MAGES SDK

Release 4.2.4

ORamaVR

Nov 10, 2022

GENERAL

1 About 3

2 Introduction 11

3 M.A.G.E.S 17

4 Getting Started 23

5 Manual 59

6 Tutorials 157

7 Video Tutorials 345

8 Class Reference 347

9 Getting Started 435

10 Manual 491

11 Tutorials 549

12 Video Tutorials 649

13 Class Reference 651

14 Cloud Services 791

15 Changelog 823

16 Changelog 831

Bibliography 833

Index 835


ORama VR

CHAPTER

ONE

ABOUT

Welcome to the official documentation of MAGES™ SDK.

1.1 MAGES™ SDK overview

1.2 MAGES™ SDK novelties

1.3 MAGES™ SDK developer experience with Unity

1.4 How MAGES™ works

Step	Process
1	**Design**: Simulation acyclic tree-like design, featuring Lessons and Stages as intermediate nodes and Actions as leaves.
2	**Assets**: 3D assets generation capable of virtual interaction.
3	**Actions**: Actions generation, utilizing our action prototype design patterns, capable of representing any action.
3.1	**Network**: Network-ready for up to 300 participants in a VR simulation.
3.2	**Analytics**: Custom analytics assignment for each action, measuring performance.
4	**Build**: Build and deploy to a wide range of supported headsets/platforms.
5	**Simulation Licensing**: Optionally connect with your cloud management license server.

1.5 MAGES™ Requirements & Specifications

Below you can find the list of requirements and specifications for **MAGES™ SDK**.

OS	Windows 10 or later / Mac OS Catalina 10.15 or later
RAM	Minimum 8GB DDR3
GPU	Integrated Intel HD Graphics 620 DX12 / AMD Radeon RX 460/560 / Nvidia GTX 1060 or better
CPU	Intel Core i5 4590 (4th gen) / Intel Core i5 7200U (7th gen mobile) / AMD Ryzen 5 1400 or better
Unity Version	2020.9.3f1
Unreal Version	4.27.2

Below you can find the list of requirements and specifications for **SIM/APPs** utilizing MAGES™ SDK.

OS	Windows 10 or later / Mac OS Big Sur 11.0 or later
RAM	Minimum 8GB DDR3
GPU	Integrated Intel HD Graphics 620 DX12 / AMD Radeon RX 460/560 / Nvidia GTX 1060 or better
CPU	Intel Core i5 4590 (4th gen) / Intel Core i5 7200U (7th gen mobile) / AMD Ryzen 5 1400 or better

1.6 Supported Platforms

1.6.1 Windows

Please first make sure that you have a Windows 10 PC which is compatible with the latest VR headsets (HMDs – head mounted displays) that allow the unique feeling of "Presence" in the virtual world. These high-end VR HMDs are coupled with hand motion controllers (type of joysticks that enable unique embodied cognition with the affordances of gesture and manipulation in the virtual environment).

Windows platform fully supports both Desktop 3D as well as VR mode for the applications built with MAGES™ SDK.

1.6.2 MacOS

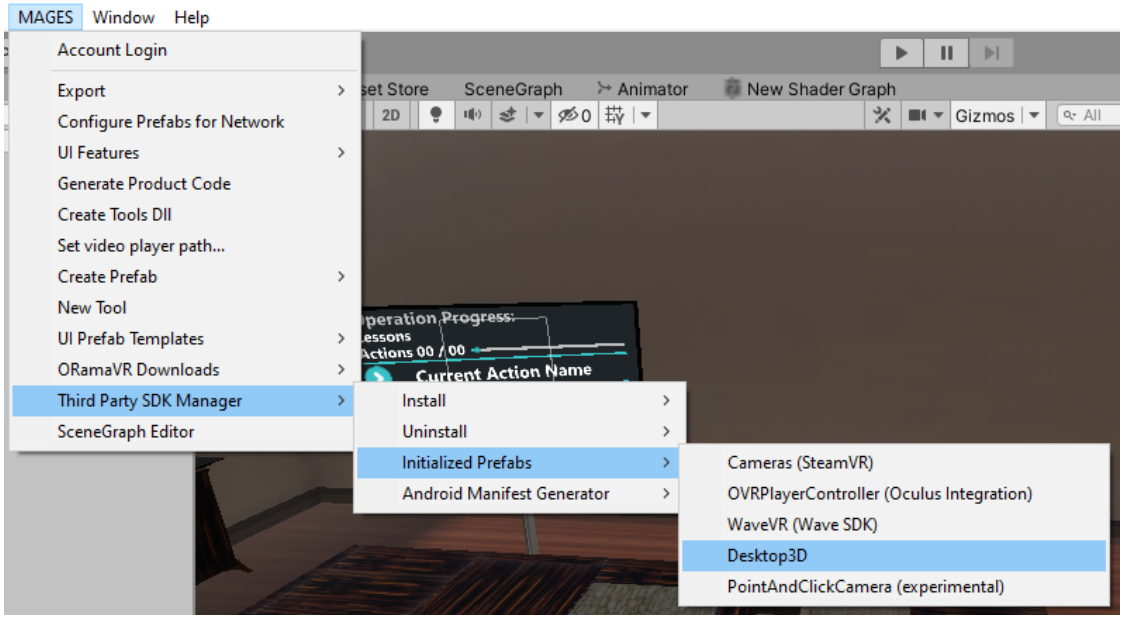
MAGES™ SDK is now available for MacOS. A new Unity project utilizing MAGES™ SDK can now be started under MacOS or even transfer your MAGES™ SDK project from Windows to your Mac and continue working there (as described in the step by step guide).

Note: Currently MacOS version only supports Desktop 3D as well as the experimental Point and Click mode. Currently, you will not be able to run a VR application on MacOS.

1.6.3 Non-VR mode

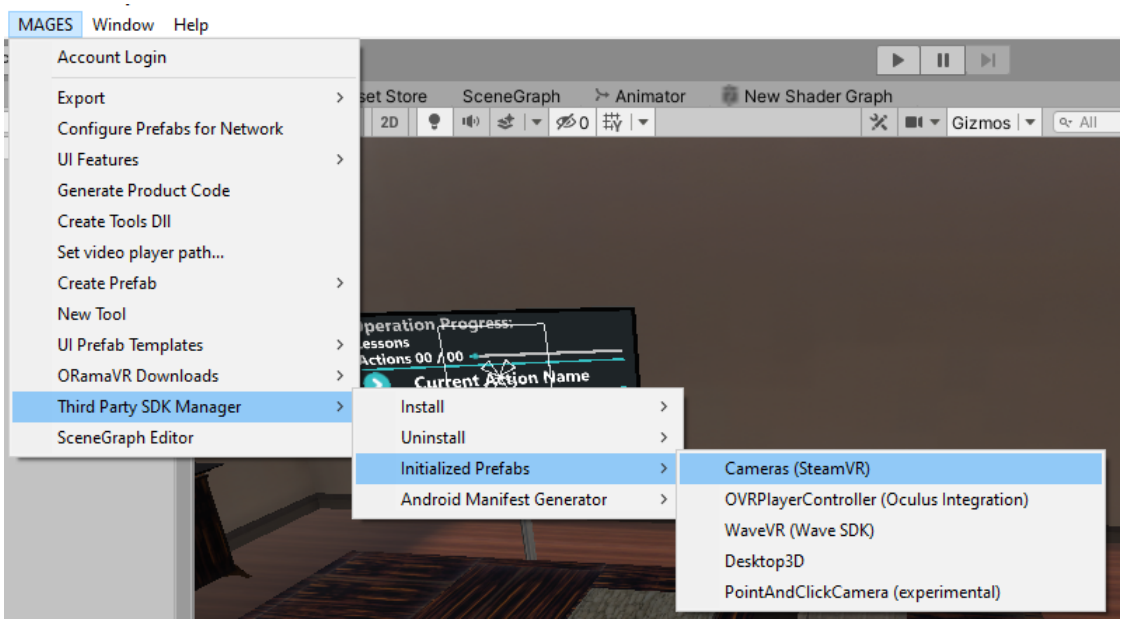
In case you do not own a VR headset, you can still utilize MAGES™ SDK in order to build applications that will run on Desktop 3D mode.

To do so, you will need to enable the Desktop 3D camera in your Unity MAGES™ SDK project. This can be done by navigating to MAGES menu (located in Unity's menu bar) and selecting Third Party SDK Manager –> Initialized Prefabs –> Desktop3D.



1.6.4 VR mode (Supported Headsets)

On Windows platform, MAGES™ SDK supports VR as well. A large number of headsets is supported through the SteamVR SDK. Once installed in your project (through MAGES menu) you can enable the Desktop VR camera for your Unity scene navigating to MAGES menu as depicted below:



This camera represents a large number of Desktop VR headsets. The VR headsets supported by MAGES™ SDK are listed below:

Headset	PC Minimum Requirements
Windows Mixed Reality	Download Windows Mixed Reality PC Check **Operating System** Windows 10 w/ Fall Creators Update (RS3). **Versions**: Home, Pro, Business, Education **CPU** Intel Core i5 4590 (4th gen) Intel Core i5 7200U (7th gen mobile) AMD Ryzen 5 1400 **RAM** 8GB DDR3 **GPU** Integrated Intel HD Graphics 620 DX12 integrated GPU (*note that this is not a separate graphics card, but it is part of specific Intel CPUs. Check if your model is greater). NVIDIA MX150 discrete GPU NVIDIA 965M DX12-capable discrete GPU AMD Radeon RX 460/560 NVIDIA GTX 1060 **Graphics Display Port** HDMI 2.0 (or 1.4) DisplayPort 1.2 **Display** External or integrated VGA (800×600) display **USB** USB 3.0 Type-A **Bluetooth** Bluetooth 4.0 (for Motion Controllers)
Oculus Rift	Download Oculus Rift PC Check (SteamVR Performance Test) **Operating System** Windows 10 **Versions**: Home, Pro, Business, Education **CPU** Intel Core i3 6100 AMD Ryzen 3 1200 **RAM** 8GB DDR3 **GPU** AMD Radeon RX 470 (alternative AMD Radeon R9 290) NVIDIA GTX 1050Ti (alternative NVIDIA GTX 960) **Graphics Display Port** HDMI 1.3 **USB** 1 USB 3.0 Type-A, 2 USB 2.0
Oculus Rift S	Download Oculus Rift S PC Check (SteamVR Performance Test) **Operating System** Windows 10 **Versions:** Home, Pro, Business, Education **CPU** Intel Core i3-6100 AMD Ryzen 3 1200, FX4350 **RAM** 8GB DDR3 **GPU** NVIDIA GTX 960 AMD Radeon R9 290 **Graphics Display Port** DisplayPort 1.2 / mini DisplayPort **USB** 1 USB 3.0 Type-A

For the case of the **Android** platform, the following headsets are supported:

Headset	
HTC Focus 3, Focus, Focus Plus,	
Meta Quest, Meta Quest 2	 Meta Quest/Quest 2 can be used also as desktop VR Headset with the use of Oculus Link
Pico Neo, Neo 2	

Read more about *MAGES*™.

CHAPTER

TWO

INTRODUCTION

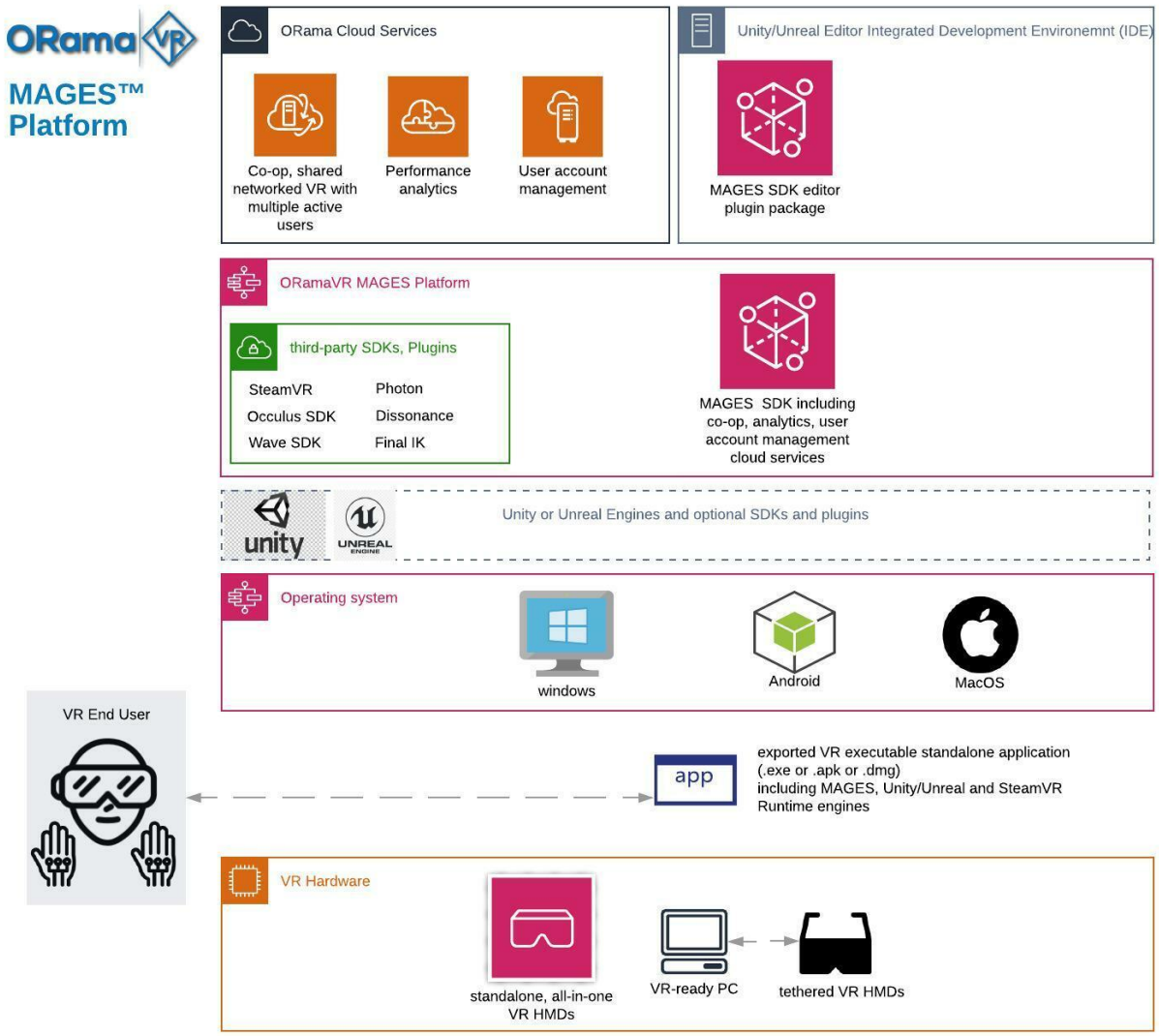
In this section we present all core components of MAGES SDK.

The aim is to provide the reader with a comprehensive overview of all fundamental pilars that ultimately compose MAGES.

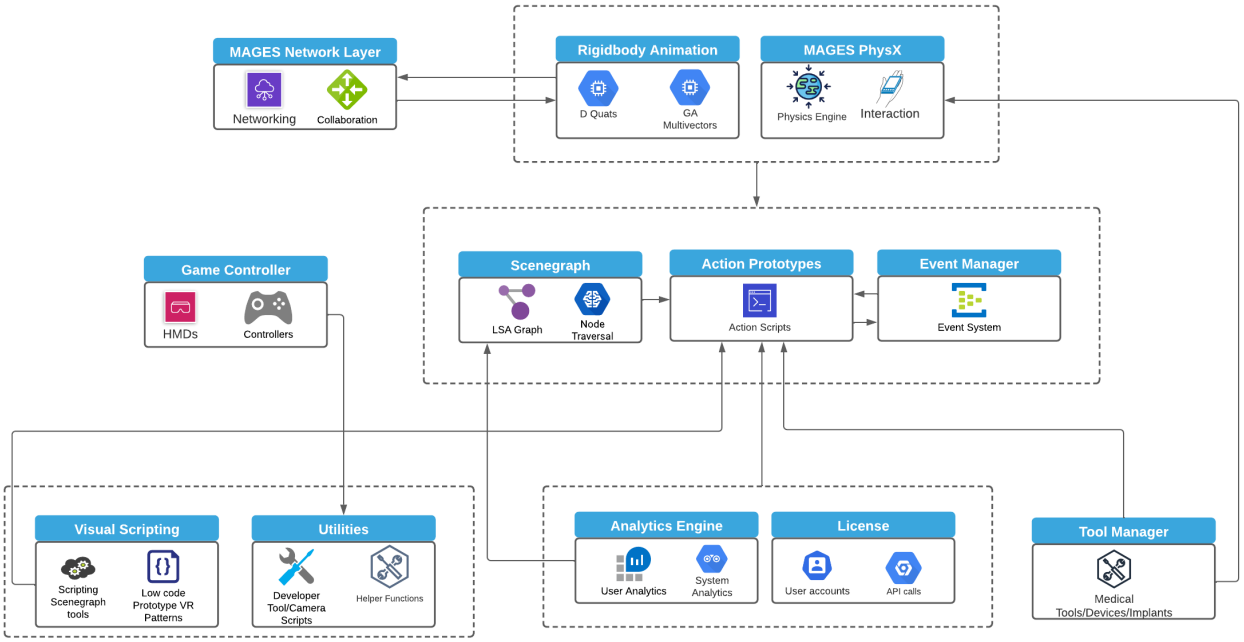
Grasping a deep understanding of the MAGES platform starts here. This section aims to familiarize readers with the terminology, development platform and software toolkits that are provided in the SDK.

Therefore, it is important for developers to pay attention to the theoretical fundamentals that will be presented in this section, as they will certainly need this knowledge to extend their development skills with MAGES SDK.

A high-level overview of the MAGES SDK architecture is depicted below.



A high-level overview of the MAGES Source code architecture is depicted below.



A complete introductory presentation of MAGES platform can be found here.

2.1 Coding with/without MAGES™ SDK

In this example we demonstrate the implementation of an Insert Action with and without MAGES SDK:

With MAGES SDK

```
public class PolyethyleneTrialAction : InsertAction
{
    public override void Initialize()
    {
        SetInsertPrefab("Lesson7/Stage2/Action0/Polyethylene",
                        "Lesson7/Stage2/Action0/PolyethyleneFinal");
        SetHoloObject(""Lesson7/Stage2/Action0/Hologram/HologramL7S2A0");

        base.Initialize();
    }
}
```

Without MAGES SDK

```
public class CraneMovement : MonoBehaviour
{

    [SerializeField]
    float moveSpeed = 3;

    [SerializeField]
    float chainMoveSpeed = 3;

    [SerializeField]
    float chainRotationSpeed = 3;

    [SerializeField]
    GameObject chainAnchor;

    // Update is called once per frame
    void Update()
    {
        transform.Translate(Vector3.forward * Time.deltaTime * Input.GetAxis(
"Vertical") * moveSpeed);
        transform.Translate(Vector3.right * Time.deltaTime * Input.GetAxis(
"Horizontal") * moveSpeed);


        // Crane Up-Down
        if (Input.GetKey("left shift") && chainAnchor.transform.localPosition.y <=
8.0f)
        {
            chainAnchor.transform.Translate(Vector3.up * chainMoveSpeed * Time.
deltaTime);
        }
        else if (Input.GetKey("space") && chainAnchor.transform.localPosition.y >=
5.0f)
        {
            chainAnchor.transform.Translate(Vector3.down * chainMoveSpeed * Time.
deltaTime);
        }

        // Crane Magned Rotate
        if (Input.GetKey("q")) // ClockWise
        {
            chainAnchor.transform.Rotate(0, -90.0f * Time.deltaTime *
chainRotationSpeed, 0, Space.Self);
        }
        else if (Input.GetKey("e")) // CounterClockWise
        {
            chainAnchor.transform.Rotate(0, 90.0f * Time.deltaTime *
chainRotationSpeed, 0, Space.Self);
        }


    }
}
```



```
public class Magnet : MonoBehaviour
{
    [SerializeField]
    float magneticForce = 10000;
```

By observing the example above, the code which results for an insert action with the absence of MAGES SDK is quite lengthy. MAGES SDK is able to perform such actions with quite a few lines of code in a robust and reliable way. The time needed to create such an action is obviously reduced, speeding up that way the application production.

There is a growing lack of medical professionals globally and not enough already are trained today for future needs [C3]. The number of new surgeons trained per year has not changed in the last 30 years, whereas the population has doubled. There is an urgent need for a paradigm shift in medical training to overcome the challenges. Previously we have proven that MAGES [C2] makes medical training more efficient. In a revolutionary clinical study [C1] in cooperation with New York University that established - for the first time in the medical bibliography - skill transfer and skill generalisation from VR to the real Operating Room in a quantifiable, measurable ROI.

Utilizing the new advances in MAGES 3.0, we released four new medical VR training modules: a dental implant placement, an endotracheal intubation, a series of emergency medical scenarios and a REBOA operation.

View the complete ORamaVR platform poster from SIGGRAPH ASIA 2018.

For a complete list of our publications related to MAGES please refer here.

(M) Multiplayer With GA Interpolation

(A) Analytics

(G) Geometric Algebra Deformable Animation, Cutting and Tearing

(E) Editor in VR with Action Prototypes

(S) Semantically Annotated Deformable, Soft and Rigid Bodies

2.2 References

CHAPTER

THREE

M.A.G.E.S

3.1 (M) Multiplayer With GA Interpolation

Our networking layer proposes a low bandwidth and high visual fidelity collaborative module, featuring multiple users based on our proprietary virtual character interpolation engine. We deploy a unique representation of the 3D deformable meshes in our Geometric Algebra (GA) framework that allows 4x improvement on reduced data network transfer and lower CPU/GPU usage for this task. This allows for a high number of multiple concurrent users in the same collaborative virtual environment.

Participants can join the same virtual OR to communicate and interact while completing the training scenario. Even non-medical related users are able to learn and perform basic surgical steps while assisted from our personalized recommendation system.

Click here for a detailed tutorial on Multiplayer

3.2 (A) Analytics

In medical training it is crucial to provide a user assessment capable of reflecting the educational impact of the used methodology. We designed an analytics system to fulfil this need.

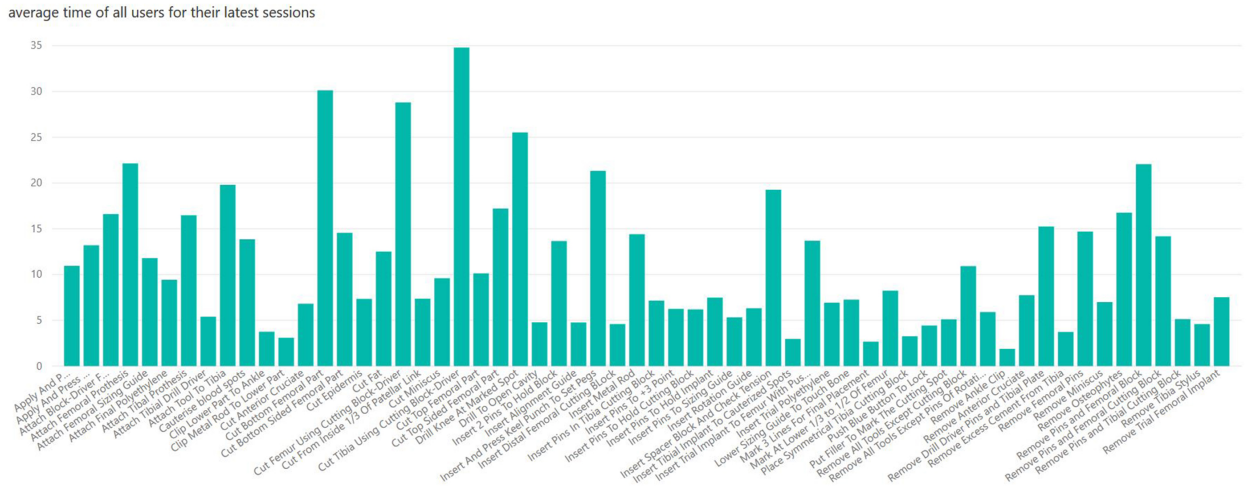
The ORamaVR Analytics engine uses a cloud-based user assessment service to track, monitor and present important feedback regarding each gamified operation. VR environments provide a variety of tracking capabilities, from hand movement tracking to measuring an unexpected approach; we utilize a variety of means to generate a surgical profile for each user that will be used for evaluation.

Each surgical action is segmented into individual parts to identify interactions such as tool handling and proper usage of medical equipment.

In our e-Learn platform we offer a monitor system for supervisors to track the progress and skills of their users. Such system enhances the traditional ways of supervision by allowing instructors to monitor an entire class of students. Real-time error tracking with visual indications, completion time of each step and global leaderboard are also supported.

We have also managed to distinguish the critical complication parts of each module and offer various error types, which augments the real-time operation with additional challenges. Gathered analytics data from users is processed and used to normalize and improve the simulation's scoring system based on the performance of the users.

Click here for a detailed tutorial on Analytics



3.3 (G) Geometric Algebra Deformable Animation, Cutting and Tearing

Our work focuses in enhancing the state-of-the-art in skinning and handling of models. The use of quaternions and dual quaternions yielded fast results, free of interpolation problems or other geometric artifacts. Another step towards that direction is the introduction of Conformal Geometric Algebra (CGA).

In a CGA framework, all points (P), translations (T), rotations (R) and dilations (D) are described as a single entity, called multivectors.

Since the interpolation of two multivectors of type $\in$, , , yields a multivector of the same type, we acquire an easier to understand and implement interpolation algorithm without the need to constantly transmute objects of two worlds such as in the case of quaternion-matrices.

The CGA framework we implemented converts the original skinning equation to a multivector-only equivalent:

$$C_k[m] = \sum_{n \in I_m} w_{m,n}(M_{n,k}B_n)c[m](M_{n,k}B_n)\star$$

where amounts to an offset matrix, [] is the image of the-th vertex of the model in CGA, ¸ is the influence of the -th bone on the -th vertex, ¸ is the transformation of the -th bone at time , is the list of bones influencing the -th vertex and [] is the image of the -th vertex of the final model at time .

The latter implementation yields animations close to the former method, while also enabling us to perform character deformable cutting and tearing. Furthermore, our engine performs animations with less intermediate keyframes, reducing bandwidth.

Click here for a detailed tutorial on Cutting, Tearing and Drilling

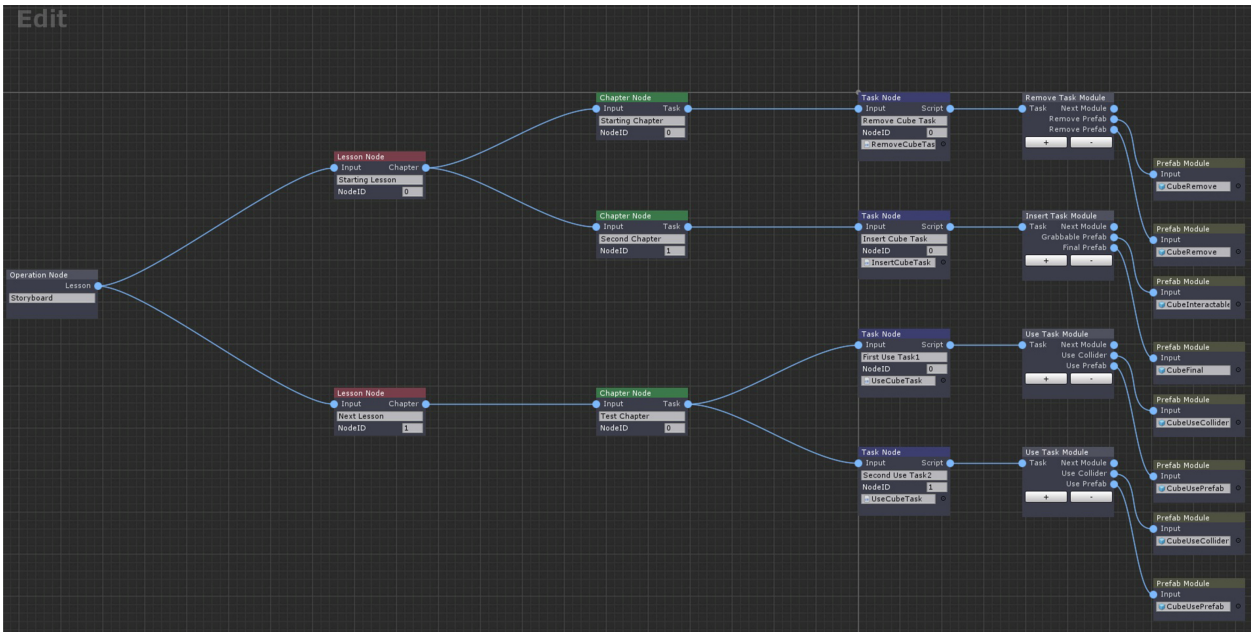
3.4 (E) Editor in VR with Action Prototypes

Our SDK encapsulates a VR editor capable of generating VR training scenarios following our modular Rapid Prototyping architecture based on storyboard, a visual representation of the VR simulation sequence, is split into discrete Action prototypes.

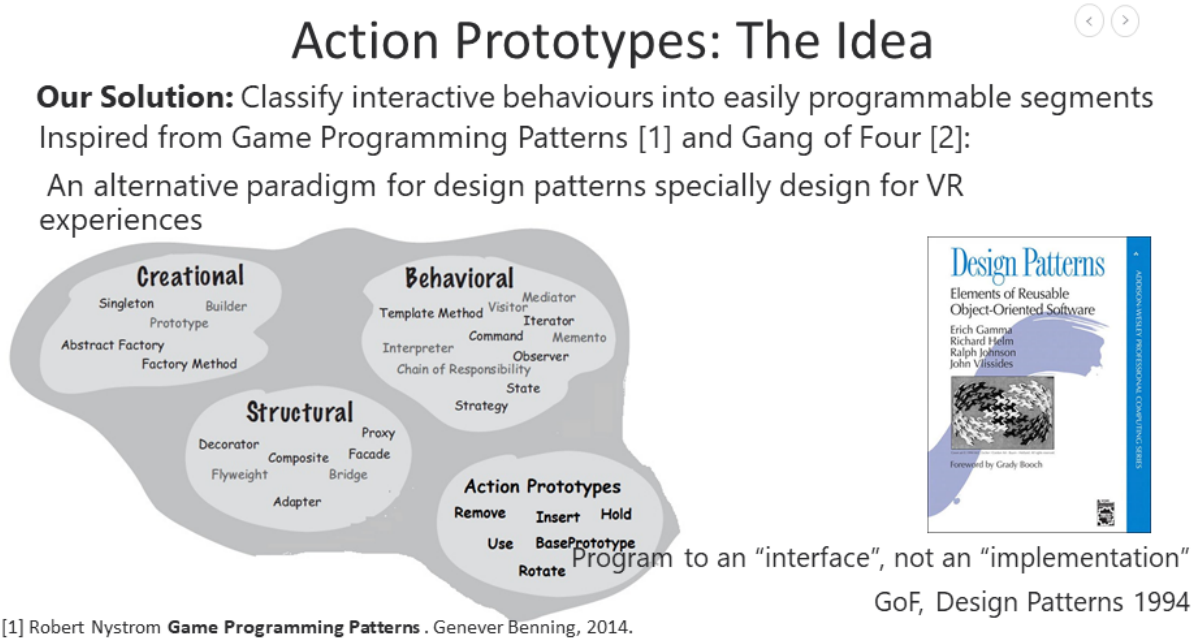
We designed our system as a collection of authoring tools combining a visual scripting system and an embedded VR editor forming a bridge from product conceptualization to product realization and development in a reasonably fast manner without the fuss of complex programming and fixtures. MAGES SDK platform is designed for any programmer or doctor to make the development of various surgical scenarios rapid and simple.

Therefore the platform allows for non-VR experts to develop new surgical modules/scenarios or modify existing ones, increasing the platform's possibilities.

Click here for a detailed tutorial on the Scenegraph Editor



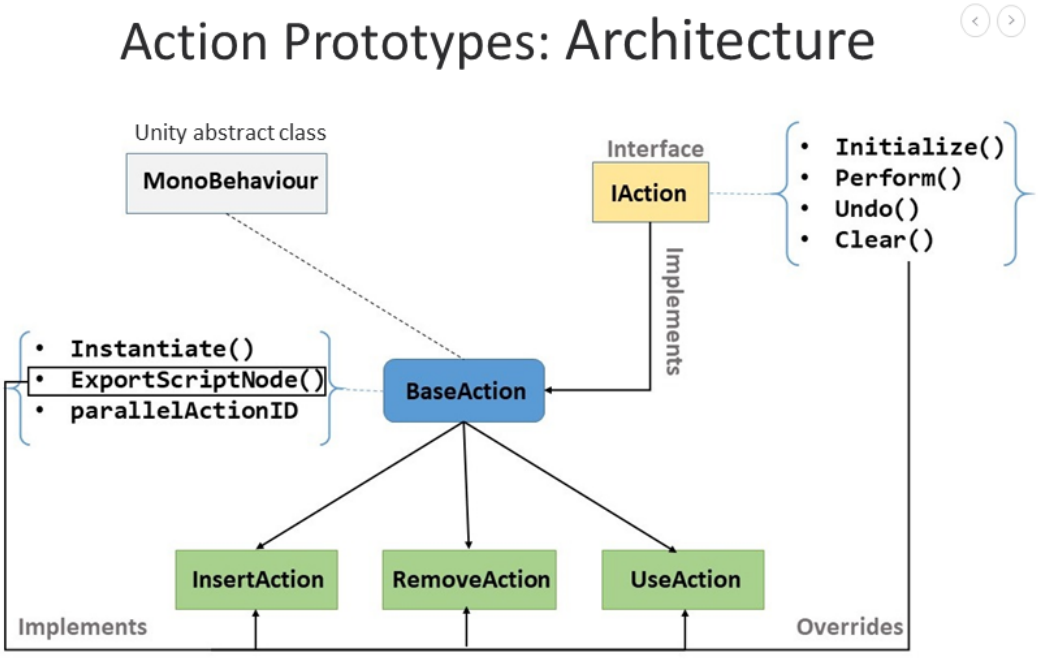
Let us try to elucidate Action prototypes, a new software design pattern suitable to replicate behavioral tasks for VR experiences. Each Action prototype implements specific methods according to the functionality we would like to support.



Click here for a detailed tutorial on Action Prototypes

Action Prototypes rapidly accelerate the development of gamified VR experiences, introducing a new way to programm interactive behaviours.

The diagram below illustrates an architectural diagram of Action prototypes to visualize better their dependencies.



The Base prototype does not represent a behaviour like the previous prototypes, is the base class where the other prototypes derive from. It contains common methods used across multiple prototypes for better organization and code

optimization.

3.5 (S) Semantically Annotated Deformable, Soft and Rigid Bodies

In the core of MAGES lies an advanced mathematical algorithm for physics-based visual techniques to allow the 3D representation of deformable soft body objects (skin, tissue, etc.), essential for VR surgical training.

Since surgical training is all about cutting and suturing soft body objects, collision detection (touching) and handling of soft bodies with other objects becomes crucial for high-realism VR. The soft deformation algorithm is based on shape matching techniques and particle-based soft body simulations.

Our methodology differs from the state of the art since it provides onthe-go control of the particles as physical objects and a centre point, which controls the entire soft body. Velocity based interaction can be applied directly to the corresponding particles while interacting with the environment as objects.

In addition we synchronize the deformable objects over network utilizing our GA interpolation engine to improve interaction with concurrent users.

Finally, we extended our physics engine to support interaction with ropes and even giving user the ability to perform knots and sutures in VR. In combination with our soft body mechanics we developed a bowel anastomosis operation where users can interact with the virtual sutures and soft tissue.

Click here for a detailed tutorial on Soft Bodies

CHAPTER

FOUR

GETTING STARTED

4.1 Step by step

4.1.1 Download and Import MAGES™ SDK

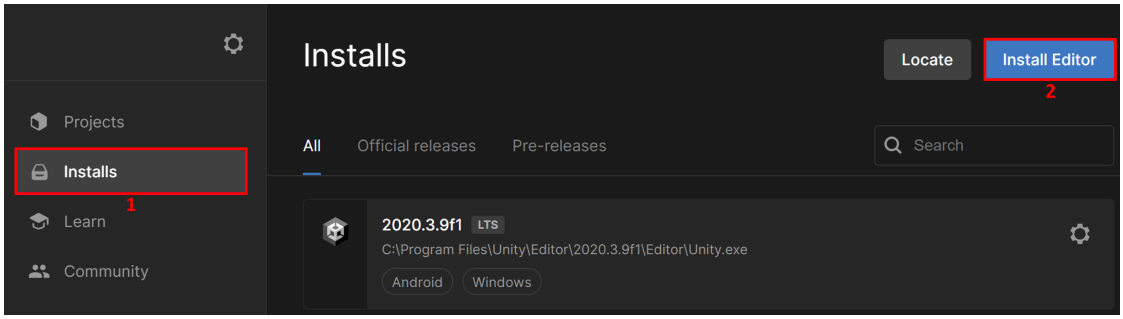
Before you start

Latest stable release of MAGES SDK runs on Unity 2020.3.9f1 LTS. Login using your developer account credentials.

Note: Support for Unity 2021.3 LTS is available in BETA version

Unity manages its latest versions through Unity Hub. You can download it by visiting this page.

After installing Unity Hub, navigate to the **Installs** tab and click the **Install Editor Button** as illustrated below.

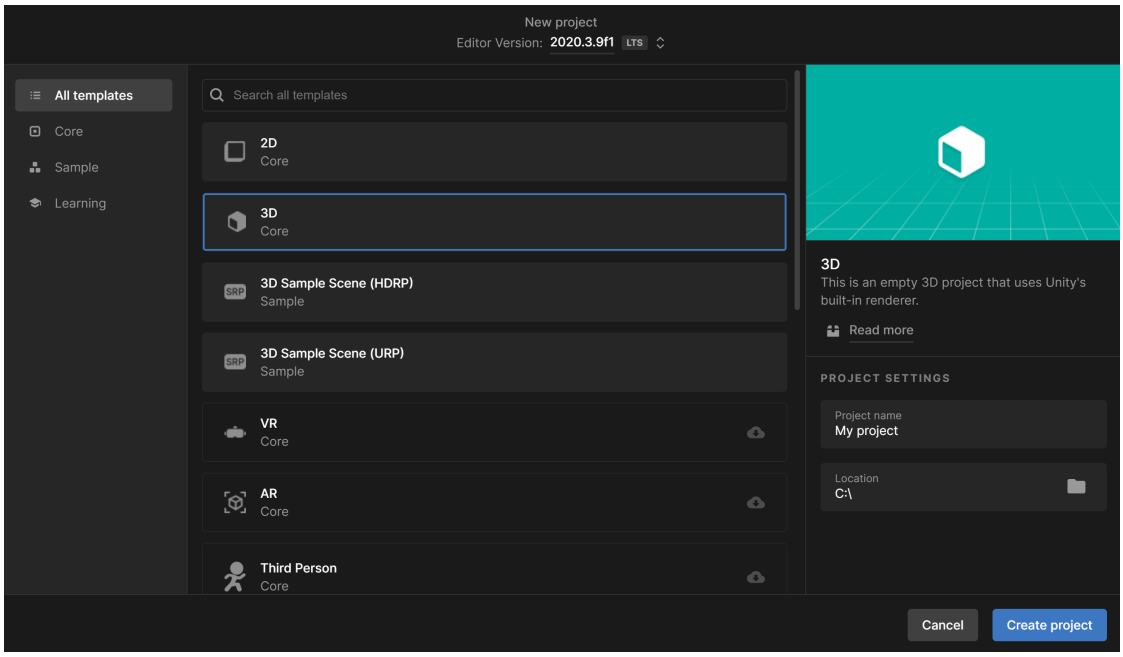


Finally, in the **Archive** tab you can locate the specified Unity version and install it.

Warning: Make sure you install this specific version otherwise you may encounter incompatibilities between internal and third party packages.

Before you download/import MAGES™ SDK you have to create a new empty Unity project.

It is recommended to start an empty Unity project by selecting the 3D preset, as shown in the image below:

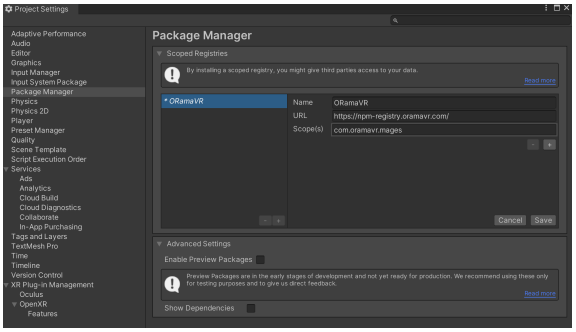


Download MAGES™ SDK

Unity is currently utilizing the package manager functionality in order to install and update third party packages with ease.

To let your Unity project access the MAGES™ package, you need to add ORamaVR registry to your project list of registries. To do so, follow the steps below.

1. Navigate to **Edit** → **Project Settings** → **Package Manager**.
2. In the **Scoped Registry** section, add a new scoped registry and fill in the details as illustrated below.



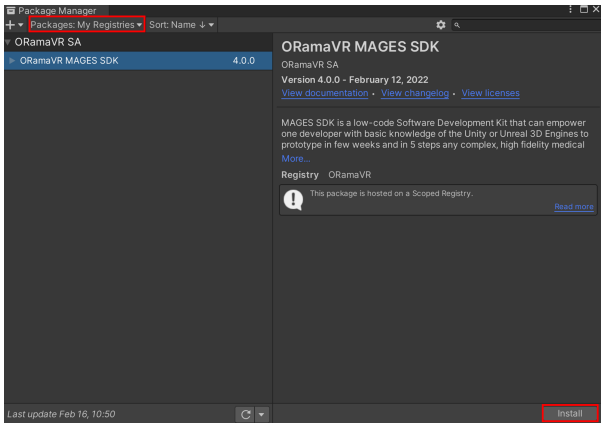
For your convenience, you can find these links below:

Name	ORamaVR
URL	https://npm-registry.oramavr.com/
Scope(s)	com.oramavr.mages

3. Click **Save**, or **Apply**.
4. Navigate to **Window** → **Package Manager**. You should be able to locate MAGES™ package under ORamaVR registry.

> **Note:** Make sure to set **Packages** to **My Registries** from the dropdown menu on the top of the window, to be able to see the MAGES™ SDK package.

5. Select the MAGES™ package and click the **Install** button.

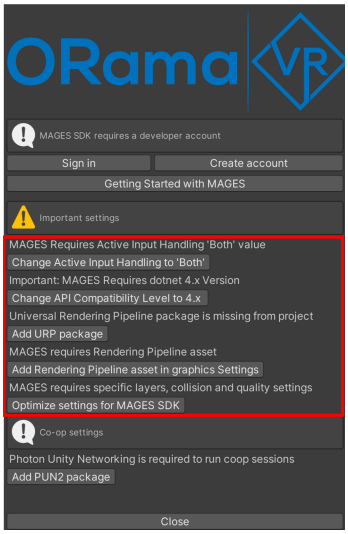


> **Note:** Depending on your internet connection speed, it may take several minutes to download the package.

6. You will be presented with the MAGES Startup window. It is recommended to enable all the settings listed below, with the order they are listed.

Change Active Input Handling to 'Both'
Change API Compatibility Level to 4.x
Add URP Package
Add Rendering Pipeline asset in graphics Settings
Optimize Settings for MAGES SDK

Below you can see the startup window. All the necessary options are highlighted.



Note: It may take some time to apply each setting.

Note: **PUN2 package** is used only if your application requires multiplayer features. If you are interested only in single player mode, you can ignore this package.

Warning: To be able to use MAGES™ SDK, you must have an ORamaVR account. You can easily create one by clicking the corresponding button in the startup window, or by clicking here.

4.1.2 Import a MAGES™ SDK Sample

Importing a Sample

Samples are short, completed applications made with Unity and MAGES™ SDK. Their purpose is to demonstrate MAGES™ SDK capabilities, as well as provide the users with examples, so they can start their own projects easily.

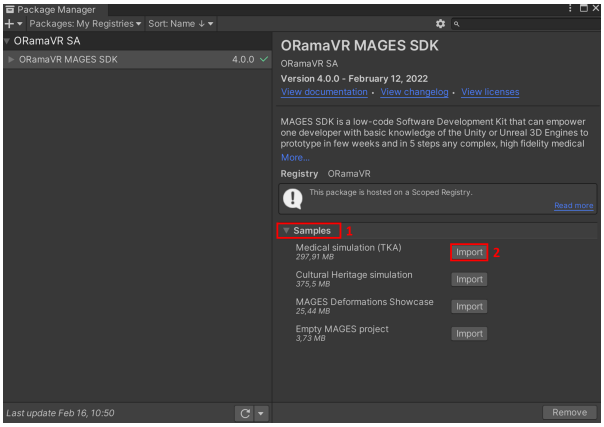
Currently, MAGES™ SDK supports the following Samples:

Sample Name	Details
Medical Simulation (TKA)	A simulation demonstrating some parts of the Total Knee Arthroplasty.
Cultural Heritage Simulation	A cultural simulation showcasing examples of the Knossos monument construction and Sponza restoration.
MAGES Deformations Showcase	Examples of different deformations of deformable meshes using MAGES™ SDK.
Empty MAGES™ Project	An empty scene, in which MAGES™ SDK is preinstalled. It is the perfect example to start a new MAGES™ project.

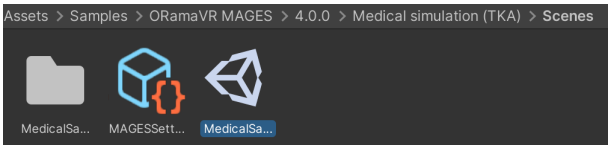
Note: As mentioned above, the Empty MAGES™ Project is the ideal scene to start a new project.

To import a MAGES™ SDK sample, Medical Simulation (TKA) in our case, kindly follow the steps below.

1. Open the Unity Package Manager and locate the MAGES™ SDK package.
2. Expand the **Samples** list and click the **Import** button next to **Medical Simulation (TKA)** as illustrated below.



3. Once the import procedure completes, you can open the scene located at `Assets/Samples/ORamaVR MAGES/4.0.0/Medical simulation (TKA)/Scenes/MedicalSampleApp.unity`.



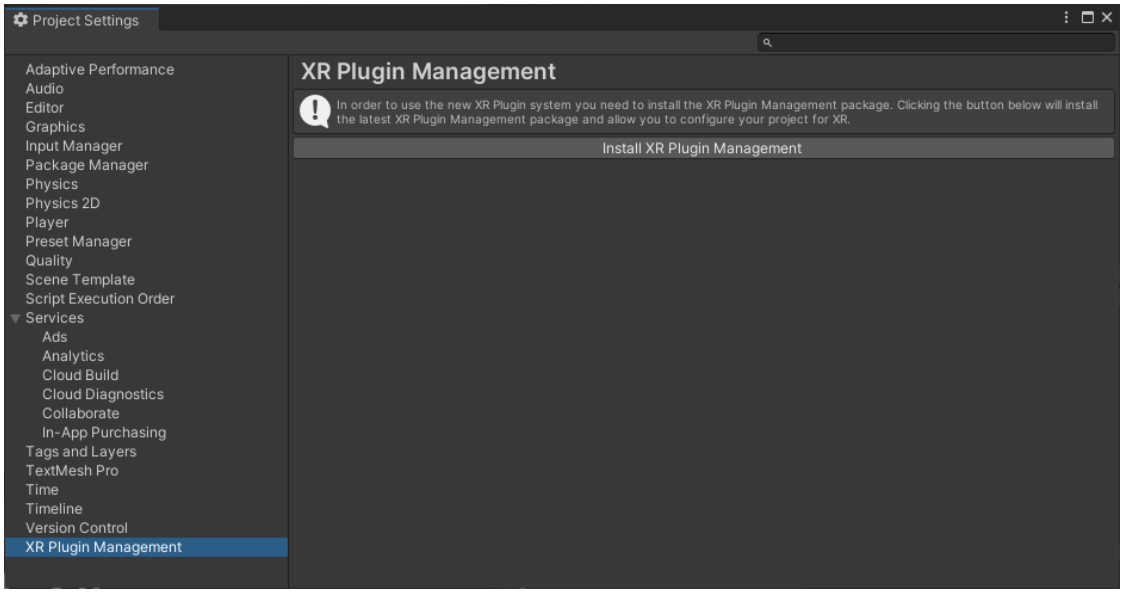
Note: The default camera is set to 2DoF mode to run the application using mouse and keyboard.

Adding Multiplayer Support

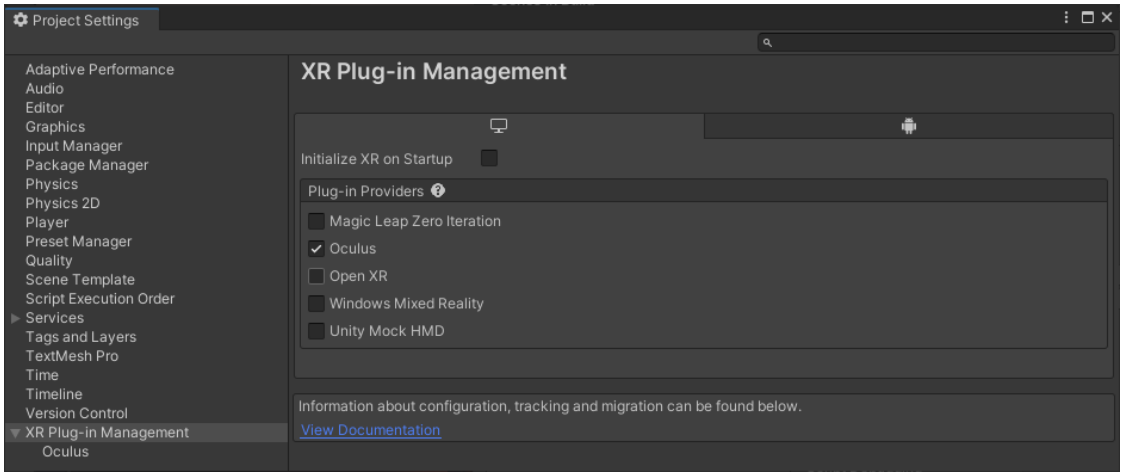
Applications built with MAGES™ SDK are multiplayer/network ready, meaning that with a few more actions needed by the developers, multiple users can cooperate and complete these simulations together online. To quickly setup your application to be multiplayer-ready please refer to *Multiplayer* guide.

Adding VR Support

To configure the application for VR support you need to install the **XR Plugin Manager**. Navigate to the **Project Settings** and click **Install XR Plugin Manager**



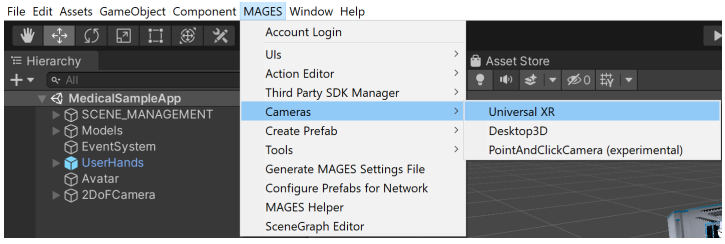
The next step is to configure your XR device. Select among the available plug-in providers. In this case, we selected Oculus.



The final step is to add the corresponding Universal XR camera to the scene.

First, delete any other MAGES camera from the scene (g.g the 2DoFCamera).

Then, from the MAGES menu navigate to **Cameras/Universal XR**. This will instantiate the **Universal_XR_Rig** camera that supports all the devices from the XR Plugin Manager.



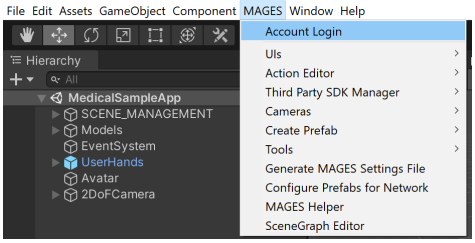
The supported cameras are the following:

1. Universal XR (for XR Plugin)
2. Desktop 3D (Non-VR camera imitating the VR controls with mouse and keyboard)

3. PointAndClickCamera (Non-VR camera for mouse only)

SDK License

Before you hit the **Play** button, make sure you have checked out a valid SDK license. To do so, open the **MAGES** tab on the top menu and click on the **Account Login** option.



Note: You can log in either with an ORamaVR account or by using your Google account.

Controls & Movement

In the table below you can find the controls for every supported platform and headset.

	Oculus Touch	VIVE	Windows Mixed Reality	Desktop3D
Grabbing Objects	Grip Button	Grip Button	Grip Button	
UI Selection & Usage of Tools	Trigger Button	Trigger Button	Trigger Button	
Toggle Movement	Press Left Touch Thumbstick	Left Menu Button	Right Menu Button	
Positional/Rotational Movement	Left/Right Touch Thumbstick	Left/Right Trackpad	Left/Right Thumbstick	Mouse and Keyboard (W,A,S,D)
Toggle In-game Options	Press Right Touch Thumbstick	Right Menu Button	Right Menu Button	

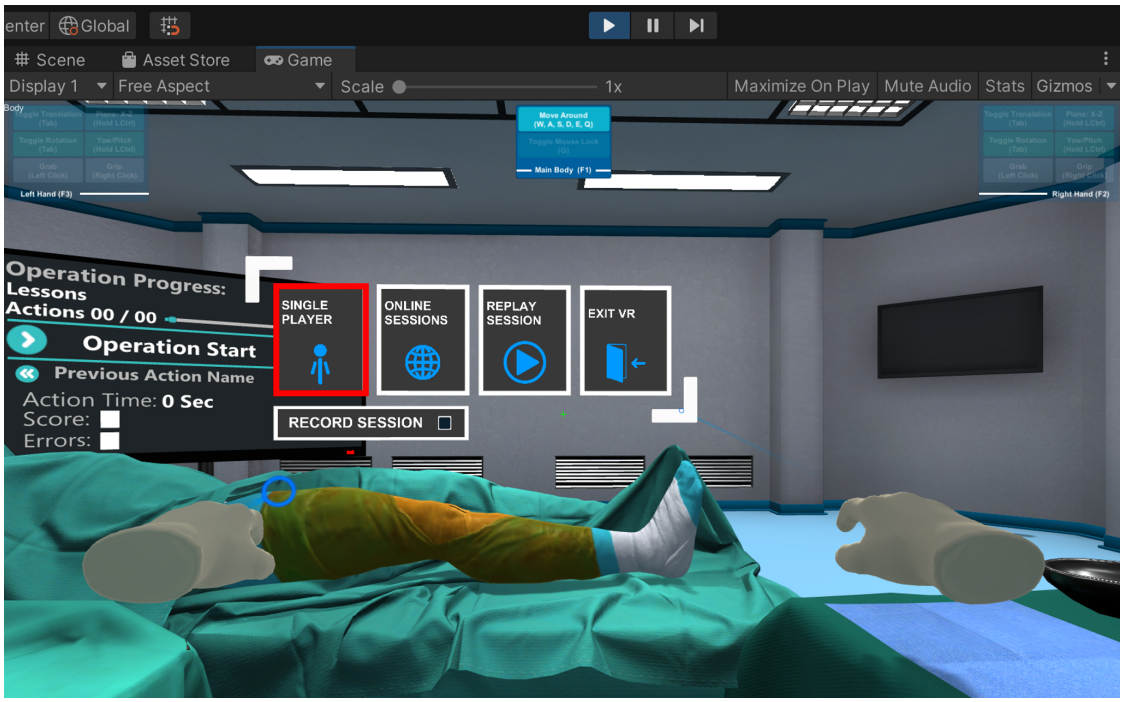
Alternatively, you can see *here* for a detailed tutorial on how to use the 3D Desktop camera if you don't have a VR headset. There is also a *video* in the video tutorials section.

How to Play

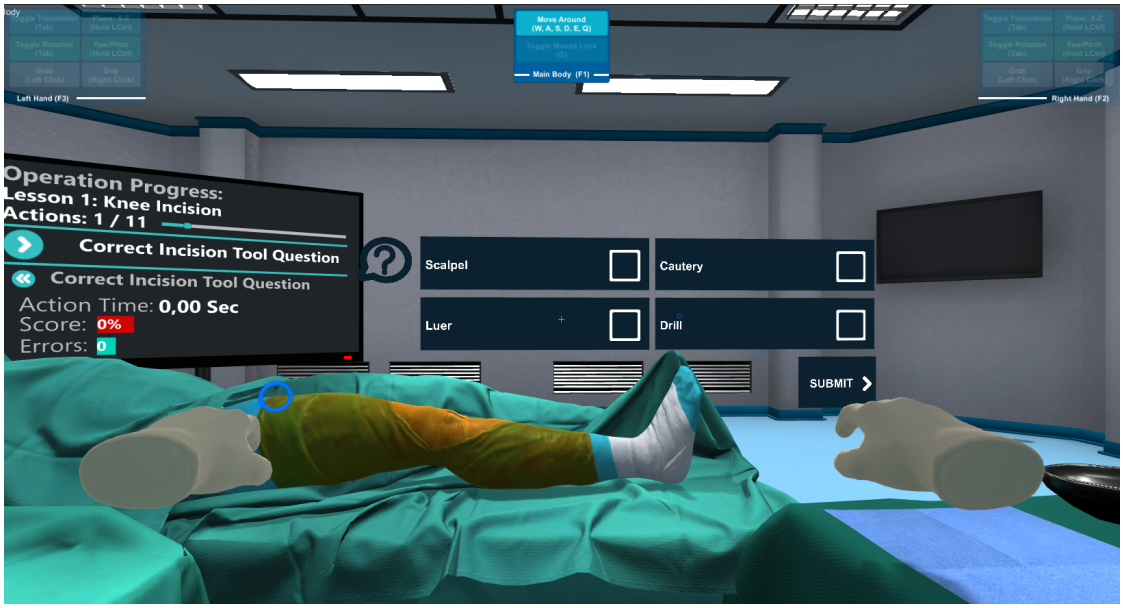
Once logged in, ensure that the MedicalSampleApp.unity scene is open and click the **Play** button on the Unity Editor.

A menu will appear, offering multiple buttons as options.

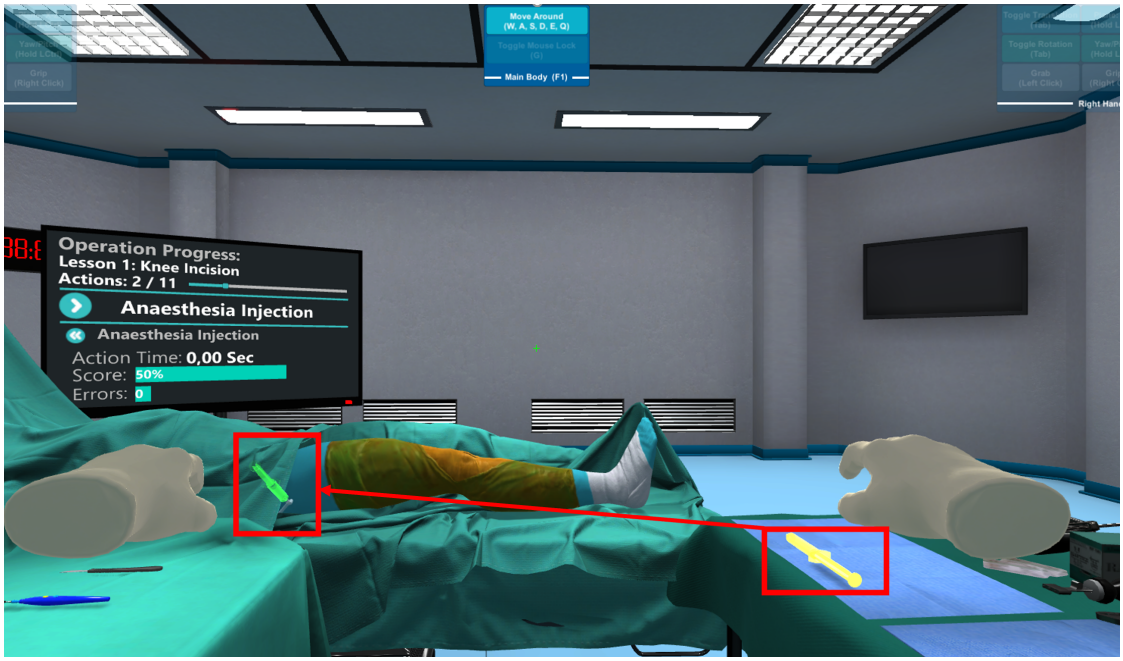
Each of the user's virtual hands, has a ray which starts at the top of the hand. Use this ray to point to the **Single Player** button and use the left mouse button to select it.



The first action of this simulation is a question action. Use the ray to point to an answer and the left mouse button to select it. Then point to the **Submit** button, using the ray and again use the left mouse button to select it and move to the next action.



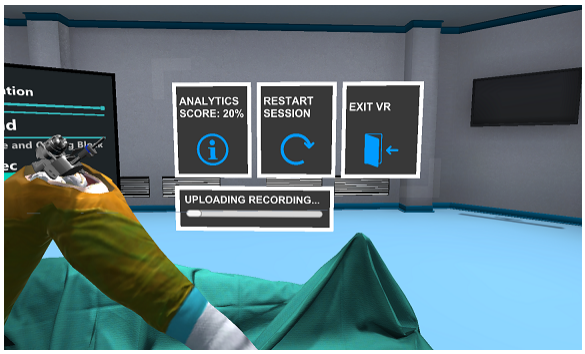
In the next action, you have to use one of the virtual hands to pick up the glowing syringe (by moving the hand close to the syringe and holding the left mouse button) and move it to the position that the green hologram represents to complete this insert action.



Once correctly inserted, click the left mouse button again to pump the syringe.

Note: Green holograms represent the position of the corresponding item or the way that this item should be used on a surface.

When the simulation is completed, the operation exit user interface will spawn. It will look like the one in the image below.



- You can view your analytics by clicking the analytics button.

- The restart button starts the simulation from the beginning.
- You can exit the simulation by clicking the exit button.

How to Build

You can find instructions on how to build the Sample apps depending on your target platform *here*

4.1.3 Build Instructions

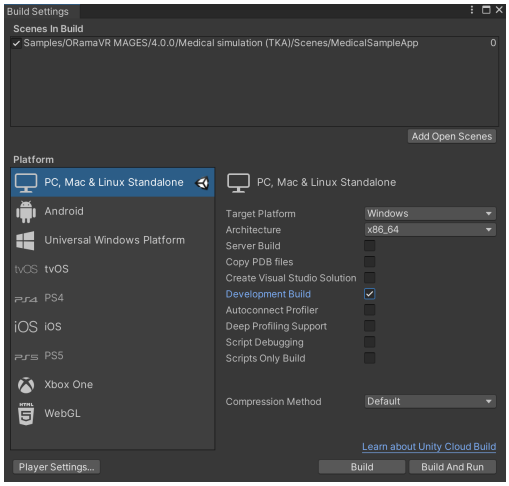
In this section we describe the build process for the Medical Sample App sample of MAGES™ SDK.

Warning: The process is significantly different for each platform so make sure you follow the exact steps if you want to avoid surprises.

Windows

To build the application for Windows you need to follow the next steps:

1. Open the **Build Settings** through **File/Build Settings** in the Unity Editor or by pressing `Ctrl+Shift+B`
    1.1. Make sure that the Medical Sample App scene is present and checked in the list.
    1.2. Make sure the selected Platform is set to Windows
    1.3. Ensure the **Architecture** is set to *x86_64*



Note: If you are building a release version, make sure to clear the `Development Build` checkbox.

2. Click the Build button and select the output path of your executable.

Note: The default ProductName is "platform" for the SDK. If you want to alter it, set it manually through the **MAGESSettingsMedical.asset** located in `Assets/Samples/ORamaVR MAGES/4.0.0/Medical simulation (TKA)/Scenes`. This step is mandatory for the StoryBoard to load correctly when running a built version.

Warning: In case you alter the **Product Code** value, you must change the folder name, where the xml resides in the Documents folder (e.g. `{User}/Documents/ORamaVR/Story/platform` → `{User}/Documents/ORamaVR/Story/{new_product_code}`).

Note: The same build technique holds for any project you would like to build with MAGES™ SDK.

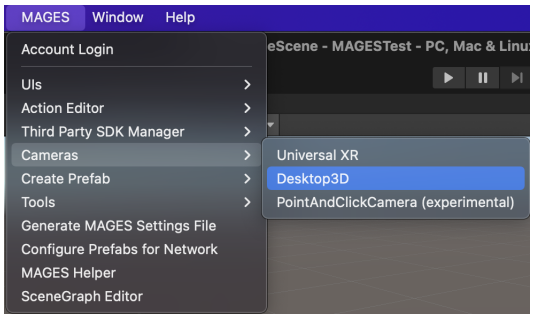
macOS

Warning: MAGES SDK does not currently support VR for the macOS platform.

Thus, applications utilizing MAGES SDK that are built on macOS platform can only run in Desktop 3D mode.

Note: You can create and build applications targeted for Android devices (e.g. Vive Focus Plus, Focus 3, Meta Quest 1 & 2) on macOS using MAGES™ SDK normally.

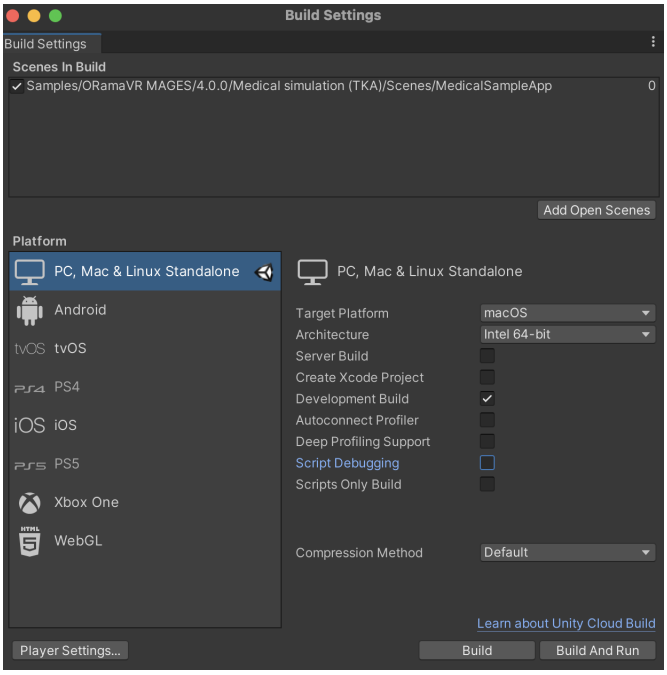
The Desktop 3D camera can be spawned through the MAGES menu, by navigating at MAGES → Third Party SDK Manager → Initialized Prefabs → Desktop3D.



To build the application for macOS you need to follow the next steps:

1. Open the **Build Settings** through **File/Build Settings** in the Unity Editor.

    3.1. Make sure the selected Platform is set to Mac OS X



Note: If you are building a release version, make sure to clear the `Development Build` checkbox.

Warning: Currently, MAGES™ SDK supports building for macOS systems with Intel processors. Make sure you set the architecture to **Intel 64-bit**, in build settings.

2. Click the Build button and select the output path of your executable.

Android

The following guide covers build instructions for *Oculus Quest* and *HTC VIVE Focus*.

In order to deploy an application utilizing the MAGES SDK to an android mobile VR headset, it is mandatory to switch your target platform to Android. This is done through **File/Build Settings** or `Ctrl+Shift+B`.

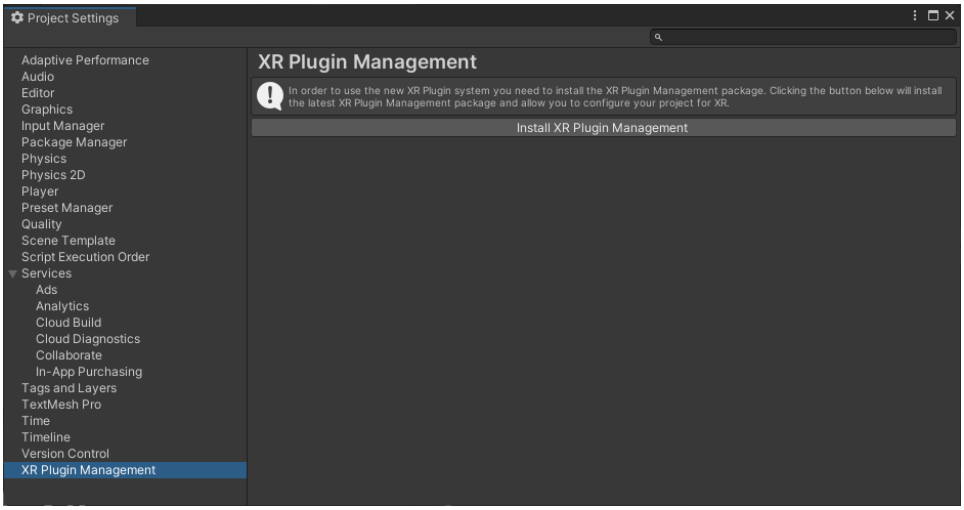
Click on the Android platform and finally click the Switch Platform button on the lower-right side of the current window.

Note: Switching between platforms induces significant compilation times. It is best that you develop on the platform you are targeting from the start.

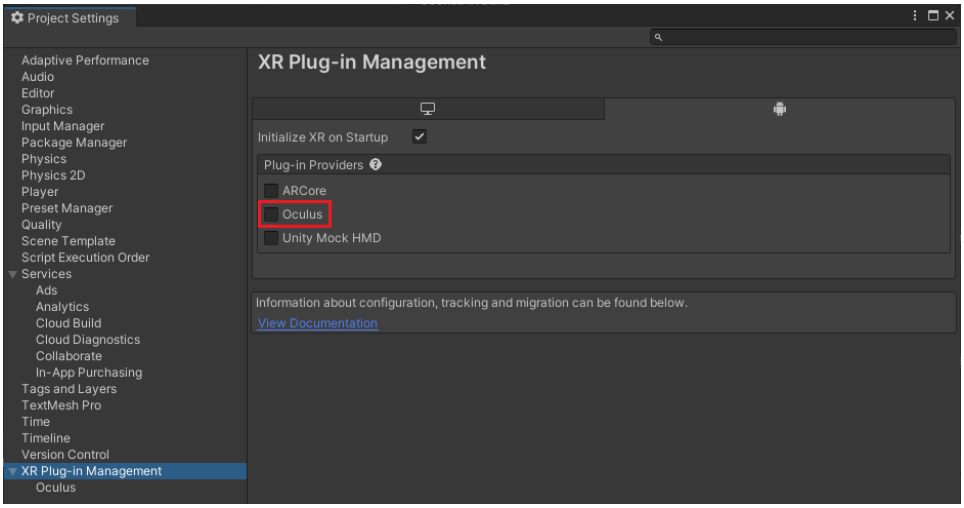
Oculus Quest

To deploy your application to Oculus Quest, follow the steps below:

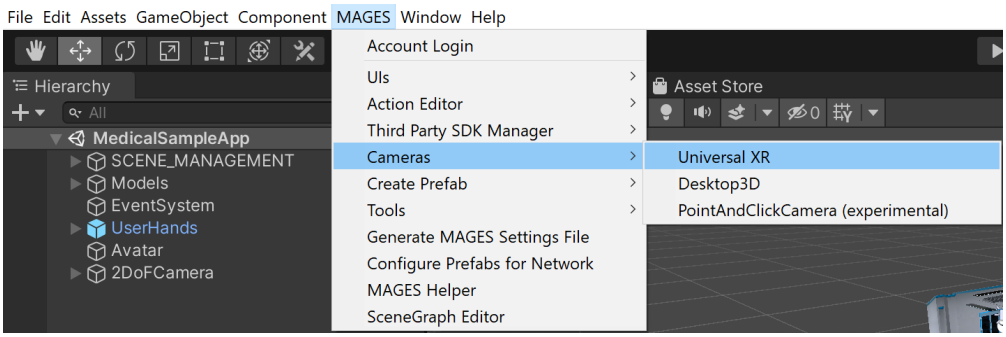
1. Install **XR Plugin**. Navigate to the **Project Settings** and click **Install XR Plugin Manager**



2. Configure your XR device. From the Android tab select the Oculus option.

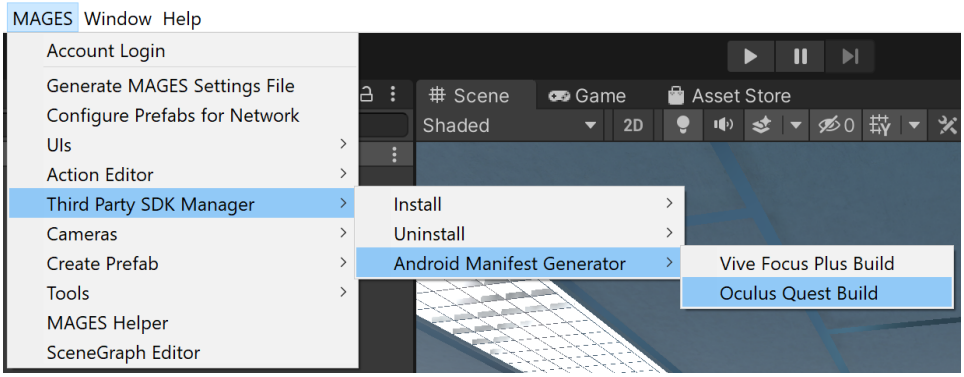


3. Add the Universal XR camera to the scene. From the MAGES menu navigate to **Cameras/Universal XR**. This will instantiate the **Universal_XR_Rig** camera that supports among others the Oculus Quest headset.



4. Android builds necessitate an *AndroidManifest.xml* in order to run correctly in the target platform.

   To generate one, navigate to **MAGES/Third Party SDK Manager/Android Manifest Generator/Oculus Quest Build**.



5. You are ready to build the application. Navigate to **File/Build Settings** and press the Build button.

VIVE Focus Plus

In order to access the **Wave SDK packages** needed to build your application for **VIVE Focus Plus**, do the following:

Open the **manifest.json** file located under **Packages** folder and add the following entry:

```
{
    "scopedRegistries": [
        {
            "name": "VIVE",
            "url": "https://npm-registry.vive.com",
            "scopes": [
                "com.htc.upm"
            ]
        }
```

(continues on next page)

(continued from previous page)

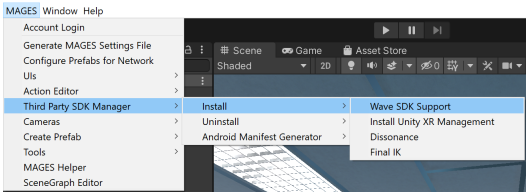
```
        ]
}
```

Warning: Please make sure that your project quality settings are set to **Low** preset for **VIVE Focus Plus**.

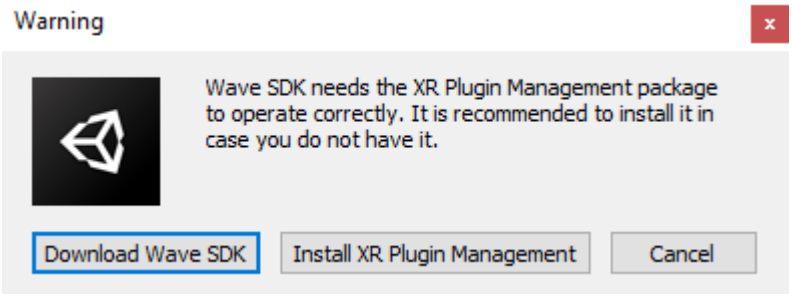
To deploy your application to **VIVE Focus Plus**, follow the steps below:

1. Install **Wave SDK packages**.

   Click on the **MAGES** menu and then **Third Party SDK Manager/Install/Wave SDK Support**.



   The following window will appear. Click **Download Wave SDK** to proceed.

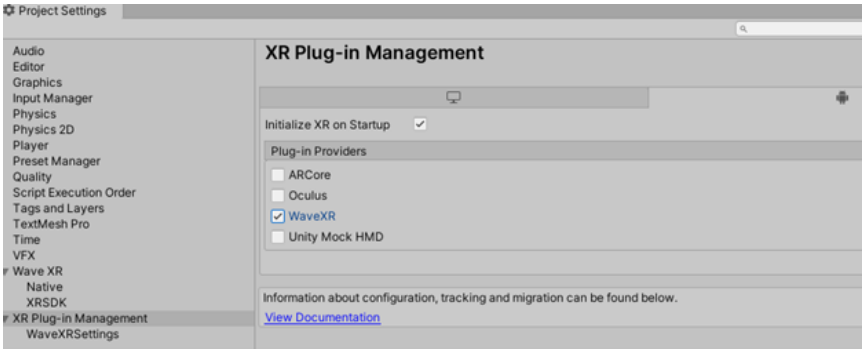


   Note: MAGES™ SDK supports Wave SDK using Unity's XR Plugin Management.

   If you do not have the XR Plugin Management package, please install it by clicking the Install XR Plugin Management button.

   Once this plugin is installed, you may click Download Wave SDK to proceed.

2. Navigate to **Project Settings/XR Plugin Management** and tick the **Wave XR** checkbox on the **Android** tab.

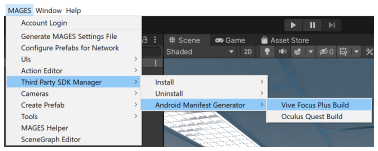


3. Make sure a player camera gameobject is present in the scene.

   Note: To create one navigate to **MAGES/Cameras/Universal XR**.

4. Android builds necessitate an *AndroidManifest.xml* in order to run correctly in the target platform.

   To generate one, navigate to **MAGES/Third Party SDK Manager/Android Manifest Generator/Vive Focus Plus Build**.



5. You are ready to build the application. Navigate to **File/Build Settings** and press the Build button.

Universal Windows Platform (HoloLens 2)

HoloLens 2

In order to deploy an application utilizing the MAGES SDK on the Microsoft HoloLens 2, it is mandatory to switch your target platform to UWP (Universal Windows Platform). This is done through **File/Build Settings** or `Ctrl+Shift+B`.

Click on the UWP platform and finally click the Switch Platform button on the lower-right side of the current window.

Note: Switching between platforms induces significant compilation times. It is best that you develop on the platform you are targeting from the start.

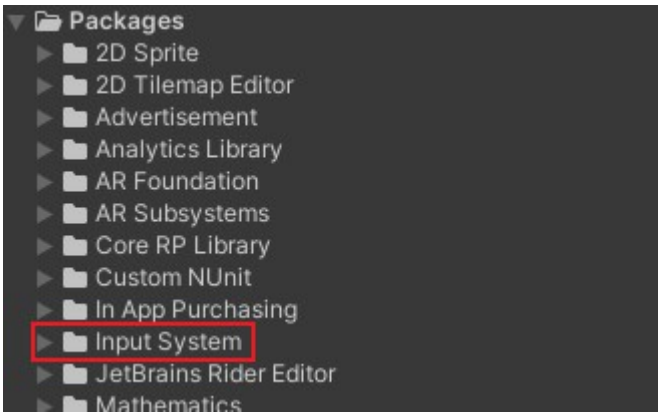
Warning: Visual Studio 2019 or later is needed for building applications for Microsoft HoloLens 2. You will also need the **Universal Windows Platform Development** and **Desktop Development with C++** components in your Visual Studio installation.

To deploy your application to the HoloLens 2, follow the steps below:

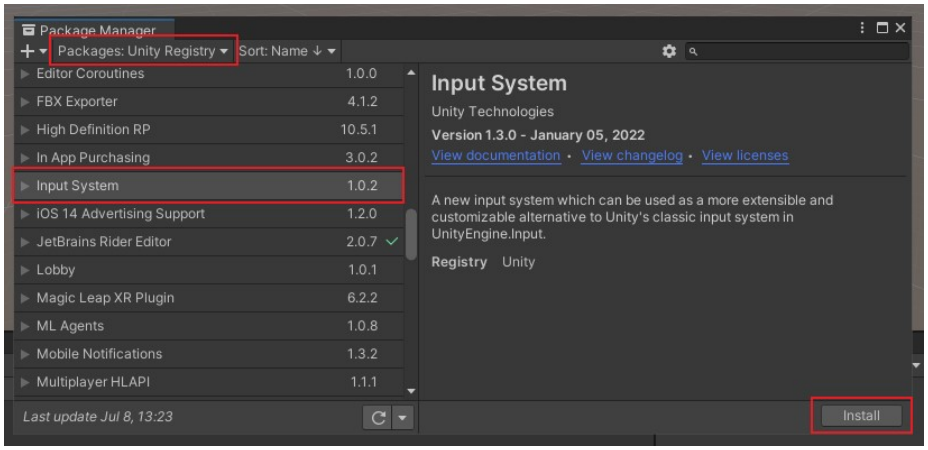
1. Download the **Mixed Reality Feature Tool** here.
2. Use the Mixed Reality Feature Tool to import the **Mixed Reality OpenXR Plugin** into your project, by following this tutorial..

Note: Mixed Reality OpenXR Plugin version 1.4.1 is officially supported by the MAGES SDK. Should you encounter any issues with other versions, please revert to this version.

Warning: Please check if the **Unity Input System** Package was installed with the Mixed Reality OpenXR Plugin.



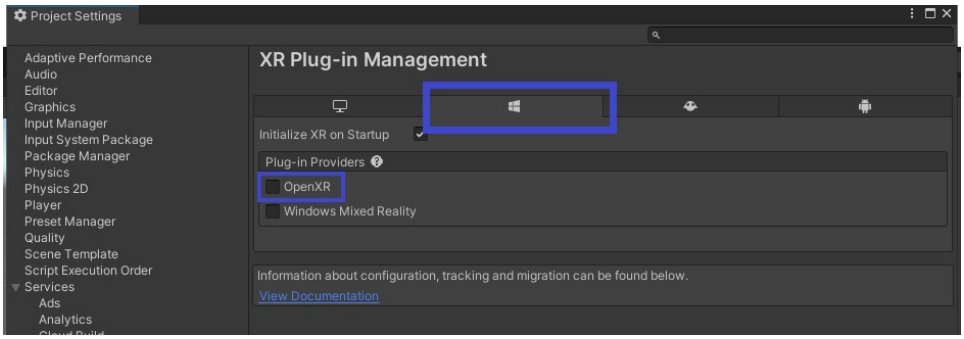
You can use the Unity Package Manager to install it in case it was not automatically installed.



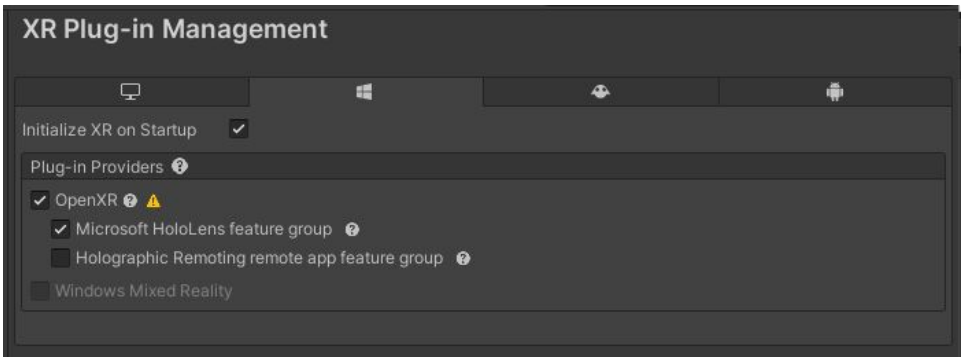
3. Set up the Mixed Reality OpenXR Plugin using the menu option **MAGES/Third Party SDK Manager/Install/Mixed Reality OpenXR Plugin**

unity/getting_started/step_by_step/img/holo2setup1.jpg

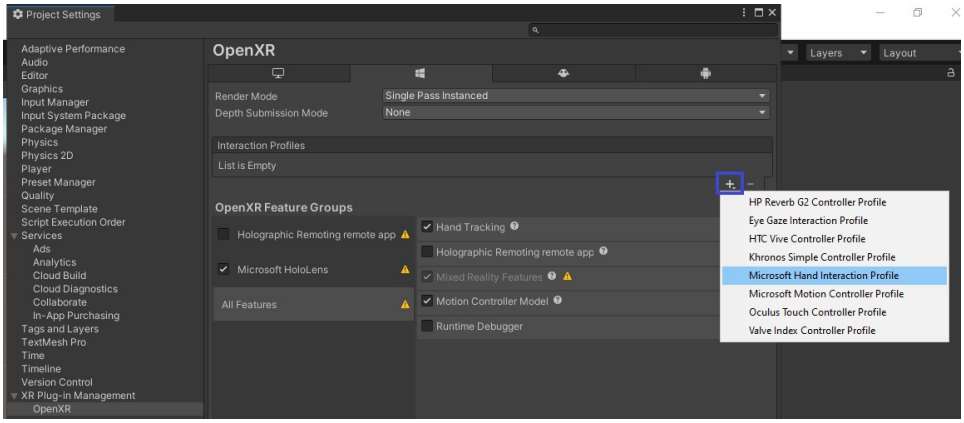
4. **Project Settings/XR Plug-in Management** will open. Click the UWP tab, and tick OpenXR to enable OpenXR Support for the UWP platform.



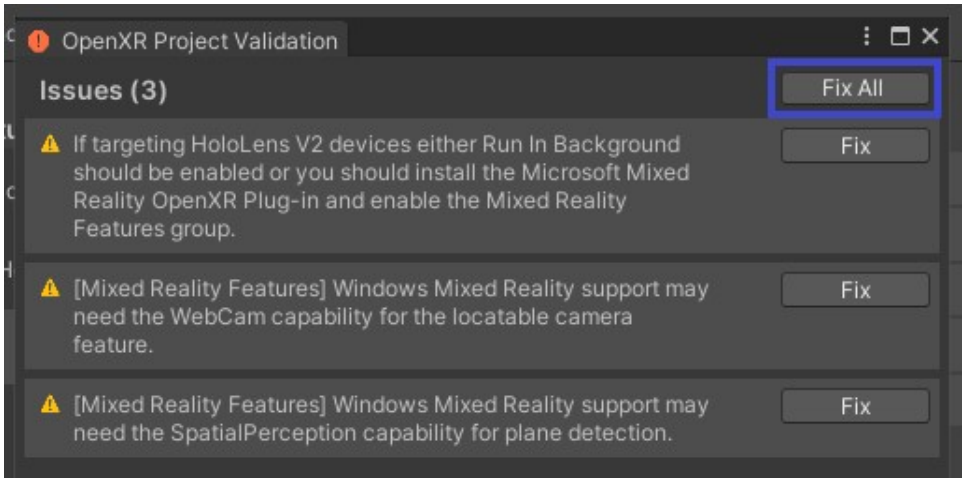
5. After enabling OpenXR support, two more sub-options will appear. Tick "Microsoft HoloLens feature group".



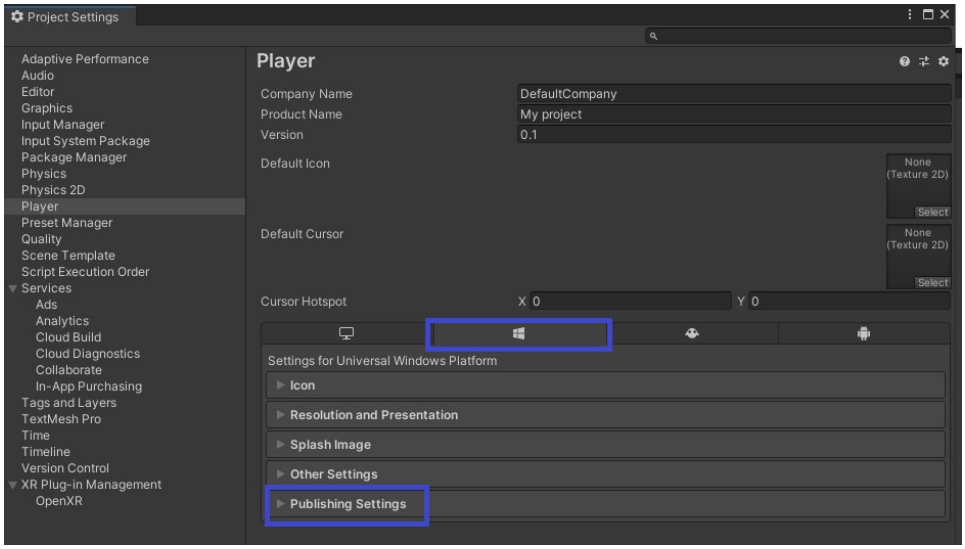
6. Go to **Project Settings/XR Plug-in Management/OpenXR** and select the UWP tab. Add the Microsoft Hand interaction profile.

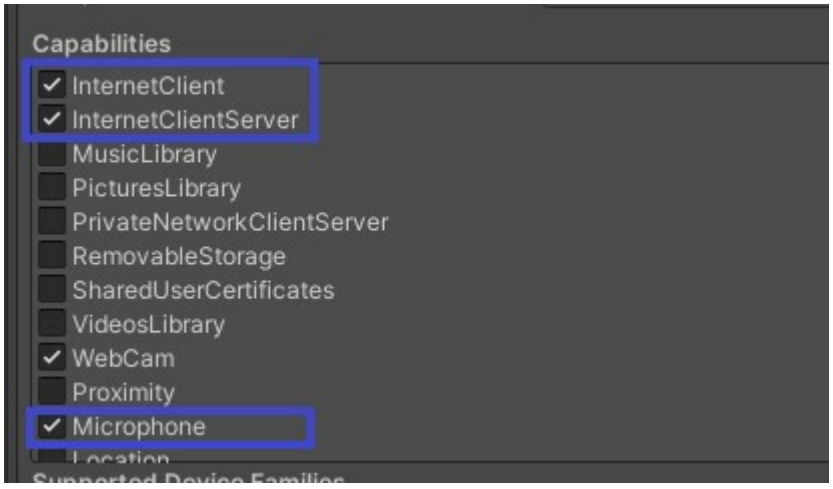


7. Click any of the warning icons, the OpenXR Project Validation window will open. Press Fix All.



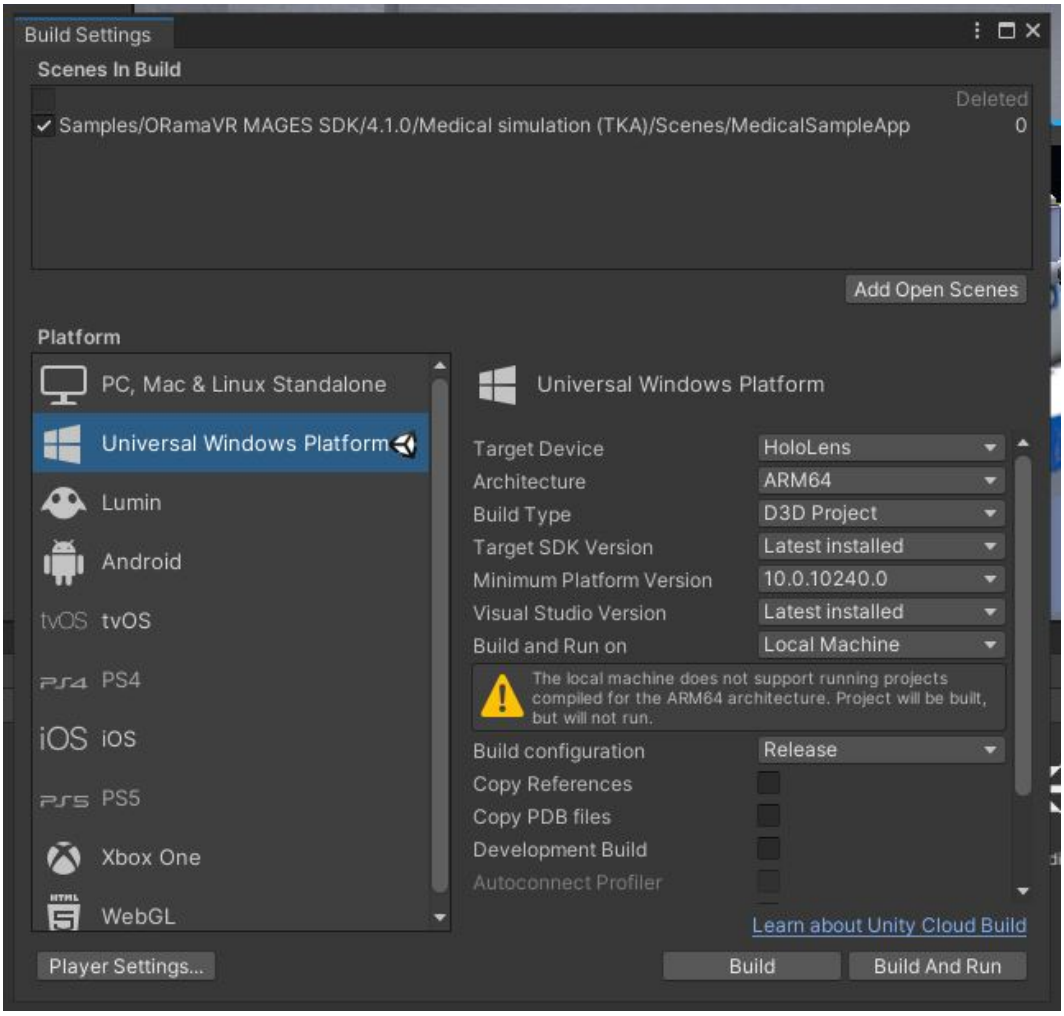
8. In order to enable cloud and multiplayer features, the appropriate permissions must be set. Navigate to **Project Settings/Player** and click on the UWP tab. Scroll to Capabilities and check the Internet, InternetClientServer and Microphone permissions.





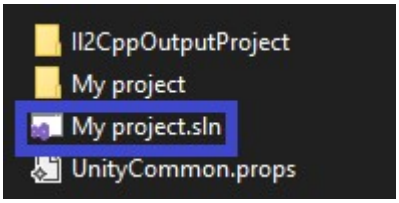
9. Spawn the HoloLens 2 Camera by navigating to **MAGES/Cameras/AR/HoloLens 2**.

10. Open Build Settings and make sure all options look like the following image. Press Build and choose the path you want your build to be stored in. The result of the build is not an executable, but a Visual Studio Solution that can be compiled and deployed on the HoloLens 2.

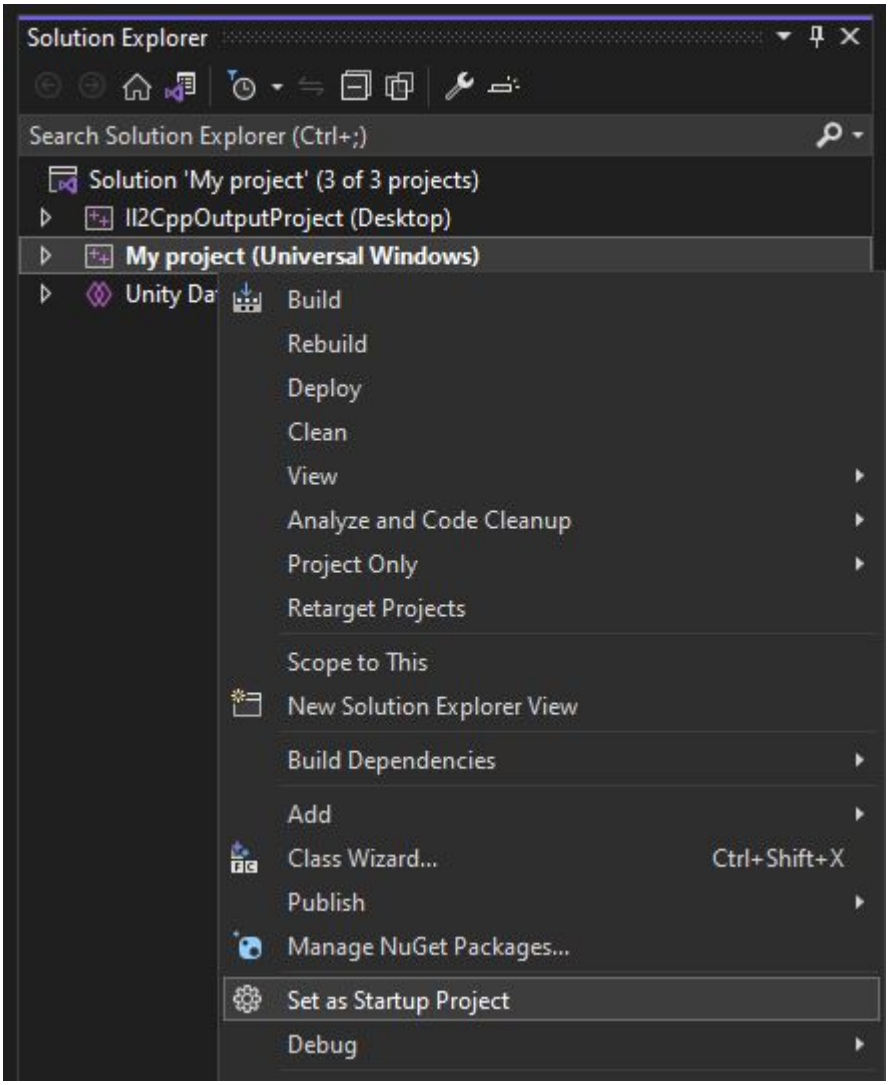


> **Warning:** Build and run is not supported when targeting USB Devices.

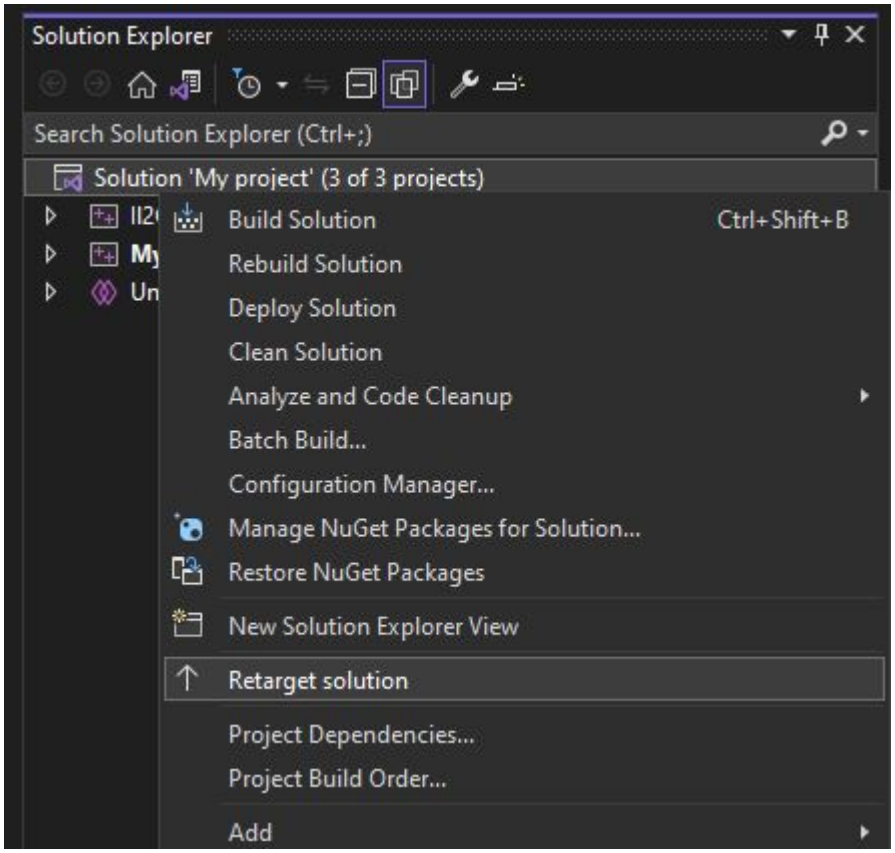
11. In the build folder, open the Visual Studio Solution using Visual Studio 2019 or Later.



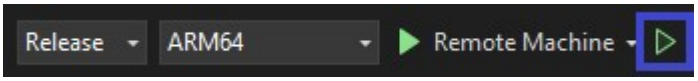
12. Set the Project with your Project's name as the Start up project.



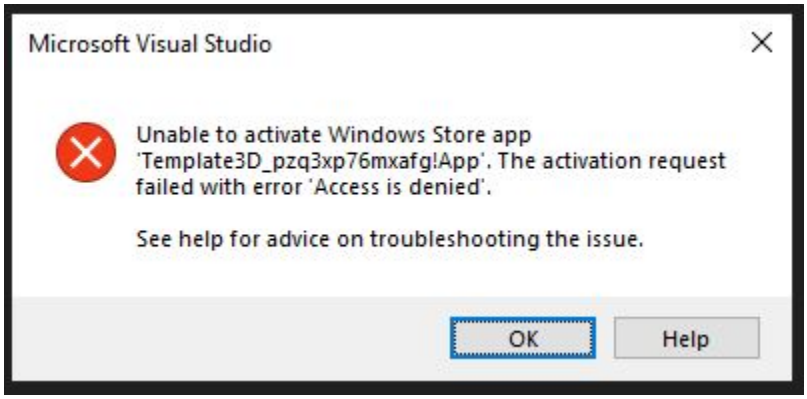
13. Depending on your Visual Studio version, HoloLens 2 operating system version, downloaded SDKs and Visual Studio components, you may have to retarget the solution to fit your needs.



14. Set Build Settings to Release, ARM64 and target the Device. After connecting your headset via USB and making sure it is turned on, press start without debugging or `Ctrl+F5`.



Warning: If the HoloLens 2 Headset goes into sleep mode while the build is deploying, this error (or similar) might appear. Despite the error, the deployment was successful. You can find the built app in the All Apps menu in your device.



Lumin OS (Magic Leap)

Magic Leap

In order to deploy an application utilizing the MAGES SDK on the Magic Leap Headset, it is mandatory to switch your target platform to Lumin. This is done through **File/Build Settings** or `Ctrl+Shift+B`. If you can not see this option, please make sure you have included the Lumin OS Build tools with your Unity installation.

Click on Lumin and finally click the Switch Platform button on the lower-right side of the current window.

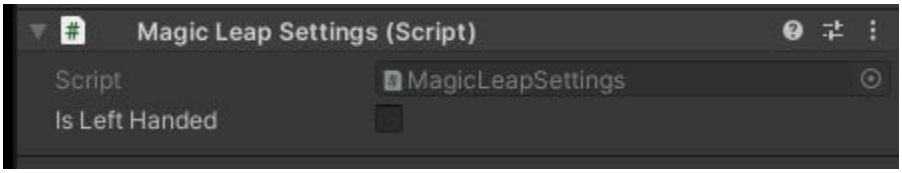
Note: Switching between platforms induces significant compilation times. It is best that you develop on the platform you are targeting from the start.

Warning: In order to deploy applications to the Hardware, Magic Leap's companion app The Lab is required.

To deploy your application to the Magic Leap Headset, follow the steps below:

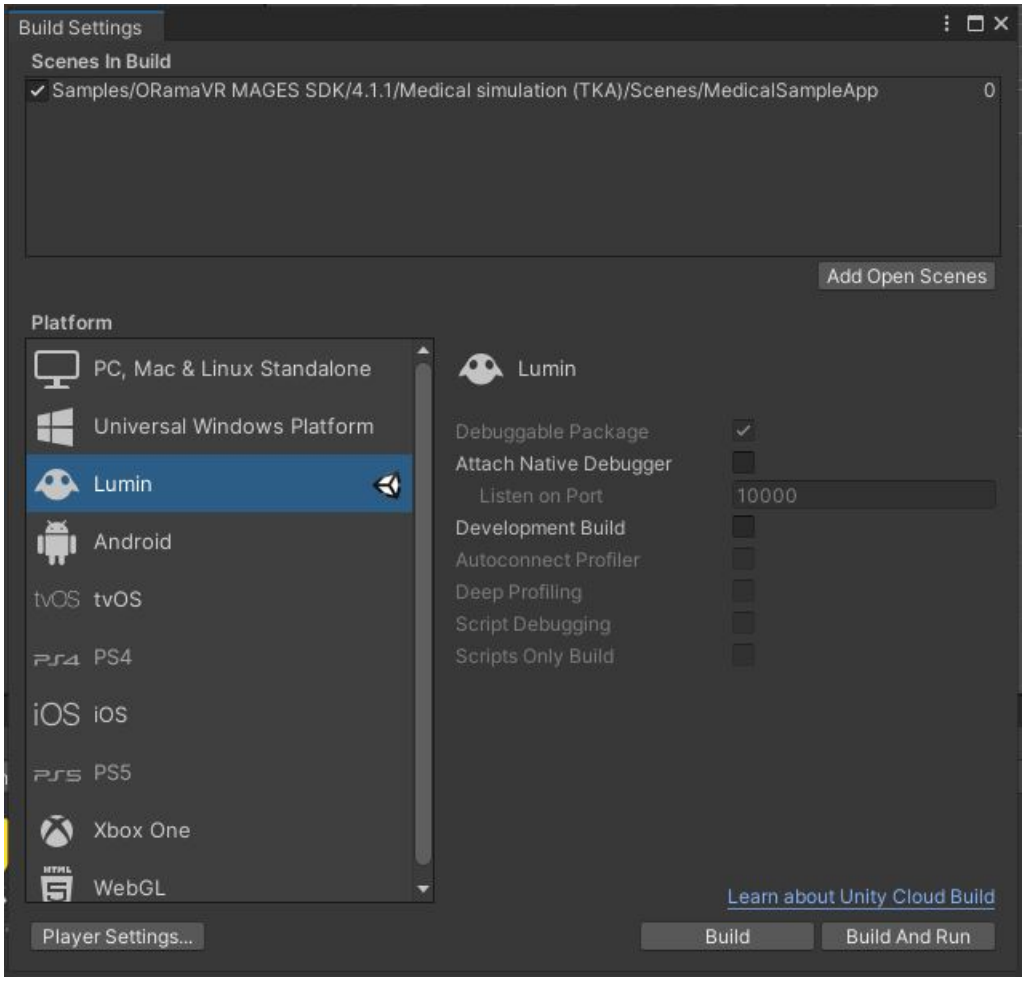
1. Use the Recommended method (Install via the Unity Asset Store) to install the latest Magic Leap SDK using this guide.
2. Follow the prompts in Unity after the asset is imported to complete the Magic Leap SDK installation, as also described in the guide.
3. Spawn the Magic Leap Camera by navigating to **MAGES/Cameras/AR/Magic Leap**.

Note: The Magic Leap camera can be used with only one hand. You can control which hand is holding the Magic Leap remote using the inspector. Select the Magic Leap XR Rig and find the Magic Leap Settings component.

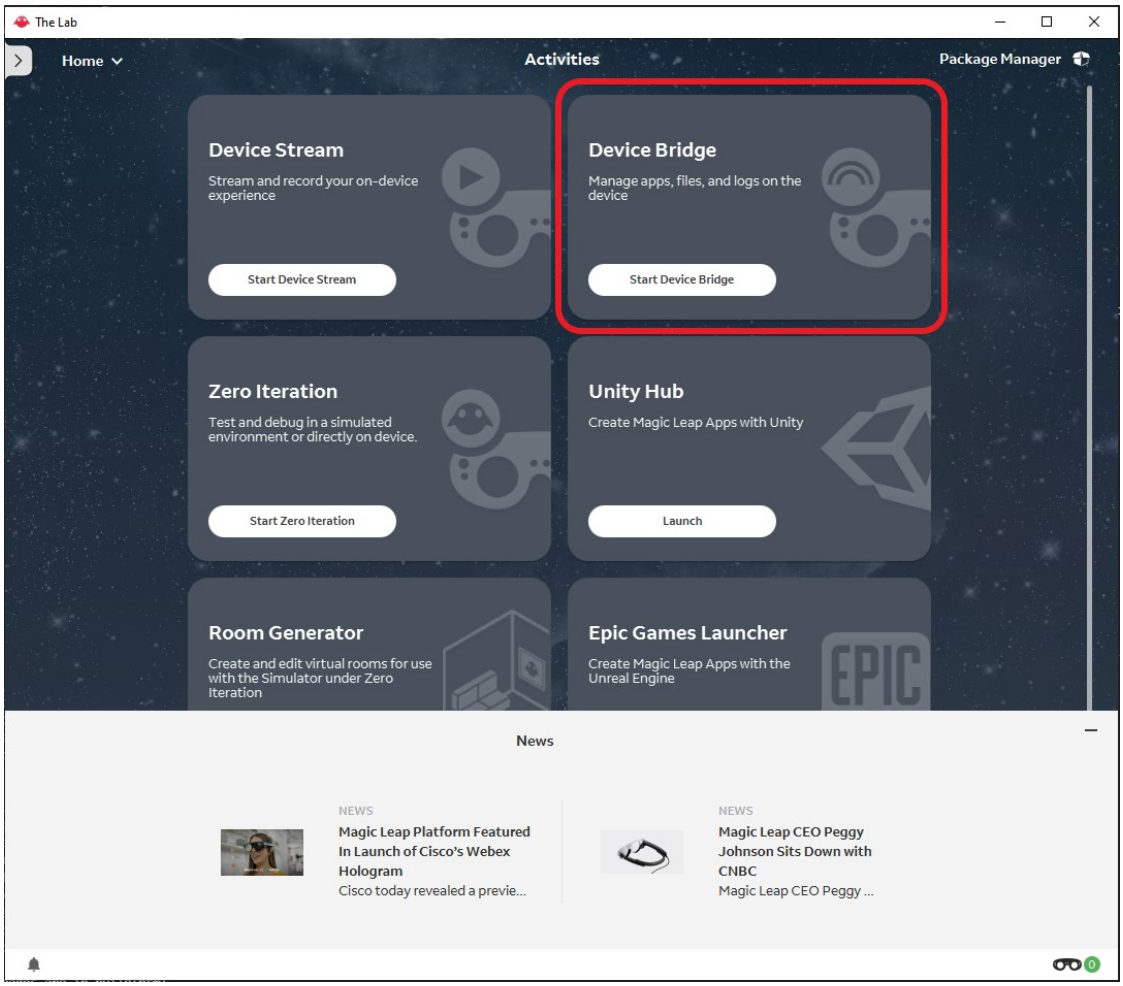


---

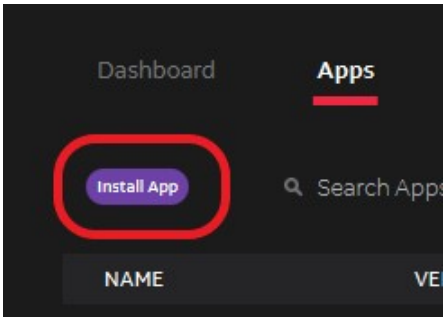
4. Open Build Settings and press build. This will open a file explorer dialogue window in which you can pick the directory where the executable .mpk file will be stored. After choosing the Build folder, Unity will build the application into the executable file.



5. When the build is completed, use Magic Leap's The Lab application to install the App on the Device via Device bridge.



6. After connecting to your device, navigate to Apps, and click Install App. Locate the .mpk file you created in Step 4.



7. You can start the application either via the Device Bridge, or on the headset itself via the app menu.

4.2 First Insert Action

4.2.1 Introduction

Once you finished following the *Step by step* guide you are ready to start developing your first insert action with MAGES SDK.

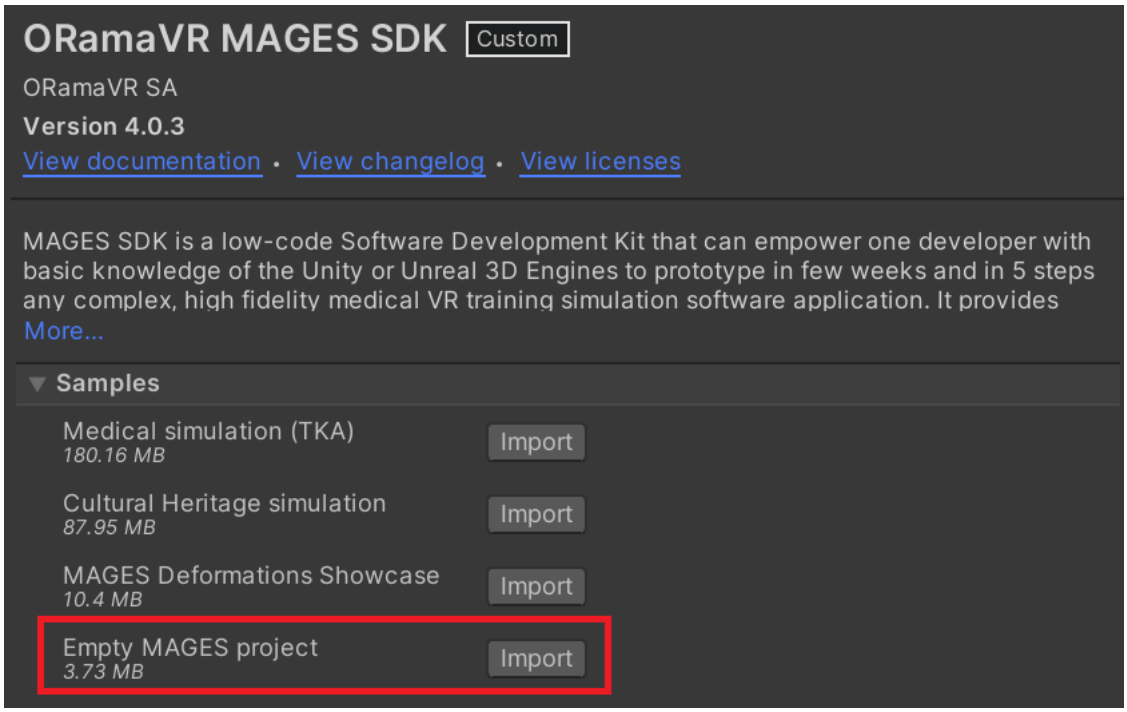
Insert Action is referring to a specific type of Action that a user has to insert an object to a specific position in order to complete it.

Note: There are **two ways** to create an insert action. One can either start **writing the script themselves**, or **create it automatically** by setting up the required parameters from the MAGES menu.

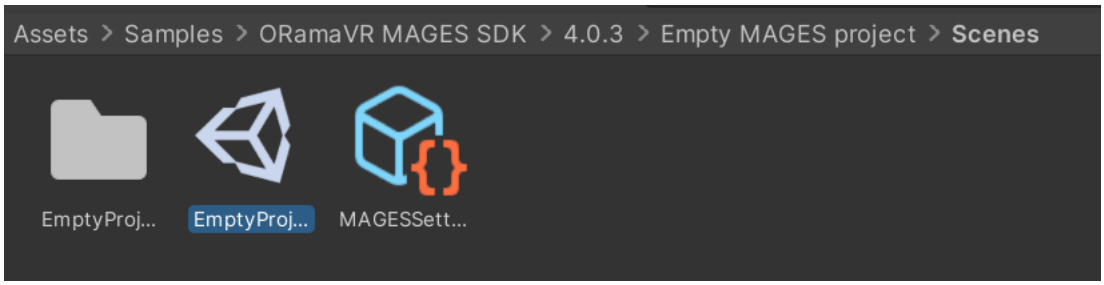
4.2.2 Download the Empty MAGES Project Sample

For this tutorial you will need the Empty MAGES Project Sample. It is an empty project with all the required components for you to start. It also has already implemented the first and last Actions of the operation.

To download the Empty project navigate to the Package manager and under the MAGES SDK package click to import the "Empty MAGES Project"



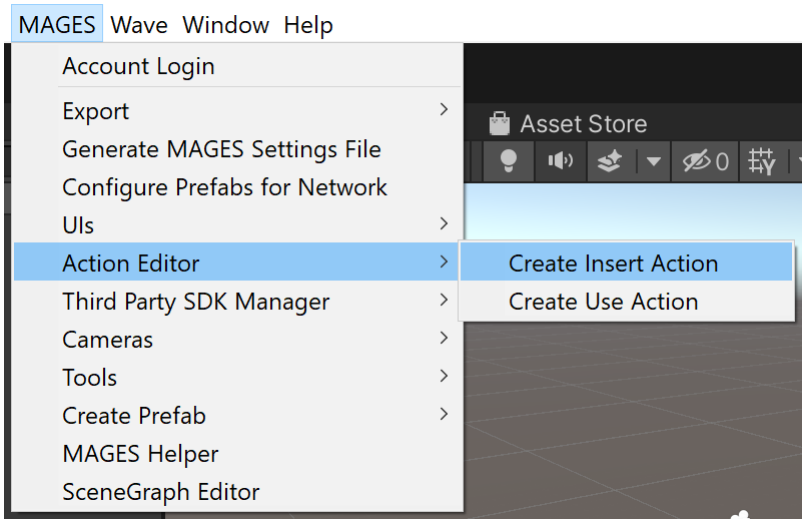
Then, open the EmptyProject scene:



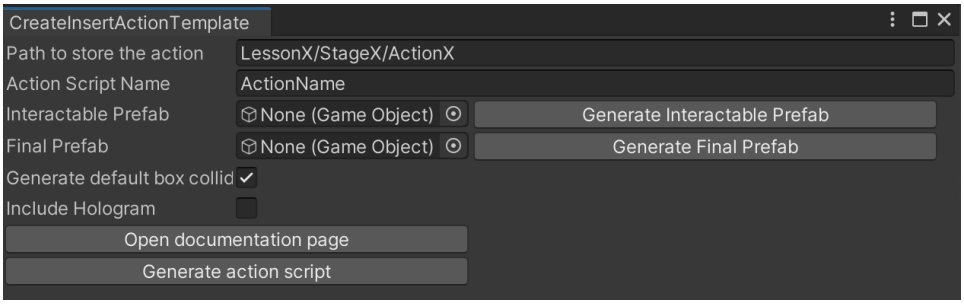
4.2.3 Automatic Generation

To create your first insert action, **without writing any code**, please follow the following steps:

1. Navigate to MAGES menu –> Action Editor –> Insert Action.



2. The following window will appear:



Each of the fields are explained below:

a. **Path to store the action**: This is the path where your prefabs will be saved. The automatically generated script will also be saved in the corresponding path under `Assets/MAGEScomp/Operation/ActionScripts/...` (Please refer to *Project File System* for more information regarding the file system.).

Note: It is recommended that the variables X, Y and Z in the path `LessonX/StageY/ActionZ/` correspond to the IDs of the current lesson, stage and action. For example, if the Action we created is the first Action of the second Lesson we will name the path: Lesson1/Stage0/Action0. However, this is not mandatory, you can name the folders as you want. The only restriction is to have a "Lesson-Prefabs" folder under the "Resources" folder as the path to load the prefabs starts from there.

---

b. **Action Script Name**: This is the name of the automatically generated script. It is recommended to name it "ActionNameAction" in which you can replace the "Action-Name" part with the actual name of your action. This naming rule is recommended but not mandatory.

c. **Interactable Prefab**: This is an empty gameobject field. In this field, the prefab of your choice must be added. This is the prefab that you would like the user to pick up in order to place it somewhere.

d. **Final Prefab**: This is an empty gameobject field. In this field, the same prefab that you added in the **Interactable Prefab** field must be added, but now it should be located in the correct position and location. In other words, this will represent the target prefab in the final/correct position that the end-user must place it.

Note: You can save this position/rotation on the final prefab through the Unity Editor. After having saved it, you can simply drag and drop it in the **Final Prefab** field.

---

e. **Generate Default Box Collider**: This is a checkbox. If it is enabled, a default box collider will be created for the **Interactable Prefab**.

f. **Include Hologram**: This is a checkbox. If it is enabled, a hologram will also be created/include in the action.

Note: The hologram is a green see-through version of the **Interactable Prefab**, located in the position of the **Final Prefab**, indicating the correct position/rotation of the **Interactable Prefab** in order to aid the user.

---

3. To create a simple insert action, to act as the first action in your application (after the operation start action), kindly follow the next steps:

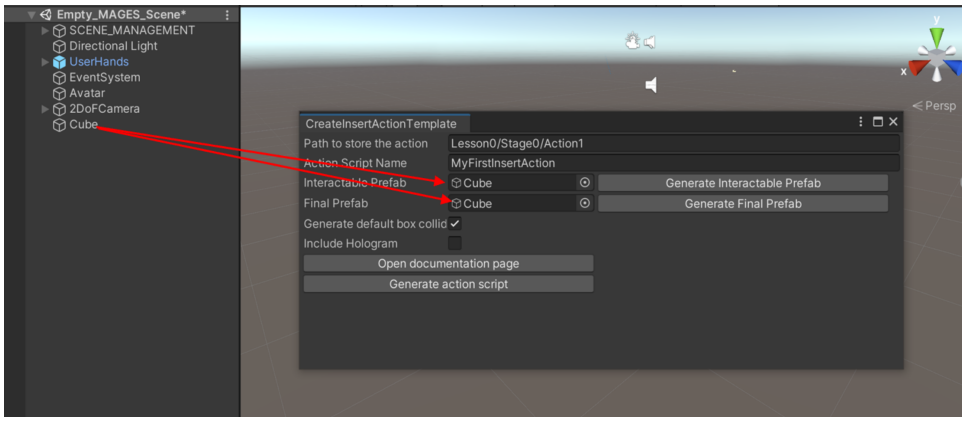
a. Locate and open the EmptyProject.unity scene.

b. From the **Create Insert Action** window, change the **Path to store the action** to `Lesson0/Stage0/Action1/`. This will create the folders if they are not already there. As we mentioned, we are following this file structure since this is the second Action of the operation (the first one is the OperationStart)

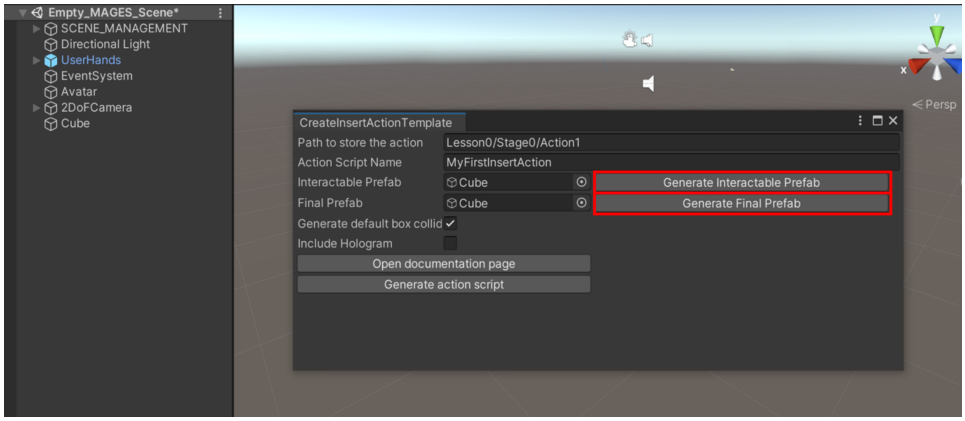
c. Set the **Action Script Name** to MyFirstInsertAction, or whichever name you prefer.

d. Create a 3D cube, through the Unity menu.

e. Drag and drop the reference of this cube to the **Interactable Prefab** and **Final Prefab** fields.

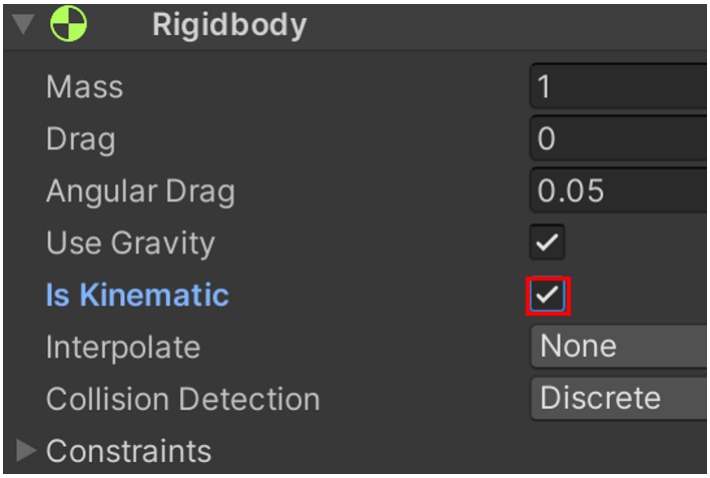


f. Click the **Generate Interactable Prefab** and **Generate Final Prefab** buttons.

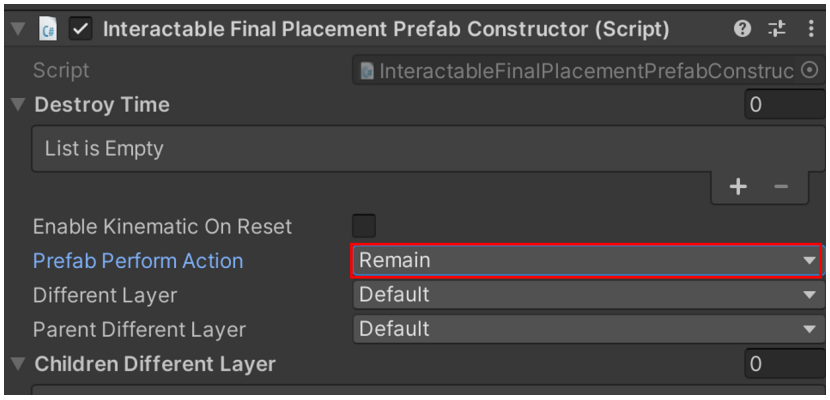


g. Through the Unity Scene, move the newly spawned objects (namely New_Interactable_Prefab and New_Final_Placement_Prefab) to the position you desire. For this example, move the **New_Interactable_Prefab** to (-19, 1.5, -16) and the **New_Final_Placement_Prefab** to (-19, 1.5, -12).

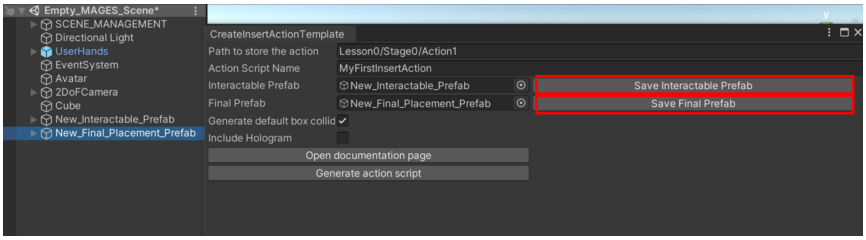
> **Warning:** The **New_Interactable_Prefab** prefab has gravity enabled by default. This will cause the cube to fall continuously in this example. It is recommended to click on the **New_Interactable_Prefab** and locate the **RigidBody** component in the Inspector. Then enable the **Is Kinematic** checkbox.



Warning: The **New_Final_Placement_Prefab** prefab is set to be destroyed by default during the end of the action. In order to prevent this from happening click on the **New_Final_Placement_Prefab** and locate the **Interactable Final Placement Prefab Constructor** component in the Inspector. Then change the **Prefab Perform Action** from **Destroy** to **Remain**.

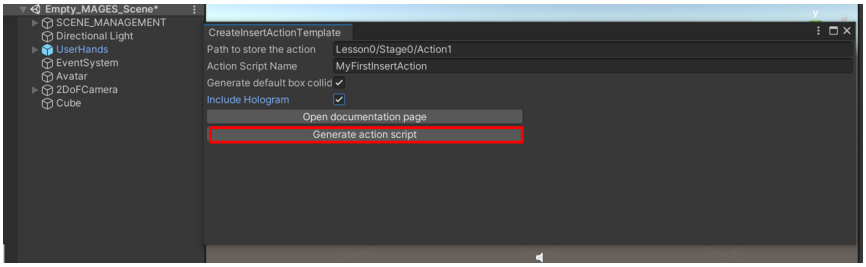


h. Click the **Save Interactable Prefab** and **Save Final Prefab** buttons.



i. Make sure that the **Include Hologram** checkbox is checked.

j. Finally click the **Generate Action Script** button to create the script.



Note: Your script will reside under `MAGES/Operation/ActionScripts/Lesson0/Stage0/Action1`

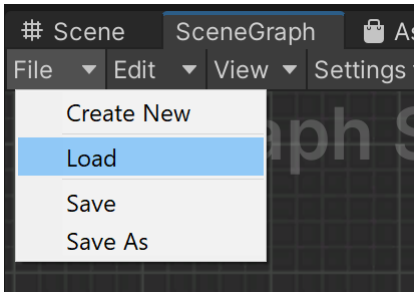
Note: Your prefabs will reside under `Resources/LessonPrefabs/Lesson0/Stage0/Action1`

4.2.4 Adding your new action to your application

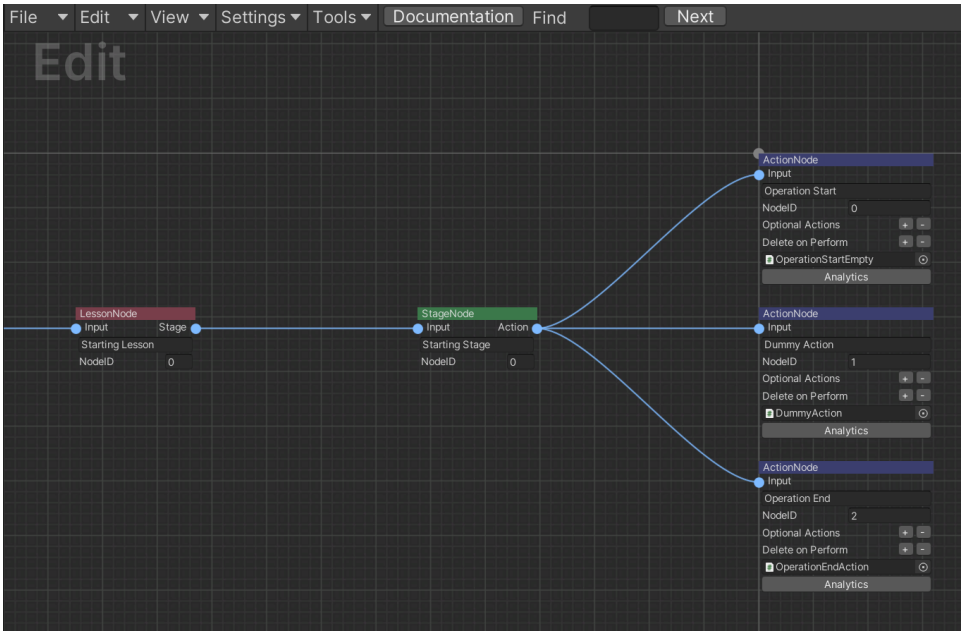
Once the necessary prefabs and script have been generated, the only thing that remains for the action to be operational is to actually be included in your application.

The empty MAGES scene contains a dummy action (i.e. an action that is empty and does nothing, which follows after the operation start action). We will now swap this action with your newly created one.

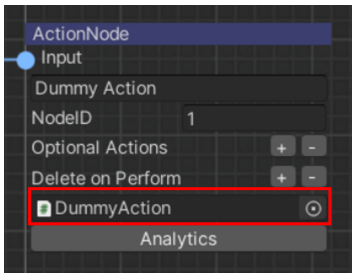
1. From the MAGES menu click the **Scenegraph Editor** option.
2. A new window will open. From this window, click the **File** button and then **Load**.



3. Navigate to the following path `Assets\Samples\ORamaVR MAGES\4.0.0\Sample App\Resources\Storyboard\platform` and open the file **EmptyProject.xml**.
4. Once the xml loads, you will see a graph like this.



5. Locate the Dummy Action node and click on the script slot to change the script.



6. From the explorer window, locate the **MyFirstInsertAction.cs** script and click Open (normally it will be in MAGES/Operation/ActionScripts/Lesson0/Stage0/Action1).
7. Click the **File** button and then Save.

4.2.5 Manual Generation

The script and the prefabs can always be created manually as well. In that case, kindly read the guide located here: *Insert Action*.

4.2.6 Destop3D camera Tutorial

The Desktop3D camera is used to run a MAGES application in Desktop 3D mode, without the need of any HMD or controllers.

In this mode, the character is controlled through the keyboard and/or mouse/trackpad.

A UI is available in this mode, by pressing the space bar. This will open the menu of Desktop3D which gives the user the ability to move, rotate, lock the hands. Below, you can see the Desktop3D menu on action.

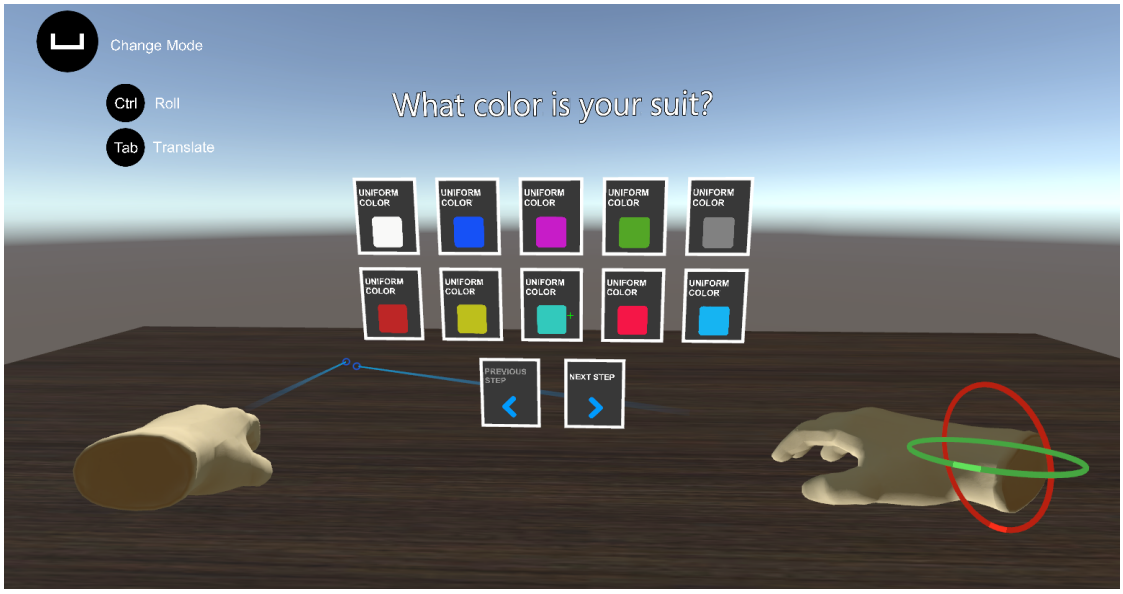
In the following table, a detailed explanation can be found, regarding each button of the Desktop 3D camera.

Button	Explanation
W or Up Arrow	Move forwards
S or Back Arrow	Move backwards
A or Left Arrow	Move left
D or Right Arrow	Move right
Q	Move downwards
E	Move upwards
F1	Toggle Body move mode
F2	Toggle Right Hand move mode
F3	Toggle Left Hand move mode
Tab	(Usually followed after pressing F1 or F2), enables the hand rotation mode with the use of mouse/trackpad
G	(Only after F1), enables/disables the cursor and freezes the camera rotation.
Left Mouse/Trackpad Click	Hand trigger on the hand targeted by F2/F3
Right Mouse/Trackpad Click	THand grip on the hand targeted by F2/F3
Ctrl	Switch hand plane movement (when F2 or F3 enabled)
Ctrl and Left Click or Right Click	Sends the Left or Right hand to the direction of the crosshair

You can utilize the Desktop 3D camera controller in order to run and execute the first insert action you created before. To do so, please follow the instructions below:

1. Ensure that the EmptyProject scene is open and click the “Play” button of the Unity Editor.

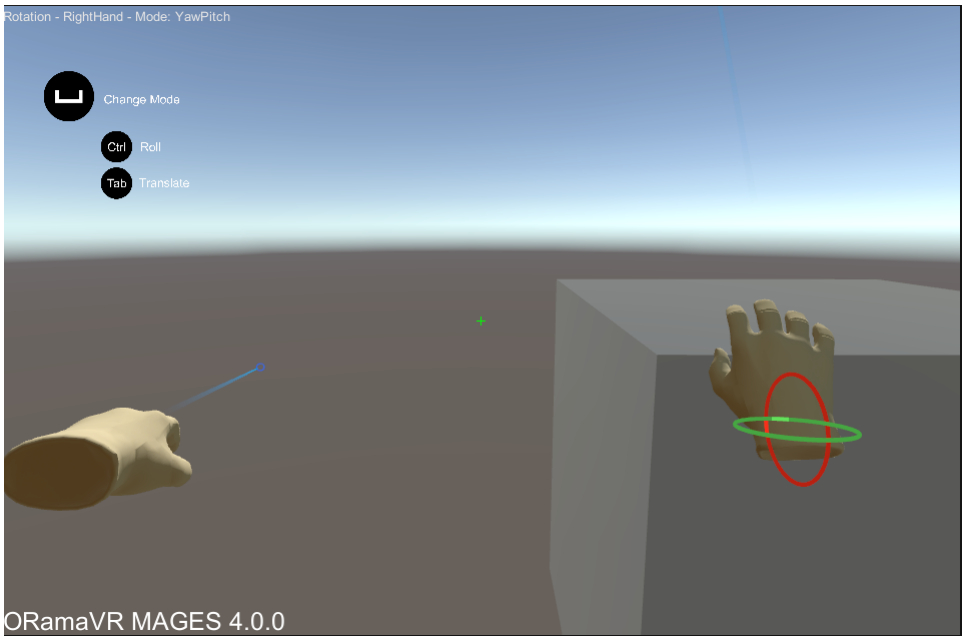
2. The first time you will open the application you will be prompted with the avatar customization UI to select the appearance of your avatar.



3. You will be presented with a menu like the following. This is the **Operation Start** menu.

You can find more information about the in-game UIs *here* (e.g the operation start buttons)

4. There are two ways to continue in this point. Either by pointing one of the virtual hands to the **Single Player** button and clicking with the left mouse button, or by pressing the **X** button on the keyboard. It is recommended to continue with the latter for now.
5. In this point, the **New_Interactable_Prefab** you set up in the tutorial above, will be present in the scene along with a hologram.
6. Activate the virtual right hand in order to be able to pick up objects. To do so, press F2+Tab from the keyboard.
7. Move close to the cube using the W/A/S/D/E/Q buttons. You will know that you are in the correct position when an outline appears around the cube like in the image below.



8. Click using the left mouse button (or trackpad) and hold it. The right virtual hand will hold the cube, as long as you hold the button.

> **Note:** In case you accidentally drop the cube, you can restart the action by moving to the next one with the **X** button and then to the previous one again with the **Z** button.

9. While holding the left mouse button, use the W/A/S/D/E/Q keys again to move next to the green hologram.
10. Try to align the cube with the hologram, using the W/A/S/D/E/Q buttons. Once you do this correctly, the cube will be inserted in the position specified by the hologram.

4.2.7 Building the application

This section will demonstrate how to create an executable with your newly created insert action (in this case for the Windows platform.)

1. Copy the **EmptyProject.xml** file, which is located under `Assets\Samples\ORamaVR MAGES\4.0.0\Sample App\Resources\Storyboard\platform`, to `{User}\Documents\ORamaVR\Story\platform`.

> **Note:** If this path does not exist in your Documents folder, create it manually.

2. Through Unity menu, navigate to **File –> Build Settings**.
3. Make sure that the **EmptyProject** is present in the list of scenes in Build Settings. If not, simply add it by click the **Add Open Scenes** button.

4. Click the **Build** button and specify the directory where you would like the executable file to be created.
5. Once the building procedure is finished, navigate to the directory you specified in step 4, and double click on the executable to start the application.

Note: For more information regarding building an application with MAGES™ SDK and all the supported platforms, please visit *Build Instructions*.

CHAPTER

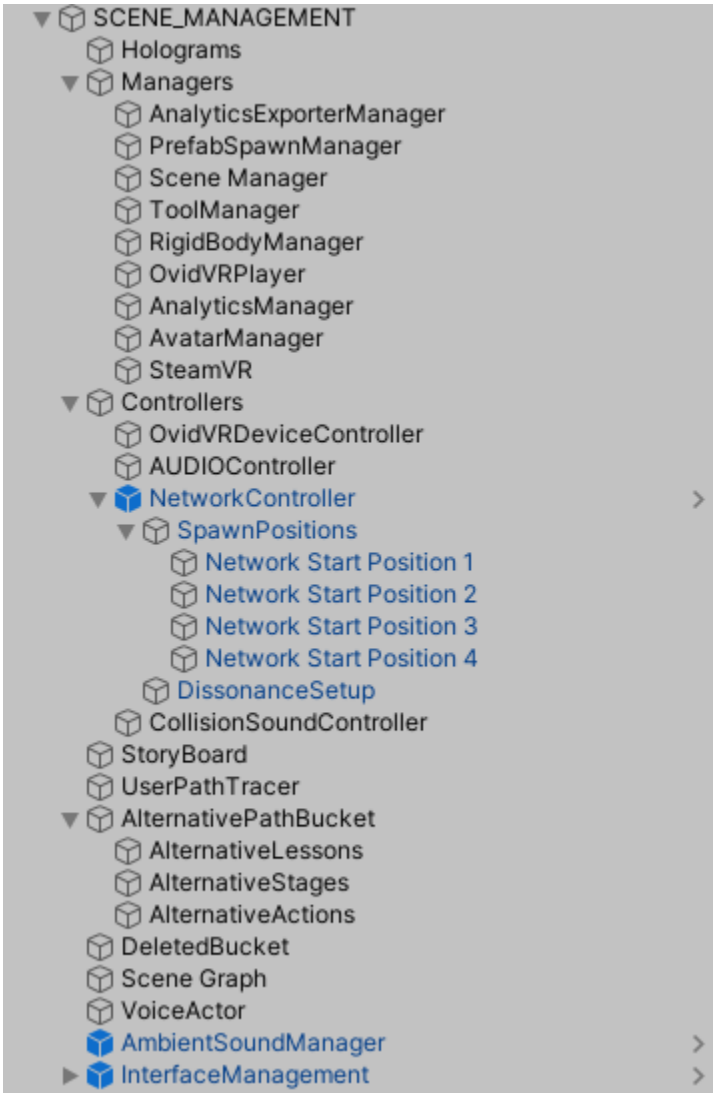
FIVE

MANUAL

5.1 Unity Scene

5.1.1 Scene Management

In order for a Unity project to run with the MAGES SDK, it is essential to create inside the Unity scene a (first in order) gameobject for the project management where all components will be located.



Managers

AnalyticsExporterManager (Required): Adds the Analytics Exporter.

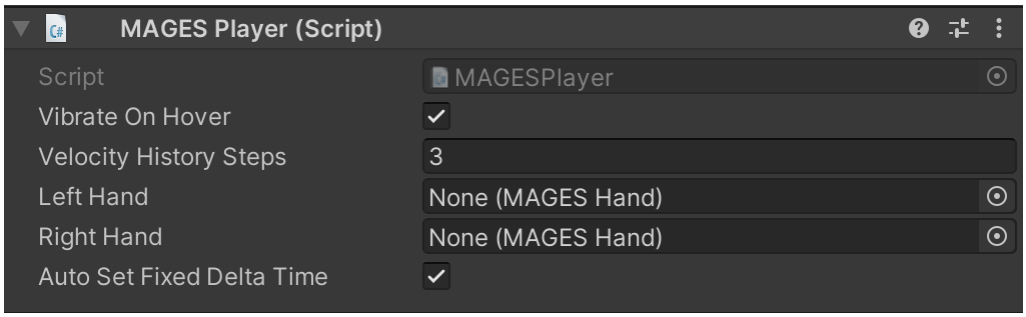
PrefabSpawnManager (Required): Adds singleton-script Prefab Spawn Manager. The developer can choose if needed to preload all assets by enabling the corresponding variable on this script.

Scene Manager (Required): Adds the singleton-scripts Event Manager and Update Manager.

ToolManager (Required): Adds singleton-script Tools Manager.

RigidBodyManager (Optional): Adds Rigidbody Animation Controller script.

MAGESPlayer (Required): Adds the similarly named script with the variables selected in Inspector as below.



AnalyticsManager(Required): Adds the manager responsible for generating the session analytics.

AvatarManager(Optional): Adds the manager responsible for the generation and management of avatars.

SteamVR Adds the manager responsible for initializing SteamVR.

Controllers

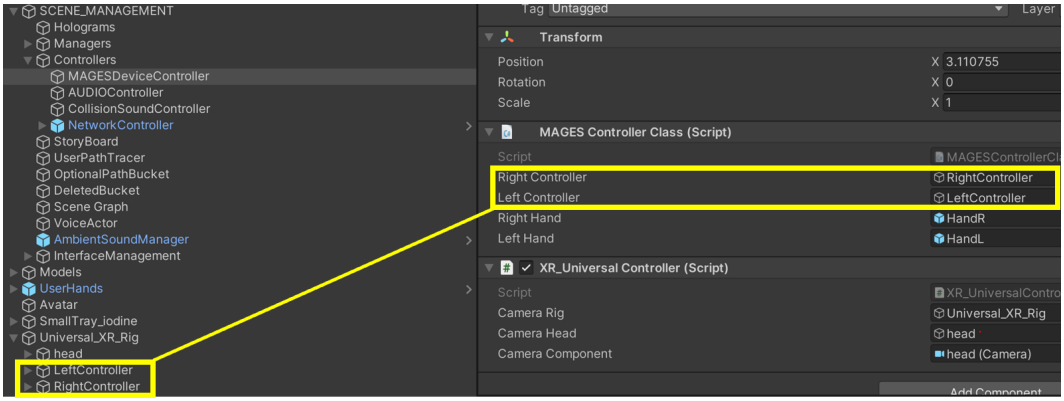
In the Sample App provided, on this particular gameobject Keyboard Controller script is attached. It is there purely for convenience purposes. It contains all the keyboard inputs, except the camera translation buttons located inside CameraRingInputController.

MAGESDeviceController (Required) Adds the Controller Load script as well as the MAGES Controller Class. In the second script place the two controllers found inside the Scene Camera.

i. **Desktop:** (VR Camera/[CameraRig]/Controller (left) & Controller (right))

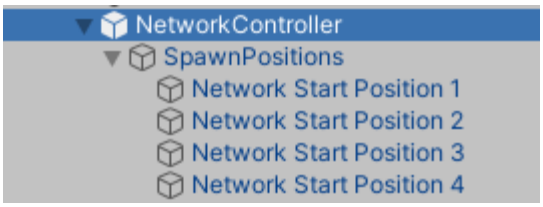
ii. **Oculus Quest:** (OVRPlayerController/OVRCameraRig/TrackingSpace/LeftHandAnchor & RightHandAnchor)

iii. **Vive Focus Plus:** (WaveVR/Generic_MC_L & Generic_MC_R)



AUDIOController (Optional) Creates an Audio Source with the Spatial Blend value set to 0 and attach the Audio Controller script.

NetworkController (Required) Adds the Network Controller Photon, Photon Network Metrics and Player Numbering scripts. Also, inside the Network controller four gameobjects must be created as children of this one containing the script Network Start Position.

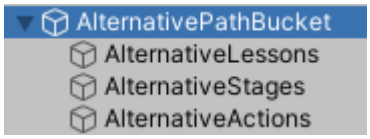


Rest of Gameobjects

StoryBoard (Required) Adds StoryBoard script.

UserPathTracer (Required) Adds UserPathTracer script.

AlternativePathBucket (Required) This is the responsible gameobject for all the alternative LSAs (Lessons Stages Actions). Attach the Alternative path and Alternative Path Importer scripts and add three empty gameobjects as its children.



DeletedBucket (Required) Nothing needs to be attached. This is handled from the Alternative Path (careful, it must not be an Alternative path's child!)

Scene Graph (Required) Add SceneGraph singleton-script here. In this gameobject, when the projects start, all LSAs will be created as children of this particular gameobject.

VoiceActor (Optional) Has the VoiceActor and VoiceActorImporter components that manage the VoiceActor audioclips.

AmbientSoundManager (Required) A manager responsible for the ambient music during the login phase or gameplay.

InterfaceManagement (Required) Manager responsible for spawning and destruction of UIs.

5.1.2 XR Camera

Camera as a GameObject

From MAGES 3.3.0 we use the Universal XR camera. A single camera instance to manage all the XR devices.

Camera Tags & Layers

Controller (left) & (right) [Desktop] must both be in the layer UserHands and the first one must have the tag LeftPalm and the second one the RightHand. This involves all their children too. Above them the CameraRig [Desktop], OVRCameraRig [Quest], WaveVR [Vive Focus Plus] where the collider for the camera is located, the layer CameraRig must be applied. Only for that specific gameobject and not its children.

UserHands Gameobject

On the children-gameObjects HandL & HandR, it is attached the script Animation Controller. As the name implies it controls all the animations for the hand depending on what button the user is pressing (e.g. pressing the trigger button, it plays the animation for the index finger). Also, an animator component is attached with the hand's animation.

Universal XR Camera Setup

Universal_XR_Rig This object contains the **Character Controller** component. This is a Capsule Collider used to restrict the camera movement inside the scene. As explained before this specific gameobject is the only child in the VRCamera that has the layer CameraRig. The developers should set up in the Project Settings → Physics what layers should be visible by this collider.

The `UniversalCameraMovement` script is responsible for the movement within the VR using the controllers.

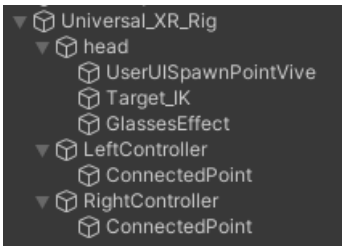
Additionally, the `KeyboardMovement` script enables movement from the keyboard as well.

head

a. **UserUISpawnPoint:** A gameObject that only counts as a reference for UI positioning. It faces the Camera Eye with a slight offset in the forward axis. Its purpose is to be the spawn point for every UI that gets spawned.

b. **Target_IK:** A gameObject that serves as the target point of avatars (in case they are used).

LeftController and RightController Contains the players virtual hands. The **ConnectedPoint** gameObjects reflecting the position of the controllers.



Universal XR Camera Setup with Microsoft HoloLens 2

When the Universal XR Camera is set up for use with the HoloLens 2, an aditional "HoloLens2Input" GameObject is added to the Universal XR Rig.

`unity/manual/unity_scene/img/holo2camera.png`

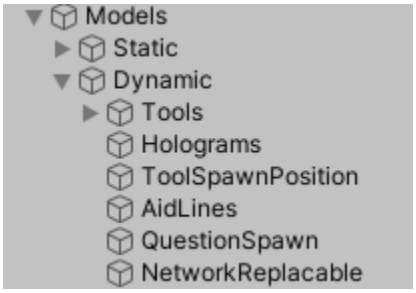
This aditional component is responsible for managing input from the hands as well as movement input.

Hover Collider This gameobject is used for detecting when a user's hand is close to the movement button, in order to make it visible.

Click Collider When a hand is inside this collider, the player will move towards the direction he is facing.

5.1.3 Models

For better scene management it is advised all models (mostly in scene - permanent models) to be grouped under one gameobject in separate children depending on their category.



Static

In this gameobject any static model should be placed to be easily recognized from the developer. Static objects are always treated differently (e.g. static lighting).

Dynamic

Tools It is advised to have placed all the developer defined tools in the Dynamic part of Models. More specifically create a Tools/ToolSet parent-child gameobjects and in that have all tools. See image above. The ToolsManager.dll expects the gameobject to be created as explained.

Holograms M.A.G.E.S. SDK provides mechanics to spawn at each action holograms. These are gameobjects similar to the ones needed for the Action completion, but with the holographic shader and their purpose is to point to a key place for the action to commence/complete. This way of creating these "visual aids" is optional serving the role of an example. See Sample App for better understanding.

ToolSpawnPosition The tools to know that they are in their default position, they observe if they have collided with a collider called ToolSpawnPosition. This condition is checked inside the ToolsManager.dll. Make sure the collider has a layer observable to the Tools and ToolsOFF layer and that the name of the gameobject is exactly the described one.

QuestionSpawn Under this gameobject some random Quiz UIs will be spawned containing questions about the current Scene.

5.2 Unity Project Settings

5.2.1 Tags & Layers

MAGES SDK expects that some specific tags and layers are created inside the Unity Scene in order to use them in specific gameobjects and prefabs. The tags are mostly needed for quick search in the Scene for frequently used or requested gameobjects. Specific layers are used for the physics engine to separate layers for collisions or lighting. The developers can add as many others tags or layers as needed.

Tags

1. Bonesaw
2. Scalpel
3. Drill
4. Pliers
5. Mallet
6. Retractor
7. RightHand
8. LeftHand
9. LeftPalm
10. RightPalm
11. FemoralMeasureTool
12. CementGun
13. RightHandIndex
14. LeftHandIndex
15. DrillTower
16. CementSpatula
17. Scissors
18. ToolAttachR
19. ToolAttachL
20. Luer
21. SewingScissors
22. Cauterizer
23. ActionTriggerCollider
24. Cautery
25. Material_Cloth
26. Material_Metal
27. SurgeryRoom
28. Material_Body_or_Plastic

Layers

1. UserHands
2. CameraRig
3. ModelSkin
4. ignore All
5. Tools
6. Lights
7. Mirror
8. SurgeryRoom
9. InternalOrgans
10. ToolsOFF
11. NoTriggerColliderLesson
12. Reflection1
13. TriggerColliderLesson
14. GrabbablePrefabs
15. AvatarLayer
16. Areas

5.3 Project File System

In this section we will introduce the project file system MAGES™ SDK utilizes.

Readers will get familiarized with the existing project structure and incentivized to work towards the same direction.

Take notes, as certain structural elements of the SDK are immutable. In other words, there is a certain structure developers are expected to store their files for the SDK to operate smoothly.

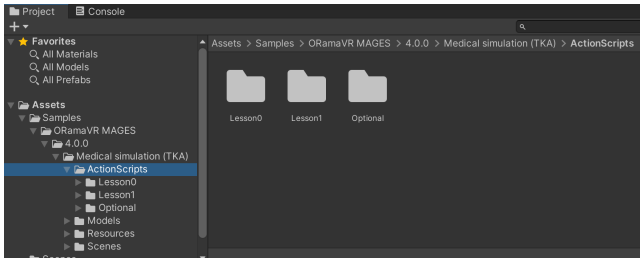
> **Warning:** Failing to follow the structure presented in this section might lead to unexpected behaviors.

5.3.1 General Guideline

Everything operation specific (e.g., Action script, Lesson Prefabs, etc.) is kept under *distinct* structure inside the SDK.

For instance, Medical Simulation (TKA) action scripts are located at:

```
Assets/Samples/ORamaVR MAGES/4.0.0/Medical simulation (TKA)/
ActionScripts/Lesson0/..
```



Warning: By default, action scripts created by MAGES SDK are saved in `Assets/ActionScripts/LessonX/StageY/ActionZ`. Mind this when creating scripts in an imported MAGES SDK sample since there will be a mismatch in these paths.

Note: The same holds for prefabs. By default, they are saved in `Assets/Resources/LessonPrefabs/LessonX/StageY/ActionZ`.

Note: Notice how directories are *created and named* per Lesson and Alternative Lesson, and more specifically per Stage and Action.

You are **strongly advised** to keep the same structure for these files.

5.3.2 Lesson Prefabs

The provided prefab importer from the ORamaVR Platform expects a specific path for all prefabs that will be spawned at runtime.

The reason is for the pre-loader to load faster all elements without parsing on every spawn the whole file path to check whether a specific element exists inside the project.

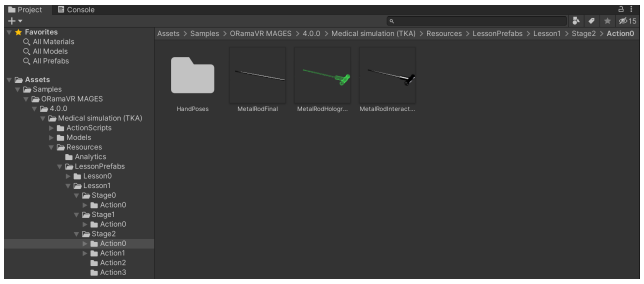
Therefore, you **cannot** alter this structure!

Lesson prefabs, e.g. for Medical Simulation (TKA), are located at:

`Assets/Samples/ORamaVR MAGES/4.0.0/Medical simulation (TKA)/Resources/LessonPrefabs`

Pay attention that the "LessonPrefabs" folder is mandatory and should be under the Resources folder, otherwise the prefabs will not spawn.

However, the prefabs can be saved in any directory under the "LessonPrefabs" folder, the structure LessonX/StageX/ActionX is only to map the scenegraph with the folders.



A clear structure of the folder system regarding the lesson prefabs can be found below:

```
Assets
├── Samples
│   └── ORamaVR MAGES
│       └──  4.0.0
│           ├── Operation1
│           │   ├── ActionScripts
│           │   ├── Models
│           │   ├── Resources
│           │   │   ├── Analytics
│           │   │   ├── LessonPrefabs
│           │   │   │   ├── Lesson0
│           │   │   │   │   ├── Stage0
│           │   │   │   │   │   ├── Action0
│           │   │   │   │   │   │   ├── Prefab1
│           │   │   │   │   │   │   ├── Prefab2
│           │   │   │   │   │   │   ├── ...
│           │   │   │   │   │   │   └── PrefabN
│           │   │   │   │   │   ├── Action1
│           │   │   │   │   │   ├── ...
│           │   │   │   │   │   └── ActionN
│           │   │   │   │   ├── Stage1
│           │   │   │   │   ├── ...
│           │   │   │   │   └── StageN
│           │   │   │   ├── Lesson1
│           │   │   │   ├── ...
│           │   │   │   ├── LessonN
│           │   │   │   └── Optional
│           │   │   └── Storyboard
│           │   └── Scenes
│           ├── Operation2
│           ├── ...
│           └── OperationN
└── Scenes
```

Note: The folder structure is named after the actions they represent. For instance, all the prefabs that have to do with the first action of the second stage of the third lesson for an operation named "Test" should be placed in the following path: `Assets/Samples/ORamaVR MAGES/4.0.0/Test/Resources/LessonPrefabs/Lesson2/Stage1/Action0`

Warning: The folder structure numbering for actions, stages, lessons starts from zero (0).

5.3.3 Action Scripts

For better organization of a MAGES SDK project, it is highly recommended (although not mandatory), the action scripts to follow the same structure as of the lesson prefabs mentioned above.

It is recommended to keep the action scripts files under:

`Assets/Samples/ORamaVR MAGES/4.0.0/(OperationName)/ActionScripts/..`

Inside this folder it is highly recommended keeping the same structure as of the lesson prefabs (meaning LessonX/StageY/ActionZ). There, inside each Action folder, the main action script of that Action should reside.

Note: Helper functions/scripts regarding a specific action can also be placed inside the corresponding action folder.

A clear structure of the folder system regarding the action scripts can be found below:

```
Assets
├── Samples
│   └── ORamaVR MAGES
│       └── 4.0.0
│           ├── Operation1
│           │   ├── ActionScripts
│           │   │   ├── Lesson0
│           │   │   │   ├── Stage0
│           │   │   │   │   ├── Action0
│           │   │   │   │   │   ├── Action0Script.cs
│           │   │   │   │   │   ├── Action0Helper0.cs
│           │   │   │   │   │   ├── ...
│           │   │   │   │   │   └── Action0HelperN.cs
│           │   │   │   │   ├── Action1
│           │   │   │   │   ├── ...
│           │   │   │   │   └── ActionN
│           │   │   │   ├── Stage1
│           │   │   │   ├── ...
│           │   │   │   └── StageN
│           │   │   ├── Lesson1
│           │   │   ├── ...
│           │   │   ├── LessonN
│           │   │   └── Optional
│           │   ├── Models
│           │   ├── Resources
│           │   └── Scenes
│           ├── Operation2
│           ├── ...
│           └── OperationN
└── Scenes
```

5.3.4 Storyboard XML files

When running in Editor mode, Scenegraph is looking at a specific folder for the .xml *storyboard* files.

The path for this is the following:

`Assets/Samples/ORamaVR MAGES/(Operation Name)/Resources/Storyboard/platform`

For instance, the .xml path for the Medical Simulation (TKA) is `Assets/Samples/ORamaVR MAGES/Medical simulation (TKA)/Resources/Storyboard/platform`

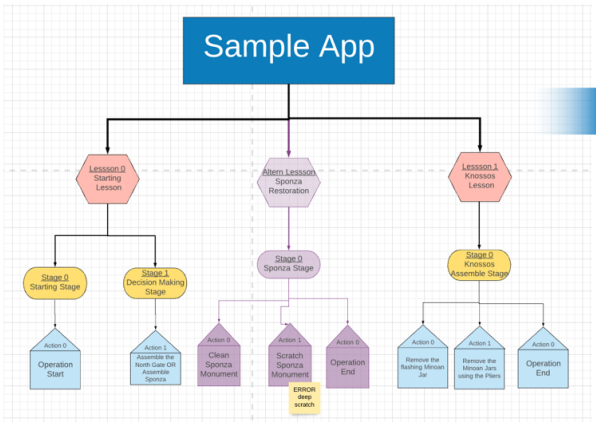
> **Warning:** If XML files are not in the directory above, Scenegraph will fail to load!

Note: In build the XML files can be placed under `{user}/Documents/ORamaVR/Story/platform/` as described in the Build Windows section in the Getting Started guide.

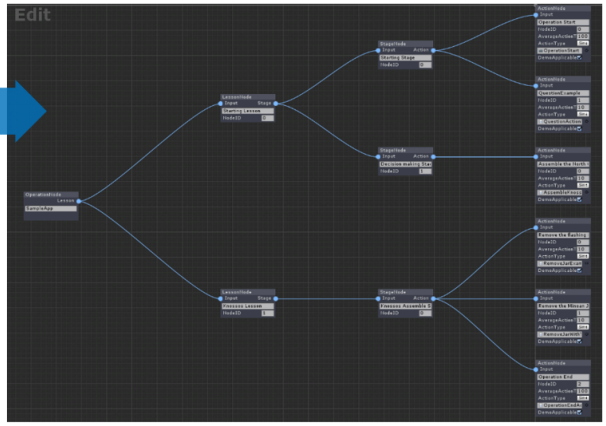
5.4 Scenegraph

Scenegraph is perhaps the most fundamental concept in MAGES SDK. It is this root module that powers and distinguishes ORamaVR's educational platform from the herd.

In this section, we proceed to present Scenegraph's architecture and how it structures all development.



Presentation of Structured Storyline



Storyline during development

The image above shows the transformation of a structured storyline into the MAGES scenegraph tree.

5.4.1 Scenegraph Explained

In order to achieve a goal whether it is the restoration of a statue or a medical operation you need to follow a list of tasks/steps in a sequential order. We are referring to those steps as Actions.

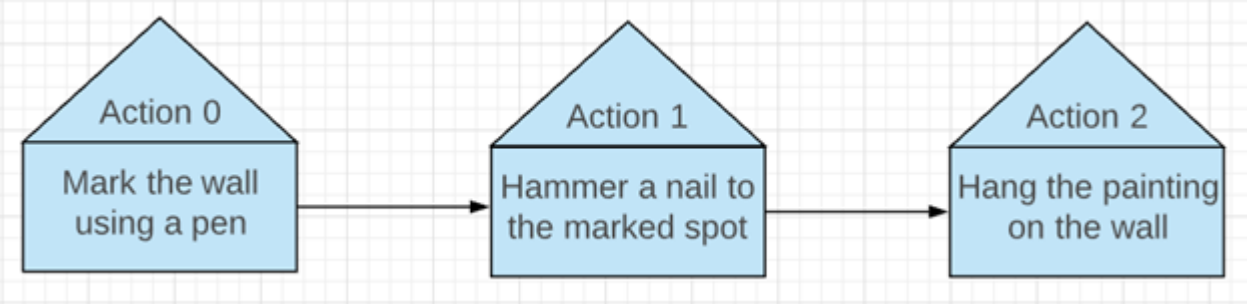
For instance, if we want to hang a painting on the wall we have to perform the following steps (Actions):

i. Mark the wall using a pen.

ii. Hammer a nail at the marked spot.

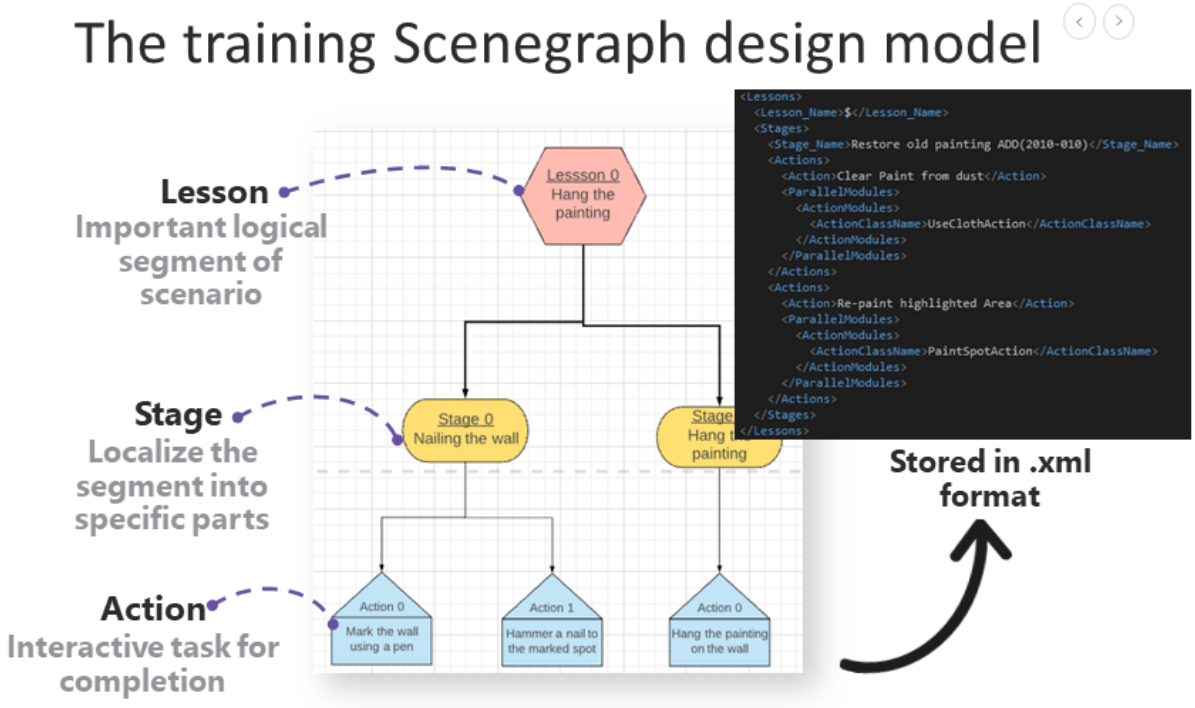
iii. Hang the painting on the wall.

Those are three steps that someone needs to complete to hang a painting on the wall. Having those steps in mind we create nodes, each one representing an Action.



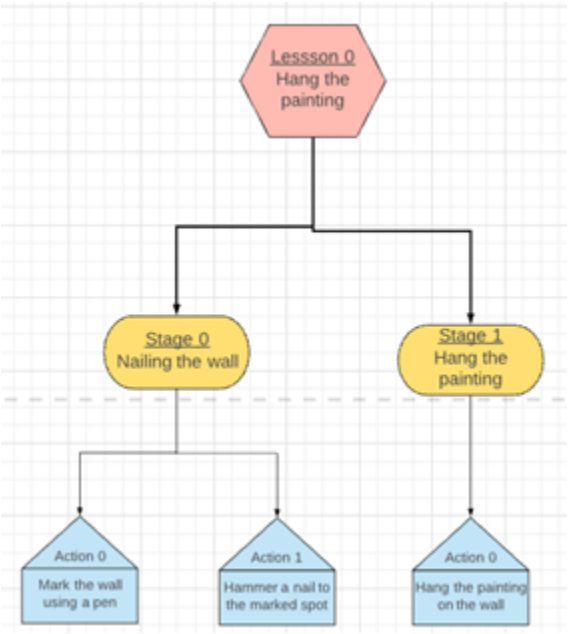
However in more complex applications there are dozens of Actions, in this case a sequential representation is not very convenient. For this reason we implemented the Scenegraph architecture.

A Scenegraph is a tree with three levels of depth. The root of the tree defines the operation/process, on depth 1 we have the Lesson nodes, depth 2 the Stage nodes and finally depth 3 the Action nodes.



The procedure runs only on Action nodes, but we also use the other nodes in a tree format to merge parts of the simulating procedure.

For instance, we can present the above 3 Actions in a tree format as follows:



In this scenario we decided to group the first two Actions in a Stage since both of them are referring to steps that are linked with the nail. The last action can be placed alone in a stage particularly for this cause.

After those optimizations, this lesson can be used in a more complex procedure with other lessons to construct a bigger Scenegraph tree. However, pay attention that even though we have multiple types of nodes (Lesson, Stage, Actions), only the Action nodes have customizable behavior.

Note: The operation runs only on Actions, the other nodes are for traversal and scene management.

Scenegraph is implemented under the "Scene Graph" gameobject in Unity scene. This gameobject will contain the Lesson, Stages and Actions. Scenegraph will manage, perform and run all the Actions as an educational pipeline.

5.4.2 The Scenegraph xml file

The Scenegraph structure is saved in an .xml file An example of this xml file can be seen in Sample App (SampleApp.xml).

```
<Lessons>
  <Lesson_Name>Knossos Lesson</Lesson_Name>
  <Tag>Normal</Tag>
  <Stages>
    <Stage_Name>Knossos Assemble Stage</Stage_Name>
    <Actions>
      <Action>Remove the flashing Minoan Jar</Action>
      <ActionClassName>RemoveJarExample</ActionClassName>
      <optionalActions>
        <string>Sponza Lesson</string>
      </optionalActions>
      <destroyOnPerform />
    </Actions>
    <Actions>
      <Action>Remove the Minoan Jars using Pliers</Action>
      <ActionClassName>RemoveJarWithToolExample</ActionClassName>
      <optionalActions />
      <destroyOnPerform />
    </Actions>
  </Stages>
</Lessons>
```

As you can see in this snippet we generate the Knossos Lesson which has one Stage and two Actions. The format of this xml needs to be as this example indicates, otherwise there will be errors in importing.

Below there is an explanation of the xml tags used in this script.

ArrayOfLessons	Contalls all the Lessons.
Lessons	Contains a Lesson.
Lesson_Name	Name of Lesson. This string will be used to timeline UI as it will be shown to the UI button in Lesson Timeline.
Tag	Lesson tag. Can be either “Normal” or “Optional”
Stages	Contains a Stage.
Stage_Name	Name of Stage. This string is only used to name the gameobject and nowhere else.
Actions	Contains all Actions.
Action	Name of Action. This string needs to address in a small sentence the purpose of current Action. This string will appear on the gamification monitor (as the current Action) so in a few word needs to address the current Action (e.g. Remove the Flashing Minoan Jar).
ActionClassName	Contains the Class name of this Action. Each Action has an Action script that needs to be added to the Action’s gameobject Node. This script will automatically run by Scenegraph when the right time comes. In order to have a clear project we recommend storing all the Action scripts under the format LessonX/StageX/ActionX to match the Scenegraph Nodes.
optionaActions	Instatiates the assigned Optional Actions. In this case, the Sponza Lesson will be initialized along with this Action.
dedstroyOnPerform	Destroys the assigned Optional Actions when performing these Actions. In his case they are none.

5.4.3 Optional Actions

Scenegraph is not just a static tree, it's a dynamic graph. Since an educational pipeline can lead to multiple paths according to user's actions and decisions, Scenegraph does it so. There are times in a procedure that user needs to choose between two predefined paths or an error he made leads to a completely different path.

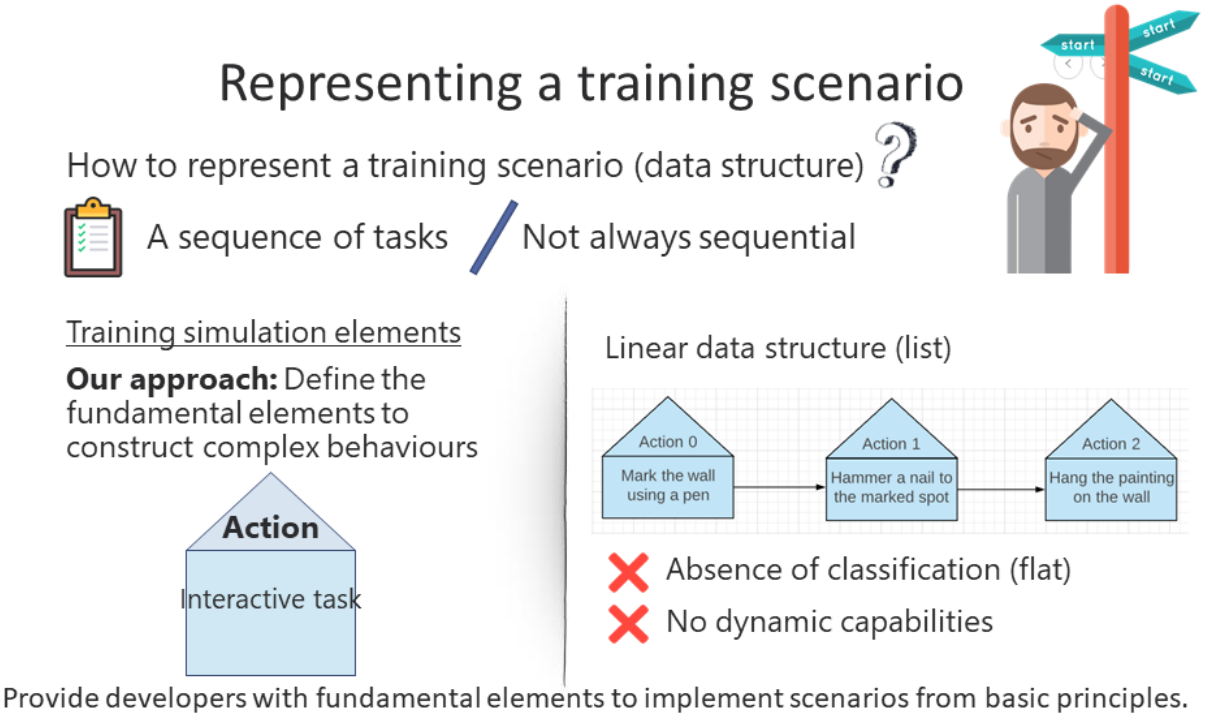
These functionalities are implemented in such a way to support real time decision making and as a result Scenegraph can change its structure (Nodes) as the procedure goes on. Scenegraph currently supports the *addition*, *deletion* and *alternation* of Lessons depending on the user's actions and decisions.

You can implement the above behavior using the Optional Actions. As mentioned on the table above, each lesson has a tag, if this tag is set to "Normal" then the Action will be spawned in the main path of the scenegraph as usual. However if this tag is set to "Optional" the Action will not spawn but wait to be spawned along with another one as Optional meaning that users can choose if they want to perform it. Those Actions may be side tasks that are not important for the operation's main path.

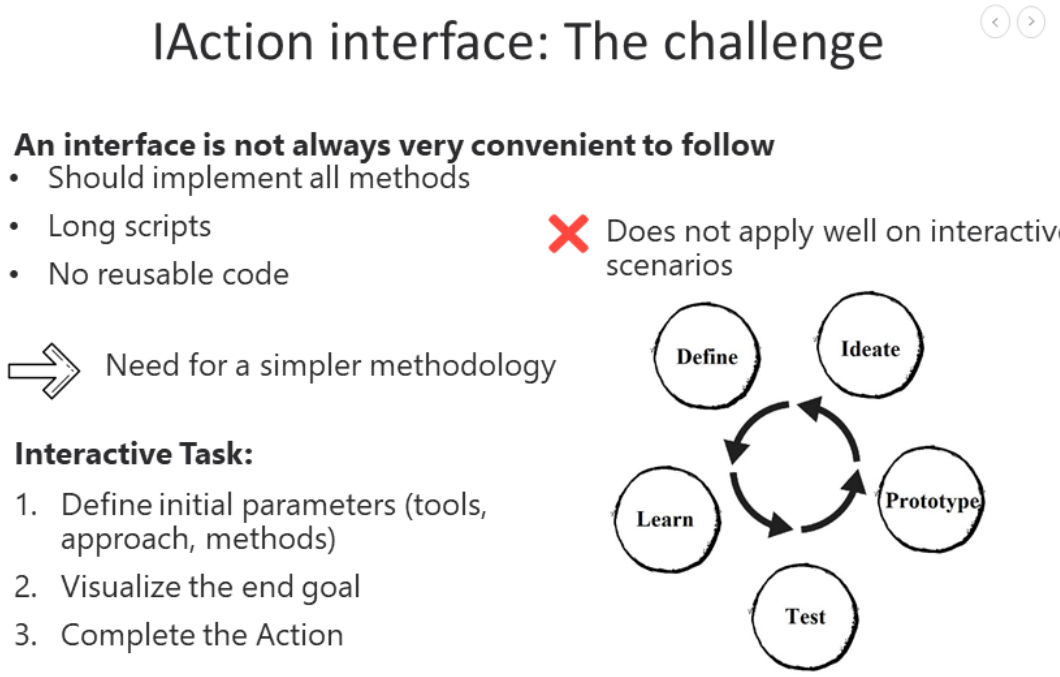
5.5 Action Prototypes

5.5.1 Introduction

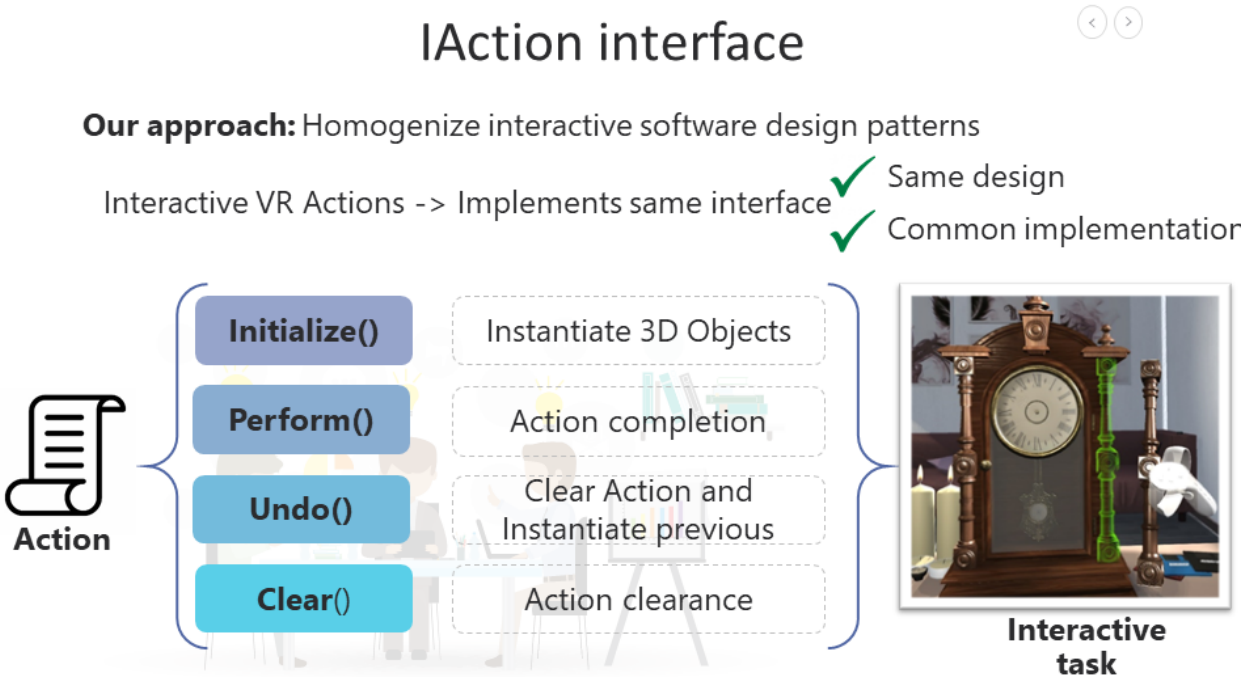
As mentioned before each step of a pipelined process is translated to an Action Script. This Script contains information to define the Action's behavior.



The Action object reflects a flexible structural module, capable to generate complex behaviors from basic ones. This is also the concept idea behind scenegraph; **provide developers with fundamental elements and tools to implement scenarios from basic principles**. Each Action script describes the behaviour in means of physical actions in the virtual environment.



In technical details, each Action script implements the **IAction** interface, which defines the basic rules every Action should follow. This interface ensures that all Actions will have the same methods.



The methods and properties of **IAction** interface are explained in detail below.

```
/// <summary>
/// This Inteface nedds to be implemented for every Action
/// Describes the functionalities the Actions should have
/// </summary>
```

(continues on next page)

(continued from previous page)

```
public interface IAction
{
    /// <summary>
    /// GameObject.name
    /// </summary>
    string ActionName
    {
        set;
        get;
    }

    /// <summary>
    /// The Gameobject refering to the current Action in Unity
    /// This implements the core of unity's scenegraph.
    /// </summary>
    GameObject ActionNode
    {
        set;
        get;
    }

    /// <summary>
    /// Go to Next Action
    /// Completes the current Action by finilizing and cleaning it
    /// Destroys prefabs, holograms
    /// Also plays animations to set the next one
    /// </summary>
    void Perform();

    /// <summary>
    /// Go to Previous Action
    /// Resets current Action by finilizing and cleaning it
    /// Destroys prefabs, holograms
    /// Plays Undo animations
    /// </summary>
    void Undo();

    /// <summary>
    /// Initialize current Action by spawning the necessary prefabs
    /// Sets each Action properties to run correctly
    /// </summary>
    void Initialize();

    /// <summary>
    /// Sets Holograms for current Action depending on the difficulty
    /// </summary>
    void InitializeHolograms();

    /// <summary>
    /// Sest the difficulty/error colliders for the current Action depending on the␣
↪difficulty
    /// </summary>
    void DifficultyRestrictions();

    /// <summary>
    /// Destroys the current Action, what is spawned from the Action it gets destroyed
    /// </summary>
```

(continues on next page)

(continued from previous page)

```
    void DestroyAction();

    /// <summary>
    /// Used only for Combined Actions
    /// Sets the next sub-Action to run after Performing the current one
    /// </summary>
    /// <param name="action">The next Action to run</param>
    void SetNextModule(Action action);
}
```

Action Prototypes

At this point, we have described the basic interface each Action should implement to initialized and performed properly. With this interface, a developer can generate action scripts that behave in a common ruleset, following the scenegraph pipeline.

To make our system more efficient we have to limit the capabilities of the Action entity to target simple but commonly used behaviors/tasks in training scenarios. Modelling those behaviours, we will generate a pool of generic behavioral patterns and tasks from which we will develop scenarios that are more specific.

Therefore, MAGES SDK introduces several specific Action behaviors that developers can utilize to simulate training scenarios. These are called **Action Prototypes** and are the following:

1. *Insert Action*
2. *Remove Action*
3. *Use Action*
4. *Tool Action*
5. *Combined Action*
6. *Question Action*
7. *Animation Action*
8. *Cut Action*

Each **Action Prototype** inherits from **BasePrototype**, an **abstract class** that utilizes a common set of methods and properties for every prototype.

5.5.2 Insert Action

Insert Action is referring to a specific type of Action that a user has to insert an object to a specific position in order to complete it.

For instance, an insert action can be scripted as follows:

```
public class PolyethyleneTrialAction : InsertAction
{
    public override void Initialize()
    {
        SetInsertPrefab("Lesson7/Stage2/Action0/Polyethylene",
                        "Lesson7/Stage2/Action0/PolyethyleneFinal");
        SetHoloObject(""Lesson7/Stage2/Action0/Hologram/HologramL7S2A0");
```

(continues on next page)

(continued from previous page)

```
            base.Initialize();
        }
}
```

Note: Notice how the developer defined action inherits from the **InsertAction** base prototype.

Action Script Explanation

1. SetInsertPrefab(string arg1, string2)

This method sets the Action's insert prefabs that will be spawned on Initialize. To set an insert Action you need to spawn two different objects, the interactable item and the final prefab. The first argument is the path to the interactable prefab while the second is the path to the final.

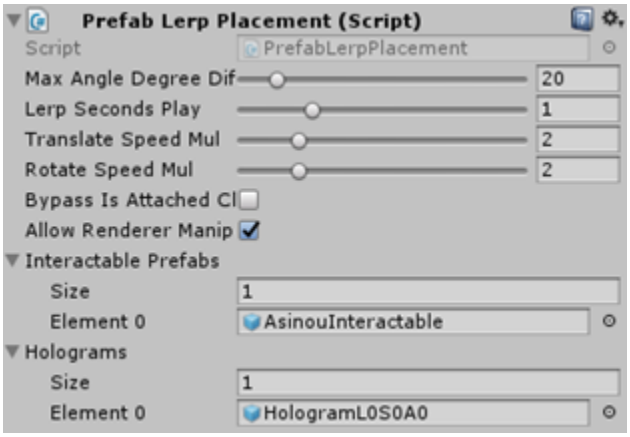
2. SetHoloObject(string arg1)

The *Hologram* is set for initialization through the *SetHoloObject* function.

3. Prefab Constructor

To create the correct prefabs you need to set the prefab constructors as follows.

For the Interactable prefab select the "Interactable" prefab Type on prefab Constructor script. This option will load all the necessary scripts to your interactable prefab. Then you need to check the interactable prefab constructor script and customize the serializable settings for your Action. To enable the Insert Action functionality for this object you must select the "Insert" option in the Prefab Interactable serializable field at Prefab constructor script. Next on the list is the final prefab. To generate a final prefab you need to select the "Interactable Final Placement" option from the prefab constructor script.



4. Finally, the **base.Initialize()** method needs to be called to set the prefabs on the **BasePrototype**.

Adding More to it

A more complex example that involves two insert actions as sub-actions is the following:

```
public override void Initialize()
{
    AnalyticsManager.AddScoringFactor<ForceScoringFactor>(2);

    //InsertAction sub-Action
    InsertAction insertFrontGateAction = gameObject.AddComponent<InsertAction>();
    insertFrontGateAction.SetInsertPrefab("Lesson0/Stage1/Action0/
↪FrontPartInteractable",
                                          "Lesson0/Stage1/Action0/FrontPartFinal");
    insertFrontGateAction.SetHoloObject("Lesson0/Stage1/Action0/Hologram/
↪FrontPartHologram");

    //-------------------------------------------------------------------------------
↪------------
    //InsertAction sub - Action
    InsertAction insertBackGateAction = gameObject.AddComponent<InsertAction>();
    insertBackGateAction.SetInsertPrefab("Lesson0/Stage1/Action0/BackPartInteractable
↪",
                                          "Lesson0/Stage1/Action0/BackPartFinal");
    insertBackGateAction.SetHoloObject("Lesson0/Stage1/Action0/Hologram/
↪BackPartHologram");
    //-------------------------------------------------------------------------------
↪------------
    //ToolAction sub - Action
    ToolAction hitWithMallet = gameObject.AddComponent<ToolAction>();
    hitWithMallet.SetToolActionPrefab("Lesson0/Stage1/Action0/BackPartHitMallet",␣
↪MAGES.ToolManager.tool.ToolsEnum.Mallet);
    hitWithMallet.SetHoloObject("Lesson0/Stage1/Action0/Hologram/MalletHologramL0S1A0
↪");

    InsertIActions(insertFrontGateAction, insertBackGateAction, hitWithMallet);


    base.Initialize();
}
```

In the above example, notice how each individual insert action follows the exact same pattern of object initialization.

5.5.3 Remove Action

Remove Action describes a step of the procedure which user has to remove an object using his hands or a tool.

Example of Remove Action Script:

```
public class RemoveJarWithToolExample : RemoveAction {

    /// <summary>
    /// Initialize() method overrides base.Initialize and sets the removable prefab
    /// </summary>
    public override void Initialize()
    {
        //Sets removable prefabs
        //Each method call adds a new removable prefab to remove prefab list
```

(continues on next page)

(continued from previous page)

```
        //Ehen user removes all of them then the Action performs
        SetRemovePrefab("Lesson1/Stage0/Action1/MinoanJar1RemovePivot");
        SetRemovePrefab("Lesson1/Stage0/Action1/MinoanJar2RemovePivot");
        SetRemovePrefab("Lesson1/Stage0/Action1/MinoanJar3RemovePivot");
        SetRemovePrefab("Lesson1/Stage0/Action1/MinoanJar4RemovePivot");

        SetHoloObject("Lesson1/Stage0/Action1/Hologram/HologramL1S0A1");

        base.Initialize();
    }
}
```

Action Script Explanation

1. SetRemovePrefab(string arg1)

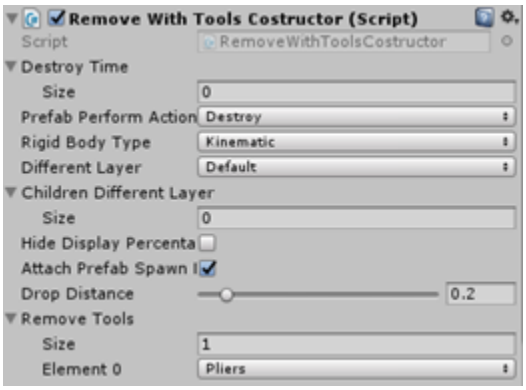
This method sets the Action's removable prefabs to initialize the Action behavior. To set a Remove prefab you need a string which contains the path to the removable prefab. This method can be called many times in an Action Script. Each time SetRemovePrefab is called a new removable prefab is added into the remove prefabs List. To perform the Action user needs to remove all of them.

2. SetHoloObject(string arg1)

Usually a remove Action does not have a hologram. Instead, we use a flashing indicator at the removable object. To enable the flashing functionality you need to check the "Attach Prefab Spawn Notifier" option at the Interactable Prefab Constructor script.

3. Prefab Constructor

To generate a removable prefab you need to select the "Interactable" option at the prefab constructor script. To enable the Remove Action functionality for this object you must select the "Remove" option in the Prefab Interactable serializable field at Prefab constructor script.



Adding More to it

A more advanced example is the following:

```
/// <summary>
/// Initialize() is the only method from basePrototype that we MUST override
/// This method initializes the Action by setting the paths to the spawned prefabs
/// </summary>
public override void Initialize()
{
    //This method sets the prefab that will be removed
    SetRemovePrefab("Lesson1/Stage0/Action0/MinoanJarRemove");
    //Sets hologram
    SetHoloObject("Lesson1/Stage0/Action0/Hologram/HologramL1S0A0Hand");
    //Set Voice Actor to play after performing the Action
    SetPerformAction(() => { VoiceActor.PlayVoiceActor("excellent"); });

    base.Initialize();
}
```

5.5.4 Use Action

Use Action is similar to a Tool Action but instead of a tool we use another object to complete the Action.

Example of Use Action Script:

```
public class CleanKnossosAction : UseAction {
    public override void Initialize()
    {
        SetUsePrefab("AlternativeLessonPrefabs/SponzaRestoration/Stage0/Action0/cloth
↪", "AlternativeLessonPrefabs/SponzaRestoration/Stage0/Action0/Dust");

        SetPhysicalColliderPrefab("AlternativeLessonPrefabs/AsinouRestoration/Stage0/
↪Action0/PhysicalCollider");
        SetHoloObject("AlternativeLessonPrefabs/AsinouRestoration/Stage0/Action0/
↪Hologram/hologram_clotha");

        base.Initialize();
    }
}
```

Action Script Explanation

1. SetUsePrefab(string arg1, string arg2)
    1.1.Sets the prefab user needs to take and place it on the use collider to perform the Action.
    1.2. Sets the collider that the “UsePrefab” will interact to complete the action.
2. Collider Prefab Constructor

An important setting you have to configure when creating a Use Action script is the amount of time the use prefab needs to interact with the use collider to Perform the Action. You can set this variable at "Stay Time" serializable field at the use collider prefab constructor (Use Collider Prefab).

Adding More to it

A more advanced example is the following:

```
public override void Initialize()
{
    //Set the interactable prefab which user will take and use it (touch collider) to␣
↪perform the Action (1st argument)
    //Sets the collider which triggers the use Prefab. For more customization␣
↪(CollisionStay time toperform) see prefab constructor(Unity editor) (2nd argument)
    SetUsePrefab("AlternativeLessonPrefabs/SponzaRestoration/Stage0/Action0/cloth",
↪"AlternativeLessonPrefabs/SponzaRestoration/Stage0/Action0/Dust");
    //Sets physical colliders that will spawn on initialize
    //For this Action we need extra non triggered colliders since the model of Sponza␣
↪dont have by default
    //These collider will be destroyed after perform/ Undo
    SetPhysicalColliderPrefab("AlternativeLessonPrefabs/SponzaRestoration/Stage0/
↪Action0/PhysicalCollider");
    //Sets the hologram for current Action
    SetHoloObject("AlternativeLessonPrefabs/SponzaRestoration/Stage0/Action0/Hologram/
↪hologram_clotha");

    sponzaHandLock = Spawn("AlternativeLessonPrefabs/SponzaRestoration/Stage0/Action0/
↪SponzaHandLock");

    base.Initialize();
}
```

5.5.5 Tool Action

Tool Action is referring to an Action that user has to take a tool and use it to complete the action.

Example of Tool Action Script:

```
public class ScratchSponzaAction : ToolAction
{
    public override void Initialize()
    {
        SetToolActionPrefab("AlternativeLessonPrefabs/AsinouRestoration/Stage0/
↪Action1/RoofDirt_Tool_Collider_Prefab", ToolsEnum.Scalpel);
        SetErrorColliders("AlternativeLessonPrefabs/AsinouRestoration/Stage0/Action1/
↪Colliders/ErrorColliders");
```

(continues on next page)

(continued from previous page)

```
        SetPhysicalColliderPrefab("AlternativeLessonPrefabs/AsinouRestoration/Stage0/
↪Action1/Colliders/PhysicalCollider");
        SetHoloObject("AlternativeLessonPrefabs/AsinouRestoration/Stage0/Action1/
↪Hologram/hologram_scalpel");

        base.Initialize();
    }
}
```

Action Script Explanation

1. SetToolActionPrefab(string arg1, ToolsEnum tool)

   This method sets the action's object that needs to interact with the tool.

2. SetErrorColliders(string arg1)

   This method sets the action's error colliders. If the user hits these colliders, the collision is marked as an error.

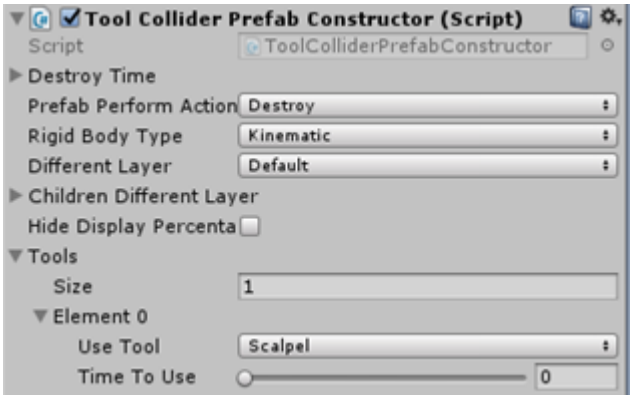
3. SetPhysicalColliderPrefab(string arg1)

This sets the physical (non-triggered) colliders. These colliders are used only in this action and are removed when it is completed.

4. SetHoloObject(string arg1)

   This sets the hologram indicating what needs to be done in this action.

5. Tool Action Prefab



   This type of action requires the tool action colliders to be specified on the tool action prefab. These colliders are specified by selecting the **Tool Collider Prefab Constructor** from the prefab constructor script. When selected, the tool colliders are created automatically by the prefab constructor.

5.5.6 Combined Action

A Combined Action has the attribute to perform multiple sub-actions sequentially. Sub-actions consist of any other type of Actions described in this section.

For example, to have an Insert Action followed by a Remove Action but consider both as one whole action, use the Combined Action prototype as described in this section.

To create a Combined Action follow the same ideology as the other prototypes with the difference that you need to define a gameobject for each sub-action you wish to insert.

For example,

```
public class AssembleKnossosPartOfAction : CombinedAction
{
    public override void Initialize()
    {
        AnalyticsManager.AddScoringFactor<ForceScoringFactor>(2);

        //InsertAction sub-Action
        InsertAction insertFrontGateAction = gameObject.AddComponent<InsertAction>();
        insertFrontGateAction.SetInsertPrefab("Lesson0/Stage1/Action0/
↪FrontPartInteractable",
                                              "Lesson0/Stage1/Action0/FrontPartFinal
↪");
        insertFrontGateAction.SetHoloObject("Lesson0/Stage1/Action0/Hologram/
↪FrontPartHologram");

        //-------------------------------------------------------------------------
↪----------------
        //InsertAction sub - Action
        InsertAction insertBackGateAction = gameObject.AddComponent<InsertAction>();
        insertBackGateAction.SetInsertPrefab("Lesson0/Stage1/Action0/
↪BackPartInteractable",
                                             "Lesson0/Stage1/Action0/BackPartFinal");
        insertBackGateAction.SetHoloObject("Lesson0/Stage1/Action0/Hologram/
↪BackPartHologram");
        //-------------------------------------------------------------------------
↪----------------
        //ToolAction sub - Action
        ToolAction hitWithMallet = gameObject.AddComponent<ToolAction>();
        hitWithMallet.SetToolActionPrefab("Lesson0/Stage1/Action0/BackPartHitMallet",␣
↪MAGES.ToolManager.tool.ToolsEnum.Mallet);
        hitWithMallet.SetHoloObject("Lesson0/Stage1/Action0/Hologram/
↪MalletHologramL0S1A0");

        InsertIActions(insertFrontGateAction, insertBackGateAction, hitWithMallet);

        base.Initialize();
    }
}
```

Action Script Explanation

The action prototype gameobjects are first defined and later set using each prototype's rule (e.g., an Insert Action needs SetInsertPrefab, a Tool Action needs SetToolActionPrefab, and so on).

Finally, you need to finalize the Combined Action's setup by calling an InsertAction's function with parameters the sub-actions objects you created before.

5.5.7 Question Action

This type is described as an action expecting the user to make a decision on choices that answer a specific question. Answers are expected to be sprites on a Button gameObject.

Choosing a wrong answer does not perform the action, but registers the selection as wrong. When the correct answer is selected, the action performs after 5 seconds.

Example of Question Action:

```
/// <summary>
/// This is an example of Question Action
/// In this Action users are asked a question and they need to answer to complete the␣
↪Action
/// The developer can set multiple answers by modifying the question prefab. In this␣
↪example we have two answers
/// </summary>
public class QuestionActionExample : QuestionAction
{
    /// <summary>
    /// Initialize() method overrides base.Initialize and sets the question prefab
    /// </summary>
    public override void Initialize()
    {
        //Sets the question prefab that will spawn
        SetQuestionPrefab("Lesson0/Stage0/Action1/QuestionPRefabExample");

        //This method enables the raycast so users can answer using their controllers
        InterfaceManagement.Get.InterfaceRaycastActivation(true);

        base.Initialize();
    }
}
```

Action Script Explanation

1. SetQuestionPrefab Spawns the Action prefab containing the question to be answered. Has two arguments:
    1.1. The prefab to be spawned
    1.2. The Question itself (expected type: string)
2. AddAnswerImage Spawns the Answer gameObject. Automatically attached onto the question Canvas.

5.5.8 Animation Action

This type of Actions is described as Actions in cases we want to insert an object, but the object needs to be inserted with an animated movement.

An example may be the insertion of a wire into a tube.

To implement this Action we would need to record the insertion of the wire, and then we will push it with our hands to the final position. The movement from the controller is translated into the normalized value of the animation [0-1].

Example of Animation Action:

```
public override void Initialize()
{
    //Helper variable to destroy the animated plug when coming back from a Perform␣
↪(so not to have two of them)
    plug = GameObject.Find("PlugAnimated(Clone)");
    if(plug)
        DestroyUtilities.RemoteDestroy(plug);

    //The animated insertion of the Plug
    //In this example the user needs to insert the cable to the plug. The movement is␣
↪recorded into an animation and played along with the movement
    //of the controllers
    SetAnimationPrefab("Lesson1/Stage0/Action3/PlugAnimated");

    //Hologram to pinpoint the correct insertion
    SetHoloObject("Lesson1/Stage0/Action3/PlugHolo");

    //We make sure the KnossosLight is turned off
    GameObject.Find("KnossosLight").GetComponent<Light>().enabled = false;

    base.Initialize();
}

public override void Perform()
{
    //After inserting the cable we light up the knossos with
    GameObject.Find("KnossosLight").GetComponent<Light>().enabled = true;

    base.Perform();
}
```

5.5.9 Non-Prototyped Actions

There are Action behaviors that have not been prototyped. In this situation we generate a custom Action that bypasses the BasePrototype and directly implements the IAction Interface. This Action needs to implement all the methods from the interface and manually spawn/destroy prefabs and any other behavior the Action needs.

When we have a non-prototyped action, we have to focus mostly on Initialize and Perform.

In an Action that implements the IAction interface we have to do **manually** all the work BasePrototype does. For example, the Initialize of this action needs to set the event manager and spawn the prefabs needed.

Additionally, perform will destroy prefabs and clear the event manager manually.

Note: It is important to consider all these manually setting s since nothing is automated by a non-prototyped Action.

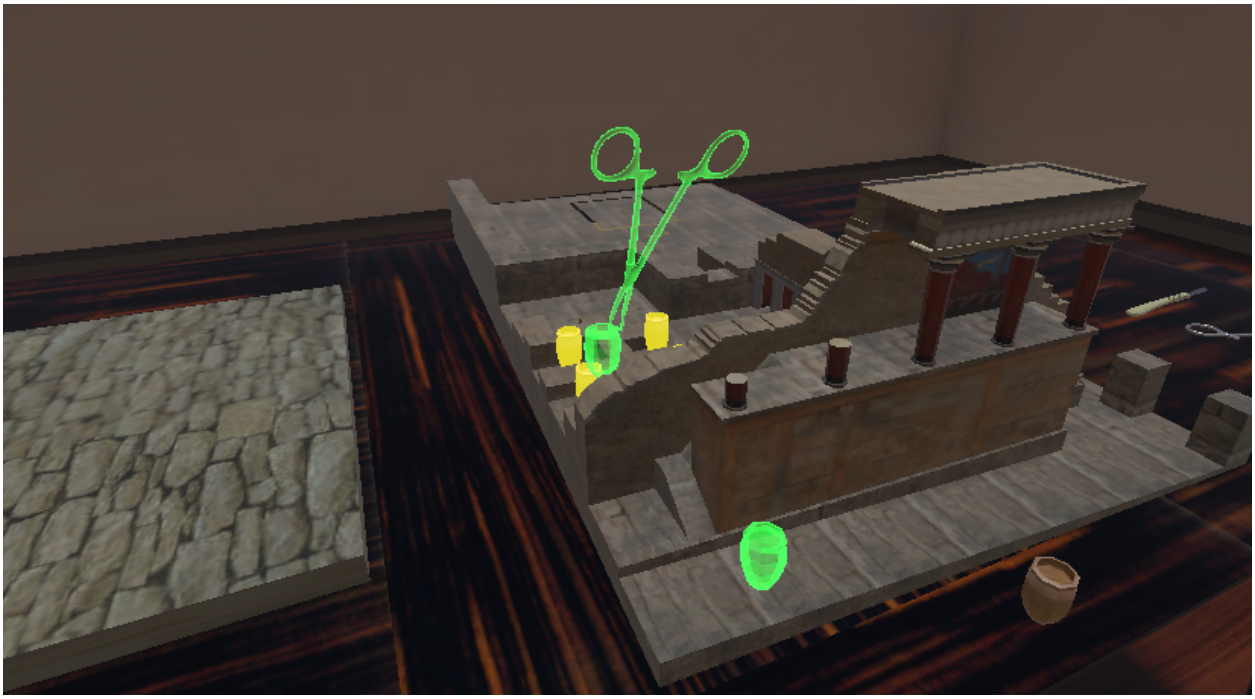
5.5.10 Optional Action

Optional Actions do not implement the IAction interface as the other ones and as a result you cannot code an Optional Action as the rest.

However, they enable the use of multiple active Lessons at the same time and decision-making. Below there is an explanation of those two functionalities.

Multiple active Actions

Imagine a Total Knee Arthroplasty scenario were we have the main tasks on the knee but at the same time there are some available Actions on your surgical table to assemble an instrument. Those Actions will wait there to be completed, but they will not affect the main path. Those are called Optional Action.

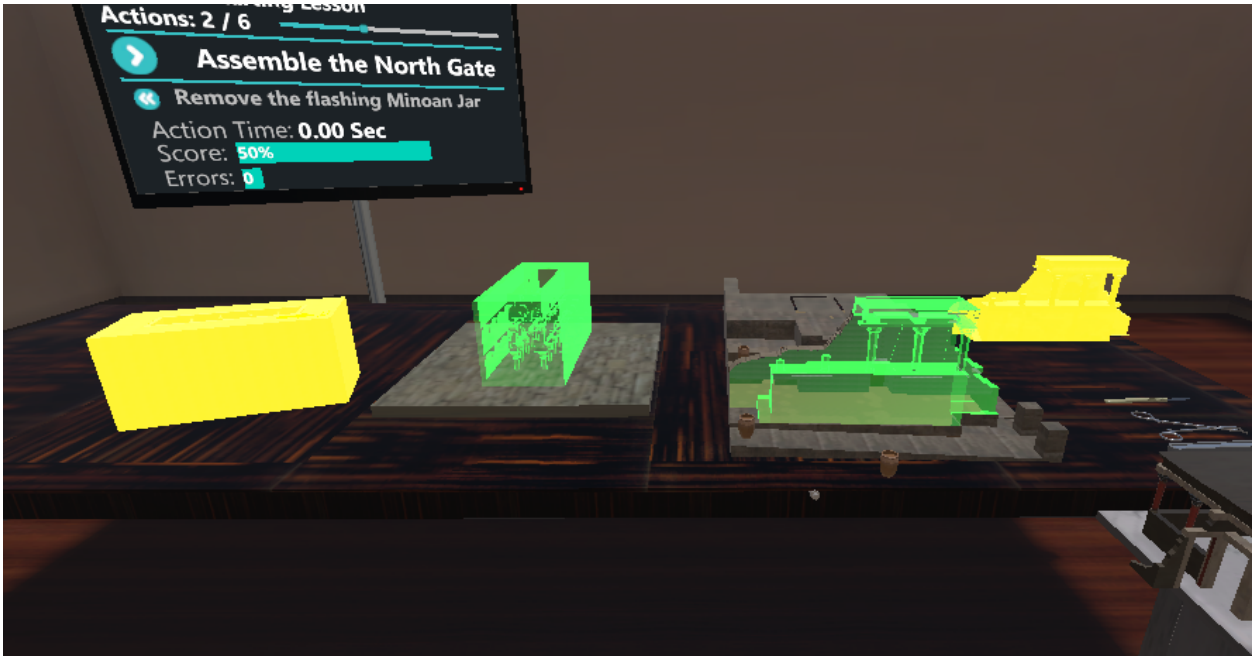


In the image above you see two active Actions: 1) The removal of jar using the pliers and 2) The insertion of jar (bottom)

You can see a tutorial on how to implement Optional Actions *here*.

Scenegraph manipulation (decision-making)

Another functionality of Optional Actions is related with the real-time manipulation (Action addition and deletion) of scenegraph and decision-making. In the same Total Knee Arthroplasty scenario we can have the main Action for the knee and an Optional Action for the tibia active at the same time. If the user decides to perform the tibia Action instead we can write a bit of code in its Perform to change the main scenegraph by adding lets say a Lesson that has all the Actions for the tibia. This Lesson will spawn inside the main scenegraph as the other main Actions.



n the image above you see two active Actions: 1) The insertion of Sponza (left) and 2) the insertion of Knossos (right). The normal path is the Knossos. However, if the user decides to insert the Sponza instead, the scenegraph will replace the Knossos Lesson with the Sponza Lesson.

You can see a tutorial on how to implement the decision-making functionality with Optional Actions *here*.

5.5.11 Cut Action

> **Warning:** It's highly recommended to read the CTD_Deformable and *Realtime Cut* sections.

Cut action is referring to a specific type of Action that a user has to cut a mesh in a specific position in order to complete it.

For instance, a Cut action can be scripted as follows:

```
public class CutSkin : CutAction
{
    public override void Initialize()
    {
        SetCutPrefabs("Lesson7/Stage2/Action0/CutSkinConstructorPrefab", "Skin");

        base.Initialize();
    }
}
```

> **Note:** Notice how the developer defined action inherits from the **CutAction** base prototype.

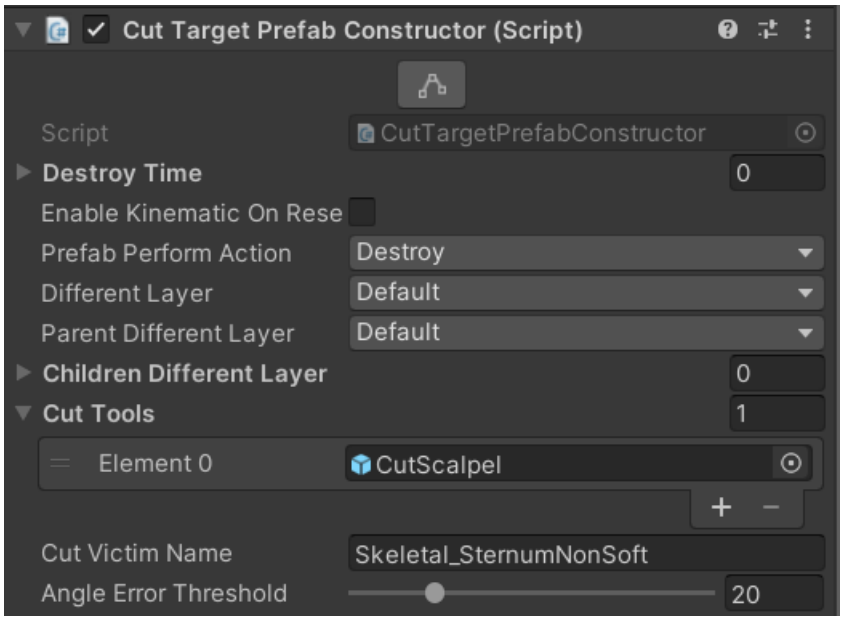
Action Script Explanation

1. SetCutPrefabs(string arg1, string arg2)

This method sets the Action's cut prefabs that will be spawned on Initialize. To set a Cut Action you need to spawn a Cut Prefab and set its parent. The first argument is the path to the cut prefab while the second argument is the name of the parent object in the scene, which is the object that will be cut.

2. Prefab Constructor To create the correct prefabs you need to set the constructor as follows.

First, there is a 'Edit Cut Plane Bounds' button in the Constructor, which allows you to edit the plane (change its position, rotation and scale) on which you want the object to be cut. In the Cut Tools option, you need to add the prefabs of the gameobjects that will be used to cut the selected object. Then in the Cut Victim Name option, you need to add the name of the object in the scene that will be cut. Lastly, you will need to set the Angle Error threshold, which refers to the threshold of the angle that the cutting tool can have in comparison with the plane.



3. Finally, the **base.Initialize()** method needs to be called to set the prefabs on the **BasePrototype**.

Adding More to it

A more complex example that involves spawning two Cuttable objects, a cutting tool and an extra Cut Constructor is the following:

```
public override void Initialize()
{
    SpawnCutObject("Lesson7/Stage2/Action0/Skin");
    SpawnCutObject("Lesson7/Stage2/Action0/Fat");
    SetCutPrefabs("Lesson7/Stage2/Action0/CutSkinConstructorPrefab", "Skin(Clone)");
    SetCutPrefabs("Lesson7/Stage2/Action0/CutFatConstructorPrefab", "Fat(Clone)");
    SpawnCuttingTools("Lesson7/Stage2/Action0/CustomCuttingTool");
    base.Initialize();
}
```

5.6 Prefab Constructors

5.6.1 Introduction

Almost all types of Actions inside the project are prototyped because they share a lot of behavioral elements. The same idea is applied to gameobjects/prefabs.

Their fundamental behavior can be split in a small amount of different Constructors. Depending on the Constructor attached, the creation for each prefab differs.

In the following sections you can find tutorials on every available Prefab Constructor:

1. *Generic Prefab Constructor*
2. *Interactable Prefab Constructor*
3. *Interactable With Parent Prefab Constructor*
4. *Interactable Final Placement Prefab Constructor*
5. *Tool Collider Prefab Constructor*
6. *Use Collider Prefab Constructor*
7. *Non Trigger Collider Prefab Constructor*
8. *Collision Hit Prefab Constructor*
9. *Remove With Tools Prefab Constructor*
10. *Question Prefab Constructor*
11. *Cut Prefab Constructor*

5.6.2 Generic Prefab Constructor

Initially, this script should not be attached to any gameobject as it is the base of all different type of constructors.

It contains selections (and functions) for the developer that are going to be found in any type of prefab constructor:

Variable Name	Type	Description
Prefab Perform Action	Destroy, Remain	Select if the prefab after action completion should remain in the scene or if it should be destroyed.
RigidBody Type	Kinematic, Gravity	Select if the prefab is Kinematic or if it has Gravity. For interactable prefab that always have physics properties this selection counts ONLY for when they are spawned, and it lasts till they are grabbed from the user. Upon user's release they automatically turn back to gravity.
Different Layer	Default, GrabbablePrefabs, TriggerColliderLesson, NoTriggerColliderLesson	Also these layers can be selected: cloth, AllBones, ModelSkin. They are not required to be created from the developer. As told before, each prefab depending on the attached constructor, is going to have a corresponding layer. If the developers need some parts of the prefab to be in another layer, then in this variable they should select the desired layer and in the next variable drag 'n' drop the parts of the prefab that this different layer is going to be applied.
Children Different Layer	Prefab's Children	Insert here all the prefab's children that need to be in a different layer. Leave empty if no child is desired to be changed.

Function Name	Description
public virtual void ResetPrefab()	Resets the prefab to its starting position and rotation.
public virtual void SetNewPrefabStartingTransform()	Changes the prefabs starting position and rotation from the values the gameobject had on spawn to the values the gameobject has on the time this function is called.
public virtual void FinalizePrefabAction()	Internal function called on Action end to finalize prefabs behavior. Can be overridden to add to the prefab functionalities.

5.6.3 Interactable Prefab Constructor

Inherits from Generic Prefab Constructor. This constructor should be applied to any prefab that will be interacted from the user and has physics properties.

It should be noted that the developer can select if the prefab is kinematic or has gravity.

This selection does NOT affect the prefabs whole lifetime. Since it's an interactable prefab it will always have gravity! If the developer selects kinematic, the prefab will be kinematic from the time it is spawned until the user grabs it.

When it will be released from the user it will turn back on the gravity. If the reset function is called the kinematic property will turn back on.

Variable Name	Type	Description
Prefab Interactable Type	Generic, Insert, Remove	Select how this prefabs is going to be used.
Prefab Detach Feature	ReInitialize, Destroy, EventTriggerCurrLSAOnDestroy	Select what the prefab should do when the user throws it away. Reset the prefab, destroy it or destroy it and call the Event Manager to trigger the Action Completion.
Allow Prefab Manual Reset	Boolean	Set to true in order for the PrefabSpawnManager to be allowed to observe and reset the specific prefab when needed.
Two Hand Interaction	Boolean	Set to true if prefab can be grabbed with both hands. Still EXPERIMENTAL, it might not work properly
Attach Prefab Spawn Notifier	Boolean	If true it attaches a script that flashes the prefab to be able to be noticed more easily from the user. When grabbed it auto destroys itself.
Prefab Material Type	enum	This is a selection for different sounds attached to the gameobject, not materials! It is still experimental and might not work properly. The developers should define their own sounds.
Min Distance Reset	enum	Set the gameobjects range from its starting to its current position where, when greater, the prefab will be reset.

Prefab Creation Requirements

1. Rigidbody
2. Non-triggered colliders, for physics collision
3. Trigger collider (optional), for action triggering & user grab trigger
4. MAGESInteractableItem script attached
5. Have no other transform as parent

5.6.4 Interactable With Parent Prefab Constructor

Inherits from Interactable Prefab Constructor. It works exactly the same as the script it inherits.

The only difference is that this script should be attached to an interactable gameobject that is a child of another gameobject.

Note: Interactable prefabs are meant to be by themselves one whole object without having a parent.

It is not advised to use often this script and to have a lot interactable that are children to another gameobjects.

5.6.5 Interactable Final Placement Prefab Constructor

This script is attached to a prefab that's duplicated from the interactable item, and it serves the role of the final placement. The way it works is that it has a collider and when the collision registers with the other gameobject it observes their transform. If their difference is below a specified margin (the rotations match to a certain point), the collision gets accepted.

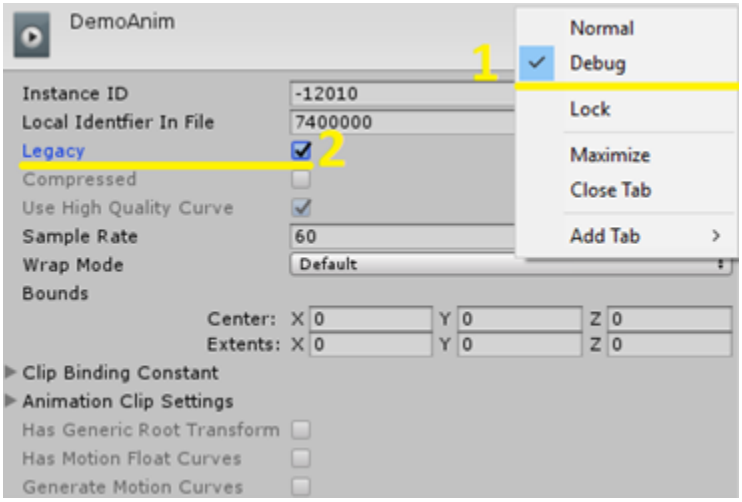
This prefab will -on start- have its renderers disabled, and it will be awaiting collision. When the collision succeeds, it detaches the interactable prefab from the user's hands, it translates it to the position (and rotates it) the final prefab is. When they are at the same position the prefab that the user had at hand gets destroyed and the final prefab enables its renderers.

When all of this it's done the final behavior of this prefab is called (e.g. some specific animation after the prefab is placed) and it triggers the Event Manager for the Action completion.

Prefab Creation Requirements

1. Rigidbody
2. Trigger Colliders
3. Prefal Lerb Placement script

For now, this script does not search for animator in the gameobjects components, but an animation component. To be able for the animation to play by itself without the animator it must be legacy supported (explained how in image below)



Prefab Lerp Placement

Everything explained above is the works of this script, PrefabLerpPlacement. That constructor only observes if the prefab has this script attached. This script can be used on its own anywhere.

Variable Name	Type	Description
Max Angle Degree	float	Observes the rotations of the final placement and the interactable prefabs. If their difference in all three axis is less that this given margin, it accepts the collision.
Lerp Seconds Play	float	When collision is accepted, the prefab at hand gets detached and transforms to the correct position of the one placed. This variable sets for how long that transition will last.
Translate Speed Mul	float	Translation speed multiplier.
Rotate Speed Mul	float	Rotational speed multiplier.
Bypass Is Attached Check	Boolean	Keep false! It should be true only for prefabs that mid-action are children to another gameobject.
Allow Renderer Manipulation	Boolean	This gameobject has its renderers disabled until it replaces the interactable prefab (after collision acceptance). Set to false if its renderers are desired to be permanently on.
Interactable Prefabs	list of gameobjects	Insert here all the gameobjects that this prefab is going to await collision with.
Holograms	list of gameobjects	Each action can have gameobjects that serve the role of visual aid for the user to understand how to perform the Action. Since this Action will have to do with inserting something, these helpful holograms need to be destroyed after the user just correctly placed the gameobject. Insert here all the prefabs that help the user understand this particular action. When the interactable gameobject is placed, this script will destroy all the inserted holograms.

5.6.6 Tool Collider Prefab Constructor

Some Actions in order to be performed, they need a lot of interaction points with specific gameobjects. These interaction points normally take the form of multiple colliders. A Tool Collider prefab is a gameobject where all these colliders are grouped underneath it as 1st depth children. Furthermore, this prefab constructor will expect collision only with gameobjects that are defined from the developers as Tools (explained later in this document).

Each collider on spawn gets as a component a ToolTriggerCollider script. Every time each collider is triggered, via the above mentioned attached script, it notifies the parent gameobject (the one with the Tool Collider Prefab Constructor) that it just accepted collision, and then it destroys itself. When the parent has no more any children-colliders, it triggers the Event Manager for the Action completion.

Variable Name	Type	Description
Tools	List of Pair<ToolsEnum, float>	Select the tools that can interact with the colliders and for each tool select how much time it needs for each collider to trigger before notifying the parent.

Prefab Creation Requirements

1. Parent has Rigidbody and NO colliders
2. Each child has one trigger collider

5.6.7 Use Collider Prefab Constructor

The idea behind this prefab is similar to the one explained in the 5.5. This script should be attached to prefabs that contain by themselves a collider that when triggered with specific gameobjects it triggers the Event Manager for the Action completion.

The difference here is that the gameobjects for the collision are type-unrestricted. Anything can be inserted for the collision acceptance.

Variable Name	Type	Description
Stay Time	float	Set the time needed for the collider to register a successful collision.
Prefabs Used	List of gameobjects	Insert any gameobject that the collider will await collision with.

Prefab Creation Requirements

1. Rigidbody
2. Trigger Collider

5.6.8 Non Trigger Collider Prefab Constructor

Attach to prefabs that have only non-triggered colliders and are expected to be interacted with other objects only via the physics engine.

It will on spawn have a layer that is observer only by itself.

Prefab Creation Requirements

1. Rigidbody
2. Non-trigger Colliders

5.6.9 Collision Hit Prefab Constructor

The general for these type of prefabs idea matches with the Use Collider mentioned in 5.6., except that it is more tailored to the needs of this type of interaction, hitting. One gameobject that awaits collision with one specific user defined Tool, and on each collision it will translate a small step into the direction given.

A behavior expected also in real life.

The developer must place the prefab to its final position (where it should be after the hits have occurred). Then, on spawn the gameobject will be translated into the opposite direction of the one selected by: total hits * translation per hit.

Variable Name	Type	Description
Tool	enum type of ToolsEnum	Select the user–defined tool for hits
Hit Step	float	Select the amount of translation per hit
Hit Count	float	Select the total hits
Min Magnitude	float	On each hit the script observes from the rigidbodies the strength of the hit. If the strength is below the given minimum magnitude it will ignore the collision.
Vector Direction	enum of all available selections for direction	Select the direction where the gameobject will translate to. In LOCAL Axis.
Collision Time Diff	float	When an accepted collision occurs, the collider will not accept another collision for a limited amount to time to avoid duplicate collisions. This amount of time can be selected with this variable.

Prefab Creation Requirements

1. Rigidbody
2. Trigger Collider

5.6.10 Remove With Tools Prefab Constructor

This script should be attached to prefabs that are already placed and need to be removed.

If the prefabs were required to be removed with the user's hands, the developer should use the Interactable Prefab Constructor (mentioned in 5.2.).

In this case the prefabs need to be removed with the use of another gameobject, in this script's case, the user-defined tools (where these tools are held from the user).

Variable Name	Type	Description
Attach Prefab Spawn Notifier	boolean	If true, it will attach to the object a script that flashes it to be recognized visually from the user.
Drop Distance	float	Set the minimum distance for the prefab. If the user removes with the tool the prefab but drops it to close to its starting position (distance smaller than the float given), the prefab will return back to and it will need to be removed again.
Remove Tools	List of ToolsEnum	Select the Tools with which the user can remove the prefab.

Prefab Creation Requirements

1. Rigidbody
2. Trigger Collider
3. Non-trigger Colliders

Warning: The non-trigger colliders are optional. It's there to help with the user presence. When the prefab is removed it needs to have all the physics properties and collide wherever the user is going to throw it.

But, the non-triggered collider will be disabled on spawn, otherwise they will bug (in physics level mostly) with the tool that will also have same type of colliders. These non-triggered colliders will be enabled only when the user will have removed and dropped the prefab away.

If the prefab still does not behave properly, leave it with ONLY the trigger colliders.

5.6.11 Question Prefab Constructor

The concept behind this prefab type is to have the user make a decision. This script should be attached to the root of the question prefab. This script handles user's interaction with the answers given to the question.

Furthermore, this constructor manages audio playbacks for the selected answers.

Variable Name	Type	Description
Correct	Audio Clip	Set the audio clip that will be played when user's answer is correct.
Wrong	Audio Clip	Set the audio clip that will be played when user's answer is wrong.

Note: Each answer prefab needs to have attached the **Question Trigger Collider** script.

Question Trigger Collider Script

This script must be attached manually on the answer gameObject. This component declares the prefab as an answer to the question. This also defines if the answer is correct or not.

5.6.12 Cut Prefab Constructor

Inherits from Generic Prefab Constructor. This constructor should be applied to any prefab that will be used for the perform of a Cut Action, since it sets the plane in which the Cuttable Object will be cut.

Variable Name	Type	Description
Cut Tools	List of GameObjects/Prefabs	The GameObjects that will be used to cut the Cuttable Object.
Cut Victim Name	String	The name of the Cuttable GameObject in the Scene
Angle Error Threshold	Integer	The error threshold in degress between the Cutting Tools and the plane of the constructor that was set.

Prefab Creation Requirements

1. Rigidbody
2. Trigger Collider that surrounds the Cuttable Object

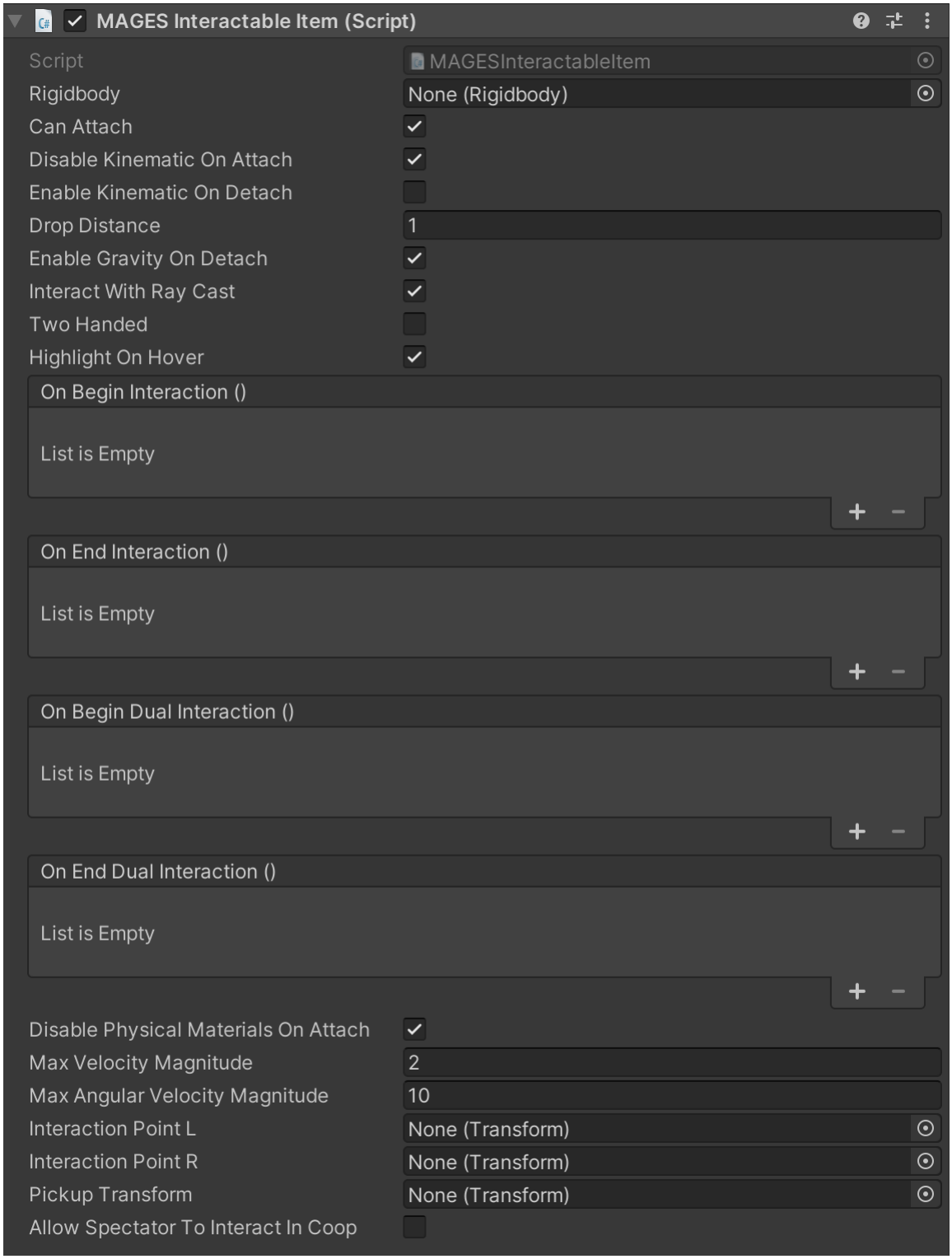
5.7 Physics

5.7.1 MAGES Interactable Item

The most important script the developer needs from the MAGESPhysX is the MAGESInteractableItem. When attached to an object it enables physics interactions to the object it is attached to.

More specifically, for an object to have physics interactions inside Unity, it is easy for the developer. Create colliders, add rigidbody and enable gravity. But if grabbing the object is implemented from the developers with the Unity's default method, parenting, upon grabbing, that object will lose its physics properties (e.g. it won't collide with object - passing through them).

Instead of parenting, the MAGES Platform provides this script for the developers to use. It is responsible for the object to be able to be grabbed from the user while maintaining its physical properties.



Parameters explained below:

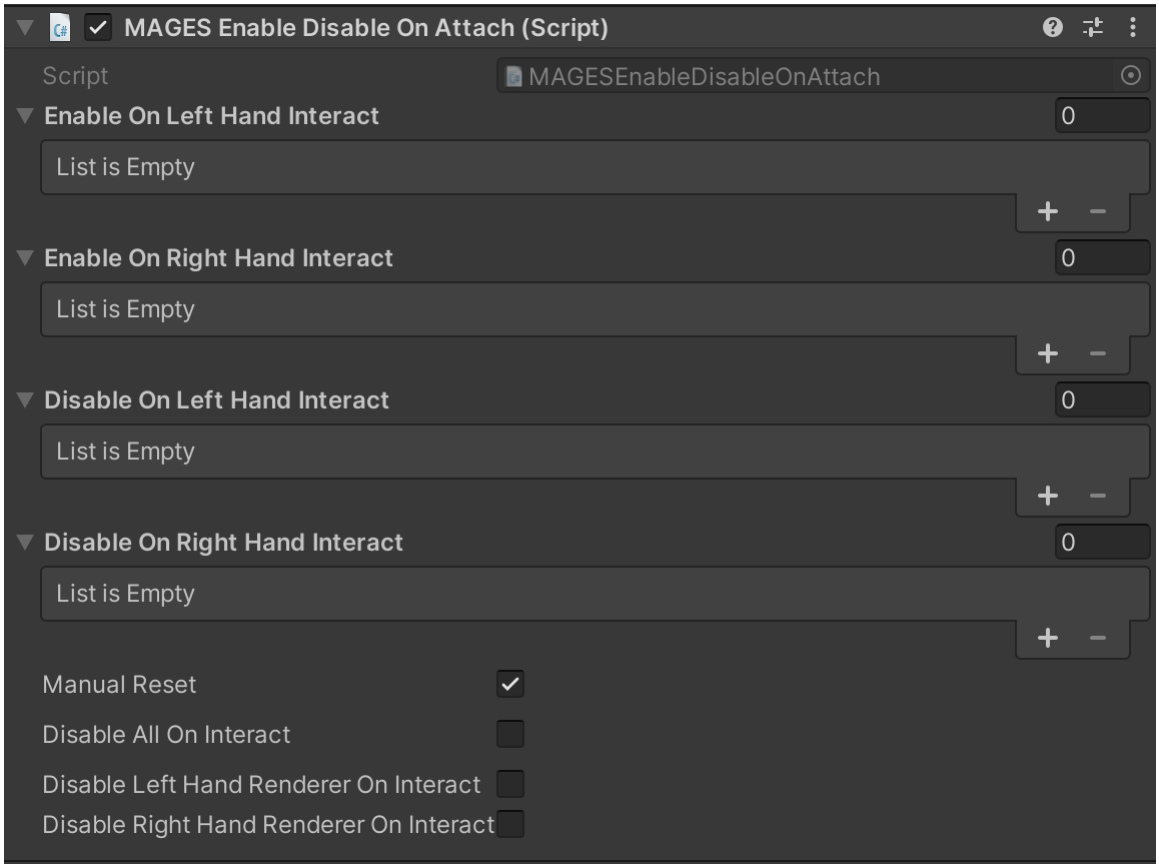
Parameter	Description
Can Attach	object can/can't be attached to the user's virtual hands
Disable Kinematic On Attach	if true, when attached to a hand it will disable its kinematic properties to re-enable objects physics interactions.
Enable Kinematic On Attach	if true, when released the object from users hands its kinematic properties will be re-enabled. (See Unity's documentation for more information on kinematic).
Drop Distance	set a distance margin between hand and object. When their distance is greater than the margin, the interaction will stop. Very useful for two hand interaction.
Enable Gravity On Detach	if true, when released the object from users hands its gravity will be turned back on.
Interact With Ray Cast	if true the user can interact with the object using the ray cast as well.
Two handed	if true it can be grabbed with both hands. Still experimental it can produce a large pivot between hands, use with caution.
Disable Physical materials On Attach	if a physical material is attached on the gameobject, it will be disabled on attach to avoid any influence in the interaction.
On Begin Interaction (Unity Action)	functions can be added here. They will be called when the user will grab the object.
On End Interaction (Unity Action)	functions can be added here. Upon objects release from the user, these functions are going to be used.
On Begin Dual Interaction (Unity Action)	functions can be added here. They will be called when the user will grab the object with both hands.
On End Dual Interaction (Unity Action)	functions can be added here. Upon objects release from the user (both hands), these functions are going to be used.
Interaction Point L & R	fixed points on the object used for snapping the object to that position and rotation on attach. More specifically, when grabbed, the object will keep its transform relative to the hand upon grasping. The developer can create specific positions and rotations for the object (for each hand) using different gameobjects as children of that object in question. If these transforms are given into these variables, the object upon grasping will automatically be changed in terms of transform to match these interactive points given.

5.7.2 MAGES Enable Disable On Attach

In the MAGESPhysX package you can also find the MAGESEnableDisableOnAttach script. In many occasions the developer might want to disable certain game objects when an interaction with an object starts.

For example, the developer might want to disable a UI indicator on attach and re-enable it on detach. It is also very common for the developer to want to disable the virtual hands' renderers and replace them with a specific grab pose while interacting with a game object.

All the above features are implemented in this script.



Parameter	Description
Enable On Left/Right Hand Interact	The prefabs referenced here will be enabled when the user interacts with the left/right hand.
Disable On Left/Right Hand Interact:	The prefabs referenced here will be disabled when the user interacts with the left/right hand.
Manual Reset	If true, once disable/enable happens on begin interaction it does not stop until reset is called. **Recommendation** keep it *false*.
Disable All On Interact	The prefabs referenced on both Disable left and right lists will be disabled when the user interacts with the item.
Disable Left Hand Renderer On Interact	If true, disable the renderer of the virtual left hand when interacting with object.
Disable Right Hand Renderer On Interact	If true, disable the renderer of the virtual right hand when interacting with object.

5.7.3 Soft Bodies

Introduction

Realistic deformations play an important role in computer graphics, games, simulations and VR environments.

Soft body simulations are used to change an object shape, when external forces are applied.

The computation of physically accurate deformation of objects when VR users uses hands controllers to interact is a liturgy, which requires much computation power. Only a few applications and simulations use soft body deformation due to computation power needed.

MAGES™ SDK & Soft bodies

In MAGES™ SDK, we provide a novel soft mesh deformation algorithm suitable for Virtual Reality interaction and collaboration.

The soft deformation algorithm is based on shape matching techniques and particle based spring mass soft body simulations. Our particle-based soft body algorithm is different from the state of the art because it provides easy control of the particles as physical objects and a center point, which controls the entire soft body position.

Velocity based interaction can be applied directly to our particles while as physical objects can interact also with the environment.

Our Virtual Reality interaction system uses velocity base approach providing the ability to pick up, hold and drop objects. Due to our soft body particles' nature, this interaction can be applied directly.

Real time soft mesh deformations

With the use of term soft body, we mean a mesh that can change its initial vertices position when interacting with other physical meshes. Our main idea of creating a soft body physics algorithm is based on three main categories:

1. Clustering

   The term clustering describes how each vertex on our mesh being grouped in another object that is used to calculate its deformation.

2. How physics are applied

   For the soft bodies to achieve some deformation, physics should be applied. To apply physics in our soft bodies efficient and with good performance, we apply them to the clusters that we used to group all the vertices.

3. Mesh deformation

   Finally, we have to calculate the mesh deformation. The resulting deformation derives from the Clustering transformation after the physics calculation. This transformation has an effect on our mesh vertices.

Soft bodies Interaction In VR

In virtual reality environments, the user can utilize hand controllers or other devices to interact with virtual objects. In this chapter, we provide a method on how this interaction can be performed based on physics properties.

We focus on how this interaction can be done by pickup and drop an object or physically interact with it. Then, we use this method to interact with our soft body mesh.

- Pickup objects

  With the use of VR controller, the user is able to pick a virtual object and move it according to the laws of physics. The object is not able to pass through another object and normally stops, slides around it or pushes it.

- Physical interaction

  The user is able to use the VR controllers to interact with object physically. For example, the user is able to push an object.

- Interact with soft body

  To interact with a soft body mesh, we use both of the above methods. The user is able to grab a soft body mesh or physically interact with it.

How-To

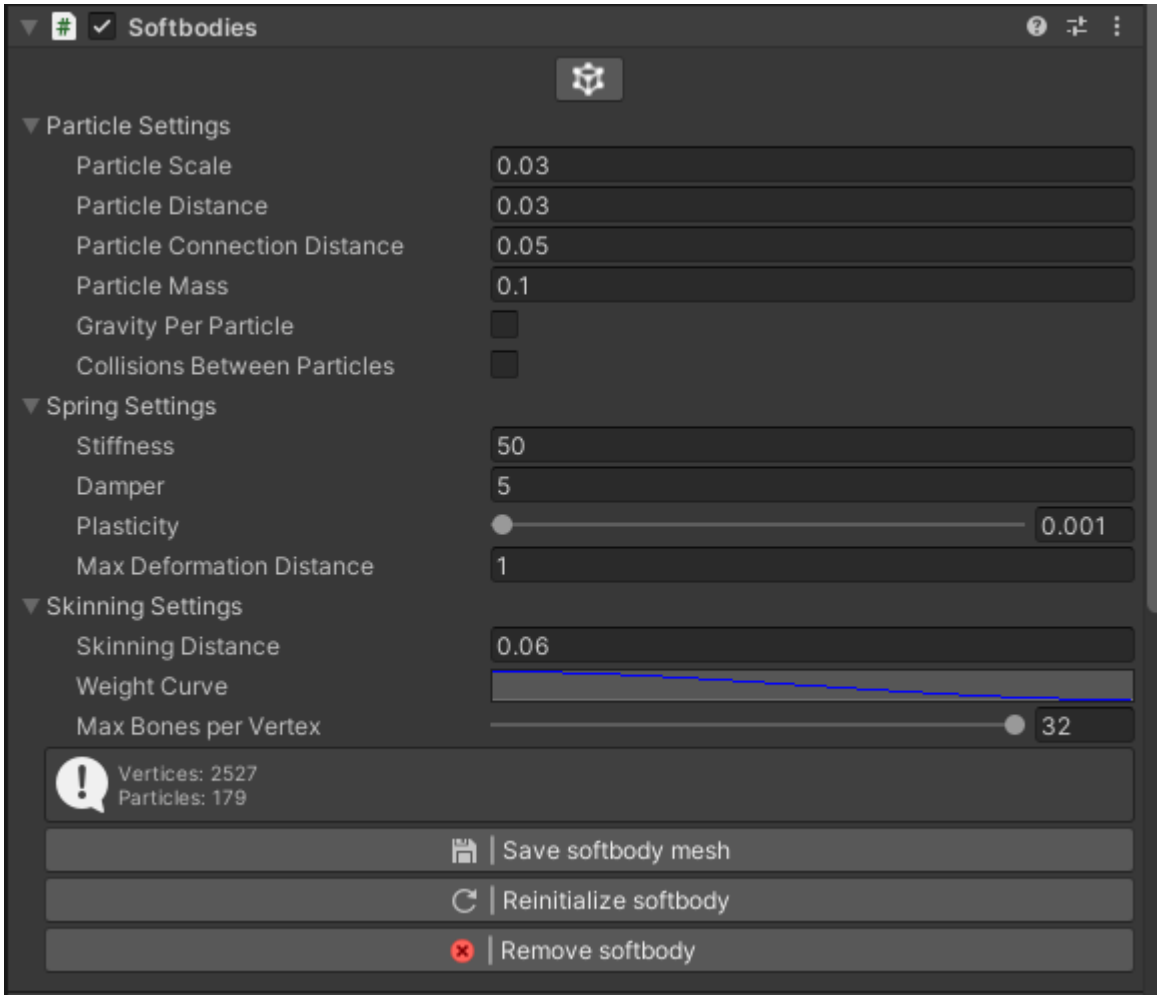
To create a soft body interaction, you need to click **Add Component** and navigate to **MAGES** → **Mesh Deformations** → **Softbody**.

Below there is a short description and the properties of the Mass Spring Softbody:

Mass Spring Softbody

MAGES SDK implements a MassSpring softbody simulation. It is suitable for all kinds of simulation both for skinned and static meshes, producing visually pleasing results.

Below are the properties of the script and what they do.


Softbodies
Particle Settings
Particle Scale
0.03
Particle Distance
0.03
Particle Connection Distance
0.05
Particle Mass
0.1
Gravity Per Particle
Collisions Between Particles
Spring Settings
Stiffness
50
Damper
5
Plasticity
0.001
Max Deformation Distance
1
Skinning Settings
Skinning Distance
0.06
Weight Curve
Max Bones per Vertex
32
Vertices: 2527
Particles: 179
Save softbody mesh
Reinitialize softbody
Remove softbody

Property	Description
`float ParticleScale`	The scale of each particle.
`float ParticleDistance`	The minimum distance two particles can be appart.
`float ParticleConnectionDistance`	The max distance of two connected particles. Increasing this value will increase realism in exchange for performance. It must be greater than `ParticleDistance`.
`float ParticleMass`	The mass of each particle.
`bool GravityPerParticle`	Wheter to calculate the gavity per particle (when enabled) or to calculate it based on the rigidbody of the object's center (when disabled). Leave disabled for increased simulation stability.
`bool CollisionsBetweenParticles`	Can the particles collide with each other?.
`float Stiffness`	The stiffness of the softbody springs. Increasing this value will make for a harder to deform object.
`float Damper`	The damper of the softbody springs. Decreasing this value will make for a more bouncy object.
`float Plasticity`	The spring tolerance. Large values will make for permanent deformations instead of elastic.
`float MaxDeformationDistance`	The maximum deformation the softbody can have. The maximum distance a particle can be from its anchor.
`float SkinningDistance`	The maximum distance a particle can be away from a vertex in order to affect it.
`AnimationCurve WeightCurve`	The skinning function.
`int MaxBonesPerVertex`	The maximum number of particles a vertex can be affected from.

Saving Utilities

All the skinning scripts have a tab called **Saving Utilities**. By selecting it you will a button in order to Save the softbodified mesh as an asset. Simply press it and **choose a path in the project's Assets directory** as well as a suitable name to save the softbodified mesh as.

Network Limitations

Limitations

Due to the lack of an authoritative server in the system the MAGES SDK use, there are some limitations in the softbody interactions when in cooperative environments. The softbody physics will be handled by the machine of the client who last grabbed a particle of the softbody. The other clients will not be able to interact, until grabbing some particle and therefore taking authority of the simulation to their machine.

Setup

Warning: Photon PUN2 must first be imported, if not already done, in order to add network support. Use the MAGES Helper menu to import it.

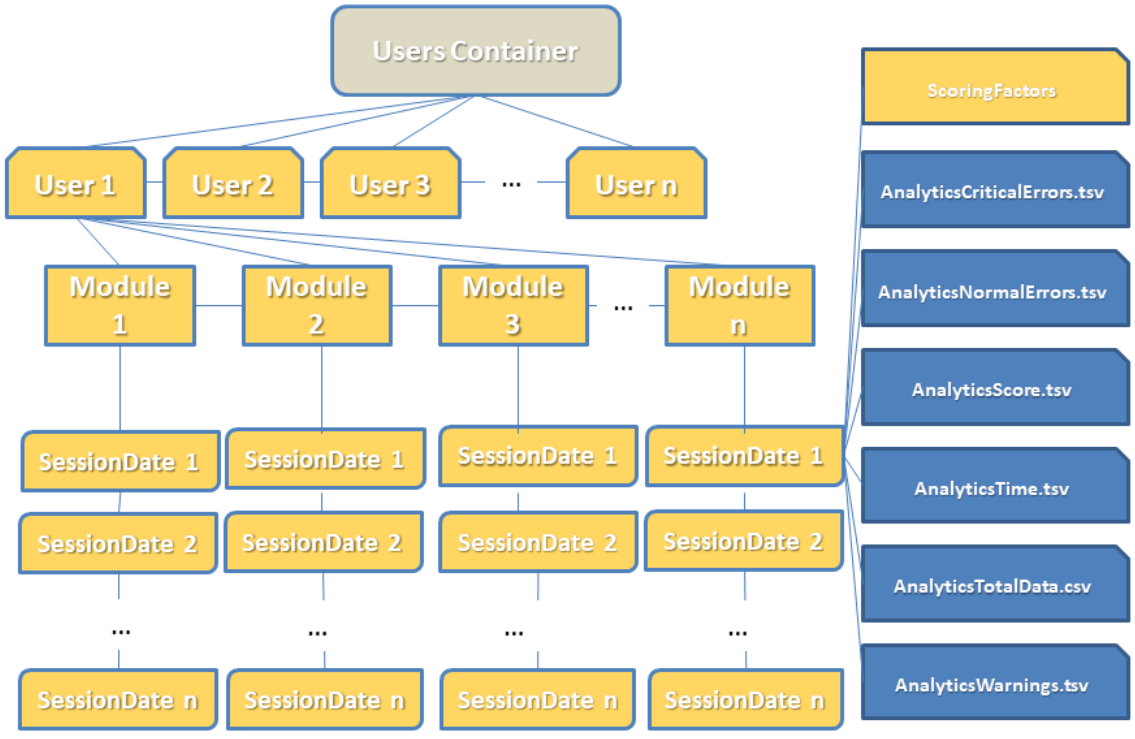
In order to add multiplayer support to a softbody object simply press the **Configure for Network** button found at the end of the simulation script of the softbody.

In order to limit the network bandwidth you may increase the **Movement and Rotation threshold** of the **MAGES Sync Transform Photon** and the **Net Multi Transform Photon** script. Keep in mind that as you increase these values the more the max difference in transform the softbody may have between devices will be.

5.8 Analytics

5.8.1 Analytics File System

A high-level overview of MAGES analytics file system is depicted in the image below.



Inside Users container (i.e., the root node) the following structure exists:

1. User Folders

One folder per user. Each of these folders contain all necessary files of their respective user progress.

2. Module Folders

   Each user folder contains one or more modules folder. Module folders are named after respective module names. Module folders are generated when user runs a module for the first time.

3. SessionDates Folders

   These folders are contained inside their respective module folder. Each of the session folders represents a single user session of the module. A session folder is created when the user finishes a complete playthrough of the specific module.

   Generally, we store the following data for each user:

   3.1. Number of critical errors in each module session and the name of the action, where they occurred.

   3.2. Number of non-critical (or normal) errors in each module session and the name of the action, where they occurred.

   3.3. The score of each action in each module session.

   3.4. The time that the user needed for each action in each module session, measured in seconds.

   3.5. The total data (all errors, critical errors, warnings, final score) for each module session.

Stored data

The content that is saved in the files mentioned above, concerns the progress of users in each step of our module.

A basic example is provided below.

Errors & Warnings Data

Files under this category concern errors and warnings occurred during users playthrough.

Errors and warnings are structured similarly as shown in the image above.

The structure is intuitive and self-explanatory. Each action the user obtains a warning/error is kept in track alongside the amount of errors.

Action	Errors
Question Action Example	1
Assemble Knossos Or Sponza Action	0
Remove Jar Example	0
Remove Jar With Tool Example	2
Apply Glue Action	0
Insert Plug Action	0

Scoring Data

We keep track of users' score for each action in the module.

Specifically, the name of the action is saved, along with the score. Score variables are integers within the range of $score \in [0, 100]$.

An example file content of users' score is exhibited in the table below.

Action	Score
Question Action Example	100
Assemble Knossos Or Sponza Action	100
Remove Jar Example	100
Remove Jar With Tool Example	50
Apply Glue Action	80
Insert Plug Action	90

Timing Data

The time users spent on each action; is another important variable we monitor. More specifically, action names are accompanied by a double-precision number, which represents the time user has spent on that action.

Timings are measured in seconds. An example is shown in the table below.

Action	Time
Question Action Example	10.25
Assemble Knossos Or Sponza Action	83.70
Remove Jar Example	20.48
Remove Jar With Tool Example	115.80
Apply Glue Action	30.67
Insert Plug Action	17.61

Accumulated User Data

For convenience, we keep the concrete form of the data presented above, augmented with relative, yet necessary, information per session.

Namely, we keep track of the following information; the session date (DD/MM/YY), time session ended (HH:MM:SS), module difficulty, handedness, total score, total time, total errors (normal, critical) and warnings, number of current session (identifier), and total time of session.

Username	IP	Session Date	Session Time	Difficulty	Total Score	Total Time In Seconds	Total Errors	Total Critical Errors	Total Warnings	Total Time
Username	IP	06/12/2020	16:00:44	Easy	36	484	0	1	0	00:08:04

Scoring Factors

In order to calculate and store scoring information more precisely and for better and more detailed presentation of the data to the users, we also keep another type of information called Scoring Factors.

There are different kinds of scoring factors, some of them are the same across all of our modules (that would be the ones that concern errors, critical errors and warnings) and some others that are a bit more specific to module actions (for instance, for a module that contains question actions, there will be different scoring factor for those actions).

Error	100
Object has been contaminated!	
Error	30
2	-1
Goggles can be recycled!	
Critical Error	0
1	-1
You are entering without PPE	

The data shown above are the scoring factors for a specific action of a module.

The first line states the word "Error", which means that this scoring factor concerns an error of this action. The integer "100" is the score credited to this factor. The "0", which is located below the word "Error" in this example, is the number of times the users made that error. "-1" means that this error can be made infinite times (or that there is not any limit to the number of times that this error can be made).

In case there is a limit, that number may be any positive integer. Finally, the name of the specific error that this scoring factor represents is given (in this example "Object has been contaminated").

An action can have one or more scoring factors. In the case of our example, this specific action had three different scoring factors. The first one is the one we just finished presenting. The rest follow exactly the same logic.

5.8.2 Generating Analytics

As discussed thoroughly in *Analytics File System*, Analytics expect certain assessment formatting (e.g., scoring data, accumulated user data, etc.) and produce *certain* output.

ORamaVR provides a simple out-of-the-box solution for generating analytics for your products. In detail, analytics are per action and have to be explicitly specified for each action in your storyboard.

Note: Recall, Actions are generated from the SceneGraph Editor.

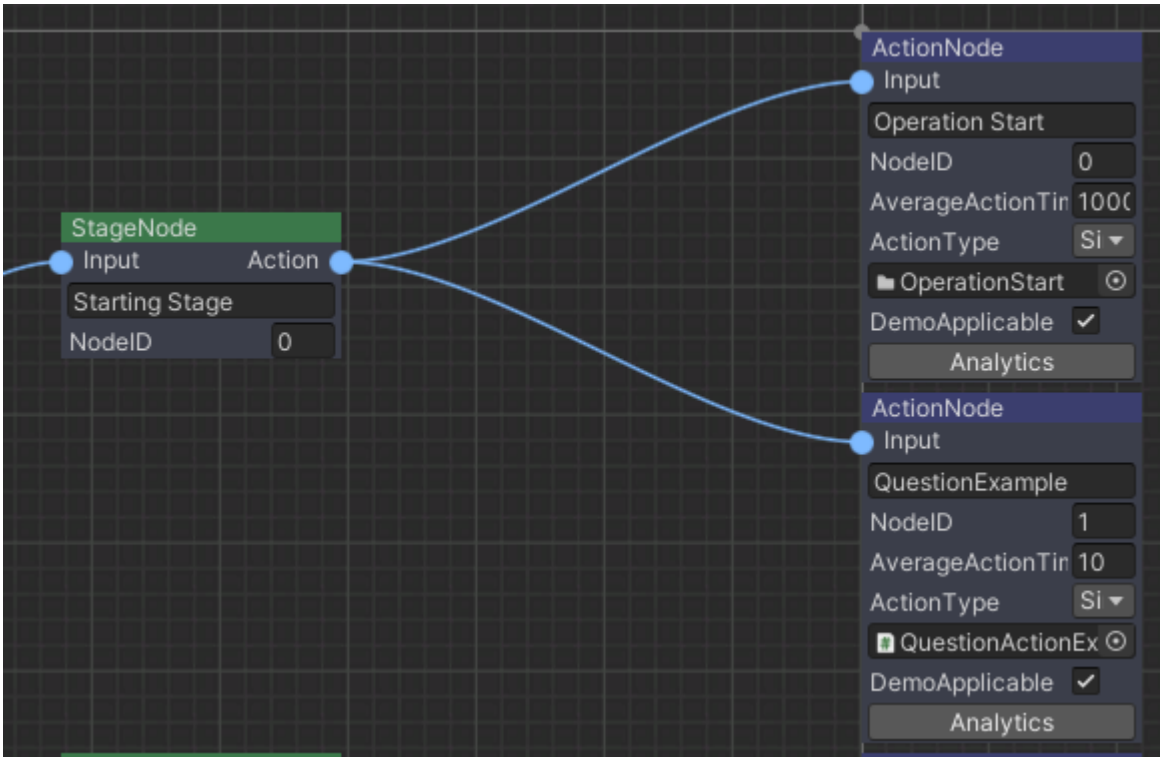
Visual Editor

To specify the assesment (i.e., analytics recording) for each Action, start by opening the SceneGraph Editor through the **MAGES menu**.

- *MAGES/SceneGraph Editor*

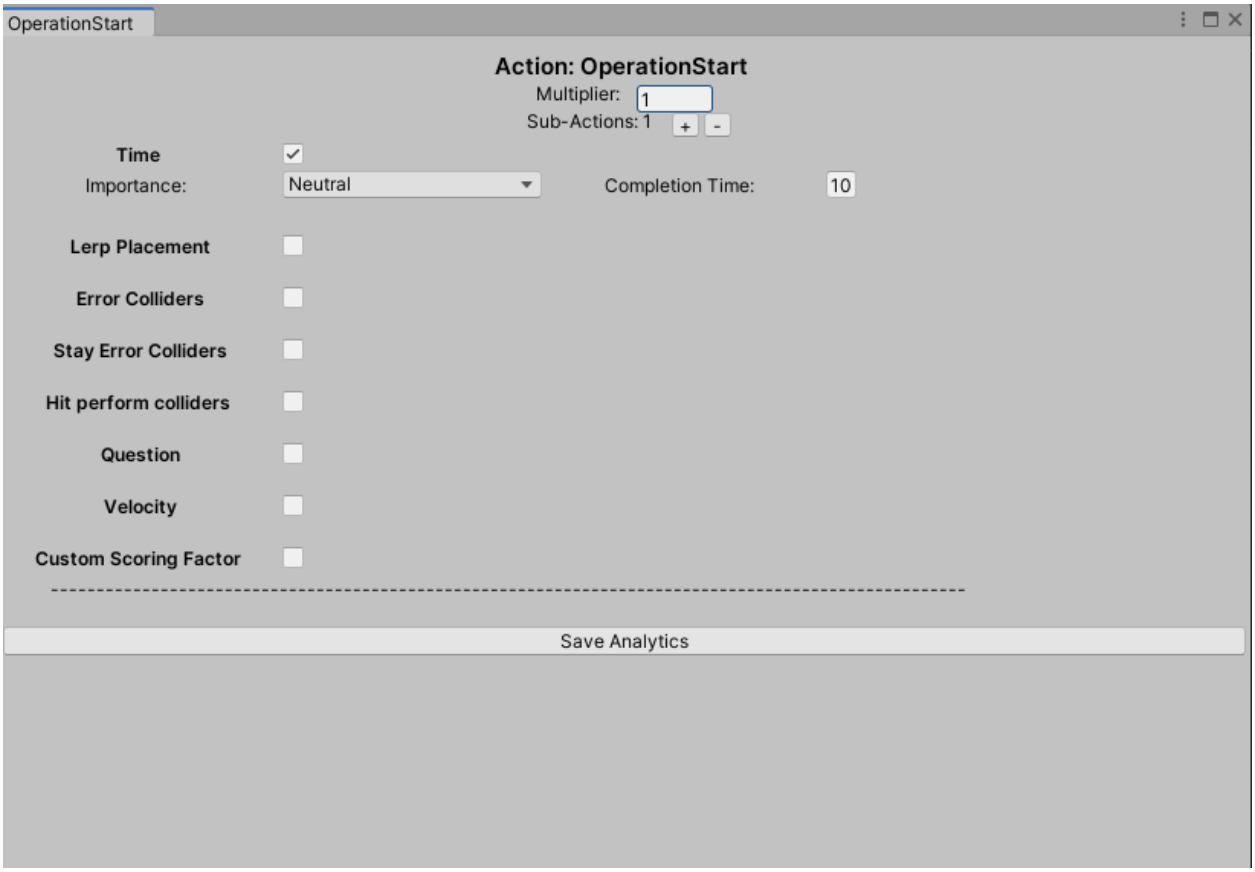
Proceed to Load the Storyboard XML by selecting **File/Load** and navigate under `Assets/Resources/Storyboard/platform/`

You will be presented with a similar view:



Under each Action there is an **Analytics** button.

Press on this button and the following window will open



In the Analytics window you can specify all scoring factors for the current Action.

Scoring factors

As described in *Analytics File System*, MAGES SDK supports a variety of predefined scoring factors.

Current available scoring factors are enumerated below in an algorithmic manner **following the order of the Scene-Graph Editor**:

1. Time

Input: MaxTime
Output: Score

```
1 OnEachFrame()
2 time_counter : time_counter + (time since last frame)
3
4 OnPerform()
5 if time_counter > MaxTime
6       diff : time_counter - MaxTime
7       Score : 100 - (diff * 10)
```

2. Lerp Placement

Input: scoringAngle , offsetAngle
Output: Score
OnInteractionEnd(Interactable I**)**
if I is in placementCollider
    $angle_{current}$: get I rotation
    rot_{diff} : $|angle_{current}| - scoringAngle$
    $angleRange$: offsetAngle - scoringAngle
    Score : $\frac{rot_{diff} * 100}{angleRange}$

3. Error Colliders

Input: Avoid_Colliders
Output: Score
OnCollision(Interactable I **, Collider** C**)**
if C in Avoid_Colliders
    errors++
    Score : Score - lostPointsForError

4. Stay Error Colliders

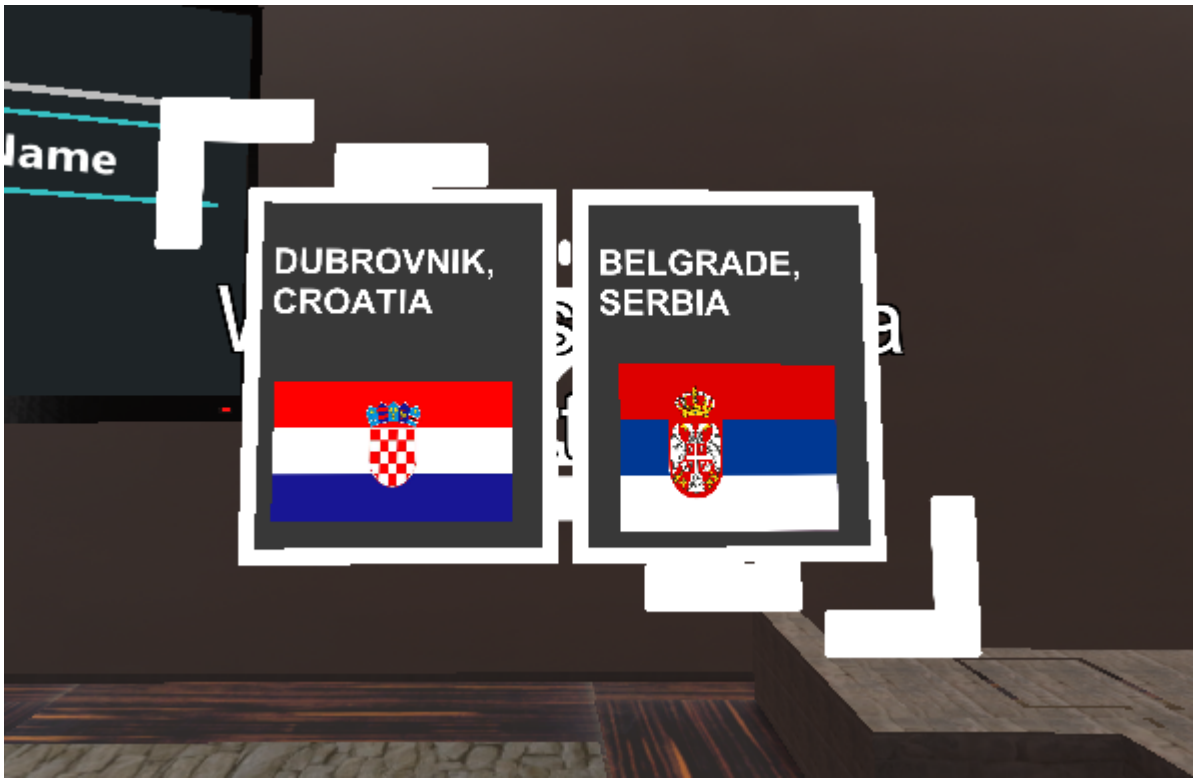
Input: Interactable I_1
Output: Score
OnUseInteractable(Interactable I_2**)**
if I_2 is used on I_1
    if I_1 is not currently held
        errors++
        Score : Score - lostPointsForError

5. Hit Perform Colliders

Input: Colliders, Colliders_{size}
Output: Score
OnCollision (Interactable I, Collider_i)
if Colliders cotains Collider_i
    remove $Collider_i$ from Colliders

OnPerform()
leftColliders : get size of Colliders
Score : $\frac{leftColliders*100}{Colliders_{size}}$

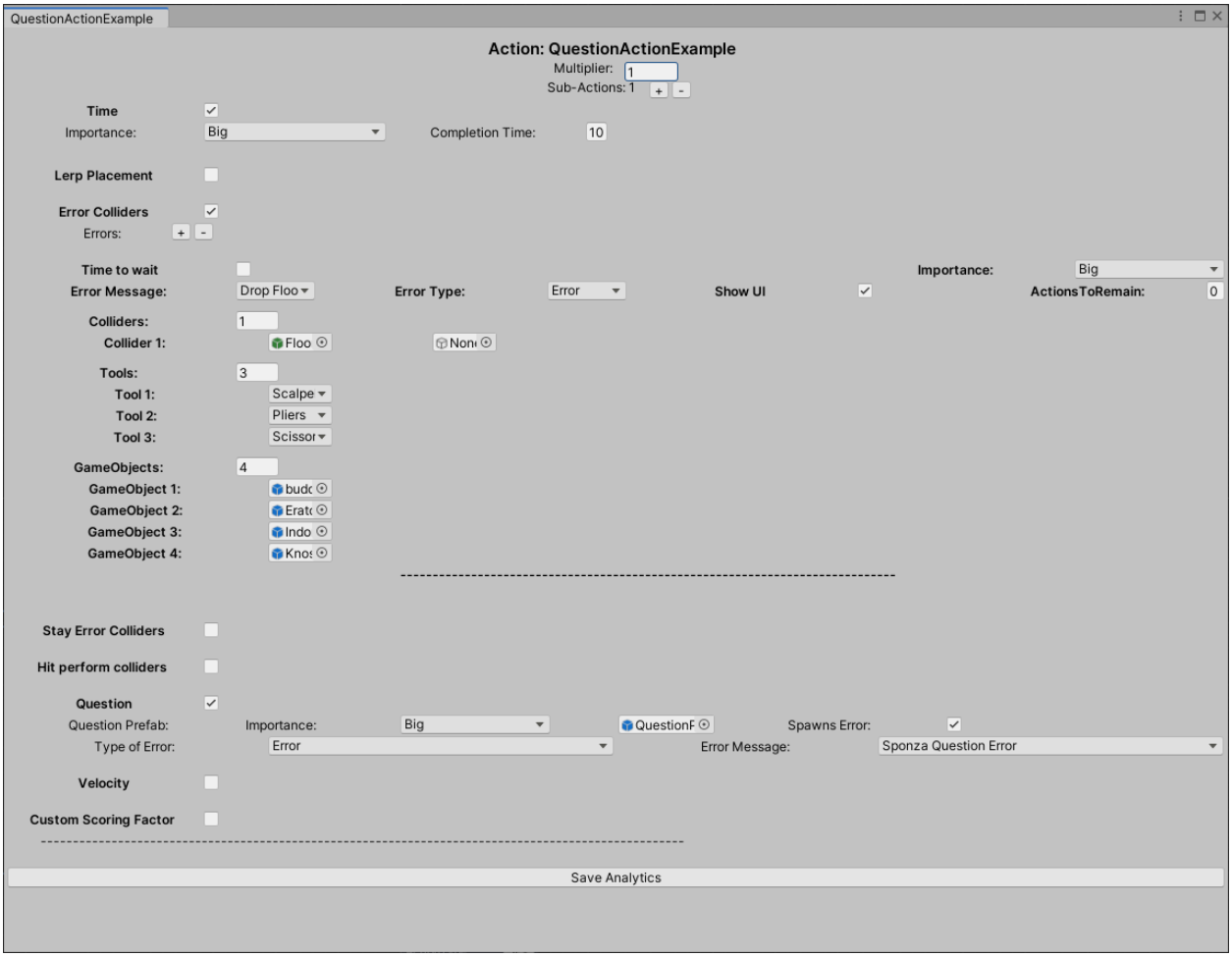
6. Question



7. Velocity

Input: MaximumVelocity
Output: Score
1 **OnEachFrame(Interactable** I**)**
2 $current_{velocity}$: get velocity from I
3 if current$_{velocity} > MaximumVelocity$
4 errors++
5 Score : Score - lostPointsForError

Using the above combinations you can produce an output similar to the following:



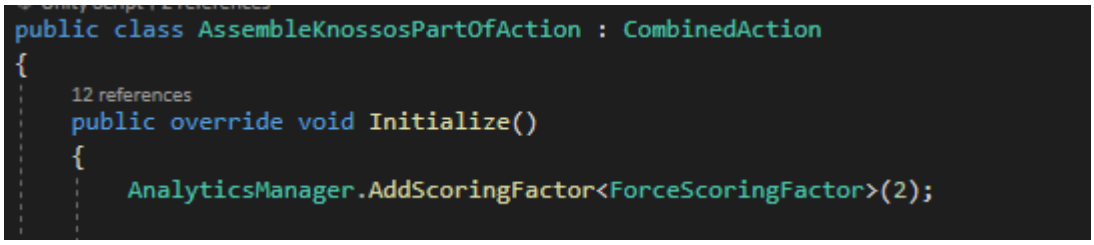
Finally, click the **Save Analytics** button down in the editor window to save your changes.

> **Warning:** If you forget to *Save your Analytics* for each action, the changes will get discarded.

Custom Scoring Factor

Custom Scoring Factor is Work in Progress (WIP) and cannot be modified from the Editor. However, you can implement it directly in code.

You can implement a *Custom Scoring Factor* as in the SampleApp example below:



And the respective ForceScoringFactor script in code as follows:

```
public class ForceScoringFactor : ScoringFactor
{
    public float maximumForce = 12;

    UpdateCollisionForce _forceScript;
    float _currentForce;
    GameObject _knossosBackPart;
    ForceScoringFactor _forceSf;
    int _score;
    LanguageTranslator _errorMsg;

    public override ScoringFactor Initialize(GameObject g)
    {
        _forceSf = g.AddComponent<ForceScoringFactor>();
        _knossosBackPart = GameObject.Find("BackPartHitMallet(Clone)");
        _forceScript = _knossosBackPart.AddComponent<UpdateCollisionForce>();
        _forceScript.Init(maximumForce,true);
        return _forceSf;
    }

    public override float Perform(bool skipped = false)
    {
        Destroy(_forceSf);
        Destroy(_forceScript);
        if (skipped) return 0;
        int errors = _forceScript.GetErrorsCounter();
        _score = 100 - (errors * 20);

        return Mathf.Clamp(_score,0,100);
    }

    public override void Undo()
    {
        _score = 0;
        Destroy(_forceSf);
        Destroy(_forceScript);
    }

    public override object GetReadableData()
    {
        ScoringFactorData sfData = new ScoringFactorData();
```

(continues on next page)

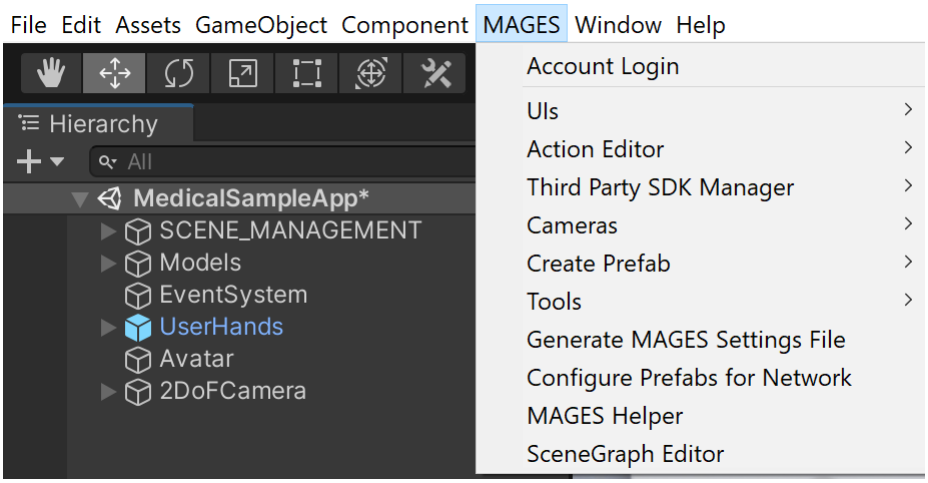
(continued from previous page)

```
        sfData.score = _score;
        sfData.outOF = (int) maximumForce;
        sfData.type = "Force Scoring Factor";
        sfData.scoreSpecific = (int) _forceScript.GetCollisionForce();
        sfData.errorMessage =  InterfaceManagement.Get.GetUIMessage(_errorMsg);

        return sfData;
    }
}
```

Note: Notice how our custom scoring factor extends **ScoringFactor**

5.9 MAGES Menu

One of the scripts provided includes code for additional functionality in Unity's progress bar.



1.Account Login This option saves the developers account to be able to run the application inside the Unity Editor.

2.UIs This option provides a variety of features regarding the UI components. From the addition and modification of text and speech UI elements to the management of different languages.

3.Action Editor This is the editor to generate the Action scripts.

4.Third Party SDK Manager This option contains different functionalities for managing third party SDKs needed for the application to operate using different headsets.

5.Cameras This option contains the available VR and non-VR cameras.

6.Create Prefab This option creates an empty gameobject in the scene with all the appropriate components attached depending on the type of prefab selected.

7.Tools This menu is responsible to generate the `ToolsEnum.dll` and create new tools.

8.Generate MAGES Settings File This options creates a MAGES asset file containing the appropriate input fields to fill in the settings of your MAGES application.

9.Configure Prefabs for Network This option searches all prefabs in the specific path displayed. If they do not contain the appropriate components to be able to connect to the network, it automatically attaches them.

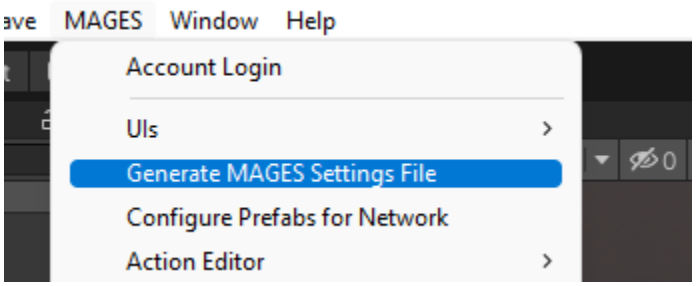
10.MAGES Helper The window that appears on startup, containing important settings for MAGES™ SDK.

11.SceneGraph Editor This option opens the SceneGraph Editor to edit the scenegraph tree (Lesson, Stages, Actions).

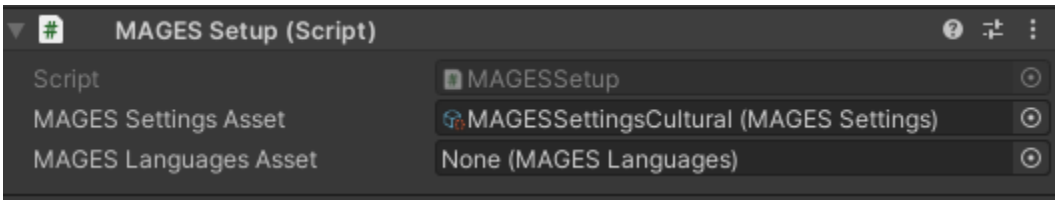
5.10 MAGES Settings

The MAGES settings asset, serves as a configuration file for each application made with the MAGES SDK.

This asset can be created from the *MAGES_Menu*.



In order to specify which MAGES Settings asset will be used, you have to reference it in the MAGES Setup script which can be found in the SceneManagement gameobject on the *MAGES Setup* script.



These fields are described below.

5.10.1 UI Settings

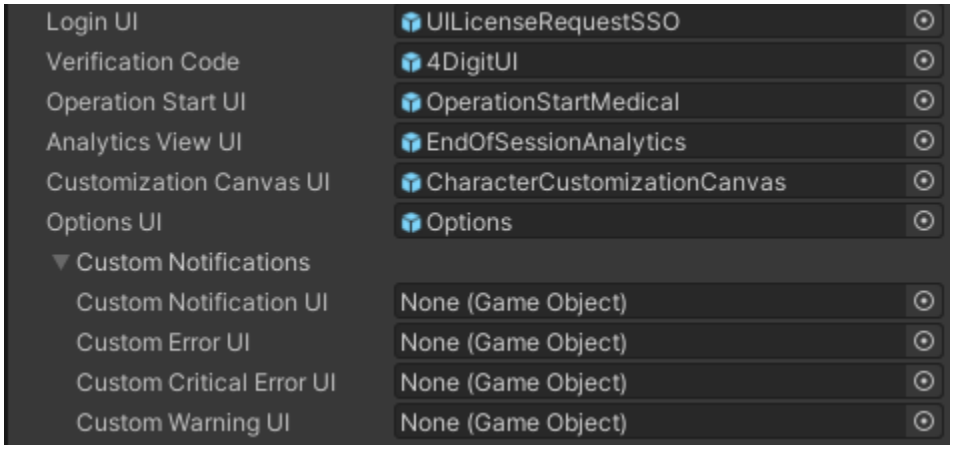


Fig. 1: In this section developers can find and configure the settings for MAGES UIs.

- **Login UI:** Spawns at the start of the application and will prompt the user to enter his credentials in order to login.

- **Verification Code:** In case of Login using SSO this UI will spawn to show to the user his verification code.
- **Operation Start UI:** Spawns at the start of the operation.
- **Operation End UI:** Spawns at the end of the operation.
- **Analytis View UI:** Spawns at the end of the session, showing to the user his analytics overview as well as informing him/her of any errors.
- **Customization Canvas UI:** Spawns after the Login UI and guides the user on how to create his/her avatar.
- **Options UI:** Spawns on the right thumbstick press and users can find there options such as skip or undo current action.
- **Custom Notifications:** Here developers can customize MAGES notifications UI. If a prefab is specified as a custom UI it will replace MAGES default notification UIs.

5.10.2 General Settings

Fig. 2: In this section developers can find and configure general settings for MAGES.

- **Product code:** String identifier for the application.
- **User Login:** Activates user's login.
- **Enable Microphone:** Enable's the user Microphone in coop.

5.10.3 Scenegraph Settings

- **Operation XML:** The XML that will be loaded at runtime.

5.10.4 API Calls Settings

- **Login Identity Url:** URL used for reaching the login identity server.
- **Login Loop back Url:** URL used for specifying which login service to use.
- **Login Client Secret:** Secret code used for the checkout of users.
- **Login ClientID:**
- **Profile Upload Url:** URL used to upload the profile of the user.
- **Register Url:** URL used to redirect user to create a new account.
- **Analytis Upload Url:** URL used to upload the analytics files.
- **Multiplayer Upload Url:** URL used to upload the analytics files of a coop session.
- **Vitals Upload Url:** URL used to upload the Vitals values, exported from the vital's manager.
- **Questionnaire Upload Url:** URL used to upload the results of the questionnaire.

Fig. 3: In this section developers can configure the settings for connecting with their cloud services.

5.10.5 VR Recorder Settings

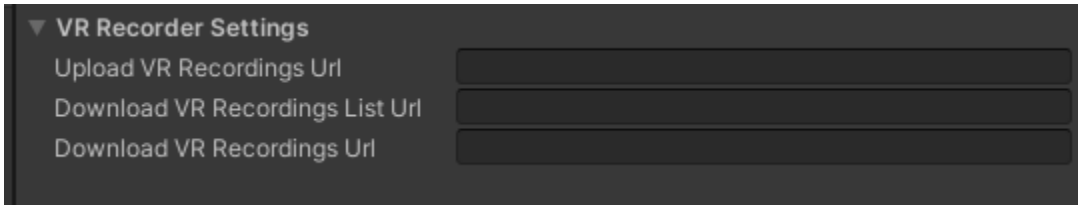


Fig. 4: In this section developers can configure the settings for uploading/downloading their recording to/from the cloud.

- **Upload VR Recording Url:** URL used for uploading the VR Recorder files.
- **Download VR Recordings List Url:** URL used for downloading the list of all available recordings.
- **Download VR Recording Url:** URL used for downloading a specific recording.

5.10.6 Networking UI Settings

- **Networking Info UI:** This UI shows information about the current networking session, such as number of users.
- **Networking Sessions UI:** This UI is used to create coop sessions and connect to them.

5.10.7 Session Start prefab

5.11 MAGES UIs

In this section we will explain the UIs that already come with the MAGES SDK and what they perform after you click them.

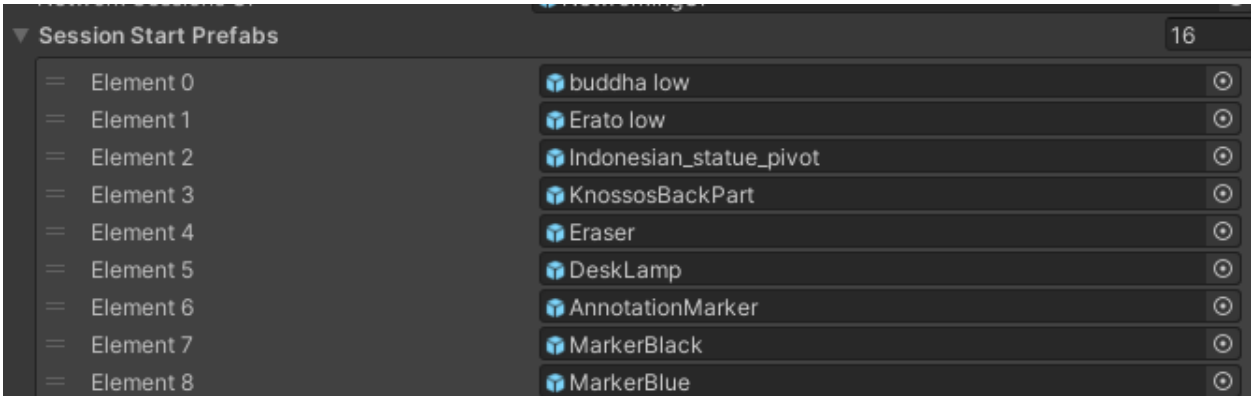
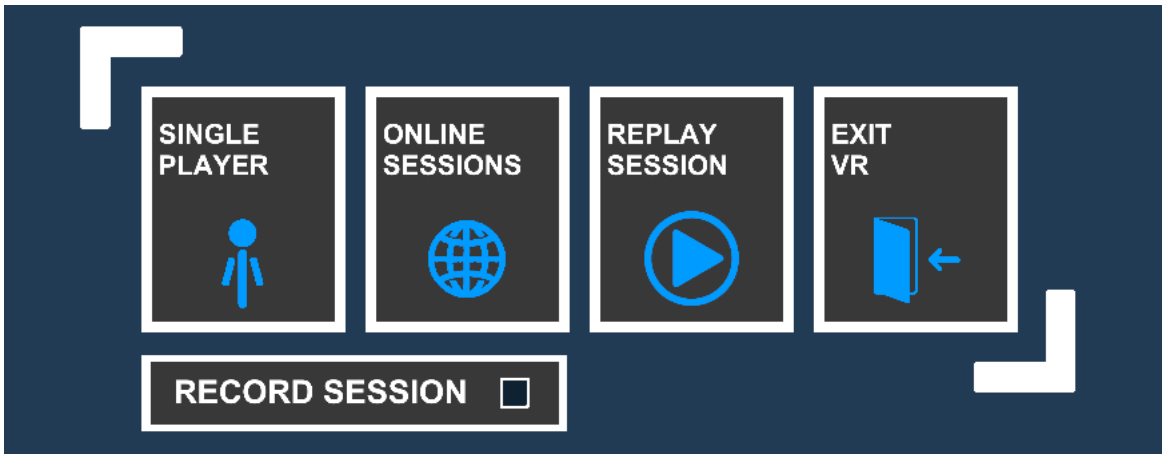


Fig. 5: In this section, developers can add prefabs that will be spawned at the start of the operation.

5.11.1 Operation Start



This is the Operation Start UI, by default it is spawned with the OperationStart Action. You can assign a new Ui to pop from the MAGESSettings.asset.

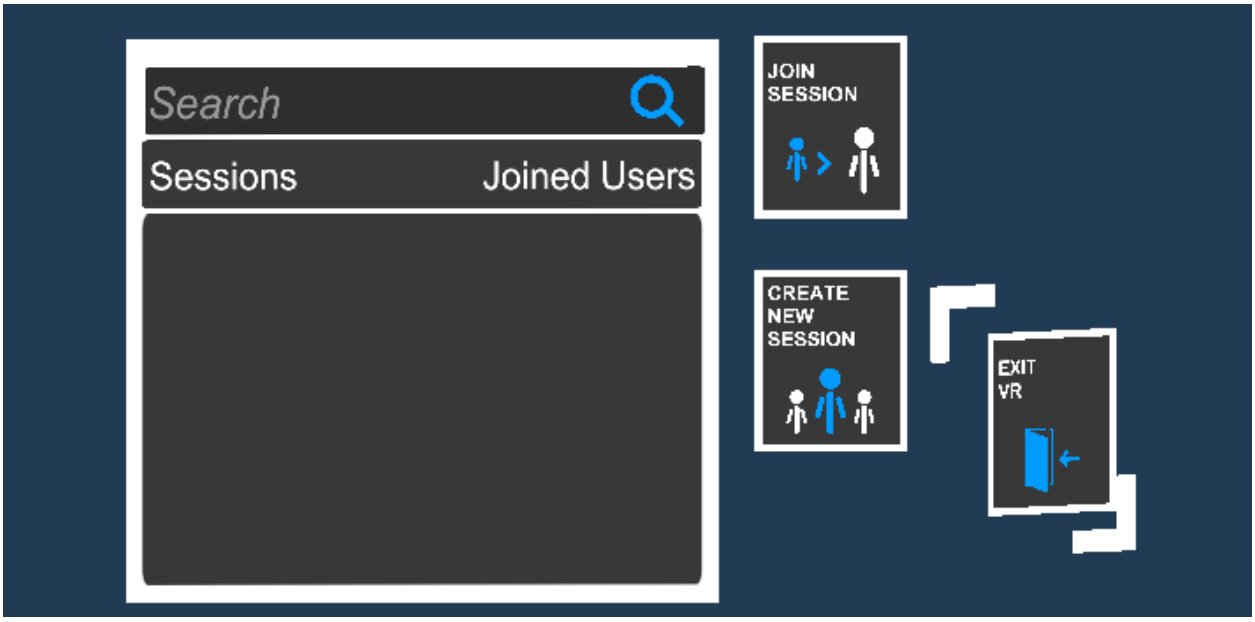
SINGLE PLAYER: Performs the Action and starts the operation.

ONLINE SESSIONS: Opens the networking panel for collaborative sessions.

REPLAY SESSION: Opens the VR Recorder panel to replay the selected session.

EXIT VR: Exits the application.

5.11.2 Networking UI

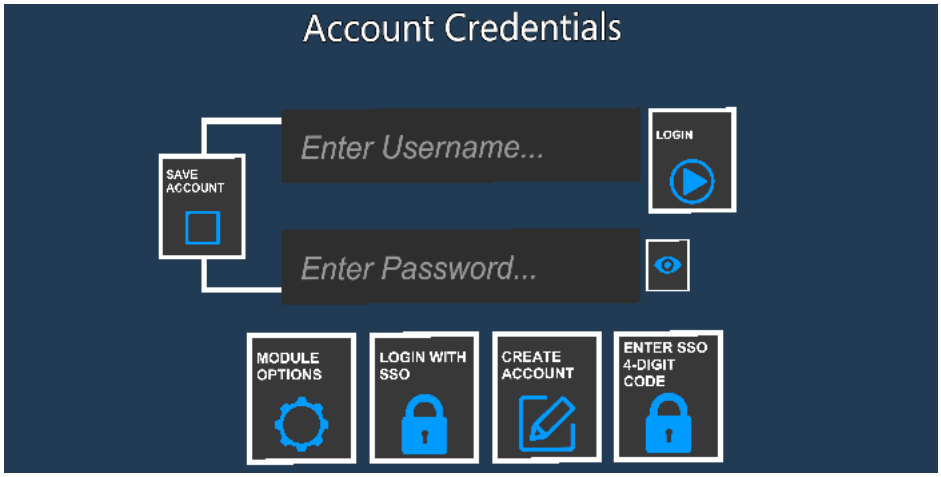


JOIN SESSION: After user has selected a session from the list, click this button to join the collaborative room.

CREATE NEW SESSION: Creates a new coop session.

EXIT VR: Exits the application.

5.11.3 Login UI



SAVE ACCOUNT: Remembers the login and password of user.

LOGIN: Logins the session if user has entered valid credentials.

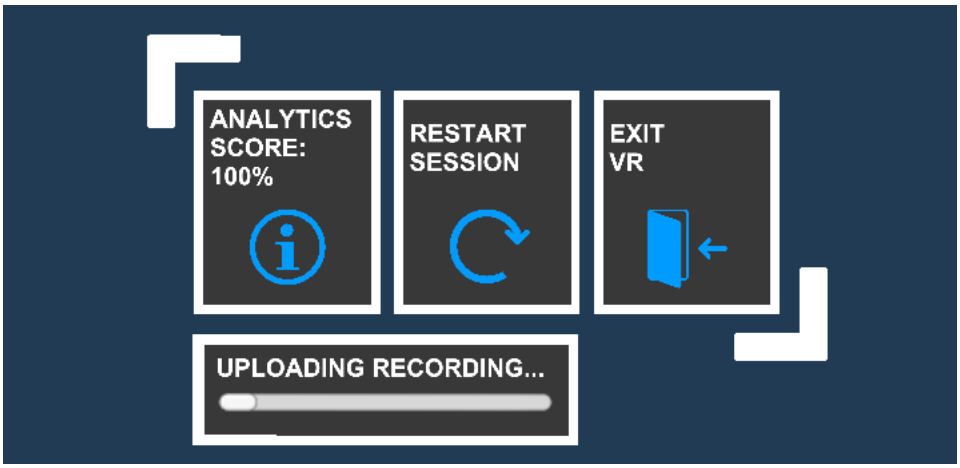
MODULE OPTIONS: Opens additional options like server region or language selection (can be customized).

LOGIN WITH SSO: Login with Single Sign Auth. Can be customized for your case.

CREATE ACCOUNT: This option can be customized and linked to your custom portal for users to create their account.

ENTER SSO 4-DIGIT CODE: Users will insert the 4-digit code they got from the SSO verification.

5.11.4 Operation Exit

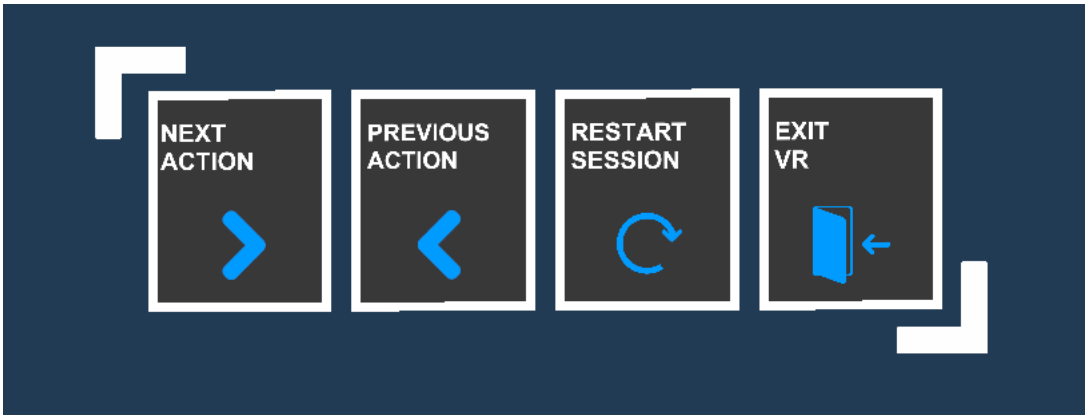


ANALYTICS SCORE: Opens the Analytics panel.

RESTART SESSION: Restarts the session.

EXIT VR: Exits the application.

5.11.5 Options



NEXT ACTION: Goes to the next Action (calls the Perform method).

PREVIOUS ACTION: Goes to the previous Action (calls the Undo method).

RESTART SESSION: Restarts the session.

EXIT VR: Exits the application.

5.11.6 Notification UIs

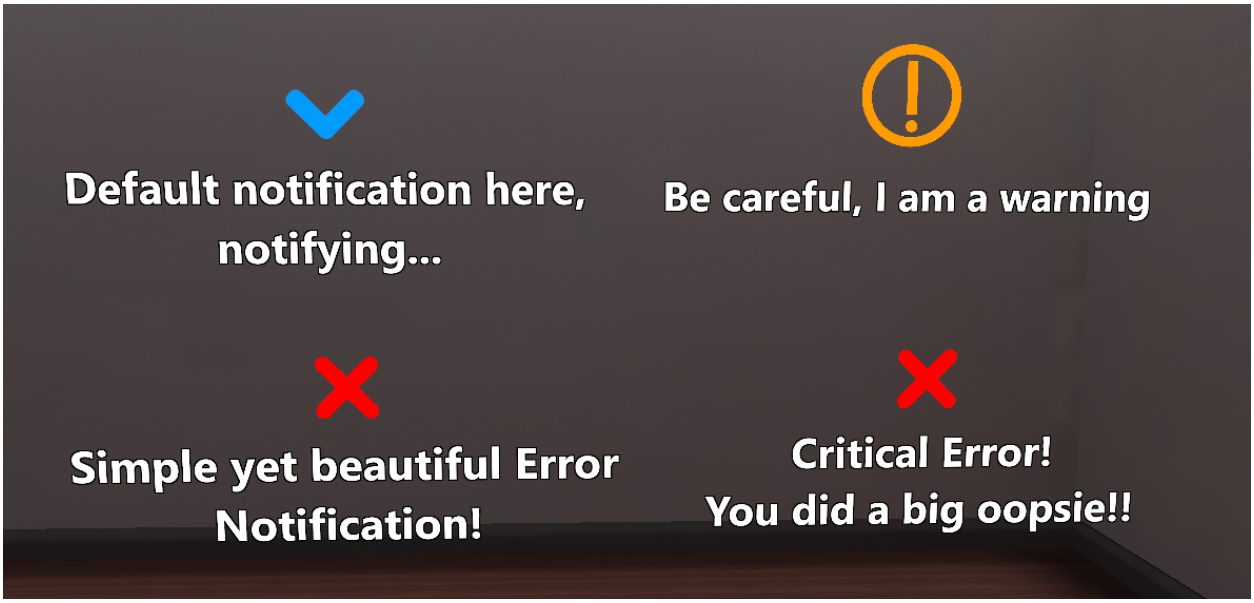


Fig. 6: MAGES notification UIs for different types of events.

Error: Pops up when the user performs an action that will trigger an error.

Critical Error: Pops up when the user performs an action that will trigger a crical error.

Warning: Pops up when the user performs an action that will trigger a warning.

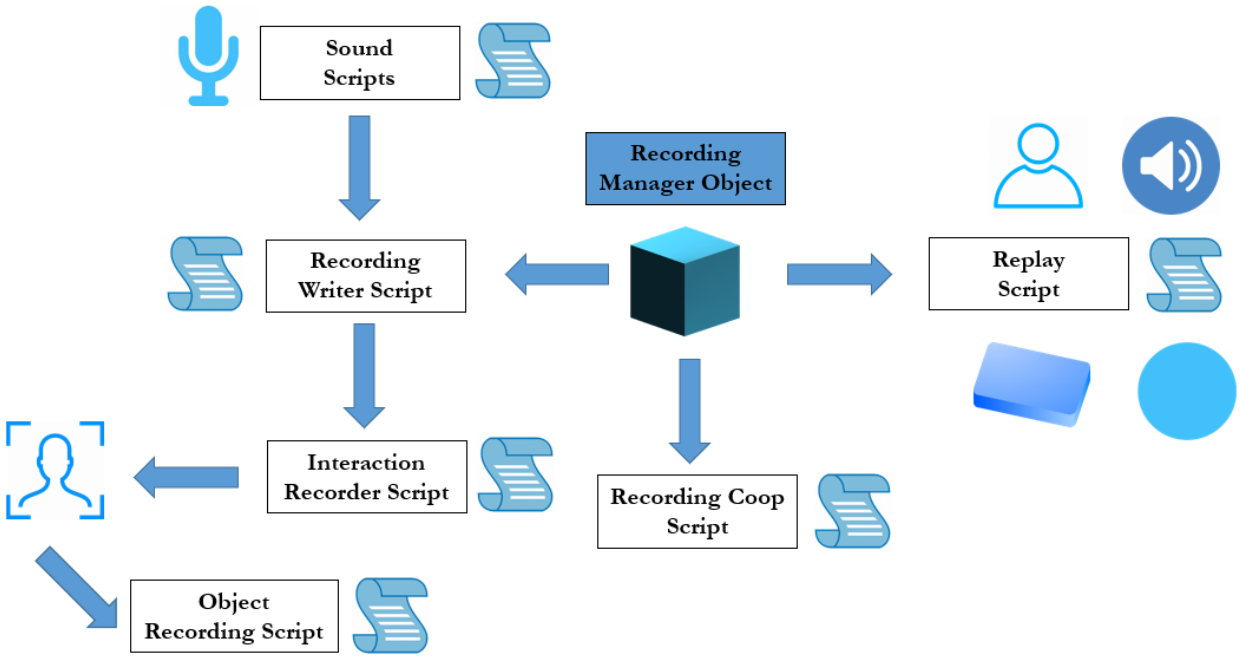
Notifications: Informs the user for a certain event.

5.12 VR Recorder

5.12.1 Introduction

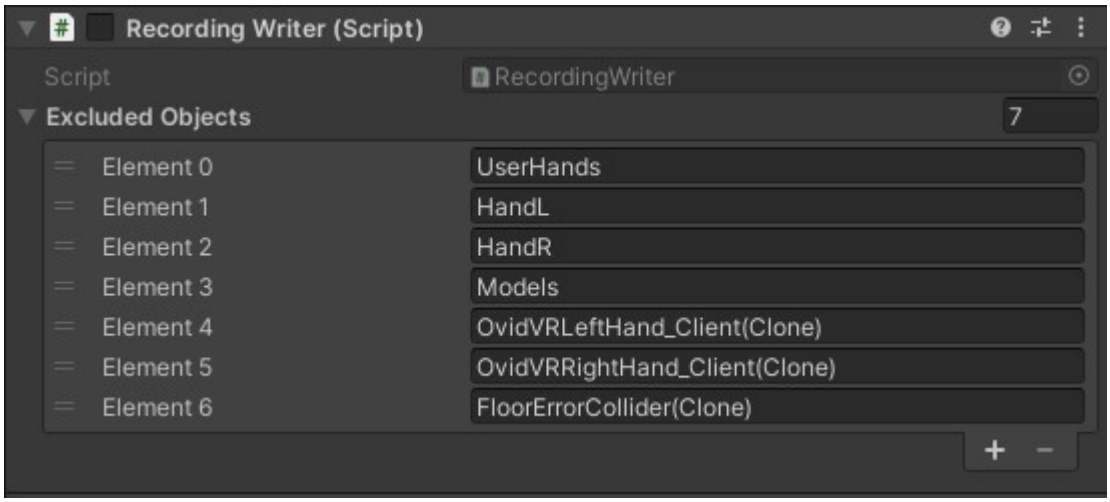
VR Recorder is a functionality that can Record and Replay an Operation, in both Single Player and Multi Player Modes. These Recordings can be synchronized with the cloud and are replayable on any device regardless of the original hardware they were recorded on.

Recording Manager



The Recording Manager GameObject is essential for the operation of the Recording and Replaying functionalities within an Operation.

Recording Writer

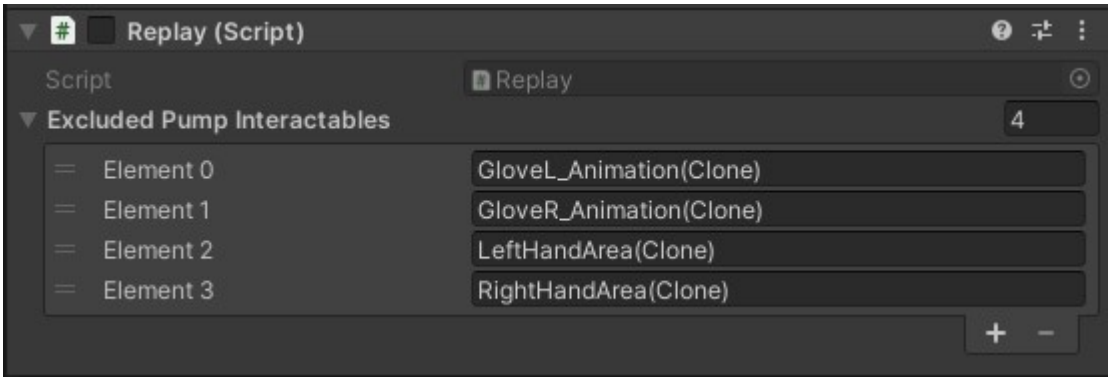


This component is used for managing the Recording of the Operation. It is responsible for creating and writing the files of the Recording for all users in the room (Multiplayer & Local).

Excluded Objects

Some GameObjects that are not essential to the recording can be ignored. These can be added to this list, and interactions all users have with them will not be recorded.

Replay

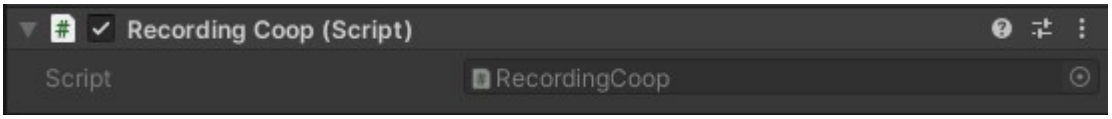


This component is used for managing the Replaying of an Operation. All access to the files of the Recording is handled by this component, and the entire Operation is controlled by it during Replay.

Excluded Pump Interactables

Some Pump Interactables are used as animation controllers and their progress is controlled via script rather than via user Input. These Interactables have to be explicitly listed here, so that they operate correctly during Replaying.

Recording Coop



This component assists in configuring Recording for Multiplayer Operations.

Other Scripts

Sound Scripts

- **Dissonance Audio Recorder, Get Audio Samples:** Records incoming network audio using Dissonance voice communications.
- **Merge Wavs, Save Wav, Sound Info:** Utilities for saving audio to WaveForm format as well as metadata regarding audio.

Object Recording Scripts

- **Propagate Recording:** Used to propagate the recording to all objects that are touched by other interacted objects.

Warning:

Known Issues:
1. Game Objects with names that begin with '<' and end with '>' (E.g. "<ObjectName>") will create issues when replayed.
2. Interactables with Drop distances bellow 0.5 cause repeated Interaction Starts and Ends, slowing down replay and desynchronizing the audio.

5.12.2 Recording

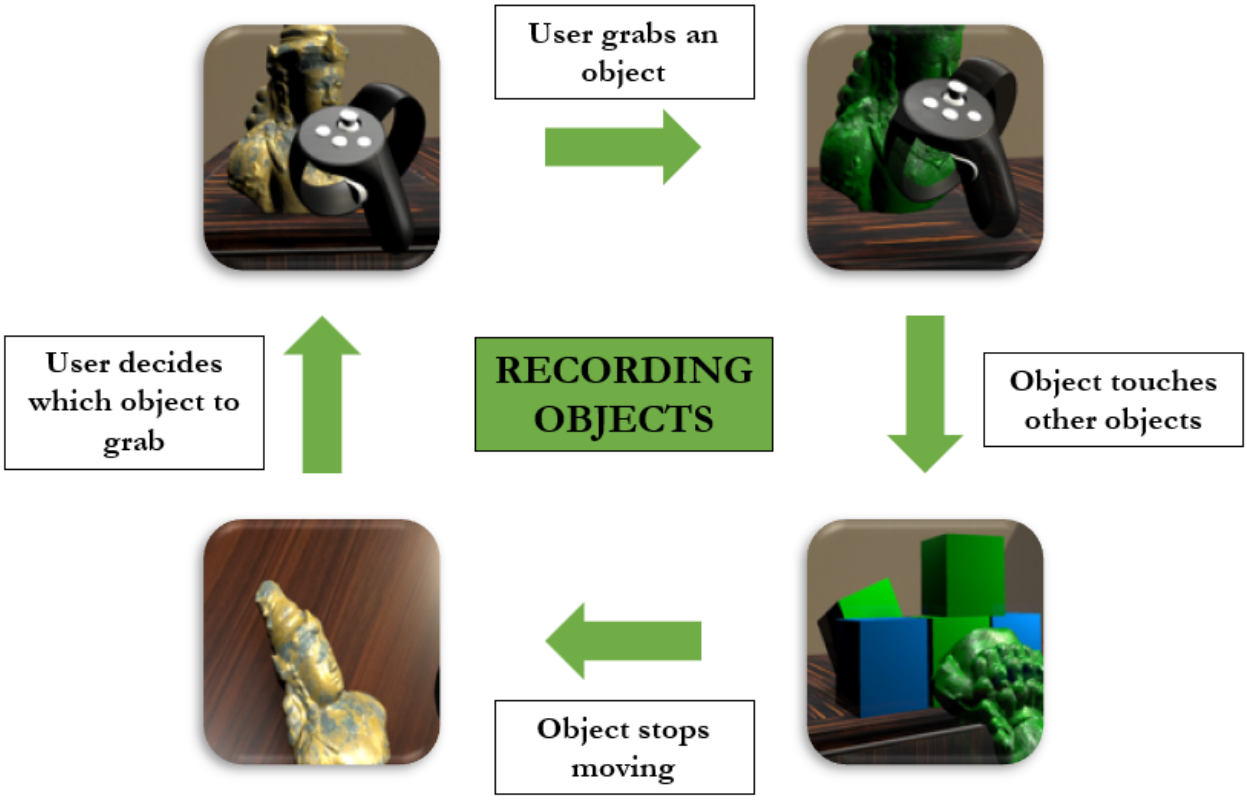
What is Recorded?

When recording is enabled, all movement of every user in the Operation is recorded, including all interactions with physics objects or Interactables, except those specified in *Introduction*.

The voice of each player is also recorded individually, including incoming voice via Dissonance in Multi Player mode.

The actions performed, as well as the complete traversal through the SceneGraph is recorded.

The objects the user interacts with might come into contact or interact with other objects and thus change their transform. The transform of all subsequent touched objects is also recorded.



Recording File structure

The file created will be under the folder **VRLogger** in the **Documents** of the user.

`C:\Users\Your_Username\Documents\VRLogger\`

Each Recording is a folder under this location, with the **date and time the recording started as a name**. The file structure of each recording is as follows:

- 2021_01_01_20_00
  - Player_0
    - MessagesCamera
    - MessagesLeftHand
    - MessagesRightHand
    - RecordingInfo
    - Sound.wav
    - TransformCamera
    - TransformLeftHand
    - TransformRightHand
  - Player_1
    - MessagesCamera
    - MessagesLeftHand

        * MessagesRightHand
        * Sound.wav
        * TransformCamera
        * TransformLeftHand
        * TransformRightHand
    - Player_2
        * ...
    - ...

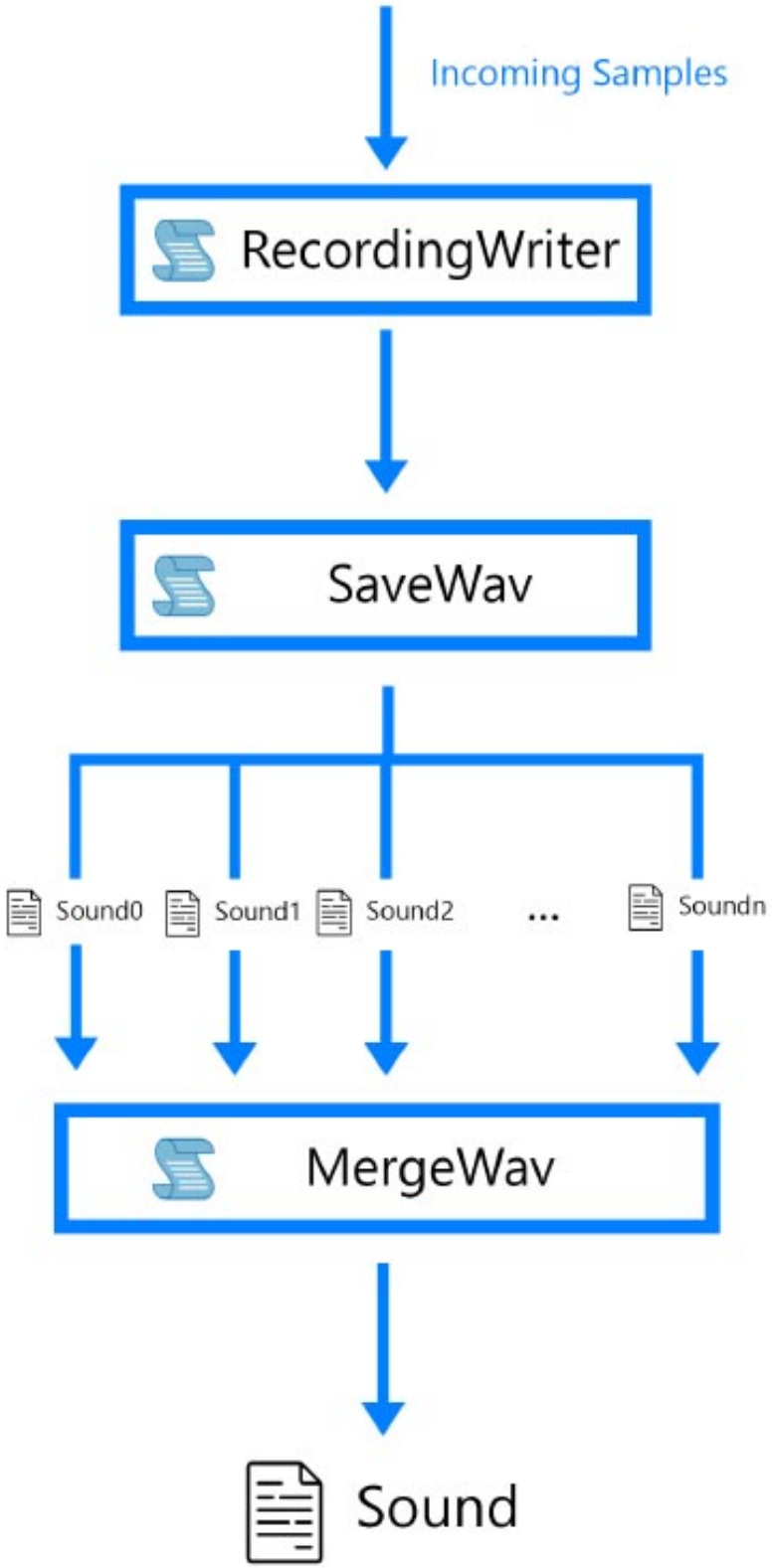
There is a Player subfolder for each user that was present in the Operation during the Recording.

- **Recording Info:** Present only in the first subfolder `Player_0`, it provides additional information regarding the recording.
- **Messages Camera, Left Hand, Right Hand:** Contains information for each of the mentioned entities. For the hands, interactions with objects, UI and button presses. For the Camera, only in the first subfolder `Player_0`, there is information regarding the traversal through the Scene Graph.
- **Sound:** contains **the microphone input** of the user that was captured during the recording, not their in-game sound.
- **Transform Camera, Left Hand, Right Hand:** Contains the transforms for each entity mentioned, as well as objects that the entity might have interacted with, following the Recording Propagation logic described in *Introduction*. For example, if a user grabs a statue, its transform is written in this file. Should he hit some other object with the statue, the other object's transform is also written here until the object is no longer moving.

Recording Sound

While Sound samples are obtained differently for the local player and network players, both follow the same procedure for saving the sound file in a recording. Sound Samples for the local player are obtained Unity's Microphone interface, and network player's sound is retrieved using Dissonance Sound communications.

The Incoming sound is initially stored as multiple files in segments of n seconds, and at the end of the recording they are merged to create the final sound file.


Incoming Samples
RecordingWriter
SaveWav
Sound0
Sound1
Sound2
...
Soundn
MergeWav
Sound

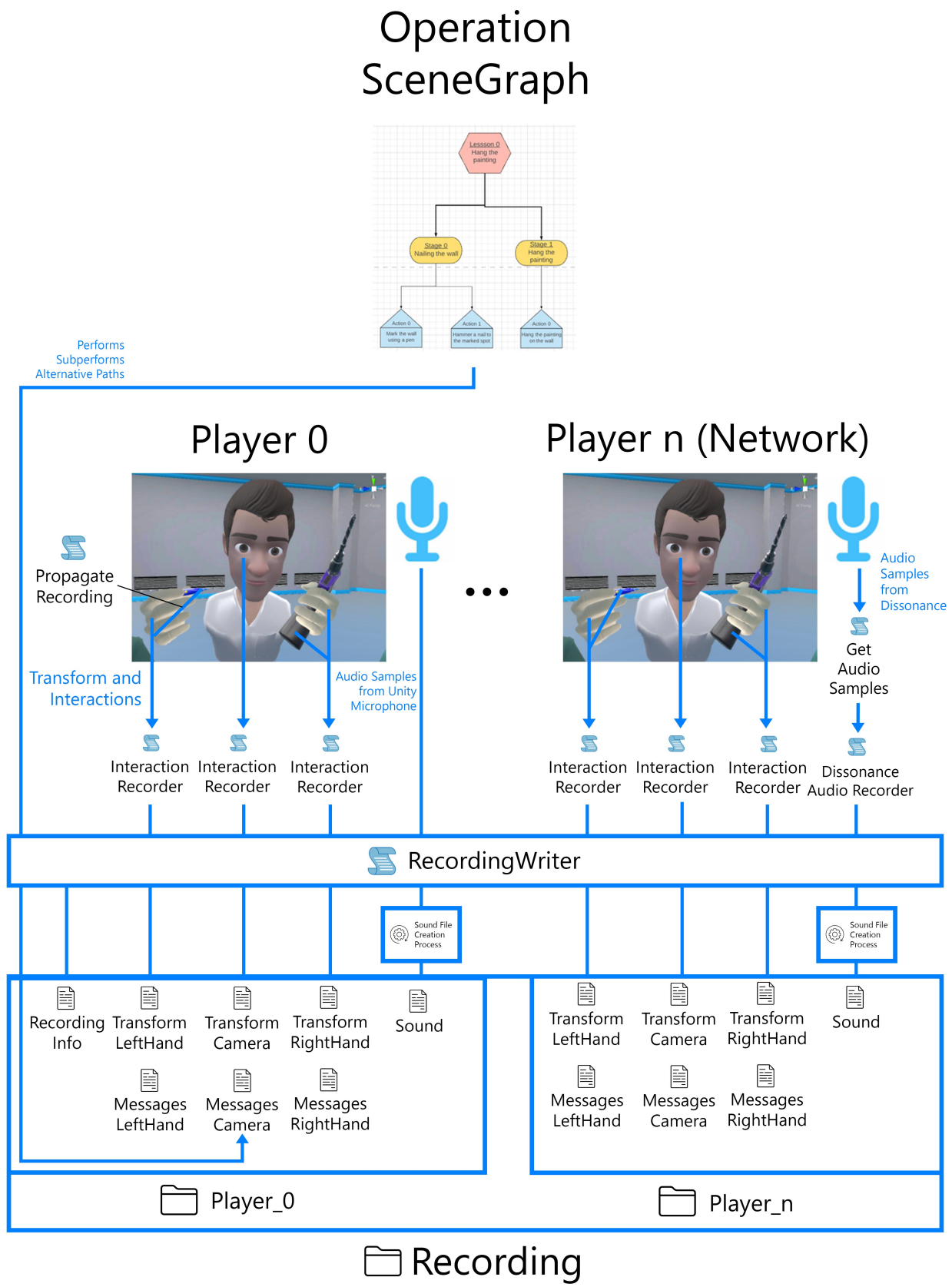
Communication between Scripts and Components

The Recording Writer component is referenced as a singleton instance. It is mostly referenced by other scripts, which use it as a means of accessing the recording's files and updating them each frame.

All Interaction Recorder scripts reference the Recording Writer instance and invoke the appropriate functions for writing transforms and interactions for the GameObject they are attached to. The Recording Writer then accesses the correct files and writes all information to them.

The Propagate Recording Scripts create an entry in the Tracking Targets List of the Recording Writer, who in turn keeps track of the position of the GameObject and writes it to the correct file in the correct level of propagation.

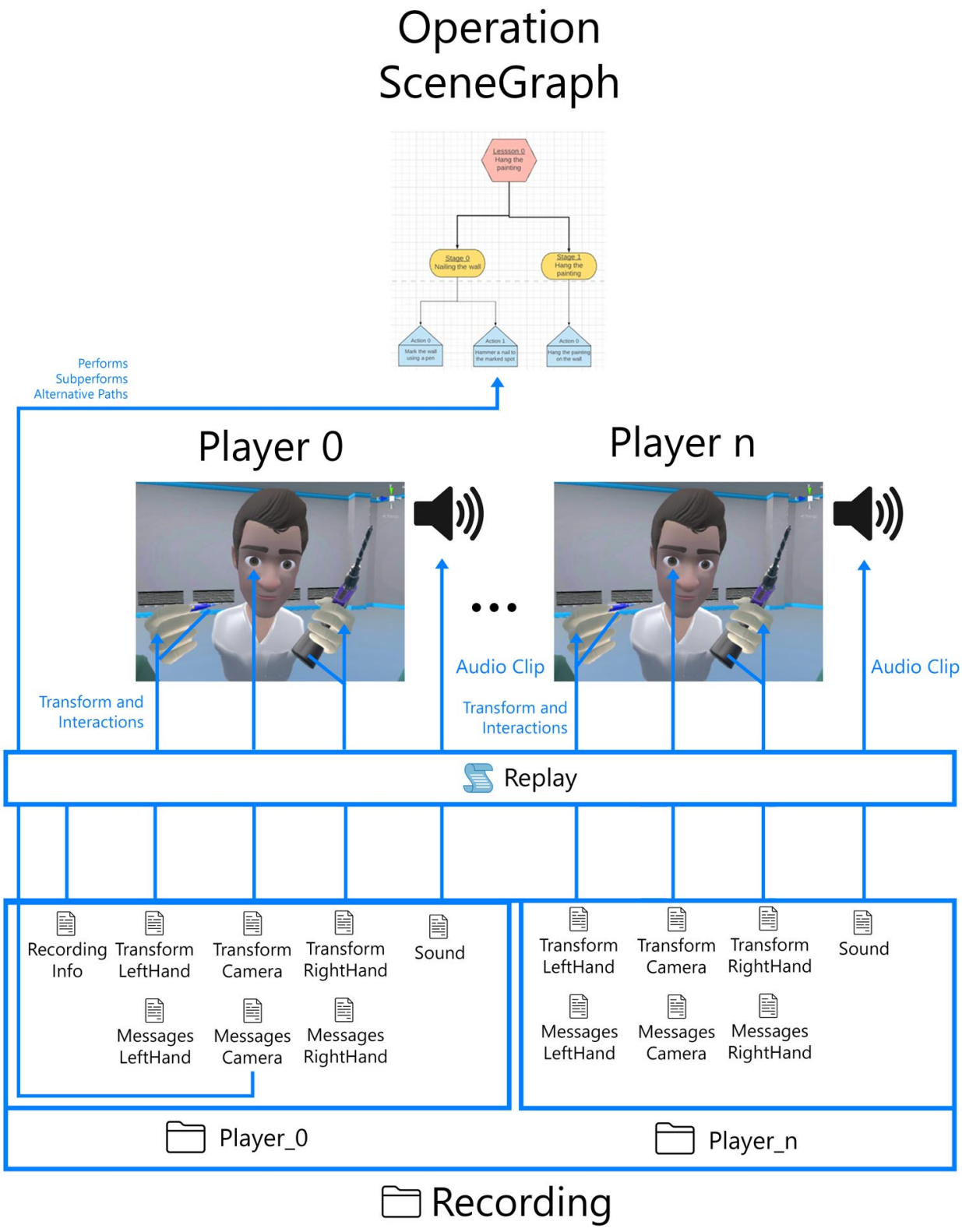
For Sound, the Recording Writer Script collects sound samples from Unity's Microphone interface for the local player, and uses the SaveWav and MergeWav Scripts to save and merge the files at the end of the recording. For Network players, The Get Audio Samples and Dissonance Audio Recorder scripts provide the RecordingWriter with samples, which then follow the same procedure as a local player's samples.


Operation
SceneGraph
Performs
Subperforms
Alternative Paths
Player 0
Player n (Network)
Propagate
Recording
Transform and
Interactions
Audio Samples
from Unity
Microphone
Audio
Samples
from
Dissonance
Get
Audio
Samples
Interaction
Recorder
Interaction
Recorder
Interaction
Recorder
Interaction
Recorder
Interaction
Recorder
Interaction
Recorder
Dissonance
Audio Recorder
RecordingWriter
Sound File
Creation
Process
Sound File
Creation
Process
Recording
Info
Transform
LeftHand
Transform
Camera
Transform
RightHand
Sound
Messages
LeftHand
Messages
Camera
Messages
RightHand
Transform
LeftHand
Transform
Camera
Transform
RightHand
Sound
Messages
LeftHand
Messages
Camera
Messages
RightHand
Player_0
Player_n
Recording

5.12.3 Replaying Recordings

How Replaying accesses Recording Files

All Replay Players are controlled by the Component **Replay** in the *Introduction*. The Component accesses all files of the Recording and manipulates the Replay Avatars, Interacted Objects and the Scene Graph of the Operation in order to fully re-create the recorded session.



The Replay component has references to all replay avatars and fully controls them during the Replaying of an Operation. It also accesses all interacted GameObjects via name in the scene hierarchy, and creates references to them to

manipulate their transform and state if they are interactables or tools.

The Sound is accessed at the beginning of the replay and formed into an audio clip that is then played back using an Audio Source. The Audio is spatial and originates from the head of the player's avatars. The sound is also manipulated by the Replay script to match the current frame rate of the replay, that might not match the frame rate of the recording.

If a user interacts with a tool or a pump interactable, a special replaying script is placed on it, that assists in replaying interactions correctly. These scripts are placed automatically when the Replay Component detects interaction with a pump interactable or a tool. These scripts then reference the Replay script and change their state according to the recorded input that is being replayed (button presses).

Replaying interactions that require user input

Replaying works out of the box for Interactables and all Tools, however in order to add the specialized functionality for Interactables that require user input (E.g. Trigger Pulls), a Replaying Component must be created and attached on all applicable objects.

> **Warning:** Tools' states are manipulated on or off, however animations/changes related to the toggling of their state are not invoked.

```
1  if (Replay.Instance.isActiveAndEnabled &&
2  GetComponent<OvidVRInteractableItem>().IsAttached)
3  {
4      int playerID = int.Parse(interactableItem.AttachedHand.name.Substring(19));
5
6      bool isButtonPressed = interactableItem.AttachedHand.IsLeft ?
7      Replay.Instance.ReplayPlayers[playerID].leftButtonsPressed[buttonName] :
8      Replay.Instance.ReplayPlayers[playerID].rightButtonsPressed[buttonName];
9
10     // Your functionality
11 }
```

Code block explanation:

Lines 1-2: If Replay has an active instance, and this Interactable Item is attached, then we can start listening for button inputs.
Line 4: Acquire Replay Player ID for accessing the correct Messages File.
Lines 6-8: Retrieve Button Press status for the given Button name and hand that the Interactable is attached to. This will either return true or false if the button was pressed during this frame.

Possible Values for buttonName are defined in `OvidVRControllerClass.OvidVRControllerButtons`

Using this snippet you can replicate all your interactions in Replay mode (Or even add to them).

5.12.4 Synchronizing Recordings with the Cloud

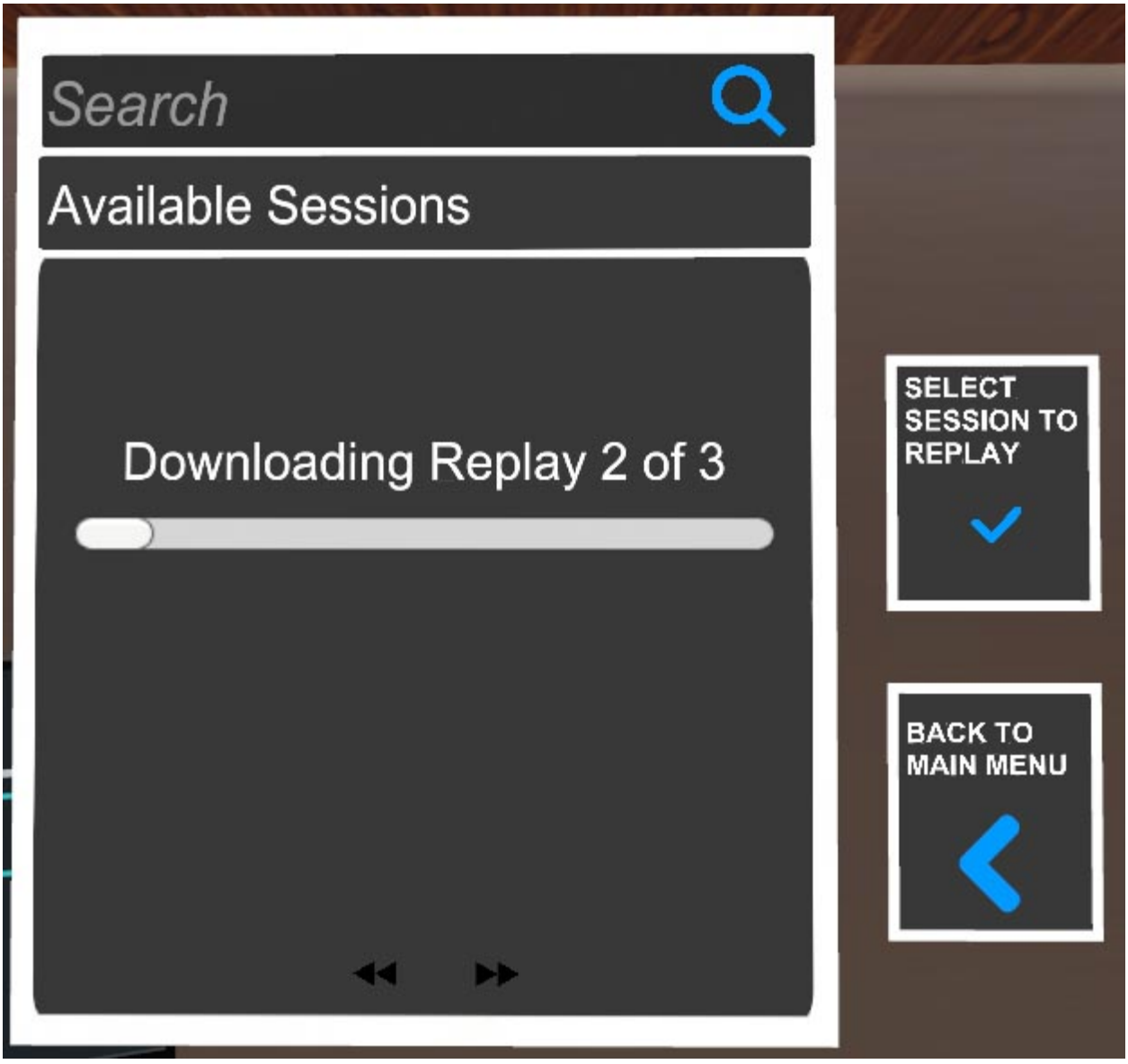
Uploading Recordings

Using the user Token and username, the application will contact the ORamaVR Cloud and upload the recording. This might take a while since the Sound file of the recording is relatively large. All files are stored in Base64 encoding in the server and are encoded during the upload process.

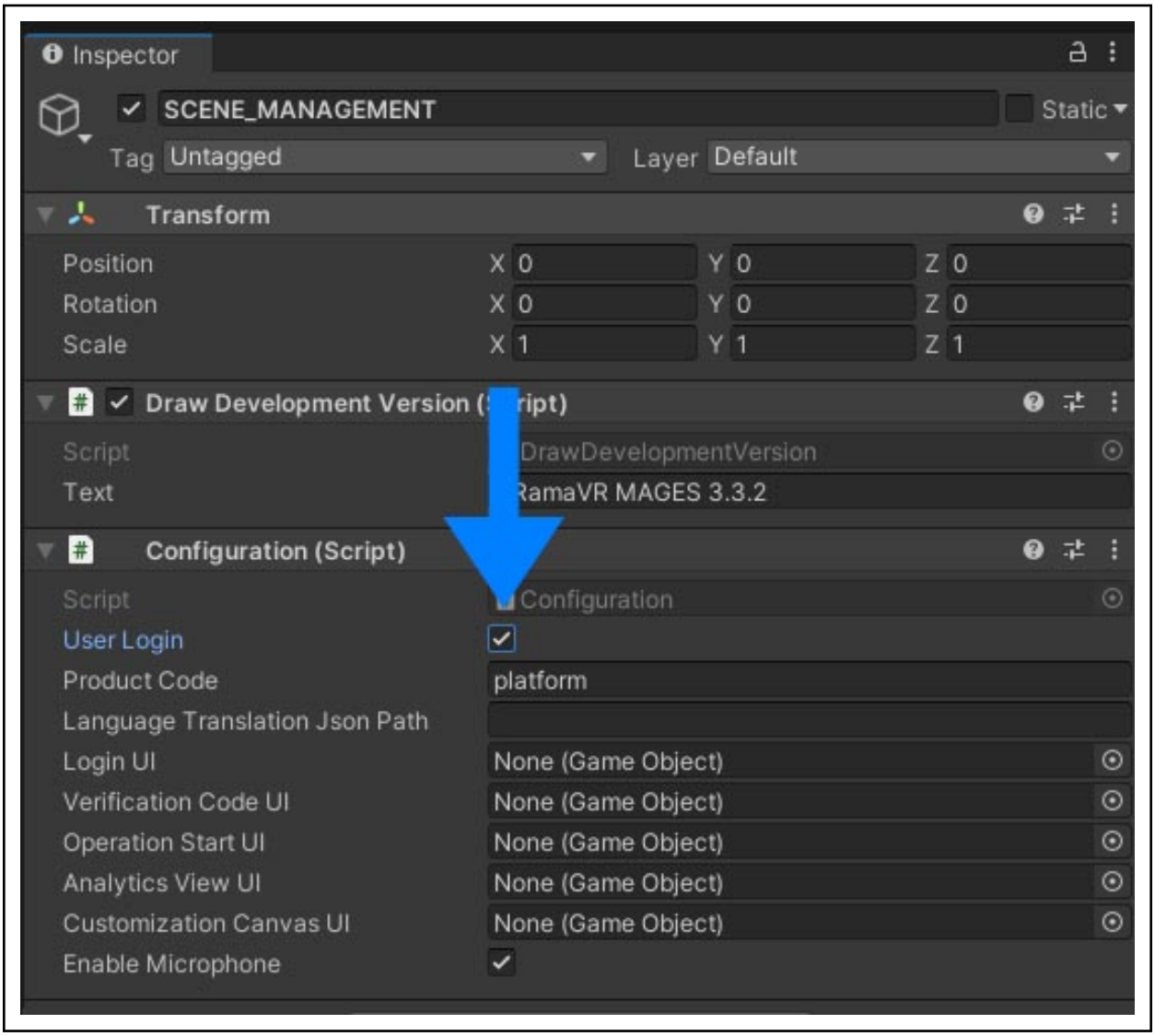
Note: Only operations that reach Operation End will be uploaded. All other recordings will be available locally for the device they were recorded on.

Downloading Recordings

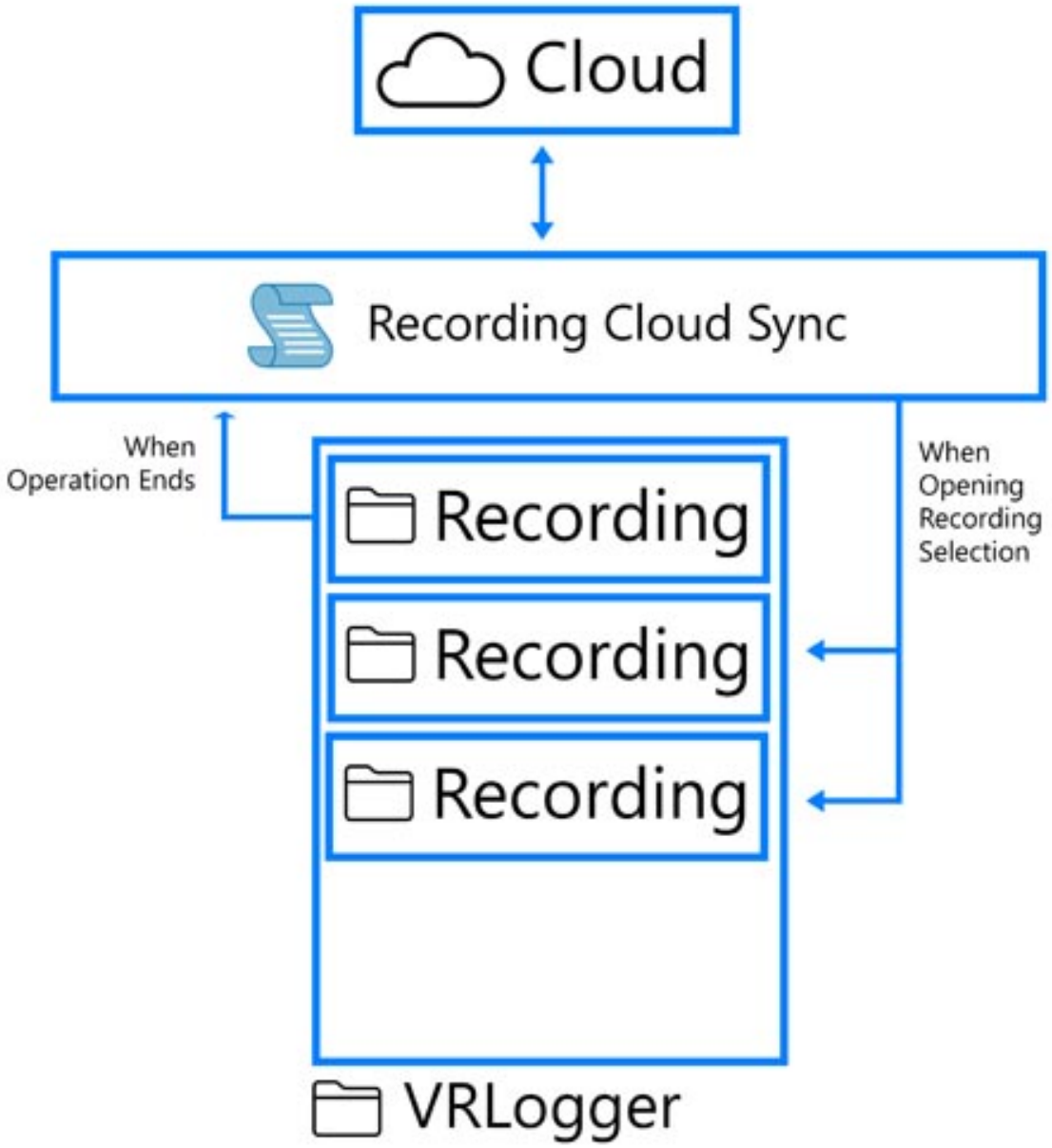
Using the user Token and username, the application will contact the ORamaVR Cloud and fetch a list of available recordings. Then, only the recordings that are not present on the local device will be downloaded from the server. They are all decoded from Base64 encoding and stored in the local file system.



Warning: Uploading and Downloading recordings require the user to be logged in to their ORamaVR account. If you are getting HTTP 500 Errors when attempting upload or download, make sure User Login is enabled in your application.


Inspector
SCENE_MANAGEMENT
Static
Tag Untagged
Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Draw Development Version (Script)
Script DrawDevelopmentVersion
Text RamaVR MAGES 3.3.2
Configuration (Script)
Script Configuration
User Login
Product Code platform
Language Translation Json Path
Login UI None (Game Object)
Verification Code UI None (Game Object)
Operation Start UI None (Game Object)
Analytics View UI None (Game Object)
Customization Canvas UI None (Game Object)
Enable Microphone

Cloud Connection Flowchart



5.13 MAGES Setup Script

In this section we will cover how you can configure –via code– certain settings of your application (mostly runtime).

Let's start with some code.

```
void Start()
{
    // Note: The order is important.
    // Here you can specify certain configuration properties that concern the following: Paths, ApplicationSettings.

    MAGESSetup.Get.ProductCode = MAGESSetup.Get.productCode; // This should only change on build! On Unity.Editor it must remain as it is
    MAGESSetup.Quality = QualityConfig.High;
    MAGESSetup.Region = Region.Auto;
    MAGESSetup.Difficulty = UserAccountManager.Difficulty.Easy;

    OperationXML = MAGESSetup.Get.OperationXML == null ? null : MAGESSetup.Get.OperationXML.name + ".xml";

    MAGESSetup.SetXmlName(OperationXML);

    MAGESSetup.Get.ConfigurePreInitialization(); // Important

    initializeSceneGraph();
}
```

The above snippet is the `Start()` MonoBehavior function override of **sceneGraph.cs**, a script that can be found inside the SampleScene under the **SCENE_MANAGEMENT** gameObject.

> **Warning:** Without sceneGraph your application won't load!

As you can observe from the snippet above, **MAGESSetup** is another script that is responsible for setting certain properties for your application.

In general, MAGESSetup offers two crucial methods:

1. `MAGESSetup.ConfigurePreInitialization();`
2. `MAGESSetup.ConfigurePostInitialization();`

From this we can deduce three main points:

1. Everything related to application settings, paths and storyboard is set prior to `MAGESSetup.ConfigurePreInitialization();`
2. `initializeSceneGraph()` is invoked straight after to load the corresponding Storyboard
3. Everything related to users and analytics (e.g., your own user management) is set after sceneGraph has initialized via `MAGESSetup.ConfigurePostInitialization();`

> **Warning:** **Do not** change the sequence/structure, or you will almost certainly experience unexpected behaviors!

By default MAGESSetup.cs has certain values set which you can inspect by opening the script.

5.13.1 Pre-Initiliazation

- The first step is to set the Product or Application name via the `string` property `MAGESSetup.Get.ProductCode;`.

  By default, ProductCode is set to "platform" for the Unity Editor.

  > **Warning:** If you are running inside the Unity Editor do not change the ProductCode, otherwise, your application will crash.

  The ProductCode gets appended to the path under Documents `{User}/Documents/ORamaVR/Story/{ProductCode}` where your application will be looking to load the Storyboard, in built-mode (i.e., when you have produced an executable).

  However, in the Unity Editor, Storyboard is by default looking under `/Assets/Resources/Storyboard/platform` to load your storyboard.

- Then, you can proceed to set the Quality settings through the `enum` property `MAGESSetup.Get.QualityConfig;`.

  These are the graphic quality settings that are set into Unity.

  The `enum` takes one of the following three values `[Low, Medium, High]`. Default is set to `High`. However, in certain builds (e.g., Android) you might need to lower the quality.

- Afterwards, you can specify a region for your COOP (–Multiplayer) servers by setting the `enum` property `MAGESSetup.Get.Region;`.

  Default is set to `Auto`.

  Other available options are: `[UnitedStates, Europe, Signapore]`.

- Further, you can configure the difficulty of your application towards the end-user behaviors through the `enum` property `MAGESSetup.Get.Difficulty;`.

  Default value is set to `Easy`.

  Other available options are: `[Medium, Hard]`.

  Note: Difficulty utilizes the `enum` from `UserAccountManager.Difficulty`.

- Finally, you **need** to set the names of the XML files your Storyboard will load from.

  To do so, invoke the function `MAGESSetup.Get.SetXMLNames(string arg1, string arg2, string arg3, string arg4);`.

  You can configure these values from the Unity Editor/Inspector under `SCENE_MANAGEMENT/Scene Graph` gameObject.

  Empty/Null values for *AlternativeLessons*, *AlternativeStages* and *AlternativeActions* are allowed.

5.13.2 initializeSceneGraph()

After you have set all pre-initialization options, `initializeSceneGraph();` will run and load your Storyboard and everything related to your specified settings.

5.13.3 Post-Initiliazation

- You can proceed to set custom properties for your analytics.

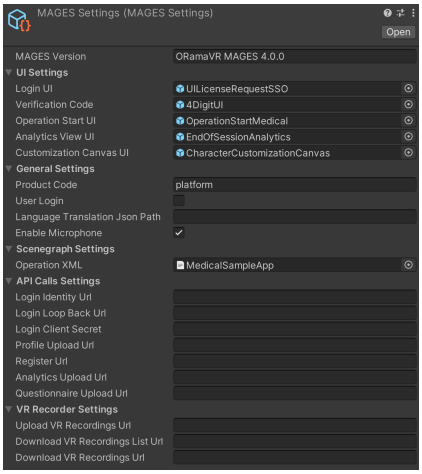
  For instance, the OnlineURL where your user analytics will be uploaded, alongside with custom `FormFields` and potentially custom `HeaderFields` for token authentication (if your platform supports that).

  In addition, you can also set the local file paths your analytics will be exported to.

A full list of available Analytics configurations is provided in the code snippet below.

```
#region Analytics
public static string EditorPathToAnalytics { get; set; }

public static string OverrideLocalWindowsPath { get; set; }

public static string OverrideLocalAndroidPath { get; set; }

public static string OnlineURL { get; set; }

public static List<AnalyticsExporter.FormField> FormFields { get; set; }

public static List<AnalyticsExporter.HeaderKey> HeaderKeys { get; set; }

#if UNITY_ANDROID
    public static string PathToAnalytics { get; set; } = "/data/data/" + PackageName␣
↪+ "/Analytics/";
#elif UNITY_STANDALONE_WIN
    public static string PathToAnalytics { get; set; } = "";
#endif
#endregion
```

Note: Alternatively, you can set important information for your project through the MAGESSettings.asset.



5.14 Cut - Tear - Drill

5.14.1 Realtime Cut

> **Warning:** Note that this feature is in beta stage, therefore it is not stable and it can create artifacts on the newly created meshes.

MAGES gives you the ability to perform realitme cuts in meshes, by providing a **Cuttable Mesh** and a **Cutter** script. Additionally a **CutAction** is available for implementing this feature in an Action Form (see *Cut Action* for more details). A combination of these can achieve results like the following:

Cuttable Mesh

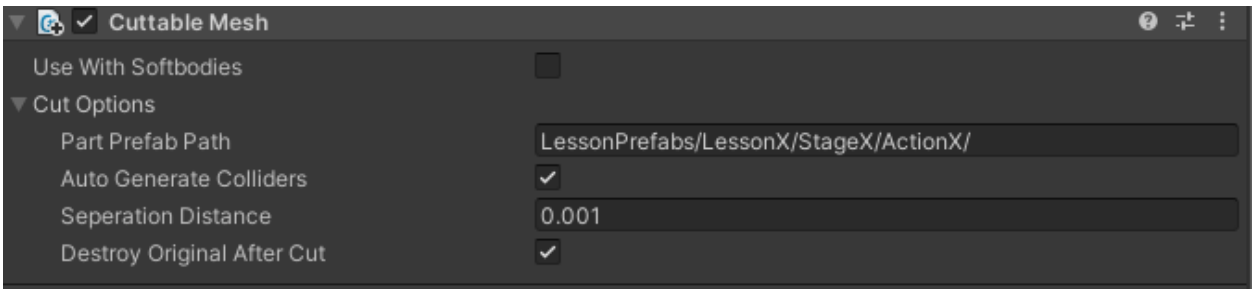
The **Cuttable Mesh** script found in the **Add Component > MAGES > Mesh Deformations > Cuttable Mesh** menu enables realtime seperation of a mesh along a plane. It can be used in combination with the **Cutter** script or as a standalone by simply calling the

```
public bool Cut(Plane cutPlane, out Mesh positiveSide, out Mesh negativeSide, out␣
↪GameObject final, bool handleMeshReconstruction = true)
```

method. Afterwards you can use

```
public bool UndoCut()
```

to undo the last cut. By attaching the script in a GameObject that containts a Renderer component you get the following options:

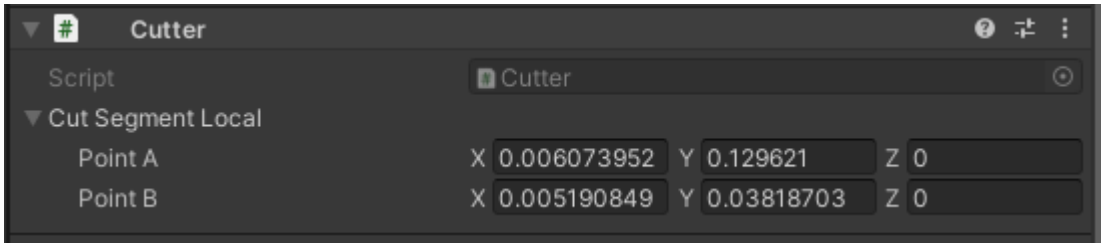


Property	Description
`bool useWithSoftbodies`	Only available when attached in a mesh beeing a Softbody, it enables/disables the cutting of the underlying softbody structure
`string partPrefabPath`	After a cut if handleMeshReconstruction is enabled the two meshes will be attached to the Renderer of the 2 new instances of the prefab found in this prefab path. Note that the prefab **must be under the Resources folder**.
`bool autoGenerateColliders`	After the cut should mesh colliders be generated and attached to the new cut parts?
`float seperationDistance`	By how much should the new parts after a cut be moved appart? Used mainly to make the cut visible by leaving a gap in the cut section.
`bool destroyOriginalAfterCut`	Should the original uncut gameobject be destroyed after a cut has been performed?

Cutter

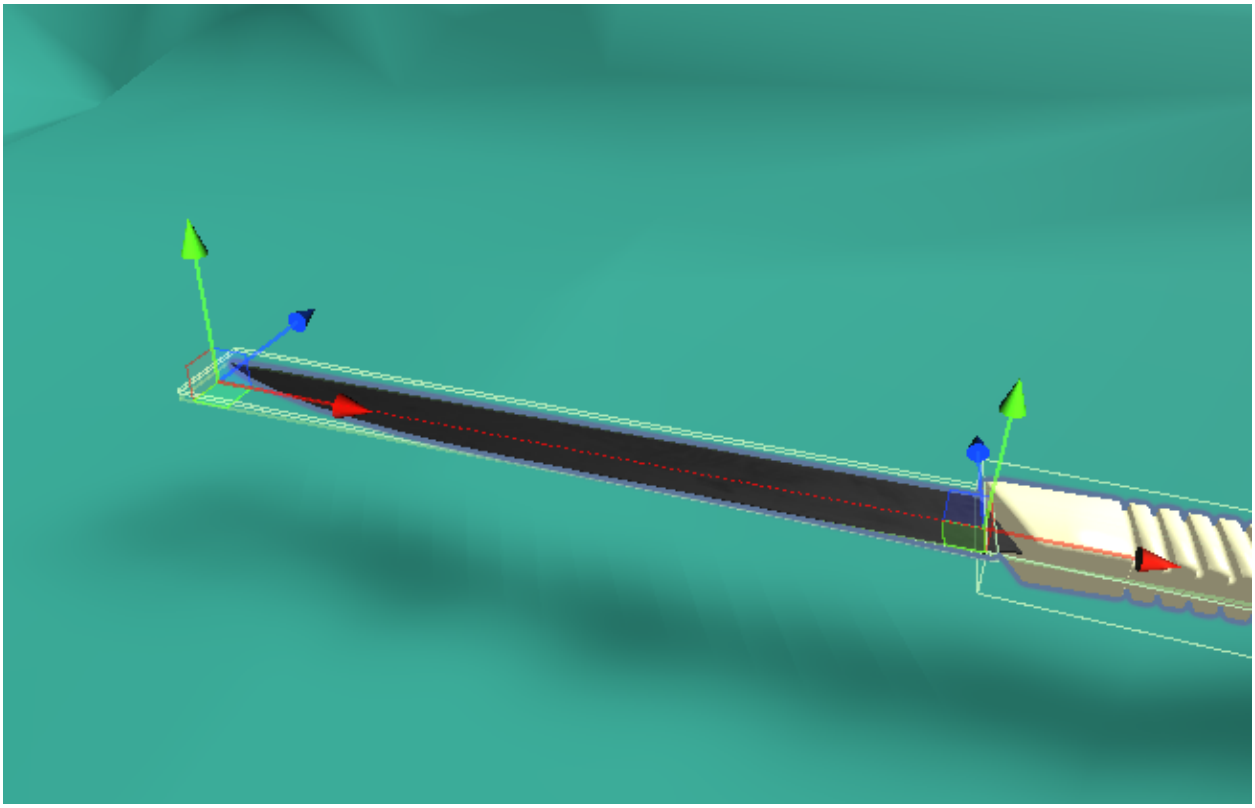
The **Cutter** script, found in the **Add Component > MAGES > Mesh Deformations > Cutter** menu, is a simple example script showing how to use the CuttableMesh. It can be attached in a gameobject that you want to act as a knife splitting the mesh in two parts. In order for the Cutter, to work we must have a trigger collider in the knife at the position of the blade and a collider (trigger or not) in the gameobject having the CuttableMesh script attached to it. These colliders will be used in order to detect the moment the knife enters the Cuttable Mesh and the moment it exits. We keep track of 3 points, two points on enter and one point on exit, in order to define the cutPlane.

By attaching the **Cutter** script to a gameobject you get the following properties:



Property	Description
`CutSegment cutSegmentLocal`	Defines the two points that will be used to create the cutPlane. Adjust them to match your blade's edges.

Note: Instead of manually setting the cutSegment's points you can use the two handles that appear in the scene. Make sure you have the editor Gizmos option enabled. If the handles do not show, they may be on the same point as the default move handle for this gameobject. If that is the case, simply select the hand tool in the editor in order to hand the default move tool and make the handles visible.



5.14.2 Realtime Tear

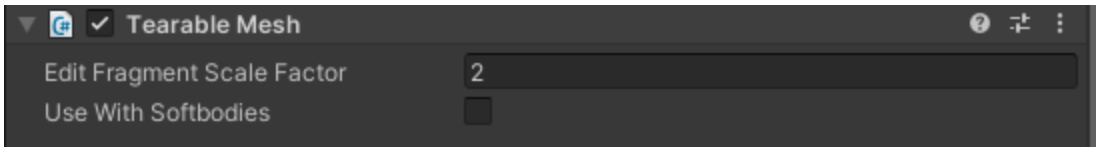
Warning: Note that this feature is in experimental stage, therefore it is not stable and it can create artifacts on the newly created meshes.

MAGES gives you the ability to perform realitme surgical like tear in meshes, by providing a **Tearable Mesh** and a **Tearer** script. A combination of these cn achieve results like the following:

Tearable Mesh

The **Tearable Mesh** script found in the **Add Component > MAGES > Mesh Deformations > Tearable Mesh** menu enables realtime surgical like tearing. It can be used with the Tearer script or as a standalone by simply calling the `public void Tear(Vector3 pointA, Vector3 pointB, Vector3 direction, float width, float height)` or the `public void Tear(Plane leftFace, Plane rightFace, Plane bottomFace, Plane topFace, Plane frontFace, Plane backFace)` method. These methods will remove a box-like section of mesh. The first method needs two points, the direction of the height of the box normalized, the width and the height of it. The second method needs 6 planes defining the 6 faces of a box with their normals facing inside the box.

By attaching the script in a GameObject that containts a Renderer component you get the following options:

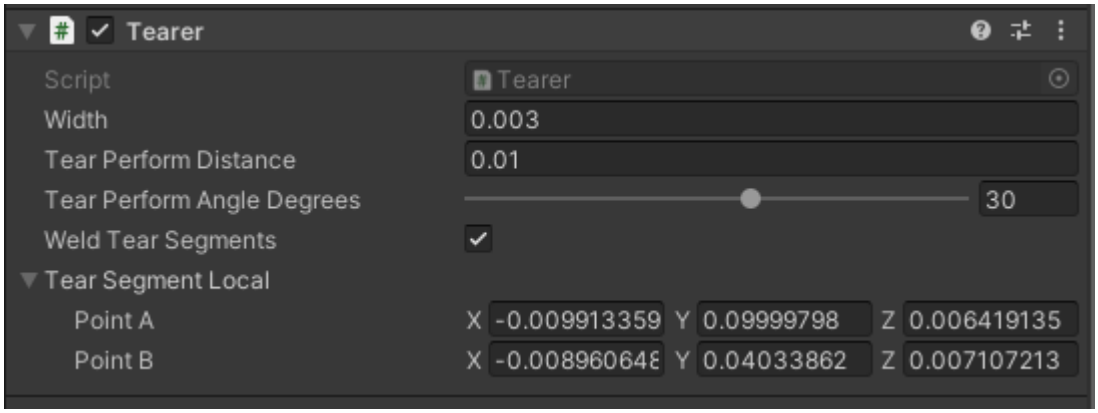


Property	Description
`bool editFragmentScaleFactor`	Used only when the `public void Tear(Vector3 pointA, Vector3 pointB, Vector3 direction, float width, float height)` method is called. To improve performance when this prperty is greater than 0 a selection sphere is created and instead of all the triangles, only the triangles inside it are checked for tear. Adjust it appropriately to the smaller value possible, while creating tears without artifacts.
`useWithSoftbodies`	Only available when the tearable mesh is attached to a softbody object. When enabled, the particle connections intersecting the tear box will be broken and the particles will not affect vertices in the oposite side of the box; The tear will be able to spread.

Tearer

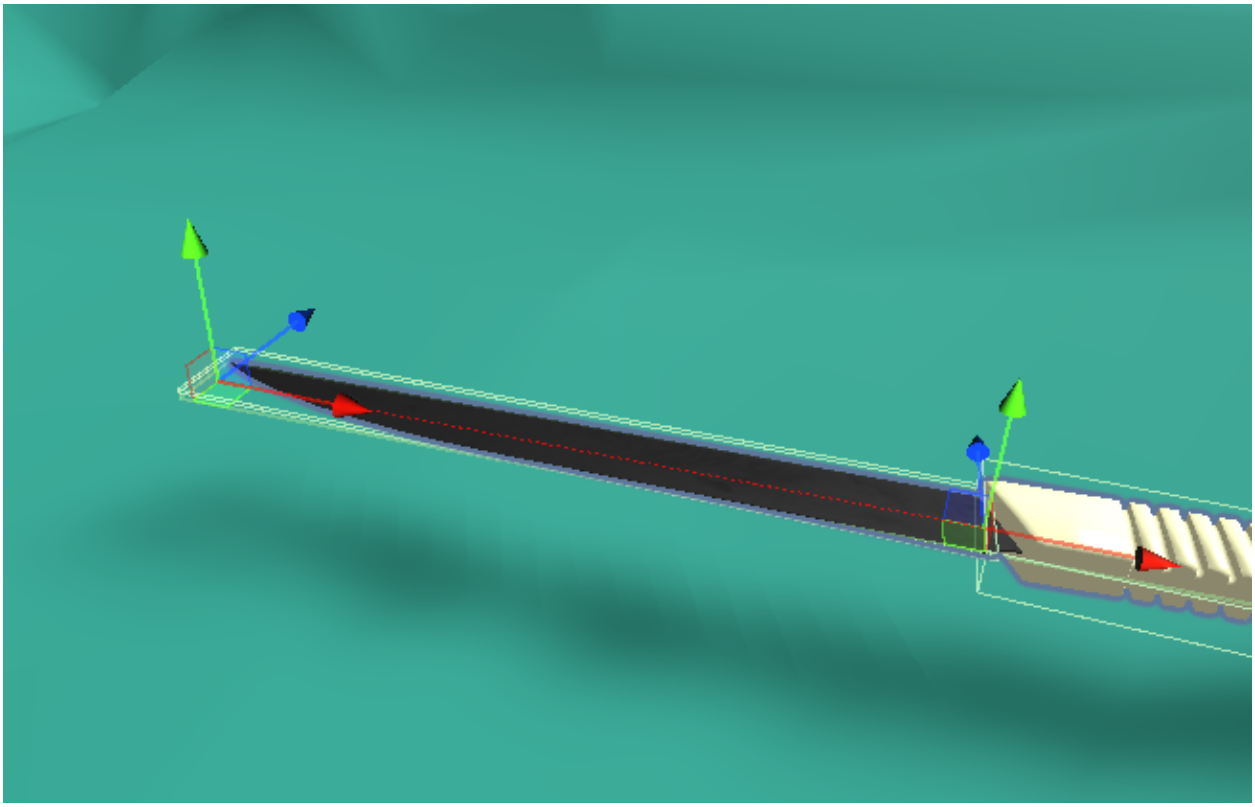
The **Tearer** script, found in the **Add Component > MAGES > Mesh Deformations > Tearer** menu, is a simple example script showing how to use the TearableMesh. It can be attached in a gameobject that you want to act as a scalpel tearring the mesh. In order for the Tearer to work, we must have a trigger collider in the scalpel at the position of the blade and a collider (trigger or not) in the gameobject having the TearableMesh script attached to it. These colliders will be used in order to detect the moment the scalpel enters the Tearable Mesh and the moment it exits.

By attaching the **Tearer** script to a gameobject you get the following properties:



Property	Description
`public float width`	The width of the tear, in worldspace.
`public float tearPerformDistance`	The distance of the sample points for creating the boxes, on the curve the movement of the tearer. Smaller values will create smoother curves, but decrease performance.
`public float tearPerformAngleDegrees`	If the angle in degrees created from the position of the tearer in relation to the old tear box is greater than this, a new tear box will be created. Smaller values will create smoother curves, but decrease performance.
`public float weldTearSegments`	Weld multiple tear boxes together in order to create a continuous curve?
`public TearSegment tearSegment`	Defines the two points that will be used for creating the tear boxes. Adjust them to match your blade's edges.

Note: Instead of manually setting the tearSegment's points you can use the two handles that appear in the scene. Make sure you have the editor Gizmos option enabled. If the handles do not show, they may be on the same point as the default move handle for this gameobject. If that is the case, simply select the hand tool in the editor in order to hand the default move tool and make the handles visible.



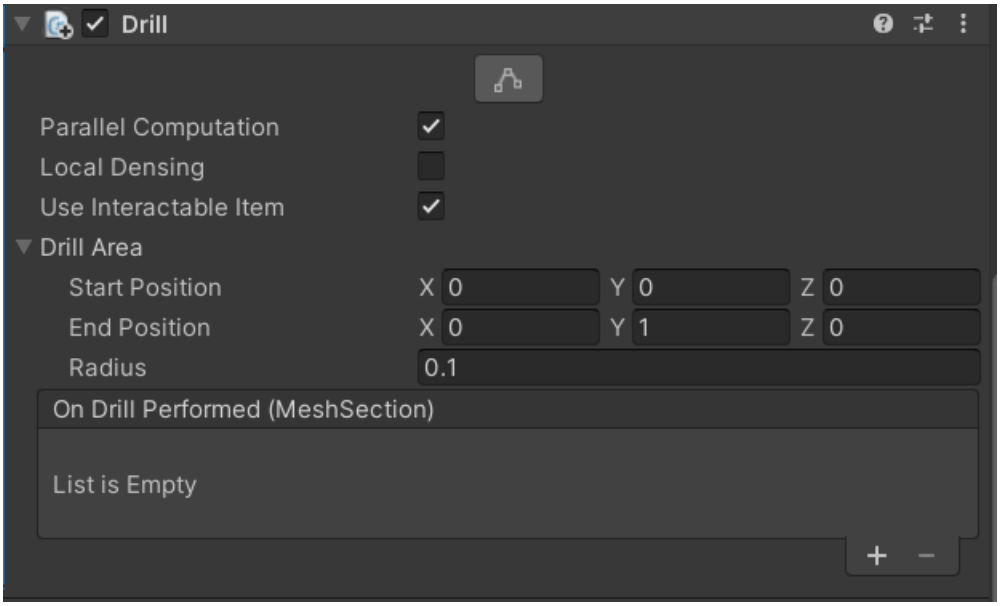
5.14.3 Drill

The drill module enables the developer to set up a Unity GameObject as a drill tool, in order to achieve real time drilling of holes on 3D rigged and static models.

Set up

To use the Drill Module, you need to attach the `Drill` component onto the gameobject that we want to use as a drill. Simply click **Add Component> MAGES > Mesh Deformations > Drill** to add the component.

Note: The scale of the gameObject that this script is attached to, affects the scaling of the drill area.



This gives you the following properties:

1. Parallel Computation:

- Option to run the code in parallel for increased performance.

2. Local Densing:

- Option to split every affected triangle of the drilling into four smaller ones, in order to prevent the smaller diameter holes looking like polygons.

3. Use Interactable Item

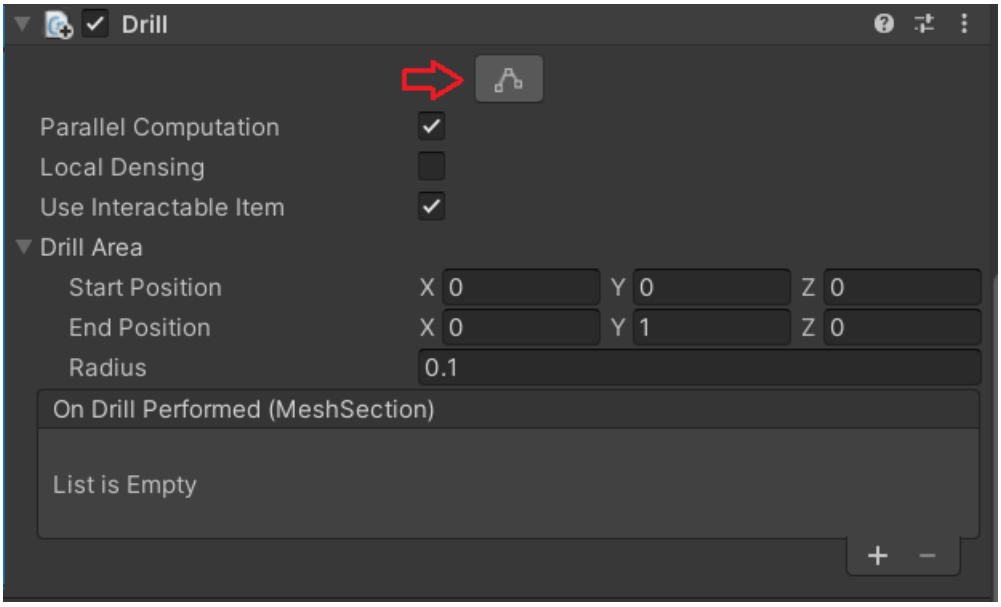
- Option to use this script in combination with an interactable item, such that when you have this object in your hand and then press the trigger button a drill is performed.

4. Drill Area:

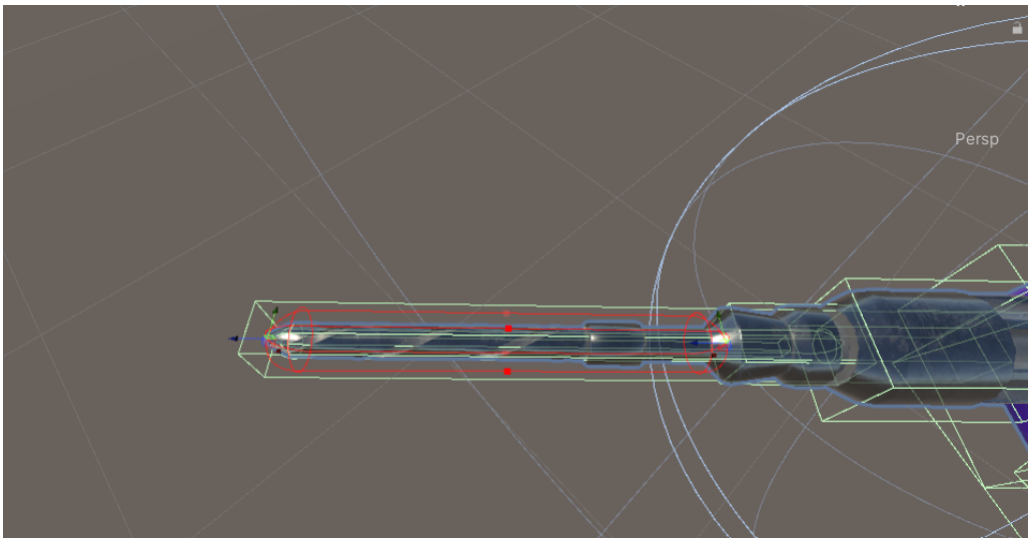
    - Option to set up the drilling axis and the radius of the drill.

Note: Make sure Gizmos are enabled on the Unity editor.

    – Click the edit button shown in the image below:



– Use the handles shown in the scene to adjust the drill area to your requirements as shown below:



That last step that remains is to add a trigger collider component to the gameObject, that is aligned with the drill axis GUI that we created before:

In our example we are going to add a `Box Collider`.

Properties

Property	Description
`bool parallelize`	Minimize running times by executing the code in parallel.
`bool localDensing`	Split every affected triangle of the drilling into four smaller ones, in order to prevent the smaller diameter holes looking like polygons.
`DrillAxis drillArea`	Set up the drilling axis and the radius of the drill
`UnityEvent OnDrillPerformed`	Called whenever after this script successfully modifies a mesh section.

5.15 Languages Support

5.15.1 Languages Support in MAGES™ SDK

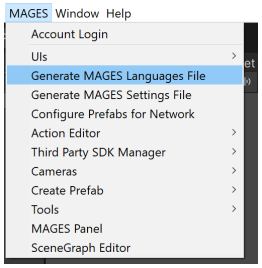
Warning: Note that this feature is still under development.

MAGES™ SDK provides multilingual capabilities. Any application that is created with MAGES™ SDK can support different languages. It is up to the developers to provide the languages of their choice, as well as the possible

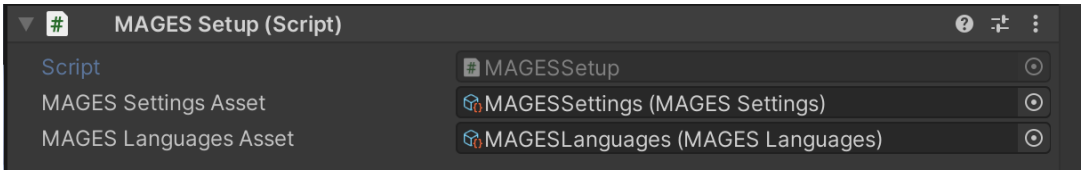
translations for the text(s) they desire in each language.

MAGES™ SDK Languages Setup

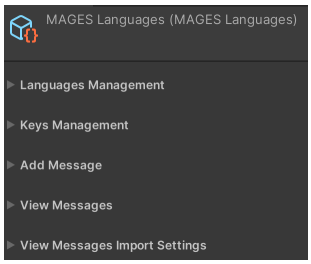
In order to be able to utilize MAGES™ SDK languages support, a MAGES Languages asset file must be created. To create a MAGES Languages asset file, navigate to the **MAGES** menu and click **Generate MAGES Languages File** as shown in the following image.



Warning: Please make sure that the MAGESLanguages asset file is also correctly setup in the MAGESSetup component (usually located in the SCENE_MANAGEMENT GameObject).



Choose the asset file location and save it. Once it is created, you will be presented with the following settings:



Option	Description
Languages Management	Expand this to present options regarding the addition/removal of a language.
Keys Management	Expand this to present options regarding the addition/removal of a key.
Add Message	Expand this to present options regarding the addition/removal of a message.
View Messages	Expand this to view added messages.
View Messages Import settings	Expand this to present options regarding the import of already existing messages from a .json file (older version of MAGES)

Adding/Removing a Language

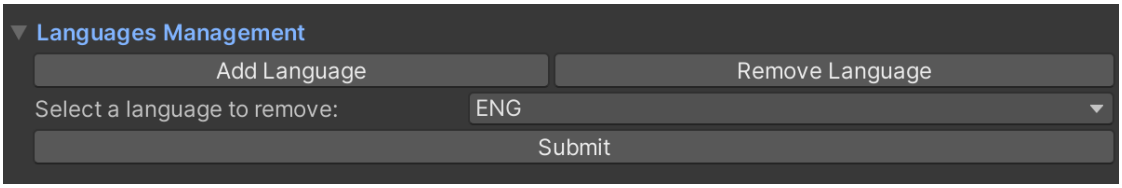
To add a new language to the application, expand the **Languages Management** section.

Type in the text field the name of the new language.

> **Warning:** Note that only the first three letters of your input will be kept, so if you type, for instance, ENGLISH, then ENG will be saved.

Finally, click the **Submit** button.

Following the same procedure, to remove a language expand the **Languages Management** section.

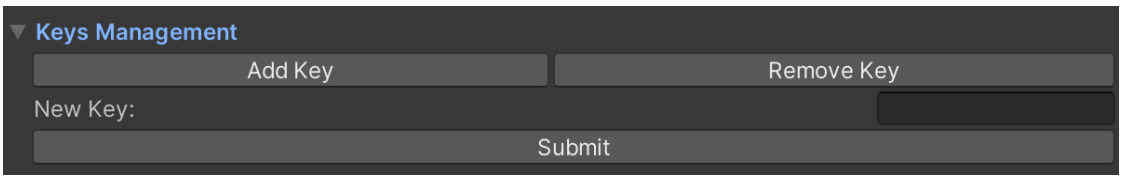


Click the **Remove Language** button.

Select the language you wish to remove from the dropdown list and click the **Submit** button.

Adding/Removing a Key

To add a new key to the application, expand the **Keys Management** section.

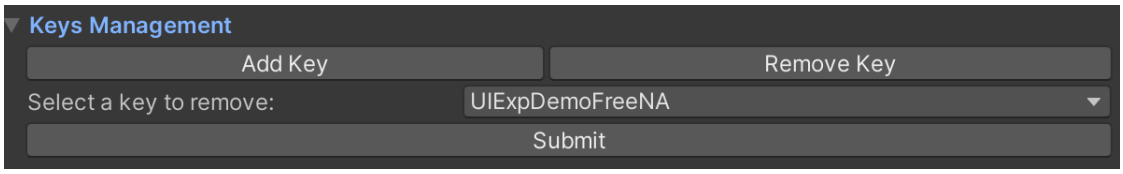


Click the **Add Key** button.

Type in the text field the name of the new key.

Finally, click the **Submit** button.

Following the same procedure, to remove a key expand the **Keys Management** section.

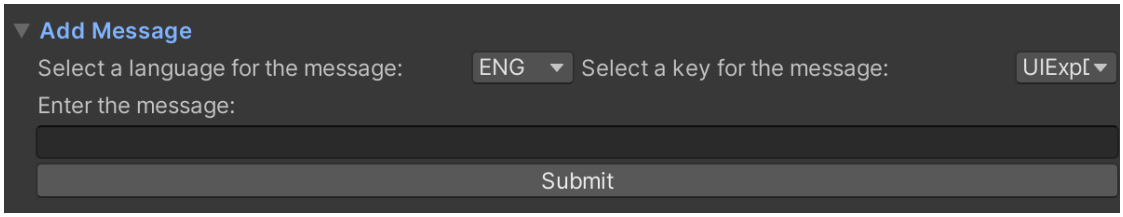


Click the **Remove Key** button.

Select the key you wish to remove from the dropdown list and click the **Submit** button.

Adding a Message

To add a new message, expand the **Add Message** section.



You will be presented with two dropdown lists and a text field.

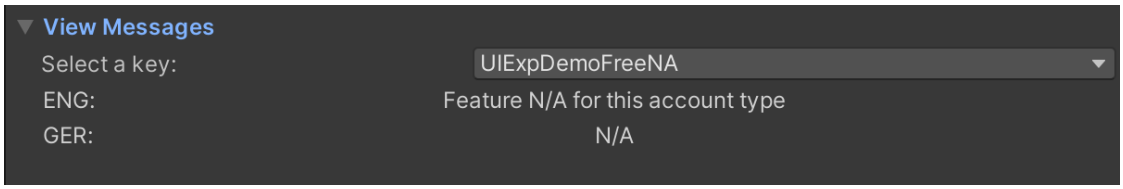
Select the language for this message (first dropdown list) and the key that this message corresponds to (second dropdown list).

Type the desired message in the text field.

Click the **Submit** button.

Viewing a Message

To view all the registered messages for a specific key, expand the **View Messages** section.



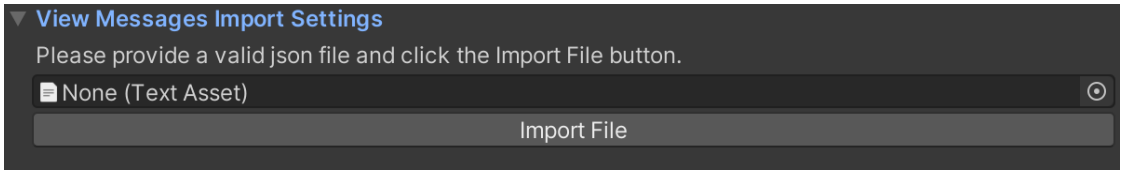
Select a key from the dropdown list, to view its registered messages for each language.

Messages Import

Note: This option is mainly for importing saved messages/lanuages/keys created with an earlier version of MAGES, that was utilizing .json files to keep this data.

To import messages/languages/keys from an earlier version of MAGES, you will need its corresponding .json file (its default name is LanguageTranslationMsg).

Expand the **View Messages Import Settings** section.



Set the .json file reference to the corresponding field.

Click the **Import File** button.

Warning: This action will overwrite all your previous messages/languages/keys.

5.15.2 Translation MAGES™ SDK

Warning: Note that this feature is still under development.

MAGES™ SDK translation feature is able to provide translation of either public fields in MonoBehaviour components or text components. The translation it provides and the languages it supports, depend on your MAGESLanguage asset file setup. The language this script translates the text to, is based on the main application language, chosen by the user on startup.

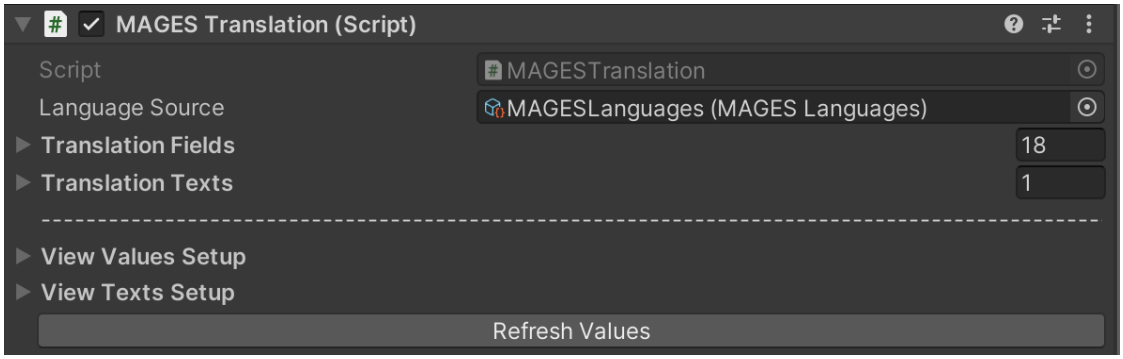
MAGES Translation Script

To translate MonoBehaviour script public text fields, or the text field of a text component, add the MAGESTranslation script to the desired GameObject.

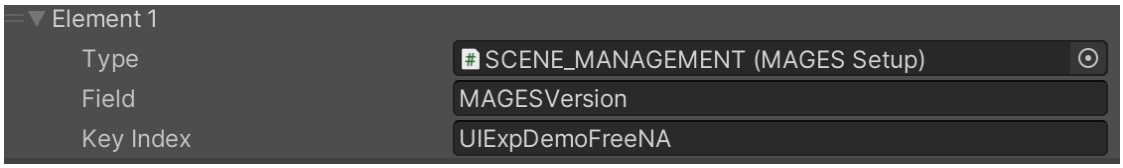
If you have already opened a MAGES scene, that contains the **SCENE_MANAGEMENT** GameObject, being correctly setup, this script will automatically locate the MAGESLanguages asset file for this application.

Note: In case the above fails, you can always drag and drop the MAGESLanguages file of your choice to the corresponding field of the script.

Once MAGESTranslation script is attached, it will automatically detect all the public text values of a MonoBehaviour script and the text field of the text component (if the GameObject has one) and add them all to the **Translation Fields** and **Translation Texts** lists respectively.



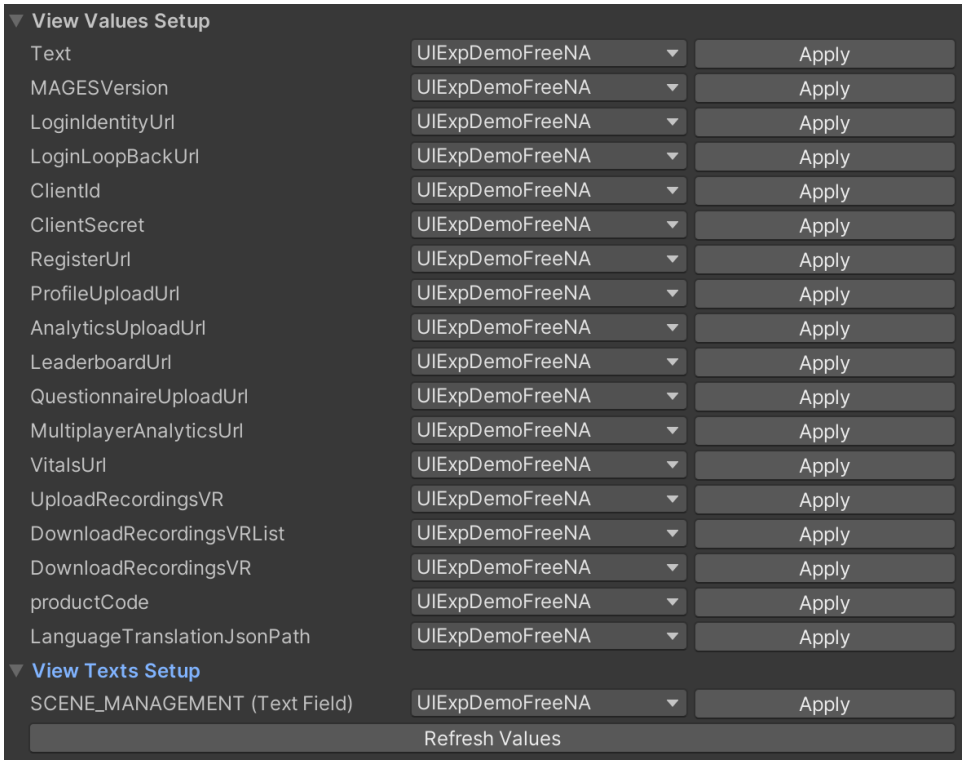
An entry of these list looks like the one below:



Field reports the name of the field to be translated (only present in case of a MonoBehaviour script field), while the **Key Index** refers to the key holding the message that will replace the value of this field.

You can set the **Key Index** value in two ways.

1. By typing the name of the key (key names are case-sensitive).
2. By expanding the **View Values Setup** section, locating the field of your choice, setting the corresponding key and clicking the **Apply** button.



Note: It is recommended to use the second way, to avoid possible errors while typing key names.

The MAGESTranslation script is constantly updated. If you add or remove a new component (MonoBehaviour/Text) please click the **Refresh Values** button, to reload it and update the translation lists.

CHAPTER

SIX

TUTORIALS

6.1 Action Prototypes

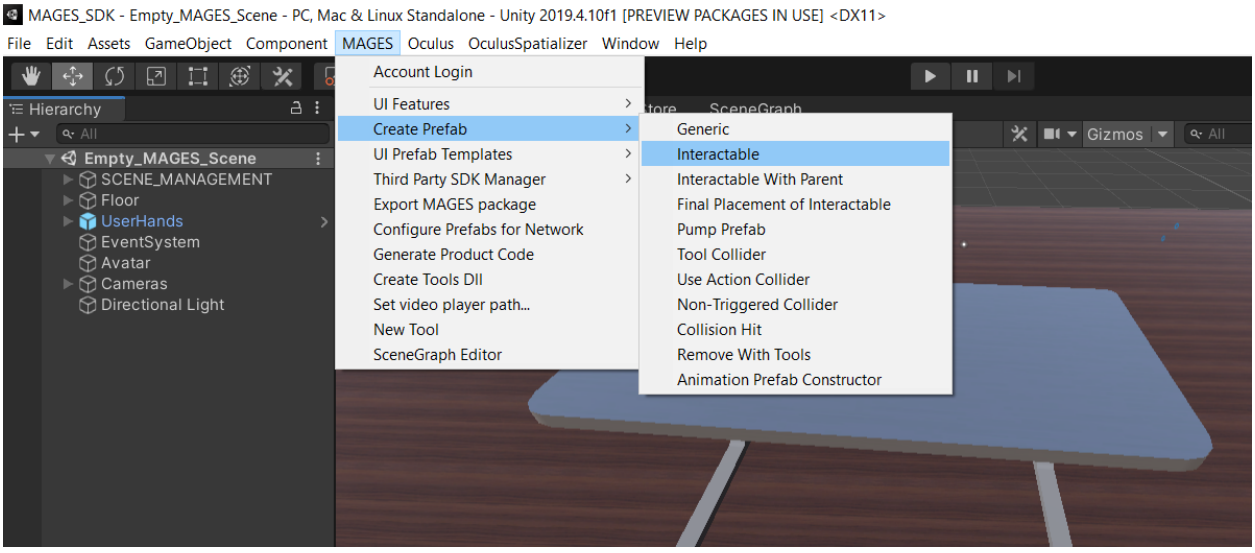
6.1.1 Insert Action

To generate an Insert Action you need the following three prefabs:

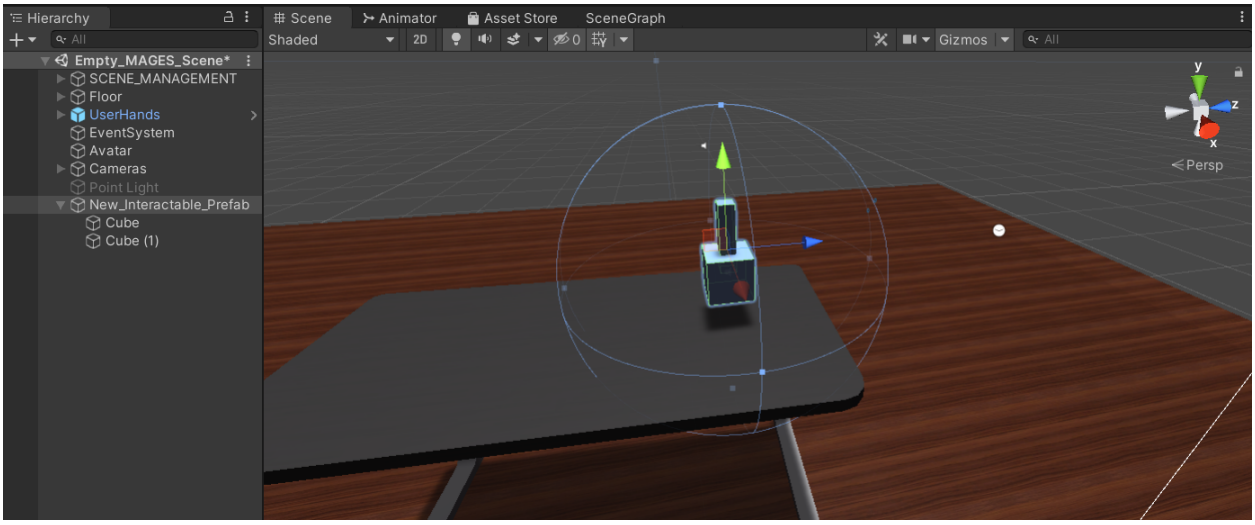
1. The interactable Prefab
2. Its final position
3. A hologram indicating the final position

Interactable Prefab

From the MAGES menu select the option CreatePrefab/Interactable



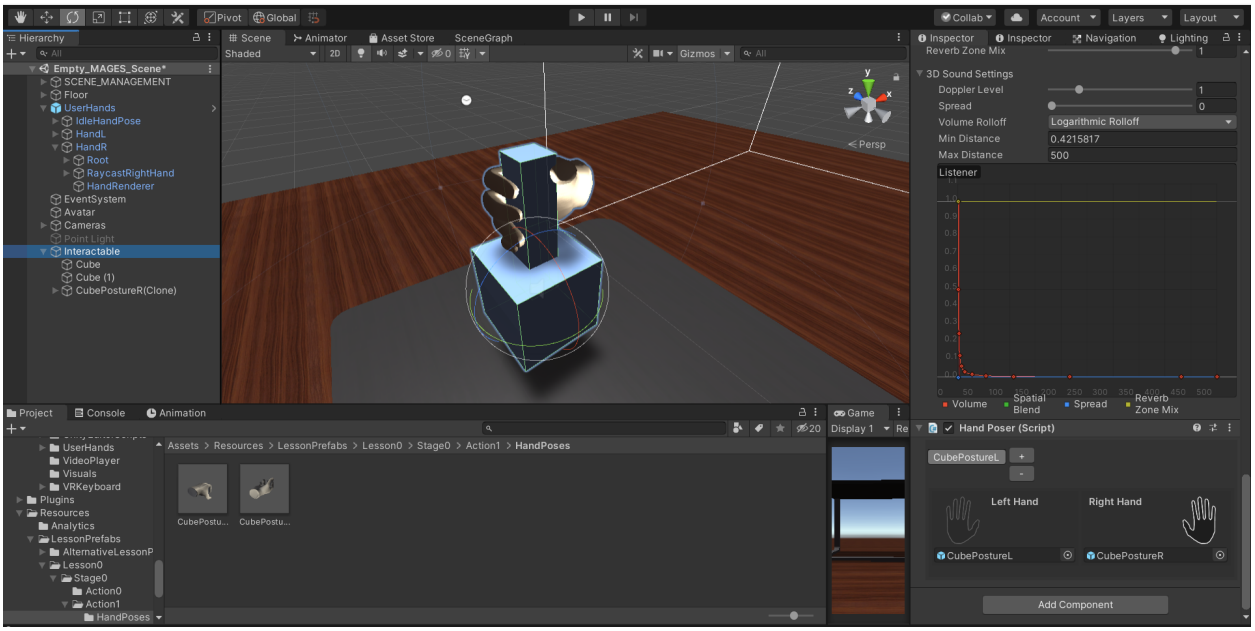
The template gameObject for the interactable prefab will appear. It is recommended to use this object as the root of your interactable prefab. Now we will populate the prefab with our object. In this case we will use two cubes. Below you can see the final result.



We renamed the gameobject to “Interactable” for our convenience. Remember to add physical colliders to the object as you need to grab it, otherwise it will pass through the table.

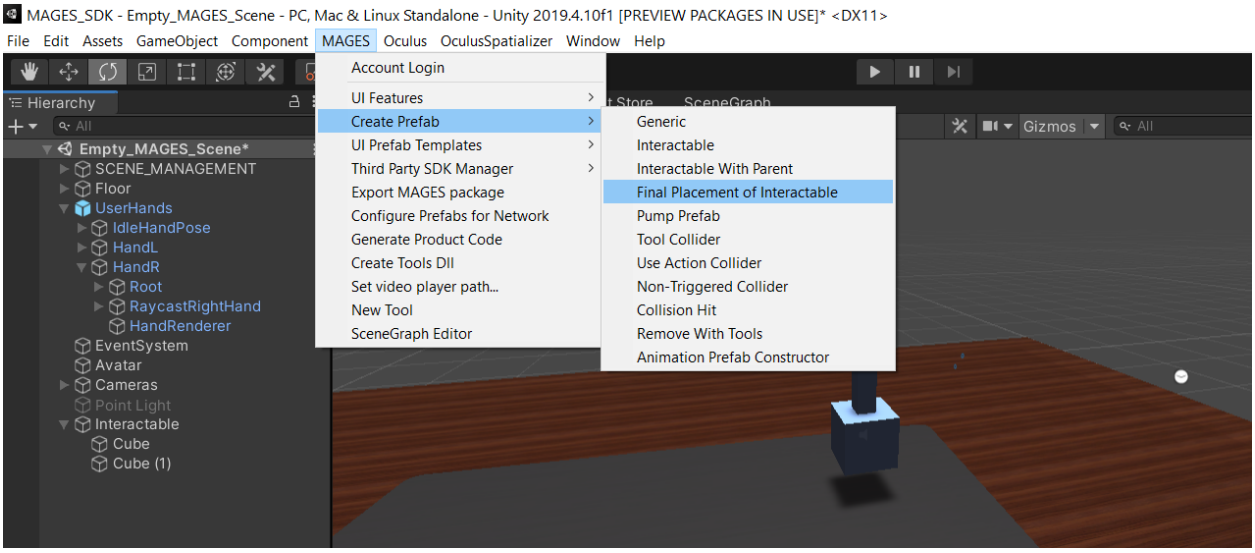
The next step is optional but recommended for a more natural interaction.

We need to configure hand postures when interacting with an interactable object. You can read *here* a detailed tutorial on how to properly setup hand postures. The image below shows the posture of the right hand attached on our object.



Final Prefab

The next step is to generate the final prefab. This indicates the correct position and the orientation of the object. In a similar way, we navigate to the MAGES menu and click the CreatePrefab/Final Placement of Interactable.

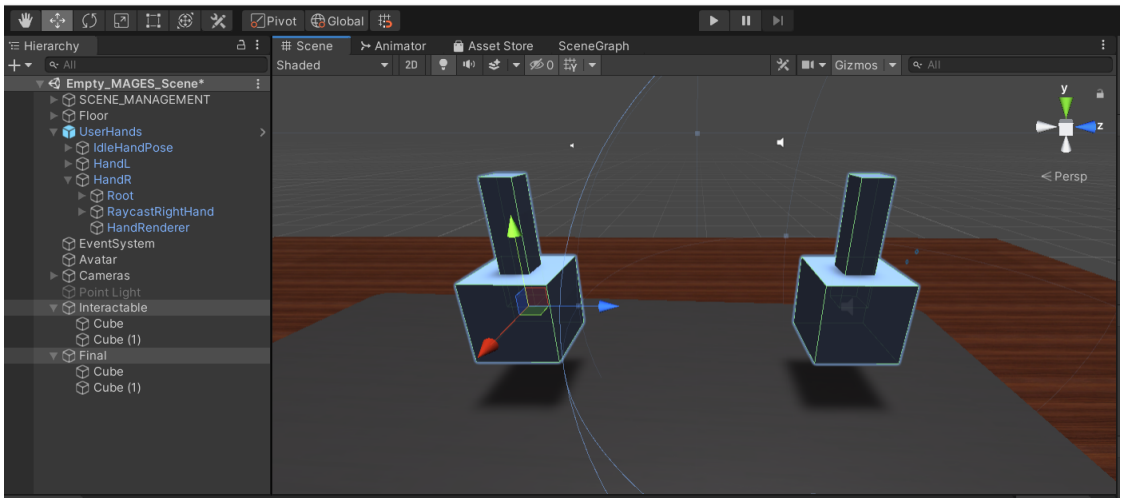


Warning: The Final prefab must have the same pivot with the interactable prefab because the PrefabLerpPlacement script checks if the orientation (position and rotation) of the objects match to perform the Action.

For this reason, the safest way to generate the final prefab is to duplicate the interactable, copy the transform of its root, paste it on the final prefab template and transfer its children to the final prefab.

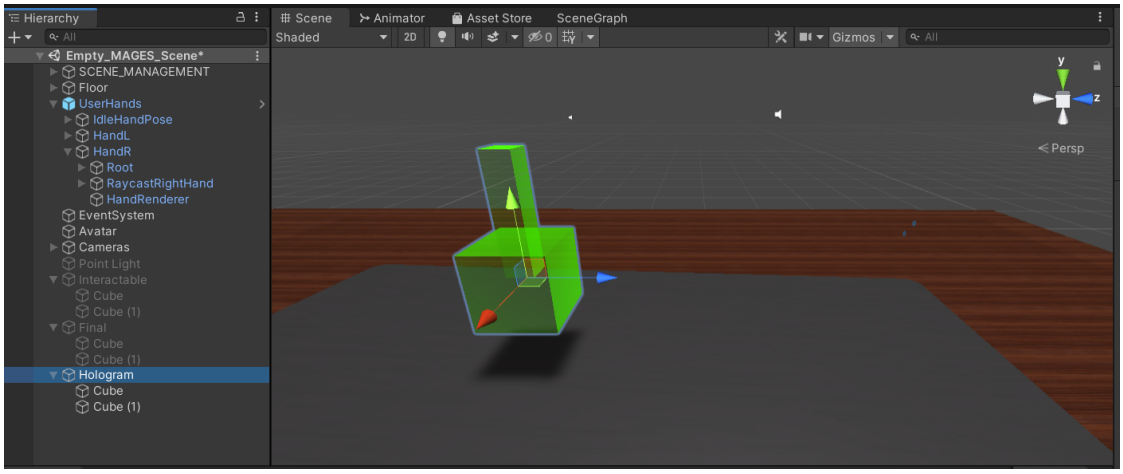
Remember to set its rigidbody component to kinematic and all its colliders to trigger.

The image below shows both the interactable (right) and the final prefab (right).



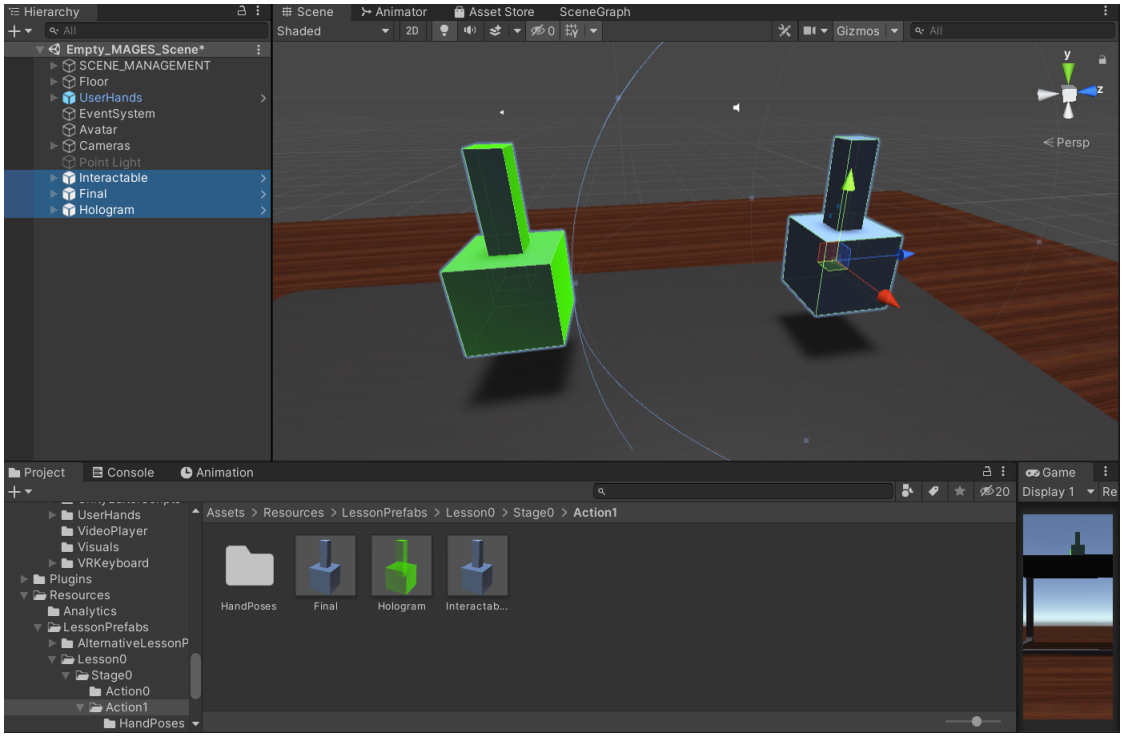
Hologram Prefab

The hologram prefab does not have any component or script attached. It is just a copy of the final prefab with the holographic material. Remember to remove its colliders as well.

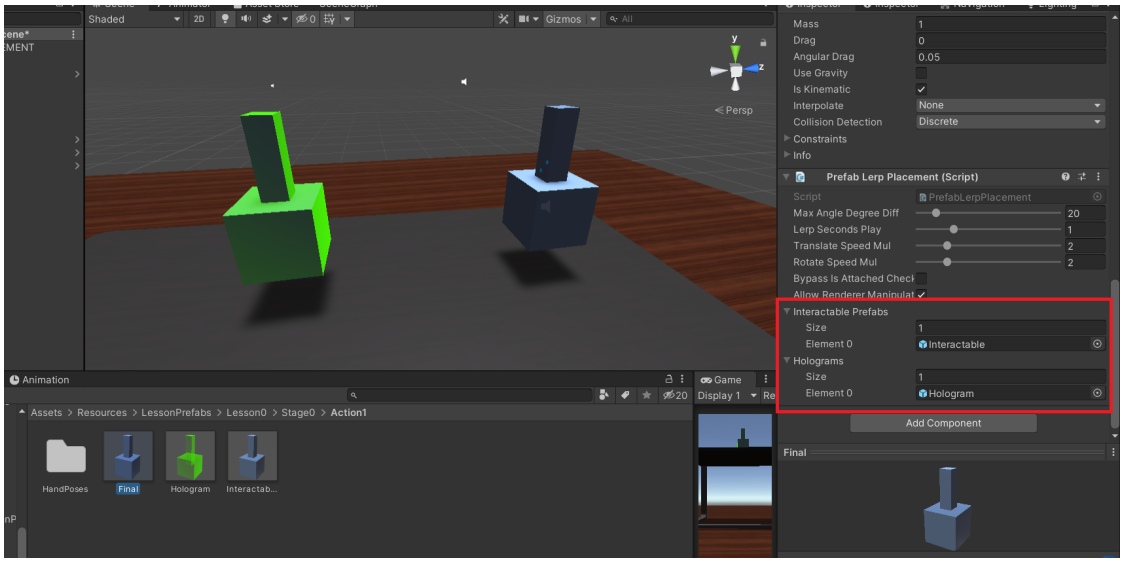


Save prefabs and final configuration

Save the prefabs in the Resources folder. It is recommended to keep the prefabs in folders according to the scenegraph structure. In this case we will save the interactable, final and hologram prefab at Resources/Lesson0/Stage0/Action1 folder.



The final step is to configure the PrefabLerpPlacement script which is attached to our final prefab. This component indicates the interactable prefab that matches with this final prefab as well as the hologram. Additionally you can set up properties like the tolerance in angle difference with the interactable or set up the lerping behavior. The image below shows the interactable along with the hologram prefab linked with the PrefabLerpPlacement component.

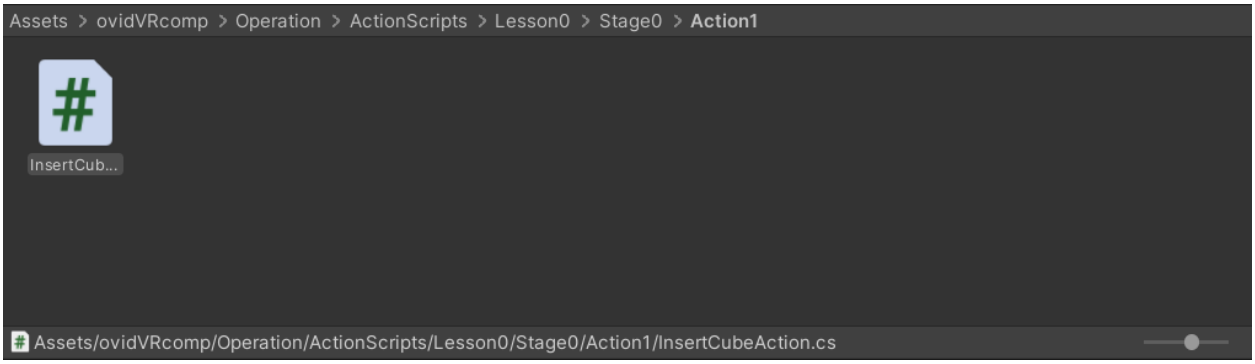


Action Script

In this step we will write the Action script. The script below initializes our interactable and final prefab and spawns the hologram prefab as well.

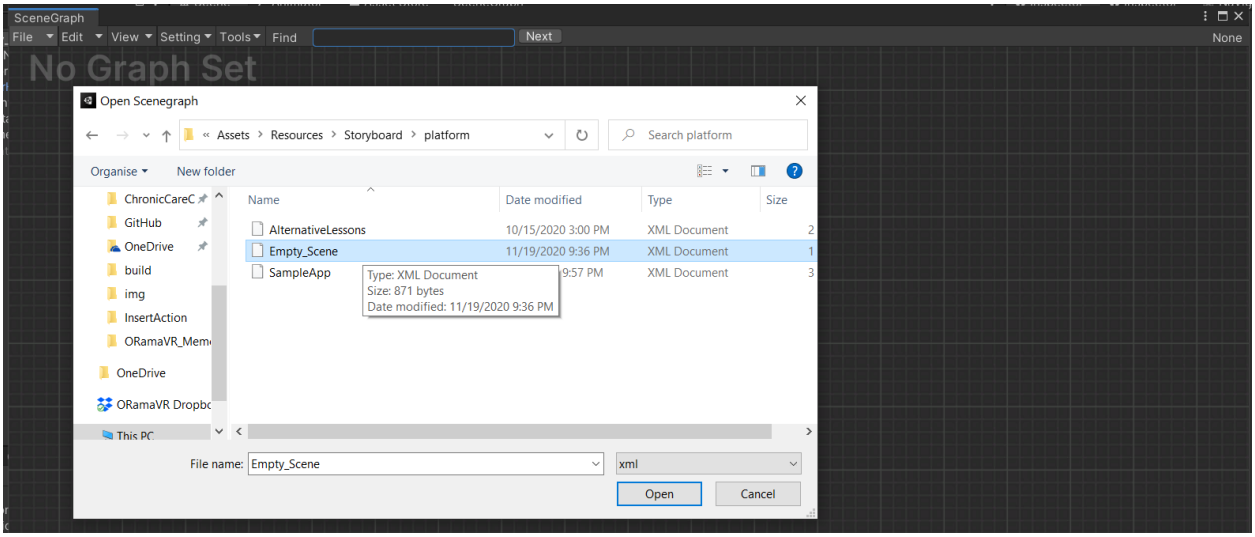
```
using MAGES.ActionPrototypes;
public class InsertCubeAction : InsertAction
{
    public override void Initialize()
    {
        SetInsertPrefab("Lesson0/Stage0/Action1/Interactable", "Lesson0/Stage0/
↪Action1/Final");
        SetHoloObject("Lesson0/Stage0/Action1/Hologram");

        base.Initialize();
    }
}
```

We save the Action script in path following the scenegraph structure

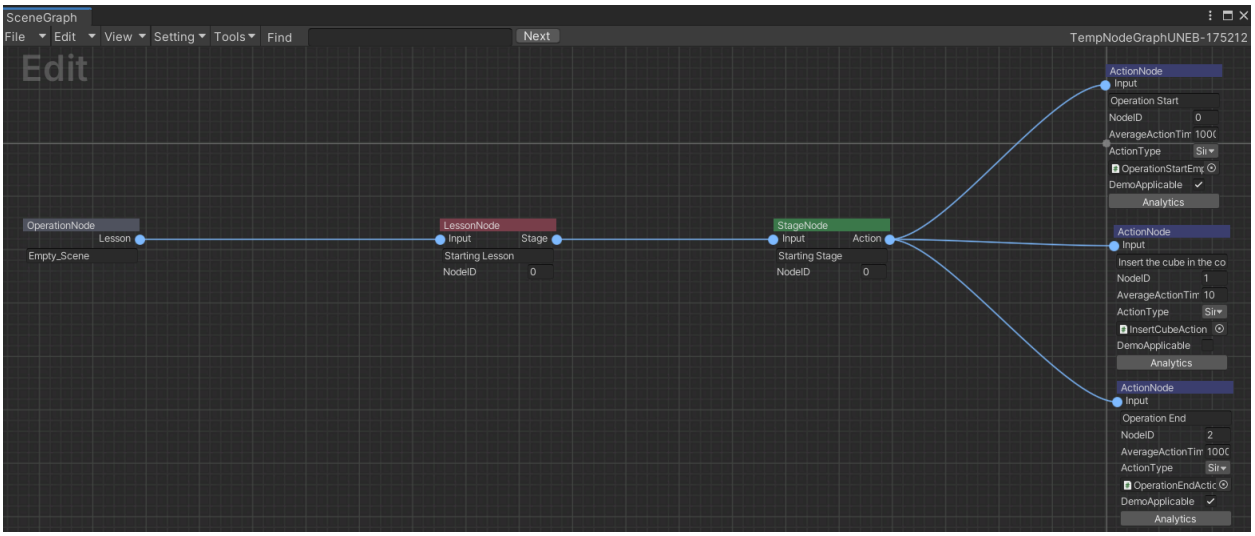


Add the Action to Scenegraph

The final step is to link the ActionScript to the scenegraph. From the MAGES menu click Scenegraph Editor. At the Scenegraph Editor tab, click File/Load and the proper .xml import the scenegraph. In this case is the Empty_Scene.xml.



To add a new Action Node right click inside the Scenegraph Editor and select “Action Node”. Fill the Action description, along with the proper NodeID (in this case is the second Action) Finally, add the reference of the Action script.



From the Scenegraph Editor menu, click File/Save to save your changes.

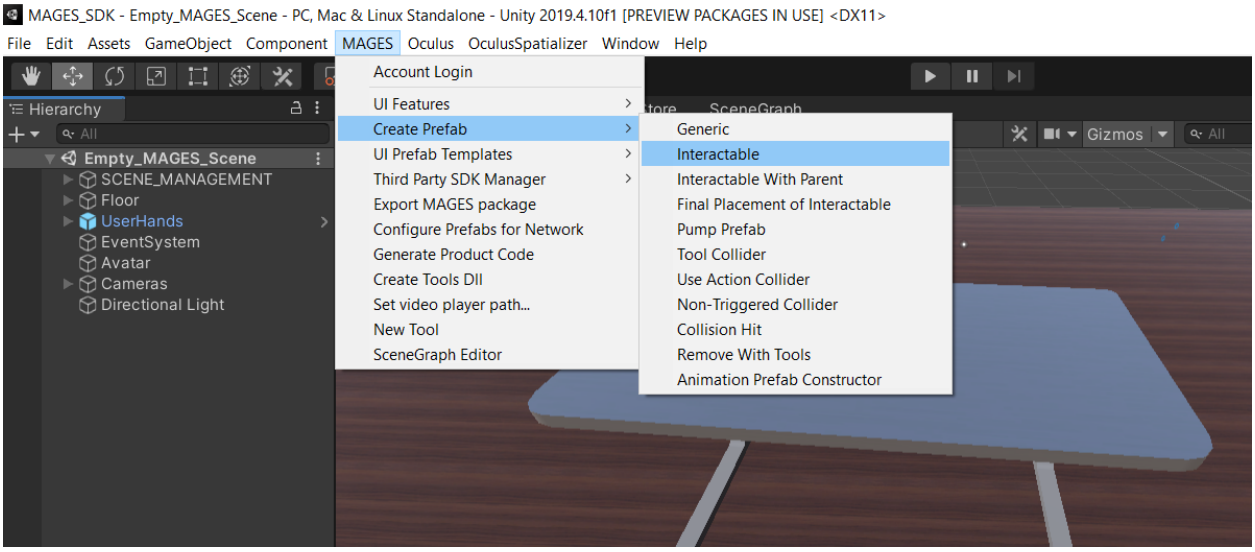
6.1.2 Remove Action

The Remove Action is used when we need to remove a specific object from the scene using our hands or a tool.

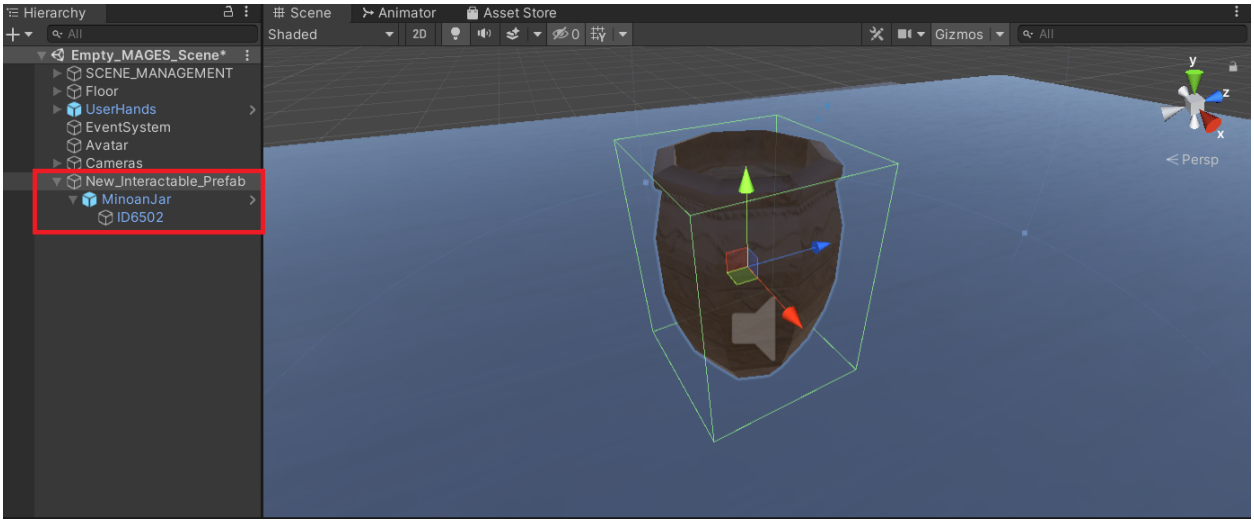
To generate a Remove Action you need the Interactable prefab (the one to be removed) and a hologram.

Remove object with hands

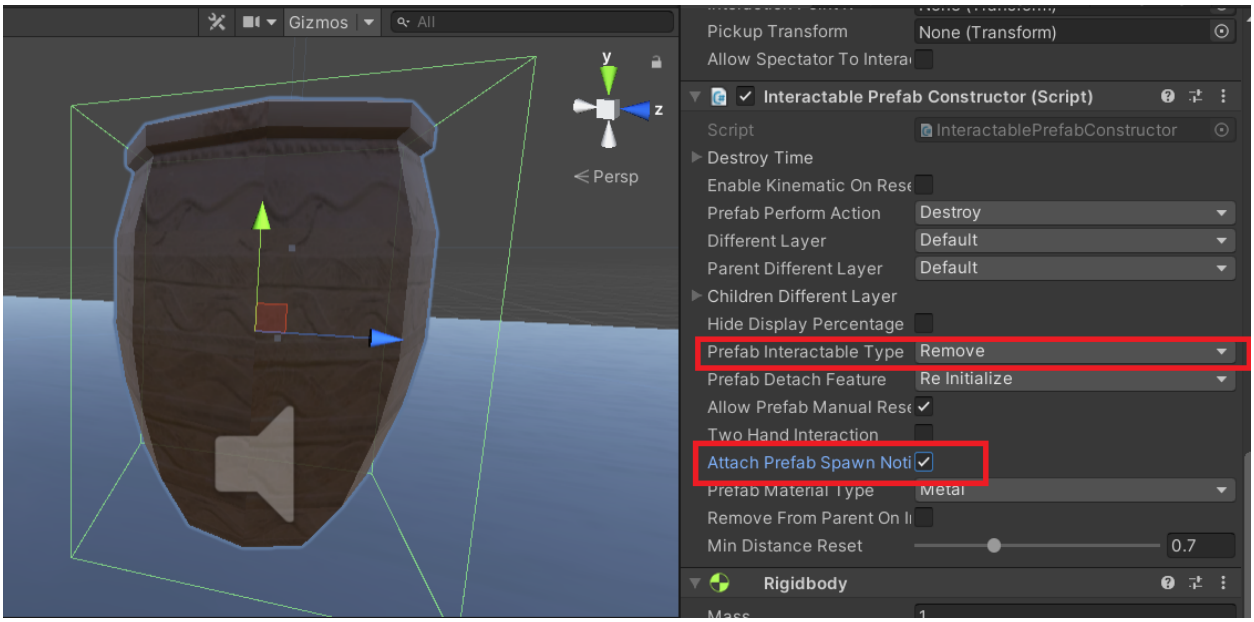
From the MAGES menu select the option CreatePrefab/Interactable



The Interactable prefab template will be instantiated. In this example we will remove a jar, thus we populate the prefab template with the jar model as seen in the image below.

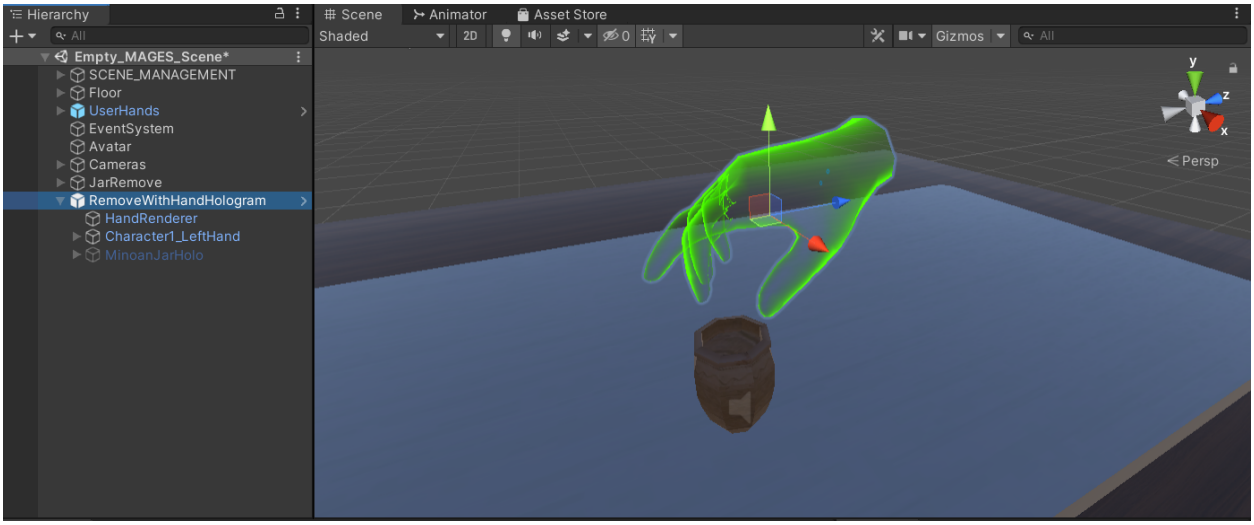


The only configuration we need is to set from the `InteractablePrefabConstructor` script the `prefabInteractableType` value to `Remove` It is also recommended to set the `AttachPrefabSpawnNotifier` value to `True`. This will set a flashing animation to the object, indicating its removal.

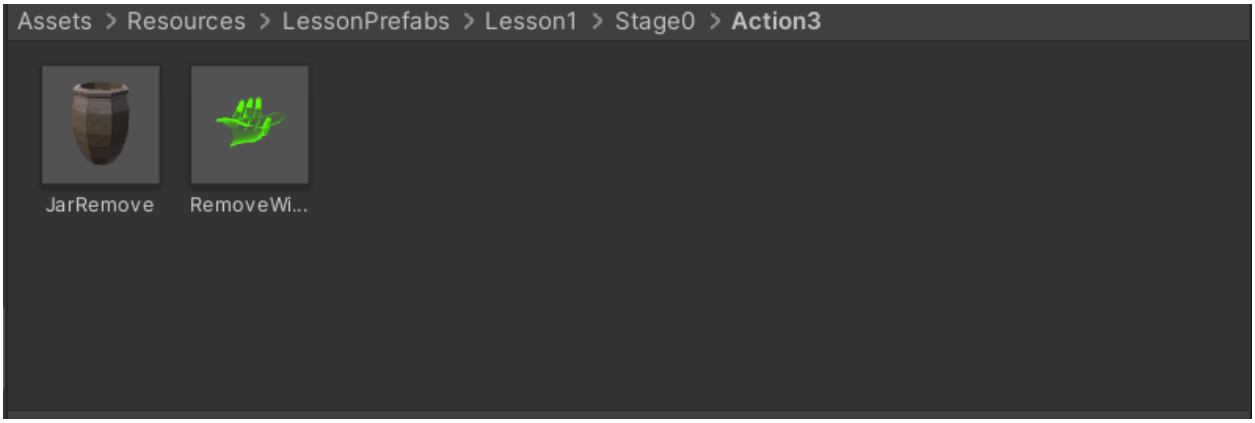


Don't forget to add proper physical colliders to the object.

Additionally, we generate an animated hologram showing a holographic hand removing the jar.



Finally, we save the prefabs in our LessonPrefab folder



Action Script

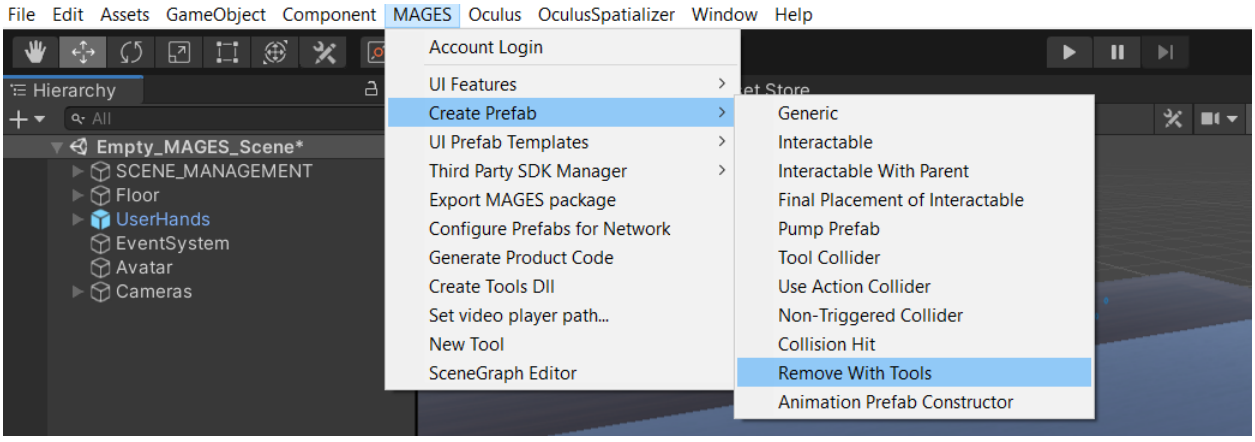
The script below initializes the interactable prefab and spawns the hologram as well.

```
using MAGES.ActionPrototypes;
public class RemoveJarAction : RemoveAction
{
    public override void Initialize()
    {
        SetRemovePrefab("Lesson1/Stage0/Action3/JarRemove");

        SetHoloObject("Lesson1/Stage0/Action3/RemoveWithHandHologram");

        base.Initialize();
    }
}
```

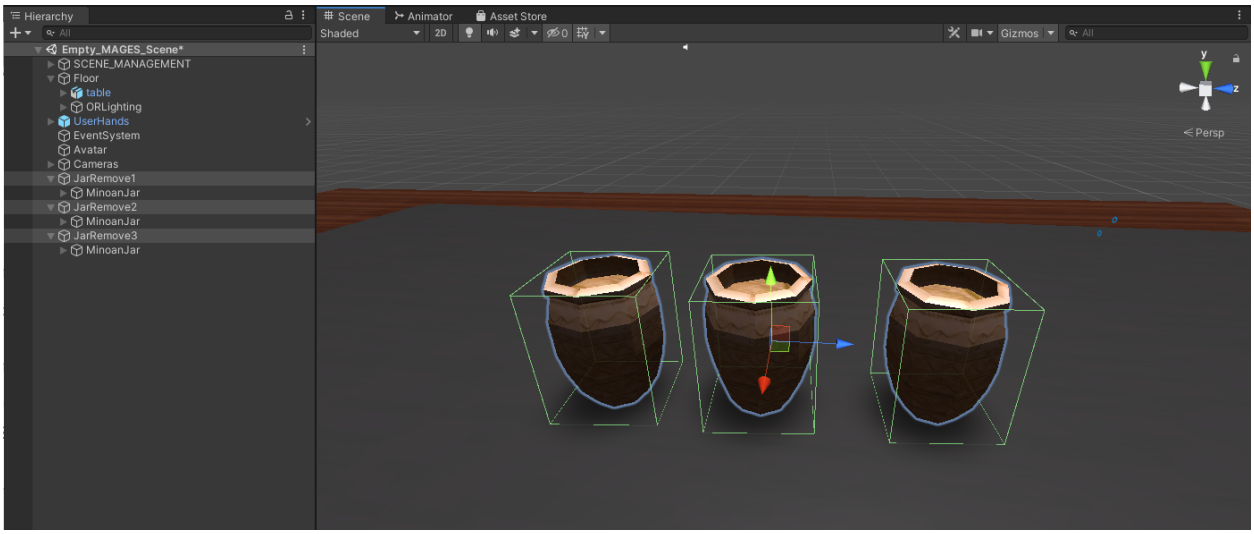
Remove object with tools

In this case, we will remove three jars using a tool (the pliers) instead of our hands.

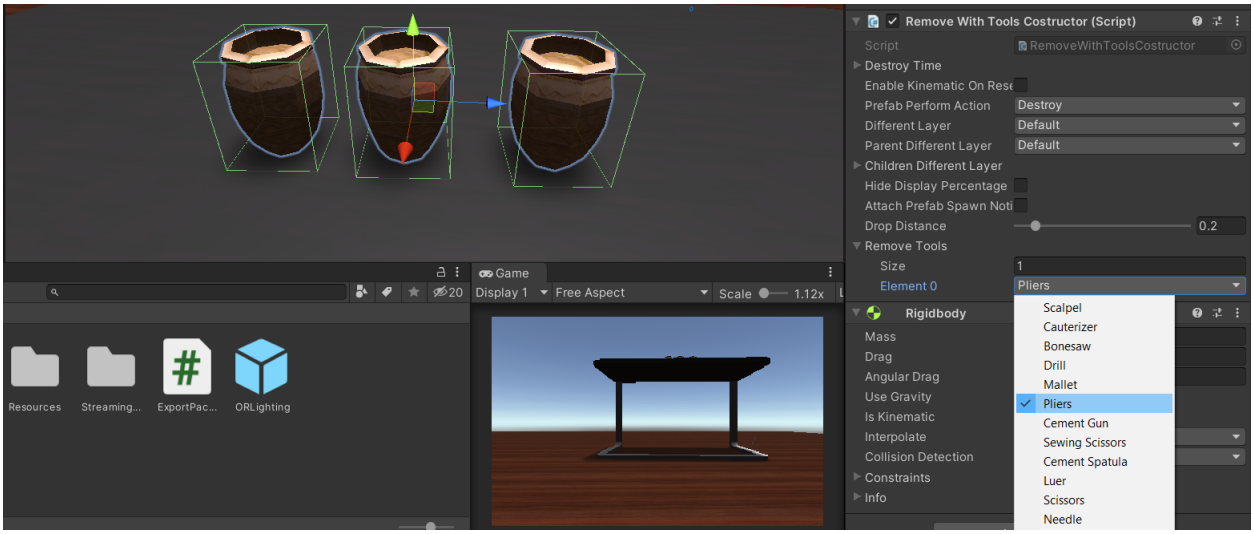
From the MAGES menu select the option CreatePrefab/Remove With Tools



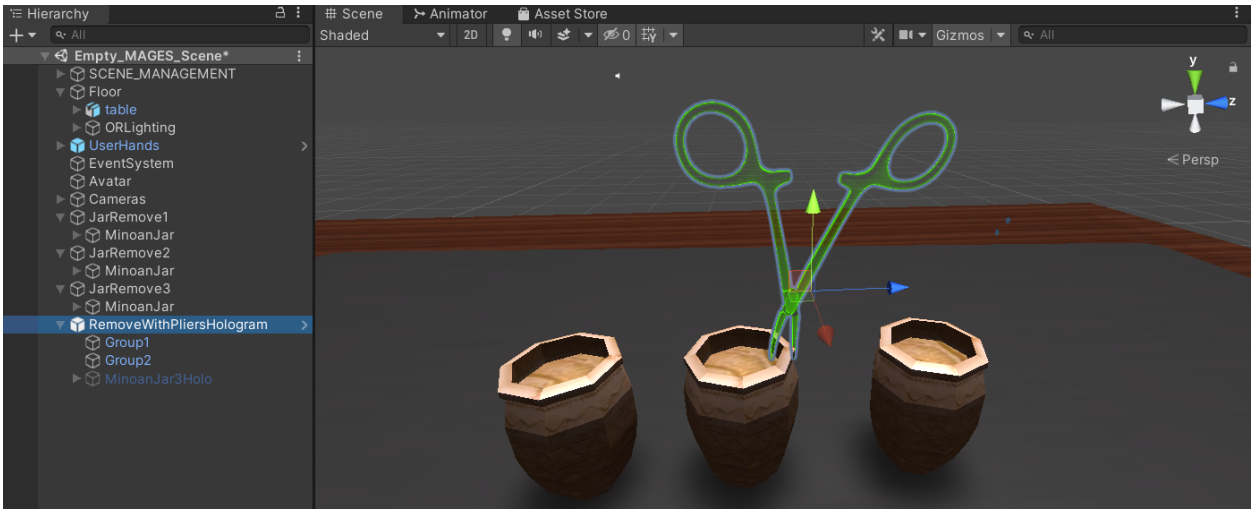
This time we will generate three prefabs as seen below.



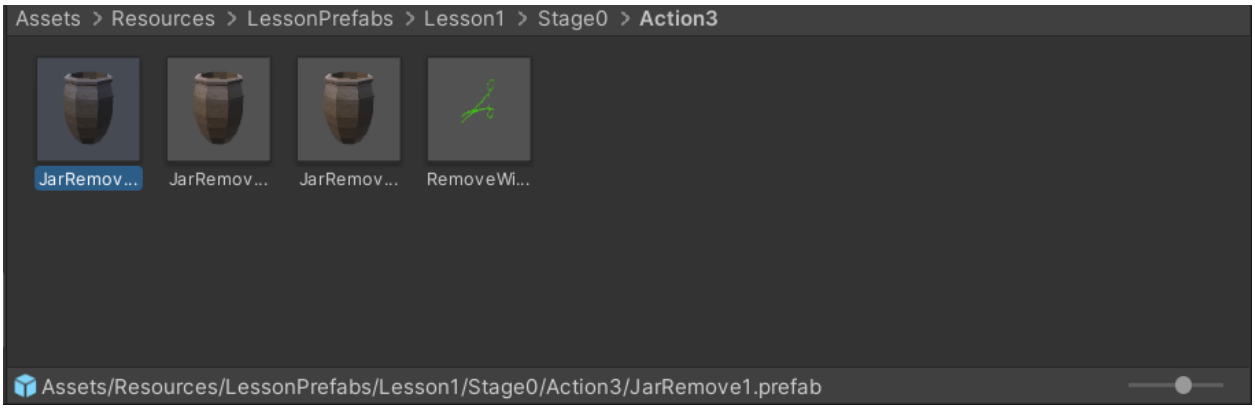
Now, we will configure our remove prefabs to be removed with the pliers tool. At the `RemoveWithToolsConstructor` script which is attached to your prefab, set the `RemoveTools` size to 1 and from the dropdown menu select the tool you want. In this case we selected the Pliers. We will do the same configuration to all the three remove prefabs.



Additionally, we make an animated hologram.



Finally, we save the prefabs in our LessonPrefab folder



Action Script

In this Action we would like to remove three jars using the pliers instead of one, thus the Action script will be the following.

```
using MAGES.ActionPrototypes;
public class RemoveJarsWithPliersAction : RemoveAction
{
    public override void Initialize()
    {
        SetRemovePrefab("Lesson1/Stage0/Action3/JarRemove1");
        SetRemovePrefab("Lesson1/Stage0/Action3/JarRemove2");
        SetRemovePrefab("Lesson1/Stage0/Action3/JarRemove3");

        SetHoloObject("Lesson1/Stage0/Action3/RemoveWithPliersHologram");

        base.Initialize();
    }
}
```

Note: The above RemoveAction script sets three removable prefabs. In this case, the Action will Perform when all

three are removed with the pliers.

6.1.3 Use Action

The Use Action is used in situations where we need to take an object and hold it in a specific area for a predefined period of time.

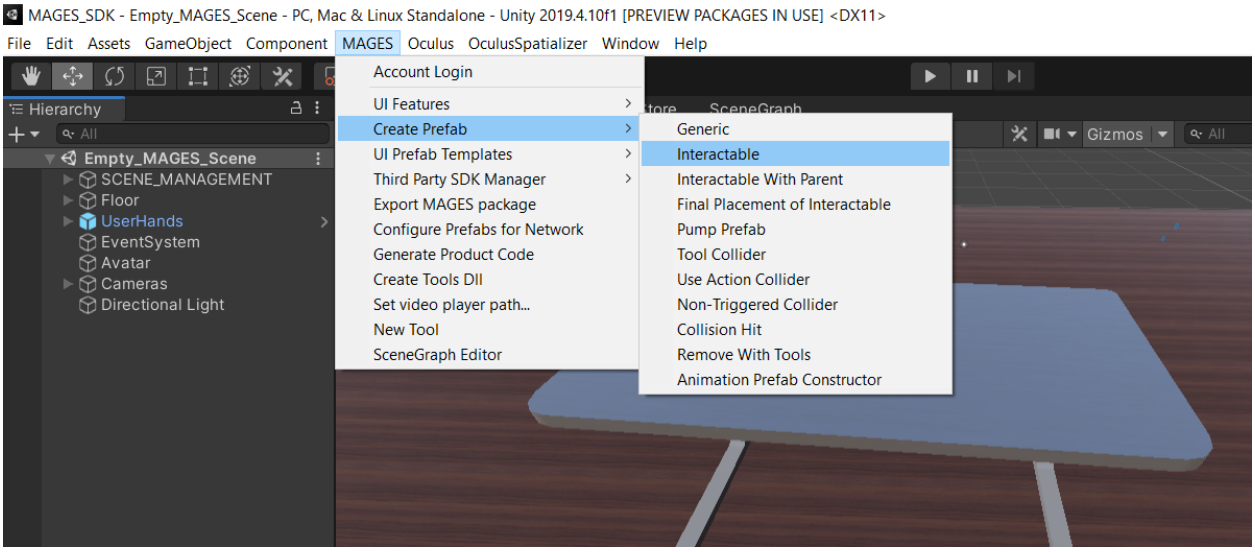
To generate a Use Action you need:

1) The interactable prefab
2) The use collider
3) An animated hologram

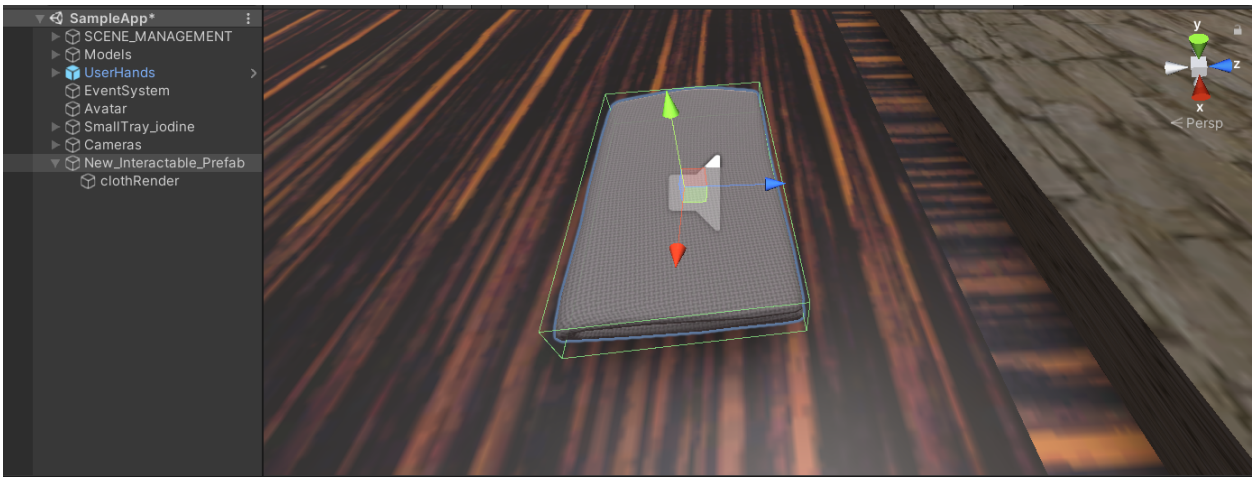
In this tutorial, we will implement a Use Action where the users should take a cloth to clean the Sponza model.

Interactable prefab

From the MAGES menu select the option CreatePrefab/Interactable

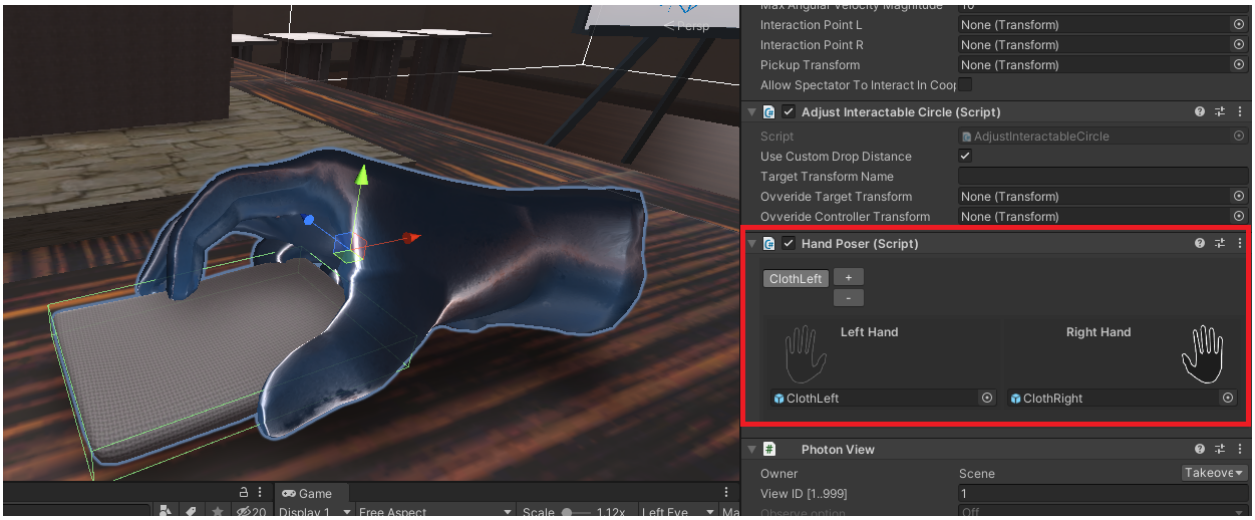


We populate the prefab template with a cloth model as seen in the image below. Remember to add physical colliders.



Note: It is recommended to set the `AttachPrefabSpawnNotifier` value from the `InteractablePrefabConstructor` script to `True`. This will enable a flashing animation till the object is grabbed.

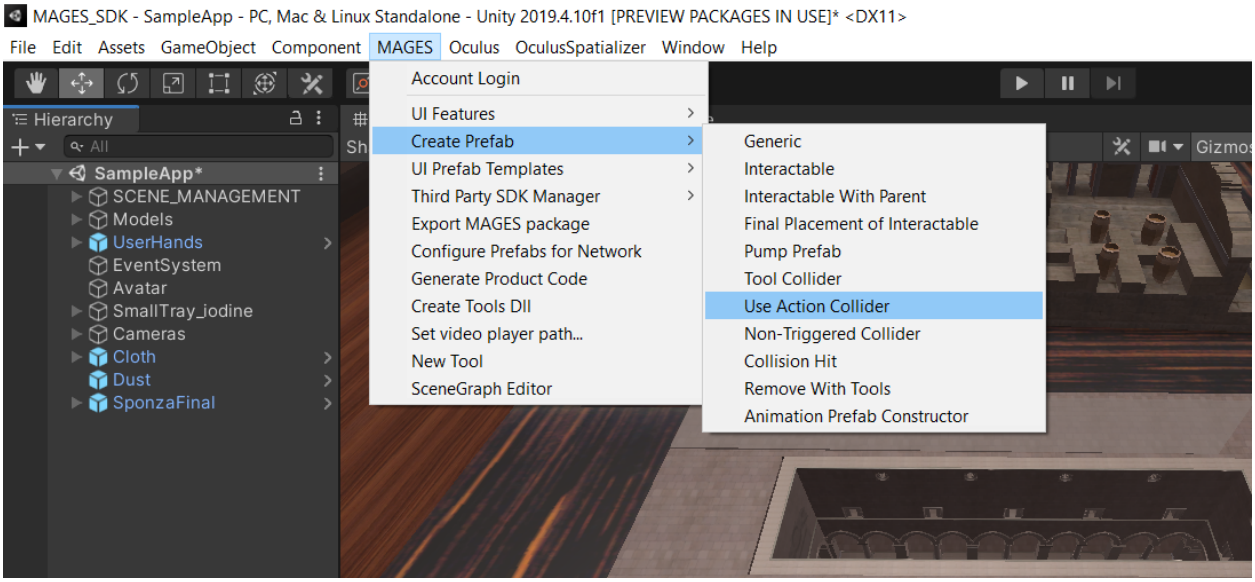
We also add custom hand postures for our interactable prefab. You can read *here* a detailed tutorial on how to properly setup hand postures. The image below shows the posture of the right hand attached on our object.



Use collider

This is the collider that will trigger with the cloth.

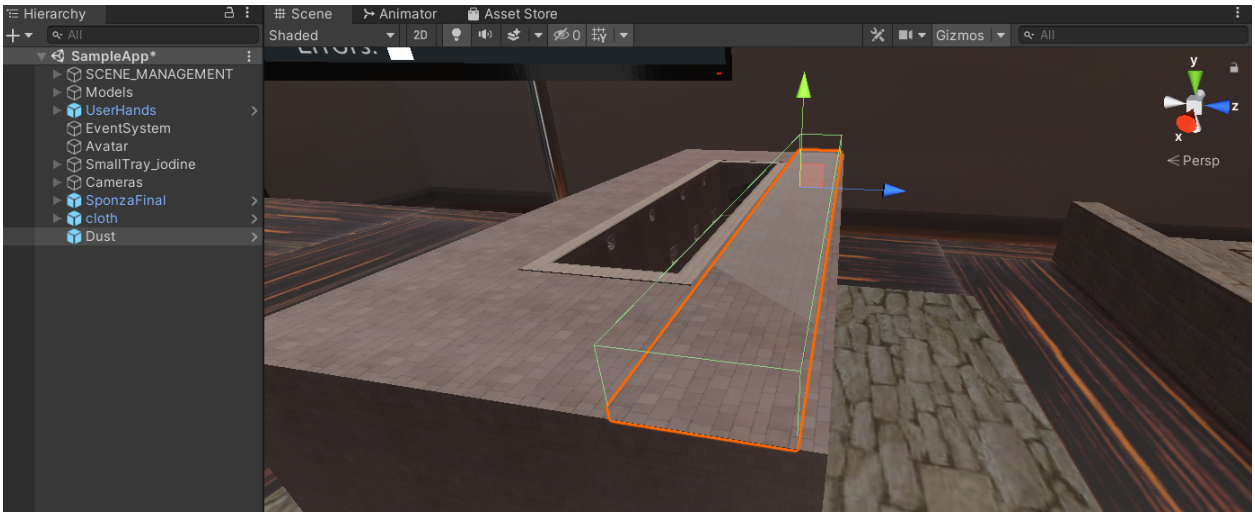
From the MAGES menu select the option CreatePrefab/Use Action Collider



In this case we add a dust textured model to represent the dust particles on top of the Sponza model.

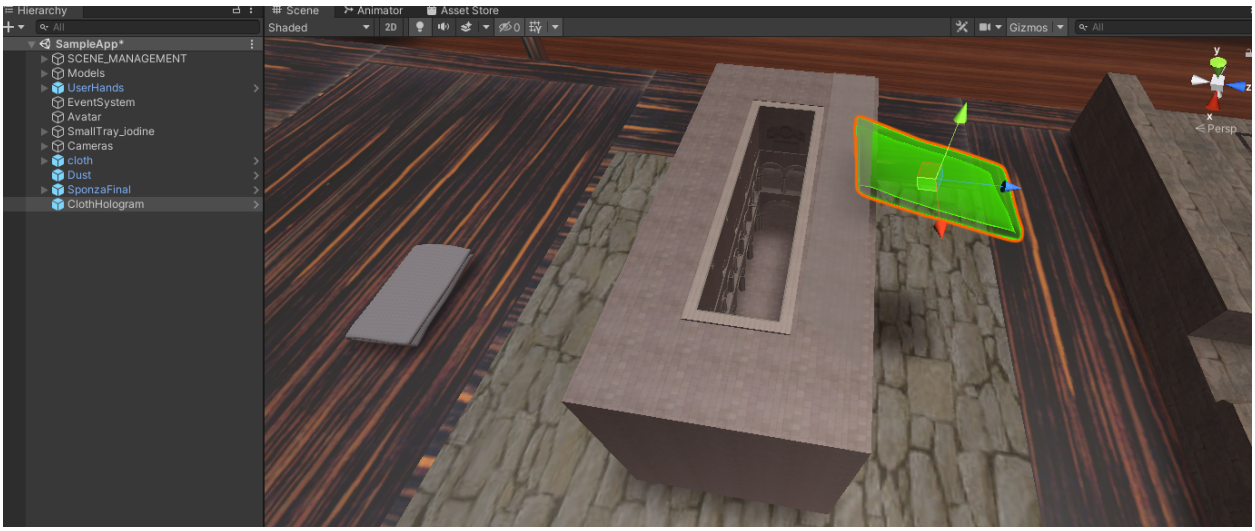
Note: Remember to add a trigger collider to your prefab

Below you can see the final use collider.



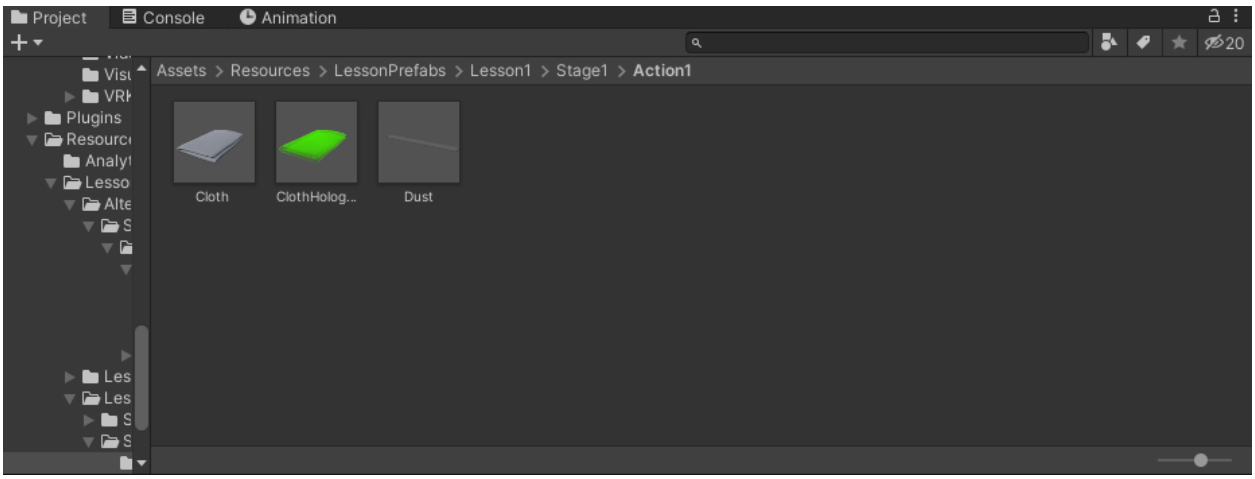
Hologram

To visualize the Action we will include an animated hologram indicating the interactable prefab and how to use it. This hologram will visualize the cloth hovering on top of the sponza. Below you can see the holographic cloth.

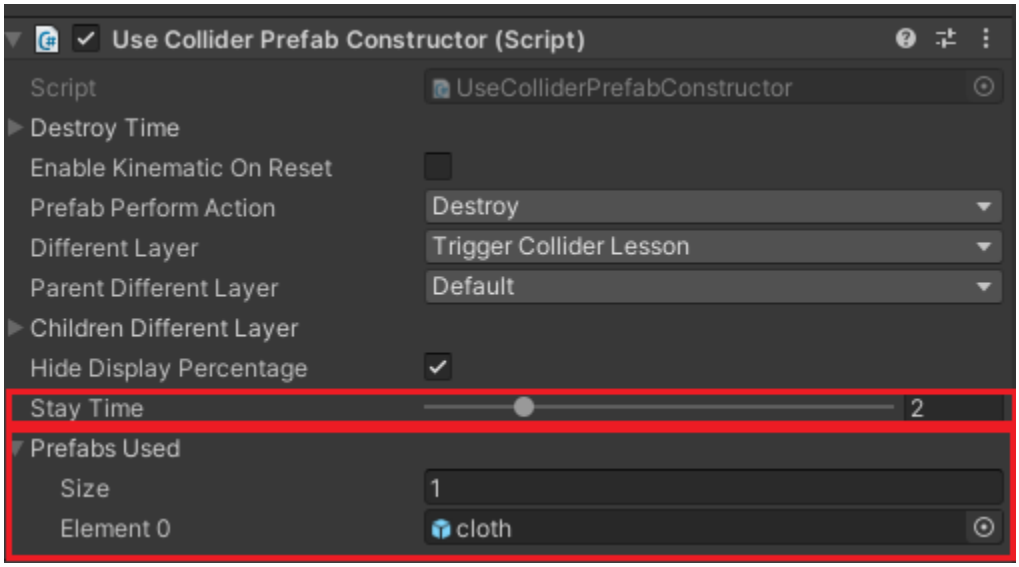


Save prefabs and final configuration

Save the prefabs in the Resources folder. It is recommended to keep the prefabs in folders according to the scenegraph structure. In this case we will save the interactable, use collider and hologram prefab at Resources/Lesson1/Stage1/Action1 folder.



Now we have to link the use collider with the interactable prefab. Navigate to our use collider (dust prefab) and from the `UseColliderPrefabConstructor` you need to set the `PrefabsUsed` variable to `1`. Then, drag and drop the interactable prefab (cloth) at the Element 0 position. Additionally, set the `StayTime` variable to the amount of time that the cloth needs to stay in contact with the use collider. In this case, we set it to `2` seconds, meaning that we have to keep the cloth in contact with the collider for 2 seconds to Perform.



Action Script

The script below initializes the use prefab.

```
using MAGES.ActionPrototypes;
public class CleanSponzaAction : UseAction
{
    public override void Initialize()
    {
        SetUsePrefab("Lesson1/Stage1/Action1/Cloth", "Lesson1/Stage1/Action1/Dust");

        SetHoloObject("Lesson1/Stage1/Action1/ClothHologram");

        base.Initialize();
    }
}
```

6.1.4 Tool Action

The Tool Action is used when we need to use a tool to complete a specific task. Examples of tool Actions may be a skin incision with the scalpel, or the cauterization of fat using the cautery.

It is important to mention here that the Tool Action has a lot of similarities with the Use Action. Both Actions need a collider and a specific object to complete a task. However, the only difference is that Tool Actions needs a tool whereas the Use Action is implemented with just a prefab.

Note: Tools are unique entities in MAGES SDK. They are spawned at the beginning of each scenario and in many cases support additional functions like turning on/off.

In this tutorial we will implement the drilling of the femoral canal.

To generate a Tool Action you need:

1. A tool

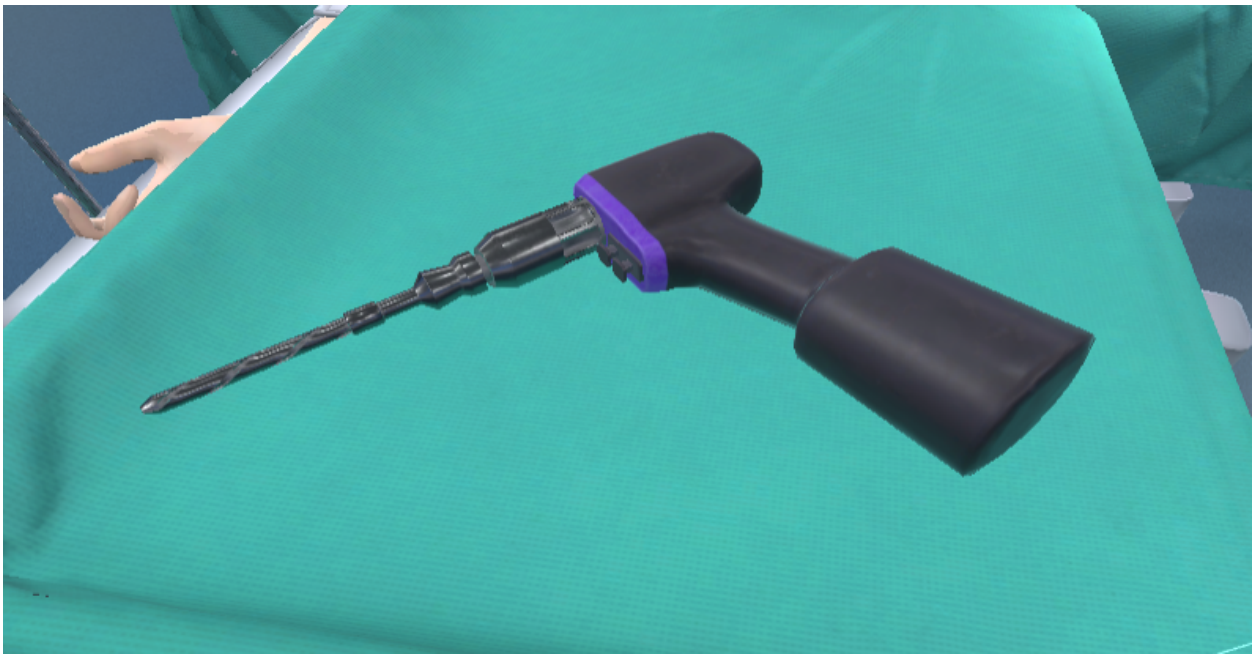
2. A tool collider
3. A hologram (optional)

The tool

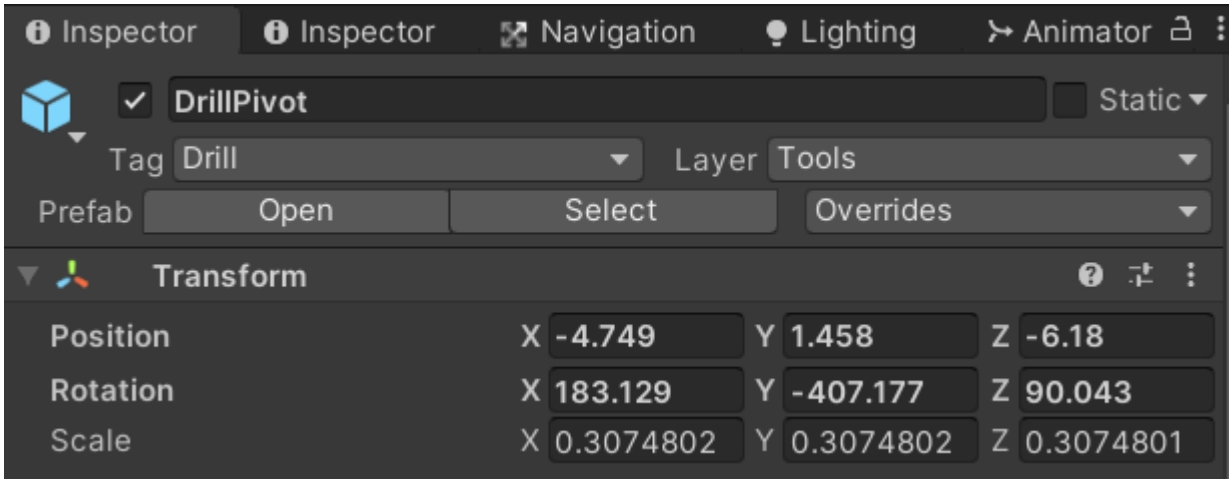
We will use the drill as our tool.

Tools difeer from other prefabs in the MAGES SDK since they have a refence saved in the **ToolsEnumDLL.dll**. This dll file is used to store the tools as data types to use them later in Action scripts.

In our case the **Drill** tool is already provided by the MAGES SDK as a part of the standard tool assets.

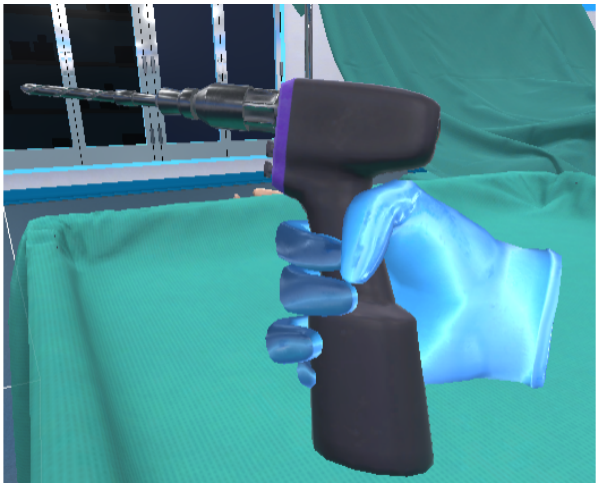


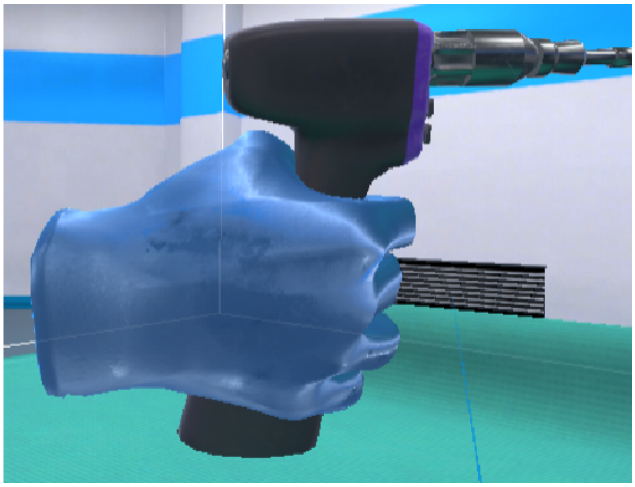
You can see in the inspector, the tool has its own tag (Drill) and Layer (Tools).



For all the tools it is recommended to add the `HandPoser` component with the proper grab postures to improve the interaction with the tool.

You can read *here* a detailed tutorial on how to properly setup hand postures.



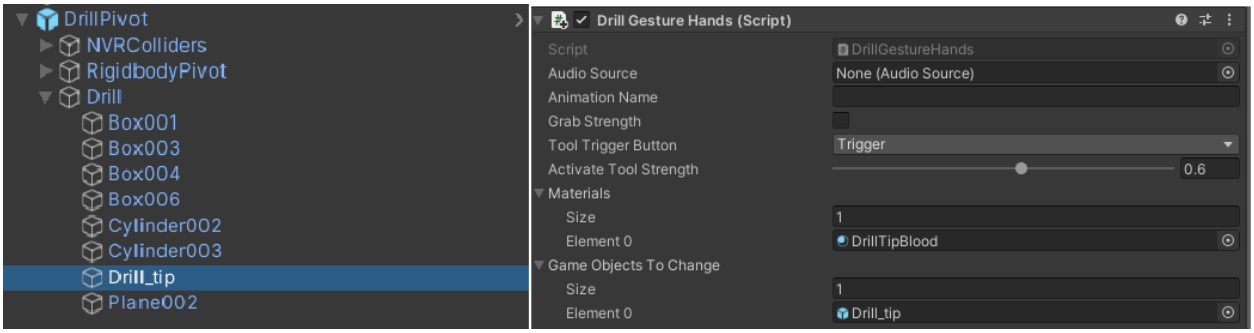


Drill gesture hands

The drill is not a static tool. To operate it you have to press the dill button. In MAGES SDK we have implemented this behavior in the `GestureHands` base class. Developers can inherit from this script, to generate their own tool behaviors. For example if you need the cautery to emit smoke particles when turned on, you should implement a `CautreyGestureHands` script and inherit from the `GestureHands`. Then you can code the tool's behavior.

In our scenario we need the drill tip to rotate when operating and change the rotating tip material to a bloody texture when inside the femoral canal.

The first step is to write the `DrillGestureHands` script and attach it to our drill tip gameObject as showed below.



Below you can see a snippet from the `DrillGestureHands` script.

```
public class DrillGestureHands : GestureHands {

    public Material[] materials;
    public GameObject[] gameObjectsToChange;

    private MAGESInteractableItem toolInteractable;

    public void SetRotatingParts(params GameObject[] GO)
    {
        rotatedParts.AddRange(GO);
    }

    public override void EndToolGesture()
    {
```

(continues on next page)

(continued from previous page)

```
        GetComponent<Renderer>().material = materials[0];

        base.EndToolGesture();
    }

    private void Start()
    {
        fullRotation = true;
        SetRotatingParts(gameObject);

        toolInteractable = tool.toolGameobject.GetComponent<MAGESInteractableItem>();
    }

    public void ChangeToolMaterial(int _materialNumber)
    {
        // If tool is not grabbed and called to change to bloody material - ignore
        if (!toolInteractable.IsAttached && _materialNumber == 1)
            return;

        if (gameObjectsToChange.Length != 0 || materials.Length != 0)
        {
            foreach (GameObject gO in gameObjectsToChange)
                gO.GetComponent<Renderer>().material = materials[_materialNumber];
        }

    }
}
```

You can override the virtual functions from the `GestureHands` to implement a custom tool behavior.

In our case we added the `ChangeToolMaterial` and the `SetRotatingParts` methods. The first one will change the texture of the tip with the assigned bloody material while the second one will rotate the drill tip upon operating the tool by pressing the trigger button.

Save your tool into a prefab.

As mentioned before all the tools are spawned from the beginning of the operation. This is mainly because we need the tools to be present constantly to the virtual environment.

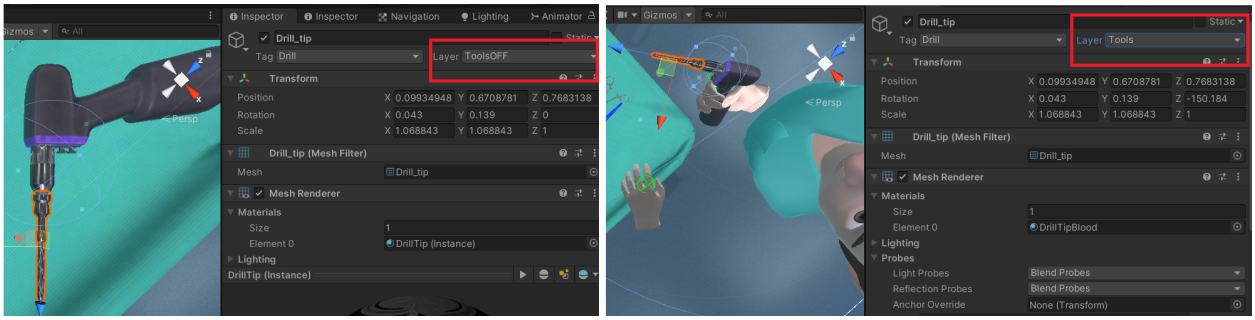
The best way to spawn the tools is at the beginning of our scenario, in the `Perform()` method of the OperationStart Action.

Below there is a method which is called at the `Perform()` function in OperationStart.

```
private void SpawnNetworkPrefabs()
{
    PrefabImporter.SpawnActionPrefab("Medical/Lesson0/Stage0/Action1/PliersPivot");
    PrefabImporter.SpawnActionPrefab("Medical/Lesson0/Stage0/Action1/ScalpelPivot");
    PrefabImporter.SpawnActionPrefab("Medical/Lesson0/Stage0/Action1/CauterizerPivot
↪");
    PrefabImporter.SpawnActionPrefab("Medical/Lesson0/Stage0/Action1/DrillPivot");
}
```

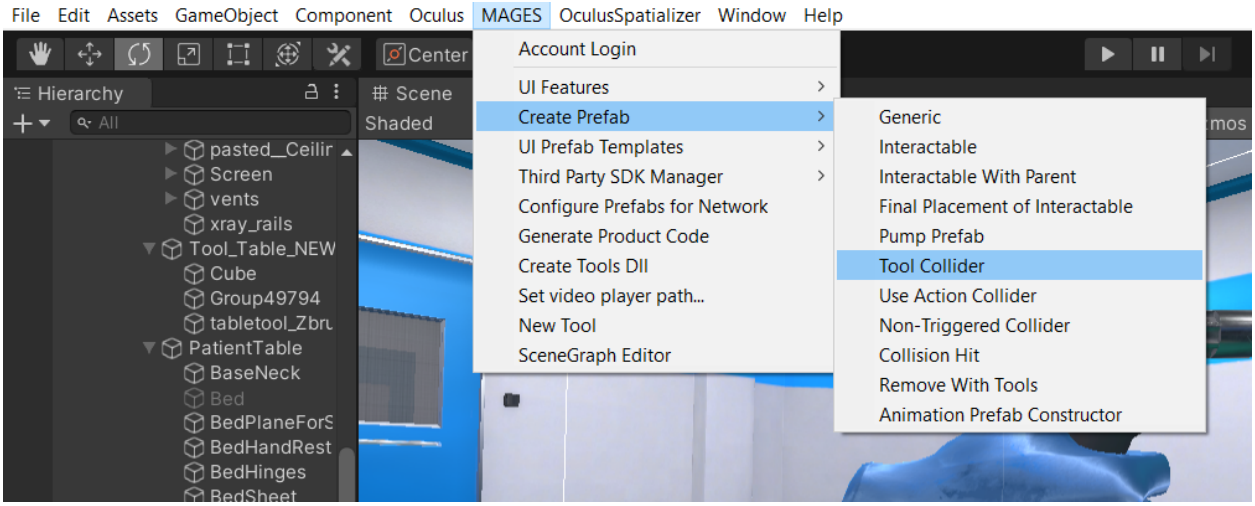
As you can see all the tools are spawned in this way.

> **Warning:** Pay attention to the tool's layers when operating the drill. The Drill_tip gameObject is initially registered with the layer toolsOFF. This layer will not trigger the tool collider. Once turning on the drill, the tip changes layer to Tools to trigger the tool collider.



The tool collider

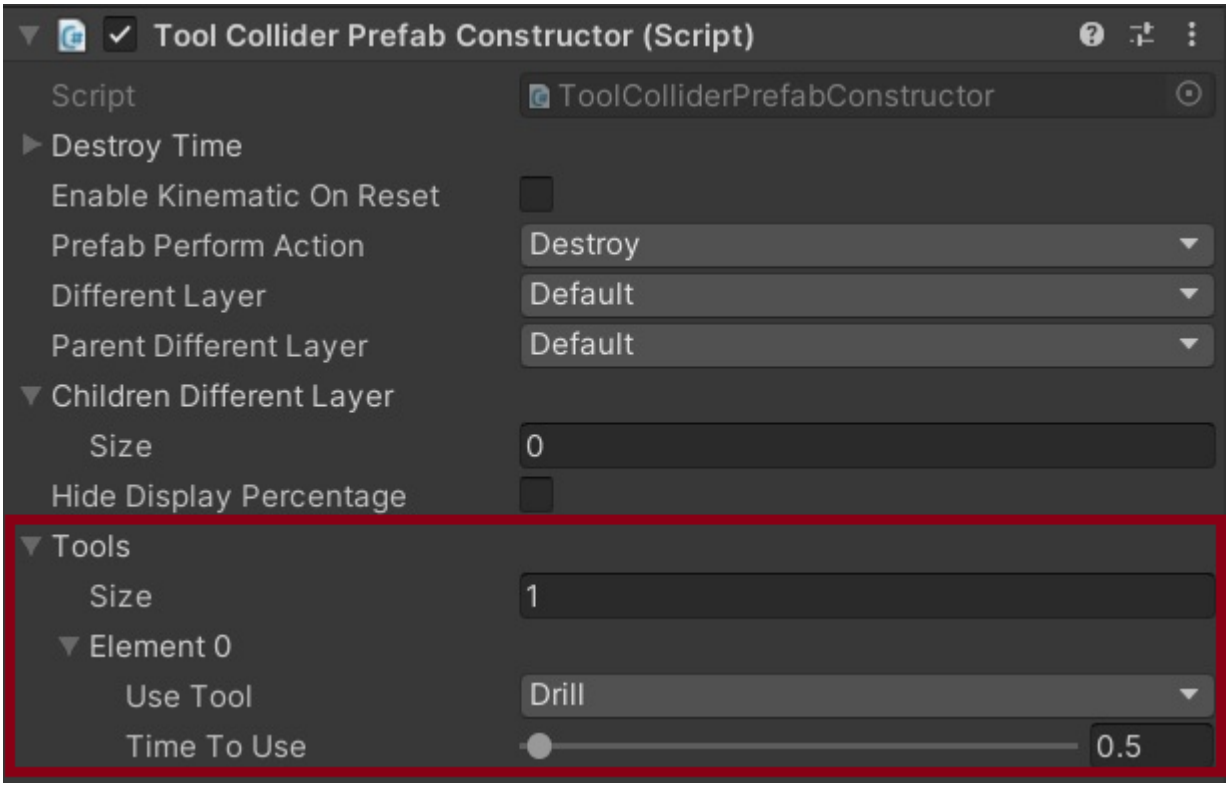
To generate the tool collider navigate to **MAGES/Create Prefab/Tool Collider** and the tool collider prefab template will appear.



You will see a **New_Tool_Collider_Prefab** object in the scene. This is the template for our tool collider.

In this scenario we need use the drill to open the intramedullary canal on the patient's femur. To implement this behaviour, we need to place the tool collider at the bottom of the femoral canal to trigger the collider only when the drill is in the proper depth.

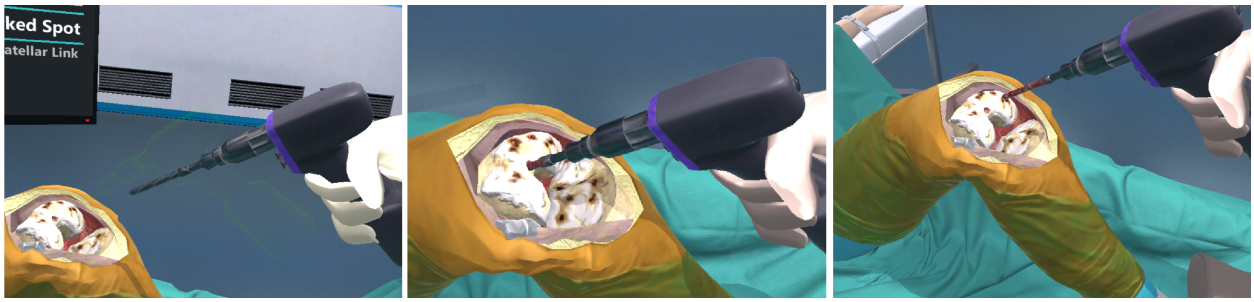
It is important to make some configuration to the `ToolColliderPrefabCosnstructor` script which is attached to the tool collider object. Make sure you select the proper tool that triggers the tool collider, in our case the drill. The `TimeToUse` variable sets the time we need to stay in contact with the collider to Perform the Action.



The drill needs to stay in contact with the tool collider for 0.5 second to perform.

Note: Of course we need physical colliders to form the femoral canal in order for the drill navigate properly to the canal

---



The image above shows the steps of this Tool Action. We take the drill (the tip is still clean) and drill the femoral canal. When we are a drill's tip length inside the canal we trigger the tool collider and the Action is performed. Additionally, when touching the interior of the femoral canal, the drill tip changes material into a bloody one.

Action Script

The script below initializes the interactable prefab and spawns the hologram as well.

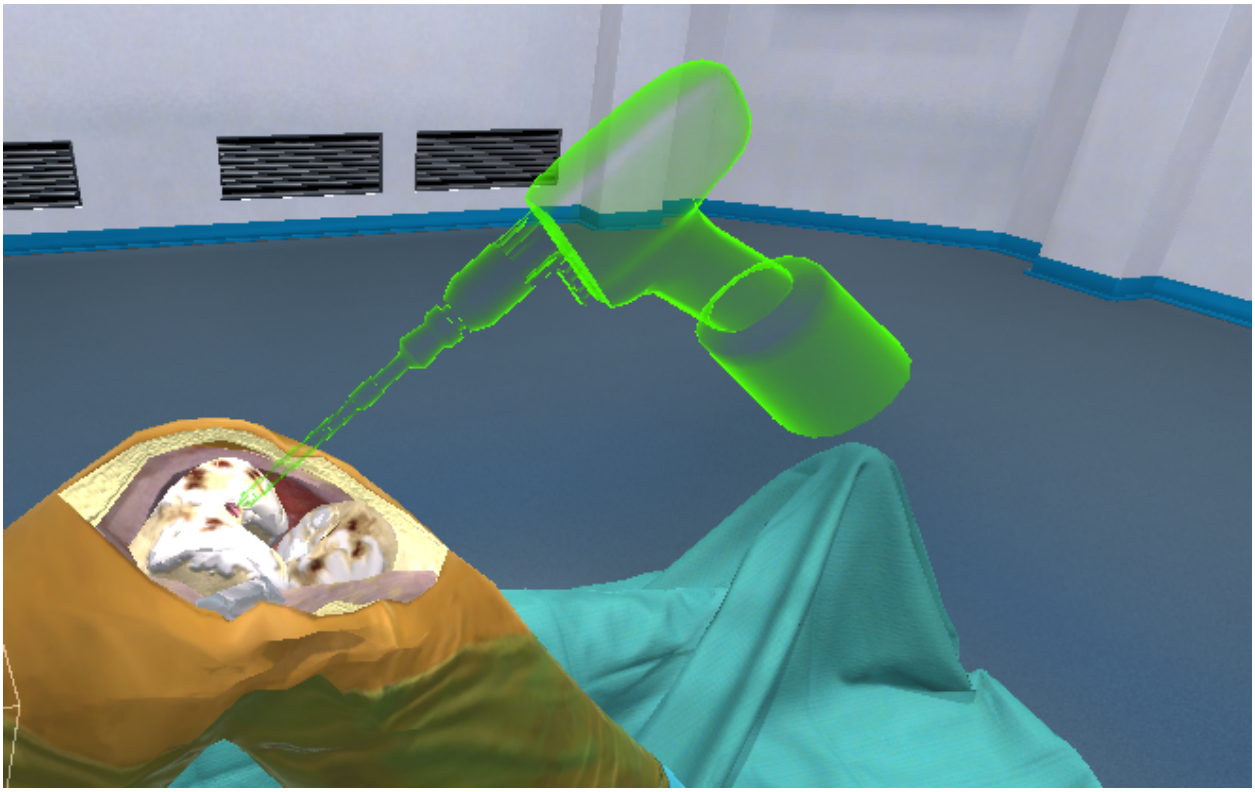
```
using MAGES.ActionPrototypes;

public class DrillKneeAction : ToolAction
{
    public override void Initialize()
    {
        SetToolActionPrefab("Medical/Lesson1/Stage0/Action0/DrillCollider", MAGES.
↪ToolManager.tool.ToolsEnum.Drill);

        SetHoloObject("Medical/Lesson1/Stage0/Action0/Hologram/DrillHolo");

        base.Initialize();
    }
}
```

The `SetToolActionPrefab` method takes two arguments.

1. The drill collider
2. A ToolsEnum enumerator with the assigned tool to trigger our collider

This is why we need the tools defined in the ToolsEnumDLL, in order to use them as data types in Action scripts.

Finally, we add an animated drill hologram showing the proper movement.



6.1.5 Combined Action

The Combined Action does not include any new VR behavior, it is a way to include multiple Actions in the same script. Combined Actions are useful in situations where we want to implement sequential tasks but incorporate them into a single entity.

In this example, we will convert the *UseAction* and the *RemoveAction* from the previous tutorials into a CombinedAction. As a result, in this Action the user would be asked to clean the sponza with a cloth (UseAction) and then remove the jar using his hand (RemoveAction).

Action Script

The script below configures the Combined Action.

```
using MAGES.ActionPrototypes;
public class UseAndCleanAction : CombinedAction
{
    public override void Initialize()
    {
        //Use sub-Action
        UseAction cleanSponzaAction = gameObject.AddComponent<UseAction>();
        cleanSponzaAction.SetUsePrefab("Lesson1/Stage1/Action1/Cloth", "Lesson1/
↪Stage1/Action1/Dust");
        cleanSponzaAction.SetHoloObject("Lesson1/Stage1/Action1/ClothHologram");
        //-------------------------------------------------------------------------
↪---------------

        //Remove sub-Action
        RemoveAction removeJarAction = gameObject.AddComponent<RemoveAction>();
        removeJarAction.SetRemovePrefab("Lesson1/Stage1/Action1/JarRemove");
        removeJarAction.SetHoloObject("Lesson1/Stage1/Action1/RemoveWithHandHologram
↪");

        InsertIActions(cleanSponzaAction, removeJarAction);

        base.Initialize();
    }

    //Example of Perform() in CombinedAction
    public override void Perform()
    {
        //This method will invoke after all the sub-Action will perform
    }
}
```

Combined Action explanation

As you can see, only the syntax of the Action script has changed while the functions and methodology remains the same.

In this example we developed two sub-Actions: 1) a UseAction where the user needs to clean the sponza model and 2) a RemoveAction where the user is asked to remove the jar.

Note: The sub-Actions will be initialized sequentially, meaning that the `Initialize()` of the UseAction will be called first. When the user completes the Action, the `Initialize()` of the RemoveAction will be called. Finally,

when all the sub-Actions are performed, the `Perform()` of the CombinedAction will be called as well.

The method `InsertIActions()` sets the sequence of the sub-Actions. For example if we set the arguments like this: `InsertIActions(removeJarAction, cleanSponzaAction);` the sub-Actions will be initialized in the reverse order (first the remove then the use). This function is mandatory for the CombinedAction to work properly.

Additionally, you can see that we manually added the sub-Actions as components in the Action's gameObject.

```
RemoveAction removeJarAction = gameObject.AddComponent<RemoveAction>();
```

To access each sub-Action's methods use the sub-Action variable.

```
removeJarAction.SetRemovePrefab("Lesson1/Stage1/Action1/JarRemove");
```

To access the `Perform()` and the `Undo()` of a sub-Action use the `SetPerformAction()` and the `SetUndoAction()`

```
cleanSponzaAction.SetPerformAction(()=>Spawn("Lesson1/Stage1/Action1/Sparkling"));
```

This example will spawn the Sparkling prefab after performing the cleanSponzaAction sub-Action.

Similarly, we save the prefabs related to this Action in a single folder, in this case "Lesson1/Stage1/Action1"

6.1.6 Optional Action

In this tutorial we will learn how to compose an Optional Action and make our applications more interesting.

Note: Optional Actions can be used as side-tasks, Actions that are not mandatory to be performed.

The scenario

In this example our main Action will be a QuestionAction, asking the user where is Sponza located. At the same time we would like for a second Action to be spawned. This Action involves the insertion of a jar into a predefined place (InsertAction). After inserting the jar we would also like to hammer it into place (ToolAction). Those two Actions would be in a lesson.

Optional Action Implementation

The implementation of the Actions is simple and does no different from implementing a simple Insert and a Tool Actions.

Insert the Jar (Insert Action)

```
using MAGES.ActionPrototypes;

public class InsertJarAction : InsertAction
{
    public override void Initialize()
    {
        SetInsertPrefab("Optional/InsertJar/Action0/JarInteractable", "Optional/
↪InsertJar/Action0/JarFinal");
        SetHoloObject("Optional/InsertJar/Action0/JarHologram");

        base.Initialize();
    }
}
```

Hammer the Jar (Tool Action)

```
using MAGES.ActionPrototypes;

public class HammerJarAction : ToolAction
{
    public override void Initialize()
    {
        SetToolActionPrefab("Optional/InsertJar/Action1/JarHitMallet", MAGES.
↪ToolManager.tool.ToolsEnum.Mallet);
        SetHoloObject("Optional/InsertJar/Action1/JarMalletHologram");

        base.Initialize();
```

(continues on next page)

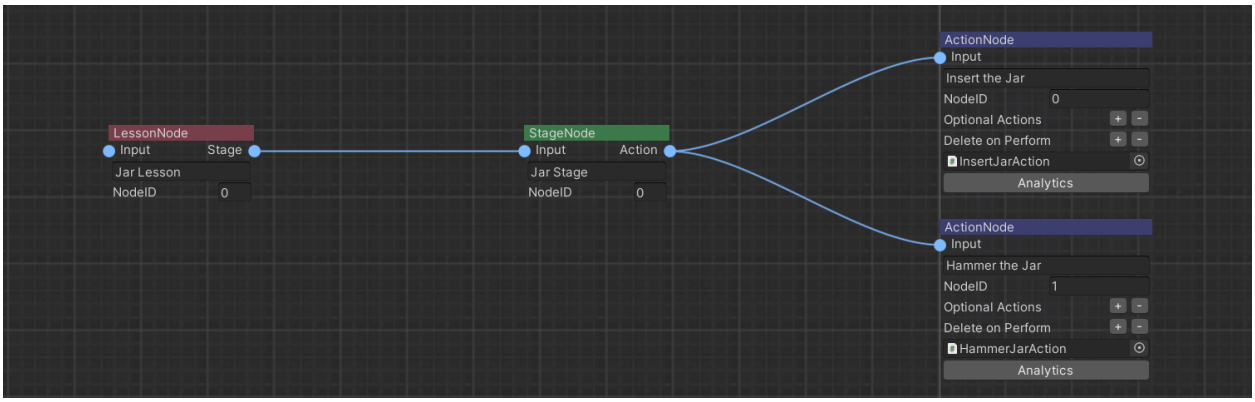
(continued from previous page)

```
        }
}
```

Add Action to scenegraph

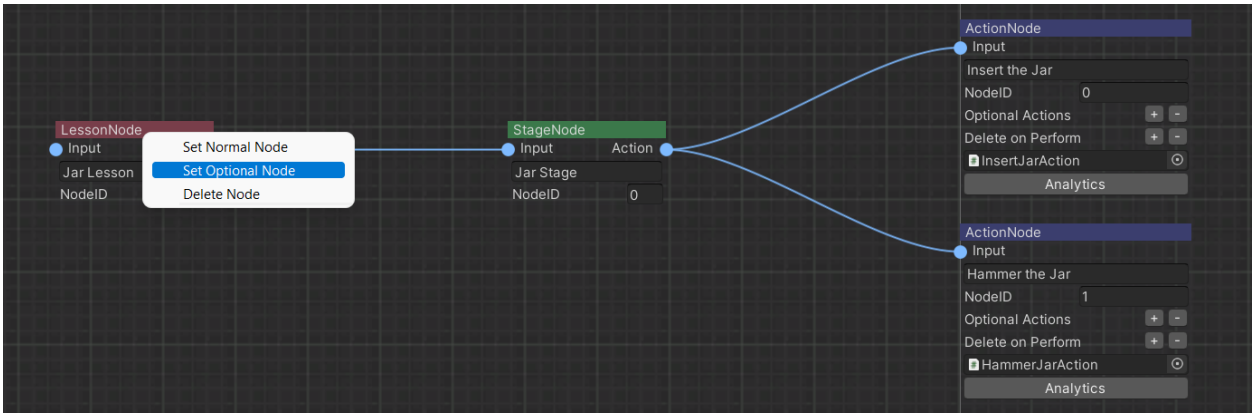
Now that we have the two Actions it is time to add them to scenegraph.

Open the existing scenegraph tree and construct a lesson with a stage containing the above two Actions. You can reconstruct the Optional Lesson as you want, in our case we included both Actions into a single Stage.

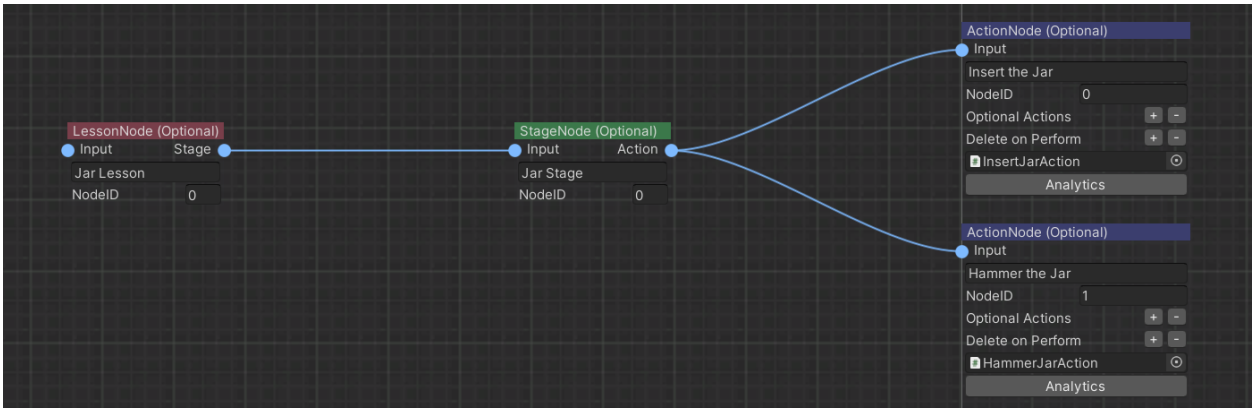


Warning: Pay attention that didn't link the Lesson we created with the operation node. This is because this Lesson is not a part of the scenegraph's main path. It is an optional Lesson.

To make the Lesson Optional, right click on the lesson node and select "Set Optional Node"

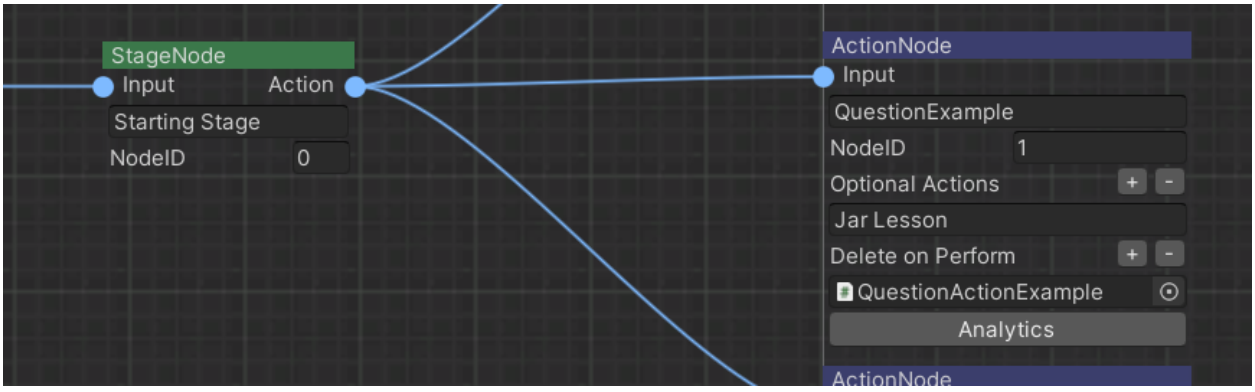


This will make all the nodes contained into the Lesson Optional Nodes. A tag "Optional" is also written into the node labels.



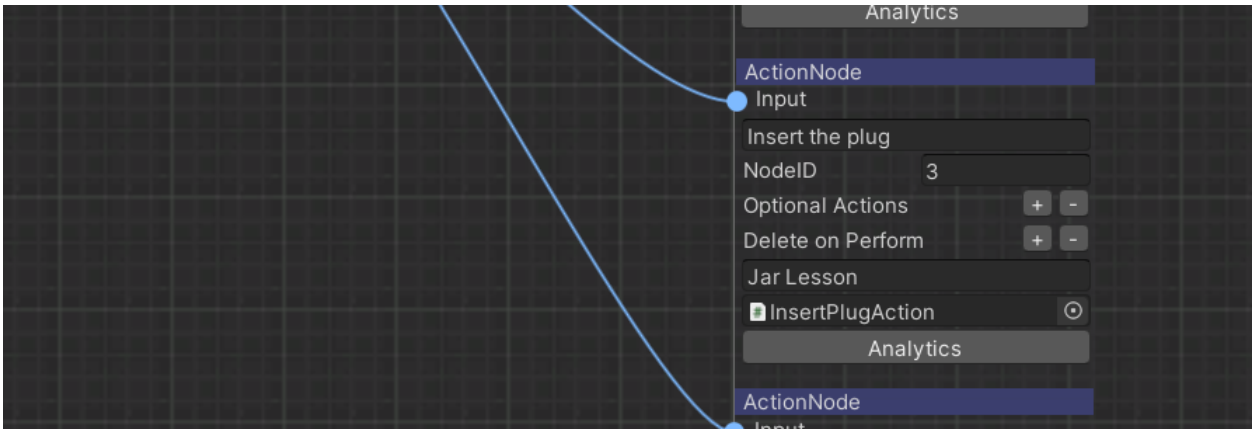
The next step is to set where this Lesson will be spawned. As mentioned before we will spawn it with our Question-Actions as an Optional.

To do that we will go to the **QuestionExample** and add type the **Jar Lesson** at the optional Actions List. You can instantiate more than one Optional Lessons. This will instantiate the first Action of the **Jar Lesson** along with the **QuestionExample** Action



We also want to set when this Optional Lesson will be deleted. In our case we want to delete it after the **Insert the plug** Action.

To do that we will go to the **Insert the plug** and add the **Jar Lesson** in the destroy on perform List. You can delete more than one Optional Lessons. By performing the **Insert the plug** Action we will also delete our Optional Lesson, if the user did not perform it till then.



Note: In this way we configure the lifetime of our Optional Action.

Remember to **save the scenegraph** after this step.

Final result

The gif below shows the Optional Action we implemented. As you can see the InsertJarAction remains active till we decide to perform it.

Scenegraph manipulation with Optional Actions

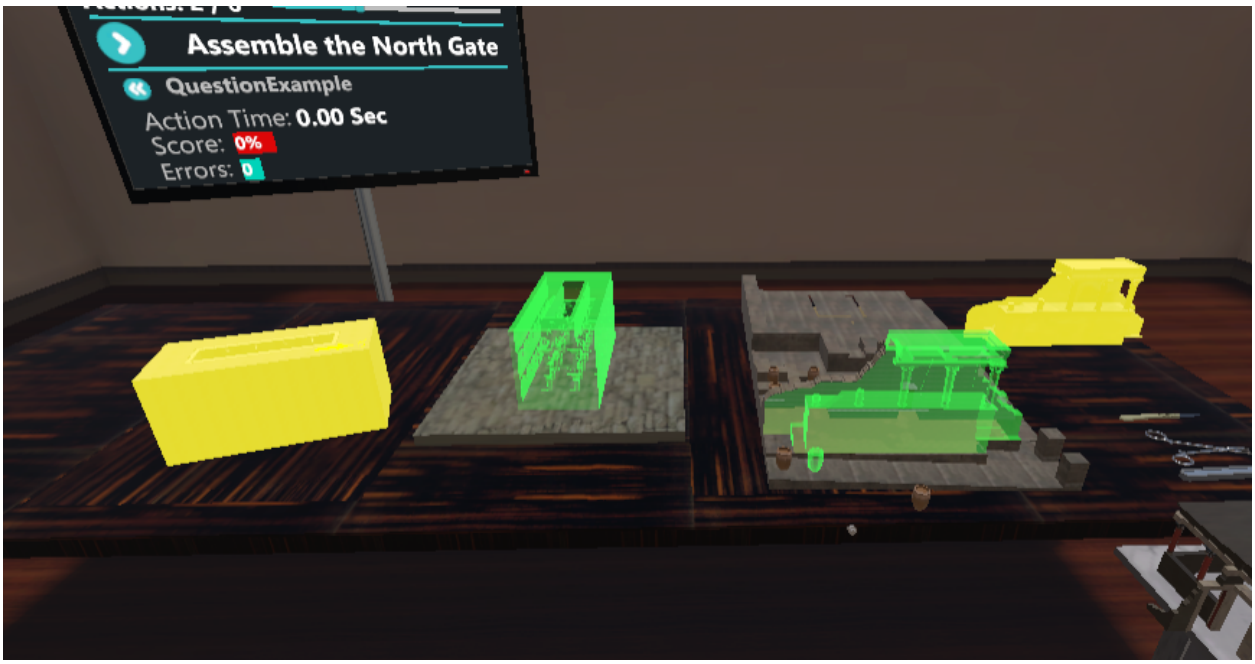
Optional Actions can also be used modify the Scenegraph tree according to the user's decisions.

Note: This functionality can be used in cases that an Action is crucial and by performing it the scenegraph will update itself by adding or removing other Actions. An example can be a completion of a wrong Action by mistake.

In this tutorial we will demonstrate this decision-making functionality.

The scenario

In our case, we would like to generate a decision-making case for our users to decide whether to assemble the Knossos or the Sponza. If the user decides to assemble the Knossos (right), the scenegraph will move to the next Action. However, if he assembles the Sponza (left), we will replace the Knossos Lesson with an Optional one, the Sponza lesson.



Action Scripts

Assemble the Knossos (Insert Action)

```
using MAGES.ActionPrototypes;

public class AssembleKnossosAction : InsertAction
{

    public override void Initialize()
    {
        SetInsertPrefab("Lesson0/Stage1/Action0/FrontPartInteractable", "Lesson0/
↪Stage1/Action0/FrontPartFinal");
        SetHoloObject("Lesson0/Stage1/Action0/Hologram/FrontPartHologram");

        base.Initialize();
    }
}
```

The Knossos Action is just a simple Insert Action.

Assemble the Sponza (InsertAction)

```
using MAGES.ActionPrototypes;
using MAGES.sceneGraphSpace;

public class AssembleSponzaAction : InsertAction
{
    public override void Initialize()
    {
        SetInsertPrefab("Lesson0/Stage1/Action0/SponzaInteractable", "Lesson0/Stage1/
↪Action0/SponzaFinal");
        SetHoloObject("Lesson0/Stage1/Action0/Hologram/HologramSponzaFinal");

        //It is important to add this line if your perform method changes the␣
↪scengergraph runtime
        // (eg uses the ReplaceNoreRuntime)
        GetComponent<ActionProperties>().actionType = ActionType.
↪OptionalConvertedToNormal;

        base.Initialize();
    }

    public override void Perform()
    {
        ScenegraphTraverse.ReplaceNodeRuntime("Knossos Lesson", "Sponza Restoration");

        base.Perform();
    }

}
```

This is the Optional Action, as you see it overrides the Perform method and includes some code. The `ReplaceNodeRuntime("Knossos Lesson", "Sponza Restoration")` function will replace the Knossos Lesson with the Sponza Restoration Lesson (this is an Optional Lesson) if the user Performs this Action.

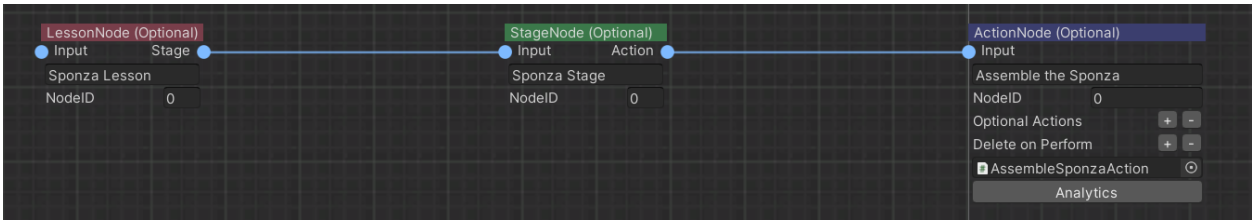
Alternatively, you can use the `AddNodeRuntime("Sponza Restoration")` to add the Sponza Restoration Lesson or the `DeleteNodeRuntime("Knossos Lesson")` that will only delete the Knossos Lesson.

Warning: Make sure to include the line `GetComponent<ActionProperties>().actionType = ActionType.OptionalConvertedToNormal;` in the initialize method if you want to Destroy the main Action and initialize the next one after performing this optional.

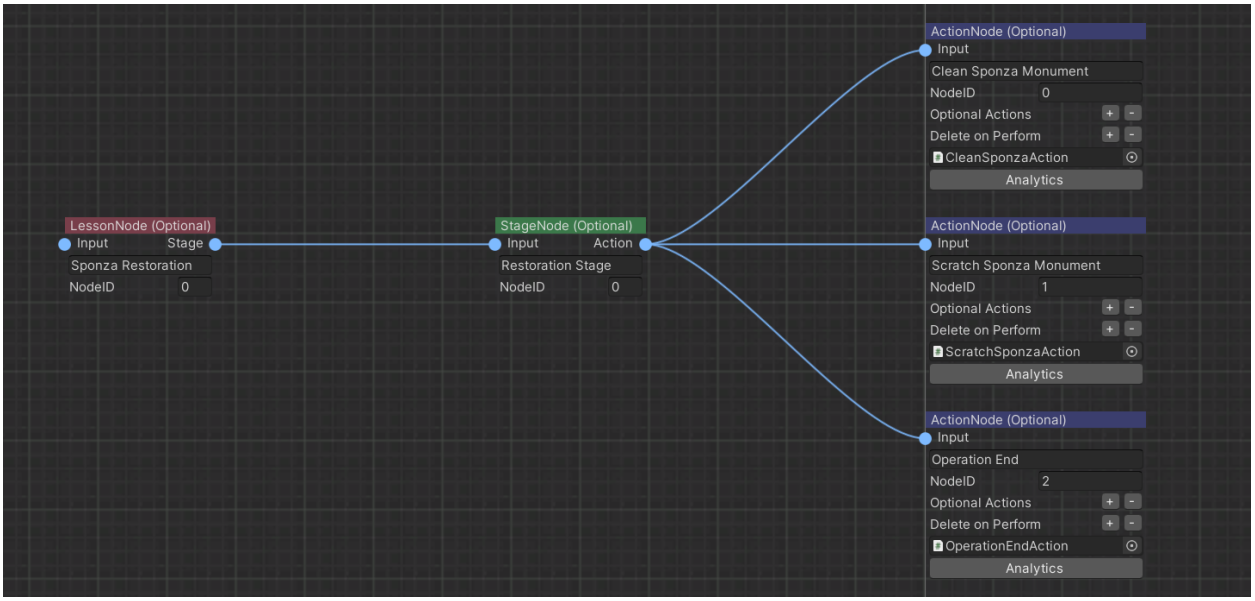
Scenegraph configuration

For this implementation we need to configure two Optional Lessons. The Optional Lesson to insert the Sponza (this will trigger the second one) and the **Sponza Restoration** Lesson that will replace the Knossos one in case the user decides to assemble the Sponza.

Below you see the Assembly of Sponza

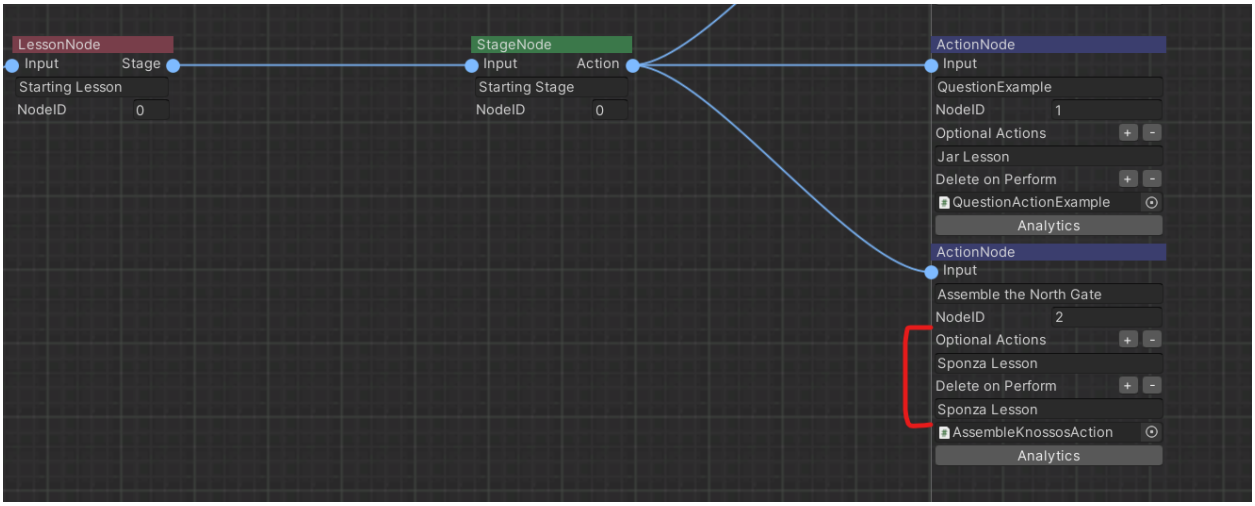


And this is the Sponza Restoration Lesson



Note: Remember to mark those nodes as Optional nodes.

Finally, we have to configure the point where we spawn the Sponza Optional Action. This would be the same Action with the Knossos, to make the user decide what he will assemble.



Pay attention that we will also destroy the Sponza Optional node if it is not performed after this Action.

Final result

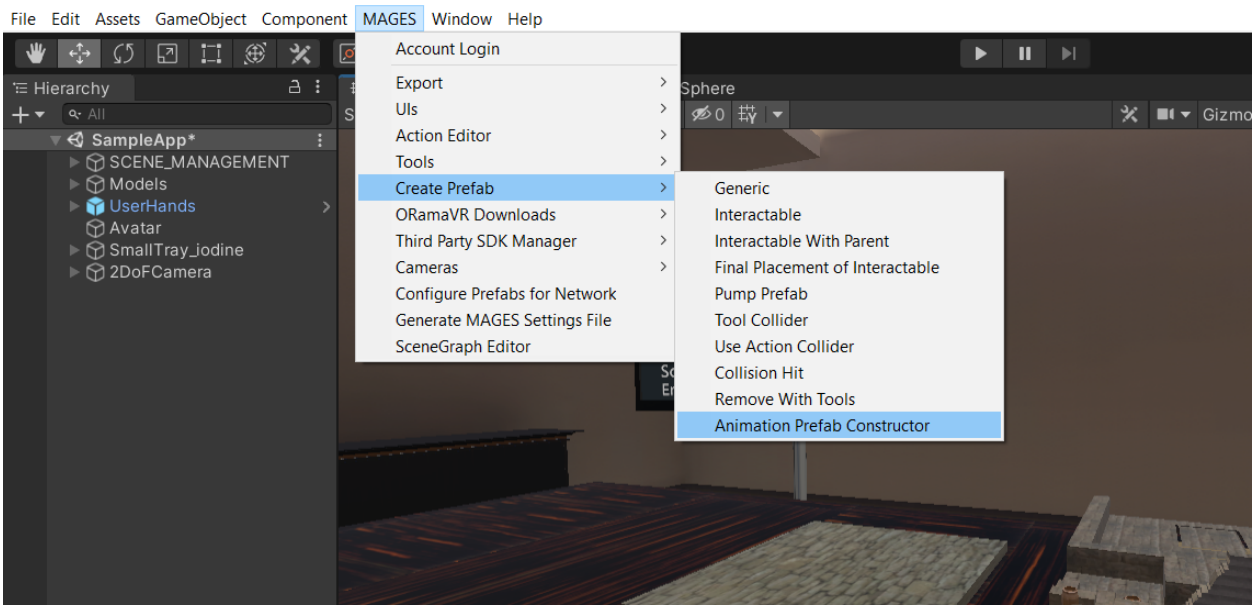
The gif below shows the Optional Action we implemented. As you can see if we perform the Knossos Action we continue with the assembly of Knossos. However, if we return and do the Sponza Action, we proceed with the Sponza restoration Actions (Optional Lesson).

6.1.7 Animation Action

In this tutorial we will learn how to create and use an Animation Action and its components.
To generate an Animation Action you only need the **Animation Move Prefab Constructor**

Animation Prefab Constructor

From the MAGES menu select the option Create Prefab/Animation Prefab Constructor

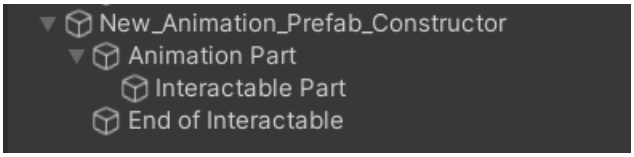


The template gameObject for the animation prefab constructor will appear. It is recommended to use this object for your Animation Action. The Animation Prefab Constructor has 3 child objects;

1. **Animation Part**
2. **Interactable Part** (Child of Animation Part)
3. **End of Interactable**

- The Animation Part is used to map the animation with the user's hand movement.
- The Interactable Part is used to interact with the object
- The End of Interactable is used to set the finishing point of the animation. When the Animation Part has the same position as the End of Interactable, then the Animation Action will be performed. We will get on that later on the tutorial.

The image below shows the gameobject that were referenced above.



So, let us begin creating our first animation action prefab.

Configuration of the interactable part

The configuration of the interactable part is pretty easy and straightforward. We will edit two scripts: - The **Ovid VR Interactable Item** - The **Configure Child Movement**

In the **Ovid VR Interactable Item** we simply drag and drop the Interactable Part gameObject from the scene to the **Rigidbody** field of the script.
This will give access to the gameObject's rigidbody.

In the **Configure Child Movement** , drag and drop the Interactable Part gameobject form the scene to the **Interactable Item** field of the script.

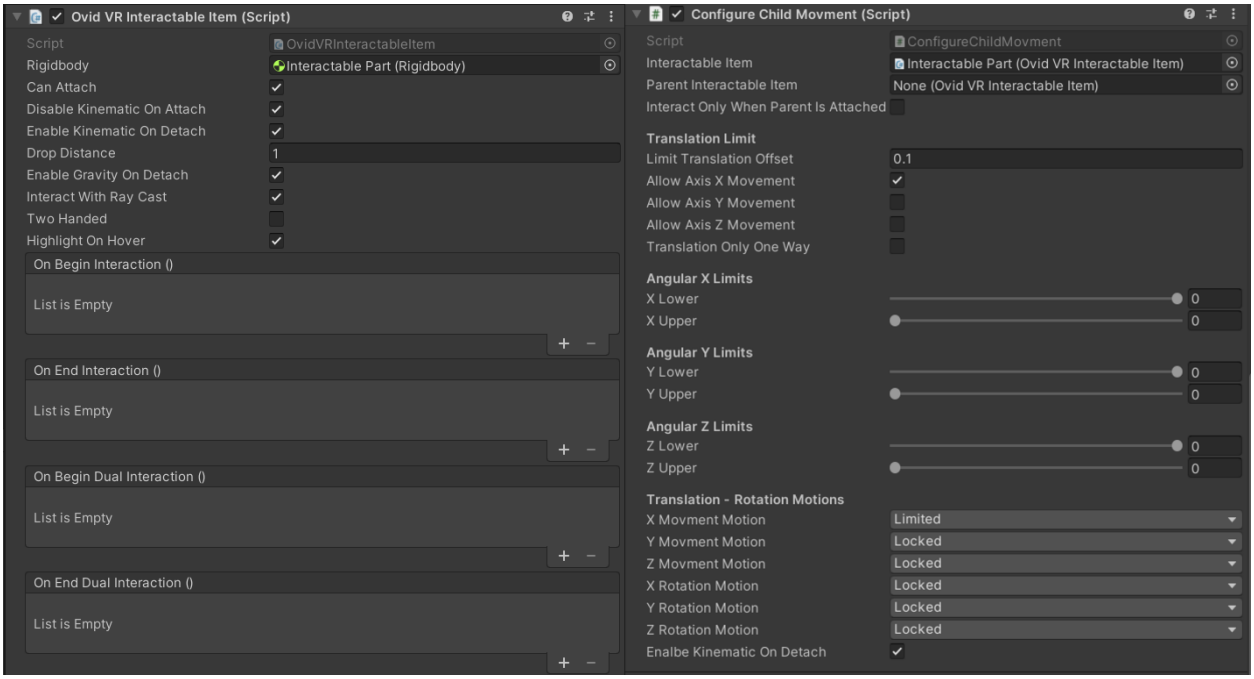
Then, we will change the **Translation Limit** and the **Translation - Rotation Motions**

In the **Translation Limit** we set the acceptable movements in the XYZ axis

In the **Translation - Rotation Motions** we set the acceptable rotation motions in the XYZ axis.

In our example, we will Allow the X Movement, and we will set the everything except the **X Movement Motion** to **Locked**

Below, you can see how we configured both the **OvidVRInteractableItem** script and the **ConfigureChildMovement** script



Configuration of the gameobject mesh

First and foremost, we have to find our model prefab that we want to animate. In our case, we will use the "B_J_Base animation" prefab. Our prefab has already an **animator** component embedded with a controller named **B_J_Base**. The model of the prefab is an auxiliary plug model. Set the prefab **as a child** of the "Animation Part" GameObject.

Your last step of model configuration is creating a **box collider** as a child gameobject of the Interactable Part. The box collider should encapsulate all the plug mesh.

Lastly, we should also move the **End of Interactable** gameobject to the final location of the animation. In our case the last point of the animation is when x = 0.0627.

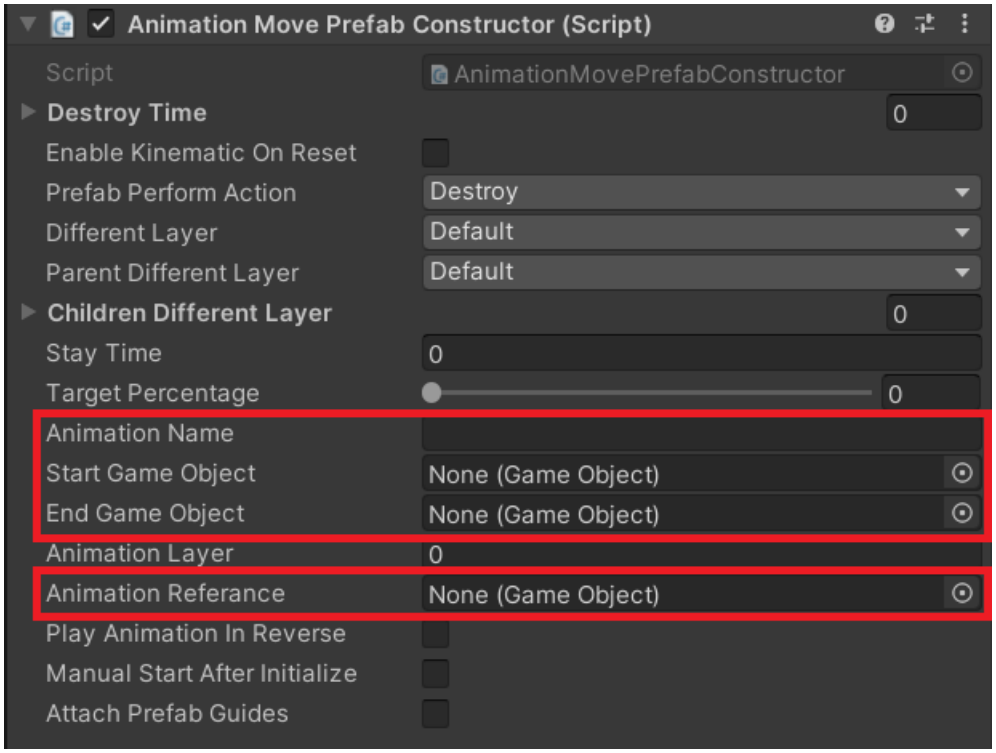
Now, let's continue our tutorial by editing the Animation Move Prefab Constructor.

Animation Move Prefab Constructor

Here we will showcase how to use the Animation Move Prefab Constructor in the Animation Prefab Constructor.

The important fields of the Animation Move Prefab Constructor are:
1) Animation Name with a type of string
2) Start Game Object with a type of GameObject
3) End Game Object with a type of GameObject
4) Animation Reference with a type of GameObject

You can see the above fields in the picture below.



The above fields will be used in this tutorial to create a simple animation action! (For the explanation of each field please click here.)

Now let's fill the above fields:

- For the Animation Name we will put **PlugInAnimation**. It is a simple animation in the X axis.
- After that, in the "Start Game Object" we put the **Interactable Part**.
- In the "End Game Object" we put the **End of Interactable**.
- For the "Animation Reference" we put the **B_J_Base** gameobject.
- Lastly, for the "Target Percentage", let's set it to 0.922.

Action Script

Click here for a detailed tutorial on how to create an animation script

Results

Below you can see the Animation Action.

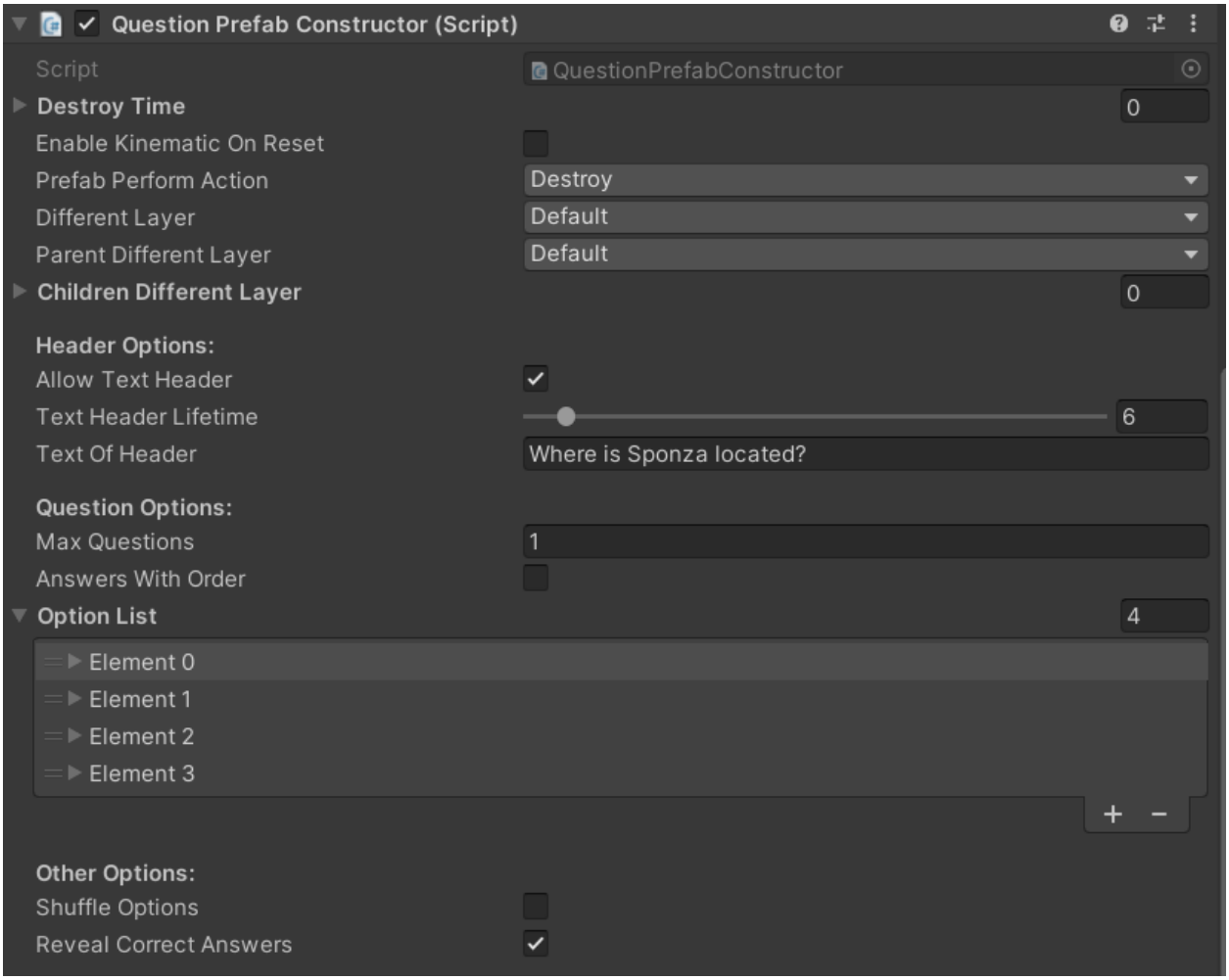
6.1.8 Question Action

In this tutorial we will learn how to create a Question Action
We will modify an already existing prefab named **QuestionPrefabExample** and create a new question action.

Question Prefab Constructor

In order to create a Question Action, the most important part is to set the **Question Prefab Constructor** correctly.

First and foremost, we will spawn our Question Prefab example. To do so, we will drag and drop it either on the hierarchy or in the scene. After spawning the prefab, click on it. In the inspector tab you will be able to see the **Question Prefab Constructor**.



In the above photo we can see an example of a ready-to-use Question Action. We can see that the script has a lot of fields. The main distinguish in the script is between the different option types. There are three option types:

- Header Options
- Question Options
- Other Options

Now, we will break up and analyze its option:

i) **Header Options:** This groups the functionalities of the header UI. It has 3 fields:

- **Allow Text Header:** If true, spawns first the header UI and then the options.
- **Text Header Lifetime:** For how many seconds the header UI will be visible.
- **Text Of Header:** The text of the header UI

ii) **Question Options:** This groups the functionalities of the Options UI. It has 3 fields:

- **Max Questions:** The total amount of questions.
- **Answers With Order:** If true, then there will be a choice order.
- **Option List:** List of the option. There you can set which option is the correct one, the order of it and the text of the option.

iii) **Other Options** This groups some functionalities of the UI. It has two fields:

- **Shuffle Options:** Spawns the options randomly each time
- **Reveal Correct Answers:** After answering the question, the system will reveal the correct answer.

So after learning the basic fields of the Question Prefab Constructor, it is time to change the fields of it and put our own question and options!
First, let's say that we want to ask *"What's 1+1?"* and have 4 Options, 2,-2,1 and 11. We will replace the **Text of Header** to *"What's 1+1?"*
After that, we will go to the **Option list**, delete the previous ones and add 4 new Options. In each Option Text we put the numbers we said above with the correct one being the 2. You can see the gif below on how to set up all the above:

Now, we just save our new prefab and copy its' path and paste it in our Question Action script.

Action Script

Click here for a detailed tutorial on how to create a question script

Results

Below you can see our new prefab while playing.

6.1.9 Cut Action

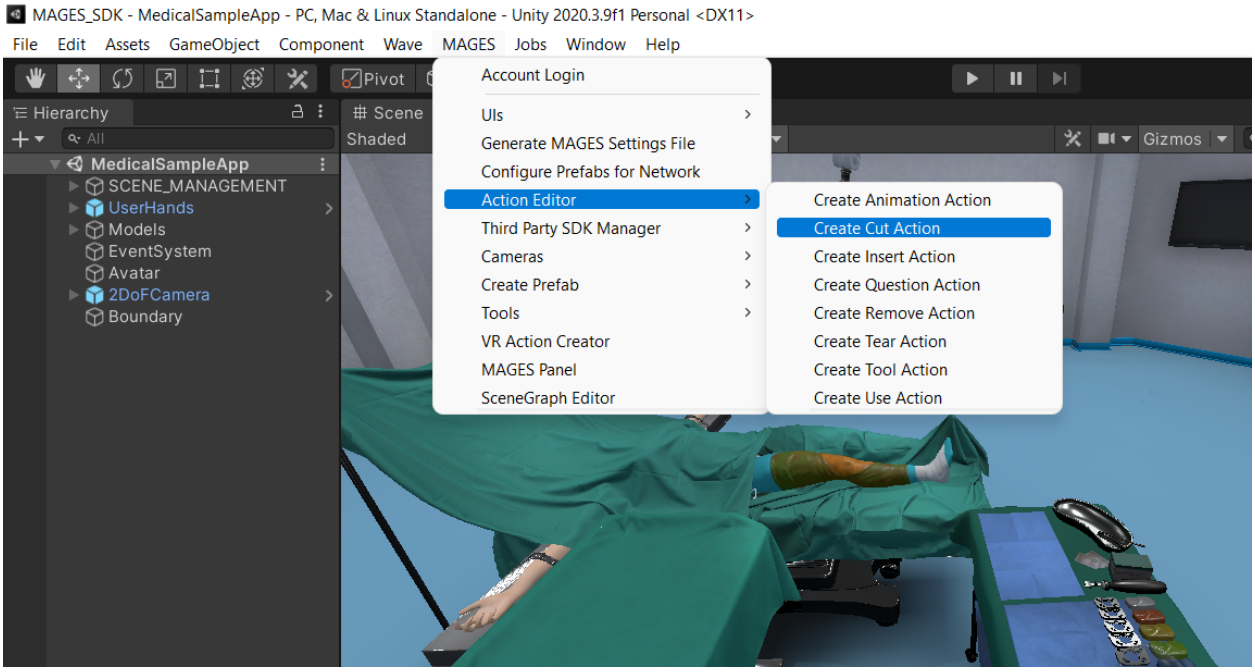
To generate a Cut Action you need the following three prefabs:

1. The Cut Constructor Prefab
2. At least one Cuttable Object with the Cuttable Mesh Script attached to them
3. At least one Cutting Tool/Object with the CuttingBlade or CuttingScissors Script attached to them

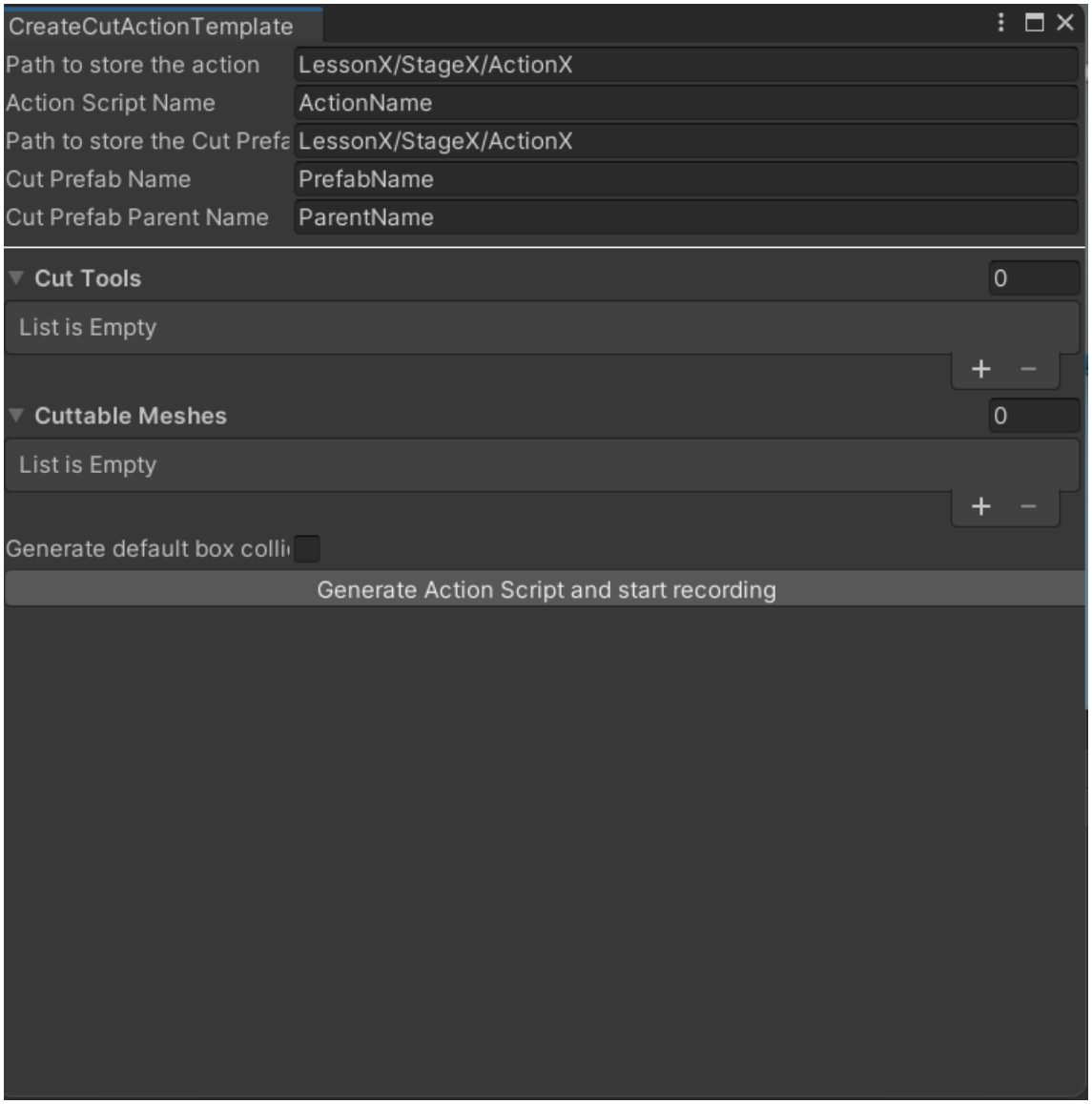
> **Warning:** The Cutting Tools need either the CuttingBlade or Scissors attached to them and not the Cutter or else the recording and the action won't be performed.

Cut Action Generation

From the MAGES menu select the option Action Editor/Create Cut Action



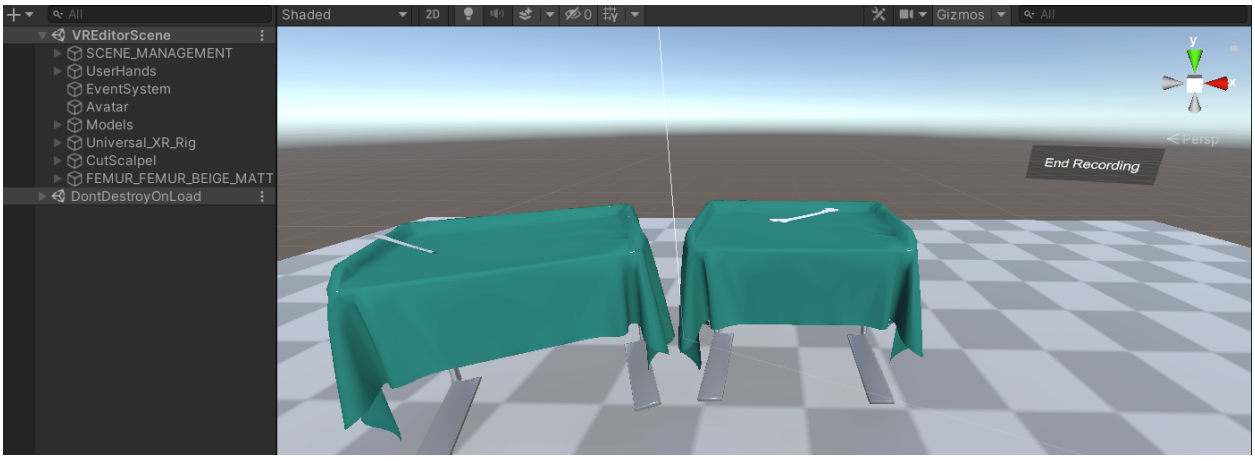
Then an editor window will appear, with the following fields:


CreateCutActionTemplate
Path to store the action
LessonX/StageX/ActionX
Action Script Name
ActionName
Path to store the Cut Prefa
LessonX/StageX/ActionX
Cut Prefab Name
PrefabName
Cut Prefab Parent Name
ParentName
Cut Tools
0
List is Empty
Cuttable Meshes
0
List is Empty
Generate default box colli
Generate Action Script and start recording

Variable Name	Type	Description
Path to Store the Action	String	The path that the Cut Action Script will be stored. The path starts from "Assets/MAGES/Operation/ActionScripts/"
Action Script Name	String	The name of the .cs file/The class name
Path to store the Cut Prefab	String	The path that the generated prefab with the Cut Prefab Constructor will be stored. The path starts from "Assets/MAGES/Operation/Resources/LessonPrefabs/"
Cut Prefab Name	String	The name of the generated prefab
Cut Prefab Parent Name	String	The name of the generated prefab's parent(the name of the cuttable object)
Cut Tools	List of GameObjects/Prefabs	The prefabs that will be used to cut the Cuttable Object
Cuttable Meshes	List of GameObjects/Prefabs	The Cuttable Objects
Generate default box colliders	Boolean	If true, it generates a trigger box collider to the Cuttable prefab

After having set these fields, you are ready to click on the 'Generate Action Script and start recording' button.

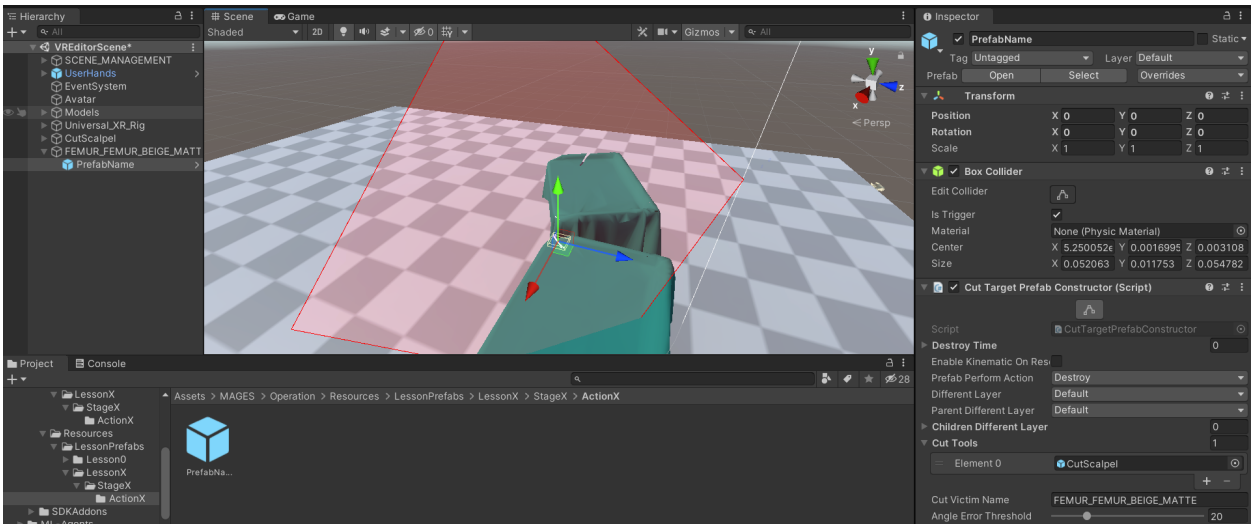
When clicked, you will be redirected in the recording scene, in play mode.



As you can see, the Cut Tool and The Cuttable Object are spawned on the two tables and there's an End Recording Button. So in order to complete the recording, we grab the tool and perform the cut and press the button.

After finishing the recording, if we navigate to the paths we added in the editor window, we can see the exported Action script and the exported Cut Prefab with its fields already setup with the information from the recording.


```
public class ActionName : CutAction
{
    public override void Initialize()
    {
        SetCutPrefabs("LessonX/StageX/ActionX/PrefabName", "ParentName");


        base.Initialize();
    }
}
```



However, you can edit both script and prefab to match your needs, maybe by adding more cut tools to the prefab and spawning them through the action script, or editing the plane.

Custom Cutting Tool Creation

Let's see how to create a custom cutting tool, using the Cutting Scalpel as an example.

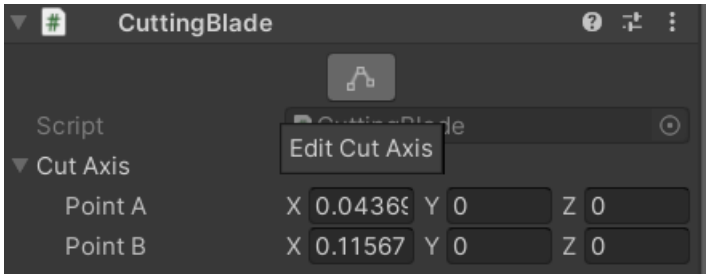
The scalpel's root is an empty GameObject which consists of two child GameObjects that represent the handle and the blade.

The parent GameObject in order for the Cutting tool to work correctly, needs the following Components:

1. The Mages Interactable Item Script, which allows you to grab the tool
2. A rigidbody
3. A handposer, so you can define the pose that the tool will be held (Optional)
4. The Cutting Blade script, which defines the area which the tool is going to use to cut objects

Here's how you define the area in the Cutting Blade script:

1. Drop the tool into a scene and then go to the Cutting Blade script that's attached to your tool and click the edit button

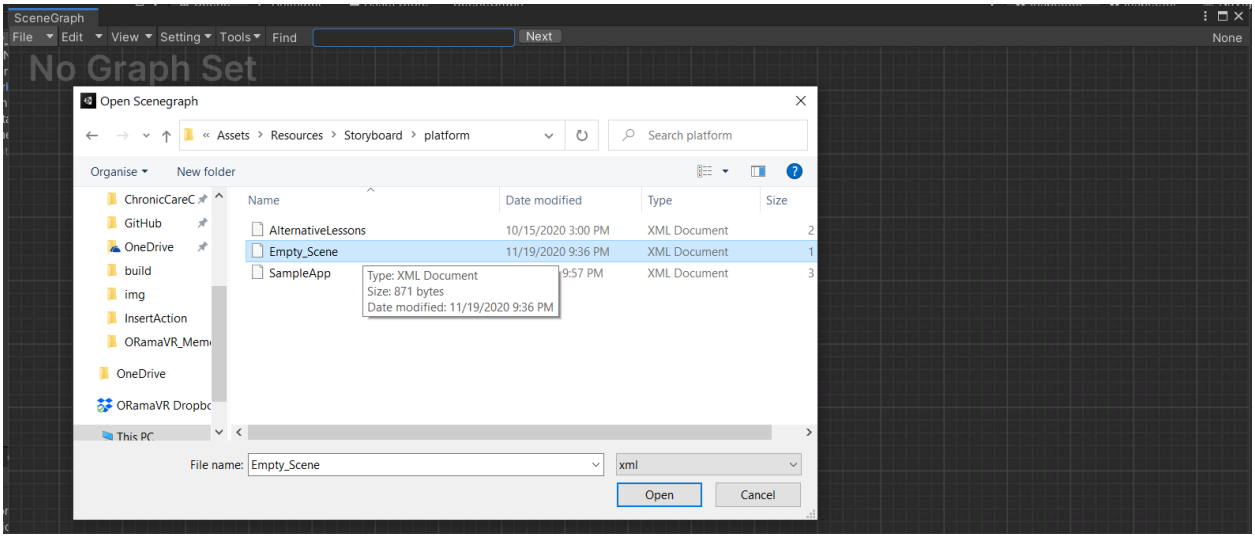


2. Then, with Gizmos enabled, you can position the two points(Point A, Point B) to define your cutting area

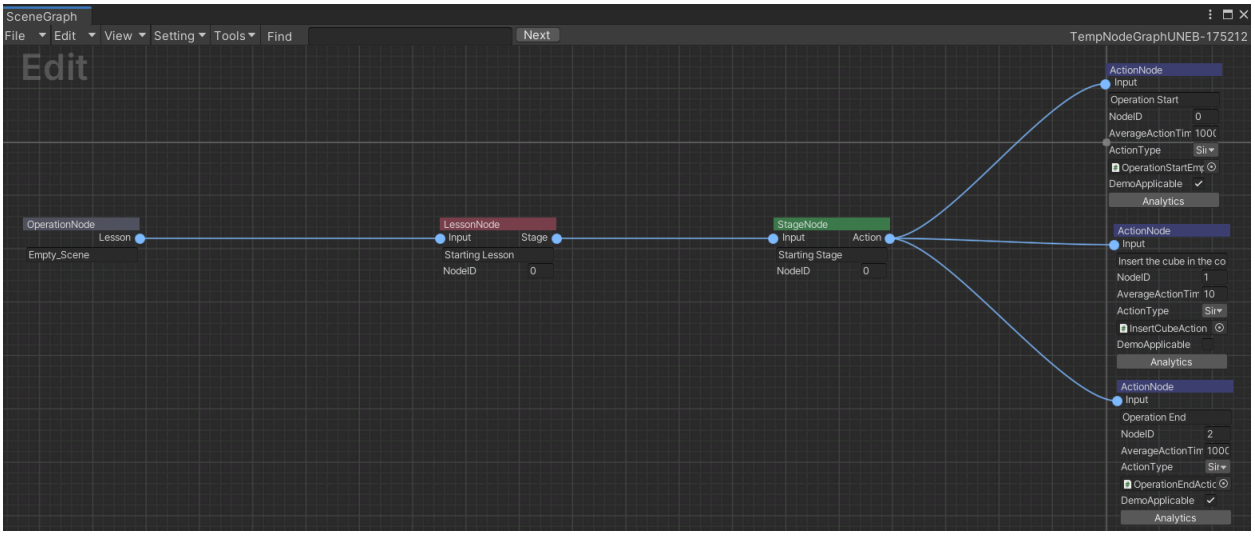
For the children GameObjects, you just need a non-trigger collider for the handle and a trigger collider for the blade.

Add the Action to Scenegraph

The final step is to link the ActionScript to the scenegraph. From the MAGES menu click Scenegraph Editor. At the Scenegraph Editor tab, click File/Load and the proper .xml import the scenegraph. In this case is the Empty_Scene.xml.



To add a new Action Node right click inside the Scenegraph Editor and select "Action Node". Fill the Action description, along with the proper NodeID (in this case is the second Action) Finally, add the reference of the Action script.



From the Scenegraph Editor menu, click File/Save to save your changes.

6.2 Action Analytics

In this tutorial we will demonstrate the different types of our analytics errors/warnings and how to properly configure them for your Actions.

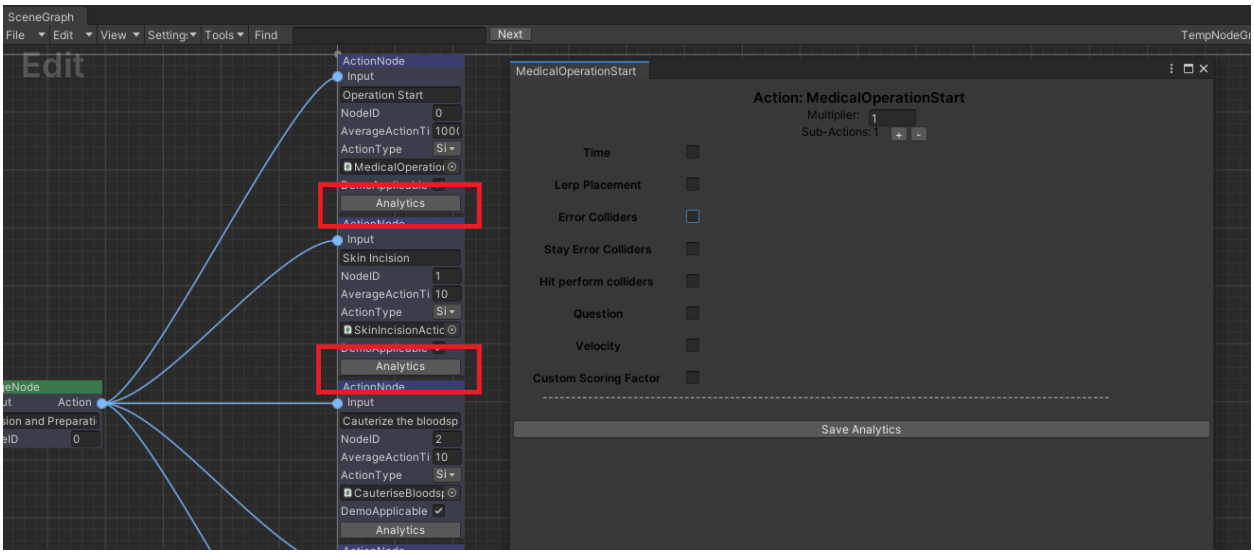
In MAGES SDK we provide a number of standard **scoring factors** to enhance your Analytics.

Note: **Scoring factors** refer to specific actionable behaviors which are important to consider for each Action, and they reflect the user scoring e.g. correct orientation, wrong collision, max velocity etc.

6.2.1 How to add scoring factors to your Action

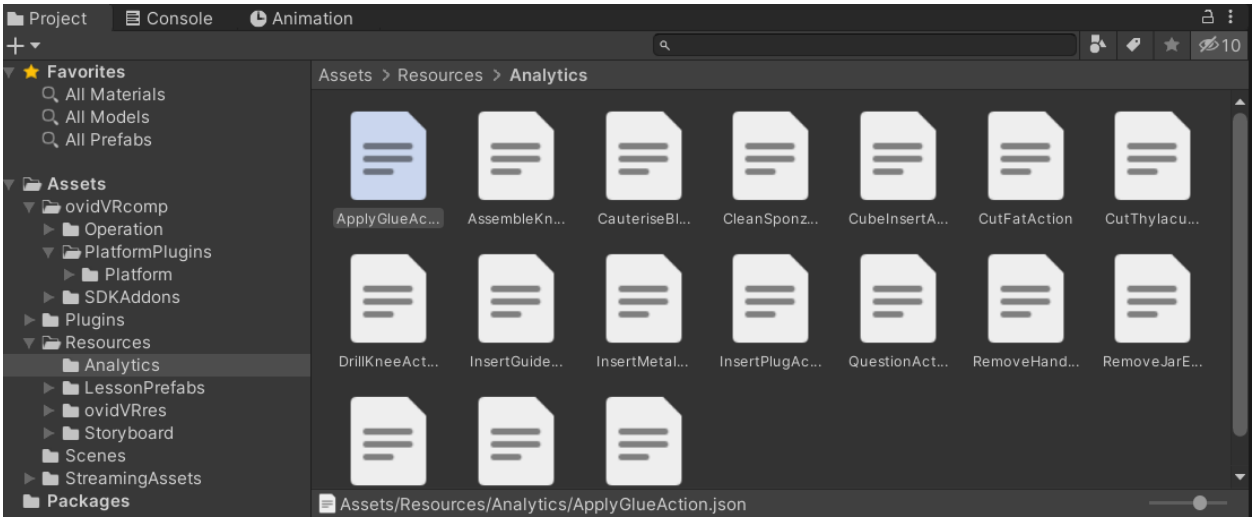
Open the scenegraph editor from **MAGES > Scenegraph Editor** and load your scenegraph.

Each Action node has an **Analytics** button, click it to open the analytics editor.



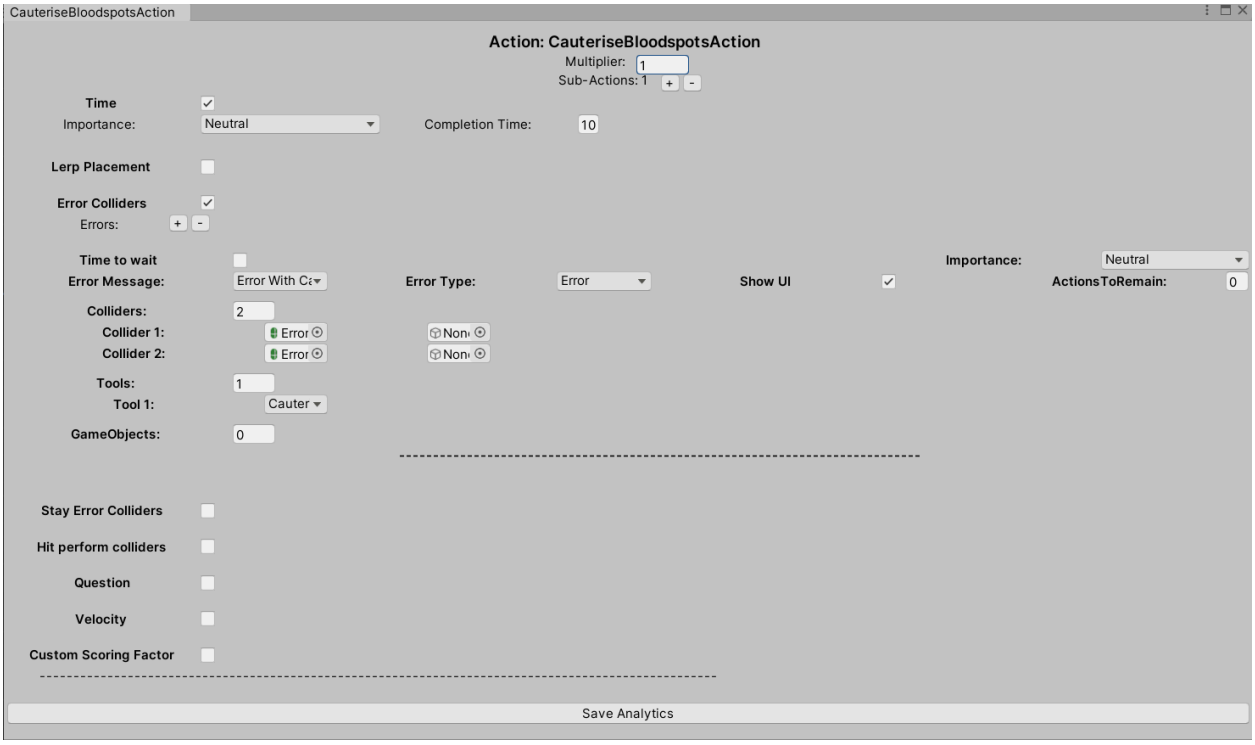
This window contains the scoring factors. To enable a new scoring factor click the corresponding checkbox. To save your changes click the **Save Analytic** button. This will generate the analytics file for your Action.

Below you can see the analytic files of our Actions. They are saved in **Assets/Resources/Analytics/**.



6.2.2 The Analytics Editor

This is an example of the analytics editor.



Note: The scoring system is calculated in 100. The maximum score of an Action is 100 and the minimum 0.

Multiplier: At the top of the window you can see a multiplier value. This can be used in case you want to multiply with a given number the score of the Action.

Sub-Actions: To add a second sub-Action in combined Actions use the "+" and "-" buttons.

Importance: This value identifies the weight of the scoring factor.

1. **VeryLittle:** 15%
2. **Little:** 30%
3. **Neutral:** 50%
4. **Big:** 80%
5. **VeryBig:** 100%

If our Action has only a **Little** scoring factor then its maximum score will be 30/100.

If we configure a **Neutral** and a **VeryLittle** scoring factor within the same Action, the maximum score will be 65/100.

Note: The score is capped at 100.

If our scoring factors overpass 100 e.g. three **Neutral** scoring factors it will be capped at 100.

Error Type: We support three different types of errors with different popup UIs for each case:

1. **Warning**

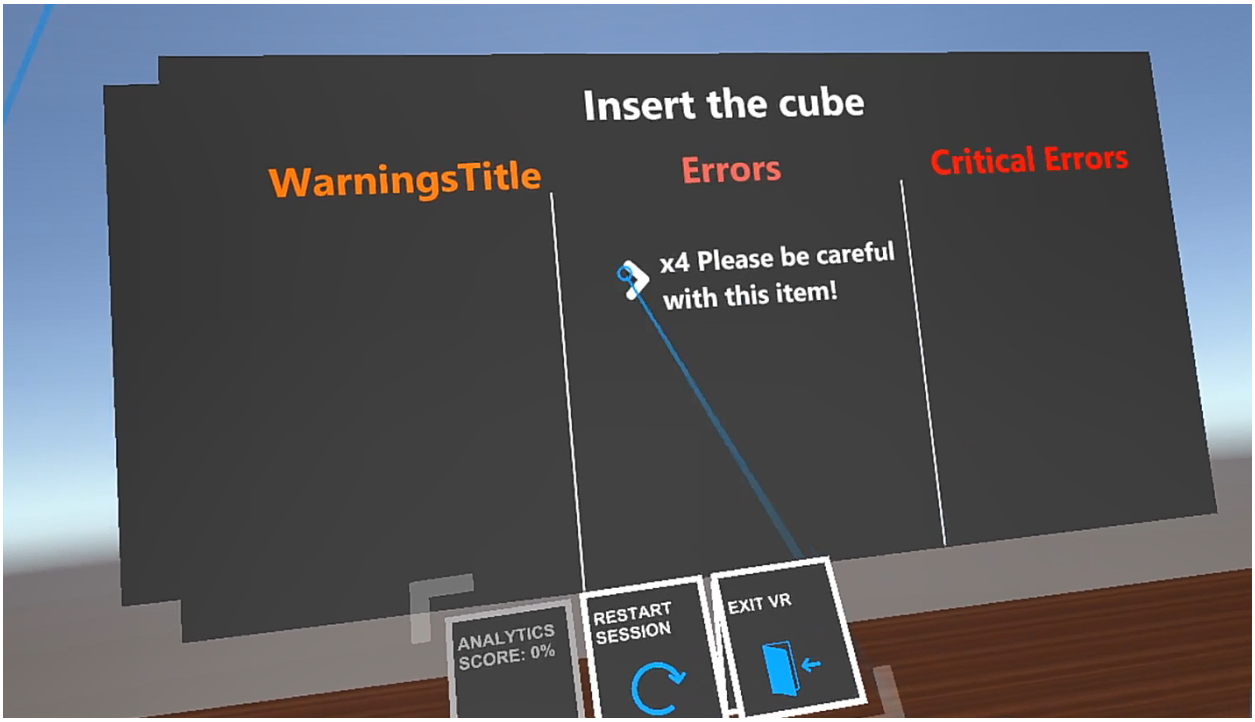
2. **Error**
3. **CriticalError**

Error Message: From the dropdown menu add the corresponding UI key containing text to inform the user.

Show UI: Boolean value to toggle the error message.

Example of an error message:

Be aware that errors, warnings and critical errors are also presented in the analytics overview at the end of the operation:



Below we present the standard scoring factors with examples on how to use them to rate your users.

Scoring factors:

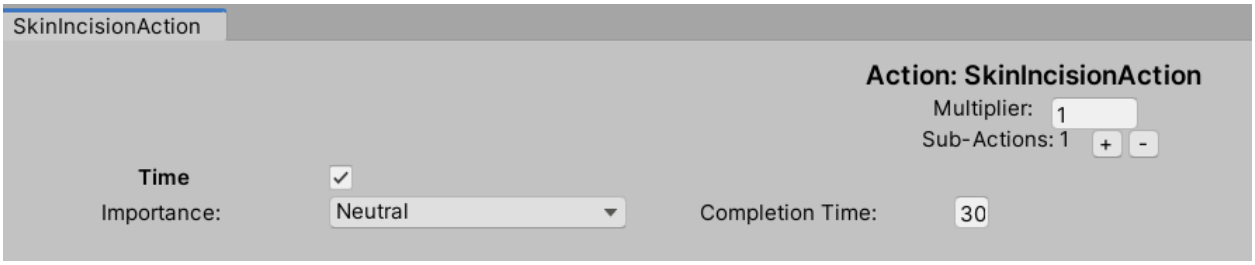
1. *Time*
2. *Lerp Placement*
3. *Error Colliders*
4. *Stay Error Colliders*
5. *Hit Perform Colliders*
6. *Question*
7. *Velocity*
8. *Custom Scoring Factor*

6.2.3 Time

Usage: Give points according to the completion time. To achieve the highest score user needs to complete the Action in fewer seconds that the **Completion Time**. Passing this time-limit results in points loss (10 points per second).

Example: In this example we give user 30 seconds to complete the Action,

Here is the analytics editor for this Action:



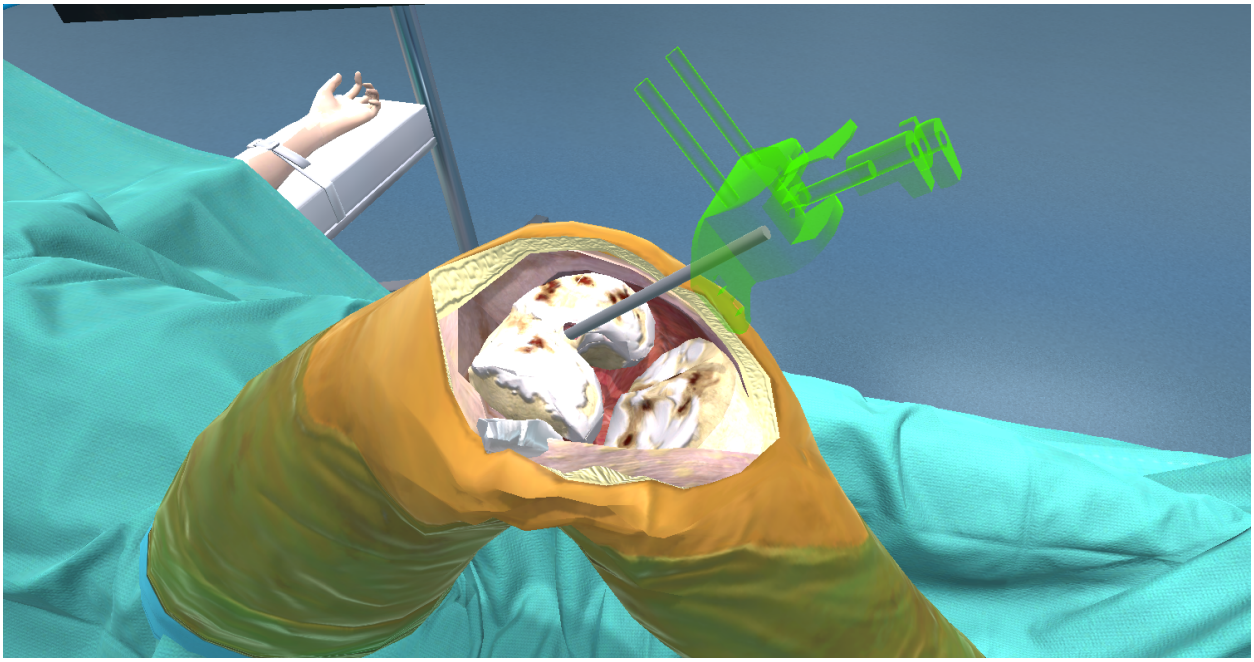
- We set the **Completion Time** to 30 seconds.
- We also set the **Importance** to **Neutral** meaning that this scoring factor will give 50/100 points to the user. If this is the only scoring factor the user can achieve the highest score of 50/100.

6.2.4 Lerp Placement

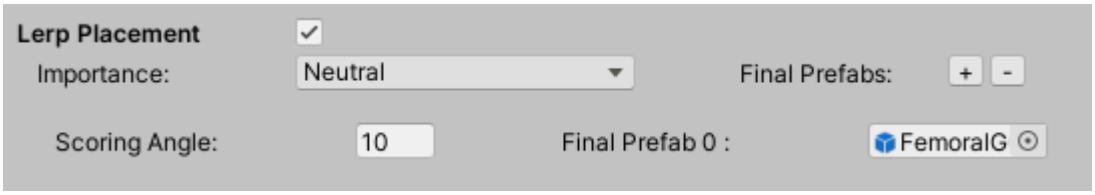
Usage: Track the correct orientation when inserting an object (InsertAction).

Example: In this example, we configure a 10 angles tolerance when inserting the alignment guide to its final position.

Here is the holographic alignment guide:

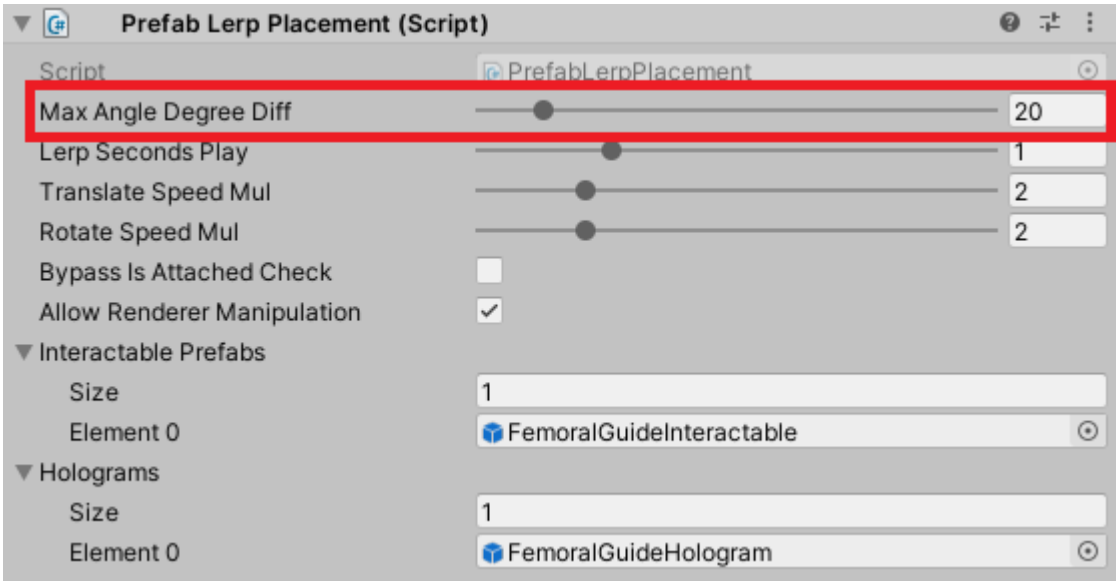


Below you can see the analytics editor for this Action:



- We drag and drop the final prefab into the **Final Prefab 0** input field
- **Importance** to **Neutral**. This factor is valued 50/100.
- We set the **Scoring Angle** to **10** degrees

Below you can see the `PrefabLerpPlacement` component, attached on our final prefab. The max angle difference value sets the maximum number of degrees for this insertion.



Note: User will be awarded with full score when inserting the object with angle less than: **max angle difference value (angle from the final placement) - Scoring Angle**.

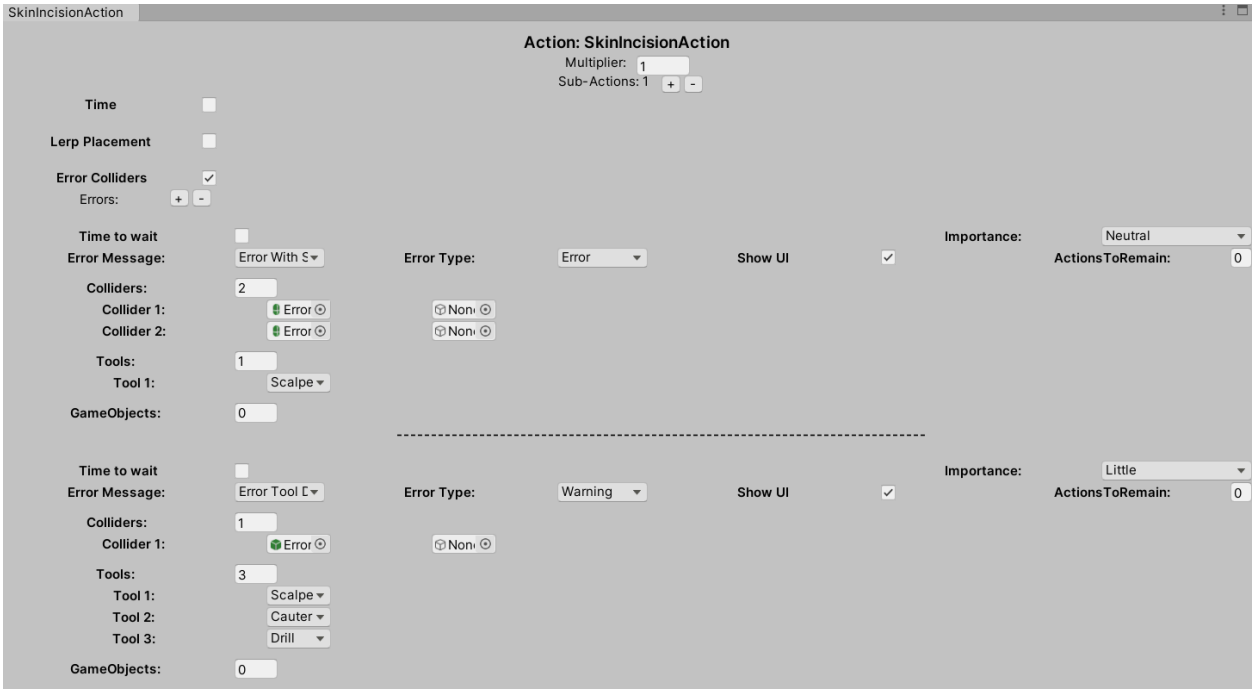
In our example 20-10 = 10 degrees. With up to 10 degrees difference, the user will get full score.

6.2.5 Error Colliders

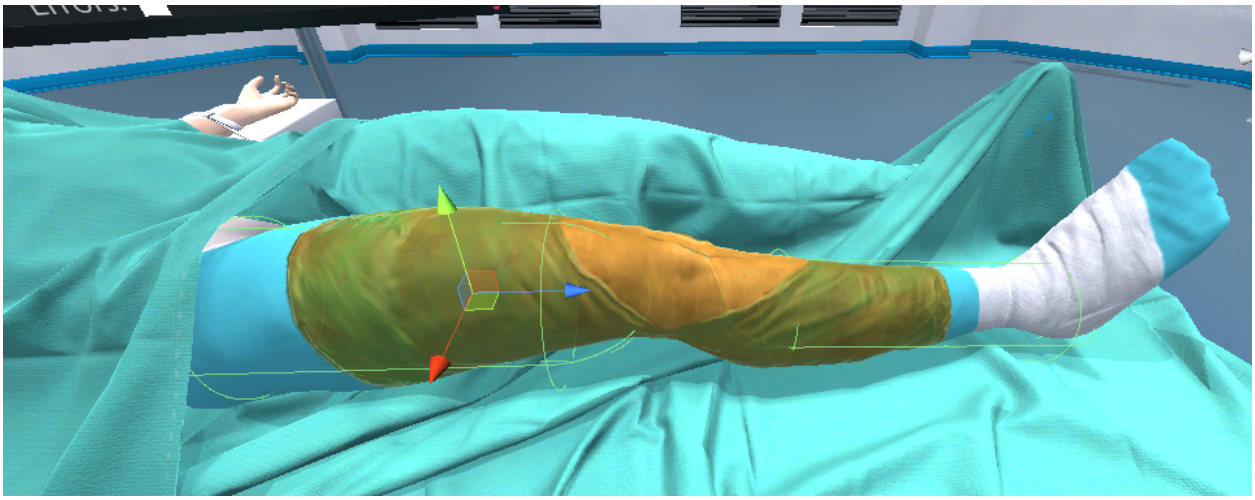
Usage: Track if an object is in contact with a collider.

Example: In this example we set two error colliders to track if the user drops a tool on the floor, and the other one if the scalpel cuts the skin in wrong position.

Below you can see the analytics editor for this Action:

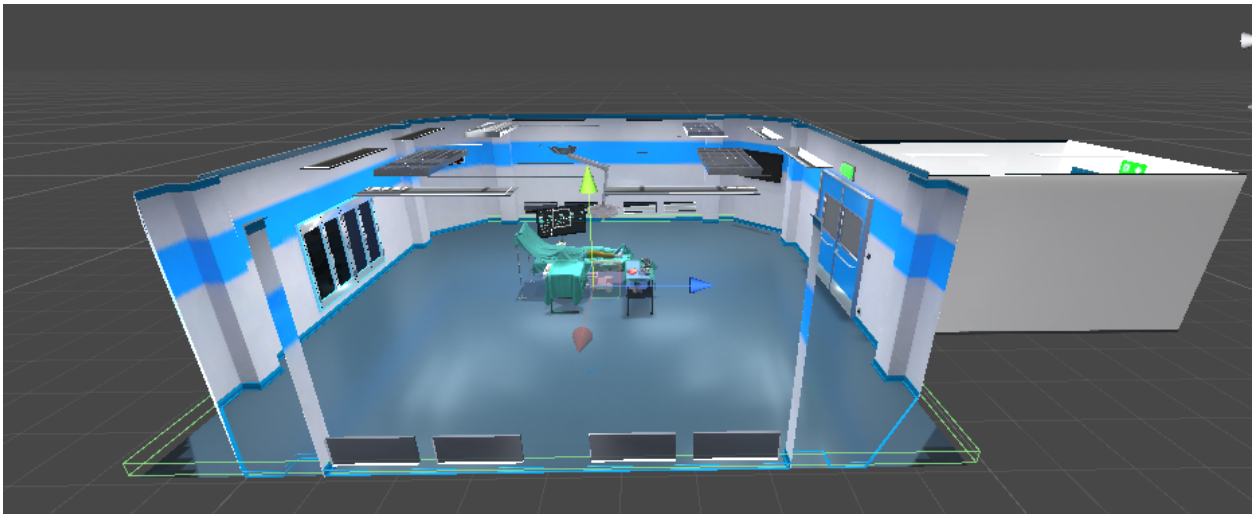


1. The first error collider item contains two error colliders: one for the femur and one for the tibia. You can see them below:



- We drag and drop both error colliders to the **Collider 1** and **Collider 2** input fields. This reference will spawn both colliders
- We select the Scalpel from the corresponding **Tools** dropdown
- We set the **Error Type** as an **Error**
- We set the **Error Message** to our custom message key from the dropdown
- **ShowUI** is enabled

2. The second error collider item an error collider for the floor.



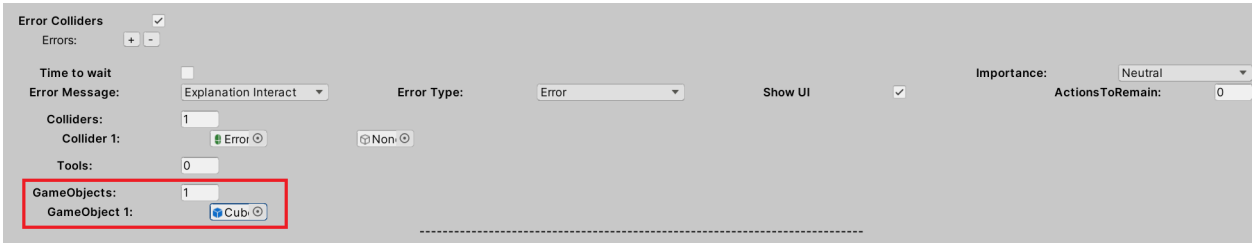
- We drag and drop the error collider to the **Collider 1** input field to spawn the collider.
- We set the **Error Type** as a **Warning**
- We set the **Error Message** to our custom message key from the dropdown
- **ShowUI** is enabled
- We select all the available tools from the corresponding **Tools** dropdown

---

Note: Check the **Time to wait** option and insert the time in seconds the object should stay in contact with the collider to trigger the error.

---

3. Alternatively you can use another object (not tool) to interact with the collider. In this case we are using a simple cube. If the cube interacts with the floor, it will trigger the error. You can use this if you want to avoid dropping or touching an object with another one or an area.



6.2.6 Stay Error Colliders

Usage: Track if an object is **not** in contact with a collider

Example: In this example we set an error collider to track if the user exposes the cautery to contamination. If the cautery exits the collider, the user will lose points.

You can see the error collider here (reflecting the safe area):

Here is the analytics editor for our Action:

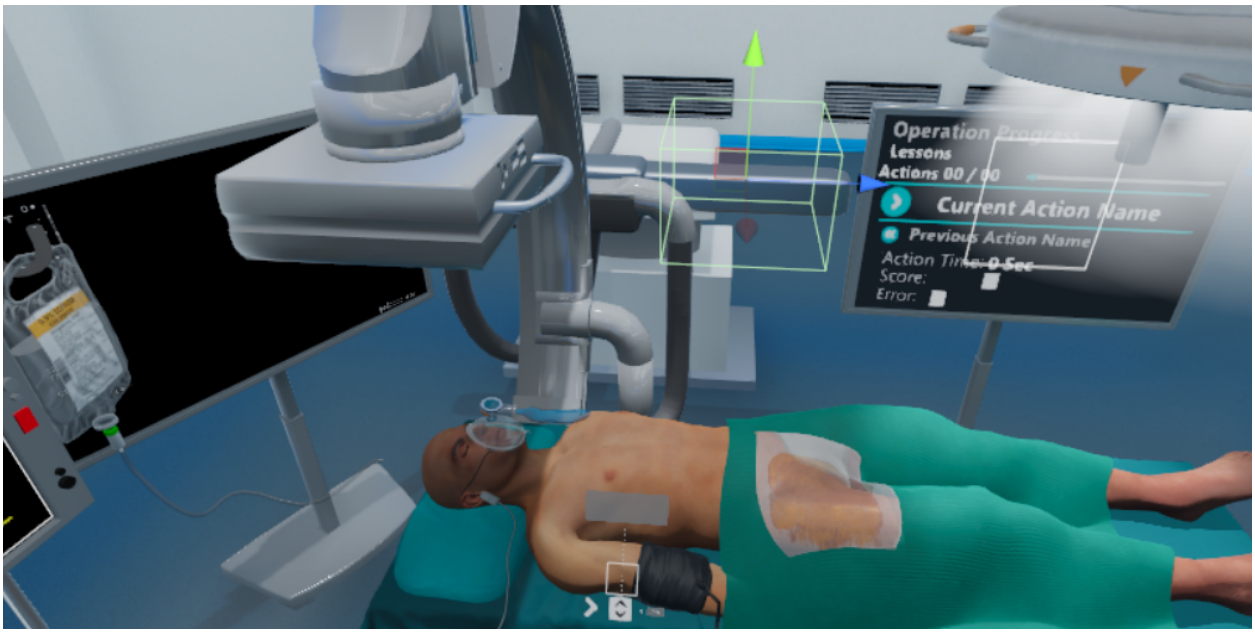
- We drag and drop the collider representing the safe area into **Collider 1**. This will spawn the safe area collider.
- We select the Cautery from the **Tools** dropdown
- We set the **Error Type** as an **Error**
- **Importance** to **Neutral**. This factor is valued 50/100.
- We set the **Error Stay Message** to our custom message key from the dropdown
- **ShowUI** is enabled

6.2.7 Hit Perform Colliders

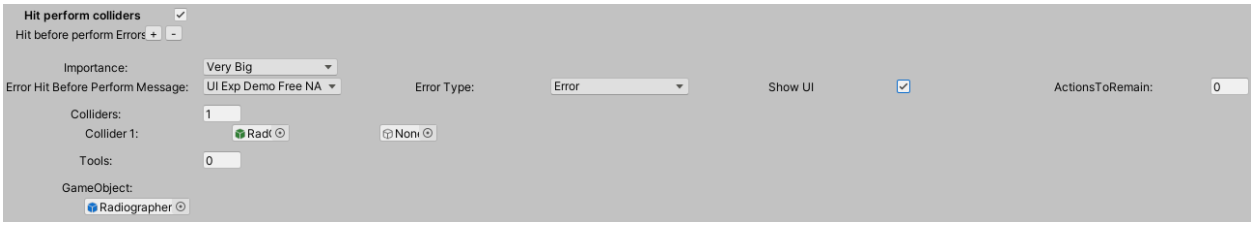
Usage: Users should interact with a collider/object before completing the Action.

Example: In this example, the user needs to answer a question about the correct placement of a vein catheter. However, to answer correctly, the user should perform an x-ray prior to that. To track if the user conducted the x-ray we spawn a collider next to the radiographer to trigger when running the x-ray.

Below you can see the collider (right) and the radiographer (left). The user will drag the radiographer from its handle and eventually hit our collider. In this way we are sure the radiographer was used properly.



Below you can see the analytics editor for this Action:



- We set the **Importance** to **VeryBig**, in this way a correct answer will give 100 points and a wrong answer 0.
- We drag and drop the radiographer prefab into the **Collider 1** input field.
- We drag and drop the radiographer collider (cube) into the **GameObject** input field. In this way the collider will be spawned.
- We set the **Error Message** to our custom message key from the dropdown
- **Spawns Error** is enabled
- We set the **Type of Error** to **Error** from the dropdown

6.2.8 Question

Usage: Lose points when answering a question incorrectly.

Example: In this example we will set the scoring factor to affect 100% of the score. Meaning that if the user answers wrong it will get zero points.

Below you can see the analytics editor for this Action:

- We set the **Importance** to **VeryBig**, in this way a correct answer will give 100 points and a wrong answer 0.
- We drag and drop the question prefab to the corresponding object field next the Importance.
- We set the **Error Message** to our custom message key from the dropdown
- **Spawns Error** is enabled
- We set the **Type of Error** to **Error** from the dropdown

6.2.9 Velocity

Usage: Track the velocity of an object. Lose points when the velocity overpasses the configured value.

Example: In this example we will set a velocity scoring factor on a fragile object since we don't want the user to grab it and move it very fast.

Below you can see the analytics editor for this Action:

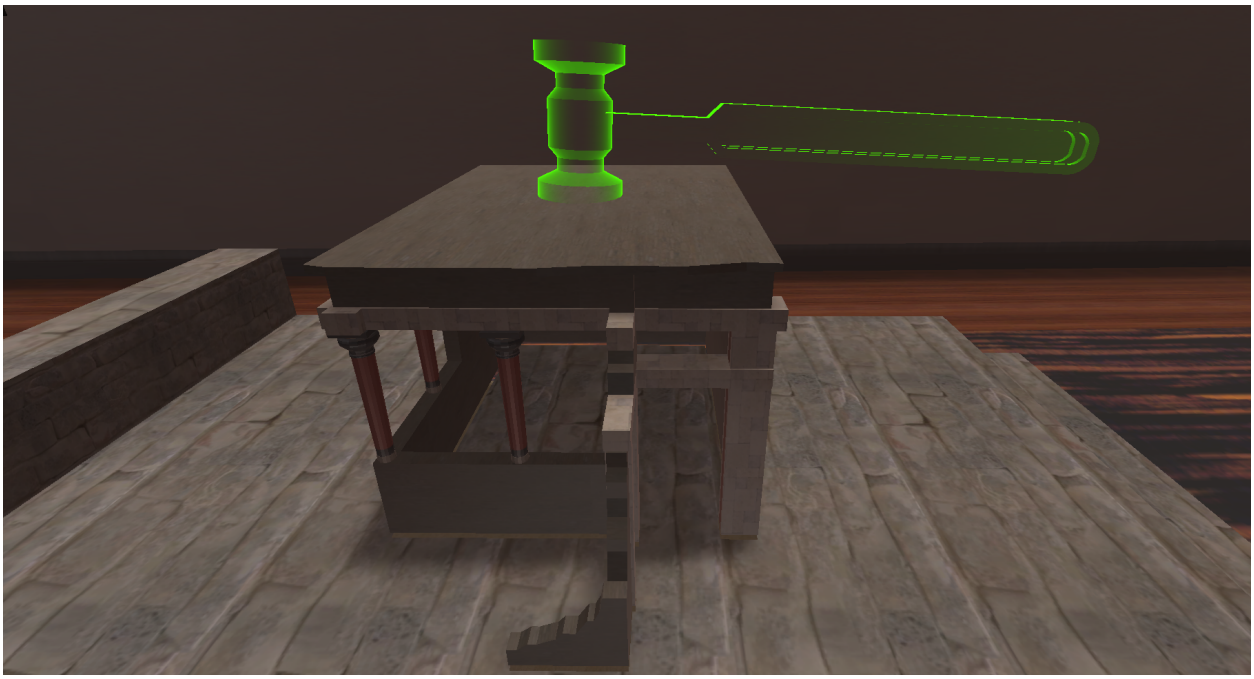
- We set the **Importance** to **Big**, in this way the Action will get a perfect score of 80/100
- We drag and drop our fragile object to the **Interactable Prefab** input field
- We set the **Velocity** value to **2.7**. If the velocity of the object overpasses 2.7 the user will lose 80 points.
- We set the **Error Message** to our custom message key from the dropdown
- **Spawns Error** is enabled
- We set the **Type of Error** to **Warning** from the dropdown

6.2.10 Custom Scoring Factor

Usage: Configure your own custom scoring factors if you need something special, not provided out-of-the-box.

Example: In this example, we set a custom scoring factor to measure the **velocity on impact** when hammering the Knossos building. User needs to take the mallet and hit the Knossos building three times. However, Knossos is fragile, so we set a maximum velocity to track the hits.

Here is the knossos building with the mallet hologram:



Note: For custom scoring factors we don't need the analytic editor. We will implement the behavior using C# code

First we create a new C# script/class that inherits from `ScoringFactor`

```
public class ForceScoringFactor : ScoringFactor
{
    public float maximumForce = 12;

    UpdateCollisionForce _forceScript;
    float _currentForce;
    GameObject _knossosBackPart;
    ForceScoringFactor _forceSf;
    int _score;
    LanguageTranslator _errorMsg;

    public override ScoringFactor Initialize(GameObject g)
    {
        _forceSf = g.AddComponent<ForceScoringFactor>();
        _knossosBackPart = GameObject.Find("BackPartHitMallet(Clone)");
        _forceScript = _knossosBackPart.AddComponent<UpdateCollisionForce>();
        _forceScript.Init(maximumForce,true);

        _forceSf._currentForce = 0;

        return _forceSf;
    }

    public override float Perform(bool skipped = false)
    {
        Destroy(_forceSf);
        Destroy(_forceScript);
        if (skipped) return 0;
        int errors = _forceScript.GetErrorsCounter();
```

(continues on next page)

(continued from previous page)

```
        _score = 100 - (errors * 20);

        return Mathf.Clamp(_score,0,100);
    }

    public override void Undo()
    {
        _score = 0;
        Destroy(_forceSf);
        Destroy(_forceScript);
    }

    public override object GetReadableData()
    {
        ScoringFactorData sfData = new ScoringFactorData();
        sfData.score = _score;
        sfData.outOF = (int) maximumForce;
        sfData.type = "Force Scoring Factor";
        sfData.scoreSpecific = (int) _forceScript.GetCollisionForce();
        sfData.errorMessage =  InterfaceManagement.Get.GetUIMessage(_errorMsg);

        return sfData;
    }
}
```

The **ForceScoringFactor** script implements our custom scoring factor.

The `ScoringFactor` class contains virtual functions for you to override in your custom scoring factors.

- **public override ScoringFactor Initialize(GameObject g)**

The Initialize method is called to setup your custom scoring factor. Everything you need to spawn or configure you should implement it in this function.

> **Warning:** Pay attention to the `_forceSf` variable. It contains an instance of your scoring factor. Use this instance to make your configurations for the Initialization.
>
> E.g. `_forceSf._currentForce = 0;`

The line below adds a script to the Knossos prefab. This script returns the applied force from the mallet. In the same way you can implement your own scripts to gather information about the user's performance.

```
_forceScript = _knossosBackPart.AddComponent<UpdateCollisionForce>();
```

- **public override float Perform(bool skipped = false)**

The Perform method is called along with the Action's Perform(). The purpose of this method is to return the score of the user. It calculates the score with data retrieved from the `UpdateCollisionForce` script.

Make sure you clamp the score at 100

```
return Mathf.Clamp(_score,0,100);
```

- **public override object GetReadableData()**

GetReadableData manages the data from the custom scoring factor that will be saved at the end of the Action.

It instantiates a new ScoringFactorData variable and overrides its fields:

1. **score:** The user's score

2. **outOF:** In case the scoring factor contains a number of possible values (e.g angles, questions etc) this variable reflects this amount
3. **type:** The scoring factor's name
4. **scoreSpecific:** The score directly from the script attached on our object. In this case the `UpdateCollisionForce`
5. **errorMessage:** The error message to spawn when triggering this error

The final step is to link this scoring factor with our Actions script.

Below you can see the `AssembleKnossosPartOfAction`, our Action script.

```
public class AssembleKnossosPartOfAction : CombinedAction
{
    public override void Initialize()
    {
        AnalyticsManager.AddScoringFactor<ForceScoringFactor>(2);

        //InsertAction sub-Action
        //......
        //-------------------------------------------------------------------------------
↪----------------
        //InsertAction sub - Action
        //......
        //-------------------------------------------------------------------------------
↪----------------
        //ToolAction sub - Action
        ToolAction hitWithMallet = gameObject.AddComponent<ToolAction>();
        hitWithMallet.SetToolActionPrefab("Lesson0/Stage1/Action0/BackPartHitMallet",␣
↪MAGES.ToolManager.tool.ToolsEnum.Mallet);
        hitWithMallet.SetHoloObject("Lesson0/Stage1/Action0/Hologram/
↪MalletHologramL0S1A0");

        InsertIActions(insertFrontGateAction, insertBackGateAction, hitWithMallet);

        base.Initialize();
    }
}
```

Warning: `AnalyticsManager.AddScoringFactor<ForceScoringFactor>(2);`

This line connects our custom scoring factor with this Action. Don't forget to add this line.

The `(2)` argument is because we attached this scoring factor to the second sub-Action.

This is the proper way to configure a custom scoring factor.

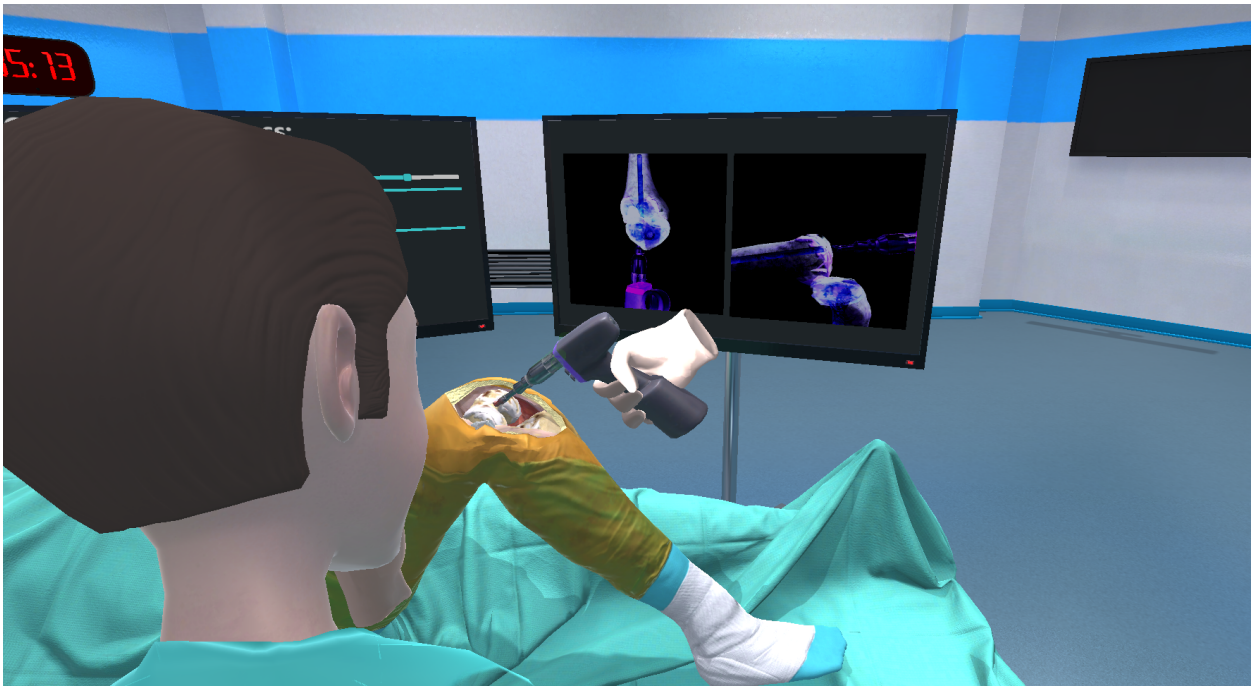
6.3 Mechanics

6.3.1 Xray Monitor

In this tutorial we will demonstrate how to develop a real-time X-ray monitor.

This mechanic is useful in scenarios that you need to have a real-time visualization of the human skeleton to proceed with a particular Action or come up with a conclusion regarding the patient.

In this example, we will extend the MedicalSampleApp scenario by adding a real-time X-ray monitor to provide guidance during the femur drilling.

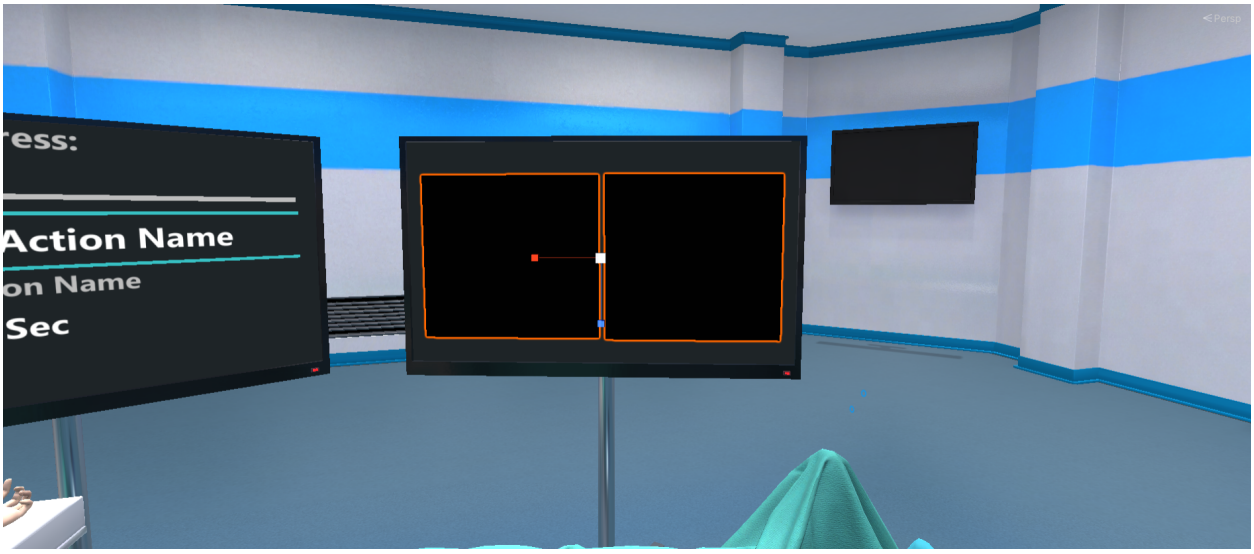


In short, we will do the following steps:

1. Attach two cameras on femur to follow the knee movement.
2. Make the cameras render only the bone and the tools (using layers).
3. Set the cameras to render on render textures.
4. Set the render textures as Albedo textures on a material.
5. Attach the materials on two planes (X-ray monitors).
6. Apply post process effects to the cameras.

Setting the scene

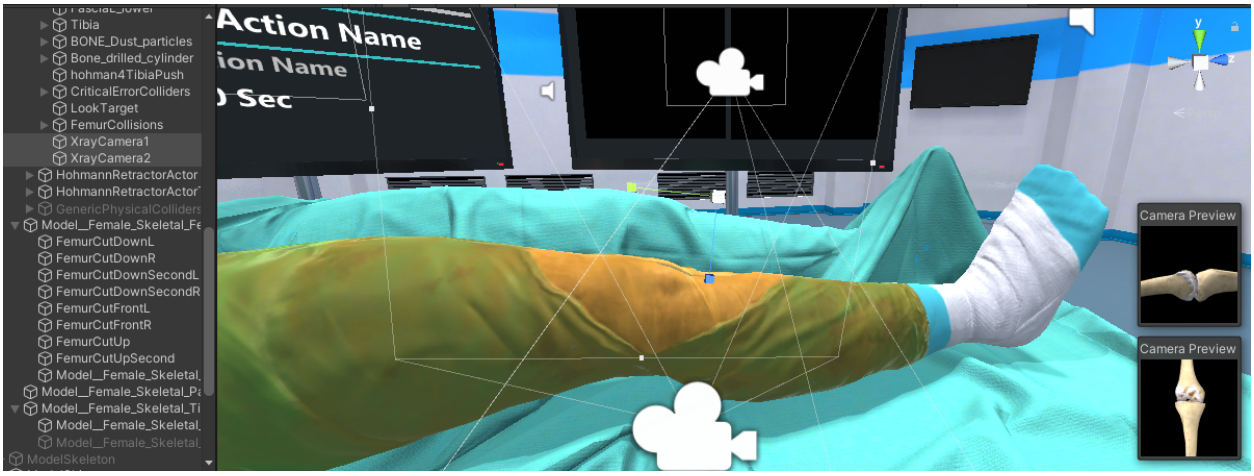
We begin by setting the scene. We need an additional monitor to visualize the X-ray. Furthermore, we duplicate the Action's monitor, remove the texts (Action names, score etc) and create two planes that will render the two X-ray visualization from two different angles. Below you can see the xray monitor:



The X-ray cameras

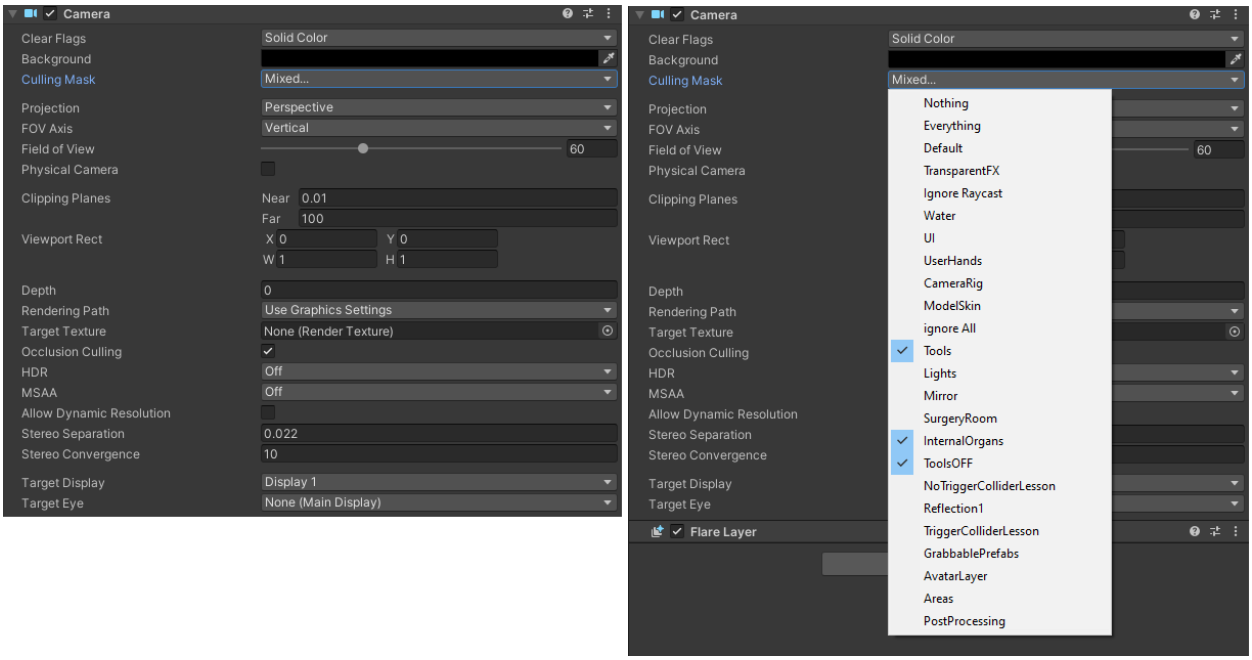
The next step is to generate two cameras that will render the X-ray image.

Here you can see the two cameras we created (**XrayCamera1** and **XrayCamera2**). For our scenario we attached both cameras to the **Femur** joint to follow the movement of the femur when rising.



The cameras should to render only the bones and the tools (drill). To implement this, we select from the **Culling Mask** option within the Camera script the appropriate Layers. We set the **Tools** and **ToolsOFF** layers to render the tools and the **InternalOrgans** layer to render the bones.

Note: Make sure you set the bones of the patient to the **InternalOrgans** layer, otherwise the cameras will not render them.



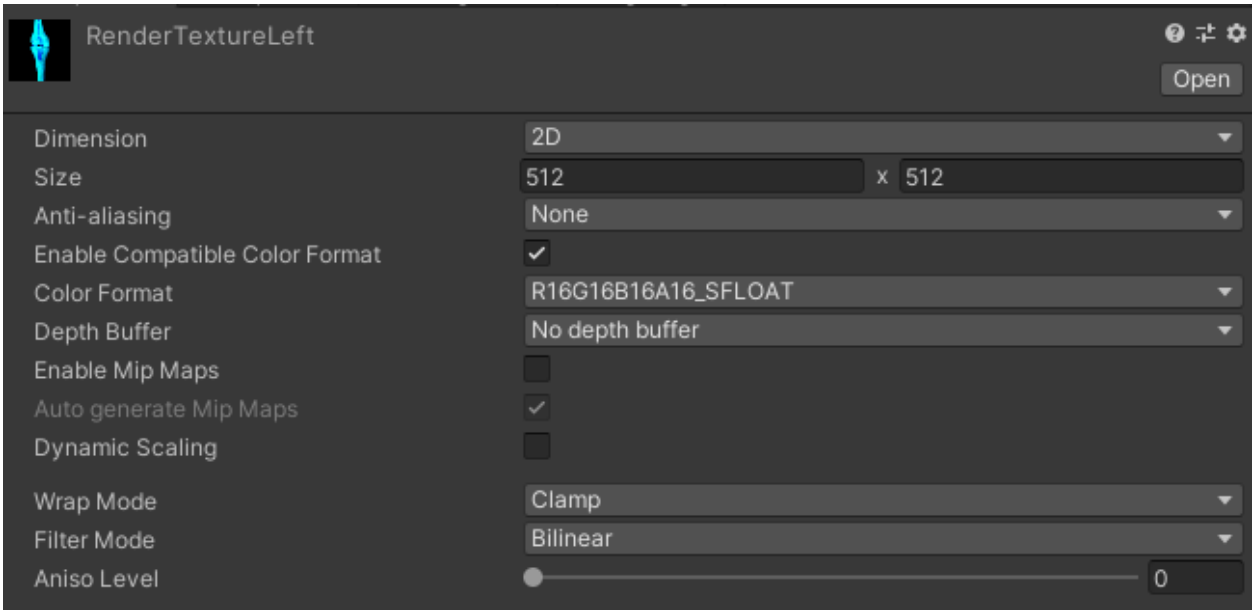
Here you can see the preview of both cameras.

Render textures

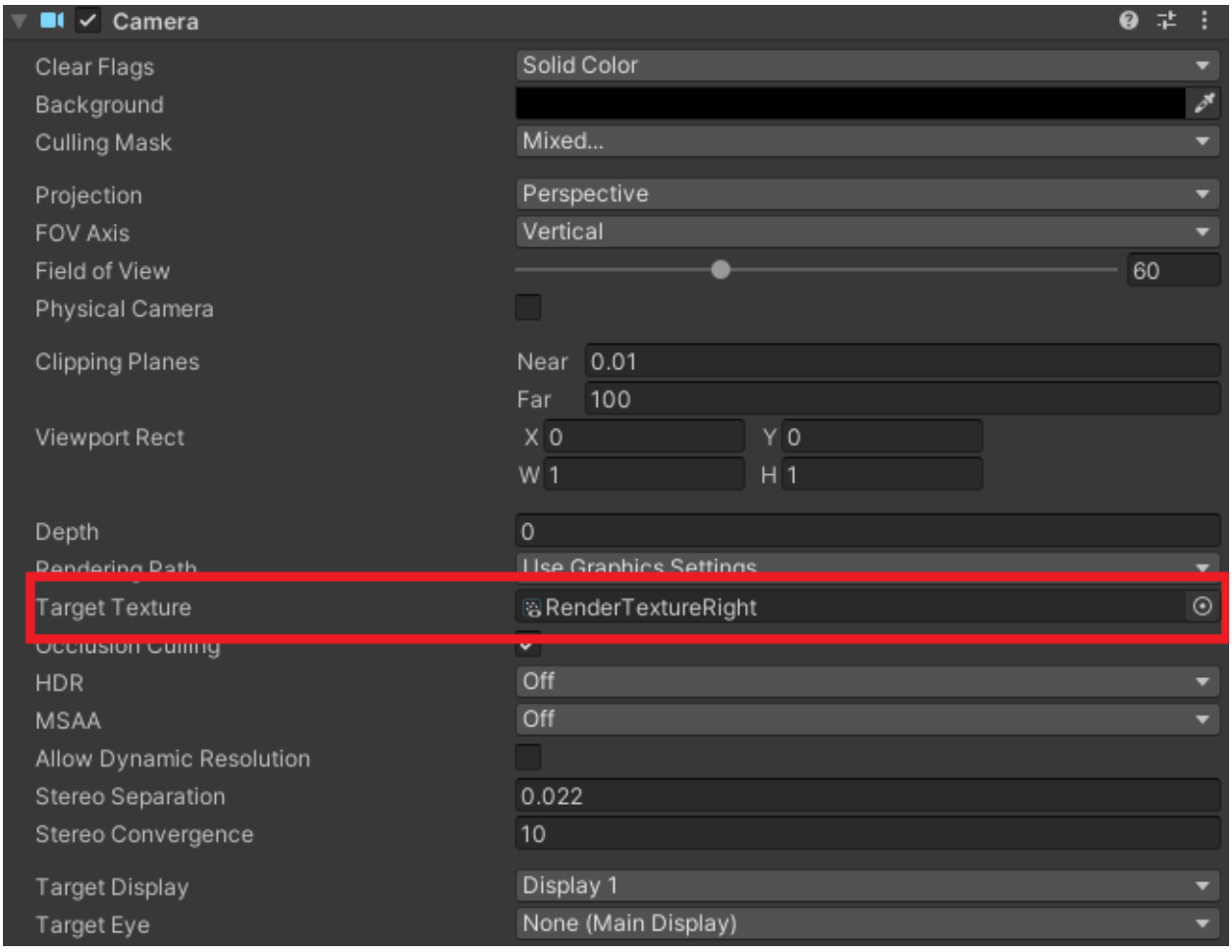
However, we don't want to render those two cameras in our main viewport. We should render them on the monitor that we created. For this reason, we will use render textures.

Right-click on the assets editor **Create > Render Texture** to generate your render texture.

Note: It is important to set the **Depth Buffer** to **No depth buffer**. This will create the X-ray effect as it will render the bones without paying attention to any culling.



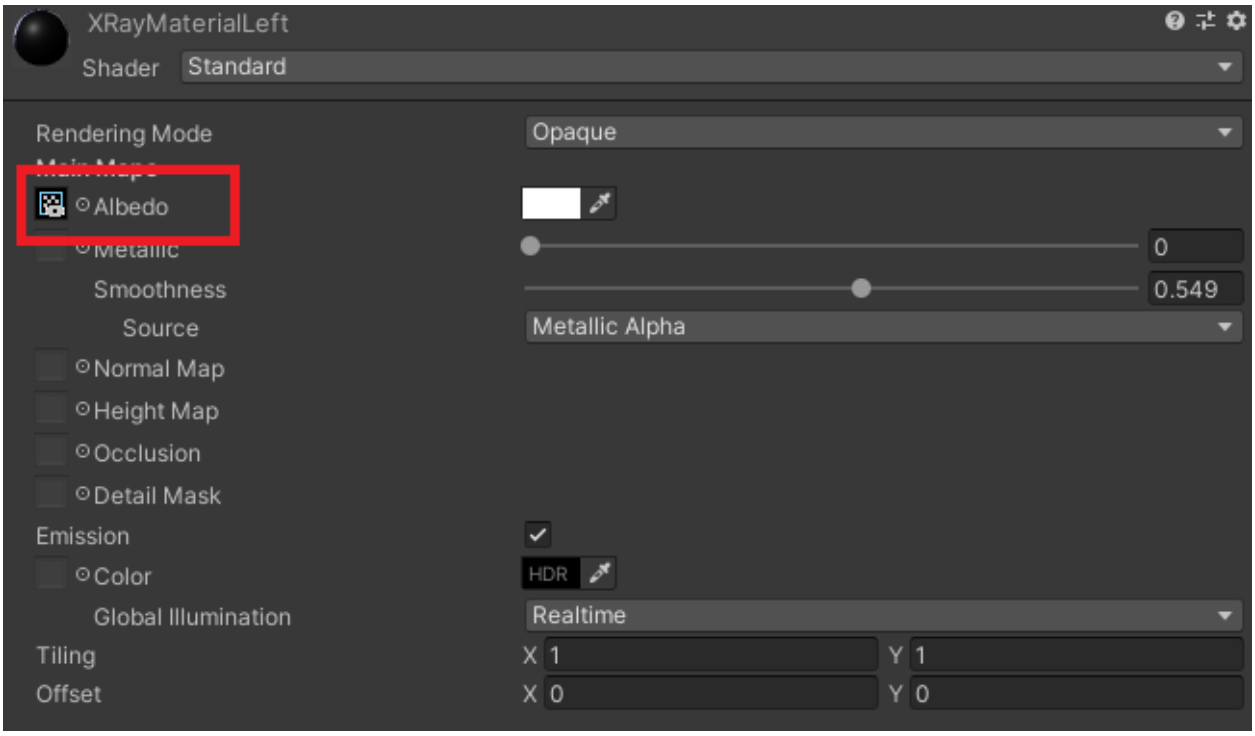
Then, link the render texture to your camera by drag and drop it to the **Target Texture** field.



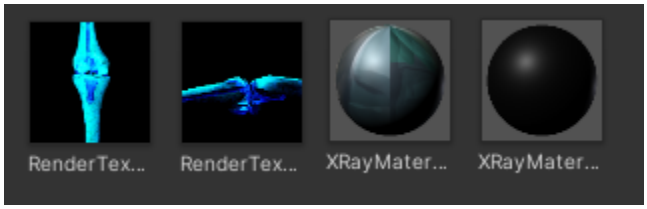
Now this camera will render on this texture.

Finally, we need a material to attach this texture and apply the material to our X-ray monitor.

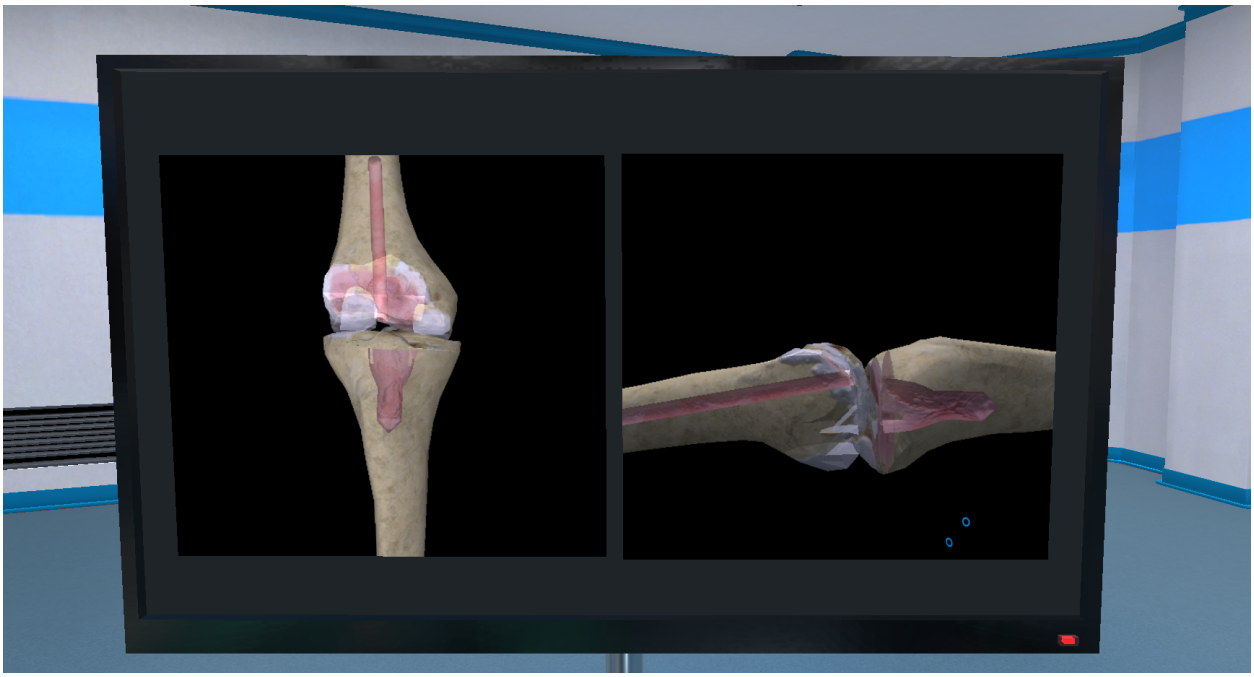
Create a new material, right-click on the assets editor **Create** → **Material** and drop the render texture at the **Albedo** field.



Below you can see the two rendered textures along with their materials.



Drag and drop the materials we created on the X-ray monitors. Below you can see the result.



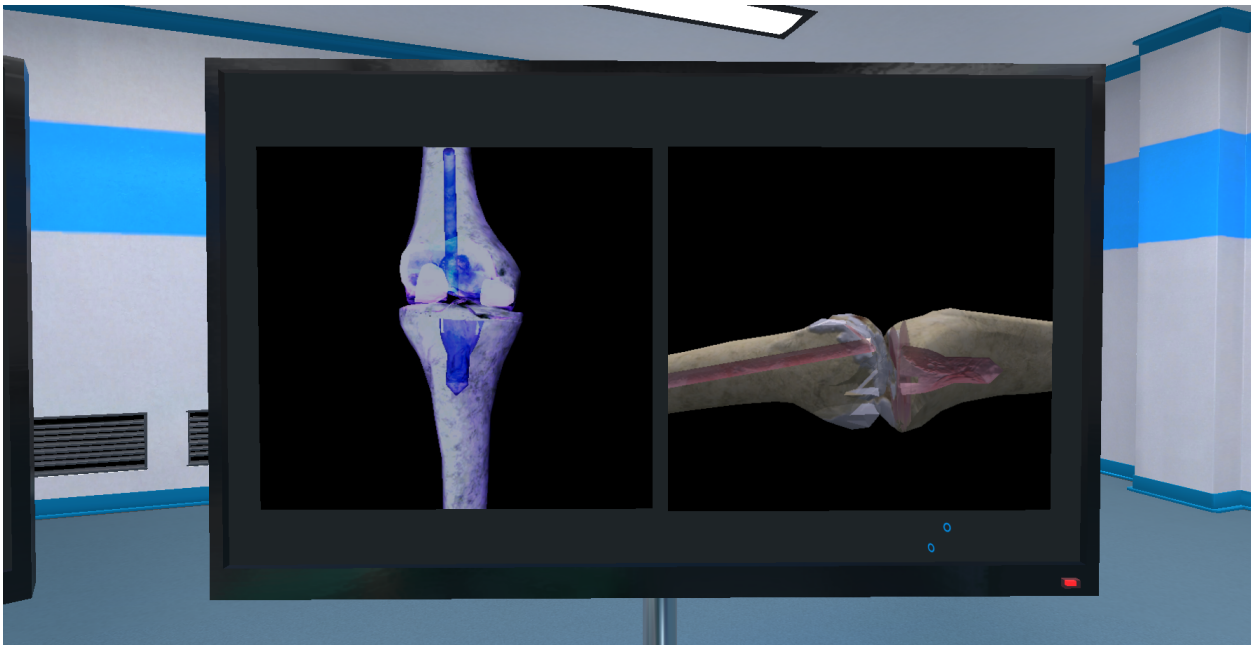
Post process effects

Note: This step is optional and depends on your design choices.

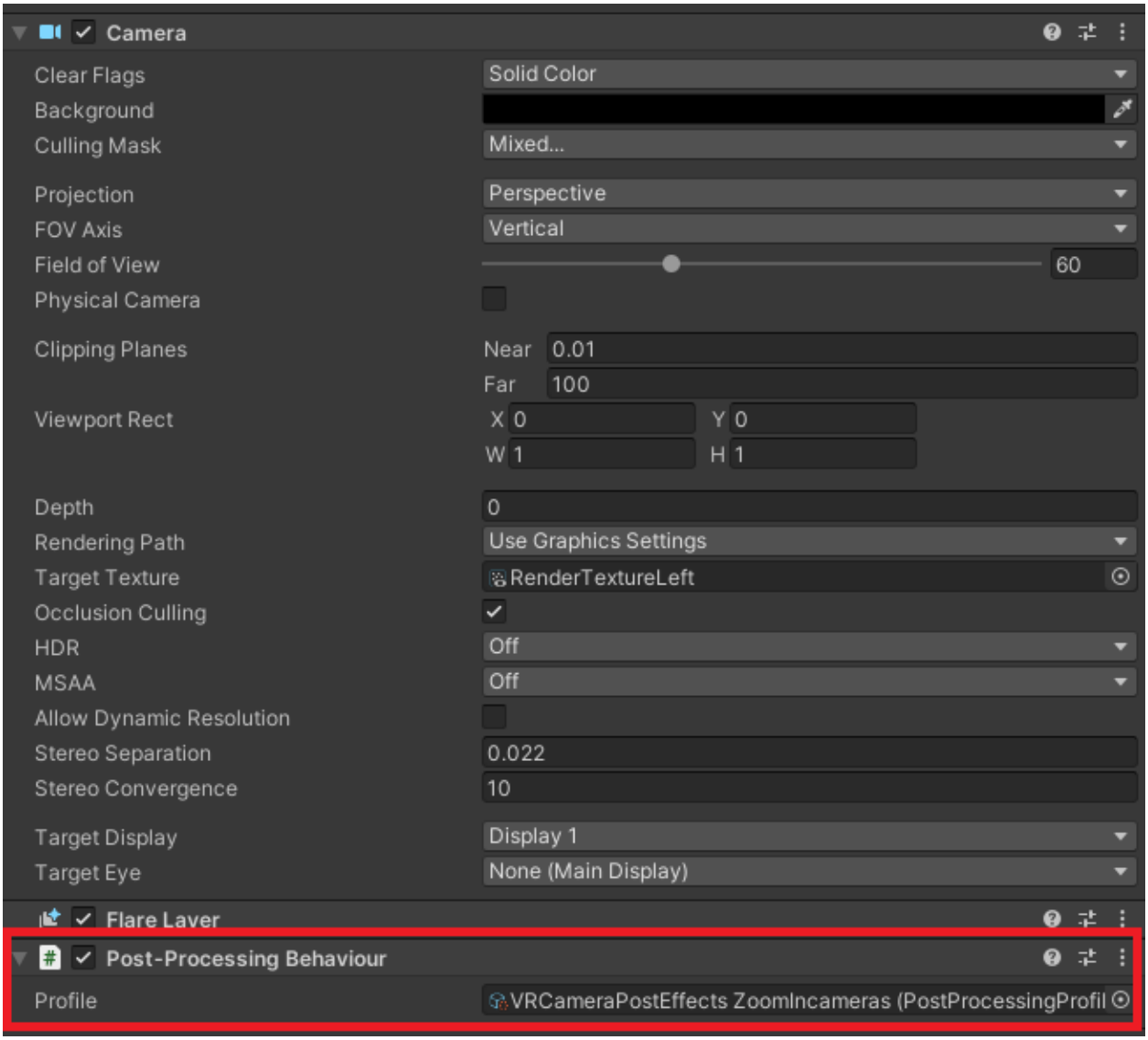
In this step we will add post process effects to our camera to replicate the X-ray effect.

In this tutorial we will use the deprecated post process effects (not from the package manager). The updated post process effects caused an unwanted behaviour by overriding the depth buffer. There are solutions to fix this issue on URP, but this project uses the standard pipeline, and you cannot modify it.

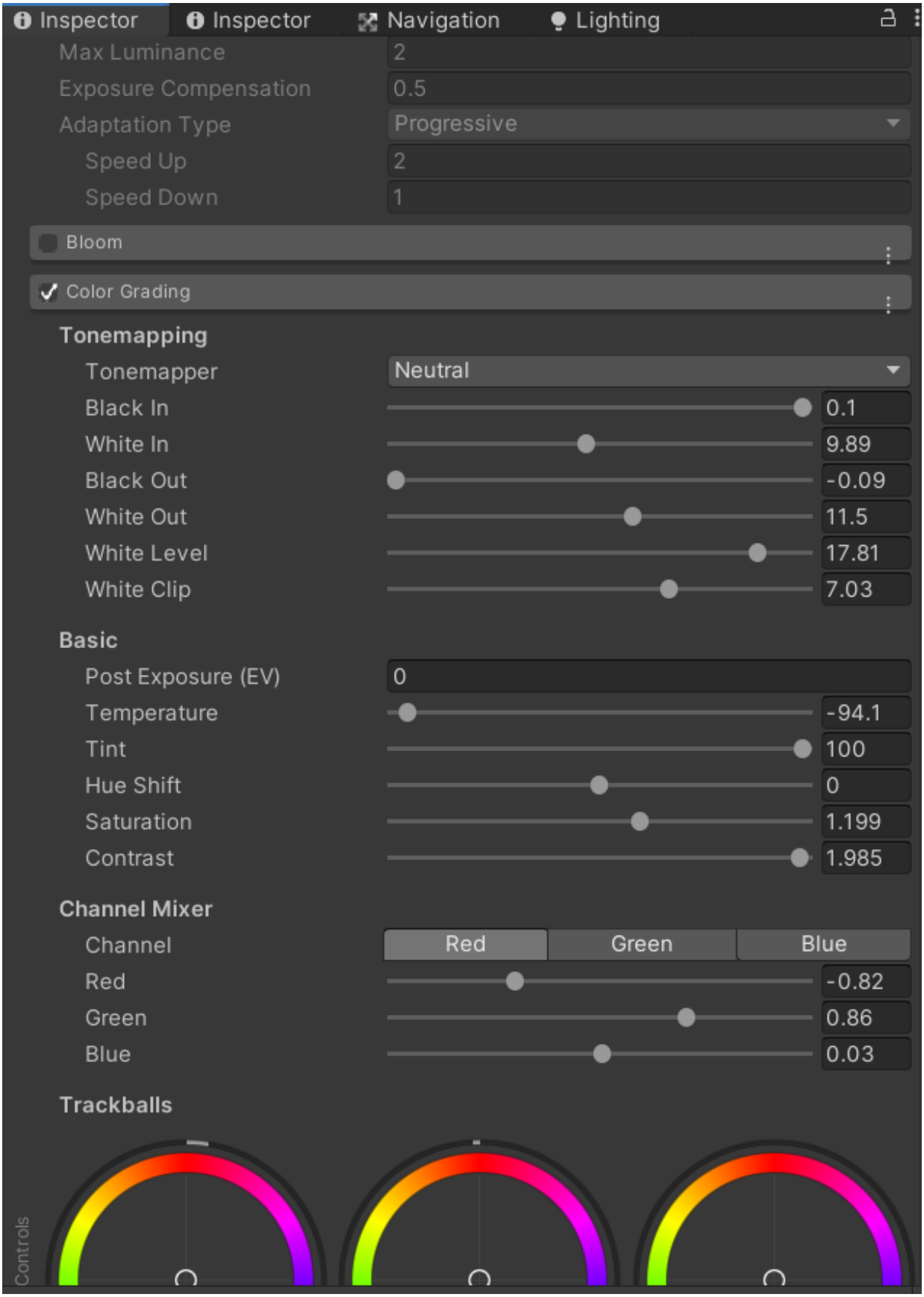
Below you can see the left camera with the post process effects applied and the right camera without any post process effect.



We attach the **PostProcessingBehaviour** script on the cameras.

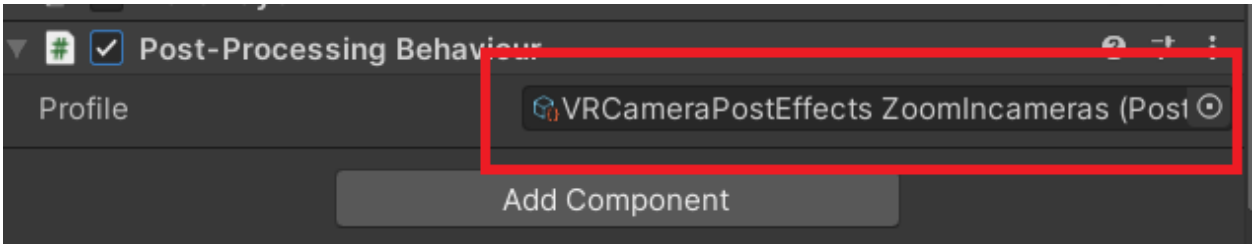


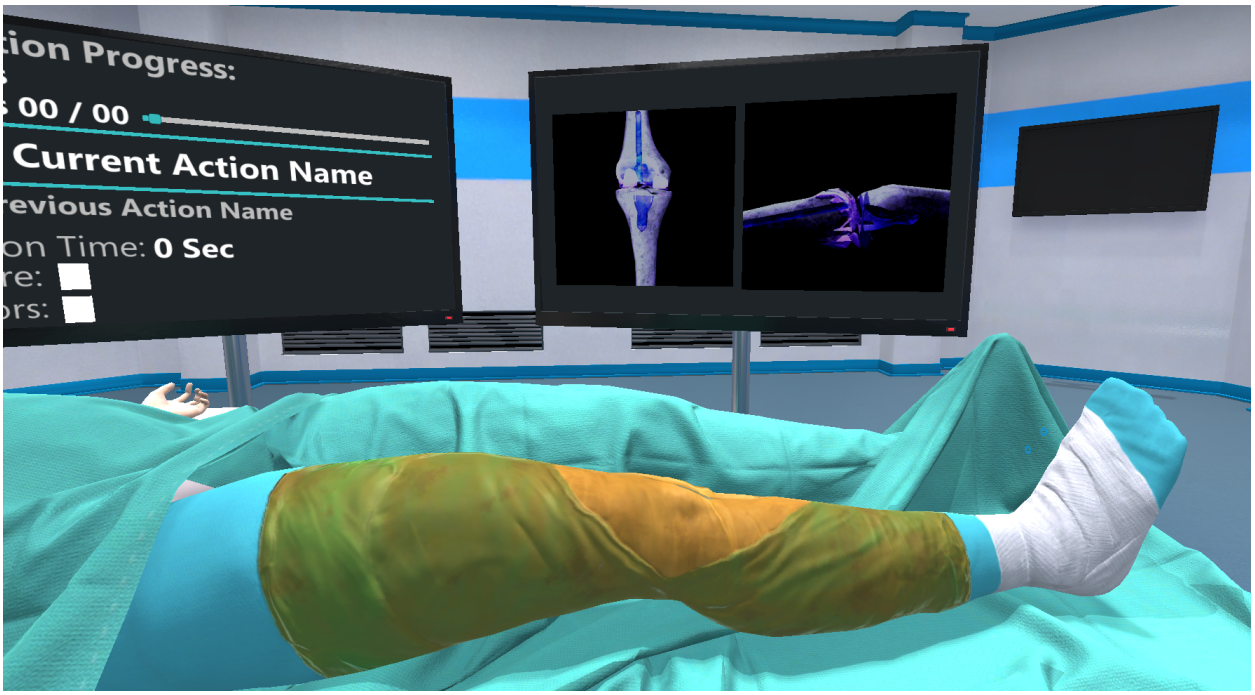
Rigt click on the assets editor **Create > Post-processing profile** to generate a post-processing profile.


Inspector
Inspector
Navigation
Lighting
Max Luminance
2
Exposure Compensation
0.5
Adaptation Type
Progressive
Speed Up
2
Speed Down
1
Bloom
Color Grading
Tonemapping
Tonemapper
Neutral
Black In
0.1
White In
9.89
Black Out
-0.09
White Out
11.5
White Level
17.81
White Clip
7.03
Basic
Post Exposure (EV)
0
Temperature
-94.1
Tint
100
Hue Shift
0
Saturation
1.199
Contrast
1.985
Channel Mixer
Channel
Red
Green
Blue
Red
-0.82
Green
0.86
Blue
0.03
Trackballs
Controls

Enable the **Color Grading** option and experiment with the **Saturation** and the **Channel Mixer** till you have the desired X-ray effect.

Finally, drag and drop the post-processing profile to the **PostProcessingBehaviour** script on the cameras. This will apply the X-ray effect.

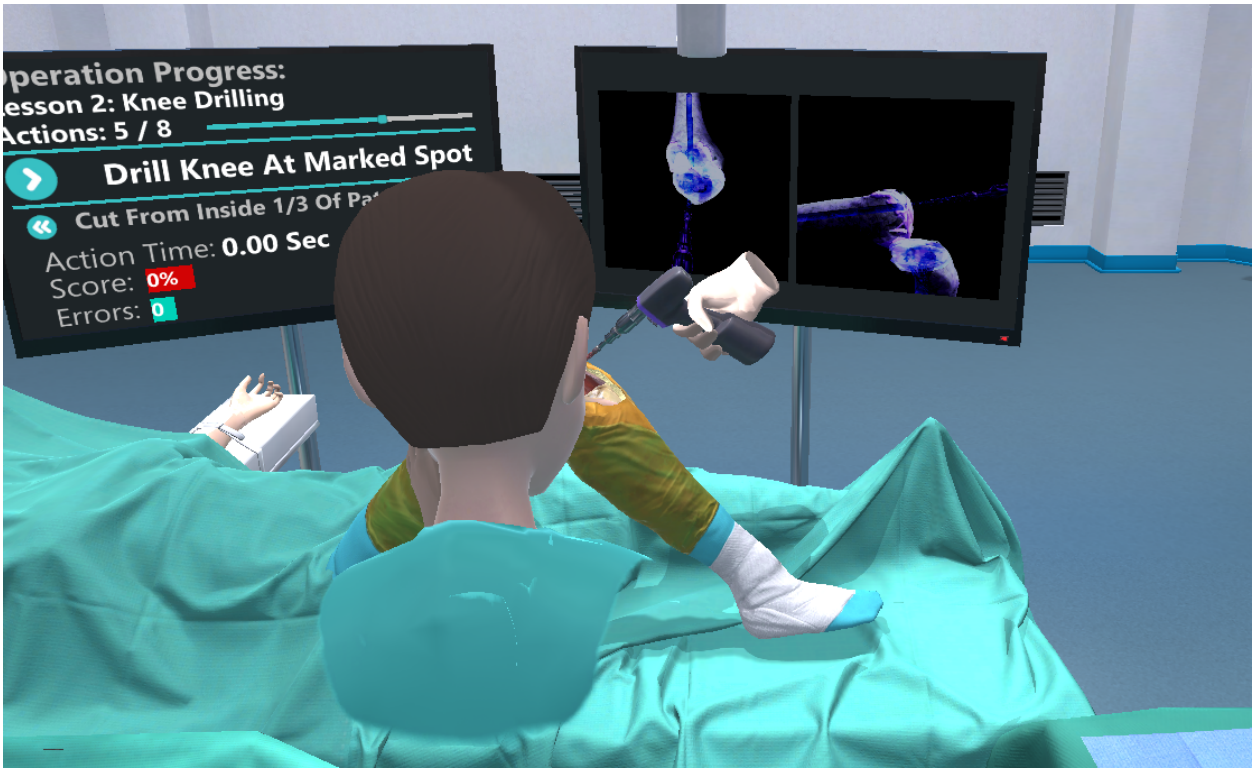


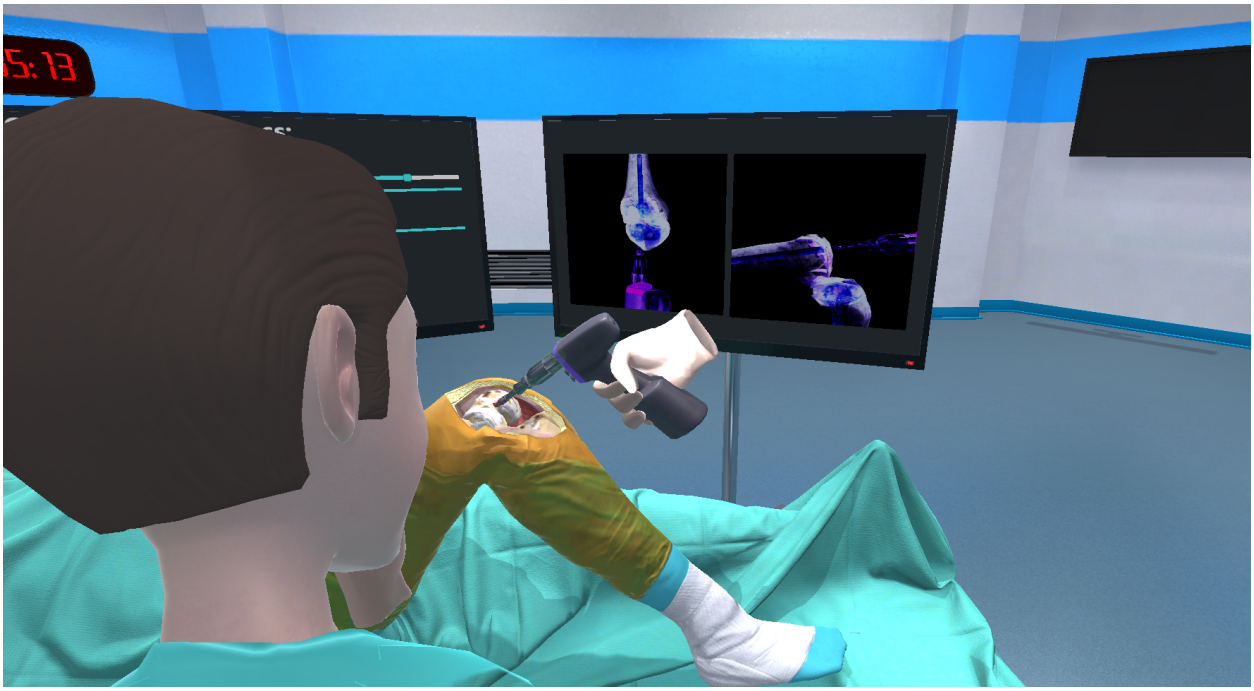


Results

As you see both the skeleton and the drill are visible to the X-ray monitor.

In this way the user can practice the femoral drilling, visualizing real time the angle in both axes.


peration Progress:
esson 2: Knee Drilling
ctions: 5 / 8
Drill Knee At Marked Spot
Cut From Inside 1/3 Of Pa
Action Time: 0.00 Sec
Score: 0%
Errors: 0


5: 13

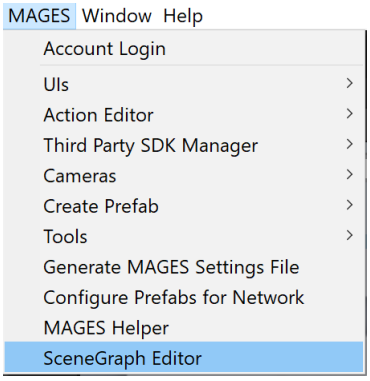
6.4 Scenegraph Generation

In this tutorial we will generate a scenegraph tree from the beginning.

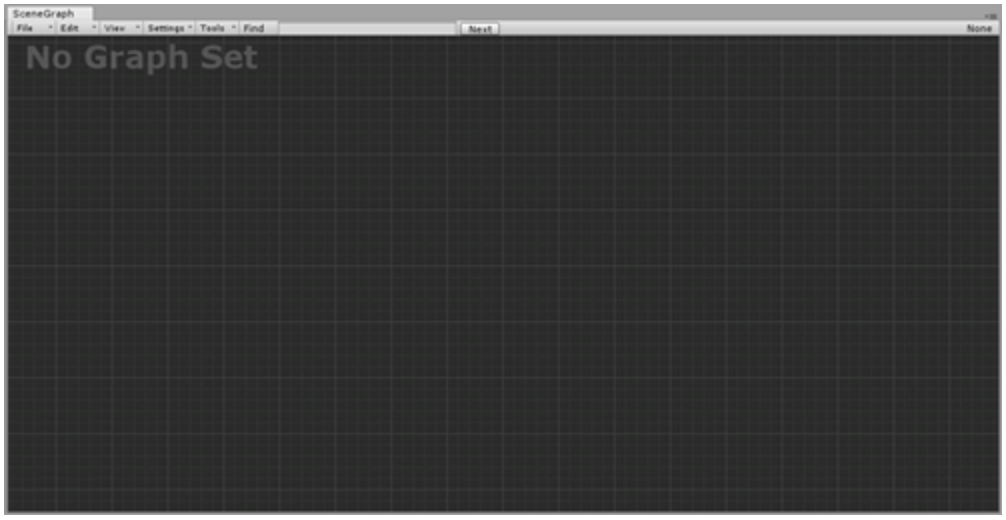
6.4.1 Visual Scripting

You don't need to edit the xml files from a scripting editor. We provide a user-friendly editor to load, generate and make all the necessary updates for your convenience.

The first step is to open the visual scritping editor, navigate to **MAGES** → **SceneGraph Editor** in the Unity Editor.



The visual scripting window will appear on your screen.



Controls

Controls for the Scenegraph Editor are depicted in the table below:

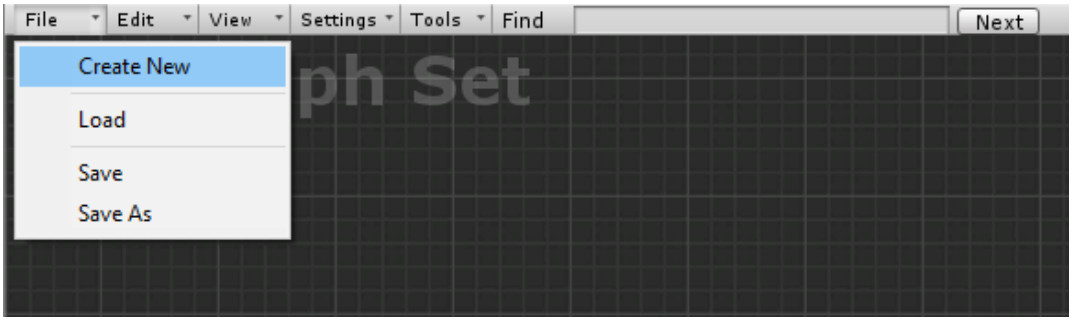
Left Click	Select Nodes and drag them to place
Right Click	When clicked on a Node opens the option menu for this Node. Also used to generate new nodes.
Middle Click	Press the middle click and drag on the window to move around the editor.
Shift + Left Click	Panning in Windows and macOS
Scroll Wheel	Zoom in - Zoom out.

Create New Scenegraph

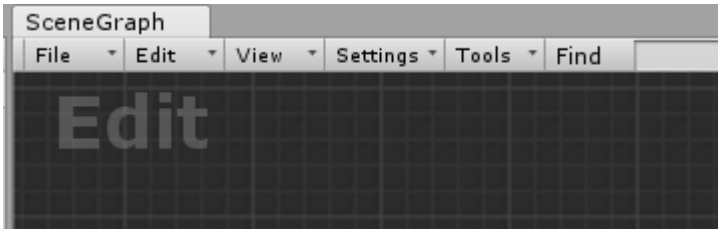
To create a new xml follow the steps below:

1. From the top menu click on **File** → **Create New**.

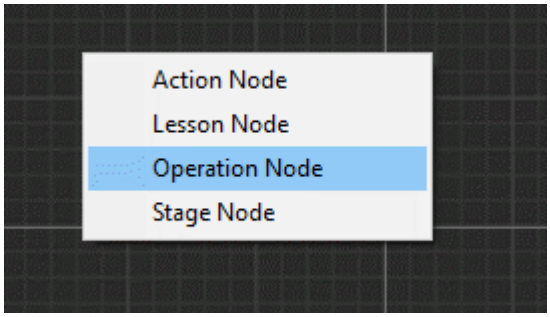
    1.1. As soon as you create a new empty graph you will be set in **Edit** mode.



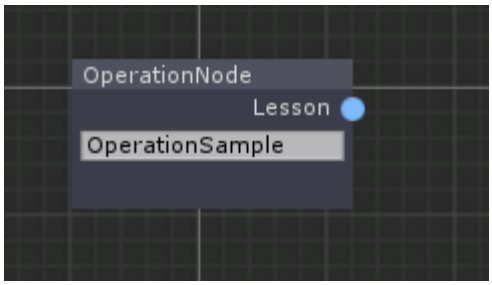
    1.2. In **Edit** mode you can start editing the graph.



2. To create Lesson-Stage-Action (LSA) nodes, right click inside the window to open the node generation menu.

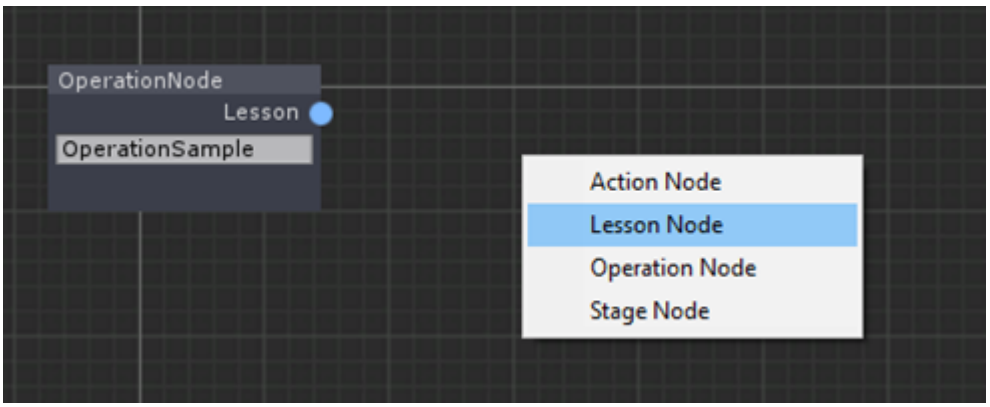


    2.1. The first node you need to create is the Operation Node. This is the base node of the Scenegraph tree. To generate the base node click on the Operation Node button.



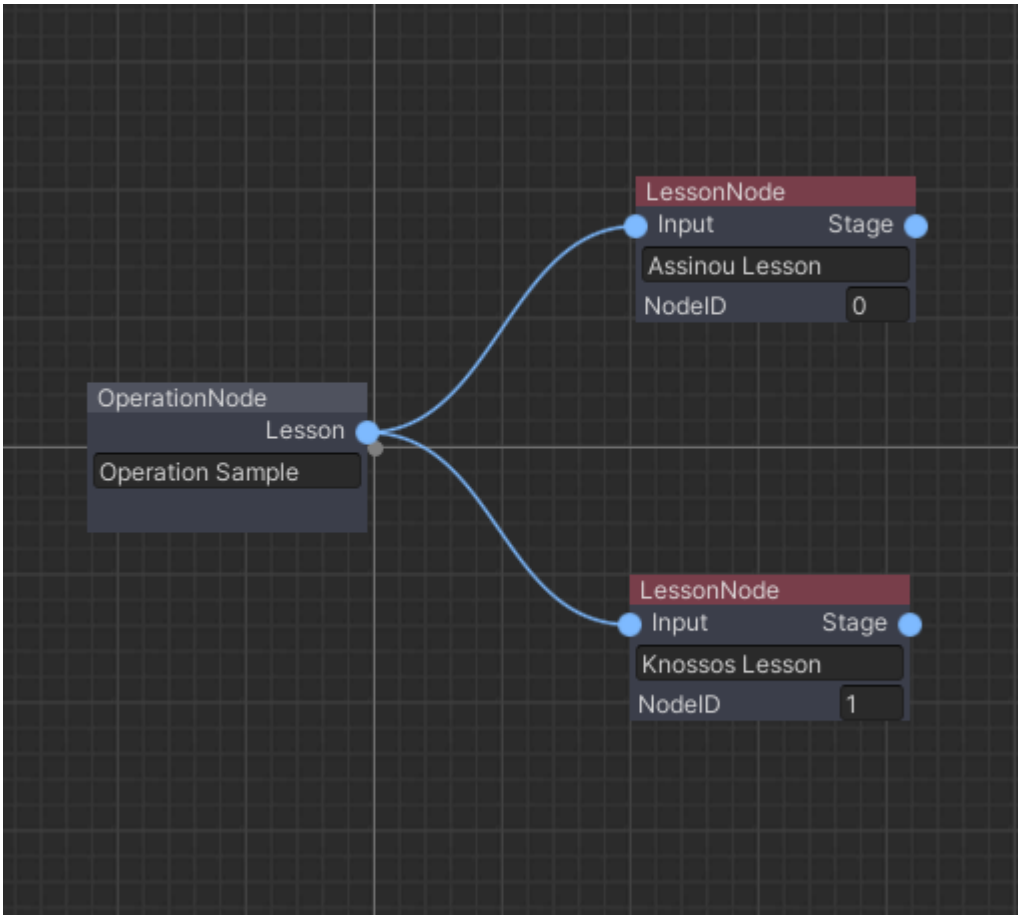
The Operation Node will appear on the editor. The next think you want to do is to set the Name. To do that click on the text box inside the node and type the name. You can drag the node around the editor by pressing and holding the right click on the node you created.

2.2. Now let's create some Lessons. Press right click again and select Lesson Node.

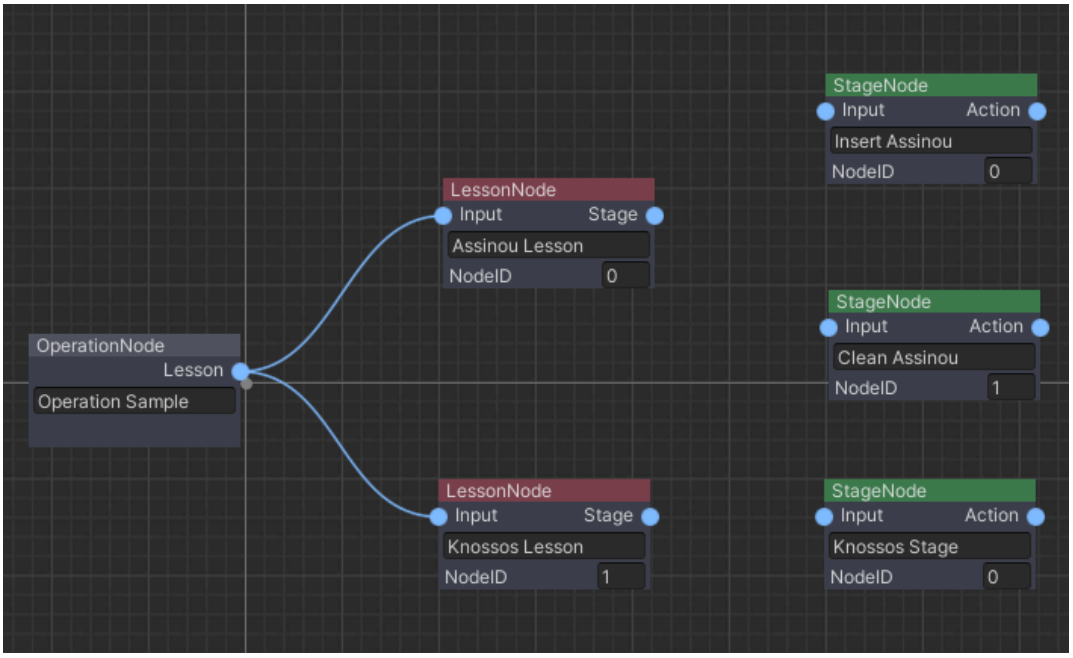


The Lesson Node will appear and is automatically connected to the Operation Node. Now you have to register two fields. First it's the name of the Lesson and the second one is the NodeID. This value refers to the order of nodes.

For example, if you want this Lesson to be the third lesson of the Scenegraph, set *NodeID=2* (numbering starts from 0). Below we have created two Lessons. Pay attention to the NodeIDs.

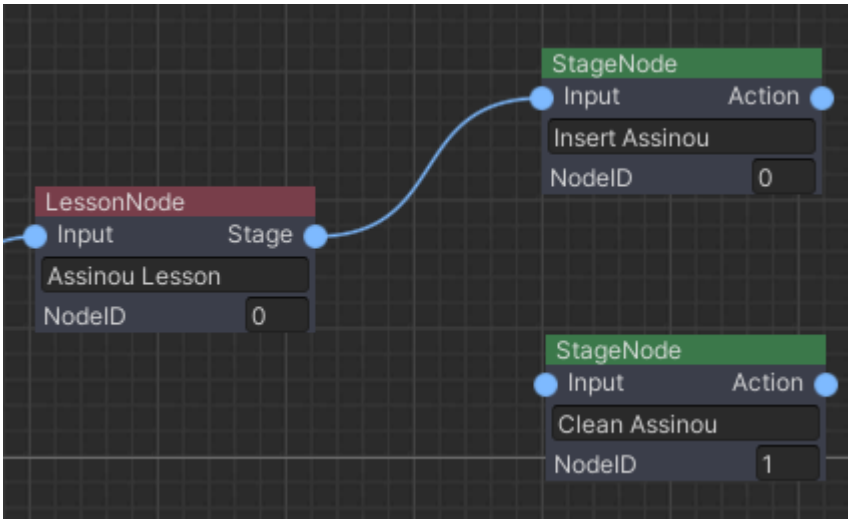


2.3. Applying the same methodology we will create some stages. Right click and select the Stage Node.

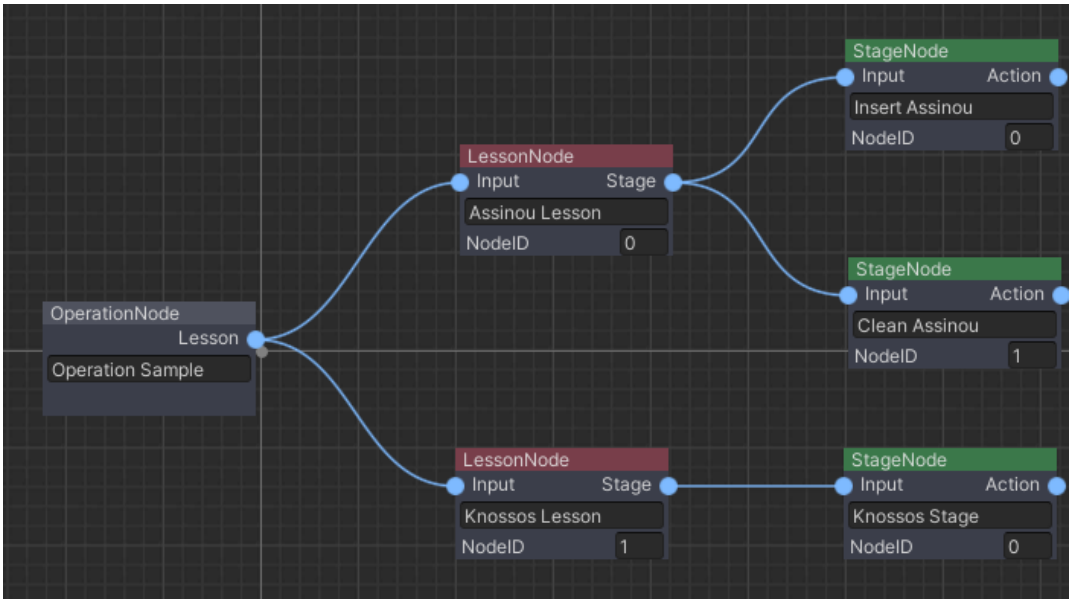


To connect the nodes together you right-click on the right blue circle of a node (Output Node) and release the click on the left blue circle (Input Node) of another node.

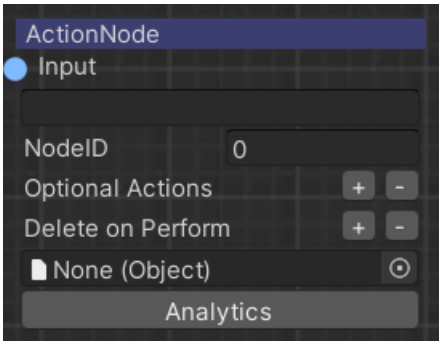
A small example below:



Now proceed to link all stages with their lessons. This is an image of the Scenegraph so far:

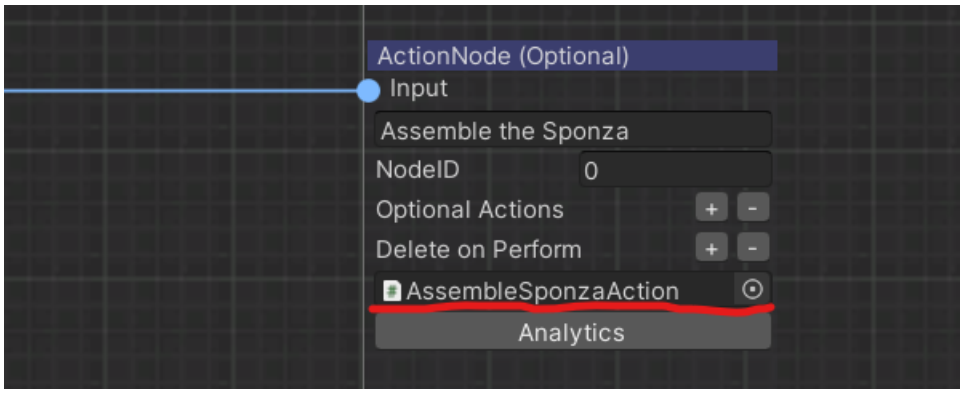


2.4. Now let's set the Action Nodes. Right click and select the Action Node.

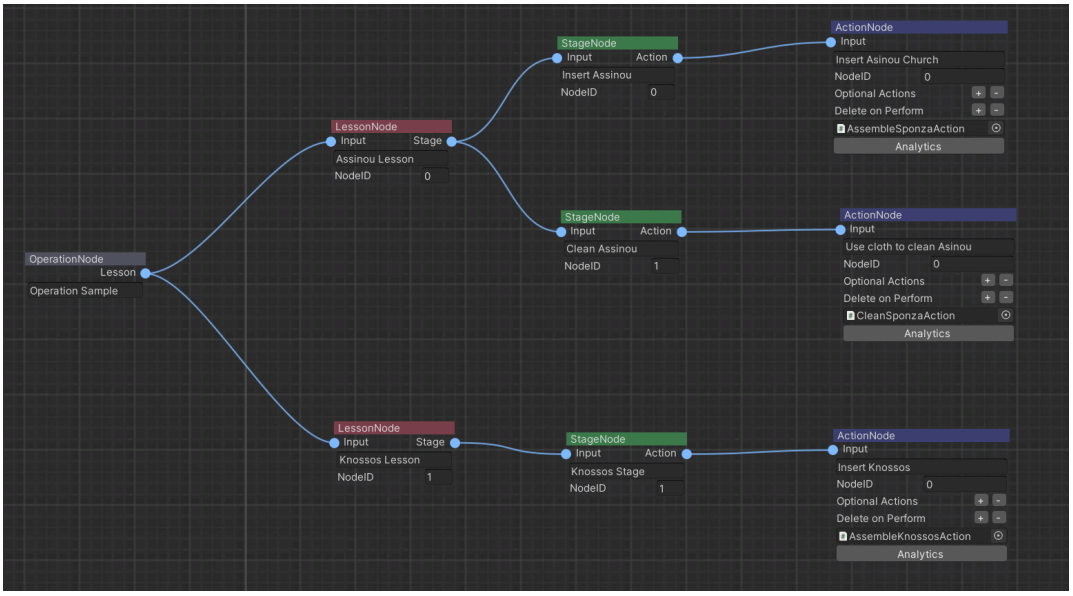


The Action Node has all the fields described above at the xml section, plus the NodeID to set the action order. The second to last field is to set the Action script. You can simply drag and drop your Action Script there or press the black circle on the left side to search it from there.

Note: Remember to drag and drop your Action Scripts!

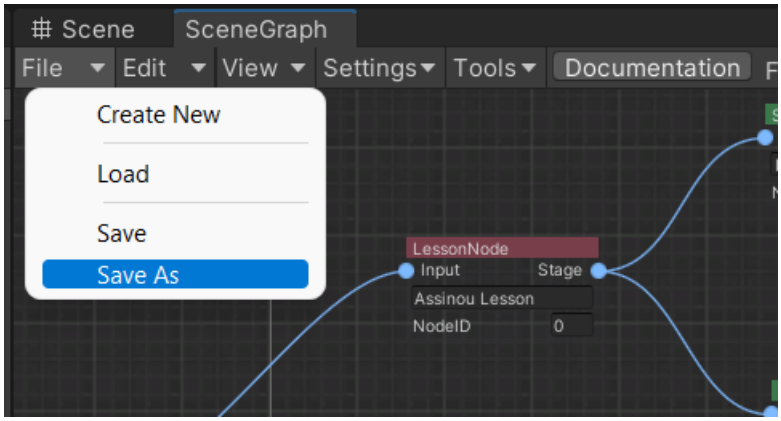


The Scenegraph tree is now complete.



Note: The Analytics button in the Action Node will be explained in the Analytics tutorial.

3. We are now ready to generate the Scenegraph.xml. Click on **File/Save as**.



A windows dialogue box will appear to save the file as an xml (If you click Save As, otherwise the Scenegraph will be saved on the existing xml). Insert the name of file and the xml will be saved. In this example we will name our scenegraph `SampleApp.xml`.

The xml has been successfully generated!

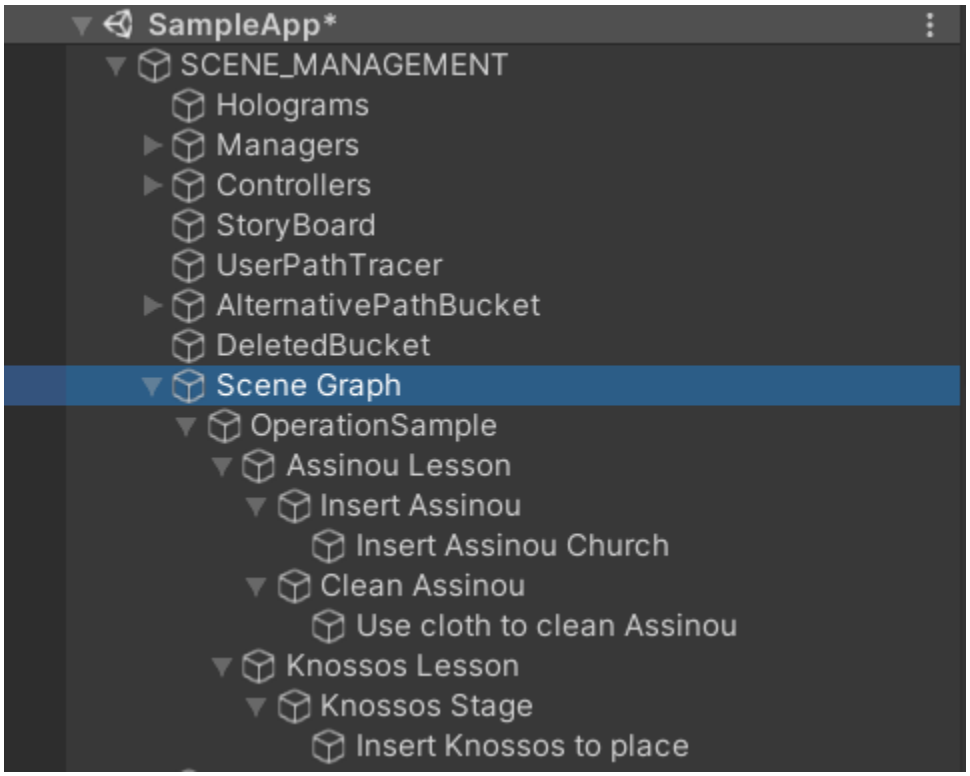
```
<?xml version="1.0"?>
<ArrayOfLessons xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Lessons>
    <Lesson_Name>Assinou Lesson</Lesson_Name>
    <Tag>Normal</Tag>
    <Stages>
      <Stage_Name>Insert Assinou</Stage_Name>
      <Actions>
        <Action>Insert Asinou Church</Action>
        <ActionClassName>AssembleSponzaAction</ActionClassName>
        <optionalActions />
        <destroyOnPerform />
      </Actions>
    </Stages>
    <Stages>
      <Stage_Name>Clean Assinou</Stage_Name>
      <Actions>
        <Action>Use cloth to clean Asinou</Action>
        <ActionClassName>CleanSponzaAction</ActionClassName>
        <optionalActions />
        <destroyOnPerform />
      </Actions>
    </Stages>
  </Lessons>
  <Lessons>
    <Lesson_Name>Knossos Lesson</Lesson_Name>
    <Tag>Normal</Tag>
    <Stages>
      <Stage_Name>Knossos Stage</Stage_Name>
      <Actions>
        <Action>Insert Knossos</Action>
        <ActionClassName>AssembleKnossosAction</ActionClassName>
        <optionalActions />
        <destroyOnPerform />
      </Actions>
    </Stages>
  </Lessons>
</ArrayOfLessons>
```

The final step is to configure the xml file in the MAGES Settings.asset.

Navigate MAGESSettings.asset on your Assets and drop the xml to the Operation XML field.

unity/tutorials/scenegraph/img/scenegraph_link.png

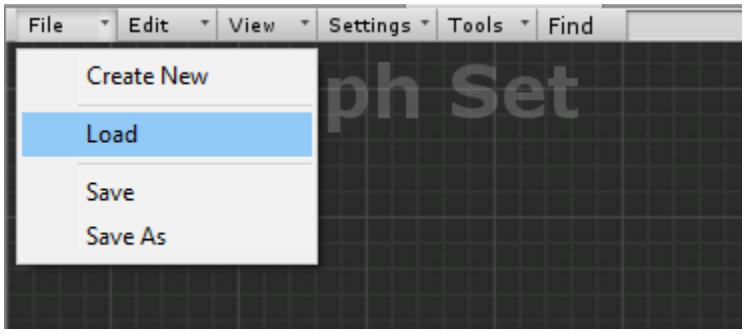
That's it! Press play on the Unity Editor and you should see the scenegraph populate under the **Scene Graph** gameobject.



Extra Functionalities

Open an existing xml

If you want to open an existing xml go to **File/Load**.



Select your XML file from the windows dialogue box.

Generate Optional Nodes

Optional nodes can be created from existing Nodes. Right-click on the node you want to make it alternative and select "Set Optional Node" option.

Delete A Node

To delete a Node right click on the node and select the "Delete Node" option.

Clear All Nodes

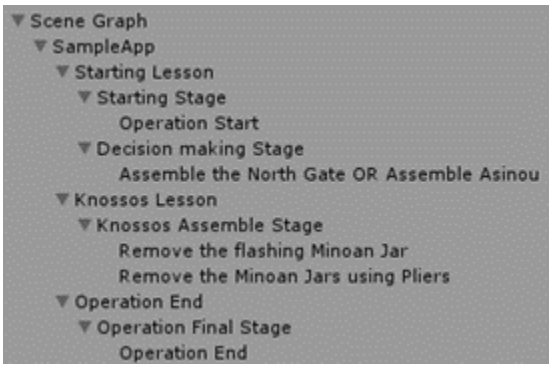
To clear (delete) all the nodes press Tools/Clear Nodes.

Find Node by Name

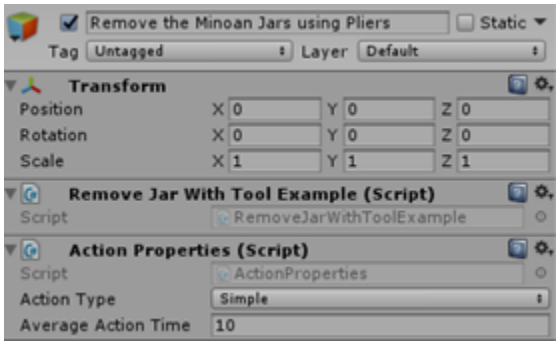
To find a node by its name you can search for it in the *Find* region.

Type in the text box and click *Next* button to get the next reference on the Scenegraph tree.

Now we have a complete Scenegraph and we are ready to start the application. After successfully generating the Scenegraph, the "Scene Graph" gameobject will look like this in play mode.



As you can see the Scenegraph gameobject has been populated with all the LSA Nodes from the xml importing. An example Action Node of Scenegraph will look like this.



6.5 HandPoser

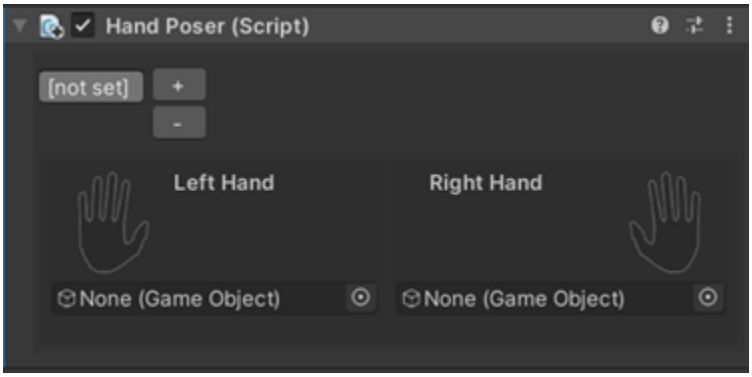
In this tutorial we will learn how to set up the Hand Poser for an interactable prefab.

Hand Poser is a mechanic contained in the MAGESPhysics module which enables grabbing objects with a specific hand posture. With the Hand Poser tool, developers can configure predefined hand postures for each object or even set more than one posture per object if needed.

It is used to interact with the physical object with an intuitive way. It is not mandatory to include it in your simulation but highly recommended for a realistic interaction system.

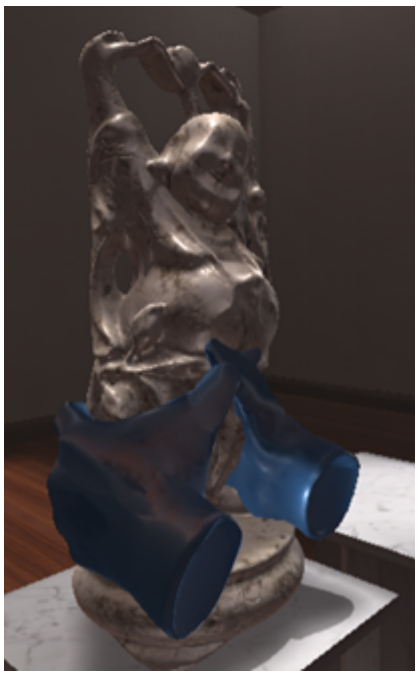
6.5.1 How to Configure Hand Poser

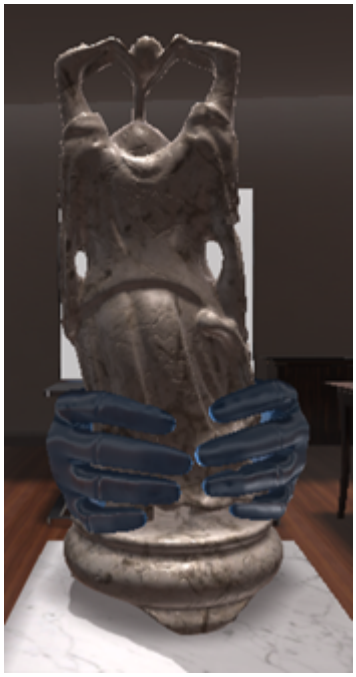
The first step is to Add the HandPoser component to your object.



This component contains two input fields for Unity prefabs, one posture per hand. To generate the postures is recommended to take an already made posture from the MAGES SDK and modify it to your needs. This posture is nothing but the hand prefab without any component attached.

Below there is an example of the left and right hands attached to our object. We will use those prefabs to save the postures for this object.





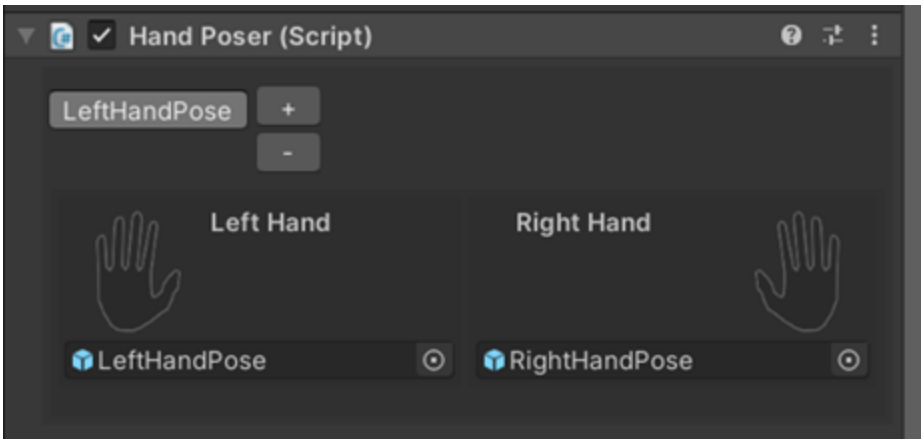
Since the left and right hands are mirrored, to generate the right hand we are inverting its y scale by assigning -1.

> **Warning:** The scale of the interactable object should be (1,1,1), otherwise the poses will be affected from the non-uniform scale values.

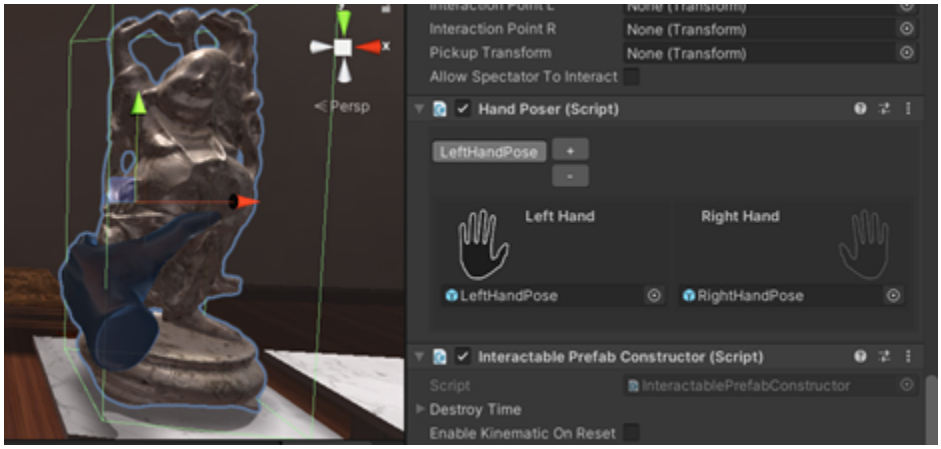
The next step is to save the postures you generated. **Both postures should be children of your object, otherwise the poser will not work properly.** Drag and drop the poses from your prefab to your assets to save posture prefab.

Below you can see the hierarchy of the Buddha model and the two postures as children.

Finally, we need to link the postures to the HandPoser component. Simply drag and drop the prefabs from your assets to the HandPoser at the object input holders (one per hand).



By pressing the hand icon on each side, the posture will be spawned on the object. In this way you can verify if the postures are implemented correctly.



Note: However, remember to delete those spawned postures by pressing again the hand icon, otherwise they will be visible once the object will be instantiated.

6.5.2 Configure Multiple Postures for an Object

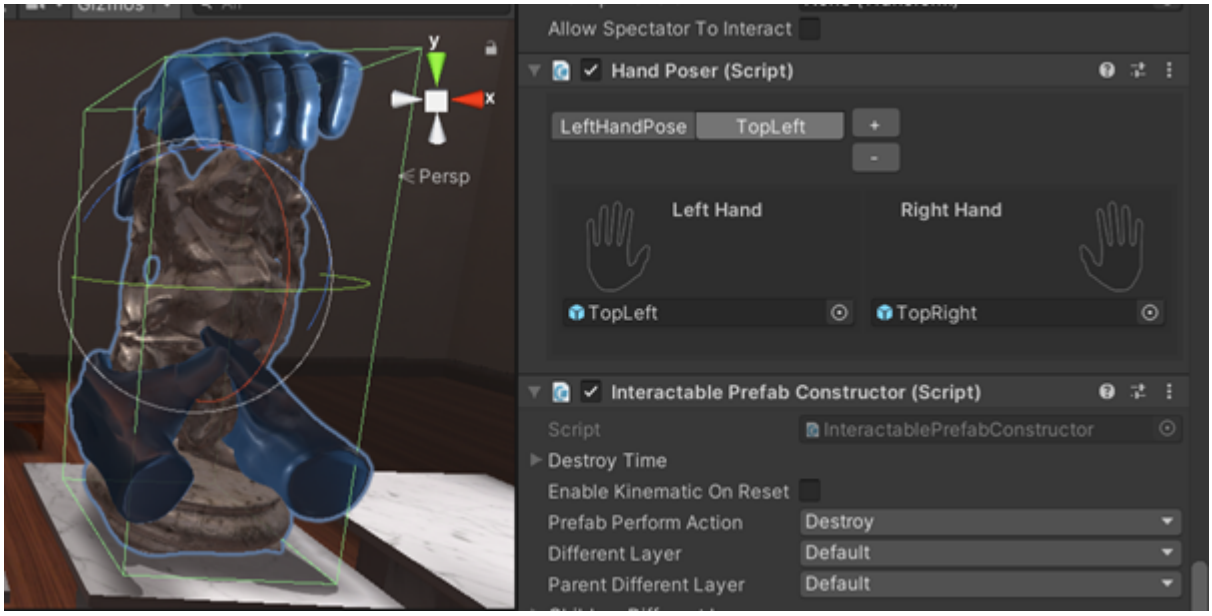
Specific objects can be grabbed in multiple ways. For this reason, Hand Poser supports the configuration of multiple postures.

To enable this feature, from the HandPoser script press the "+" button on the top left, in this way more postures can be added.

Each posture has a pair of hands (left and right). To delete a pair of postures select it and press the "-" button.

Note: If an object has more than one postures, the poser calculates the distance of the player's hands related to the object to enable the most suitable posture (distance based).

Example of multiple postures on an object:

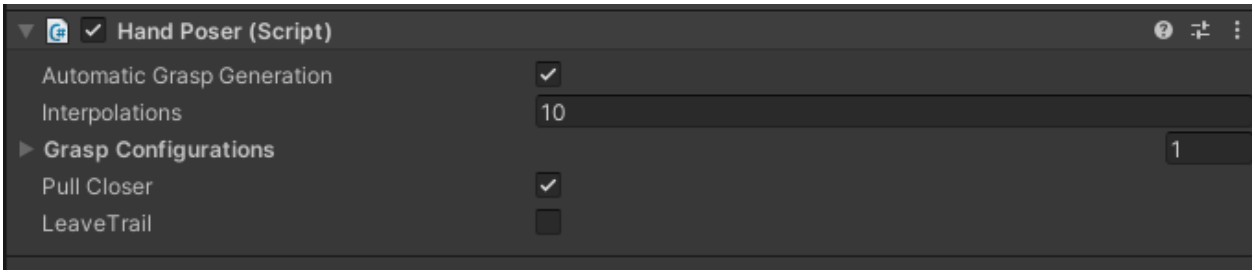


In this case we have a pair of poses at the bottom of the statuette and another pair at the top. The hand poser will select the more suitable pose at the beginning of the interaction.

6.5.3 Automatic Grasp Generation

In order to save time and effort spent on this task, we developed a feature to automate this procedure. When there is no need for a precise hand posture, but a realistic grasping is still needed, this option could be a decent solution.

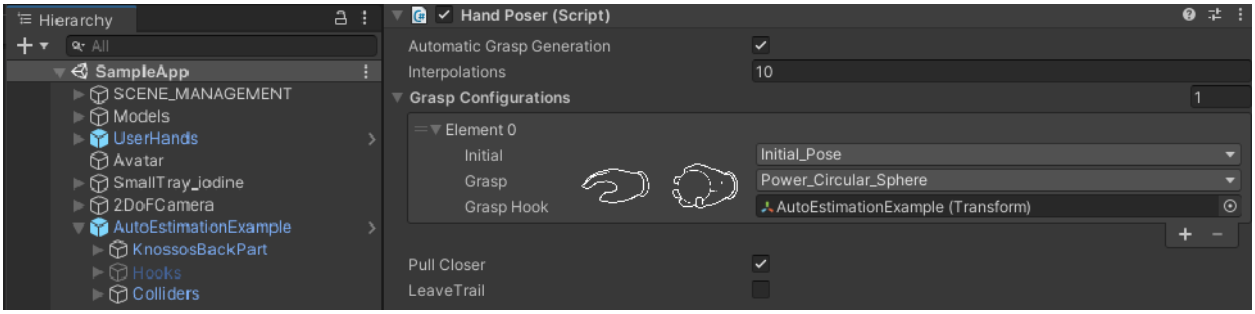
On hand poser editor there's an option for automatic grasp generation, if toggled real-time hand pose estimation will be enabled.



The implementation is imitating the human intuitive way of grasping. Human brain classifies objects according to shared features into certain grips. In order to replicate this, we have two sets of poses to choose accordingly and achieve a better result, one containing the initial hand poses and the other the final grips.

Objects often have abstract shapes. When setting up this type of object you'll have to define different spots, or else hooks, with different final grips. First, you'll create an empty game object which will be the hook for a particular area. Finally, you will choose a corresponding grip for the hook, which will be applied when trying to grasp an object from all the points near to it.

Following the same steps you can have only one final grip for object by just adding at the grasp configuration only one hook, and that would be the pivot of your object.



Interpolations A certain number of interpolations between the initial and final poses is computed. For each interpolation, collisions between bones and target objects are checked. The larger the number the more collision checks you'll have.

> **Warning:** The number of interpolations influences the performance. A minimum of 10 interpolations is required in order to avoid problems. Indeed, if the interpolations are not enough, mesh penetration issues can arise because collisions are detected with a too noticeable delay.

Pull Closer The interactable item is pulled towards the grasp center when trying to grab it from an unreasonable distance.

Leave Trail When it's on, it leaves a temporary trail, only when running from inside the editor, and you can see the closest interactable point detected and change your colliders accordingly to have a more accurate hand lock.

Note: Automatic hand pose estimation functionality is based on the proper colliders' setup of the interactable item. That's because collision detection is tracked using rigid bodies and bounds. Using primitive colliders improves performance, but possibly they won't fit perfectly with the shape of the object. In order to detect unwanted collisions, you can use this option for debug in such cases.

6.6 Import Existing MAGES™ Project to Different Platform

A MAGES SDK project can easily be transferred between macOS and Windows platforms.

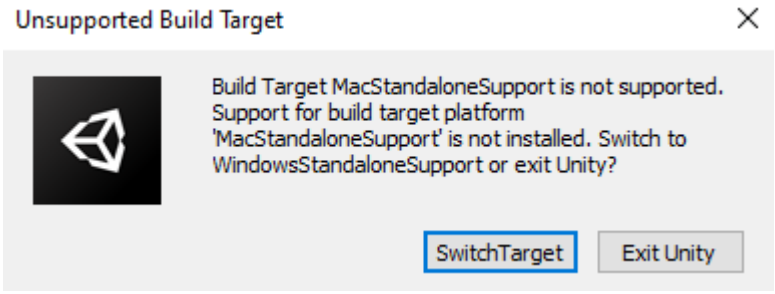
To transfer such a project from macOS to Windows, for instance, kindly follow the steps below:

1. Save and close your Unity project on your macOS machine.
2. Locate the project folder, and compress it for easier transfer.

   > **Note:** In order to compress a folder on macOS, right-click on the folder and then select "*Compress {Folder Name}*".

3. Transfer the compressed folder to your target Windows machine.
4. Decompress the folder.
5. Open Unity Hub, and click *ADD*, located in the top right corner of the Unity Hub window.
6. Navigate to the decompressed project folder and select it.

Note: A window will appear, notifying that the Mac OS X target platform is not supported. Simply click *Switch Target*.



7. Once the importing is finished, Unity will open the project, and you can continue your work.

Note: Your SDK license login account will be retained between the platforms (in case you did not log out before the project transfer). Since Windows platform supports VR, you can add VR support to your project while on Windows.

6.7 Actions with deformable skinned meshes

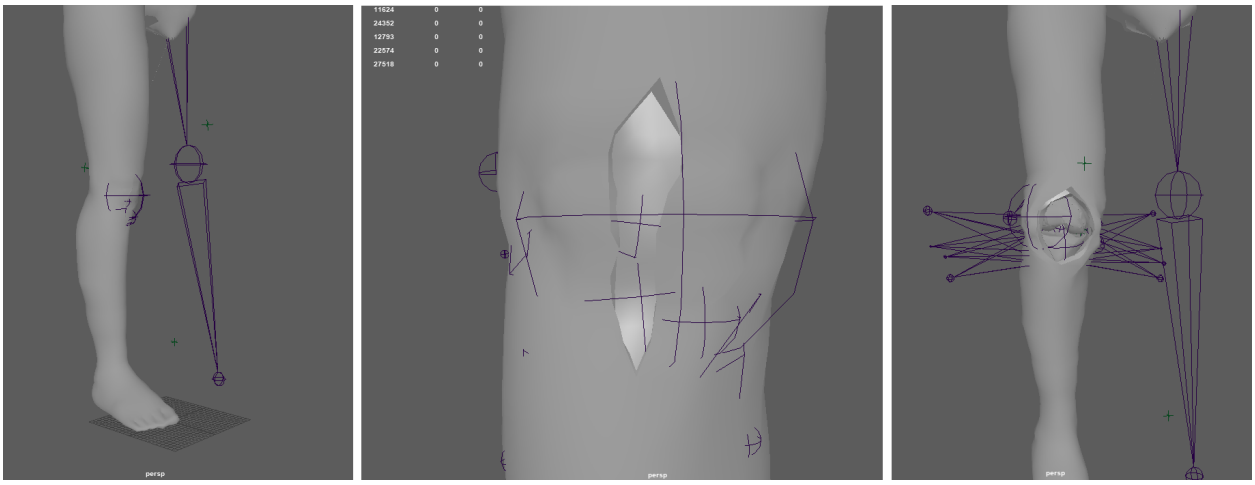
In this tutorial we will demonstrate the complete pipeline on creating an Action with a deformable skinned mesh. From the 3D model to the unity importing then all the configurations and finally the Action script. We will implement a medical example, in particular the initial incision from the Total Knee Arthroplasty operation.

Tutorial overview:

1. Generate the animated 3D model
2. Import the 3D model to Unity
3. Split the animation into clips
4. Animator setup
5. Configure the CharacterController
6. Generate the Action prefabs
7. Implement the Action script

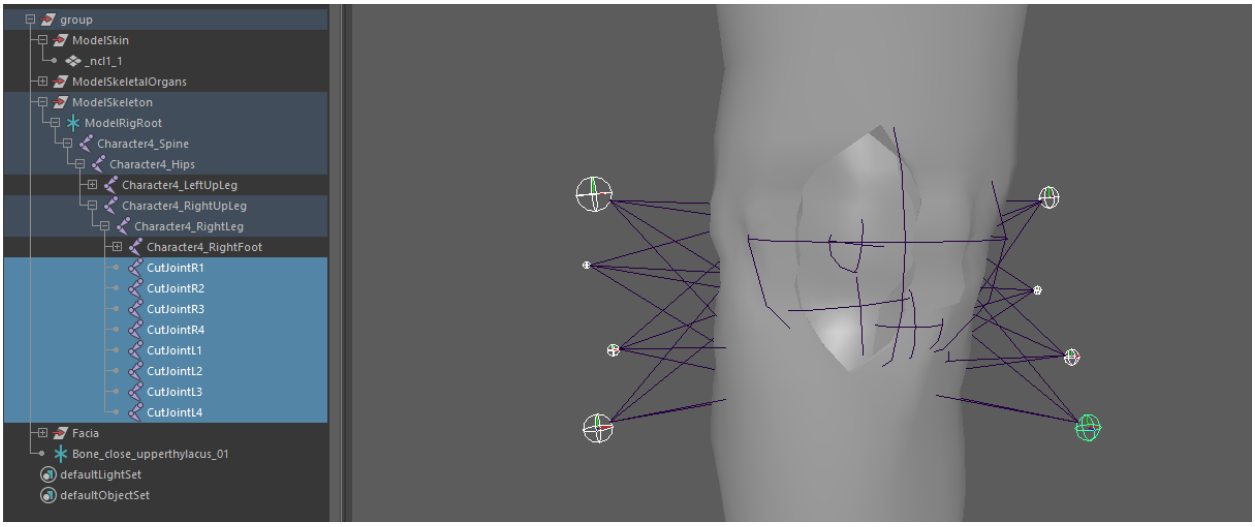
6.7.1 Generate the animated 3D model

In this scenario we will use an animated leg, with various baked skinned deformations. The image below shows our 3D model in Maya.



As you can see, in Maya we design the full animation using standard joints and key frames. The animation process depends on the scenario you want to design. In this case, we rigged the right foot adding joints to cut the skin and the different muscle layers until we have a clear view of the knee.

The image below shows the joints we used for the skin animation along with their key frames.

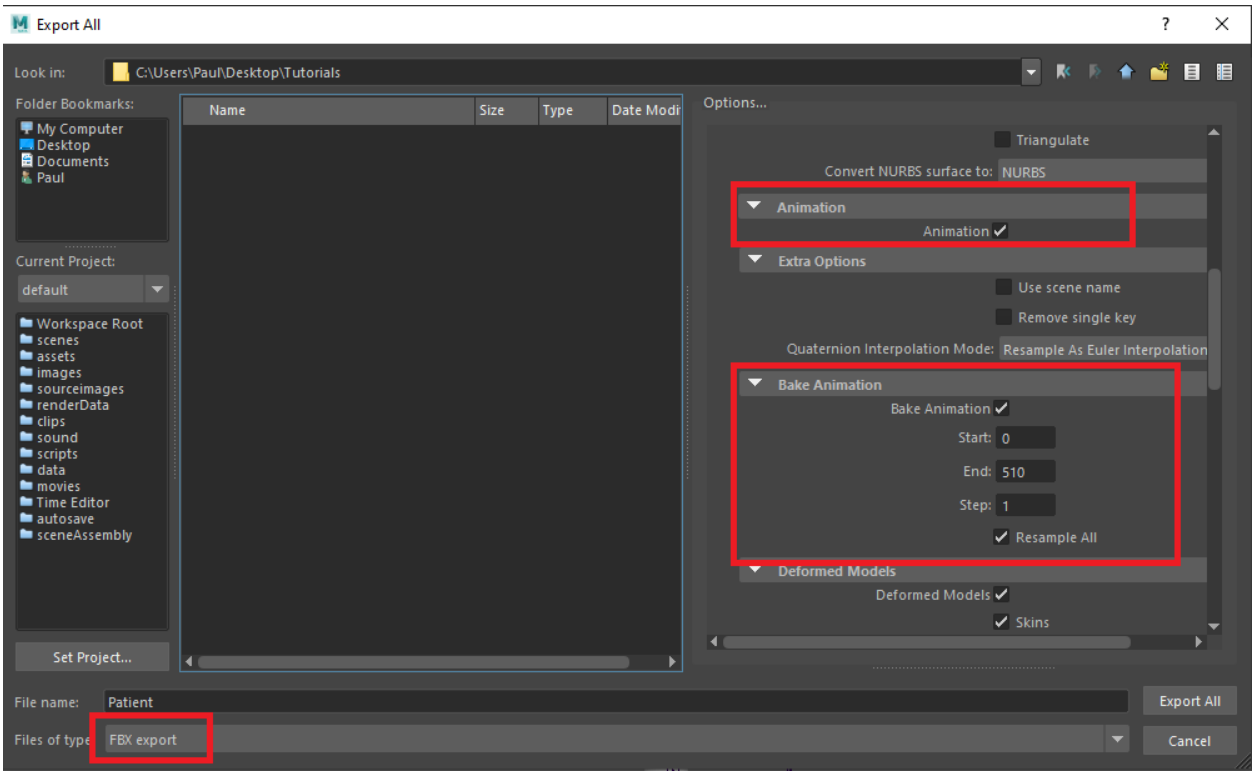


Now we have to export the 3D model. We use the FBX format to export our 3D models since it is the most reliable format to work with unity.

From the Maya menu bar navigate to **File/Export all**. Now you need to configure some options first. Make sure the Animation option is checked to export the animation as well. Make sure the **Files of type** option is set to FBX export.

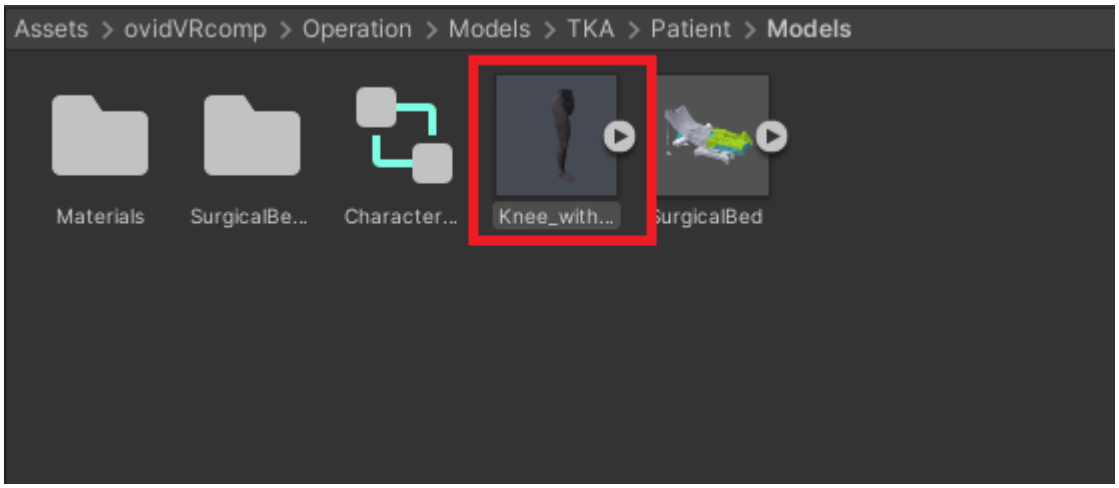
There are specific situations where you need to bake the animation instead of exporting it without the bake option. If the animation has complex animation behaviors or blends there is a chance that they will not import right in Unity. If you face some issues on the importing and the animation does not appear to work right check the Bake Animation option at the export window.

Warning: If you bake the animation, Maya will generate key frames for all joints. After that, the modifying the animation would be really inconvenient. Remember to keep the Maya binary file to modify the animation when baking.

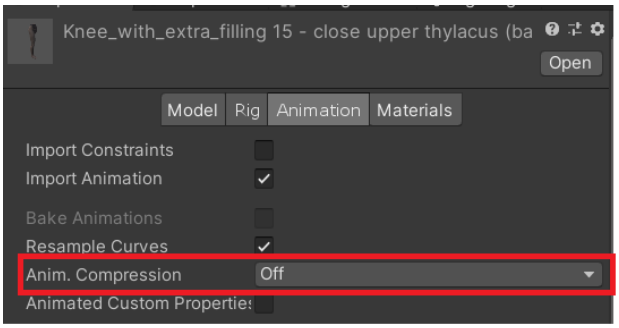


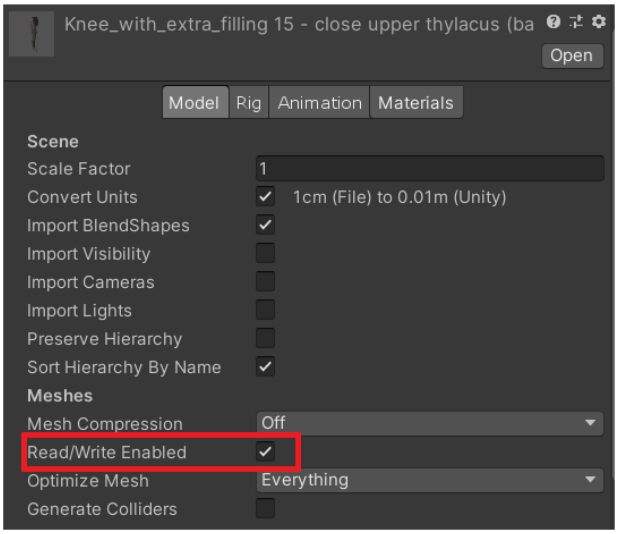
6.7.2 Import the 3D model to Unity

To import the model into Unity just drag and drop the FBX to your project, as seen below.

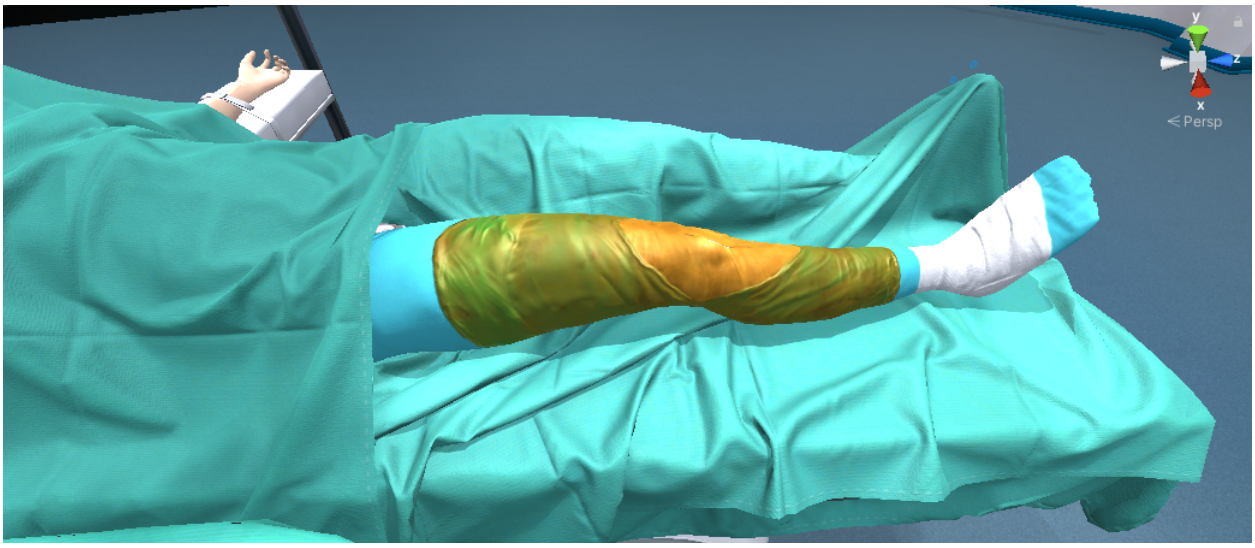


Click the FBX model and from the inspector navigate to the model tab and make sure the Read/Write Enabled is checked. Also in the Animation tab turn off the Animation Compression. Animation compression may cause unwanted animation behavior in certain situations.





This is the final result



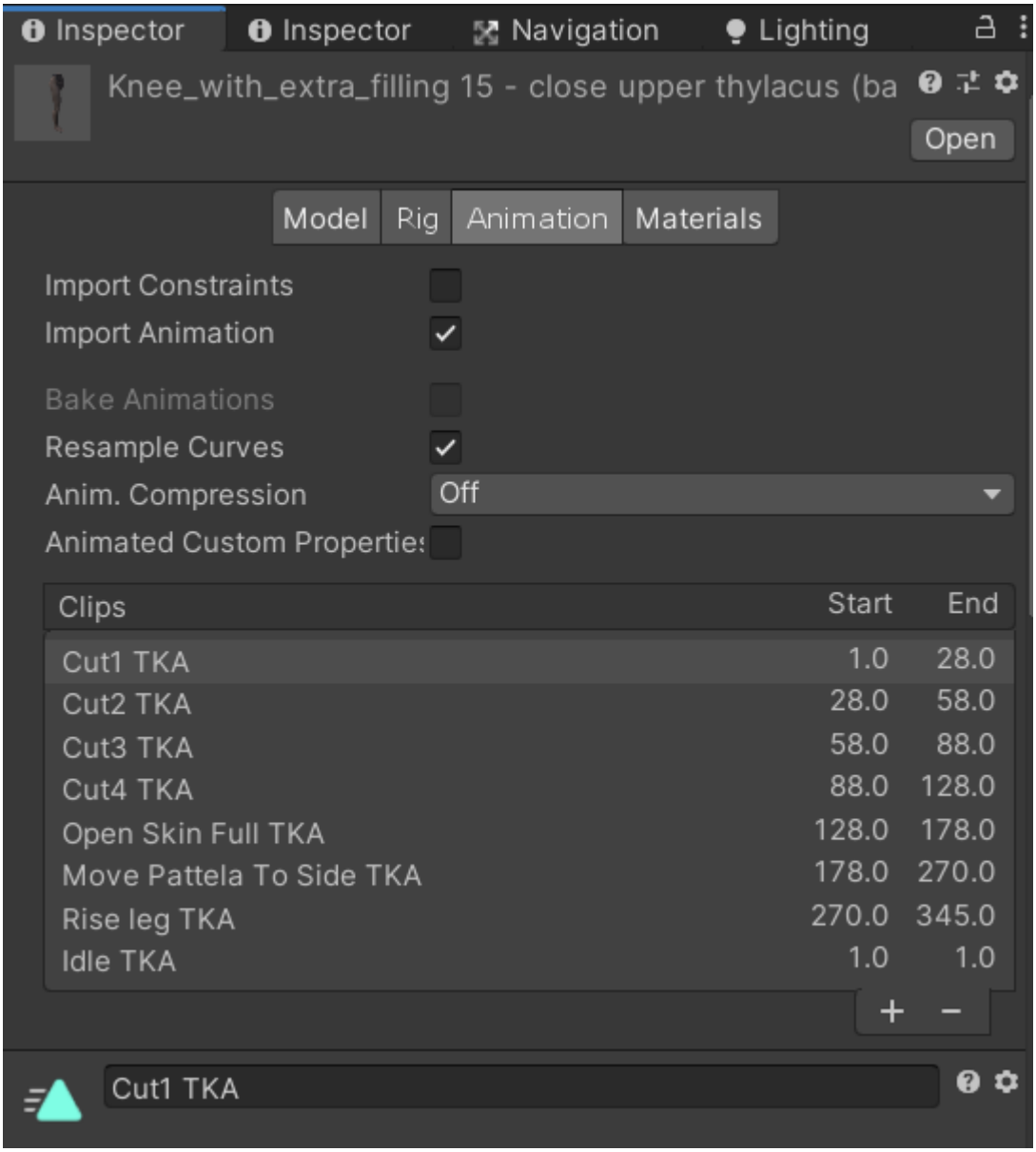
6.7.3 Split the animation into clips

We will split the animation into smaller clips to feed our CharacterController, a class that controls our patient's animations.

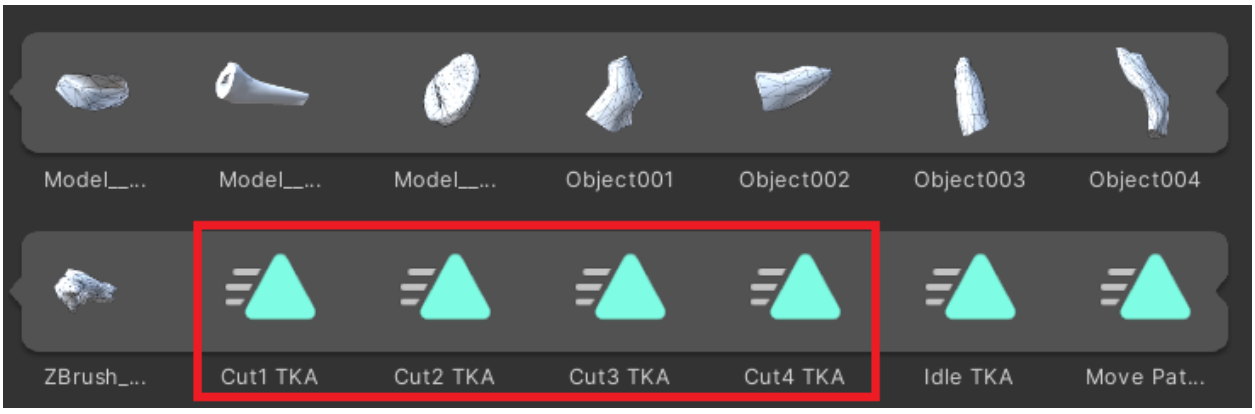
To split the animation into smaller clips, click on the FBX, then at the Inspector navigate to the Animation tab. At the Clips window click the "+" icon to add a new animation clip and then select the Start and End point from the original animation. In our scenario we will split the four parts of the initial incision animation (Cut1 TKA, Cut2 TKA, . . .)

To save your changes click the "Apply" button at the bottom of the inspector.

Below you can see the clips we made for this operation. In this tutorial we will use the first four.



The clips will also appear within the FBX model

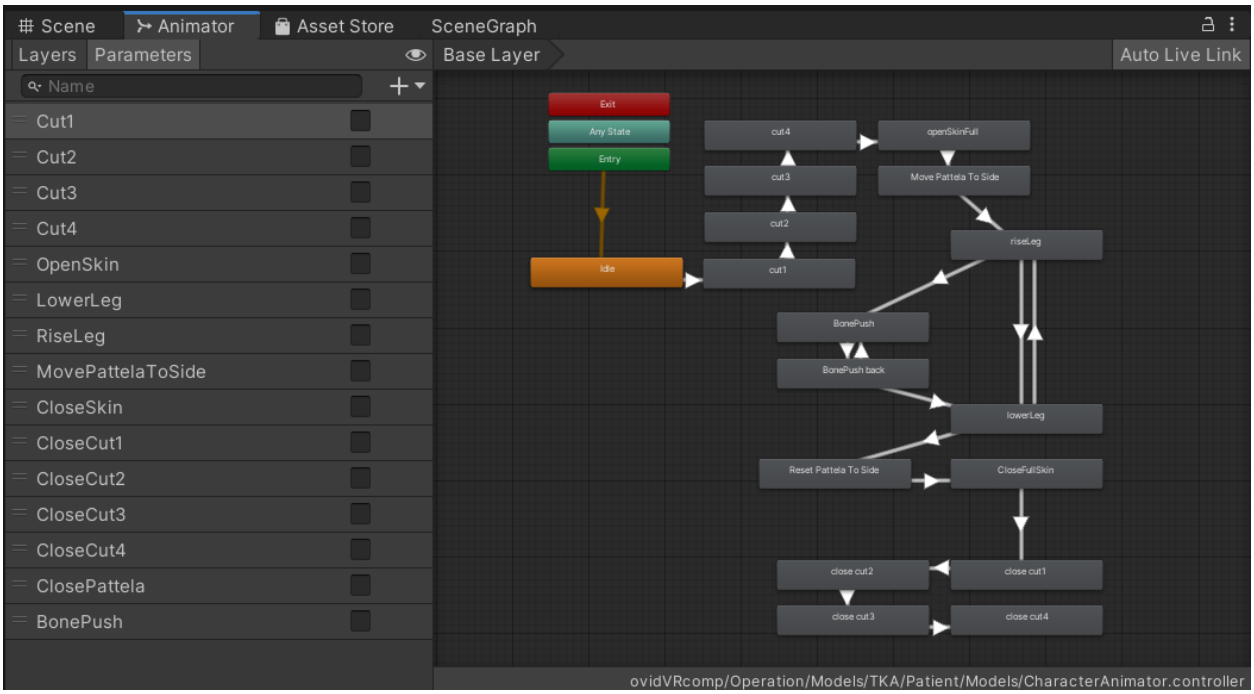


6.7.4 Animator setup

To properly execute the animations and manage their transitions we have to set up an Animator for our model.

From the Unity scene, navigate to your prefab (in this case the leg), then from the Window menu on the top bar select **Animation/Animator**. Here you need to configure your own Animation to match your needs. In this operation we designed a sequential state Animator since our animations will always play in the same linear order.

Below you can see our Animator.



Note: We are using Booleans to traverse the animation states.

6.7.5 Configure the CharacterController

The next step is to setup the CharacterController that will control our Animator. This step is optional, you can play your animations in your own way, but we figured out this was the most convenient solution for our needs.

CharacterController is a **Singleton class** attached on our animated prefab. You can find this script in our SampleApp project. The basic idea is to have a script that controls the Animator and more specific the boolean parameters to switch between the states.

In the CharacterController you can see functions as the following

```
public void PlayCut1Animation()
{
    animator.SetBool("Cut1", true);
}

public void PlayCut2Animation()
{
    animator.SetBool("Cut2", true);
}
```

In this way we can call the following method from an Action script to play the Cut1 animation

```
CharacterController.CharacterController.instance.PlayCut1Animation();
```

6.7.6 Generate the Action prefabs

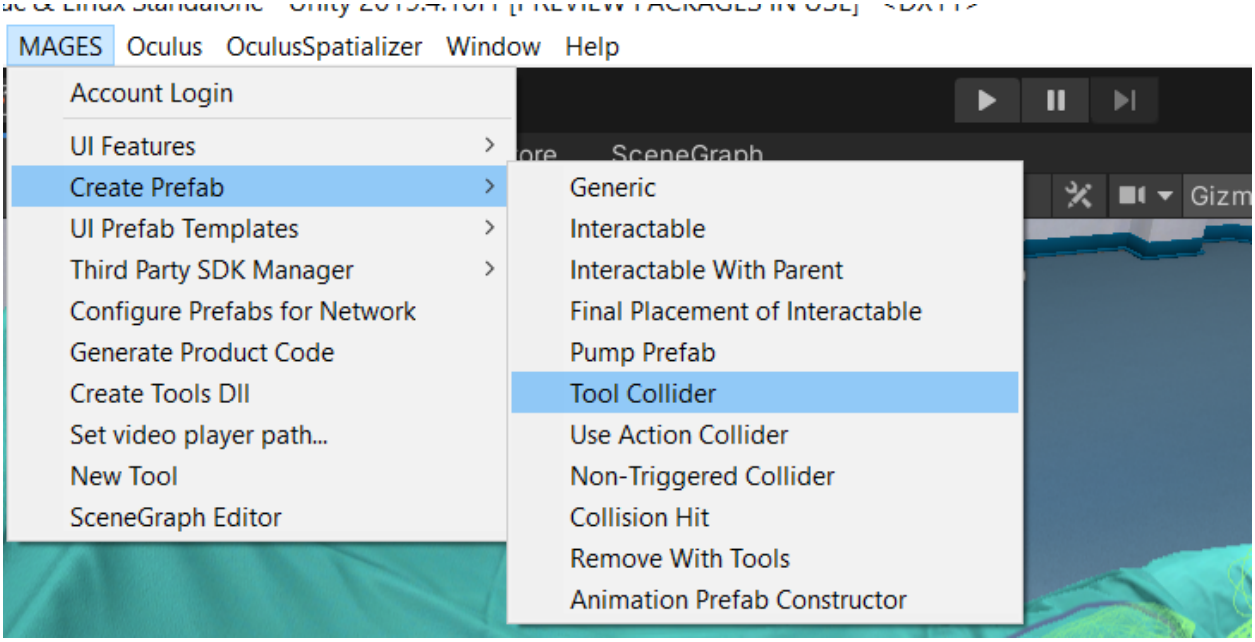
As we mentioned, the MAGES metaphor of the skin incision will be a ToolAction. We use the ToolAction in cases where we need to grab a certain tool (pliers, saw etc) to complete an objective by "touching" the tool on a predefined area.

For the ToolAction we need to prepare the following prefabs

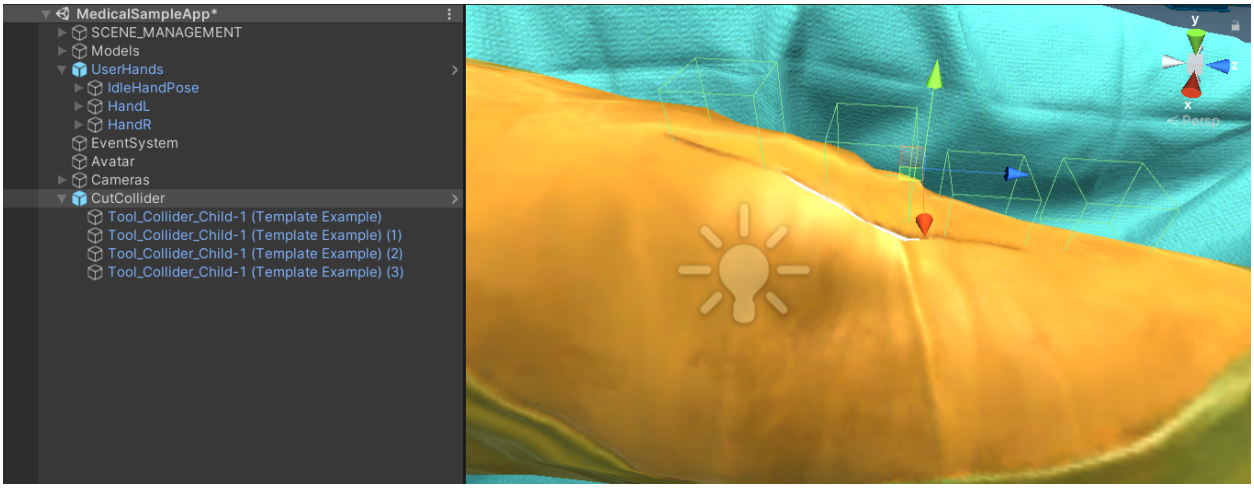
1. Tool Collider
2. The actual tool (a scalpel)

Tool Collider

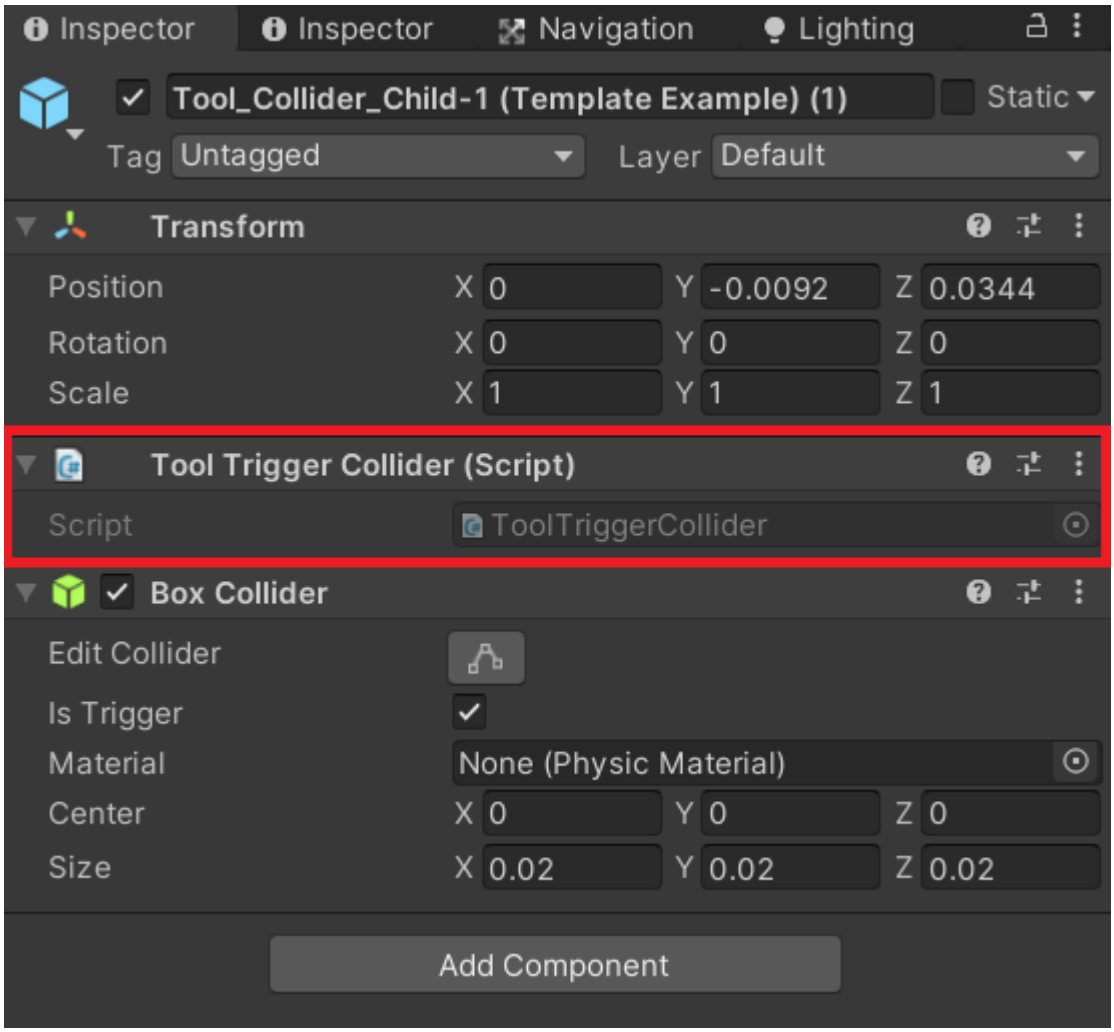
To generate the tool collider navigate to **MAGES/Create Prefab/Tool Collider** and the tool collider prefab template will appear.



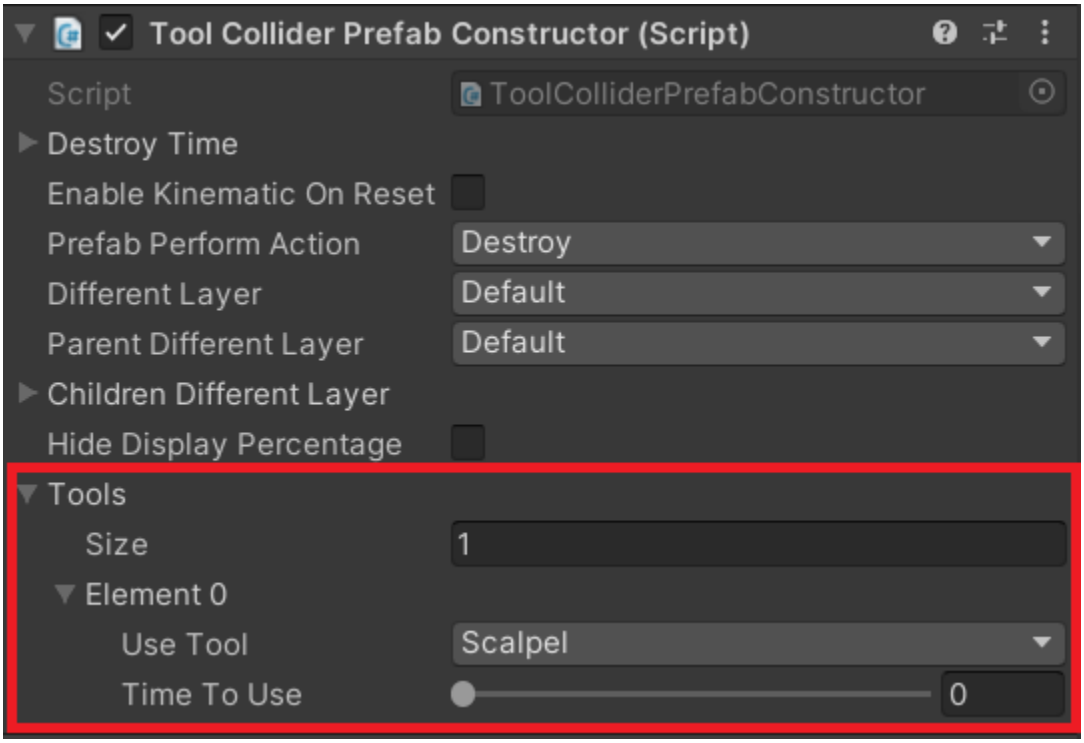
The tool colliders are waiting for the tool to trigger them. Once all the colliders are triggered the Action will perform. In this scenario we will use four colliders since we have four cut animations. The image below shows our tool colliders. Pay attention that all the colliders have the same parent.



Warning: All the tool colliders should have the `ToolTriggerCollider` script attached.



Now we need to assign that only the scalpel will trigger these colliders. To do that, navigate to the root object and at the ToolColliderPrefabConstructor add a new tool element at the Tools list, then select the scalpel (image below)



Tool prefab

The next step is to configure the tool that we will use for this scenario, the scalpel. In this case, the scalpel is already configured, so no additional actions are needed.

Here you can find more information on how to generate your own tools.

6.7.7 Implement the Action script

The final step is to write the Action script. Below you can see the basic ToolAction script that describes this ToolAction.

```
using MAGES.ActionPrototypes;

public class SkinIncisionAction : ToolAction
{
    public override void Initialize()
    {
        SetToolActionPrefab("Medical/Lesson0/Stage0/Action1/CutCollider", MAGES.
↪ToolManager.tool.ToolsEnum.Scalpel);

        SetPerformAction(MAGES.CharacterController.CharacterController.instance.
↪PlayCut1Animation, 1);
        SetPerformAction(MAGES.CharacterController.CharacterController.instance.
↪PlayCut2Animation, 2);
        SetPerformAction(MAGES.CharacterController.CharacterController.instance.
↪PlayCut3Animation, 3);
        SetPerformAction(MAGES.CharacterController.CharacterController.instance.
↪PlayCut4Animation, 4);
```

(continues on next page)

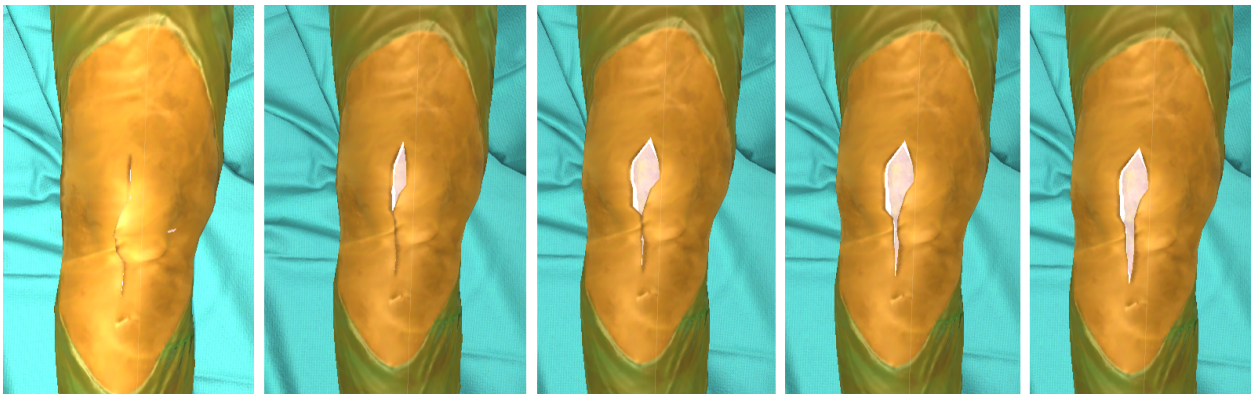
(continued from previous page)

```
        base.Initialize();
    }
}
```

Let's explain the Action a bit. You can see that we call the `SetToolActionPrefab` method to instantiate our tool colliders and set the tool that we will use to trigger them (`MAGES.toolManager.tool.ToolsEnum.Scalpel`)

The `SetPerformAction` method takes a Unity Action (method) as the first argument and an integer as the second. The second argument indicates the trigger callback ID. For example when the user hits the first collider with the scalpel, the `PlayCut1Animation` method will invoke. Hitting the second collider will invoke the `PlayCut2Animation` method and so on.

Below you can see the cutting animation playing when triggering the tool colliders.



Note: To perform the ToolAction, the user needs to hit all the registered tool colliders.

6.8 Soft Bodies

6.8.1 Upgrading from previous version

The new version of MAGES offers a more stable and easier to use softbody simulation. All the different scripts and versions have been replaced by only one script called **SpringMassSoftbody**. In order to update from previous version, attach the new script in the original model and use the same values as the old one for each field. More details about each variable can be found below or in the *Softbodies manual*.

6.8.2 Introduction

The MAGES SDK supports real time soft bodies as a part of its standard features

Note: Soft bodies are used to simulate soft and deformable surfaces like the human skin, liquids, internal organs, cloths and any other deformable surfaces.

You can read *here* more information about our soft bodies.

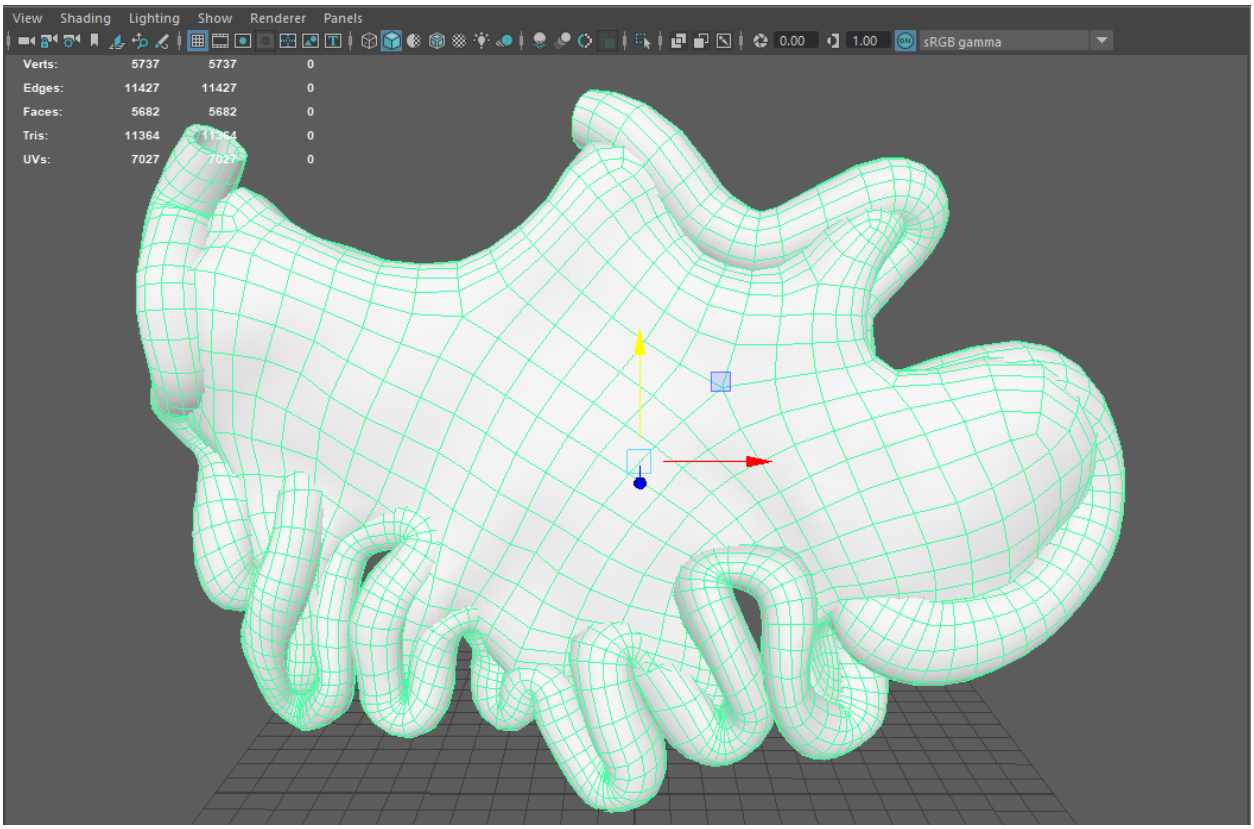
In this tutorial we will demonstrate how to set up the soft body behavior on a human bowel model (small intest and mesenter).

Tutorial overview:

1. Exporting the model from Maya
2. Unity importing
3. Soft body setup
4. Run-time and interaction
5. General discussion

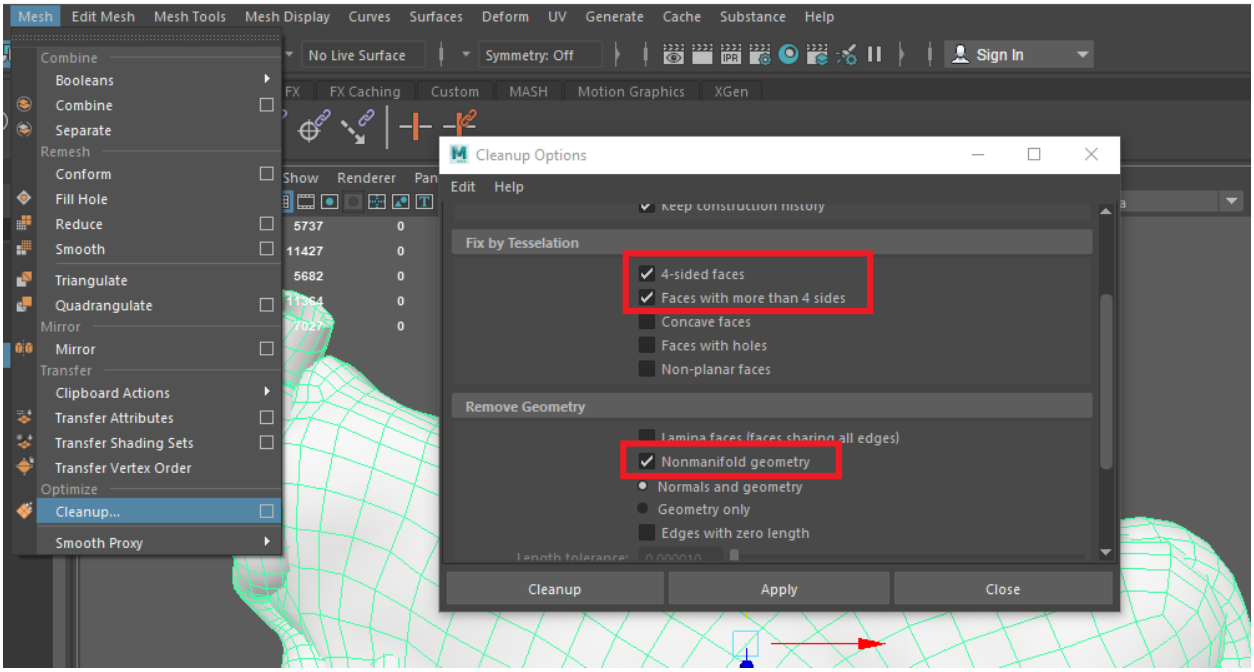
6.8.3 Exporting the model from Maya

The image below shows the bowel model in Maya.



This model has 5700 vertices. The number of vertices is really important in our soft body system both for the real-time simulation updates and the preload configuration and caching preprocessing time. Both operations (pre-processing and real-time updates) all depend heavily on the number of vertices under processing

Make sure to do a cleanup before exporting the FBX. In the cleanup window select the **4-sided faces**, **Faces with more than 4 slides** and the **Nonmanifold geometry**.

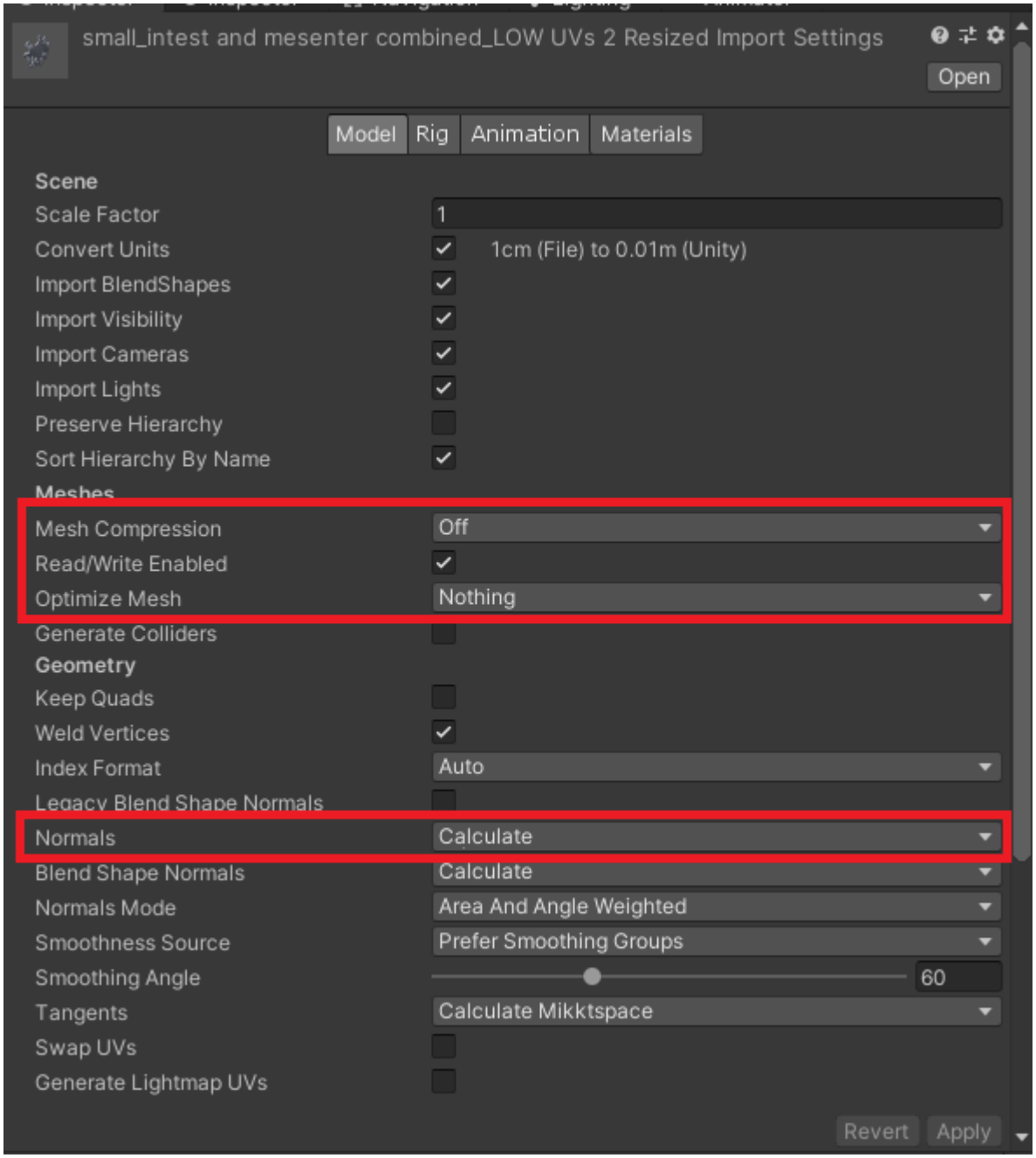


Then export the model using the FBX format.

6.8.4 Unity importing

When you import the model to Unity make sure you check the following parameters:

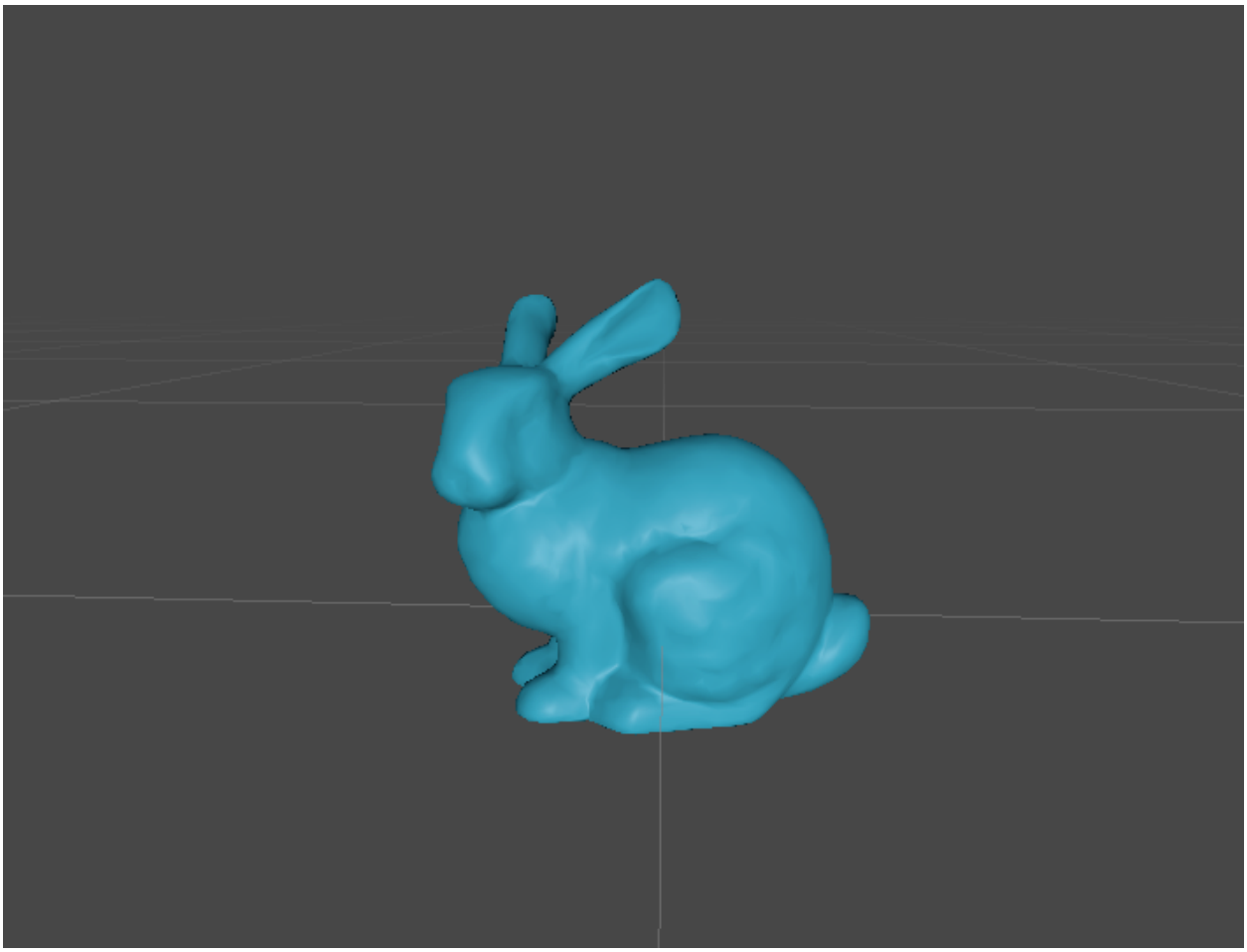
1. **Mesh compression** should be **off**
2. **Read/Write Enabled** should be **checked**
3. Set **Optimize Mesh** to **Nothing**.
4. Set **Normals** to **Calculate**.



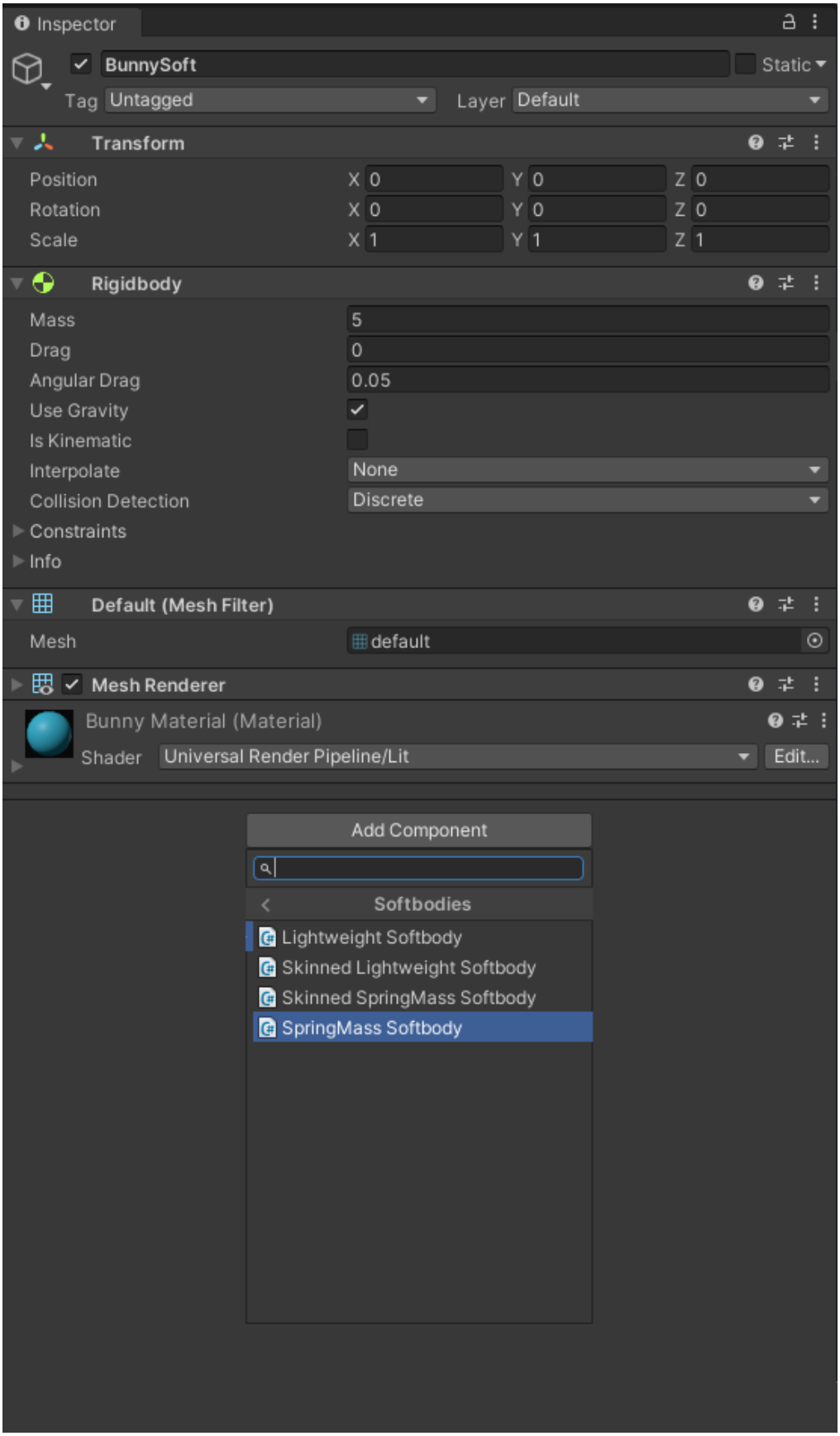
If you face any other issues with the model e.g missing vertices when starting the application, consider reviewing those configurations. **The underlying geometric topology plays an important role in the final quality of the simulation.**

6.8.5 Softbody setup

Drag and drop your FBX model in the unity scene. In this example I will be using the Stanford bunny model.



Right click on the object in hierarchy and unpack the prefab completely. Then attach the **SpringMassSoftbody** script to it.


Inspector
BunnySoft
Static
Tag Untagged
Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Rigidbody
Mass 5
Drag 0
Angular Drag 0.05
Use Gravity
Is Kinematic
Interpolate None
Collision Detection Discrete
Constraints
Info
Default (Mesh Filter)
Mesh default
Mesh Renderer
Bunny Material (Material)
Shader Universal Render Pipeline/Lit
Edit...
Add Component
Softbodies
Lightweight Softbody
Skinned Lightweight Softbody
Skinned SpringMass Softbody
SpringMass Softbody

The initialization process consists of 2 steps:

1) Initialization of the Softbody

2) Saving the generated mesh and object for future use.

We will see more details for each step below.

Initialization

The most important part of this step is to configure the Clustering options consisting of the **Particle Distance**, the **Particle Connection Distance** and the **Skinning Distance**. All the other parameters can be changed on the fly and upon experimenting with the softbody in game.

Note: **Particle Distance** is the minimum geodesic distance between two particles.

Particle Connection Distance is the max distance two particles can have in order to be connected together. (Meaning that, in game, disturbing one particle will also affect all its connected particles)

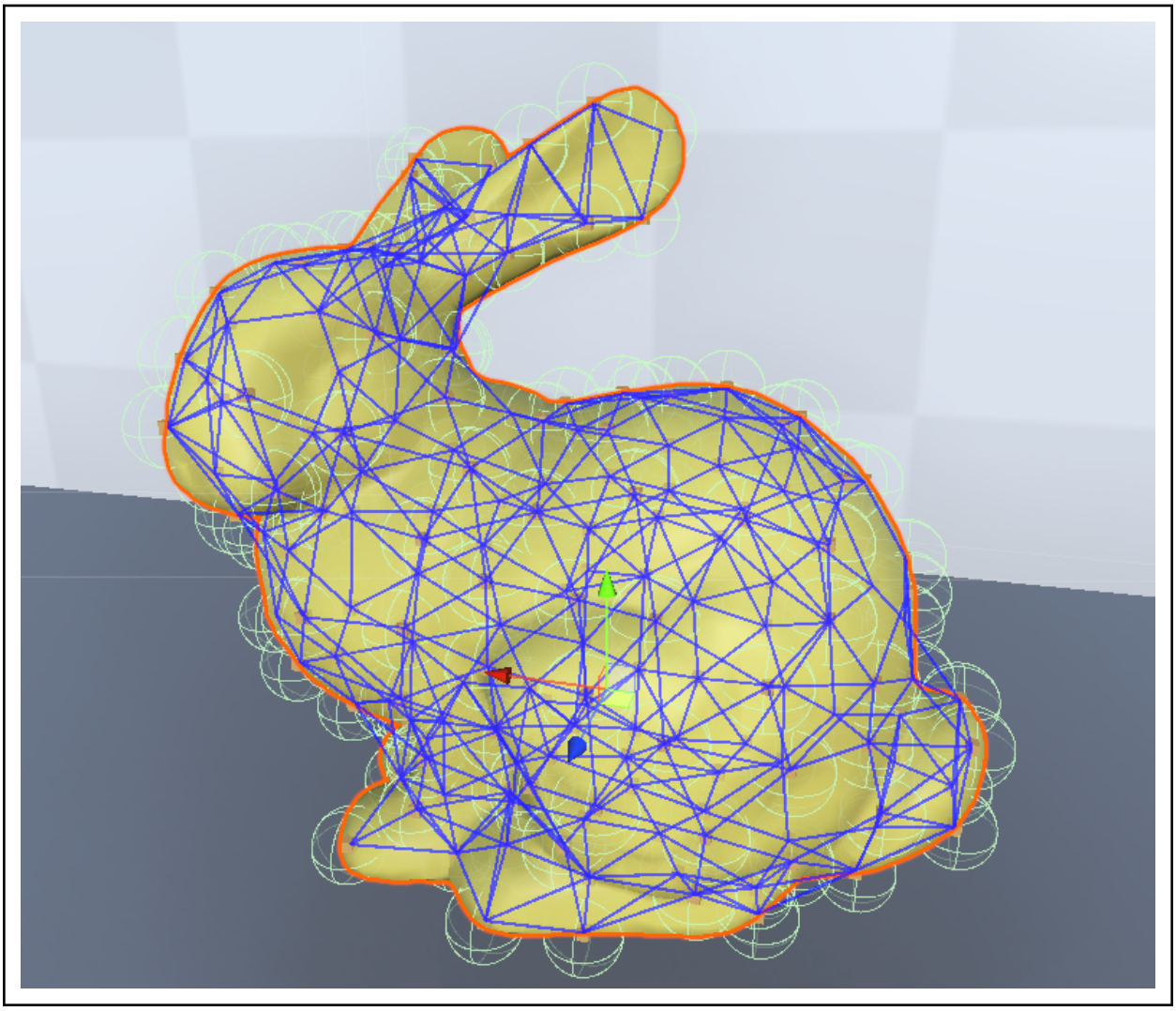
Skinning Distance is the maximum distance a vertex can have from a particle in order for it to be affected.

Keep in mind the smaller the particle distance is, the most particles will be created and therefore the simulation, although more realistic, will be more computationally expensive.

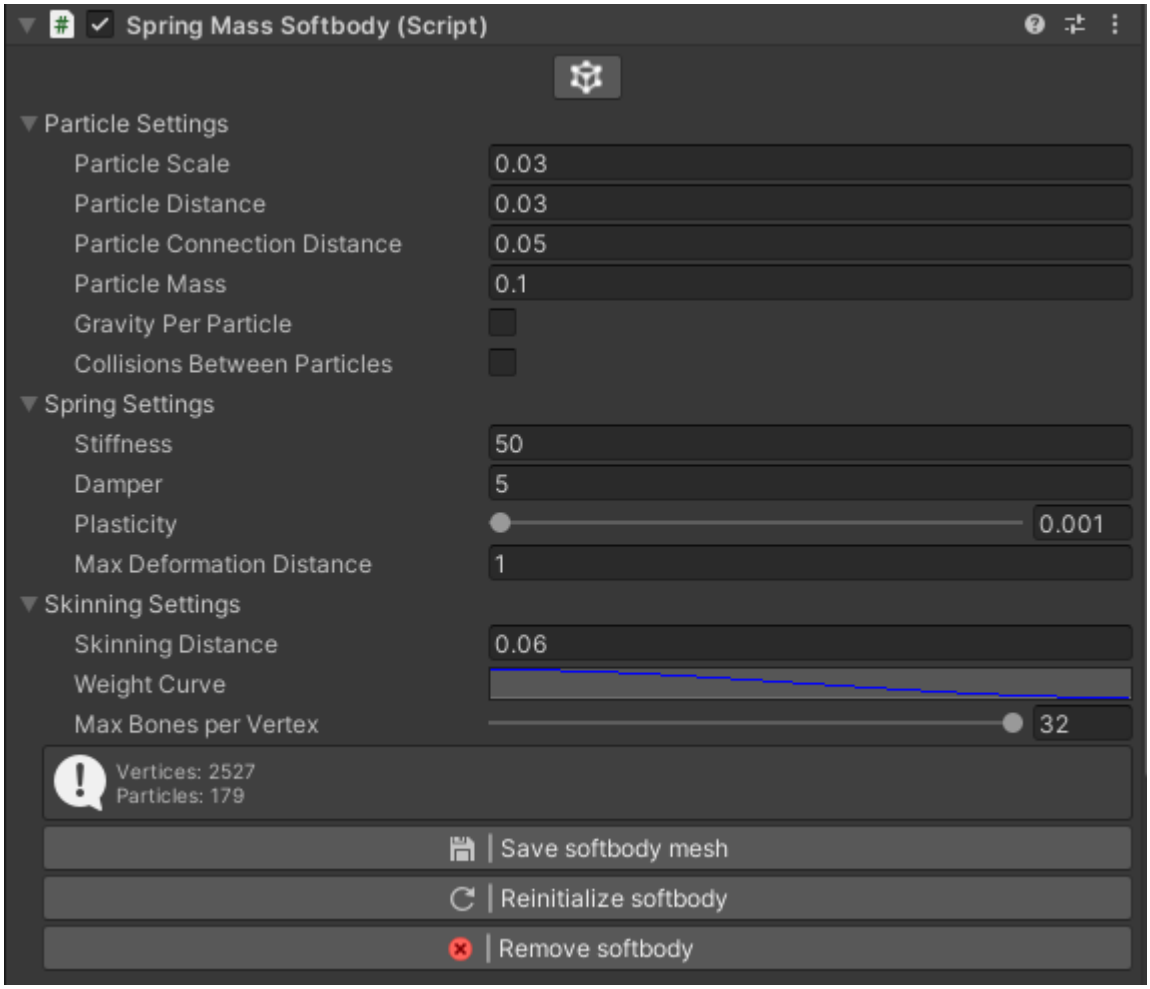
Warning: After choosing the desired Particle Distance and Particle Connection Distance we must ensure that all our particles are connected. Too many connections will make the simulation run slower and too little will make parts of the mesh unstable or detach completely.

We can visualize and check our connections by pressing the **Show Spring Connections** button in the inspector.

If we have now Gizmos enabled we must see a result similar to this. (In order to make lines visible we can disable the mesh renderer of our object temporarily)



Here are all the settings used for the Bunny Model:

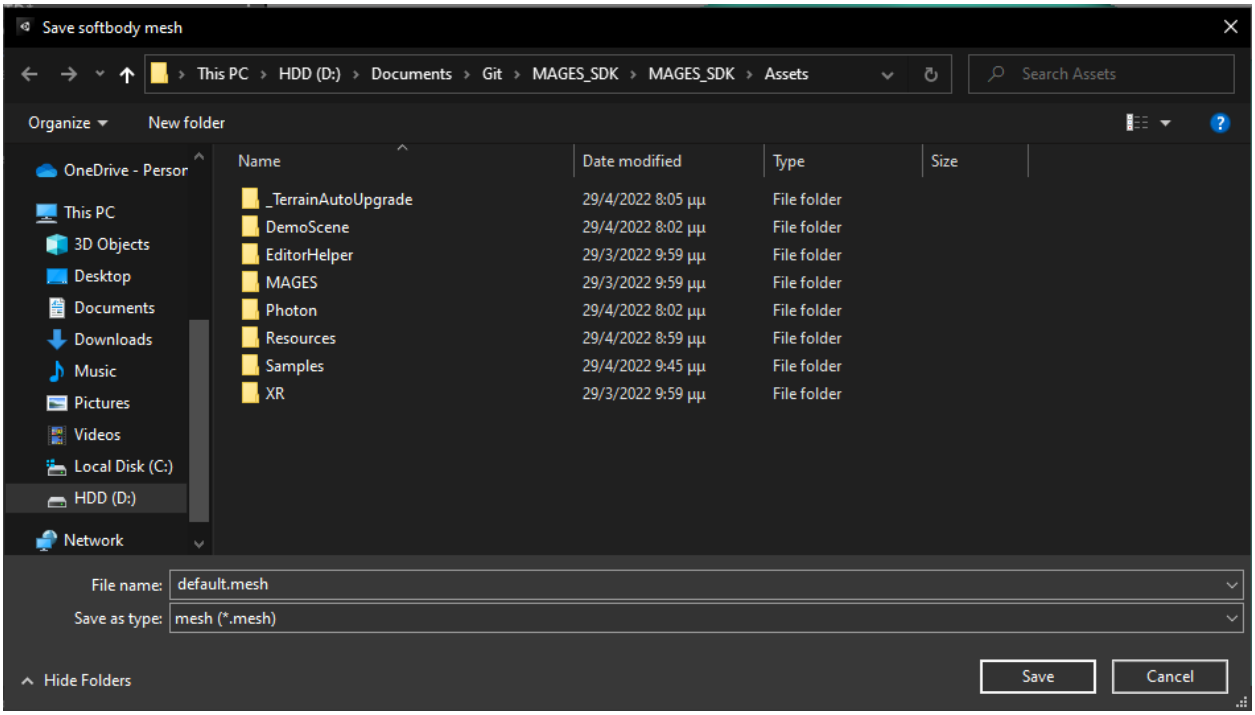


Upon choosing the appropriate values we must click the **Initialize Softbody** Button. When the Simulation is Initialized successfully a message will appear in the Console Log. If we need to change the values again we need to press the **Reinitialize Softbody** Button.

Saving the Softbody

In this step we will save the mesh as an asset and our object as a prefab for later use.

We press the **Save Softbody Mesh** button. If the mesh is not already saved, this will open the window shown below:



We select the folder we want to save the mesh, fill the name of the mesh asset and press Save.

> **Warning:** Keep in mind that the folder you select to save the mesh must either be the Assets folder or a subfolder of it.

We can now simply drag and drop our object in the Assets and create a Prefab. We can later use the prefab as many times as we want without needing Initialization again.

6.8.6 Run-time and interaction

When entering play mode the simulation starts immediately. The user can grab any particle and stretch or hold the object.

6.8.7 General discussion

Soft bodies introduce a powerful way to simulate deformable surfaces. The MAGES SDK offers a plug-and-play solution for particle-based soft bodies in Unity, ideal for medical or any other use.

It is important to consider that real-time soft body simulation uses advanced physics algorithms to calculate the object's movement per vertex, paying respect to its original shape. This is why you need to be extra careful with the script configuration as well as the model's topology for best results.

6.9 Questionnaire

In this tutorial, we will learn how to set up and develop a questionnaire for gathering information from respondents.

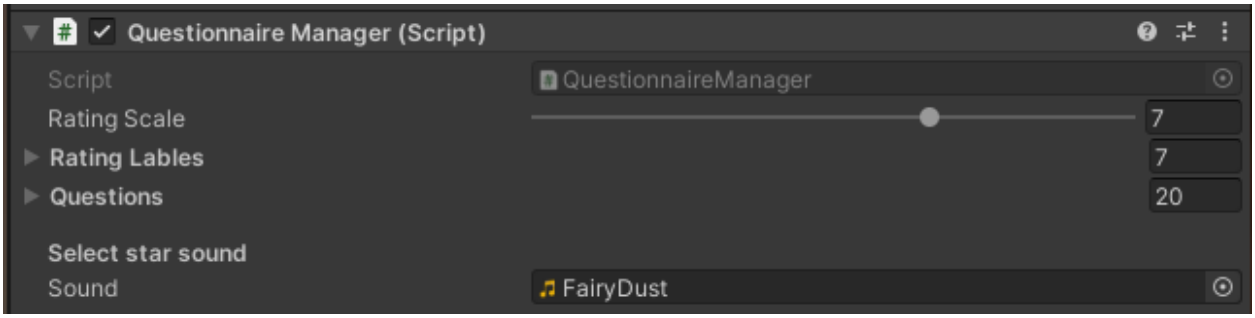
6.9.1 Evaluation Survey Template

We provide an evaluation survey template for measuring the sense of presence experienced in a virtual environment. This template can be used as a prototype and form a basis for other custom questionnaires.

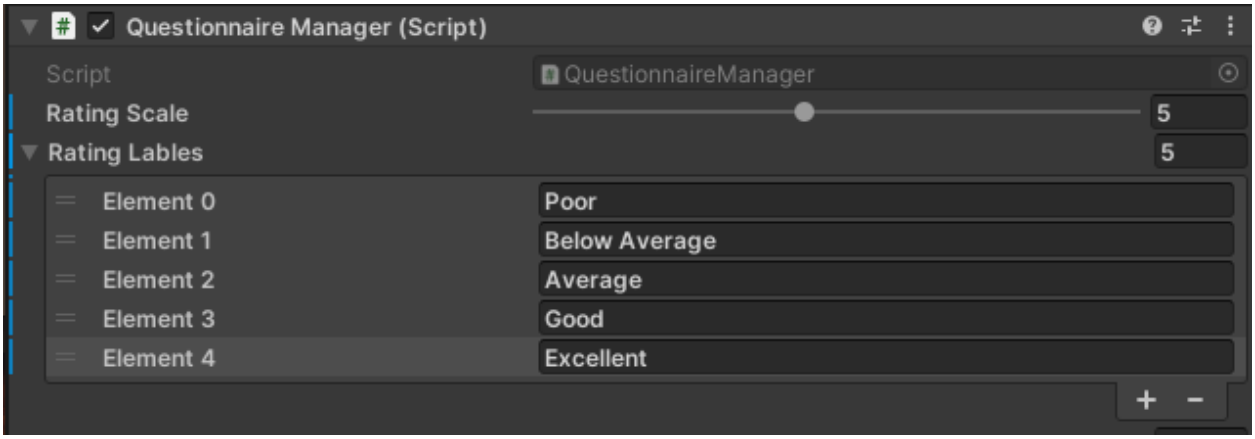
The questionnaire is placed at the end of the simulation, under the "Survey" option.

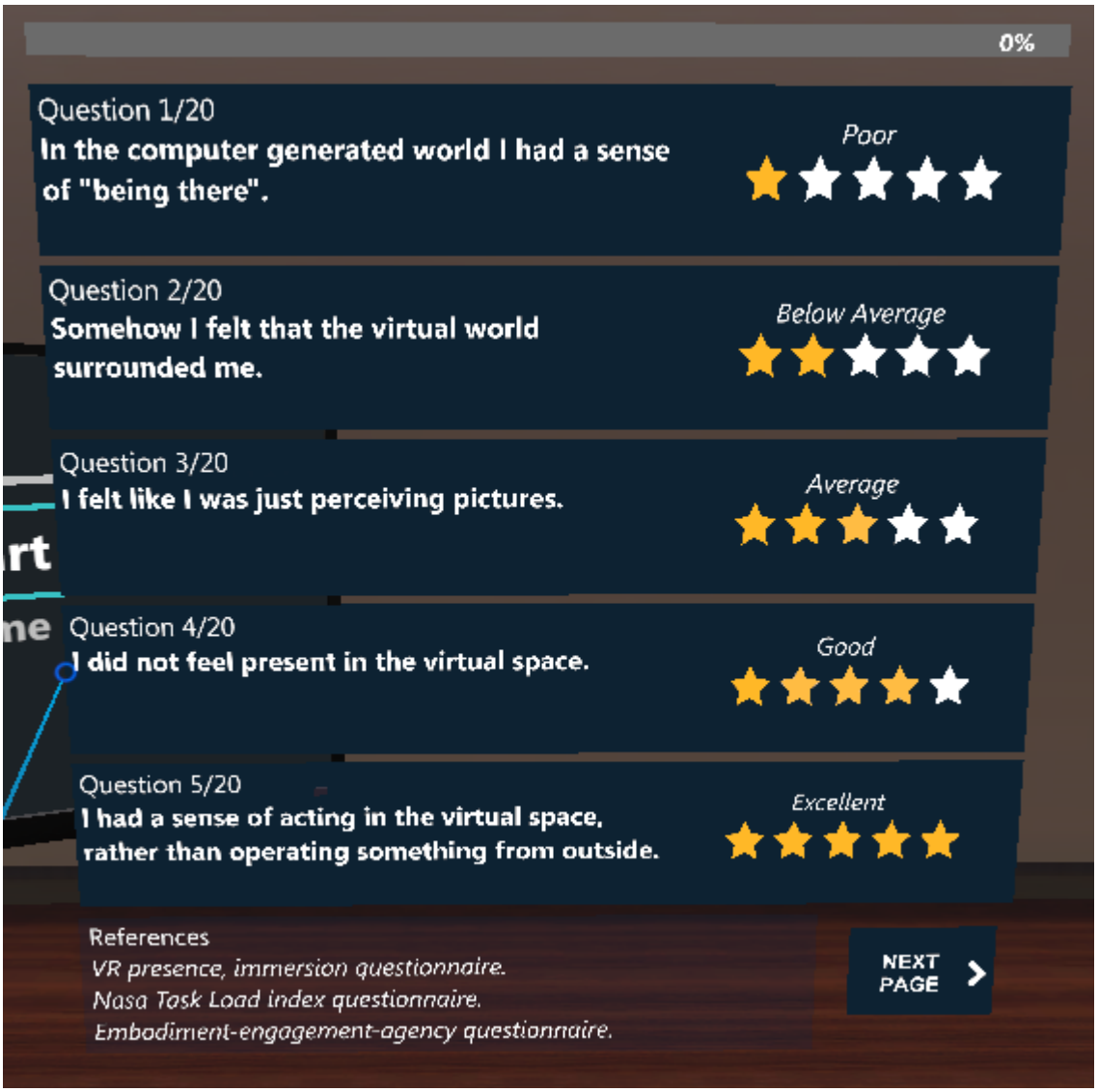
6.9.2 Creating a New Questionnaire

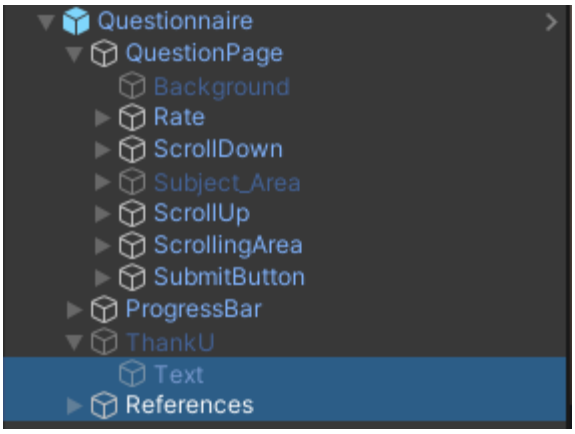
Navigate project file system following this path **Packages** → **com.oramavr.mages** → **Runtime** → **MAGES** → **SDKAddons** → **Questionnaire**. In this folder you will find questionnaire's prefab and its components, organized into folders. In 'Scripts' folder, there' s the "Questionnaire Manager" script, which you can access and add or modify the existing code and create any features you want. To spawn a new questionnaire, you just need to instantiate the prefab "Questionnaire".



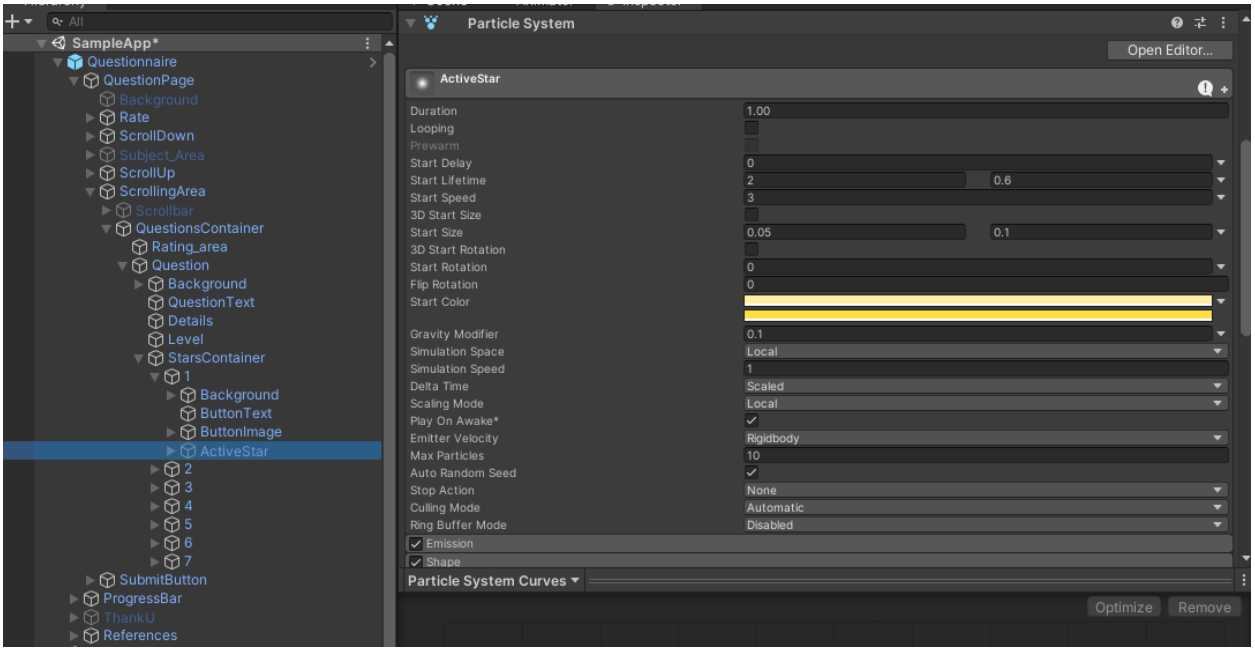
You can also edit the questionnaire settings from the inspector. You can change the Questionnaire's Manager "Questions" list, to add, remove or modify questions. Each question has a rating area. By default, it's set up with 7, but you can have as many stars as you want. After choosing your rating scale, you can also label each rating.


Questionnaire Manager (Script)
Script
QuestionnaireManager
Rating Scale
5
Rating Lables
5
Element 0
Poor
Element 1
Below Average
Element 2
Average
Element 3
Good
Element 4
Excellent


0%
Question 1/20
In the computer generated world I had a sense of "being there".
Poor
Question 2/20
Somehow I felt that the virtual world surrounded me.
Below Average
Question 3/20
I felt like I was just perceiving pictures.
Average
Question 4/20
I did not feel present in the virtual space.
Good
Question 5/20
I had a sense of acting in the virtual space, rather than operating something from outside.
Excellent
References
VR presence, immersion questionnaire.
Nasa Task Load index questionnaire.
Embodiment-engagement-agency questionnaire.
NEXT PAGE



When the user submits a rating, there's a sparkle effect playing. All rating stars have a particle system component simulating the effect, which you can customize, drag and drop a sound of your choice to play along with or even disable it. Finally, you can disable the "References" part and change the animated "Thank You" message, appeared at the end, by modifying its text on the prefab.



Note: The results are saved in JSON format. File destination is predefined for different application platforms.

- Android */../data/*
- Windows & macOS *C:/../Documents/ORamaVR/Questionnaire/Results/*
- Editor *Assets/Resources/Storyboard/Account/*

6.9.3 Cloud Service

You can modify the survey to your needs and get feedback. The survey is uploaded on the cloud. This one is configured in our system, but it's customizable, and you can integrate it into yours.

```
1 reference
IEnumerator UploadRating()
{
    // find stored file
    string jsonPath = $"{FolderPath}/QuestionnaireResults.json";
    if (!File.Exists(jsonPath))...

    // read it into memory
    byte[] jsonBytes = File.ReadAllBytes(jsonPath);

    // construct form
    WWWForm form = new WWWForm();
    form.AddField("Operation", ""); // <--- fill this
    form.AddBinaryData("File", jsonBytes, Path.GetFileName(jsonPath));

    // construct request
    string URI = ""; // <--- fill this
    UnityWebRequest request = UnityWebRequest.Post(URI, form);
    request.SetRequestHeader("Authorization", $"Bearer {UserAccountManager.Get.GetUserToken()}");

    // fire request
    yield return request.SendWebRequest();
    if (request.isNetworkError || request.isHttpError)...
    else...
}
```

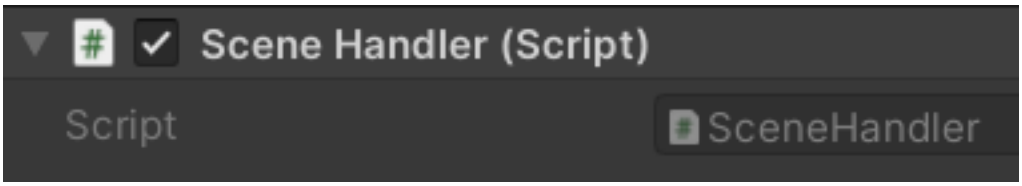
6.10 SceneHandler

In this tutorial, we will learn how to set up Scene Handler in our scene and how to transition between scenes.

Scene Handler is a mechanic contained in MAGES.Utilities module which enables scene transition. With the Scene Handler tool, developers can switch between scenes easily and effortlessly.

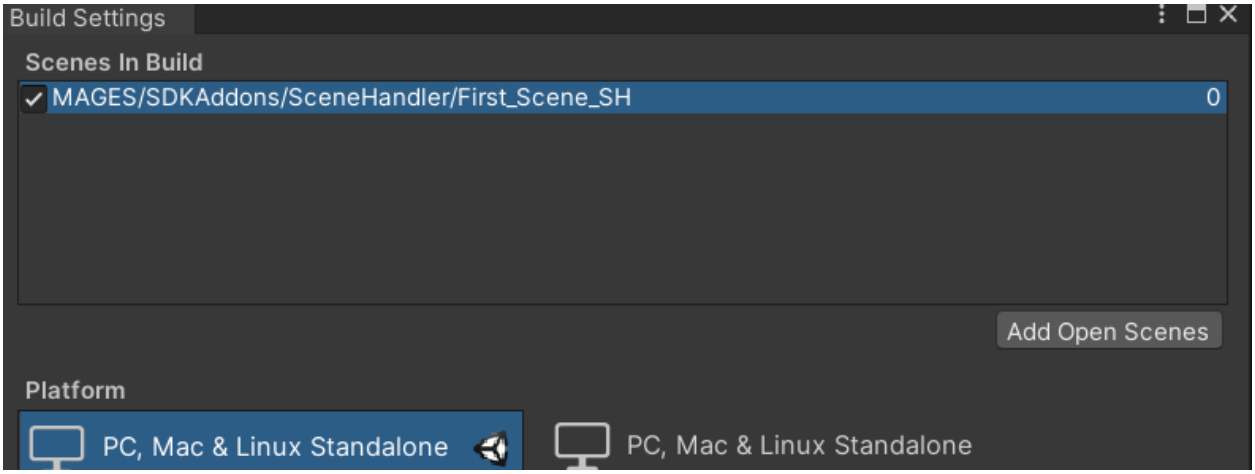
6.10.1 How to set up Scene Handler

The first is to create an empty scene and add the SCENE_MANAGEMENT component to your newly created scene. Create an empty gameobject with the name SceneHandler and Add the SceneHandler component to your object.

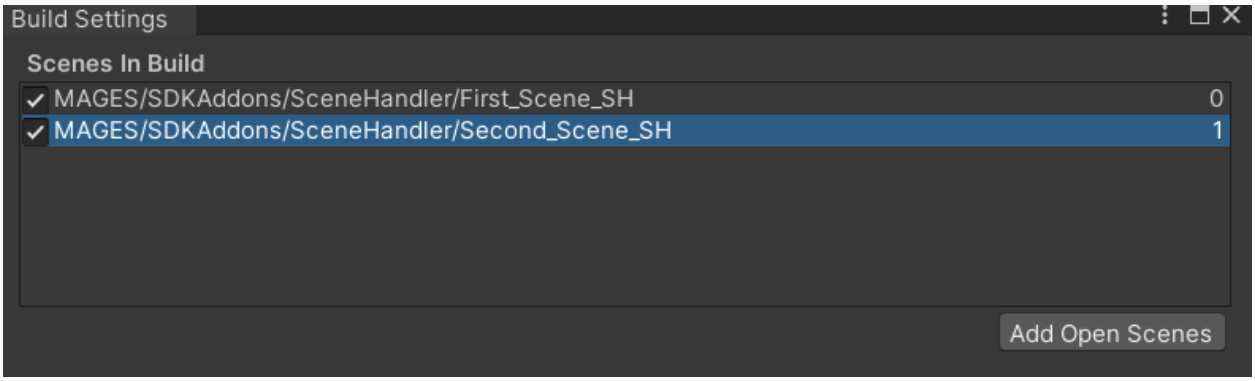


After that, you need to add the CameraFadeManagement component to a gameobject. We encourage you to add it on the InterfaceManagement gameobject.

Your next step is to edit the *"Scene in Build"* in Build Settings. In order to access Build Settings you need to go to File → Build Settings. After clicking Build Settings, a window will appear



Finally, we have to add all the scenes that we will perform a transition. We need to open each scene, and when they are loaded in the editor, press the *"Add Open Scenes"* button from the Build Settings window.



6.10.2 How to use Scene Handler

In this step we will write the method call to make a scene transition with and without fade.

In order to do a scene transition with fade, the method you can use is called SwitchScene and its members are a string for the name of the Scene that you are going to transition at, a Vector3 for the camera position and a Quaternion for the camera rotation. You can access this method from the MAGES.Utilities.SceneHandler.

```
using MAGES.Utilities
public class SceneChangerScriot : MonoBehaviour
{
    private public Start(){
        //Switches to the "Second_Scene" scene, with the given camera position and␣
↪rotation
        SceneHandler.Get.SwitchScene("Second_Scene",new Vector3(1,1,1),Quaternion.
↪Euler(0f, 90f, 0f));
    }
}
```

Below is an example of scene transitioning between the first scene and the second scene.

Another helpful method is the AddScene. With this function you can add a scene when none is there (without the fade-out effect). Usually, it will be the first scene. This function only has one member. A string for the name of the Scene that you are going to add.

```
using MAGES.Utilities
public class SceneChangerScriot : MonoBehaviour
{
    private public Start(){
        //Adds the "First_Scene" without a fade-out effect
        SceneHandler.Get.AddScene("First_Scene");
    }
}
```

6.11 Unity Integration with the Cloud

In this tutorial we will demonstrate the complete pipeline on integrating Unity with the web services provided in MAGES 3.2.

We will start of by creating a product for the Sample App through the Web Portal, and then create a user and assign him a valid product license.

After the *online* walkthrough is completed, we will proceed to integrate the Login service with the Sample App Unity side. Additionally, we will provide a configuration for uploading the user analytics to the AnalyticsAPI.

The end goal of this tutorial is to make you **product-ready**, meaning that you are able to build your VR Module and authenticate/check out any user licenses you want. Further, you will be able to upload user analytics to the AnalyticsAPI and display them in the Web Portal.

Tutorial Overview:

1. *Prerequisites*
2. *Product & User Creation*
3. *MAGES™ SDK Sample App*

6.11.1 Prerequisites

- You have configured all three web services whether for local Development or in a Production setting as described in the Web Services Manual section and the separate service sections per se:
  - For the Login Service: *Getting Ready for Development* & *Getting Ready for Production*
  - For the AnalyticsAPI: *Getting Ready for Development* & *Getting Ready for Production*
  - For the Portal: *Getting Ready for Development* & *Getting Ready for Production*
- All three web services are up and running whether locally or in the Azure Cloud
- You have an active user with Admin role permissions
- (Optional) Azure Storage Account and Azure Blob storage must be in place for uploading user Analytics

Note: For brevity, we assume in this tutorial that you have set up and running a Development –localhost environment of the Web Services.

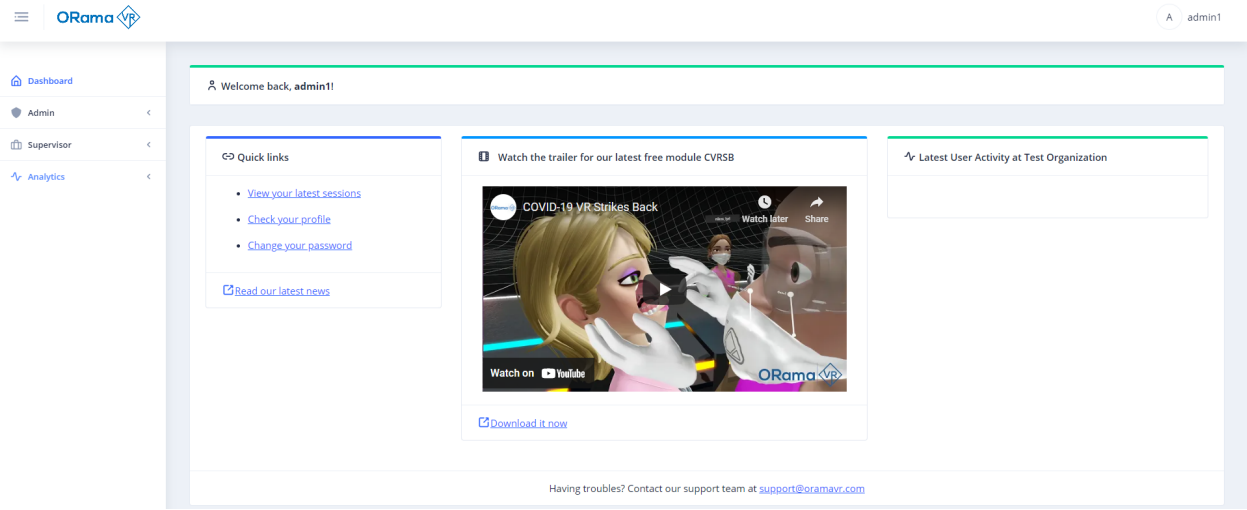
In any case, it is easy to translate the following steps to the equivalent Production configuration.

6.11.2 Product & User Creation

The first step is to login to the portal with an Admin user account.

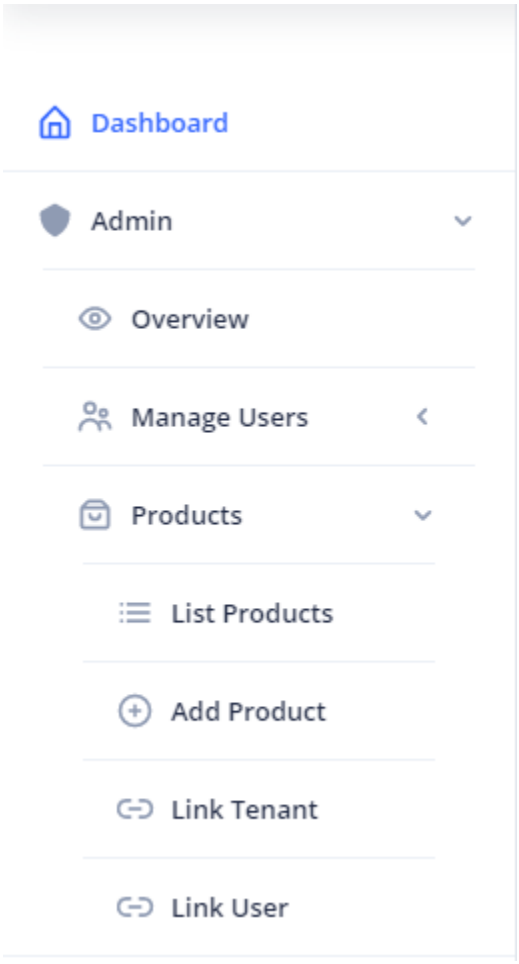
Point your browser to `localhost:4200` and you will be redirect to the Login Service `https://localhost:44355` for authentication.

Upon successful login, you will be redirected to the Portal Dashboard as below:



Add New Product

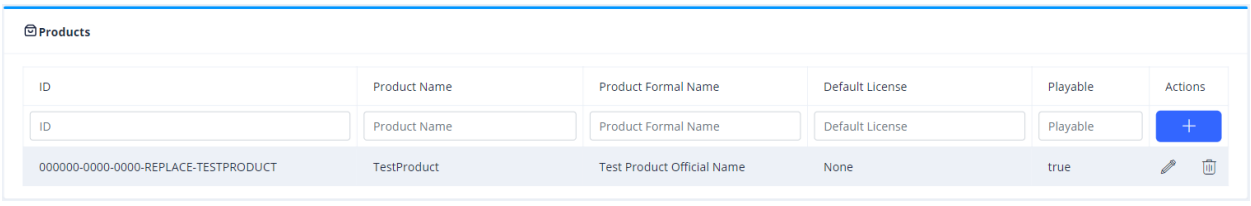
Proceed to expand the **Admin** menu item from the main sidebar, and further the child **Products** menu item as depicted in the figure:



From the **Products** menu, you can either click on the **List Products** menu item, or on the **Add Product** menu item.

List Products Option

If you clicked on the **List Products** you will see an overview of all your Products as the one below:



Click on the **Plus Icon button** on the right and a new table row will appear with an inline form. There, go on to create a product with the following configuration:

```
{
    "ProductName": "Test2021",
```

(continues on next page)

(continued from previous page)

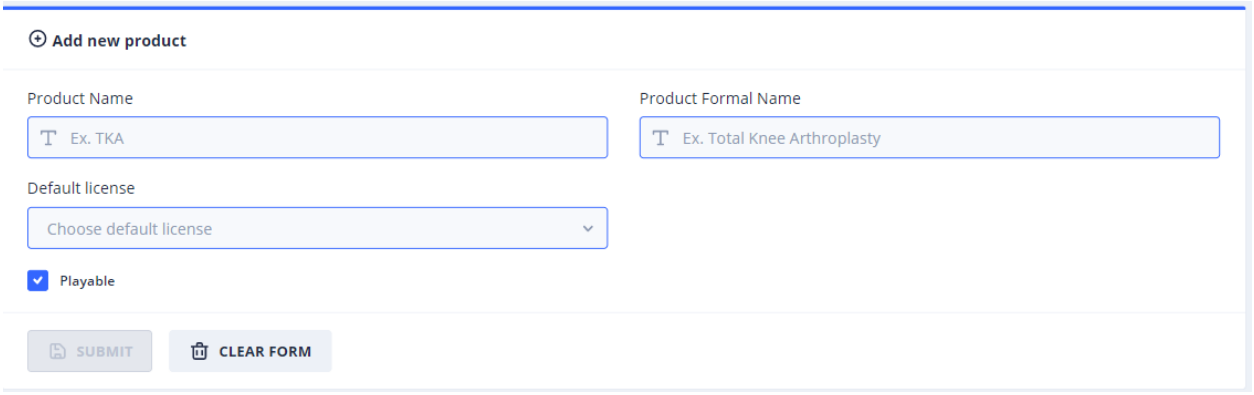
```
    "ProductFormalName": "This is the formal name of the product",
    "DefaultLicense": "None",
    "Playable": "True"
}
```

Click on the **Check Icon** on the right and a successful notification will appear on the top right of the page!

Note: Notice that the ID field is grayed out, this will be populated by the Login Service.

Add Product Option

If you clicked on the **Add Product** menu item, you will end up in the following page:



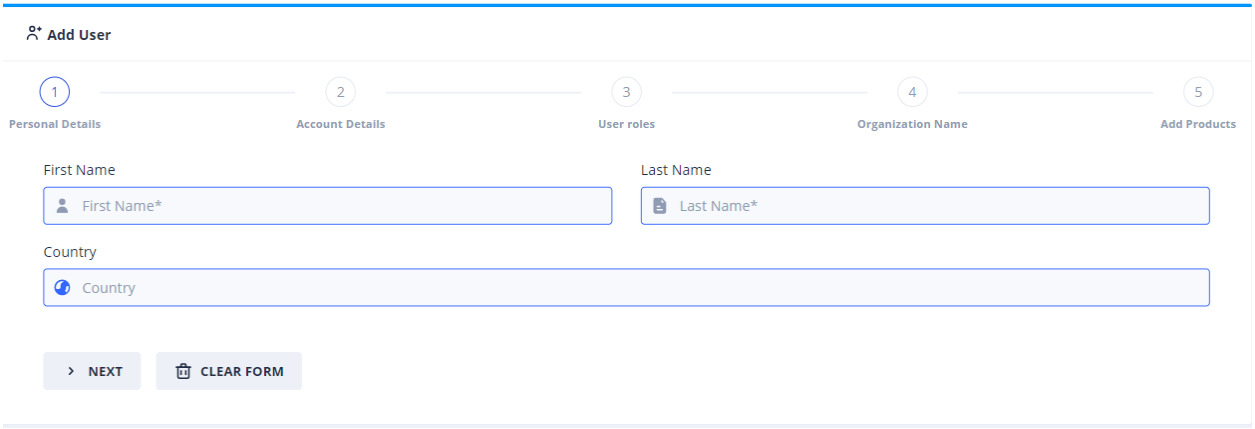
Here is a dedicated form that has also predefined options for the **DefaultLicense** and the **Playable** fields, instead of writing these info by hand.

Fill it in with the values in the JSON snippet above and click the **Submit** button to create the product.

User Creation

After the Product is created, navigate to the **Manage Users** expandable item and click on the **Create User** menu item.

The following *stepper* form will load:



With this form you can create a user and optionally assign him product licenses.

Step 1.

Proceed to fill in the **Personal Details** of the dummy user as you see fit.

Step 2.

In the **Account Details** step, create a username and a password for your user.

> **Warning:** When creating a user manually through the Portal, an email confirmation link will not be sent to the user's email address!
>
> The email address is considered **Confirmed** for the Login Service.
>
> By default, we assume admins are able to correctly register their users. This also helps testing user creation, because you do not have to enter a valid email address.

Step 3.

If the password meets the *Validator*, you can proceed to select the **User Roles**. If you user is a typical VR-only application user, select the *User* role only.

Otherwise, you can proceed to add more roles – but recall, roles are inclusive! (e.g. An admin has all three roles assigned, etc.)

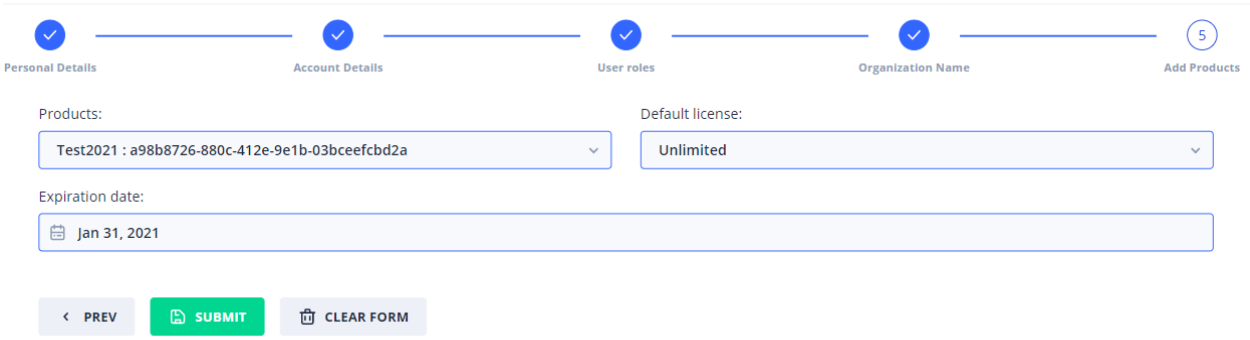
Step 4.

Select the Organization this user belongs to. For simplicity, select your own organization name or the **Default** one if applicable.

Step 5.

In the final step, we assign the user a license of the product we created earlier.

Make sure you input a date in the future and the **DefaultLicense** of the user is set to *Unlimited* as below:



Note: Unlimited means that the user can access the VR application frivolously, until the expiration day.

Go ahead and click the **Submit** button in the Step 5. to create the user. If everything went smoothly, you should see a green Notification bar at the top right corner.

Your user is now created with a product license attached! Congrats!

6.11.3 MAGES™ SDK Sample App

After the product and the user is set, let's move on to the SDK Sample App for the integration part with the Login service.

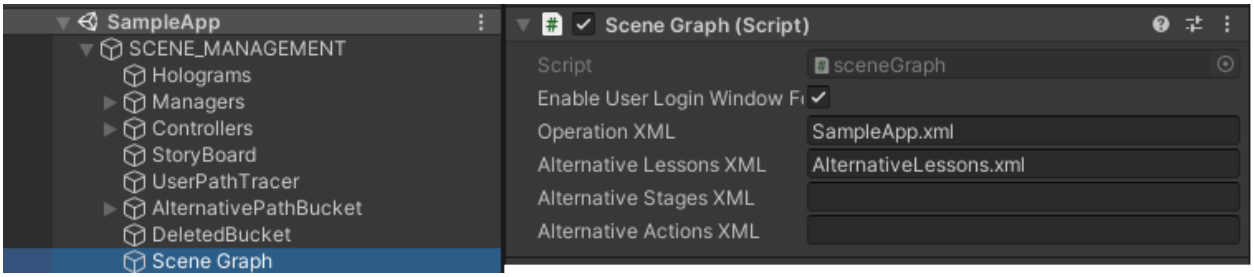
Open your MAGES SDK project in Unity and select the **Sample App** scene from `ovidVRcomp/Operation/Scenes/`.

Enable User Login Window

First, we need to make sure the checkbox `Enable User Login Window For Build` is checked in the Unity Editor.

To do so, expand the *Scene Manager* component and click on the *Scene Graph* component.

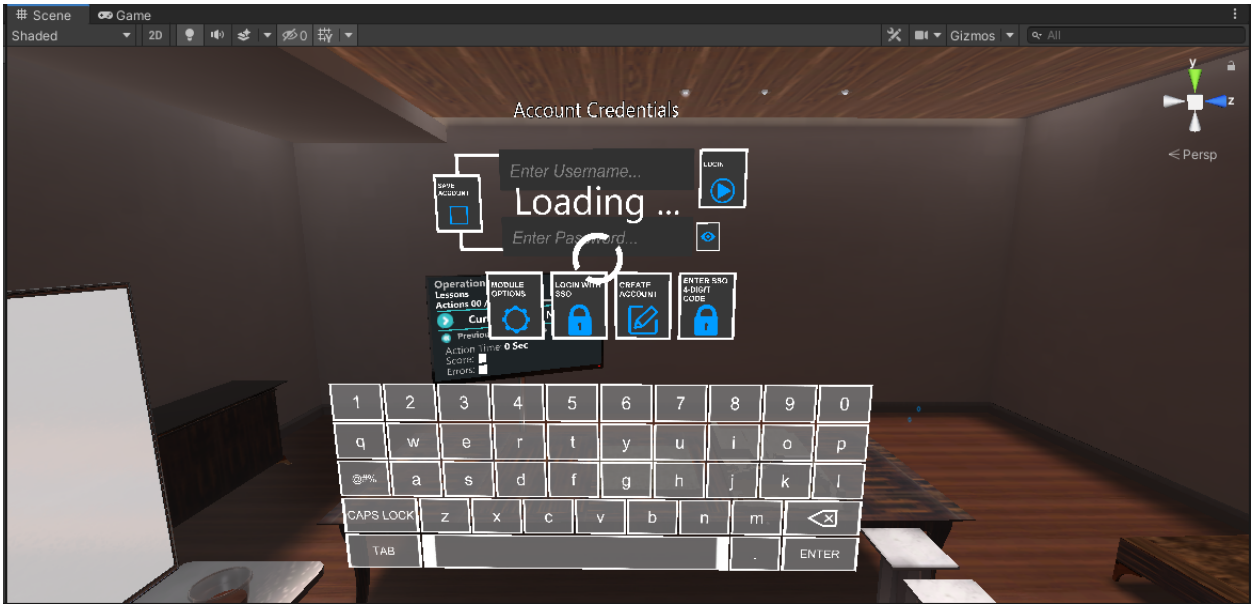
On the Inspector you can see if the `Enable User Login Window For Build` checkbox is ticked, as in the figures below:



This variable is set in the `scenegraph.cs` script and will make sure that when you build your VR application, the Login window prefab will spawn automatically, asking for user authentication before the user can access your application!

Login Flow UI

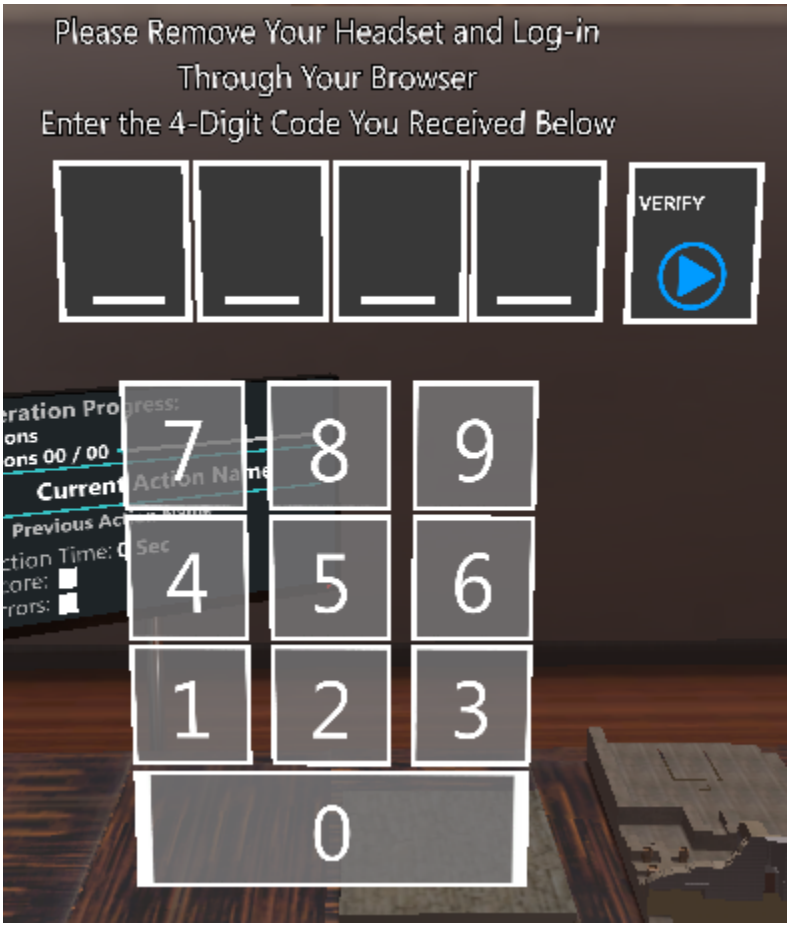
By default, the prefab of the Login window is set to `UILicenseRequestSSO.prefab` under `Resources/ovidVRres/Operation/Prefabs/UI 2.0/`.



Notice that the UI prefab offers 3 available options for user sign in & authentication.

1. Authenticate by typing Username & Password and pressing the **Login** button on the right.

2. Authenticate by clicking on the **Login with SSO** button. This will open and point the user's browser at your Login service, where the user can sign in either with username/password credentials or by clicking on a Single-SignOn button (e.g. Google). Then, the user will be presented with a 4-digit code that has to fill in at the next screen in the picture below.



3. If already at the 4-digit code on the browser, the user can click on the **Enter SSO 4-DIGIT Code** button on the bottom right to directly enter the 4-digit code.

Client Configurations

Now we need to set the Client configurations to connect Unity with the Login service.

There are two distinct parts that need to be defined, each for a separate purpose:

1. User Authentication with Username & Password (w/o SingleSignOn)
2. User Authentication via browser with SSO capabilities

To properly work with the provided UI Login prefab, you need to configure both settings.

However, if you only wish to authenticate users with either of the two methods, you can simply edit the prefab and remove one of the options.

User Authentication with Username & Password

For the first part, you need to open the `LicenseRequest.cs` script that is under the `ovidVRcomp/SDKAddons/ovidVRScripts/License/` directory.

Navigate to the `Login` function at line `162`, it should be identical to the one below:

```
public void Login()
{

    if ((!string.IsNullOrEmpty(userField.text) && !string.IsNullOrEmpty(passwordField.text)))
    {
        loginButton.ButtonActivation(false);
        StartCoroutine(FadeMusic(false));
        StartCoroutine(DelayLogin());
    }

    #if UNITY_STANDALONE_WIN
        KeyboardController.SetKeyboardState(true);
    #endif
    user = userField.text;
    pass = passwordField.text;

    #if !UNITY_EDITOR
        // Login Flow and Checkout User License
        ClientConfiguration client = new ClientConfiguration()
        {
            ClientId = "",
            ClientSecret = "",
        };
        var identityUrl = "";
        AuthenticationHandler.Instance.LoginUserWithoutSSO(client, identityUrl, user, pass, "SampleApp", result =>
        {
            if (result == LoginStatus.Success)
            {
                Debug.Log("User authenticated");
                hasLic = true;
                cameraRig.enabled = true;
                Destroy(this.gameObject);
            }
            else
            {
                Debug.Log("Error authentication");
                Debug.Log(result);
                Init(true);
            }
        });
    #endif
    InterfaceManagement.Get.applicationLanguage = selectedLang;
}
```

The `Login` function performs the Sign In directly from the UI prefab, with the use of Username & Password.

To do so, it utilizes the Resource Owner Password and Client Credentials (ROPC) grant type as described in the official IdentityServer4 documentation.

Note: ROPC is considered a valid client only for *trusted* and *native (legacy)* applications.

You should use it only for your Unity projects, and not for web-based applications, as we do on the Portal.

You need to populate the configuration between lines `19` and `25`.

To get you started, we have already defined a similar Client to the Login service inside the `Config.cs` file at the *root* directory.

For brevity, we will use the same configuration with `ClientId: UnityModuleWithoutSSO`.

Danger: Make sure to alter the **ClientSecret** value to a secure and private one, never use the defaults provided in a Production setting!

We proceed to replace lines `19-25` with the following:

```
// Login Flow and Checkout User License
ClientConfiguration client = new ClientConfiguration()
{
    ClientId = "UnityModuleWithoutSSO",
    ClientSecret = "unity.module.without.sso.2020",
};
var identityUrl = "https://localhost:44355/";
AuthenticationHandler.Instance.LoginUserWithoutSSO(client, identityUrl, user, pass,
↪"Test2021", result =>
{
    if (result == LoginStatus.Success)
    {
        Debug.Log("User authenticated");
        hasLic = true;
        cameraRig.enabled = true;
        Destroy(this.gameObject);
    }
    else
    {
        Debug.Log("Error authentication");
        Debug.Log(result);
        Init(true);
    }
});
```

Notice also, the parameter `Test2021` we passed in function `LoginUserWithoutSSO` matches the *ProductName* of the one we created through the Portal.

Essentially, this function will authenticate the user with given credentials from the prefab's text boxes (i.e., username and password) and checkout a user license for the product **Test2021**. Additionally, the underlying service will attempt to retrieve the **User Data** from the Login Service, that you can further utilize to upload analytics or populate your COOP, etc.

If any of the above steps fails, you will be prompted with an error message, and the Login screen will remain in place.

If the operation was successful, the Login screen will self-destroy and the user shall be able to continue with your module.

User Authentication via Browser (SSO support)

Let's proceed to configure our settings for the second enriched Login flow that provides us with SSO capabilities.

In the same code file `LicenseRequest.cs`, navigate to the `LoginSSO` function, it should be identical to the one below:

```
public void LoginSSO()
{
    // Login Flow and Checkout User License
    ClientConfiguration client = new ClientConfiguration()
    {
        ClientId = "",
        ClientSecret = "",
        AllowedScopes = ""
    };
    var identityUrl = "";
    var loopbackUrl = "";
    AuthenticationHandler.Instance.LoginUserBrowser(client, identityUrl, loopbackUrl,␣
↪result =>
    {
    });
    Spawn4DigitUI();
}
```

When a user clicks on the **Login with SSO** button, this function will spawn up a browser and point to your Login service for authentication.

The delegate is empty due to the fact that it will only spawn a browser page, and does not need to handle anything on return.

Sequentially, the 4DigitUI will be spawned, waiting for the user to enter his 4-Digit code.

To get you started, we have already defined a similar Client to the Login service inside the `Config.cs` file at the *root* directory.

For brevity, we will use the same configuration with `ClientId: UnityModule`.

Proceed to replace the snippet above with the following:

> **Danger:** Make sure to alter the **ClientSecret** value to a secure and private one, never use the defaults provided in a Production setting!

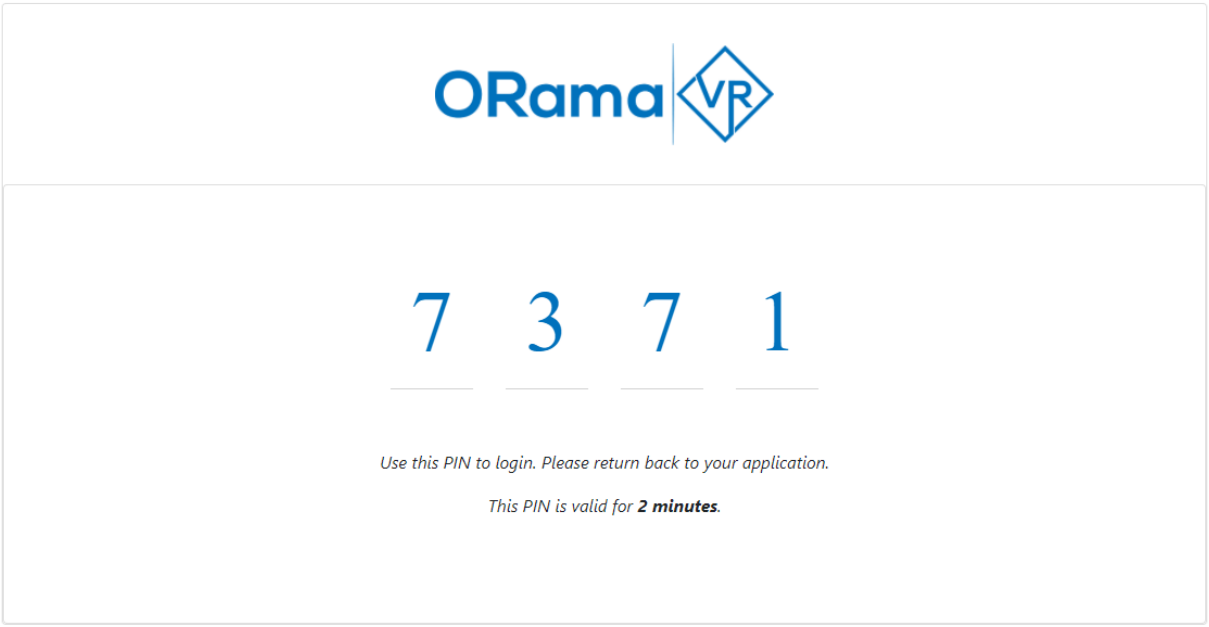
```
public void LoginSSO()
{
    // Login Flow and Checkout User License
    ClientConfiguration client = new ClientConfiguration()
    {
        ClientId = "UnityModule",
        ClientSecret = "unity.module.2020",
        AllowedScopes = "openid profile offline_access IdentityServerApi"
    };
    var identityUrl = "https://localhost:44355/";
    var loopbackUrl = "https://localhost:44355/unity";
    AuthenticationHandler.Instance.LoginUserBrowser(client, identityUrl, loopbackUrl,␣
↪result =>
    {
```

(continues on next page)

(continued from previous page)

```
    });
    Spawn4DigitUI();
}
```

When a user logs in through the browser, he will be redirected to the *unity* endpoint defined in the Login service, which will serve him with the 4-Digit code, as in the example below:



Then, the user heads back to Unity and enter the code.

Note: Each code expires after 3 minutes even though it states 2. This is to handle sloths.

Notice also, that no license checkout happens at this place. The checkout will happen when the user enters the 4-digit code at Unity and clicks on the **Verify** button.

Final step - 4Digit Code

To complete the flow, we also need to provide configurations for the `4DigitUI.prefab`.

Proceed to open the `VRNumpadController.cs` file under the `ovidVRcomp/SDKAddons/VRKeyboard/Scripts/` directory.

Navigate to `CheckCode` function and it should be identical to the one below:

```
public void CheckCode()
{
    string code = GameObject.Find("Digit1/Placeholder").GetComponent<Text>().text +␣
↪GameObject.Find("Digit2/Placeholder").GetComponent<Text>().text + GameObject.Find(
↪"Digit3/Placeholder").GetComponent<Text>().text + GameObject.Find("Digit4/
↪Placeholder").GetComponent<Text>().text;

    ClientConfiguration client = new ClientConfiguration()
```

(continues on next page)

(continued from previous page)

```
    {
        ClientId = "",
        ClientSecret = "",
        AllowedScopes = ""
    };
    var identityUrl = "";
    AuthenticationHandler.Instance.CheckoutUser(client, identityUrl, code, "SDK",␣
↪result =>
    {
        if (result == LoginStatus.Success)
        {
            Debug.Log("User authenticated");
            LicenseRequest.hasLic = true;
            GameObject.Find("UILicenseRequestSSO(Clone)").GetComponent<LicenseRequest>
↪().cameraRig.enabled = true;
            Destroy(GameObject.Find("UILicenseRequestSSO(Clone)"));
            Destroy(GameObject.Find("4DigitUI(Clone)"));
        }
        else
        {
            Debug.Log("Error authentication");
            Debug.Log(result);
            _header.GetComponent<Text>().text = "The 4-Digit Code was incorrect.";
        }
    });
}
```

To make sure that our application is authenticated and no one else can start posting up 4-digit codes, we use an additional Client for this request.

As before, to get you started, we have already defined a similar Client to the Login service inside the `Config.cs` file at the *root* directory.

For brevity, we will use the same configuration with `ClientId: UnityInternal`.

Proceed to update the following fields like below:

```
public void CheckCode()
{
    ClientConfiguration client = new ClientConfiguration()
    {
        ClientId = "UnityInternal",
        ClientSecret = "unity.internal.2020",
        AllowedScopes = "IdentityServerApi"
    };
    var identityUrl = "https://localhost:44355/;
    AuthenticationHandler.Instance.CheckoutUser(client, identityUrl, code, "Test2021",
↪ result =>
    {
        // redundant
    });
}
```

And we are done!

But before you go, notice also here that we have to provide the **ProductName** we are checking out against, as in the Username/Password flow.

Further, the `UnityInternal` client authenticates/protects an API resource with Client credentials only (i.e., Clien-

tID, ClientSecret), make sure to check it out on the official IdentityServer4 documentation.

Now you can proceed to build a version of the SampleApp (whether Windows standalone or Android), and you can checkout your users product licenses ;).

Otherwise, keep reading for more.

User Account Manager

By default, and for convenience, we populate the `UserAccountManager` class with the authenticated user information.

This includes username, the JWT token, and other optional values such as first and last name.

Therefore, you don't need to acquire any user information throughout the whole application experience. Everything is there for you to utilize as you see fit.

Uploading User Analytics

Last but not least, to upload user analytics we have to configure `Scenegraph.cs` script to populate your Upload with the info from **UserAccountManager** described above.

Proceed to open `Scenegraph.cs` and navigate to line `174`.

It should be identical to the one here:

```
// Configuration properties that regard post Scenegraph initilization such as␣
↪Analytics, Users go here.
//Configuration.User = new ApplicationUser
//{
//    id = "app-user",
//    firstName = "User",
//    lastName = "User",
//    country = "Devland",
//    userName = "proplayer"
//};
//Configuration.UserPassword = ("default");

// For uploading analytics specify your endpoint and add any extra form fields or␣
↪headers required from your service.
Configuration.OnlineURL = "";
//Configuration.OnlineURL = "";
Configuration.FormFields = new List<AnalyticsExporter.FormField>
{
    new AnalyticsExporter.FormField { key = "Username", value = UserAccountManager.
↪Get.GetUsername() },
    new AnalyticsExporter.FormField { key = "Operation", value = "SDK" },
};
Configuration.HeaderKeys = new List<AnalyticsExporter.HeaderKey>
{
    new AnalyticsExporter.HeaderKey { key = "Authorization", value = "Bearer " }
};
Configuration.ConfigurePostInitialization(); // Important
```

Since we will be using the **UserAccountManager** to upload user analytics, you can proceed to remove the commented lines that create an **ApplicationUser**:

Ideally, the end result should look like this:

```
// For uploading analytics specify your endpoint and add any extra form fields or␣
↪headers required from your service.
Configuration.OnlineURL = "http://localhost:5002/Upload";
Configuration.FormFields = new List<AnalyticsExporter.FormField>
{
    new AnalyticsExporter.FormField { key = "Username", value = UserAccountManager.
↪Get.GetUsername() },
    new AnalyticsExporter.FormField { key = "Operation", value = "Test2021" },
};
Configuration.HeaderKeys = new List<AnalyticsExporter.HeaderKey>
{
    new AnalyticsExporter.HeaderKey { key = "Authorization", value = "Bearer " +␣
↪UserAccountManager.Get.GetUserToken() }
};
Configuration.ConfigurePostInitialization(); // Important
```

All we did is specify the API endpoint, set the *Operation* name, in this case the Product we created earlier, and finally, use the authenticated user JWT to authenticate against the Login service.

Recall that AnalyticsAPI is connected with Login to work properly. Therefore, tokens are delegated to the Login service for authorization.

This is particularly useful for making authorized requests on behalf of the user.

Note: In a full playthrough with an authenticated user, if Analytics are uploaded correctly, you should be able to see them via the Web Portal.

This is everything required to integrate Unity with the Web Services seamlessly.

Next step is to actually build your application and test it!

Happy coding!

6.12 Multiplayer

6.12.1 Multiplayer in MAGES™ SDK

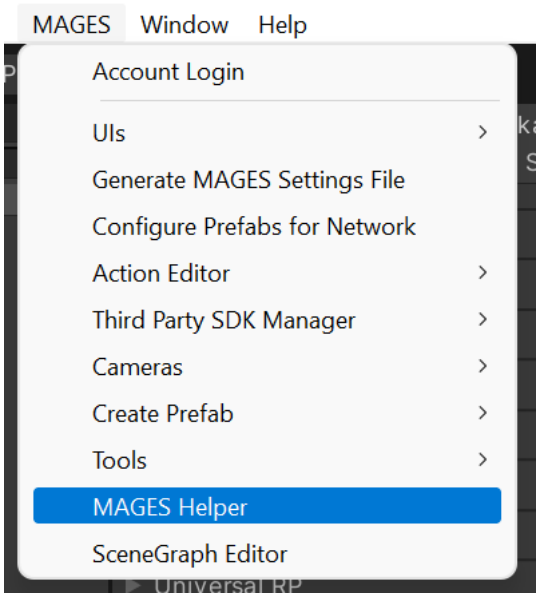
Applications built with MAGES™ SDK are multiplayer/network ready, meaning that with a few more actions needed by the developers, multiple users can cooperate and complete these simulations together online [M1].

In this section we will go through the details that developers need to check to ensure that their application is multiplayer/network ready, as well as a tutorial on how a user can create and/or join multiplayer sessions within a MAGES™ application.

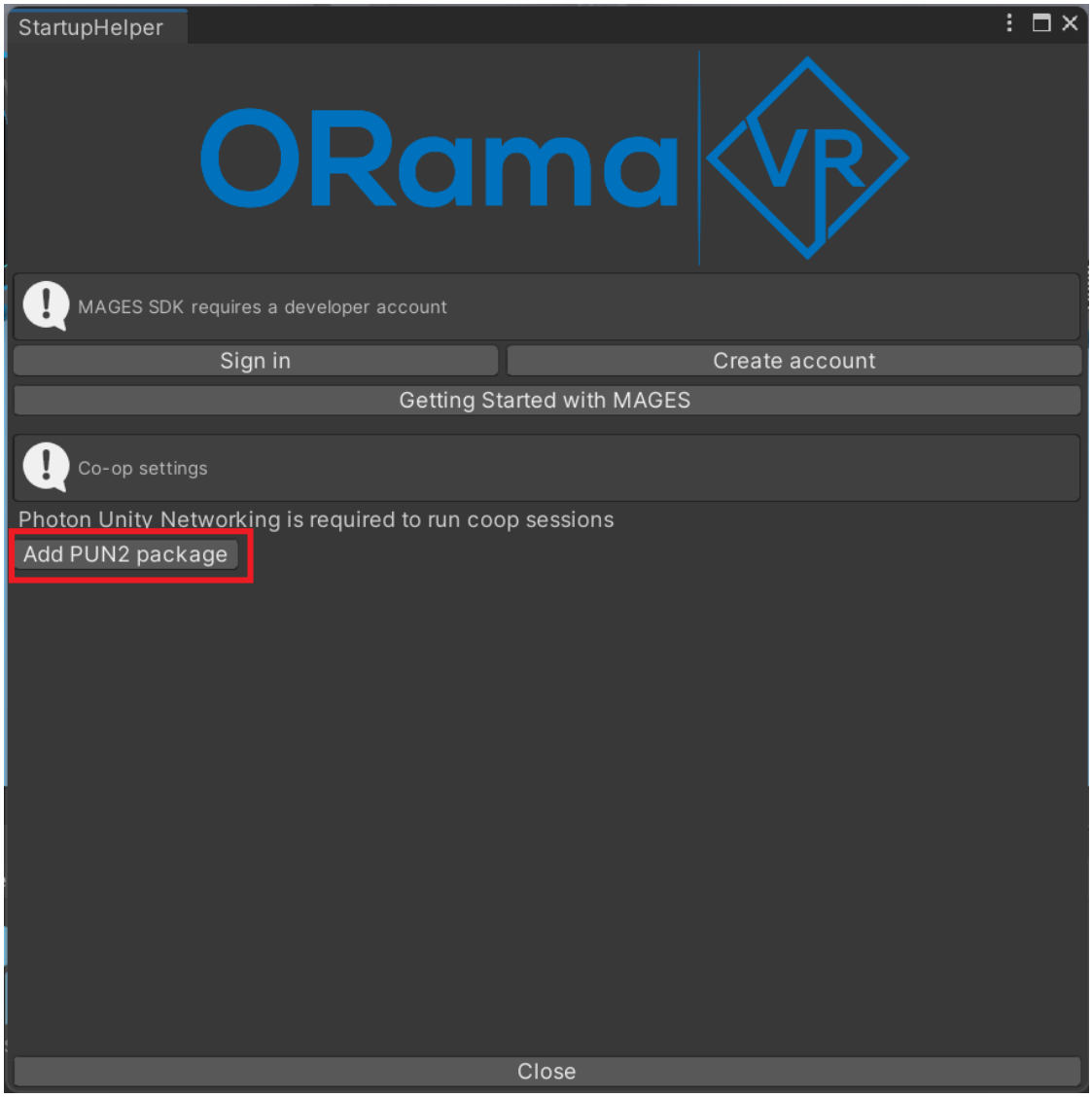
Networking Developer Guidelines

MAGES™ SDK supports Photon networking as the default networking API. For cooperation/multiplayer mode you need to setup Photon (playing in the same room with other players).

First you need to install the Photon PUN2 asset (it is free). Navigate to the MAGES menu and click the "MAGES Helper"

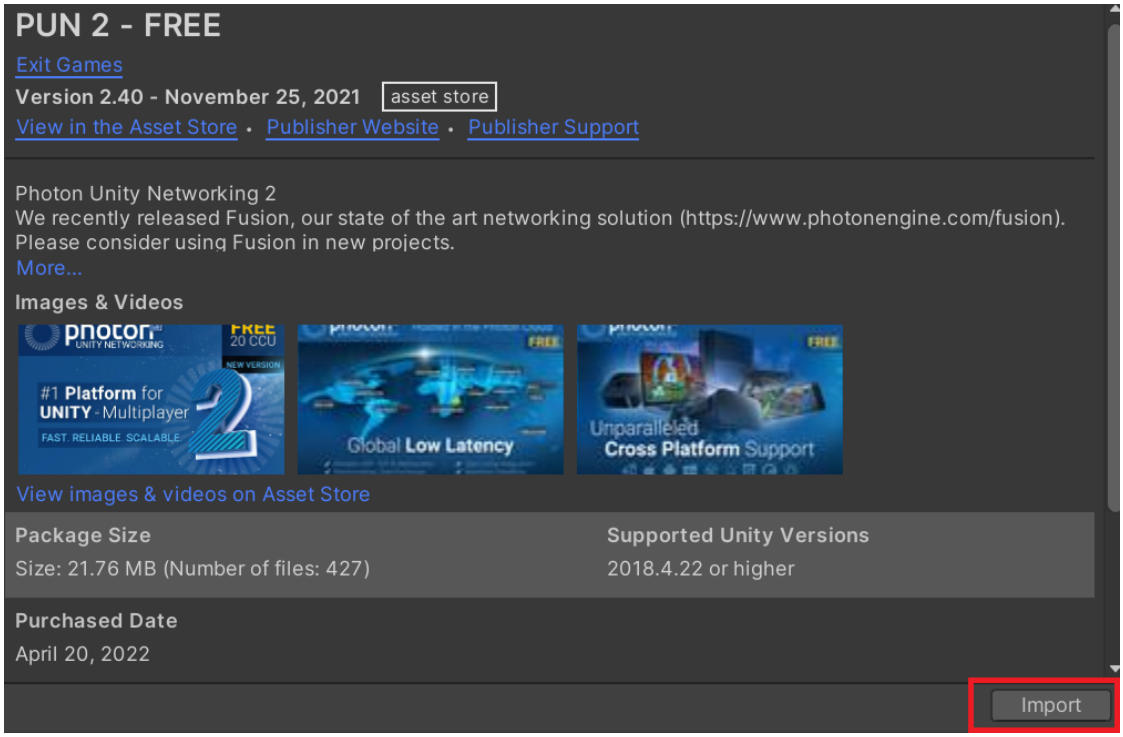


The Helper window will pop and you will see that you need to install the PUN2 package. Click the "Add PUN2 package".



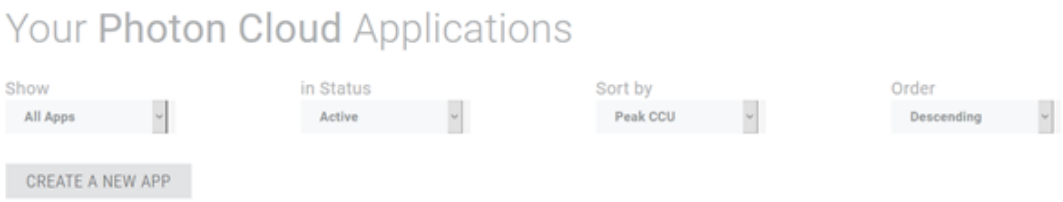
This will navigate you to the asset store. From there you can add the package to your assets.

And import the package to your project.



After installing the package, you need to create an account at Photon.

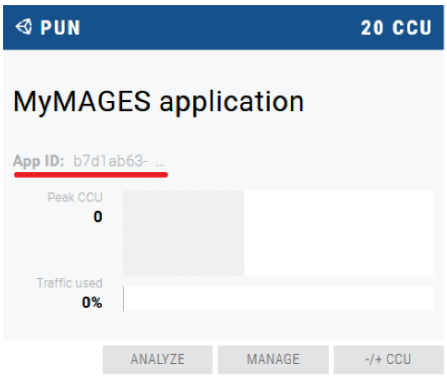
After successful registration, go to the Photon Dashboard and click the *CREATE A NEW APP* button.

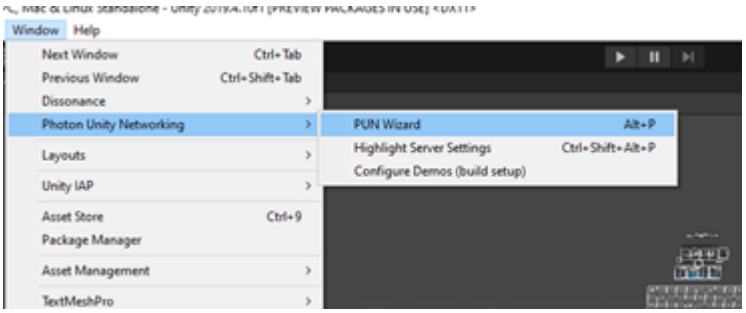


Select Photon PUN from the Photon type dropdown, give a name to your application and click the *CREATE* button.

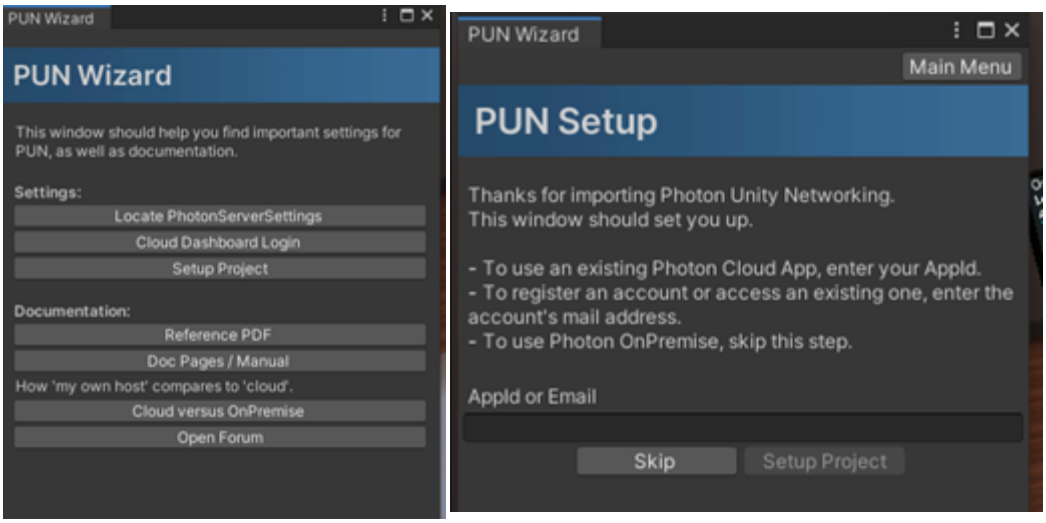
Navigate back to the main dashboard page and you will see your new PUN application. You need to copy the App ID. Open the Unity project and from the top bar go to **Windows/Photon Unity Networking/PUN Wizard**.

Note: Click the code next to the App ID to reveal the full key.



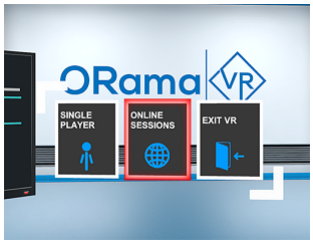


Click **Setup Project**, paste your App ID at the **Appid or Email** field and click **Setup Project**.



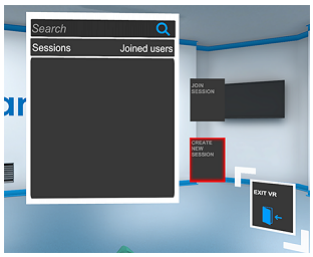
Host Online Session

1. Once you start the simulation, you will be greeted with two options. Starting the simulation in Single Player, or go to Online Sessions.



2. Select the Online Sessions. There you will be able to create a new session and wait for other to join, or join an existing session that is demonstrated on the Sessions board.

3. To Create a session, select the Create New Session, as illustrated.



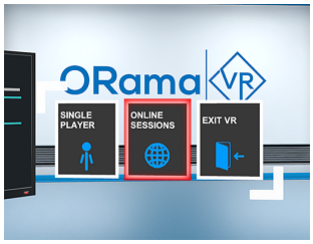
4. Wait for others to join. If at least one more user has joined, you will be able to start your online session.

5. If you want to go back, select Exit VR to quit.

   Note: If you are using older versions of our simulation, please refer to this video for instructions.

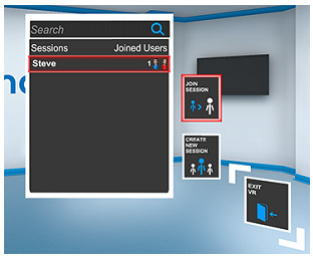
Join Online Session

1. Once you start the simulation, you will be greeted with two options. Starting the simulation in Single Player, or go to Online Sessions.



2. Select the Online Sessions. There you will be able to create a new session and wait for others to join, or join an existing session that is demonstrated on the Sessions board.

3. Select an available session from the Sessions board, and select Join Session. Each available session will demonstrate all connected users.



4. If you want to go back, select Exit VR to quit.

   Note: If you are using older versions of our simulation, please refer to this video for instructions.

References

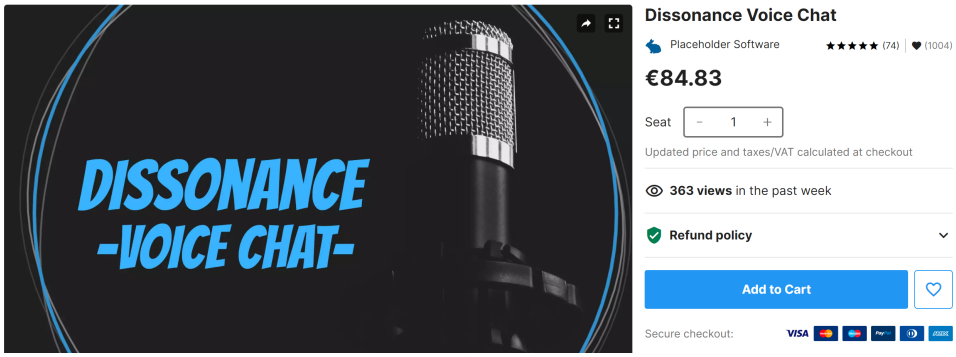
6.12.2 Adding Voice Chat (Dissonance)

You can use the Dissonance plugin in Unity, in order for your users to be able to talk to each other while being in a multiplayer session in MAGES™ SDK. Follow the steps below to install the Dissonance plugin in your Unity MAGES™ SDK project.

Note: Using Dissonance is **optional**.

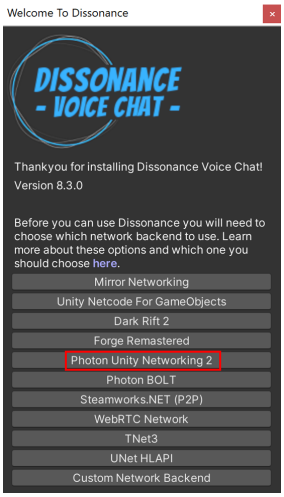
---

1. Download the Dissonance package. You can find Dissonance in Unity Asset Store, or by clicking this link.

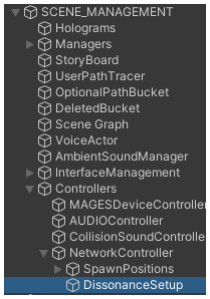


2. Import the package you downloaded in step 1, to your Unity MAGES™ SDK project.

3. Once Dissonance is imported, a window like the following will appear. Choose **Photon Unity Networking 2**.



4. Create an empty gameobject under **SCENE_MANAGEMENT/Controllers/NetworkController**. Name it DissonanceSetup.



5. Navigate to the newly created DissonanceSetup gameobject and add the following four components to it.

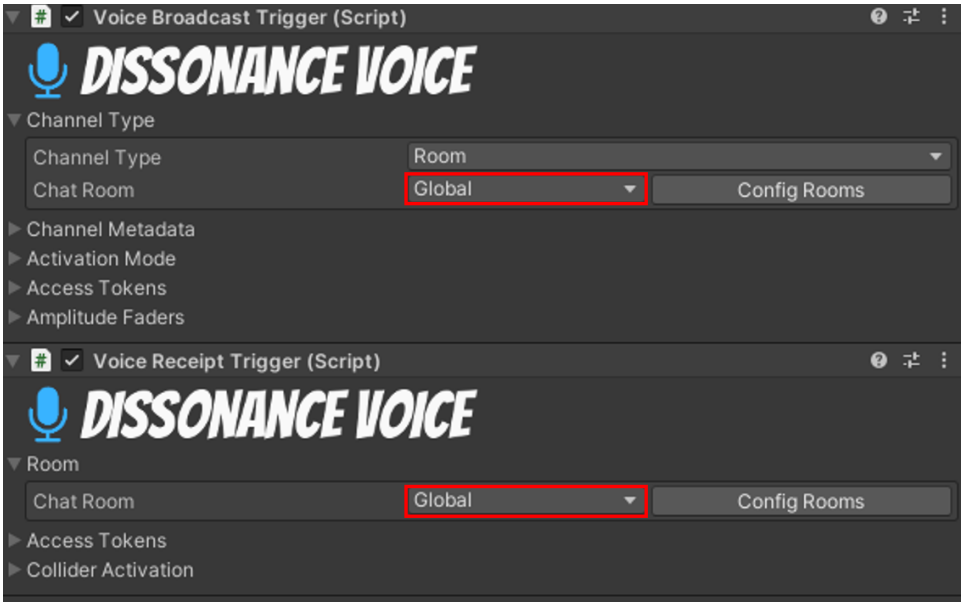
    a. Dissonance Comms.

    b. Photon Comms Network.

    c. Voice Broadcast Trigger.

    d. Voice Receipt Trigger.

Note: Regarding Voice Broadcast/Receipt Trigger components, you will need to identify the room that will be used. For most of the cases, the **Global** room is a good choice.



6. Dissonance is ready. Next time you will join a multiplayer MAGES™ SDK session, you will be able to speak with the rest of the users.

6.13 Upgrade from MAGES v3.x to v4.x

In this tutorial we will upgrade an existing project made with MAGES v3.x to the latest version of MAGES v4.x (currently the v4.0.2).

6.13.1 Step 1: Folder structure

First, gather everything operation related into 4 folders. This is optional but it helps to separate the SDK from the operation. The 4 folders are: ActionScripts, Models, Resources, Scenes. Those folders will only contain assets and scripts from your operation, nothing related to the MAGES SDK.

In more detail:

ActionScripts: Contains the Actions for the operation.

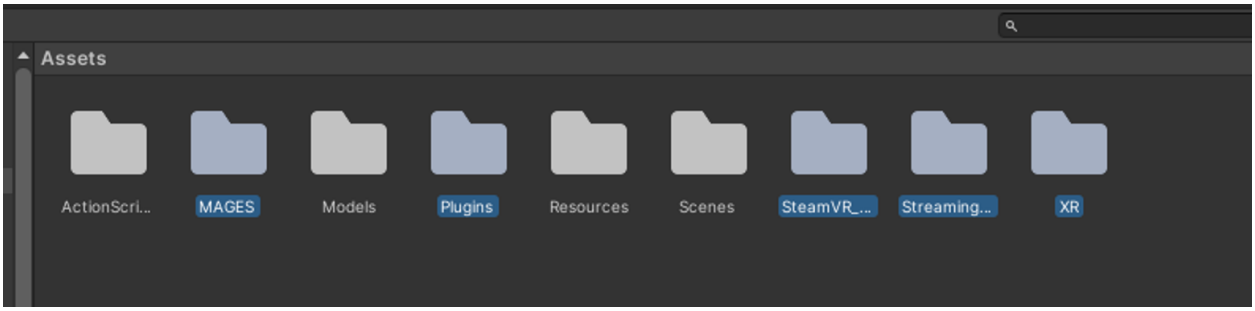
Models: Contains the models, animations, textures and materials. Basically, every visual asset that is not required to be in the resources folder (spawning runtime).

Resources: All the prefabs and items that will spawn in this operation.

Scenes: Contains the scenes and the MAGESSettings.asset file.

6.13.2 Step 2: Delete old MAGES

Delete the rest. This includes anything MAGES related. From SDK Addons to plugins, in our case we will only keep the 4 folders we created in the previous step. Make sure you also delete the MAGESres folder in the resources.



It is very important to delete the Assets/Plugins folder.

Also before moving on you have to do the following. It is a bit weird but required. Remove the Asset/Resources folder from your project before importing the MAGES SDK. After importing the SDK please put it back in place where there was. This is required cause otherwise the SDK will not be imported as it should (the folder com.oramavr.mages@4.0.2RuntimeResources will not be visible from the Unity).

After this step. lots of errors will appear, don't worry it is normal. We ll fix them in the following steps.

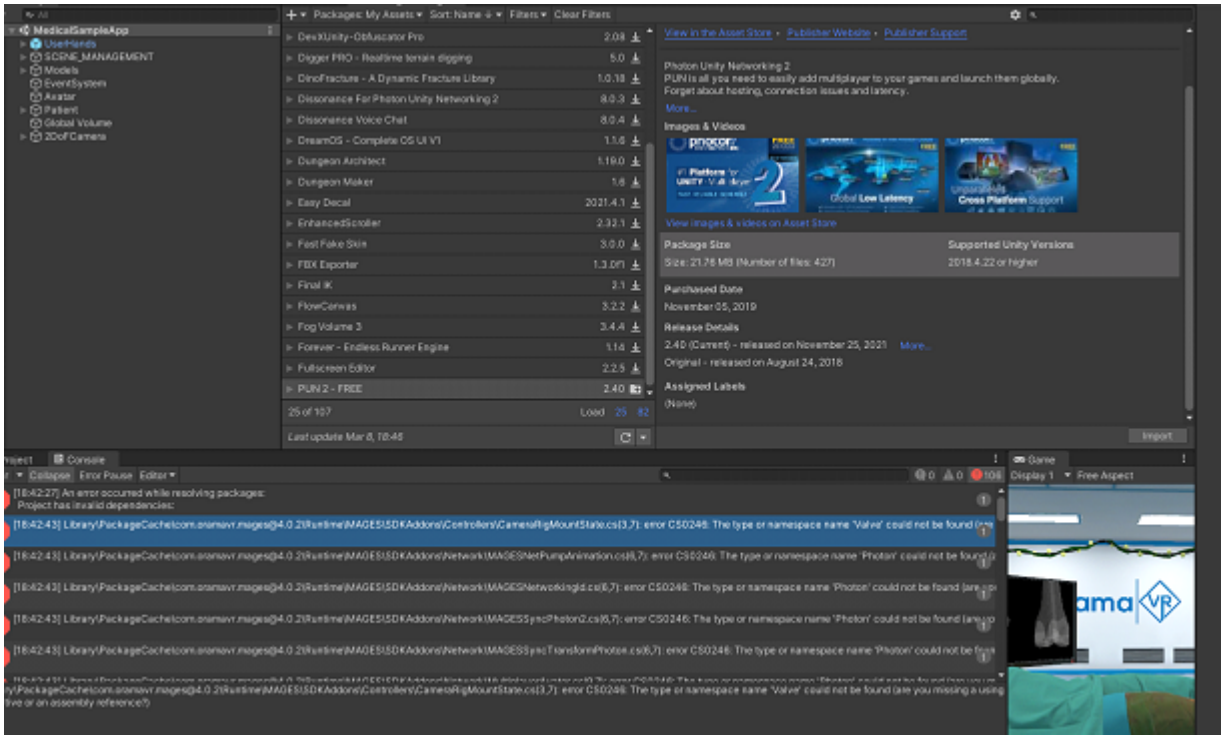
6.13.3 Step 3: Download and Import latest MAGES SDK

Now we have to download the newest version of MAGES SDK. In this example we will download and import to our project the MAGES 4.0.2 version.

Here you can find instructions on how to download and import the latest MAGES SDK.

6.13.4 Step 4: Fix compile errors (Third party assets)

After importing the MAGES SDK the majority of the errors will be fixed. However, some of them may remain. In this step we need to fix those errors before moving on. In our case we had some errors from cause the photon and Dissonance package was required for this project and we deleted it along with the rest of MAGES folders in step 2. We will download and reimport the photon PUN asset in our case.

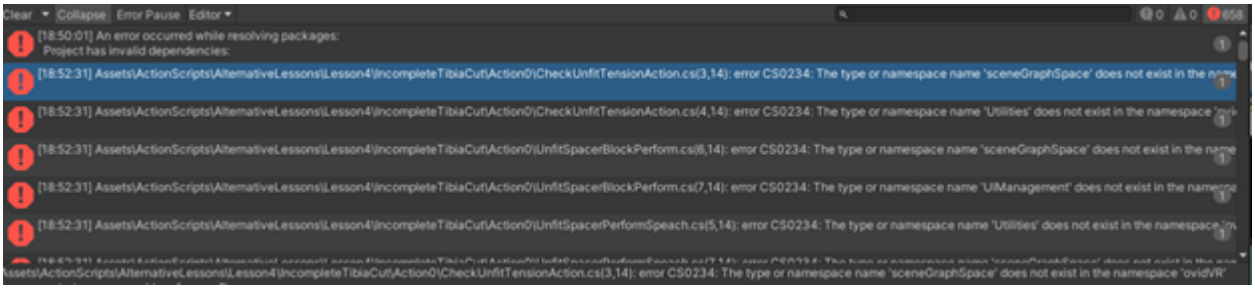


Also if you have any conflicts like this, make sure you resolve them and keep the ones from the MAGES package folders, not the ones in your Asset folder.

In your case, fix any error to move to the next step.

6.13.5 Step 5: Fix compile errors (Renaming)

More errors will pop. In this case it will be due to the renaming of our libraries. Double click an error from the Unity console to open the visual studio.



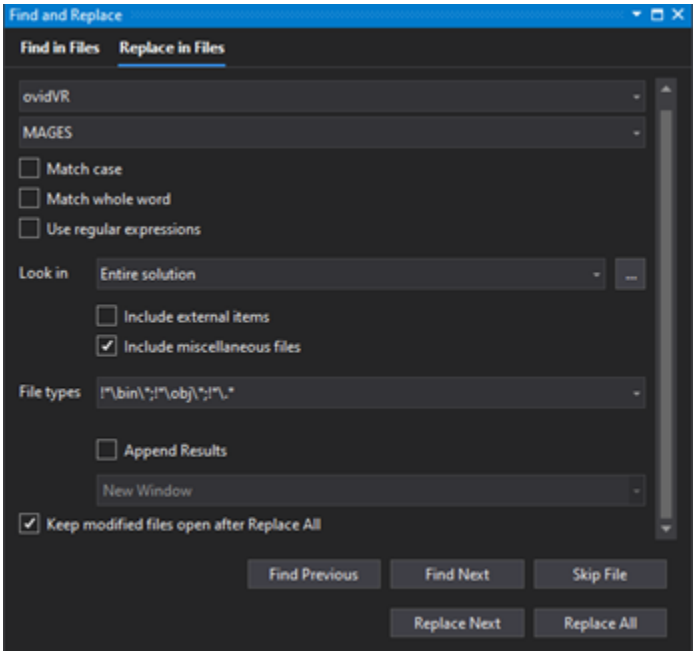
On visual studio, press CTRL + SHIFT + H to open the Find and Replace window and replace the following words across the project (press the replace all option):

“ovidVR” -> “MAGES”

“MAGES.toolManager” -> “MAGES.ToolManager”

“toolGameobject” -> “ToolGameobject”

“).isActive” -> “).IsActive”



Press CTRL + SHIFT + S in visual studio to save all the changes.

6.13.6 Step 6: Fix compile errors (Final fixes)

Now that we have finished renaming we need to fix some more errors. You may not have some of them but in any case here they are

1. If you have any Action that inherits the IAction Interface you will see an error that says your class is not implementing the IAction interface. Hover on the IAction, press "Show potential fixes" and click the "Implement Interface" option. This was occurred since we added the DestroyAction() method in the interface.

2. SetPhysicalColliderPrefab is no longer user. Use the Spawn() method instead but remember to delete the spawned prefab on Perform() and Undo() if needed.

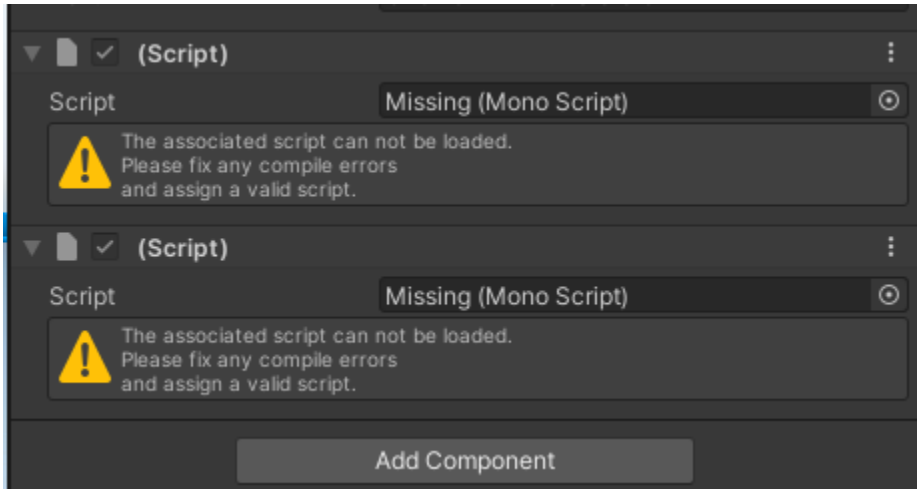
3. Also anything related to the AlternativePaths and Parallel Actions is deprecated and removed. Consider using OptionalActions.

Here is a tutorial on how to use Optional Actions:

After fixing all the errors the Unity project will reload itself.

6.13.7 Step 7: Fix missing references

At this point we don't have any compile errors. However, you will see that almost all of the prefabs have missing references. This is caused since we renamed many functions and namespaces in MAGES dlls.



To fix this issue we build an external tool that will do all the work for you.

Script: https://www.dropbox.com/s/kbvc2r1kk44lxg8/MAGESUpgradeFix.cs?dl=0

Add this script to your asset folder. Open it and change the values of the variables **projectPath** and **prefabsPath**. The variable projectPath contains the path to your project (the folder that contains the Asset folder). The prefabsPath variable contains the path to your prefabs you want to fix the references.

```
public class MAGESUpgradeFix : EditorWindow
{
    static string projectPath = "C:/GitHub/TKA/TKA_Remaster/";
    static string prefabsPath = "Assets/Resources/LessonPrefabs";
```

Let the Unity compile it and then go to the MAGES top bar menu and click the "Fix missing references" button. This will fix all the missing references for you.

However, since the "SCENE_MANAGEMENT" Gameobject in the scene is not linked with a prefab will still have missing references. To fix it we propose to take it from one of our sample applications. To import a sample application navigate to the package manager. More info *here*.

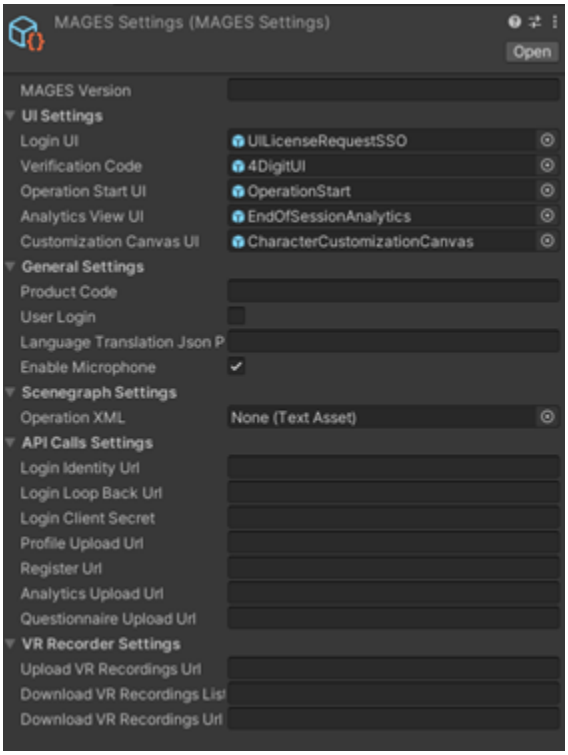
After adding the new SCENE_MANAGEMENT object you need to delete your cameras and create them again from the MAGES top bar menu since the references on the SCENE_MANAGEMENT will be flushed.

Make sure that the MAGESControllerClass component on the MAGESDeviceController is properly set. Otherwise add the missing hand references.

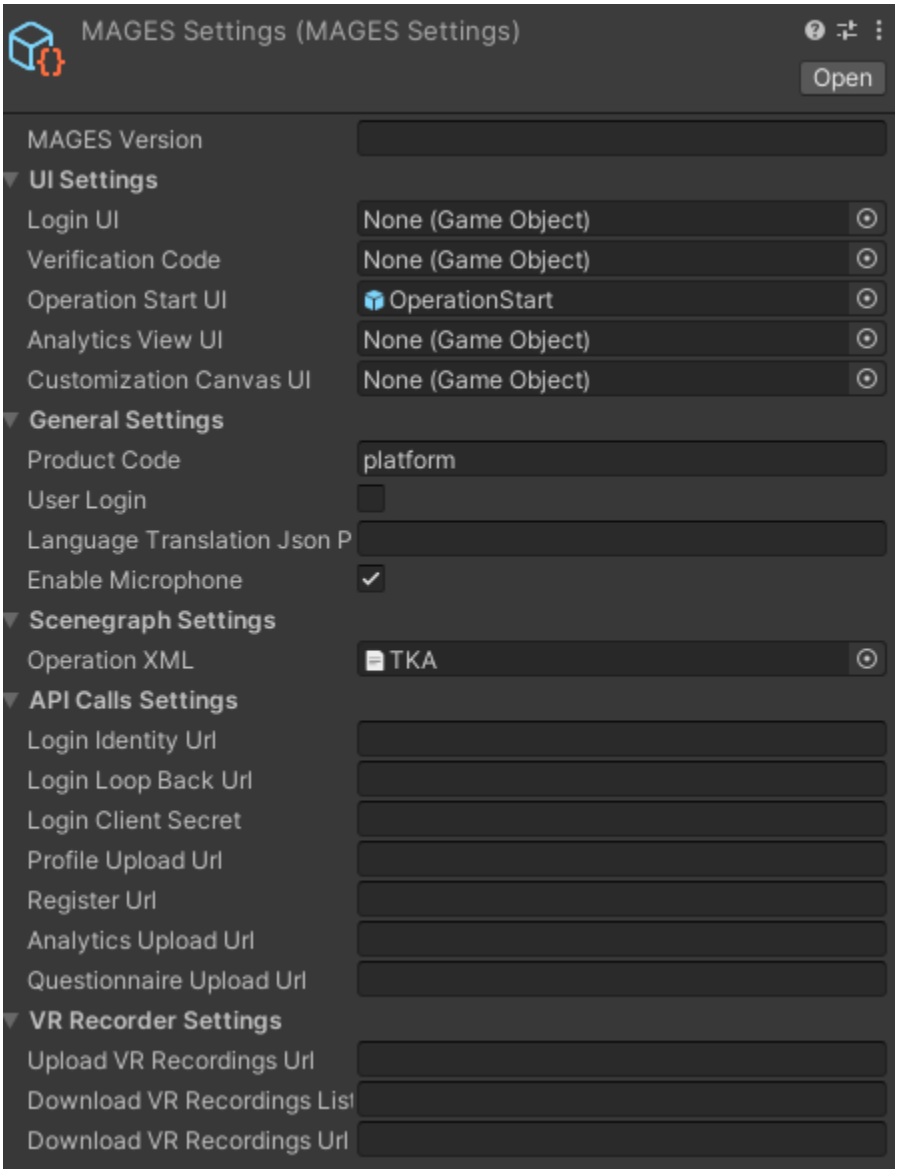
6.13.8 Step 8: Generate Asset file

The next step is to generate the MAGES Settings asset file. From MAGES 4.x we use this asset file to store all the essential links and some prefabs. Navigate to the MAGES top bar menu and click "Generate MAGES Settings File", then select the location to save your file. This will generate an .asset file.

At the beginning it will be like this:



It is important to drag and drop your XML file in the "Operation XML" field. You can also add the product code. In our case it will be "platform". In this case the XML should be saved in the following path: AssetsResourcesStoryboard**platform**. It is also important to drag and drop your prefabs at the beginning of the asset. If you have a different Operation Start UI or a customization Canvas UI for example drag and drop your own into those fields. The final result will be similar to this one.



Then click the "SCENE_MANAGEMENT" Gameobject in your scene and at the inspector you will see a MAGES Setup scrip. Drag and drop the MAGESSettings.asset we created to this field.

6.13.9 Step 9: Login

Before log in you need to delete the previous account folder. It is located in the Assets/Resources/Storyboard/Account folder. Delete the Account folder.

Then from the MAGES top bar menu click the MAGES Helper and then login from this window by inserting your credentials.

6.13.10 Step 10: Upgrade the XML

The final step is to update the XML format as from MAGES 4.x we updated the formatting of the xml. First open the scenegraph editor from the MAGES top bar menu and load your old xml. All you have to do now is to click File > Save and it will be saved with the new format.

From the MAGES top bar menu click the MAGES Helper button and click to any remaining jobs to properly configure your project.

You are ready to go!

6.14 VR Annotations

The VR Annotations are responsible for creating static or object-following Annotation Labels with ease.

6.14.1 How To

In this tutorial we will explain how to use the **VRAnnotator** and how to create standalone Annotation Labels using the **AnnotationLabel** script.

Use the Default Annotator Prefab

In the folder **Packages/com.oramavr.mages/Runtime/MAGES/SDKAddons/VRAnnotations/Prefabs** you can find the **AnnotationMarker.prefab**.

Note: The **Packages** folder is in the same level as the **Assets** folder. An easy way to locate the **AnnotationMarker.prefab** would be to use the search function of Unity, located in the **Project** tab. (You may have to click the "All" tag after your search in order to locate the prefab.)

To use it just drop it in your scene, or if you are creating an Operation, spawn it from its action scripts.

Below we describe the main functionalities of the marker:

Create new static Annotation

Take the marker and press the trigger button. Then with the raycast, click the text area and type your label. Press the "Save" button to save the label.

Reposition the Annotation

With the marker, touch the reposition icon and press the trigger button. Then to release the Annotation press the trigger button again.

Edit the Annotation text

With the marker, touch the edit icon and press the trigger button. Then with the raycast, click the text area and type your label. Press the "Save" button to save the label.

Hide the Annotation

With the marker, touch the edit icon and press the trigger button. Then with the raycast, click the hide button. To un-hide click again the hide icon.

Delete the Annotation

With the marker, touch the edit icon and press the trigger button. Then with the raycast, click the delete button.

Create new object-following Annotation

If the Annotation Marker's tip was inside a grabbable object (non-static object in the GrabbablePrefabs Layer) the label will move with the prefab. Otherwise, the label will be in a static world position, at the spawned place.

Note:

- **Buttons outside of the Annotation Editor** are pressed using the marker, moving it to a close distance (the button gets highlighted) and then pressing it with the trigger on your controller.
- **Buttons inside the editor** are standard UI buttons and can be used with the raycast.

Create a Custom Annotator

In order to create a custom Annotator similar to the marker above, you need to follow these steps:

1. In the GameObject you want to be your Annotator **attach a collider and make it trigger**. This is needed because the script needs to check the `void OnTriggerEnter()` and `OnTriggerExit()` to determine whether the label to be spawned is attached to a grabbable prefab, and therefore will move with it, or not.
2. In the same GameObject **attach the VRAnnotator script**. This script will be initialized with default values for the LabelSpawnPoint(The transform of the same Attached GameObject) and the LabelRotationMethod(AutoHorizontal). You can change these values now. The **LabelSpawnPoint** field signifies the point at which the labels will be spawned and also clamped when you are moving them. The **LabelRotationMethod** field is the setting for the rotation of the labels. If set to AutoHorizontal the labels once created and done editing will rotate automatically around the local y axis to always face the camera head. If set to Initial

---

Note: Keep in mind that when attaching the VRAnnotator script the tag of the attached GameObject will be changed to **AnnotationMarker**. It needs to stay that way in order for the Annotator to work.

---

3. The Annotator now works by grabbing it and pressing the trigger button.

Note: Note that it is not possible to change the label that the annotator spawns. If you want to make appearance changes to that simply edit the **Label** prefab, located in the package.

Create a Standalone Label in the Editor

With the MAGES Annotation System it is easy to create a Label in the Editor and make it a scenery component. Simply Drag and Drop the **StandaloneLabel** Prefab in your scene, from the folder **Packages/com.oramavr.mages/Runtime/MAGES/SDKAddons/VRAnnotations/Prefabs**. Find the child **Text** in the **StandaloneLabel > DisplaysOrigin > DefaultLabel > Canvas > Label** GameObject in hierarchy. You can edit the text component on this GameObject to edit the Annotation Label's Display Text.

Note:

- You can make the Label follow a transform, by simply assigning it to the inspector field `Transform transformParent` and setting the enum `PlacementMethods placementMethod` to **Parent Transform Follow**.
- You can also select the RotationMethod of the Label by setting the `RotationMethods rotationMethod` field in the inspector to **Auto Horizontal to always look at the camera** or **Initial Rotation to not rotate at all**.

6.15 VR Slider

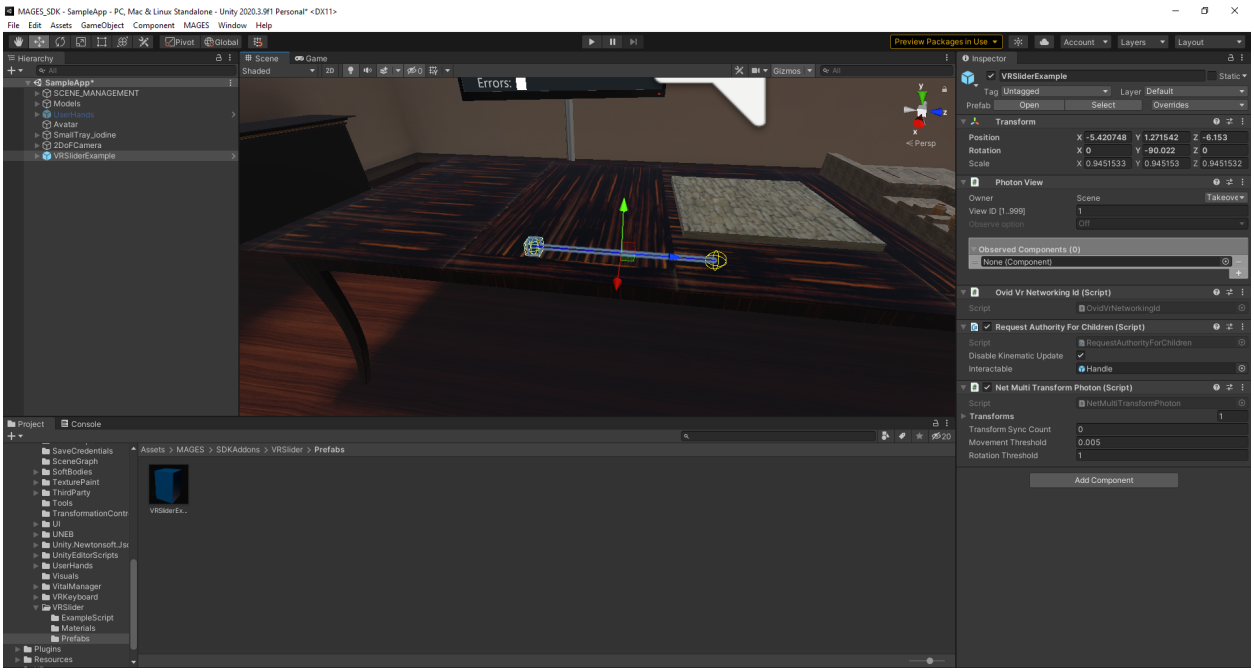
The `MAGESSlider` script is responsible for creating a VR interactive Slider with ease. The MAGES SDK provides an already configured generic VR Slider Prefab, but the VR Slider component is easily modifiable to meet any developer's need.

6.15.1 How-To

Below there are tutorials not only on how to use the default Slider Prefab included in the MAGES SDK, but also how to create one yourself and customize it according to your likings.

6.15.2 Using the existing VR Slider Prefab

In order to use the default VRSliderExample Prefab, found in Packages → com.oramavr.mages → Runtime → MAGES → SDKAddons → VRSlider → Prefabs, simply drag and drop it in your scene.



The Slider should now be working and be interactable both with the ray and the hands. In order to get/set the slider's value you can read/write the `float sliderValue` variable of the `MAGESSlider` script.

> **Warning:** The `float sliderValue` variable of the `MAGESSlider` script must be a value from 0 to 1.

Note: Remember you can toggle the raycast using the following command:

```
InterfaceManagement.Get.InterfaceRaycastActivation(bool);
```

The raycast can be used to interact with the slider. Make sure you have enabled the InteractWithRaycast value from the InteractableItem component.

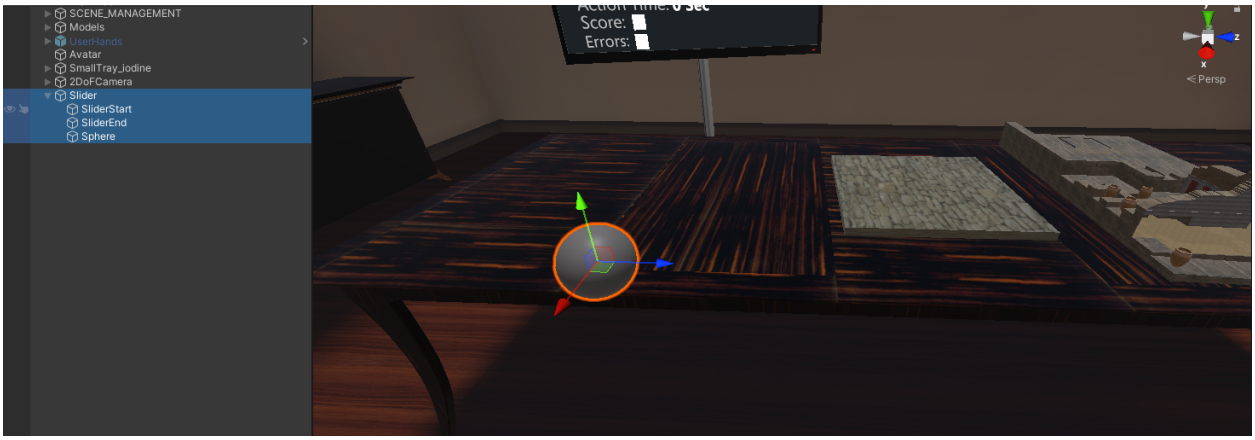
6.15.3 Creating A Custom VR Slider

In order to create a Custom VR Slider we first need to create some GameObjects.

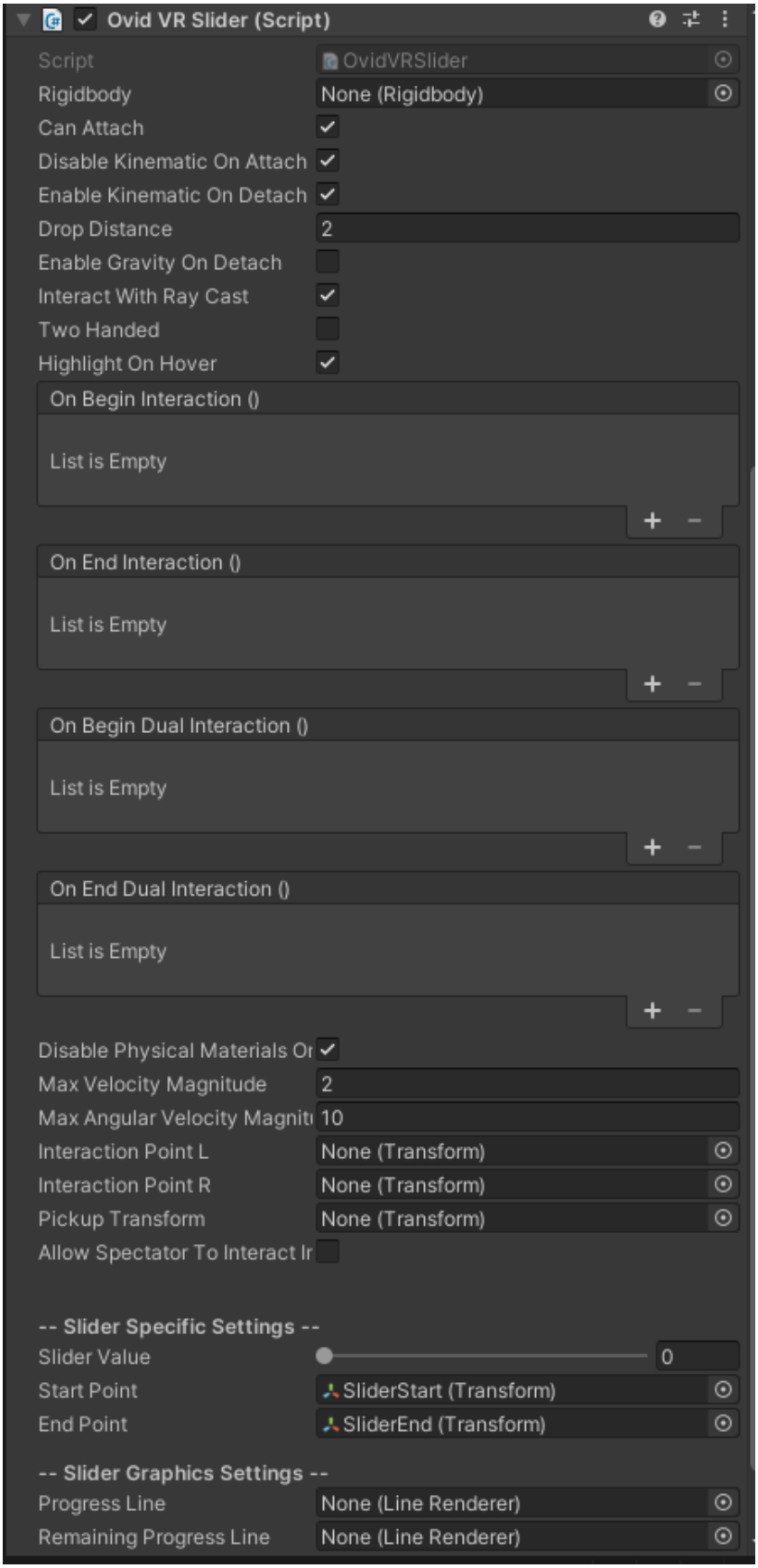
1. First we need 2 Transform references for the Slider Start position and the Slider End Position.
2. We need a GameObject which will represent the handle of our slider. Here we have used a Sphere Mesh with a sphere Collider for our handle.

Note: We created 2 Empty GameObjects (SliderStart and SliderEnd), which will be used for the Transform References, in order to signify the start and end positions of the slider. The Handle is the actual handle the user has to

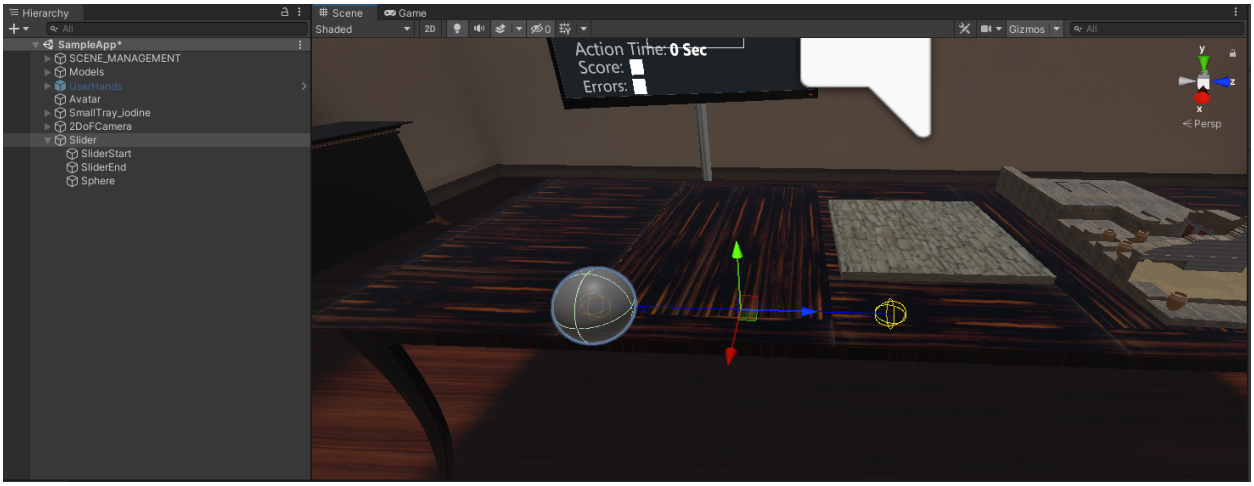
grab to move the slider. I have also created a parent Empty GameObject Called Slider in order to wrap all the slider components and make them easier to manipulate.



3. Finally we attach the `MAGESSlider` script to our handle gameobject and attach the SliderStart and SliderEnd Transforms to the **Start Point** and **End Point** fields in the inspector.


Ovid VR Slider (Script)
Script
OvidVRSlider
Rigidbody
None (Rigidbody)
Can Attach
Disable Kinematic On Attach
Enable Kinematic On Detach
Drop Distance
2
Enable Gravity On Detach
Interact With Ray Cast
Two Handed
Highlight On Hover
On Begin Interaction ()
List is Empty
On End Interaction ()
List is Empty
On Begin Dual Interaction ()
List is Empty
On End Dual Interaction ()
List is Empty
Disable Physical Materials Or
Max Velocity Magnitude
2
Max Angular Velocity Magniti
10
Interaction Point L
None (Transform)
Interaction Point R
None (Transform)
Pickup Transform
None (Transform)
Allow Spectator To Interact Ir
-- Slider Specific Settings --
Slider Value
0
Start Point
SliderStart (Transform)
End Point
SliderEnd (Transform)
-- Slider Graphics Settings --
Progress Line
None (Line Renderer)
Remaining Progress Line
None (Line Renderer)

Note: As soon as we attach the **Start Point** and **End Point** transforms in the inspector we shall see gizmos showing the slider's range in the scene. We can also try changing the **sliderValue** from the inspector and see the handle move across the slider line in the scene.



4. We are now ready to use the slider both with raycast and the hand.

6.15.4 Adding Slider Graphics

In order to add graphics to the slider line we will use 2 line renderers. For simplicity, instead of creating new gameobjects, we can attach them in the already existing SliderStart and SliderEnd gameobjects. Then we can attach them in the `MAGESSlider` script in the inspector fields **Progress Line** and **Remaining Progress Line**.

Note: The line renderer attached in the **Progress Line** script will represent the slider line portion from the start point to the handle, when the line renderer attached in the **Remaining Progress Line** script will represent the slider line portion from the handle to the end point. These line renderers will update automatically to cover the span of the mentioned line portions.

6.15.5 Inspector Fields

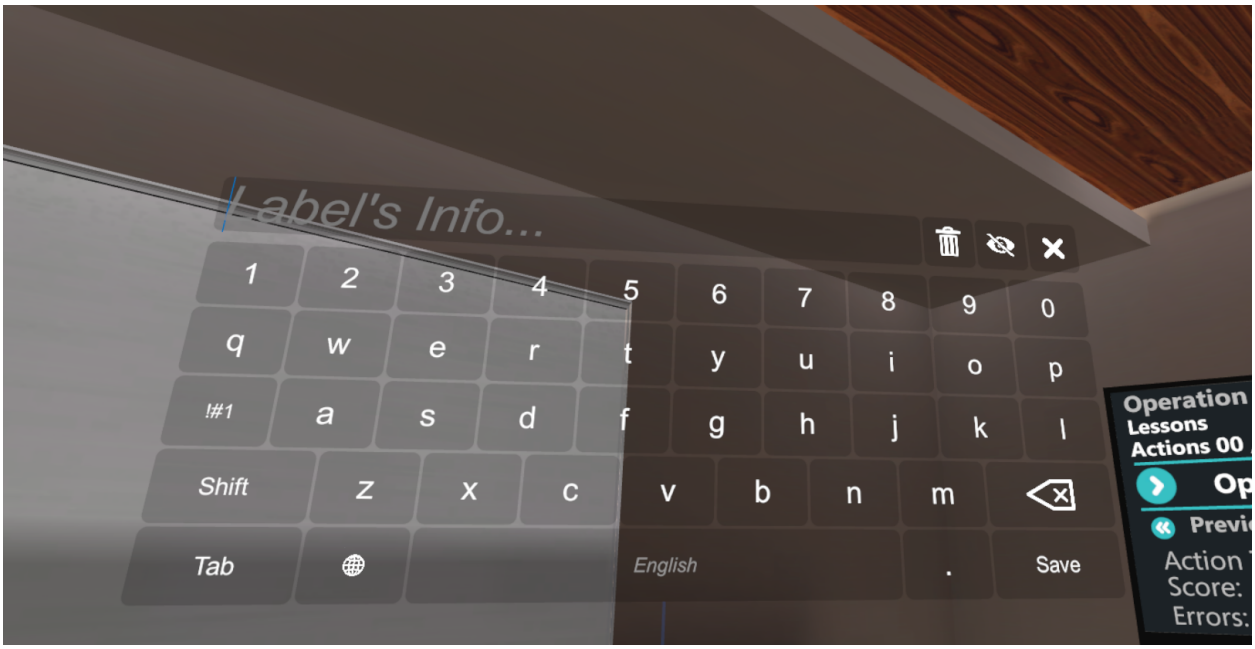
Below there is the table of the inspector extra slider parameters and what they do.

`class MAGESSlider :: MAGESInteractableItem`

Parameter	Description
`bool InteractWithRaycast`	Is the slider handle able to be used with the raycast?
`float sliderValue`	The position of the handle between `Transform startPoint` and `Transform endPoint`. It has a min value of 0 (inclusive), when the slider is at `Transform startPoint`, and max value of 1(inclusive) when the slider is at `Transform endPoint`.
`Transform startPoint`	The transform of the start position of the line segment the slider will move in.
`Transform endPoint`	The transform of the end position of the line segment the slider will move in.
`LineRenderer progressLine`	The LineRenderer that will be used to display the line segment from `Transform startPoint` to the handle's current position.
`Linerenderer remainingProgressLine`	The LineRenderer that will be used to display the line segment from the handle's current position to the `Transform endPoint`.

6.16 VR Keyboard

The MAGES SDK has a versatile, customizable and easy to use VRKeyboard that fits every need. It supports custom languages as well as layouts and can be attached in any GameObject. Below there is a How To Use and a Customization section.



6.16.1 How to Use

In order to use the VRKeyboard you must follow the steps below:

1. Drag and Drop the VRKeyboardFull prefab in your scene from the **Packages** → **com.oramavr.mages** → **Runtime** → **MAGES** → **SDKAddons** → **VRKeyboard** → **VRKeyboard** → **Prefabs** folder.

Note: You can either place the keyboard at a desired static position or as a child of a GameObject.

This Prefab has the `VRKeyboardController` script attached, which is responsible for the Keyboard Interaction and Settings

2. Call the `bool Initialize()` on the Start Function of your custom script. This will set the Keyboard's Layout to the **first KeyLayout** in the **list KeyLayouts** shown in inspector.

Warning: If the `Intialize()` function **returns false**, it means that there is a problem with the attached Key Layouts, check them in the inspector and fix it to continue.

Note: If you need to attach extra **KeyLayouts** in the VRKeyboard do it before calling the `void Initialize()` function.

---

3. Give the string you want to edit to the keyboard using the `void InitializeInputBuffer(string inputBuffer)` function. This will pass the string to the keyboard, and it will set the caret position at the end of it.

Note: Alternatively, you can set the input buffer using the `void SetInputBuffer(string inputBuffer)` which will only set the string you want to be edited by the keyboard. Then you can use the `void SetCaretPosition(int position)` to set a custom caret position in the string. (The position is clamped from 0 to the length of the string in the input buffer)

---

4. Use the `void SetSubmitText(string text)` to modify the enter button's text according to the application's needs. This step is **optional** and if not done the default text **"OK"** will appear.
5. If not already done through the editor, you can attach here the UnityAction Events to be called in `UnityAction OnSubmit()`, `UnityAction OnTab()`, `UnityAction OnAnyTextKey()` and `UnityAction OnAnyKey()` actions.
6. Finally, you can use the `void GetInputBuffer()` function to retrieve the new/edited string.

Note: In order to improve CPU performance this can be called when the `UnityAction OnAnyTextKey()` is invoked, instead of every frame.

6.16.2 Keyboard Customization

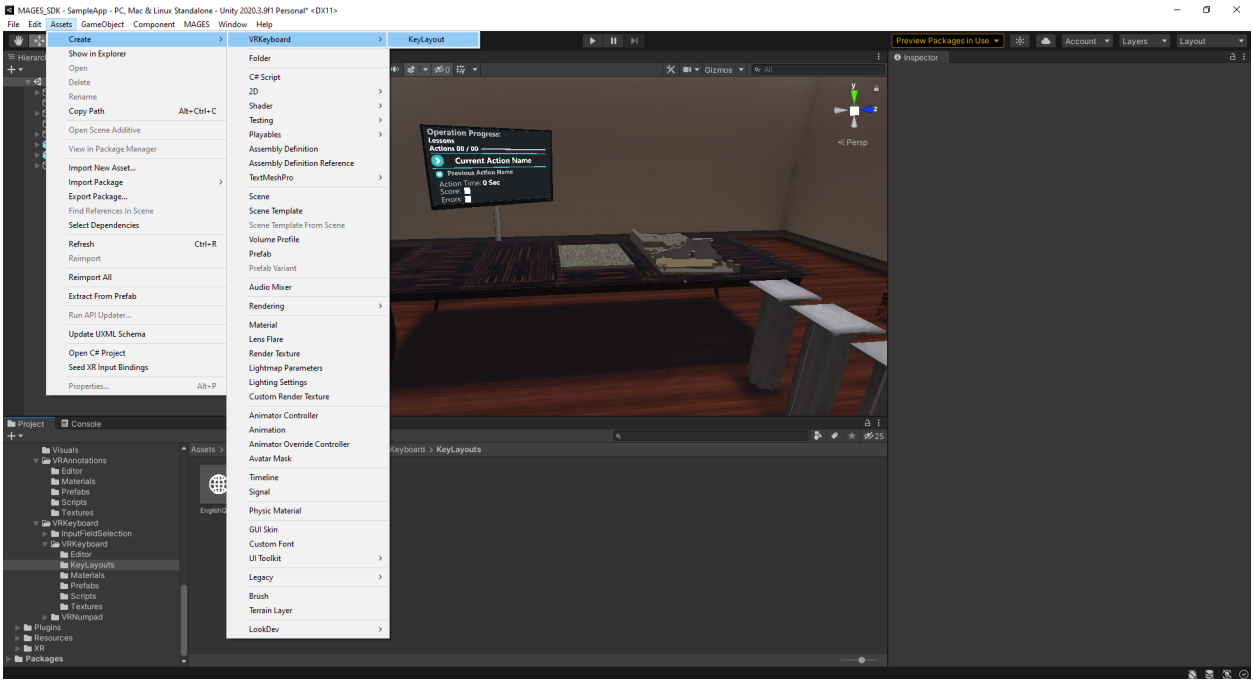
Customizing the Look and Feel

The keyboard can be customized for button looks as well as sound and haptic feedback. On the text field of every key on the keyboard there is the `VRKeyHighlightAndHaptic` script. This is responsible for the sounds, haptics and transitions.

Adding Custom Layouts/Languages

Through our KeyLayout Scriptable object the MAGES SDK has you can easily create new Layouts for the VRKeyboard. In order to create a new Layout follow the steps below:

1. Go to the ToolBar and click on Assets → Create → VRKeyboard → KeyLayout. This will create a new empty KeyLayout.



2. Go to the Inspector and edit the **KeyLayout**.

Note:

- The field `string layoutName` is for the name you want to be displayed in the spacebar. Usually used for language name (e.g. “English”, “Greek”)
- The field `List<string> mainKeys` is the list holding all the letter bindings of the main keys.
- The field `List<string> shiftMainKeys` is the list holding all the letter bindings of the main keys when shift button is enabled.
- The field `List<string> alternativeKeys` is the list holding all the letter bindings of the alternative keys accessed by pressing the “#!1” Button and in page 1.
- The field `List<string> alternativeKeys` is the list holding all the letter bindings of the alternative keys accessed by pressing the “#!1” Button and in page 2.

- You can also import the key bindings from a txt file. Switch to the **Import Utilities** Tab and add the file path as well as the separator in the fields. Then Click **Import From File**.

> **Warning:** The keys must be assigned in the order you want them to be displayed. The script assigns them line-by-line from left to right in the blank spots of the VRKeyboard prefab.

3. Attach the new `KeyLayout` created to the `VRKeyboardController` script found in the VRKeyboardFull prefab instance you have on the scene. This will enable the new prefab, and it will be cycled through using the language change button of the VRKeyboard.

Note: By default the VRKeyboard has two KeyLayouts with English and Greek letters.

6.17 VR Recorder

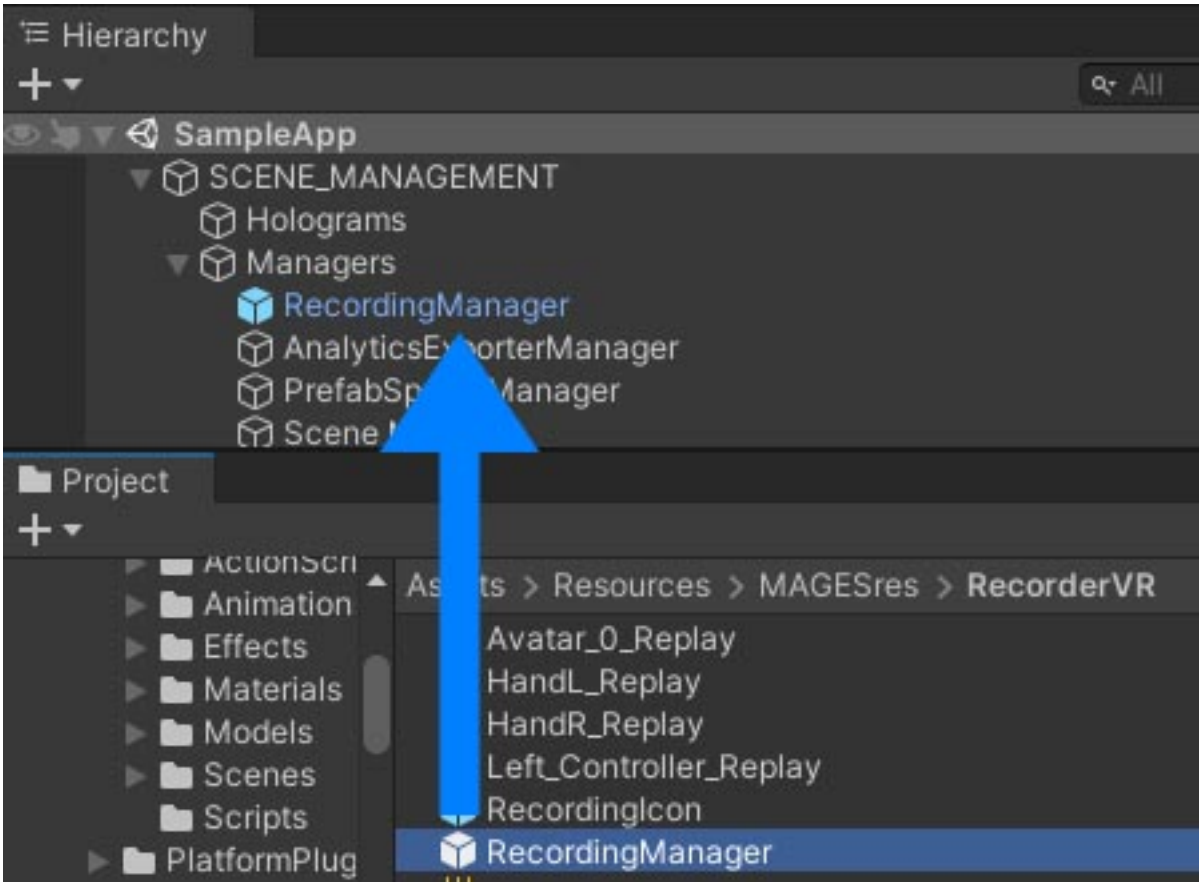
6.17.1 How to Use

The MAGES SDK offers a unique solution for recording your past Operations and reviewing them after they are over. VR Recorder is a functionality that can record and replay an Operation, in both **Single and Multi player** Modes. Recordings can be synchronized with the cloud and are **replayable on any device** regardless of the original hardware they were recorded on. These are not just video recordings of your in-game view but rather **a full recreation of the Operation** as it happened when it was recorded. When replaying you are **free to move around** the Operation Room and watch from any angle you like, as well as **hear your voice and your colleagues.**

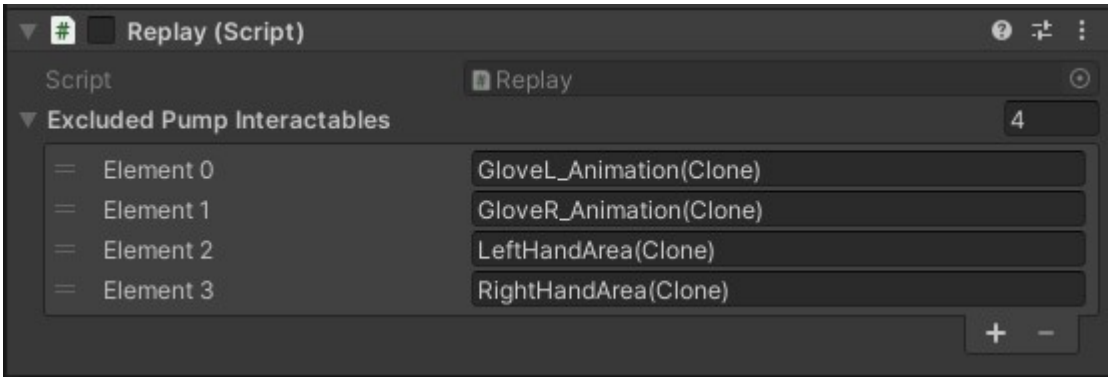
Recording Manager

The Recording Manager Prefab contains all necessary scripts to enable the functionality of the VR Recorder. Its presence in the scene is required for enabling both Recording and Replaying Operations.

You can find the prefab in the folder **Assets/Resources/MAGESres/RecorderVR** . Place it in the scene under **SCENE_MANAGEMENT/Managers**.

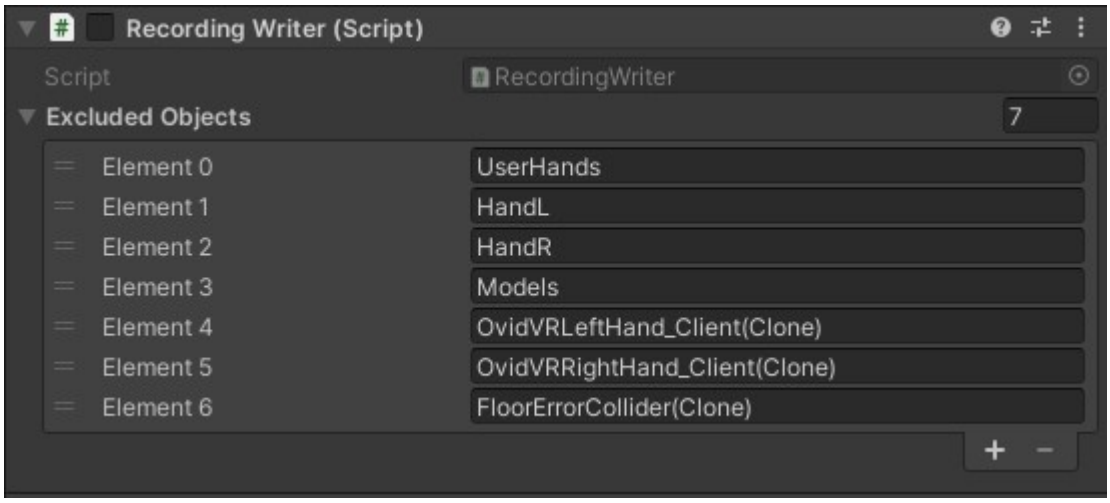


Excluded Pump Interactables



Some Pump Interactables are used as animation controllers and their progress is controlled via script rather than via user Input. These Interactables have to be explicitly listed here, so that they operate correctly during Replaying.

Excluded Objects



Some GameObjects that are not essential to the recording can be ignored. These can be added to this list, and interactions all users have with them will not be recorded.

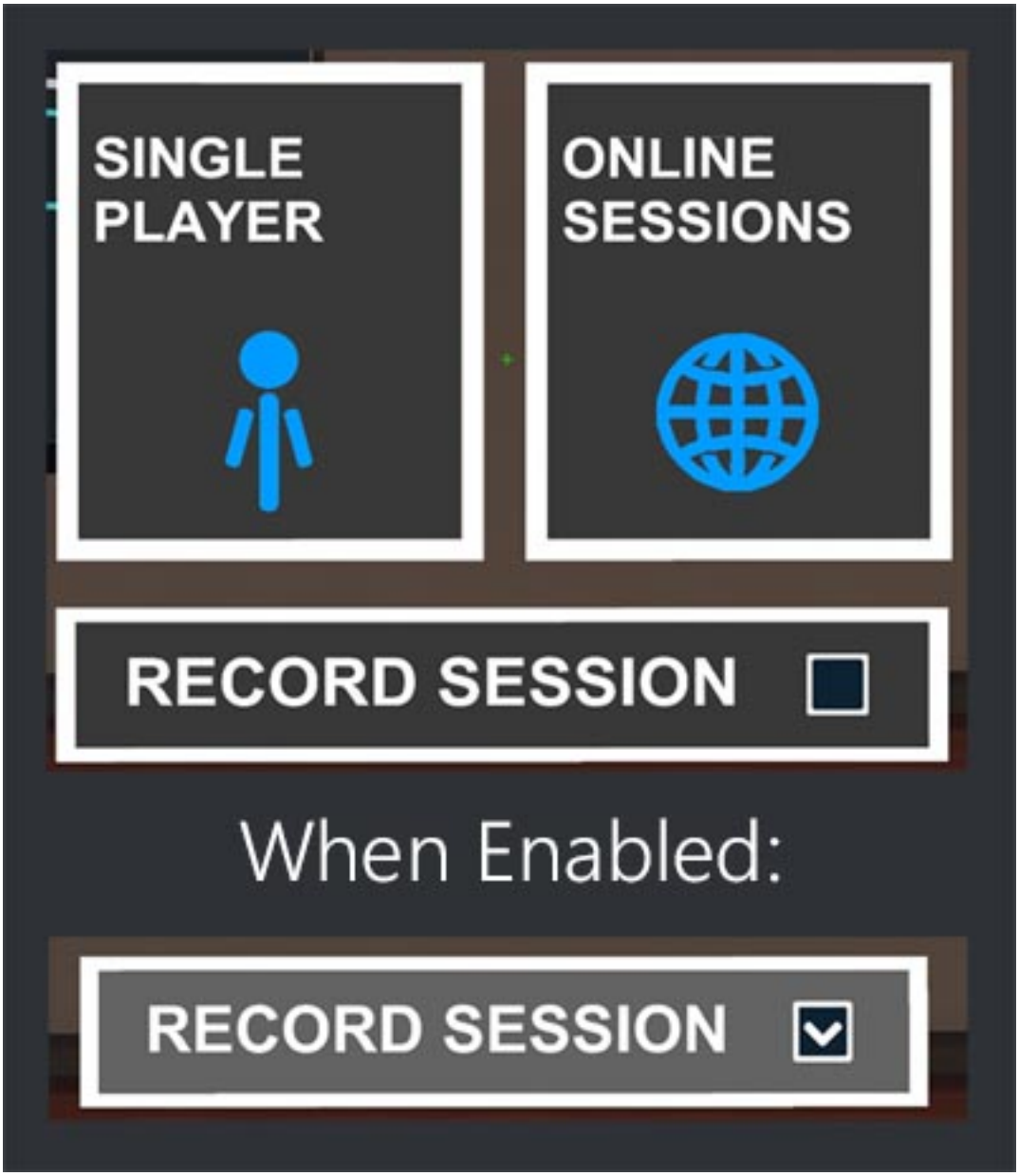
Warning:

Known Issues:

1. Game Objects with names that begin with '<' and end with '>' (E.g. "<ObjectName>") will create issues when replayed.
2. Interactables with Drop distances bellow 0.5 cause repeated Interaction Starts and Ends, slowing down replay and de-synchronizing the audio.

Recording

In the starting menu, there is now the option of enabling Recording for both Single Player and Multi Player sessions. For Multi Player sessions, it does not matter if you are creating a session or joining one.



During the operation, you can see an indication on your left wrist that Recording is enabled for this Session.

Warning: Game Objects with names that begin with '<' and end with '>' (E.g. "<ObjectName>") will create issues when replayed.

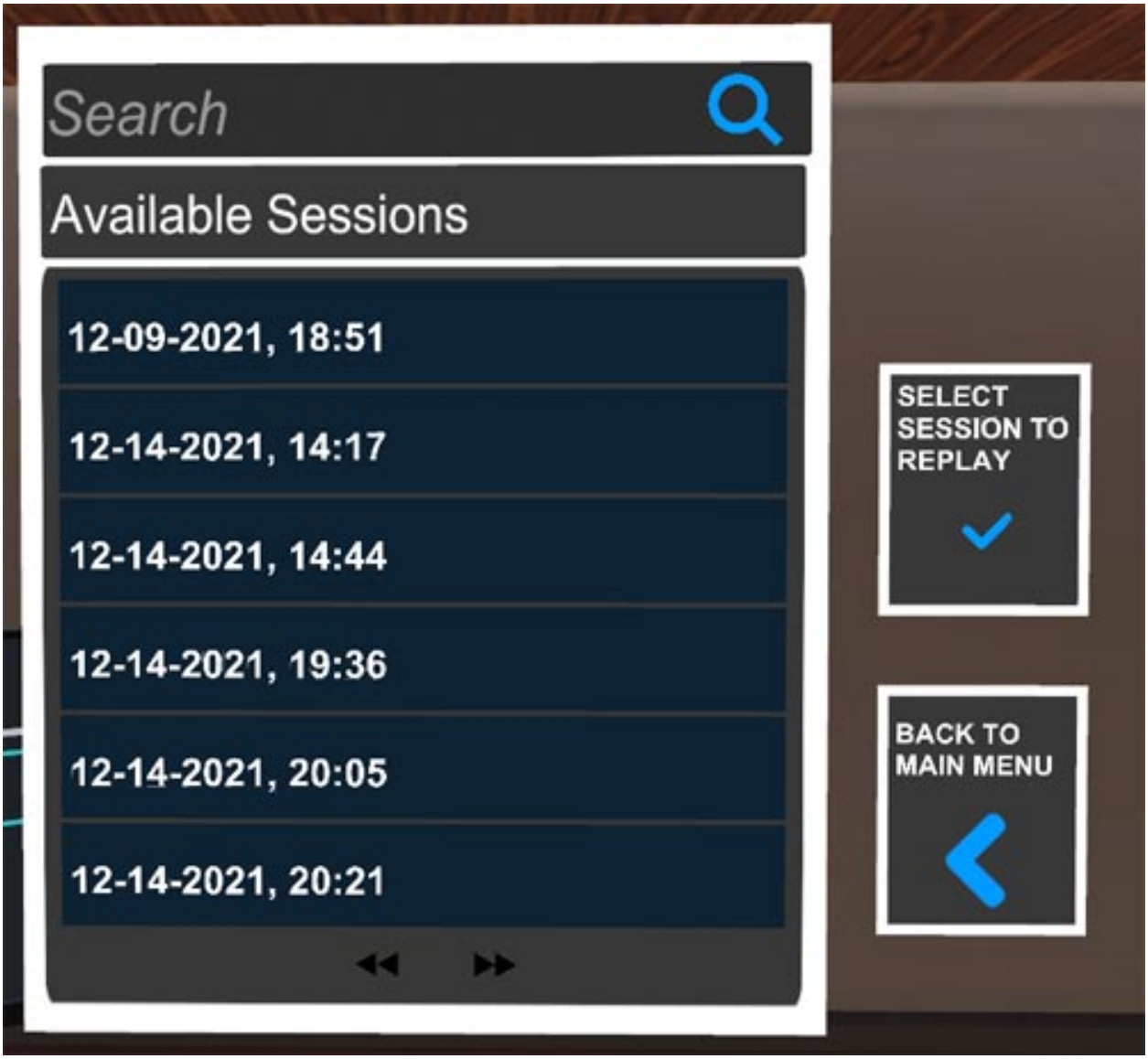
Note: The recording will be saved even if you quit the application without completing operation.

Replaying

Accessing Recordings

The user can access their recordings using the Operation Start Menu option.

When pressed, a list of all recordings will be shown, with the option to either replay one of your choosing, or return to the main menu. You can search for recordings using the search bar, or scroll through the pages using the arrows at the bottom of the UI.



Watching a Replay

When watching a replay, an indication that you are in a Replay Session is shown in your Left wrist. You can move around as you would in a normal Operation, but you will not be able to interact with any objects in the Scene.

Cloud Synchronization

Uploading Recordings

When a session is recorded and the Operation reaches its end, the following progress bar will appear. Do not close the application until it is completed, otherwise the recording will not be uploaded.

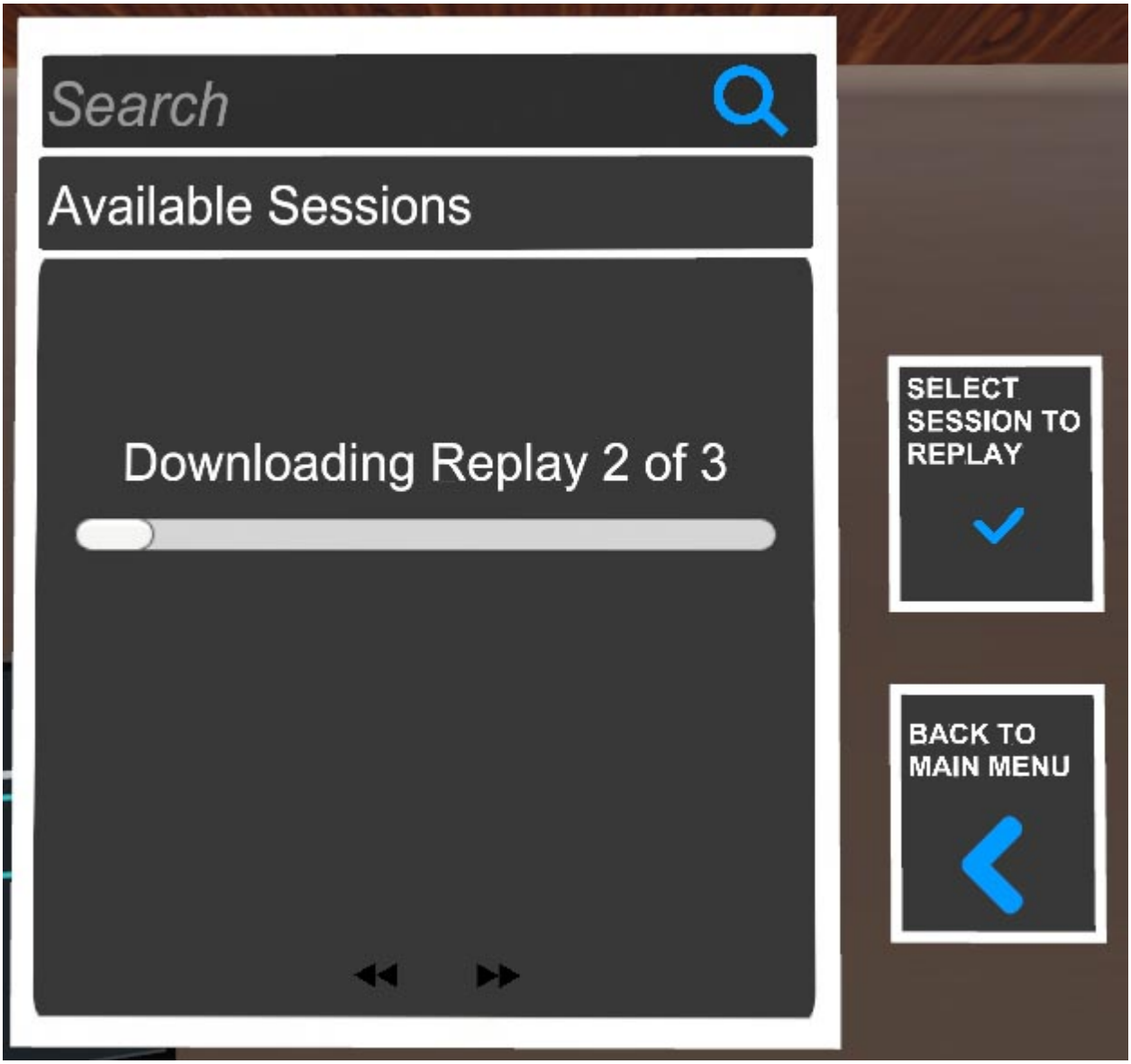
The recording is now on the cloud and will be available for download.

Note: Only operations that reach Operation End will be uploaded. All other recordings will be available locally for

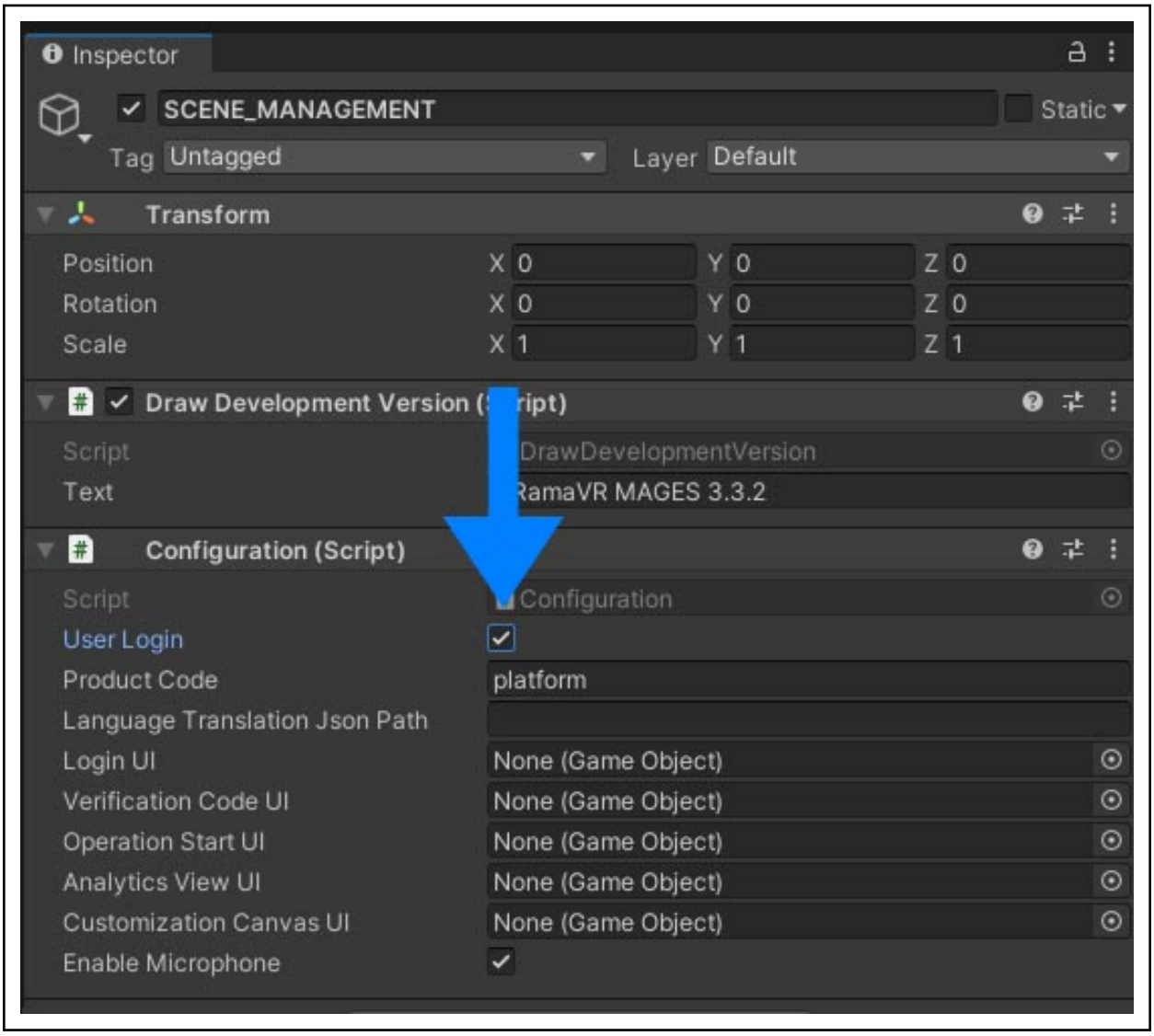
the device they were recorded on.

Downloading Recordings

When opening the Recording selection menu, all recordings from the cloud not present locally will be downloaded and become available for Replaying.



Warning: Uploading and Downloading recordings require the user to be logged in to their ORamaVR account. If you are getting HTTP 500 Errors when attempting upload or download, make sure User Login is enabled in your application.



6.17.2 Record & Replay Custom Events

During the development process of a new simulation with the MAGES SDK, there are cases where some events are not replayed. This is caused by the fact that the functionality which is under development is not recorded correctly. This happens because the feature is custom-made and does not use a script or an object provided by the SDK. In that case, one can utilize the **CustomRecord.cs** and **CustomReplay.cs** scripts.

Record Custom Messages

In order to record a specific event one has to create a script that inherits from the CustomRecord script. In the following code example one can see how a custom message is recorded:

```
using MAGES.RecorderVR;
using System.Collections;
using UnityEngine;

public class MyAppCustomRecord : CustomRecord
{
    void Start()
    {
        AddMessage("PRINT");
        StartCoroutine("CustomRecordMethod");
    }

    IEnumerator CustomRecordMethod()
    {
        yield return new WaitForSeconds(10);
        WriteMessage("PRINT", "Custom_Event_1");
    }
}
```

In the Start method the custom messages have to be registered. In that example we create a new message called "PRINT". One has to register all the custom messages in the start of that MonoBehaviour script. Afterwards, we call a coroutine that after 10 seconds from when the Start method is executed it writes a new message. Keep in mind that **WriteMessage** can be called by any method that is executed in your simulation. So, one can call it by writing:

```
GameObject.Find("RecordingManager").GetComponent<MyAppCustomRecord>().WriteMessage(
↪"PRINT", "Custom_Event_2");
```

In that example we have written two different messages in the VR Recorder's files. Both messages are of the same type, i.e. "PRINT". Thus, when in replay, the same method will be called when these two messages are read. However, the messages have two different signals, which are the second arguments of the **WriteMessage** method. The signals are used as an argument to the CustomReplay's method that is associated with a specific message. Note, that at the moment the VR Recorder can understand only one signal of type **string**. Finally, if you don't want to write a signal, just omit the second argument of the **WriteMessage** method.

Replay Custom Messages

In order to replay a specific event one has to create a script that inherits from the CustomReplay script. In the following code example one can see how a custom message is replayed:

```
using MAGES.RecorderVR;

public class MyAppCustomReplay : CustomReplay
{
    void Start()
    {
        AddMessage("PRINT", PrintMessage);
    }

    void PrintMessage(string arg)
    {
        print(arg);
```

(continues on next page)

(continued from previous page)

```
        }
}
```

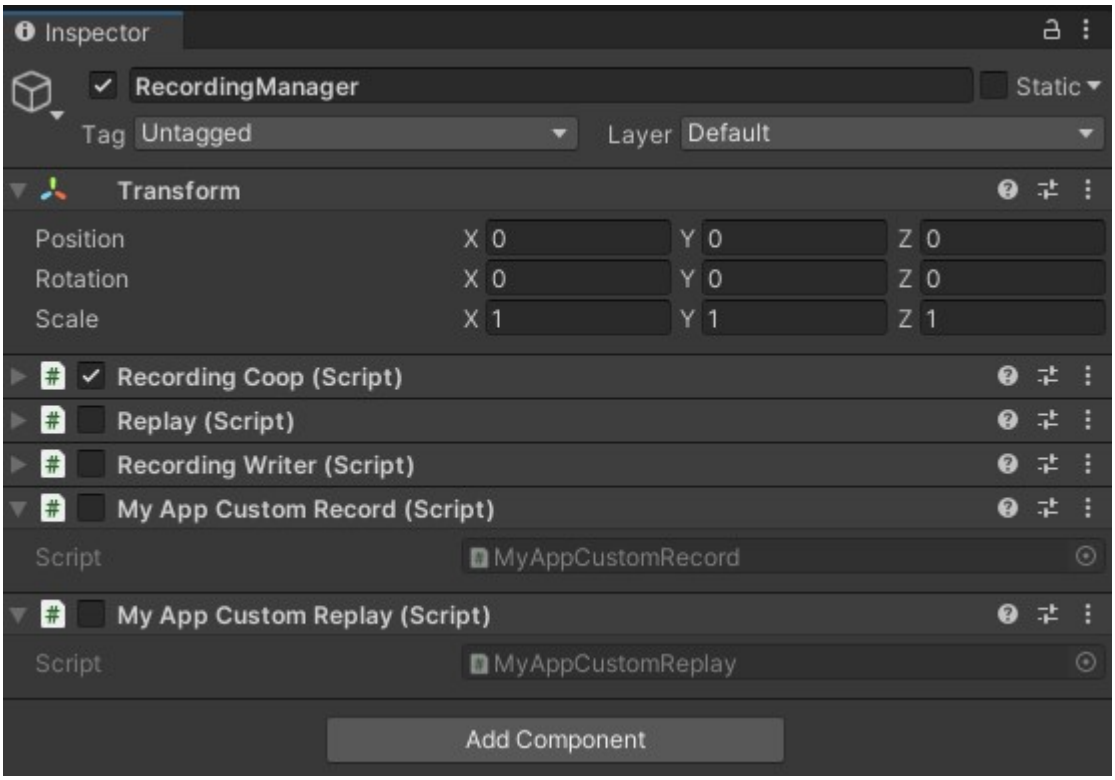
In this CustomReplay script we register the custom messages we created, the same way we did for the CustomRecord script. This time, the second argument of the **AddMessage** method is another method, called **PrintMessage** in our example. This method will be called when the Message "PRINT" is read from the recorded files.

In our example the **PrintMessage** method just prints the signal of the message that was recorded. Thus, the string "Custom_Event_1" will be printed aproximately after ten seconds from when the replay starts. Keep in mind that the frame rate when the session is recorded is different from the frame rate when it is replayed. This is the reason why the string will be printed approximately after ten seconds.

In your methods linked with the new custom messages in your CustomReplay class, you write the lines of code you want to be executed when the messages is read. Those lines are almost the same with the ones executed when the message is written in recording mode. Finally, if you didn't send a signal when you recorded a message don't use the **arg** parameter of the method. However, a string argument has to be written in the parameters due to the way that the VR Recorder handles custom messages.

Integrate the scripts into the RecordingManager

The **MyAppCustomRecord** and **MyAppCustomReplay** have to be added at the **RecordingManager** GameObject. Note that the scripts have to be disabled since they are only enabled if we record or replay a session. Also, keep in mind that only one script inheriting from **CustomRecord** and **CustomReplay** has to be added in the **RecordingManager**. Thus, after adding those scripts, your **RecordingManager** GameObject should look like the picture below:



6.18 VitalManager and Linecreator

In this tutorial we will learn how to set up and use the Vital Manager with the example prefab. Moreover, we will learn how to set up the Line Creator or a new waveform.

Vital Manager is a mechanic contained in the ovidVR.Utilities module, which enables controlling waveforms and values of a custom vital. Line Creator is a mechanic contained in ovidVR.Utilities module which gives the ability to the developer to create a line. The line has smaller lines, distinguished between curved and straight lines

6.18.1 Configure Line Creator

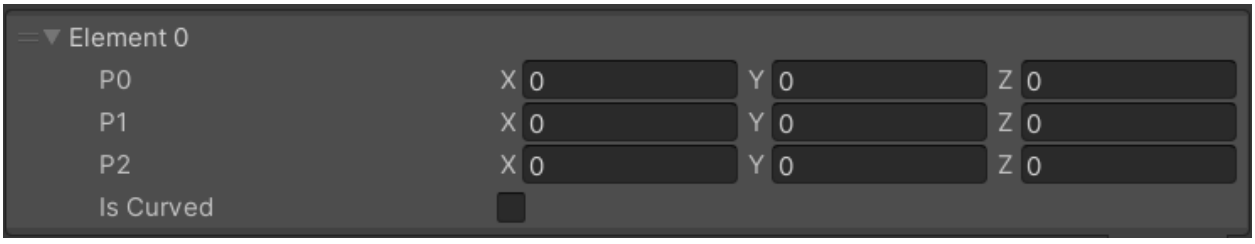
The first step is to Add the LineCreator component to your object. After doing that, you will be able to see that a LineRenderer component is also added. Your next step is to configure a small part of LineRenderer and **add** the material we are giving you with the name **"LineCreatorMaterial"**. To do so, go to LineRenderer and **untick** the **"Use World Space"**

After that, you are ready to create your own custom line! In the Line Creator there are 4 fields;

- A color named **Color**
- A float named **Thickness**
- An integer named **Wave Multiplier**
- A List of points named **Custom Line**

The color is used to change the color of the line, the thickness variable is used to change the thickness of the line. The wave multiplier, duplicates the wave and lastly the list of points lets us create smaller lines in order to create the final one.

When you add new points in the Custom Line field, you will be faced with the results below.

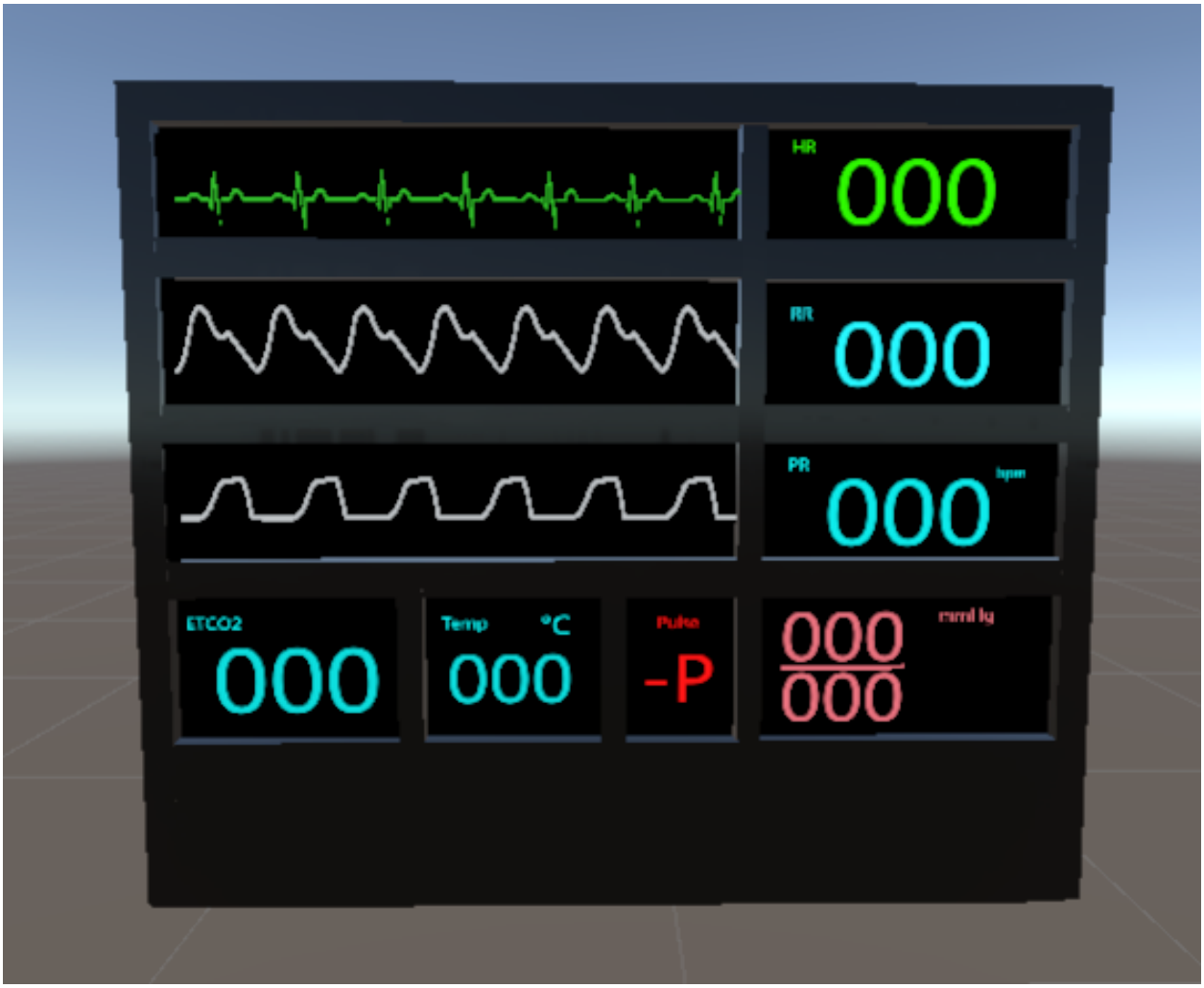


The P0, P1 and P2 are the points of the line. The boolean **"Is Curved"** is ticked only when we want to have **a curved line**. If we want to have a straight line, we only use the P0 and P1 points. If we want to have a curved line we use all the points plus we set true the "Is Curved"

Below is an example of adding both a curved and a straight line.

6.18.2 Importing the Vital Manager Prefab

The first step is to insert the prefab named "Vitals" in the Scene. To do that, just simply drag and drop the prefab into the Hierarchy window. By doing that, you will be able to see in the Scene a Vitals Monitor, like the picture below.



This is an example prefab and you can practice with that to understand how the VitalManager works. Moreover, if you want to practice with the configuration of the vital manager, you can simply load the example scene we have set called *"VitalsManagerScene"*

In order to access and configure the VitalManager, go to the gameobject named "Lines". There, you will see the Vital Manager script with some examples.

Vital Manager consists of three variables:

1. A List of the Vital UI and the Line Waveform
2. A boolean for storing the Vitals Values when they are changed.
3. A List of the Grouped UIs on the monitor.

Note: The type of Line Waveform is LineCreator. LineCreator is user to create your own lines (curved and straight), the color of it, the thickness of it and how many times to duplicate it. If you want to learn more about LineCreator and

how to create your own lines, you should read the tutorial about LineCreator.

Now, let's break down our first variable, the List of the Vital UI and the Line Waveform.

The Vital UI has 5 fields:

- A Text field named **Text UI**
- A Float field named **Vital Value**
- An Integer named **Lerp Speed**
- A boolean named **Vivid Status**
- list of 2 floats and one string names **Custom Text**

The Text UI is the text that we are going to set up the value of the vital, the vital value is the value of the vital, the lerp speed is the time that it needs for a value to be changed, and the vivid status is used to change the value of the vital by +1 and -1 every few seconds. For the list, there are 2 variables type float, one named min and one named max. If the vital value is between the min and max values, then the vital text is replaced with a string.

So, let's create a Vital Text now! In the example video below, I will create a Vital UI for the ETCO2 line. First, I will select my ETCO2 gameobject and put it in the Text UI field. After that, I will set my Vital Status to 40 and my Lerp Speed to 10. Moreover, we will not include any custom texts. Lastly, I will set the Vivid status to true as well.

After showing the VitalUI, it's time to showcase the Line Waveform fields. It has 7 fields:

- A Gameobject named **Normal Line**
- A list of Gameobjects named **Abnormal Lines**
- A boolean named **Can Relocate**
- A boolean named **Call Draw Reverse**
- An integer named **Wave Speed**
- A float named **Z Difference**
- A Gameobject named **Hidder**

The normalLine and the abnormalLines variables are used for the waveforms. In the Normal Line we put the standard waveform and in the abnormal Lines we put all the non-standard waveforms. The *"Can Relocate"* boolean is used to shift line to the left, creating a moving effect. The *"Call Draw Reverse"* redraws the previous line with the new one in order to have the vital effect. With the "Wave Speed" we can manage the time in which the waveform will be drawn. The *"Z Difference"* and the Hidder gameobject are connected. The hidder is the prefab we need to hide the previous line while the Z Difference is the position in Z axis where the old line will be spawned. Lastly, the hidder is an image gameobject, and thus, you can update the height and width of it.

Below is a set-up of all these fields we mentioned.

Warning: The hidder object must be child of the normal line. If you want to create patient's monitor vital, you will need to activate both *"Call Draw Reverse"* and *"Can Relocate"*

Lastly, we have the third field named Monitor Groups and it is a list that contains:

- A string called **Group Name**
- A gameobject called **Group Gameobject**

The *"Group name"* variable is used to store the name of the grouped ui and. In the *"Group Gameobject"* variable we store the grouped ui gameobject.

Note: Monitor Groups is *not* a precondition of the vital manager to work. This field is used only to help the developer be tidy on the vital's screen.

6.18.3 Changing the Vitals Values and Lines from code

With the Vital Manager comes 4 public functions where you can change real-time a line that is being drawn and a vital's value.

- To change a drawing line
- To change a vital's value
- To find line
- To find a text of a vital's value.

```
using ovidVR.ActionPrototypes;
using ovidVR.Utilities.VitalManager;

public class UseAndCleanAction : BasePrototype
{
    public override void Initialize()
    {
        //Searches from the abnormal and normal lines of every vital waveform for the␣
↪input name. Returns a gameobject
        Gameobject lineO2 = VitalManager.Instance.FindLine("O2_PP");

        //Changes the drawing line of the waveform that has the input line of the␣
↪function.
        //For the example below, we the O2_PP waveform is an abnormal line of the O2␣
↪waveforms.
        //Thus, it will change the drawing line of the O2.
        VitalManager.Instance.ChangeDrawLine(lineO2);

        //Searches from the text fields of the vital manager, the text with the same␣
↪name.
        Text textCardiacRythm = VitalManager.Instance.FindTextUI(
↪"CardiacRythmVitalText");

        //Changes the value of the text, with a fixed speed that can be changed from␣
↪vital manager, to the input value.
        VitalManager.Instance.ChangeVitalValue(120,textCardiacRythm);
        base.Initialize();
    }
}
```

6.19 Cut - Tear - Drill

6.19.1 Realtime Cut

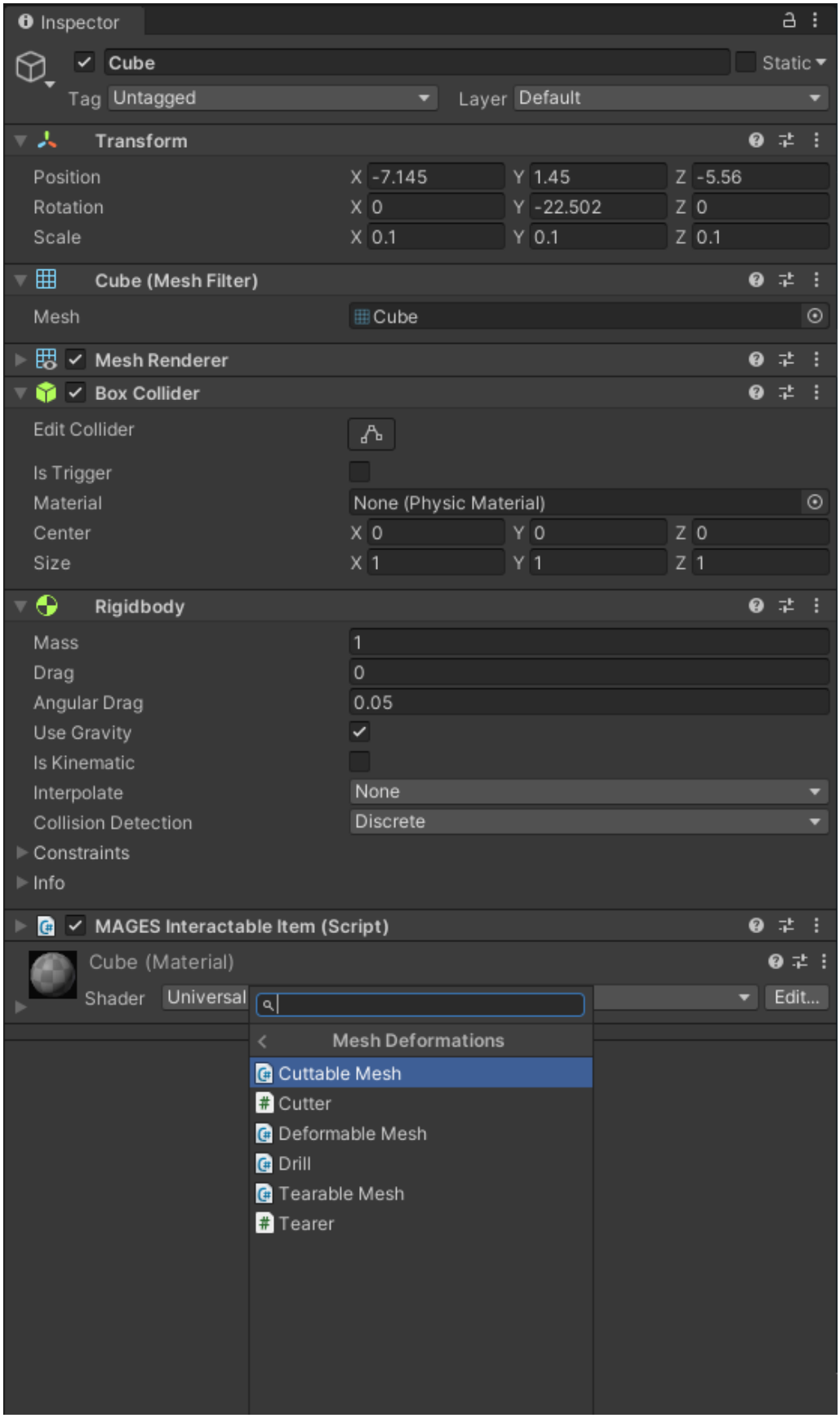
> **Warning:** Note that this feature is in Beta stage, therefore it is not stable and it can create artifacts on the newly created meshes.

MAGES gives you the ability to perform realitme cuts in meshes, by providing a **Cuttable Mesh** and a **Cutter** script. A combination of these can achieve results like the following:

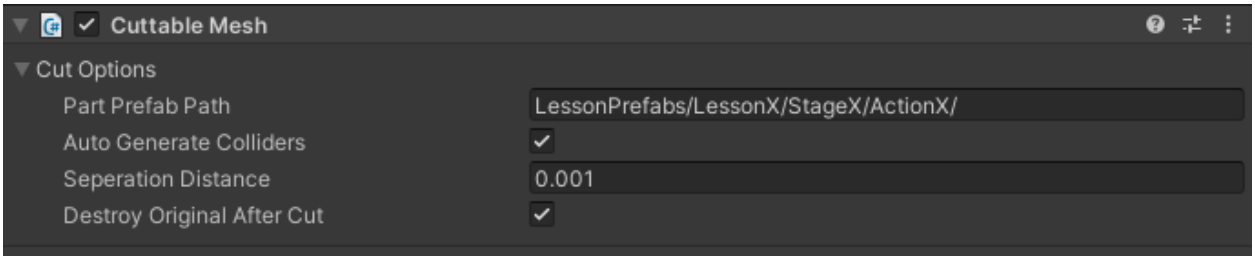
In this tutorial we will demonstrate how to setup the cube as a cuttable mesh and how to setup a custom cutter.

Cuttable Mesh

First drag and drop the mesh you want to setup as cuttable in the scene (The box in this case). Then go to **Add Component > MAGES > Mesh Deformations > Cuttable Mesh** and add the **Cuttable Mesh** script to the object.


Inspector
Cube
Static
Tag Untagged
Layer Default
Transform
Position X -7.145 Y 1.45 Z -5.56
Rotation X 0 Y -22.502 Z 0
Scale X 0.1 Y 0.1 Z 0.1
Cube (Mesh Filter)
Mesh Cube
Mesh Renderer
Box Collider
Edit Collider
Is Trigger
Material None (Physic Material)
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1
Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity
Is Kinematic
Interpolate None
Collision Detection Discrete
Constraints
Info
MAGES Interactable Item (Script)
Cube (Material)
Shader Universal
Edit...
Mesh Deformations
Cuttable Mesh
Cutter
Deformable Mesh
Drill
Tearable Mesh
Tearer

This will expose the following properties:



Afterwards, we will create an empty GameObject and we will add:

- a MeshRenderer
- an empty MeshFilter
- the same material as our cube
- a Rigidbody
- the MAGESInteractableItem script

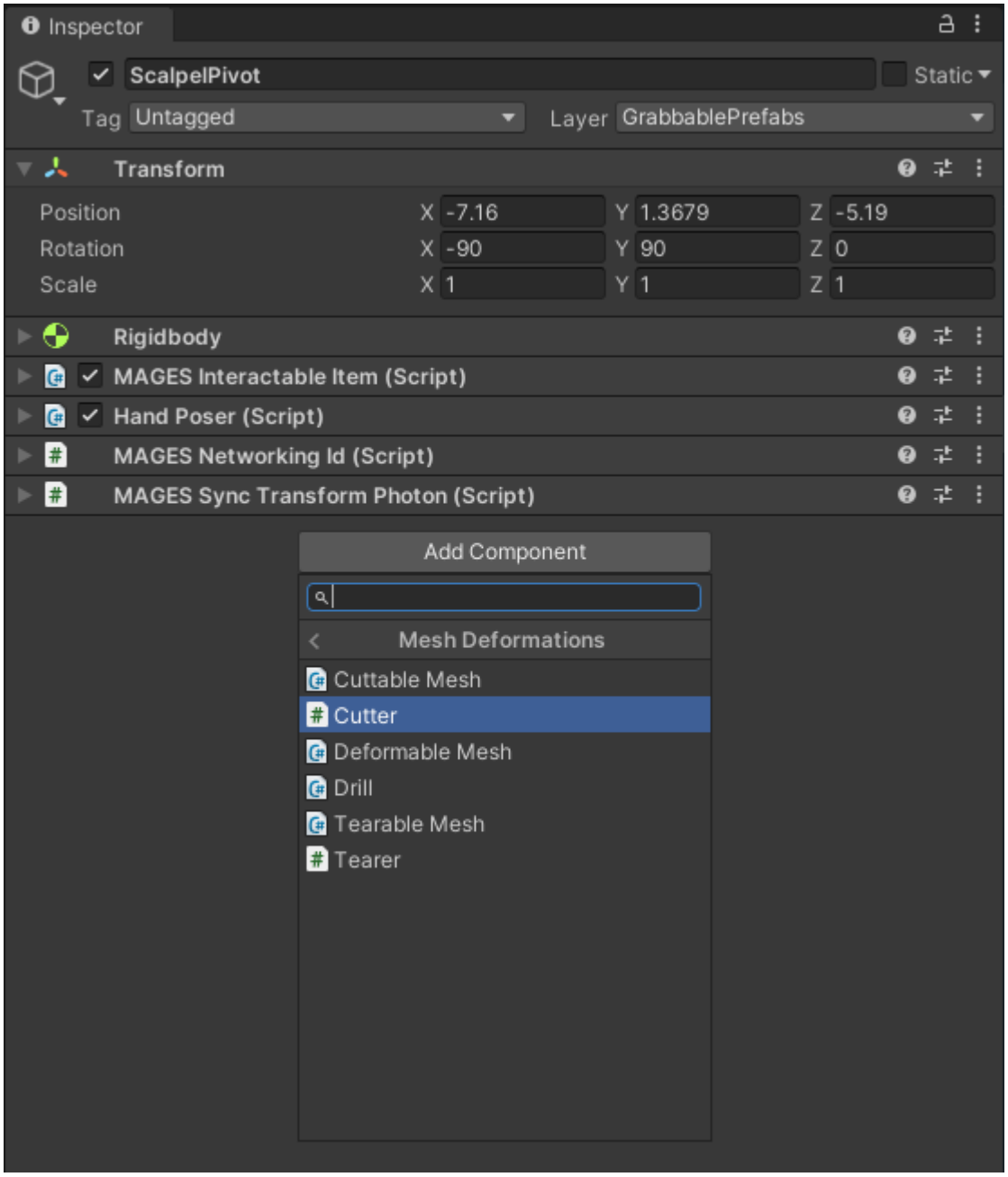
We will then save our prefab under the Resources folder in a desired path. In the **CuttableMesh** inspector we will specify that path in the field partPrefabPath. For the other options we will keep their default values.

We can now use our **Cuttable Mesh** by calling the `public void Cut(Plane cutPlane, out Mesh positiveSide, out Mesh negativeSide, out GameObject final, bool handleMeshReconstruction = true)` function, or by using the already made **Cutter Script** as described below.

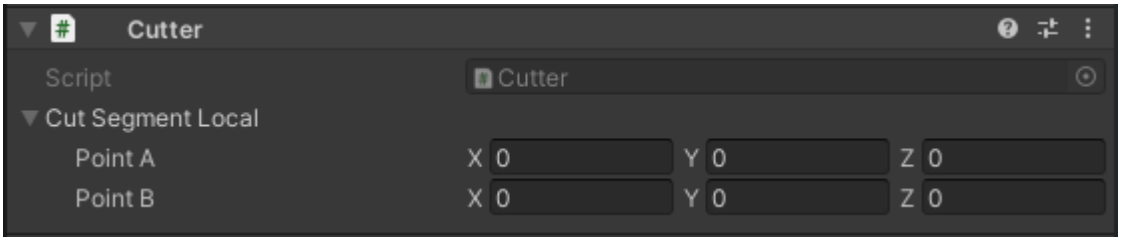
Note: You can find the above example, already set up, in the Mesh Deformations Sample Scene.

Cutter

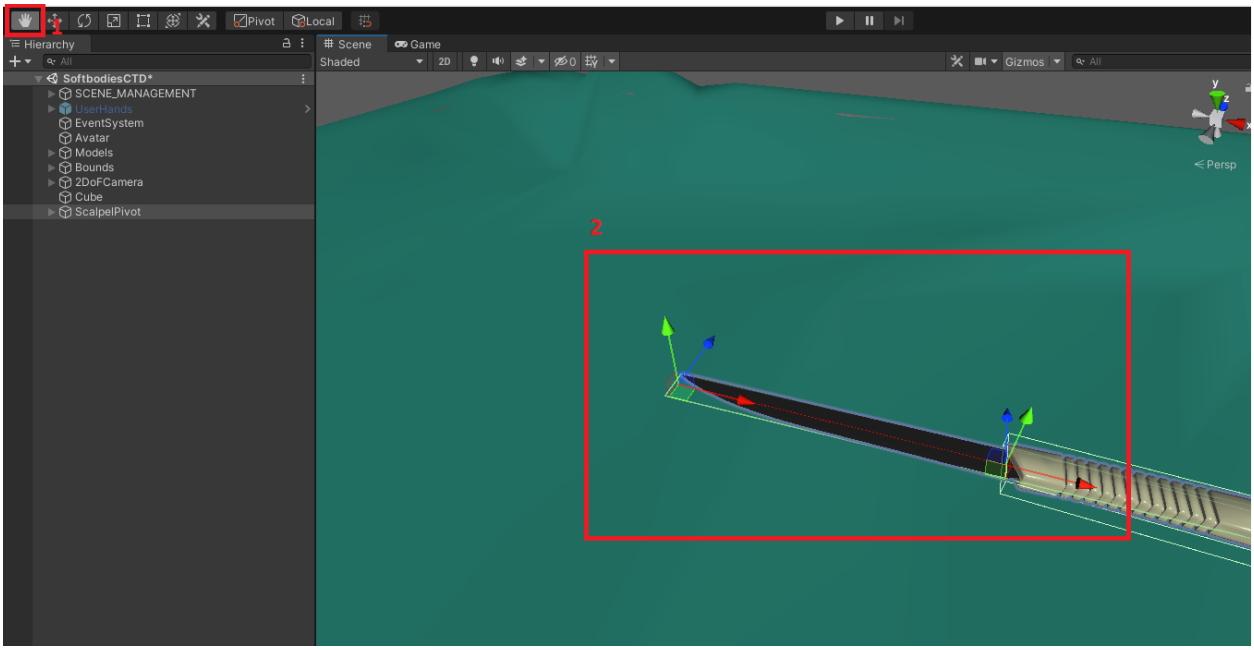
First drag and drop the tool you will use to cut the mesh (here we will use a knife). Then go to **Add Component > MAGES > Mesh Deformations > Cutter** and add the **Cutter** script to the tool.



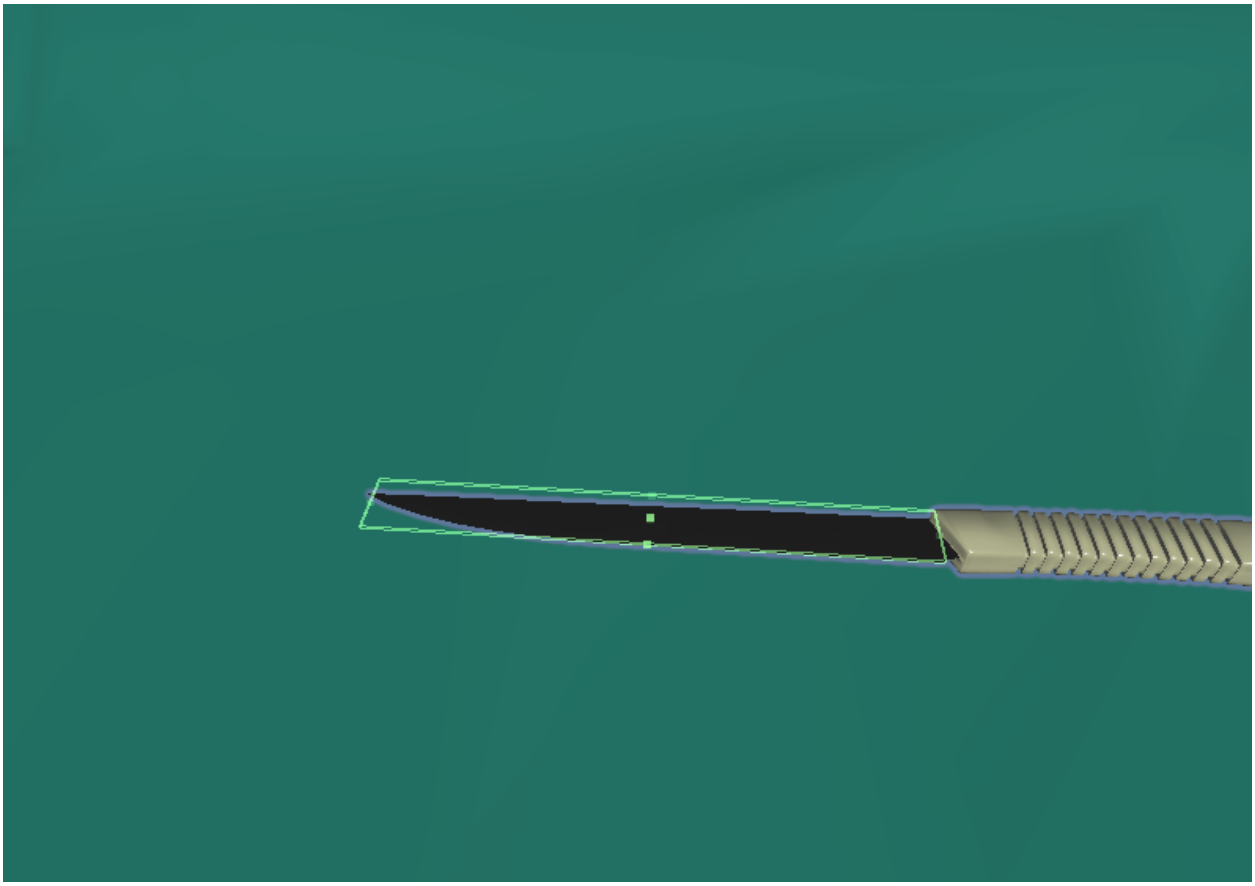
This will expose the following properties:



We will then (1) click the hand tool to hide the unity tool gizmos and then (2) move the two handles to match the knife's blade.



The next step is to add a trigger collider containing the blade of the knife.



Finally, we need to add a collider (either triger or not) on the GameObject containing the **CuttableMesh** script in order to specify the area a cut can be done.

We can now use the knife to cut our **Cuttable Mesh** inside the specified area.

Note: You can find the above example, already set up, in the Mesh Deformations Sample Scene.

6.19.2 Realtime Tear

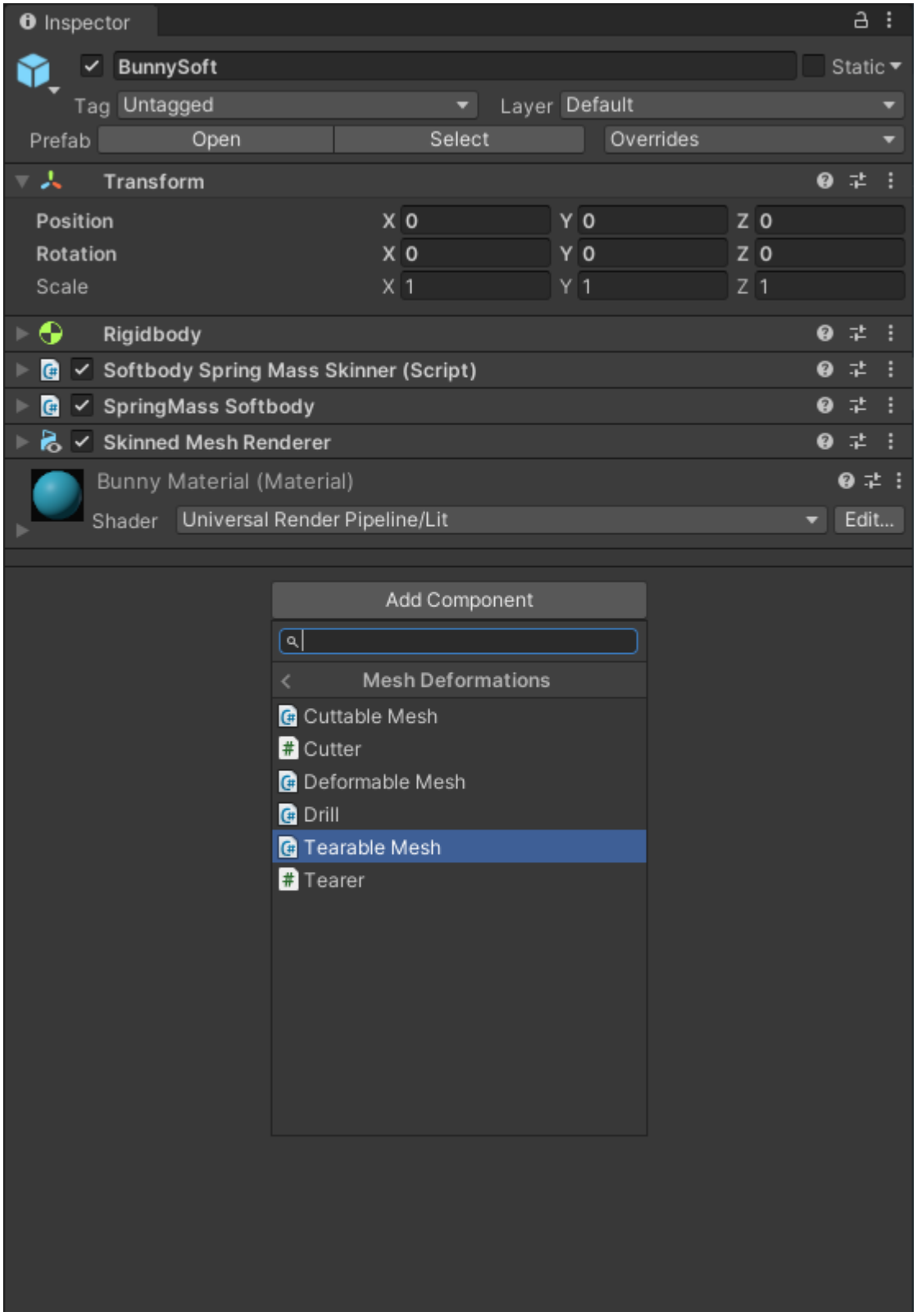
Warning: Note that this feature is in experimental stage, therefore it is not stable and it can create artifacts on the newly created mesh.

MAGES gives you the ability to perform realitme tears in meshes, by providing a **Tearable Mesh** and a **Tearer** script. A combination of these can achieve results like the following:

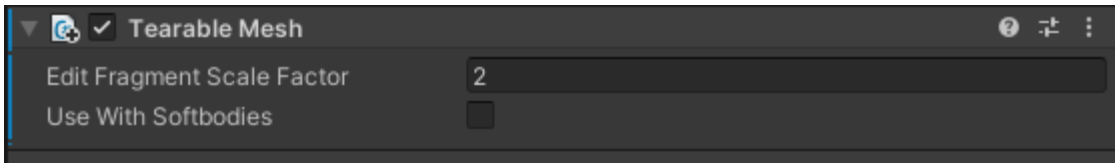
In this tutorial we will demonstrate how to setup a softbody bunny as a tearable mesh as well as how to setup a scalpel to perform the tear.

Tearable Mesh

First drag and drop the **BunnySoft** prefab located in **Packages > ORamaVR MAGES SDK > Runtime > SDKAddons > Softbodies > Examples > Bunny** into the scene. Then go to **Add Component > MAGES > Mesh Deformations > Tearable Mesh** and add the **Tearable Mesh** script to the object.

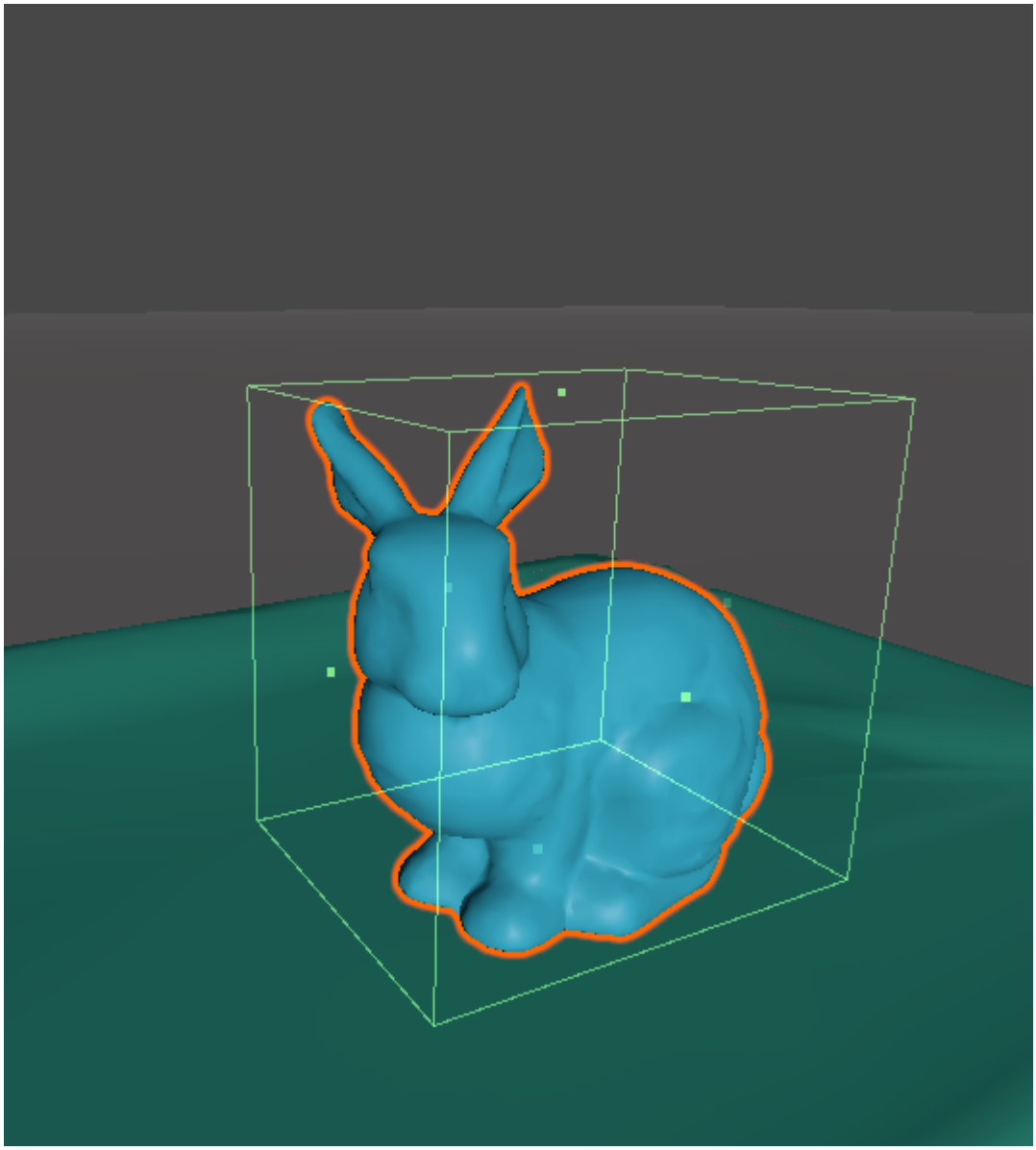


This will expose the following properties and you will set them as below:



If you do not use the bunny prefab you can set the `float editFragmentScaleFactor` to 0 and figure it out later when you will have the tearer setup. The `bool useWithSoftbodies` option will be available only if the tearer is attached in a softbody, and when selected it will enable tearing of the underlying softbody particle connections in addition to the mesh.

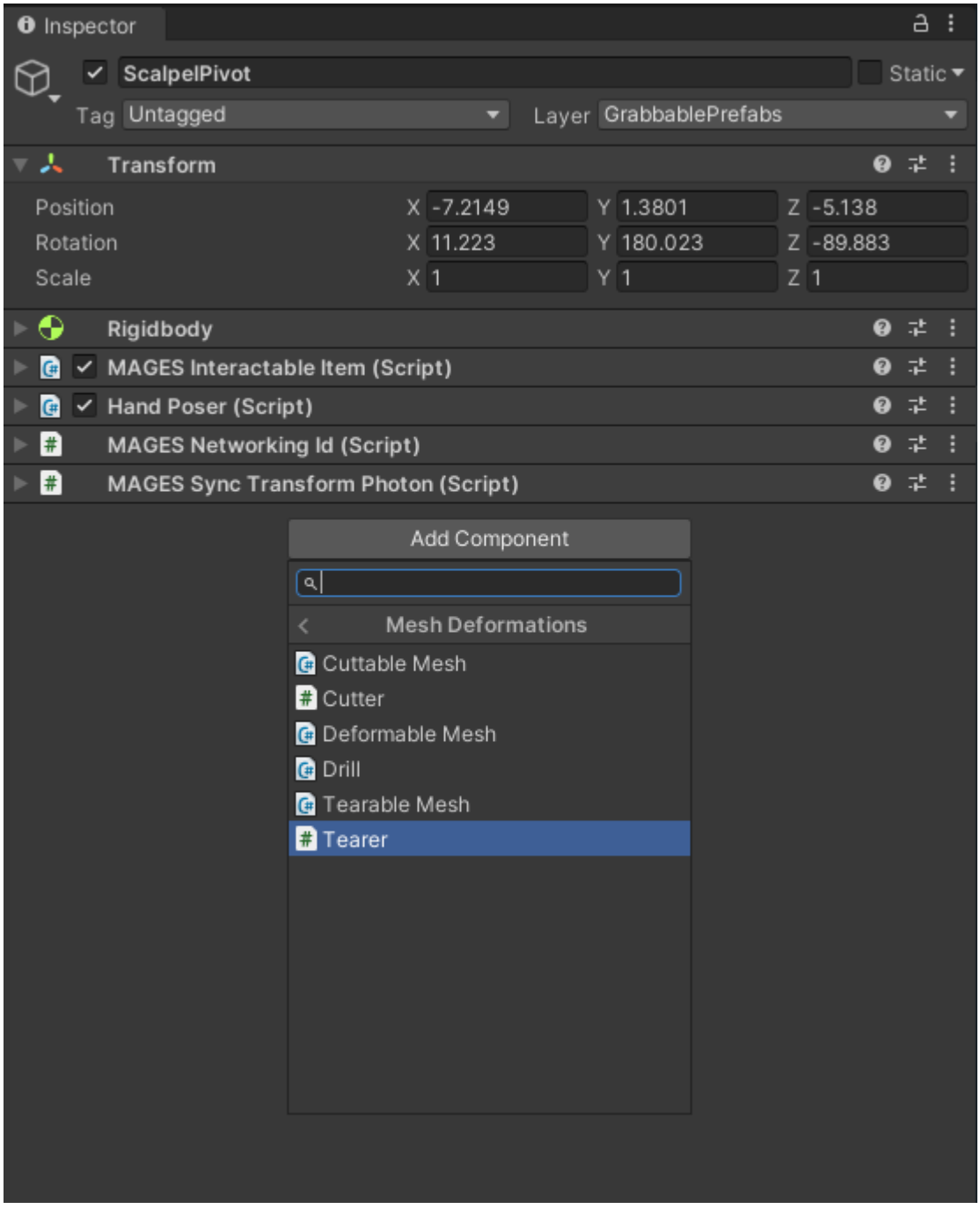
The final step to complete setup is to add a trigger collider containing our mesh.



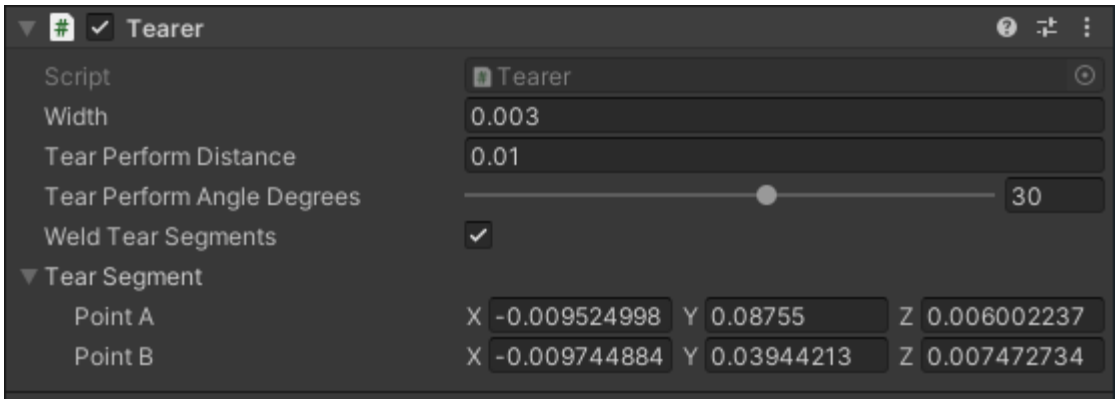
Note: This step is optional if you want to use the **Tearable Mesh** by calling the Tear functions from a custom script, instead of using the **Tearer** script described below.

Tearer

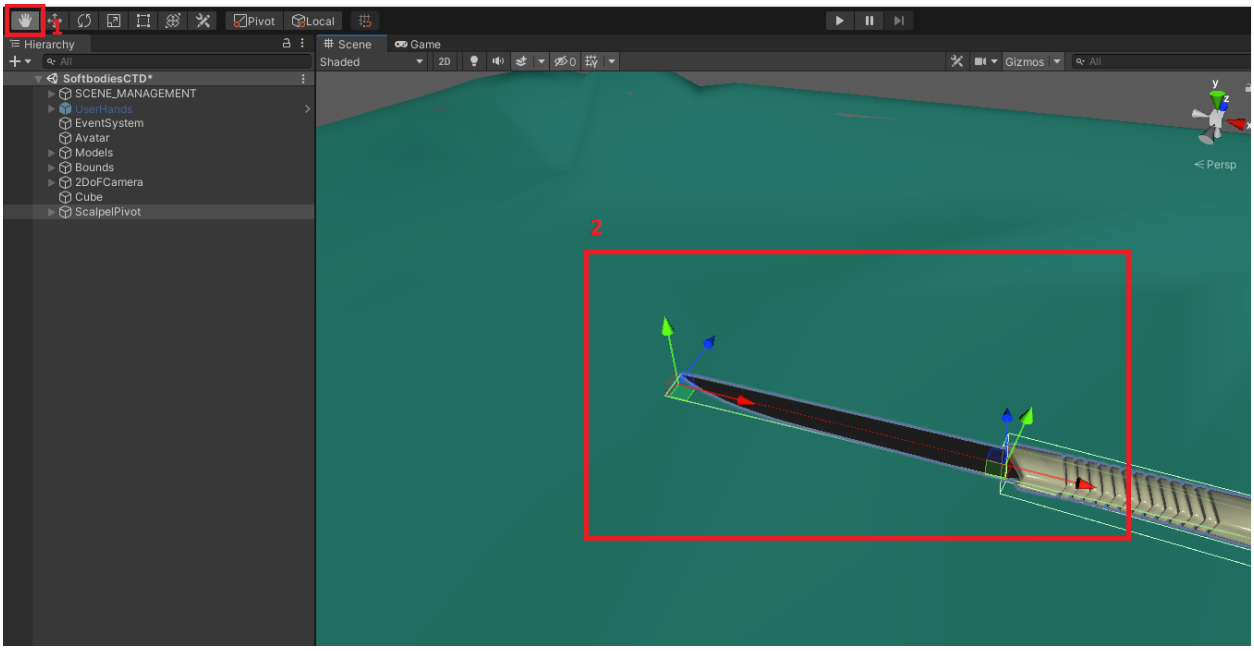
First drag and drop the tool you will use to tear the mesh (here we will use a scalpel). Then go to **Add Component > MAGES > Mesh Deformations > Tearer** and add the **Tearer** script to the tool.



This will expose the following properties:

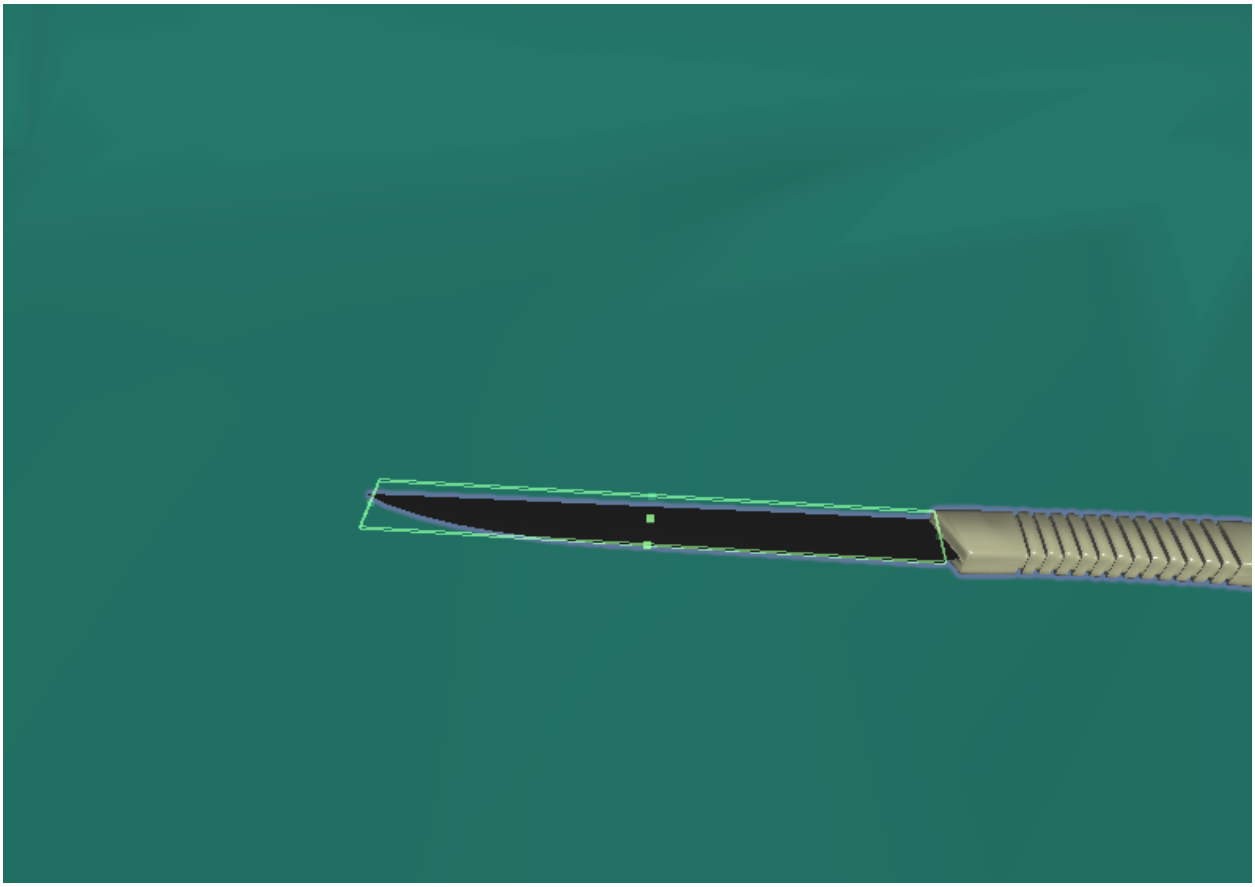


You will then (1) click the hand tool to hide the unity tool gizmos and then (2) move the two handles to match the scalpel's blade.



You can modify the width and the length of each tear segment in the tearer script according to your needs.

The final step to complete the setup is to add a trigger collider containing the blade of the knife.



You can now use the scalpel to tear any **Tearable Mesh**.

6.19.3 Realtime Drill

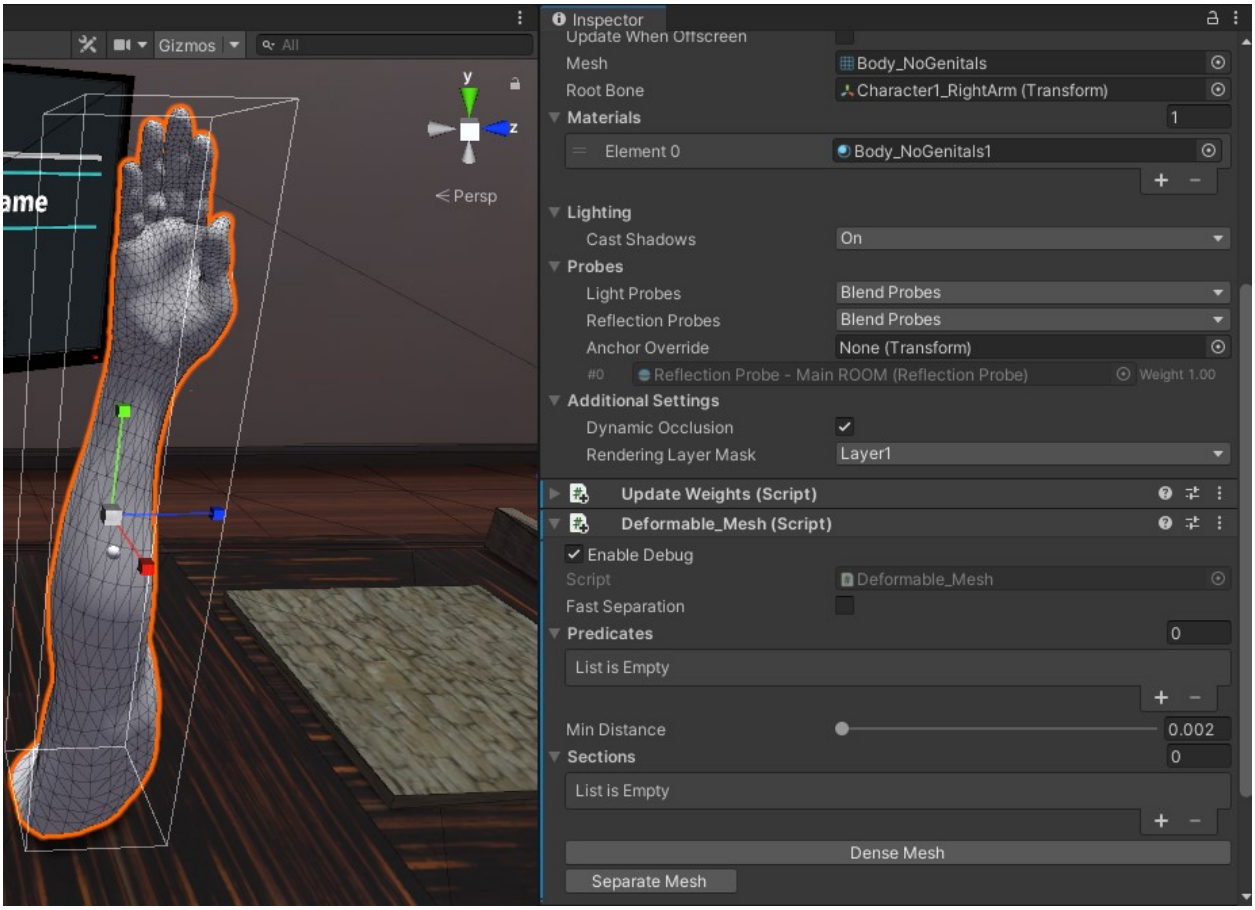
> **Warning:** Note that this feature is in experimental stage, therefore it is not stable and it can create artifacts on the newly created mesh.

MAGES gives you the ability to perform realitme drilling in meshes. In order to achieve this you will need to attach the Deformable Mesh script to the object containing the mesh and the Drill script to the tool.

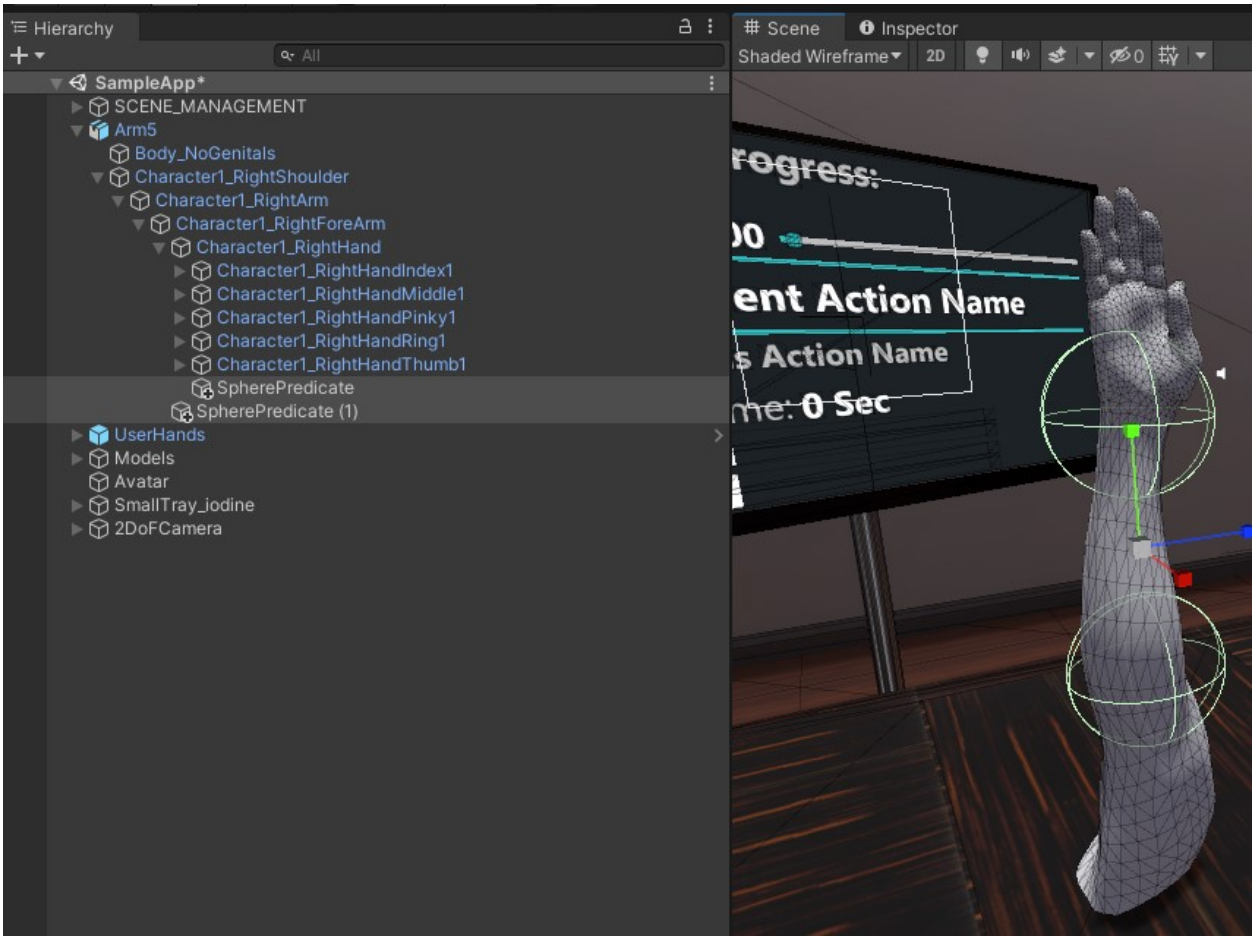
Deformable Mesh

Setting up a deformable mesh

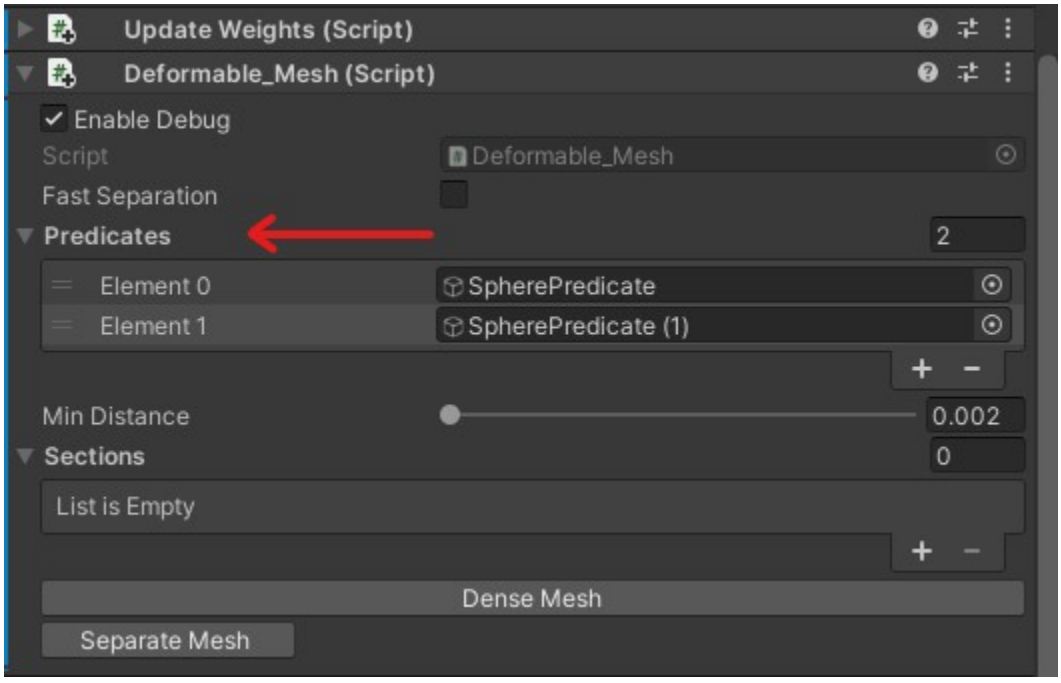
To convert a mesh into a deformable mesh, attach the "Deformable Mesh" component on the Game Object that contains its renderer.



The deformable mesh can be separated into multiple sections, based on what parts of it may be changed, as well as for optimization purposes. To separate a mesh, a list of predicates is used, which are gameobject hierarchies that contain colliders:

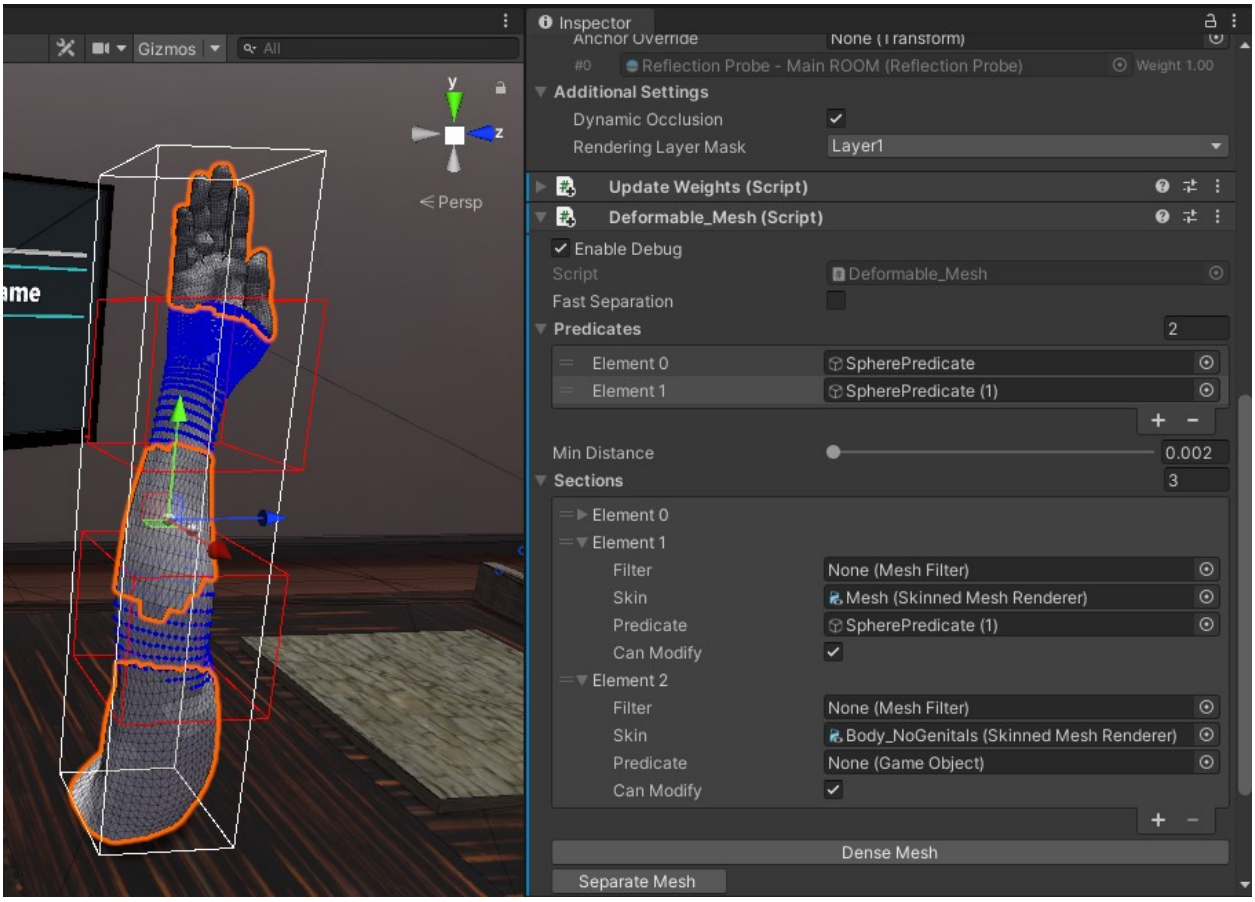


You can add predicates by adding to the "Predicates" property in the Deformable Mesh component:



After adding the colliders, you can convert the mesh to a deformable mesh, by clicking on the "Separate Mesh" button.

This may take a few moments, based on the complexity of the mesh in question:

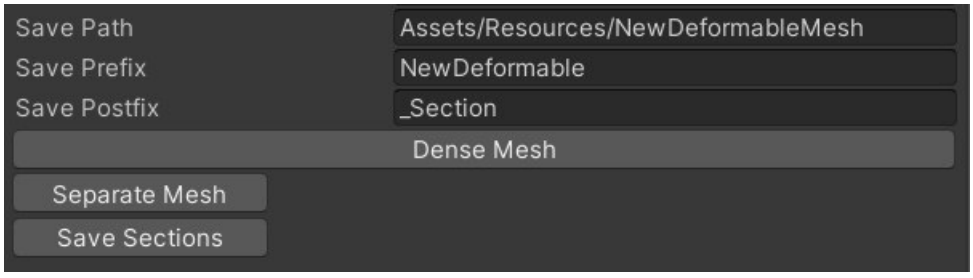


In this case, since we used, 2 predicates, there are a total of 3 different mesh sections:

- The original mesh
- And the two separated sections from the vertices that were inside the sphere colliders

In the sections list, you can check the property “Can Modify” to disable modification of the mesh section in question by all CTD scripts.

Finally, in order to be able to save the deformable mesh as a prefab select a valid path and name in the fields shown below and click **Save Sections**.



Then you can simply parent all the deformable mesh’s components in an empty gameobject and drag n drop it in your assets to create a reusable prefab.

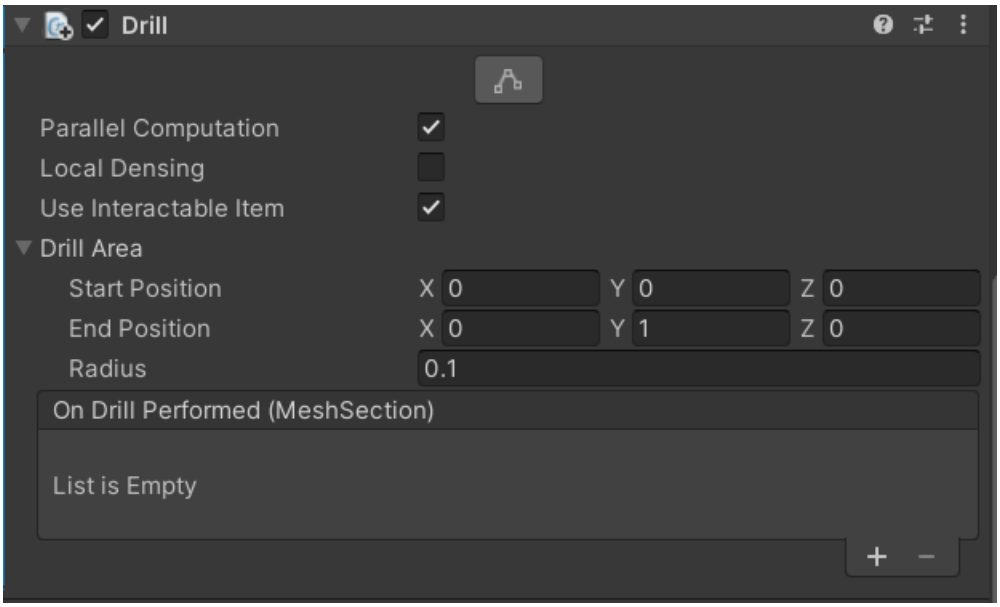
Drill

The drill module enables the developer to set up a Unity GameObject as a drill tool, in order to achieve real time drilling of holes on 3D rigged and static models.

Set up

To use the Drill Module, you need to attach the `Drill` component onto the gameobject that we want to use as a drill. Simply click **Add Component> MAGES > Mesh Deformations > Drill** to add the component.

Note: The scale of the gameObject that this script is attached to, affects the scaling of the drill area.



This gives you the following properties:

1. Parallel Computation:

- Option to run the code in parallel for increased performance.

2. Local Densing:

- Option to split every affected triangle of the drilling into four smaller ones, in order to prevent the smaller diameter holes looking like polygons.

3. Use Interactable Item

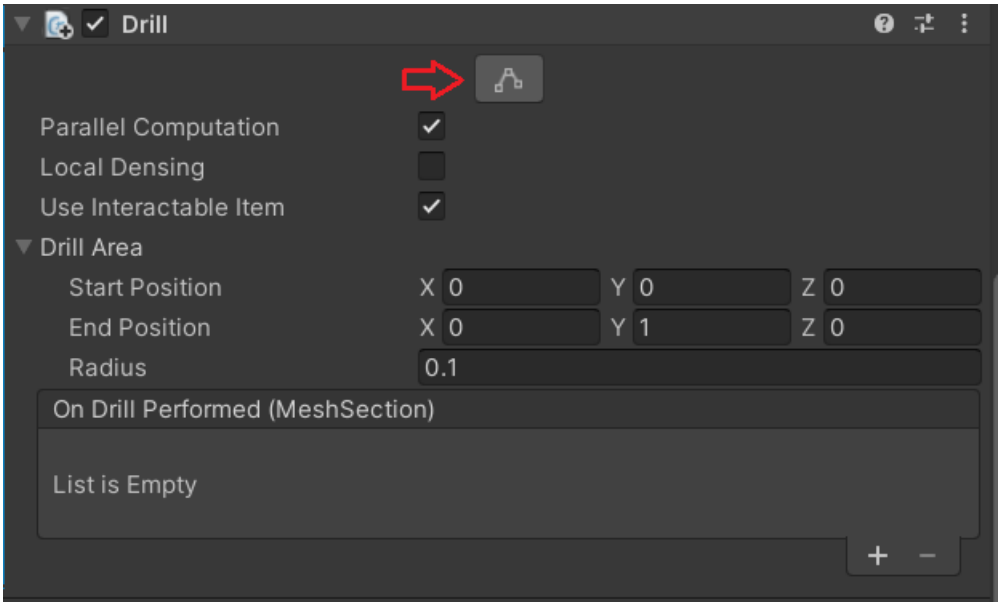
- Option to use this script in combination with an interactable item, such that when you have this object in your hand and then press the trigger button a drill is performed.

4. Drill Area:

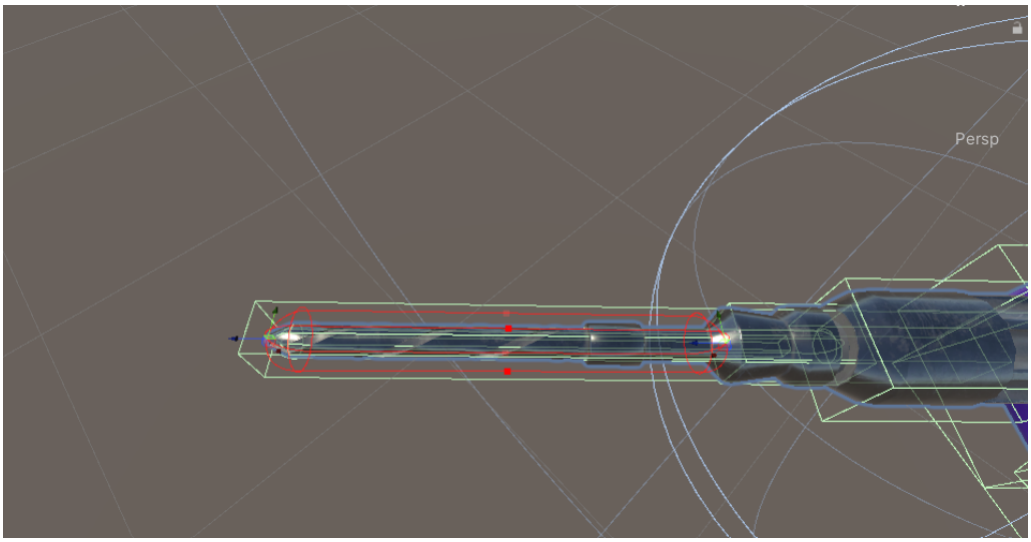
    - Option to set up the drilling axis and the radius of the drill.

Note: Make sure Gizmos are enabled on the Unity editor.

  – Click the edit button shown in the image below:



– Use the handles shown in the scene to adjust the drill area to your requirements as shown below:



That last step that remains is to add a trigger collider component to the gameObject, that is aligned with the drill axis GUI that we created before:

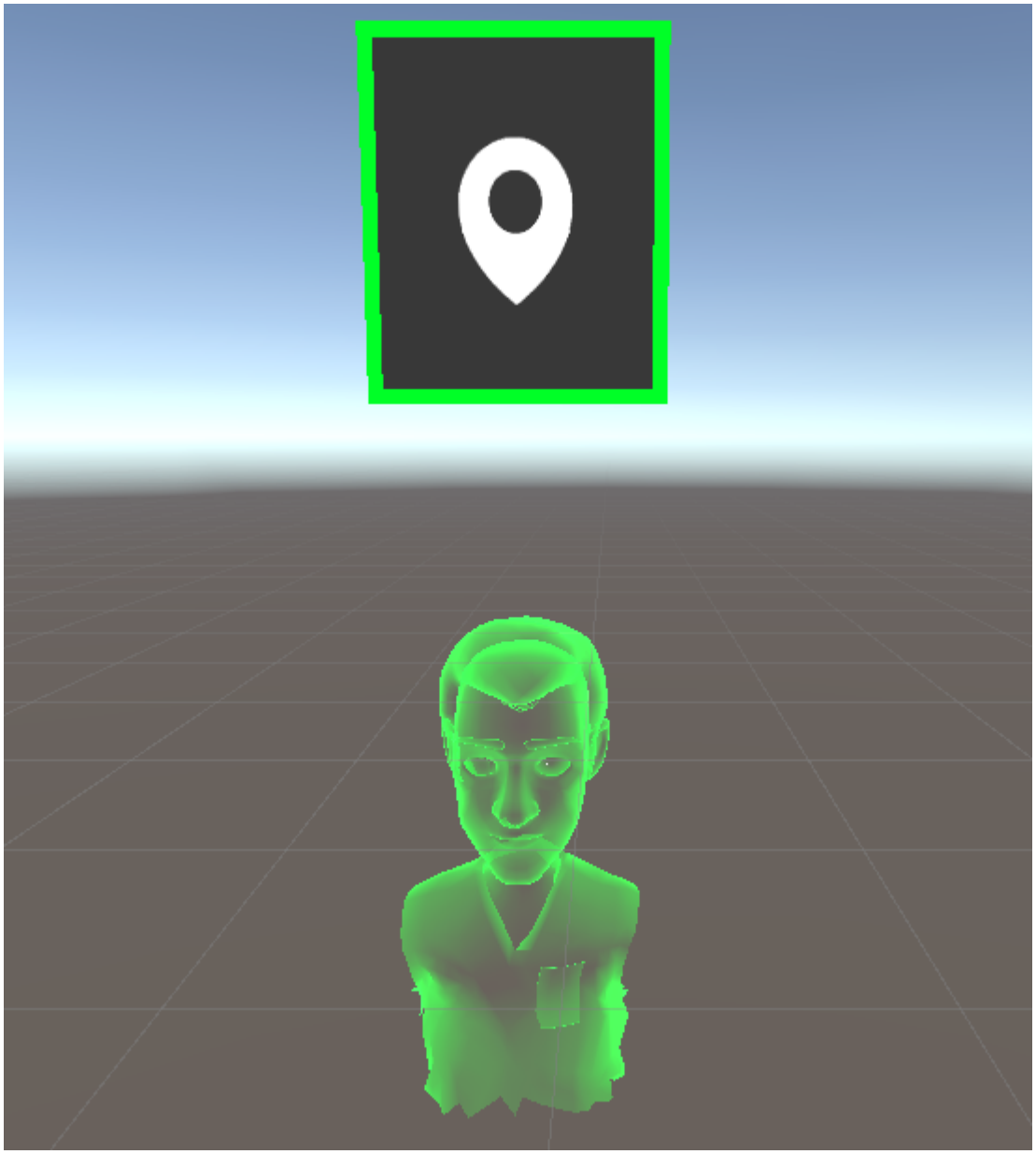
In our example we are going to add a `Box Collider`.

Properties

Property	Description
`bool parallelize`	Minimize running times by executing the code in parallel.
`bool localDensing`	Split every affected triangle of the drilling into four smaller ones, in order to prevent the smaller diameter holes looking like polygons.
`DrillAxis drillArea`	Set up the drilling axis and the radius of the drill
`UnityEvent OnDrillPerformed`	Called whenever after this script successfully modifies a mesh section.

6.20 Teleport

The `TeleportLocation` prefab makes teleporting to a location easy, regardless of the camera in use. Simply place the prefab to the location that you want to teleport to. When you press the button, you will get teleported to that location with a smooth camera fade in / out animation.

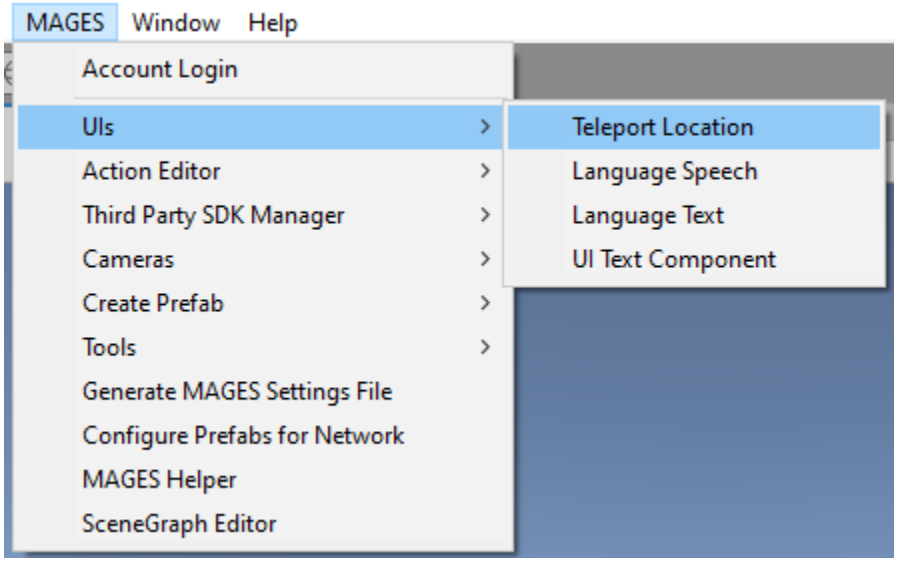


Above you can see the `TeleportLocation` prefab, which consists of a button and an avatar. The avatar indicates the position and orientation of the player after the teleport. When you hover over the button, the avatar gets enabled and when you stop hovering, it gets disabled again.

Note: When the player gets within a predefined `Hiding_Distance` (see Inspector Fields section) from the `TeleportLocation` , the button gets disabled so that it does not block / interfere with the vision of the player.

6.20.1 How-To

You can find the `TeleportLocation` prefab on the MAGES menu under the path: `MAGES/UIs/Teleport Location`.



When you do that, the `TeleportLocation` prefab will get spawn on the scene and you can place it wherever you want to teleport to. You can start the teleport by clicking the button with your raycasts.

Note: Remember you can toggle the raycast using the following command:

```
InterfaceManagement.Get.InterfaceRaycastActivation(bool);
```

The raycast can be used to interact with the teleport button.

6.20.2 Inspector Fields

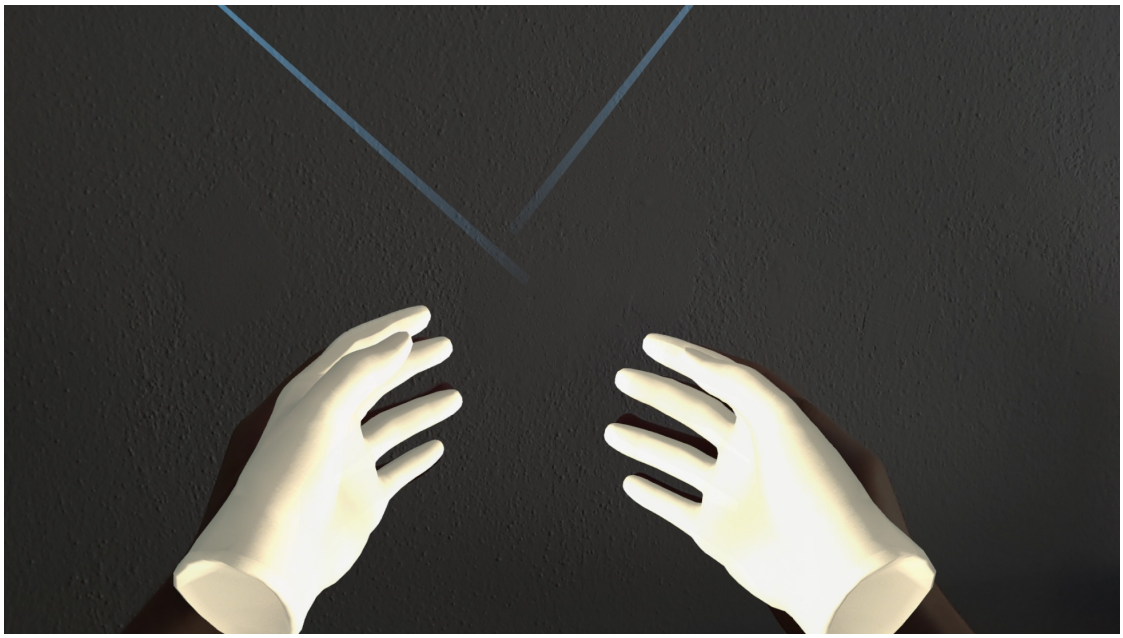
Below there is the table of the inspector extra slider parameters and what they do.

`class TeleportLocationControl`

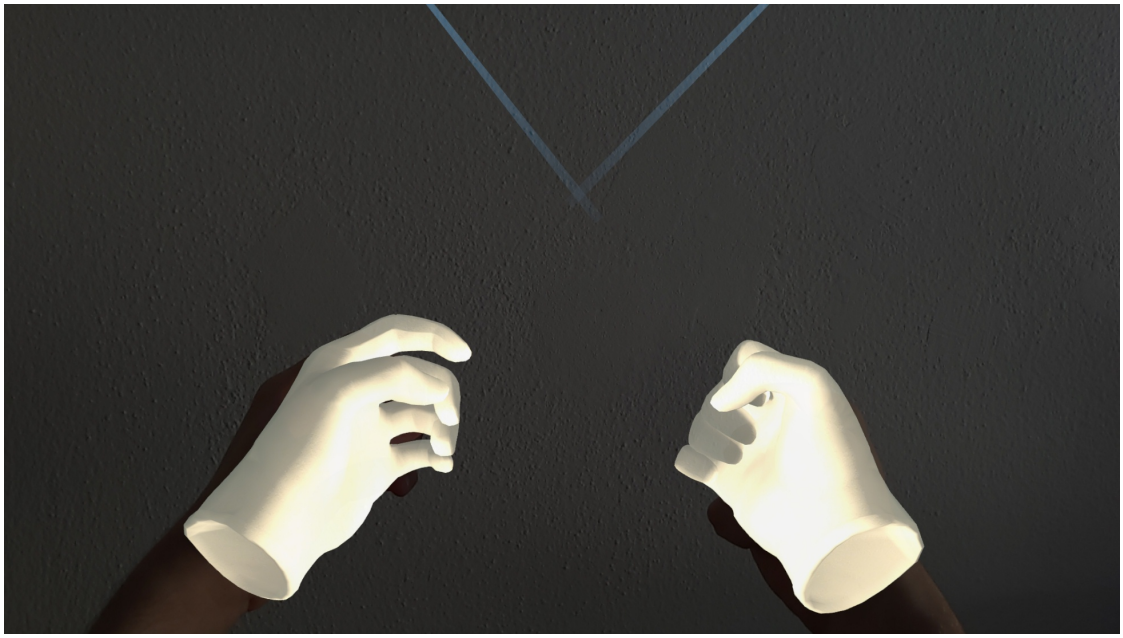
Parameter	Description
`Color Available_Color`	The color of the border of the button when the location is available.
`Color Unavailable_Color`	The color of the border of the button when the location is unavailable.
`float Hiding_Distance`	This indicates the distance between the teleport position and the player to hide the teleport button.
`float Move_Threshold`	The distance that the player can move on the teleport position without changing its availability.
`float Camera_Fade_Duration`	The duration of the camera fade animation.
`bool Disable_Availability_Check`	Disables the availability checks for this teleport position. This position will always be available.

6.21 MAGES Hands with HoloLens 2

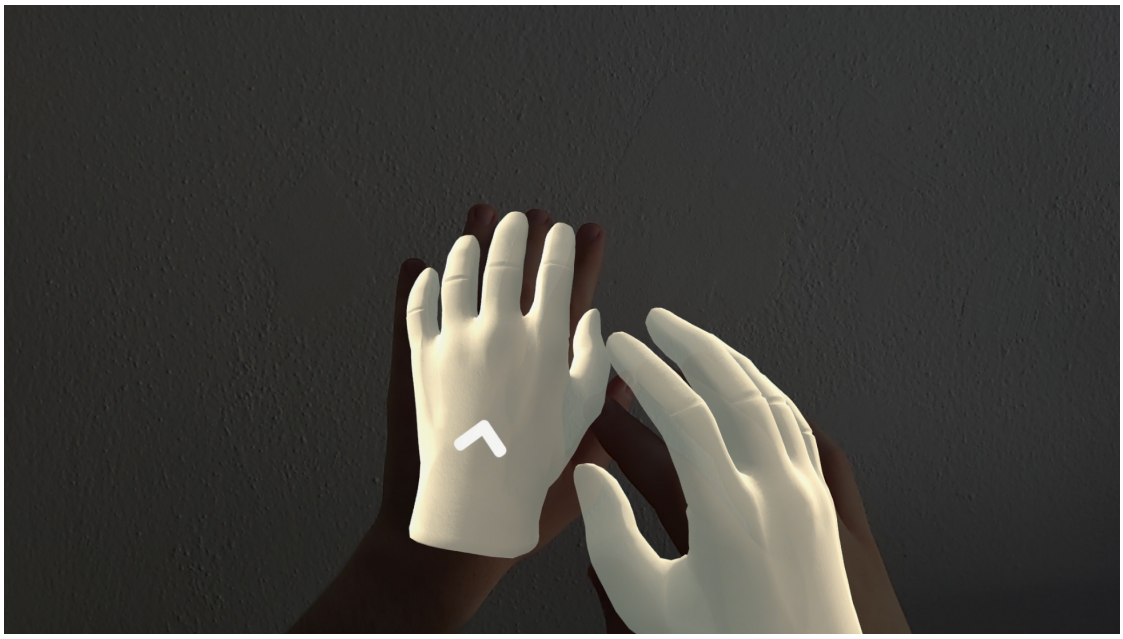
MAGES supports using Hand Tracking with the HoloLens 2 headset, providing a natural way of interacting with items in AR.



The user is able to see their digital hands and interact with the world digital world using them. They may also close or open their real hand, and the digital hand will follow.



For movement, the user can move towards the direction they are looking by touching the back of their left palm. When the right hand gets close the the left hand, an arrow will appear, hinting at the movement control ready to be used.



It is suggested that the right hand is in the form of a fist when doing this to avoid tracking issues.

Interaction with UI elements is achieved by either touching them or making a closed fist when pointing at them with a ray cast.

CHAPTER

SEVEN

VIDEO TUTORIALS

In addition to our written resources, we also provide video tutorials on how to create actions from start to end.

7.1 Getting started with MAGES

7.2 How to configure Action Analytics

7.3 How to setup 3D Desktop Camera

7.4 Insert Action

7.5 How to program an Insert Action on another object (to assemble objects)

7.6 Use Action

7.7 Remove Action

7.8 Question Action

7.9 Combined Action

Note: The following video is work in progress and do not fully reflect the latest stable release of MAGES SDK, however it can still be used at almost all cases. New videos will come out shortly. Stay tuned!

7.10 Tool Action

CHAPTER

EIGHT

CLASS REFERENCE

8.1 MAGES™ SDK

8.1.1 Action Prototypes

namespace *MAGES::ActionPrototypes*

namespace *MAGES::AutomaticDestroy*

namespace *MAGES::AutomaticDestroyFunctions*

class *MAGES::ActionPrototypes::AnimationAction::AnimationGroup* Helper class to group the animation Action prefabs

class [MAGES::ActionPrototypes::BasePrototype::InstrumentTransforms::CustomTransfrom] (#classovid_v_r_1_1_action_prototypes_1_1_base_prototype_1_1_instrument_transforms_1_1_custom_transfrom) Instrument transformation values

class *MAGES::ActionPrototypes::BasePrototype::HoloGroup* Helper class for holograms

class *MAGES::ActionPrototypes::RemoveAction::InsertGroup* For Remove Action we need an extra class to group the prefabs that will be spawned

class *MAGES::ActionPrototypes::InsertAction::InsertGroup* Helper class to group the insert Action prefabs

class *MAGES::ActionPrototypes::BasePrototype::InstrumentTransforms* This class is used to store information about the instrument's position There are some cases we want to know the position of an instrument to spawn another one relative to this position. This methodology is used in actions that user assembles instrumets. Since user handles the instruments with shis hand we need to sknow where those intstruments (or his hands) are located.

class *MAGES::AutomaticDestroyFunctions::AutomaticObjectDestroyFuncs::PlayAnimationAfterEnal* Interal class need to Play again animation if the action is perform before the animation finish Automatic removed when it is finished

class *MAGES::ActionPrototypes::PumpAction::PumpGroup* For Pump Action we need an extra class to group the prefabs that will be spawned

namespace MAGES::ActionPrototypes

Summary

class *MAGES::ActionPrototypes::AnimationAction*

class *MAGES::ActionPrototypes::BasePrototype* Base Prototype contains all the shared variables and methods other Action prototypes have It encaptulates the basic functionality to run ang configure each Action All Action prototypes inherit from Base prototype and override the needed functions

class *MAGES::ActionPrototypes::CombinedAction* The combined is used to perform actions that consist of multiple sub-actions sequentially. An example of combined action is when you want to open a door. This can be described as OpenDoorAction consisting of PullHandleAction and PullDoorAction. In general, the combined action is used to describe an action that in order to be performed, the user needs to perform 2 or more 'smaller' actions.

class *MAGES::ActionPrototypes::InsertAction* Insert Action is referring to a specific type of Action that user has to insert an object to a specific position in order to complete it. For such an action we need three basic elements. First the Interactable prefab which is the object user will grab using his virtual hands or any other input device. The second one is the position we have to place the object. This position is called the Final position of our prefab. The correct orientation of the interactable object relative to the Final position is the key element to complete the Action. The final element to complete the prototyping of an Insert Action is to use a hologram to indicate the correct final position of the hologram. An Insert Action can have many objects to be inserted to the final position. For this reason we can set many prefabs in one Action.

class *MAGES::ActionPrototypes::PumpAction* The Pump Action is used in situations where the user needs to press a button or a tool (e.g. syringe) It is also used in Actions where user needs to to use a tool (e.g. syringe) to insert liquid or air by making the pump gesture (technically pressing the trigger button). In those Actions we need an object for user to interact with and press the trigger button from the controller (as many times he set beforehand) to Perform the Action.

class *MAGES::ActionPrototypes::QuestionAction*

class *MAGES::ActionPrototypes::RemoveAction* Remove Action describes a step of the procedure which user has to remove an object using his hands or a tool.To implement a Remove Action we have to set the object that will be removed.This object will spawn as the removable prefab and to complete the Action user needs to reach and remove it.The removable prefab will flash to indicate that it needs to be removed to complete the Action.Remove Action also supports removing an object using a tool instead of hands. This tool can be a pair of plyers or another tool that supports the grab mechanism.

class *MAGES::ActionPrototypes::ToolAction* Tool Action is referring to an Action that user has to take a tool and use it to complete the action.Technically speaking, a Tool Action needs only a set of colliders to trigger a specific tool when they will come in contact.This set of colliders are referring as the Collider prefab and contain the necessary information for the selected tool to trigger them.

class *MAGES::ActionPrototypes::UseAction* Use Action is similar to a Tool Action but instead of a tool we use another object to complete the Action.In order to properly set the Action we have to spawn the collider prefab(as the tool Action) but in this case we also need to provide an extra object to interact with (instead of the tool).

class MAGES::ActionPrototypes::AnimationAction

```
class MAGES::ActionPrototypes::AnimationAction
  : public MAGES.ActionPrototypes.BasePrototype
```

Summary

`public inline virtual override void Initialize()` Initialize method for Animation Action overrides base.Initialize() and 1)Spawns all animation prefabs from the prefab list 2)Sets the counter that will Perform the Action to the number of the animated prefabs

`public inline virtual override void Perform()` Perform method for Animation Action overrides base.Perform() and 1)Finalizes all Animation prefabs 2)Clears Action Lists

`public inline virtual override void Undo()` Undo method for Animation Action overrides base.Undo() and 1)Destroys Animation Prefabs 2)Clears Action Lists

`public inline AnimationGroup SetAnimationPrefab(string grabbablePrefabPath, GameObject parentGameObject,InheritTransformFrom _inheritTransformGrabbale)` Set prefabs for a single instrument

`protected List< AnimationGroup>animationGroupsList` This list contains all the animations objects of current Action User needs to play all of the animations to perform the Action

Members

public inline virtual override void Initialize()

Initialize method for Animation Action overrides base.Initialize() and 1)Spawns all animation prefabs from the prefab list 2)Sets the counter that will Perform the Action to the number of the animated prefabs

public inline virtual override void Perform()

Perform method for Animation Action overrides base.Perform() and 1)Finalizes all Animation prefabs 2)Clears Action Lists

public inline virtual override void Undo()

Undo method for Animation Action overrides base.Undo() and 1)Destroys Animation Prefabs 2)Clears Action Lists

public inline AnimationGroup SetAnimationPrefab(string grabbablePrefabPath, GameObject parentGameObject,InheritTransformFrom _inheritTransformGrabbale)

Set prefabs for a single instrument

Parameters

- `grabbablePrefabPath` The prefab user will grab with animation

protected List< AnimationGroup>animationGroupsList

This list contains all the animations objects of current Action User needs to play all of the animations to perform the Action

class MAGES::ActionPrototypes::BasePrototype

```
class MAGES::ActionPrototypes::BasePrototype
  : public MonoBehaviour
  : public IAction
```

Base Prototype contains all the shared variables and methods other Action prototypes have It encaptulates the basic functionality to run ang configure each Action All Action prototypes inherit from Base prototype and override the needed functions

Summary

`{property} string` *`ActionName`*

`{property} GameObject` *`ActionNode`*

`public inline virtual void` *`Initialize`*`()` Initialize function of *BasePrototype* sets the shared elements of all Action Prototypes for the Action to run properly These are the event Listeners, the set of physical colliders and the reset of all counters Initialize is a virtual function since it can be overridden from each Action

`public inline virtual void` *`Perform`*`()` Perform function of *BasePrototype* clears current Action to get ready for the next one It also runs all the Unity Actions in performActionList The process of cleaning the current Action includes the deletion of prefabs (holograms), reseting the Event Manager, the kill of Voice Actor and clearing the list of Actions This method is overridden by any Action

`public inline virtual void` *`Undo`*`()` Undo function of Base Prototype has the same functionalities with Perform but instead of running the performActionList it runs the Actions in undoActionList This method is overridden by any Action

`public inline virtual void` *`ActionCall`*`()` ActionCall method is called when a trigger event from ACtions occur. An Action has its own Listener and performs an ActionCall (this method) on a specific situation For example in a Tool Action when the correct tool hits one of teh colliders then *ActionCall()* is triggered This method is overridden by any Action

`public inline virtual void` *`InitializeHolograms`*`()` InitializeHologram method is used to spawn the selected hologram in the current Action This method is overridden by any Action

`public inline virtual void` *`DifficultyRestrictions`*`()` DifficultyRestrictions performs different functionalities depending on the difficulty level Easy: Holograms,Easy ErrorColliders, Aidlines Medium: Medium ErrorColliders Hard: Hard Colliders This method is overridden by any Action

`public inline virtual void` *`SetNextModule`*`(Action action)` Connects the next Action in a combined Action A combimed Action has many actions to perform before going to the next one This methid links the next Action of current combined Action with the given Action

`public inline void` *`SetAidLine`*`(string spawnAidLine)` Sets the aidLines for the specific action

`public inline void SetAfterSpawnAction(Action afterSpawnFunction)` Sets a Unity Action to run right after Action's Initialization

`public inline void SetErrorColliders(string _errorCollidersPath)` Set the error colliders for the Action

`public inline void SetUndoAction(Action undoActionFunction)` Set an action to play on *Undo()*

`public inline void SetPerformAction(Action performActionFunction,int numOfTriggerToPlay)` Set an action to play after a numner of triggers. It will also play on Perform if Not triggererd

`public inline void SetHoloObject(string holoPath,GameObject finalPrefabParent)` Set the hologram for the Action

`public inline void ColliderPrefabCallBackOnTimerChange(params Action< float >[] _callBacks)`

Parameters

`public inline void SetEventListener(string _event)` Sets Event Listener for the current Action Event Listener is being set with a string key that is most of the times the Action's name Event Manager binds this key with the ActionCall of *BasePrototype* to Run when the trigger from the Action occurs

`public inline void SetPhysicalColliderPrefab(string _physicalColliderPath)` Sets the string path for the ACtion's phusical colliders Physical colliders of an Action are all those colliders that are static and non triggered

`public inline void SetNextAction(int lessonID,int stageID,int actionID)`

`protected Dictionary< int, Action > performActionList` Dictionary that contains Unity Actions to run on Perform For Tool Action you can use the key (int) to set a Unity Action to run after specific collider triggers For example: SetPerformAction(MAGES.CharacterController.CharacterController.instance.PlayCut2Animation, 2); Which plays PlayCut2Animation on the second trigger event of Collider prefab

`protected List< Action > undoActionList` List of Unity Actions to execute (Same idea with performActionList) but this time runs on Undo. Important: Triggers BEFORE scenegraph changes to next Action

`protected Action ActionCallOverrideOnFinish` This unity Action is set to the next Action that will run after performing this one This variable is for combined Actions and remains null otherwise

`protected GameObject errorColliders` A reference to the errorColliders gameObject

`protected string physicalColliderPath` A string varible to store the path to the physicalColliderPath prefab

`protected GameObject physicalCollider` A reference to the physicalCollider gameObject

`protected int callBackCounter` A counter to store how many times a specific action the has called its trigger event

`protected int callBackEnd` Counter to set how many trigger events are nedded to Perform the action

`protected List< Action< float > > colliderPrefabCallBackOnTimerChange` This variable is used from Tool and Use Actions to set and restore the use/tool timing

`protected inline void StopFlashing()` Stops controllers flashing

`protected inline GameObject Spawn(string _path,GameObject _parent, InheritTransformFrom _inheritTrans)` Spawns the prefabs for the current action. Beforehand we have store the paths to those prefabs so now its the time to spawn them

`protected inline void DestroyHolograms()` Destroys current Action Holograms

`enum ControllerFlashing` Enumerator to set the proper flashing

`enum InheritTransformFrom` Enumerator to define the positional inheritance of spawned GameObject Used when we want to spawn a prefab relative to another position eg. Users hand, as part of another tool

Members

{property} string ActionName

{property} GameObject ActionNode

public inline virtual void Initialize()

Initialize function of *BasePrototype* sets the shared elements of all Action Prototypes for the Action to run properly These are the event Listeners, the set of physical colliders and the reset of all counters Initialize is a virtual function since it can be overridden from each Action

public inline virtual void Perform()

Perform function of *BasePrototype* clears current Action to get ready for the next one It also runs all the Unity Actions in performActionList The process of cleaning the current Action includes the deletion of prefabs (holograms), reseting the Event Manager, the kill of Voice Actor and clearing the list of Actions This method is overridden by any Action

public inline virtual void Undo()

Undo function of Base Prototype has the same functionalities with Perform but instead of running the performAction-List it runs the Actions in undoActionList This method is overridden by any Action

public inline virtual void ActionCall()

ActionCall method is called when a trigger event from ACtions occur. An Action has its own Listener and performs an ActionCall (this method) on a specific situation For example in a Tool Action when the correct tool hits one of teh colliders then *ActionCall()* is triggered This method is overridden by any Action

public inline virtual void InitializeHolograms()

InitializeHologram method is used to spawn the selected hologram in the current Action This method is overridden by any Action

public inline virtual void DifficultyRestrictions()

DifficultyRestrictions performs different functionalities depending on the difficulty level Easy: Holograms,Easy Error-Colliders, Aidlines Medium: Medium ErrorColliders Hard: Hard Colliders This method is overridden by any Action

public inline virtual void SetNextModule(Action action)

Connects the next Action in a combined Action A combimed Action has many actions to perform before going to the next one This methid links the next Action of current combined Action with the given Action

Parameters

- `action` The nect Action to link the combined Action

public inline void SetAidLine(string spawnAidLine)

Sets the aidLines for the specific action

public inline void SetAfterSpawnAction(Action afterSpawnFunction)

Sets a Unity Action to run right after Action's Initialization

Parameters

- `afterSpawnFunction` The Action to run

public inline void SetErrorColliders(string _errorCollidersPath)

Set the error colliders for the Action

public inline void SetUndoAction(Action undoActionFunction)

Set an action to play on *Undo()*

public inline void SetPerformAction(Action performActionFunction,int numOfTriggerToPlay)

Set an action to play after a numner of triggers. It will also play on Perform if Not triggererd

Parameters

- `numOfTriggerToPlay` The number of triggers to play

public inline void SetHoloObject(string holoPath,GameObject finalPrefabParent)

Set the hologram for the Action

public inline void ColliderPrefabCallBackOnTimerChange(params Action< float >[] _callBacks)

Parameters

- `_callBacks`

public inline void SetEventListener(string _event)

Sets Event Listener for the current Action Event Listener is being set with a string key that is most of the times the Action's name Event Manager binds this key with the ActionCall of *BasePrototype* to Run when the trigger from the Action occurs

Parameters

- `_event` The string key to bind the Event Manager with *BasePrototype*'s Action Call

public inline void SetPhysicalColliderPrefab(string _physicalColliderPath)

Sets the string path for the ACtion's phusical colliders Physical colliders of an Action are all those colliders that are static and non triggered

Parameters

- `_physicalColliderPath` Path to the physical collider prefab

public inline void SetNextAction(int lessonID,int stageID,int actionID)

protected Dictionary< int, Action > performActionList

Dictionary that contains Unity Actions to run on Perform For Tool Action you can use the key (int) to set a Unity Action to run after specific collider triggers For example: SetPerformAction(MAGES.CharacterController.CharacterController.instance.PlayCut2Animation, 2); Which plays PlayCut2Animation on the second trigger event of Collider prefab

protected List< Action > undoActionList

List of Unity Actions to execute (Same idea with performActionList) but this time runs on Undo. Important: Triggers BEFORE scenegraph changes to next Action

protected Action ActionCallOverrideOnFinish

This unity Action is set to the next Action that will run after performing this one This variable is for combined Actions and remains null otherwise

protected GameObject errorColliders

A reference to the errorColliders gameObject

protected string physicalColliderPath

A string varible to store the path to the physicalColliderPath prefab

protected GameObject physicalCollider

A reference to the physicalCollider gameObject

protected int callBackCounter

A counter to store how many times a specific action the has called its trigger event

protected int callBackEnd

Counter to set how many trigger events are nedded to Perform the action

protected List< Action< float > > colliderPrefabCallBackOnTimerChange

This variable is used from Tool and Use Actions to set and restore the use/tool timing

protected inline void StopFlashing()

Stops controllers flashing

protected inline GameObject Spawn(string _path,GameObject _parent, InheritTransformFrom _inheritTrans)

Spawns the prefabs for the current action. Beforehand we have store the paths to those prefabs so now its the time to spawn them

Parameters

- `_path` Path of the prefab we want to spawn
- `_parent` GameObject to spawn under a parend. By default null
- `_inheritTrans` Transformation of another object to spawn relative to this position

Returns

protected inline void DestroyHolograms()

Destroys current Action Holograms

enum ControllerFlashing

TriggerButtonGripButtonNoFlashing

Enumerator to set the proper flashing

enum InheritTransformFrom

nonegrabbablePrefabfinalPrefabremovePrefab

Enumerator to define the positional inheritance of spawned GameObject Used when we want to spawn a prefab relative to another position eg. Users hand, as part of another tool

class MAGES::ActionPrototypes::CombinedAction

```
class MAGES::ActionPrototypes::CombinedAction
  : public MAGES.ActionPrototypes.BasePrototype
```

The combined is used to perform actions that consist of multiple sub-actions sequentially. An example of combined action is when you want to open a door. This can be described as OpenDoorAction consisting of PullHandleAction and PullDoorAction. In general, the combined action is used to describe an action that in order to be performed, the user needs to perform 2 or more 'smaller' actions.

public class OpenDoorAction : *CombinedAction* { public override void *Initialize()* { *UseAction* pullHandleAction = gameObject.AddComponent(); pullHandleAction.SetUsePrefab("Lesson10/Stage0/Action0/pullHandle"); pullHandleAction.SetHoloObject("Lesson10/Stage0/Action0/Hologram/pullHandleActionHologram"); *UseAction* pullDoorAction = gameObject.AddComponent(); pullDoorAction.SetUsePrefab("Lesson10/Stage0/Action0/pullDoor"); pullDoorAction.SetHoloObject("Lesson10/Stage0/Action0/Hologram/pullDoorHologramb");

InsertIActions(pullHandleAction, pullDoorAction); base.Initialize(); } }

Summary

`public inline virtual override void Initialize()` Initialize method for Combined Action overrides base.Initialize() and 1)Deletes and Undoes all previous Actions to set the combined Action 2)Sets the event Listener for current Combined Action

`public inline virtual override void Perform()` Perform method for Combined Action overrides base.Perform() and 1)Performs current sub-Action and Initializes next sub-Action If this is the last sub-Action of Combined Action then Perform the Action 2)Sets event listener to teh next Action

`public inline virtual override void Undo()` Undo method for Combined Action overrides base.Undo() and 1)Undoes Each Action from iActionListQueue 2)Clears Lists 3)Calls undoActionList Actions 4)Resets and initializes Actions

`public inline virtual override void DifficultyRestrictions()` DifficultyRestrictions method for Combined Action overrides base.DifficultyRestrictions() Sets Holograms depending on the difficulty level

`public inline virtual override void SetNextModule(Action action)` Set the next module for the last element of queue

`public inline void InsertIActions(params IAction[] iActions)` Inserts sub-Action to Action Queue

`public inline void NextModuleFinish()` Method to manage sub-Action performs 1)Performs current active sub-Action 2)Sets event Listener for the next sub-Action 3)Initializes next sub-Action

`public inline void SetEventListenerCombined(string _event)` Sets the Event listener with the given key

`public inline new void SetPerformAction(Action performActionFunction,int numOfTriggerToPlay)` Set an action to play after a numner of triggers. It will also play on Perform if Not triggererd

`public inline new void SetUndoAction(Action undoActionFunction)` Set an action to play on *Undo()*

`public inline int GetCurrentSubActionName()`

Members

public inline virtual override void Initialize()

Initialize method for Combined Action overrides base.Initialize() and 1)Deletes and Undoes all previous Actions to set the combined Action 2)Sets the event Listener for current Combined Action

public inline virtual override void Perform()

Perform method for Combined Action overrides base.Perform() and 1)Performs current sub-Action and Initializes next sub-Action If this is the last sub-Action of Combined Action then Perform the Action 2)Sets event listener to teh next Action

```
public inline virtual override void Undo()
```

Undo method for Combined Action overrides base.Undo() and 1)Undoes Each Action from iActionListQueue 2)Clears Lists 3)Calls undoActionList Actions 4)Resets and initializes Actions

```
public inline virtual override void DifficultyRestrictions()
```

DifficultyRestrictions method for Combined Action overrides base.DifficultyRestrictions() Sets Holograms depending on the difficulty level

```
public inline virtual override void SetNextModule(Action action)
```

Set the next module for the last element of queue

Parameters

- `action` Next sub-Action to enQueue

```
public inline void InsertIActions(params IAction[] iActions)
```

Inserts sub-Action to Action Queue

Parameters

- `iActions` Action to set

```
public inline void NextModuleFinish()
```

Method to manage sub-Action performs 1)Performs current active sub-Action 2)Sets event Listener for the next sub-Action 3)Initializes next sub-Action

```
public inline void SetEventListenerCombined(string _event)
```

Sets the Event listener with the given key

Parameters

- `_event` String key to bind current Action call to Event Manager

public inline new void SetPerformAction(Action performActionFunction,int numOfTriggerToPlay)

Set an action to play after a numner of triggers. It will also play on Perform if Not triggererd

Parameters

- `numOfTriggerToPlay` The number of triggers to play

public inline new void SetUndoAction(Action undoActionFunction)

Set an action to play on *Undo()*

public inline int GetCurrentSubActionName()

class MAGES::ActionPrototypes::InsertAction

```
class MAGES::ActionPrototypes::InsertAction
  : public MAGES.ActionPrototypes.BasePrototype
```

Insert Action is referring to a specific type of Action that user has to insert an object to a specific position in order to complete it. For such an action we need three basic elements. First the Interactable prefab which is the object user will grab using his virtual hands or any other input device. The second one is the position we have to place the object. This position is called the Final position of our prefab. The correct orientation of the interactable object relative to the Final position is the key element to complete the Action. The final element to complete the prototyping of an Insert Action is to use a hologram to indicate the correct final position of the hologram. An Insert Action can have many objects to be inserted to the final position. For this reason we can set many prefabs in one Action.

Example of Insert Action: Insert Trial Polyethylene to Tibia. To set this action we have to spawn the interactable polyethylene on a surgical table, the final position of polyethylene and lastly the hologram for the polyethylene in the correct position. To complete the Action user needs to grab the polyethylene and place it with the correct rotation on the final position that the hologram indicates.

public class PolyethyleneTrialAction : *InsertAction* { public override void *Initialize()* { SetInsertPrefab("Lesson7/Stage2/Action0/Polyethylene", "Lesson7/Stage2/Action0/PolyethyleneFinal", "Lesson7/Stage2/Action0/Hologram/HologramL7S2A0"); base.Initialize(); } }

Summary

`public inline virtual override void` *`Initialize`*`()` Initialize method for Insert Action overrides base.Initialize() and 1)Destroys all final prefabs from remoteDestroyPrefabsList 1)Spawns all insert prefabs from the prefab list 2)Sets the counter that will Perform the Action to the number of the inserted prefabs

`public inline virtual override void` *`Perform`*`()` Perform method for Insert Action overrides base.Perform() and 1)Finalizes all insert prefabs 2)Clears Action Lists

`public inline virtual override void` *`Undo`*`()` Undo method for Insert Action overrides base.Undo() and 1)Destroys insert Prefabs 2)Clears Action Lists

`public inline InsertGroup SetInsertPrefab(string grabbablePrefabPath,string finalPrefabPath,GameObject finalPrefabParent,GameObject grabbableParent, InheritTransformFrom _inheritTransformGrabbale,InheritTransformFrom _inheritTransformFinal)` Set prefabs for a single instrument

`protected List< InsertGroup>insertGroupsList` This list contains all the insert objects of current Action User needs to insert all of them to perform the Action

Members

public inline virtual override void Initialize()

Initialize method for Insert Action overrides base.Initialize() and 1)Destroys all final prefabs from remoteDestroyPrefabsList 1)Spawns all insert prefabs from the prefab list 2)Sets the counter that will Perform the Action to the number of the inserted prefabs

public inline virtual override void Perform()

Perform method for Insert Action overrides base.Perform() and 1)Finalizes all insert prefabs 2)Clears Action Lists

public inline virtual override void Undo()

Undo method for Insert Action overrides base.Undo() and 1)Destroys insert Prefabs 2)Clears Action Lists

public inline InsertGroup SetInsertPrefab(string grabbablePrefabPath,string finalPrefabPath,GameObject finalPrefabParent,GameObject grabbableParent, InheritTransformFrom _inheritTransformGrabbale,InheritTransformFrom _inheritTransformFinal)

Set prefabs for a single instrument

Parameters

- `grabbablePrefabPath` The prefab user will grab
- `finalPrefabPath` The final position of the instrument
- `hologramPath` The hologram of the final position

protected List< InsertGroup>insertGroupsList

This list contains all the insert objects of current Action User needs to insert all of them to perform the Action

class MAGES::ActionPrototypes::PumpAction

```
class MAGES::ActionPrototypes::PumpAction
  : public MAGES.ActionPrototypes.BasePrototype
```

The Pump Action is used in situations where the user needs to press a button or a tool (e.g. syringe) It is also used in Actions where user needs to to use a tool (e.g. syringe) to insert liquid or air by making the pump gesture (technically pressing the trigger button). In those Actions we need an object for user to interact with and press the trigger button from the controller (as many times he set beforehand) to Perform the Action.

A pump Action needs a single or multiple pump prefabs The pump prefab should have the following components: 1) MAGESInteractableItem with small drop distance (0.08) and a rigidbody 2) NetworkIdentity and MAGESNetTransform for Coop 3) MAGESAutoAttach to enable the hand lock mechanic automatically 4) PumpPrefabConstructor and PumpBehaviour with the configured settings and animations 5) MAGESEnableDisableOnAttach for the hand lock mechanic. We suggest to set both hands 6) AudioSource in case the pump Action makes any sound 7) Collider to enable the auto attach mechanic when triggered with the hands

Example of Pump Action: Inflate Balloon Catheter In this Action user needs to take a syringe and inflate the balloon catheter. The first step is to place the syringe on the catheter (insert Action). To perform the Pump Action he/she needs to hover his hand on top of the syringe, then the hand lock mechanic will be enabled and his hand will lock in place (first frame from the recorded animation) to register the pump, user needs to press the trigger button. By pressing the trigger button the syringe will start playing the prerecorded animation according to the input from the trigger button. The Action will Perform when the user completes the desired amount of pumps.

Pump Actions can register different pump prefabs within the same Action by calling the SetPumpPrefab method multiple times. In this case the Action will perform as soon as user completes all the pumps from all the different prefabs

In addition the SetPumpPrefab method has an optional argument (performActionFunction) that a developer can use to invoke a method after completing all the pumps of the spawned pump prefab (e.g the SyringePumping)

public class InflateBalloonCatheter : *PumpAction* { public override void *Initialize()* { SetPumpPrefab("Lesson0/Stage1/Action1/SyringePumping", catheter);

base.Initialize(); }

}

Summary

`public inline virtual override void Initialize()` Initialize function of *BasePrototype* sets the shared elements of all Action Prototypes for the Action to run properly These are the event Listeners, the set of physical colliders and the reset of all counters Initialize is a virtual function since it can be overridden from each Action

`public inline virtual override void Perform()` Perform function of *BasePrototype* clears current Action to get ready for the next one It also runs all the Unity Actions in performActionList The process of cleaning the current Action includes the deletion of prefabs (holograms), reseting the Event Manager, the kill of Voice Actor and clearing the list of Actions This method is overridden by any Action

`public inline virtual override void Undo()` Undo function of Base Prototype has the same functionalities with Perform but instead of running the performActionList it runs the Actions in undoActionList This method is overridden by any Action

`public inline void SetWaitForAllPumps(bool wait)`

`public inline void SetPumpPrefab(string _pumpPrefabPath)` Sets a single pump prefab

`public inline void SetPumpPrefab(string _pumpPrefabPath,GameObject _pumpParent,Action _performActionFunction)` Sets a single pump prefab

`public inline void SetPumpPrefab(string _pumpPrefabPath,Action _performActionFunction)` Sets a single pump prefab

Members

public inline virtual override void Initialize()

Initialize function of *BasePrototype* sets the shared elements of all Action Prototypes for the Action to run properly These are the event Listeners, the set of physical colliders and the reset of all counters Initialize is a virtual function since it can be overridden from each Action

public inline virtual override void Perform()

Perform function of *BasePrototype* clears current Action to get ready for the next one It also runs all the Unity Actions in performActionList The process of cleaning the current Action includes the deletion of prefabs (holograms), reseting the Event Manager, the kill of Voice Actor and clearing the list of Actions This method is overridden by any Action

public inline virtual override void Undo()

Undo function of Base Prototype has the same functionalities with Perform but instead of running the performAction-List it runs the Actions in undoActionList This method is overridden by any Action

public inline void SetWaitForAllPumps(bool wait)

public inline void SetPumpPrefab(string _pumpPrefabPath)

Sets a single pump prefab

Parameters

- `_pumpPrefabPath` Path to the pump prefab

public inline void SetPumpPrefab(string _pumpPrefabPath,GameObject _pumpParent,Action _performActionFunction)

Sets a single pump prefab

Parameters

- `_pumpPrefabPath` Path to the pump prefab
- `_pumpParent` Parent for the pump prefab
- `_performActionFunction` Action call to invoke after completing all the pumps on a prefab

public inline void SetPumpPrefab(string _pumpPrefabPath,Action _performActionFunction)

Sets a single pump prefab

Parameters

- `_pumpPrefabPath` Path to the pump prefab
- `_performActionFunction` Action call to invoke after completing all the pumps on a prefab

class MAGES::ActionPrototypes::QuestionAction

```
class MAGES::ActionPrototypes::QuestionAction
  : public MAGES.ActionPrototypes.BasePrototype
```

Summary

`public bool` *`m_Ready`*

`public inline void` *`SetQuestionPrefab`*`(string _questionPrefabPath)`

`public inline virtual override void` *`ActionCall`*`()` ActionCall method is called when a trigger event from ACtions occur. An Action has its own Listener and performs an ActionCall (this method) on a specific situation For example in a Tool Action when the correct tool hits one of teh colliders then *ActionCall()* is triggered This method is overridden by any Action

`public inline virtual override void` *`DifficultyRestrictions`*`()` DifficultyRestrictions performs different functionalities depending on the difficulty level Easy: Holograms,Easy ErrorColliders, Aidlines Medium: Medium ErrorColliders Hard: Hard Colliders This method is overridden by any Action

`public inline virtual override void` *`Initialize`*`()` Initialize function of *BasePrototype* sets the shared elements of all Action Prototypes for the Action to run properly These are the event Listeners, the set of physical colliders and the reset of all counters Initialize is a virtual function since it can be overridden from each Action

`public inline virtual override void` *`InitializeHolograms`*`()` InitializeHologram method is used to spawn the selected hologram in the current Action This method is overridden by any Action

`public inline virtual override void` *`Perform`*`()` Perform function of *BasePrototype* clears current Action to get ready for the next one It also runs all the Unity Actions in performActionList The process of cleaning the current Action includes the deletion of prefabs (holograms), reseting the Event Manager, the kill of Voice Actor and clearing the list of Actions This method is overridden by any Action

`public inline virtual override void` *`SetNextModule`*`(Action action)` Connects the next Action in a combined Action A combimed Action has many actions to perform before going to the next one This methid links the next Action of current combined Action with the given Action

`public inline virtual override void` *`Undo`*`()` Undo function of Base Prototype has the same functionalities with Perform but instead of running the performActionList it runs the Actions in undoActionList This method is overridden by any Action

Members

public bool m_Ready

public inline void SetQuestionPrefab(string _questionPrefabPath)

public inline virtual override void ActionCall()

ActionCall method is called when a trigger event from ACtions occur. An Action has its own Listener and performs an ActionCall (this method) on a specific situation For example in a Tool Action when the correct tool hits one of teh colliders then *ActionCall()* is triggered This method is overridden by any Action

public inline virtual override void DifficultyRestrictions()

DifficultyRestrictions performs different functionalities depending on the difficulty level Easy: Holograms,Easy ErrorColliders, Aidlines Medium: Medium ErrorColliders Hard: Hard Colliders This method is overridden by any Action

public inline virtual override void Initialize()

Initialize function of *BasePrototype* sets the shared elements of all Action Prototypes for the Action to run properly These are the event Listeners, the set of physical colliders and the reset of all counters Initialize is a virtual function since it can be overridden from each Action

public inline virtual override void InitializeHolograms()

InitializeHologram method is used to spawn the selected hologram in the current Action This method is overridden by any Action

public inline virtual override void Perform()

Perform function of *BasePrototype* clears current Action to get ready for the next one It also runs all the Unity Actions in performActionList The process of cleaning the current Action includes the deletion of prefabs (holograms), reseting the Event Manager, the kill of Voice Actor and clearing the list of Actions This method is overridden by any Action

public inline virtual override void SetNextModule(Action action)

Connects the next Action in a combined Action A combimed Action has many actions to perform before going to the next one This methid links the next Action of current combined Action with the given Action

Parameters

- `action` The nect Action to link the combined Action

public inline virtual override void Undo()

Undo function of Base Prototype has the same functionalities with Perform but instead of running the performAction-List it runs the Actions in undoActionList This method is overridden by any Action

class MAGES::ActionPrototypes::RemoveAction

```
class MAGES::ActionPrototypes::RemoveAction
  : public MAGES.ActionPrototypes.BasePrototype
```

Remove Action describes a step of the procedure which user has to remove an object using his hands or a tool.To implement a Remove Action we have to set the object that will be removed.This object will spawn as the removable prefab and to complete the Action user needs to reach and remove it.The removable prefab will flash to indicate that it needs to be removed to complete the Action.Remove Action also supports removing an object using a tool instead of hands. This tool can be a pair of plyers or another tool that supports the grab mechanism.

Example of Remove Action: Remove osteophytes To set this remove Action we need to spawn the osteophytes with the flashing mechanic and also set the correct parameters that enables removing with tool. Remove Action supports multiple removal of objects.As a result in this example we can set all the osteophytes as removable prefabs within the same Action. To complete the Action we have to take the plyers from the tool table and remove all the osteophytes.

public class RemoveOsteophytesAction : *RemoveAction* { public override void *Initialize()* { SetRemovePrefab("Lesson0/Stage4b/Action0/FemoralSpur1", femur); SetRemovePrefab("Lesson0/Stage4b/Action0/FemoralSpur2", femur); SetRemovePrefab("Lesson0/Stage4b/Action0/FemoralSpur3", femur); SetRemovePrefab("Lesson0/Stage4b/Action0/TibialSpur1", tibia); SetHoloObject("Lesson0/Stage4b/Action0/Hologram/HologramL0S4bA0"); SetErrorColliders("Lesson0/Stage4b/Action0/Restrictions/ErrorsColliders");

base.Initialize(); } }

Summary

`public inline virtual override void` *`Initialize`*`()` Initialize method for Remove Action overrides base.Initialize() and 1)Spawns all removable prefabs from the prefab list 2)Sets the counter that will Perform the Action to the number of the removable prefabs

`public inline virtual override void` *`Perform`*`()` Perform method for Remove Action overrides base.Perform() and 1)Destroys Remove prefabs 2)Clears Action Lists

`public inline virtual override void` *`Undo`*`()` Undo method for Remove Action overrides base.Undo() and 1)Destroys Remove Prefabs 2)Clears Action Lists

`public inline void` *`SetRemovePrefab`*`(string prefabPath,GameObject toBeParent)` Set the prefab user will remove

`public inline void` *`SetRemovePrefab`*`(string grabbablePrefabPath,string removePrefabPath,GameObject removePrefabParent,`*`InheritTransformFrom`*` _inheritTransformGrabbale,`*`InheritTransformFrom`*` _inheritTransformRemove)` Set the prefab user will remove

Members

public inline virtual override void Initialize()

Initialize method for Remove Action overrides base.Initialize() and 1)Spawns all removable prefabs from the prefab list 2)Sets the counter that will Perform the Action to the number of the removable prefabs

public inline virtual override void Perform()

Perform method for Remove Action overrides base.Perform() and 1)Destroys Remove prefabs 2)Clears Action Lists

public inline virtual override void Undo()

Undo method for Remove Action overrides base.Undo() and 1)Destroys Remove Prefabs 2)Clears Action Lists

public inline void SetRemovePrefab(string prefabPath,GameObject toBeParent)

Set the prefab user will remove

Parameters

- prefabPath

public inline void SetRemovePrefab(string grabbablePrefabPath,string removePrefabPath,GameObject removePrefabParent,InheritTransformFrom _inheritTransformGrabbale,InheritTransformFrom _inheritTransformRemove)

Set the prefab user will remove

Parameters

- prefabPath

class MAGES::ActionPrototypes::ToolAction

```
class MAGES::ActionPrototypes::ToolAction
  : public MAGES.ActionPrototypes.BasePrototype
```

Tool Action is referring to an Action that user has to take a tool and use it to complete the action.Technically speaking, a Tool Action needs only a set of colliders to trigger a specific tool when they will come in contact.This set of colliders are referring as the Collider prefab and contain the necessary information for the selected tool to trigger them.

Example of Tool Action: Cut the cloth using scissors To set this Tool Action we need to spawn the collider that will trigger with the scissors tool.It is also recommended to provide a holographic animated scissors performing the action indicating the correct cut or simple line-dots at the colliders we want to cut. To complete the Action user has to take the scissors from the tool table an cut the spawned tool colliders. When scissors interact with all colliders the Action will perform.

public class CutClothAction : *ToolAction* { public override void *Initialize()* { SetToolActionPrefab("Lesson0/Stage0/Action0/CutClothColliders" , ToolsEnum.Scissors); SetErrorColliders("Lesson0/Stage0/Action0/Restrictions/ErrorsColliders"); SetHoloObject("Lesson0/Stage0/Action0/Hologram/HologramL0S0A0");

base.Initialize(); } }

Summary

`public inline virtual override void` *`Initialize`*`()` Initialize method for Tool Action overrides base.Initialize() and 1)Spawns tool collider 2)Sets the counter that will Perform the Action if all the colliders will be triggered by the tool IMPORTANT: This collider will be set to the childs of tool collider that will have the ToolTriggerCollider

`public inline virtual override void` *`Perform`*`()` Perform method for Tool Action overrides base.Perform() and 1)Destroys colldider prefab

`public inline virtual override void` *`Undo`*`()` Undo method for Tool Action overrides base.Undo() and 1)Destroys tool collider prefab

`public inline void` *`SetToolActionPrefab`*`(string prefabPath,ToolsEnum _tool, GameObject _PrefabParent)` Set the trigger colliders that will interact with the tool

Members

public inline virtual override void Initialize()

Initialize method for Tool Action overrides base.Initialize() and 1)Spawns tool collider 2)Sets the counter that will Perform the Action if all the colliders will be triggered by the tool IMPORTANT: This collider will be set to the childs of tool collider that will have the ToolTriggerCollider

public inline virtual override void Perform()

Perform method for Tool Action overrides base.Perform() and 1)Destroys colldider prefab

public inline virtual override void Undo()

Undo method for Tool Action overrides base.Undo() and 1)Destroys tool collider prefab

public inline void SetToolActionPrefab(string prefabPath,ToolsEnum _tool, GameObject _PrefabParent)

Set the trigger colliders that will interact with the tool

class MAGES::ActionPrototypes::UseAction

```
class MAGES::ActionPrototypes::UseAction
  : public MAGES.ActionPrototypes.BasePrototype
```

Use Action is similar to a Tool Action but instead of a tool we use another object to complete the Action.In order to properly set the Action we have to spawn the collider prefab(as the tool Action) but in this case we also need to provide an extra object to interact with (instead of the tool).

Example of Use Action: Clean statue with a cloth To set this action we have to spawn the cloth and the collider that will interact with. To simulate the cleaning process it is recommended to set the use time variable around 3-5 seconds to wait that much time in contact with the cloth and then trigger the Perform to complete the Action. It is also recommended to spawn an animated hologram to indicate the proper object and the correct way to use it. To complete the action user has to take the cloth and place it on the statue for the amount of time we have set.

public class CleanStatueAction : *UseAction* { public override void *Initialize()* { SetUsePrefab("Lesson0/Stage2/Action0/ClothPrefab"); SetColliderPrefab("Lesson0/Stage2/Action0/ClothCollider"); SetHoloObject("Lesson0/Stage2/Action0/Hologram/HologramL0S2A0");

base.Initialize(); } }

Summary

`public inline virtual override void` *`Initialize`*`()` Initialize method for Use Action overrides base.Initialize() and 1)Spawns collider and use Prefabs

`public inline virtual override void` *`Perform`*`()` Perform method for Use Action overrides base.Perform() and 1)Destroys colldider prefab 2)Sets use prefab to be destroyed after thrown by user

`public inline virtual override void` *`Undo`*`()` Undo method for Use Action overrides base.Undo and 1)Destroys collider and use Prefabs

`public inline void` *`SetUsePrefab`*`(string _usePrefabPath,string _colliderPrefabPath,GameObject _colliderParent)` Set the Grabbable prefab to use in Action

`protected inline GameObject` *`GetUsePrefab`*`()`

Members

public inline virtual override void Initialize()

Initialize method for Use Action overrides base.Initialize() and 1)Spawns collider and use Prefabs

public inline virtual override void Perform()

Perform method for Use Action overrides base.Perform() and 1)Destroys colldider prefab 2)Sets use prefab to be destroyed after thrown by user

public inline virtual override void Undo()

Undo method for Use Action overrides base.Undo and 1)Destroys collider and use Prefabs

public inline void SetUsePrefab(string _usePrefabPath,string _colliderPrefabPath,GameObject _colliderParent)

Set the Grabbable prefab to use in Action

Parameters

- usePrefabPath

protected inline GameObject GetUsePrefab()

namespace MAGES::AutomaticDestroy

Summary

class *MAGES::AutomaticDestroy::AutomaticObjectDestroy* This script when set to an object, spawn and destroyed it as set in DestroyTime List

class MAGES::AutomaticDestroy::AutomaticObjectDestroy

```
class MAGES::AutomaticDestroy::AutomaticObjectDestroy
  : public MonoBehaviour
```

This script when set to an object, spawn and destroyed it as set in DestroyTime List

Summary

public List< DestroyTime > *destroyTime* Selected the actions where the prefab can be deleted. Leave empty if the prefab is going to be destroyed by the same action that spawned it

Members

public List< DestroyTime > destroyTime

Selected the actions where the prefab can be deleted. Leave empty if the prefab is going to be destroyed by the same action that spawned it

namespace MAGES::AutomaticDestroyFunctions

Summary

class *MAGES::AutomaticDestroyFunctions::AutomaticObjectDestroyFuncs* Internal class containing all the functionalities for automatic/spawn Destroy objects

class MAGES::AutomaticDestroyFunctions::AutomaticObjectDestroyFuncs

Internal class containing all the functionalities for automatic/spawn Destroy objects

class MAGES::ActionPrototypes::AnimationAction::AnimationGroup

Helper class to group the animation Action prefabs

Summary

public string *grabbablePrefabPath* Paths to the prefabs that will spawned

public GameObject *grabbablePrefab* GameObjects to store spawned prefabs

public GameObject *parentGameObject* Parent GameObject

public *InheritTransformFrom inheritGrabbable* Transforms of objects we need to spawn the prefabs in relative position

public inline *AnimationGroup*(string _grabbablePrefabPath,GameObject _parentGameObject,*InheritTransformFrom* _inheritGrabbable) Constructor to generate an animation group

Members

public string grabbablePrefabPath

Paths to the prefabs that will spawned

public GameObject grabbablePrefab

GameObjects to store spawned prefabs

public GameObject parentGameObject

Parent GameObject

public InheritTransformFrom inheritGrabbable

Transforms of objects we need to spawn the prefabs in relative position

public inline AnimationGroup(string _grabbablePrefabPath,GameObject _parentGameObject,InheritTransformFrom _inheritGrabbable)

Constructor to generate an animation group

class MAGES::ActionPrototypes::BasePrototype::InstrumentTransforms::CustomTransfrom

Instrument transformation values

Summary

public Vector3 *position*

public Quaternion *rotation*

Members

public Vector3 position

public Quaternion rotation

class MAGES::ActionPrototypes::BasePrototype::HoloGroup

Helper class for holograms

Summary

public string *holoPath* Path for the prefab

public GameObject *holoParent* Possible holographic parent

public GameObject *holoObject* The actual hologram

public inline void *SetHoloObject*(GameObject _holoObject) Set Hologram Gameobject to class variable

public inline *HoloGroup*(string _holoPath,GameObject _holoParent)

Members

public string holoPath

Path for the prefab

public GameObject holoParent

Possible holographic parebn

public GameObject holoObject

The actual hologram

public inline void SetHoloObject(GameObject _holoObject)

Set Hologram Gameobject to class variable

Parameters

- _holoObject

public inline HoloGroup(string _holoPath,GameObject _holoParent)

Summary

public Action *action* Action to call after user completes this specific Action

public inline *IActionGroup*(int _numberofPath,IAction _iAction,Action _action) Constructor to set each Action

Members

public int numberofPath

The number of scenegraph path that will be triggered by Perfrorming this Action

public Action action

Action to call after user completes this specific Action

public inline IActionGroup(int _numberofPath,IAction _iAction,Action _action)

Constructor to set each Action

class MAGES::ActionPrototypes::RemoveAction::InsertGroup

For Remove Action we need an extra class to group the prefabs that will be spawned

Summary

`public string` *grabbablePrefabPath* Paths to the prefabs that will be spawned

`public GameObject` *grabbablePrefab* GameObjects of prefabs that will be spawned

`public` *InheritTransformFrom inheritGrabbable* Transforms of objects we need to spawn the prefabs in relative position

`public inline` *InsertGroup*`(string _grabbablePrefabPath,string _removeprefabPath,GameObject _parentPrefab,`*InheritTransformFrom*` _inheritGrabbable,`*InheritTransformFrom*` _inheritRemove)` Constructor to generate an insert group

Members

public string grabbablePrefabPath

Paths to the prefabs that will be spawned

public GameObject grabbablePrefab

GameObjects of prefabs that will be spawned

public InheritTransformFrom inheritGrabbable

Transforms of objects we need to spawn the prefabs in relative position

public inline InsertGroup(string _grabbablePrefabPath,string _removeprefabPath,GameObject _parentPrefab,InheritTransformFrom _inheritGrabbable,InheritTransformFrom _inheritRemove)

Constructor to generate an insert group

class MAGES::ActionPrototypes::InsertAction::InsertGroup

Helper class to group the insert Action prefabs

Summary

`public string` *grabbablePrefabPath* Paths to the prefabs that will spawned

`public GameObject` *grabbablePrefab* GameObjects to store spawned prefabs

`public GameObject` *finalParent* GameObject to set the parent of spawned prefab if needed

`public` *InheritTransformFrom inheritGrabbable* Transforms of objects we need to spawn the prefabs in relative position

`public inline` *InsertGroup*`(string _grabbablePrefabPath,string _finalPrefabPath, GameObject _finalParent,GameObject _grabbableParent,` *InheritTransformFrom* `_inheritGrabbable,` *InheritTransformFrom* `_inheritFinal)` Constructor to generate an insert group

Members

public string grabbablePrefabPath

Paths to the prefabs that will spawned

public GameObject grabbablePrefab

GameObjects to store spawned prefabs

public GameObject finalParent

GameObject to set the parent of spawned prefab if needed

public InheritTransformFrom inheritGrabbable

Transforms of objects we need to spawn the prefabs in relative position

public inline InsertGroup(string _grabbablePrefabPath,string _finalPrefabPath, GameObject _finalParent,GameObject _grabbableParent,InheritTransformFrom _inheritGrabbable,InheritTransformFrom _inheritFinal)

Constructor to generate an insert group

class MAGES::ActionPrototypes::BasePrototype::InstrumentTransforms

This class is used to store information about the instrument's position There are some cases we want to know the position of an instrument to spawn another one relative to this position. This methodology is used in actions that user assembles instrumets. Since user handles the instruments with shis hand we need to sknow where those intstruments (or his hands) are located.

class MAGES::AutomaticDestroyFunctions::AutomaticObjectDestroyFuncs::PlayAnimationAfterEnabl

```
class␣
↪MAGES::AutomaticDestroyFunctions::AutomaticObjectDestroyFuncs::PlayAnimationAfterEnable
  : public MonoBehaviour
```

Interal class need to Play again animation if the action is perform before the animation finish Automatic removed when it is finished

Summary

`public string` *prefabAnimName* Animation name

Members

public string prefabAnimName

Animation name

class MAGES::ActionPrototypes::PumpAction::PumpGroup

For Pump Action we need an extra class to group the prefabs that will be spawned

Summary

`public string pumpPrefabPath` Paths to the prefabs that will be spawned

`public Action performActionFunction` Action call to invoke after completing all the pumps on a prefab (optional)

`public GameObject pumpPrefab` GameObjects of prefabs that will be spawned

`public inline PumpGroup(string _pumpPrefabPath,GameObject _pumpPrefabParent, Action _performActionFunction)` Constructor to generate a *PumpGroup*

Members

`public string pumpPrefabPath`

Paths to the prefabs that will be spawned

`public Action performActionFunction`

Action call to invoke after completing all the pumps on a prefab (optional)

`public GameObject pumpPrefab`

GameObjects of prefabs that will be spawned

`public inline PumpGroup(string _pumpPrefabPath,GameObject _pumpPrefabParent, Action _performActionFunction)`

Constructor to generate a *PumpGroup*

8.1.2 Game Controller

`namespace MAGES::GameController`

`namespace MAGES::GameController::Network`

namespace `MAGES::GameController`

Summary

`enum ControllerDOF`

`class MAGES::GameController::AudioController` Controls the sounds used for each action in the application `class MAGES::GameController::DeviceController` This must be overriden by each controller `class MAGES::GameController::MAGESControllerClass` This class is the link between the device manager and network manager The Device manager needs to be overriden for each controller type

Members

enum ControllerDOF

TwoDOFThreeDOFSixDOF

class MAGES::GameController::AudioController

```
class MAGES::GameController::AudioController
  : public MonoBehaviour
```

Controls the sounds used for each action in the application

Summary

{property} *AudioController* [Get](#classovid_v_r_1_1_game_controller_1_1_audio_controller_1ae47ce91b9223f780ba2c

public inline void *ResetAudio*() Resets the audio source properties public inline void *PlayPerform*() Play 'perform' sound public inline void *PlayError*() Play 'error' sound public inline void *PlayCriticalError*() Play 'critical error' sound public inline void *PlayUndo*() Play 'undo' sound protected inline *AudioController*()

Members

{property} AudioController Get

public inline void ResetAudio()

Resets the audio source properties

public inline void PlayPerform()

Play 'perform' sound

public inline void PlayError()

Play 'error' sound

public inline void PlayCriticalError()

Play 'critical error' sound

public inline void PlayUndo()

Play 'undo' sound

protected inline AudioController()

class MAGES::GameController::DeviceController

```
class MAGES::GameController::DeviceController
  : public MonoBehaviour
```

This must be overriden by each controller

Summary

public *ControllerDOF DOF* Current *DeviceController* degrees of freedom

public inline virtual Material *GetThumpStickMaterial*(*MAGESControllerClass.MAGESHand* _hand) Controller Material functions that returns the material of the "_hand" thumpstick/trackpad

public inline virtual void *SetThumpSticMaterial*(*MAGESControllerClass.MAGESHand* _hand,Material mat) Sets the material of the thumpstick or trackpad

public abstract void *SetHandColor*(Color _handColor,float _flashSpeed, *MAGESControllerClass.MAGESHand* _hand) Sets the Color of the desired hand

public abstract void *ResetHandsColor*() Resets the color of both hands and stops the flashing

public abstract void *SetDefaultHandsColor*(Color _color) Sets the default color for both hands

public abstract void *SetControllerState*(*MAGESControllerClass.MAGESHand* _hand, bool state) Sets the state of the controller (enable or disabled)

public abstract void *SetLeftHandTransparency*(float _alpha)

public abstract void *SetRightHandTransparency*(float _alpha)

public abstract void *SetMenuButtonFlashing*(*MAGESControllerClass.MAGESHand* _hand)

public abstract void *StopMenuButtonFlashing*(*MAGESControllerClass.MAGESHand* _hand)

public abstract void *SetButtonFlashing*(bool _enabled,*MAGESControllerClass.MAGESHand*_hand,params*MAGESControllerClass.MAGESControllerButtons* _buttons) Sets a flashing animation to the given buttons

public abstract void *SetHandTransparency*(float _alpha,*MAGESControllerClass.MAGESHand* _hand) Sets hand transparency

public abstract bool *MovementEnabled*(*MAGESControllerClass.MAGESHand* _hand)

public abstract bool *IsMoving*()

public abstract bool *GetTrackPadPressed*(*MAGESControllerClass.MAGESHand* _hand) Gets the trackpad/thumpstick current state

public abstract bool *GetTriggerPressed*(*MAGESControllerClass.MAGESHand* _hand) Gets the trigger current state

public abstract bool *GetGripPressed*(*MAGESControllerClass.MAGESHand* _hand) Gets the grip state

public inline virtual *MAGESControllerClass.ControllerTypes GetControllerType*() Gets the state of the controller

public inline virtual void *SetControllerType*(*MAGESControllerClass.ControllerTypes* _controllerTypes) Sets the controller state

public abstract float *GetControllerGrabStrength*(*MAGESControllerClass.MAGESHand* _hand) Function to get the controllers grab strength value.

public abstract string *GetHandTag*(GameObject child) Get a string representation of the hand tag

public abstract Transform *GetHandTransform*(GameObject child) Returns the Transform of the current controllers hand gameobject

public abstract void *ControllerHapticPulse*(*MAGESControllerClass.MAGESHand* _hand,float strength,float _freq,float _duration) Creates a haptic pulse on vive controller

public abstract bool *GetIsGrabed*(*MAGESControllerClass.MAGESHand* _hand) Get the state if the grip button is pressed

public abstract void *SetBothControllersConnection*(bool _bothContrConnected)

public abstract Vector2 *GetThumpPosOnController*(*MAGESControllerClass.MAGESHand* _hand) Returns the position of the thump

public abstract *MAGESControllerClass.MAGESControllerButtons GetPressedButtons*() Sees which buttons are pressed

Members

public ControllerDOF DOF

Current *DeviceController* degrees of freedom

public inline virtual Material GetThumpStickMaterial(MAGESControllerClass.MAGESHand _hand)

Controller Material functions that returns the material of the "_hand" thumpstick/trackpad

Parameters

- `hand`

Returns

Returns the thumpstick/trackpad material

public inline virtual void SetThumpSticMaterial(MAGESControllerClass.MAGESHand _hand,Material mat)

Sets the material of the thumpstick or trackpad

Parameters

- `_hand` Which hand
- `mat` Material to set

public abstract void SetHandColor(Color _handColor,float _flashSpeed, MAGESControllerClass.MAGESHand _hand)

Sets the Color of the desired hand

Parameters

- `_hand`
- `_handColor`
- `_flashSpeed`

public abstract void ResetHandsColor()

Resets the color of both hands and stops the flashing

public abstract void SetDefaultHandsColor(Color _color)

Sets the default color for both hands

Parameters

- `_color`

public abstract void SetControllerState(MAGESControllerClass.MAGESHand _hand, bool state)

Sets the state of the controller (enable or disabled)

Parameters

- `_hand`
- `state` true -> enabled

public abstract void SetLeftHandTransparency(float _alpha)

public abstract void SetRightHandTransparency(float _alpha)

public abstract void SetMenuButtonFlashing(MAGESControllerClass.MAGESHand _hand)

public abstract void StopMenuButtonFlashing(MAGESControllerClass.MAGESHand _hand)

public abstract void SetButtonFlashing(bool _enabled,MAGESControllerClass.MAGESHand_hand,paramsMAGESControllerClass.MAGESControllerButtons _buttons)

Sets a flashing animation to the given buttons

Parameters

- `_enabled` Enable OR disable flashing animation
- `_hand` The hand to use
- `_buttons` Array or list of buttons to use

public abstract void SetHandTransparency(float _alpha,MAGESControllerClass.MAGESHand _hand)

Sets hand transparency

Parameters

- `_alpha` Transparency Value
- `_hand` Hand: left OR right

```
public abstract bool MovementEnabled(MAGESControllerClass.MAGESHand _hand)
```

```
public abstract bool IsMoving()
```

```
public abstract bool GetTrackPadPressed(MAGESControllerClass.MAGESHand _hand)
```

Gets the trackpad/thumpstick current state

Parameters

- `_hand`

Returns

Returns true if the thumpstick/trackpad is pressed

```
public abstract bool GetTriggerPressed(MAGESControllerClass.MAGESHand _hand)
```

Gets the trigger current state

Parameters

- `_hand`

Returns

Returns trus if trigger is pressed

```
public abstract bool GetGripPressed(MAGESControllerClass.MAGESHand _hand)
```

Gets the grip state

Parameters

- `_hand`

Returns

Returns true if grip buttons is pressed

public inline virtual MAGESControllerClass.ControllerTypes GetControllerType()

Gets the state of the controller

Returns

Returns true if the controller is enabled, false otherwise

public inline virtual void SetControllerType(MAGESControllerClass.ControllerTypes _controllerTypes)

Sets the controller state

Parameters

- `_controllerTypes`

public abstract float GetControllerGrabStrength(MAGESControllerClass.MAGESHand _hand)

Function to get the controllers grab strength value.

Returns -2.0f if no controller is specified

Parameters

- `hand` Can be either “left” or “right”

Returns

public abstract string GetHandTag(GameObject child)

Get a string representation of the hand tag

Parameters

- `child` The child gameobject, usually the colliding object

Returns

Returns the Tag of the current controllers hand gameobject

Returns null if no controller is specified

```
public abstract Transform GetHandTransform(GameObject child)
```

Returns the Transform of the current controllers hand gameobject

Parameters

- `child` The child collider of the hand

Returns

Returns the transform of the current hand gameobject

Returns null if no controller is specified

```
public abstract void ControllerHapticPulse(MAGESControllerClass.MAGESHand
_hand,float strength,float _freq,float _duration)
```

Creates a haptic pulse on vive controller

Parameters

- `hand` Left OR right controller
- `strength` The pulse strength : [Range 0 -> 1]

```
public abstract bool GetIsGrabed(MAGESControllerClass.MAGESHand _hand)
```

Get the state if the grip button is pressed

Parameters

- `hand` Hand to look if pressed

Returns

Returns true if the grip button is pressed

public abstract void SetBothControllersConnection(bool _bothContrConnected)

public abstract Vector2 GetThumpPosOnController(MAGESControllerClass.MAGESHand _hand)

Returns the position of the thump

Parameters

- `hand` The hand that grabs an object in the scene

Returns

Returns an cartesian representation of the thump on the controller

public abstract MAGESControllerClass.MAGESControllerButtons GetPressedButtons()

Sees which buttons are pressed

Returns

Returns a list of the presseed buttons

class MAGES::GameController::MAGESControllerClass

```
class MAGES::GameController::MAGESControllerClass
  : public MonoBehaviour
```

This class is the link between the device manager and network manager The Device manager needs to be overriden for each controller type

Summary

{property} bool *IsInNetwork*

{property} *ControllerDOF DOF* Controller's degrees of freedom, if *DeviceController* is null will find the first available *DeviceController*

{property} *MAGESControllerClass Get* Singleton Instance of MAGESControllerClass. public *ControllerTypes controllerType*

public bool *isServer*

public bool *isClient*

public GameObject *rightController*

public GameObject *rightHand*

public inline virtual GameObject *AttachToolRight*()

public inline virtual GameObject *AttachToolLeft*()

public inline virtual GameObject *AttachTool*(*MAGESHand* _hand)

public inline virtual *ControllerTypes GetControllerType*() User accessible function to get the current controller type

public inline virtual void *SetControllerType*(*ControllerTypes* _controllerTypes)

public inline void *SetIsGrabed*(bool value)

public inline float *GetControllerGrabStrength*(*MAGESHand* _hand) Function to get the controllers grab strength value.

public inline string *GetHandTag*(GameObject child) Returns the Tag of the current controllers hand gameobject

public inline void *ControllerHapticPulse*(*MAGESHand* _hand,float strength) Creates a haptic pulse on the controller

public inline bool *GetIsGrabed*(*MAGESHand* _hand) Returns true if the grip button is pressed

public inline void *SetBothControllersConnection*(bool _bothContrConnected)

public inline bool *GetTriggerPressed*(*MAGESHand* _hand) Get the status of trigger button

public inline *MAGESControllerButtons GetPressedButtons*() Sees which buttons are pressed

public inline bool *GetIsSpectator*()

protected inline *MAGESControllerClass*() MAGESControllerClass protected constructor

enum *ControllerTypes*

enum *MAGESHand*

enum *MAGESControllerButtons*

Members

{property} bool IsInNetwork

{property} ControllerDOF DOF

Controller's degrees of freedom, if *DeviceController* is null will find the first available *DeviceController*

{property} MAGESControllerClass Get

Singleton Instance of MAGESControllerClass.

Returns

Returns the main controller.

public ControllerTypes controllerType

public bool isServer

public bool isClient

public GameObject rightController

public GameObject rightHand

public inline virtual GameObject AttachToolRight()

public inline virtual GameObject AttachToolLeft()

public inline virtual GameObject AttachTool(MAGESHand _hand)

public inline virtual ControllerTypes GetControllerType()

User accessible function to get the current controller type

Returns

Returns the type of current controller (HTC-Vice/Oculus Rift)

```
public inline virtual void SetControllerType(ControllerTypes _controllerTypes)
```

```
public inline void SetIsGrabed(bool value)
```

```
public inline float GetControllerGrabStrength(MAGESHand _hand)
```

Function to get the controllers grab strength value.

Returns -2.0f if no controller is specified

Parameters

- `hand` Can be either left or right

Returns

```
public inline string GetHandTag(GameObject child)
```

Returns the Tag of the current controllers hand gameobject

Returns null if no controller is specified

Parameters

- `child` The child gameobject, usually the colliding object

Returns

```
public inline void ControllerHapticPulse(MAGESHand _hand,float strength)
```

Creats a haptic pulse on the controller

Parameters

- `hand` Left or right controller
- `strength` The pulse strength : [Range 0 -> 1]

public inline bool GetIsGrabed(MAGESHand _hand)

Returns true if the grip button is pressed

Parameters

- `hand` Hand to look if pressed

Returns

public inline void SetBothControllersConnection(bool _bothContrConnected)

public inline bool GetTriggerPressed(MAGESHand _hand)

Get the status of trigger button

Parameters

- `_hand` In which hand

Returns

Returns if trigger is pressed

public inline MAGESControllerButtons GetPressedButtons()

Sees which buttons are pressed

Returns

Returns a list of the presseed buttons

public inline bool GetIsSpectator()

protected inline MAGESControllerClass()

MAGESControllerClass protected constructor

Initializes the Singleton Instance!!!

enum ControllerTypes

HTCViveControllerOculusTouchControllerWindowsMixedRealityControllerNoController

enum MAGESHand

leftright

enum MAGESControllerButtons

TriggerButtonGripButtonMenuButtonABXYThumbStick

namespace MAGES::GameController::Network

Summary

class *MAGES::GameController::Network::MAGESNetworkmanager* Base class for network managment Some functions need to be overriden to add application specific functionalities

class MAGES::GameController::Network::MAGESNetworkmanager

Base class for network managment Some functions need to be overriden to add application specific functionalities

Summary

public Transform *spawnPosition*

public int *curPlayer*

public inline virtual void *OnStartServer*() Called when starting a network session as server

public inline virtual void *OnStartClient*() Called when starting a network session as client

public inline bool *GetIsInNetwork*()

public inline bool *GetIsServer*() Gets the status of the current application instance

public inline bool *GetIsClient*() Gets the status of the current application instance

public inline void *SetIsServer*(bool value) Sets the status of the current application instance

public inline void *SetIsClient*(bool value) Sets the status of the current application instance

public inline bool *GetIsSpectator*()

public inline void *SetIsSpectator*(bool _isSpectator)

Members

public Transform spawnPosition

public int curPlayer

public inline virtual void OnStartServer()

Called when starting a network session as server

public inline virtual void OnStartClient()

Called when starting a network session as client

Parameters

- client

public inline bool GetIsInNetwork()

Returns

True if the local machine is in a network session.

public inline bool GetIsServer()

Gets the status of the current application instance

Returns

Returns true if the application is the server

public inline bool GetIsClient()

Gets the status of the current application instance

Returns

Returns true if the application is client

public inline void SetIsServer(bool value)

Sets the status of the current application instance

public inline void SetIsClient(bool value)

Sets the status of the current application instance

public inline bool GetIsSpectator()

public inline void SetIsSpectator(bool _isSpectator)

8.1.3 Scenegraph

Summary

namespace *MAGES::sceneGraphSpace*

namespace MAGES::sceneGraphSpace

Summary

enum *Difficulty* The selected difficulty of *Operation*

enum *ActionType*The Action type to calculate score for analytics

class *MAGES::sceneGraphSpace::ActionNodeProperties*

class *MAGES::sceneGraphSpace::Operation* Singleton Class *Operation* is the game object (Node) that has all the Lesson as children. This Node will be the root of scenegraph after Initialization

class *MAGES::sceneGraphSpace::ScenegraphTraverse* Class with helper functions to traverse through the *Operation*'s Nodes

Members

enum Difficulty

EasyMediumHard

The selected difficulty of *Operation*

enum ActionType

SimpleBasicCrucial

The Action type to calculate score for analytics

class MAGES::sceneGraphSpace::ActionNodeProperties

Summary

public int *lessonID*

public int *stageID*

public int *actionID*

public string *actionName*

public inline *ActionNodeProperties*(int _lessonID,int _stageID,int _actionID, string _actionName)

Members

public int lessonID

public int stageID

public int actionID

public string actionName

public inline ActionNodeProperties(int _lessonID,int _stageID,int _actionID, string _actionName)

class MAGES::sceneGraphSpace::Operation

```
class MAGES::sceneGraphSpace::Operation
  : public MonoBehaviour
```

Singleton Class *Operation* is the game object (Node) that has all the Lesson as children. This Node will be the root of scenegraph after Initialization

Summary

{property} *Operation Get*

public Action *onPerformAllActions*

public inline void *LoadLSA*()

public inline void *AddActionOnPerform*(Action setAction) Set custom invokes to run on perform

public inline void *AddActionOnUndo*(Action setAction) Set custom invokes to run on undo

public inline void *AddActionAfterUndo*(Action setAction) Set custom invokes to run on undo

public inline void *AddActionAfterInitialize*(Action setAction) Set custom invokes to run after initialize

public inline void *RemoveActionOnUndo*(Action removeActon) Remove an action from Undo

public inline void *RemoveActionOnPerform*(Action removeActon) Remove an action from Perform

public inline void *AddActionOnStagePerform*(Action setAction) Set custom invokes to run on the perform of a Stage

public inline void *RemoveActionAfterInitialize*(Action removeActon) Remove an action from [after initialize list]

public inline List< Action > *GetOnStagePerformInvokesList*()

public inline void *AddActionOnStageUndo*(Action setAction) Set custom invokes to run on the perform of a Stage

public inline List< Action > *GetOnStageUndoInvokesList*()

public inline bool *Perform*() This is the main function for Traversing the graph Perform is being called from the *Operation* node and moves through the graph to find the current Action that needs to be performed. Then recursively travels back the root

public inline bool *Undo*() Undo goes to the previous Action in the graph The methodology is the same as Perform

public inline int *GetLessonID*() Returns the current Lesson ID

public inline int *GetStageID*() Returns the current Stage ID

public inline int *GetActionID*() Returns the current Action ID

public inline bool *GetSkipAction*() Returns if current mode is jump lesson

public inline GameObject *GetOperationNode*() Returns the gameObject the operation is assigned to

public inline string *GetOperationName*() Returns the name of operation

public inline int *GetNumberOfLessons*() Returns the total number of Lessons

public inline bool *GetHologramOption*() Returns the current holographic state

public inline *Difficulty GetOperationDifficulty*() Returns the operations difficulty

public inline *ActionType GetCurrentActionType*() Returns the current Action's Type for analytics

public inline float *GetCurrentAverageActionTime*() Returns the current Action's average time for analytics

`public inline float GetDemoActionSkipTimer()` IEnumeration timer for demo actions that will be skipped

`public inline void SetOperationName(string opName)` Set the operation's name

`public inline void SetOperationDifficulty(Difficulty difficulty)` Sets the operations difficulty

`public inline void SetHolograms(bool option)` Sets the holographic option

`public inline void SetJumpingLessons(bool _setJump)` Don't Use outside of scenegraph!

`public inline void SetDemoActionSkipTimer(float _timer)` IEnumeration timer for demo actions that will be skipped

`public inline bool PerformByServer()` Sets the perform by server

`public inline bool UndoByServer()` Sets the Undo by server

`public inline void SkipCurrentAction(bool _isManuallyCalled)` Skips current Action for Analytics

`public inline void SetNextActionForInitialization(int lessonID,int stageID,int actionID)`

`public inline GameObject GetCurrentLesson()` Returns the Current Lesson of this active operation

`public inline GameObject GetCurrentStage()` Returns the gameObject of current Stage on the scenegraph

`public inline GameObject GetCurrentAction()` Returns the gameObject of current Action on the scenegraph

Members

`{property} Operation Get`

`public Action onPerformAllActions`

`public inline void LoadLSA()`

`public inline void AddActionOnPerform(Action setAction)`

Set custom invokes to run on perform

`public inline void AddActionOnUndo(Action setAction)`

Set custom invokes to run on undo

```
public inline void AddActionAfterUndo(Action setAction)
```

Set custom invokes to run on undo

```
public inline void AddActionAfterInitialize(Action setAction)
```

Set custom invokes to run after initialize

```
public inline void RemoveActionOnUndo(Action removeActon)
```

Remove an action from Undo

```
public inline void RemoveActionOnPerform(Action removeActon)
```

Remove an action from Perform

```
public inline void AddActionOnStagePerform(Action setAction)
```

Set custom invokes to run on the perform of a Stage

```
public inline void RemoveActionAfterInitialize(Action removeActon)
```

Remove an action from [after initialize list]

```
public inline List< Action > GetOnStagePerformInvokesList()
```

```
public inline void AddActionOnStageUndo(Action setAction)
```

Set custom invokes to run on the perform of a Stage

```
public inline List< Action > GetOnStageUndoInvokesList()
```

```
public inline bool Perform()
```

This is the main function for Traversing the graph Perform is being called from the *Operation* node and moves through the graph to find the current Action that needs to be performed. Then recursively travels back the root

After Performing an action we also have to Initialize the properties of the next one to be ready for the next

Returns

True if there is a next lesson OR False if this is the last one

public inline bool Undo()

Undo goes to the previous Action in the graph The methodology is the same as Perform

After Undo we have to set the current Actions properties to Perform this one again

Returns

True if there is a next lesson OR False if this is the last one

public inline int GetLessonID()

Returns the current Lesson ID

public inline int GetStageID()

Returns the current Stage ID

public inline int GetActionID()

Returns the current Action ID

public inline bool GetSkipAction()

Returns if current mode is jump lesson

public inline GameObject GetOperationNode()

Returns the gameObject the operation is assigned to

public inline string GetOperationName()

Returns the name of operation

public inline int GetNumberOfLessons()

Returns the total number of Lessons

public inline bool GetHologramOption()

Returns the current holographic state

public inline Difficulty GetOperationDifficulty()

Returns the operations difficulty

public inline ActionType GetCurrentActionType()

Returns the current Action's Type for analytics

public inline float GetCurrentAverageActionTime()

Returns the current Action's average time for analytics

public inline float GetDemoActionSkipTimer()

IEnumeration timer for demo actions that will be skipped

public inline void SetOperationName(string opName)

Set the operation's name

public inline void SetOperationDifficulty(Difficulty difficulty)

Sets the operations difficulty

Parameters

- `difficulty` Selected Dificulty

public inline void SetHolograms(bool option)

Sets the holographic option

Parameters

- `option` True to enable holograms and False to disable

public inline void SetJumpingLessons(bool _setJump)

Dont Use outside of scenegraph!

public inline void SetDemoActionSkipTimer(float _timer)

IEnumeration timer for demo actions that will be skipped

Parameters

- `_timer` 0.1f to 10f seconds

public inline bool PerformByServer()

Sets the perform by server

public inline bool UndoByServer()

Sets the Undo by server

public inline void SkipCurrentAction(bool _isManuallyCalled)

Skips current Action for Analytics

Parameters

- `_isManuallyCalled` Set to True if called manually from user input, e.g. button press

public inline void SetNextActionForInitialization(int lessonID,int stageID,int actionID)

public inline GameObject GetCurrentLesson()

Returns the Current Lesson of this active operation

public inline GameObject GetCurrentStage()

Returns the gameObject of current Stage on the scenegraph

public inline GameObject GetCurrentAction()

Returns the gameObject of current Action on the scenegraph

class MAGES::sceneGraphSpace::ScenegraphTraverse

Class with helper functions to traverse through the *Operation*'s Nodes

8.1.4 Utilities

Summary

namespace *MAGES::Utilities*

namespace *MAGES::Utilities::Keyboard*

class *AdjustInteractableCircle*

class *MAGESParenting* Unity's parenting is very usefull but in many cases impractical (e.g. Scaling issues). This script has nothing to do with Unity's parenting. it just translates and rotates an object according to another gameobject's transform. class *RequestAuthorityForChildren*

namespace MAGES::Utilities

Summary

class *MAGES::Utilities::DestroyUtilities* Used for networking class *MAGES::Utilities::RenderModeChager* Changes Standard shader rendering mode property class *MAGES::Utilities::RequestAuthority* Used for networking.

class MAGES::Utilities::DestroyUtilities

Used for networking

Only the server can destroy network objects. When a network object is deleted on the server, it is deleted on all clients.

class MAGES::Utilities::RenderModeChager

Changes Standard shader rendering mode property

Rendering Modes: Opaque, Cutout, Fade, Transparent

Summary

`enum` *`BlendMode`* Available render modes for standard shader

Members

enum BlendMode

OpaqueCutoutFadeTransparent

Available render modes for standard shader

class MAGES::Utilities::RequestAuthority

```
class MAGES::Utilities::RequestAuthority
  : public MonoBehaviour
```

Used for networking.

Player must have the authority to move an network object. Objects MUST have local player authority enabled!!!

On Client: This asks the server for permission to move an object. The server then marks that object as Kinematic the this client disables Kinematic and simulate physics for the object

On Server: The server accepts the authority request from a client and grants him permission for that objects manupulation. Marks the object Kinematic for himself. When the server grabs an object it sends a message to the client and starts simulating physics. The client enables Kinematic.

Summary

`public bool` *`disableKinematicUpdate`*

`public inline bool` *`GetSpawnedAsKinematic`*`()`

`public inline void` *`SetSpawnedAsKinematic`*`(bool spawnedKinematic)`

`public inline bool` *`GetEnableKinematicOnDetatch`*`()`

`public inline void` *`SetEnableKinematicOnDetach`*`(bool enableKinDetach)`

Members

public bool disableKinematicUpdate

public inline bool GetSpawnedAsKinematic()

public inline void SetSpawnedAsKinematic(bool spawnedKinematic)

public inline bool GetEnableKinematicOnDetatch()

public inline void SetEnableKinematicOnDetach(bool enableKinDetach)

namespace MAGES::Utilities::Keyboard

Summary

class *MAGES::Utilities::Keyboard::KeyboardController*

class MAGES::Utilities::Keyboard::KeyboardController

```
class MAGES::Utilities::Keyboard::KeyboardController
  : public MonoBehaviour
```

Summary

{property} *KeyboardController Get*

Members

{property} KeyboardController Get

class AdjustInteractableCircle

```
class AdjustInteractableCircle
  : public MonoBehaviour
```

Summary

public bool *UseCustomDropDistance*

public string *TargetTransformName*

public Transform *OvverideTargetTransform*

public Transform *OvverideControllerTransform*

public inline IEnumerator *Start*()

public inline void *RetargetTransform*(Transform _retarget)

Members

public bool UseCustomDropDistance

public string TargetTransformName

public Transform OvverideTargetTransform

public Transform OvverideControllerTransform

public inline IEnumerator Start()

public inline void RetargetTransform(Transform _retarget)

class MAGESParenting

```
class MAGESParenting
  : public MonoBehaviour
```

Unity's parenting is very useful but in many cases impractical (e.g. Scaling issues). This script has nothing to do with Unity's parenting. it just translates and rotates an object according to another gameobject's transform.

Basically regarding Translation and Rotation it works like Unity's parenting, without the need for the gameobject to be attached to another one.

NOTICE after the game starts and the gameObject's MAGESparenting is in use, it's scale SHOULD NOT be changed to avoid weird results in transformation

Summary

{property} bool *StopParenting*

public Transform *parentTransform*

Members

{property} bool StopParenting

public Transform parentTransform

class RequestAuthorityForChildren

```
class RequestAuthorityForChildren
  : public MonoBehaviour
```

Summary

public bool *disableKinematicUpdate*

Members

public bool disableKinematicUpdate

8.1.5 Utilities/Camera

Summary

namespace *MAGES::SecondVieport*

class *FPSDisplay*

namespace MAGES::SecondVieport

Summary

enum *CameraFixedPositions*

enum *XrayMonitor*

class *MAGES::SecondVieport::SecondVieportCameraConfiguration* Internal Use mostly, XRAY effect for secondary cameras applied to given rendered textures

Members

enum CameraFixedPositions

frontFacingRightLowSideLeftMediumSideTopFemurFacing

enum XrayMonitor

rightleft

class MAGES::SecondVieport::SecondVieportCameraConfiguration

```
class MAGES::SecondVieport::SecondVieportCameraConfiguration
  : public MonoBehaviour
```

Internal Use mostly, XRAY effect for secondary cameras applied to given rendered textures

Summary

{property} *SecondVieportCameraConfiguration Cam_inst*

Members

{property} SecondVieportCameraConfiguration Cam_inst

class FPSDisplay

```
class FPSDisplay
  : public MonoBehaviour
```

Summary

{property} *FPSDisplay Instance*

public bool *showFps*

public inline void *Update*()

public inline void *OnGUI*()

Members

{property} FPSDisplay Instance

public bool showFps

public inline void Update()

public inline void OnGUI()

8.1.6 Utilities/Constructors

Summary

namespace *MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor*

struct *MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::InteractableWithParent*

namespace MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor

Summary

enum *PrefabType*

enum *PrefabActionOnPerform*

enum *PrefabAvailableLayers*

enum *PrefabMaterialType*

enum *PrefabInteractableType*

enum *OnPrefabDetachFeature*

enum *PumpMode*

class *MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::AnimationMovePrefabCons*

class *MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::CollisionHitPrefabConst*
This script should be attached to gameObjects that their purpose is to be hit into place. Most common example is the gameObject being a pin ready to be hit by a hammer.

class *MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::DestroyTime*
Most prefabs spawned for a specific action are destroyed but the time the action is completed. If a prefab needs to be destroyed in another action, it can be achieved using this class to select the action where it's going to be destroyed.

class *MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::GenericPrefabConstructo*
Almost all prefabs used in a scene have similar properties. Many of these properties are grouped into different types of PrefabConstructor scripts that all inherit this one the *GenericPrefabConstructor*. These scripts initialize the prefabs with the values provided in the inspector by the script itself. They also check if the prefabs contain all the essential components and it automatically adds them if anything is missing otherwise it produces a runtime error explaining what is wrong with the prefab.

class *MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::InteractableFinalPlacer*
This script is for the prefabs that are placed to the final position, and wait for the duplicated interactable prefabs to be inserted by the user in their own position

class *MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::InteractablePrefabConst*
This script is for any prefab that can be grabbed from the User. It has physics properties and when spawned it is stored to the PrefabSpawnManager and it's observed regularly. In case of a bug (mostly due to physics) it will be respawned with it's starting transform.

class *MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::InteractableWithParenti*
This script inherits from *InteractablePrefabConstructor* and works in exactly the same way Only difference is that the gameObject this script is attach to, it is a child of another gameObject that does NOT have interactable attributes

class *MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::NonTriggerColliderPrefa*
This script should be attached to prefabs with only non triggered colliders that can be (physically only) interacted with the user (e.g. pushing)

class *MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::PumpPrefabConstrutor*

class *MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::QuestionPrefabConstruct*

class *MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::RemoveWithToolsCostruct*
This script is for any prefab that needs to be interacted by the user with the use of another Tool. This script as a component by itself will have a dropdown list for the developer to choose the tools that can grab the gameObject that this script is attached to.

class *MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::ToolAndTime*
Keeps for each tool how much time it needs to interact with each collider

class *MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::ToolColliderPrefabConst*
This script should be attach to a gameObject which is just a grouping parent of many children where each one is a separate collider. This gameObject will interact with any tool given with the dropdown list. By interacting we mean that when the tool collides with one of the colliders, it destroys the collider after the time given. if all the colliders-children of the gameObject are destroyed, this script will trigger the EventManager as the gameObject's function is considered complete.

class *MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::UseColliderPrefabConst*
Generally the purpose of this script is that the collider attached to the gameObject containing this script expects a collision OLNY with all the prefabs that are dragged n dropped in this component's dropdown list. If it detects that the collision comes from one of the desired prefabs, and the timer during this collision passed the minimum amount of time given (variable stayTime), this gameObject's function is considered complete.

Members

enum PrefabType

GenericInteractableInteractableWithParentInteractableFinalPlacementToolActionColliderUseActionColliderNonTriggerColliderCollisi

enum PrefabActionOnPerform

DestroyRemain

enum PrefabAvailableLayers

DefaultGrabbablePrefabsTriggerColliderLessonNoTriggerColliderLessonAvatarLayerignoreAll

enum PrefabMaterialType

MetalPlasticCloth

enum PrefabInteractableType

GenericInsertRemove

enum OnPrefabDetachFeature

ReInitializeDestroyEventTriggerCurrLSAOnDestroyNothing

enum PumpMode

FullPumpHalfPump

class MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::AnimationMovePrefabConst

```
class␣
↪MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::AnimationMovePrefabConstructor
  : public MAGES.Utilities.prefabSpawnManager.prefabSpawnConstructor.
↪GenericPrefabConstructor
```

Summary

public float *stayTime*

public float *targetPercentage*

public inline virtual override void *FinalizePrefabAction*() when action is complete it calls this function on the prefab to finalise it's behavior

public inline virtual override void *FinalizeByNetwork*() Internal function for the multiplayer, Ignore

public inline void *SetOnStayTimeAction*(Action< float > _action) Set a callback function for the collision This function will be called every fixed frame during the collision

protected inline virtual override void *Start*()

Members

public float stayTime

public float targetPercentage

public inline virtual override void FinalizePrefabAction()

when action is complete it calls theis function on the prefab to finalise it's behavior

public inline virtual override void FinalizeByNetwork()

Internal function for the multiplayer, Ignore

public inline void SetOnStayTimeAction(Action< float > _action)

Set a callback function for the collision This function will be called every fixed frame during the collision

Parameters

- `_action` set function with float parameter

protected inline virtual override void Start()

class MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::CollisionHitPrefabConstr

```
class␣
↪MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::CollisionHitPrefabConstructor
  : public MAGES.Utilities.prefabSpawnManager.prefabSpawnConstructor.
↪GenericPrefabConstructor
```

This script should be attached to gameObjects that their purpose is to be hit into place. Most common example is the gameObject being a pin ready to be hit by a hammer.

The gameObject's set up is very specific. It must be placed to it's final position (where it should be after the hits). Then select in what direction it should go after each hit and finally insert the hit count and the step size per hit. On start the gameObject will translate in the opposide direction of the hit direction selected from the developer. The translated distance is: hitStep * hitCount

The gameObject upon impact with the tool will measure the tool's velocity. If the velocity is smaller than the value given (minMagnitude) it will ignore the collision.

NOTICE both gameObject and tool need to have Rigidbodies to measure velocity as it is a Physics variable.

Summary

`public ToolsEnum` *`tool`*

`public float` *`hitStep`*

`public int` *`hitCount`*

`public float` *`minMagnitude`*

`public float` *`collisionTimeDiff`*

`public inline virtual override void` *`FinalizePrefabAction`*`()` when action is complete it calls this function on the prefab to finalize it's behavior

`public inline virtual override void` *`FinalizeByNetwork`*`()` Called when a hit activated by remote user Called for each hit, also display percentage for all other users

`protected` *`VectorSelection vectorDirection`*

`protected inline virtual override void` *`Start`*`()`

`protected inline virtual override void` *`OnTriggerEnter`*`(Collider other)`

Members

public ToolsEnum tool

public float hitStep

public int hitCount

public float minMagnitude

public float collisionTimeDiff

public inline virtual override void FinalizePrefabAction()

when action is complete it calls theis function on the prefab to finalise it's behavior

public inline virtual override void FinalizeByNetwork()

Called when a hit activated by remote user Called for each hit, also display persentage for all other users

protected VectorSelection vectorDirection

protected inline virtual override void Start()

protected inline virtual override void OnTriggerEnter(Collider other)

class MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::DestroyTime

Most prefabs spawned for a specific action are destroyed but the time the action is completed. If a prefab needs to be destroyed in another action, it can be achieved using this class to select the action where it's going to be destoryed.

Summary

public int *Lesson*

Members

public int Lesson

class MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::GenericPrefabConstructor

```
class␣
↪MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::GenericPrefabConstructor
  : public MonoBehaviour
```

Almost all prefabs used in a scene have similar properties. Many of these properties are grouped into different types of PrefabConstructor scripts that all inherit this one the *GenericPrefabConstructor*. These scripts initialize the prefabs with the values provided in the inspector by the script itself. They also check if the prefabs contain all the essential components and it automatically adds them if anything is missing otherwise it produces a runtime error explaining what is wrong with the prefab.

NOTICE This perticular script should not be used by itself on any prefab as it just contains all the common functions. Only the inherited members should be attached.

PrefabMaterialType is used for different sounds. Being still under development it will not -in most cases- work

REQUIREMENTS The Unity Project MUST have at least the layers below in order for the Prefab Constructors to work* Default

- GrabbablePrefabs
- TriggerColliderLesson
- NoTriggerColliderLesson

This is important in order to minimize the observed collision between layers. This can be done by changing the Layer Collision Matrix in Edit->Project Settings->Physics

Summary

`{property} string EventManagerTriggerKey`

`public List< DestroyTime>destroyTime` Selected the actions where the prefab can be deleted. Leave empty if the prefab is goind to be destroyed by the same action that spawned it

`public bool enableKinematicOnReset` If true forces kinematic on rigidbody on reset.

`public PrefabType prefabType`

`public PrefabActionOnPerform prefabPerformAction`

`public Action onPerformDelete`

`public inline void SetOnPerformAction(Action call)`

`public inline virtual void ResetPrefab()` Resets the prefab to the position and rotation it had when spawned

`public inline virtual void [SetNewPrefabStartingTransform]` (#classo-vid_v_r_1_1_utilities_1_1prefab_spawn_manager_1_1prefab_spawn_constructor_1_1_generic_prefab_constructor_1a9ace2d16ef224f
Set new starting position and rotation for the prefab if the ones when spawned are not desired it applies the position and rotation at the time this function is called

`public inline virtual void SetNewPrefabCustomStartingTransform(Vector3 ResetPosition,Quaternion ResetRotation)` THIS FUNCTION MUST BE CALLED AFTER INITIALIZE OR IT WILL BE IGNORED

`public inline virtual void FinalizePrefabAction()` when action is complete it calls this function on the prefab to finalize it's behavior

`public inline virtual void FinalizeByNetwork()` Internal function for the multiplayer, Ignore

`public inline virtual void FindEventKey()` Set the prefab's EventManagerTriggerKey internally with teh correct key (action's name)

`protected string eventTriggerKey`

`protected PrefabAvailableLayers differentLayer`

protected *PrefabAvailableLayers parentDifferentLayer*

protected GameObject[] *childrenDifferentLayer*

protected List< Transform > *allChildTransforms*

protected Rigidbody *addedRigidBody*

protected Vector3 *startPosition*

protected Quaternion *startRotation*

protected AudioSource *prefabAudio*

protected bool *hideDisplayPercentage*

protected Action *onPerformFunction* This method will be set to Invoke after completing a specific pump prefab object

protected inline virtual void *Start*()

protected inline void *SetUpLayer*(int _layer)

protected inline void *SetUpRigidBody*(bool _kinematic) Sets the prefab's attributes according to the Rigidbody component

protected inline void *SetUpAudioSource*(string _soundName)

protected inline void *SetUpPrefabSpawnNotifier*()

protected inline virtual void *OnTriggerEnter*(Collider other)

protected inline virtual void *OnDestroy*()

Members

{property} string EventManagerTriggerKey

public List< DestroyTime>destroyTime

Selected the actions where the prefab can be deleted. Leave empty if the prefab is goind to be destroyed by the same action that spawned it

public bool enableKinematicOnReset

If true forces kinematic on rigidbody on reset.

public PrefabType prefabType

public PrefabActionOnPerform prefabPerformAction

public Action onPerformDelete

public inline void SetOnPerformAction(Action call)

public inline virtual void ResetPrefab()

Resets the prefab to the position and rotation it had when spawned

public inline virtual void SetNewPrefabStartingTransform()

Set new starting position and rotation for the prefab if the ones when spawned are not desired it applies the position and rotation at the time this function is called

public inline virtual void SetNewPrefabCustomStartingTransform(Vector3 ResetPosition,Quaternion ResetRotation)

THIS FUNCTION MUST BE CALLED AFTER INITIALIZE OR IT WILL BE IGNORED

Set new starting position and rotation for the prefab if the ones when spawned are not desired it applies the position and rotation at the time this function is called

Parameters

- `ResetPosition` New reset postion
- `ResetRotation` New reset rotation

public inline virtual void FinalizePrefabAction()

when action is complete it calls theis function on the prefab to finalise it's behavior

public inline virtual void FinalizeByNetwork()

Internal function for the multiplayer, Ignore

public inline virtual void FindEventKey()

Set the prefab's EventManagerTriggerKey internally with teh correct key (action's name)

protected string eventTriggerKey

protected PrefabAvailableLayers differentLayer

protected PrefabAvailableLayers parentDifferentLayer

protected GameObject[] childrenDifferentLayer

protected List< Transform > allChildTransforms

protected Rigidbody addedRigidBody

protected Vector3 startPosition

protected Quaternion startRotation

protected AudioSource prefabAudio

protected bool hideDisplayPercentage

protected Action onPerformFunction

This method will be set to Invoke after completing a specific pump prefab object

protected inline virtual void Start()

protected inline void SetUpLayer(int _layer)

protected inline void SetUpRigidBody(bool _kinematic)

Sets the prefab's attributes according to the Rigidbody component

The Rigidboby Component Initializes The Prefab Constructor

Parameters

- _kinematic

protected inline void SetUpAudioSource(string _soundName)

protected inline void SetUpPrefabSpawnNotifier()

protected inline virtual void OnTriggerEnter(Collider other)

protected inline virtual void OnDestroy()

class MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::InteractableFinalPlaceme

```
class␣
↪MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::InteractableFinalPlacementPrefabConst
  : public MAGES.Utilities.prefabSpawnManager.prefabSpawnConstructor.
↪GenericPrefabConstructor
```

This script is for the prefabs that are placed to the final position, and wait for the duplicated interactable prefabs to be inserted by the user in their own position

REQUIREMENTS PrefabLerpPlacement

Summary

`public inline virtual override void FinalizePrefabAction()` when action is complete it calls this function on the prefab to finalize it's behavior

`public inline virtual override void FinalizeByNetwork()` Internal function for the multiplayer, Ignore

`public inline virtual override void FindEventKey()` Set the prefab's EventManagerTriggerKey internally with teh correct key (action's name)

`protected inline virtual override void Start()`

Members

public inline virtual override void FinalizePrefabAction()

when action is complete it calls theis function on the prefab to finalise it's behavior

public inline virtual override void FinalizeByNetwork()

Internal function for the multiplayer, Ignore

public inline virtual override void FindEventKey()

Set the prefab's EventManagerTriggerKey internally with teh correct key (action's name)

protected inline virtual override void Start()

class MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::InteractablePrefabConstr

```
class␣
↪MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::InteractablePrefabConstructor
  : public MAGES.Utilities.prefabSpawnManager.prefabSpawnConstructor.
↪GenericPrefabConstructor
```

This script is for any prefab that can be grabbed from the User. It has physics properties and when spawned it is stored to the PrefabSpawnManager and it's observed regularly. In case of a bug (mostly due to physics) it will be respawned with it's strarting transform.

NOTICE ALL prefabs with this script HAVE enabled gravity! if rigidbody is selected to be kinematic then the prefab will be frozen to it's spawned position as a kinematic until it is grabbed. Upon grabbing the rigidbody will disable it's kinematic property and re-enable gravity. This is useful for prefabs that spawn into a weird position that in theory should not be affected by gravity.

REQUIREMENTS* Rididbody

- Colliders
- MAGESInteractableItem

Summary

public *PrefabInteractableType prefabInteractableType*

public *OnPrefabDetachFeature prefabDetachFeature*

public bool *allowPrefabManualReset*

public inline virtual override void *ResetPrefab*() Resets the prefab to the position and rotation it had when spawned

public inline virtual override void *FinalizeByNetwork*() Internal function for the multiplayer, Ignore

protected float *minDistanceReset*

protected MAGESInteractable *prefabInteract*

protected string *audioMaterial*

protected bool *prefabDropped*

protected bool *isRightHand*

protected inline virtual override void *Start*()

protected inline virtual IEnumerator *SetUpInteractableItemListeners*()

protected inline void *PrefabPicked*()

protected inline void *PrefabDropped*()

protected inline virtual override void *OnTriggerEnter*(Collider other)

Members

public PrefabInteractableType prefabInteractableType

public OnPrefabDetachFeature prefabDetachFeature

public bool allowPrefabManualReset

public inline virtual override void ResetPrefab()

Resets the prefab to the position and rotation it had when spawned

public inline virtual override void FinalizeByNetwork()

Internal function for the multiplayer, Ignore

protected float minDistanceReset

protected MAGESInteractable prefabInteract

protected string audioMaterial

protected bool prefabDropped

protected bool isRightHand

protected inline virtual override void Start()

protected inline virtual IEnumerator SetUpInteractableItemListeners()

protected inline void PrefabPicked()

protected inline void PrefabDropped()

protected inline virtual override void OnTriggerEnter(Collider other)

class MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::InteractableWithParentPr

```
class␣
↪MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::InteractableWithParentPrefabConstruct
  : public MAGES.Utilities.prefabSpawnManager.prefabSpawnConstructor.
↪InteractablePrefabConstructor
```

This script inherits from *InteractablePrefabConstructor* and works in exactly the same way Only difference is that the gameObject this script is attach to, it is a child of another gameObject that does NOT have interactable attributes

e.g. for a simple tool in the scene that has no parent we attach *InteractablePrefabConstructor*. But for a lever that is interactable and it is a child of a non interactable wall (wall as a gameObject as well) we attach the InteractableWithParentPrefabConstructorto the lever to be able to be used by the user.

NOTICE Unity's parenting does not work well with physics behavior and interaction attached to gameObjects. This script is still under development

Summary

`public inline virtual override void ResetPrefab()` Resets the prefab to the position and rotation it had when spawned

`public inline virtual override void FinalizeByNetwork()` Internal function for the multiplayer, Ignore

`protected inline virtual override void Start()`

`protected inline virtual override IEnumerator [SetUpInteractableItemListeners(#classovid_v_r_1_1`

`protected inline virtual override void OnTriggerEnter(Collider other)`

Members

public inline virtual override void ResetPrefab()

Resets the prefab to the position and rotation it had when spawned

public inline virtual override void FinalizeByNetwork()

Internal function for the multiplayer, Ignore

protected inline virtual override void Start()

protected inline virtual override IEnumerator SetUpInteractableItemListeners()

protected inline virtual override void OnTriggerEnter(Collider other)

class MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::NonTriggerColliderPrefab

```
class␣
↪MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::NonTriggerColliderPrefabConstructor
  : public MAGES.Utilities.prefabSpawnManager.prefabSpawnConstructor.
↪GenericPrefabConstructor
```

This script should be attached to prefabs with only non triggered colliders that can be (physically only) interacted with the user (e.g. pushing)

Summary

protected inline virtual override void *Start*()

Members

protected inline virtual override void Start()

class MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::PumpPrefabConstrutor

```
class␣
↪MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::PumpPrefabConstrutor
  : public MAGES.Utilities.prefabSpawnManager.prefabSpawnConstructor.
↪GenericPrefabConstructor
```

Summary

public int *pumpsToPerform*

public *PumpMode pumpMode*

public bool *pauseAnimationOnPerform*

public float *vibration*

public bool *reverseAnimation*

public bool *playSound*

public bool *continious*

public bool *pressTrigger*

public bool *enableRangeMode*

public float *lowEndToPerform*

public float [highEndToPerform(#classovid_v_r_1_1_utilities_1_1prefab_spawn_manager_1_1prefab_spawn_constructor_

public float *stayTime*

public AnimationClip *rightHandAnimation*

public AnimationClip *leftHandAnimation*

public string *gameObjectName*

public string *externalStateName*

public bool *interacting*

public bool *waitForAllPumpPrefabs*

public inline void *PlayPumpSound*()

public inline void *SetTotalPumps*(int pumpsID)

public inline void *SetInteracting*(bool value)

public inline string *GetAttachedHand*()

public inline void *SetRestPumpInteractables*(List< GameObject > pumpGoList) Sets references to all the rest pump gameobjects of the current action.

public inline virtual override void *FinalizePrefabAction*() Finalizes the Action by removing unnecessary components or destroying the gameObject based on the PrefabActionOnPerform option

public inline float *GetPumpProgressValue*()

public inline virtual override void *FinalizeByNetwork*() Internal function for the multiplayer, Ignore

protected inline virtual override void *Start*()

Members

```
public int pumpsToPerform

public PumpMode pumpMode

public bool pauseAnimationOnPerform

public float vibration

public bool reverseAnimation

public bool playSound

public bool continious

public bool pressTrigger

public bool enableRangeMode

public float lowEndToPerform

public float highEndToPerform

public float stayTime

public AnimationClip rightHandAnimation

public AnimationClip leftHandAnimation

public string gameObjectName

public string externalStateName

public bool interacting

public bool waitForAllPumpPrefabs

public inline void PlayPumpSound()

public inline void SetTotalPumps(int pumpsID)

public inline void SetInteracting(bool value)

public inline string GetAttachedHand()
```

public inline void SetRestPumpInteractables(List< GameObject > pumpGoList)

Sets references to all the rest pump gameobjects of the current action.

public inline virtual override void FinalizePrefabAction()

Finalizest the Action by removing unesessary components or destroying the gameObject based on the PrefabActionOnPerform option

public inline float GetPumpProgressValue()

public inline virtual override void FinalizeByNetwork()

Internal function for the multiplayer, Ignore

protected inline virtual override void Start()

class MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::QuestionPrefabConstructo

```
class␣
↪MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::QuestionPrefabConstructor
  : public MAGES.Utilities.prefabSpawnManager.prefabSpawnConstructor.
↪GenericPrefabConstructor
```

Summary

public string *setChildEventByNetwork*

public inline virtual override void *FinalizePrefabAction*() when action is complete it calls theis function on the prefab to finalise it's behavior

public inline virtual override void *FinalizeByNetwork*() Internal function for the multiplayer, Ignore

public inline int *GetNumOfWrongAnswers*()

protected inline virtual override void *Start*()

Members

public string setChildEventByNetwork

public inline virtual override void FinalizePrefabAction()

when action is complete it calls theis function on the prefab to finalise it's behavior

public inline virtual override void FinalizeByNetwork()

Internal function for the multiplayer, Ignore

public inline int GetNumOfWrongAnswers()

protected inline virtual override void Start()

class MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::RemoveWithToolsCostructo

```
class␣
↪MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::RemoveWithToolsCostructor
  : public MAGES.Utilities.prefabSpawnManager.prefabSpawnConstructor.
↪GenericPrefabConstructor
```

This script is for any prefab that needs to be interacted by the user with the use of another Tool. This script as a component by itself will have a dropdown list for the developer to choose the tools that can grab the gameObject that this script is attached to.

The list will contain the tools created inside Unity using MAGES/Create Tools DLL.

NOTICE This script currently ONLY works for prefabs that are already placed somewhere and have the need to be removed from there and be thrown away.

Summary

`public inline virtual override void` *`FinalizeByNetwork`*`()` Internal function for the multiplayer, Ignore

`public inline virtual override void` *`FinalizePrefabAction`*`()` when action is complete it calls theis function on the prefab to finalise it's behavior

`protected inline virtual override void` *`Start`*`()`

`protected inline virtual override void` *`OnTriggerEnter`*`(Collider other)` On Trigger Enter check if it is a Tool that can grab this prefab

`protected inline virtual override void` *`OnDestroy`*`()`

Members

public inline virtual override void FinalizeByNetwork()

Internal function for the multiplayer, Ignore

public inline virtual override void FinalizePrefabAction()

when action is complete it calls theis function on the prefab to finalise it's behavior

protected inline virtual override void Start()

protected inline virtual override void OnTriggerEnter(Collider other)

On Trigger Enter check if it is a Tool that can grab this prefab

Parameters

- other

protected inline virtual override void OnDestroy()

class MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::ToolAndTime

Keeps for each tool how much time it needs to itneract with each collider

Summary

public ToolsEnum *useTool* public float *timeToUse*

Members

public ToolsEnum useTool

public float timeToUse

class MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::ToolColliderPrefabConstr

```
class␣
↪MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::ToolColliderPrefabConstructor
  : public MAGES.Utilities.prefabSpawnManager.prefabSpawnConstructor.
↪GenericPrefabConstructor
```

This script should be attach to a gameObject which is just a grouping parent of many children where each one is a separate collider. This gameObject will interact with any tool given with the dropdown list. By interacting we mean that when the tool collides with one of the colldiers, it destroys the collider after the time given. if all the colliders-children of the gameObject are destoryed, this script will trigger the EventManager as the gameObject's function is considered complete.

When spawned, every 1st depth child is attached with the script ToolTriggerCollider

REQUIREMENTS* only 1st depth children

- all children must be identical (simple colliders)

Summary

{property} List< *ToolAndTime*>*ToolsList*

public int *numberOfColliders*

public string *setChildEventByNetwork*

public inline void *CollisionEvent*(GameObject colliderTriggered) For Internal Use between plug-ins DO NOT call manually

public inline void *SetOnStayTimeAction*(Action< float > _action) Set a callback function for the collision This function will be called every fixed frame during the collision

public inline virtual override void *FinalizeByNetwork*() Internal function for the multiplayer, Ignore

protected inline virtual override void *Start*()

Members

{property} List< ToolAndTime>ToolsList

public int numberOfColliders

public string setChildEventByNetwork

public inline void CollisionEvent(GameObject colliderTriggered)

For Internal Use between plug-ins DO NOT call manually

Parameters

- colliderTriggered

public inline void SetOnStayTimeAction(Action< float > _action)

Set a callback function for the collision This function will be called every fixed frame during the collision

Parameters

- _action set function with float parameter

public inline virtual override void FinalizeByNetwork()

Internal function for the multiplayer, Ignore

protected inline virtual override void Start()

class MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::UseColliderPrefabConstru

```
class␣
↪MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::UseColliderPrefabConstructor
  : public MAGES.Utilities.prefabSpawnManager.prefabSpawnConstructor.
↪GenericPrefabConstructor
```

Generally the purpose of this script is that the collider attached to the gameObject containing this script expects a collision OLNY with all the prefabs that are dragged n dropped in this component's dropdown list. If it detects that the collision comes from one of the desired prefabs, and the timer during this collision passed the minimum amount of time given (variable stayTime), this gameObject's function is considered complete.

Summary

public float *stayTime*

public inline virtual override void *FinalizeByNetwork*() Internal function for the multiplayer, Ignore

public inline void *SetOnStayTimeAction*(Action< float > _action) Set a callback function for the collision This function will be called every fixed frame during the collision

protected inline virtual override void *Start*()

Members

public float stayTime

public inline virtual override void FinalizeByNetwork()

Internal function for the multiplayer, Ignore

public inline void SetOnStayTimeAction(Action< float > _action)

Set a callback function for the collision This function will be called every fixed frame during the collision

Parameters

- `_action` set function with float parameter

protected inline virtual override void Start()

struct MAGES::Utilities::prefabSpawnManager::prefabSpawnConstructor::InteractableWithParentPr

Summary

public Transform *parentTransform*

public Rigidbody *parentRigidbody*

public MAGESInteractable *parentInteract*

public inline *InteractableParent*(Transform _t,Rigidbody _r,MAGESInteractable _i)

Members

public Transform parentTransform

public Rigidbody parentRigidbody

public MAGESInteractable parentInteract

public inline InteractableParent(Transform _t,Rigidbody _r,MAGESInteractable _i)

8.1.7 Utilities/Prefab Manager

Summary

namespace *MAGES::Utilities*

namespace *MAGES::Utilities::prefabSpawnManager*

namespace *MAGES::Utilities::prefabSpawnNotifier*

class *HologramPrefabBehavior*

class *PathAnimation*

class *PrefabLerpPlacement* When an interactable gameObject needs to be placed somewhere with a correct angle, it is difficult to measure if it's correctly placed because we do not have something to compare it to. A solution is to have the same gameObject duplicated and have this script attached to it.

class *MAGES::Utilities::prefabSpawnManager::PrefabSpawnManager::PrefabSpawnValues*

class *QuestionTriggerCollider*

class *ToolTriggerCollider* This script is attached to every child of the prefab that contains the ToolColliderPrefabConstructor Everytime each child is triggered it calls the parent when it's done via this script

namespace MAGES::Utilities

Summary

class *MAGES::Utilities::OnDestoyCallBackForFakePrefabs*

class *MAGES::Utilities::PrefabImporter* Prefab Importer is a helper class that loads and instantiates prefabs

class MAGES::Utilities::OnDestoyCallBackForFakePrefabs

```
class MAGES::Utilities::OnDestoyCallBackForFakePrefabs
  : public MonoBehaviour
```

Summary

public Action *onDestoy*

Members

public Action onDestoy

class MAGES::Utilities::PrefabImporter

```
class MAGES::Utilities::PrefabImporter
  : public MonoBehaviour
```

Prefab Importer is a helper class that loads and instantiates prefabs

namespace MAGES::Utilities::prefabSpawnManager

Summary

class *MAGES::Utilities::prefabSpawnManager::PrefabSpawnManager* Many gameobjects have physics properties (e.g. gravity). So there can be cases when the physics engine bugs and the object disappears or a user making a mistake, and have the object being thrown to an unreachable spot.

class MAGES::Utilities::prefabSpawnManager::PrefabSpawnManager

```
class MAGES::Utilities::prefabSpawnManager::PrefabSpawnManager
  : public MonoBehaviour
```

Many gameobjects have physics properties (e.g. gravity). So there can be cases when the physics engine bugs and the object disappears or a user making a mistake, and have the object being thrown to an unreachable spot.

This script purpose is to store each interactable -with physics properties- gameobject and observe its behavior. It constantly observes each gameObject per some seconds and resets it under specific conditions such as: i. not beng attached by anything (using the MAGESInteractableItem script) ii. being away from it's starting position by a given

offset. This offset can be set from the gameObject itself if it has attached the script InteractablePrefabCostructor OR InteractableWithParentPrefabConstructor.

PrefabSpawnManager is also responsible for preloading all prefabs on start of the Application

NOTICE Every function of this class can be used for debug purposes. However since evey function is called internally they SHOULD NOT be called from the developers.

This sigleton contains a unity function Update()

Summary

`{property}` *PrefabSpawnManager Pm_inst*

`public bool` *preLoadAssets*

Members

{property} PrefabSpawnManager Pm_inst

public bool preLoadAssets

class MAGES::Utilities::prefabSpawnManager::PrefabSpawnManager::PrefabSpawnValues

Summary

`public GameObject` *spawnedPrefab*

`public InteractablePrefabConstructor` *spawnedPrefabConstructor*

`public Coroutine` *spawnedPrefabCoroutine*

`public inline` *PrefabSpawnValues*`(GameObject _gameObject)`

Members

public GameObject spawnedPrefab

public InteractablePrefabConstructor spawnedPrefabConstructor

public Coroutine spawnedPrefabCoroutine

public inline PrefabSpawnValues(GameObject _gameObject)

namespace MAGES::Utilities::prefabSpawnNotifier

Summary

`class` *MAGES::Utilities::prefabSpawnNotifier::PrefabSpawnNotifier* Script is responsible for flashing the gameobject it's attached to. Helpful for the user to notice specific gameobject. When attached to one, the gameobject will automatically start flashing. If it contains colliders, then when the users hand (ONLY if it's tagged

"RightPalm" and/or "LeftPalm") is hovering above the gameObject it will flash faster with another to notify the user that it is in a grabbable range (if selected).

class MAGES::Utilities::prefabSpawnNotifier::PrefabSpawnNotifier

```
class MAGES::Utilities::prefabSpawnNotifier::PrefabSpawnNotifier
  : public MonoBehaviour
```

Script is responsible for flashing the gameobject it's attached to. Helpful for the user to notice specific gameobject. When attached to one, the gameobject will automatically start flashing. If it contains colliders, then when the users hand (ONLY if it's tagged "RightPalm" and/or "LeftPalm") is hovering above the gameObject it will flash faster with another to notify the user that it is in a grabbable range (if selected).

When grabbed from the user, OR the StopNotification Function is called, the gameObject stops flashing and the script gets destoryed.

The public values are only viable to change for the Inital Setup, before the script rins (e.g. setup on Editor).

NOTICE It may NOT work for all types of shaders.

Summary

`public Color` *`mainFlashColor`*

`public bool` *`allowDifferentColorOnHover`*

`public Color` *`hoverColor`*

`public float` *`mainFlashSpeedMul`*

`public float` *`hoverFlashSpeedMul`*

`public inline void` *`StopNotification`*`()` resets the material of the gameObject, stops the flashing and the script is auto-destoryed

Members

public Color mainFlashColor

public bool allowDifferentColorOnHover

public Color hoverColor

public float mainFlashSpeedMul

public float hoverFlashSpeedMul

public inline void StopNotification()

resets the material of the gameObject, stops the flashing and the script is auto-destoryed

class HologramPrefabBehavior

```
class HologramPrefabBehavior
  : public MonoBehaviour
```

class PathAnimation

```
class PathAnimation
  : public MonoBehaviour
```

Summary

public bool *conditionsAreMet*

public inline void *SetUpPathAnimation*(string animName,GameObject startTransform,GameObject endTransform,float targetProgress,int animationLayer,bool reverseAnimation,bool lateStart)

public inline void *Initialize*()

public inline void *FinalizeScript*()

public inline float *GetAnimationValue*()

public inline void *ManualStartAnimation*()

public inline void *RecalculateTotalDistance*()

Members

public bool conditionsAreMet

public inline void SetUpPathAnimation(string animName,GameObject startTransform,GameObject endTransform,float targetProgress,int animationLayer,bool reverseAnimation,bool lateStart)

public inline void Initialize()

public inline void FinalizeScript()

public inline float GetAnimationValue()

public inline void ManualStartAnimation()

public inline void RecalculateTotalDistance()

class PrefabLerpPlacement

```
class PrefabLerpPlacement
  : public MonoBehaviour
```

When an interactable gameObject needs to be placed somewhere with a correct angle, it is difficult to measure if it's correctly placed because we do not have something to compare it to. A solution is to have the same gameObject duplicated and have this script attached to it.

The gameObject with this script must be placed to its final -placed- position. When started, the renderers for this gameObject will be disabled. When the other gameObject (the similar one!) collides with this gameObect, it observes their angle difference. When this difference is below the offset given from the developer, this scirpt detaches the gameObect from anywhere that it is attached using the script 'MAGESInteractableItem' and adjust its to position and rotation to match the ones of this -final-gameObject. After the Lerping time (or adjustment time) is finished, the other gameObject gets destroyed, this gameObject's renderers are enabled and every other behavior provided to this gameObject is executed (e.g. an animation). In the end the script auto disables itself.

REQUIREMENTS:* Both must have rigidbodies and colliders

- The other gameobject must contain in it's parent the MAGESInteractableItem or MAGESInteractableRotator script.
- If this gameObject has an animation, it must be an Animation Component NOT an Animator and the animation iself must be Legacy Supported (see debug option)

Summary

`public float` *`maxAngleDegreeDiff`*

`public float` *`acceptedAngle`*

`public List< GameObject >` *`InteractablePrefabs`*

`public inline void` *`FinalizePrefabAction`*`()` If the action was skipped and the prefab that has attached this script isn't in the state where it should be if the user whould do the whole action correctly, call this function on the action's Perform function to turn this prefab into it's final state

`public inline void` *`FinalizePrefabActionByNetWork`*`()` When server or client finishes the action, the prefab goes to its final state ONLY when the game is in online mode!

Members

`public float maxAngleDegreeDiff`

`public float acceptedAngle`

`public List< GameObject > InteractablePrefabs`

`public inline void FinalizePrefabAction()`

If the action was skipped and the prefab that has attached this script isn't in the state where it should be if the user whould do the whole action correctly, call this function on the action's Perform function to turn this prefab into it's final state

public inline void FinalizePrefabActionByNetWork()

When server or client finishes the action, the prefab goes to its final state ONLY when the game is in online mode!

NOTICE This function should not be called manually. it's internal function (between dlls)

class QuestionTriggerCollider

```
class QuestionTriggerCollider
  : public MonoBehaviour
```

Summary

public bool *m_IsCorrect*

public inline void *ActionCall*()

public inline void *SetOnFinishSend*(Action a) Internally used function Set the callback when the collider is finished

Members

public bool m_IsCorrect

public inline void ActionCall()

public inline void SetOnFinishSend(Action a)

Internally used function Set the callback when the collider is finished

Parameters

- a

class ToolTriggerCollider

```
class ToolTriggerCollider
  : public MonoBehaviour
```

This script is attached to every child of the prefab that contains the ToolColliderPrefabConstructor Everytime each child is triggered it calls the parent when it's done via this script

If the parent has the ToolColliderPrefabConstructor script it automatically attaches this script to every 1st depth child

REQUIREMENTS: each child must have a trigger collider

Summary

public inline void *SetUsedTool*(List< ToolAndTime > _usedTools) Provide at runtime a new tool that can complete this action and the time it will need to trigger every collider

public inline void *SetOnFinishSend*(Action a) Internally used function Set the callback when the collider is finished

Members

public inline void SetUsedTool(List< ToolAndTime > _usedTools)

Provide at runtime a new tool that can complete this action and the time it will need to trigger every collider

Parameters

- _usedTools List of class that contains the tool and a float for the collider's trigger timer

public inline void SetOnFinishSend(Action a)

Internally used function Set the callback when the collider is finished

Parameters

- a

8.1.8 Utilities/UI

Summary

namespace *MAGES::UIManagement*

namespace MAGES::UIManagement

Summary

enum *NotificationUITypes*

enum *UserUITypes*

enum *ProgressUITypes*

public delegate void *Action< T1, T2, T3, T4, T5 >*(T1 p1,T2 p2,T3 p3,T4 p4,T5 p5)

class *MAGES::UIManagement::DecisionInterfaceMediator*

class *MAGES::UIManagement::InterfaceManagementMediator*

class *MAGES::UIManagement::UIManagementMediator*

Members

enum NotificationUITypes

UINotificationUIWarningUIErrorUICriticalError

enum UserUITypes

UIOptionsUIOperationStartUIOperationExitUIOperationTimelineUIMenuUIOnlineSessionsUICreateSessionUIJoinSession

enum ProgressUITypes

ProgressUIProgressUI100

public delegate void Action< T1, T2, T3, T4, T5 >(T1 p1,T2 p2,T3 p3,T4 p4,T5 p5)

class MAGES::UIManagement::DecisionInterfaceMediator

```
class MAGES::UIManagement::DecisionInterfaceMediator
  : public MonoBehaviour
```

class MAGES::UIManagement::InterfaceManagementMediator

```
class MAGES::UIManagement::InterfaceManagementMediator
  : public MonoBehaviour
```

class MAGES::UIManagement::UIManagementMediator

```
class MAGES::UIManagement::UIManagementMediator
  : public MonoBehaviour
```

CHAPTER

NINE

GETTING STARTED

9.1 Step by step

9.1.1 Download and Import MAGES™ SDK

Before you start

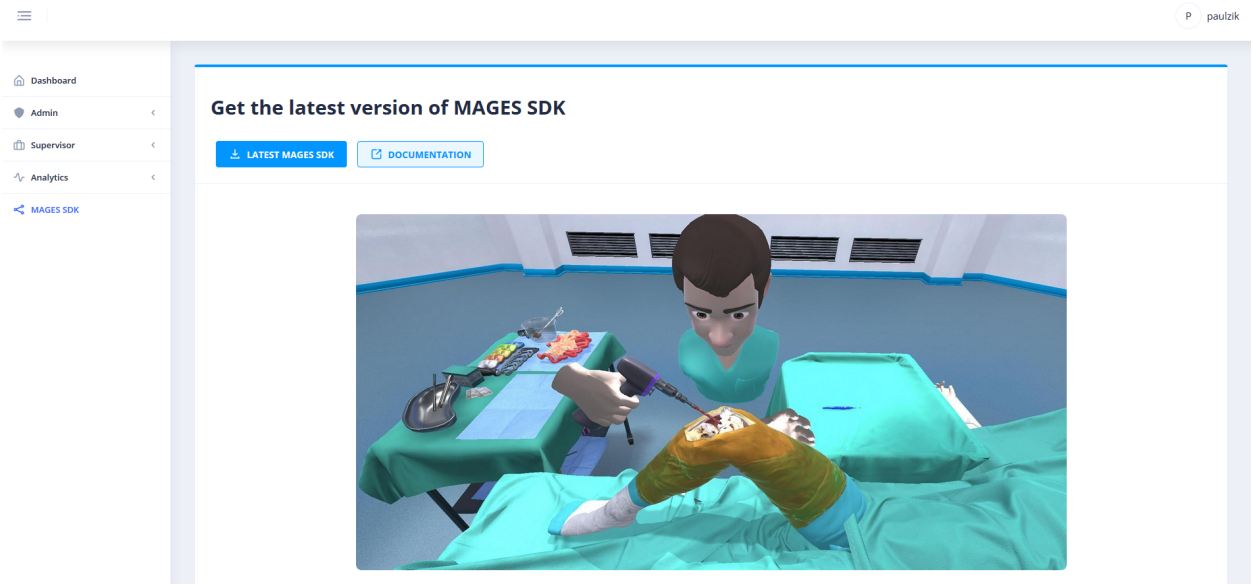
Latest release of MAGES Unreal runs on Unreal Engine 4.27.

You will have to download the Epic Installer and then from the Unreal Engine tab install editor version 4.27 with the latest minor version (4.27.2 at the time of writing).

> **Warning:** Make sure you install this specific major version otherwise you may encounter incompatibilities between internal and 3 rd party packages.

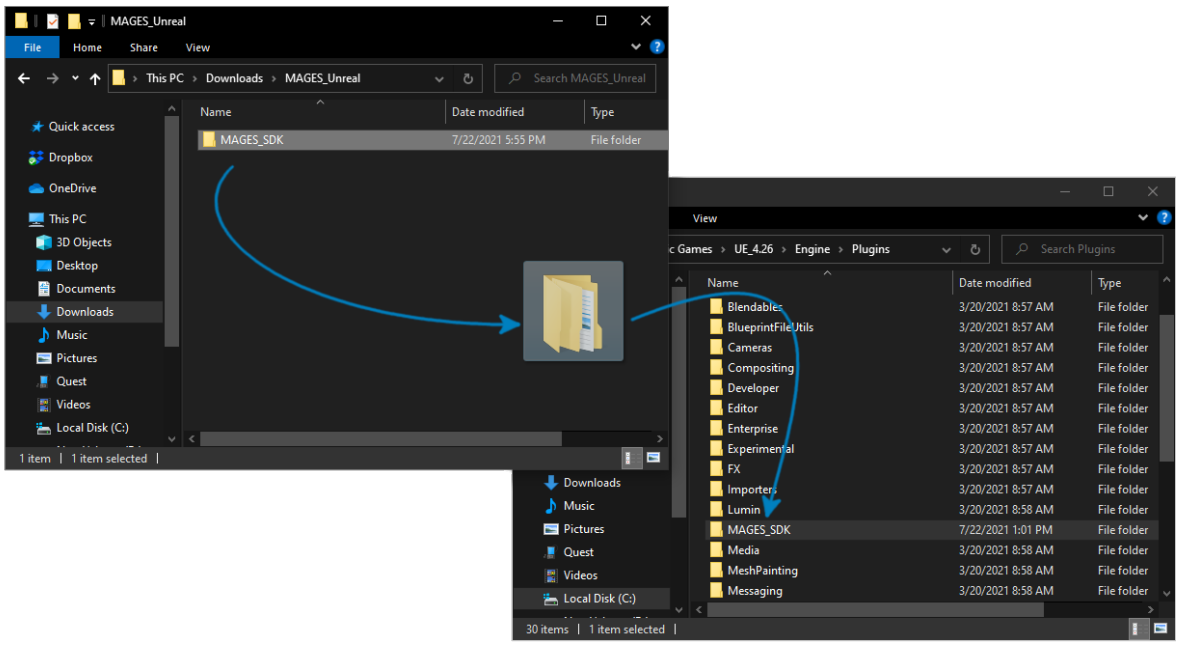
Download MAGES™ SDK

Download the latest MAGES SDK Unreal plugin from the ORamaVR Portal.



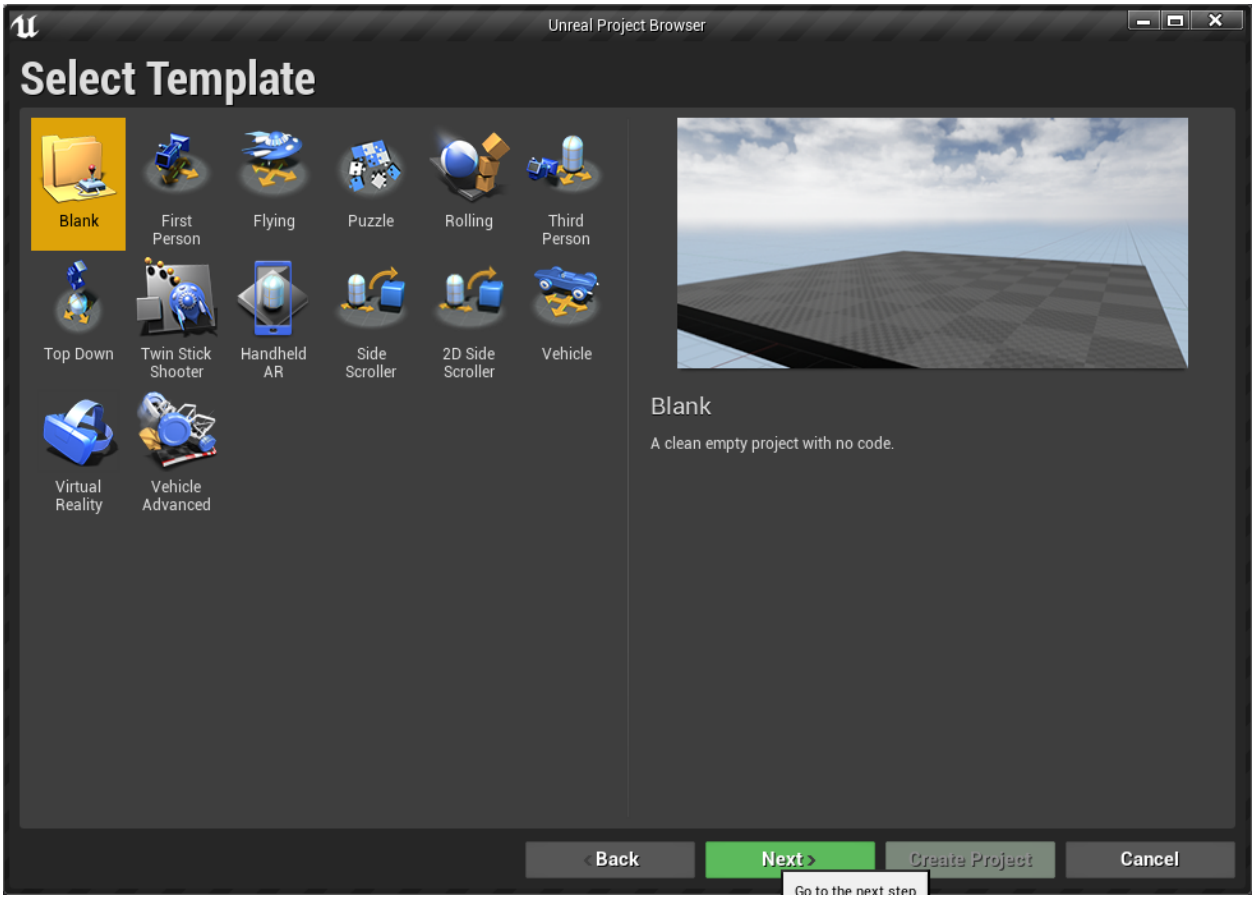
Import MAGES™ SDK

It is imperative that MAGES_SDK is installed as an engine plugin (and not as a project plugin) by copying the "MAGES_SDK" plugin folder into the "Plugins" folder inside the engine (that would typically be `C:\Program Files\Epic Games\UE_4.27\Engine\Plugins`)
Navigate to the engine plugin folder and paste the "MAGES_SDK" folder into it.



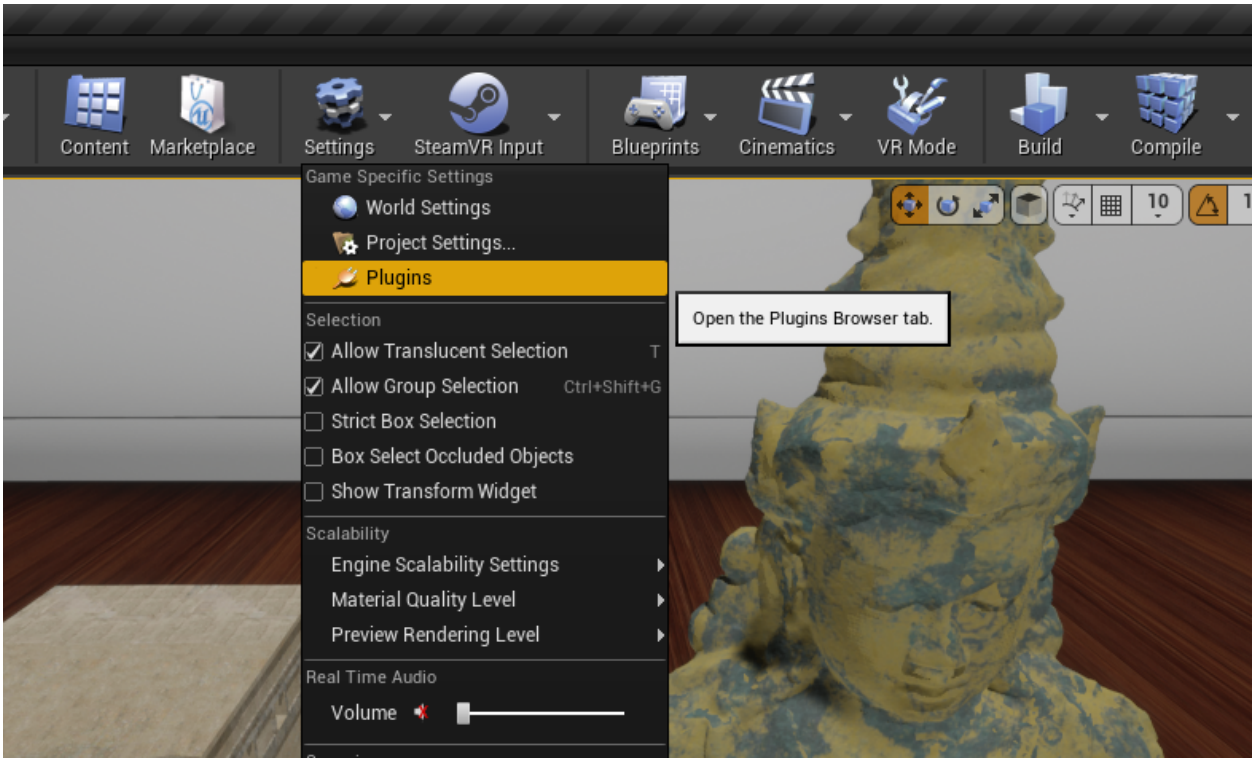
Project Setup

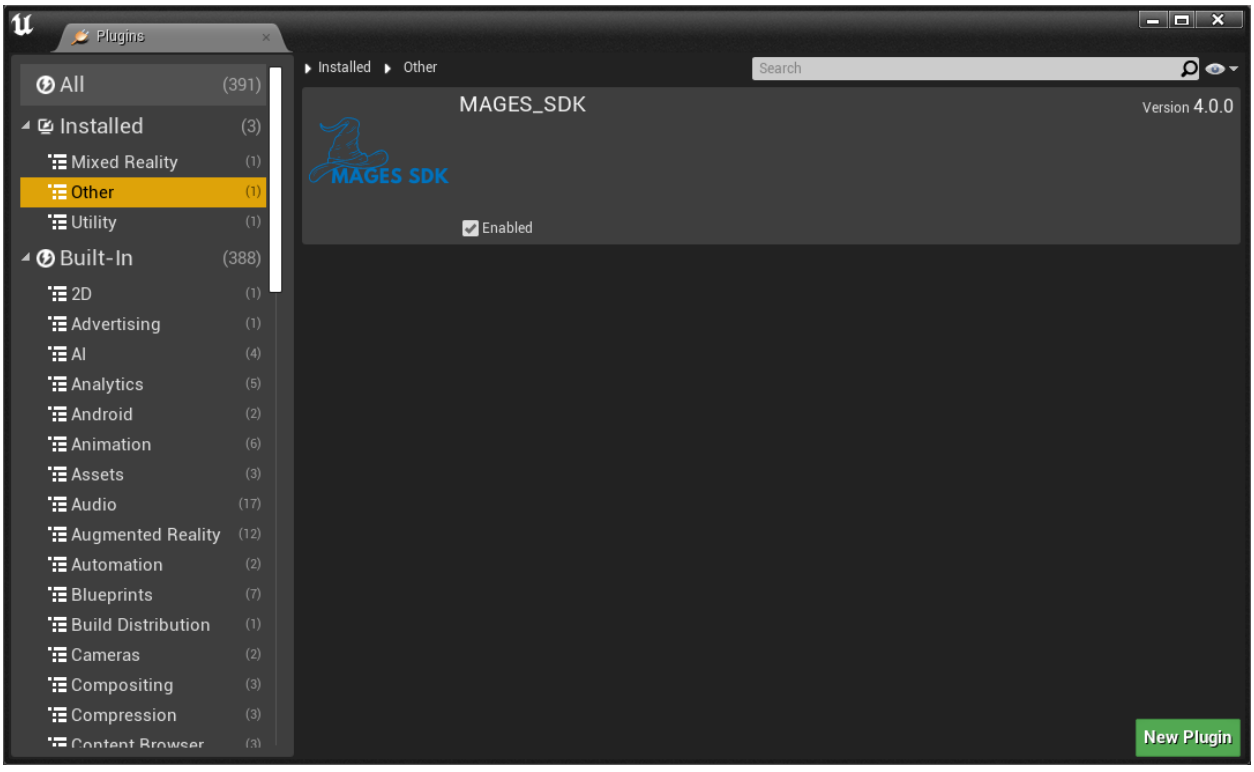
Launch Unreal Engine 4.27, select the Games category and create a blank project.



As of version 4.0.1, MAGES Unreal utilizes all the core features required for development

From the Unreal Editor navigate to **Settings** on the top menu, select **Plugins**, search for *MAGES_SDK* and enable as shown in the figure below.





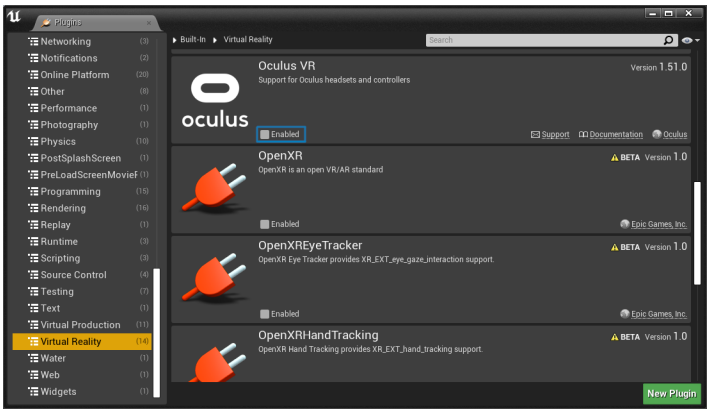
Note: If you want to test the SDK with Non-VR mode, you can skip the following section

Enable the corresponding plugin for your VR headset

Generally, the following applies:

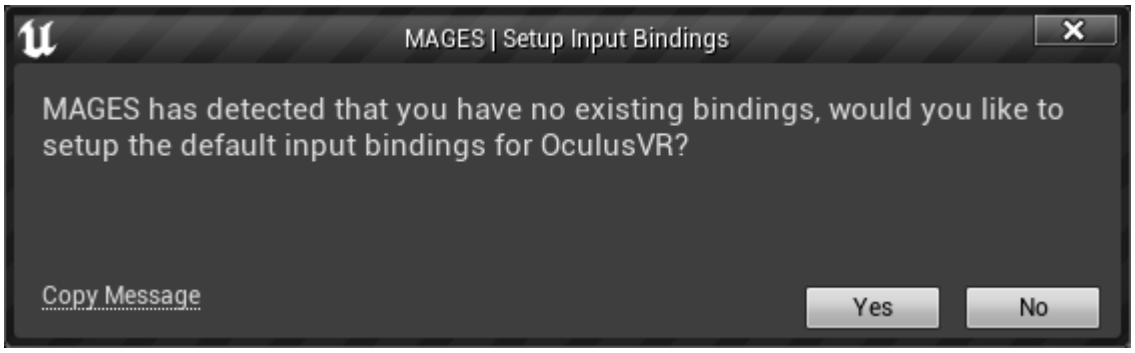
VR SDK	Supported Devices	Remarks
OculusVR	Oculus Rift, Oculus Rift S, Oculus Quest 1&2 w/ Oculus Link	
SteamVR	Most desktop HMDs, as well as mobile HMDs with Oculus Link like capabilities	You need to install SteamVR, as well as the vendor specific drivers
OpenXR	Most desktop HMDs, as well as Oculus Quest 1&2 w/ Oculus Link	Experimental in Unreal; can cause many crashes, and does not work in Packaged games

Warning: Disable all of the other VR plugins except the one that you have selected to use



Afterwards, you will have to restart the editor.

Note: If you enabled any of the VR plugins, MAGES will prompt you on the next restart of the editor with the following:



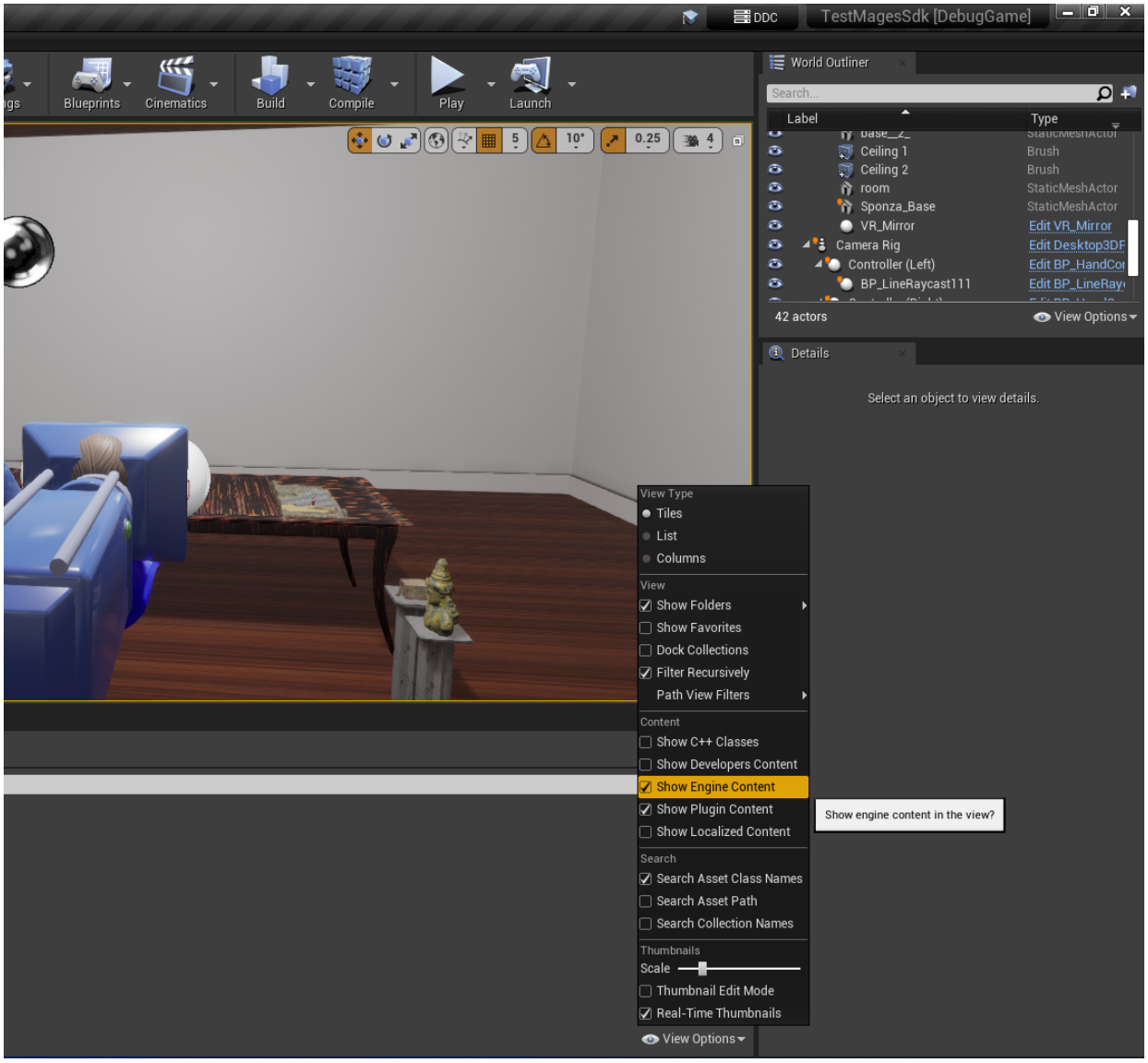
Click “Yes” here, otherwise you will need to manually setup the key bindings based on your installed VR SDK instructions.

9.1.2 Load a MAGES SDK sample

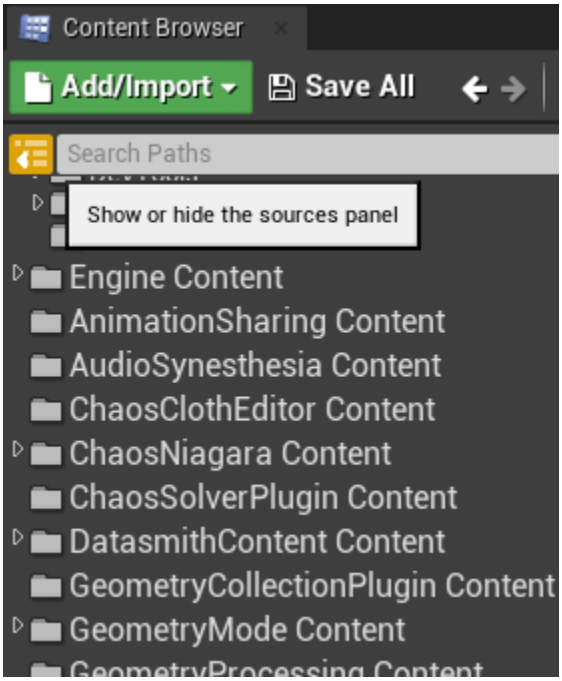
With MAGES Unreal, 3 sample applications are provided:

- Sample App (Cultural Heritage)
- Medical Sample App
- CVRSB (see here before running)

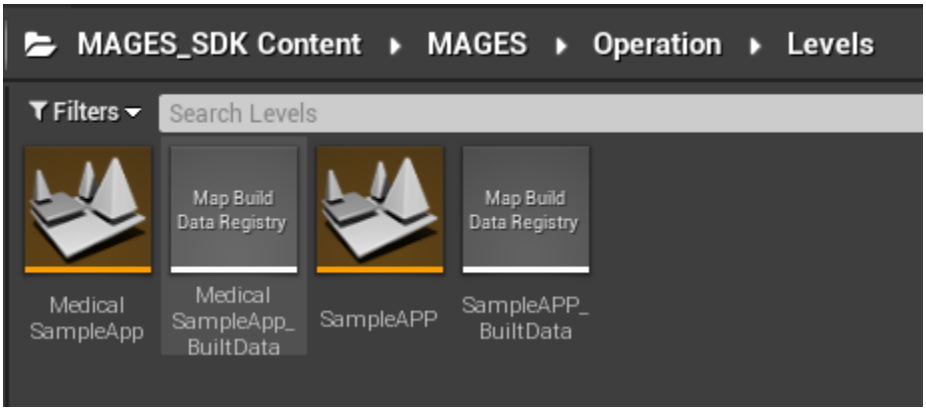
After the editor has launched, in the Content Browser, check if the Engine and the Plugins folders are visible, in the Content Browser. “Show Engine Content”, and “Show Plugin Content” must both be ticked:



Additionally, make sure that the sidebar is enabled, by clicking on the following button on the left side of the Content Browser:



From the *Content Browser* navigate to **MAGES_SDK Content > MAGES > Operation > Levels** and open the **SampleApp** level.



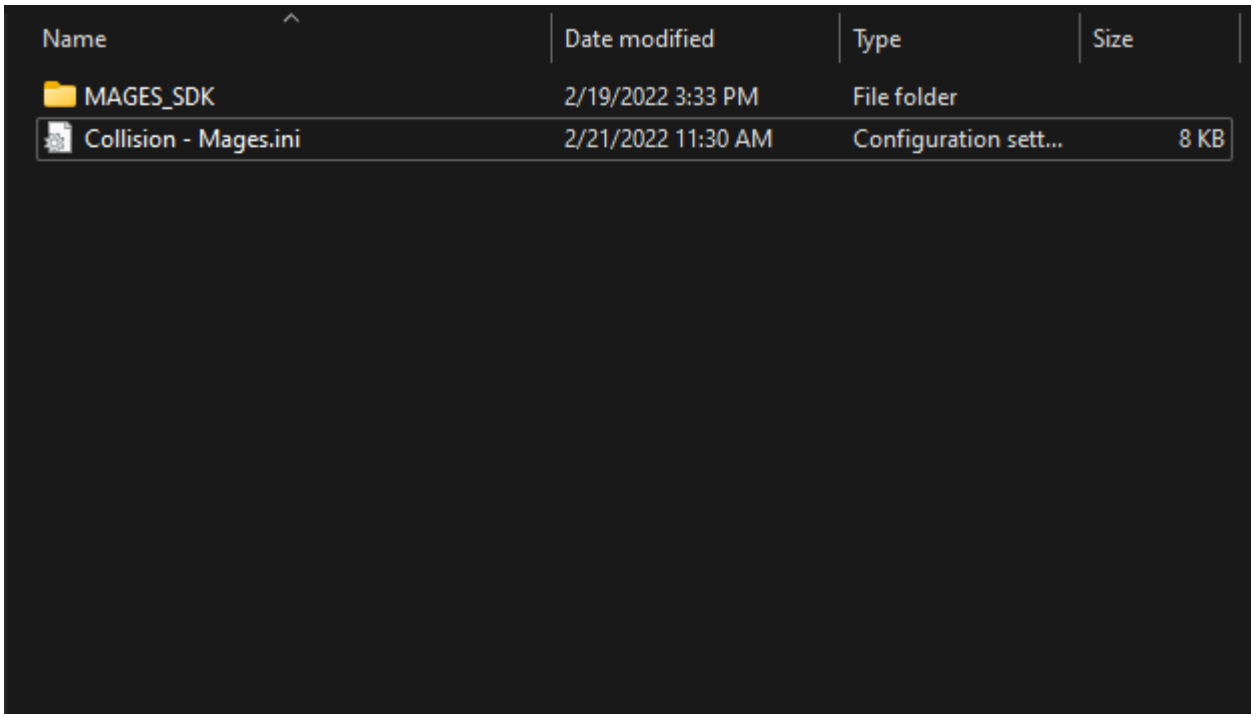
Warning: In case you want to try a level other than SampleAPP, follow the instructions mentioned *here*

Collision Settings

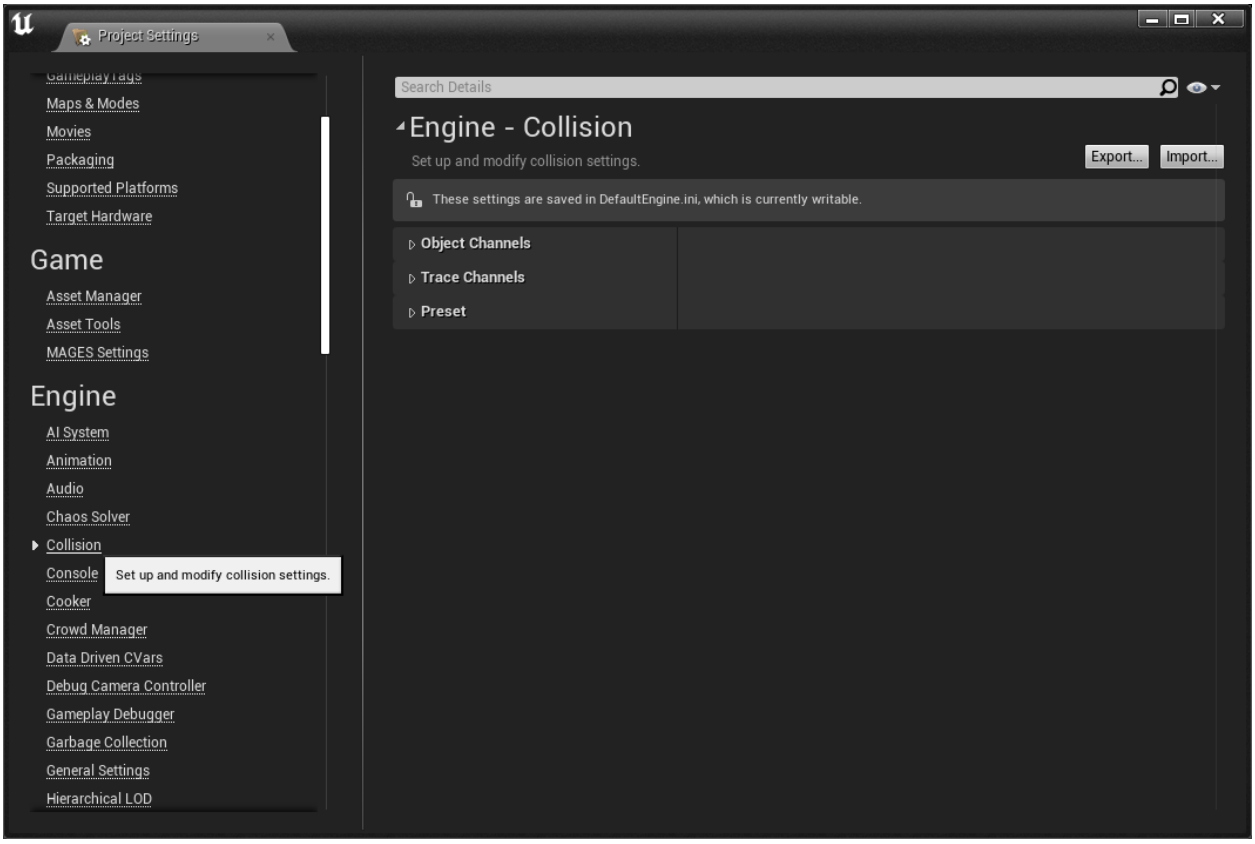
MAGES_SDK uses custom collision presets for some important parts of interaction and hit-testing. You can either choose to import the ini file from here, or set-up collision by yourself.

Import INI File

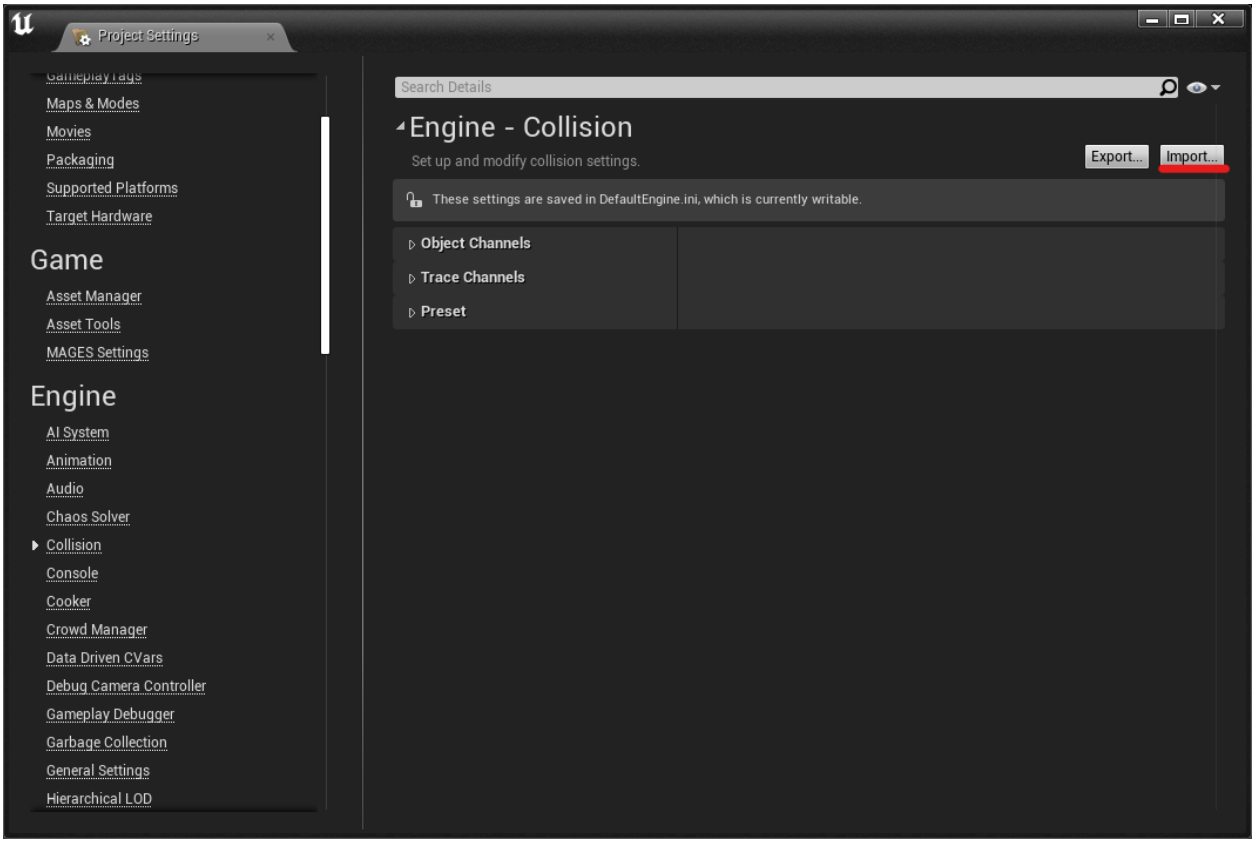
You can find this configuration file inside the plugin download folder:



Open the Project Settings, and look for the Collision section in the sidebar:



On the top right of the window, click Import and select the file you have just downloaded.

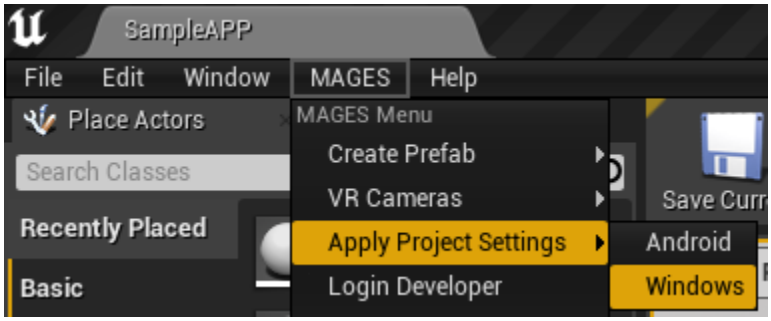


If you want to manually install these collision settings you can find the detailed steps *here*.

Note: You will have to restart the editor for the collision settings to be applied.

Default Project Settings

From the MAGES Menu, click on "Apply Project Settings > Windows". This will configure the project with some good default settings regarding rendering and performance. **You will have to restart the editor in order for these changes to be applied.**

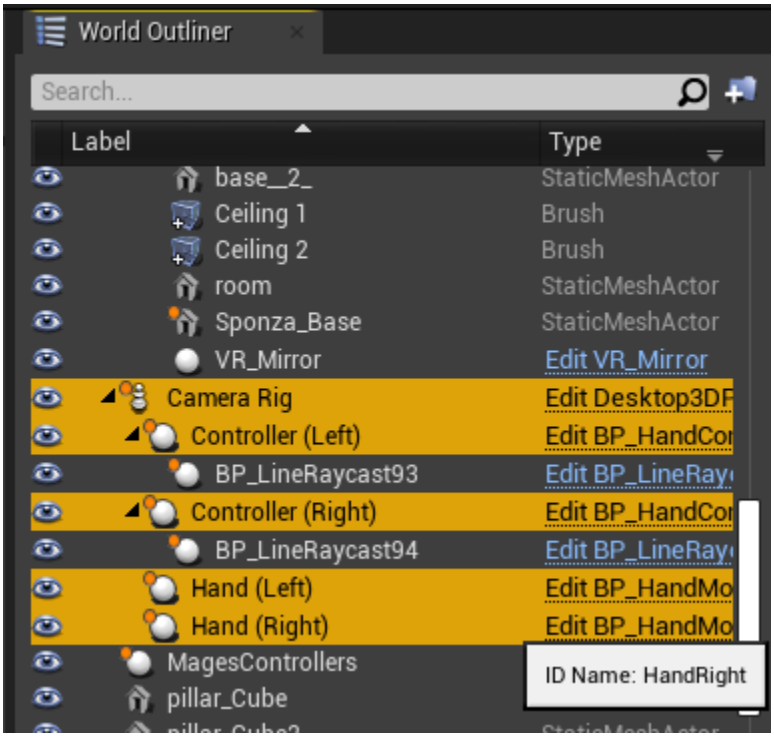


Place a VR Pawn into the level

Note: You do not have to follow this section if you have chosen to play the sample application in Non-VR mode.

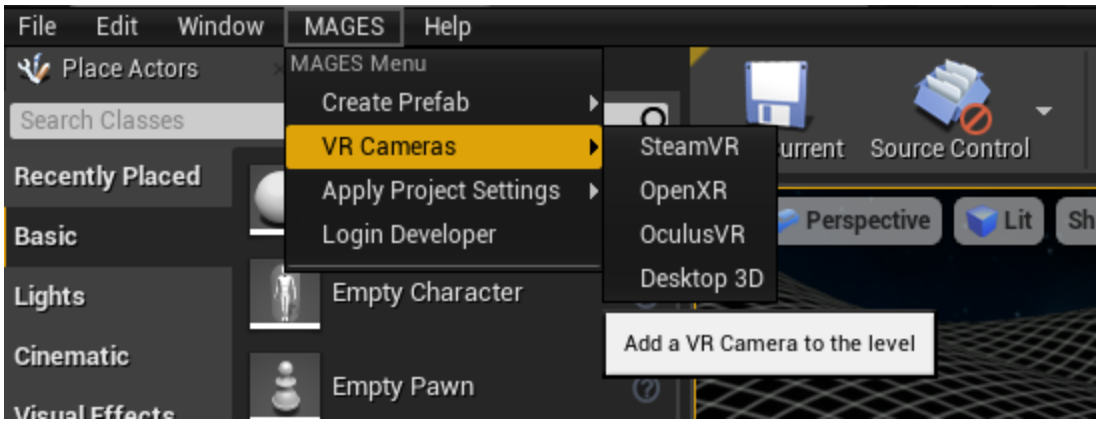
MAGES_SDK supports OpenXR, SteamVR and OculusVR. Based on what VR SDK you selected in the *previous step*, you must setup the corresponding VR pawn (camera) in the current level.

By default, the map will include the Desktop 3D pawn, which allows for testing and running operations without using a VR HMD. To setup a VR enabled pawn, delete the camera rig, controllers and hands that are already present in the level:



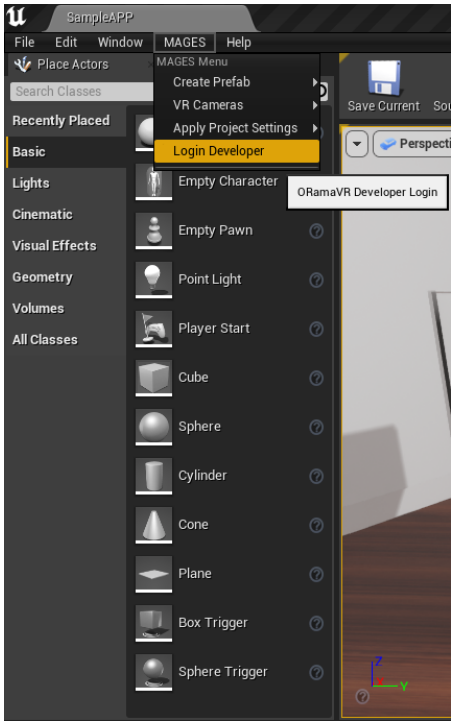
Note: To delete any number of actors, select them in the world outliner or in the level, and press "Delete" on your keyboard

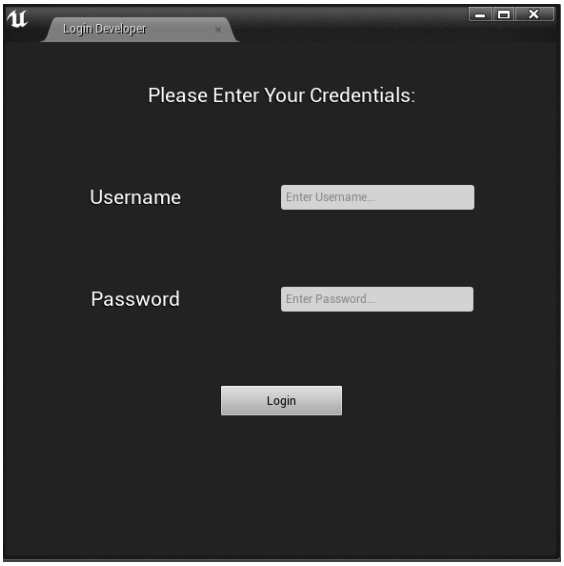
Following, we need to select the appropriate camera from the dropdown MAGES menu. Go to "MAGES > VR Cameras" and select the pawn that matches your VR SDK plugin.



SDK License

Before you hit the **Play** button, make sure you have checked out a valid SDK license. To do so, open the **MAGES** tab on the top menu and click on the **Login Developer** option.





Controls & Movement

In the table below you can find the controls for every supported platform and headset for playing in Non-VR mode:

Move Around	Look Around	Open Radial Menu
WASD Keys	Mouse	Spacebar

Note: You read more about this mode *here*

In the table below you can find the controls for every supported platform and headset for playing in VR.

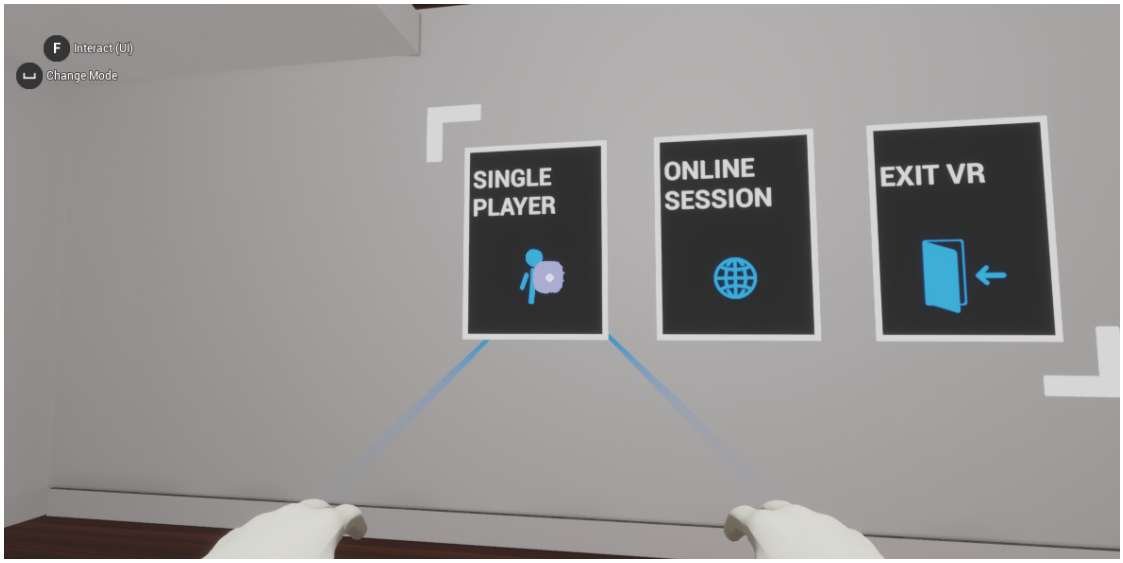
	Oculus Touch	VIVE	Windows Mixed Reality
Grabbing Objects	Grip Button	Grip Button	Grip Button
UI Selection & Usage of Tools	Trigger Button	Trigger Button	Trigger Button
Toggle Movement	Press Left Touch Thumbstick	Press Left Touch Thumbstick	Press Left Touch Thumbstick
Positional/Rotational Movement	Left/Right Touch Thumbstick	Left/Right Trackpad	Left/Right Thumbstick
Toggle In-game Options	Press Right Touch Thumbstick	Press Right Touch Thumbstick	Press Right Touch Thumbstick
Next/Previous Step	X,Z Keyboard Button	X,Z Keyboard Button	X,Z Keyboard Button

How to Play in Desktop 3D

If you want to try MAGES in Desktop 3D mode simply click the play button.

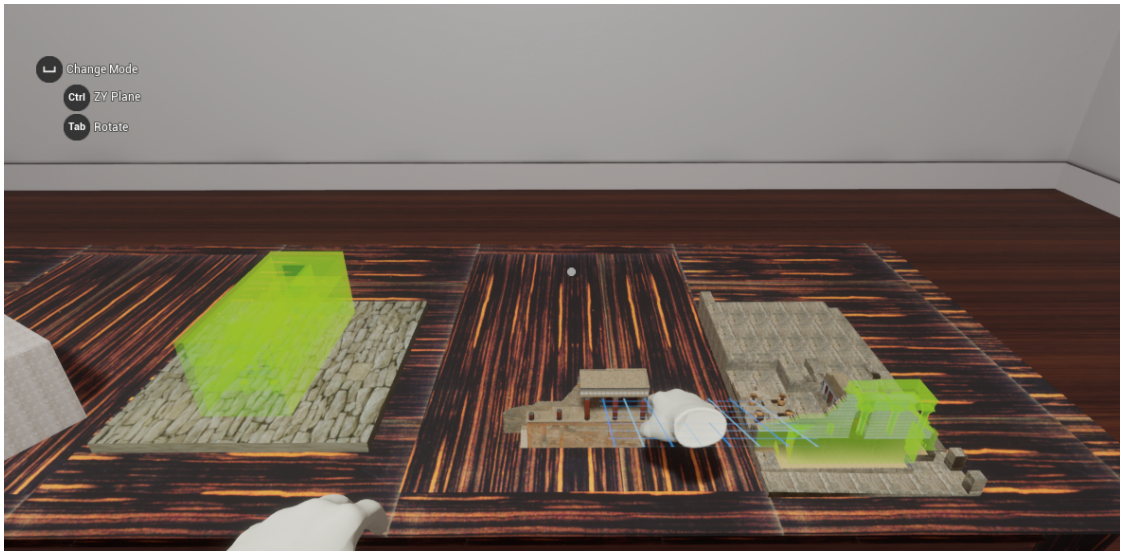
A menu will appear, offering multiple buttons as options.

Each of the user's virtual hands, has a ray which starts at the top of the hand. These exist only for debugging purposes. In the middle of the screen you will notice a small circle. Point this circle to the Single Player button and click the left mouse button.



The first action of this simulation is a question action. Use the ray to point to an answer and the left mouse button to select it. Then point to the **Submit** button, using the ray and again use the left mouse button to select it and move to the next action.

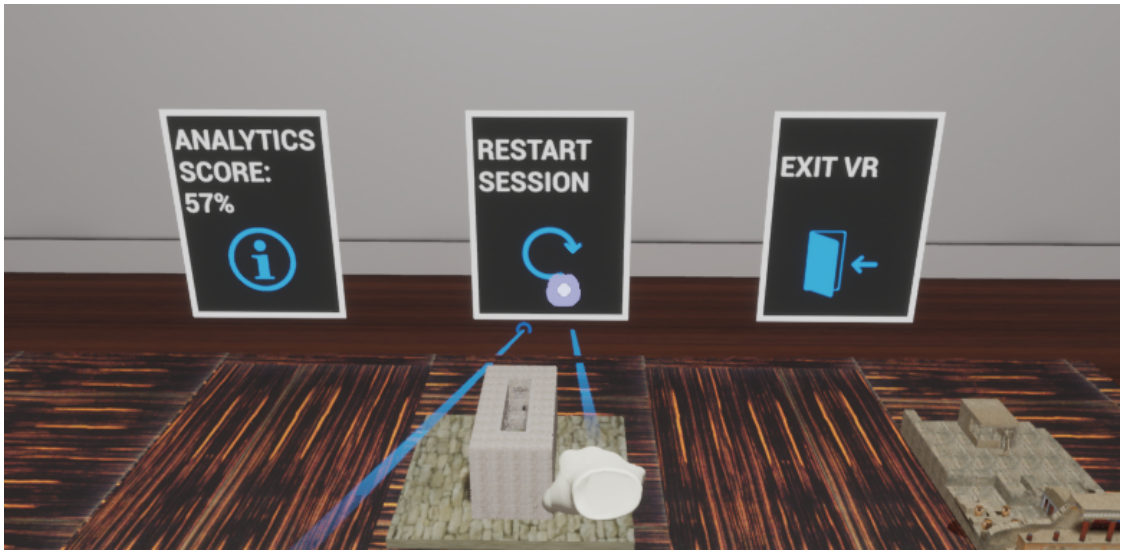
In the next action, you have to use one of the virtual hands to pick up the glowing knossos front part (by moving the hand close to the front part) and move it to the position that the green hologram represents to complete this insert action.



Once correctly inserted, the next action will start.

Note: Green holograms represent the position of the corresponding item or the way that this item should be used on a surface.

When the simulation is completed, the operation exit user interface will spawn. It will look like the one in the image below.

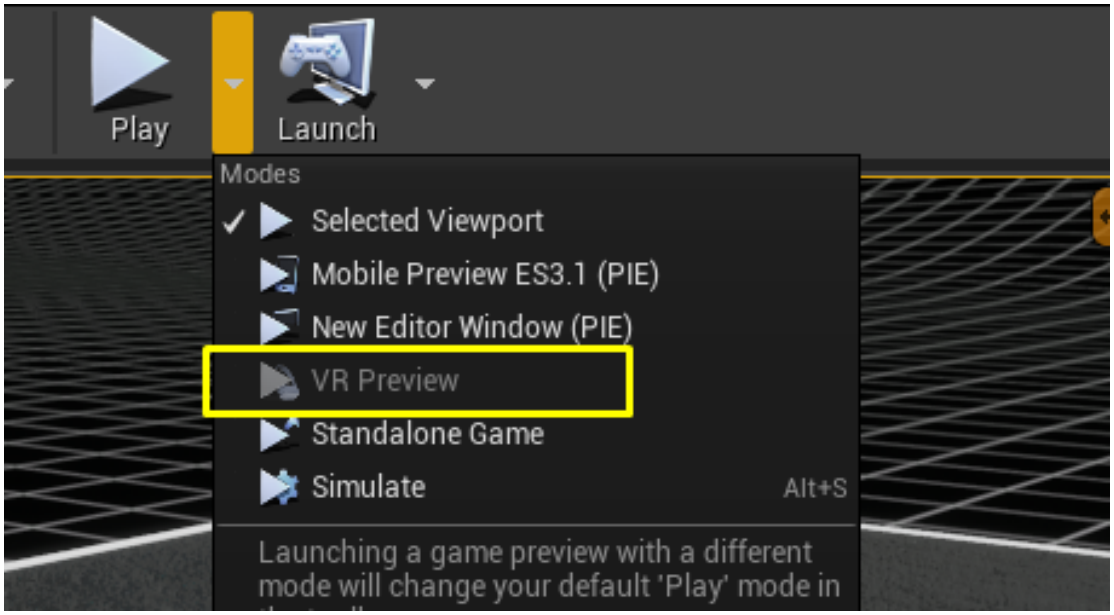


- You can view your analytics by clicking the analytics button.

- The restart button starts the simulation from the beginning.
- You can exit the simulation by clicking the exit button.

How to Play in VR

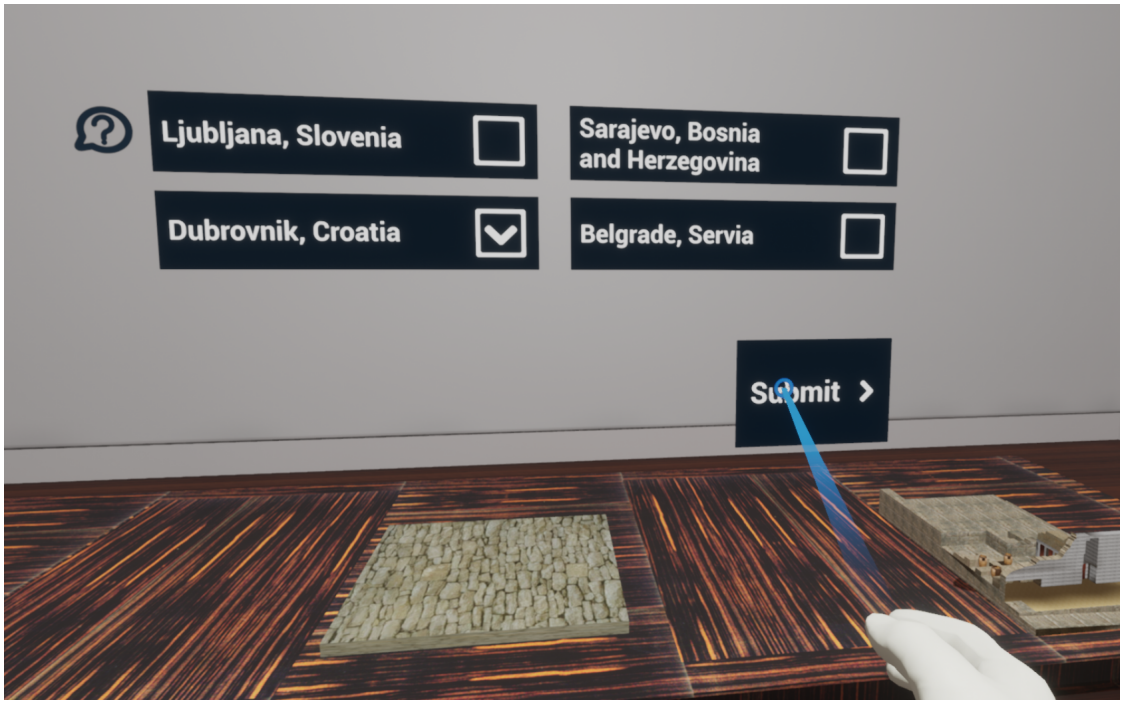
You are now ready to play the operation. If you followed the steps to play in VR click the arrow next to the play button and select *Play in VR Preview*



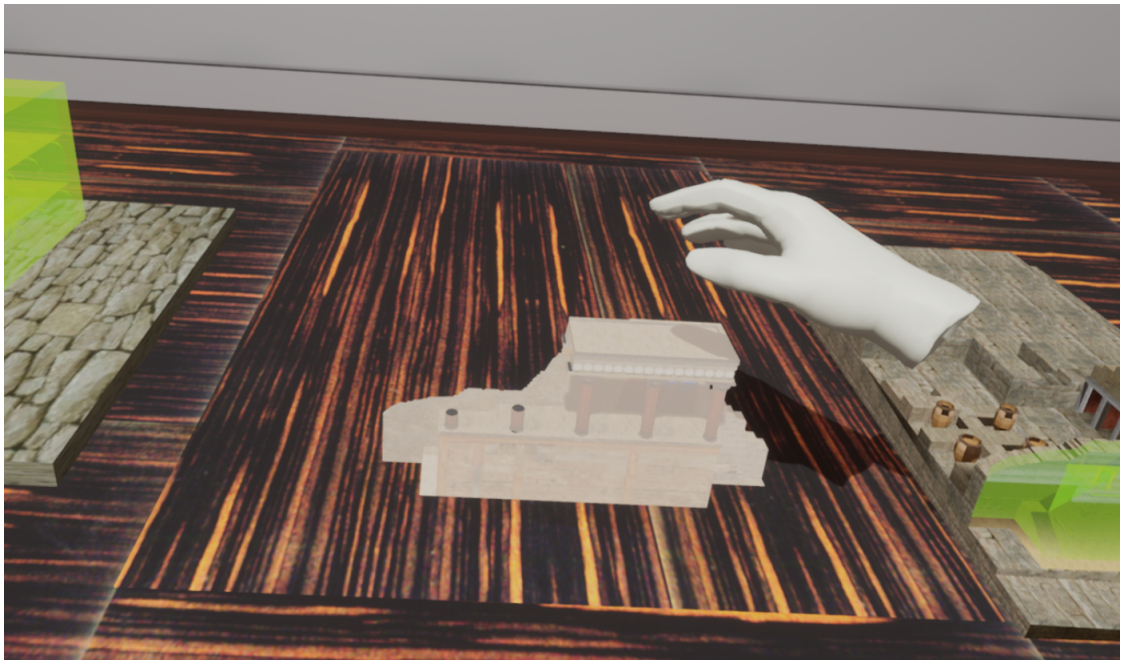
Once you start, a menu will appear, offering multiple buttons as options.

Each of the user's virtual hands, has a ray which starts at the top of the hand. Point this ray to the Single Player button and click the trigger button on the pointing controller.

The first action of the SampleApp simulation is a question action. Use the ray to point to an answer and click the trigger button to select it. Then point to the **Submit** button, using the ray and again use the left mouse button to select it and move to the next action.



In the next action, you have to grab glowing knossos front part (by moving the hand close to the front part) and pressing the trigger button and place it to the position that the green hologram indicates to complete this insert action.



Once correctly inserted, the next action will start.

Note: Green holograms represent the position of the corresponding item or the way that this item should be used on a surface.

When the simulation is completed, the operation exit user interface will spawn. It will look like the one in the image below.

- You can view your analytics by clicking the analytics button.
- The restart button starts the simulation from the beginning.
- You can exit the simulation by clicking the exit button.

MAGES Multiplayer

MAGES SDK supports Photon networking as the default networking API. For cooperation/multiplayer mode you need to setup Photon (playing in the same room with other players).

You can find instructions on how to set it up *here*.

9.1.3 First Insert Action

Introduction

Once you've finished setting up MAGES, you are ready to create your first insert action with MAGES. An Insert action involves the user manipulating an object to place it in a specific position designated by the designer of the action.

We'll guide you through the implementation of this action step-by-step from content creation to the end.

Note that, any position or rotation values that look like the following:

```
(X=1.000000,Y=1.000000,Z=1.000000)
```

can be copied from the documentation and pasted directly into the editor verbatim by right-clicking on the target property:

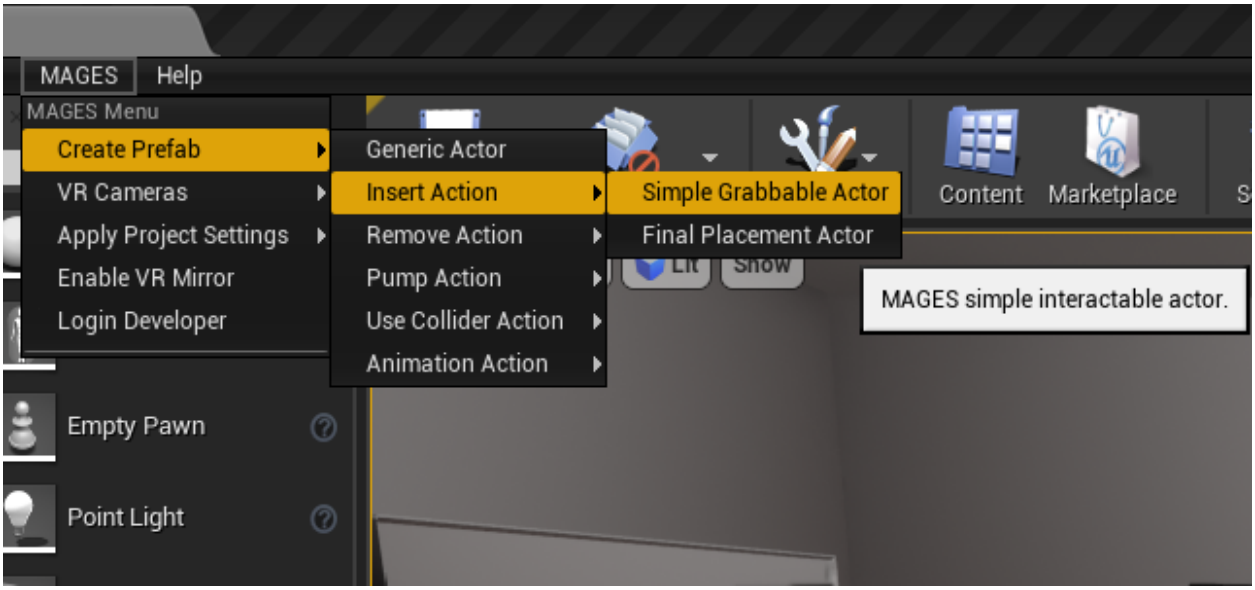
Overview

The Insert Action requires at least two blueprints:

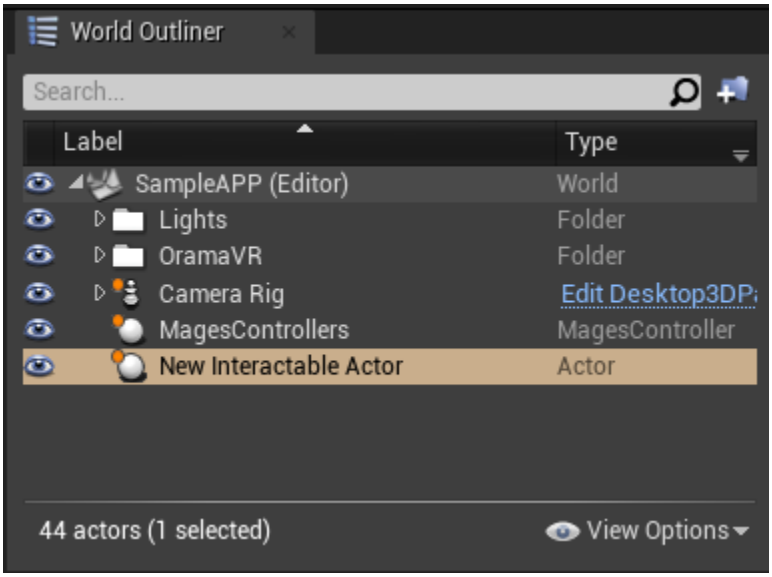
- The Insert Blueprint – the interactive item
- The Final Blueprint – the location and orientation where this item is meant to be placed
- A Hologram – Used to help guide the user visually

Creating the Insert Blueprint

We'll get started by opening the MAGES Menu, and selecting "Create Prefab" > "Insert Action" > "Simple Grabbable Actor":



In the World Outliner, you should see a new actor named "New Interactable Actor". Select it as shown below:

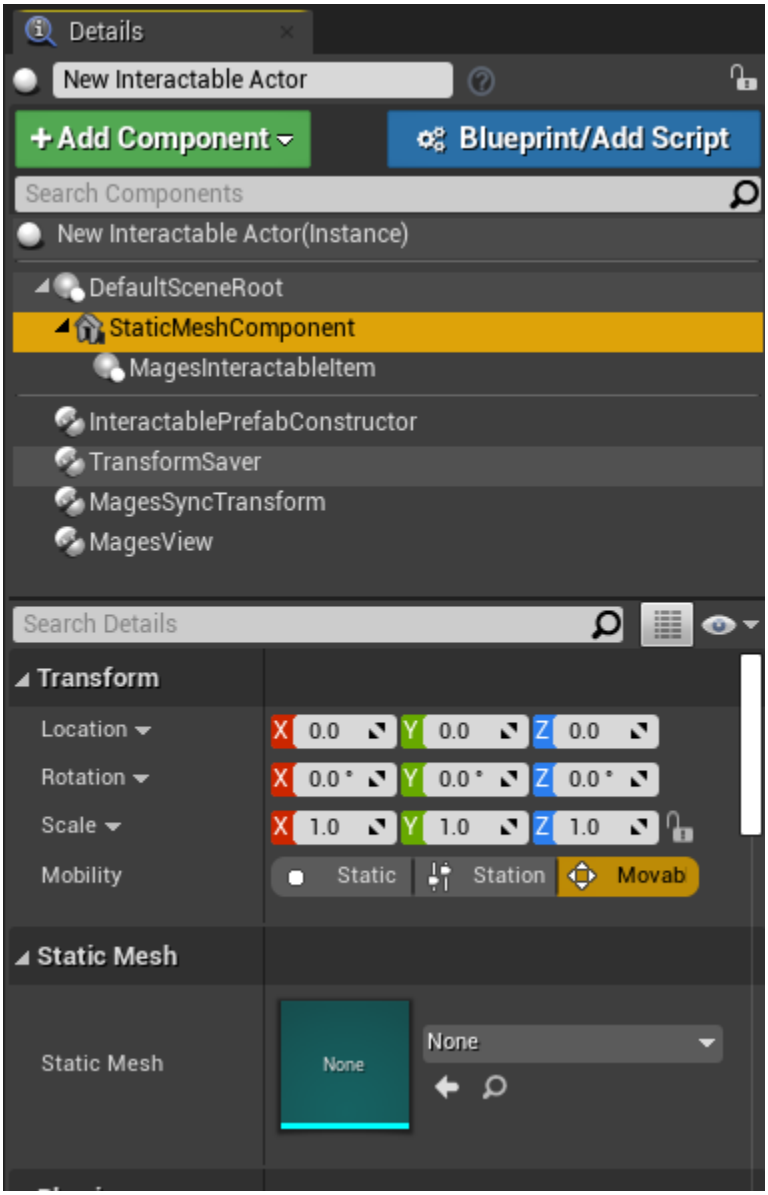


Let's set the actor to show a very simple cube.

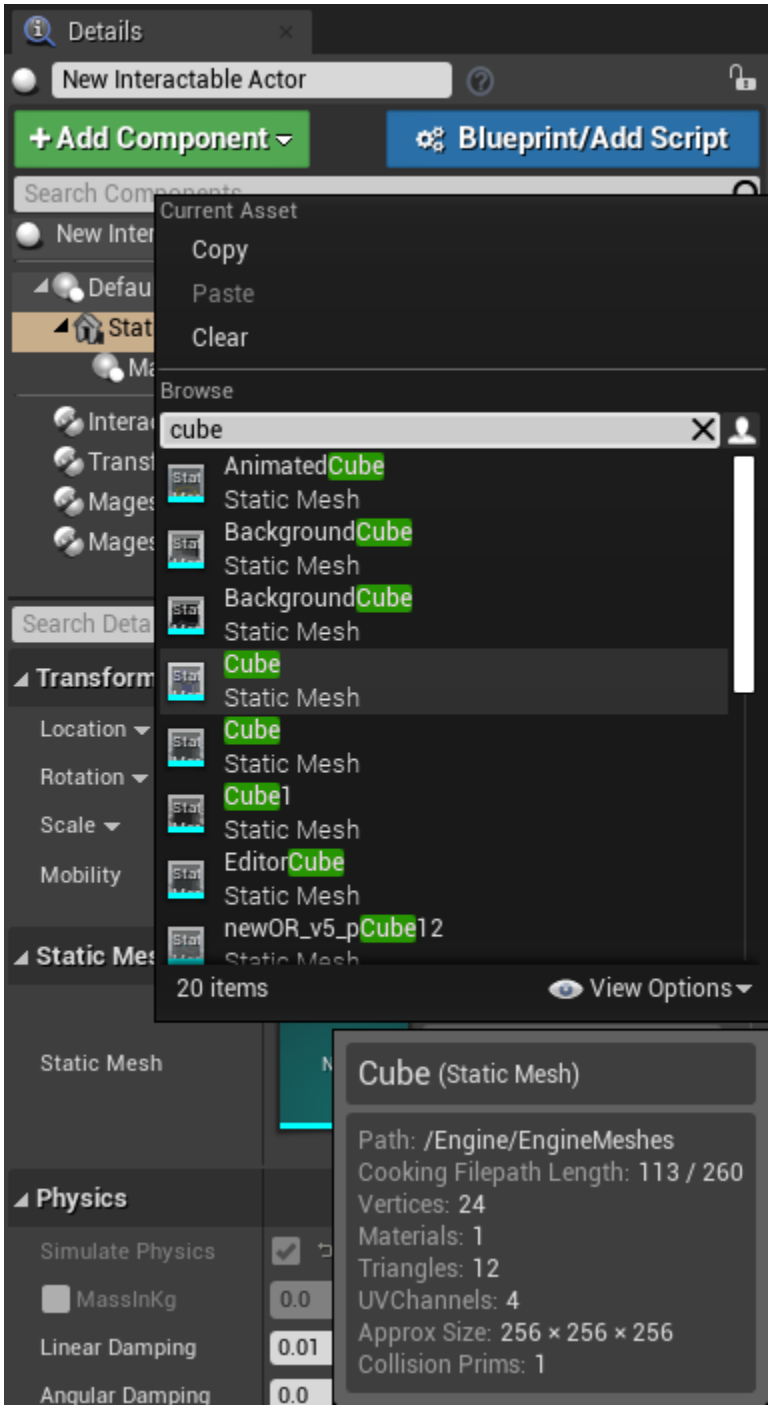
In the Details Panel, find and select the component named "StaticMeshComponent":

Note: If you cannot see any components there, try deselcting the actor by clicking somewhere in the viewport, or on another actor in the World Outliner

---



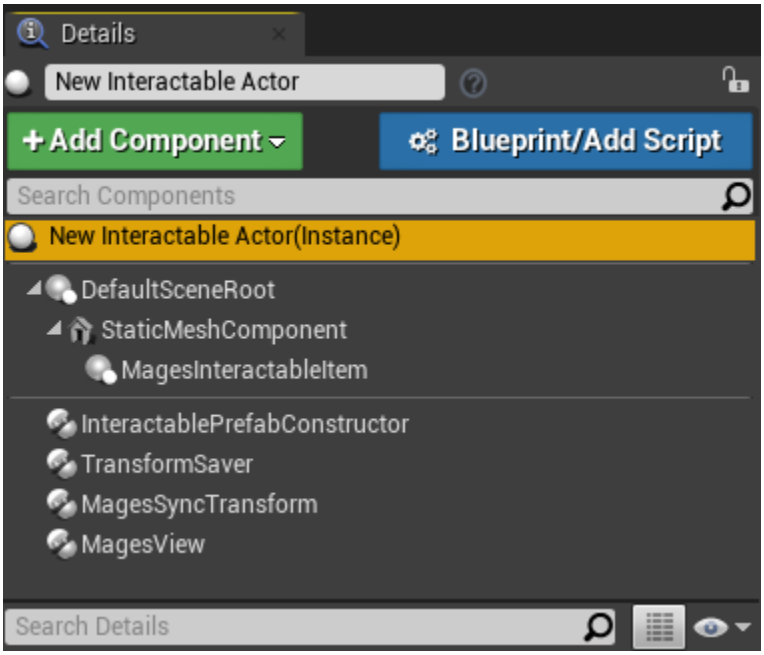
In the Component's Detail panel, set the "Static Mesh" property to a mesh of your choice. For this example, we'll use the "Cube" mesh.



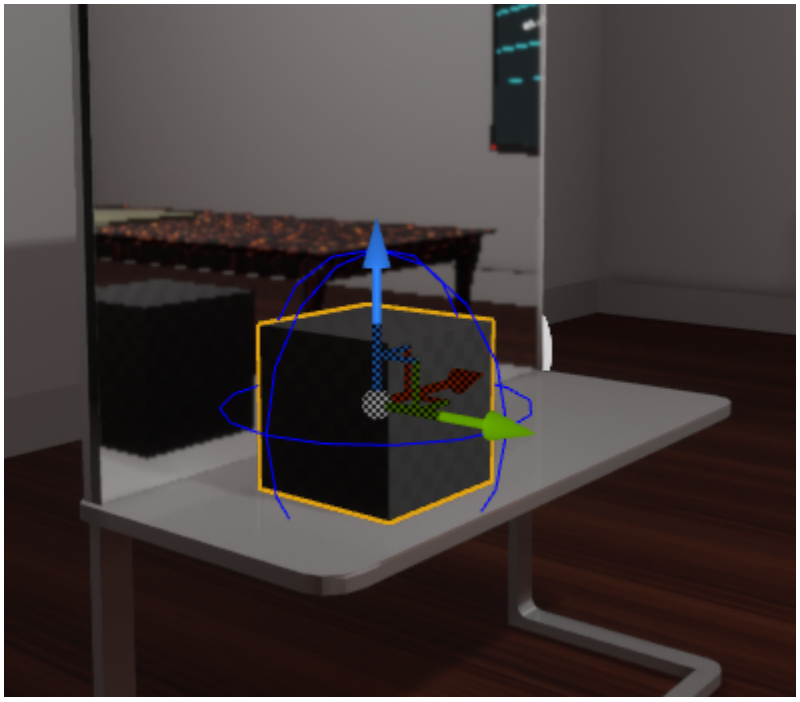
In this case, our cube is a little too big for something that would be realistically grabbable, so we'll change its "Scale" property to `(X=0.100000,Y=0.100000,Z=0.100000)`

This is a bit more realistic now. Next up, we'll move the actor to a more plausible location by doing the following:

In the Components panel, select the "New Interactable Actor(Instance)" item so that any change we make will be applied to the actor's transform, and not to the component's transform:

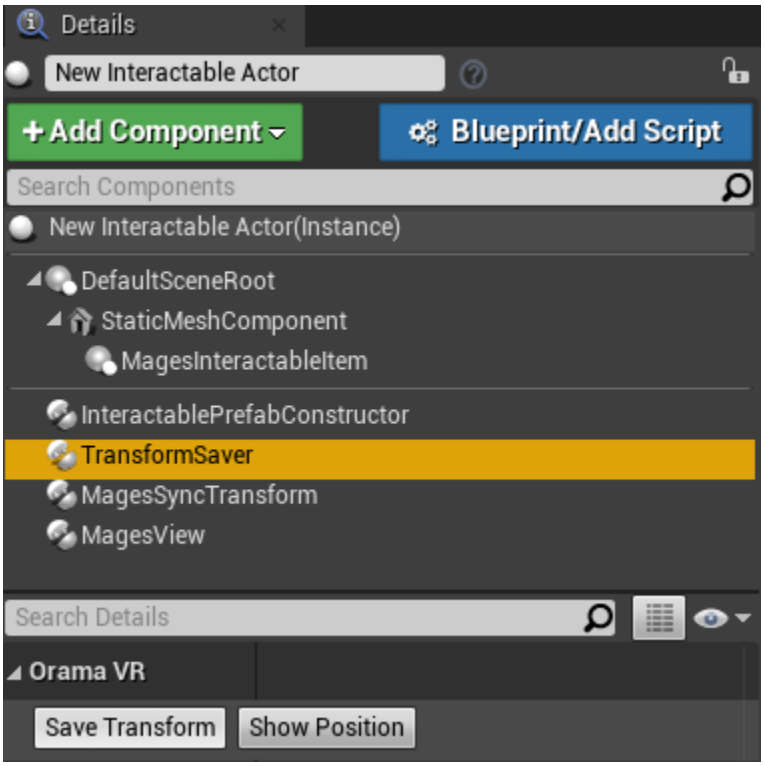


Now we'll move the actor on top of the mirror surface (specifically, `(X=0.001282,Y=-245.366364,Z=131.245087)`):

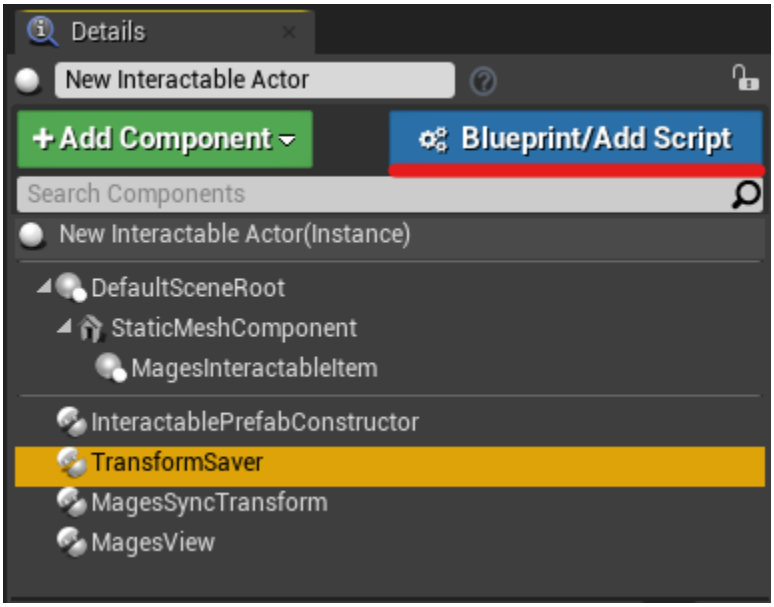


Afterwards, we have to save the Transform of the actor, so that it will be spawned at the exact position we've specified:

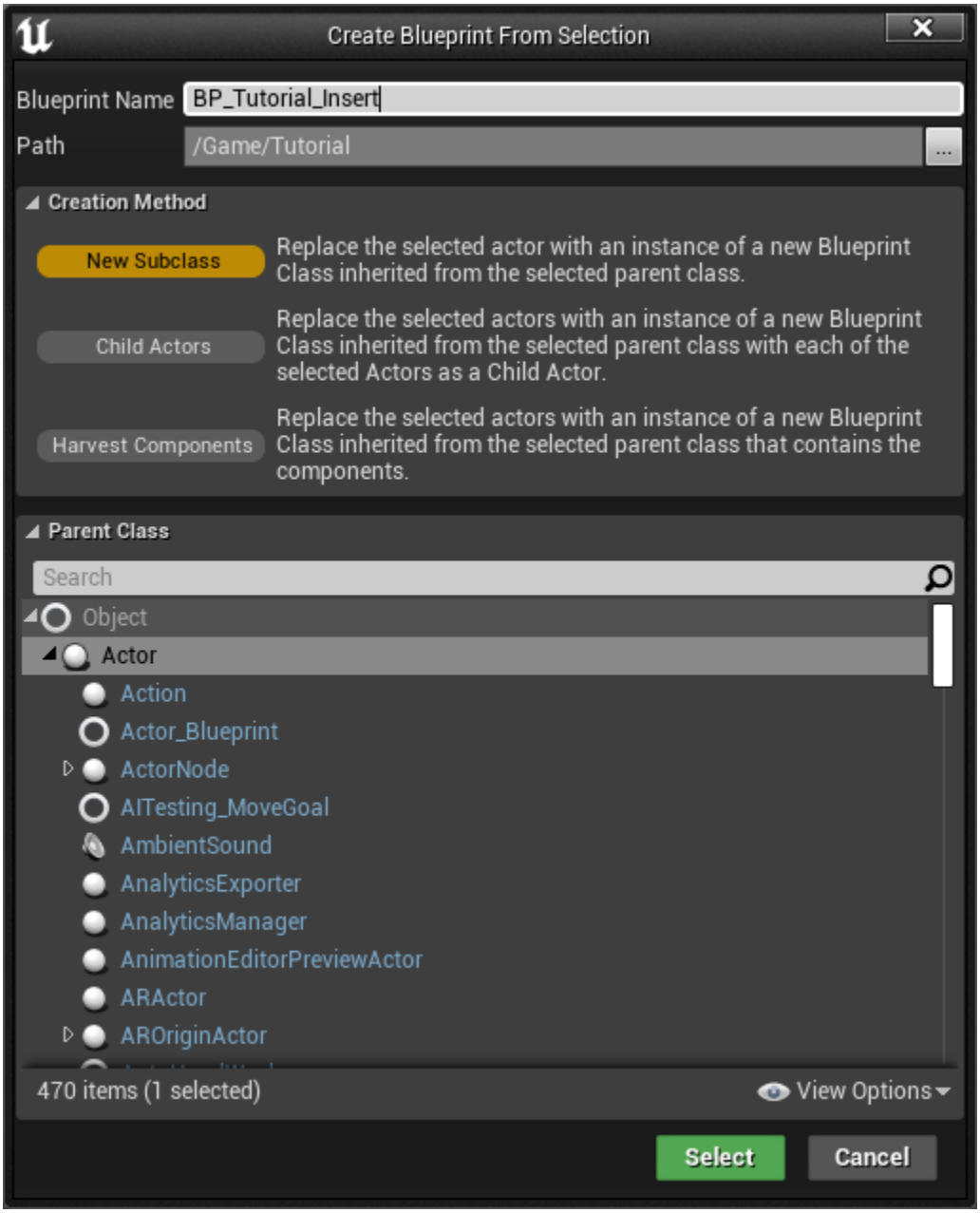
In the Components panel, select the "Transform Saver" component and below click the "Save Transform" button:



Finally, we need to convert this Level Actor into a blueprint. Click the "Blueprint/Add Script" button:



A prompt will open up, asking you for the name, and save location of the blueprint. For this tutorial, we chose to name it “BP_Tutorial_Insert”:



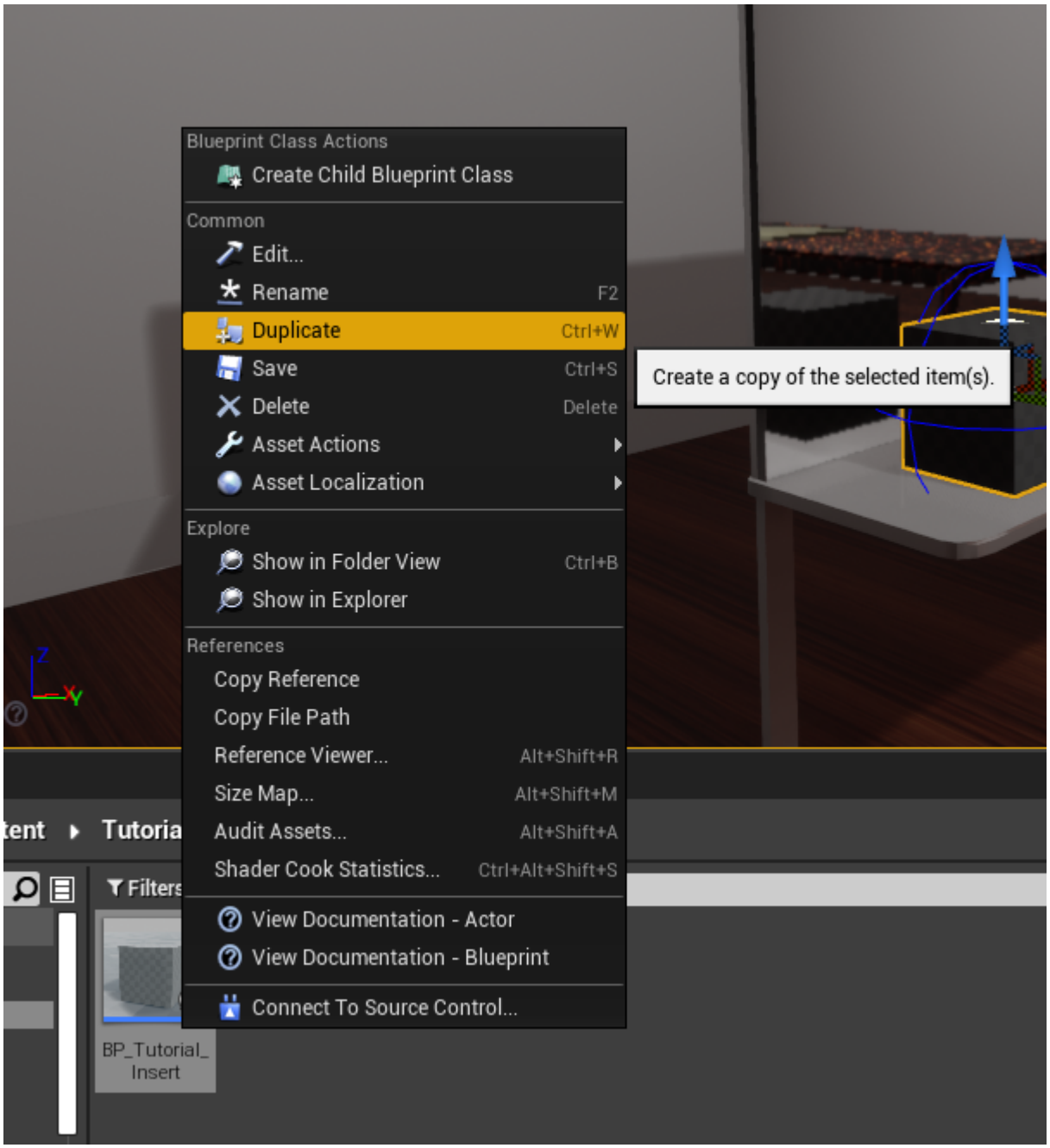
The blueprint will open in the editor. Save it and close the editor.

We have finished creating the Interactable Insert Blueprint! Now we can go about creating the Final Blueprint in a similar manner as with the Insert Blueprint (using the MAGES Menu), but most of the time, the Final Blueprint has very similar properties to the Insert Blueprint, so we'll show you a faster way:

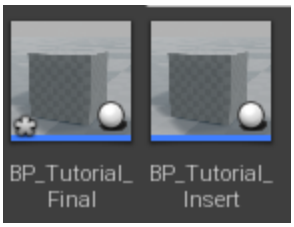
Creating the Final Blueprint

To create the final blueprint, we will duplicate the Insert Blueprint, and change some of its properties:

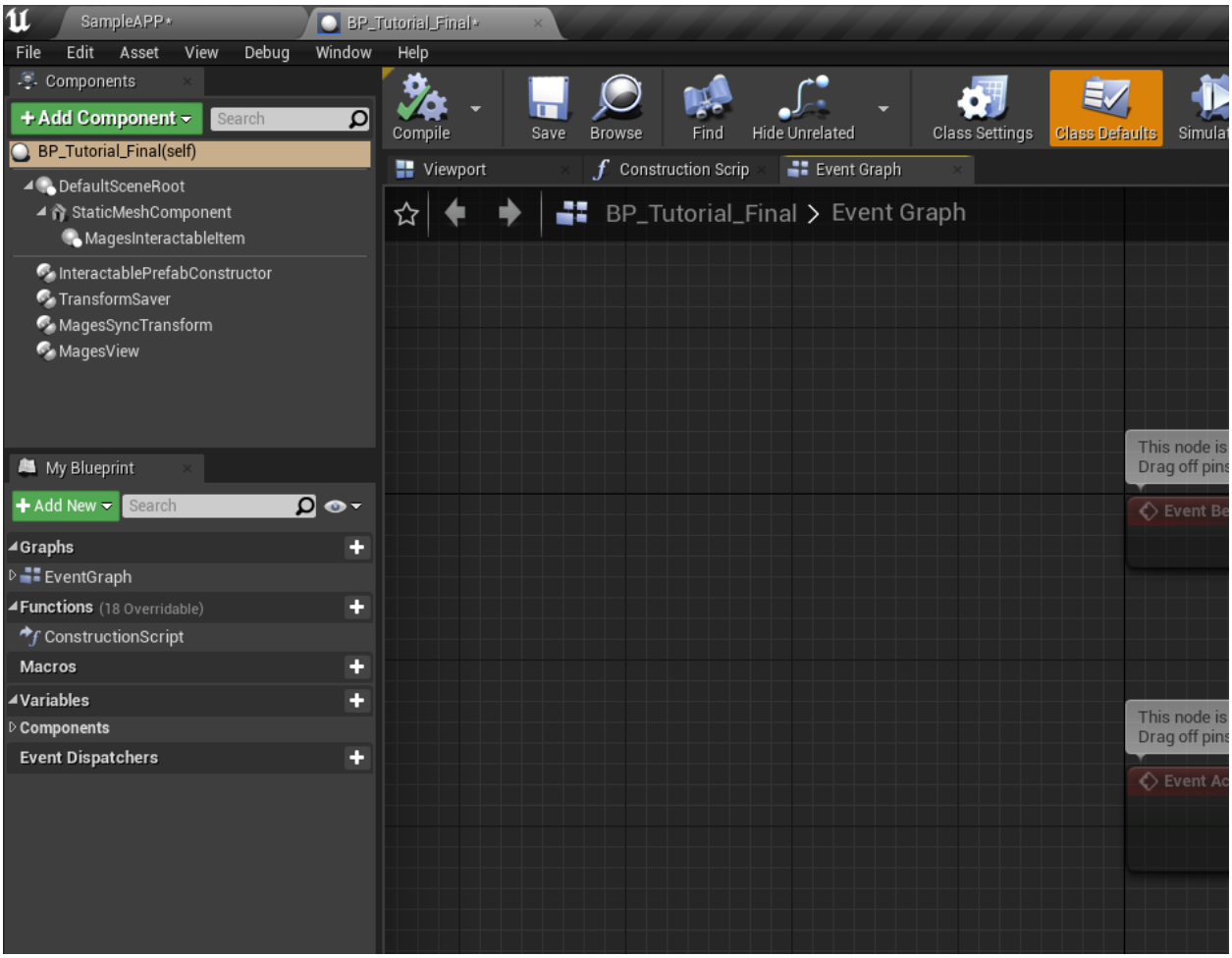
Find the Insert Blueprint inside the Content Browser (the name used in this tutorial is "BP_Tutorial_Insert"), right-click on it, and choose "Duplicate":



A new Blueprint will be created; an exact replica of "BP_Tutorial_Insert". We'll name it something more appropriate, like "BP_Tutorial_Final":



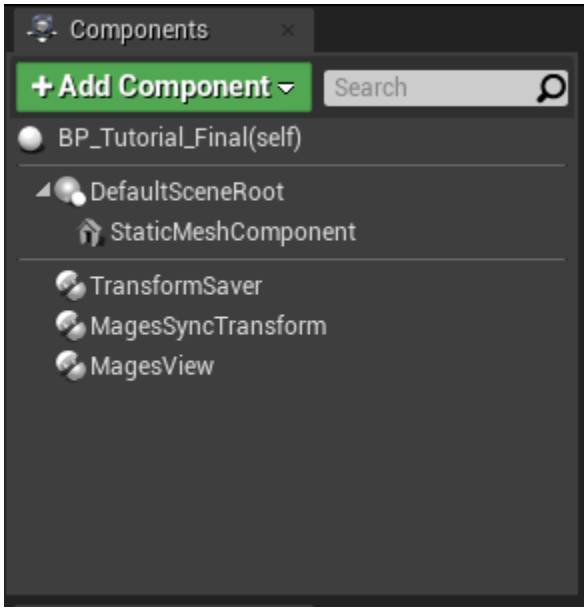
Now, open the Final Blueprint, and find the Components panel (in the top-left of the window):



Delete the following components by selecting them, and pressing the Delete key on the keyboard, or by right-clicking and choosing “Delete”:

- MagesInteractableItem
- InteractablePrefabConstructor

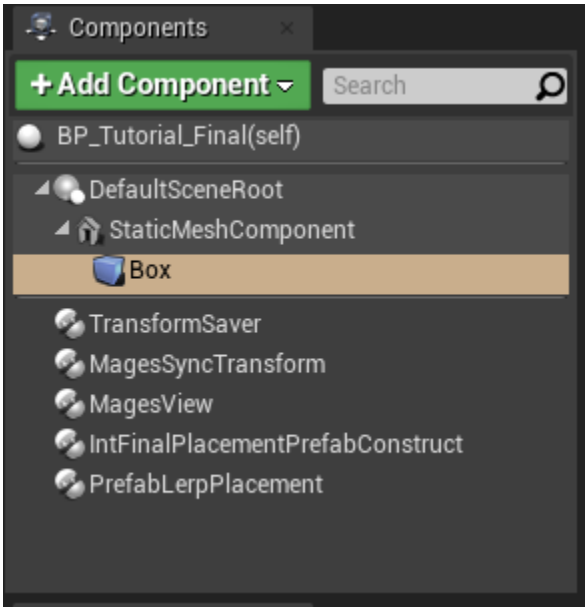
The Components panel should now look like this:



Using the "Add Component" button, we'll add the following components:

- Int Final Placement Prefab Construct
- Prefab Lerp Placement

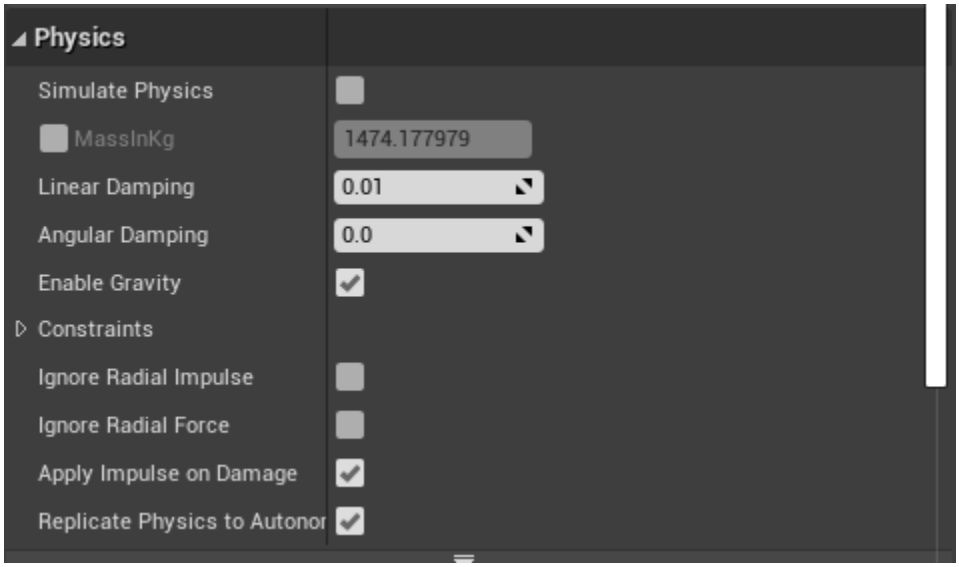
Additionally, we'll add a "Box Collision" component as a child of the "StaticMeshComponent":



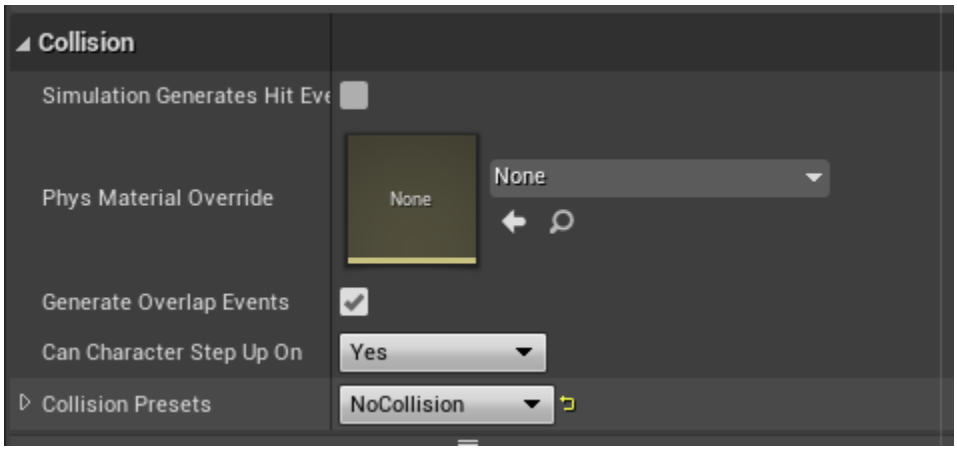
Select the "Box" component, and in the Details panel on the right, set its Scale property to `(X=4.125000,Y=4.125000,Z=4.125000)`

Next up, we have to configure the "StaticMeshComponent" so that it does not simulate physics, or collider with anything else:

Select the "StaticMeshComponent" in the Components panel, and in the details panel, find the section named: "Physics". Make sure that "Simulate Physics" is unchecked:



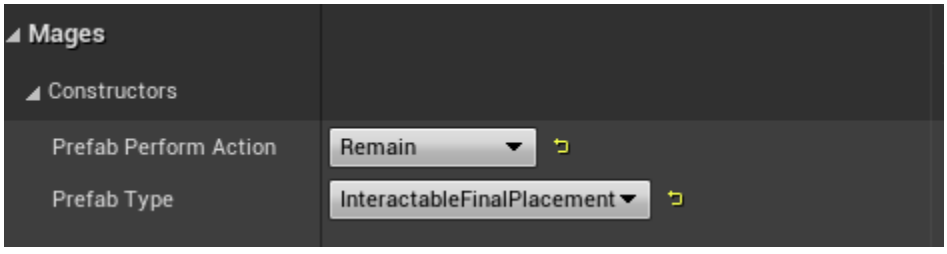
Just underneath this section, there is the "Collisions" section. Set the "Collision Presets" property to "NoCollision".



Now, we need to configure the other two components we added so that this Blueprint works like a Final Blueprint.

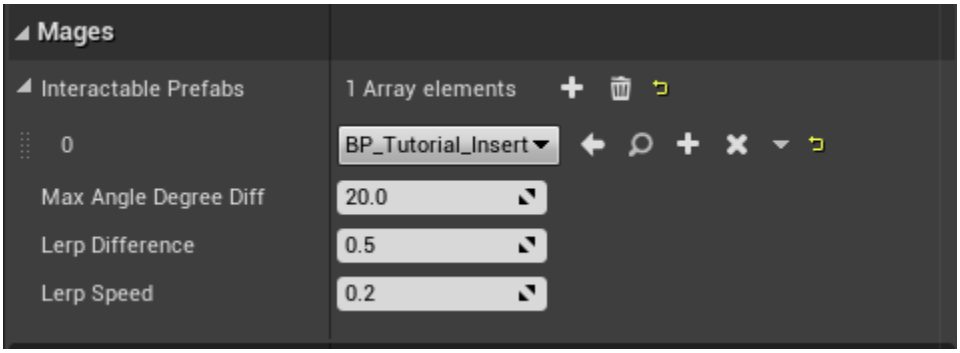
Select the "IntFinalPlacementPrefabConstruct" in the Components panel, and in the details panel set the following properties (under Mages > Constructors):

- Prefab Perform Action: Remain
- Prefab Type: InteractableFinalPlacement



Almost there for the Final Blueprint! Select the "PrefabLerpPlacement" component. In the Details panel, find the property named "Interactable Prefabs", and press the '+' button next to it.

This allows you to declare which blueprints may be placed into this Final Blueprint; we'll use just one for this action, namely "BP_Tutorial_Insert":

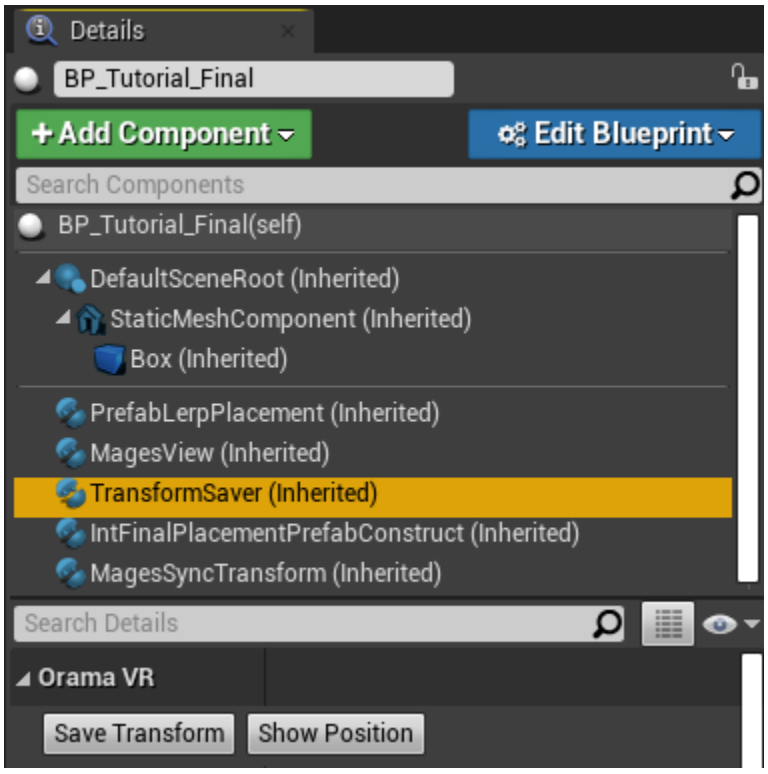


Afterwards, press "Compile" and "Save" in the toolbar.

We've finished setting up the Final Blueprint, but remember, we cloned it from "BP_Tutorial_Insert". This means, that its transform will be the exact same as the Insert Blueprint's transform. Let's change that:

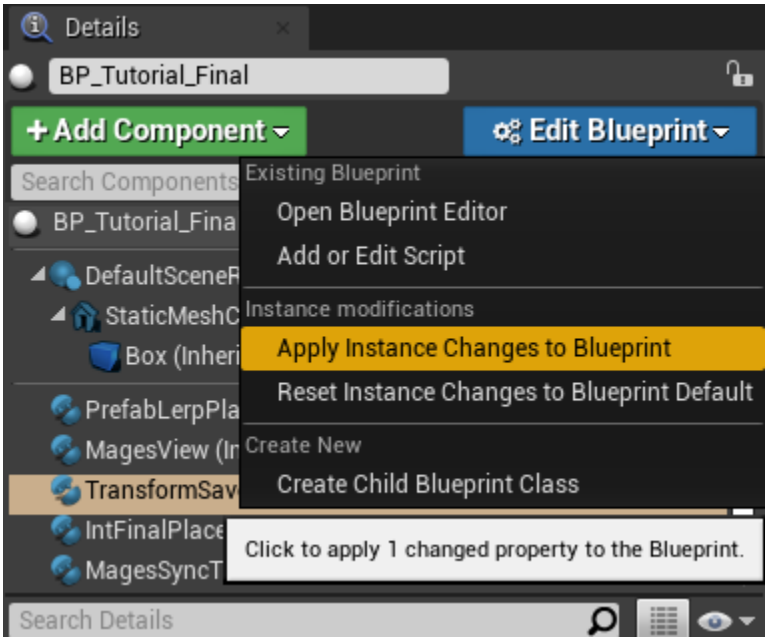
In the Content Browser, find the "BP_Insert_Final" Blueprint and drag it into the level. You'll see that it snaps to the exact position as the Insertion Blueprint.

Let's move it to the right of Insert Blueprint, specifically to `(X=57.065460,Y=-245.366364,Z=131.245087)`. We need to save its Transform now, so as before, select the TransformSaver in the Details panel, and press "Save Transform":

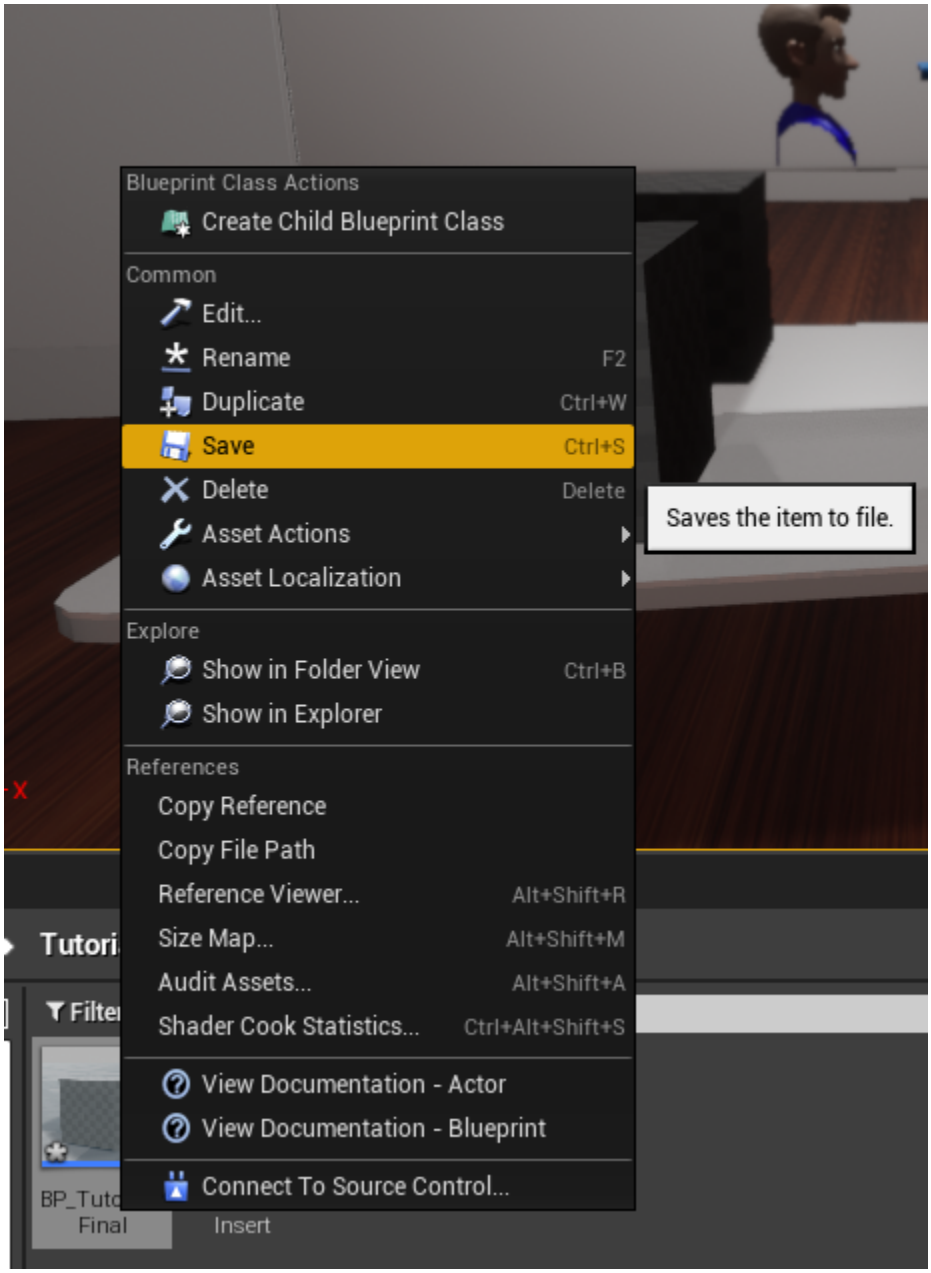


Additionally, we'll apply the instance's changes to the blueprint; to propagate the transform value:

Click "Edit Blueprint" and choose "Apply Instance Changes to Blueprint":



In the Content Browser, you will see that your blueprint has been marked with an asterisk icon. This means that it has unsaved changes; right-click on it and choose “Save”:

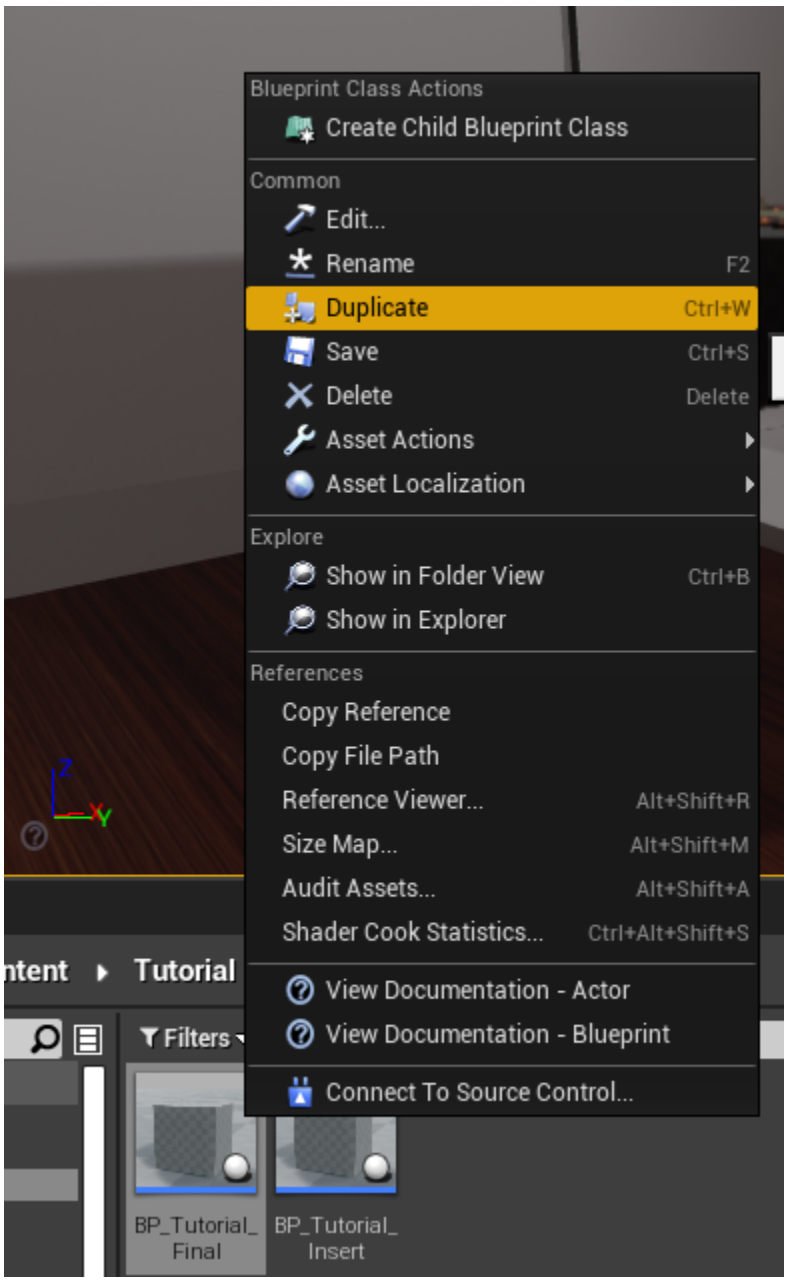


We are done creating the Final Blueprint. Now this action would work correctly, but how is the user supposed to know where to place the cube? Remember; the Final Blueprint will be invisible until the action has been completed. In real-world scenarios, context would help, but a hologram blueprint would be best. So let's create it.

Creating the Hologram Blueprint

In the same manner as with how we created the Final Blueprint, we will duplicate the Final Blueprint, and make the necessary changes to turn it into a Hologram.

In the Content Browser, find the Final Blueprint ("BP_Tutorial_Final" in our case), right-click on it, and choose "Duplicate":



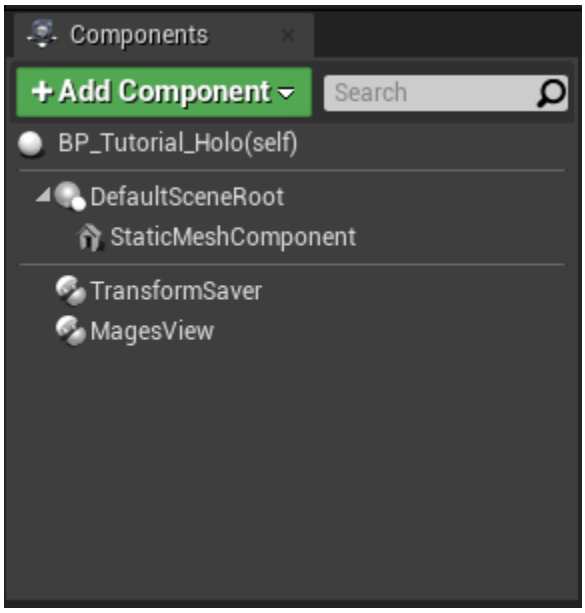
Name it accordingly; we chose to name it "BP_Tutorial_Holo", and afterwards, open the blueprint.

In the Components panel, find and delete the following items:

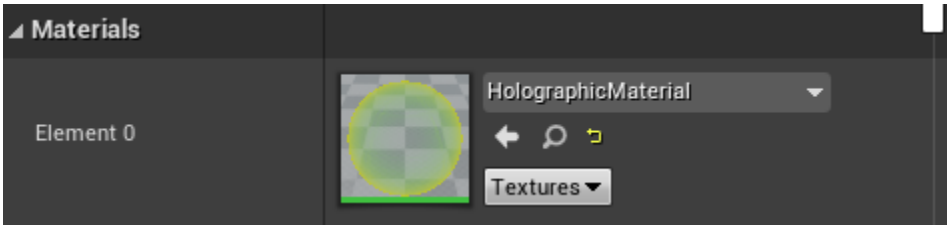
- MagesSyncTransform
- IntFinalPlacementPrefabConstruct

- PrefabLerpPlacement
- Box

You should have something like the following afterwards:



Now, we need to change the material of the cube, so that it looks like a hologram. Select the "StaticMeshComponent" and in the Details panel on the right, find the section titled "Materials". Change the property "Element 0" to the "HolographicMaterial":



Finally, hit "Compile" and "Save" to save the changes, and close the editor.

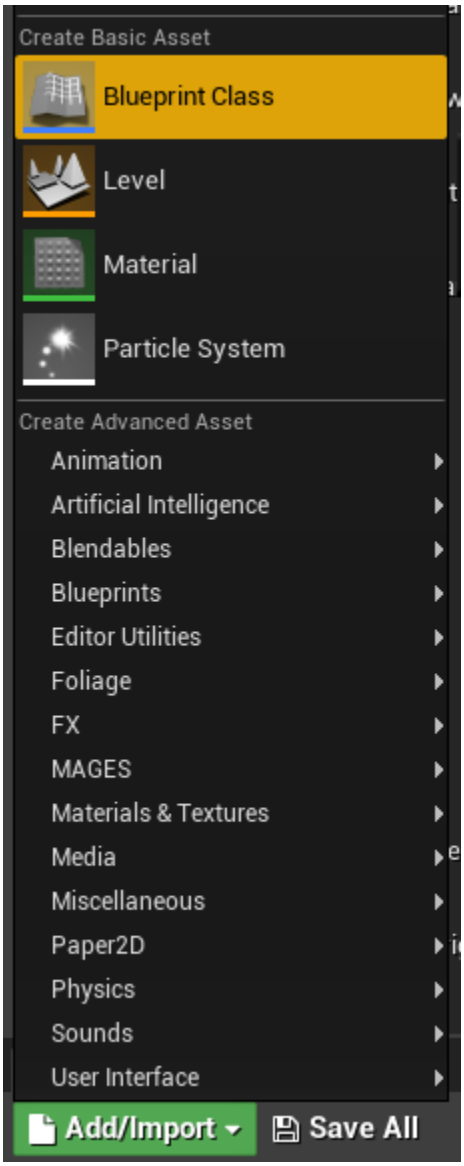
We've finished creating the Blueprints required for the Insert Action. Now we need to create the Insert Action Blueprint, and use the blueprints we've created.

In the Level, delete the following actors (they will be spawned when the action takes place):

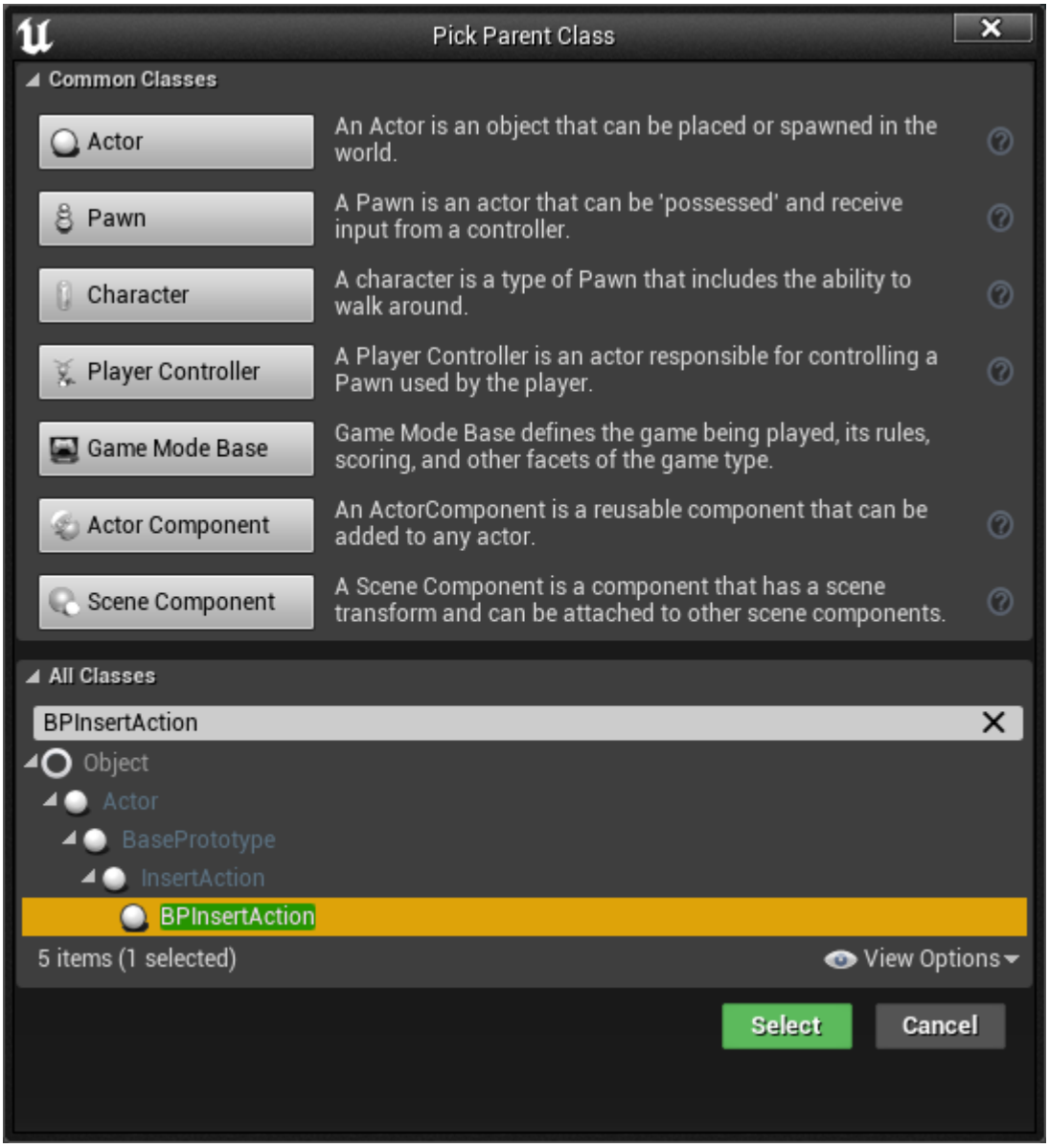
- Insert Blueprint "BP_Tutorial_Insert"
- Final Blueprint "BP_Tutorial_Final"

Creating the Insert Action

In the Content Browser, press “Add/Import” and choose “Blueprint Class”:



In the dialog, expand the “All Classes” section, and search for “BPInsertAction”, select it and press “Select”:

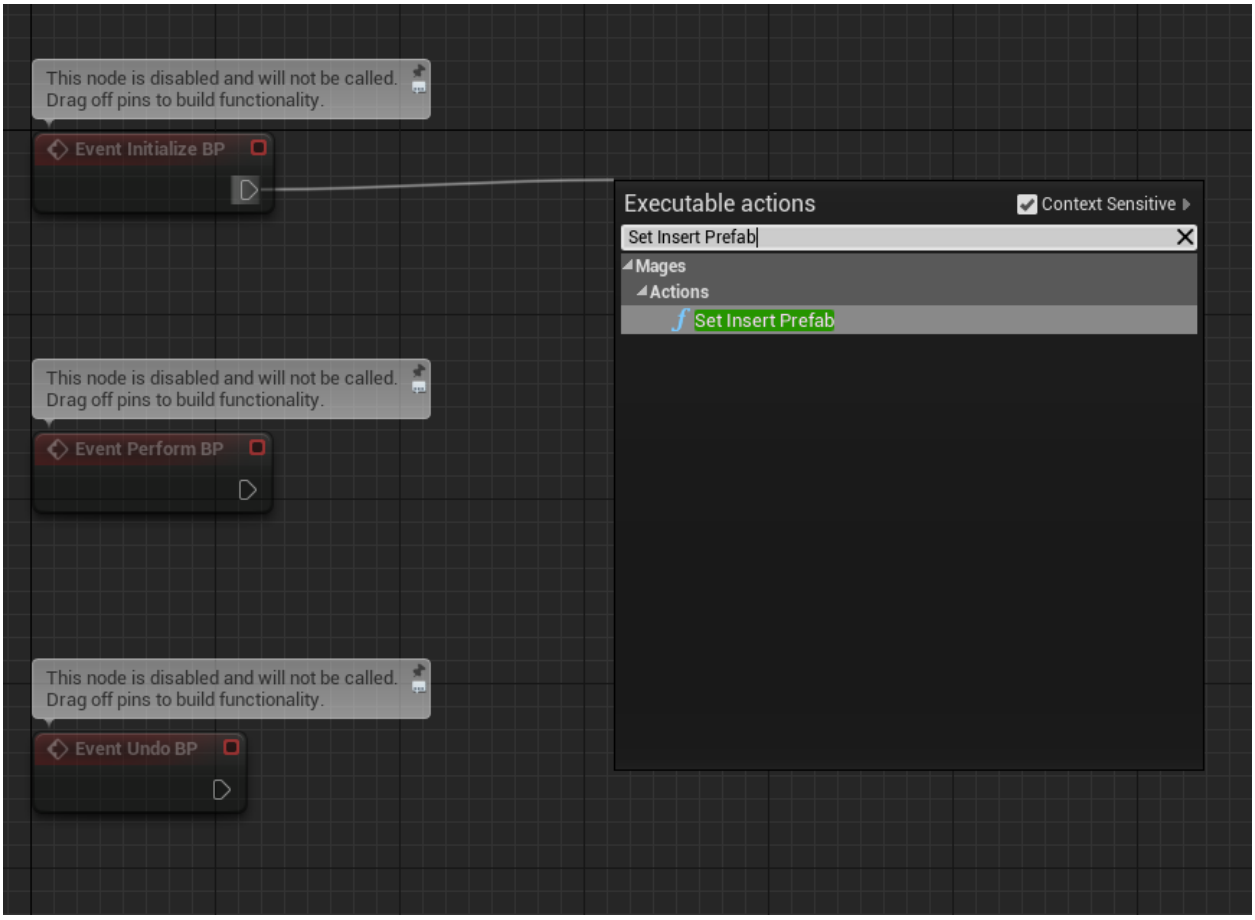


Name the new blueprint accordingly (we chose “TutorialInsertAction”), and open it.

In the Event Graph tab, there are three nodes:

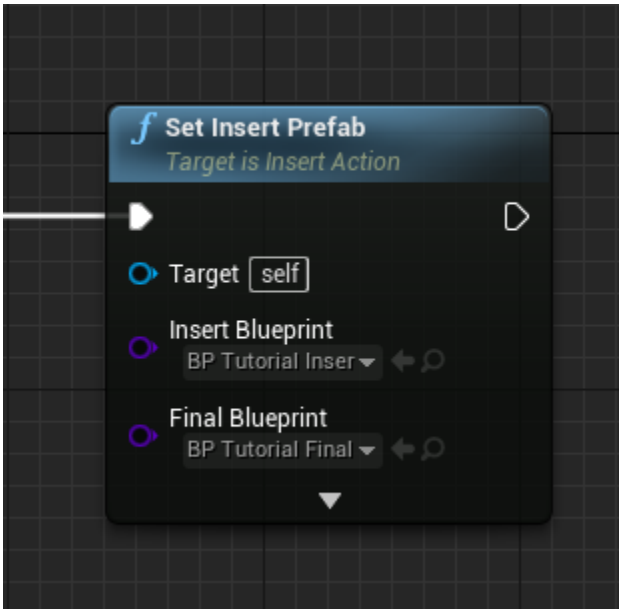
- Event Initialize BP
- Event Perform BP
- Event Undo BP

Drag a connection from the “Initialize BP” node - a popup will appear with all of the available nodes. Search for “Set Insert Prefab” and hit Enter:



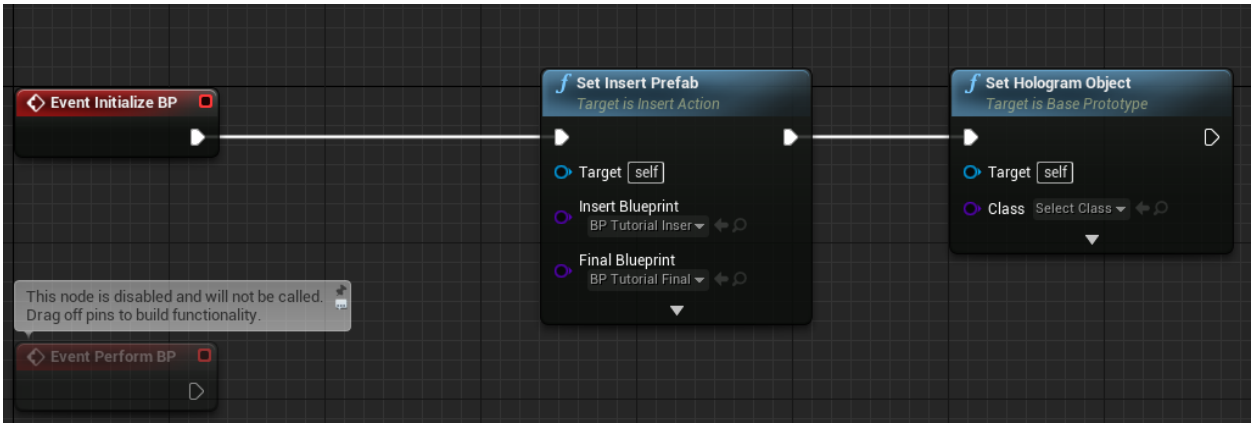
A new node will be created and connected to the "Initialize BP" node. With this node, we can declare which blueprints will be used for the Insert Action.

Set the Insert and Final Blueprint parameters to the blueprint's we created earlier. It should look something like the following image:

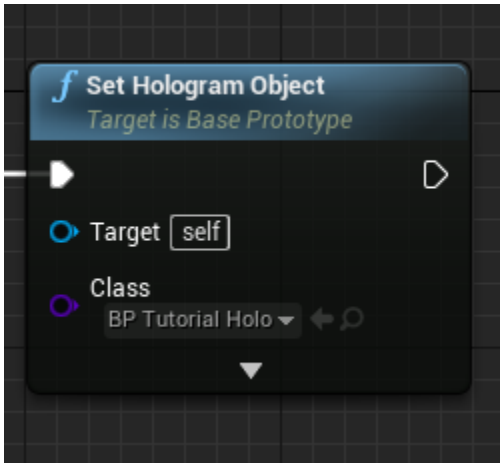


Next up, let's also add our hologram blueprint. Drag a connection from the output pin of the "Set Insert Prefab" node, search for "Set Hologram Object" and press enter.

You should now have something like this:



Set the "Class" property to the hologram we created for the action ("BP_Tutorial_Holo"):

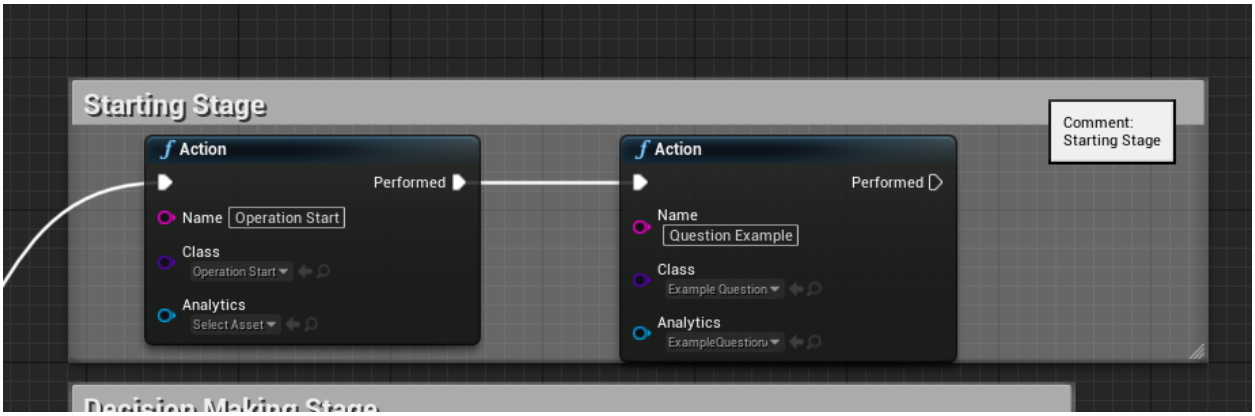


Done! Hit "Compile" and "Save", and you can close the blueprint afterwards. Now, all we need to do is to add it to our Scene Graph.

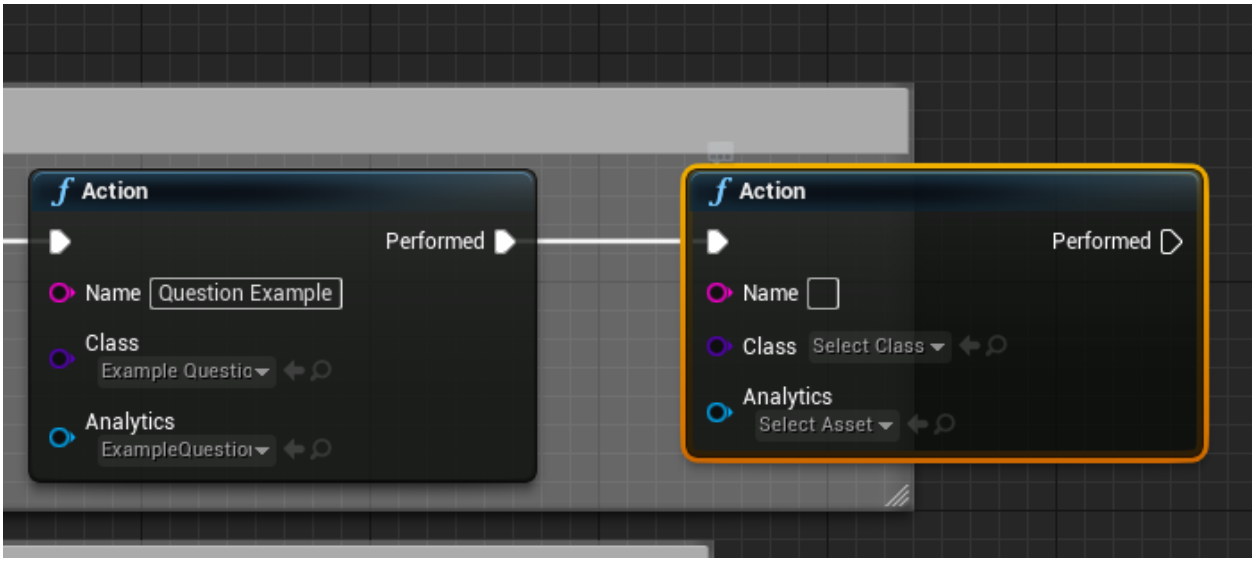
Adding the New Action to the Scene Graph

Open the blueprint named "SceneGraph_SampleApp". You can find it in the following path inside the Content Browser: "MAGES_SDK/Content/MAGES/Operation/ActionBlueprints/".

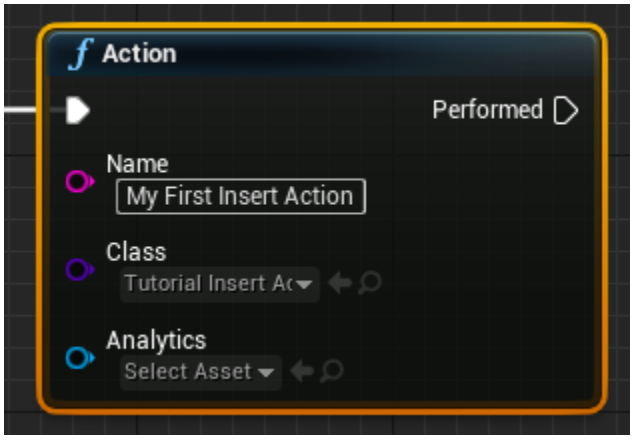
In the Event Graph tab, navigate to the segment marked "Starting Stage". We'll add our action right after the Question Action of the Sample App Scene Graph.



Drag a connection from the "Performed" pin of the "Question Example" node, search for "Action" and press Enter:



Se the "Name" property to something memorable, like "My First Insert Action", and the "Class" proerty to the name of the action blueprint we last created ("TutorialInsertAction"):



Hit "Compile" and "Save"; you can close the blueprint editor afterwards.

We've added the action to the scene graph, and now we're ready to test it out! By default, the Desktop 3D controller is used, so let's look at how it works.

2DoF Controller Tutorial

The 2DoF controller is used to run a MAGES application in Desktop 3D mode, without the need of any HMD or controllers.

In this mode, the character is controlled through the keyboard and/or mouse/trackpad.

In Desktop 3D mode, the gameobject in the Unity hierarchy, which represents the camera is called 2DoF-Camera.

A UI is available in this mode, by pressing the space bar. This will open the menu of 2DoF which gives the user the ability to move, rotate, lock the hands. Below, you can see the 2DoF menu on action.

In the following table, a detailed explanation can be found, regarding each button of the 2DoF Controller.

Button	Explanation
W or Up Arrow	Move forwards
S or Back Arrow	Move backwards
A or Left Arrow	Move left
D or Right Arrow	Move right
Q	Move downwards
E	Move upwards
F1	Toggle Body move mode
F2	Toggle Right Hand move mode
F3	Toggle Left Hand move mode
Tab	(Usually followed after pressing F1 or F2), enables the hand rotation mode with the use of mouse/trackpad
G	(Only after F1), enables/disables the cursor and freezes the camera rotation.
Left Mouse/Trackpad Click	Hand trigger on the hand targeted by F2/F3
Right Mouse/Trackpad Click	THand grip on the hand targeted by F2/F3
Ctrl	Switch hand plane movement (when F2 or F3 enabled)
Ctrl and Left Click or Right Click	Sends the Left or Right hand to the direction of the crosshair

You can utilize the Desktop 3D camera controller in order to run and execute the first insert action you created before. To do so, please follow the instructions below:

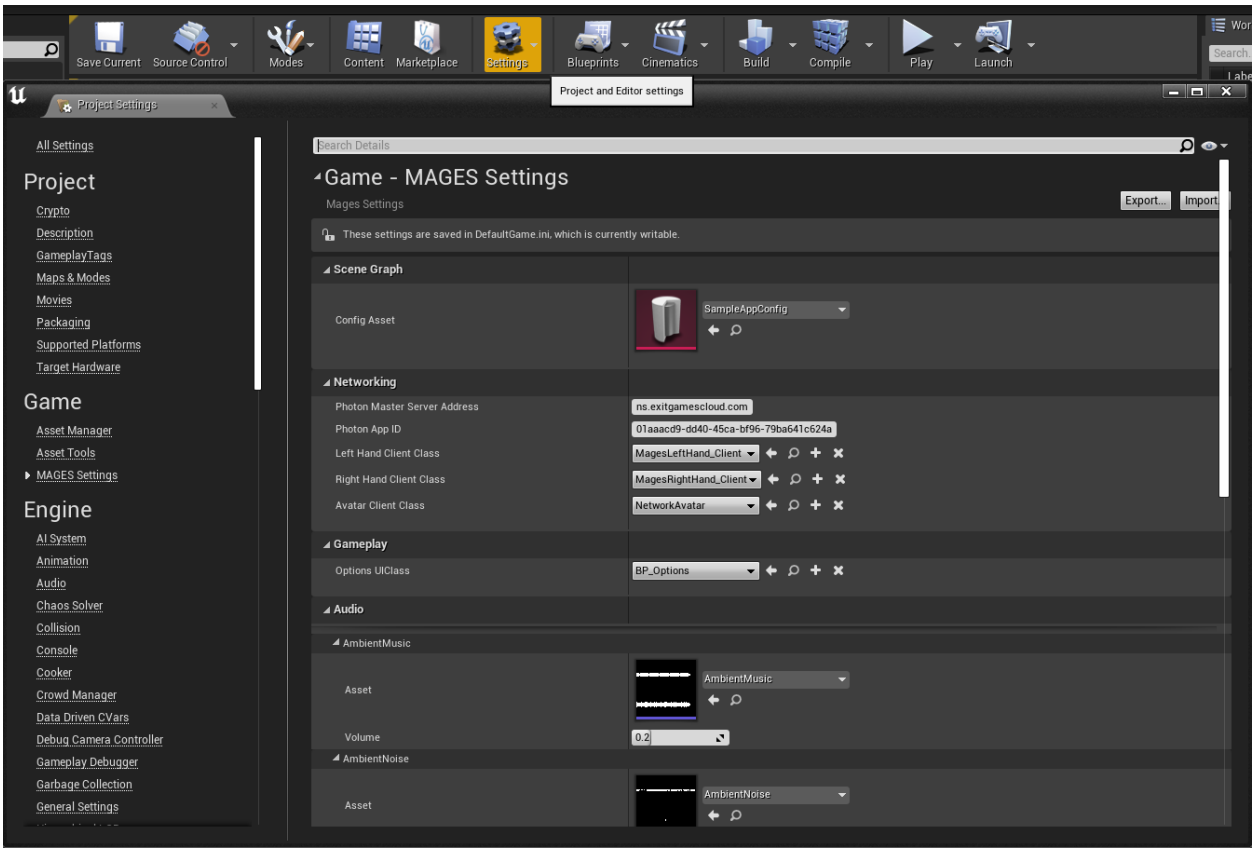
1. Ensure that the Sample App scene is open, and click the “Play” button in the level editor
2. You will be presented with a menu like the following. This is the **Operation Start** menu.
3. There are two ways to continue in this point. Either by pointing the crosshair **Single Player** button and clicking with the left mouse button, or by pressing the **X** button on the keyboard. It is recommended to continue with the latter for now.
4. Afterwards, the Question Action will appear, you can complete it by answering and pressing the “Submit” button, or by pressing “X” on your keyboard.

4. In this point, the “BP_Tutorial_Insert” we set up in the tutorial above, will be present in the level along with a hologram, atop the desk on the mirror to your left.
5. Move close to the cube using the W/A/S/D/E/Q buttons, and by moving the mouse to turn
6. Activate the virtual right hand in order to be able to pick up objects. To do so, press ‘2’ from the number row on the keyboard.
7. Using the mouse, move the hand towards the cube, and grab it using the left mouse button
8. Drag it towards the hologram to the right, and it will snap to that position

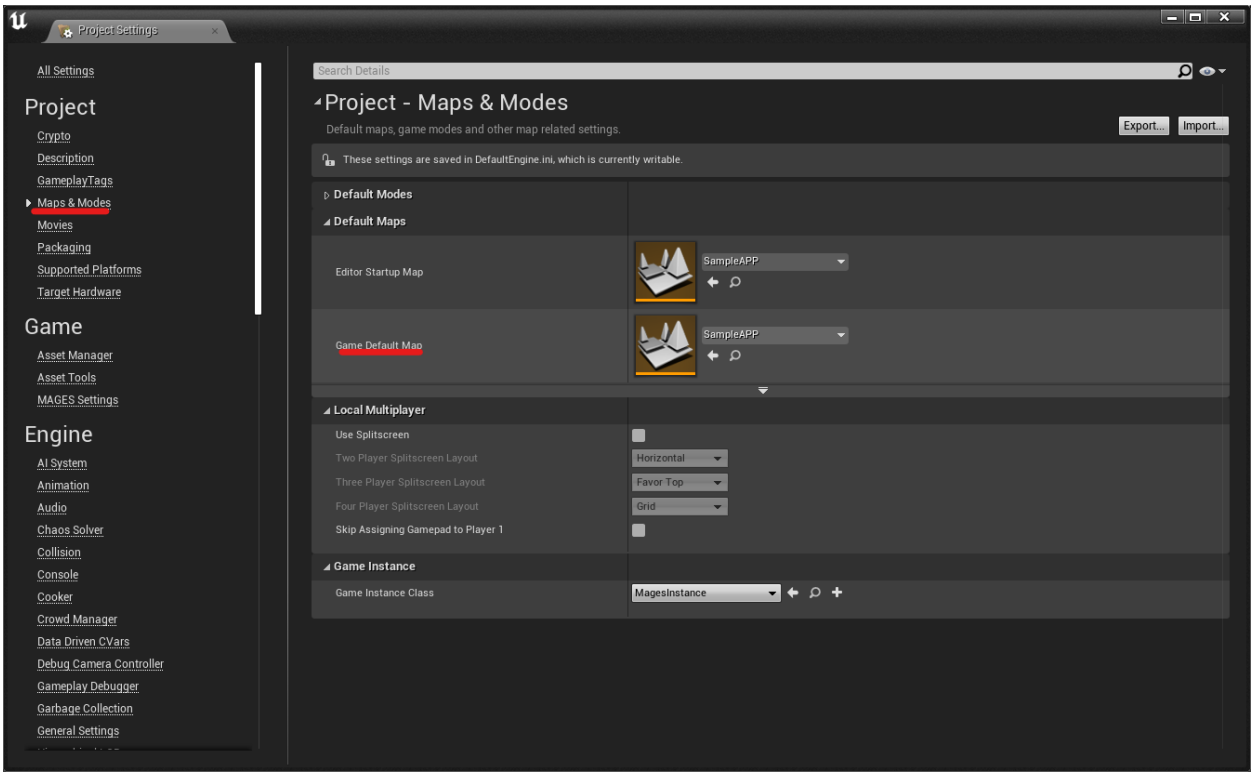
Building the Application

This section will demonstrate how to Package the project for the Windows platform, with your newly created Insert Action.

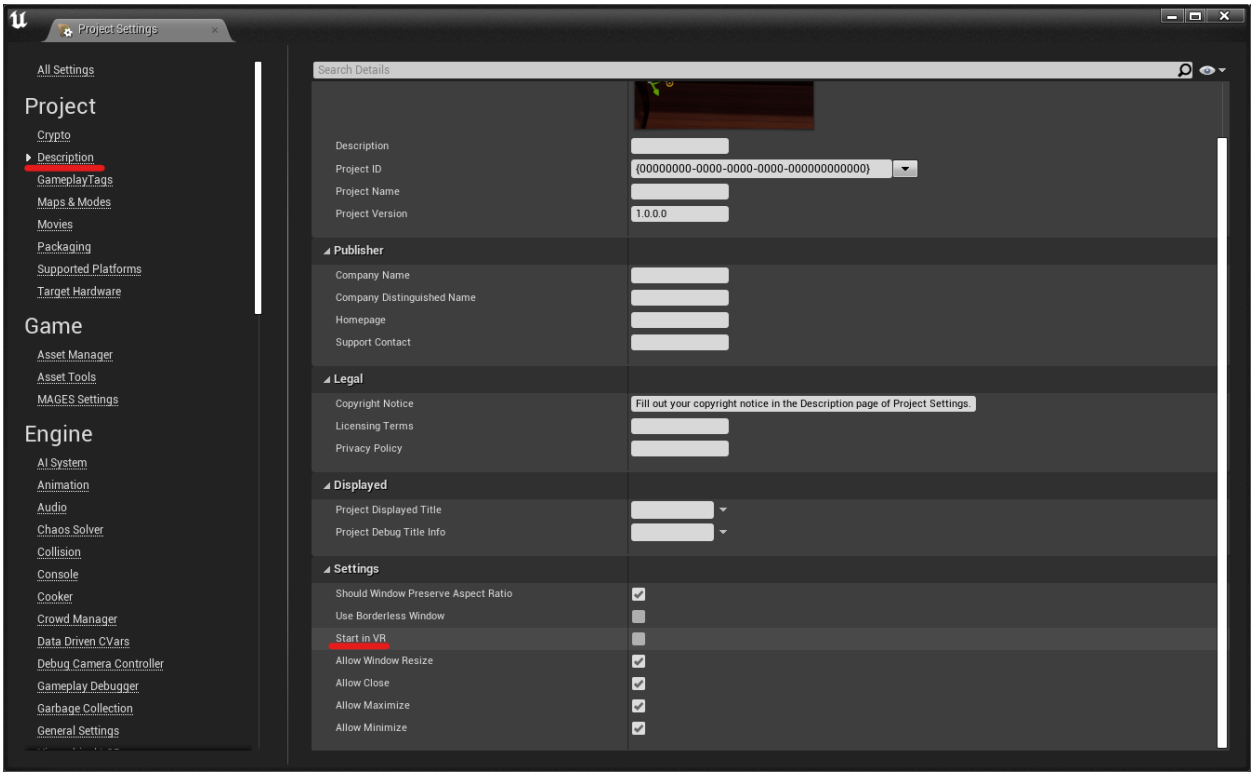
1. Open the Project Settings, and click the “MAGES Settings” section on the sidebar



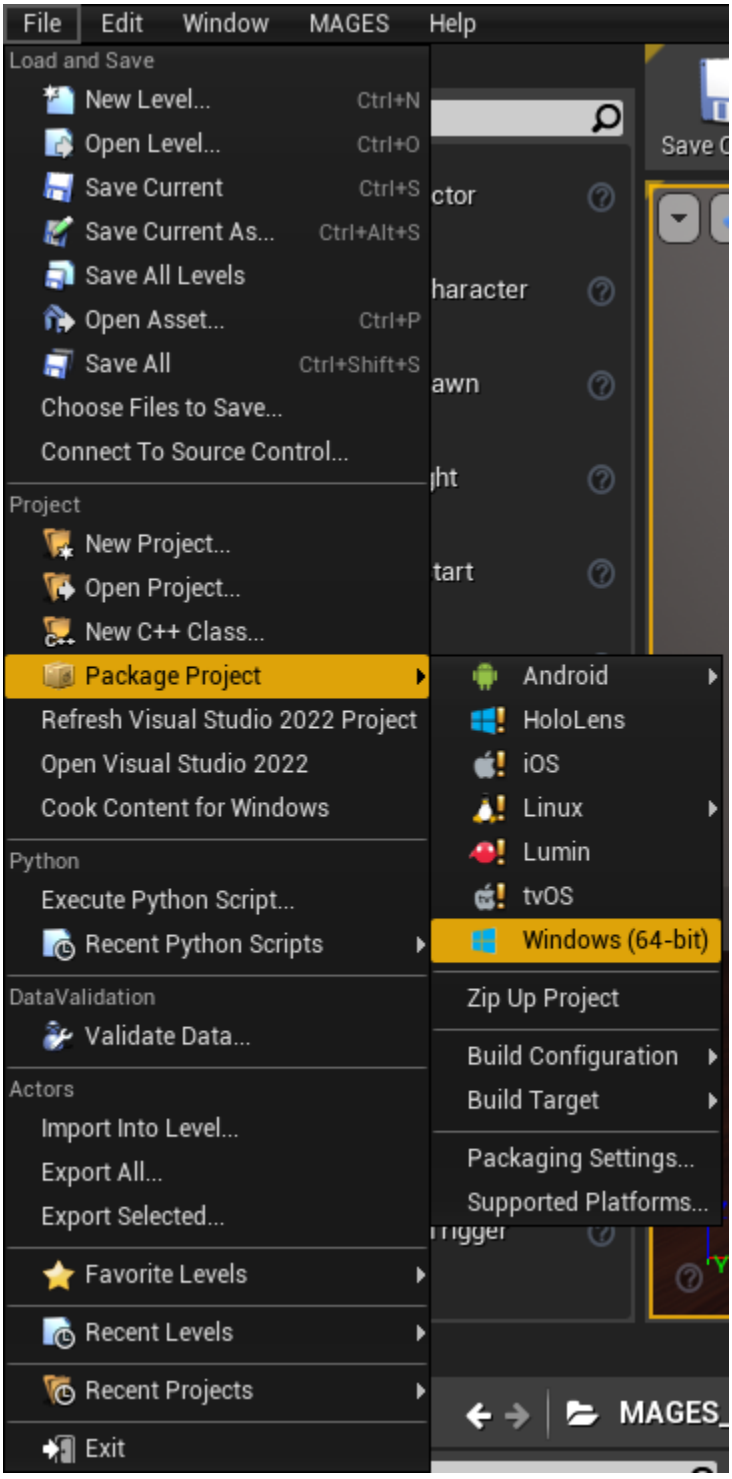
2. Ensure that the “Config Asset” property is set to “SampleAppConfig”, by clicking on the field, selecting “Clear”, and then resetting it to “SampleAppConfig” in the same manner
3. In the Maps & Modes section, ensure that the “Game Default Map” property is set to the “Sample App” map



4. In the “Description” section, ensure that “Start in VR” is disabled:



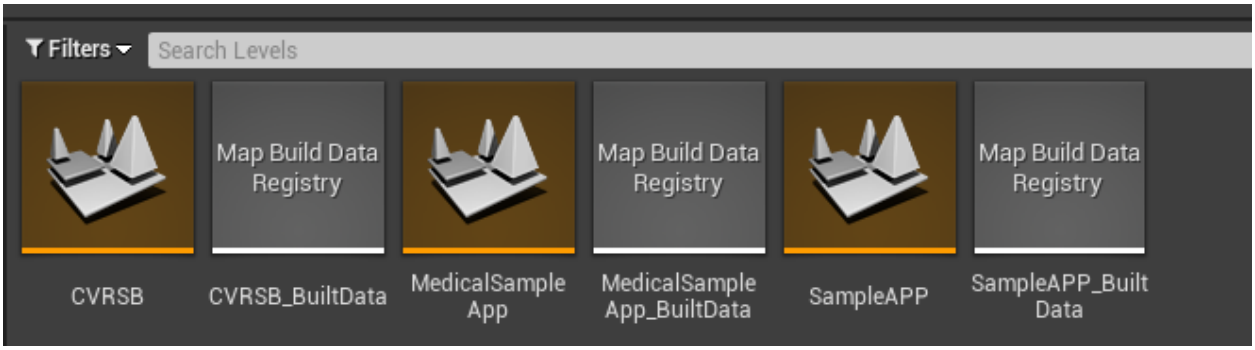
5. Close the Project Settings window, and in the level editor select “File > Package Project > Windows (64-bit)”



Note: For more information regarding packaging using the MAGES SDK and all of the supported platforms, please visit *Build Instructions*

9.1.4 Swapping to another level

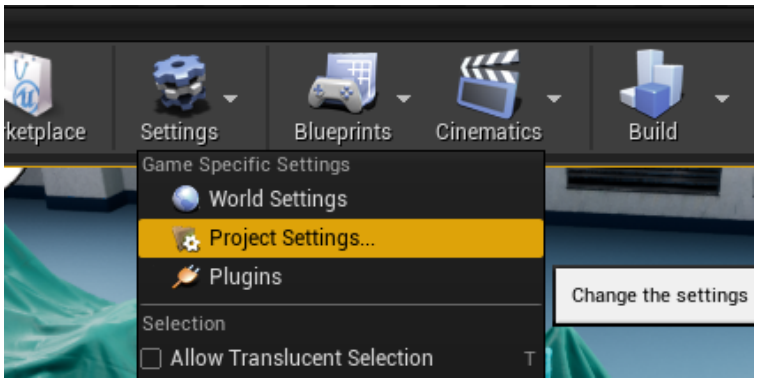
In case you want to try one of the other MAGES sample applications. First of all, navigate to *MAGES_SDK Content > MAGES > Operation > Levels* and there you will find all available MAGES levels.

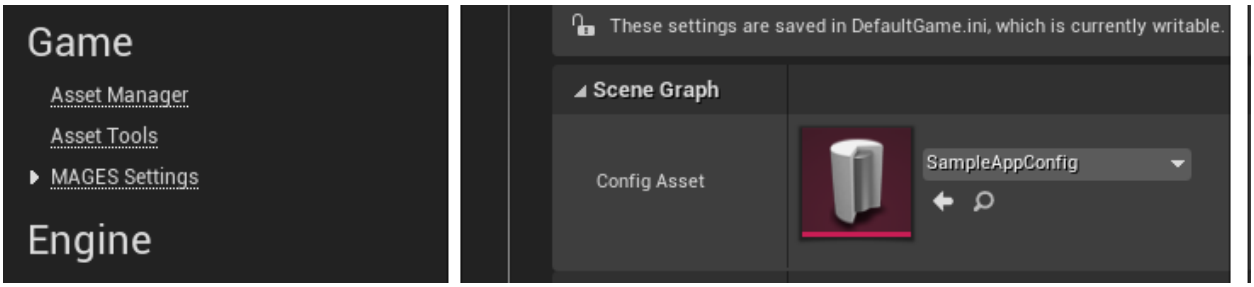


Double click the level file of the sample want to try out.

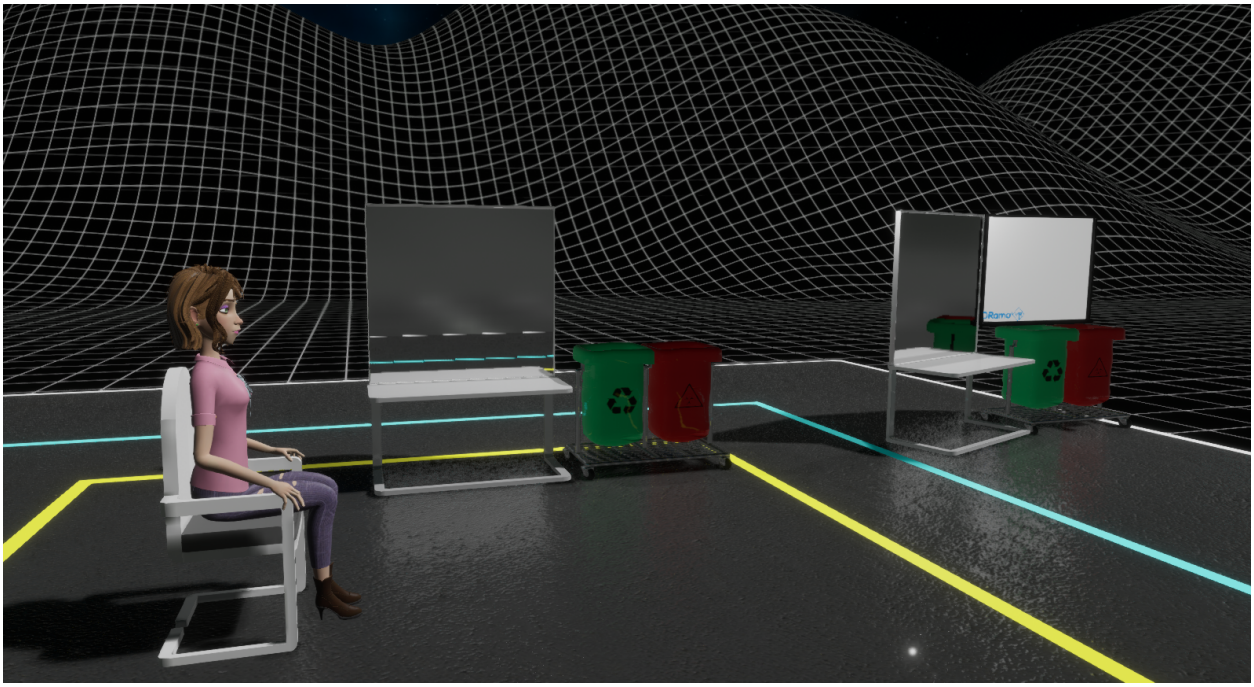
Note: All of the MAGES samples come with a 3D Desktop Pawn camera as default. In case you want to try them in VR follow the steps mentioned *here*

Once, you've set up your camera you need to change the scenegraph config asset. Navigate to your project settings, select MAGES settings on the left panel and change the config asset to the one corresponding to your level.



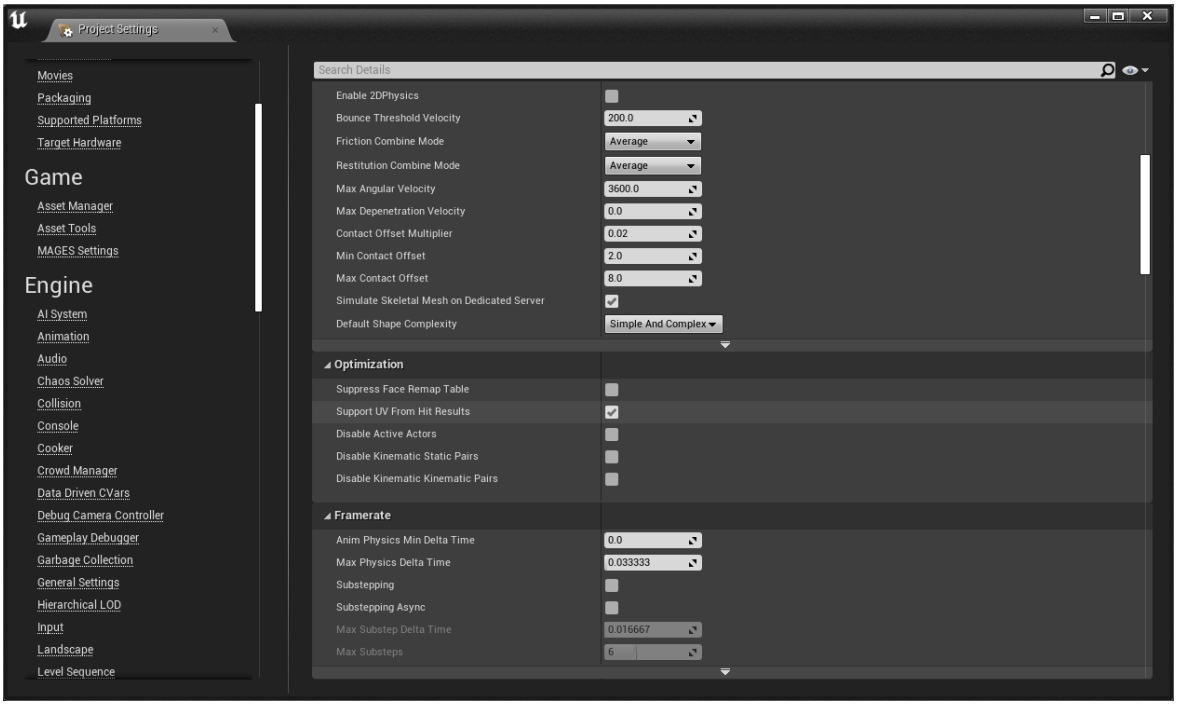


CVRSB Requirements



For the Early Access version of CVRSB specifically, the board will **not** work unless you do the following:

1. Open the Project Settings window
2. In the Physics Section, find the sub-section named “Optimization” and make sure that “Support UV From Hit Results is enabled”:

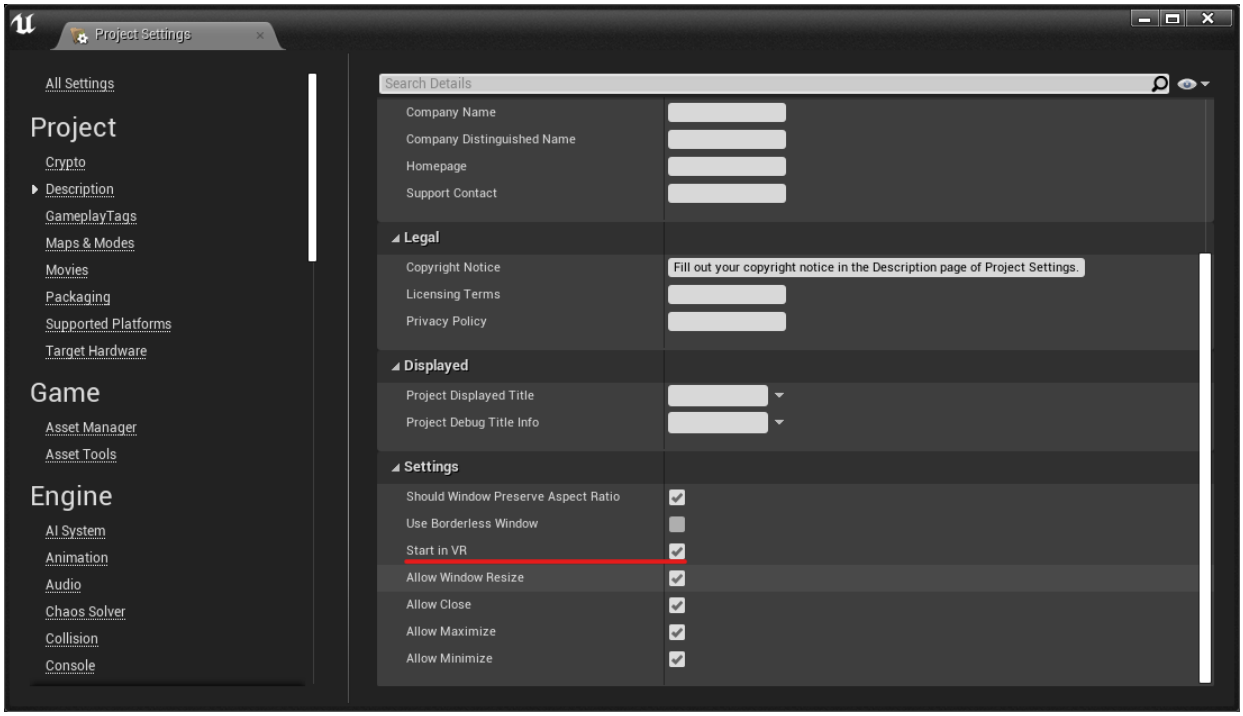


9.1.5 Build Instructions

In this section we describe the process of packaging a MAGES enabled project. To produce a packaged application, you will need to:

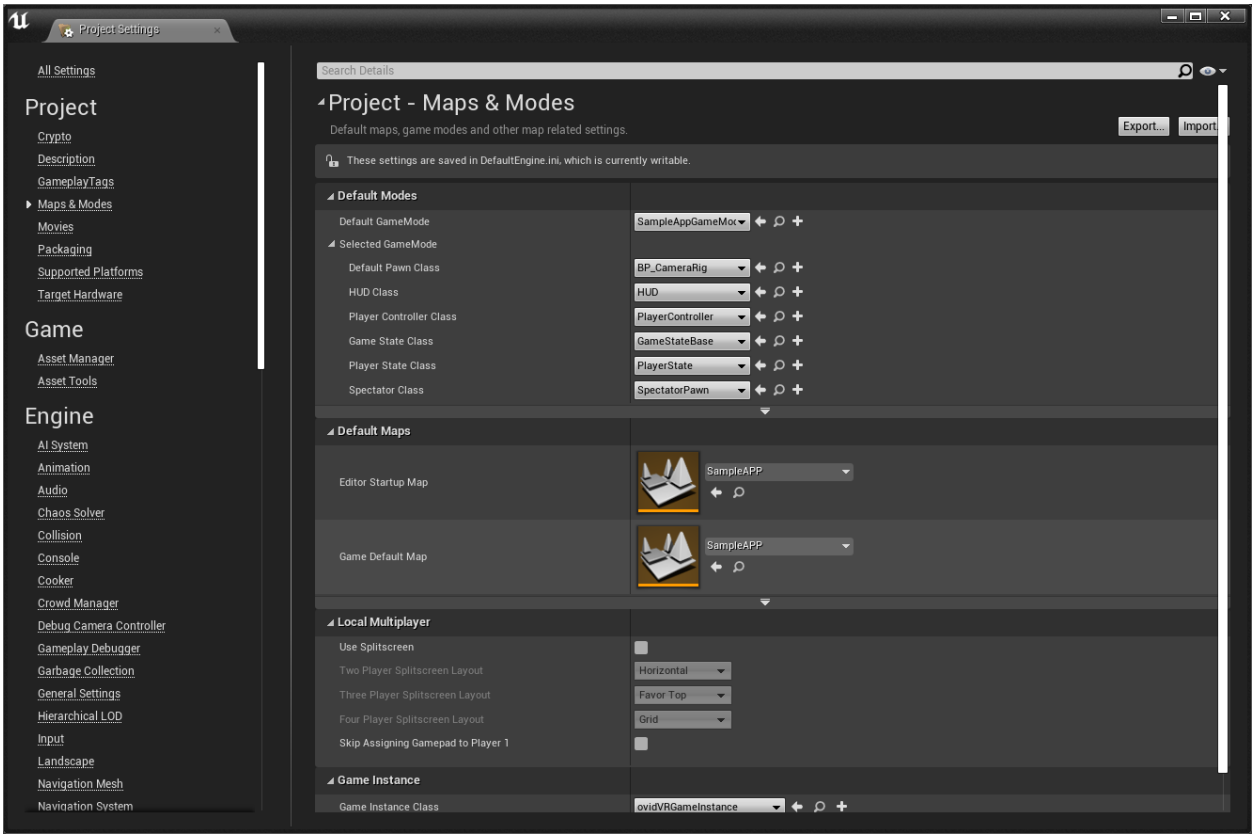
- Select the startup level
- Switch to the appropriate engine settings for your platform of choice

If you are building for VR, in the `Project/Description` section, make sure that "Start in VR" is enabled

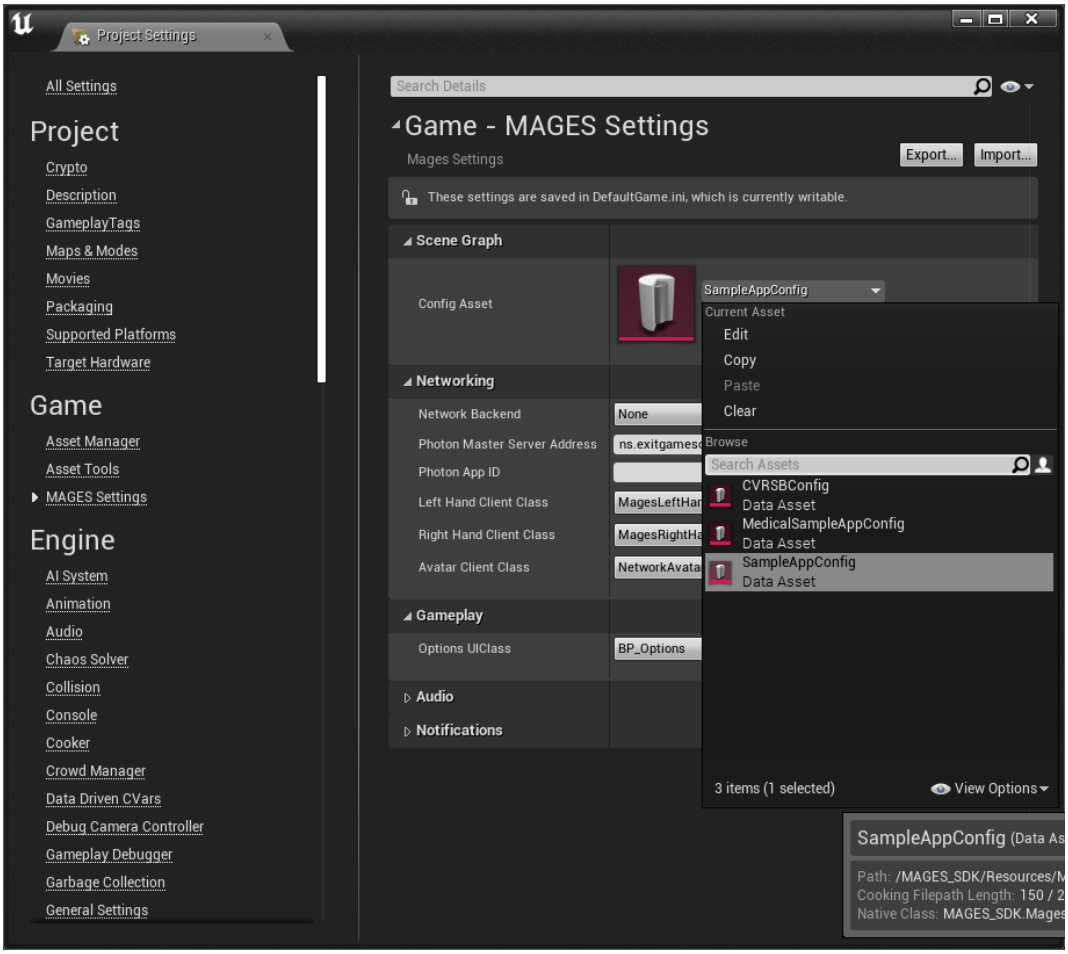


Maps & Modes Settings

In the `Maps & Modes` section, choose the application startup map through `Game Default Map`.

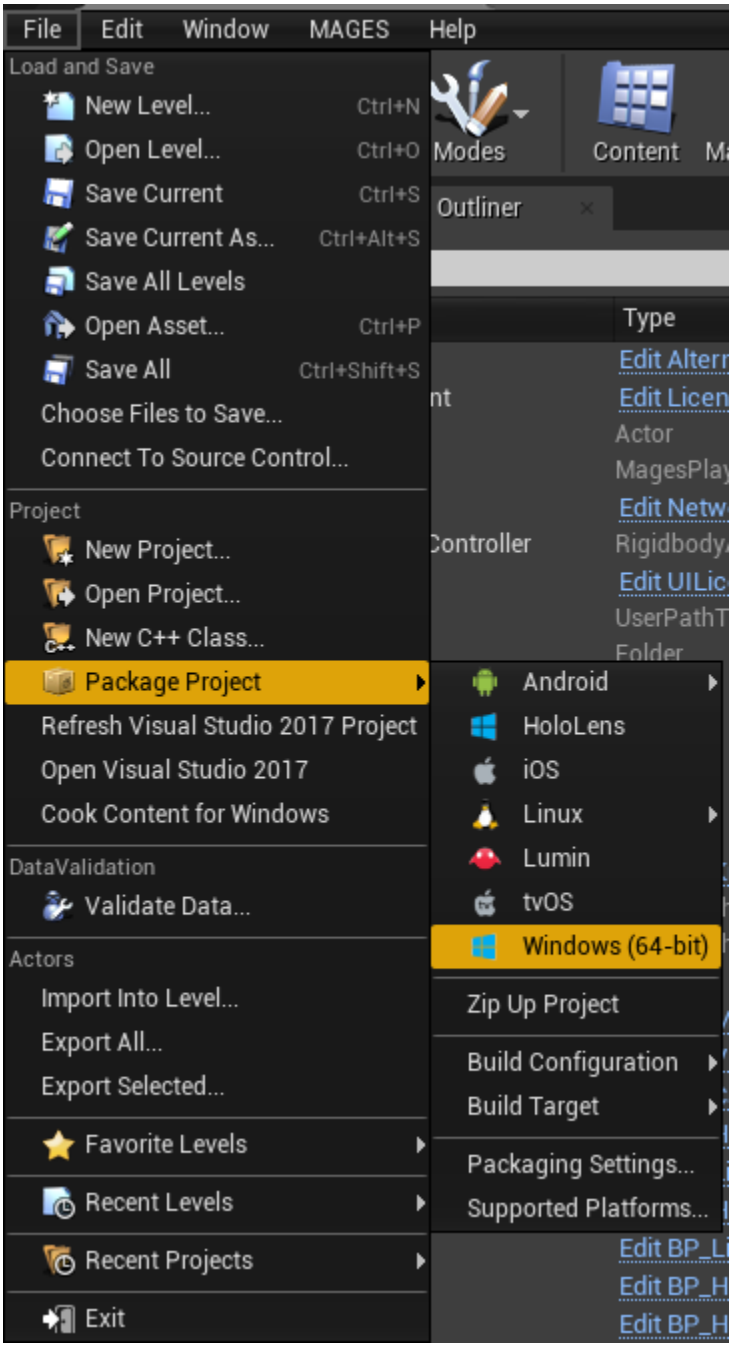


Note: It is recommended to clear and reset the configuration asset in the “MAGES Settings”:



Packaging - Windows

From the menu select “File”, choose “Package Project” and click on “Windows (64-bit)”.

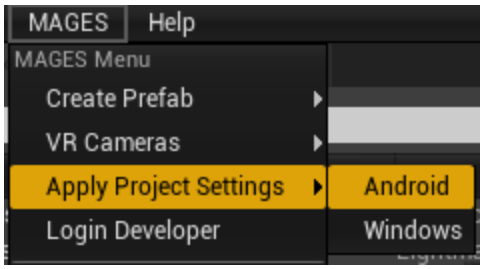


Packaging - Oculus Mobile

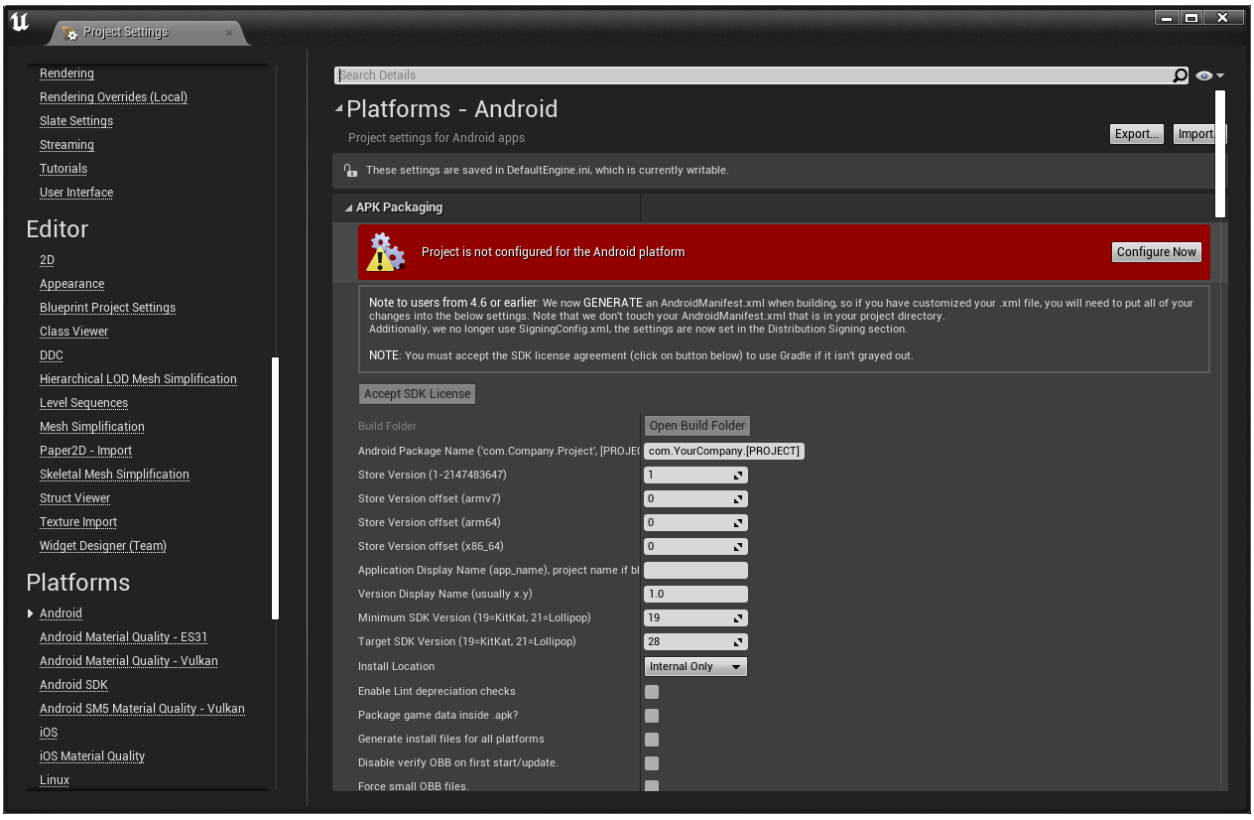
Note: Before building, make sure you've *set up the application startup map*

To get started, follow the Android setup from Unreal's documentation.

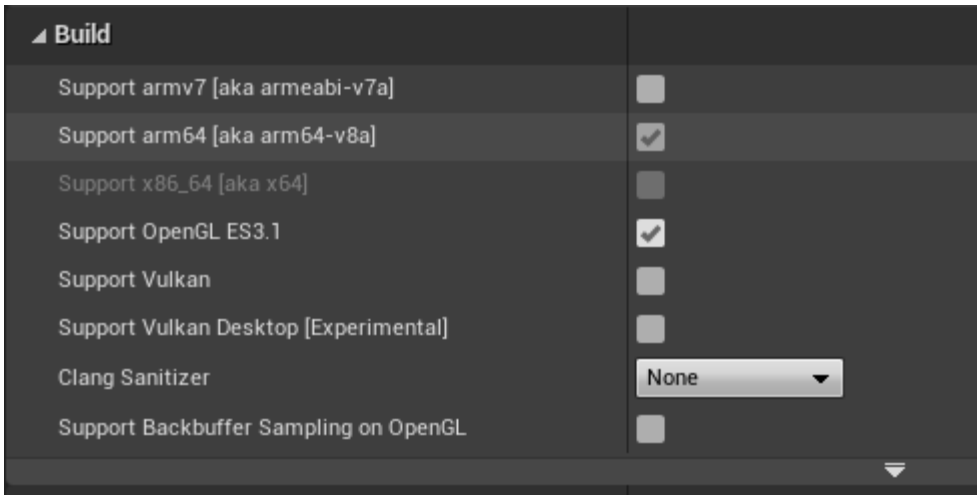
MAGES Unreal can setup some good default settings for exporting for Android (Oculus), and it is highly recommended to use them. On the menu bar, click on "MAGES", select "Apply Project Settings", and then click on "Android":



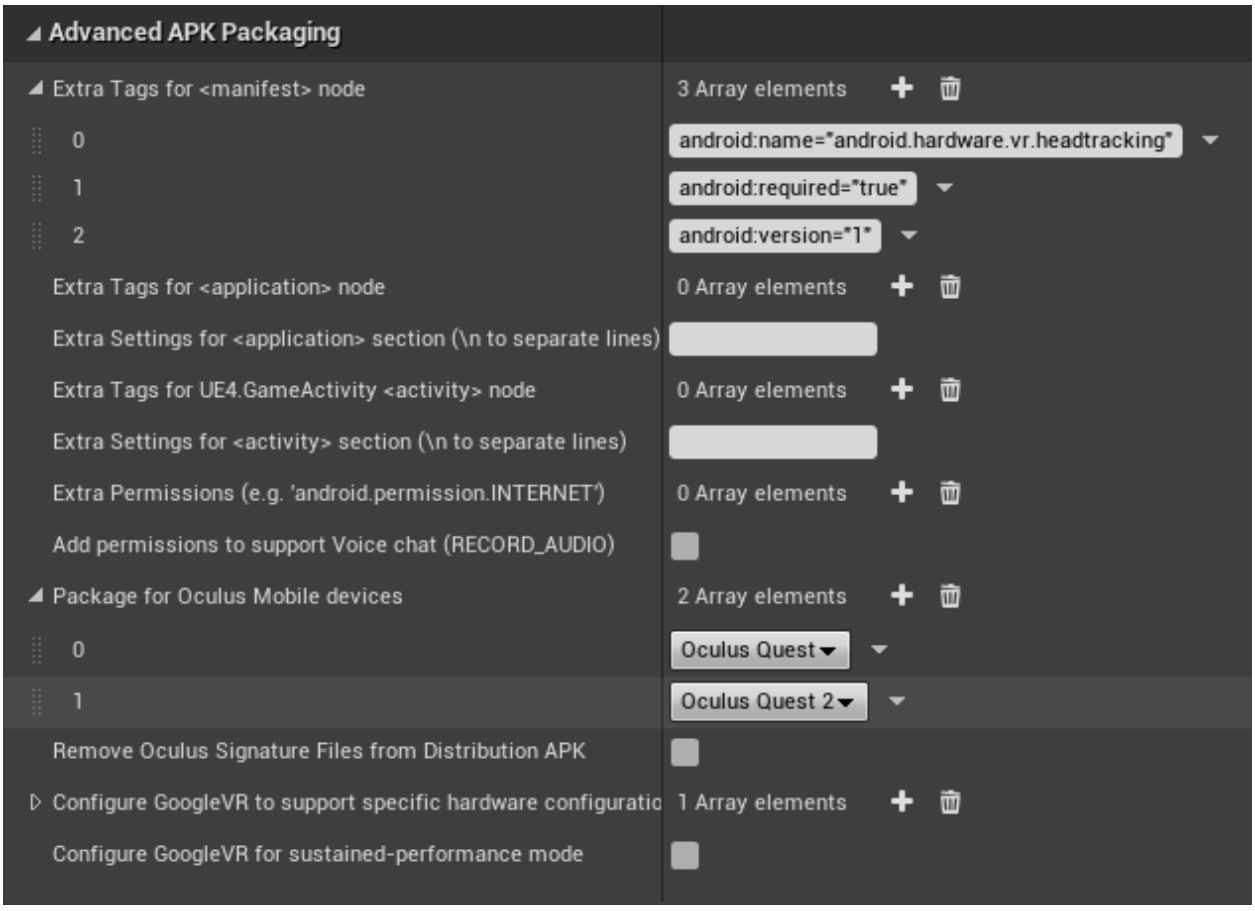
Afterwards, open the Project Settings, and inside the "Android" section under Platforms, press the Configure Now button, to enable packaging for android in the project



In the same section, scroll down to find the "Build" sub-section, and make sure that you are building for arm64:

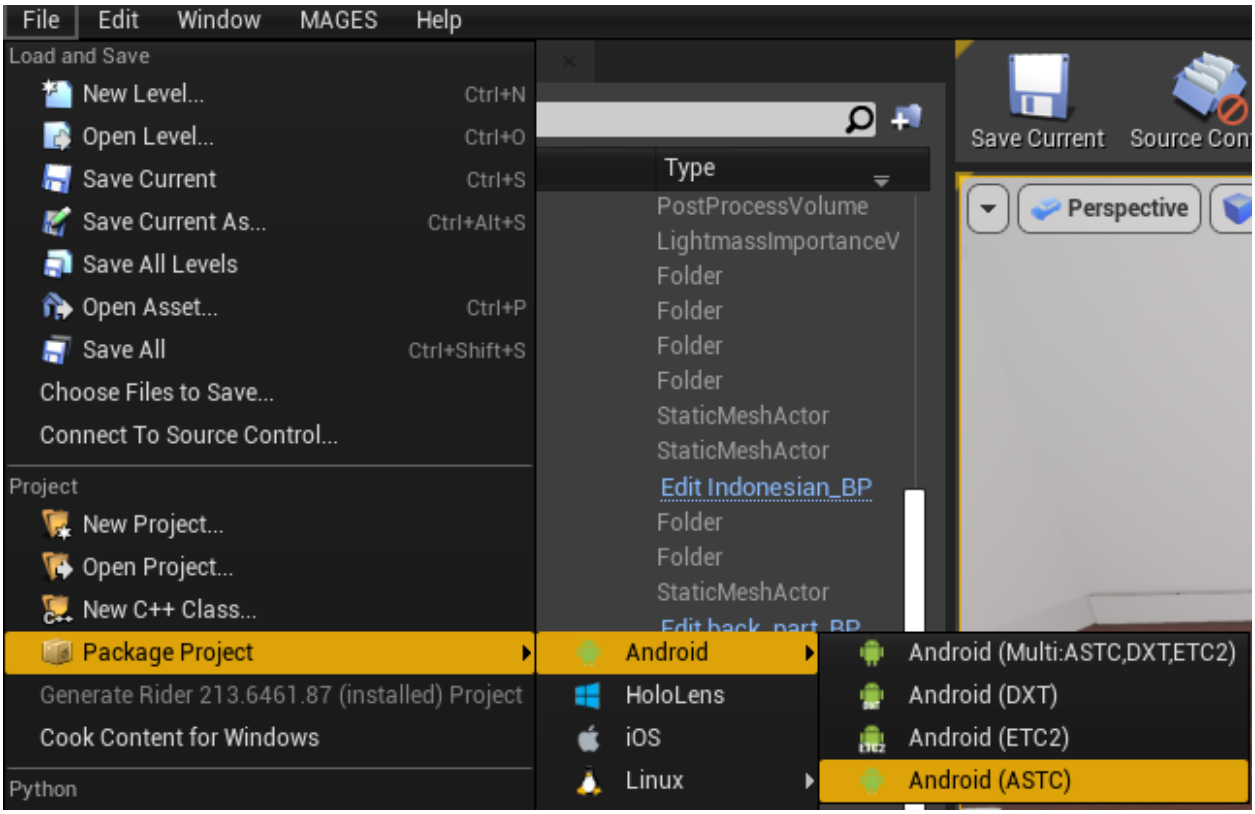


Last, scroll further down to find the "Advanced APK Packaging", clear the "Package for Oculus Mobile Devices" property, and add "Oculus Quest", and "Oculus Quest 2"



Package the Project

To package for Oculus Quest 1&2, click on File > Package Project > Android > `Platform of your choice`. Android (ASTC) is recommended by Oculus for building for their mobile devices.

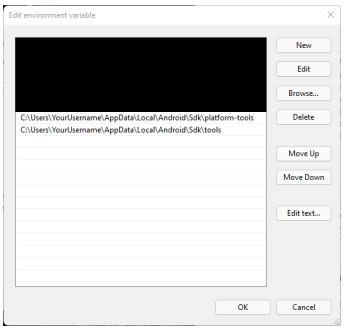


After the build has finished, go to your packaged project's folder, and run the "Install_[Your Project Name]-arm64.bat" batch file.

Common Issues

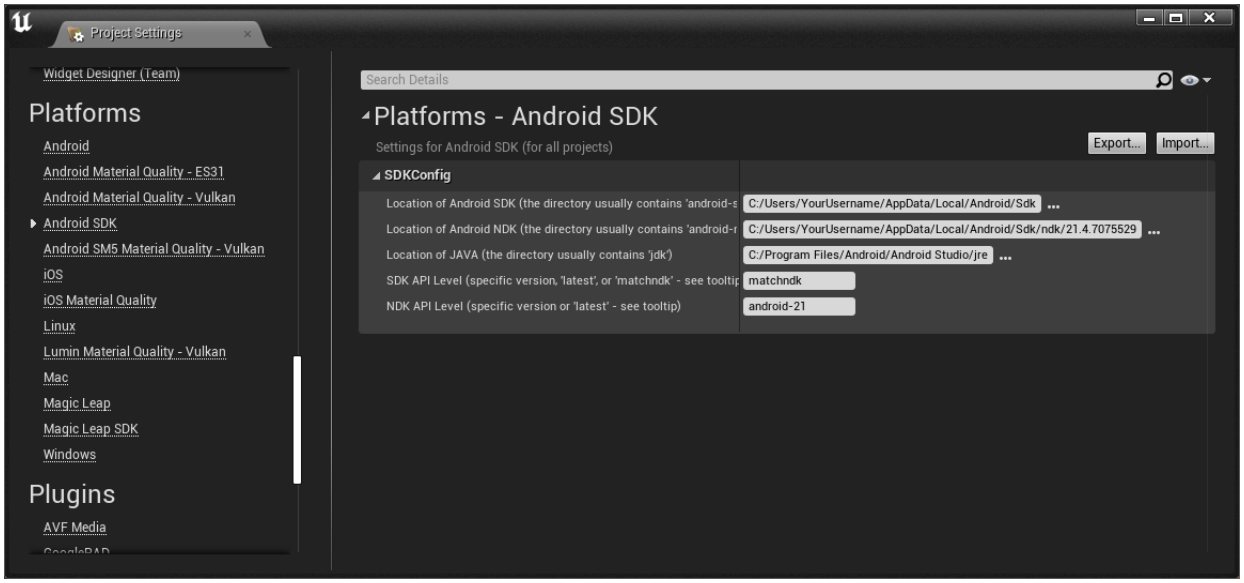
System cannot find ADB

When executing the "Install_[Your Project Name]-[arch].bat" file, the system may not be able to find "adb.exe". Make sure it is included in your PATH in the user's "Environment Variables":



Unreal Engine 4 Android build error

When pacakging, Unreal may complain that it cannot use the version of the Android NDK or SDK that you have installed. Make sure that your "Android SDK" settings (inside the Project Settings) are not empty values; Unreal has trouble finding your SDK installation otherwise:



Project starts in a non-VR windowed mode

Make sure that "Start in VR" under Project Settings > Description. Even if it is, set it and unset it, to ensure that Unreal has saved the value.

Additionally, if you're building for Android, redo *this step*

9.2 Supported Platforms

9.2.1 Windows

Please first make sure that you have a Windows 10 PC which is compatible with the latest VR headsets (HMDs – head mounted displays) that allow the unique feeling of "Presence" in the virtual world. These high-end VR HMDs are coupled with hand motion controllers (type of joysticks that enable unique embodied cognition with the affordances of gesture and manipulation in the virtual environment).

Windows platform fully supports both Desktop 3D as well as VR mode for the applications built with MAGES SDK.

9.2.2 Non-VR mode

Non-VR mode is available mainly for testing and development purposes by using the *Dekstop 3D Pawn*

9.2.3 VR mode (Supported Headsets)

MAGES Unreal supports the three most major plugins for VR support:

- OculusVR
- SteamVR, and
- OpenXR

You can find more about this *here*.

9.2.4 VR mode (Supported Headsets)

The VR headsets supported by MAGES SDK are listed below:

Headset	PC Minimum Requirements
Windows Mixed Reality	Download Windows Mixed Reality PC Check **Operating System** Windows 10 w/ Fall Creators Update (RS3). **Versions**: Home, Pro, Business, Education **CPU** Intel Core i5 4590 (4th gen) Intel Core i5 7200U (7th gen mobile) AMD Ryzen 5 1400 **RAM** 8GB DDR3 **GPU** Integrated Intel HD Graphics 620 DX12 integrated GPU (*note that this is not a separate graphics card, but it is part of specific Intel CPUs.* Check if your model is greater). NVIDIA MX150 discrete GPU NVIDIA 965M DX12-capable discrete GPU AMD Radeon RX 460/560 NVIDIA GTX 1060 **Graphics Display Port** HDMI 2.0 (or 1.4) DisplayPort 1.2 **Display** External or integrated VGA (800×600) display **USB** USB 3.0 Type-A **Bluetooth** Bluetooth 4.0 (for Motion Controllers)
Oculus Rift	Download Oculus Rift PC Check (SteamVR Performance Test) **Operating System** Windows 10 **Versions**: Home, Pro, Business, Education **CPU** Intel Core i3 6100 AMD Ryzen 3 1200 **RAM** 8GB DDR3 **GPU** AMD Radeon RX 470 (alternative AMD Radeon R9 290) NVIDIA GTX 1050Ti (alternative NVIDIA GTX 960) **Graphics Display Port** HDMI 1.3 **USB** 1 USB 3.0 Type-A, 2 USB 2.0
Oculus Rift S	Download Oculus Rift S PC Check (SteamVR Performance Test) **Operating System** Windows 10 **Versions:** Home, Pro, Business, Education **CPU** Intel Core i3-6100 AMD Ryzen 3 1200, FX4350 **RAM** 8GB DDR3 **GPU** NVIDIA GTX 960 AMD Radeon R9 290 **Graphics Display Port** DisplayPort 1.2 / mini DisplayPort **USB** 1 USB 3.0 Type-A

CHAPTER

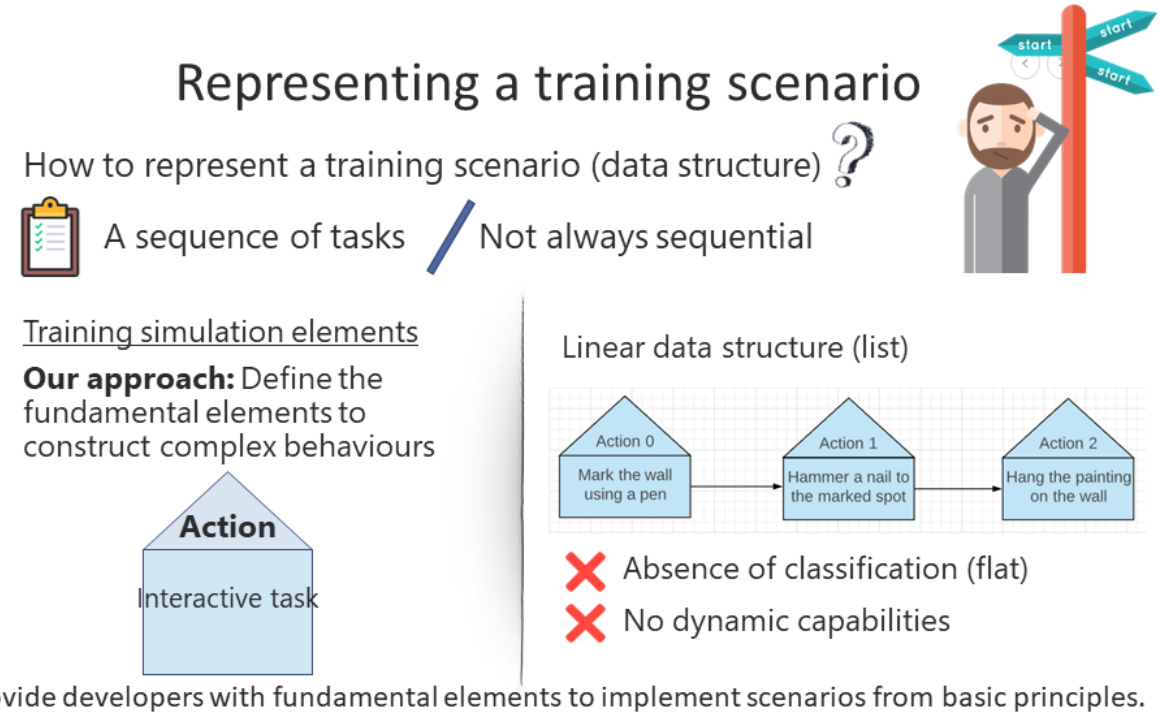
TEN

MANUAL

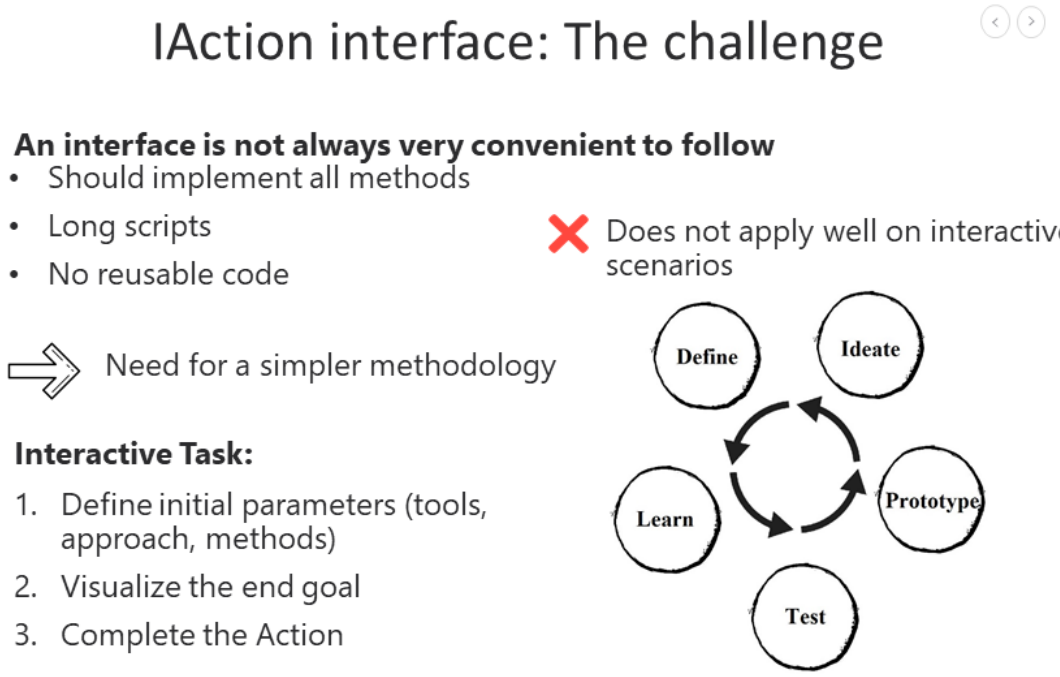
10.1 Action Prototypes

10.1.1 Introduction

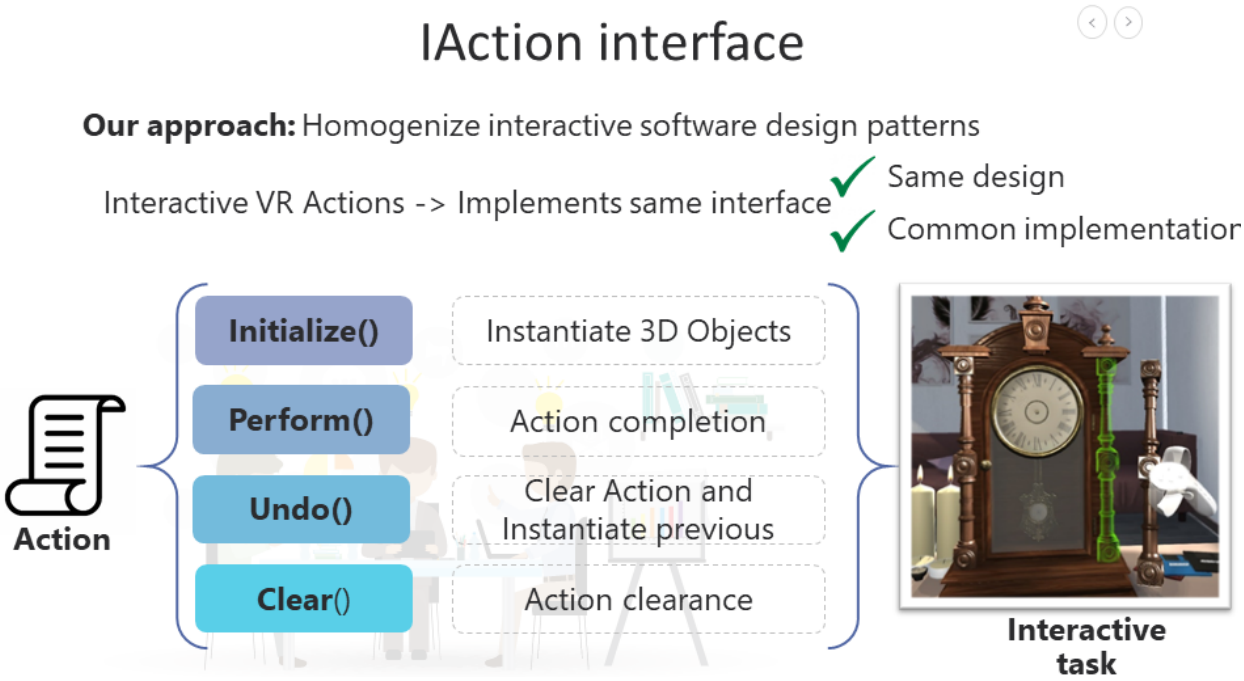
As mentioned before each step of a pipelined process is translated to an Action Blueprint. This Blueprint contains information to define the Action's behavior.



The Action object reflects a flexible structural module, capable to generate complex behaviours from basic ones. This is also the concept idea behind scenegraph; **provide developers with fundamental elements and tools to implement scenarios from basic principles**. Each Action blueprint describes the behaviour in means of physical actions in the virtual environment.



In technical details, each Action script implements the **IAction** interface, which defines the basic rules every Action should follow. This interface ensures that all Actions will have the same methods.



The methods and properties of **IAction** interface are explained in detail below.

```
/// <summary>
/// This Inteface nedds to be implemented for every Action
/// Describes the functionalities the Actions should have
/// </summary>
```

(continues on next page)

(continued from previous page)

```
class MAGES_SDK_API IIAction
{
    GENERATED_BODY()

public:
    /// <summary>
    /// Get the name of the action
    /// </summary>
    FString GetActionName();

    /// <summary>
    /// Get the actor referring to the current Action in Unreal.
    /// This implements the core of unreal's scenegraph.
    /// </summary>
    AAction* GetActionNode();

    /// <summary>
    /// Sets the alternative path for current Action
    /// -1 : Default path
    /// </summary>
    int32 GetAlternativePath();

    /// <summary>
    /// Go to Next Action
    /// Completes the current Action by finilizing and cleaning it
    /// Destroys prefabs, holograms
    /// Also plays animations to set the next one
    /// </summary>
    virtual void Perform() = 0;

    /// <summary>
    /// Go to Previous Action
    /// Resets current Action by finilizing and cleaning it
    /// Destroys prefabs, holograms
    /// Plays Undo animations
    /// </summary>
    virtual void Undo() = 0;

    /// <summary>
    /// Initialize current Action by spawning the necessary blueprints
    /// Sets each Action properties to run correctly
    /// </summary>
    virtual void Initialize() = 0;

    /// <summary>
    /// Sets Holograms for current Action
    /// </summary>
    virtual void InitializeHolograms() = 0;

    /// <summary>
    /// Used only for Combined Actions
    /// Sets the next sub-Action to run after Performing the current one
    /// </summary>
    /// <param name="action">The next Action to run</param>
    virtual void SetNextModule(Action action) = 0;

    /// <summary>
```

(continues on next page)

(continued from previous page)

```
    /// Used only for Parallel Actions
    /// Sets the next action which can be in a different path of scenegraph
    /// </summary>
    /// <param name="Action">The next Action to run</param>
    /// <param name="pathToSet">The different path</param>
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
};
```

Action Prototypes

At this point, we have described the basic interface each Action should implement to be initialized and performed properly. With this interface, a developer can generate action scripts that behave in a common ruleset, following the scenegraph pipeline.

To make our system more efficient we have to limit the capabilities of the Action entity to target simple but commonly used behaviours/tasks in training scenarios. Modelling those behaviours, we will generate a pool of generic behavioural patterns and tasks from which we will develop scenarios that are more specific.

Therefore, MAGES SDK introduces several specific Action behaviors that developers can utilize to simulate training scenarios. These are called **Action Prototypes** and are the following:

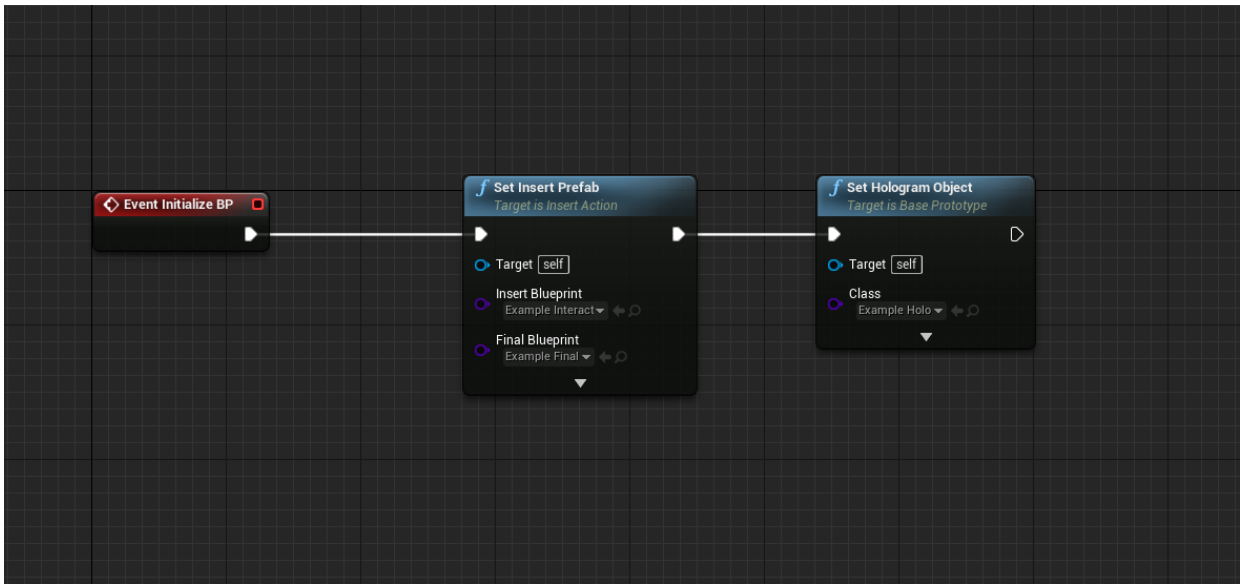
1. *Insert Action*
2. *Remove Action*
3. *Use Action*
4. *Combined Action*
5. *Parallel Action*
6. *Question Action*
7. *Animation Action*

Each **Action Prototype** inherits from **BasePrototype**, a **base class** that utilizes a common set of methods and properties for every prototype.

10.1.2 Insert Action

Insert Action is referring to a specific type of Action that a user has to insert an object to a specific position in order to complete it.

For instance, an insert action blueprint can be seen below:



Note: This blueprint inherits from the `BPInsertAction` class.

Action Blueprint Explanation

1. Set Insert Prefab

This method sets the Action's insert actors that will be spawned on Initialize. To set an insert Action you need to spawn two different objects, the grabbable (interactable) item and the final item. The first argument is the path to the interactable blueprint while the second is the path to the final.

The Grabbable Parent and the Final Parent arguments are optional and are used in case these objects need to be spawned as children of already spawned actors.

2. Set Hologram Object

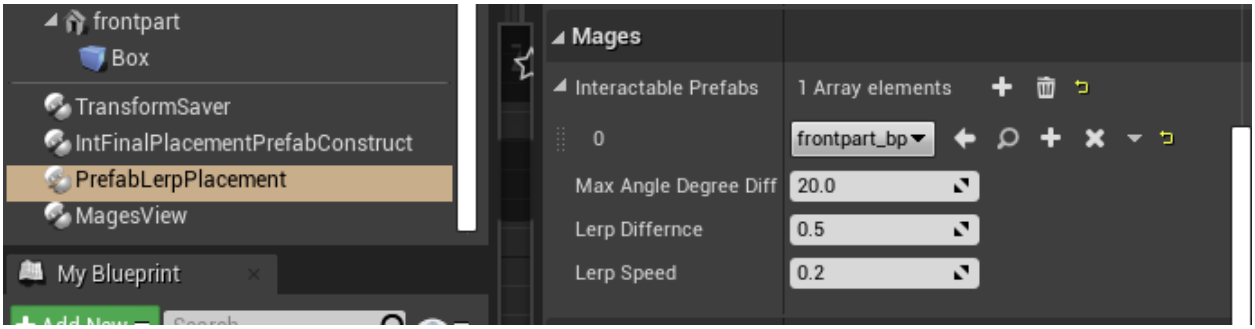
The *Hologram* is set for initialization through the *Set Hologram Object* function.

3. Prefab Constructors

To create the correct actors you need to set their components as follows.

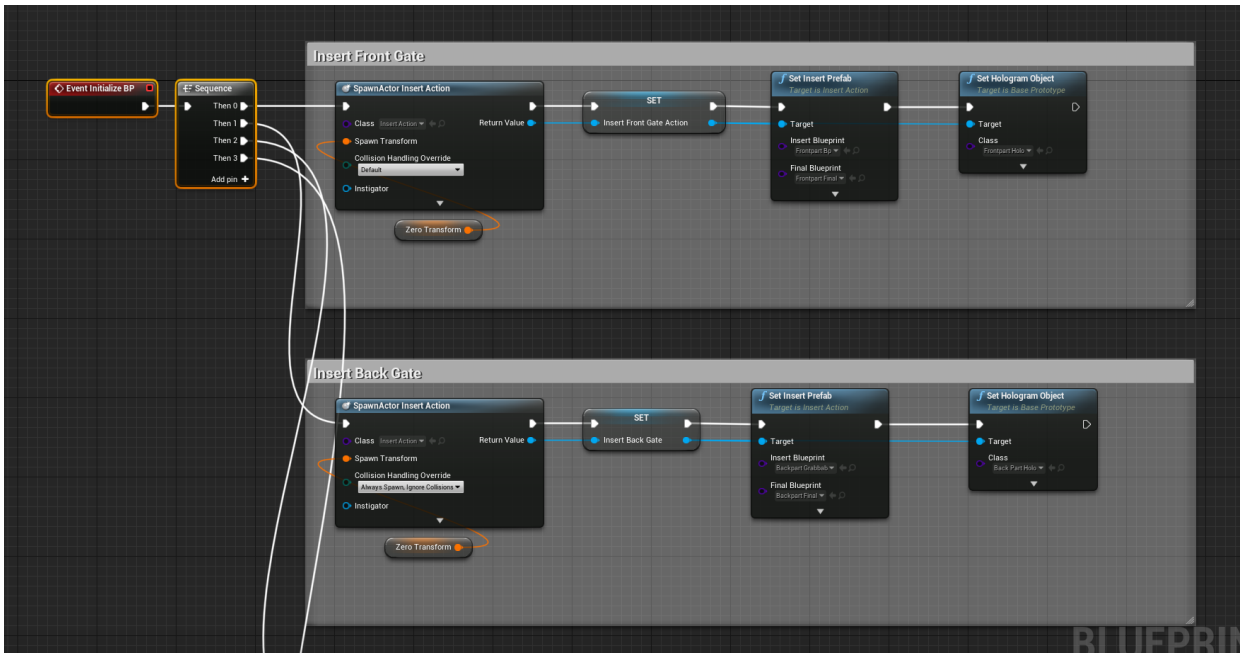
The Interactable actor needs a prefab constructor component, specifically the InteractablePrefabConstructor. From this component you need to define that this actor will be used in an insert action, this is done in the Prefab Interactable Type property. Next on the list is the final placement actor. The final placement actor needs the IntFinalPlacementConstruct component, where you need to setup the Prefab Type as `InteractableFinalPlacement`. In addition, you need to reference you interactable actor in the PrefabLerpPlacement component.

You can find theses actor pre-configured from the MAGES Menu, `Create Prefab -> Insert Action`.



Adding More to it

A more complex example that involves two insert actions as sub-actions is the following:

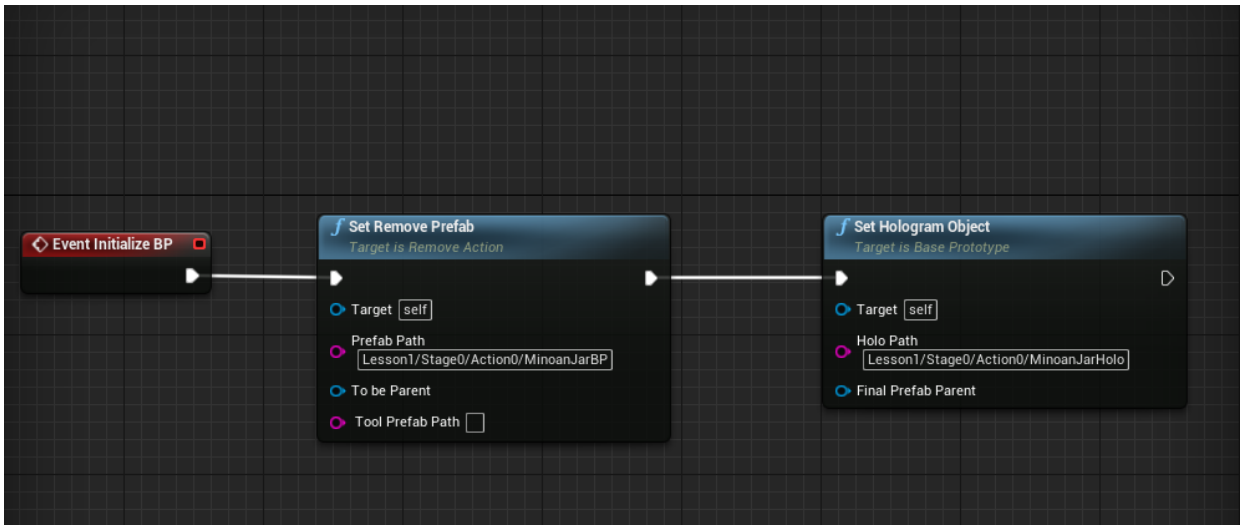


In the above example, notice how each individual insert action follows the exact same pattern of object initialization.

10.1.3 Remove Action

Remove Action describes a step of the procedure which user has to remove an object using his hands or a another interactable.

Example of Remove Action Script:



Action Script Explanation

1. Set Remove Prefab

This method sets the Action's removable actors to initialize the Action behavior. To set a Remove actor you need a string which contains the path to the removable blueprint. This method can be called many times in an Action Blueprint. Each time Set Remove Prefab is called a new removable actor is added and spawned into the remove prefabs List. To perform the Action, the user needs to remove all of them.

2. Set Hologram Object

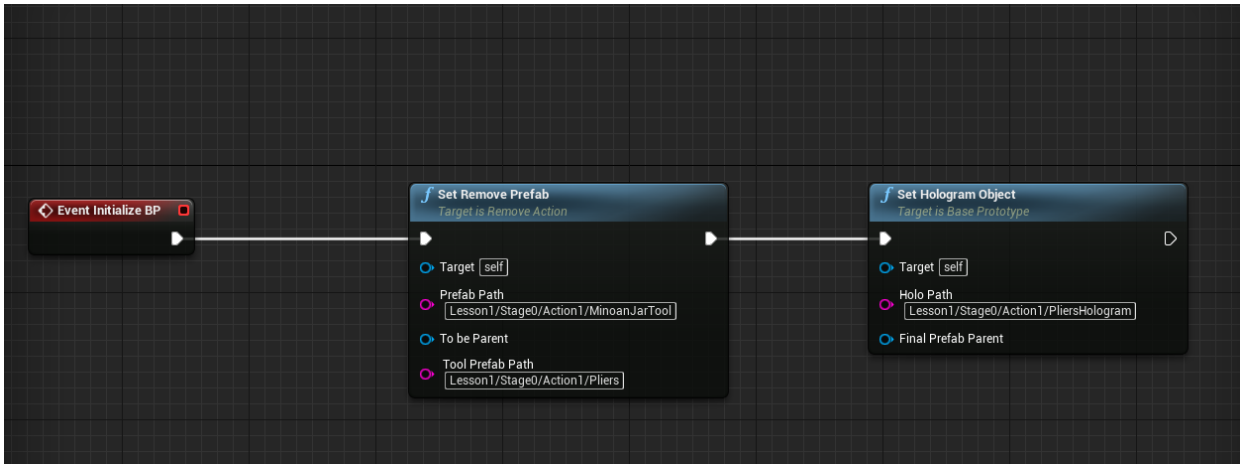
Usually a remove Action does not have a hologram. Instead we use a flashing indicator at the removable object. To enable the flashing functionality you need to check the "Attach Prefab Spawn Notifier" option at the Interactable Prefab Constructor script.

3. Prefab Constructor

To generate a removable prefab you need to select the "Simple Grabbable Actor" option at the create prefab sub-menu. To enable the Remove Action functionality for this object you must select the "Remove" option in the Prefab Interactable Type property field at Prefab constructor script.

Adding More to it

A more advanced example is the following:



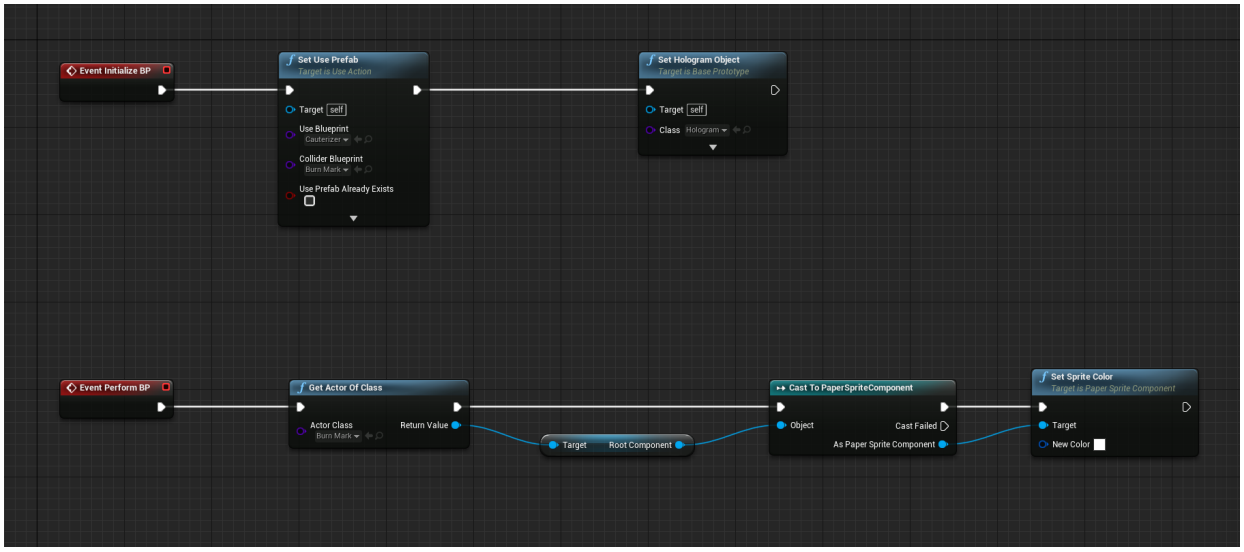
In this example we are also specifying a tool which will be used for removing the object by adding the blueprint's path in the `Tool Prefab Path` argument of Set Remove Prefab function. In the interactable constructor of the removable object it is important to set the Prefab Interactable Type to `Remove With Tools` as well as the Prefab Detach Feature to `EventTriggerCurrLSAOnDestroy`.

10.1.4 Use Action

Use Action is used in situations where the user must use an object to complete the Action. There are different types of usages from which the user can select, which are,

- **Simple** The user must hold the interactable object in a specific area for a specified amount of time.
- **UseWithTool** The user must hold the interactable object in a specific area for a specified amount of time while also holding the trigger button for activating the object.
- **Hit** The user must collide the interactable object with a specific area for a specific amount of times with enough force.

Example of Use Action Script:



Action Script Explanation

1. **Set Use Prefab** 1.1. Sets the blueprint actor user needs to take and place it on the use collider to perform the Action.

   1.2. Sets the collider blueprint that the 'Use Actor' will interact to complete the action.

   1.3. You can specify a parent in order for the collider to be spawned as its child.

   1.4. In cases where the use actor is already in the scene, e.g. due to being used in a previous action, you can tick the `Use Prefab Already Exists` option and it will not be spawned a second time.

2. **Collider Prefab Constructor** The `UseColliderPrefabConstructor` component must be added on the collider actor. You can read more about this constructor in *Use Collider Prefab Constructor*.

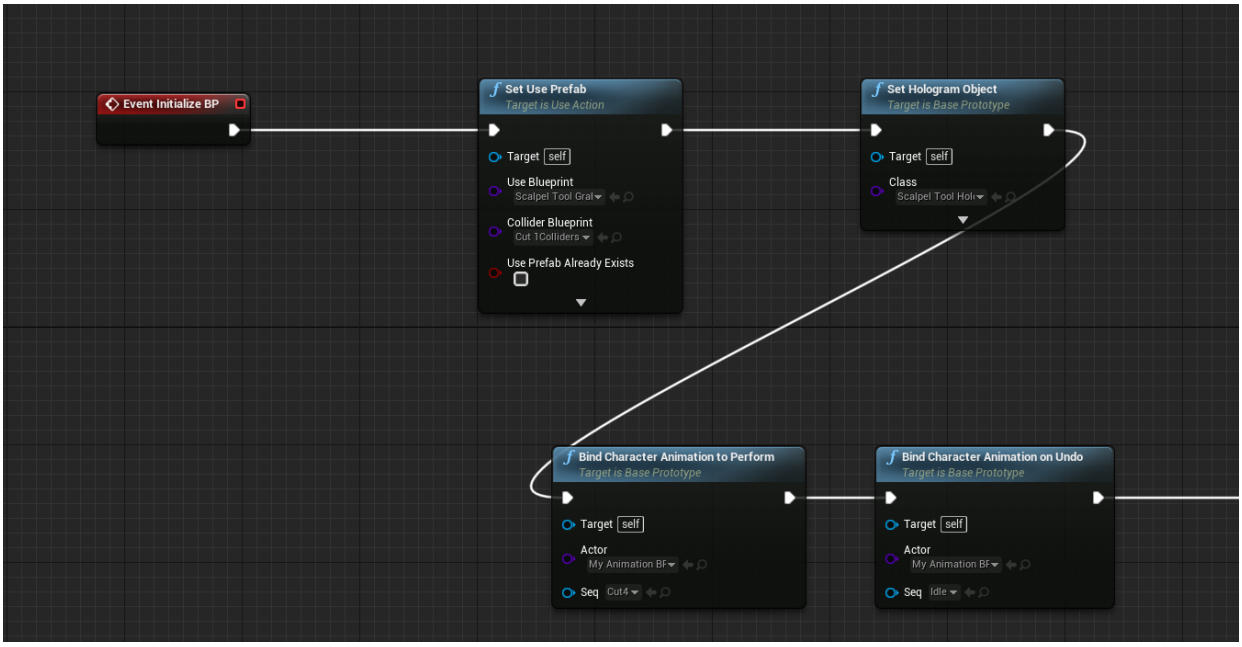
3. **Set Hologram Object** Spawns the hologram from the specified path.

Depending on the behaviour needed to complete the action the user needs to set the corresponding field in the `Collider Trigger` property of the `UseColliderPrefabConstructor` component. Based on the above setting different properties will be used from the Constructor.

- **Stay Time** This is used for Simple or UseWithTool trigger colliders. It defines the time the user needs to hold the object inside the trigger.
- **Hit Times** This is used for Hit trigger colliders. It defines the number of times the user needs to hit the box.
- **Hit Force** This is used for Hit trigger colliders. It defines the force needed for each hit to register.
- **Hit Movement** This is used for Hit trigger colliders. The offset that will be applied to the collider after each hit.

Adding More to it

A more advanced example is the following:



In this example we add one animation that will be played at the end of the action on the patient and another one that will be played in case we undo this action.

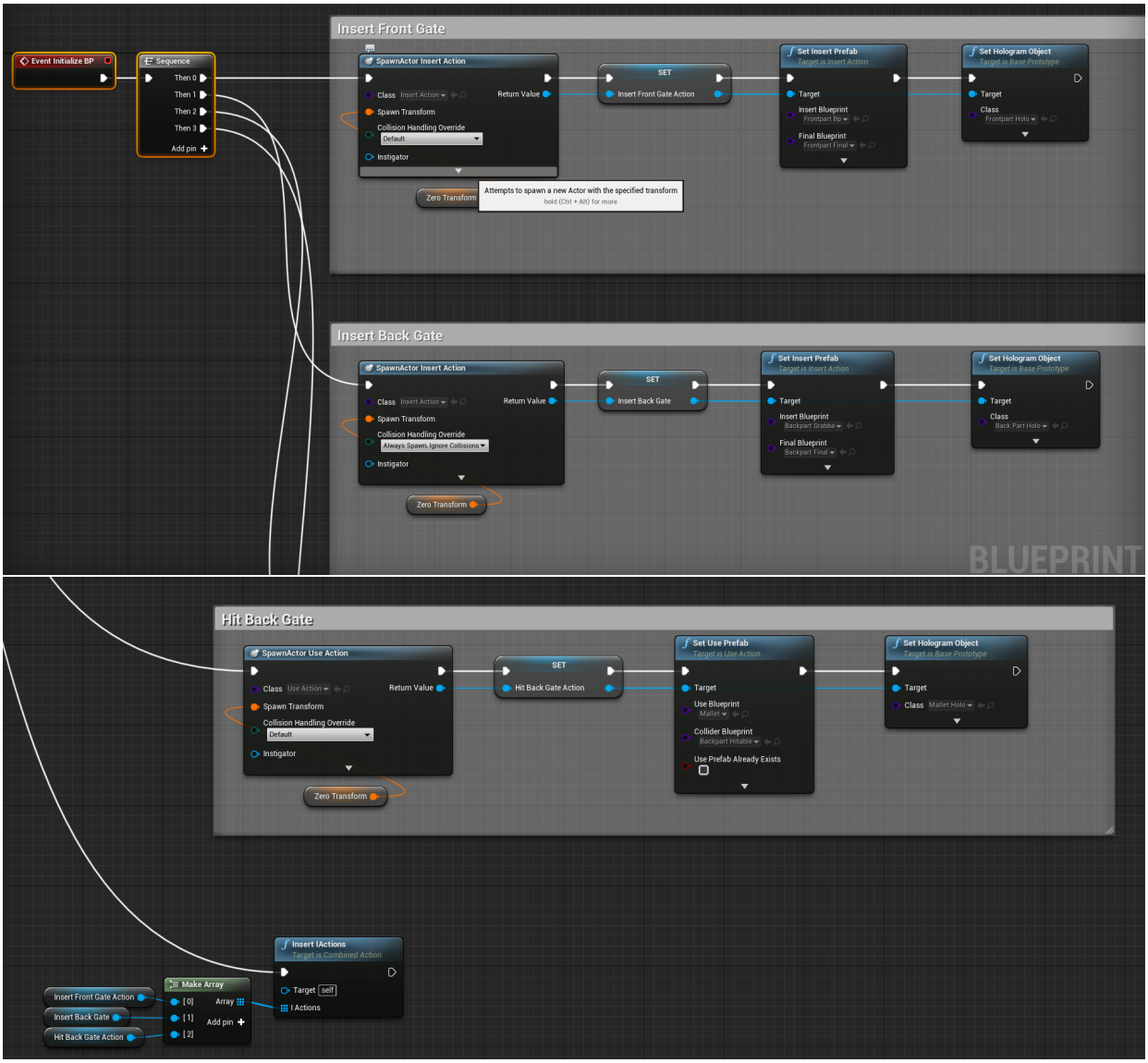
10.1.5 Combined Action

A Combined Action has the attribute to perform multiple sub-actions sequentially. Sub-actions consist of any other type of Actions described in this section.

For example, to have an Insert Action followed by a Remove Action but consider both as one whole action, use the Combined Action prototype (BPCombinedAction Class) as described in this section.

To create a Combined Action follow the same ideology as the other prototypes with the difference that you need to create an actor for each sub-action you wish the user to perform.

For example,



Action Blueprint Explanation

The action prototype actors are first spawned and then set to a parameter for later use. Similarly to the other action prototypes we need to define the actors for each sub-action. (e.g., an Insert Action needs SetInsertPrefab, a Use Action needs SetUsePrefab, and so on).

Note: Notice, the action class in the SpawnActor blueprint.

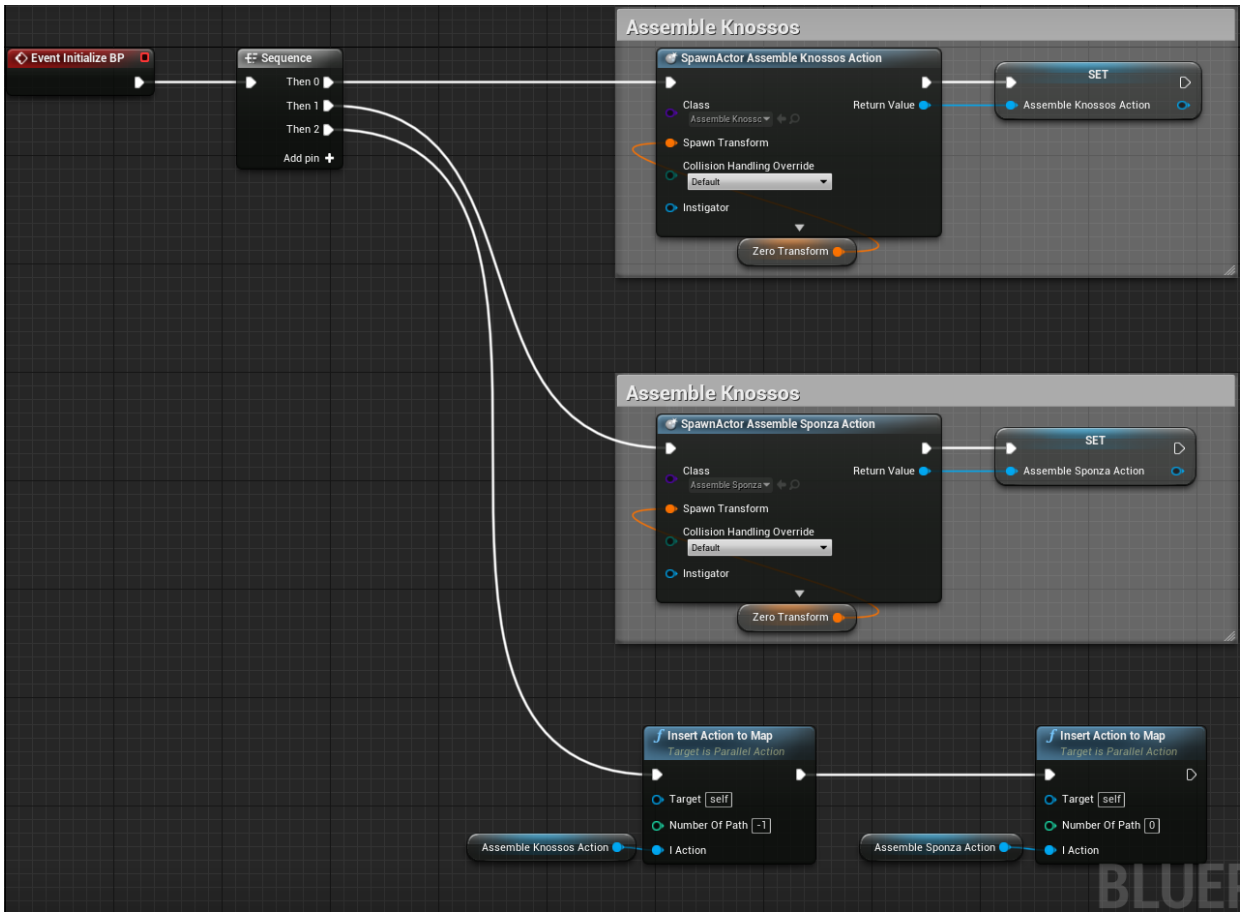
Finally, you need to finalize the Combined Action's setup by calling the InsertIAction's function which takes as a parameter an array containing all the sub-action actors created in the previous steps.

10.1.6 Parallel Action

A Parallel Action is used to create an Alternative Path decision making action.

Multiple actions can be initialized and according to which action is completed, the session will follow the according path.

Example of Parallel Action Blueprint:

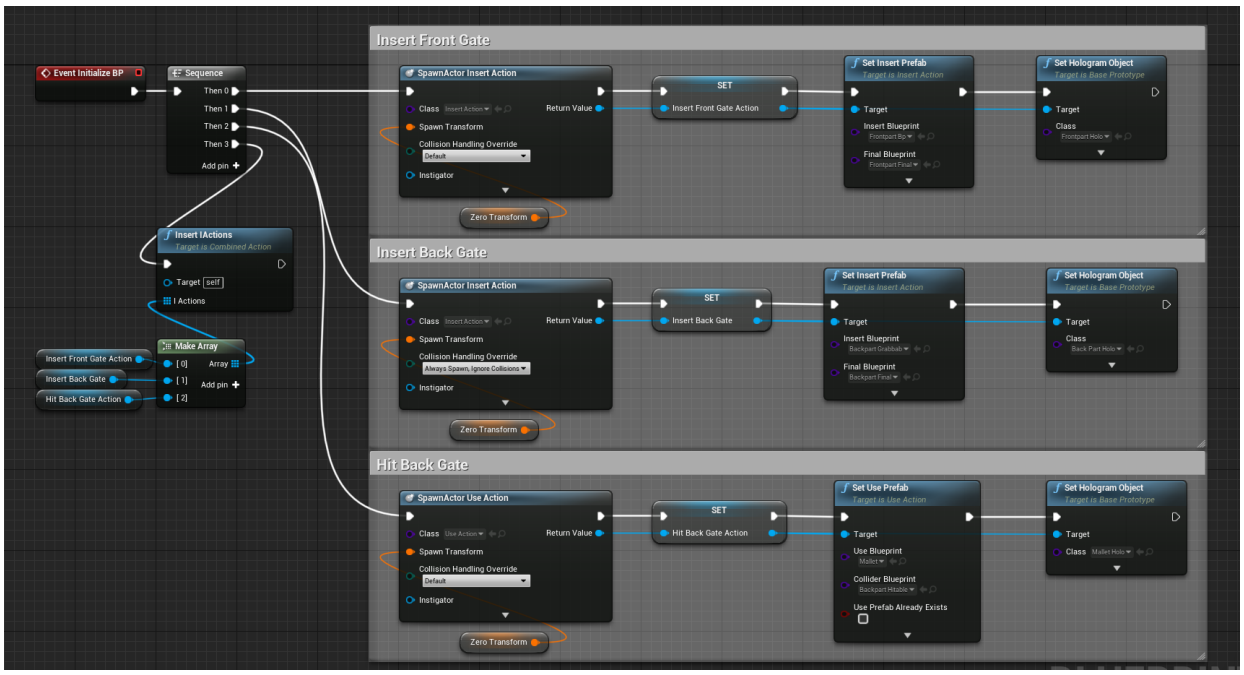


Action Blueprint Explanation

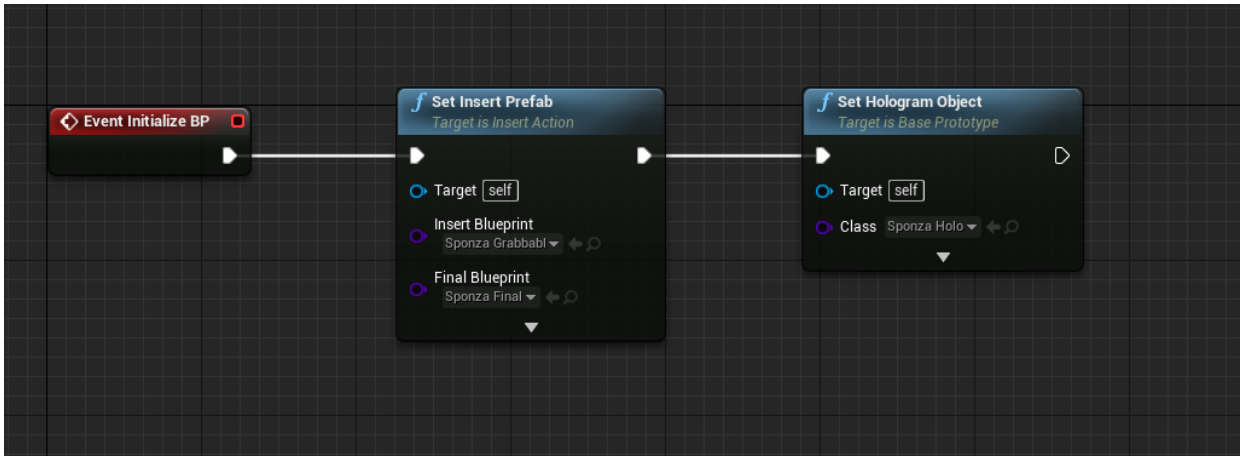
In this example we create two Actions (one Combined Action and one Insert Action) and initialize them asynchronously. To correctly initialize a Parallel Action add the classes of the actions inside the Parallel Action class. Then, add them as a Component as shown in the example above.

For reference, here are the blueprint graphs of the Combined Action and the Insert Action classes used in the previous graph:

Combined Action



Insert Action

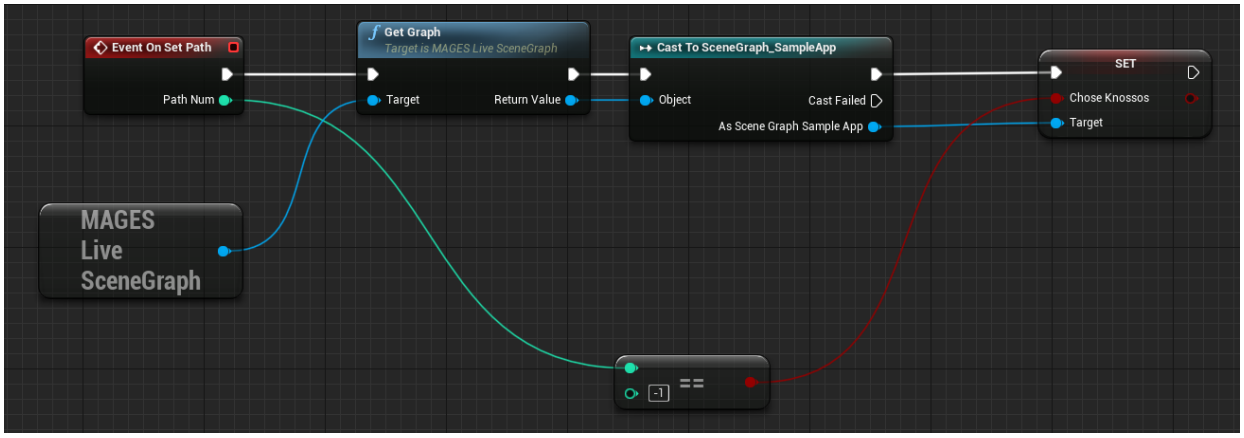


The final step is to set each action to trigger a specific Path, using the **InsertIActionToDictionary** function.

Note: Keep in mind, that once performed, the Parallel Action will trigger all Perform() functions of each action it contains. This may result in actor deletion or other unintended behavior, so make sure you handle each occasion.

The Parallel Action includes an event called `On Set Path`. This event informs about which of the actions in the

map was actually completed by the user. With this event, you can set custom variables inside the Scene Graph (as seen in the next image), or do anything else specific to the user's decision.

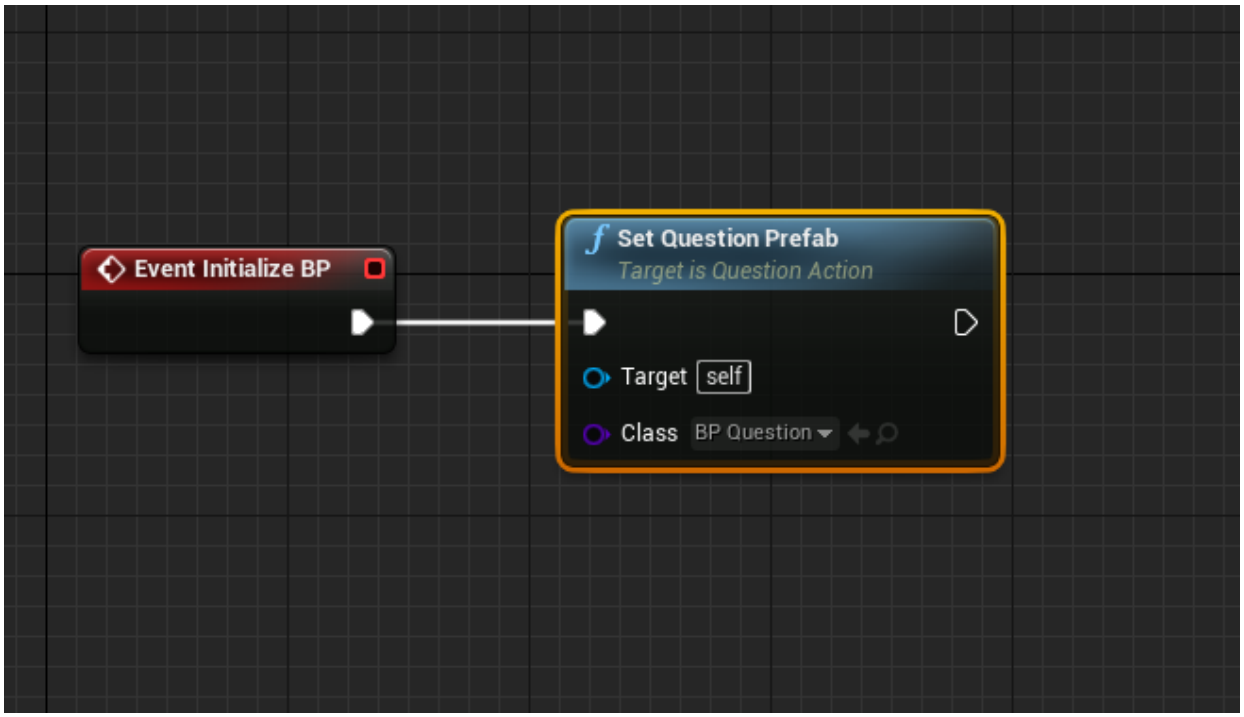


10.1.7 Question Action

This type is described as an action expecting the user to make a decision on choices that answer a specific question. Answers are expected to be widgets inheriting from `Mages Question Button Widget`.

Choosing any answer will perform the action, registering which answer was selected, as well which answers were correct. Upon submitting the answer, the action performs after 5 seconds.

Example of a Question Action:



Action Blueprint Explanation

1. Set Question Prefab => Spawns the Action blueprint containing the question to be answered. Has one argument:
    1.1. The blueprint to be spawned

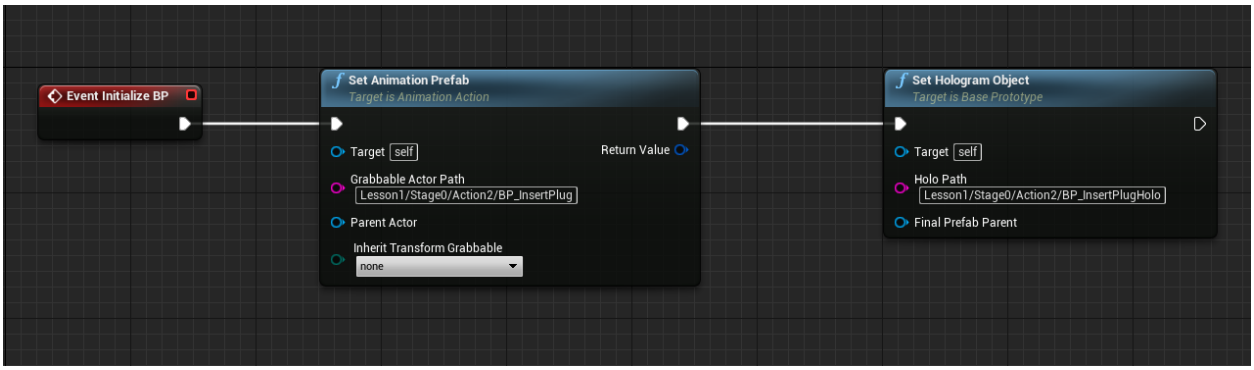
10.1.8 Animation Action

This type of Actions is described as Actions in cases we we want to insert an object but the object needs to be inserted with an animated movement.

An example may be the insertion of a wire into a tube.

To implement this Action we would need to record the insertion of the wire and then we will push it with our hands to the final position. The movement from the controller is translated into the normalized value of the animation [0-1].

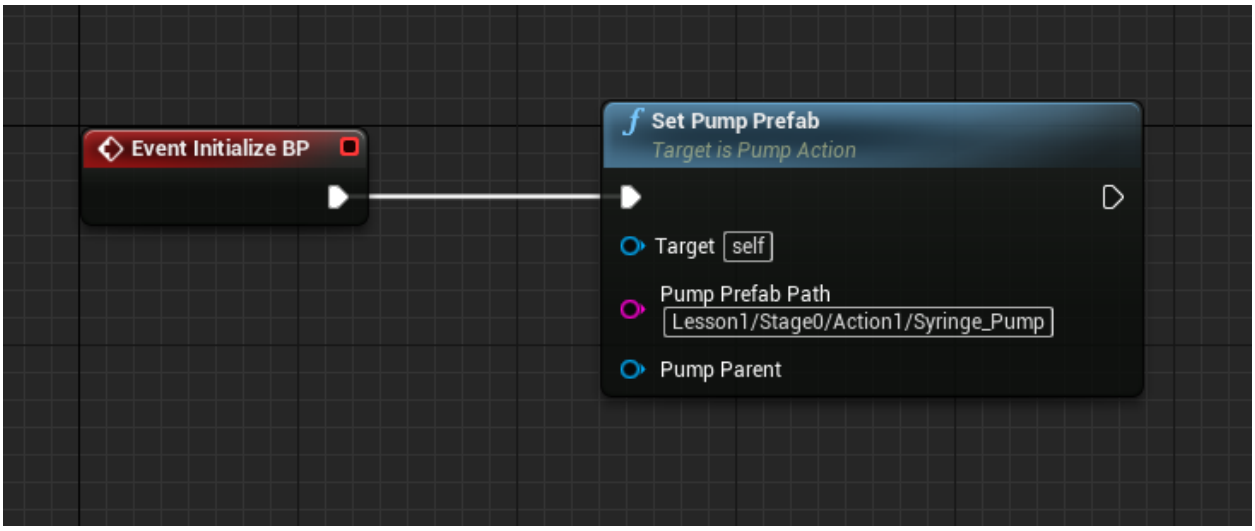
Example of Animation Action:



10.1.9 Pump Action

A Pump Action models an interaction where a user may squeeze, apply pressure (and more) to an interactive item. This is accomplished using the analog buttons on the controller(s)

Example of a Pump Action:



Action Blueprint Explanation

1. Set Pump Prefab => Spawns the Action blueprint containing the question to be answered. Has two arguments:

    1.1. The prefab to be spawned 1.2. The parent of the prefab (optional)

Multiple calls to `Set Pump Prefab` will have the user perform the pump action on all of the spawned blueprints

10.2 Physics

10.2.1 Mages Interactable Item

The most important component the developers need from the Mages Physics is the MagesInteractableItem. When added to an actor it enables physics interactions with that actor.

More specifically, for an object to have physics interactions inside Unreal, it is easy for the developer. Create colliders, enable physics simulation and enable gravity. But if grabbing the object is implemented from the developers with Unreal's default method, attachment, upon grabbing, that object will lose its physics properties (e.g. it won't collide with object - passing through them).

Instead of parenting, MAGES SDK provides this component. It is responsible for the object to be able to be grabbed from the user while maintaining its physical properties.

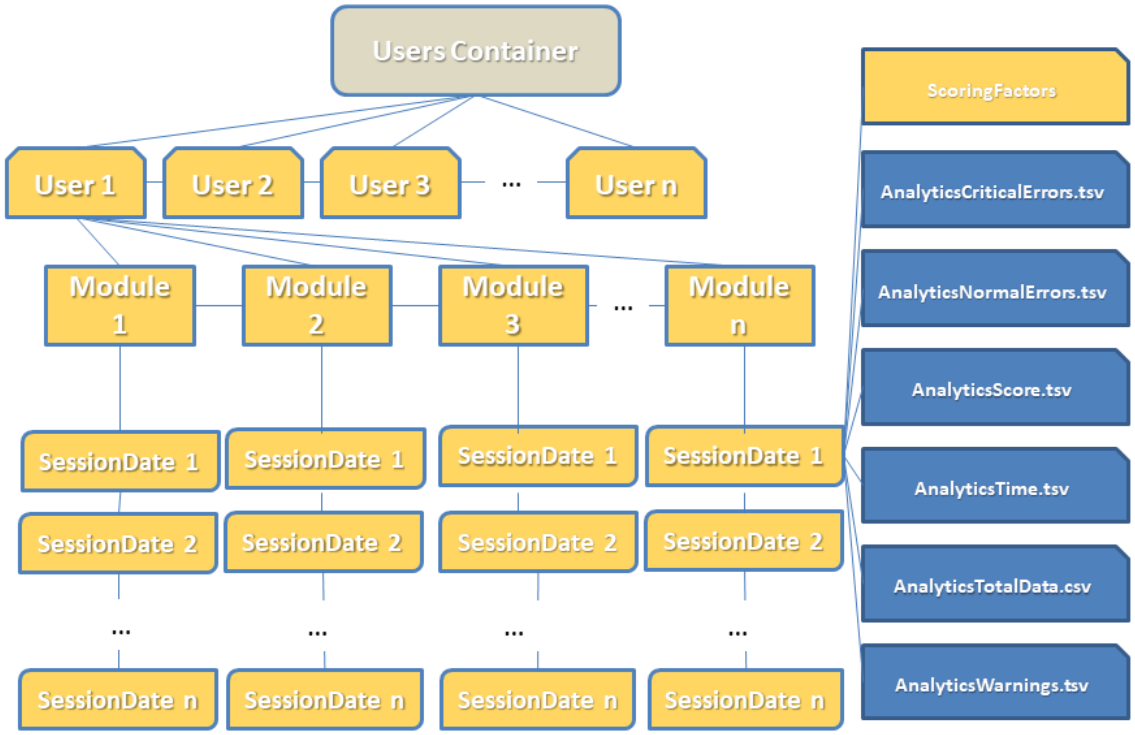
Parameters and usage explained below:

Parameter	Description
Can Attach	object can/can't be attached to the user's virtual hands
Disable Kinematic On Attach	if true, when attached to a hand it will disable its kinematic properties, enabling the object's physics interactions.
Enable Kinematic On Detach	if true, when released the object from users hands its kinematic properties will be re-enabled, disabling physical interactions.
Drop Distance	set a distance margin between hand and object. When their distance is greater than the margin, the interaction will stop, useful for objects that might get stuck between other colliders.
Enable Gravity On Detach	if true, when released the object from users hands its gravity will be turned back on.

10.3 Analytics

10.3.1 Analytics File System

A high-level overview of MAGES analytics file system is depicted in the image below.



Inside Users container (i.e., the root node) the following structure exists:

1. User Folders

   One folder per user. Each of these folders contain all necessary files of their respective user progress.

2. Module Folders

   Each user folder contains one or more modules folder. Module folders are named after respective module names. Module folders are generated when user runs a module for the first time.

3. SessionDates Folders

   These folders are contained inside their respective module folder. Each of the session folders represents a single user session of the module. A session folder is created when the user finishes a complete playthrough of the specific module.

   Generally, we store the following data for each user:

   3.1. Number of critical errors in each module session and the name of the action, where they occured.

3.2. Number of non-critical (or normal) errors in each module session and the name of the action, where they occured.

3.3. The score of each action in each module session.

3.4. The time that the user needed for each action in each module session, measured in seconds.

3.5. The total data (all errors, critical errors, warnings, final score) for each module session.

Stored data

The content that is saved in the files mentioned above, concerns the progress of users in each step of our module.

A basic example is provided below.

Errors & Warnings Data

Files under this category concern errors and warnings occurred during users playthrough.

Errors and warnings are structured in a similar manner as shown in the image above.

The structure is intuitive and self-explanatory. Each action the user obtains a warning/error is kept in track alongside the amount of errors.

Action	Errors
Question Action Example	1
Assemble Knossos Or Sponza Action	0
Remove Jar Example	0
Remove Jar With Tool Example	2
Apply Glue Action	0
Insert Plug Action	0

Scoring Data

We keep track of users' score for each action in the module.

Specifically, the name of the action is saved, along with the score. Score variables are integers within the range of $score \in [0, 100]$.

An example file content of users' score is exhibited in the table below.

Action	Score
Question Action Example	100
Assemble Knossos Or Sponza Action	100
Remove Jar Example	100
Remove Jar With Tool Example	50
Apply Glue Action	80
Insert Plug Action	90

Timing Data

The time users spent on each action; is another important variable we monitor. More specifically, action names are accompanied by a double-precision number, which represents the time user has spent on that action.

Timings are measured in seconds. An example is shown in the table below.

Action	Time
Question Action Example	10.25
Assemble Knossos Or Sponza Action	83.70
Remove Jar Example	20.48
Remove Jar With Tool Example	115.80
Apply Glue Action	30.67
Insert Plug Action	17.61

Accumulated User Data

For convenience, we keep the concrete form of the data presented above, augmented with relative, yet necessary, information per session.

Namely, we keep track of the following information; the session date (DD/MM/YY), time session ended (HH:MM:SS), module difficulty, handedness, total score, total time, total errors (normal, critical) and warnings, number of current session (identifier), and total time of session.

Username	IP	Session Date	Session Time	Difficulty	Total Score	Total Time In Seconds	Total Errors	Total Critical Errors	Total Warnings	Total Time
Username	IP	06/12/2020	16:00:44	Easy	36	484	0	1	0	00:08:04

Scoring Factors

In order to calculate and store scoring information more precisely and for better and more detailed presentation of the data to the users, we also keep another type of information called Scoring Factors.

There are different kinds of scoring factors, some of them are the same across all of our modules (that would be the ones that concern errors, critical errors and warnings) and some others that are a bit more specific to module actions (for instance, for a module that contains question actions, there will be different scoring factor for those actions).

Error	100
0	-1
Object has been contaminated!	
Error	30
2	-1
Goggles can be recycled!	
Critical Error	0
1	-1
You are entering without PPE	

The data shown above are the scoring factors for a specific action of a module.

The first line states the word "Error", which means that this scoring factor concerns an error of this action. The integer "100" is the score credited to this factor. The "0", which is located below the word "Error" in this example, is the number of times the users made that error. "-1" means that this error can be made infinite times (or that there is not any limit to the number of times that this error can be made).

In case there is a limit, that number may be any positive integer. Finally, the name of the specific error that this scoring factor represents is given (in this example "Object has been contaminated".

An action can have one or more scoring factors. In the case of our example, this specific action had three different scoring factors. The first one is the one we just finished presenting. The rest follow exactly the same logic.

10.3.2 Generating Analytics

As discussed thoroughly in *Analytics File System*, Analytics expect certain assessment formatting (e.g., scoring data, accumulated user data, etc.) and produce *certain* output.

ORamaVR provides a simple out-of-the-box solution for generating analytics for your products. In detail, analytics are per action and have to be explicitly specified for each action from the analytics editor.

Note: Currently, the analytics panel needs the action that they refer to be specified. In version 1.0 this configuration will take place through the SceneGraph Editor.

Visual Editor

To specify the analytics that will be recorded for each Action, start by creating a MAGES Analytics asset. This can be found by right clicking in the content browser and selecting

- *MAGES -> Analytics Asset*

Once you create it and open it. You will be presented with a similar view:

i. **Save:** Status window, which will be updated after each action with a relative message. You can also find the asset in the editor by pressing the browse button.

ii. **Scoring Factors:** Detailed explanation can be found here GeneratingAnalytics

iii. **Multiplier:** The weight of this action regarding the rest actions.

In the Analytics window you can specify all scoring factors for the current Action.

Scoring factors

As described in *Analytics File System*, MAGES SDK supports a variety of predefined scoring factors.

Current available scoring factors are enumerated below in an algorithmic manner **following the order of the Analytics Editor**:

1. Time

Input: MaxTime
Output: Score
OnEachFrame()
$time_{counter}$: $time_{counter}$ + (time since last frame)

OnPerform()
if $time_{counter} >$ MaxTime
    diff : $time_{counter}$ - MaxTime
    Score : 100 - (diff * 10)

2. **Error Colliders** 2.1 Avoid Object Factor

Input: Avoid_Colliders
Output: Score
OnCollision(Interactable I **, Collider** C**)**
if C in Avoid_Colliders
    errors++
    Score : Score - lostPointsForError

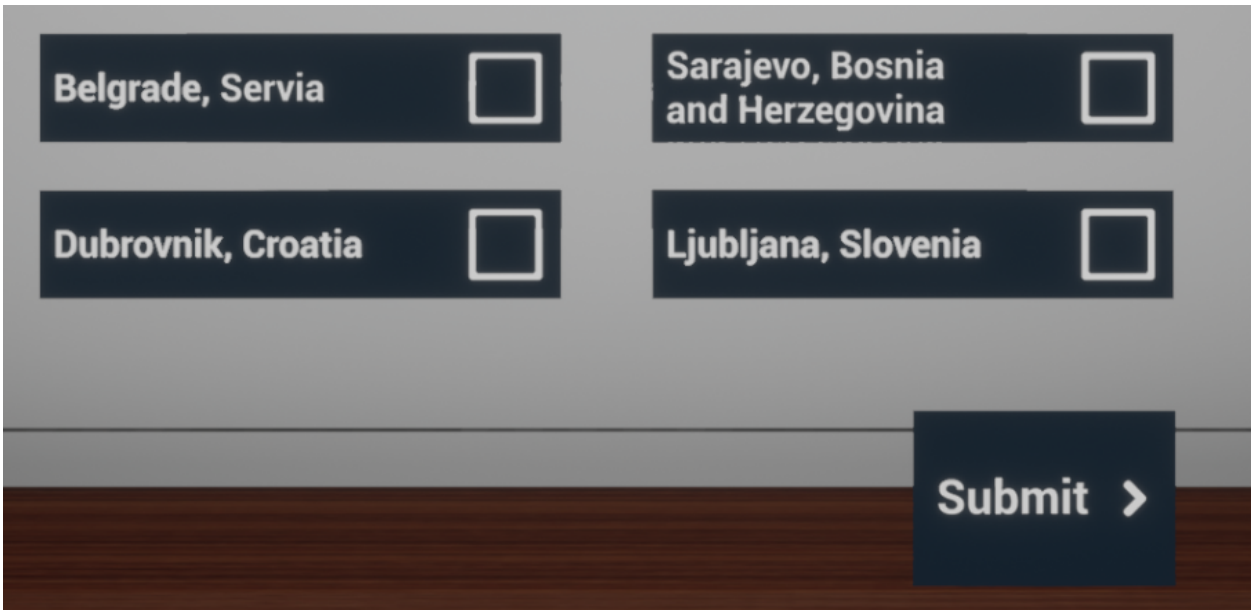
2.2 Stay Error Colliders

Input: Interactable I_1
Output: Score
OnUseInteractable(Interactable I_2**)**
if I_2 is used on I_1
    if I_1 is not currently held
        errors++
        Score : Score - lostPointsForError

2.3 Hit Perform Colliders

Input: Colliders, Colliders_{size}
Output: Score
1 **OnCollision** (Interactable I, Collider_i)
2 if Colliders cotains Collider_i
3 remove $Collider_i$ from Colliders
4
5 **OnPerform()**
6 leftColliders : get size of Colliders
7 Score : $\frac{leftColliders*100}{Colliders_{size}}$

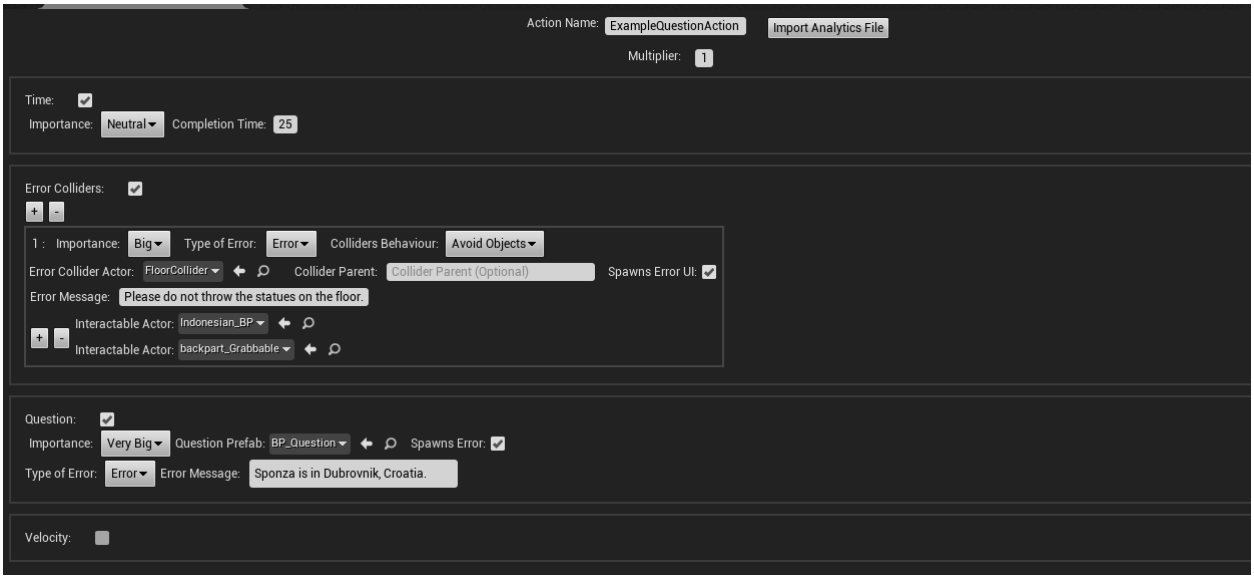
3. Question



4. Velocity

Input: MaximumVelocity
Output: Score
1 **OnEachFrame(Interactable** I)
2 $current_{velocity}$: get velocity from I
3 if $\text{current}_{velocity} > MaximumVelocity$
4 errors++
5 Score : Score - lostPointsForError

Using the above combinations you can produce an output similar to the following:



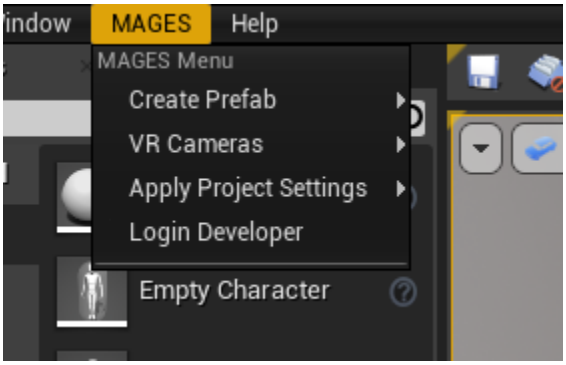
Finally, click the **Save** button down in the editor window to save your changes.

> **Warning:** If you forget to *Save your Analytics* for each action, the changes will get discarded.

10.4 MAGES™ Menu

10.4.1 Introduction

MAGES SDK enhances the functionality of Unreal's menu bar, with the MAGES menu.



In this dropdown menu, you will find:

Create Prefab With this sub-menu developers can create preconfigured actors in the scene with all the appropriate components added depending on the type of actor selected.

VR Cameras With this sub-menu users can quickly generate a preconfigured MAGES VR Camera.

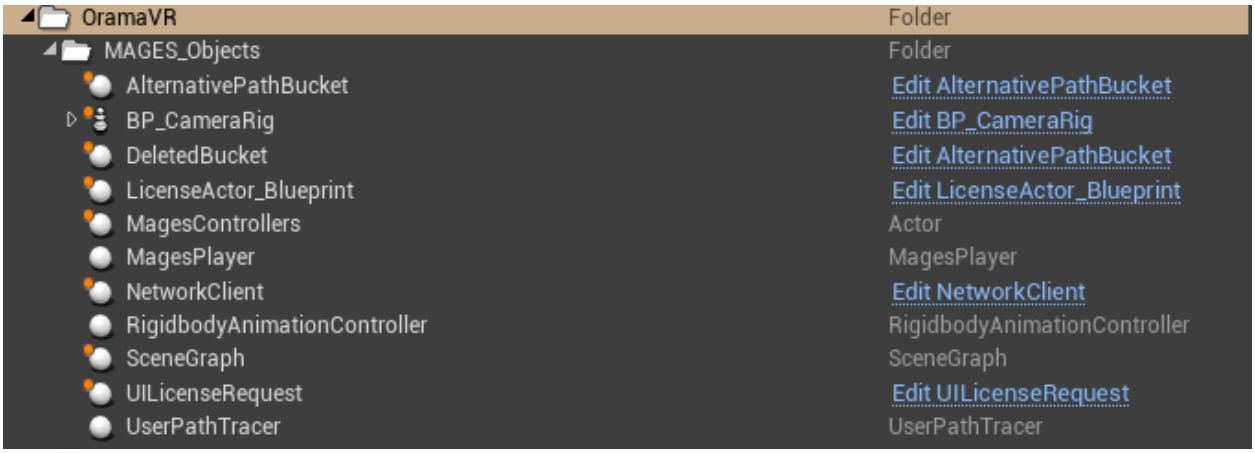
Apply Project Settings This initializes some good default settings for packaging projects on the corresponding platform of choice.

Login Developer This option saves the developers account to be able to run and develop their VR application inside the Unreal Editor.

10.5 Unreal Level

10.5.1 MAGES™ Objects

In order for a Unreal project to run with the MAGES SDK, it is essential to have inside the Unreal level specific actors derived from specific classes used as managers and controllers of the different functionalities of MAGES SDK.



Actors

RigidBodyAnimationController (Required) Enables and handles the usage of GA's Dual Quaternions.

UserPathTracer (Required) Computes and stores data regarding the user's session.

Scene Graph (Required) When the projects start, all LSAs will be created as children of this particular actor. In addition this actor requires the `ScenegraphPathDefinitions` data asset where critical information is stored regarding the project's structure.

DeletedBucket (Required) Nothing needs to be attached. This is handled from the Alternative Path (careful, it must not be an Alternative path's child!)

AlternativePathBucket (Required) This is the responsible actor for all the alternative LSAs (Lessons Stages Actions). Add the Alternative path and Alternative Path Importer components.

MagesControllers (Required) Adds the current VR SDK controller component as well as the MAGES Controller Class. In the second component (MagesControllerClass), the two hand controllers found inside the BP Camera Rig need to be referenced, as well as their hand models.

MAGESPlayer (Required) Responsible for initializing the virtual hands and connecting them with the virtual camera.

NetworkClient (Optional) Responsible for establishing the connection with the photon cloud server. It is not needed in case of single player.

LicenseActorBlueprint (Required) Responsible for checking out the licensing of users and developers.

UILicenceRequest (Optional) Responsible for handling user login on packaged projects.

10.5.2 VR Camera

Camera as a Pawn

The VR Camera is an implementation of SteamVR's camera and assets from the MAGES SDK. It contains a connection to a generic set of hands created from ORamaVR (Hand Models) in order to be the same for any type of commercial HMD used.

Desktop VR Camera

Camera Rig Input Controller Our VR Camera Rig derives from the **Camera Rig Input Controller** class, which contains all code for the camera movement (x-z translation, rotation, height) using the controllers.

Hand Controllers The actors HandControllerLeft and HandControllerRight serve as mediators between the real VR controllers and the virtual hands. They contain:

1. a motion controller which captures the movement of the real VR controllers and apply the necessary translation to the virtual hands.
2. a physical constraint binding the user controllers with the virtual hands.
3. a widget interaction controller which is used for interacting with MAGES UIs.

Hand Models The models HandModelLeft and HandModelRight contain the skeletal meshes of the VR Hands .

Main Camera

Under the Camera Rig Blueprint there are the following components:

1. **VR Camera:** The player's main camera component.
2. **Avatar:** A skeletal mesh component containing the user's avatar.

Oculus Quest Camera Setup

Note: Currently, under development.

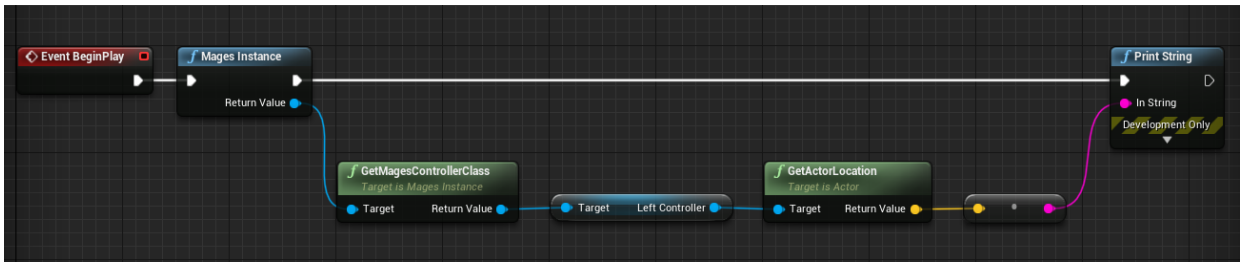
Vive Focus Plus Camera Setup

Note: Currently, under development.

10.5.3 MAGES Instance

There are many components or subsystems in our SDK that you may need to access at any point, from any point. For example, getting the user's left controller would involve searching it in the scene, or searching for the character first. For this purpose, we store all important references to our objects inside the `MAGES Instance`.

Example of using the MAGES Instance:



In this example, we get the MAGES Instance, and get the controller class. We use that to get the left controller inside the level.

10.6 Project File System

In this section we will introduce the project file system MAGES SDK utilizes.

Readers will get familiarized with the existing project structure and incentivized to work towards the same direction.

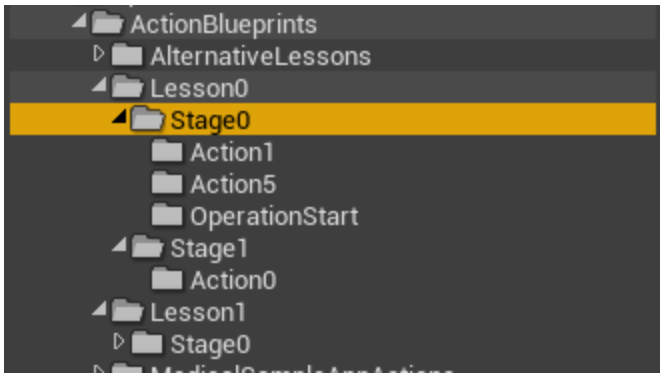
Take notes, as certain structural elements of the SDK are immutable. In other words, there is a certain structure developers are expected to store their files for the SDK to operate smoothly.

Warning: Failing to follow the structure presented in this section might lead to unexpected behaviors.

10.6.1 General Guidelines

Everything operation specific (action and lesson blueprints, etc...) is stored using a *specific* structure.

For instance, action blueprints are stored as follows:

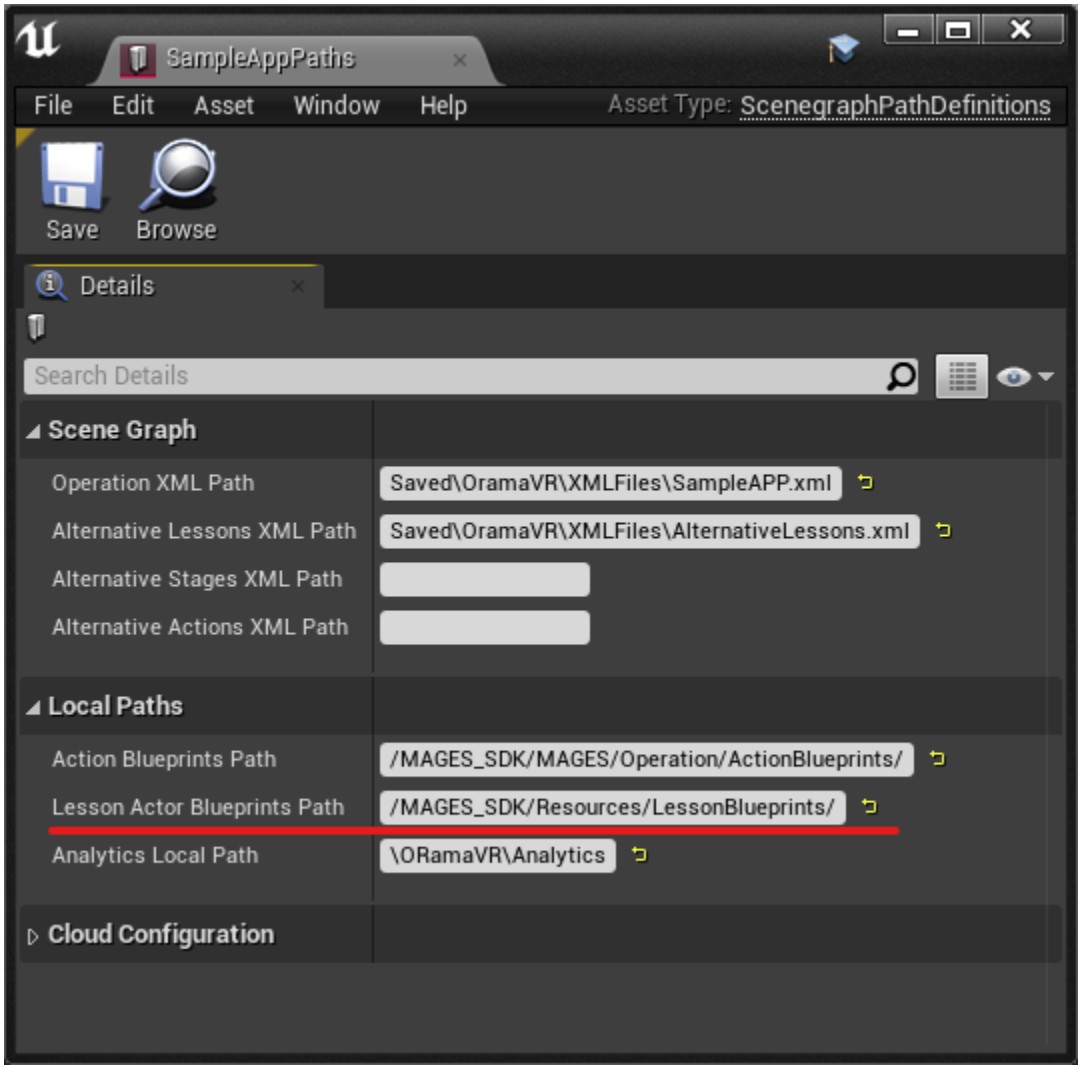


Every action blueprint is located under one general directory, and then furthermore organized by lesson, stage and action indices. You are **strongly advised** to keep the same structure for these assets.

Note: The base directory for action blueprints and lesson blueprints is configured in the `SceneGraphPathDefinitions` asset.

10.6.2 Lesson Prefabs

The provided Asset importer from the ORamaVR platform will spawn lesson blueprints (blueprints spawned with the `Spawn` function inside any action blueprint) using a base path prefix, specified in the paths asset.

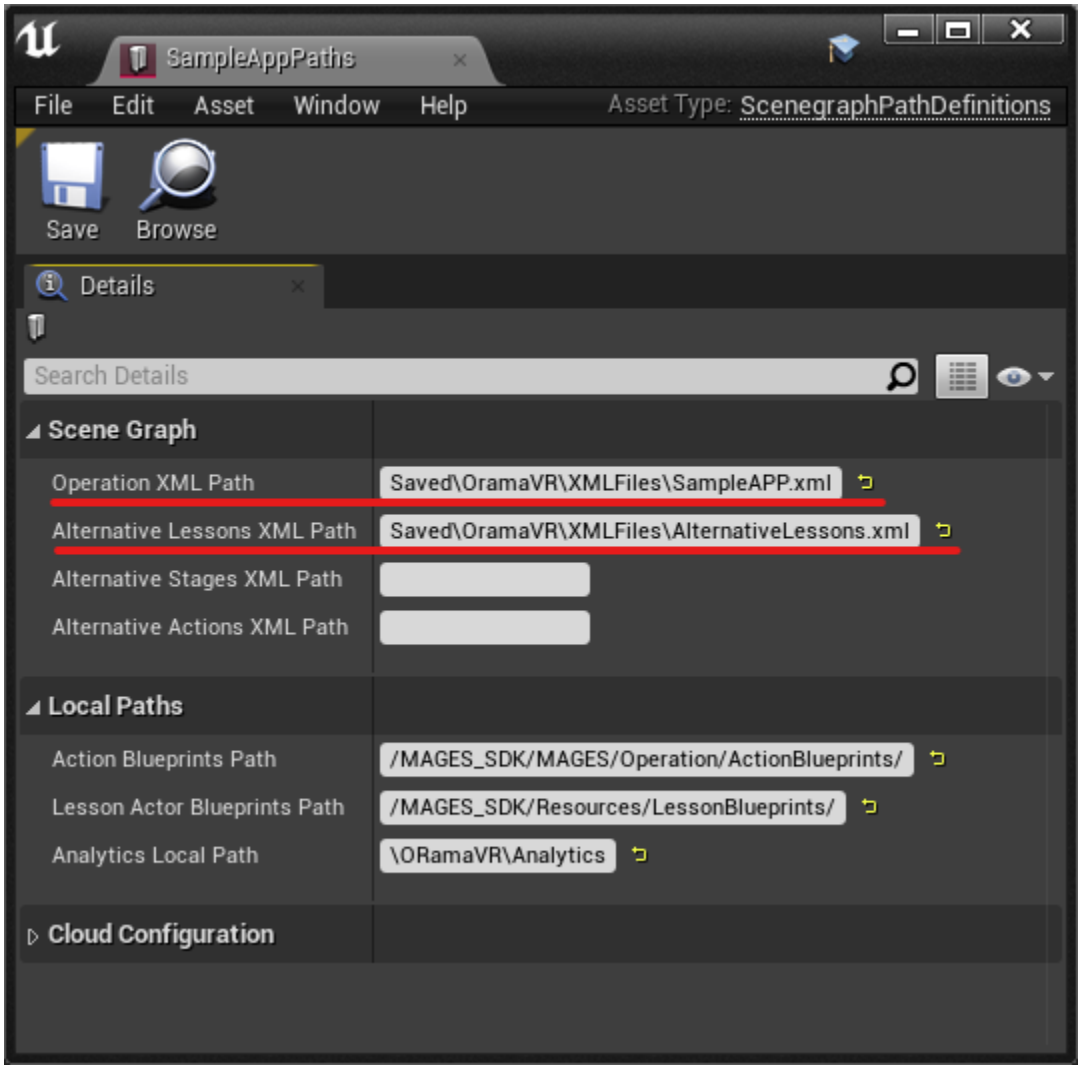


10.6.3 Storyboard XML Files

In the same manner, the locations of the storyboard XML files are specified in the paths asset. In the case seen below, this operation only needed the:

- Operation XML path (required for all training modules), and the
- Alternative Lessons XML path

So the remaining fields (`Alternative Stage XML Path` and `Alternative Actions XML Path`) are left empty.



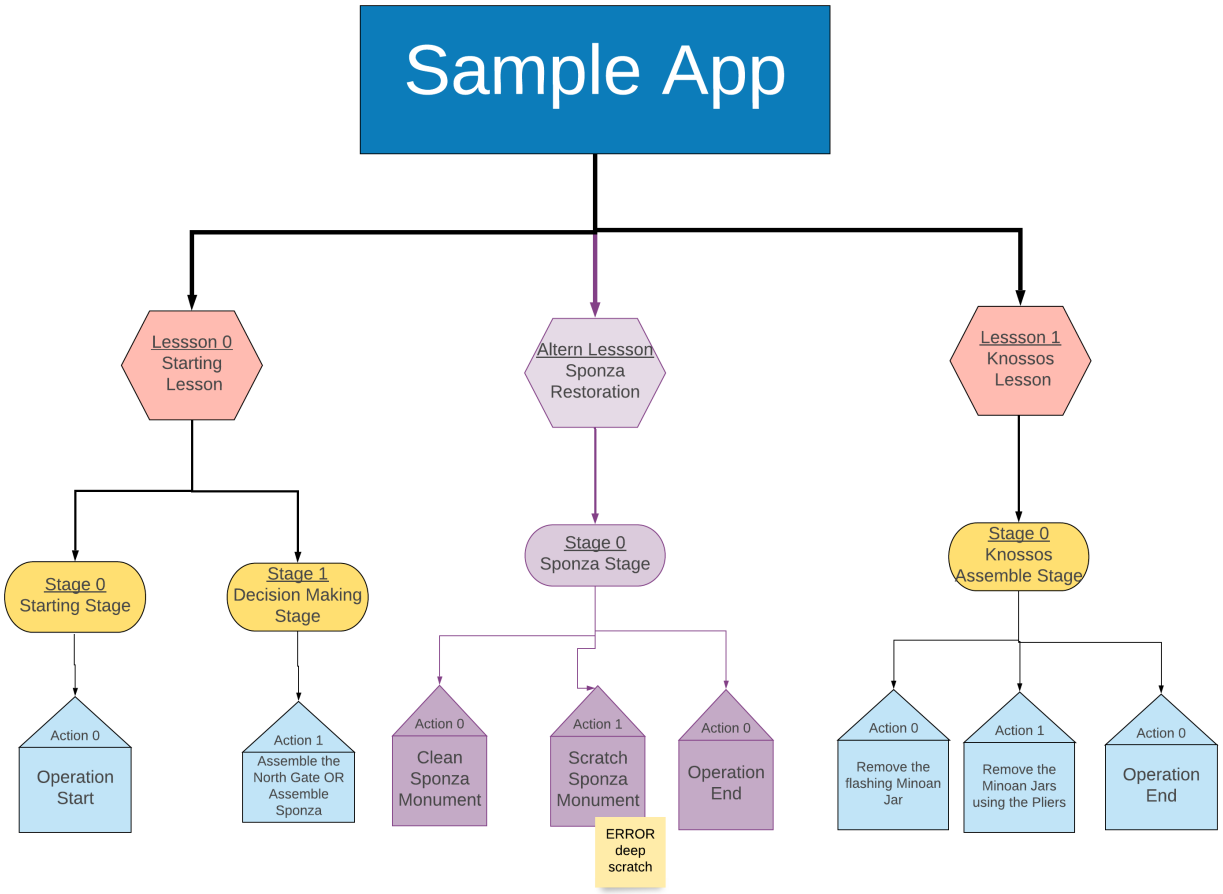
The same path is used for packaged applications.

10.7 Scenegraph

10.7.1 Introduction

Scenegraph is perhaps the most fundamental concept in MAGES SDK. It is this root module that powers and distinguishes ORamaVR's educational platform from the herd.

In this section, we proceed to present Scenegraph's architecture and how it structures all development.



The image above shows the transformation of a structured storyline into MAGES Live Scenegraph visual script.

Live Scenegraph Explained

In order to achieve a goal whether it is the restoration of a statue or a medical operation you need to follow a list of tasks/steps in a sequential order. We are referring to those steps as Actions.

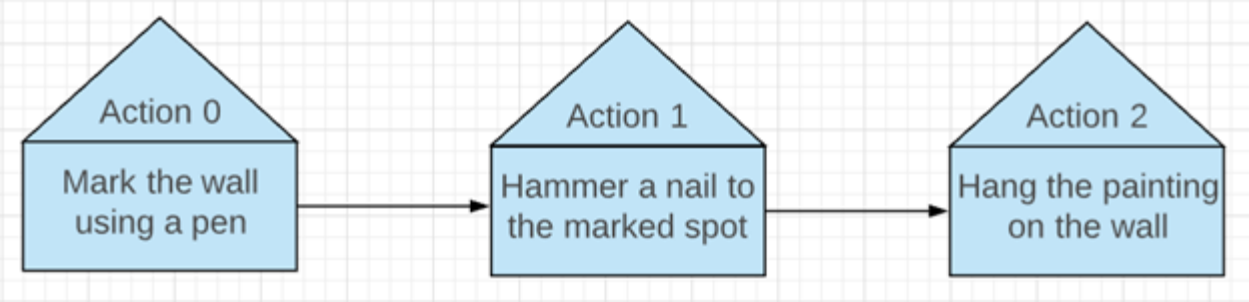
For instance, if we want to hang a painting on the wall we have to perform the following steps (Actions):

i. Mark the wall using a pen.

ii. Hammer a nail at the marked spot.

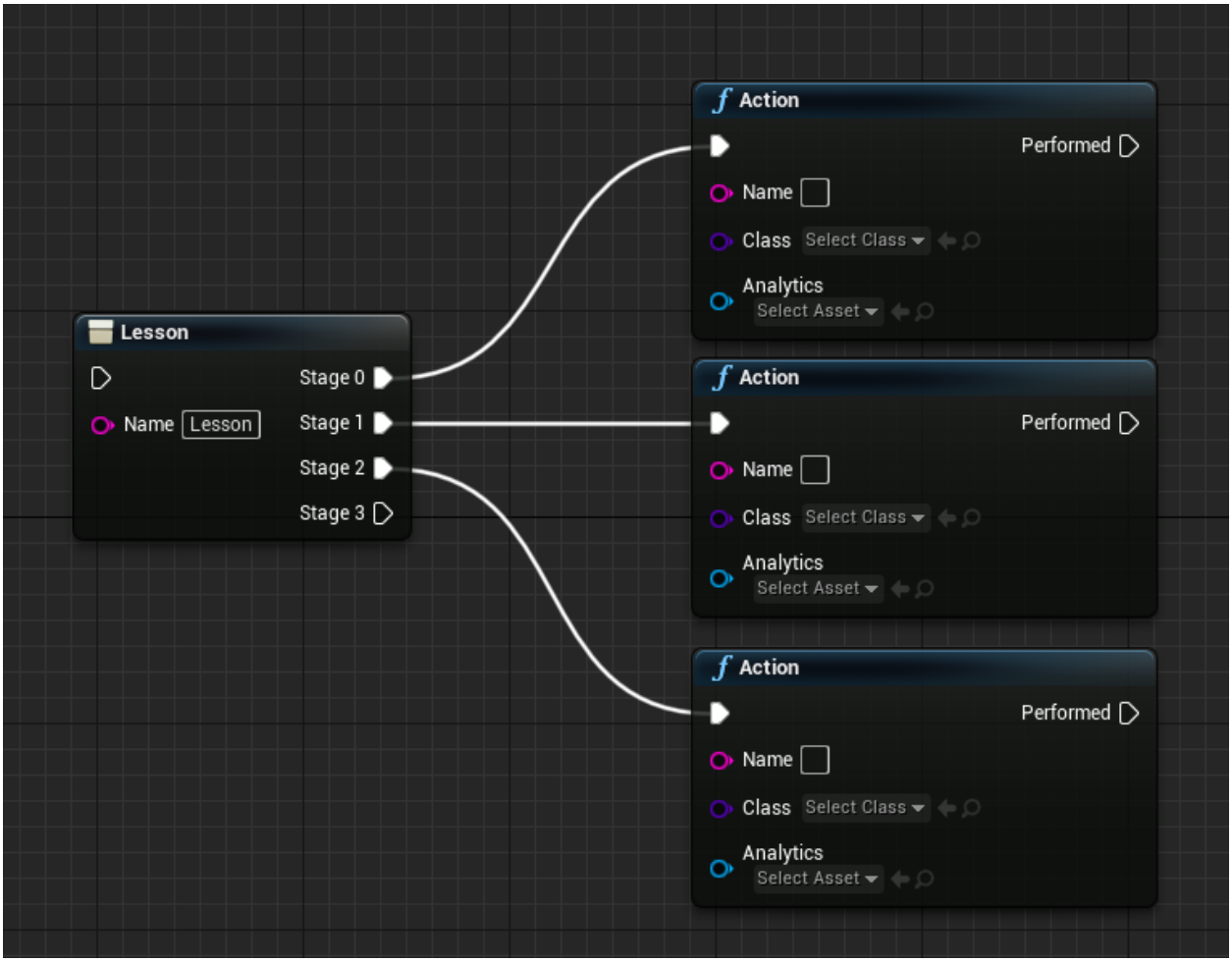
iii. Hang the painting on the wall.

Those are three steps that someone needs to complete to hang a painting on the wall. Having those steps in mind we create nodes, each one representing an Action.

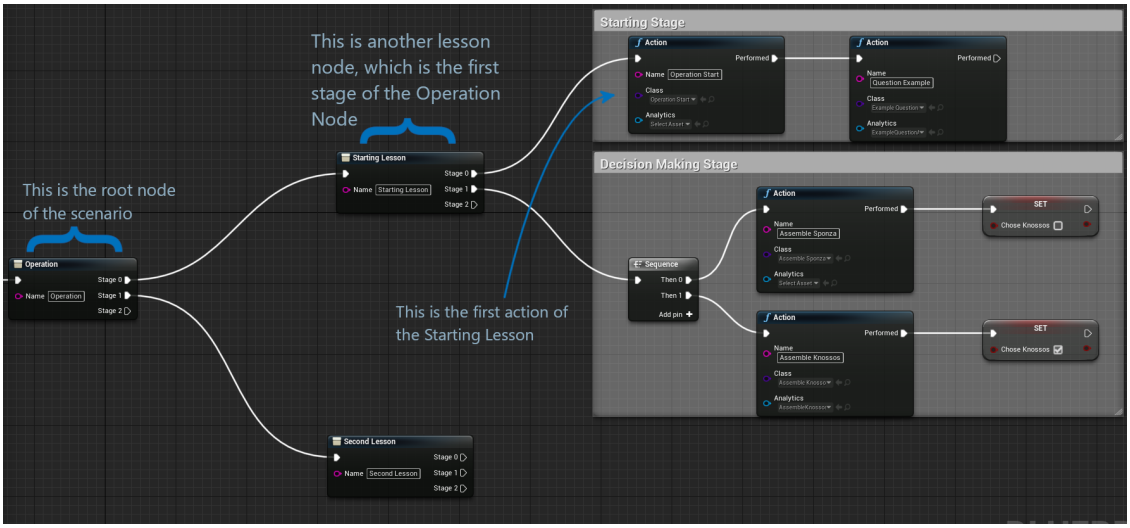


However in more complex applications there are dozens of Actions, in this case a sequential representation is not very convenient. For this reason we implemented the Scenegraph architecture.

A Scenegraph is a tree with varying levels of depth. The grouping node is called a lesson but really, it is up to the designer of the story to determine the segmentation and semantics of every lesson node.

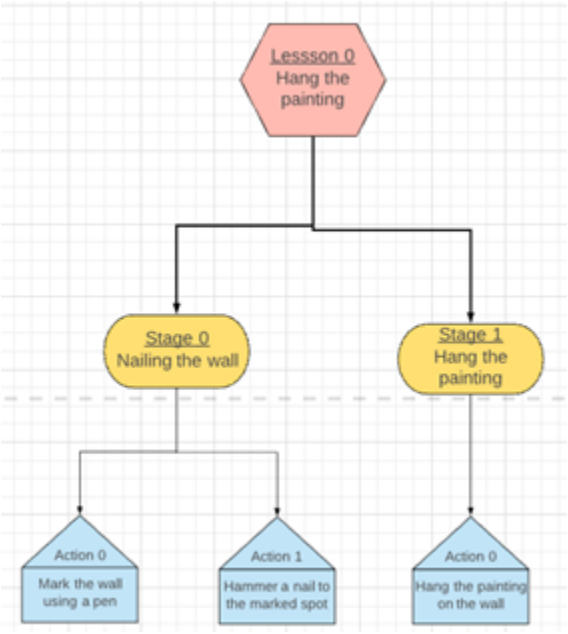


What follows is an example of how the lesson node is used to group sub-lessons or stages into other lessons:



The procedure runs only on Action nodes but we also use the other nodes in a tree format to merge parts of the simulating procedure.

For instance, we can present the above 3 Actions in a tree format as follows:



In this scenario we decided to group the first two Actions in a Stage since both of them are referring to steps that are linked with the nail. The last action can be placed alone in a stage particularly for this case.

After those optimizations, this lesson can be used in a more complex procedure with other lessons to construct a bigger Scenegraph tree. However, pay attention that even though we have multiple types of nodes (Lesson, Stage, Actions), only the Action nodes have customizable behavior.

Note: The operation runs only on Actions, lesson nodes are for traversal and scene management.

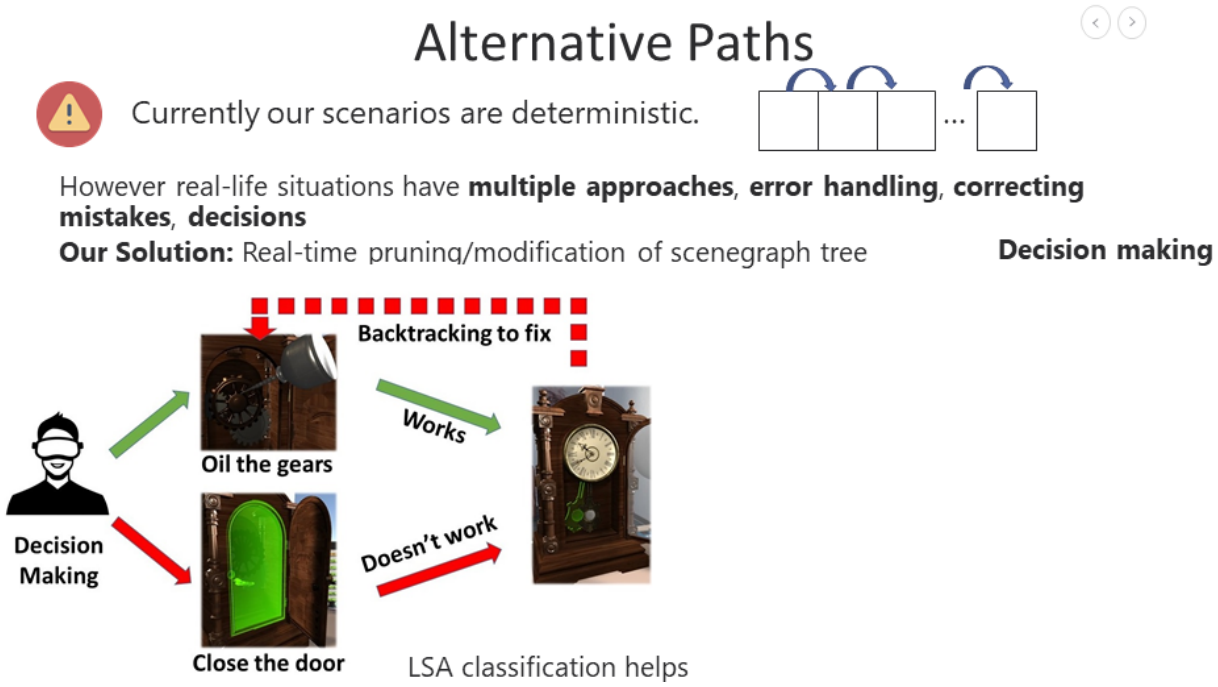
In Unreal, the scenegraph is a simple actor, spawned at the start of the application. This actor will spawn the relevant action nodes, when needed. Scenegraph will manage, perform and run all the Actions as an educational pipeline.

Any SceneGraph is a blueprint of the type `Mages Scene Graph`, and contains the logic and the actions that represent the whole scenario.

10.7.2 Logic & Decision Making

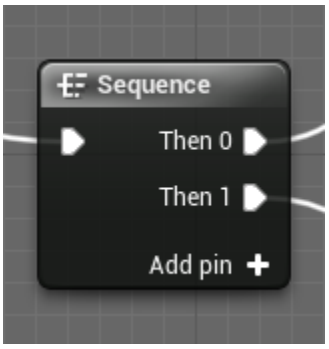
Scenegraph is not just a static tree, it's a dynamic graph. Since an educational pipeline can lead to multiple paths according to user's actions and decisions, Scenegraph does it so. There are times in a procedure where the user needs to choose between two predefined paths or an error they made leads to a completely different path.

These functionalities are implemented in such a way to support real time decision making and as a result Scenegraph can change its structure (Nodes) as the procedure goes on. Any logic node that is provided by Unreal for blueprints can be used to change the behavior of the graph according to the user's decisions.

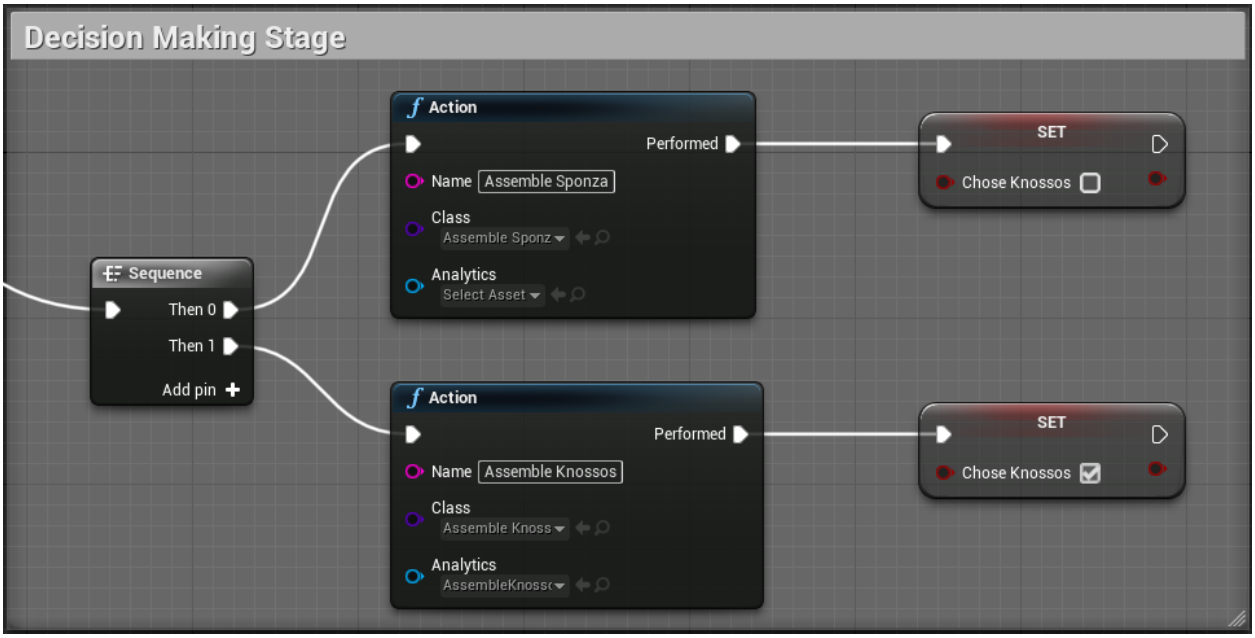


Alternative Paths - An Example

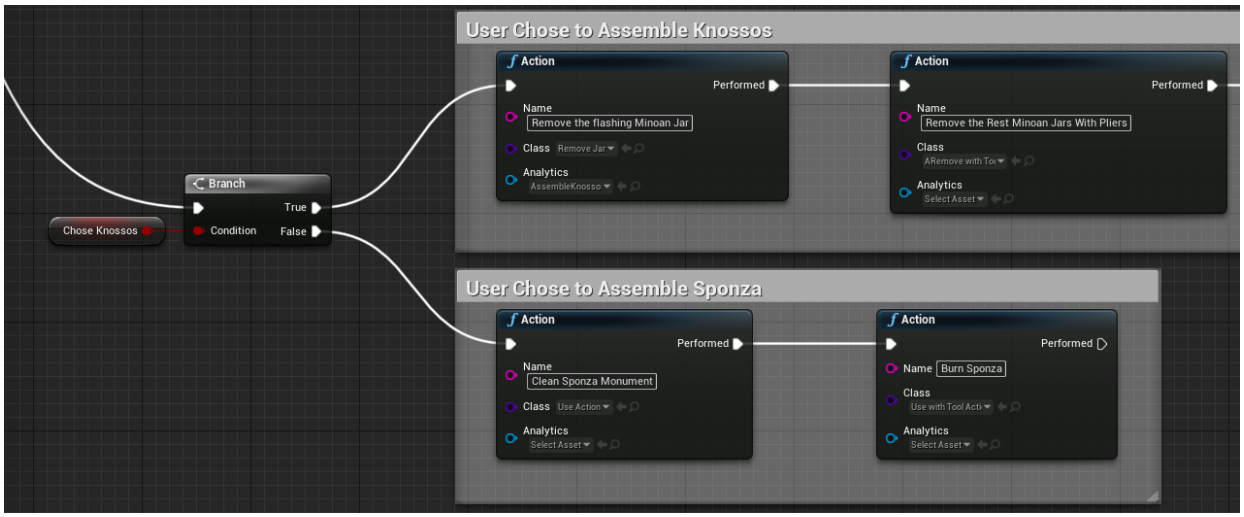
Since the Live Scene Graph is now a simple blueprint with the SDK's custom functionalities added on top, code logic can be applied to implement decision handling. The simplest manner in which this can be done is by making use of the Sequence node by Unreal.



The Sequence node, unlike the Lesson node (provided by the SDK), will not wait for the previous action or lesson group to be completed in order to execute the next stage. Instead, all actions within the sequence will be created simultaneously, but only one of them will execute logic after it has been performed by the user.



As illustrated above, if the user chooses to assemble Knossos by completing the corresponding action, the logic linked to the pin labelled "Performed" will be executed, setting the boolean variable of "Chose Knossos" to true. Afterwards, this boolean can be examined by the graph, with different logic for either case:



10.8 Prefab Constructors

10.8.1 Introduction

Almost all types of Actions inside the project are prototyped because they share a lot of behavioral elements. The same idea is applied to blueprints & actors.

Their fundamental behavior can be split in a small amount of different Constructors. Depending on the Constructor attached, the creation for each Action actor differs.

In the following sections you can find tutorials on every available Constructor:

1. *Generic Prefab Constructor*
2. *Interactable Prefab Constructor*
3. *Interactable Final Placement Prefab Constructor*
4. *Use Collider Prefab Constructor*
5. *Collision Hit Prefab Constructor*
6. *Remove With Tools Prefab Constructor*
7. *Question Prefab Constructor*

10.8.2 Generic Prefab Constructor

Initially, this script should not be attached to any gameobject as it is the base of all different type of constructors.

It contains selections (and functions) for the developer that are going to be found in any type of prefab constructor:

Variable Name	Type	Description
Prefab Perform Action	Destroy, Remain	Select if this actor after action completion should remain in the scene or if it should be destroyed.
Prefab Type	Generic, Interactable, InteractableWithParent, InteractableFinalPlacement, UseActionCollider, Question, CollisionHit, AnimationPrefab, Pump	Select the type of usage for this actor. Based on the selection here this actor will be initialized differently.

Function Name	Description
public virtual void ResetPrefab()	Resets the prefab to its starting position and rotation.
public virtual void SetNewPrefabStartingTransform()	Changes the prefabs starting position and rotation from the values the gameobject had on spawn to the values the gameobject has on the time this function is called.
public virtual void FinalizePrefabAction()	Internal function called on Action end to finalize prefabs behavior. Can be overridden to add to the prefab functionalities.

10.8.3 Interactable Prefab Constructor

Inherits from Generic Prefab Constructor. This constructor should be applied to any actor that will be interacted from the user and has physics properties.

Variable Name	Type	Description
Prefab Interactable Type	Generic, Insert, Remove, Remove with tool	Select for what type of action this actor will be used.
Prefab Detach Feature	ReInitialize, Destroy, EventTrigger, CurrLSAOnDestroy, Nothing	Select what the prefab should do when the user throws it away. Reset the prefab, destroy it or destroy it and call the Event Manager to trigger the Action Completion.
Wait for Drop Action	float	Set a time delay that the prefab detach feature action will be called after the user drops the interactable object.

Actor Creation Requirements

1. A primitive component. This can be a skeletal mesh, a static mesh or a shape collider (box, capsule, etc..).
2. In the primitive's component physics section `Simulate Physics` should be enabled
3. In the primitive component collision section `Generate Overlap Events` should be enabled, as well as `Collision Enabled` should be set to `Query Only` or `CollisionEnabled`
4. MagesInteractableItem component attached

Warning: If the mass of the primitive component is above 100kg the virtual hand will not be able to lift the object.

10.8.4 Interactable Final Placement Prefab Constructor

This script is attached to an actor that's duplicated from the interactable item and it serves the role of the final placement. The way it works is that it has an overlap collider and when the collision registers with the other interactable actor it observes their transform. If their transform difference is below a specified margin (the rotations match to a certain point), the collision gets accepted and the action performs.

This actor will -on start- have all its renderers disabled and it will be awaiting an overlap event. When the collision succeeds, it detaches the interactable actor from the user's hands, it translates it to the position (and rotates it) the final actor is. When they are at the same position the actor that the user had at hand gets destroyed and the final actor enables its renderers.

When all of this it's done the final behavior of this actor is called (e.g. some specific animation after the actor is placed) and it triggers the Event Manager for the Action completion.

Prefab Creation Requirements

1. Overlap Colliders
2. Prefal Lerb Placement component
3. In the primitive component collision section `Generate Overlap Events` should be enabled, as well as `Collision Enabled` should be set to `Query Only`

Prefab Lerp Placement

Everything explained above is the works of this component, PrefabLerpPlacement. The constructor observes if the prefab has this component attached and it initializes it accordingly. This component can also be used on its own anywhere for lerping between two transforms using Dual Quaternions.

Variable Name	Type	Description
Max Angle Degree	float	Observes the rotations of the final placement and the interactable prefabs. If their difference in all three axis is less that this given margin, it accepts the collision. A value of zero means that any rotation is accepted.
Lerp Difference	float	The max fault tolerance for the interpolation. A higher value leads to better performance but might lead to inaccurate results.
Lerp Speed	float	Speed multiplier for how fast the interactable object should reach the final position/rotation.
Interactable Prefabs	list of actors	Insert here all the actors that this prefab is going to await collision with.

10.8.5 Use Collider Prefab Constructor

The idea behind this prefab is similar to the one explained in the 5.5. This component should be attached to prefabs that contain by themselves a collider that when triggered with specific actors it triggers the Event Manager for the Action completion. In addition, based on the desired usage of the interactable different configuration is needed. The difference here is that the actors for the collision are type-unrestricted. Anything can be inserted for the collision to be accepted.

Variable Name	Type	Description
Stay Time	float	In case of `collider trigger` set to simple or use with tool, set the time needed for the collider to register a successful collision.
Prefabs Used	List of actors	Insert the actors that the collider will await collision with.
Hit Times	int	In case of `collider trigger` set to hit, set the amount of successful collisions needed.
Hit force	float	In case of `collider trigger` set to hit, set the amount of force needed to register a successful collision.
Hit Movement Vector	Vector3	In case of `collider trigger` set to hit, after each successful hit, the offset that will be applied to the primitive component.
Proceed Animation on Collider Hit	boolean	If enabled, iIn each successful collision play the next animation from the animation list `Animation Names`.
Proceed Animation on Perform	boolean	If enabled, on action's perform the next animation from the `CharacterAnimationController` will be played.
Animation Names	List of strings (animation names)	The names of animation assets that will be played on each successful collision hit. These should referenced be in the `CharacterAnimationController` component.
Character Actor Name	string	The name of the actor that has the `CharacterAnimationController` component. This is usually the actor that has the patient's skeletal mesh component.
Promote Collider Components	boolean	If false, all components under the collider being hit(destroyed) will be destroyed as well. Otherwise, the child components will take its place when destroyed
Collider Trigger	Simple, UseActionCollider, Hit	Specify the behaviour needed from the user in order to register a successful collision.

Prefab Creation Requirements

1. Primitive component
2. Overlap Collider(s)

10.8.6 Question Prefab Constructor

The concept behind this component is to have the user make a decision. This component should be attached to the question lesson blueprint. This component handles user interaction with the answers given to the question.

Constructor Properties

Using the properties specified below, the constructor will dynamically generate the Question UI upon initialization.

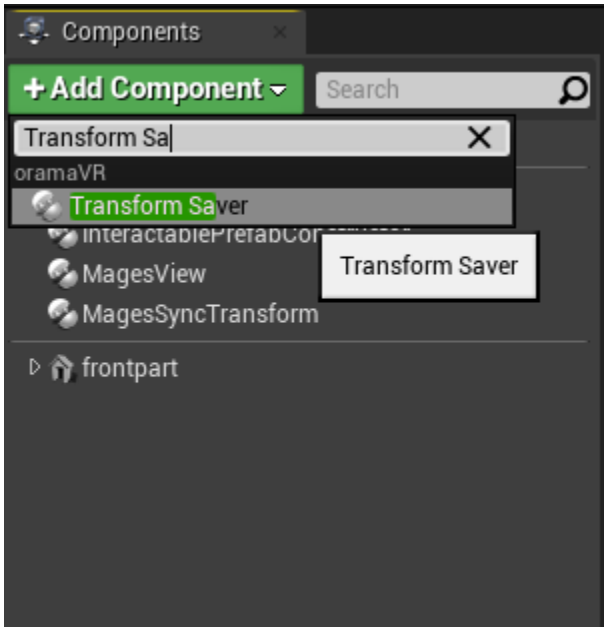
Property	Type	Description
Text Header Lifetime	Number (float)	If Text Header is enabled, Set Text lifetime.
Header Widget Class	Mages Text Widget	Header widget class for option UI
Header Widget Size	Vector 2D	Size of the Header widget
Max Questions	Number (integer)	Number of maximum permitted questions. For single selection UI set 1
Option List	Array of Options	Designates the different available answers for the question
Option Widget Class	Mages Question Button Widget	The Widget class to use for the buttons
Option Widget Size	Vector 2D	Size of the button widgets
Option Widget Pivot	Vector 2D	Pivot of the button widgets
Submit Widget Class	Mages Button Widget	Button widget class for the submission button
Submit Widget Size	Vector 2D	Size for the submission button
Submit Widget Pivot	Vector 2D	Pivot for the submission button
Shuffle Options	Boolean	Set to true for options shuffling (randomly changes order of the answers)
Reveal Correct Answers	Boolean	Set to true to reveal at the end the correct answers

10.8.7 Transform Saver

Blueprints, by default, do not save their transform values in the content browser. The `Transform Saver` component does just that. It is present on *every* lesson blueprint, since they are spawned dynamically.

Usage

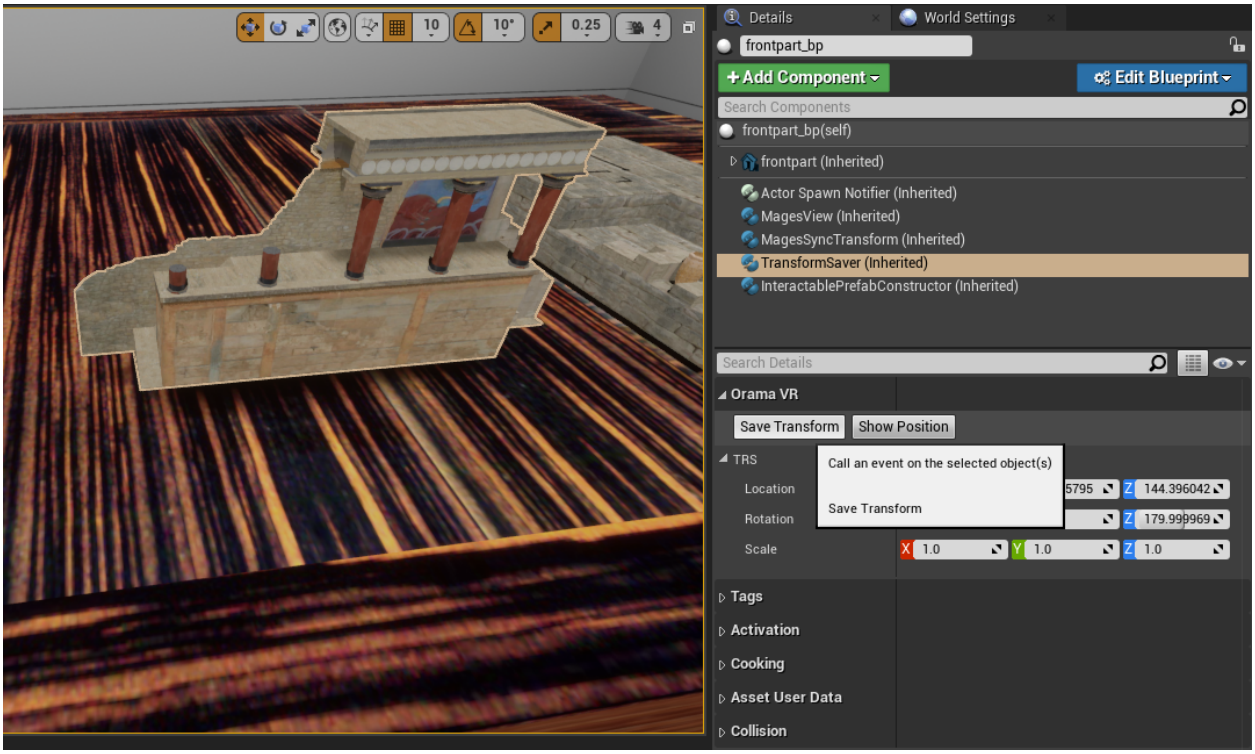
Inside a blueprint, click on the **Add Component** button and search for "Transform Saver".



Having added this component, anytime this blueprint is spawned through the MAGES SDK Spawn methods, the transform stored inside this component will be used.

In order to change the saved transform of the blueprint, drag it inside the level, and place it where'd you'd like it to be spawned.

After you are satisfied with its transform, select the `Transform Saver` component that was added previously and click "Save Transform".



Afterwards, click on **Edit Blueprint** and choose "Apply Instance Changes to Blueprint". The transform (position, rotation and scale) of this object have now been saved to the blueprint inside the content browser.

10.8.8 Actor Spawn Notifier

This component adds a flashing effect to any mesh components inside an actor, for a set time. It is primarily used to notify the user about a newly spawned object that they can grab.

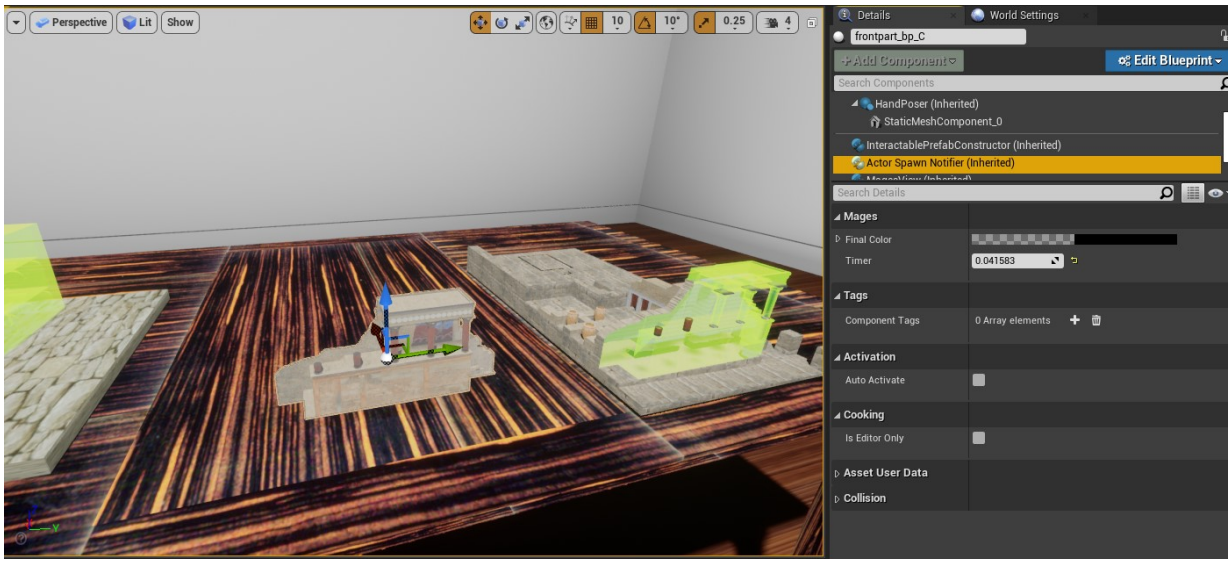


Fig. 1: Notice that the object is flashing. This is the Actor Spawn Notifier changing the object's emission property dynamically. After a predetermined amount of time, this component will be destroyed.

This component is automatically added to any spawned actor that has an *Interactable*

Prefab Constructor component, so you do not need to add it manually in most cases.

10.8.9 Pump Prefab Constructor

The Pump Prefab Constructor component is used to perform a *pump action*.

Constructor Properties

Property	Type	Description
Pumps To Perform	Integer	How many pumps to the prefab needs for completion
Pump Mode	PumpMode	FullPump: On trigger button clicked, HalfPump: On Trigger button down
Pause Animation On Perform	Boolean	True: The animation will lock in last frame when performed, False: The animation will return to initial position
Vibration	Float	Adds continuous vibration when trigger button pressed
Reverse Animation	Boolean	Plays the animation in reverse (instead of recording a new one)
Play Sound	Boolean	Plays sound when the trigger button is pressed (Better results with short clips)
Continuous	Boolean	For continuous interaction (time)
Press Trigger	Boolean	For continuous interaction only. Specifies if the animation plays automatically or when trigger is pressed.
Enable Range Mode	Boolean	Enable the functionality to perform between two values of the input percentage of the trigger
Low End To Perform	Float	Define the low percentage that above this value will perform after remaining for stayTime seconds
High End To Perform	Float	Define the high percentage that above this value will perform after remaining for stayTime seconds
Stay Time	Float	Define how much time the trigger needs to stay within the low and high values to perform
Right Hand Animation	Animation Asset	Animation to play when interacting with the right hand and pressing the trigger button
Left Hand Animation	Animation Asset	Animation to play when interacting with the left hand and pressing the trigger button
External Actor Label	String	The actor that contains the external skeletal mesh component. This animation will share the same progress as the action.

10.9 MAGES UI

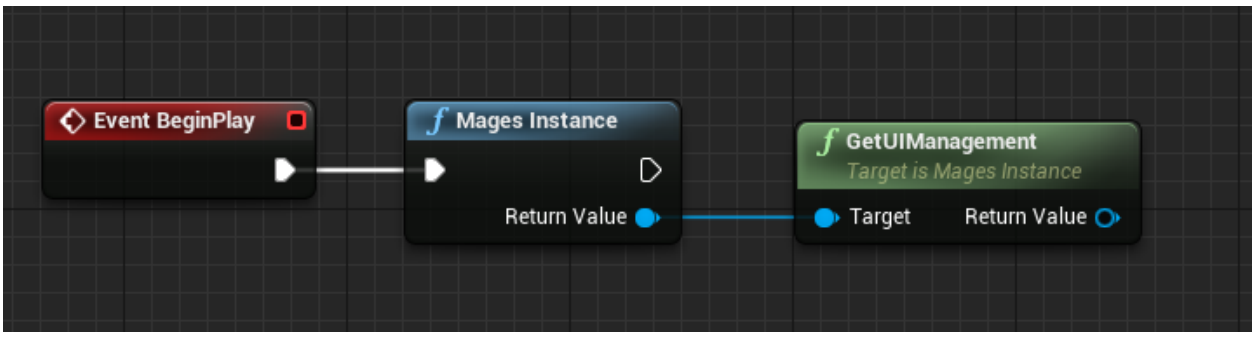
Our User Interface is built on top of Unreal Engine's UMG; we only make customizations when it is absolutely necessary to do so, like our `Mages Button Widget` class.

Every widget designated to be used within the SDK inherits from the Mages Widget class. Additionally, it is used inside a `Mages Widget` component instead of the `Widget Component` to enable some extra functionality.

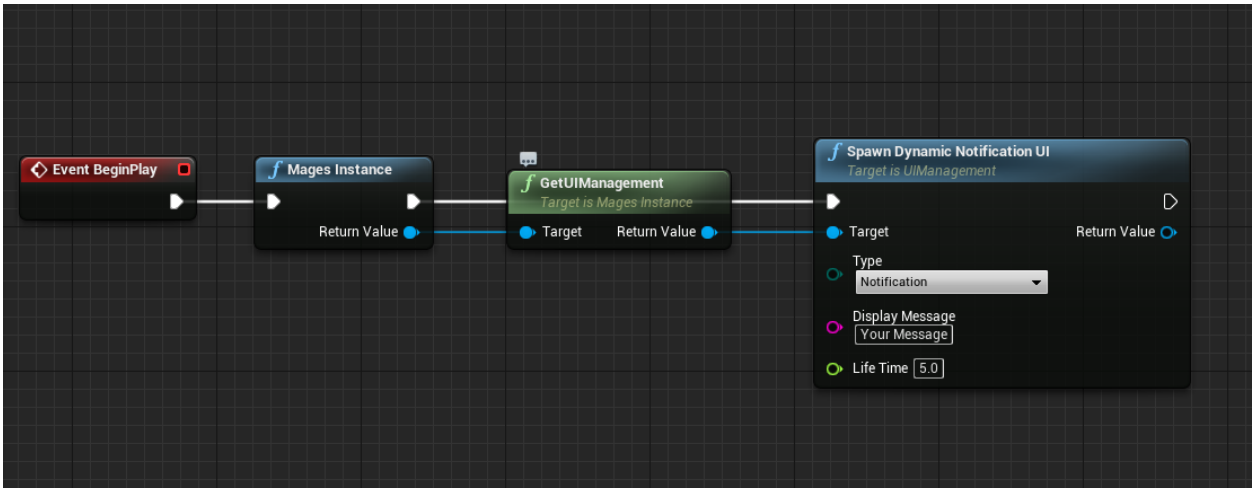
10.9.1 UI Notifications

Some subset of user-feedback for their actions cannot be implemented though controller rumble, or subtle queues in general, so a textual notification is sometimes necessary. For this reason, we have a dynamic notification management subsystem.

To access these functions, you need to get the UIManagement object from the *Mages Instance*:



In order to spawn a notification, call the `Spawn Dynamic Notification UI` function:



This will place a notification object in front of the user, and adjust it dynamically, should their position change so that it is not seen. There are 4 types of notifications:

- Notification: Used for basic information
- Warning: Usually used for non-action specific checks, i.e. dropping fragile items on the floor
- Error: Used for errors during an action
- CriticalError: Used for errors that would have the operation be aborted in a real-life scenario

The `Life Time` parameter will close the notification after the specified amount of seconds; set it to 0 for the default lifetime.

10.10 Deformations

10.10.1 Softbodies

Introduction

Realistic deformations play an important role in computer graphics, games, simulations and VR environments.

Soft body simulations are used to change an object shape, when external forces are applied.

The computation of physically accurate deformation of objects when VR users uses hands controllers to interact is a liturgy, which requires much computation power. Only a few applications and simulations use soft body deformation due to computation power needed.

MAGES™ SDK & Soft bodies

In MAGES™ SDK, we provide a novel soft mesh deformation algorithm suitable for Virtual Reality interaction and collaboration.

The soft deformation algorithm is based on shape matching techniques and particle based spring mass soft body simulations. Our particle-based soft body algorithm is different from the state of the art because it provides easy control of the particles as physical objects and a center point, which controls the entire soft body position.

Velocity based interaction can be applied directly to our particles while as physical objects can interact also with the environment.

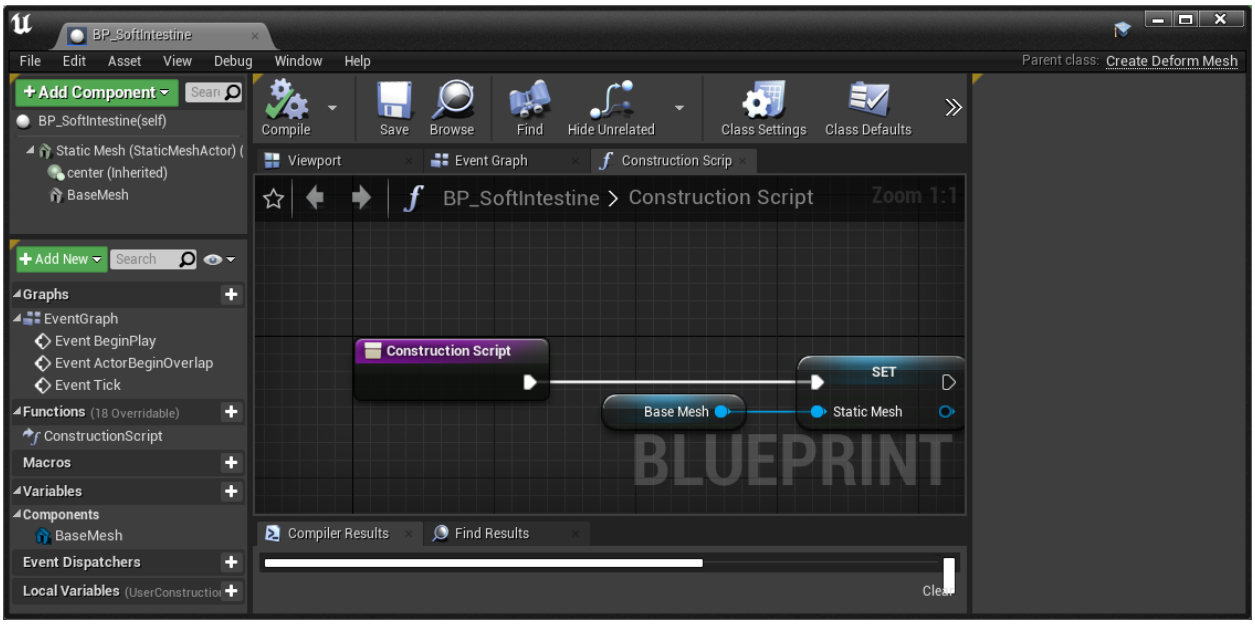
Our Virtual Reality interaction system uses velocity base approach providing the ability to pick up, hold and drop objects. Due to our soft body particles nature, this interaction can be applied directly.

How-To

In MAGES Unreal, an example for a soft-body enabled intestine is located under "MAGES SDK Content/SDKAddons/Softbodies".

Any soft-body blueprint is derived from the `CreateDeformMesh` native class. It requires a static mesh component (the actual soft body) from which to copy transformation and mesh data.

Inside the construction script of the blueprint, we set the Static Mesh property to the component instance:



On Begin Play, the mesh will be initialized with soft body interaction:

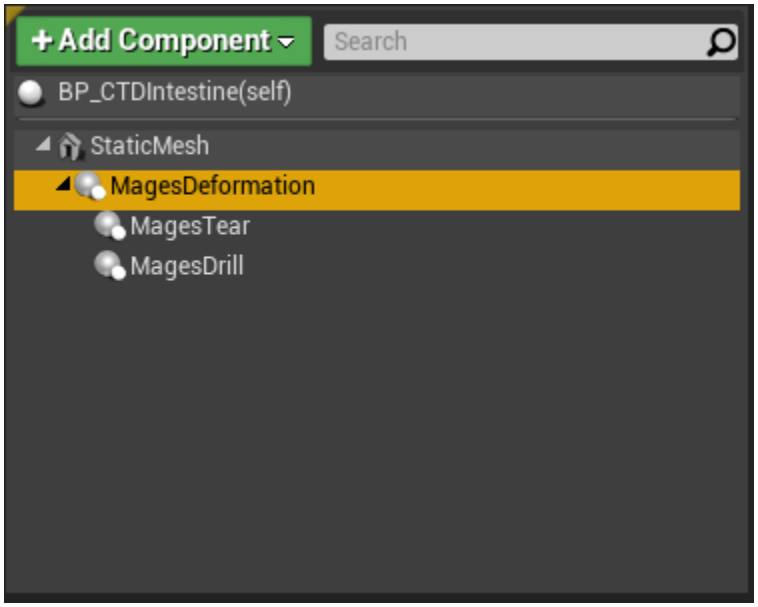
The green spheres represent the actual simulated particles of the softbody

10.10.2 Deformation Component

Any Static Mesh Component can be modified by the CTD algorithms, using the `Mages Deformation` component. This provides a unified access method to the mesh data, so that they may be updated in a simple and fast manner.

Creating a Deformable Mesh

To create a deformable mesh from a static mesh component, add the deformation component under the static mesh:

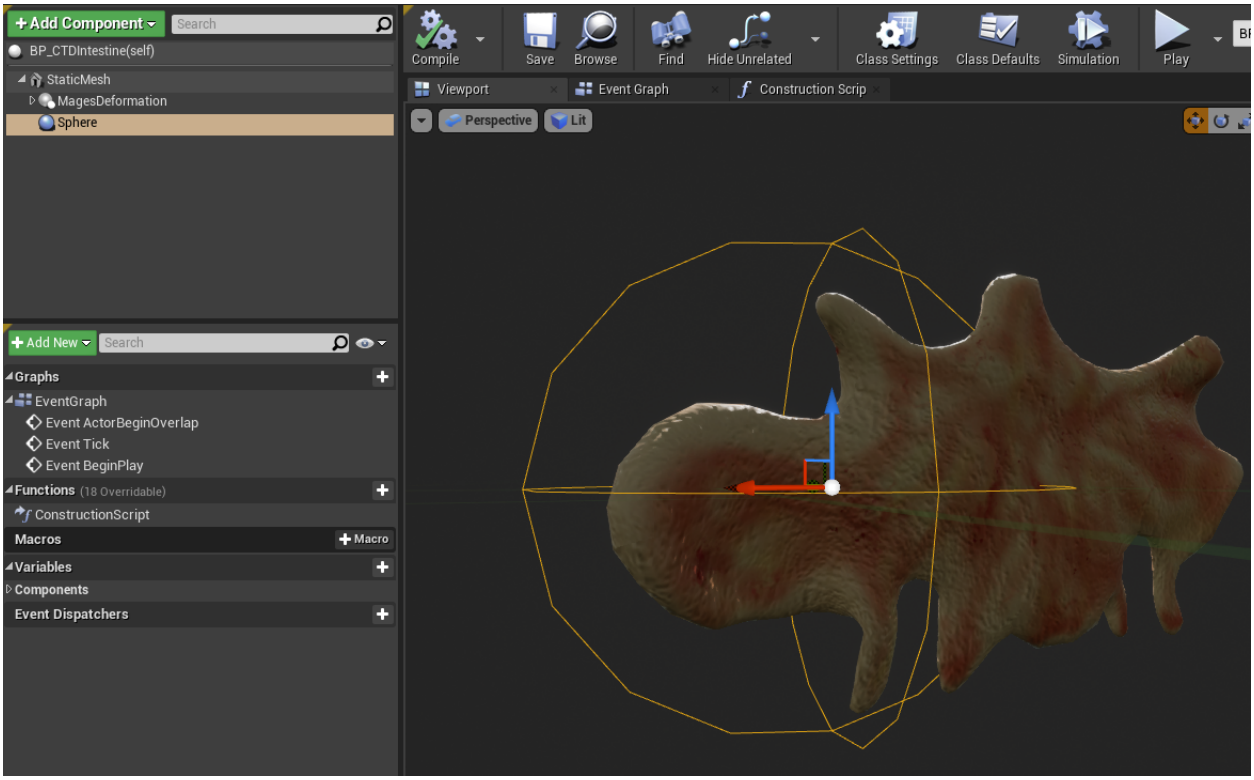


Mesh Predicates

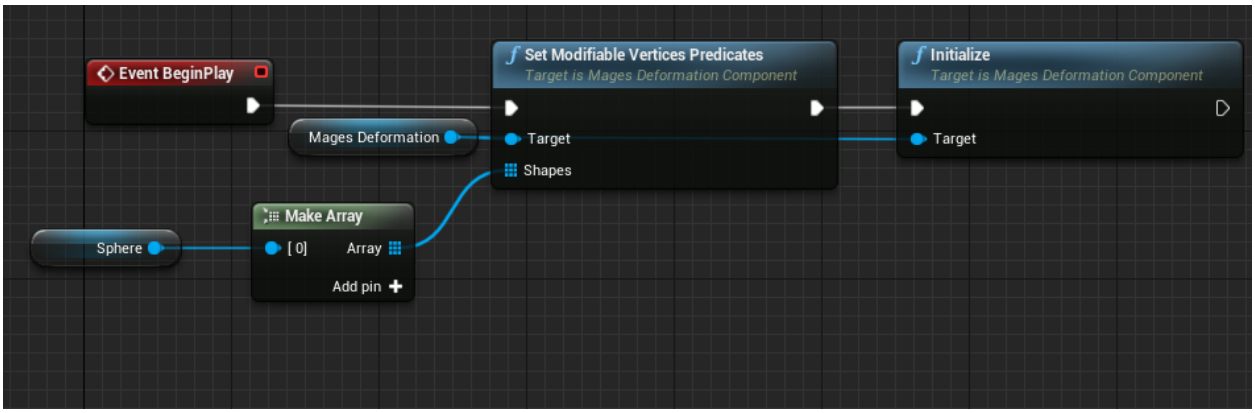
The whole mesh has now been converted into a deformable mesh, and can have ctd operations applied on to it. For optimization purposes, we can choose to split the mesh in multiple sections according to our predicate system. This can be done by adding shapes (spheres, boxes, capsules) and using them to construct the mesh.

As an example, we'll use one predicate (a Sphere Collision component) to separate the mesh into two sections.

First, we add the predicate to the mesh, as shown below:

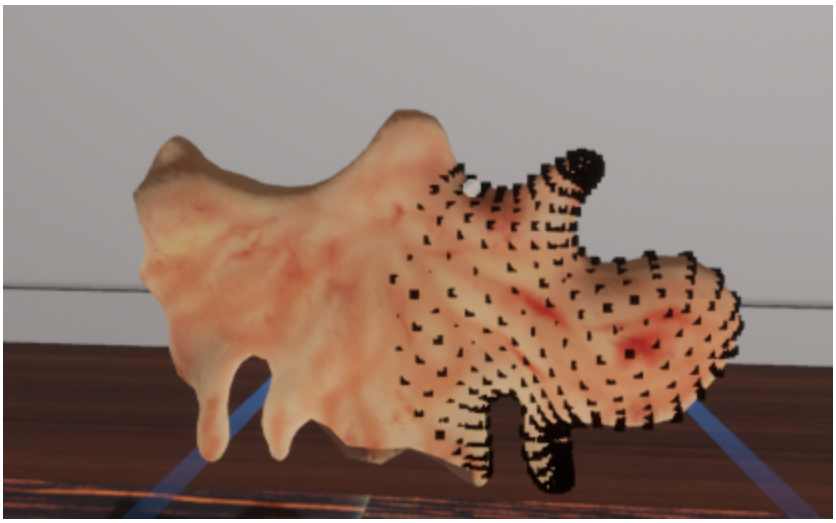


Then, in the Begin Play event, we call the relevant functions to declare the predicates, and then initialize the deformable mesh:



Warning: If you are using predicates, make sure to turn off "Initialize on Begin Play" in the Mages Deformation component properties.

If we've checked "Enable Debug" on the Mages Deformation component, the we can see that the new mesh section we've defined has black rectangles on its vertices.

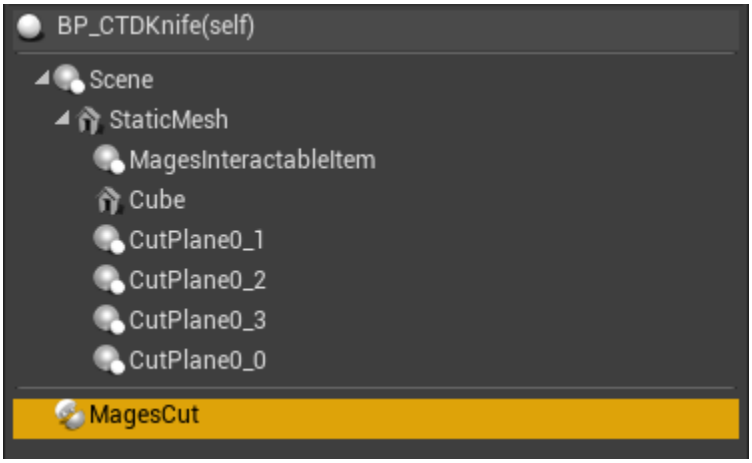


10.10.3 Cut

The cut component can be attached to tools, and will cut the mesh according to a plane, defined in world space. The left section of the mesh will be discarded, while the right section will remain

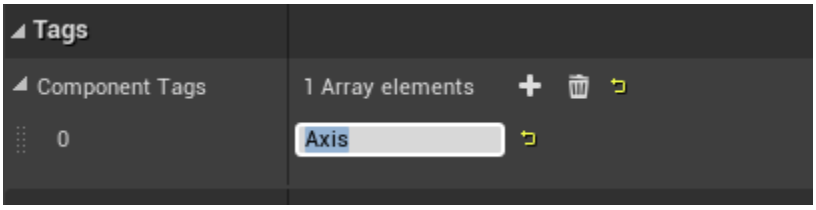
Set up

Inside the tool blueprint, add the `Mages Cut` component:



To setup the cut plane, we add 4 new components named: * “CutPlane0_0” * “CutPlane0_1” * “CutPlane0_2” * “CutPlane0_3”

All of these scene components need to include the tag “Axis” in their component tags



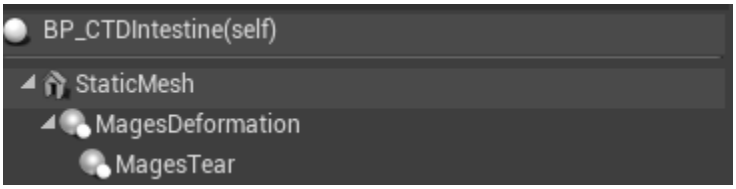
That’s it! Make sure to add some sort of overlap collider to trigger the cut.

10.10.4 Tear

The Tear operator splits the mesh in a manner similar to how a scalpel would work in the real world.

Set up

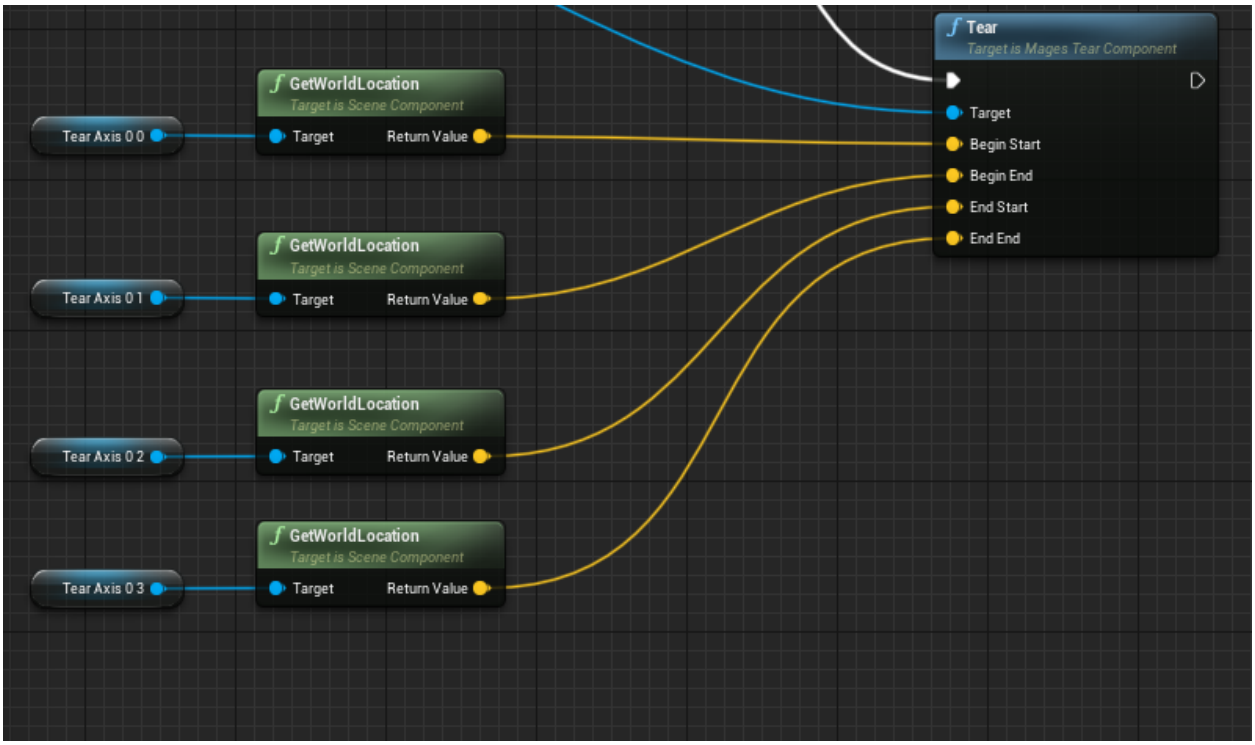
In the deformable mesh's blueprint, add the `Mages Tear` component under the deformation component to enable tearing on the mesh:



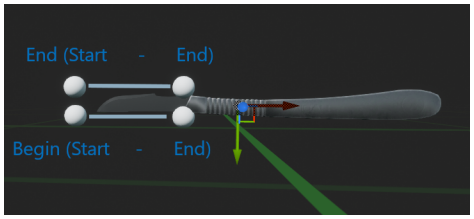
The "Alpha" property designates by how much the section will "open up"

Performing Tear

The `Mages Tear` component exposes a single method to blueprints, named "Tear". This takes in four location parameters, which specify a plane (the Tear plane), which is used to apply the tear operation. Inside the `BP_CTDScalpel` tool, we use four scene components to determine the tear plane:



And this is how they look inside the tool's viewport:

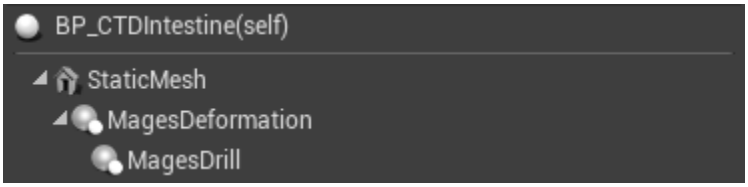


10.10.5 Drill

The Drill operator can be used to create cylindrical wholes inside a deformable mesh object.

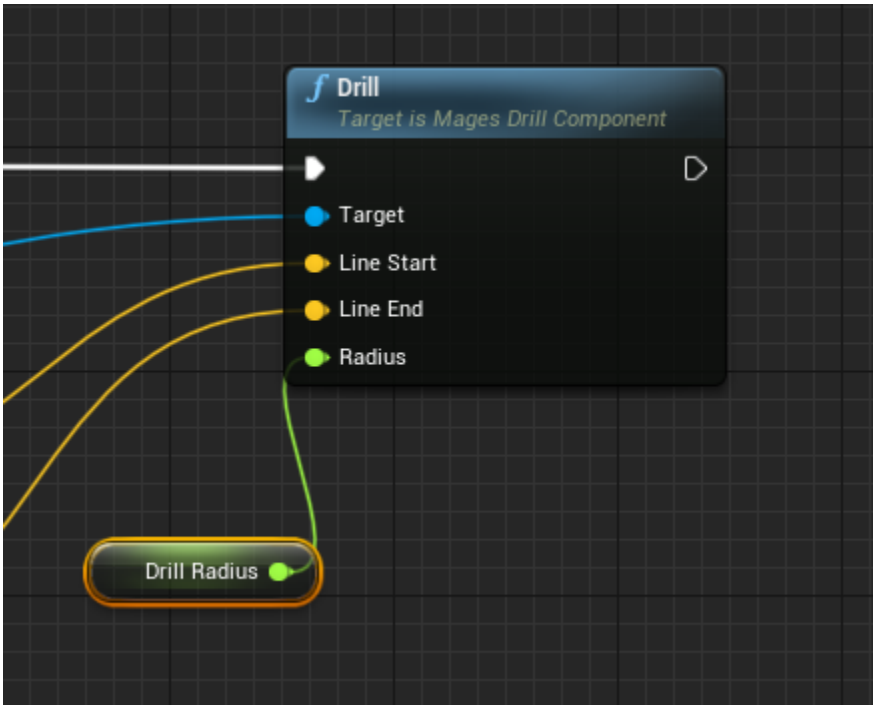
Set up

In the deformable mesh's blueprint, add the `Mages Drill` component under the deformation component to enable drilling on the mesh:



Performing Drill

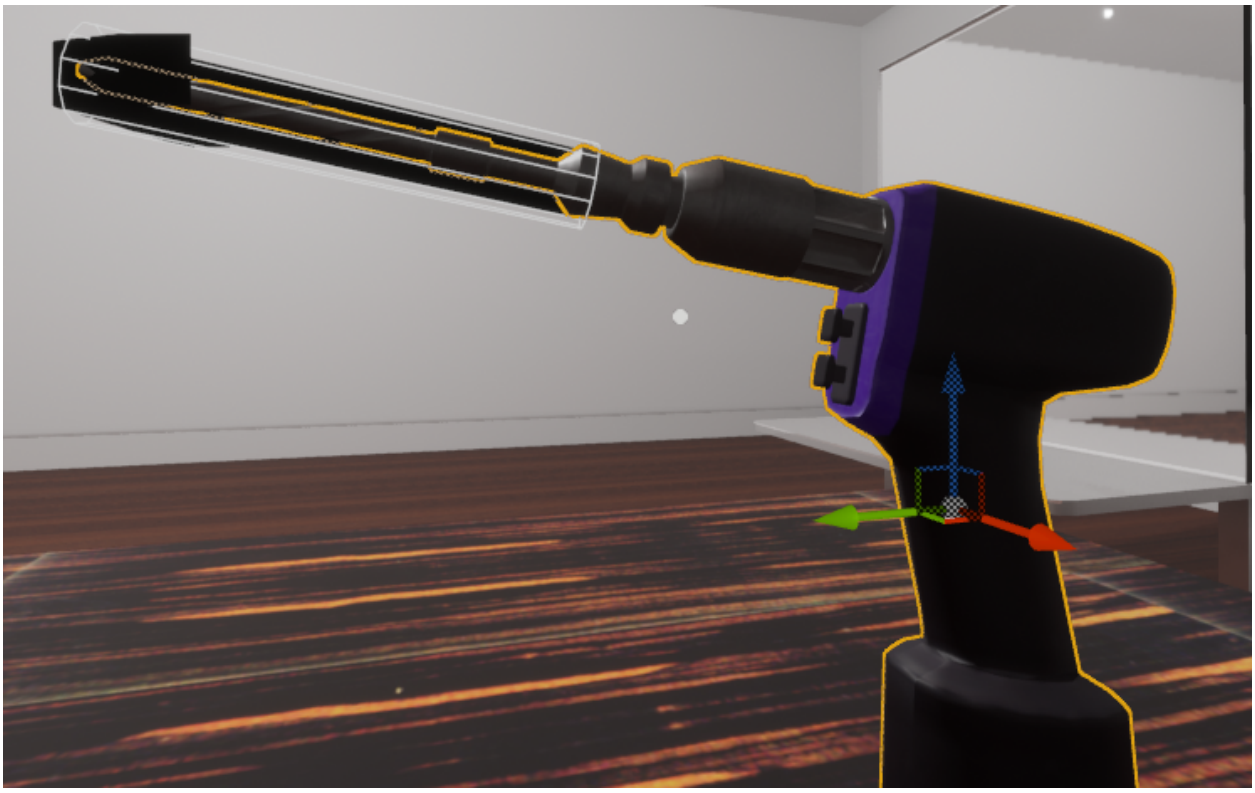
The drill component exposes once function, named "Drill", which takes in 3 parameters that define a cylinder in world space:



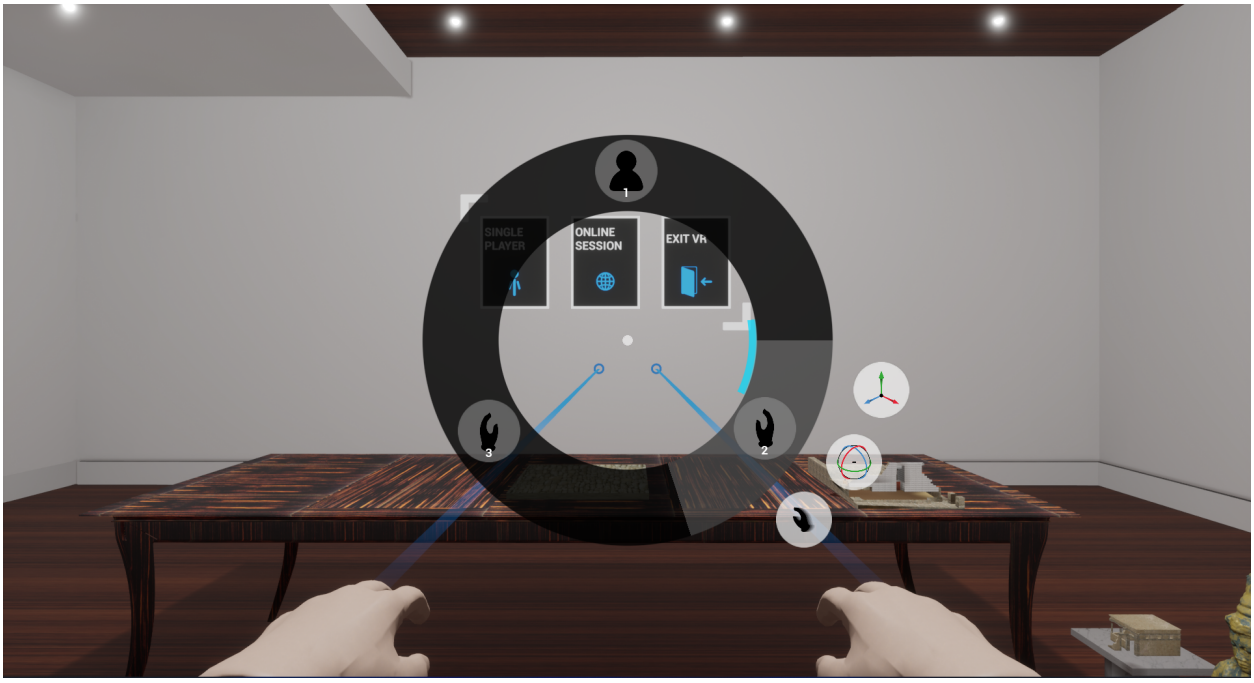
- Line Start: The start location of the cylinder; this would be the drill's base

- Line End: The end location of the cylinder; this would be the tip of the drill
- Radius: The radius of the drilling cylinder

These can be defined and extracted from the tool blueprint that may be used (see `BP_CTDDrill`):



10.11 Desktop 3D Controller



10.11.1 Intro

The Desktop 3D controller is used as the default pawn (camera) for all three different operations provided with MAGES Unreal. It is a useful development tool for accelerating iteration time when creating VR content; as immersive as VR can be, sometimes you need a quick way to check that some interaction works.

Additionally, the Desktop 3D controller allows you to test networking logic within a single instance of the editor.

> **Warning:** To re-iterate: the Desktop 3D controller is **not** intended to be the pawn that you will ship the operation with, but **only** as a tool for development and testing purposes.

We'll be going through an explanation of the main controls and concepts of the Desktop 3D controller by looking into the reasoning behind them.

10.11.2 Vision & Intention

It is impossible to represent all possible movements of a VR user's head and hands using the mouse & keyboard, without resorting to an interface that makes you feel more like a pilot than anything else. Thus it was obvious from the start that the user will have to choose to control one aspect of a VR character at a time:

1. The whole body (Avatar), or
2. One of the hands

Note: You can use the number row on the keyboard to quickly change modes:

- 1 – Avatar Mode

- 2 – Left Hand
- 3 – Right Hand

Additionally, for the hands, sub-modes needed to be implemented since our SDK allows for very fine-grained controller motion requirements, which cannot be emulated with keyboard keys. So, the translation and rotation of the hand needed to be mapped to the user's mouse movement. In this manner: Through the radial menu (activated using the Spacebar), you can choose to switch to controlling one of the hands in one of two sub-modes:

1. Position (Translation)
2. Orientation (Rotation)

Note: The hotkey for switching between translation and rotation without going through the radial menu is the "Tab" key

10.11.3 Interaction

Interaction is easy to map: the left mouse button corresponds to the trigger button on a VR controller, and the right mouse button to the grip button, accordingly. Switching to a different mode while having grabbed an object will keep the hand in the same state, so you can hold multiple objects simultaneously.

Note: If you switch back to controlling one of the hands that has grabbed an object, you do not need to hold any of the mouse buttons; the item will stay grabbed. You can press the corresponding mouse button to let go of the item.

10.11.4 Swapping Axes

But there is still a problem here: The mouse can only input 2D coordinate movements: horizontal and vertical, which severely limits the user's options in both cases:

By holding down "Left Ctrl", you can temporarily change the axis of the translation or rotation

10.11.5 Throw Hand (or Quick Grab)

This takes care of a partial mapping of fine-grained movement to the keyboard & mouse. But what about mapping one of the most common aspects of any VR simulation? What about Grabbing?

The "Throw Hand" command does exactly this: It moves the left or right hand to the object at the user's center of the screen, and tries to grab anything once it's there.

Note: You can execute the "Throw Hand" command either through the radial menu, or by using Ctrl + Click:

- Ctrl + Left Mouse Button Click — Throw right hand
- Ctrl + Right Mouse Button Click — Throw left hand

10.11.6 Basic Controls

By default, the controller will start in the "Avatar" mode. This is the closest mode to any typical 3D application that uses a first-person perspective. You can:

- Look around with the mouse
- Move with the W,A,S and D keys
- Interact with UIs using the left mouse button, or 'F'
- Open the radial menu with the Spacebar

Note: Movement with WASD is enabled on all different modes, so even when you're controlling one of the hands you can still move the whole avatar around.

10.11.7 Outro

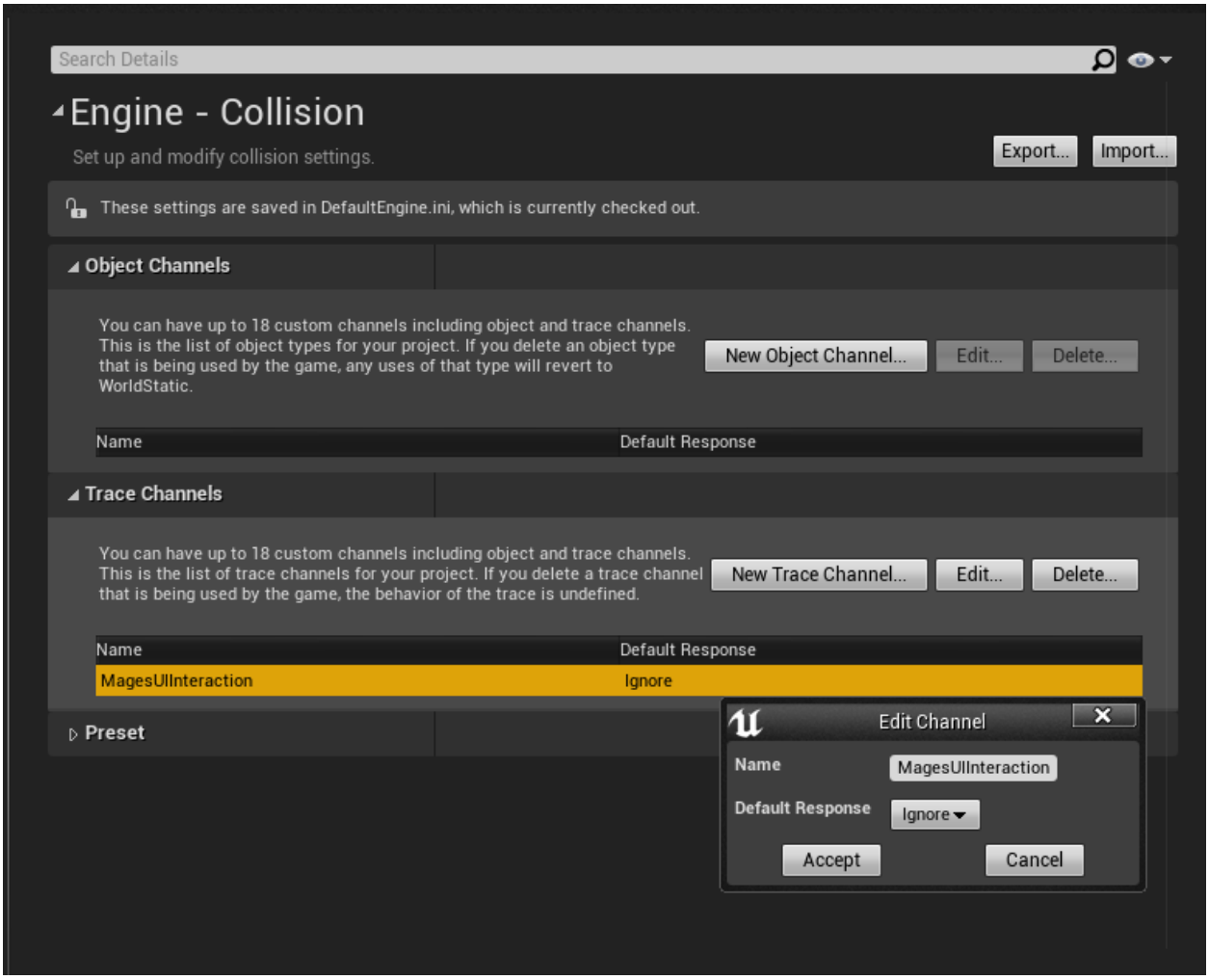
The Desktop 3D controller is self documenting and context sensitive, so it may feel difficult to control at first, but given time, you can learn the hotkeys through the radial menu and the context prompts at the top left of your screen, and become much faster with it.

10.12 MAGES Collision Settings

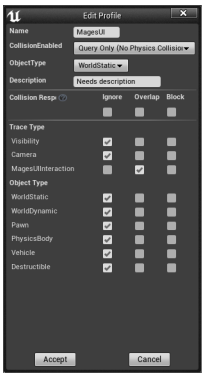
If you have imported MAGES collision settings using the provided configuration file, you won't need to complete the following steps.

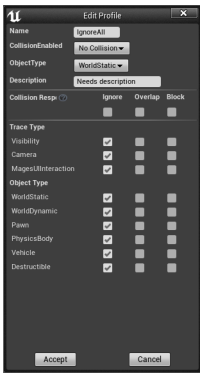
In the "Project Settings" window, navigate to the "Collision" section.

In the "Trace Channels" section, create a new Trace Channel named "MagesUIInteraction", with the default response method set to "Ignore"

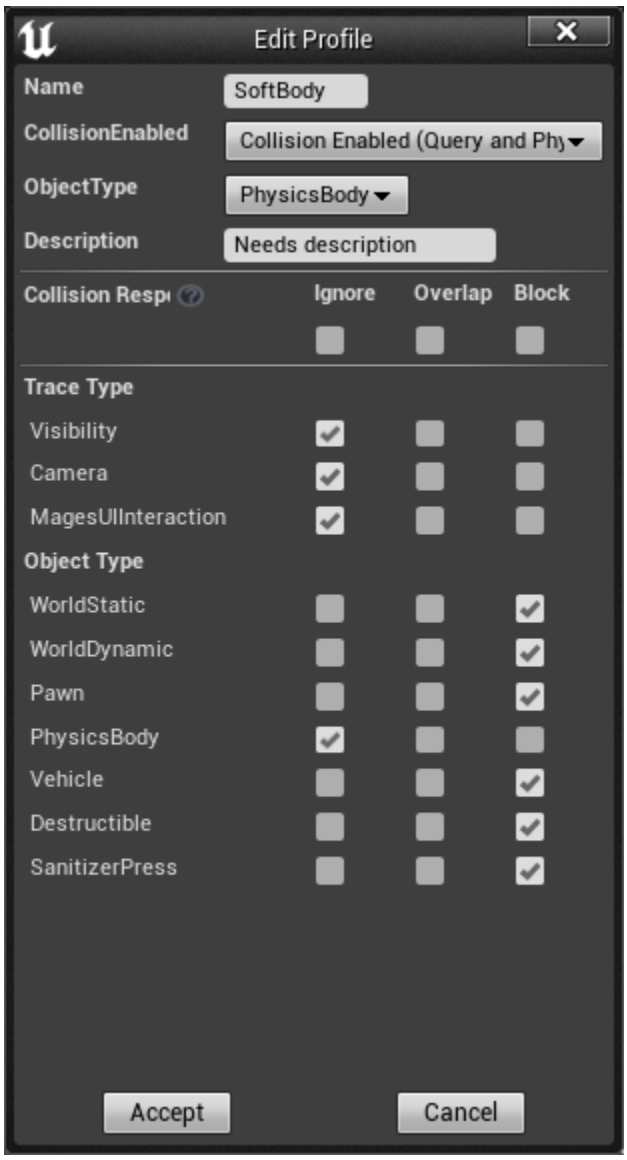


Expand the "Preset" section, and create two new presets, "MagesUI" and "IgnoreAll". The configuration for both is shown below:

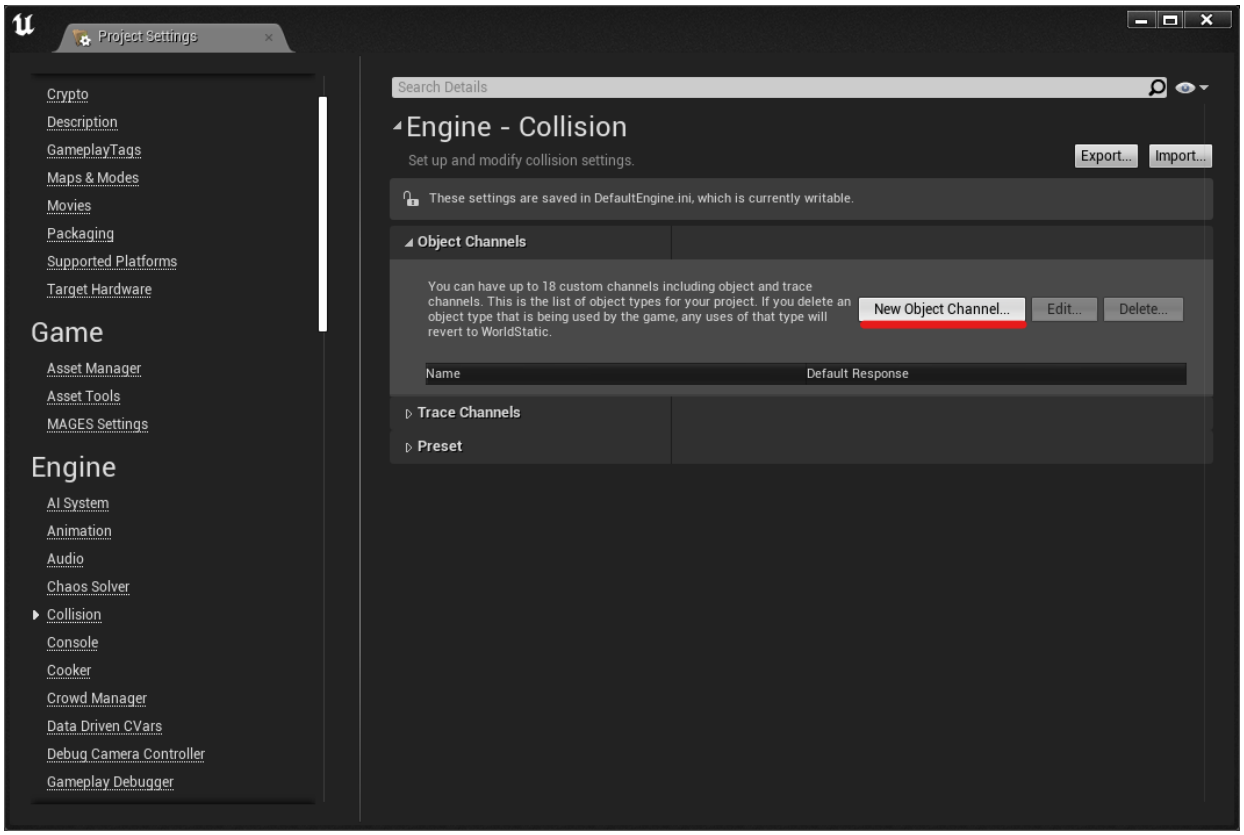




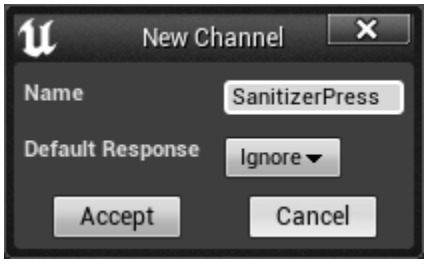
Softbodies need another collision preset named "SoftBody":



For the CVRSB sample, an object channel named “SanitizerPress” is needed. Click on “New Object Channel”:



Name the channel “SanitizerPress”, and set the default response property to “Ignore”:



CHAPTER

ELEVEN

TUTORIALS

11.1 Action Prototypes

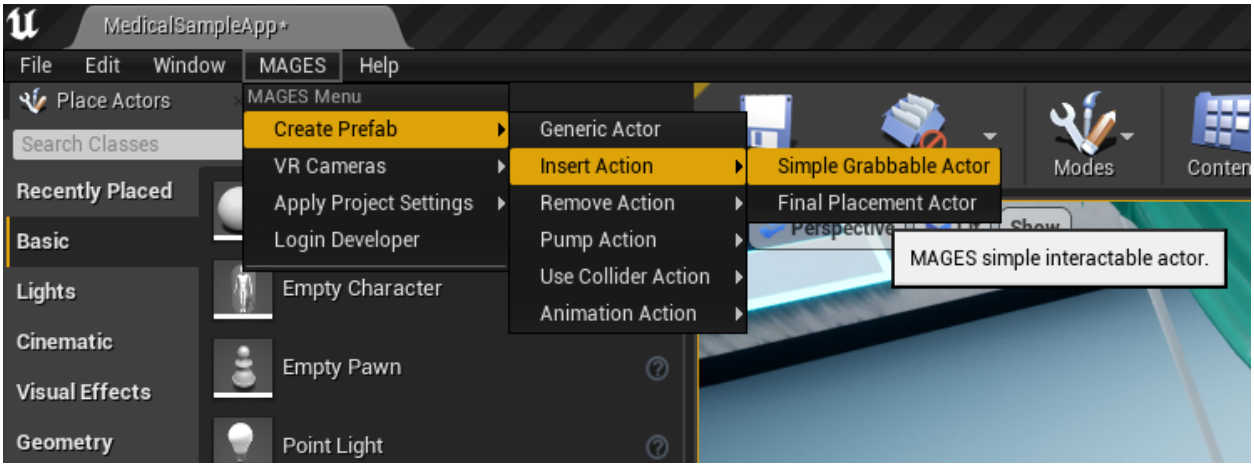
11.1.1 Insert Action

To generate an Insert Action you need the following three blueprint actors:

1. A grabbable actor
2. Its final position
3. A hologram indicating the final position

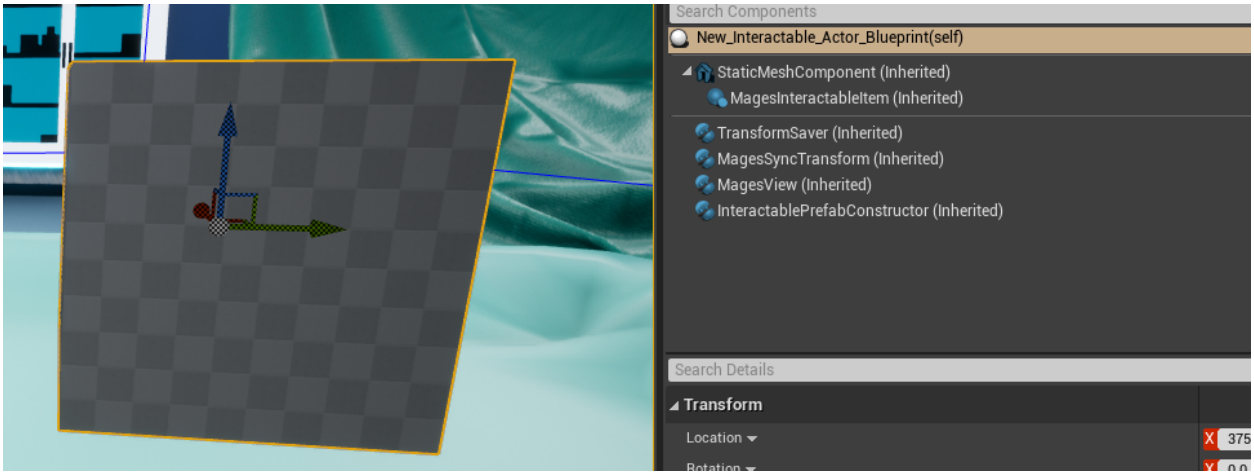
Interactable Actor

From the MAGES menu select the option Create Prefab/Insert Action/Simple Grabbable Actor



The template actor for the interactable object will appear. It is recommended to use this object as the root and starting point of your grabbable actor. In the actor's components you will find its static mesh component, there you can add its static mesh. Below you can see the final result.

Note: Keep in mind that unreal might not refresh your mesh in the editor immediately. In order to force a refresh you can create a blueprint using this instance.



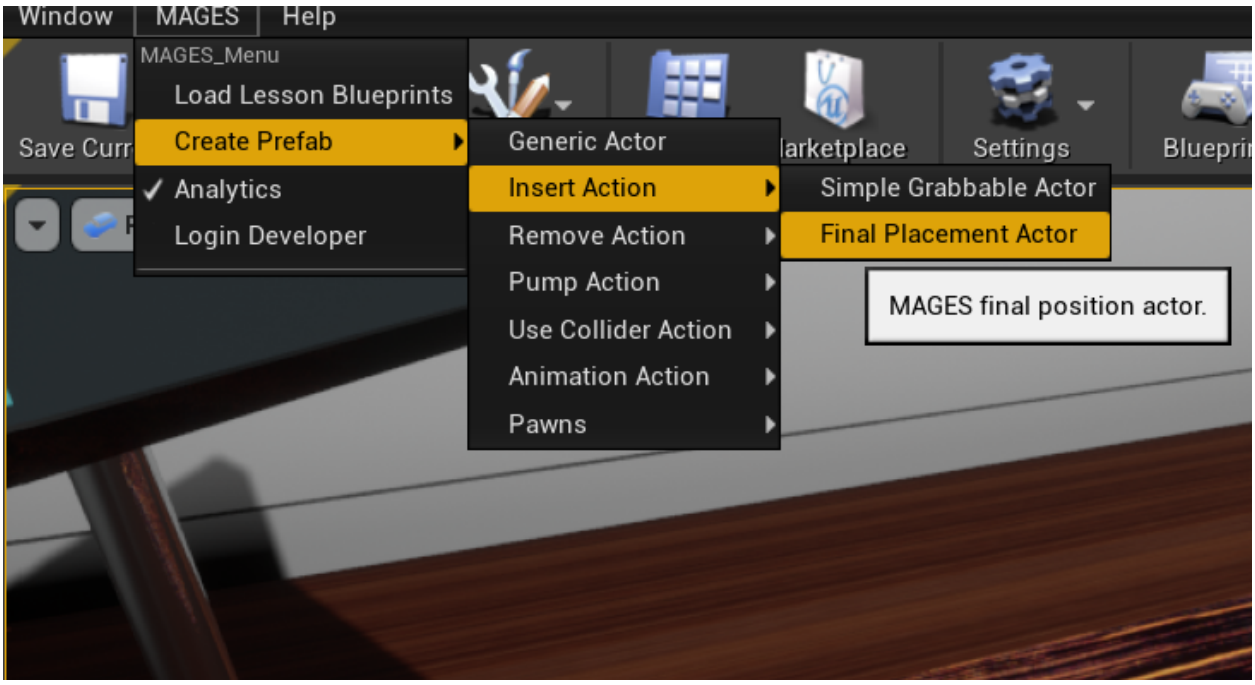
Remember to add colliders to the object as you need to grab it, and to configure its collision properties otherwise it will pass through the table.

The next step is optional but recommended for a more natural interaction.

We need to configure hand postures when interacting with a grabbable object. You can read *here* a detailed tutorial on how to properly setup hand postures.

Final Prefab

The next step is to generate the final placement actor. This indicates the correct position and the orientation of the object. In a similar way, we navigate to the MAGES menu and click the Create Prefab/Insert Action/Final Placement Actor.

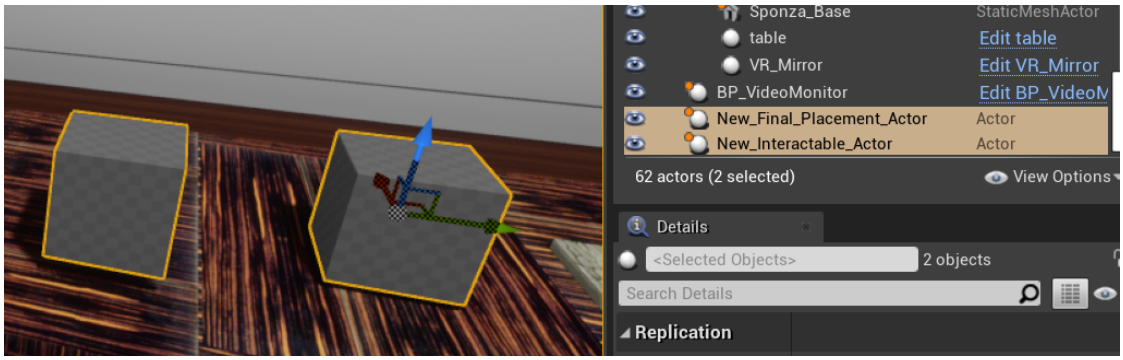


Warning: The Final prefab must have the same pivot with the interactable actor because the PrefabLerpPlacement script checks if the orientation (position and rotation) of the objects match to perform the Action.

For this reason, the safest way to generate the final prefab is to duplicate the interactable, copy the transform of its root, paste it on the final prefab template and transfer its children to the final prefab.

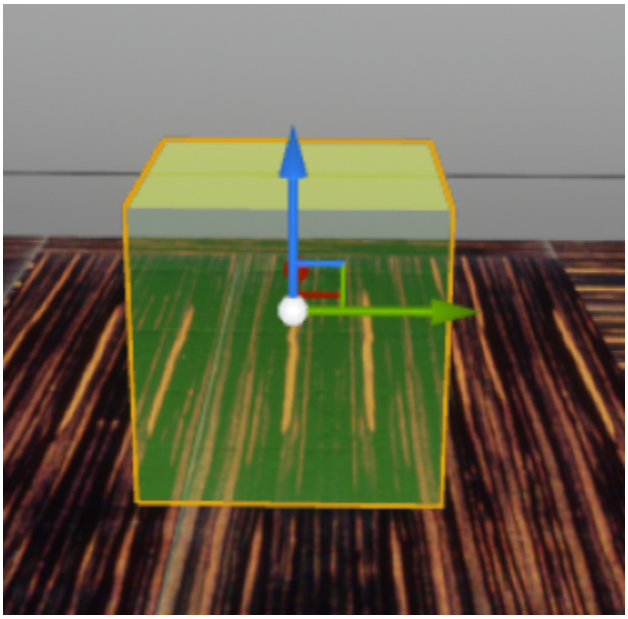
Remember to disable all Collisions and enable the Generate overlap events field.

The image below shows both the interactable (left) and the final prefab (right).



Hologram Actor

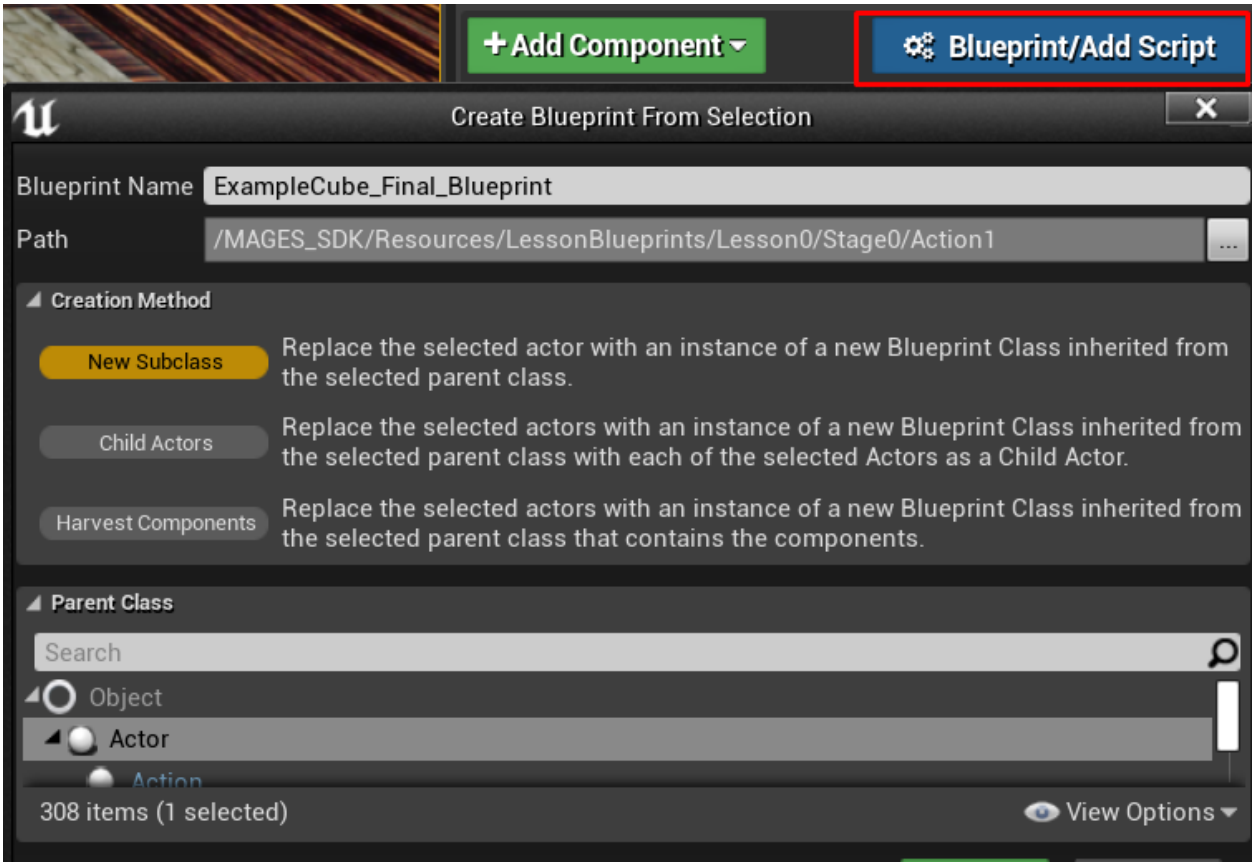
The hologram actor only needs the transform saver component and the MagesView component (for multiplayer support). It is just a copy of the final placement actor with the holographic material. Remember to remove its colliders as well.



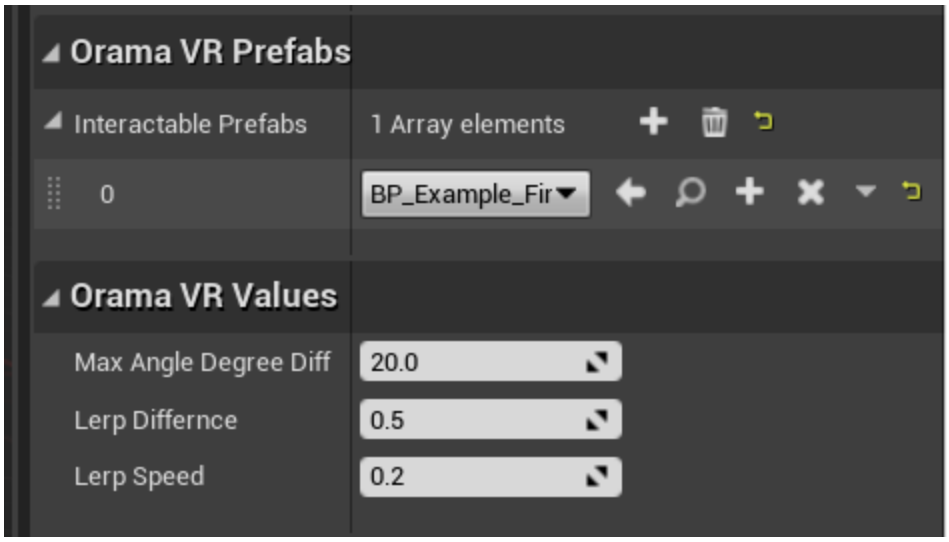
Save prefabs and final configuration

Now we need to save the actor's transform in order to spawn it at that location and orientation. Once we've placed all actors in the decired position we select the Transform Saver component on each of them and click Save Transform. Then we create a blueprint class based on that specific actor, by clicking the Blueprint/Add Script button. It is recommended to keep the action actors in folders according to the scenegraph structure. In this case we will save the interactable, final and hologram prefab at Lesson0/Stage1/Action0 folder.

unreal/tutorials/action_prototypes_ue4/img/insert_action_

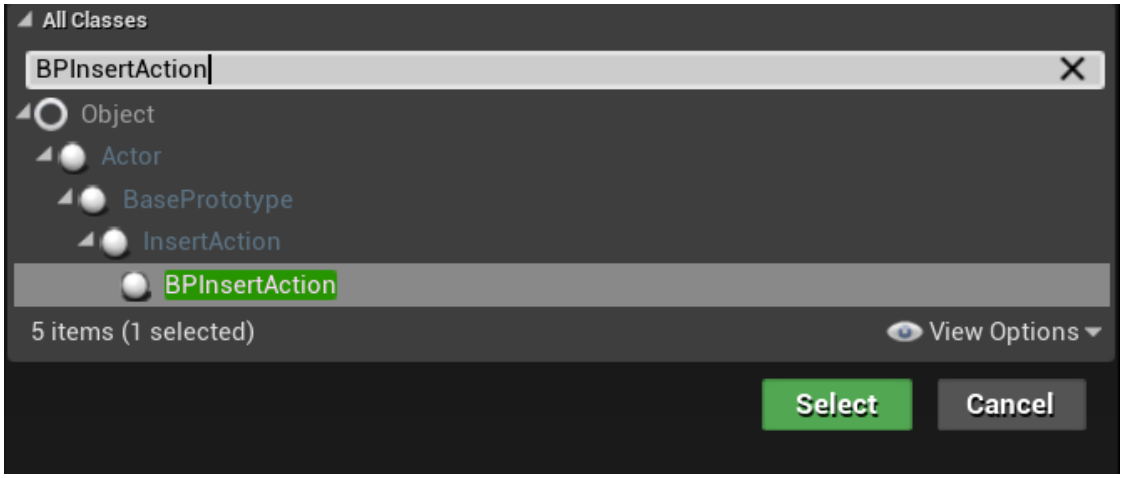


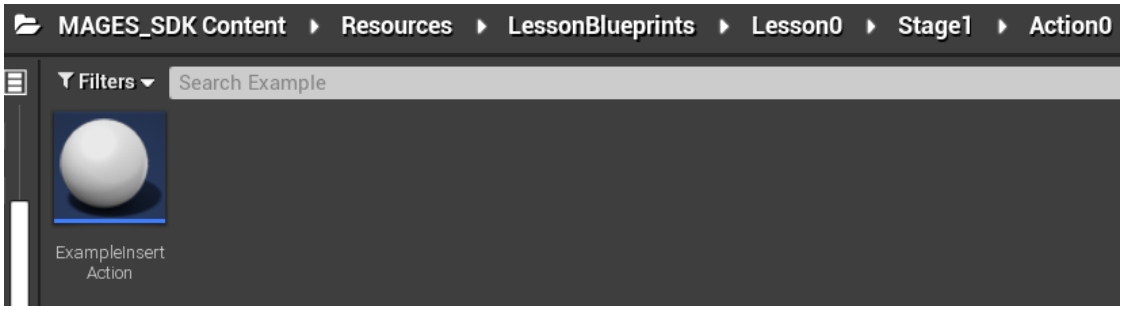
The final step is to configure the PrefabLerpPlacement component which is attached to our final actor. This component indicates the interactable prefab that matches with this final placement actor. Additionally, you can setup properties like the tolerance in angle difference with the interactable or setup the lerping behaviour. The image below shows the interactable along linked with the PrefabLerpPlacement component.



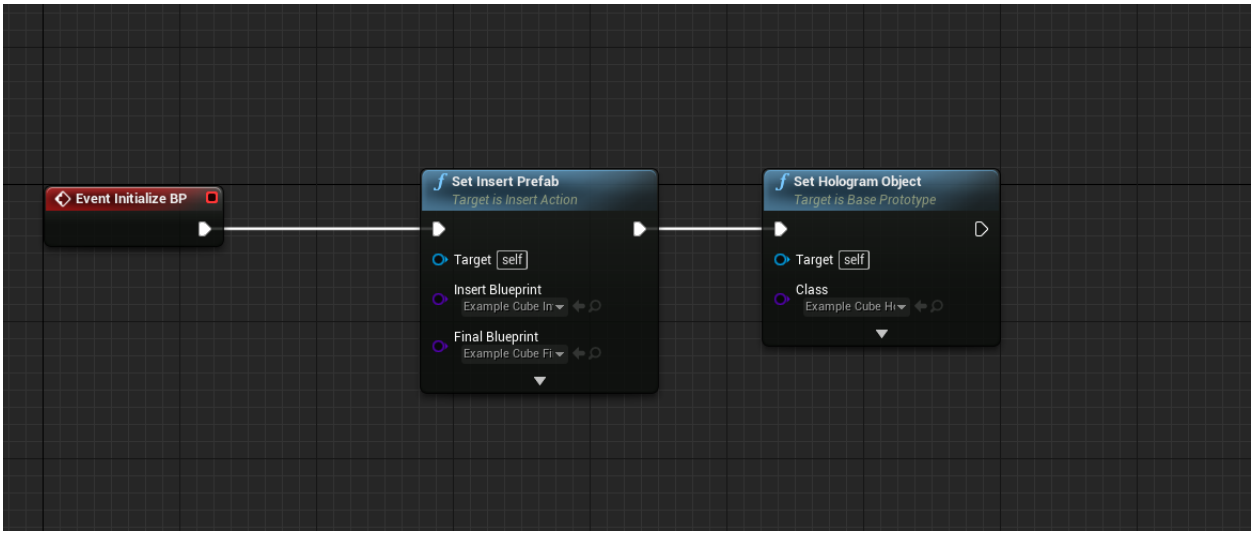
Action Blueprint

In this step we will create the Insert Action Blueprint. Create a new blueprint that inherits from BPInsertAction class. We save the Action bleuprint in a path similar to the scenegraph structure, in this case Lesson0/Stage1/Action0.





The blueprint nodes can be seen below. It initializes our grabbable and final actor and spawn the hologram actor as well.



Add the Action to Scenegraph

Finally, we need to link our newly created action with scenegraph. You can read the detailed tutorial on this procedure in the *scenegraph tutorial*.

11.1.2 Use Action

This guide will describe how to implement a use action from scratch using MAGES SDK

Use Action consists of three possible ways to perform the corresponding action accordingly to user needs

These are:

- Simple
- Use With Tool
- Use With Collision Hit

To generate a Use Action you need:

- The interactable prefab
- The use collider
- An animated hologram

Simple

This form of Use Action requires the user to “use” an object for a specified amount of time

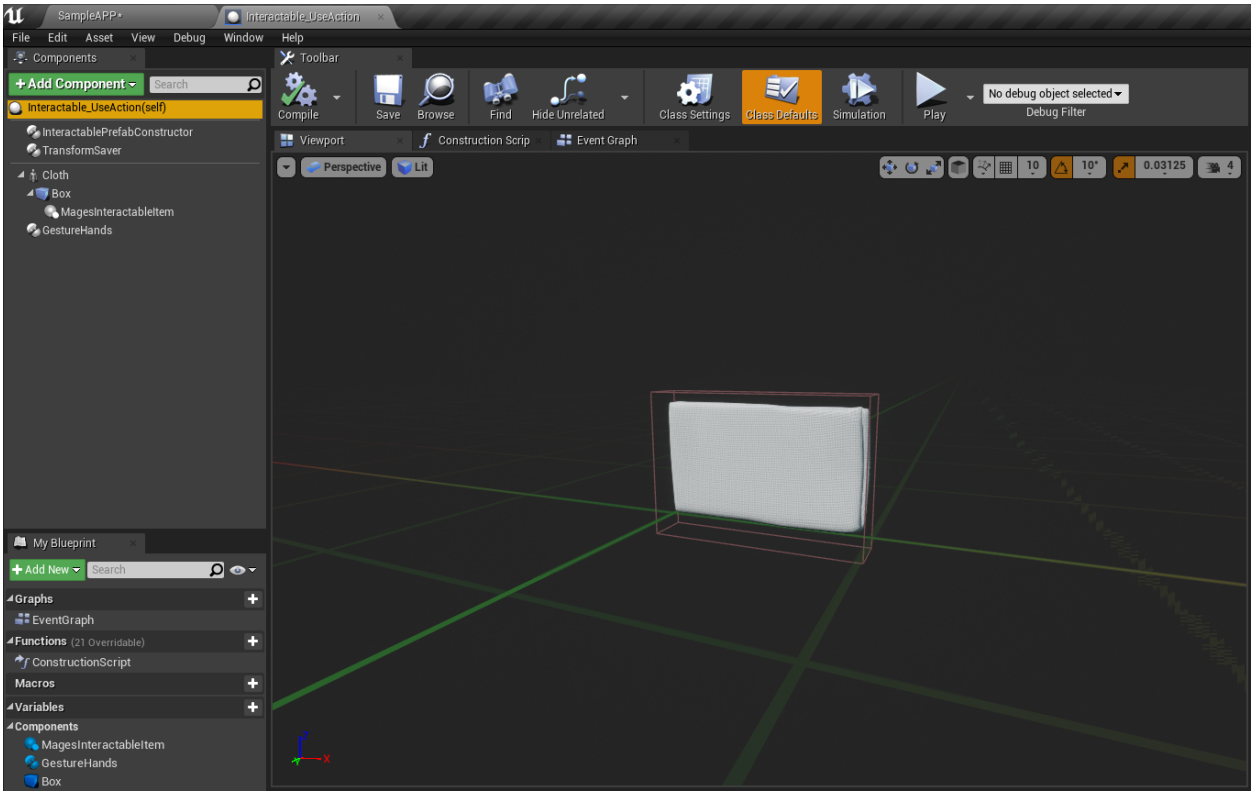
In this tutorial, we will implement a Use Action where the users should take a cloth to clean the Sponza model.

Interactable prefab

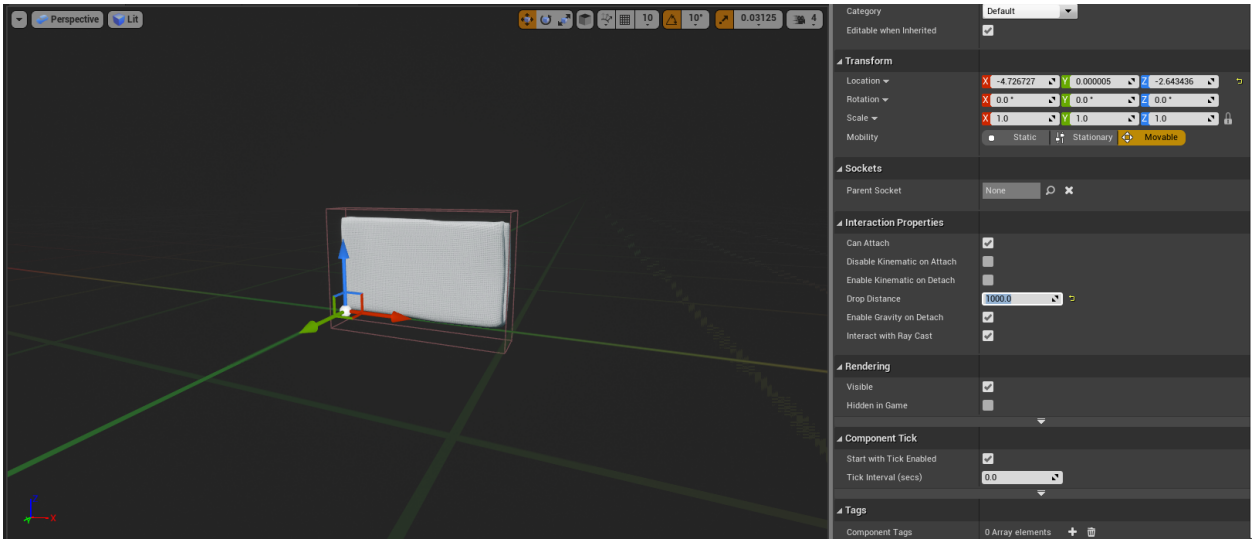
To create an interactable actor you need:

- First, you need to create a blueprint class having your interactable model as root
  - Add a collision component on that mesh
    - Attach the component 'Mages Interactable Item' on the collision component
- Attach the Interactable Prefab Constructor component on the actor
- Attach the Transform Saver Component
- Attach the Gesture Hands component

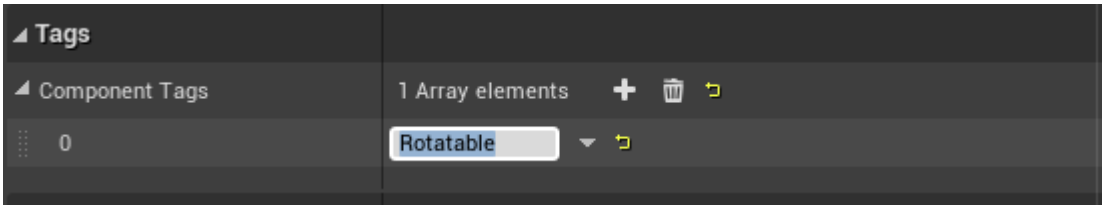
An example of what the blueprint call actor should look like is shown below:



Note: On the 'Mages Interactable Item' component go to Details panel -> Interaction properties -> Drop Distance and modify the value to a higher one (e.g. 1000)

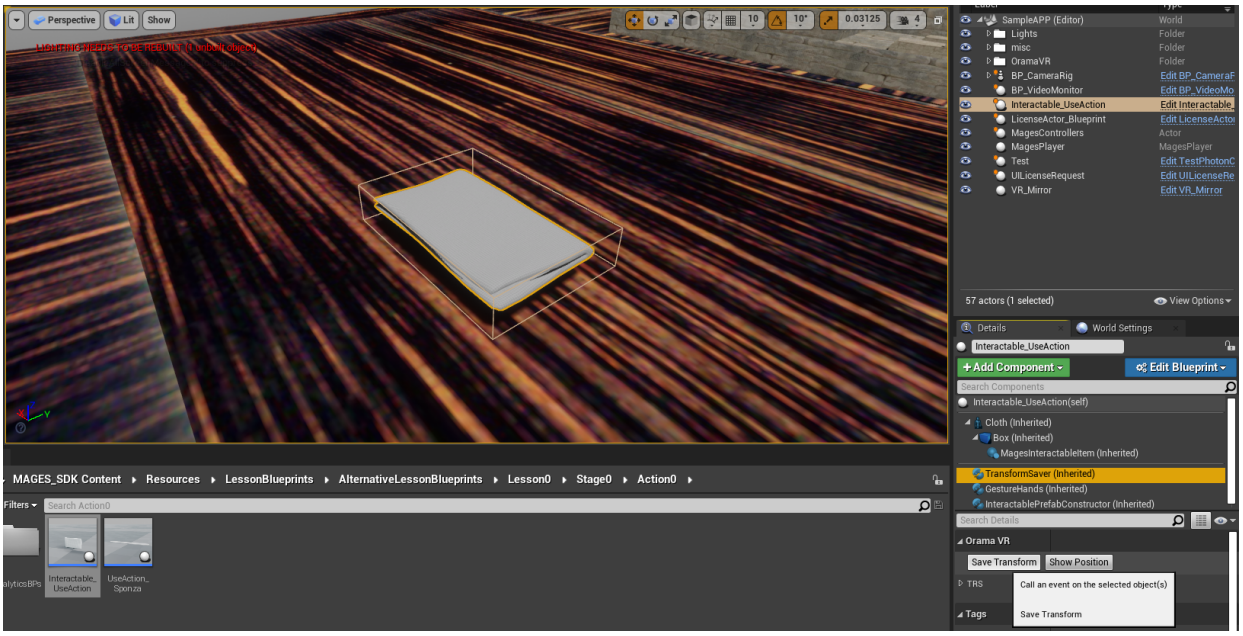


Afterwards select the mesh component (root) and add the tag “Rotatable”

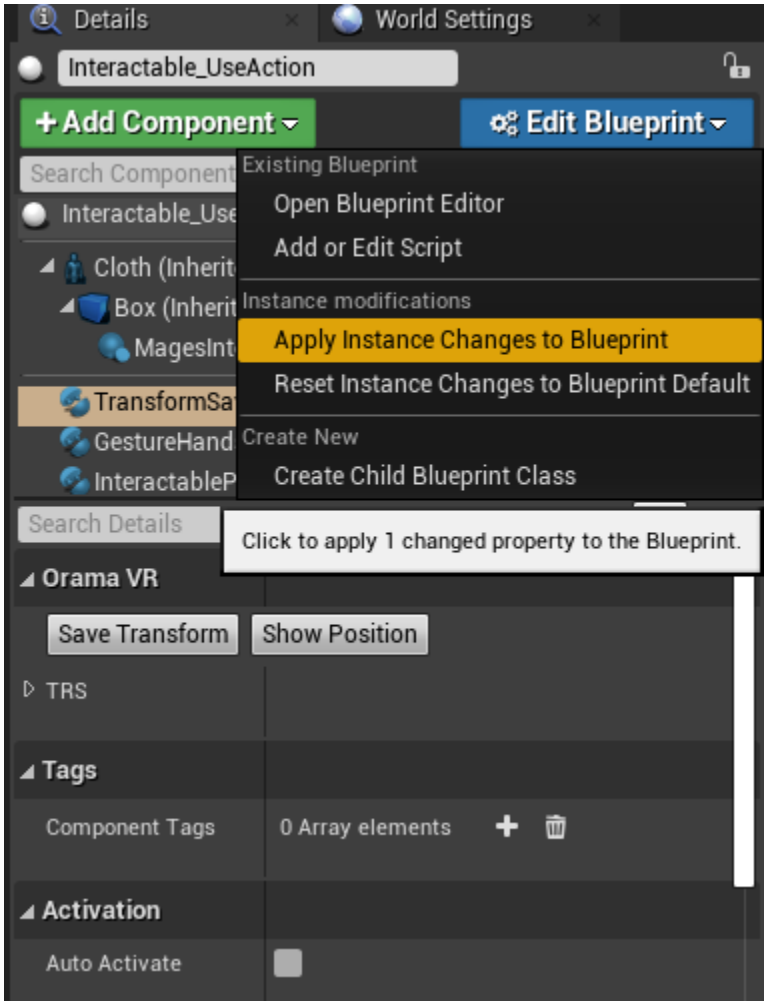


Finally, compile and save your actor blueprint and put it inside the level

Adjust its position to where you want it to be spawned and then through the details panel navigate to trasform saver and on the **Orama VR** tab click *Save Transform*.



Once done, click Edit blueprint -> Apply Instance Changes To Blueprint and save the actor again

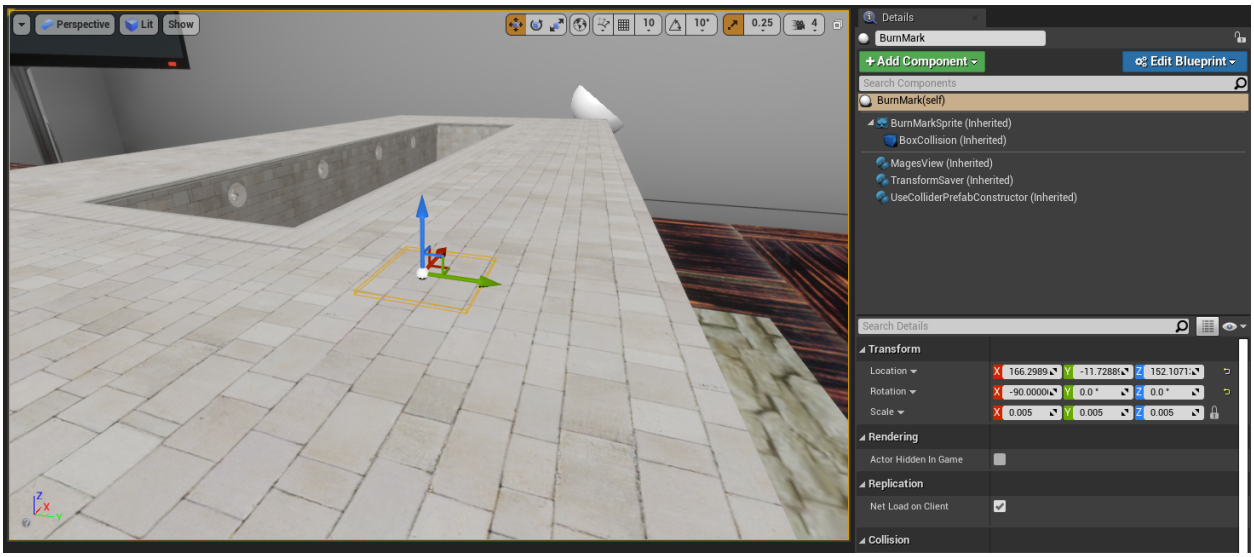


Use With Tool

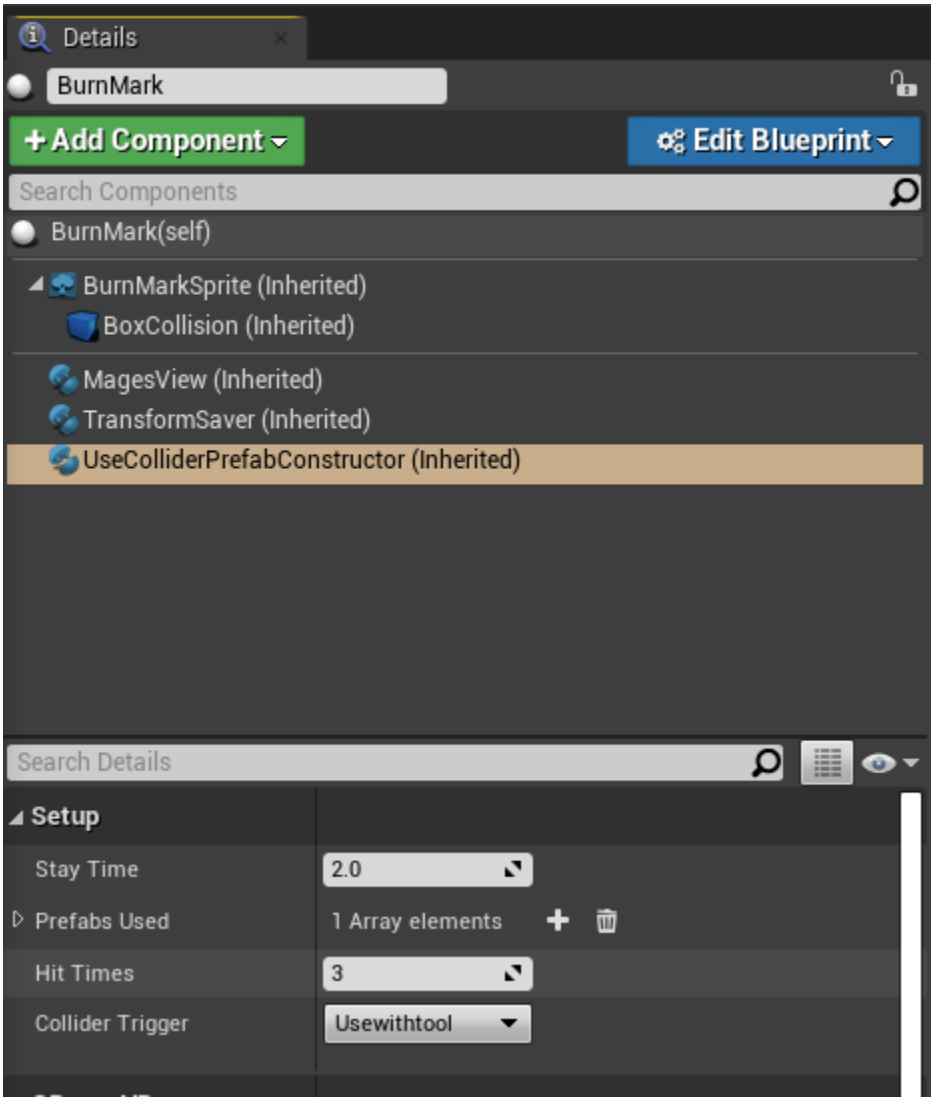
In this case, we will use the cauterizer on a specific collider on top of Sponza.

Use Collider

We will generate a new actor, with a box collision component (the BurnMarkSprite is only use for visual feedback; it does not factor into the action itself) and the `UseColliderPrefabConstructor`:



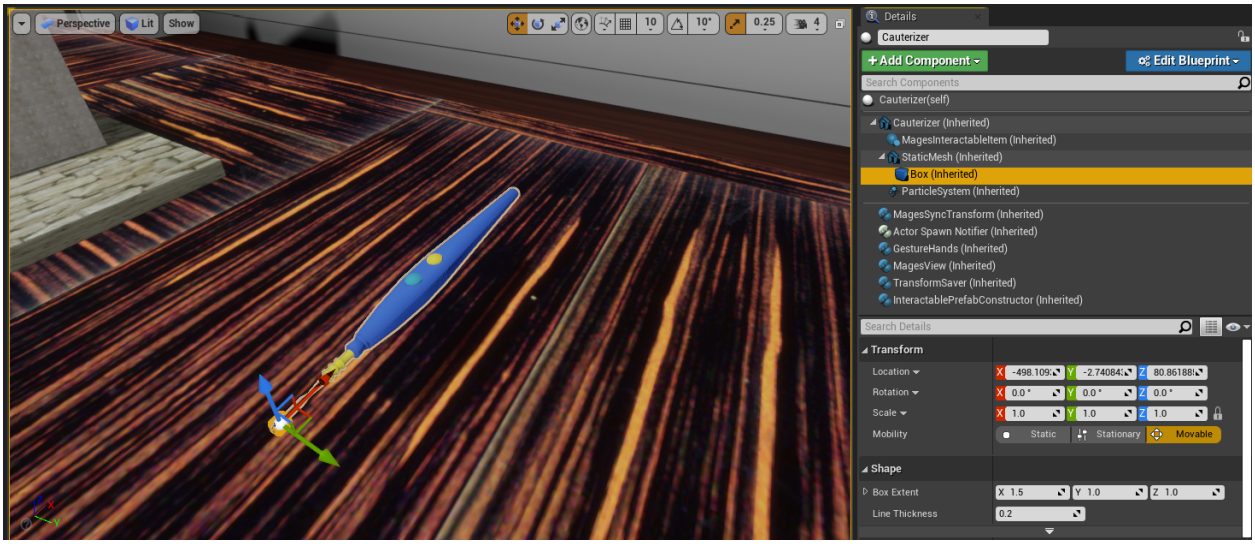
In the `UseColliderPrefabConstructor` component's properties (in the details panel), set the `Collider Trigger` property to "Usewithtool". The `Stay Time` property designates how long the tool has to interact with the item in order for the action to be completed.



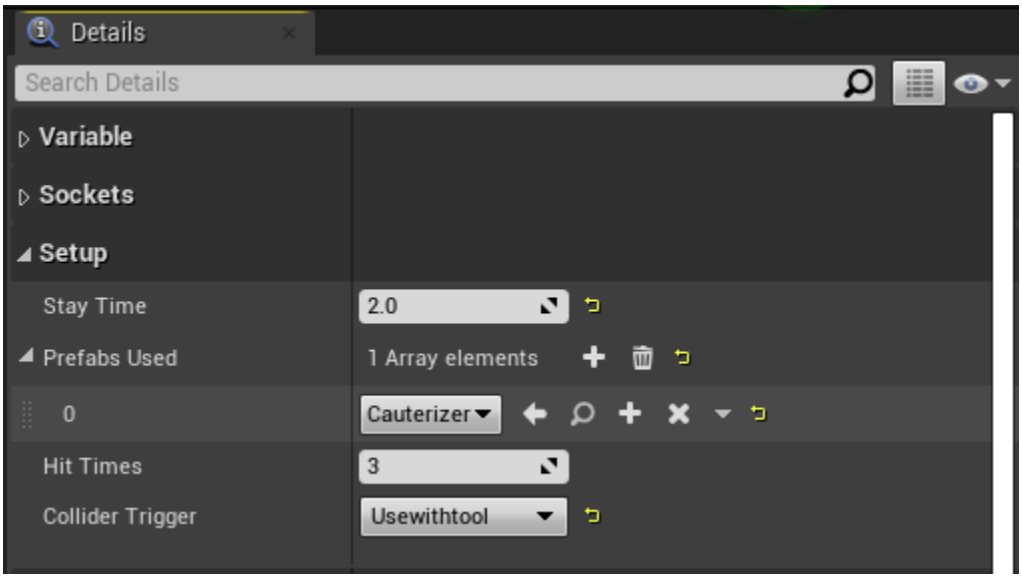
Next up, we will create the "Cauterizer" actor. This actor will need:

- The `Mages Interactable Item` component
- The `Interactable Prefab Constructor` component
- The `Gesture Hands` component

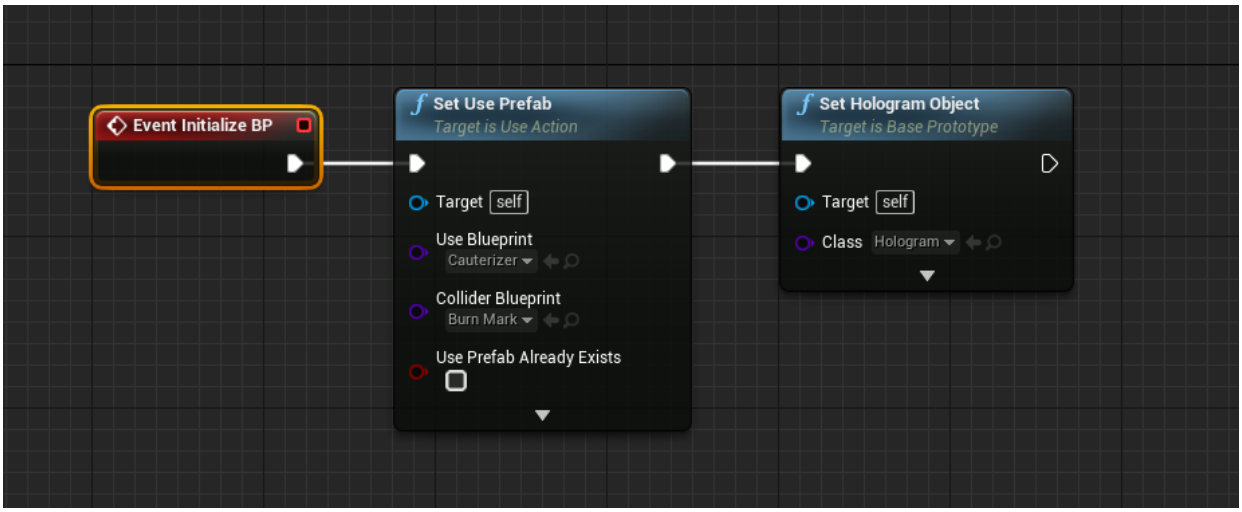
We will also add a Box Collision component to the tip of the cauterizer.



Now that we've created our tool, we have to tell the Use actor (on which it will applied), what kind of objects can be applied on to it. Select the `Use Collider Prefab Constructor` component, and add the newly created tool to the `Prefabs Used` array property:



The following blueprint will spawn the necessary actors for this action:



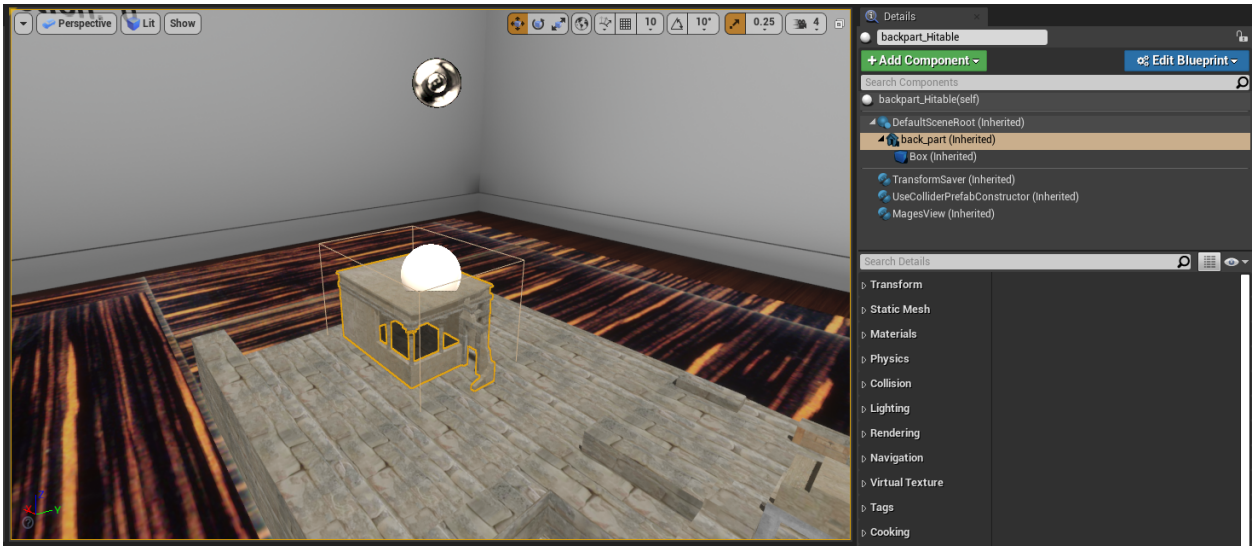
Use With Collision Hit

There are cases where you might need the user to *hit* an interactable using a tool, rather than apply it continuously on the interactable item. We will use a mallet to set-up an action where the user has to hit the back gate into place.

Hit-able Item

We generate an actor with the following components:

- The `Transform Saver` component
- The `Use Collider Prefab Constructor` component
- And a collision shape component (in this case, we will use a `Box Collision` component)



To configure this actor so that the user has to hit it with a tool, we change the following properties of the `Use Collider Prefab Constructor` component, in the details panel:

- Set the `Collider Trigger` property to “Hit”
- Set the `Hit Times` property to however many times the user has to hit the object (in this case we set it to 3)

- Set the `Hit Movement Vector` to "0.0, 0.0, -1.0", so that the object will move downwards with each subsequent hit

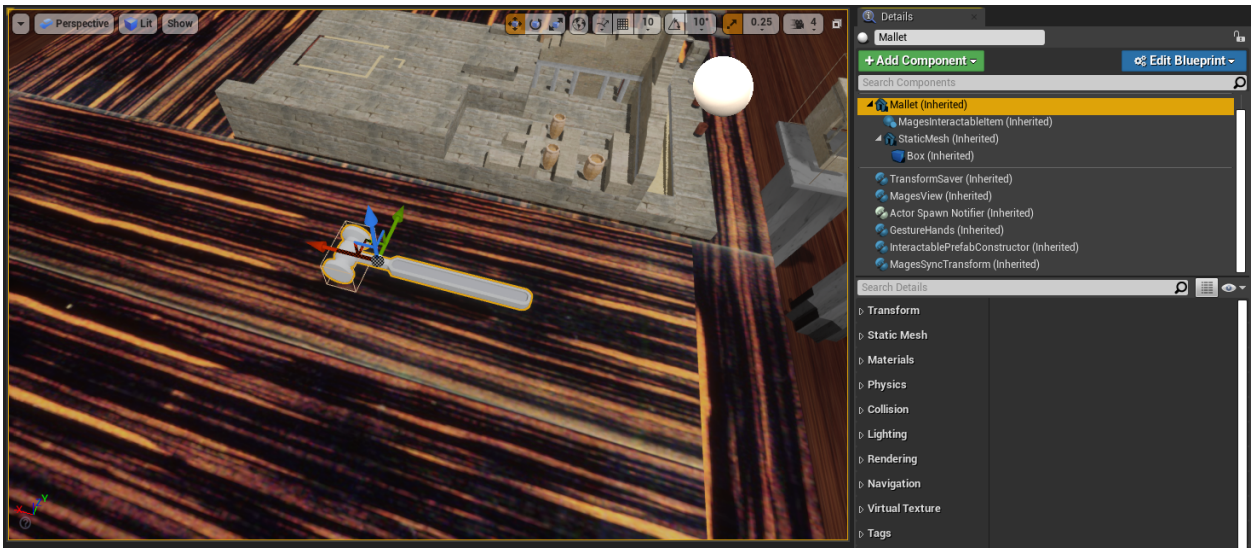
---

Note: The hit-able item's saved transform (applied using the `Transform Saver` component), should be its final location. The component will automatically offset the object, depending on the `Hit Movement Vector` and `Hit Times` properties.

Mallet

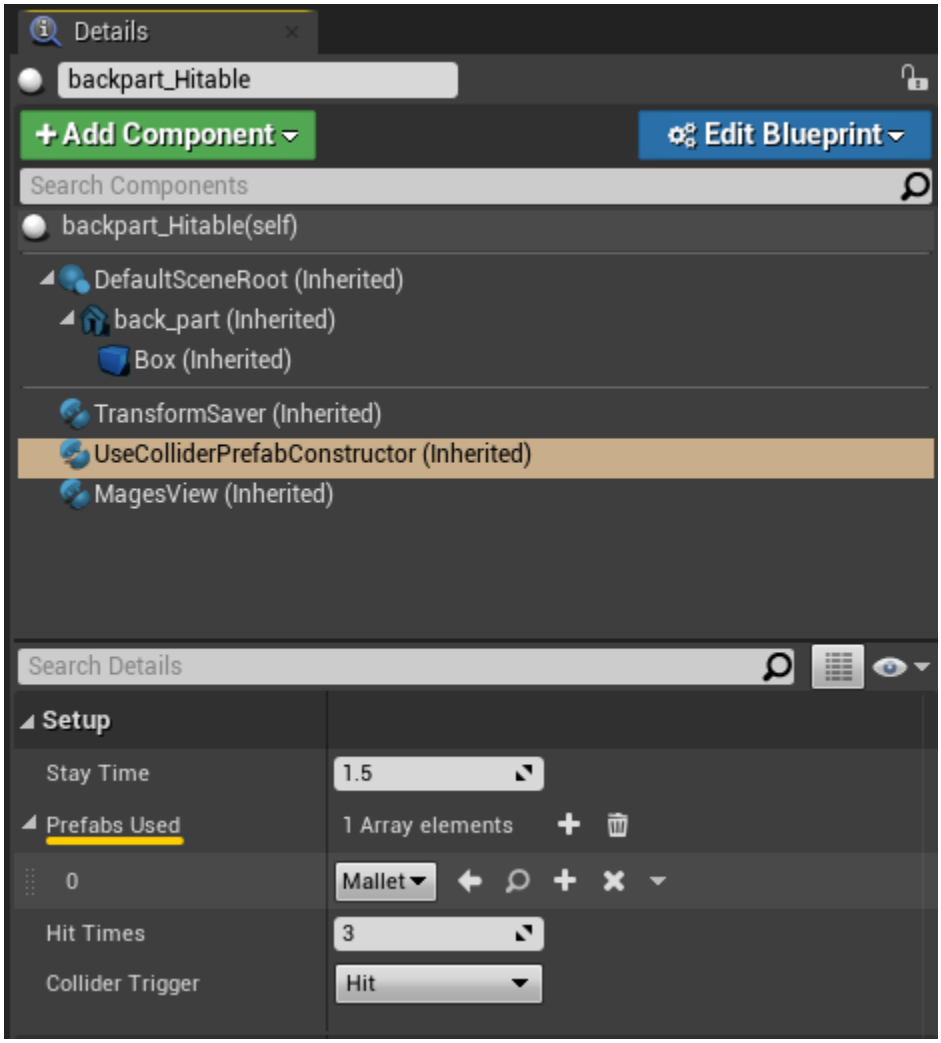
To create the tool which will be used to hit the object, create a new actor with the following components:

- `Transform Saver` component
- `Mages Interactable Item` component
- `Interactable Prefab Constructor` component
- `Gesture Hands` component
- A collision shape component; in this case we use a `Box Collision` component

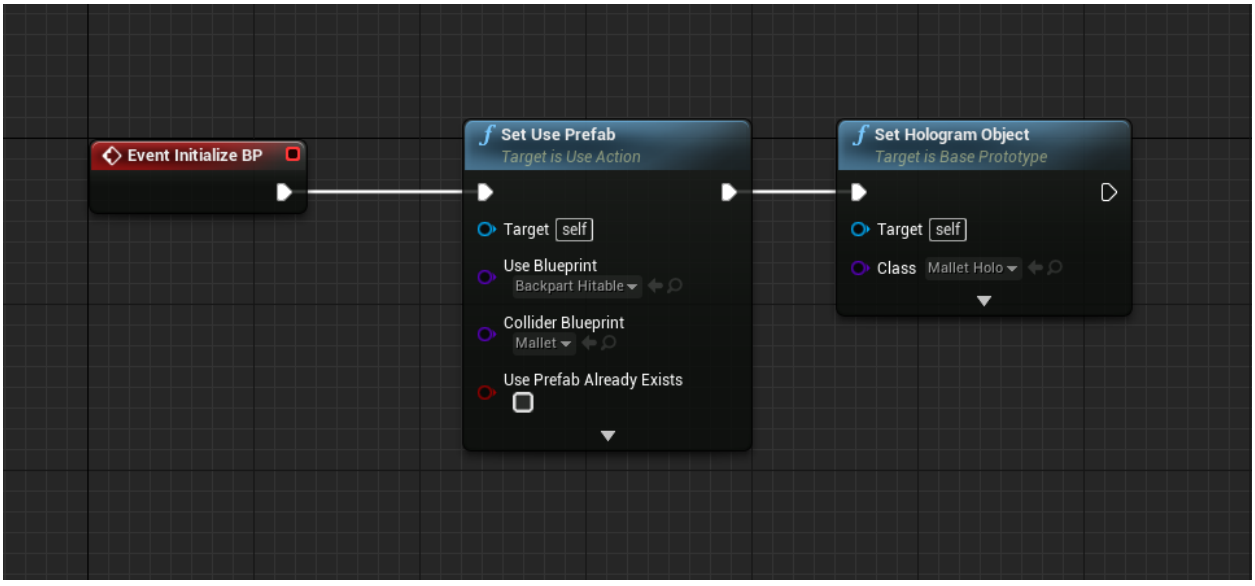


Note: The collision shape component will *only* be used for striking the hit-able item we created previously, the collision of the `Static Mesh Component` named "Mallet" will be used for grabbing the tool, since the `Mages Interactable Item` component is placed under it in the component hierarchy.

Open the Hit-able item's blueprint and in the `Use Collider Prefab Constructor` component properties, add the mallet to the `Used Prefabs` array property:



The following blueprint graph will spawn the two actors for this action:



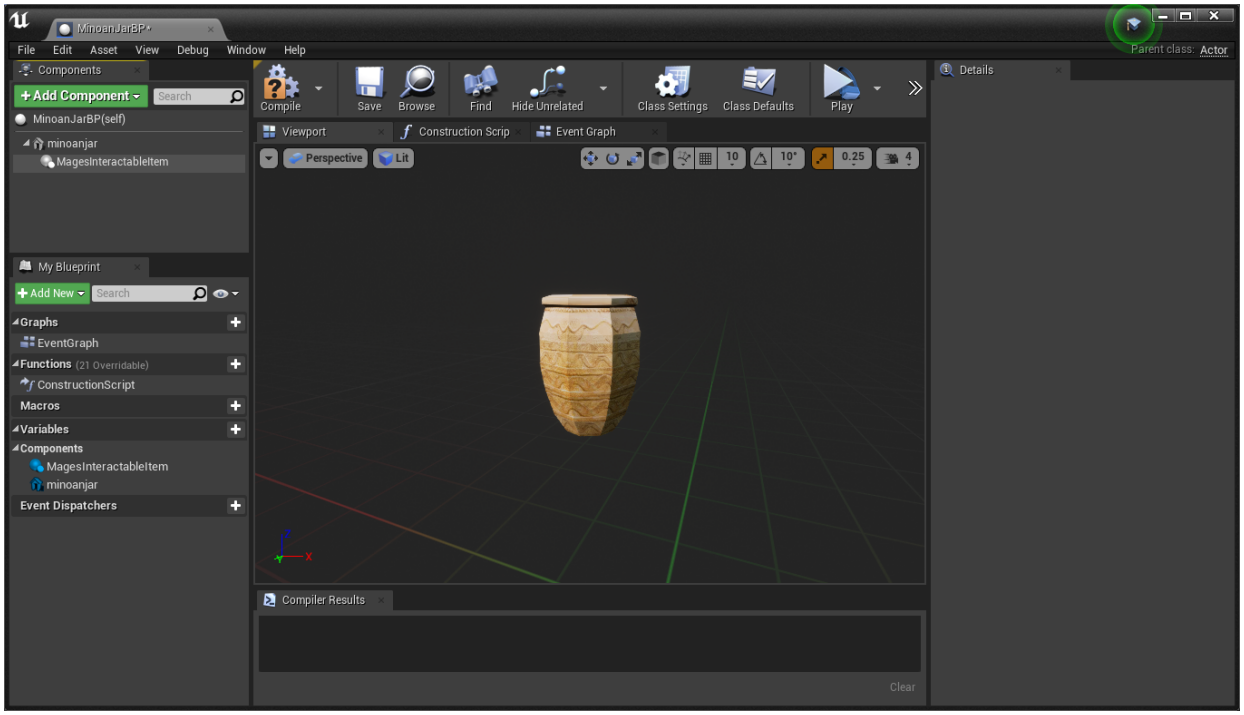
11.1.3 Remove Action

The Remove Action is used when we need to remove a specific object from the scene using our hands or a tool.

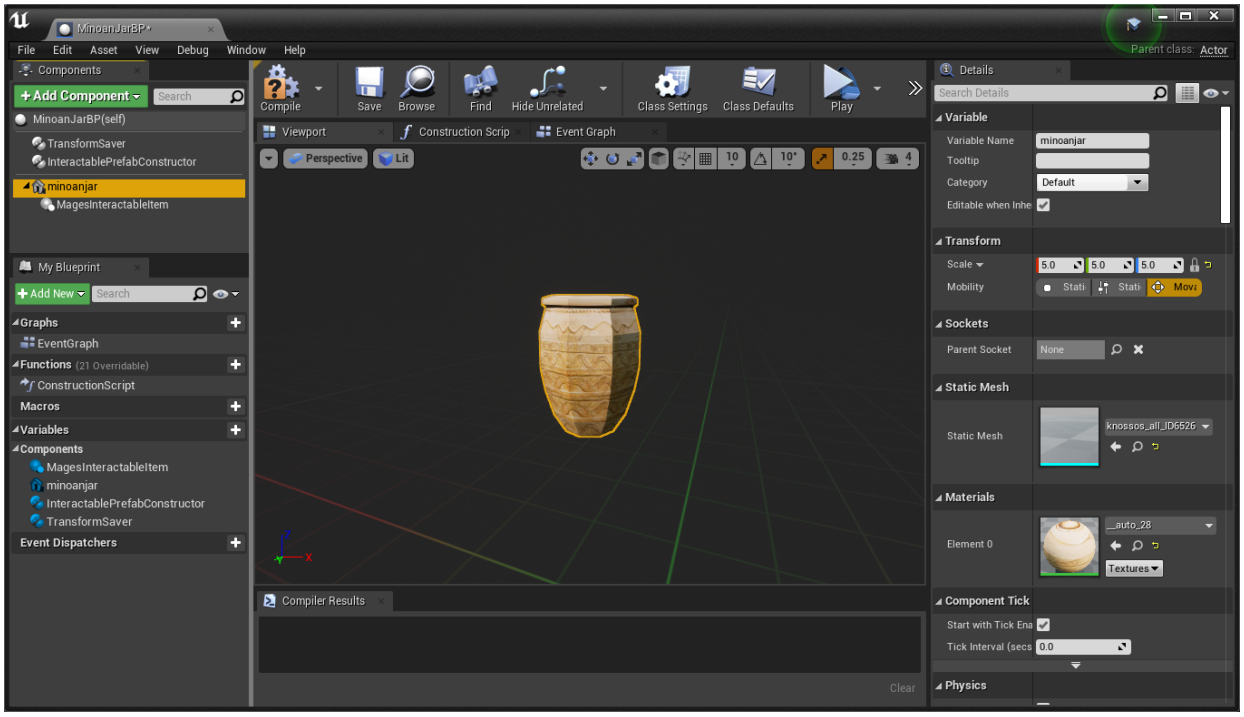
To generate a Remove Action you need an *interactable item* with an "Interactable Prefab Constructor" component which the user will have to remove, and (optionally) a hologram.

Interactable Item

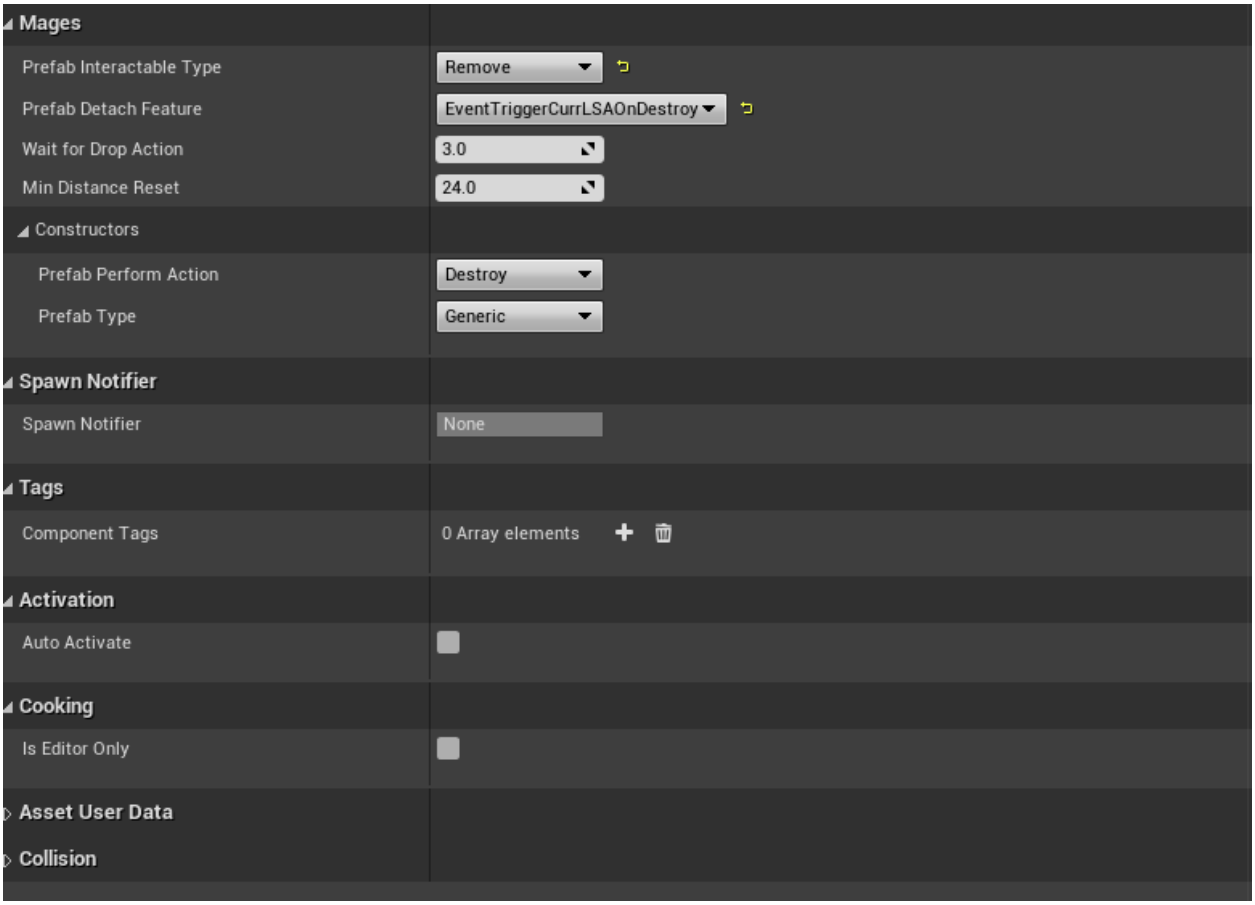
Create a new actor, and add the `Mages Interactable Item` component under the primitive where it should attach to upon user interaction:



Add the `Interactable Prefab Constructor` component and the `Transform Saver` components:

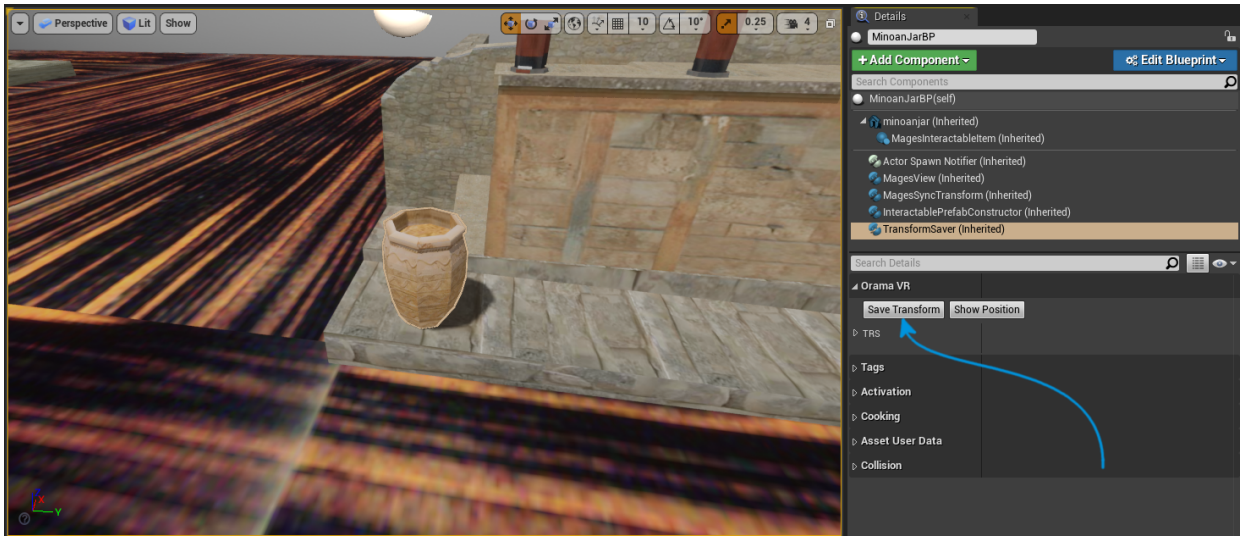


The only configuration needed is to set the `Prefab Interactable Type` to `Remove`, under the Mages/Constructors category.

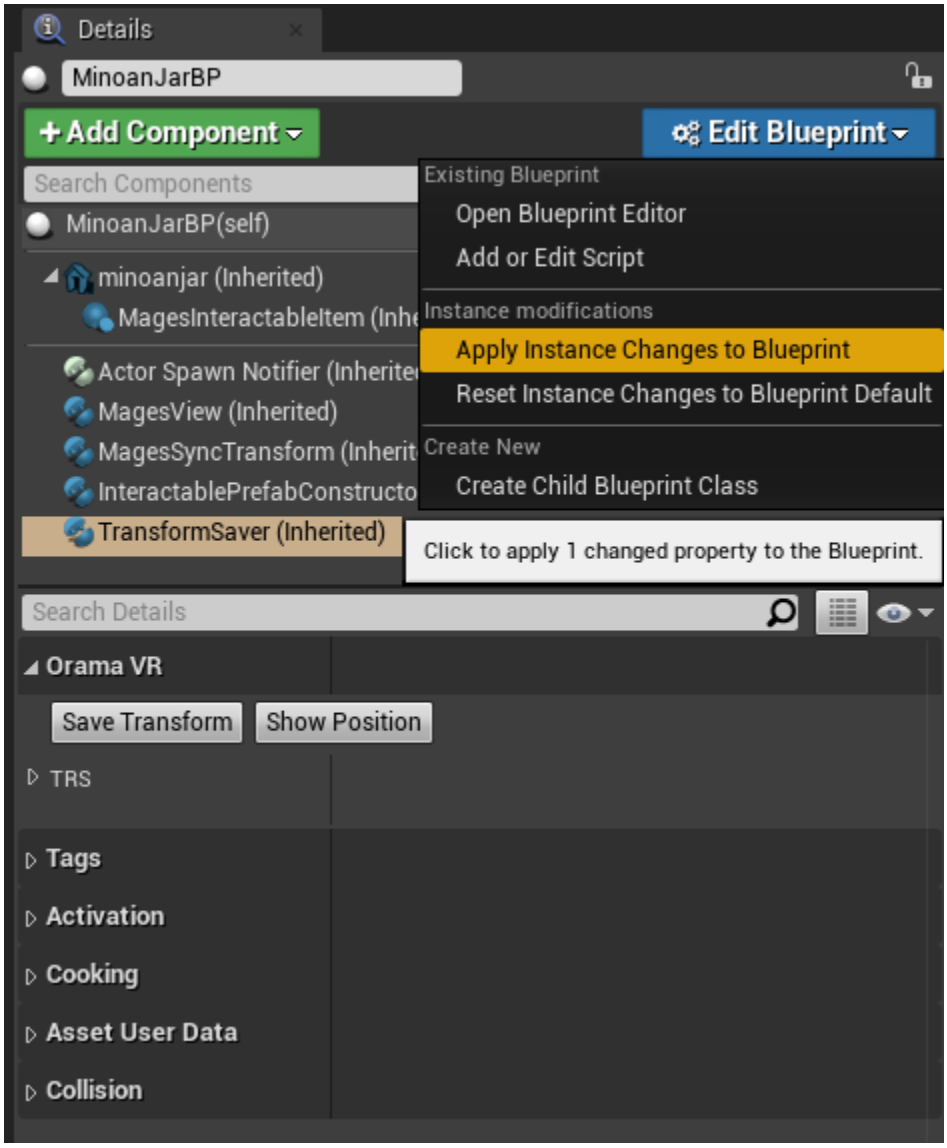


Compile and save the blueprint, then place it into the scene.

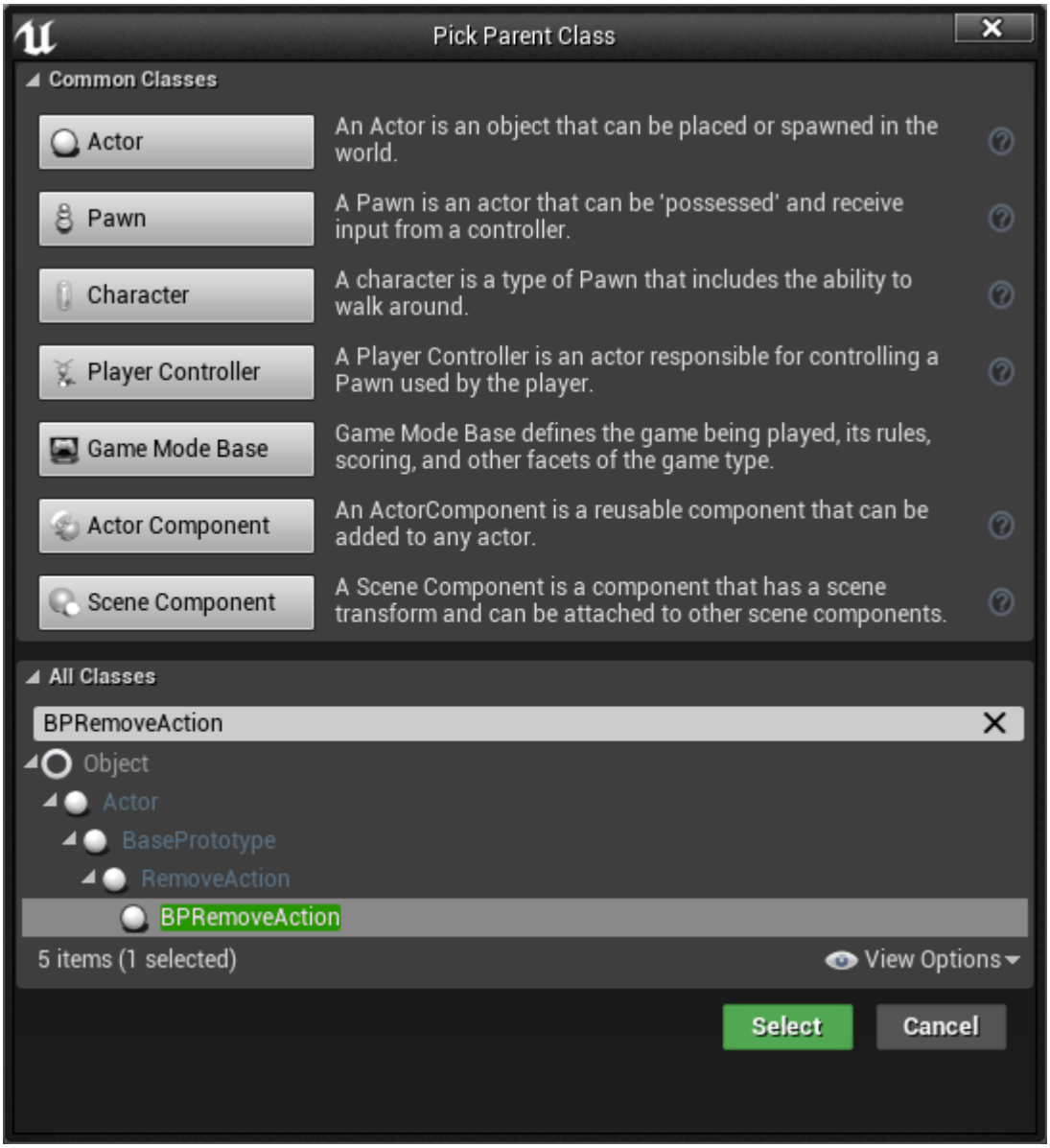
To save the current location, rotation and scale of the actor, have the blueprint selected in the scene, then select the `Transform Saver` component, and click on "Save Transform" in the details panel. This will save the transform of the current blueprint instance.



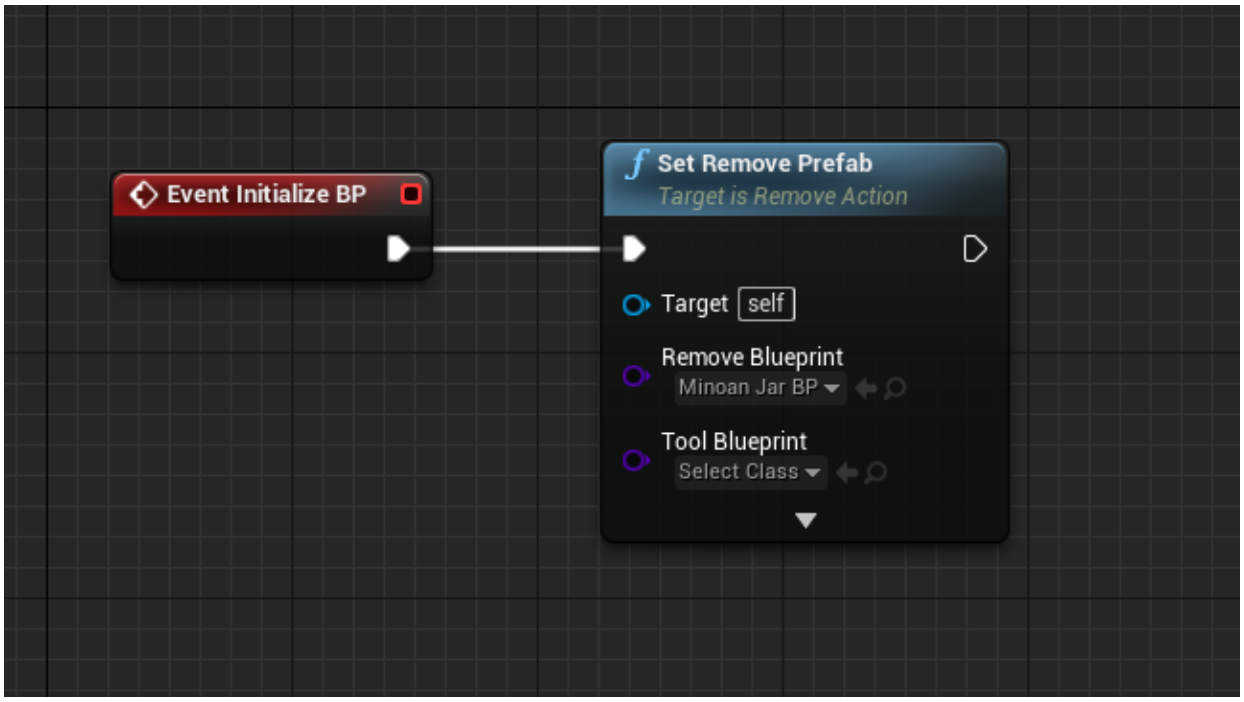
To apply it to the blueprint itself, click on "Edit Blueprint" and click "Apply Instance Changes to Blueprint Default".



Create a new Blueprint deriving from `BPRemoveAction`:



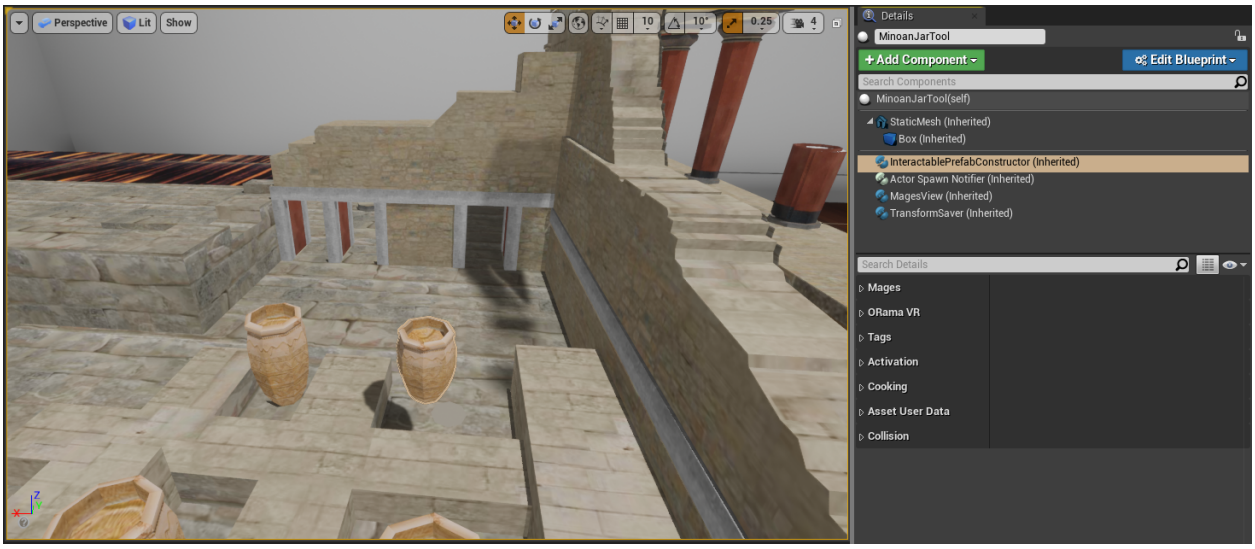
Edit the blueprint's event graph to spawn the interactable item:



Remove Object with Tools

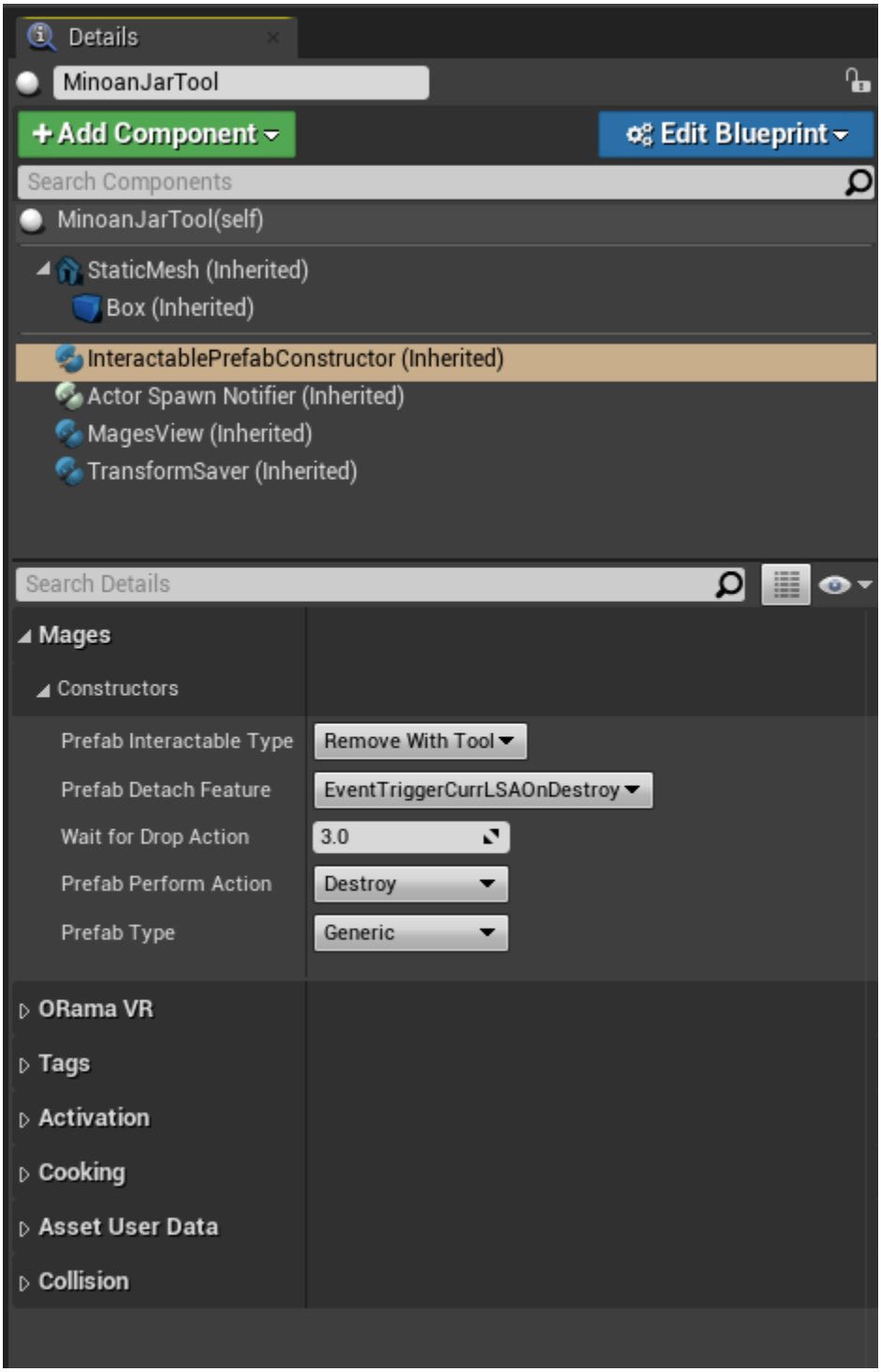
In this case, we will remove a jar using a tool (the pliers) instead of our hands.

This time, we will generate one actor as seen below:

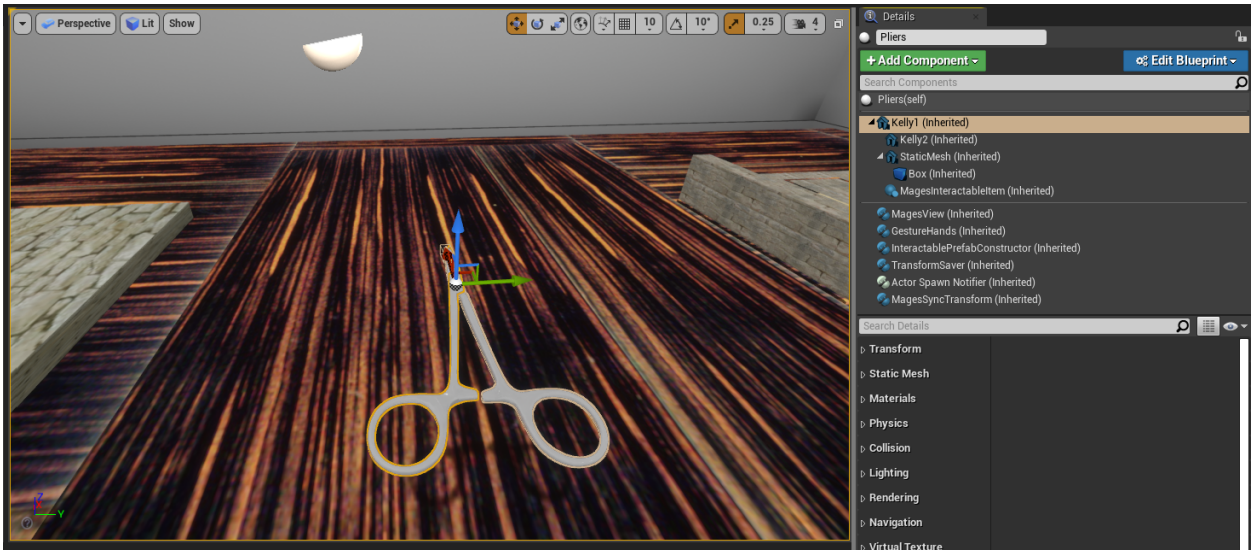


Now, we will configure our remove prefabs to be removed with the pliers tool. At the `Interactable Prefab Constructor` component of your actor:

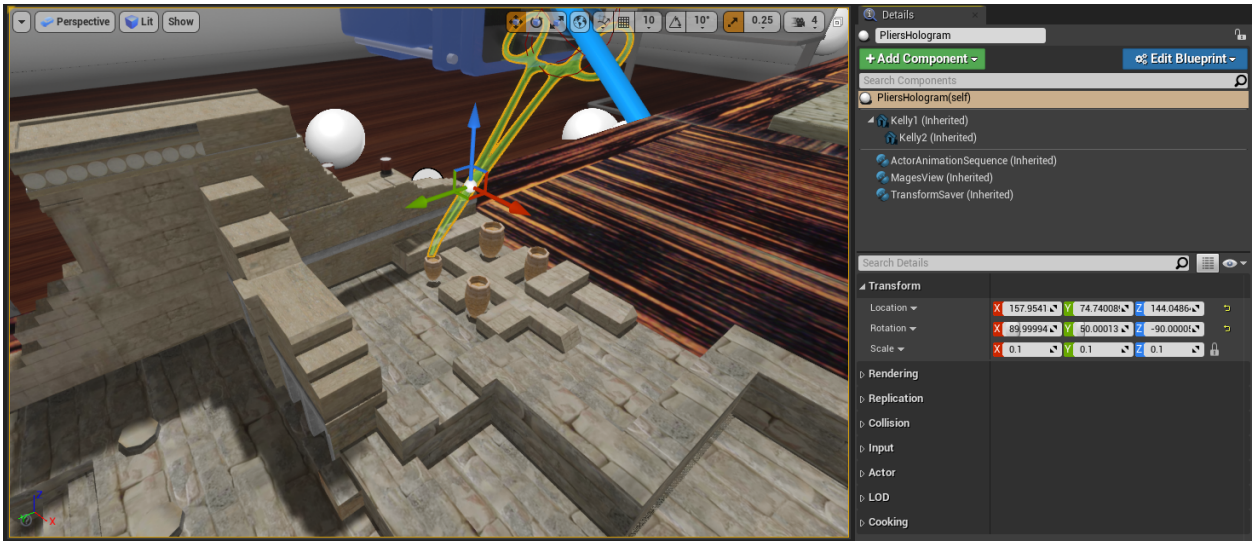
- Set the `Prefab Interactable Type` to `Remove With Tool`
- Set the `Prefab Detach Feature` to `EventTriggerCurrLSAOnDestroy`



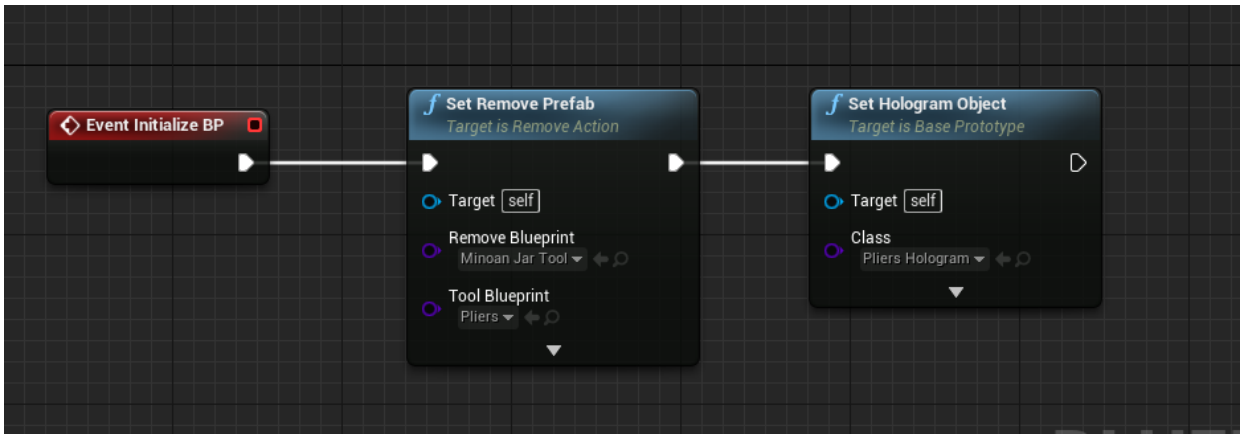
Will also generate the pliers actor. This actor must have the `Gesture Hands` component attached to it:



Additionally, we can create an animated hologram:



In this blueprint, we want to use the pliers in order to remove the jar, thus the graph will be the following:



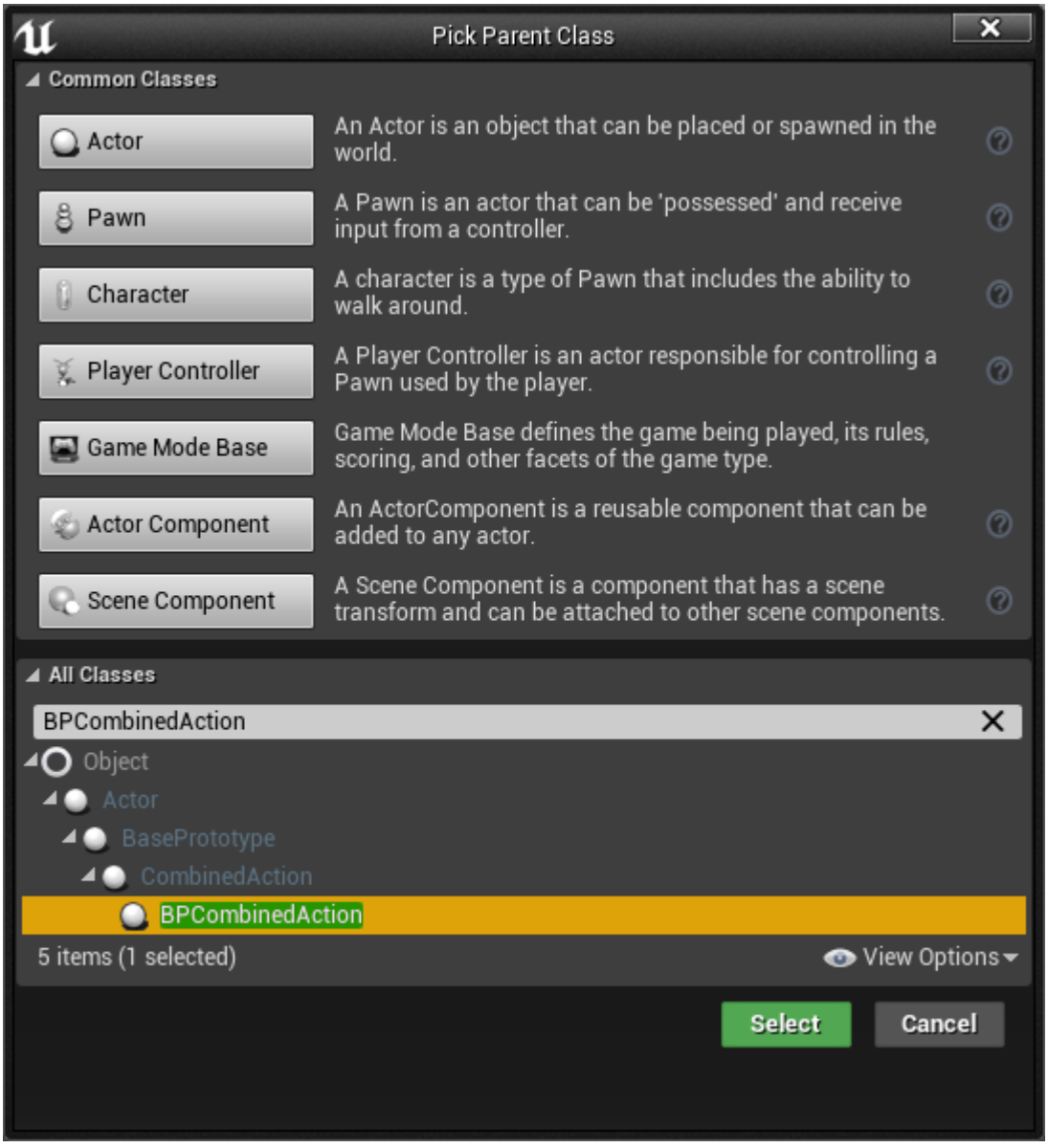
11.1.4 Combined Action

The Combined Action does not include any new VR behavior, it is a way to include multiple Actions in the same script. Combined Actions are useful in situations where we want to implement sequential tasks but incorporate them into a single entity.

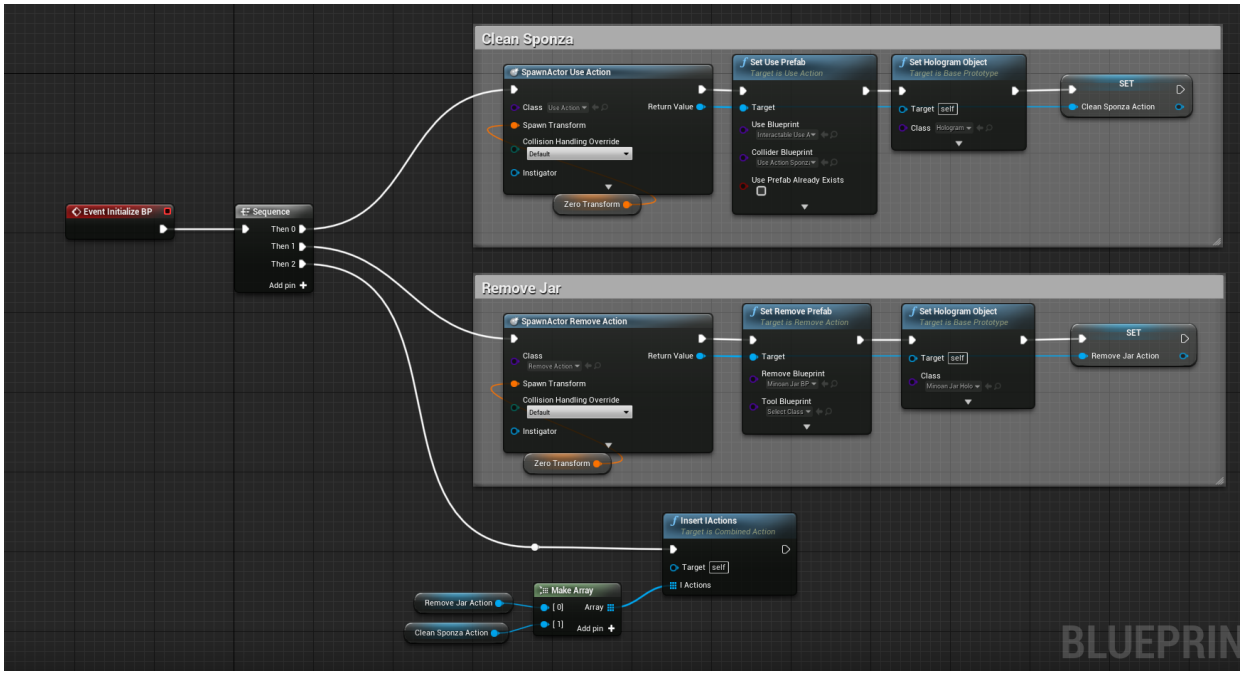
In this example, we will convert the *UseAction* and the *RemoveAction* from the previous tutorials into a CombinedAction. As a result, in this Action the user would be asked to clean the Sponza with a cloth (UseAction) and then remove the jar using their hand (RemoveAction).

Action Blueprint

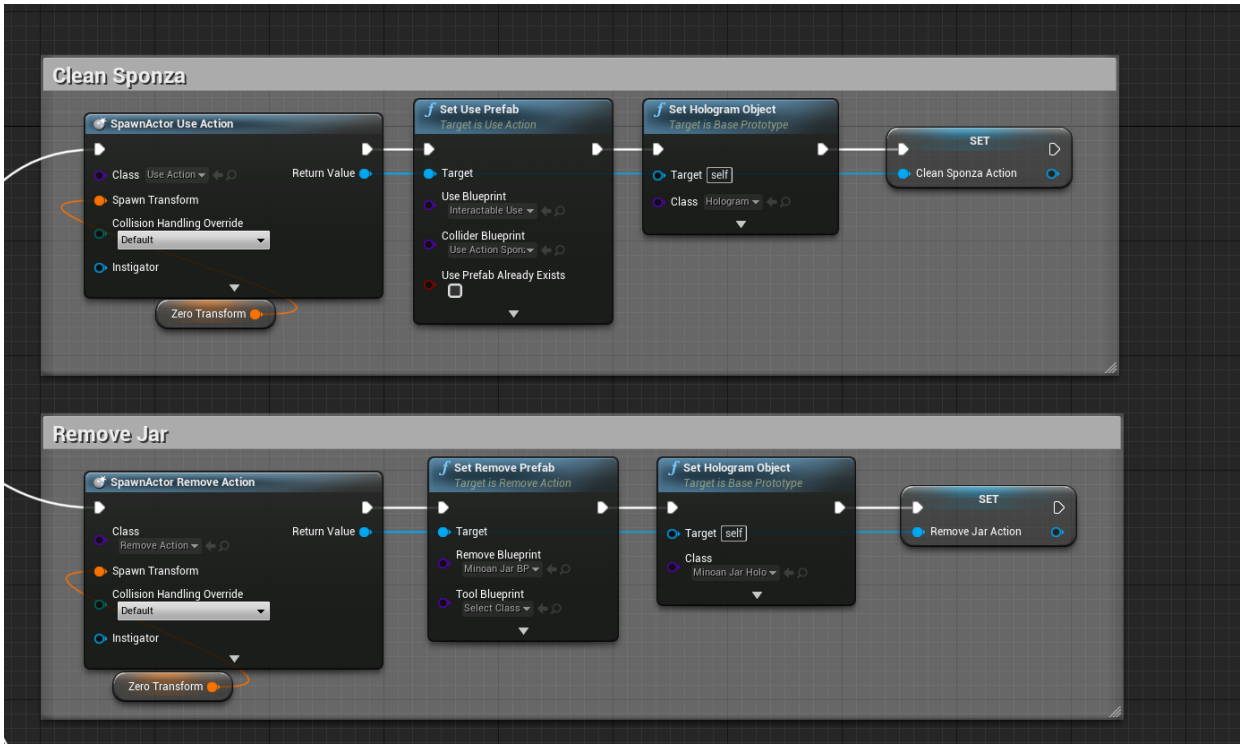
Create a new blueprint, and choose `BPCombinedAction` as its parent class:



The graph below configures the combined action:



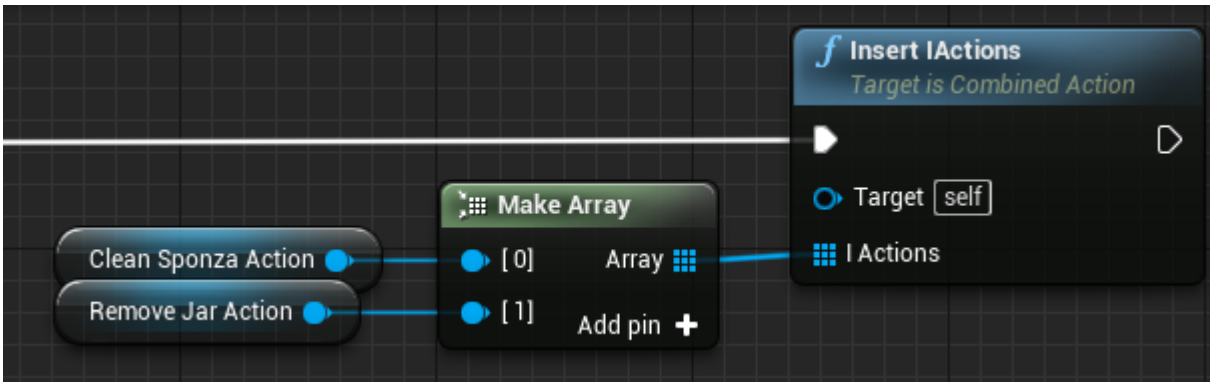
Blueprint Walk-through



By comparing this action blueprint with the blueprints from the *UseAction* and the *RemoveAction* tutorials, we can see that this blueprint contains all of the function calls from the aforementioned actions, as well as some extra calls to set-up the actions themselves, and to register them.

Note: The sub-actions will initialized sequentially; the `Initialize` event of the UseAction will be

called first. When the user completes the action, the `Initialize` event of the RemoveAction will be called. After all the sub-actions have been completed, the `Perform` event of the combined action will be called as well.



The method `Insert IActions` accepts an array of actions (actors that inherit from `BasePrototype`). Actions will be initialized in the order they appear inside the array. This function is mandatory for the CombinedAction to work properly.

11.1.5 Parallel Action

The ParallelAction is used to instantiate two or more Actions simultaneously. Then the user can decide which one to complete first.

Additionally, ParallelAction can modify the Scenegraph tree according to the user's decisions. ParallelActions are usually linked with critical errors. For example, we may implement a ParallelAction which has a correct and a wrong Action to Perform. If the user decides to complete the correct Action, then the simulation will continue as usual. However, if the user completes the wrong Action (for instance inserting an object in a wrong place) then this behavior will trigger the alternative path and the scenegraph will be modified runtime by adding or removing specific Actions.

For instance, if the user causes damage to a human bone, the scenegraph will populate a new stage that forces the user to correct their mistake by repairing the fracture. In another example, if the user paints a car with the wrong color, the scenegraph can add new Actions in order to repaint it.

Note: Parallel actions can now be simulated directly inside the *Live SceneGraph* with the Sequence node

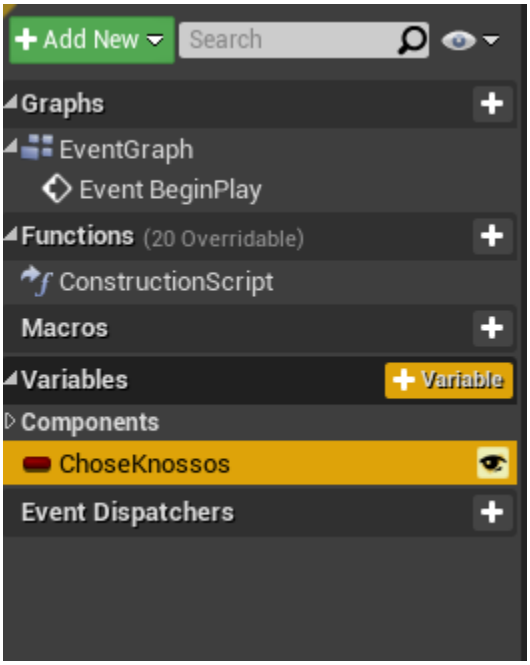
In this example, we will implement a ParallelAction where the user needs to decide whether to assemble the Knossos model or the Sponza model. This Action is part of the SampleApp level named `AssembleKnossosORSponzaAction`

Note: The "normal" path involves the assembly of Knossos and the "alternative" path the assembly of Sponza.

As a result, if the user decides to assemble the Knossos, the scenegraph will move to the next Action. However, if they assemble the Sponza, the scenario will change accordingly to accomodate the decision.

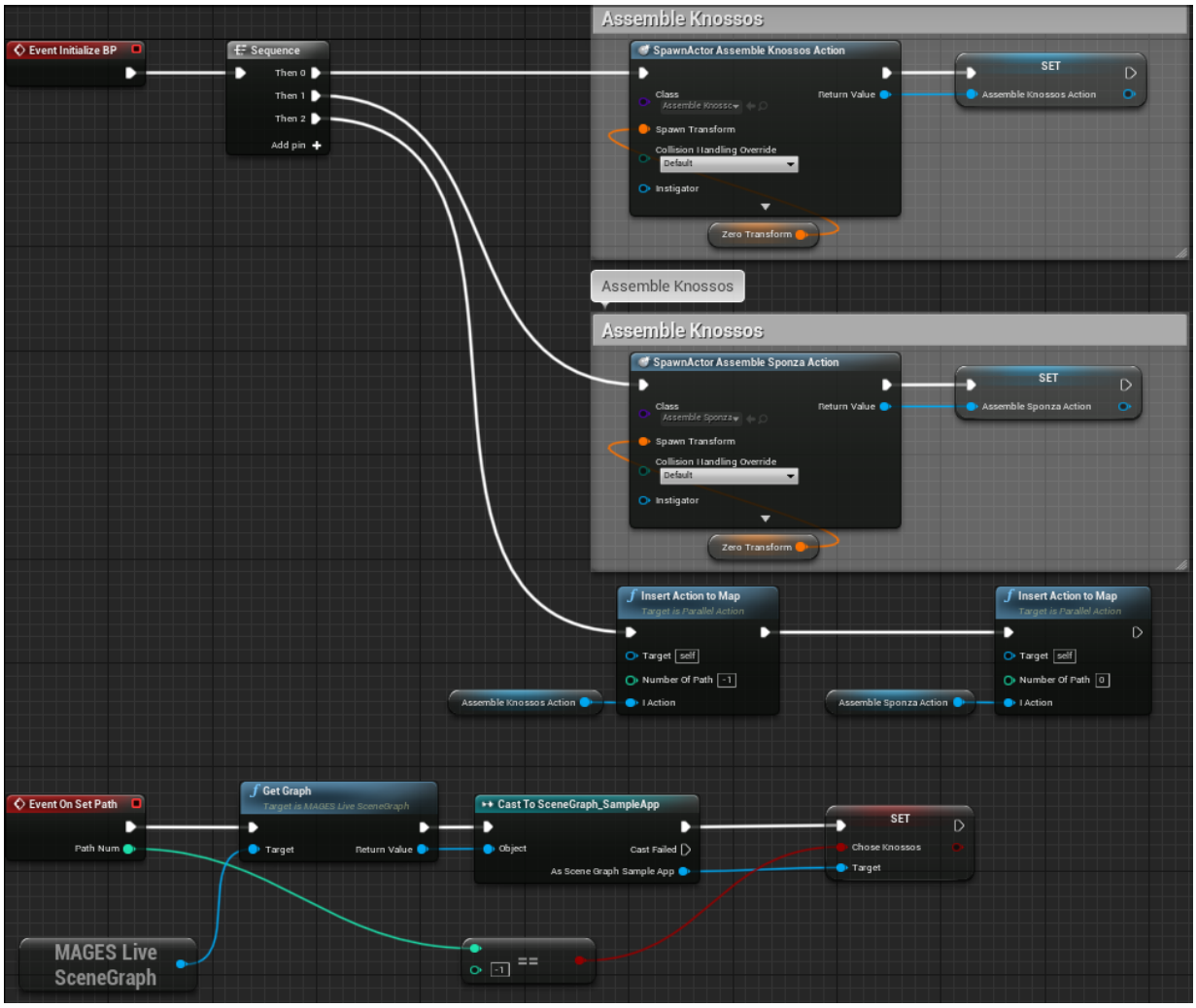
SceneGraph Setup

Inside the SceneGraph (SceneGraph_SampleApp), we create a new boolean variable named "ChoseKnossos". This will be set to true if the user chose to assemble the Knossos model, and false otherwise.



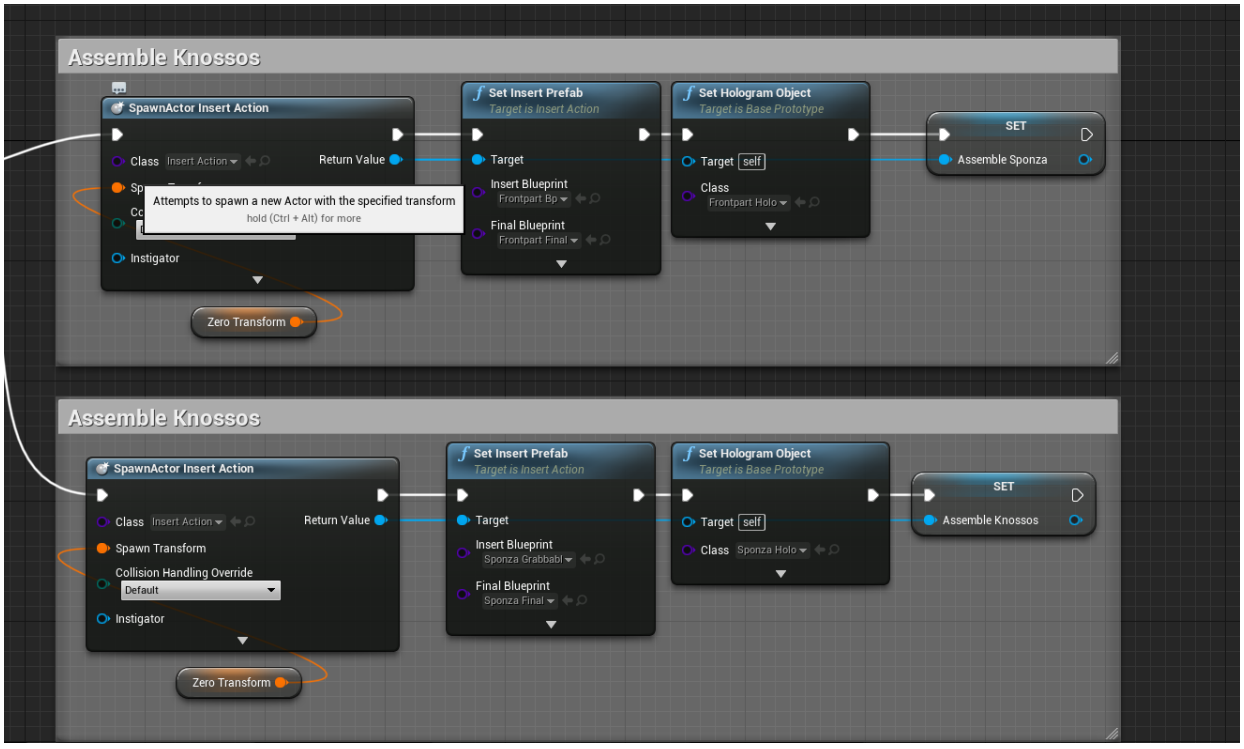
Action Blueprint

The graph below implements the action:

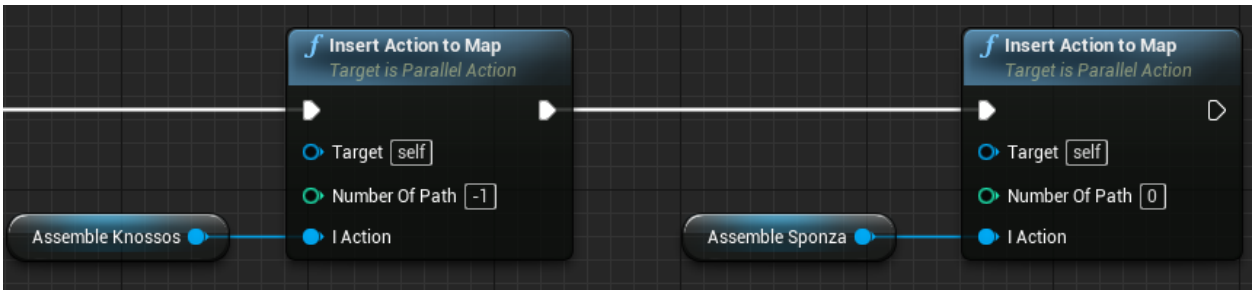


Blueprint Walk-through

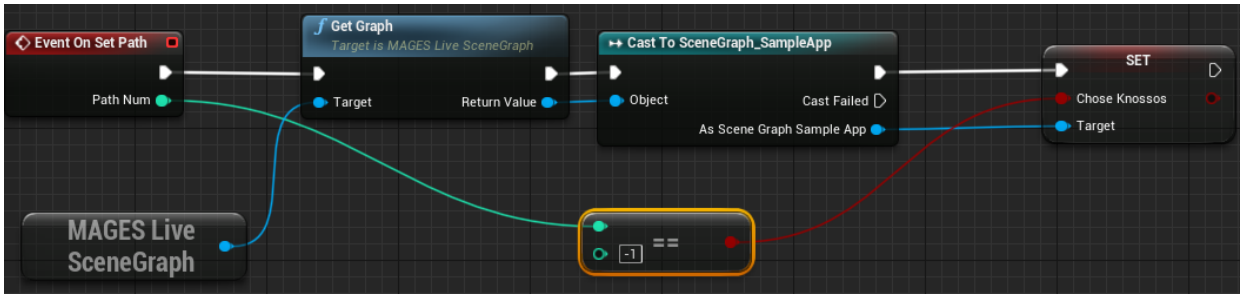
Initially, the two actions which will determine the path are created and configured. They are both InsertActions, and are stored as temporary variables for use later on when the ParallelAction is set-up.



Next up, we configure the number of the alternative path that each action will trigger upon completion. In our example, the `Assemble Knossos` action will trigger the default path (making no change to the scene graph) which is `-1`. If the user chooses to assemble Sponza, it will trigger the alternative path `0`

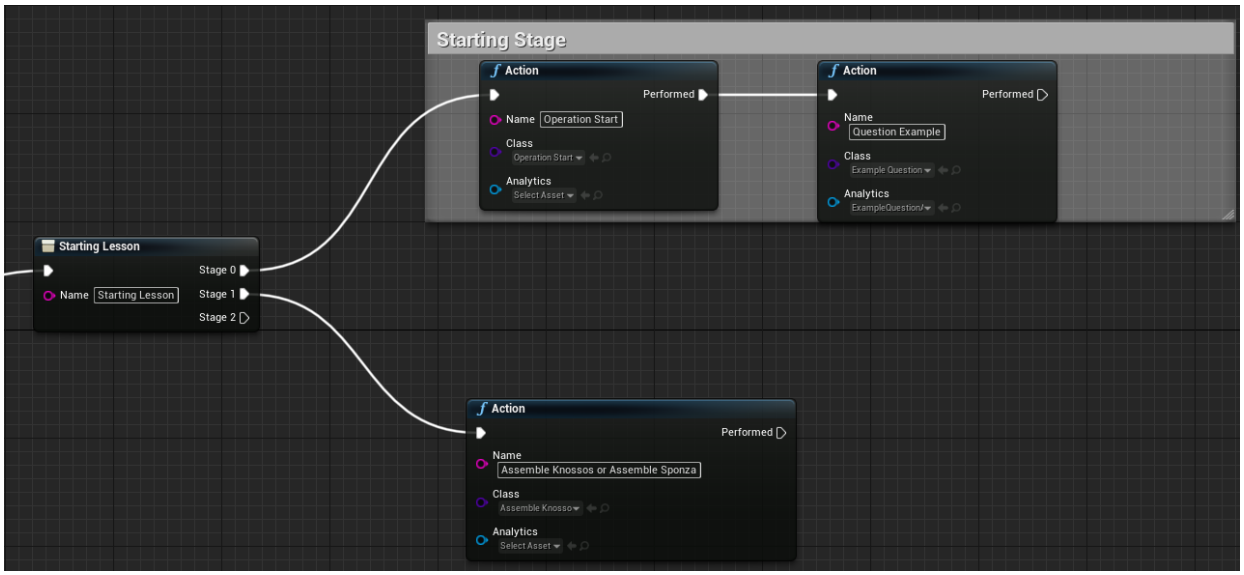


Finally, we implement the event "On Set Path", and set the variable of "ChoseKnossos" depending on the path number:

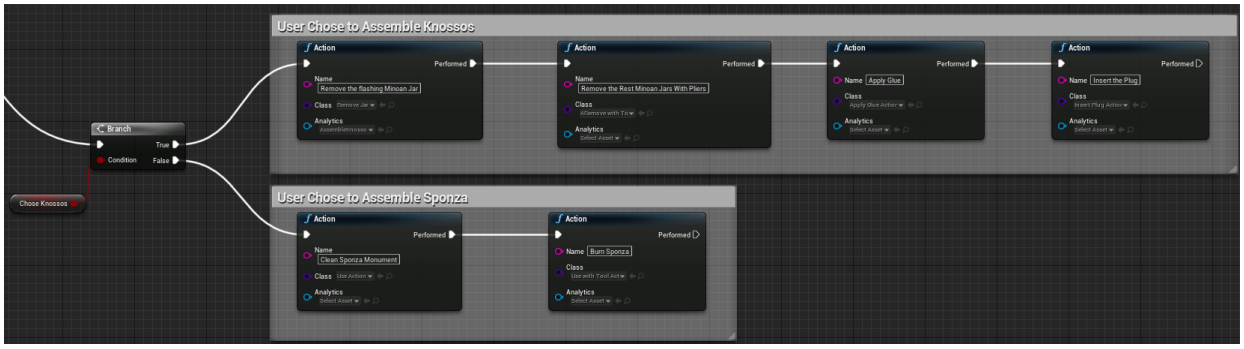


SceneGraph Configuration

The final part is to handle the actual logic after the action has been completed, from the Scene Graph:

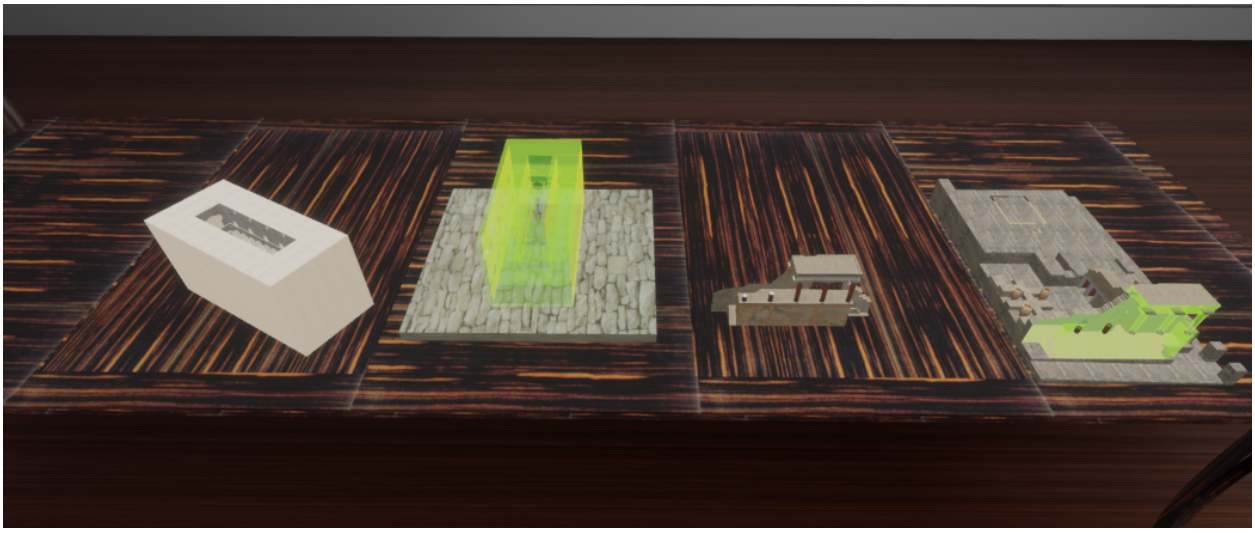


In the image above, the action in the second stage of the Starting Lesson includes the class for our new parallel action. After this action is performed, the variable of "ChoseKnossos" will have a valid value depending on what the user chose to do.



Parallel Action Explanation

Now let's see what happens when we start the application. Move to the `AssembleKnossosORSponzaAction` Action that will set the "ChoseKnossos" variable. As you can see both Actions are Initialized simultaneously, the user should decide which one will complete. To trigger the alternative path we have to insert the Sponza model.



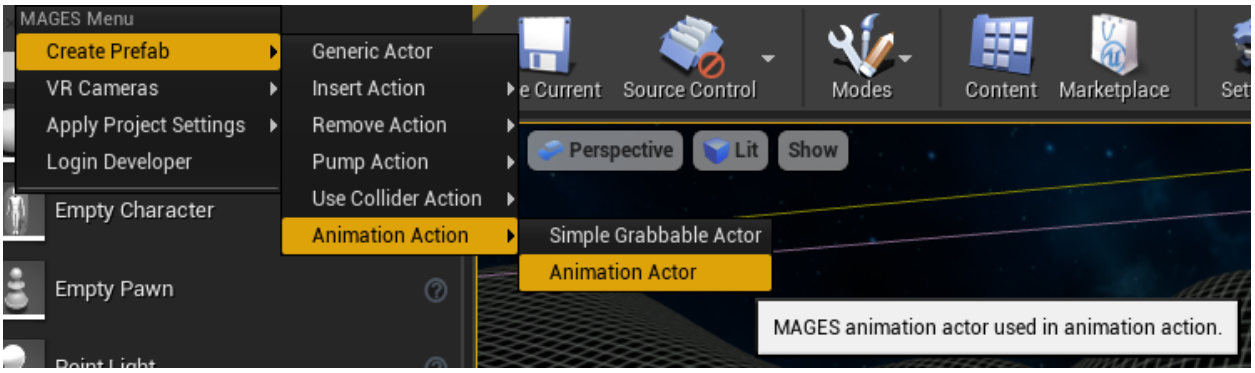
Upon performing the action by choosing to assemble Sponza, the variable of "ChoseKnossos" will be set to false, and the scene graph will spawn the actions relevant to Sponza.

11.1.6 Animation Action

In this tutorial we will learn how to create and use an Animation Action and its componenets.
Animaton Actions use two actors the **Animation Actor** and the **Interactable Actor**.

Animation Prefab Constructor

From the MAGES menu select the option Create Prefab/Animation Action/Animation Actor to create a new instance of an animation actor, as well as Create Prefab/Animation Action/Simple Grabbable Actor to create a new instance of an interactable actor.



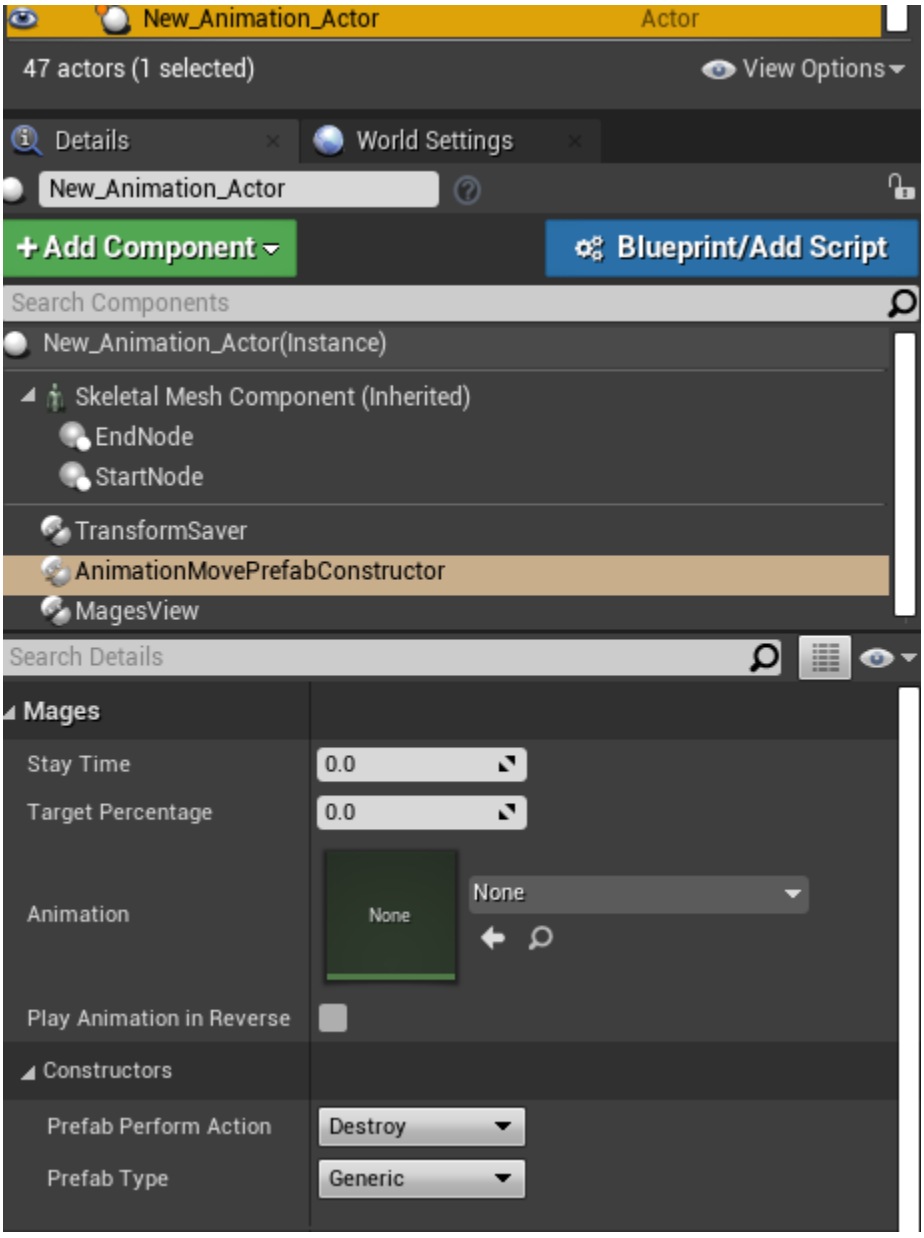
The template Actors will appear in the scene. It is recommended to configure these objects for your Animation Action. The Animation Actor has 3 important components;

1. **AnimationMovePrefabConstructor** 4. **End node** 5. **Start node**

- The AnimationMovePrefabConstructor is used to reference all the needed assets for this action as well as configure its variables.
- The End Node is used to set the finishing point of the animation. When the Interactable Actor has the same position as the End Node, then the Animation Action will be performed. We will get on that later on the tutorial.

- The Start Node, similarly with the end node, is the starting point of the animation.

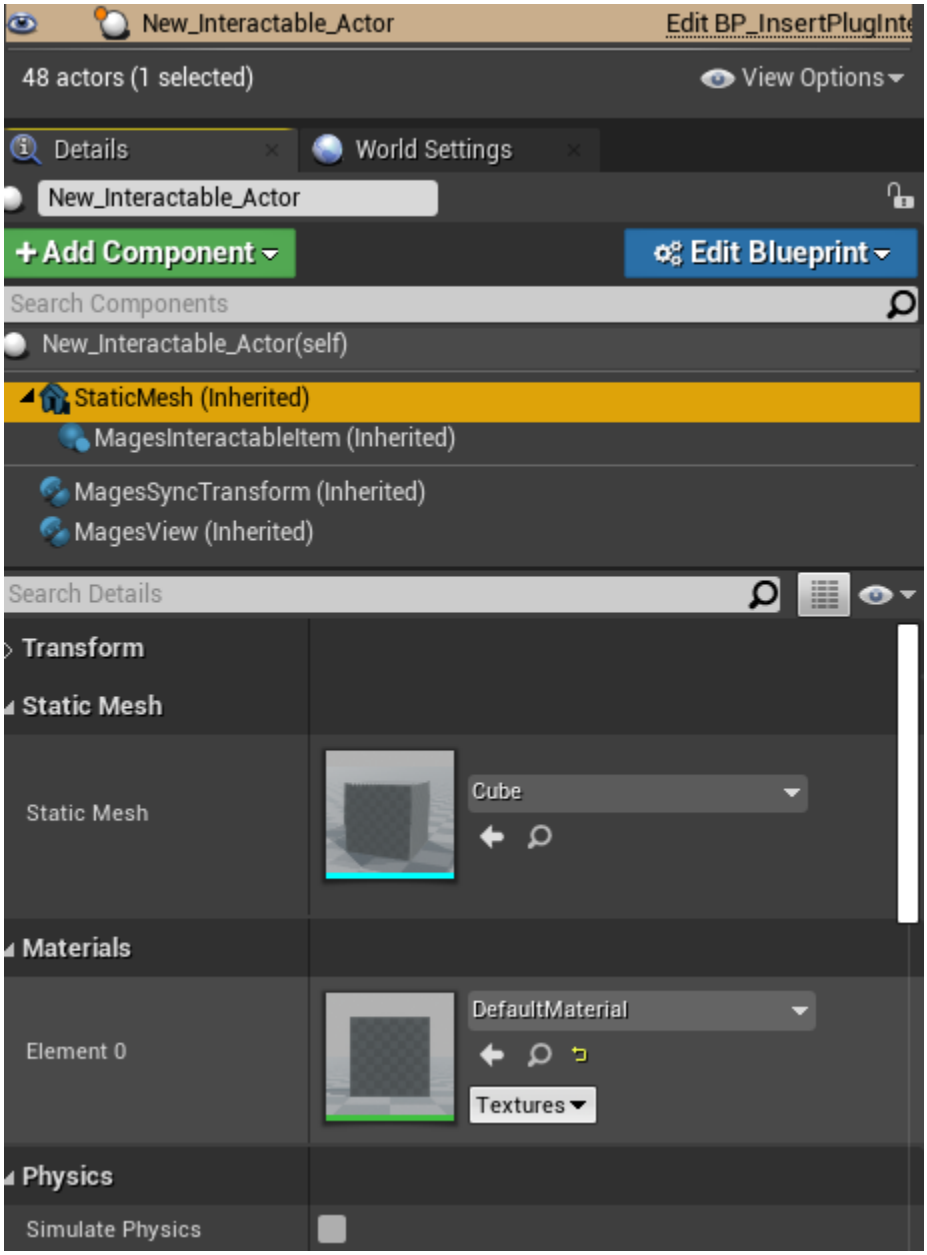
The image below shows the actor and its components.



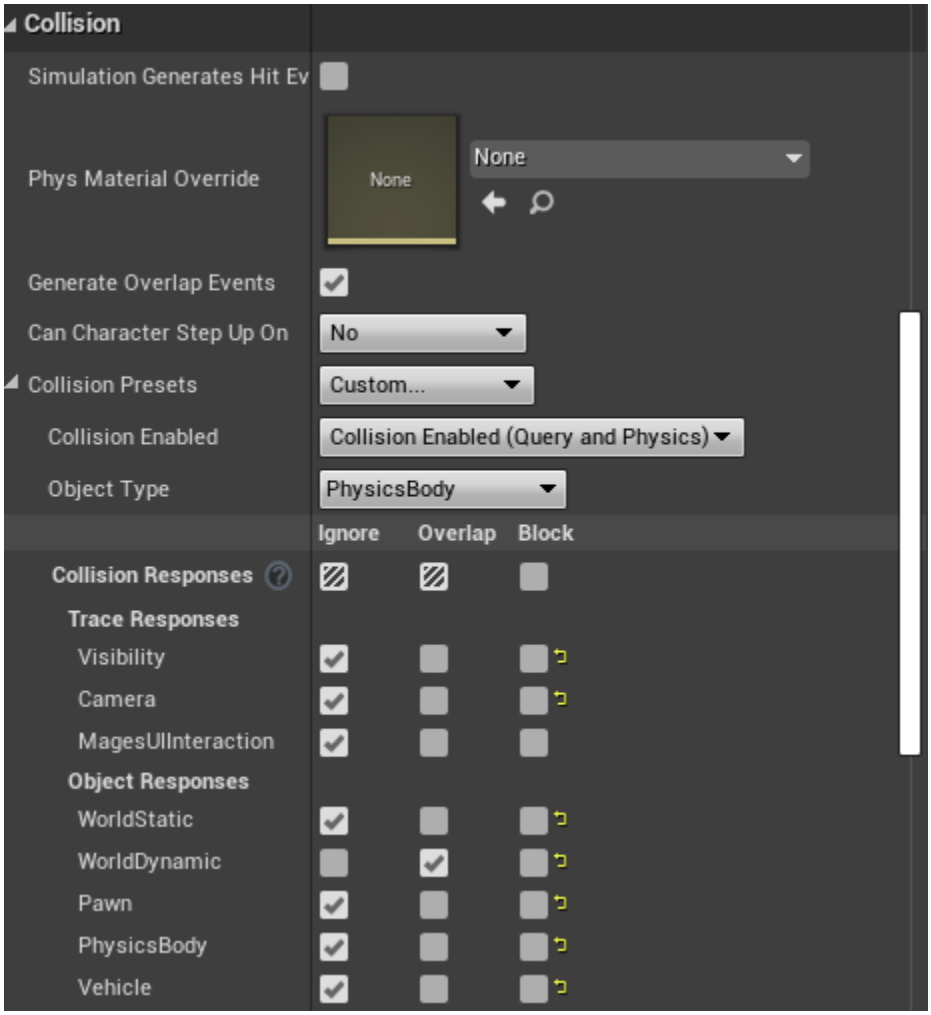
So, let us begin creating our first animation action blueprint.

Configuration of the interactable Actor

The configuration of the interactable actor is pretty easy and straightforward. | We need to add a mesh in the static mesh component as well as turn off Simulate Physics. This mesh will define the overlap boxes of our interactable actor. | Make sure that Simulate Physics is turned off



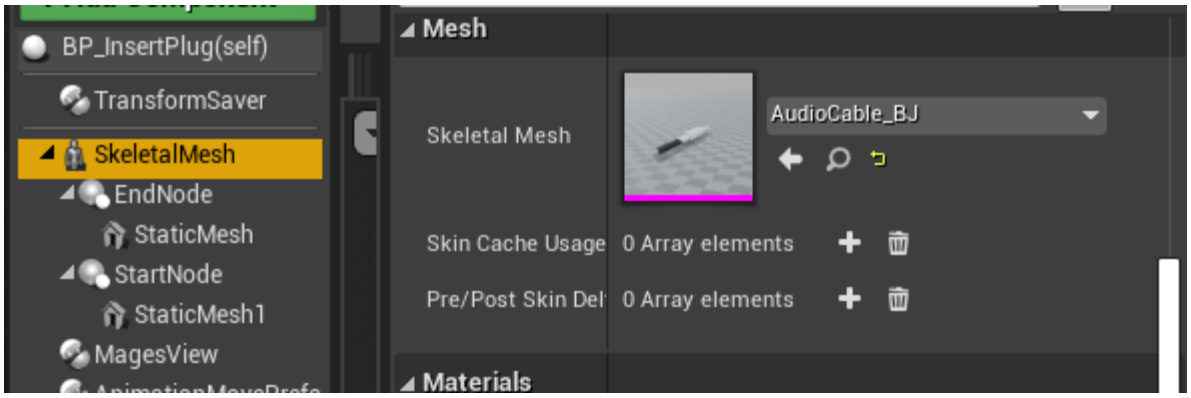
Also in the physics component of the static mesh, we need to make sure that it generates overlap events when overlapping with Dynamic objects (the user hands).



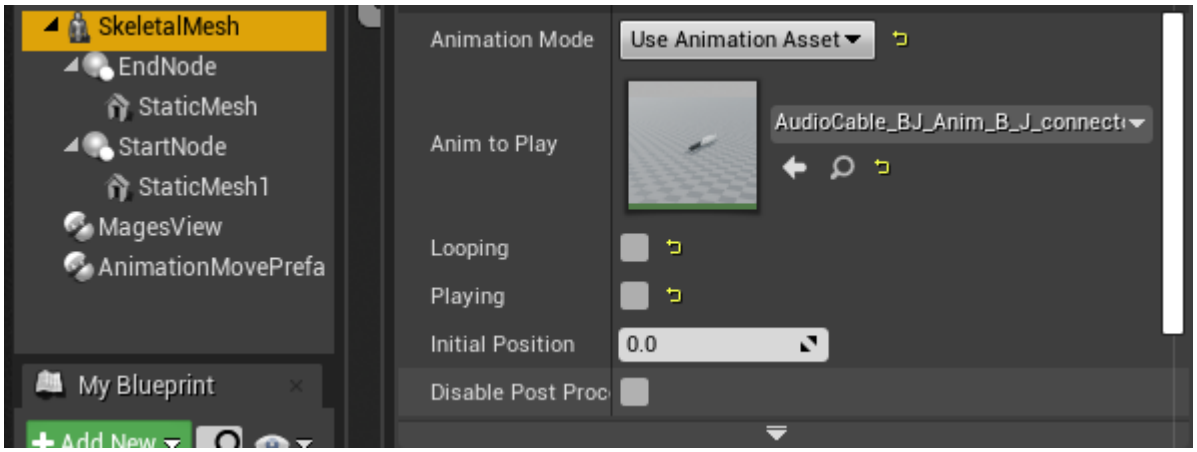
Finally we need to save our actor as blueprint class that will be referenced by the **AnimationMovePrefabConstructor**.

Configuration of the Animation Actor

First of all we need to create a new blueprint class fromt he animation actor template in the scene. | Once we have our blueprint we can open it and select the skeletal mesh component. There we need to reference our skeletal mesh object, that has been animated.

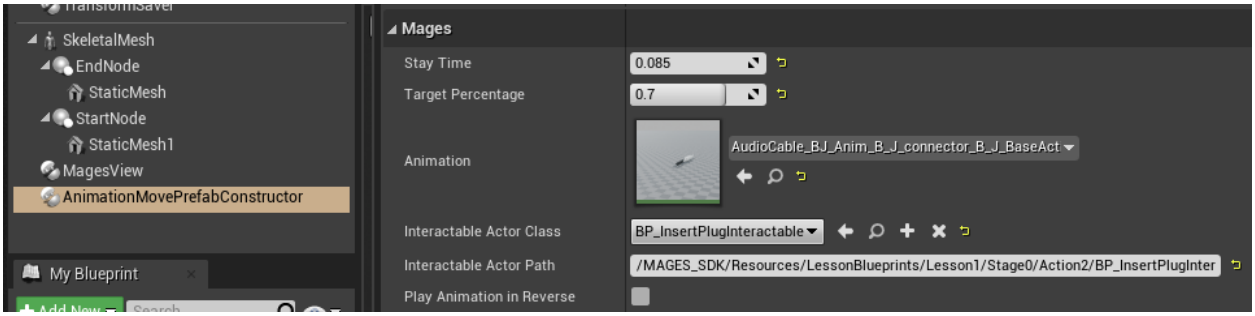


Next we need to reference the animation that we'll be played while we are moving the interactable actor.
Make sure that looping and playing check boxes are not selected since these will be handled internally from MAGES.



Continuing to the **AnimationMovePrefabConstructor** component, we have to configure the following variables:

1) Stay time : How much time the interactable actor needs to stay at its final position in order for the action to perform.
2) Target Percentage : The percentage of the distance that needs to be traversed in order for the action to be completed (or the stay timer to start counting)
3) Interactable Actor Class : The blueprint interactable actor class for this action.
4) Interactable Actor Path : The path to the interactable actor.
5) Play Animation in Reverse : In case we need the animation to play in reverse as we move the interactable actor from the start node to the end node.



We also need to define the start and end positions of the object. As we move the interactable actor in this spectrum the animation will play from start to finish.
This is defined by the transform of the **strat node** and **end node** components.
In the following picture we've highlighted with blue cube the starting transform (left) and end transform (right) in our example.

We need to define where this actor will be spawned. Drag and drop it in the scene and after placing it in its correct transform, we select the transform saver from its components and click the **Save Transform** button.

Finally, we click edit blueprint and select **Apply Instance Changes to Blueprint**

Creating the Animation Action Blueprint

As with all actions we need to create its blueprint class. We right click in the content browser, select blueprint class and select to inherti from BPAnimationAction.

Below you can see our example animation action, which requires to blueprint nodes: | 1) Set Animation Prefab, where we reference the Animation Actor Blueprint. | 2) Set Hologram Prefab, where we reference the assistive Hologram.

Finally, simply save the blueprint and reference it from the scenegraph blueprint and we can test our animation action.

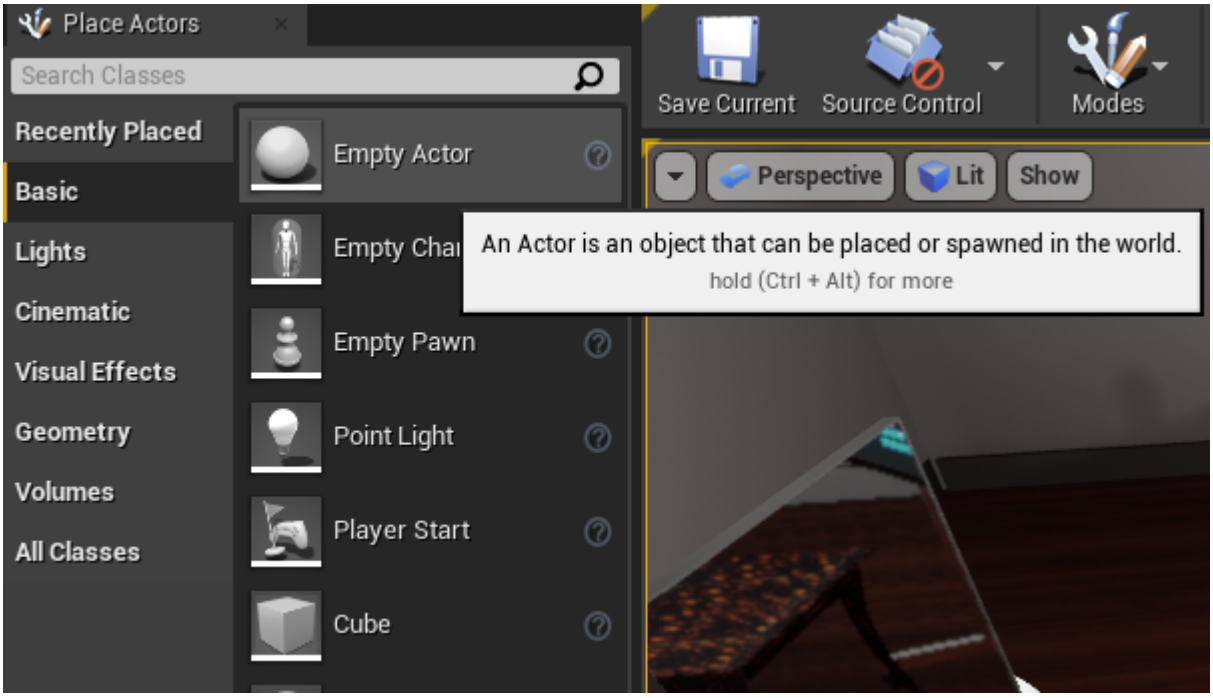
11.1.7 Question Action

To create a question action, a Blueprint with a `Question Prefab Constructor` component is required. In this tutorial, we'll recreate the Question Action from the Sample Application level. The question will involve a prompt asking the user: "Where is Sponza located?", with 3 incorrect answers, and 1 correct answer.

We'll start by creating the question blueprint first, and then use it inside the action.

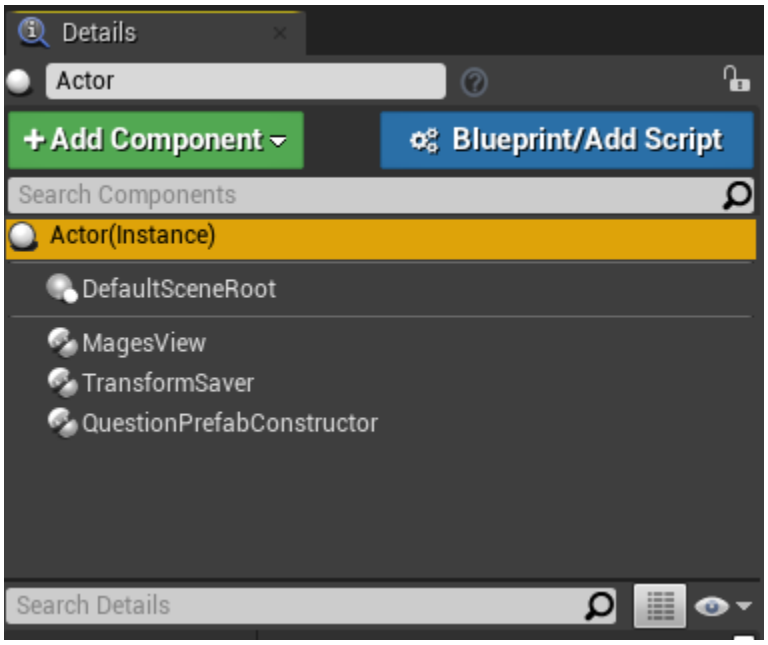
Question Blueprint

From the Place Actors tab, drag an empty actor into level, and place it where the question needs to appear:

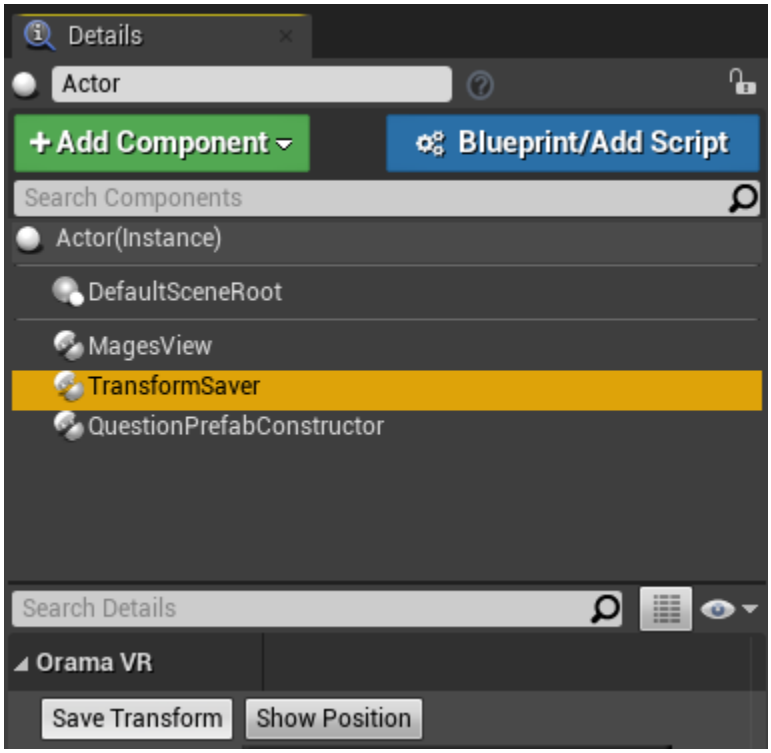


Add the following components to the actor:

- Transform Saver
- Mages View
- Question Prefab Constructor

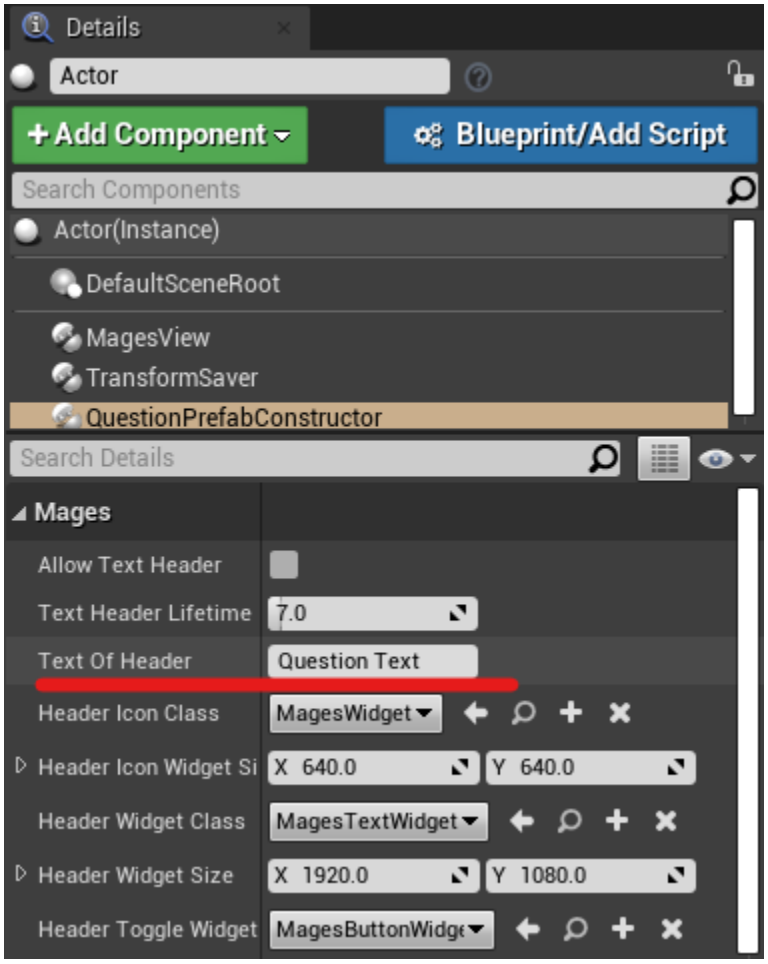


Select the Transform Saver component, and click “Save Transform” to save the location and orientation of the blueprint actor:



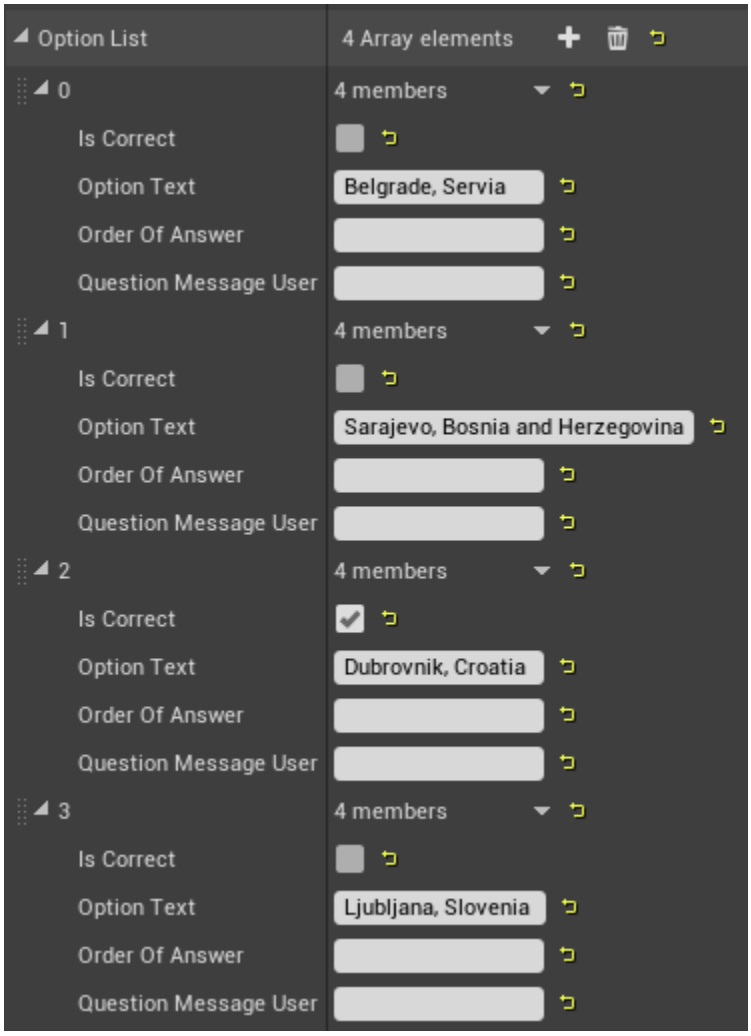
Next up, the Prefab Constructor needs to be configured with the Question text itself, as well as the various options that will be available to the user.

Select the `QuestionPrefabConstructor` component, and find the "Text of Header" property:

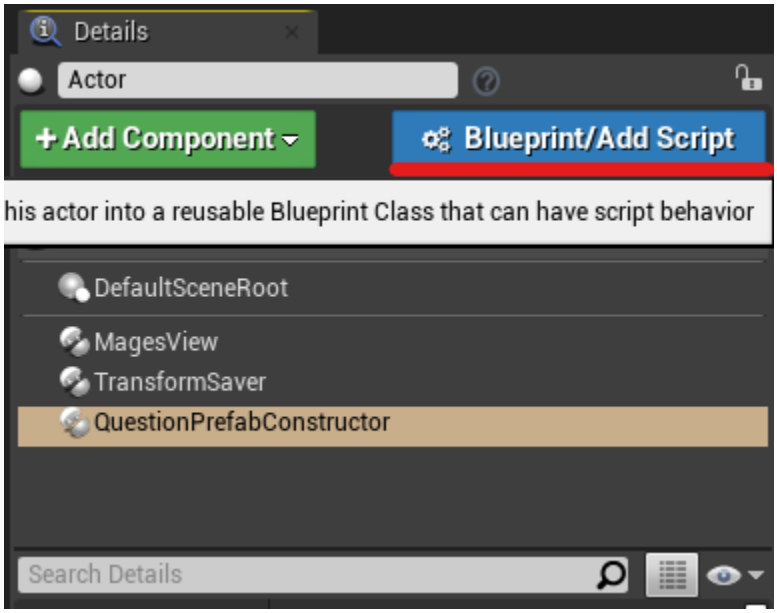


This will be the prompt seen by the user, before the options appear; Set it to: “Where is Sponza located?”.

Additionally, in the “Option List” property, click the ‘+’ button 4 times, to add 4 new possible answers. Expand all of the new items, and set their properties as seen below:

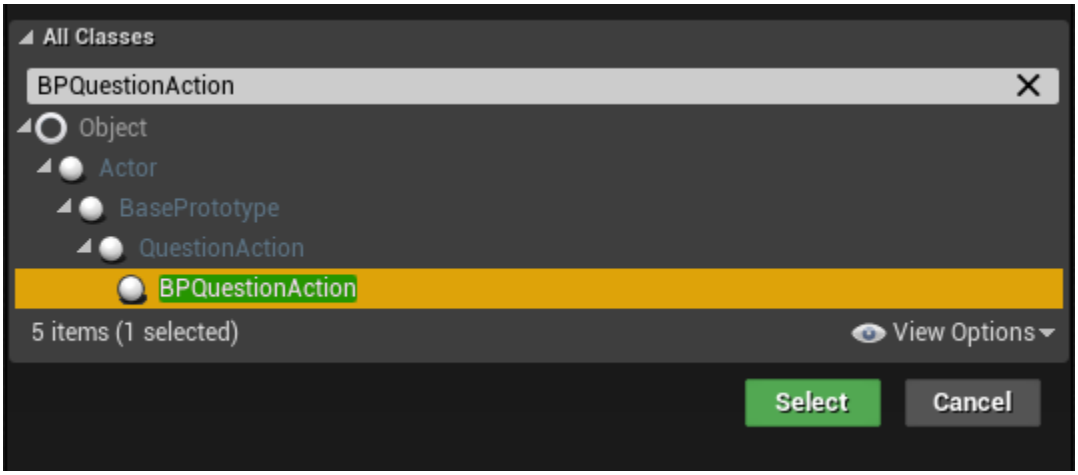


We have finished configuring the constructor, so all that remains is to save it as a blueprint: Click the "Blueprint/Add Script" button, and save it wherever you would like:

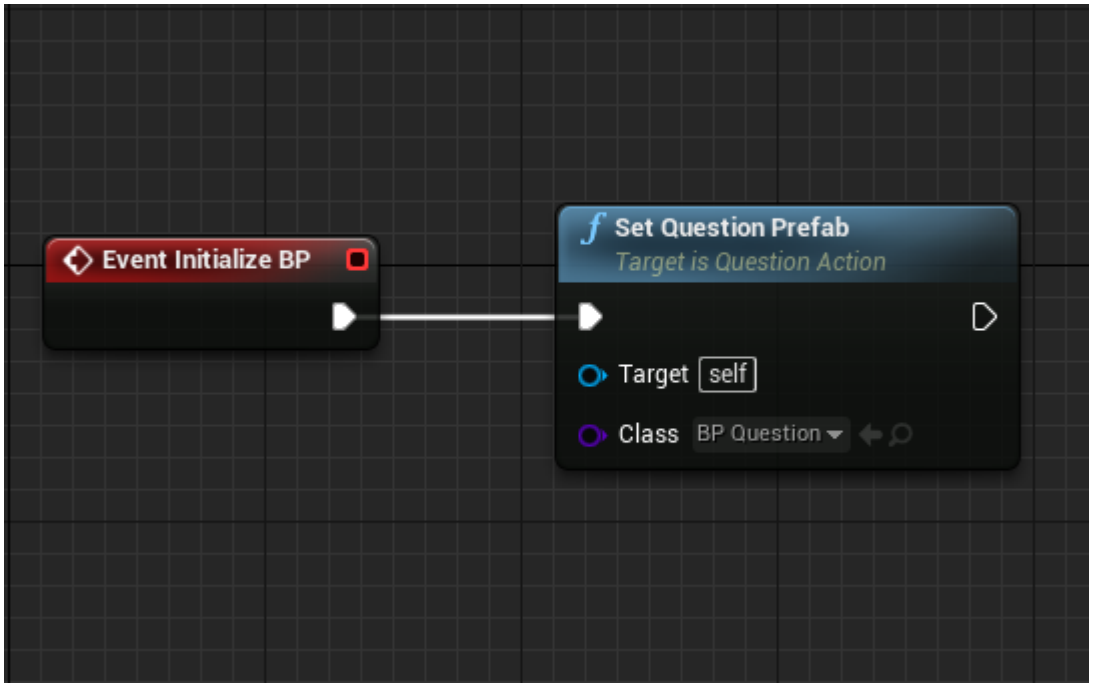


Question Action

To create the question action, create a new blueprint that inherits from the BPQuestionAction class:



The blueprint node for the question action can be seen below. Essentially, it registers our Question Blueprint as the blueprint to be used for the action itself:



Add Action to the Scene Graph

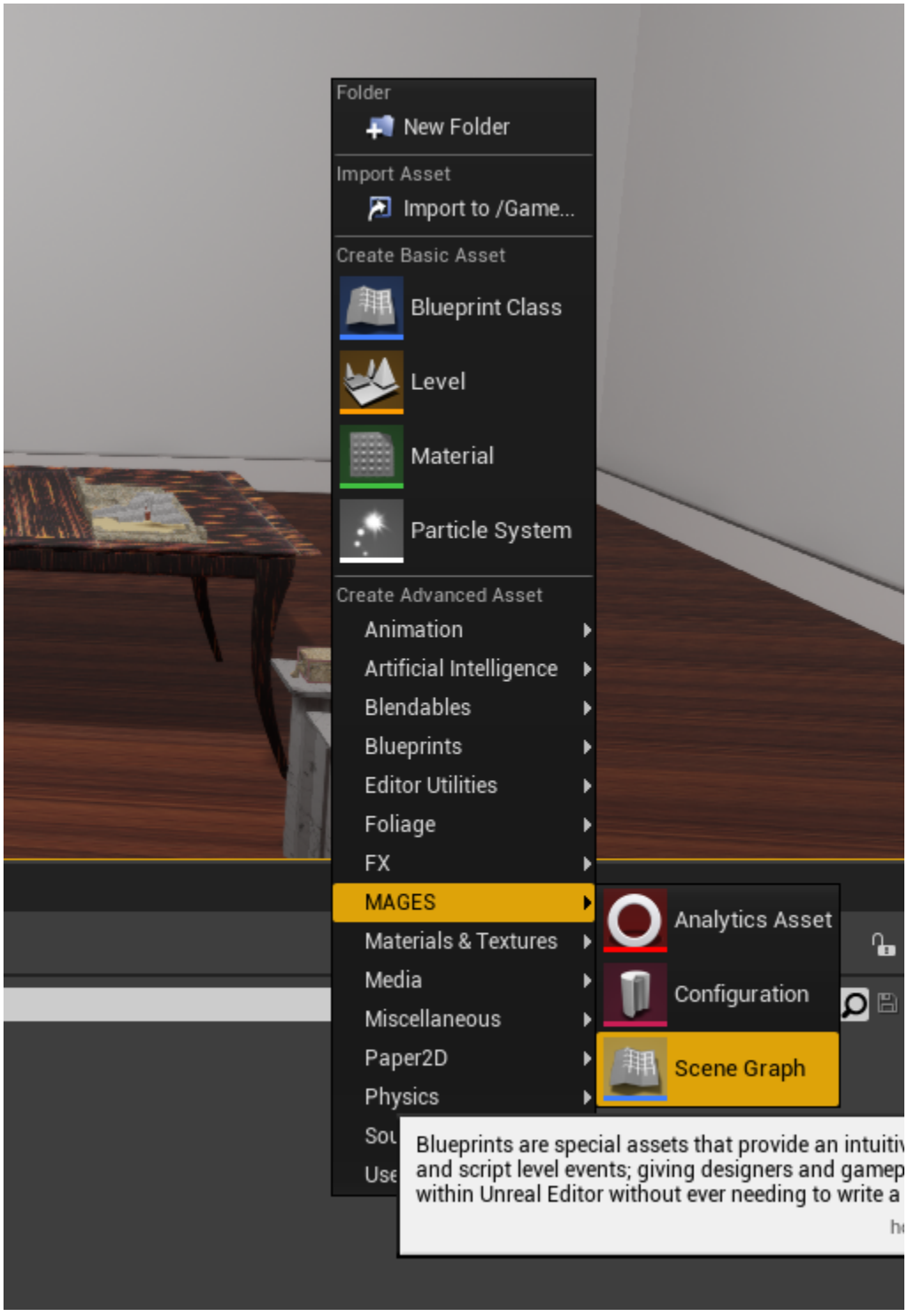
Finally, we have to add this action to the Scene Graph. You can view a detailed tutorial on how to do this *here*.

11.2 Scenegraph Generation

In this tutorial we will generate a scenegraph blueprint from scratch. We strongly suggest that you take a glance at Unreal's documentation on how to use the Blueprint Graph Editor

11.2.1 Creating a New Scenegraph

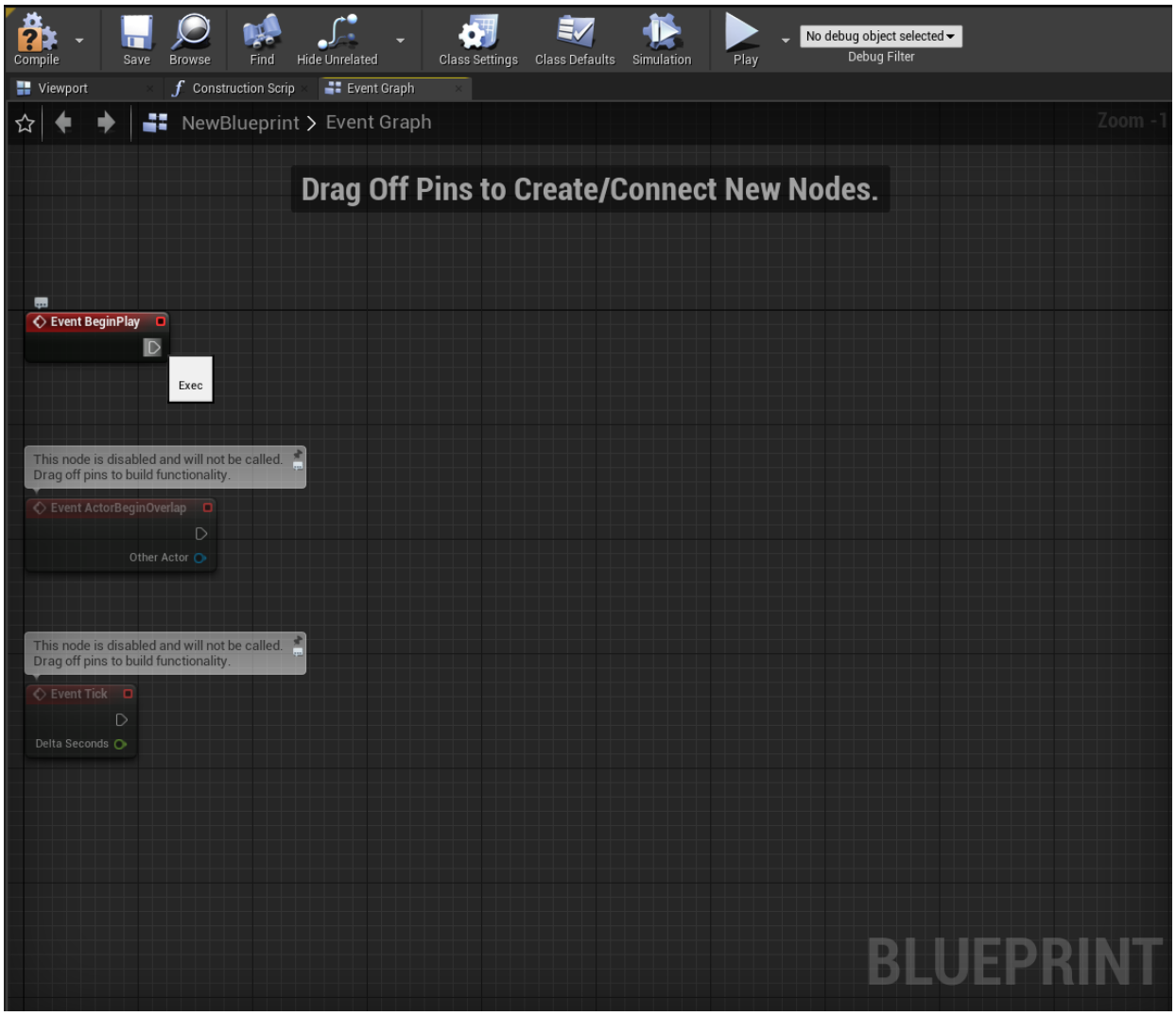
Right-click on an empty space in the **Content Browser** and choose MAGES > Scene Graph:



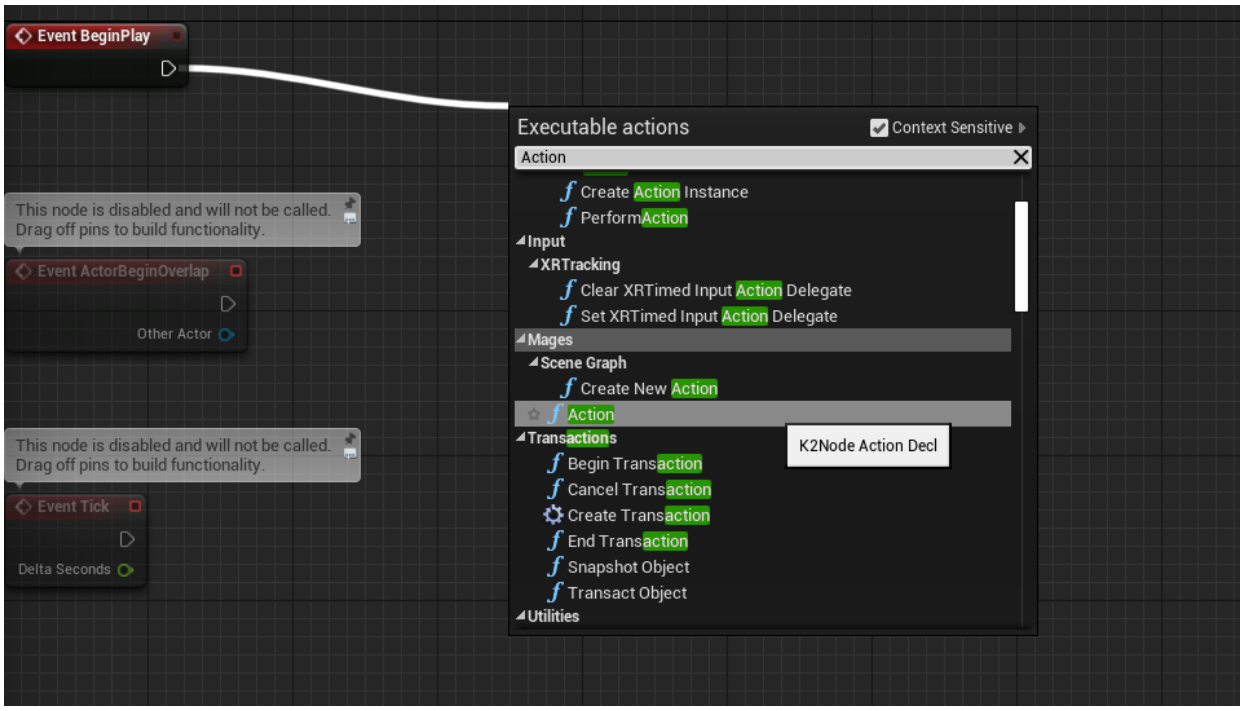
And open the Blueprint.

11.2.2 Creating a New Action

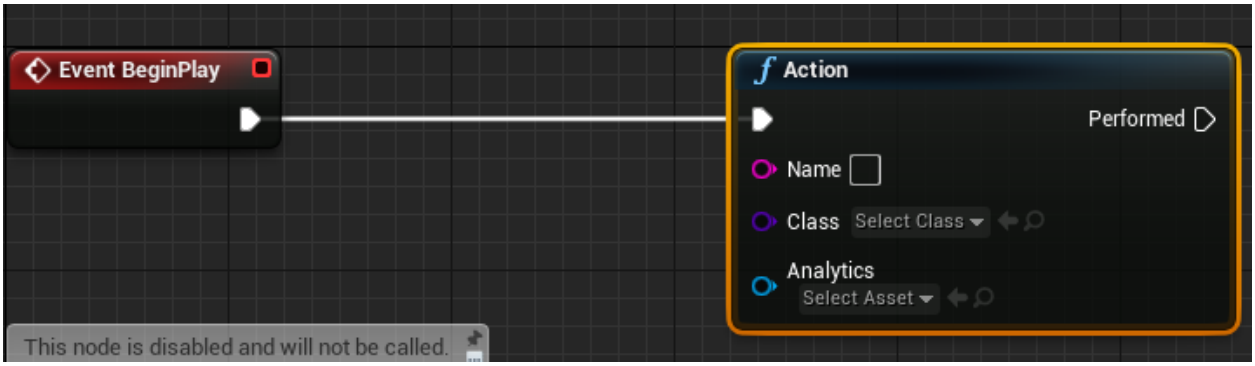
In the Event Graph tab, drag a connection link from the node “Event Begin Play”:



Once you let go of the left mouse button, a popu menu will appear listing all of the available nodes that you can use. Search for the node named “Action”:



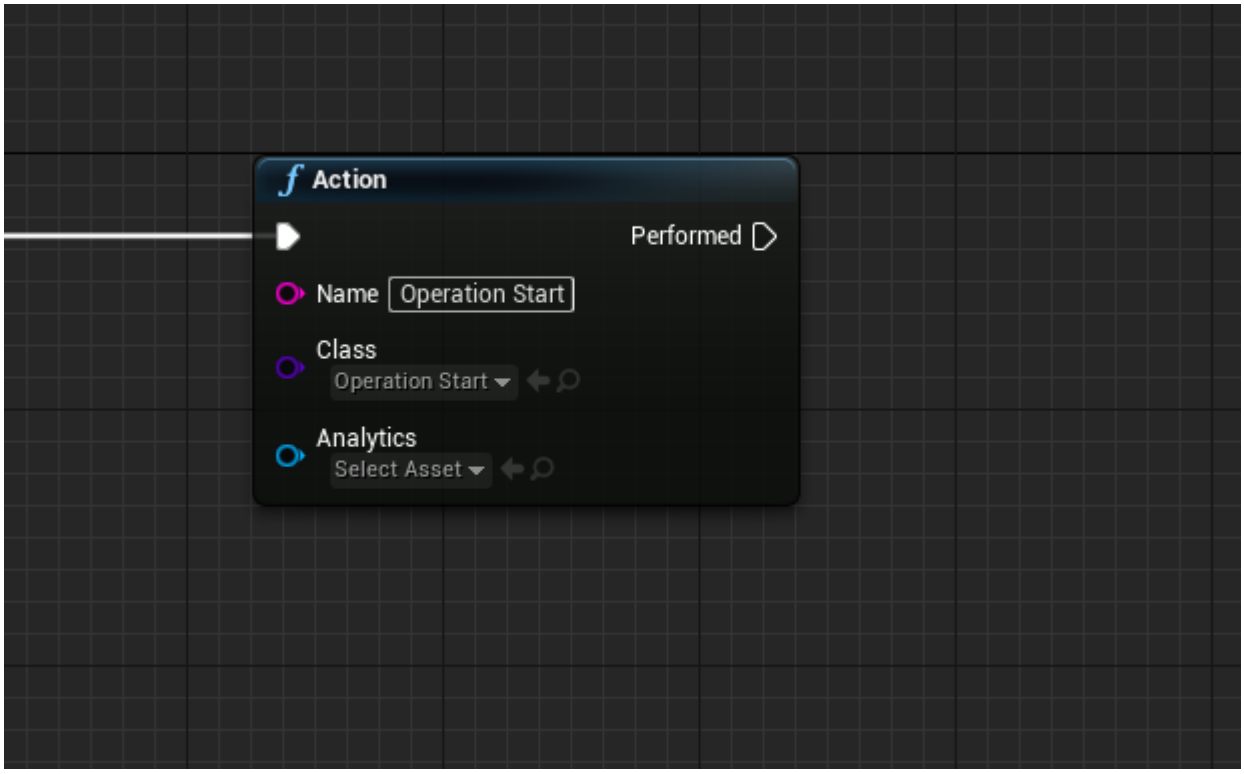
And press enter, or click on the search result. You should see a node like this:



This node is the basis of any MAGES Scene Graph. It's parameters are described in order:

Parameter	Description
Name	The identifier of the action. Do **not** leave this empty!
Class	The class of the action blueprint to use for this action
Analytics	The analytics asset to use for this action

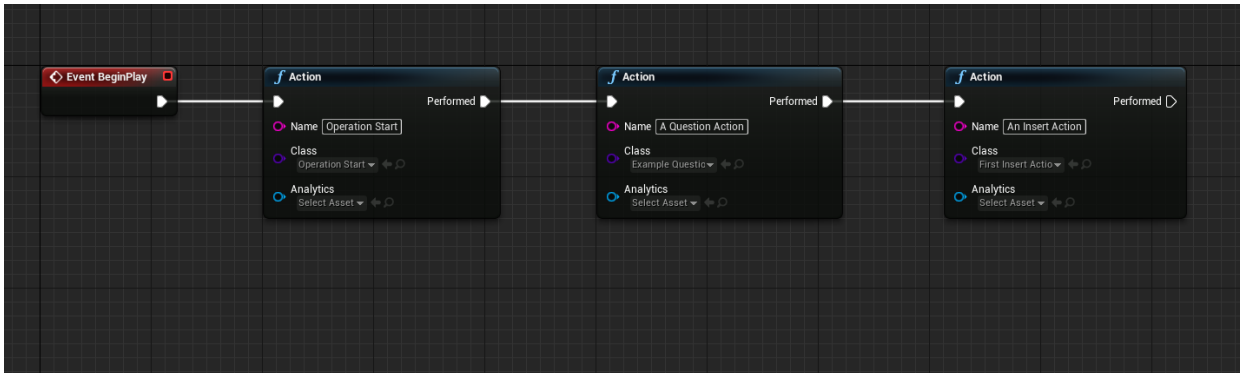
Let's set these values to something meaningful:



Usually, the first action of any Scene Graph is called: "Operation Start". The Analytics parameter can be left empty.

11.2.3 Adding More Actions

In the same manner you can create more actions to be executed after this one, by dragging connection links from the **Performed** pin of the last node:

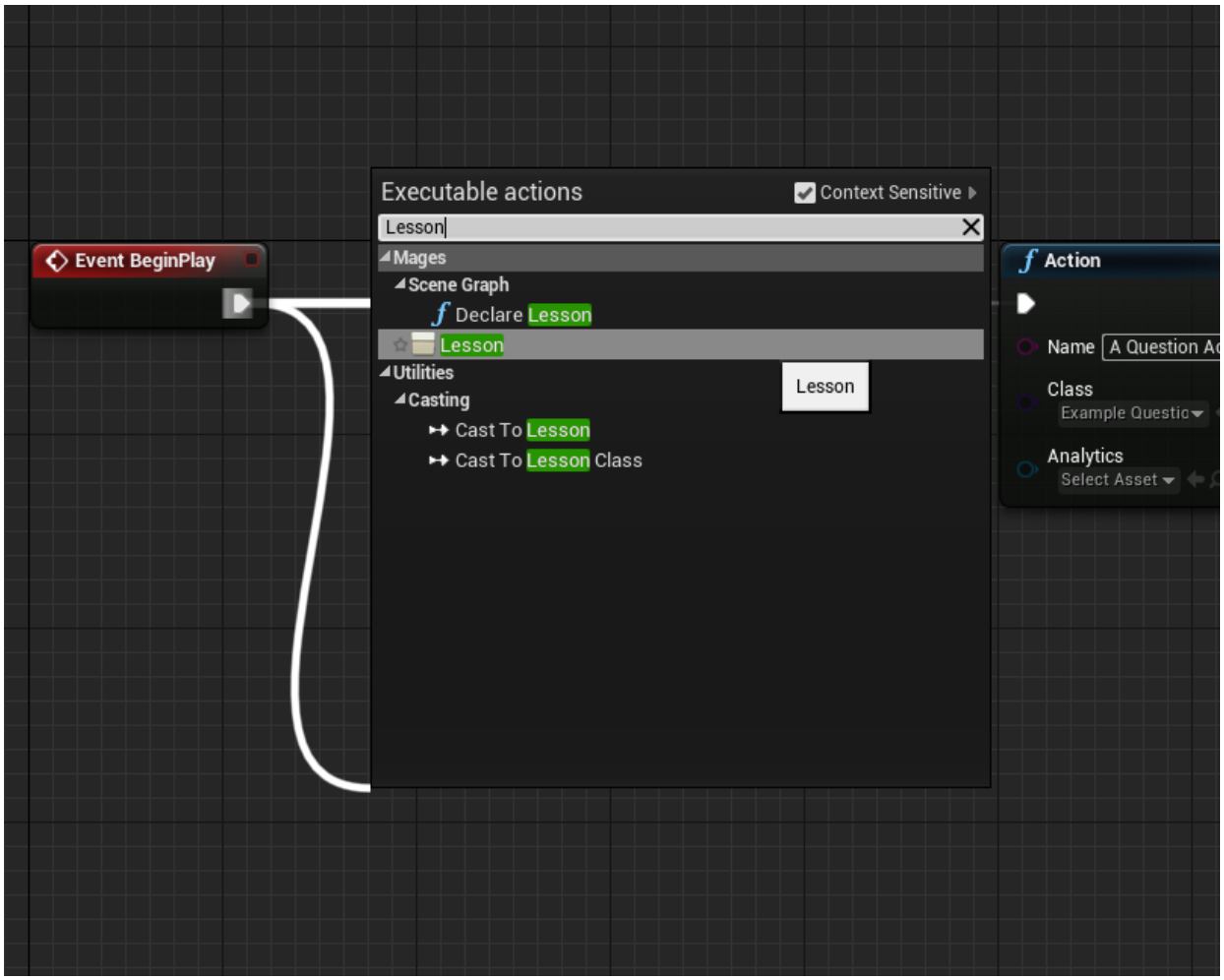


The order in which the action will be executed will be:

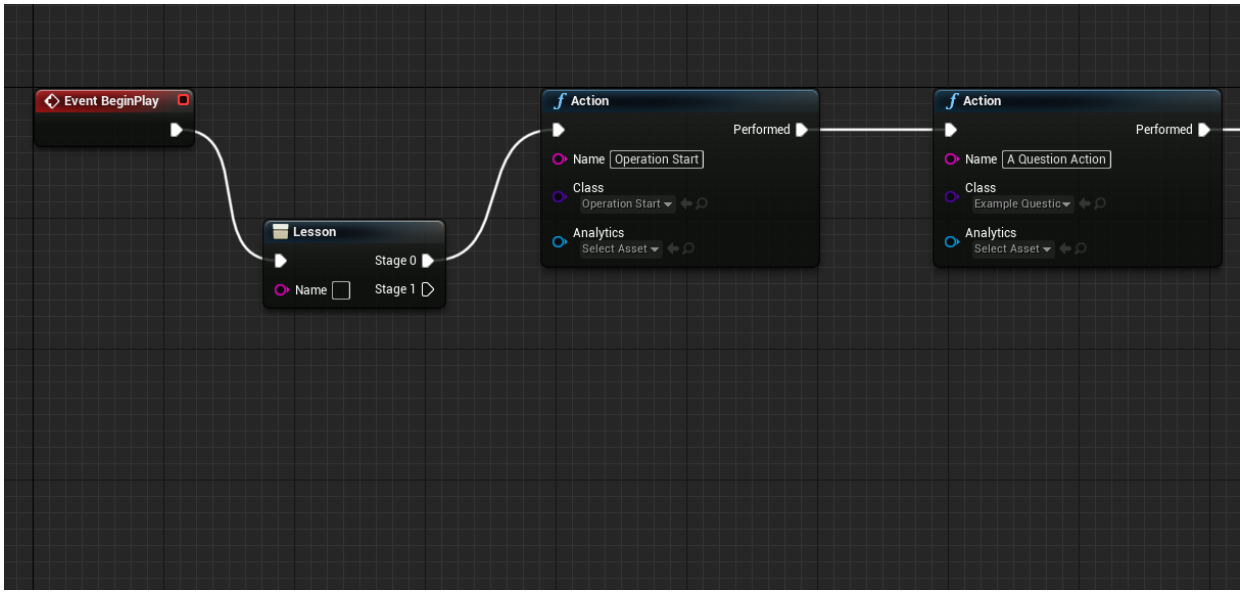
- 1. Operation Start
- 2. A Question Action
- 3. An Insert Action

But this is getting a bit unwieldy, imagine 10 or more actions linearly connected - with no structure; you can do it, but it would be very difficult to manage visually!

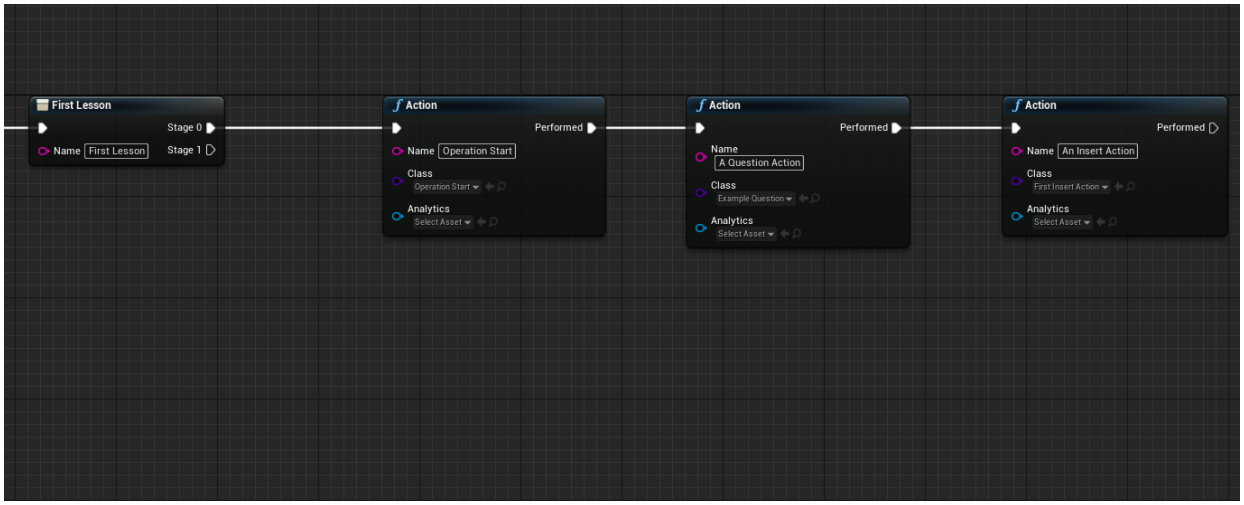
This is where the “Lesson” node comes in. From the node “Event Begin Play”, drag a new connection link, and search for “Lesson”:



You should see something like the following:



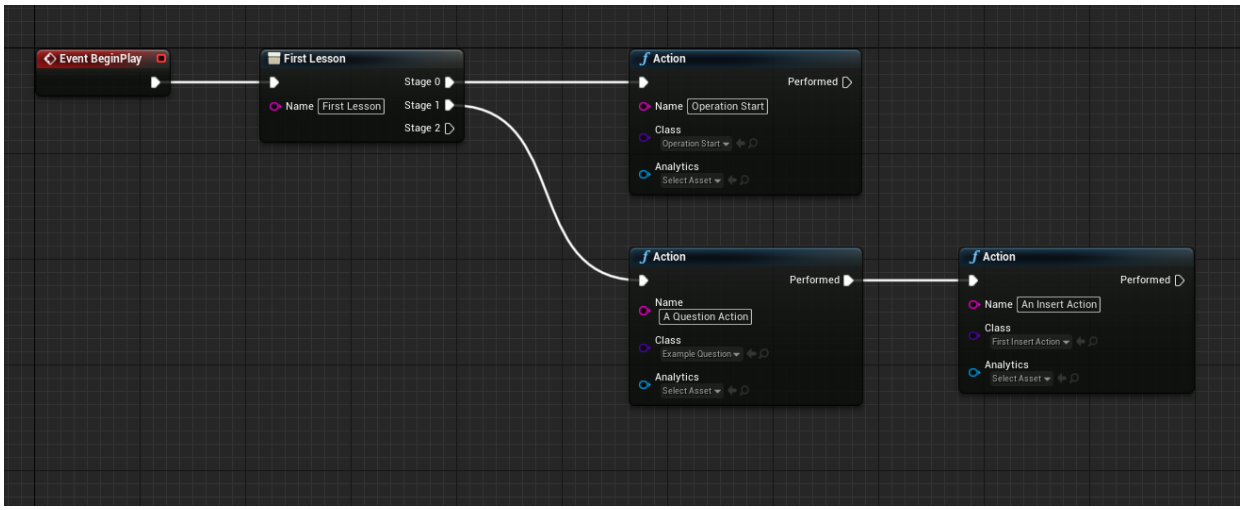
Now click inside the text entry box of the lesson node, and name it something meaningful, like “First Lesson”:



You can now see that the node’s title has also changed to reflect the name you’ve entered. Let’s discuss about what the Lesson node does:

11.2.4 The Lesson Node

The Lesson node is a node that executes all of the actions conencted to it **in order**. For example, if we disconnect the Action named “A Question Action” (by grabbing the input pin with `Left Ctrl` held down), and reconnect it to the “Stage 1” link of the lesson node:



We can see that:

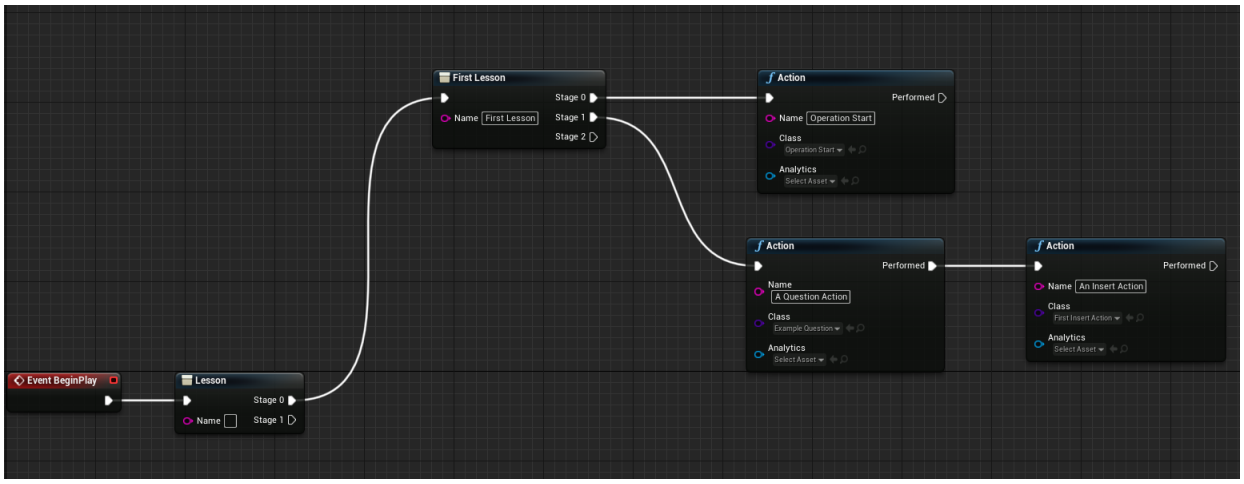
- **Stage 0** has only once action
- **Stage 1** has two actions; one connected to the next
- The Lesson node has now created another Stage pin, named **Stage 2**

Now, this Blueprint is strating to look more like a tree, but really, nothing has changed; the actions will be executed in the same order as before:

- 1. Operation Start

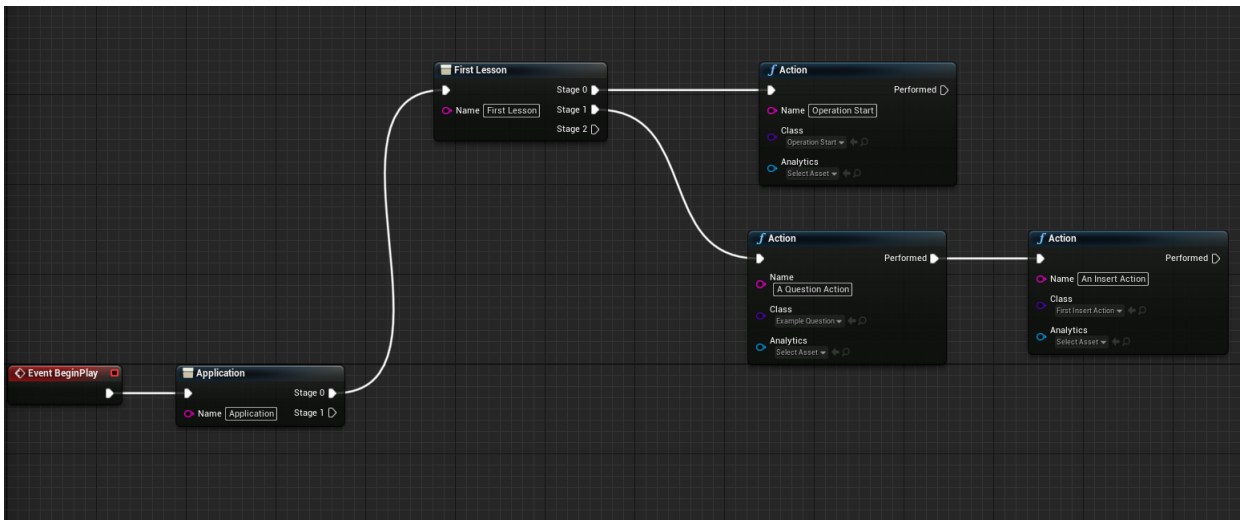
- 2. A Question Action
- 3. An Insert Action

The Lesson node is only used for organizational purposes, so that we don't end up with a huge linear graph of actions. In that vain, you can also add lesson nodes within other lesson nodes:



What've done now, is we created another lesson node, and relinked "First Lesson" to the first stage ("Stage 0") of our new lesson node.

In all of our sample applications, you will see that the root lesson node (i.e. the node that is directly linked to the "Event Begin Play") is called "Operation", but this is really only our convention, so you can name it however you'd like, "Application", for example:

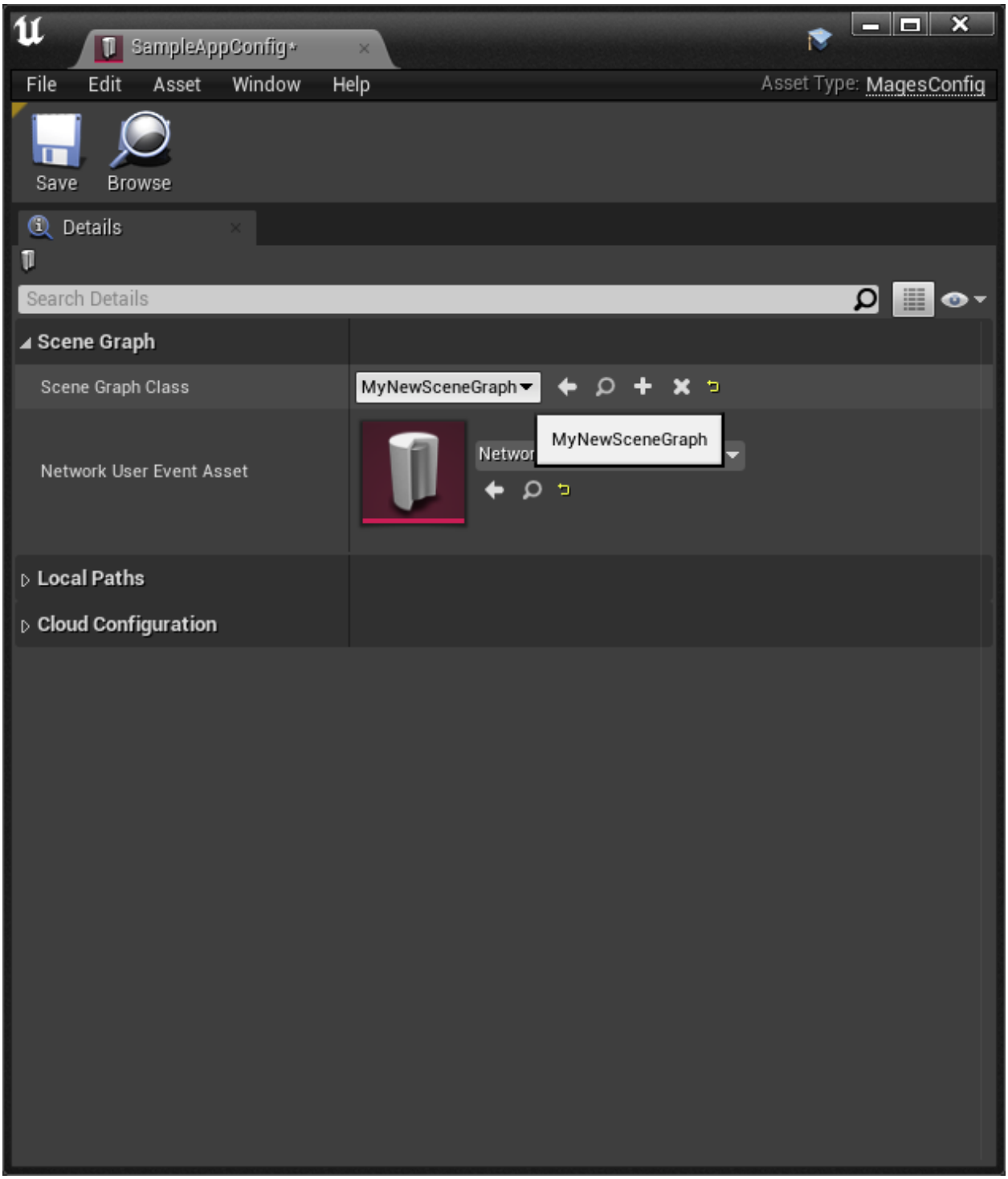


The final step is to set this scene graph in the configuration asset:

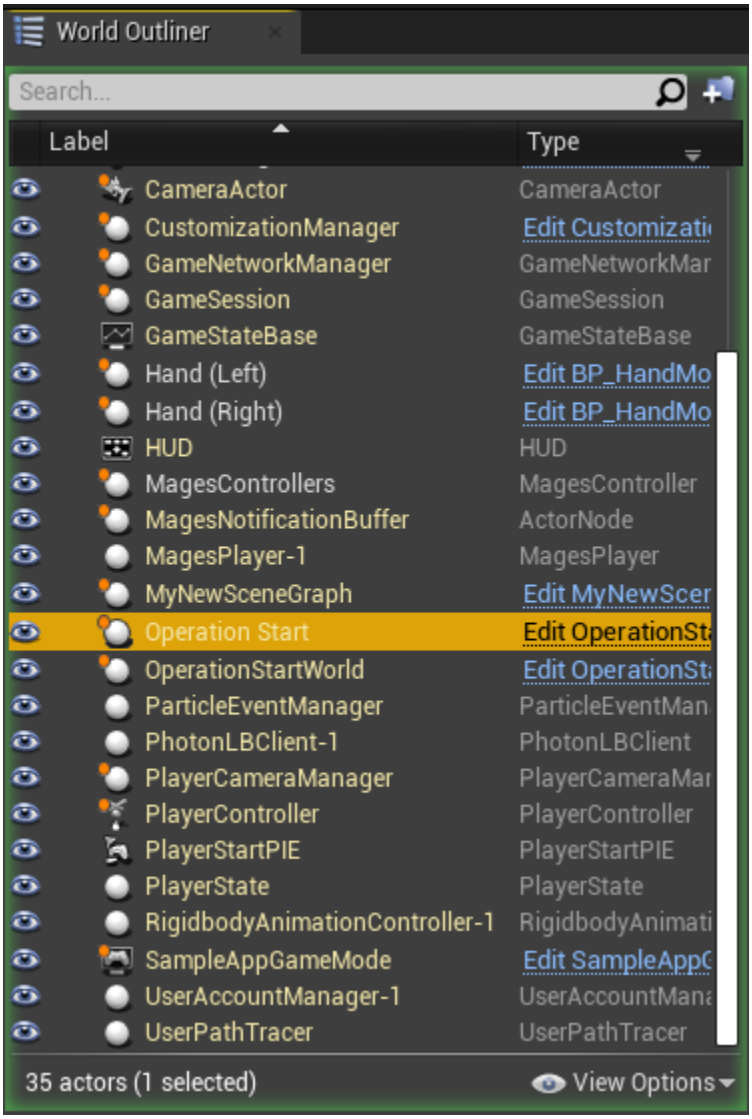
11.2.5 Setting the Scene Graph in the Configuration Asset

Navigate to the Config asset used in the "Scene Graph" actor and open the asset.

Set the Scene Graph Class property to the Scene Graph you created



That's it! Press play inside the editor and the first action (as well as the your Scene Graph) will be visible inside the World Outliner:



You can read more about the Scene Graph *here*. It covers the basics of the Scene Graph as well as more advanced topics like logic and alternative paths.

11.3 Action Analytics

In this tutorial we will demonstrate the different types of our analytics errors/warnings and how to properly configure them for your Actions.

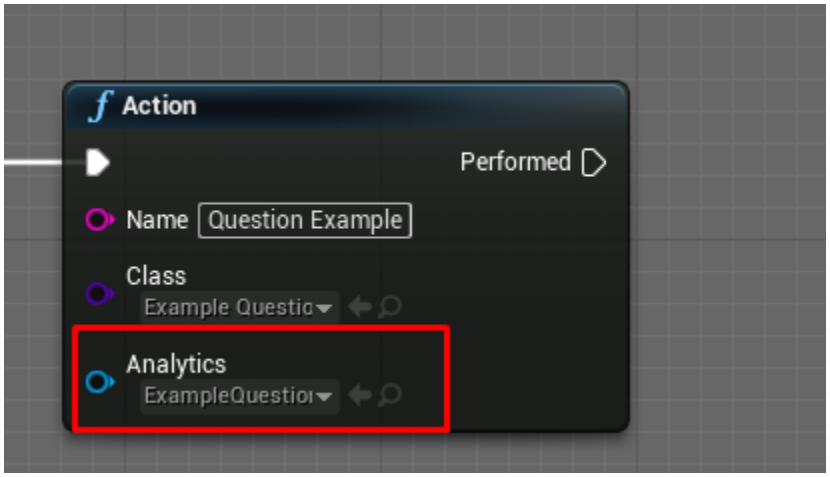
In MAGES SDK we provide a number of standard **scoring factors** to enhance your Analytics.

Note: **Scoring factors** refer to specific actionable behaviours which are important to consider for each Action and they reflect the user scoring e.g. wrong collision, action time, max movement velocity etc.

11.3.1 How to add scoring factors to your Action

Right click on Unreal's content browser, on the folder you want to save your analytics asset. From the MAGES submenu select **Analytics Asset** in order to create an analytics configuration.

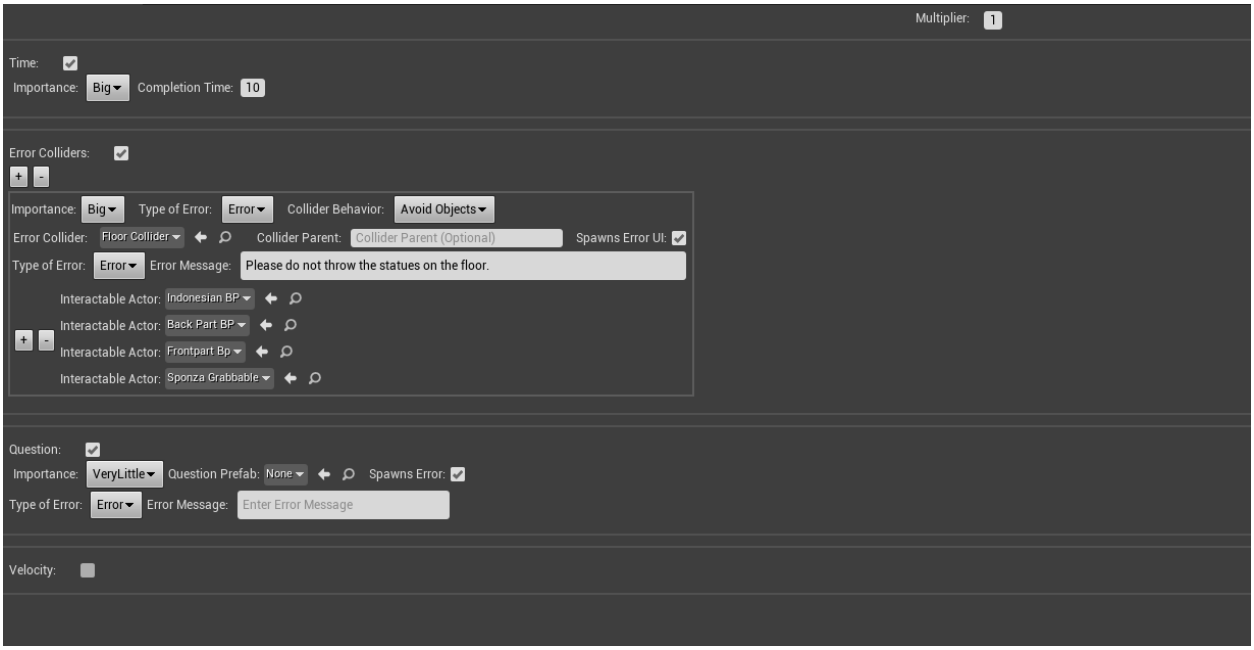
Each Action has each own **Analytics** configuration. In order to specify which action this asset is referring to, you need to reference it from respective action node in the Scenegraph Blueprint.



This window contains the scoring factors. To enable a new scoring factor click the corresponding checkbox. To save your changes click the **Save** button.

11.3.2 The Analytics Editor

This is an example of the analytics editor.



Note: The scoring system is calculated in 100. The maximum score of an Action is 100 and the minimum 0.

Multiplier: At the top of the window you can see a multiplier value. This can be used in case you want to multiply with a given number the score of the current Action, making it count as multiple actions.

Importance: This value identifies the weight of each scoring factor.

1. **VeryLittle:** 15%
2. **Little:** 30%
3. **Neutral:** 50%
4. **Big:** 80%
5. **VeryBig:** 100%

If our Action has only a **Little** scoring factor then its maximum score will be 30/100.

If we configure a **Neutral** and a **VeryLittle** scoring factor within the same Action, the maximum score of the Action will be 65/100.

Note: The score is capped at 100.

If our scoring factors overpass 100 e.g. three **Neutral** scoring factors it will be capped at 100, allowing the user to have 50 "bonus" points.

Error Type: We support three different types of errors with different popup UIs for each case:

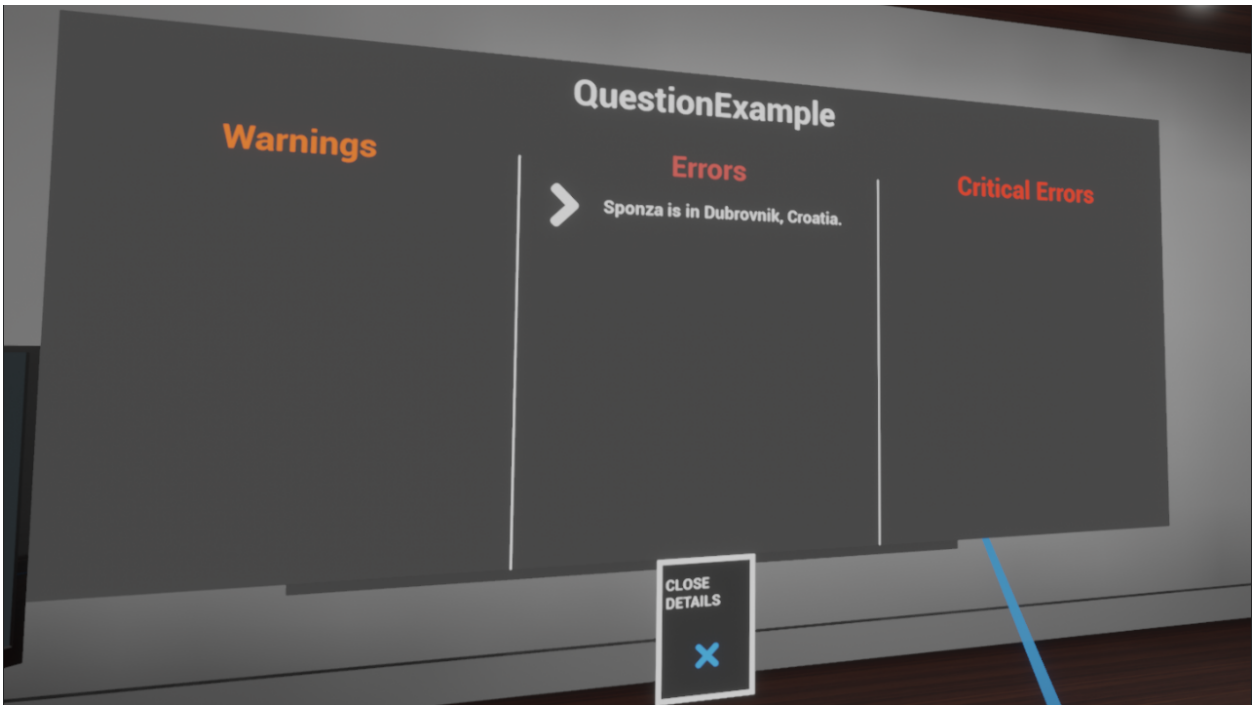
1. **Warning**
2. **Error**
3. **CriticalError**

Error Message: In the error message input field you can type the message that will be shown to the user, in case the user performs this error.

Show UI: Boolean value to toggle the error message. In case of false, the error will be logged but not shown to the user.

Example of an error message:

Be aware that errors, warnings and critical errors are also presented in the analytics overview at the end of the operation:



Below we present the standard scoring factors with examples on how to use them to rate your users.

Scoring factors:

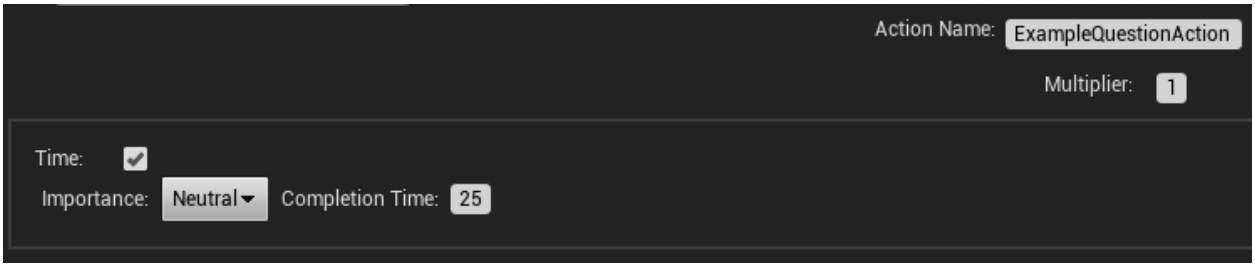
1. *Time*
2. *Error Colliders*
3. *Stay Error Colliders*
4. *Hit Perform Colliders*
5. *Question*
6. *Velocity*

11.3.3 Time

Usage: Give points according to the completion time. To achieve the highest score user needs to complete the Action in less seconds than the **Completion Time**. Passing this time-limit results in points loss (10 points per second).

Example: In this example we give user 25 seconds to complete the Action,

Here is the analytics editor for this Action:



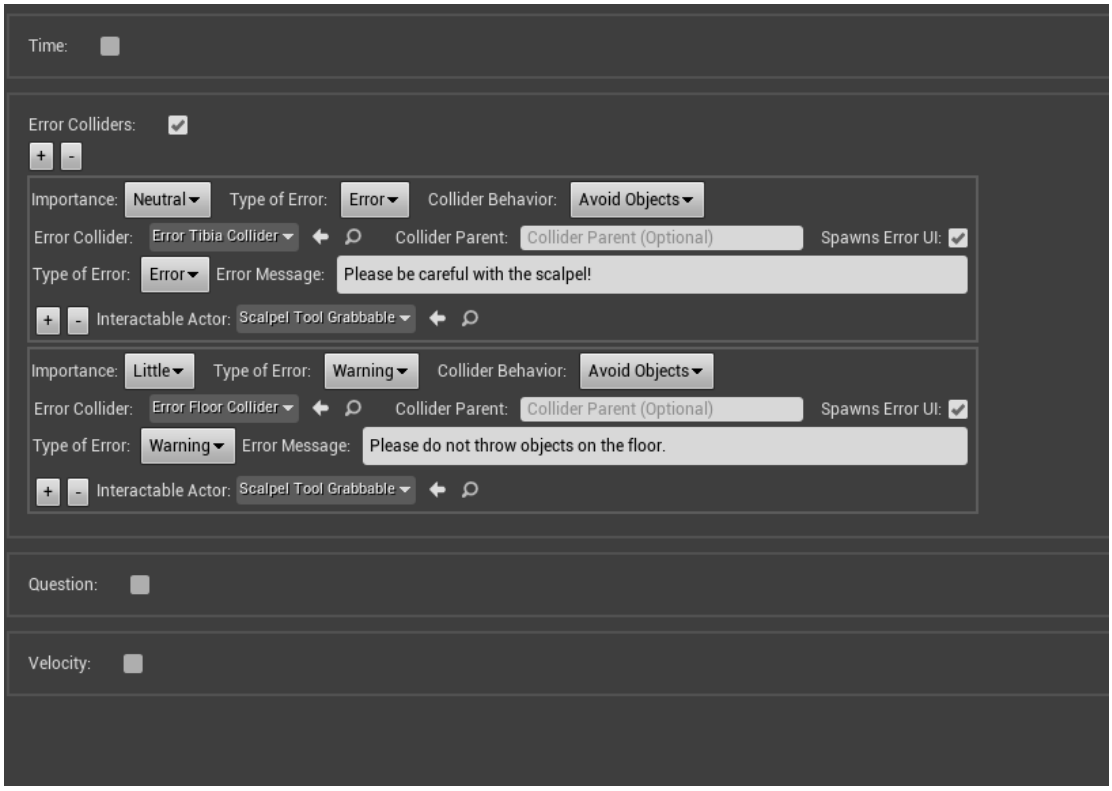
- We set the **Completion Time** to 25 seconds.
- We also set the **Importance** to **Neutral** meaning that this scoring factor will give 50/100 points to the user. If this is the only scoring factor the user can achieve a highest score of 50/100.

11.3.4 Error Colliders

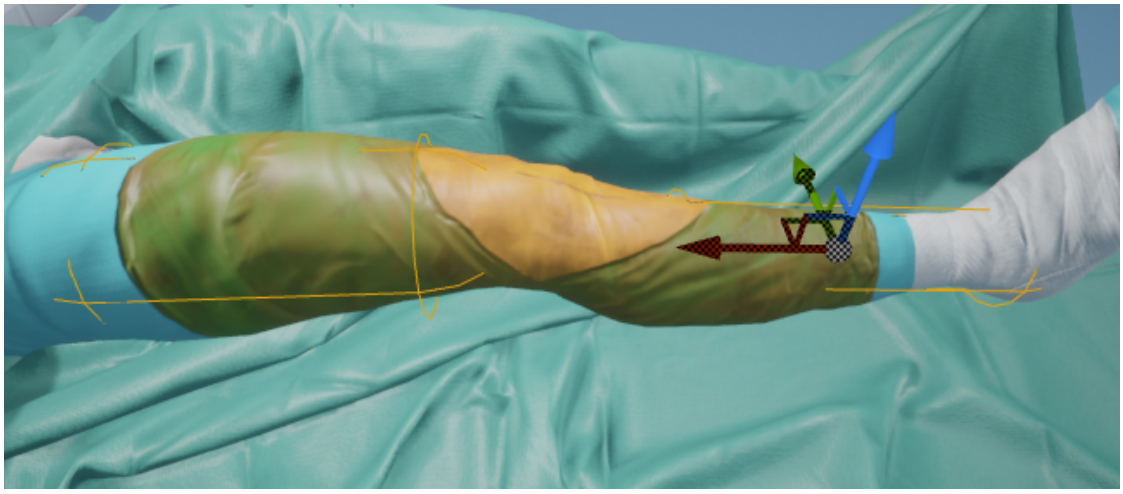
This scoring factor refers to the usage of overlapping colliders in order to define invalid events the user can perform in the simulation. The collider behaviour field defines when an error should be triggered. The available options are: 1. Avoid Objects 2. Stay in Collider 3. Must hit objects

Avoid Objects

> **Behaviour Usage:** Track if an object is in contact with a collider.
>
> **Example:** In this example we set two error colliders to track if the user drops a tool on the floor, and the other one if the scalpel cuts the skin in wrong position.
>
> Below you can see the analytics editor for this Action:

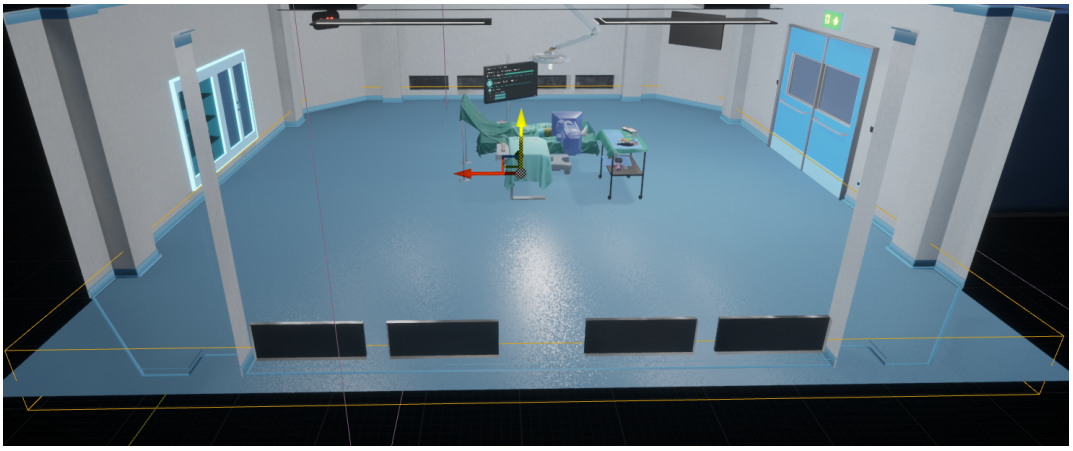


1. The first error collider actor contains two error trigger components: one for the femur and one for the tibia. You can see them below:



- We select the actor containing the two error triggers in the ErrorColliderActor input field. This reference will spawn the actor
- We select the ScalpelToolGrabbable from the corresponding **Interactable Actors** fields
- We set the **Error Type** as an **Error**
- We type the **Error Message** to our custom message
- **ShowUI** is enabled in order to show the error UI to the user

2. The second error collider actor is an error trigger for the floor.



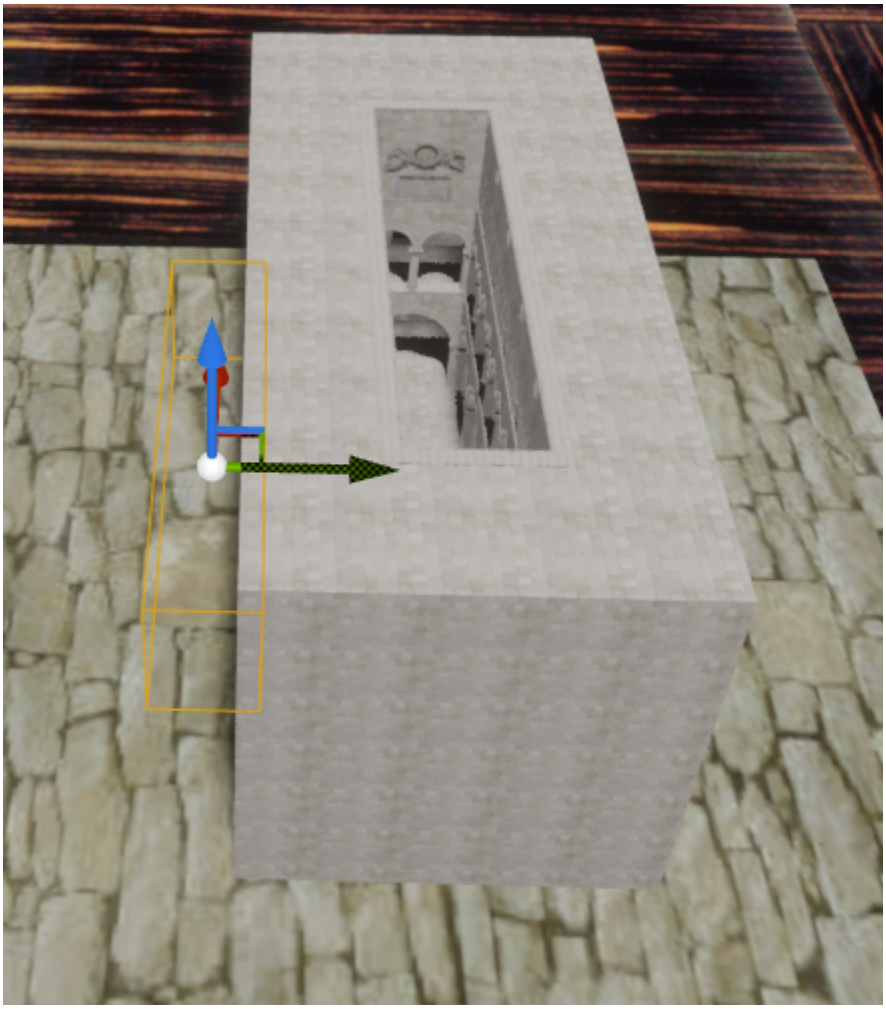
- We select the actor containing the floor error trigger in the ErrorColliderActor input field. This reference will spawn the actor
- We set the **Error Type** as a **Warning**
- We type in the **Error Message** our custom message
- **ShowUI** is enabled
- We add all the available items from the corresponding **Interactable Actors** fields

Stay Error Colliders

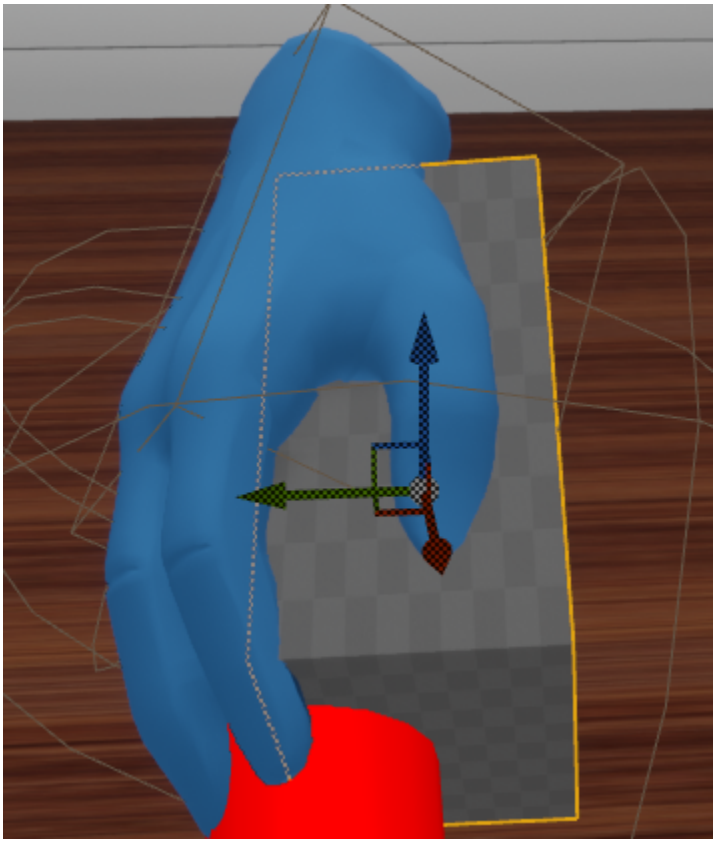
Usage: Track if an object is **not** in contact with a collider.

Example: In this example we set an error collider to track if the user holds the sponza while cleaning it with the cloth. If the hand exits the trigger box, the user will lose points.

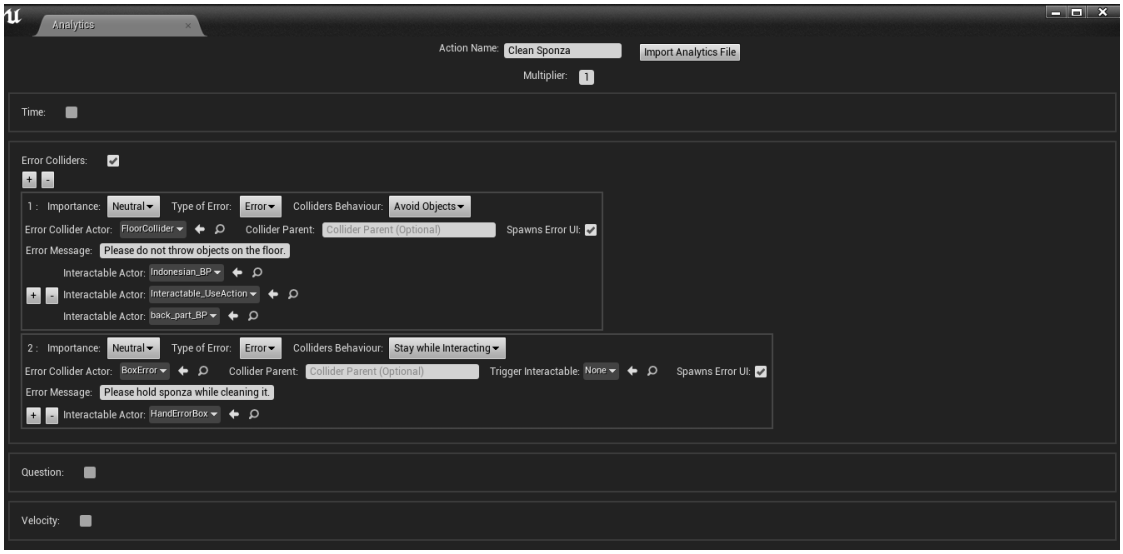
You can see the error box here on sponza:



In addition a non-visible static mesh actor is spawned as child of the user hands. This is actor is spawned through the action blueprint and is not automatically spawned from the analytics editor.



Here is the analytics editor for this action:



- We select the colliders behaviour to **Stay while Interacting**
- We select the collider representing the area that the user needs to place his hand on top of sponza into **Error Collider Actor**. This will spawn the safe area collider.
- We select the Cloth (Interactable_UseAction) from the **Trigger Interactable** dropdown
- We set the **Error Type** as an **Error**
- We set the **Importance** to **Neutral**. This factor is valued 50/100.

- We type in the **Error Message** our custom message
- **ShowUI** is enabled

Hit Perform Colliders

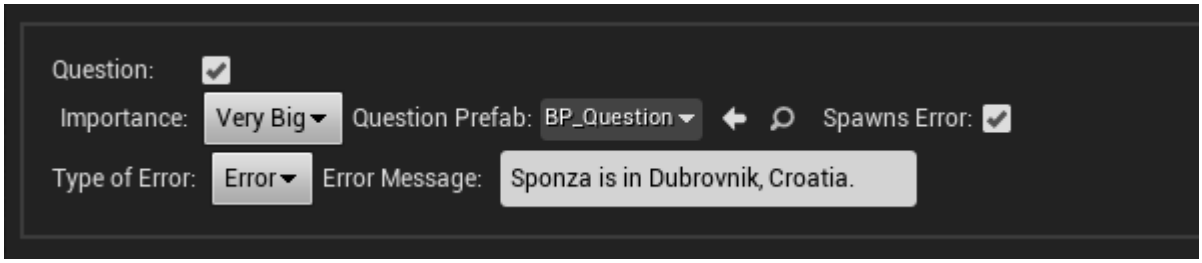
> **Note:** Currently under development.

11.3.5 Question

Usage: Lose points when answering a question incorrectly.

Example: In this example we will set the scoring factor to affect 100% of the score. Meaning that if the user answers wrong it will get zero points.

Below you can see the analytics editor for this Action:



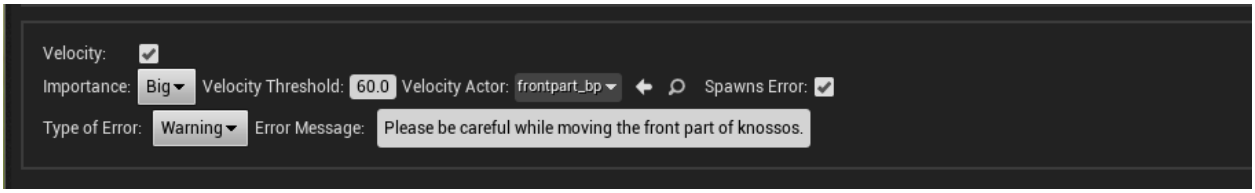
- We set the **Importance** to **VeryBig**, in this way a correct answer will give 100 points and a wrong answer 0.
- We select the question blueprint in the corresponding object field next to the Importance.
- Since, **Spawns Error** is enabled we need to set the type of error and the message that will be shown to the user.
- We type the **Error Message**.
- We set the **Type of Error** to **Error** from the dropdown field.

11.3.6 Velocity

Usage: Track the velocity of an object. Lose points when the velocity overpasses the configured value.

Example: In this example we will set a velocity scoring factor on a fragile object since we don't want the user to grab it and move it very fast.

Below you can see the analytics editor for this Action:



- We set the **Importance** to **Big**, in this way the Action will get a perfect score of 80/100
- We add the **Velocity Interactable Actor** that will be observed in the **Velocity Actor** field
- We set the **Velocity Threshold** value to **60**. If the velocity of the object overpasses 60 the user will lose 80 points.

  - A general guideline for velocity thresholds is:

    30: The user must move his hand extremely slowly. 40: The user must move his hand slower than the average speed. 60: The user must not do rapid movements with his hand while holding this object.

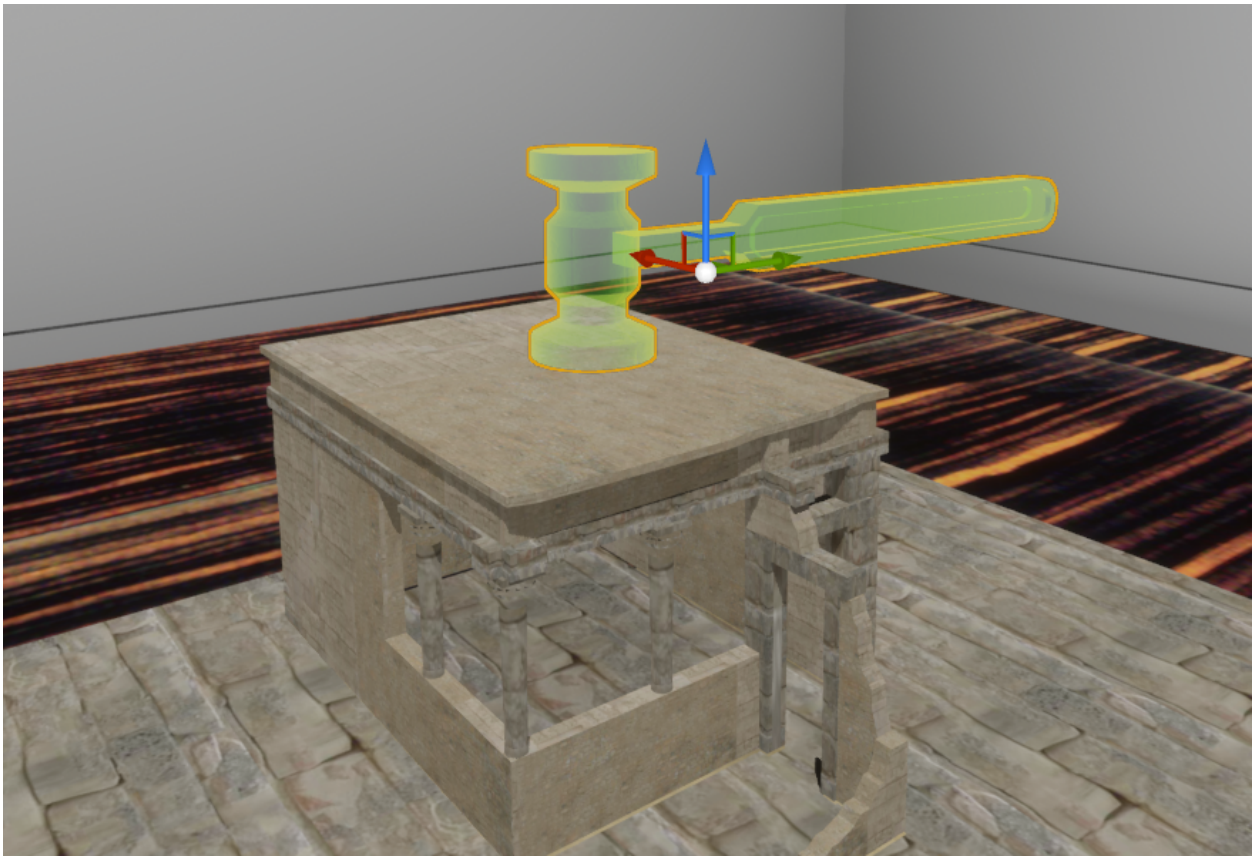
- Since, **Spawns Error** is enabled we need to set the type of error and the message that will be shown to the user.
- We type the **Error Message**.
- We set the **Type of Error** to **Error** from the dropdown field.

11.3.7 Custom Scoring Factor

Usage: Configure your own custom scoring factors if you need something special, not provided out-of-the-box.

Example: In this example, we set a custom scoring factor to measure the **velocity on impact** when hammering the knossos building. User needs to take the mallet and hit the Knossos building three times. However, Knossos is fragile so we set a maximum velocity to track the hits.

Here is the knossos building with the mallet hologram:

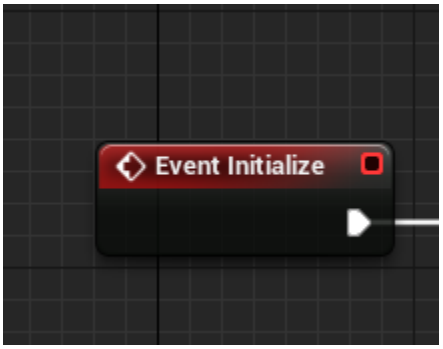


Note: For custom scoring factors we don't need the analytic editor. We will implement the behaviour using Blueprints

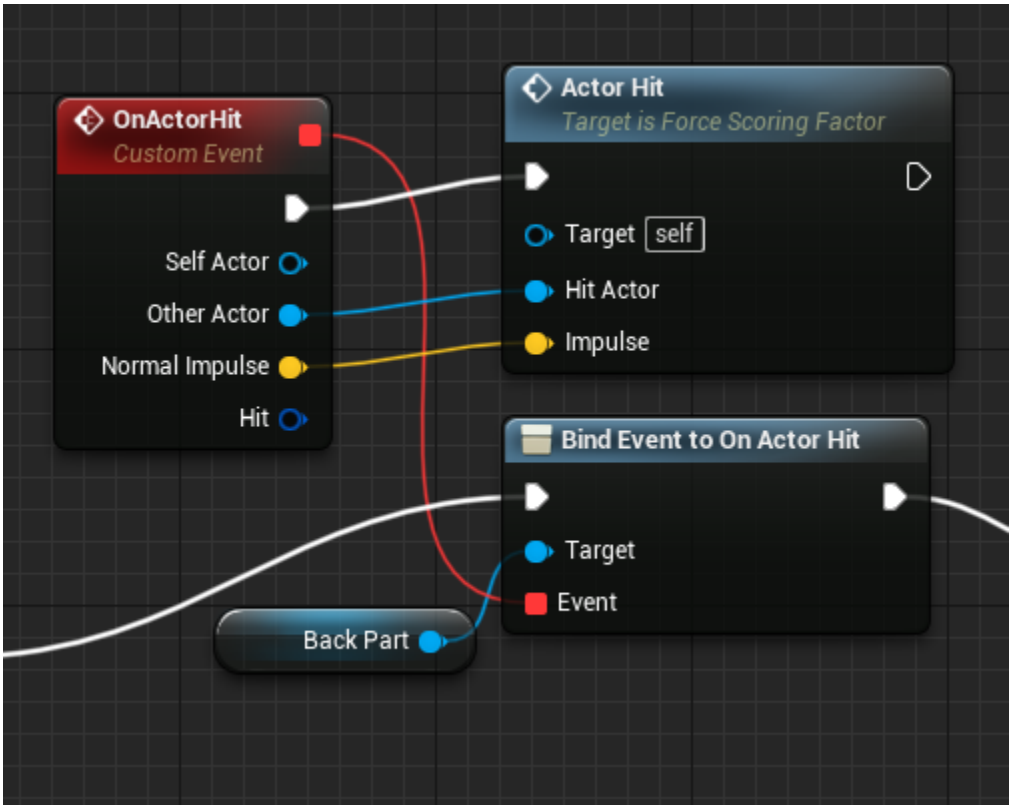
First we create a new blueprint class that inherits from `ScoringFactor`

The **ForceScoringFactor** blueprint implements our example custom scoring factor.

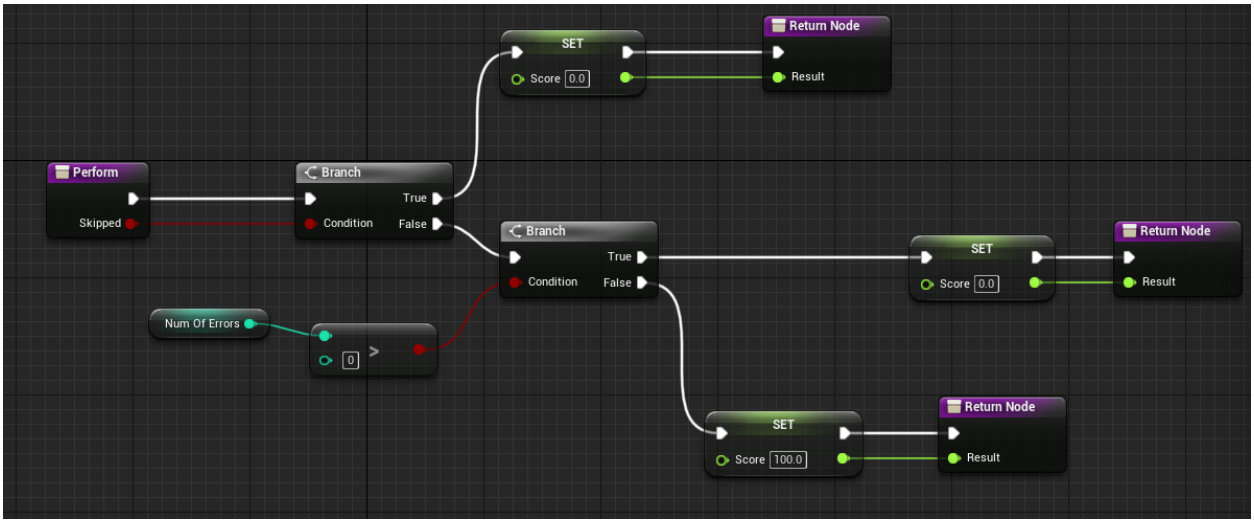
The `ScoringFactor` class contains virtual functions and events for you to override in your custom scoring factors.



The Initialize event is called to setup your custom scoring factor. Everything you need to spawn or configure you should implement it in this function.

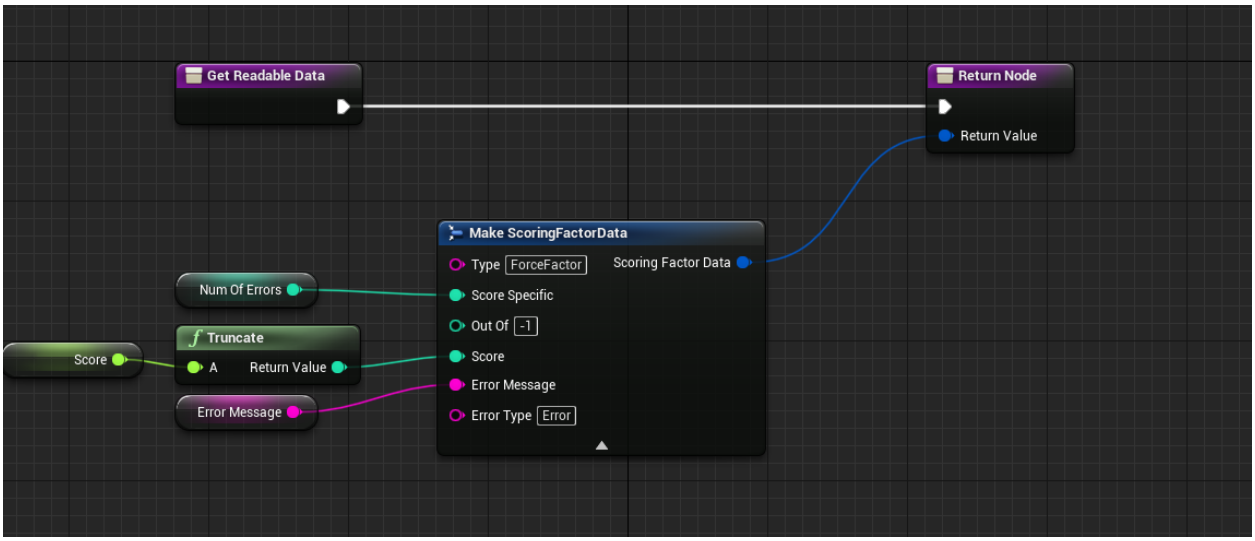


In this case we add an event on the Actor Hit listener of the back part actor. On each hit the Actor Hit function is called which determines if the applied force results in an error. In the same way you can implement your own logic to gather information about the user's performance.



The Perform function of the custom scoring factor needs to be overridden. It is called along with the Action's Perform(). The purpose of this method is to calculate and return the score of the user. In this example, it calculates the score with data retrieved from the Actor Hit Function.

Make sure the score in the range [0,100]



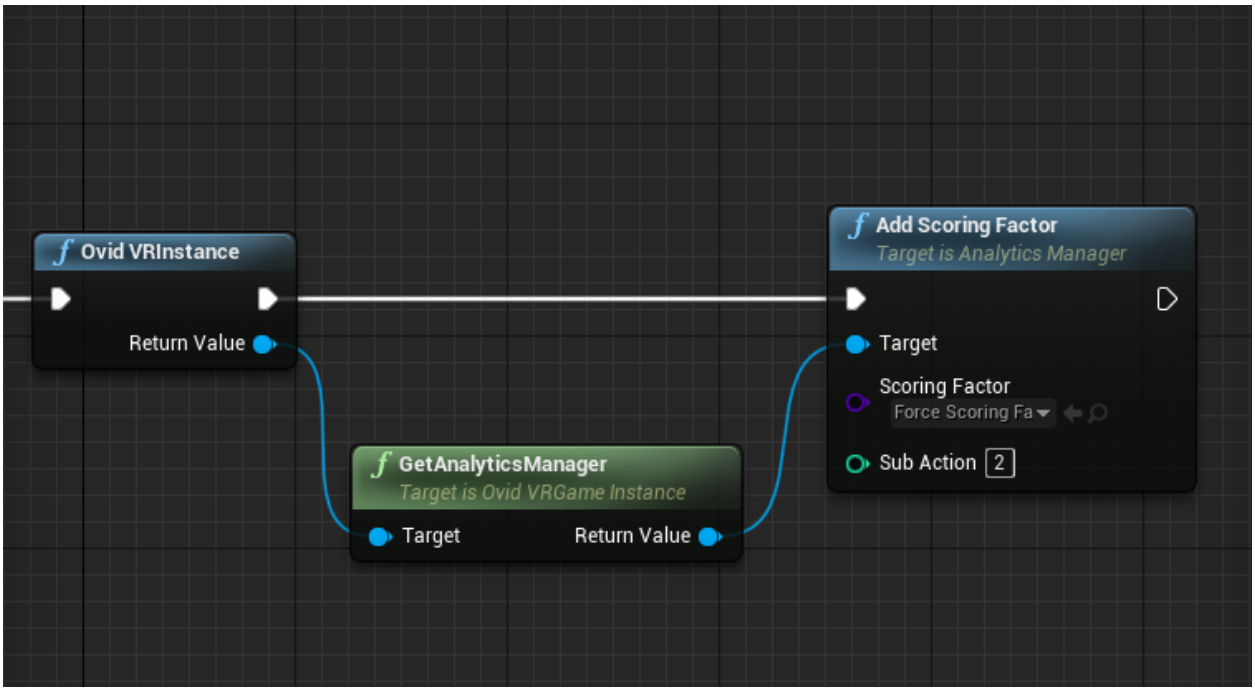
GetReadableData manages the data from the custom scoring factor that will be saved at the end of the Action in human-readable form.

A new ScoringFactorData struct needs to be created, which contains:

1. **Score:** The user's score
2. **Out Of:** In case the scoring factor contains a number of possible values (e.g maximum velocity, maximum time etc) this variable reflects this amount
3. **Type:** The scoring factor's description name
4. **Score Specific:** The specific metric of this scoring factor. E.g. In time it is the seconds the user needed to complete the action.
5. **Error Message:** The error message to spawn when triggering this error
6. **Error Type:** The type of error (warning, normal, critical error)

The final step is to link this scoring factor with our Actions script.

Below you can see the Blueprints that are responsible for adding a custom scoring factor in the current Action.



Warning: The `Add Custom Scoring Factor` Blueprint connects our custom scoring factor with this Action and the Analytics Manager. It needs to be called in the Initialization State of the current Action.

The `Sub Action` argument is used in Combined Actions to specify the sub-action which will be added. In all other type of actions this field should be 0.

This is the proper way to configure a custom scoring factor.

11.4 Actions with deformable skinned meshes

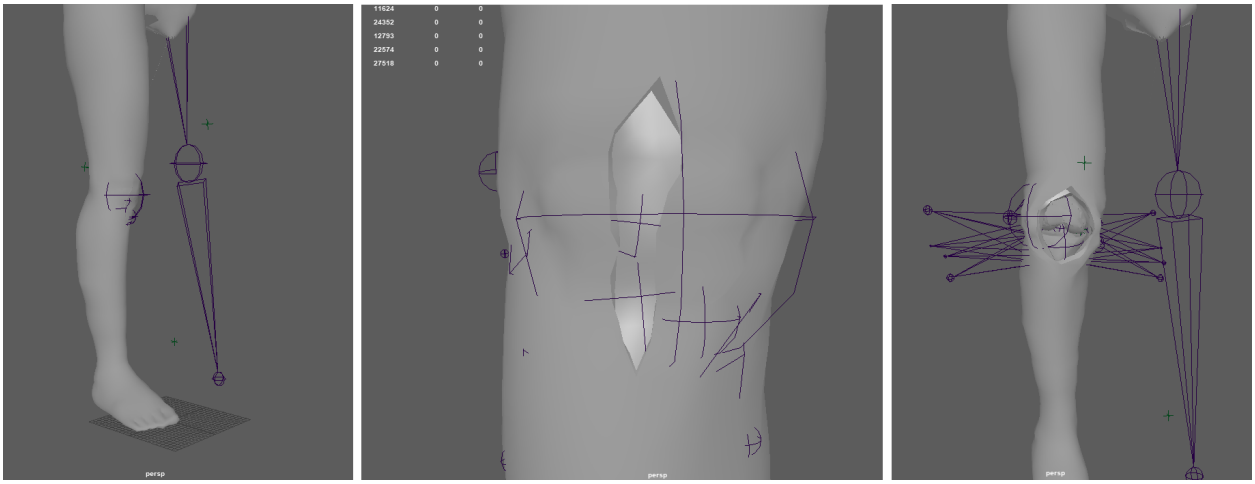
In this tutorial we will demonstrate the complete pipeline on creating an Action with a deformable skinned mesh. From the 3D model to the unity importing then all the configurations and finally the Action script. We will implement a medical example, in particular the initial incision from the Total Knee Arthroplasty operation.

Tutorial overview:

1. Generate the animated 3D model
2. Import the 3D model to Unreal
3. Split the animation into smaller animation assets
4. Animator setup
5. Configure the CharacterController
6. Generate the Action Actors
7. Implement the Action Blueprint

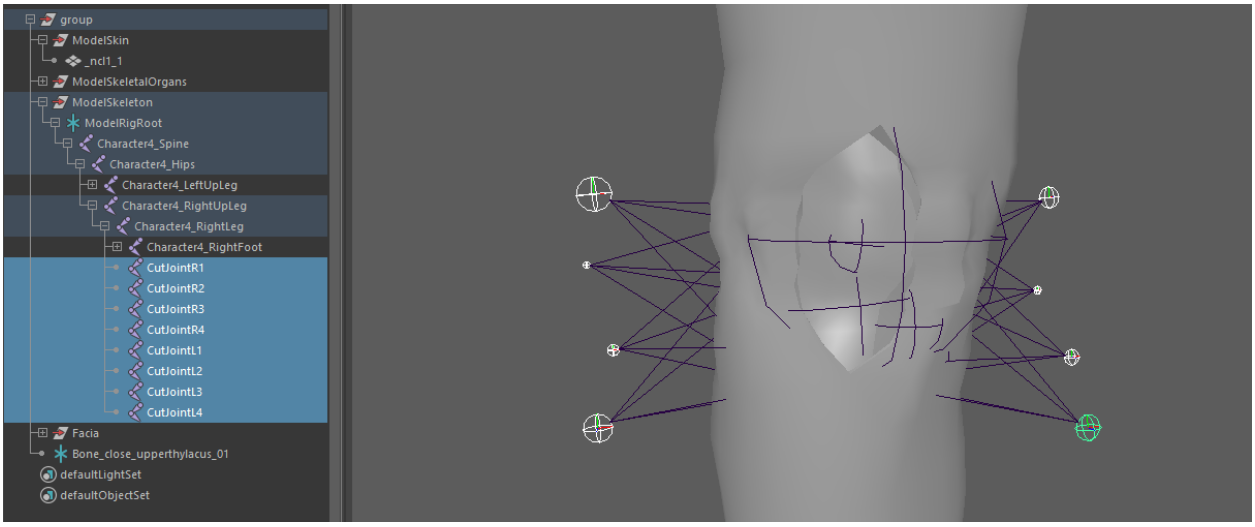
11.4.1 Generate the animated 3D model

In this scenario we will use an animated leg, with various baked skinned deformations. The image below shows our 3D model in Maya.



As you can see, in Maya we design the full animation using standard joints and key frames. The animation process depends on the scenario you want to design. In this case, we rigged the right foot adding joints to cut the skin and the different muscle layers until we have a clear view of the knee.

The image below shows the joints we used for the skin animation along with their keyframes.

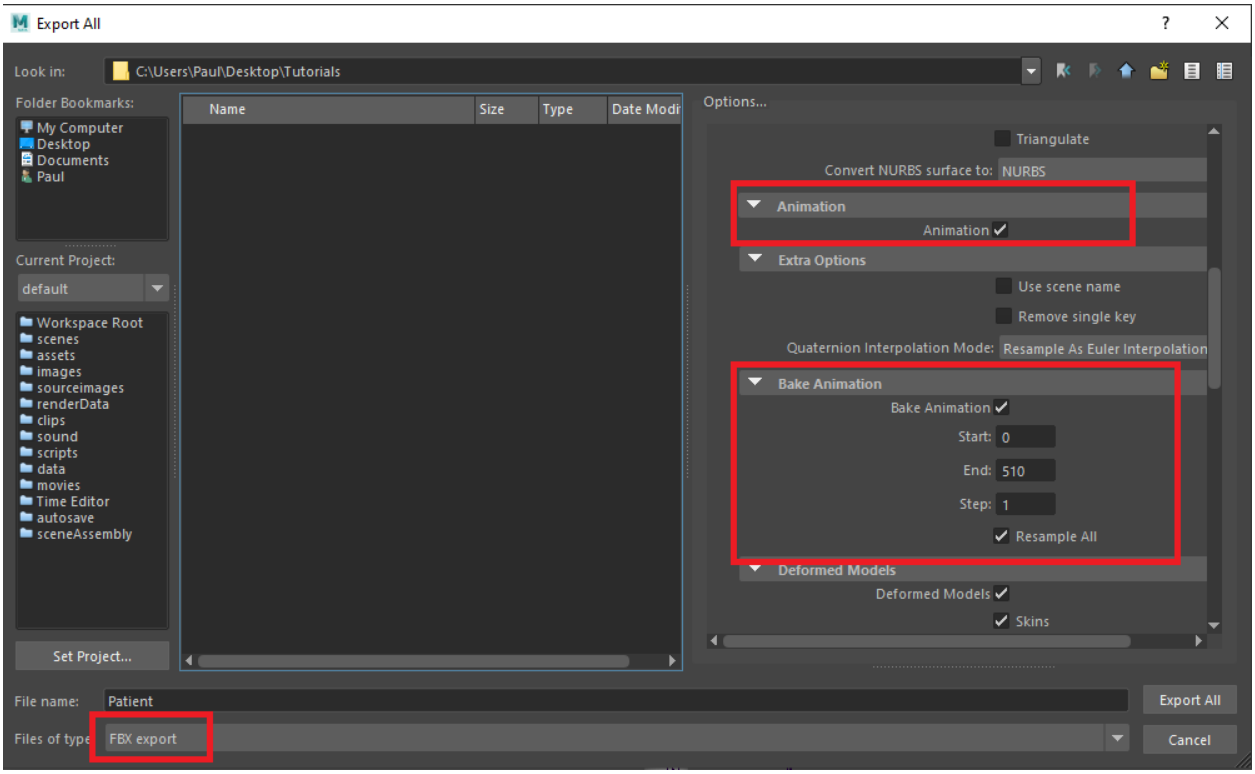


Now we have to export the 3D model. We use the FBX format to export our 3D models since it is the most reliable format to work with unreal.

From the Maya menu bar navigate to **File/Export all**. Now you need to configure some options first. Make sure the Animation option is checked to export the animation as well. Make sure the **Files of type** option is set to FBX export.

There are specific situations where you need to bake the animation instead of exporting it without the bake option. If the animation has complex animation behaviours or blends there is a chance that they will not import right in Unreal. If you face some issues on the importing and the animation does not appear to work right check the Bake Animation option at the export window.

Warning: If you bake the animation, Maya will generate key frames for all joints. After that, the modifying the animation would be really inconvenient. Remember to keep the Maya binary file to modify the animation when baking.



11.4.2 Import the 3D model to Unreal

To import the model into Unreal just drag and drop the FBX to your project. Depending on the import settings you choose different .uasset files will be created. These can be:

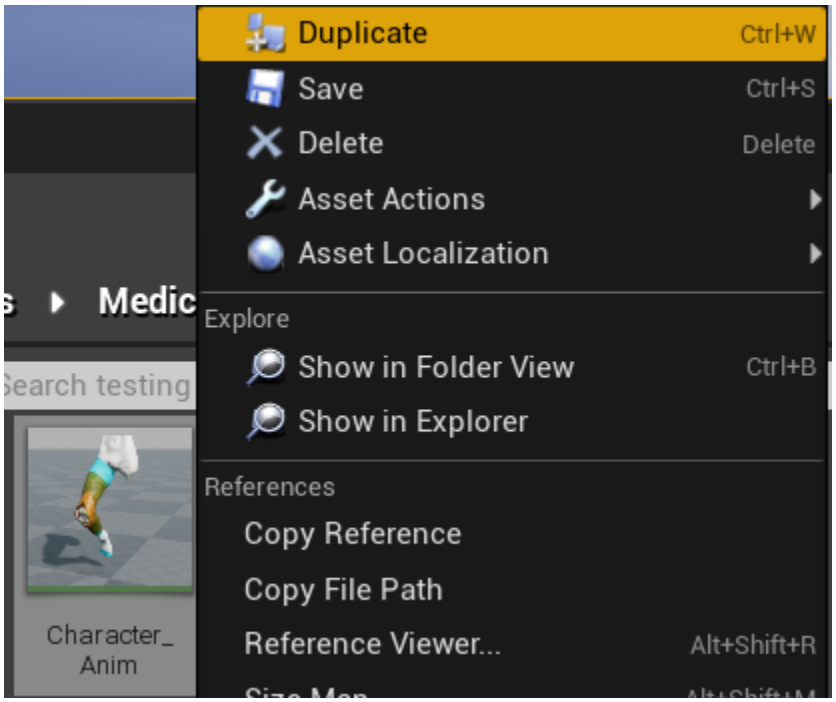
1. Skeleton
2. Mesh
3. Animations
4. Materials

This is the final result

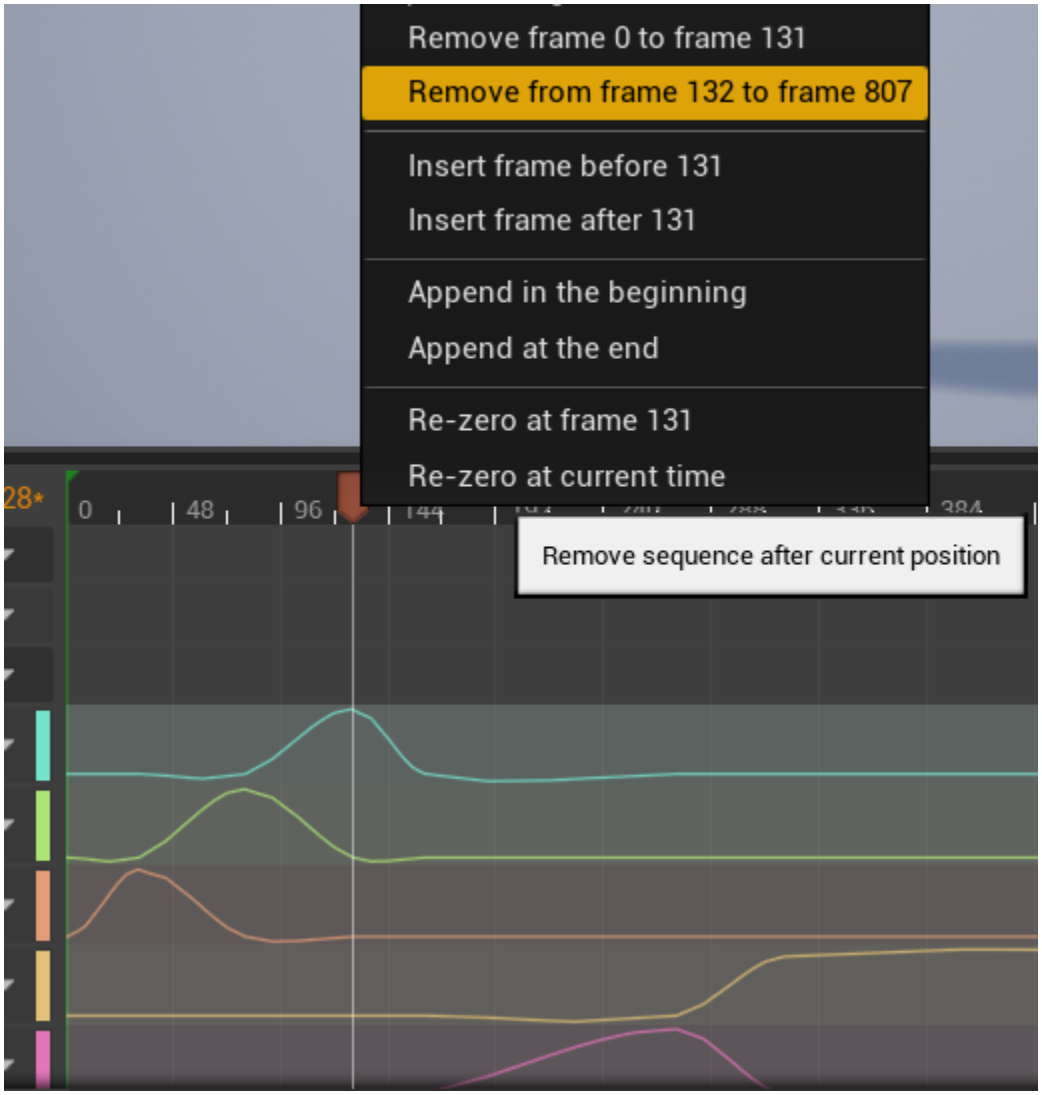
11.4.3 Split the animation into clips

We will split the animation into smaller clips to feed our CharacterController, a class that controls our patient's animations.

To split the animation into smaller clips, select the imported animation and right click it, then select duplicate.



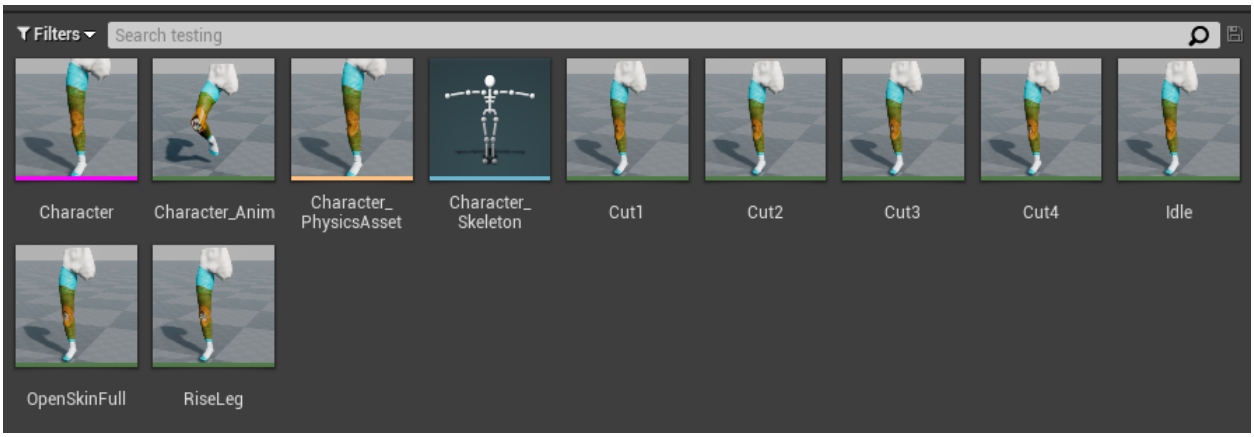
Open the duplicated animation and then move the key frame cursor (the red bar) to the last frame of this animation segment. Right click and select remove key frames from :code:'XX frame' to :code:'XX frame' (the last frame of the whole animation). Remove in a similar way the starting redundant frames.



Finally, save your changes and rename the animation asset.

Repeat this process for all your animation segments.

Below you can see the clips we made for this operation. In this tutorial we will use Cut1, Cut2, Cut3, Cut4.



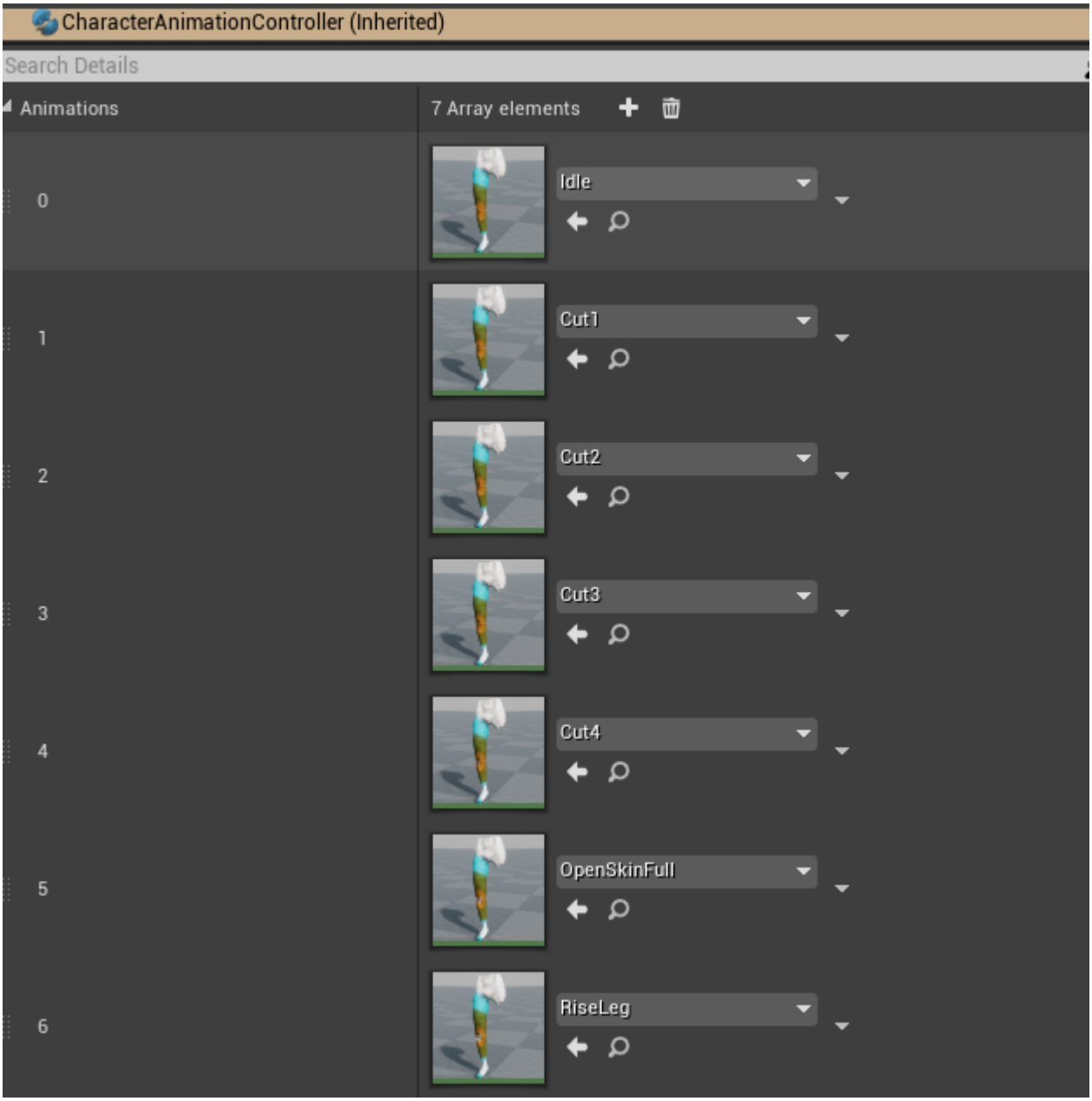
11.4.4 Character Controller Setup

To properly execute the animations and manage their transitions we have to setup an Animator for our model.

From the Unreal scene, navigate to your Actor (in this case the leg), then select Add Component and add a CharacterAnimationController. Here you need to configure your own Animation to match your needs.

The next step is to add the animations in our CharacterAnimationController. In the `Animations` array of the CharacterAnimationController component add all the animation assets you created in the previous step.

Below you can see our CharacterAnimationController.



11.4.5 Generate the Action actors

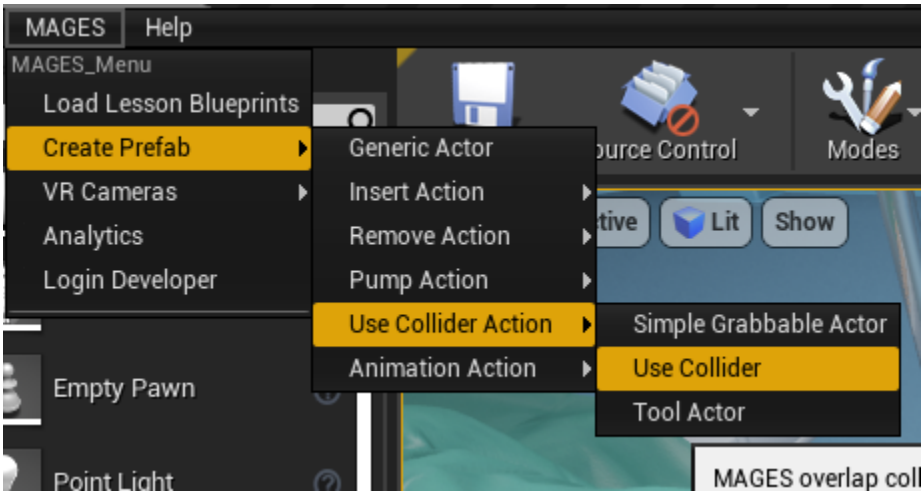
As we mentioned, the MAGES metaphor of the skin incision will be a UseAction. We use the UseAction in cases where we need to grab a certain interactable object (pliers, scalpel etc) to complete an objective by "touching" the tool on a predefined area.

For the UseAction we need to prepare the following prefabs

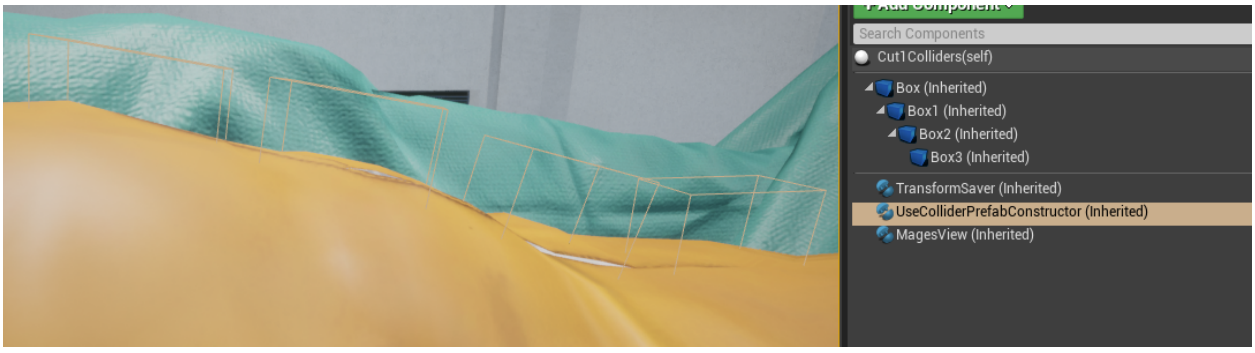
1. Use Colliders
2. The actual interactable (a scalpel)

Use Collider

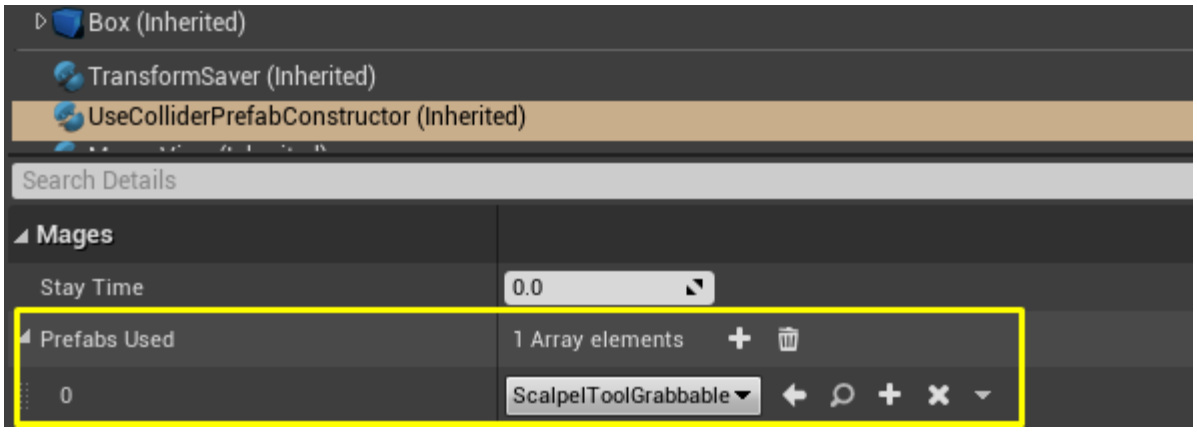
To generate the use collider navigate to **MAGES/Create Prefab/Use Collider** and the use collider actor template will appear in the world outliner.



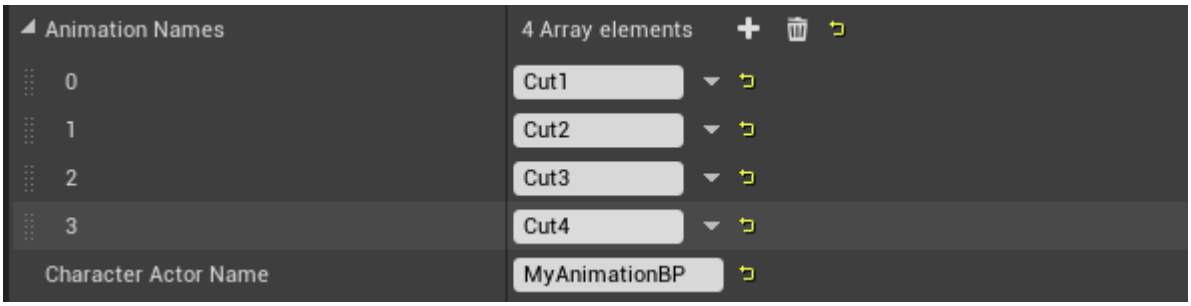
The use colliders are waiting for the use actor to trigger them. Once all of the colliders are triggered the Action will perform. In this scenario we will use four colliders since we have four cut animations. The image below shows our actor with the four colliders components.



Now we need to assign that only the scalpel will trigger these colliders. To do that, navigate to the UseColliderPrefab-Constructor add a new Actor element at the Prefabs Used List, then select the scalpel (image below)

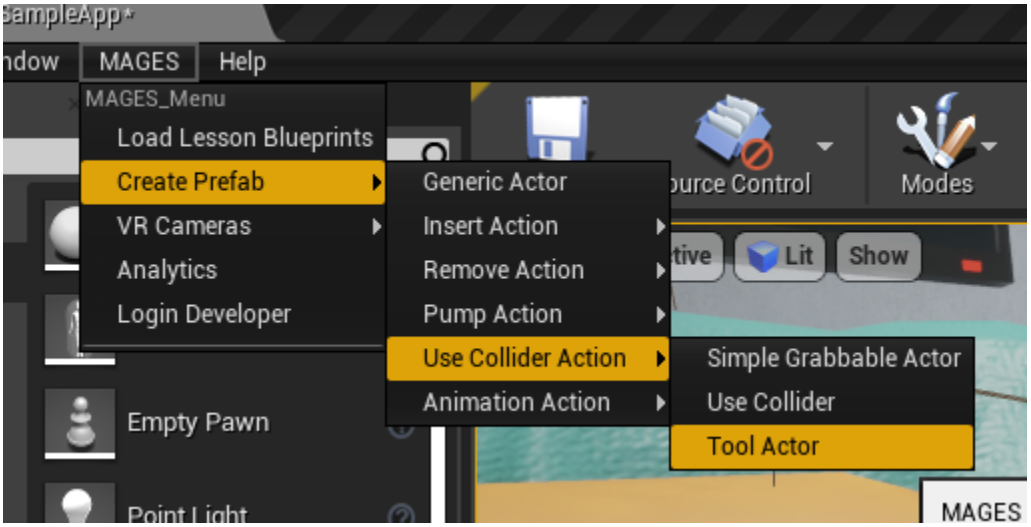


Finally, we need to add the animations that will be played with each box. These animations are added in the UseColliderPrefabConstructor component in the field, `Animation Names`.



Use Interactable Actor

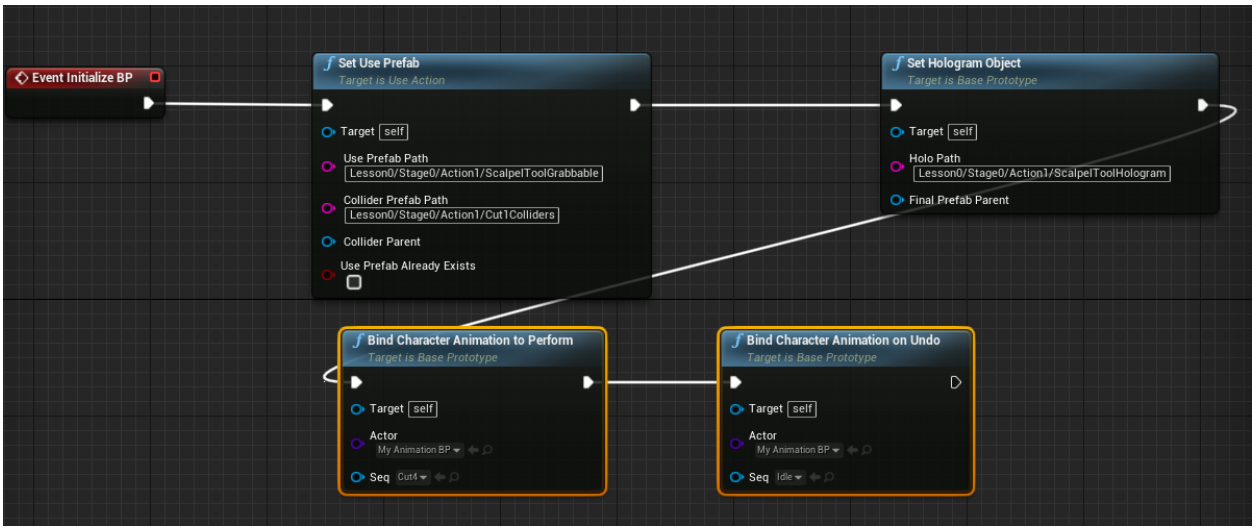
The next step is to configure the Interactable actor that we will use for this scenario, the scalpel. We need to create a Tool Actor. A tool actor is the same as an interactable actor with an extra component, GestureHands.



This component allows us to enhance its interacting behaviour, you can read more about it *here*.

11.4.6 Implement the Action Blueprint

The final step is to write the Action script. Below you can see the basic ToolAction script that describes this ToolAction.



Lets explain the Action a bit. You can see that we call the `SetToolActionPrefab` method to instantiate our tool colliders and set the actor that we will use to trigger them. In addition two functions are called, the `Bind Character Animation to Perform` and :code:`Bind Character Animation on Undo which take the actor that has the CharacterAnimationController component (the patient) and an animation sequence as arguments. The first function will add an animation to be played once the user performs this action, while the latter will play an animation on undo. These functions are useful in case the user skips or undoes the action.

Note: To perform the ToolAction, the user needs to hit all the registered tool colliders.

11.5 HandPoser

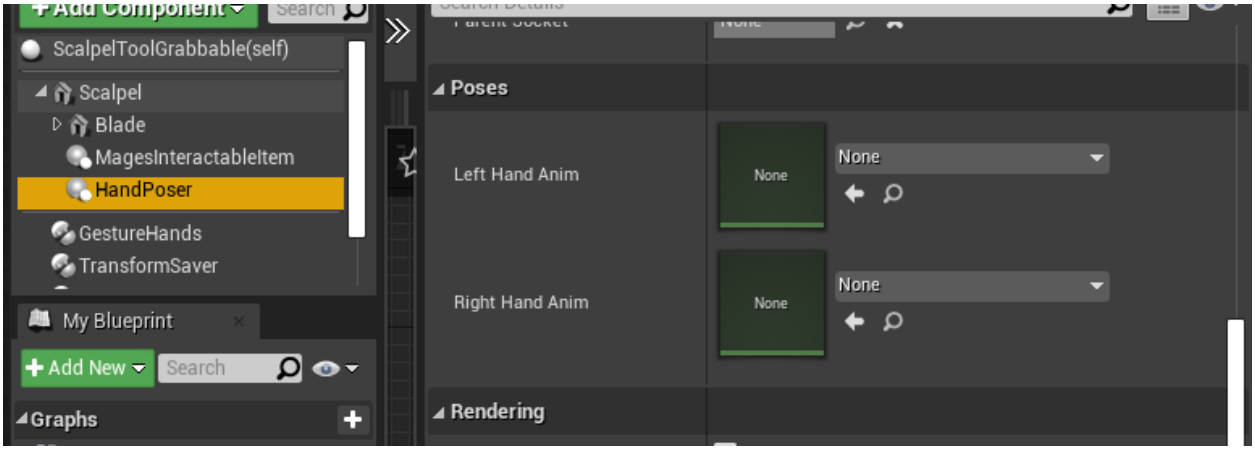
In this tutorial we will learn how to setup the Hand Poser for an interactable actor.

Hand Poser is a mechanic contained in the MAGESPhysics module which enables grabbing objects with a specific hand posture. With the Hand Poser tool, developers can configure predefined hand postures for each object.

It is used to interact with the physical object with an intuitive way. It is not mandatory to include it in your simulation but highly recommended for a realistic interaction system.

11.5.1 How to Configure Hand Poser

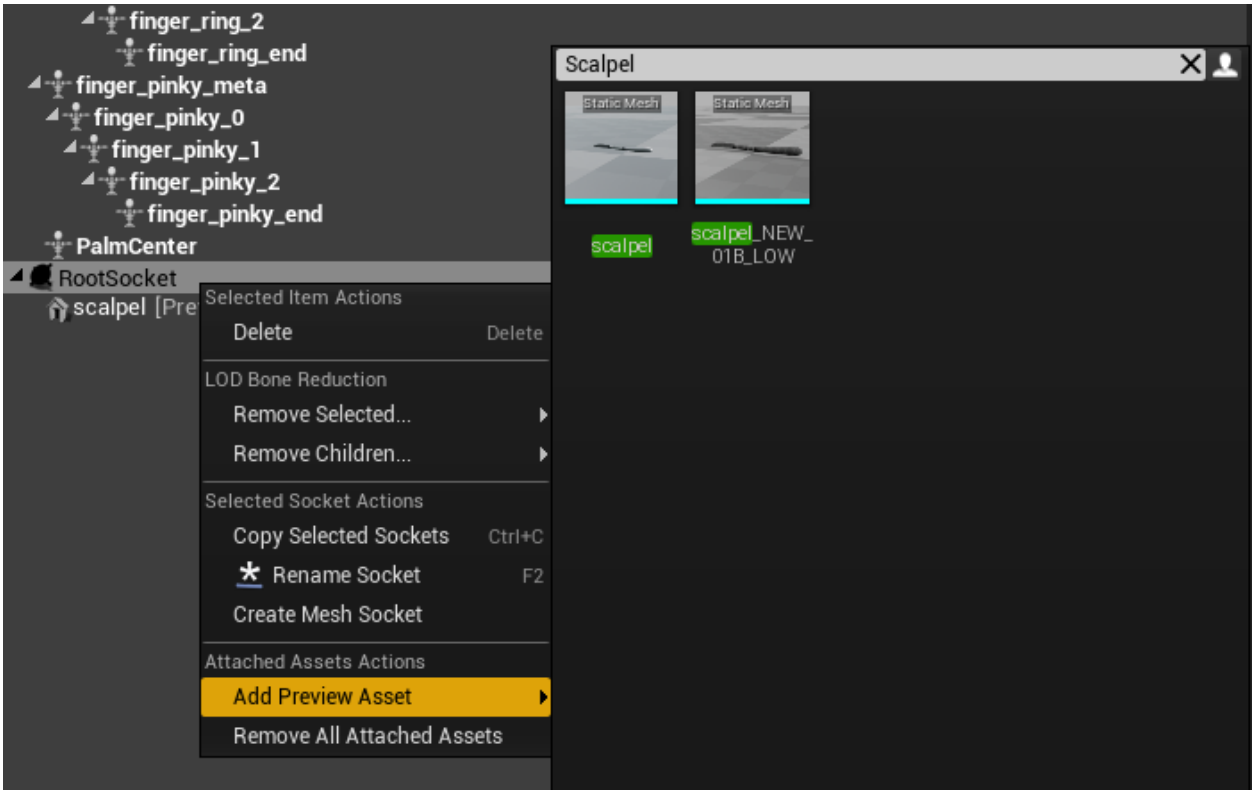
The first step is to Add the HandPoser component to your object. Make sure the component is added a child of the static mesh component you will interact with.



This component contains two input fields for Animation assets, one posture per hand. To generate postures for the left hand open the skeletal mesh **LeftHandMesh** which is in the folder **MAGES_SDK_Content/MAGES/SDKAddons/UserHands/Models/LeftHand**. Similarly, the **RightHandMesh** can be found under the folder **MAGES_SDK_Content/MAGES/SDKAddons/UserHands/Models/RightHand**.

In the skeletal hierarchy of the hand skeleton you will notice a socket called **RootSocket** under the root bone.

In order to create a hand posture that will accurately fit on an object we need to be able to to preview that mesh while authoring our posture. For this reason we add a preview mesh on the **RootSocket**, in this case we choose the scalpel.



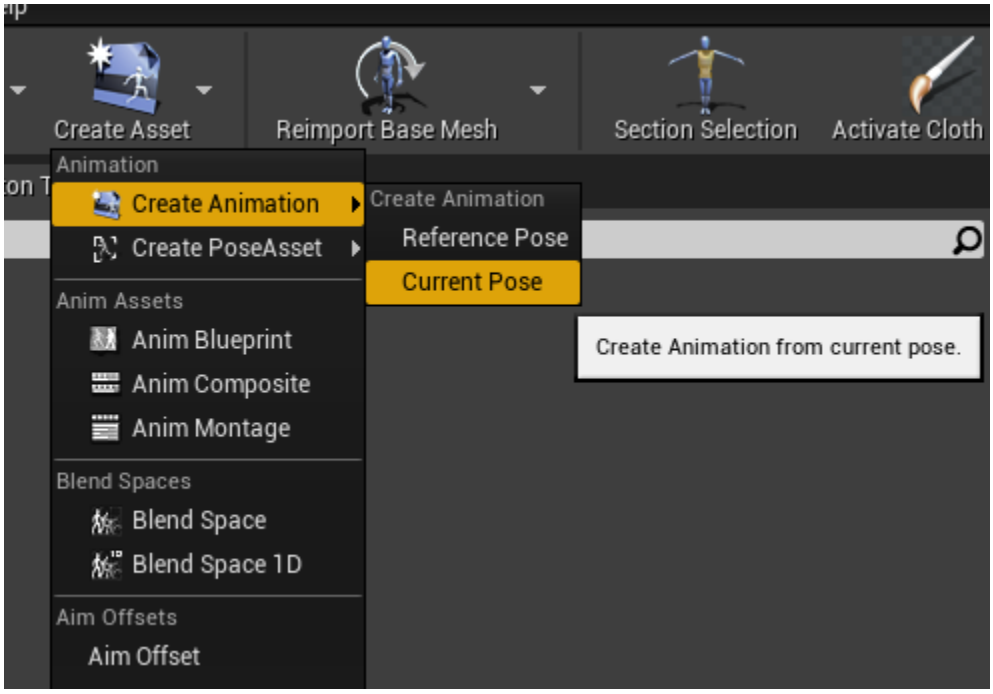
Warning: The scale of the static mesh should be (1,1,1) and its pivot should be as close the its center as possible, otherwise the poses will lead to unexpected behaviour.

Now, we need to translate and rotate the wrist bone and its children bones in order for the hand to accurately hold the scalpel. Below there is an example of the right hand posture for the scalpel.

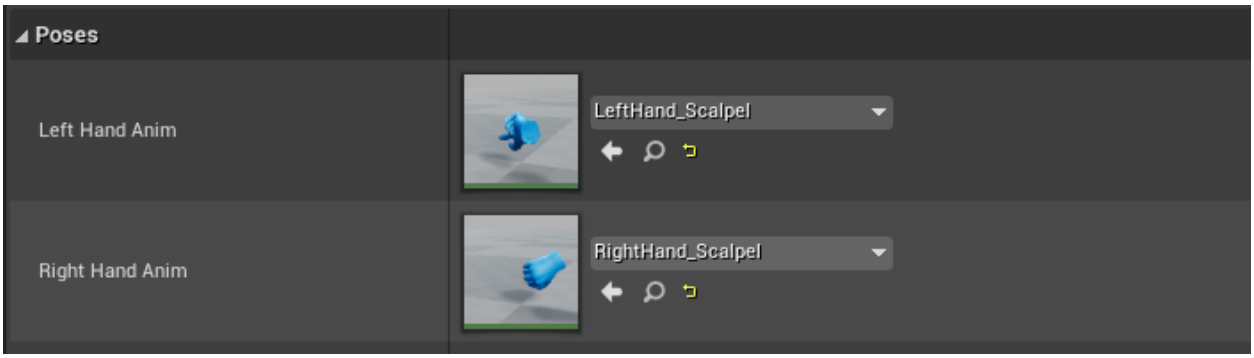
Note: The static mesh that we have added is only a preview, changing its transform, or the transform of the socket will not have any effects on our posture.

The next step is to save the postures you generated.

Once you are done authoring the transformation of the joints click on the Unreal Menu navigate to **Create Asset/Create Animation/Current Pose**, and create a new Animation Asset.



We need to link the postures to the HandPoser component. Simply drag and drop (or select them from the dropdown menu) the Animation Assets in the HandPoser component under the Poses section.



Finally, save the Blueprint Actor.

11.6 Unreal Integration with the Cloud

In this tutorial we will demonstrate the complete pipeline on integrating Unreal with the web services provided in MAGES 4.0.1

We will start of by creating a product for the Sample App through the Web Portal, and then create a user and assign him a valid product license.

After the *online* walkthrough is completed, we will proceed to integrate the Login service with the Sample App Unreal side. Additionally, we will provide a configuration for uploading the user analytics to the AnalyticsAPI.

The end goal of this tutorial is to make you **product-ready**, meaning that you are able to build your VR Module and authenticate/check out any user licenses you want. Further, you will be able to upload user analytics to the AnalyticsAPI and display them in the Web Portal.

Tutorial Overview:

1. *Prerequisites*
2. *Product & User Creation*
3. *MAGES™ SDK Sample App*

11.6.1 Prerequisites

- You have configured all three web services whether for local Development or in a Production setting as described in the Web Services Manual section and the separate service sections per se:
  - For the Login Service: *Getting Ready for Development* & *Getting Ready for Production*
  - For the AnalyticsAPI: *Getting Ready for Development* & *Getting Ready for Production*
  - For the Portal: *Getting Ready for Development* & *Getting Ready for Production*
- All three web services are up and running whether locally or in the Azure Cloud
- You have an active user with Admin role permissions
- (Optional) Azure Storage Account and Azure Blob storage must be in place for uploading user Analytics

Note: For brevity, we assume in this tutorial that you have set up and running a Development –localhost environment of the Web Services.

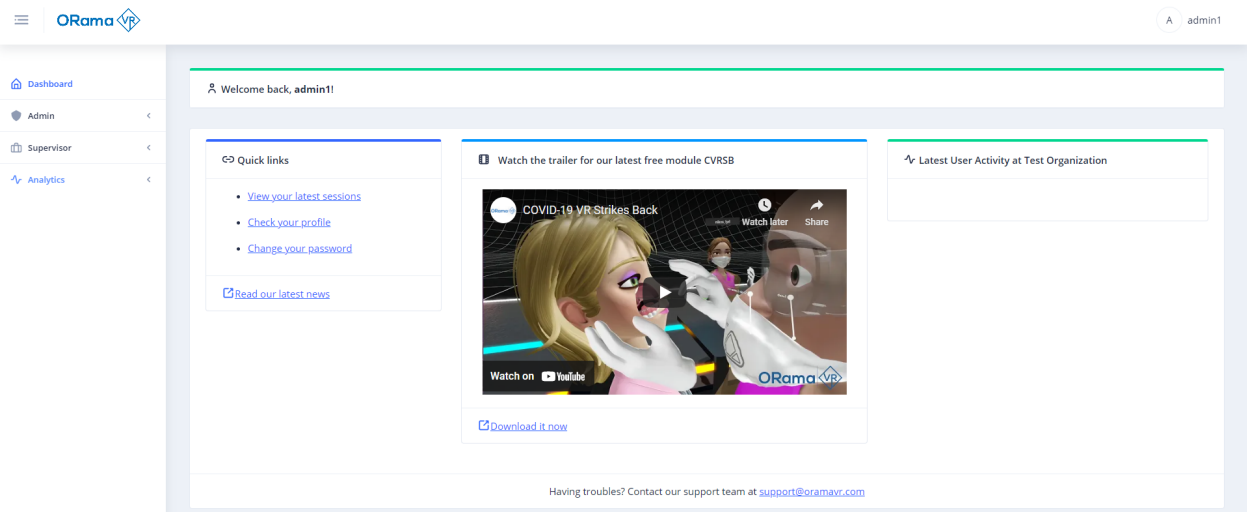
In any case, it is easy to translate the following steps to the equivalent Production configuration.

11.6.2 Product & User Creation

The first step is to login to the portal with an Admin user account.

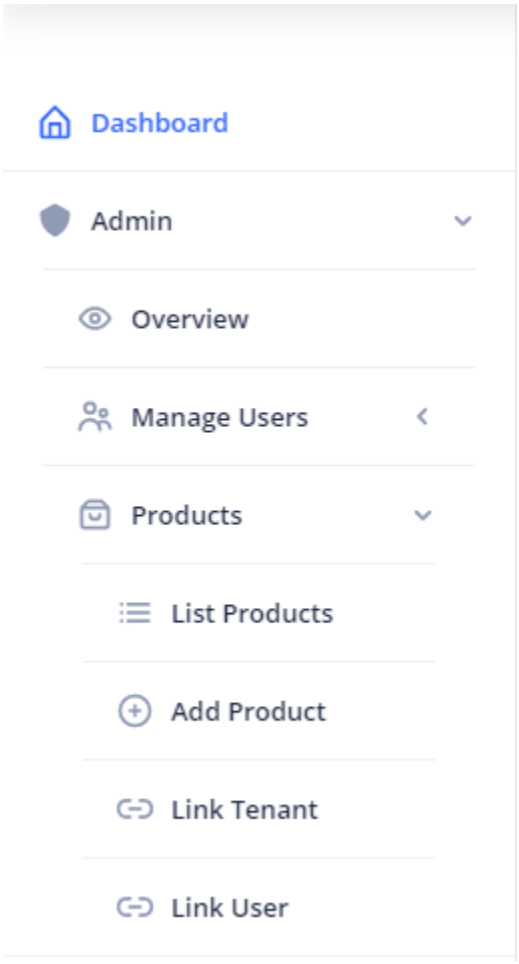
Point your browser to `localhost:4200` and you will be redirect to the Login Service `https://localhost:44355` for authentication.

Upon successful login, you will be redirected to the Portal Dashboard as below:



Add New Product

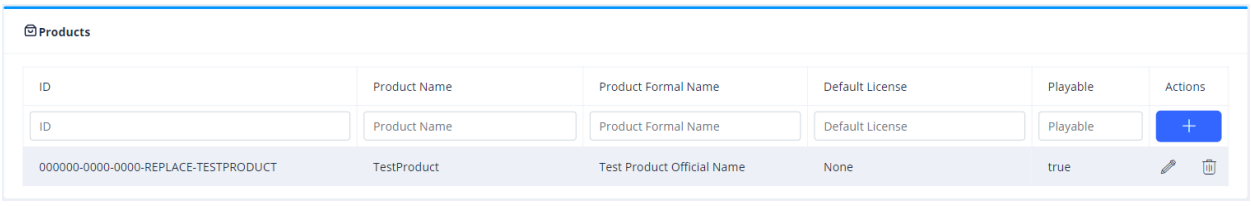
Proceed to expand the **Admin** menu item from the main sidebar, and further the child **Products** menu item as depicted in the figure:



From the **Products** menu, you can either click on the **List Products** menu item, or on the **Add Product** menu item.

List Products Option

If you clicked on the **List Products** you will see an overview of all your Products as the one below:



Click on the **Plus Icon button** on the right and a new table row will appear with an inline form. There, go on to create a product with the following configuration:

```
{
    "ProductName": "Test2021",
```

(continues on next page)

(continued from previous page)

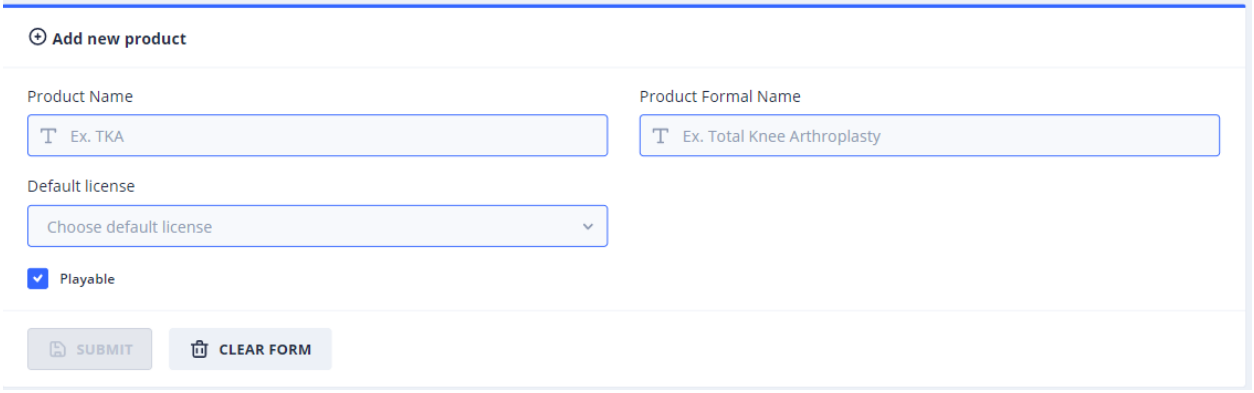
```
    "ProductFormalName": "This is the formal name of the product",
    "DefaultLicense": "None",
    "Playable": "True"
}
```

Click on the **Check Icon** on the right and a successful notification will appear on the top right of the page!

Note: Notice that the ID field is grayed out, this will be populated by the Login Service.

Add Product Option

If you clicked on the **Add Product** menu item, you will end up in the following page:



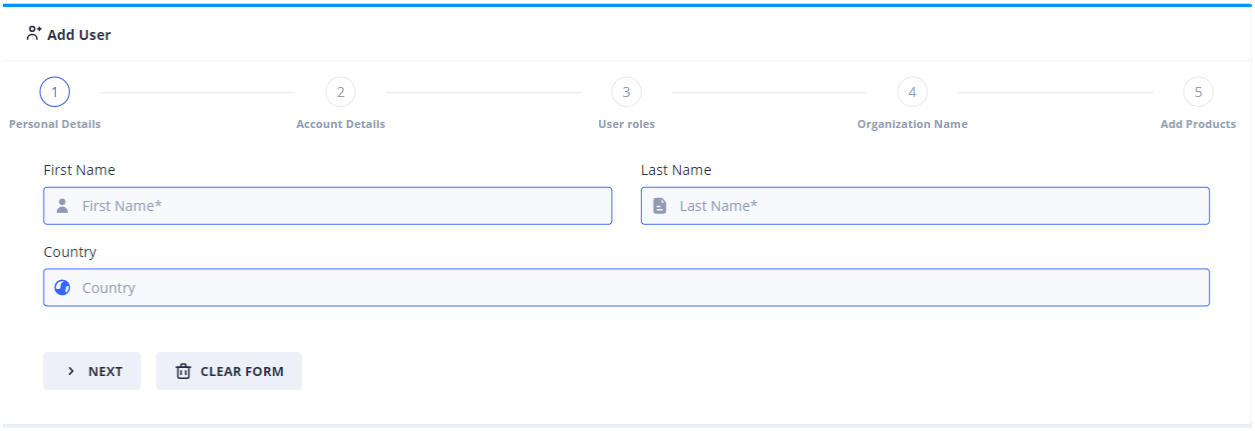
Here is a dedicated form that has also predefined options for the **DefaultLicense** and the **Playable** fields, instead of writing these info by hand.

Fill it in with the values in the JSON snippet above and click the **Submit** button to create the product.

User Creation

After the Product is created, navigate to the **Manage Users** expandable item and click on the **Create User** menu item.

The following *stepper* form will load:



With this form you can create a user and optionally assign him product licenses.

Step 1.

Proceed to fill in the **Personal Details** of the dummy user as you see fit.

Step 2.

In the **Account Details** step, create a username and a password for your user.

> **Warning:** When creating a user manually through the Portal, an email confirmation link will not be sent to the user's email address!
>
> The email address is considered **Confirmed** for the Login Service.
>
> By default, we assume admins are able to correctly register their users. This also helps testing user creation, because you do not have to enter a valid email address.

Step 3.

If the password meets the *Validator*, you can proceed to select the **User Roles**. If you user is a typical VR-only application user, select the *User* role only.

Otherwise, you can proceed to add more roles – but recall, roles are inclusive! (e.g. An admin has all three roles assigned, etc.)

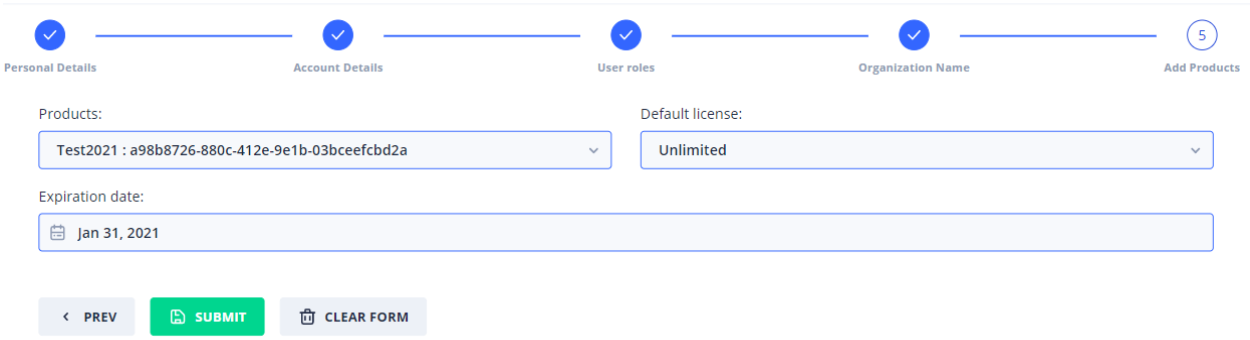
Step 4.

Select the Organization this user belongs to. For simplicity, select your own organization name or the **Default** one if applicable.

Step 5.

In the final step, we assign the user a license of the product we created earlier.

Make sure you input a date in the future and the **DefaultLicense** of the user is set to *Unlimited* as below:



Note: Unlimited means that the user can access the VR application frivolously, until the expiration day.

Go ahead and click the **Submit** button in the Step 5. to create the user. If everything went smoothly, you should see a green Notification bar at the top right corner.

Your user is now created with a product license attached! Congrats!

11.6.3 MAGES™ SDK Sample App

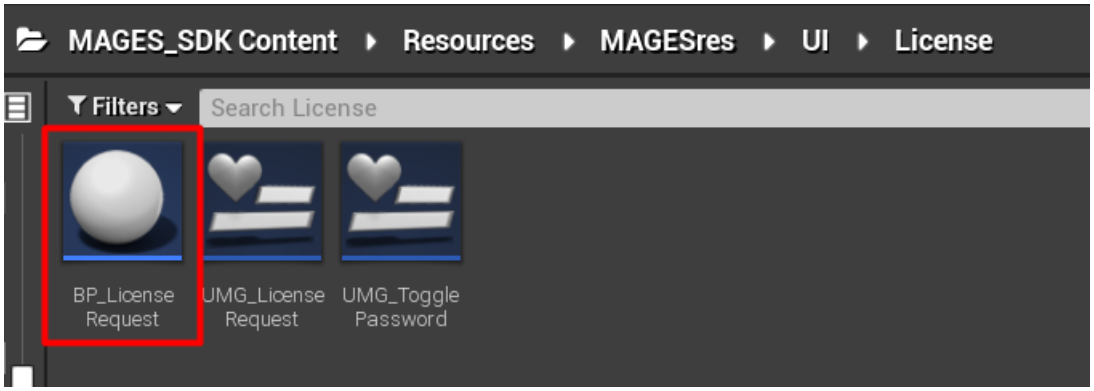
After the product and the user is set, let's move on to the SDK Sample App for the integration part with the Login service.

Open your MAGES SDK project in Unreal and select the **Sample App** scene from `MAGES_SDK Content > MAGES > Operation > Levels`.

Enable User Login Window

In order for the users we need to add in the scene the Login UI window. This can be found in :code:`MAGES_SDK Content > Resources > MAGESres > UI > License.

Drag and drop in the scene the preconfigured code::*BP_License_Request* blueprint.



Essentially, this blueprint will authenticate the user with given credentials from the UI's text boxes (i.e., username and password) and checkout a user license for the product defined in the scenegraph data asset (see below). Additionally, the underlying service will attempt to retrieve the **User Data** from the Login Service, that you can further utilize to upload analytics or populate your COOP, etc.

If any of the above steps fails, you will be prompted with an error message, and the Login screen will remain in place.

If the operation was successful, the Login screen will self-destroy and the user shall be able to continue with your module.

Client Configurations

Now we need to set the Client configurations to connect Unreal with the Login service.

We have to configure the `ScenegraphPdathDefinitions` data asset that has been referenced in your scenegraph actor.

The following fields need to be set: **Login URL:** The URL pointing to your login service needs to be set here. E.g. https://login.test.com/

Product Code: The product name (product code) you have set for this project through your portal. In our case Test2021.

Client ID: This ID defines how the VR application will attempt to autherize your users. By default this is UnityModuleWithoutSSO.

Client Secret: The client secret generated through the portal regarding this VR application. It is a 16 digit key, where each 4 digits are seperated by '-'.

Note: Currently MAGES SDK supports only User Authentication with Username & Password, but we plan to support also SSO authentication through browsers.

User Account Manager

By default, and for convenience, we populate the `UserAccountManager` class with the authenticated user information.

This includes username, the JWT token, and other optional values such as first and last name.

Therefore, you don't need to acquire any user information throughout the whole application experience. Everything is there for you to utilize as you see fit.

Uploading User Analytics

Proceed to open your data asset and add the URL pointing to the Upload function of your Analytics API. E.g. https://analytics.test2021.com/api/Upload

All we did is specify the API endpoint, and the Product Code from previous steps. MAGES_SDK automatically uses the user credentials to authorize and authenticate the user when the upload function is called.

Recall that AnalyticsAPI is connected with Login to work properly. Therefore, tokens are delegated to the Login service for authorization.

This is particularly useful for making authorized requests on behalf of the user.

Note: In a full playthrough with an authenticated user, if Analytics are uploaded correctly, you should be able to see them via the Web Portal.

This is everything required to integrate Unreal with the Web Services seamlessly.

Next step is to actually build your application and test it!

Happy coding!

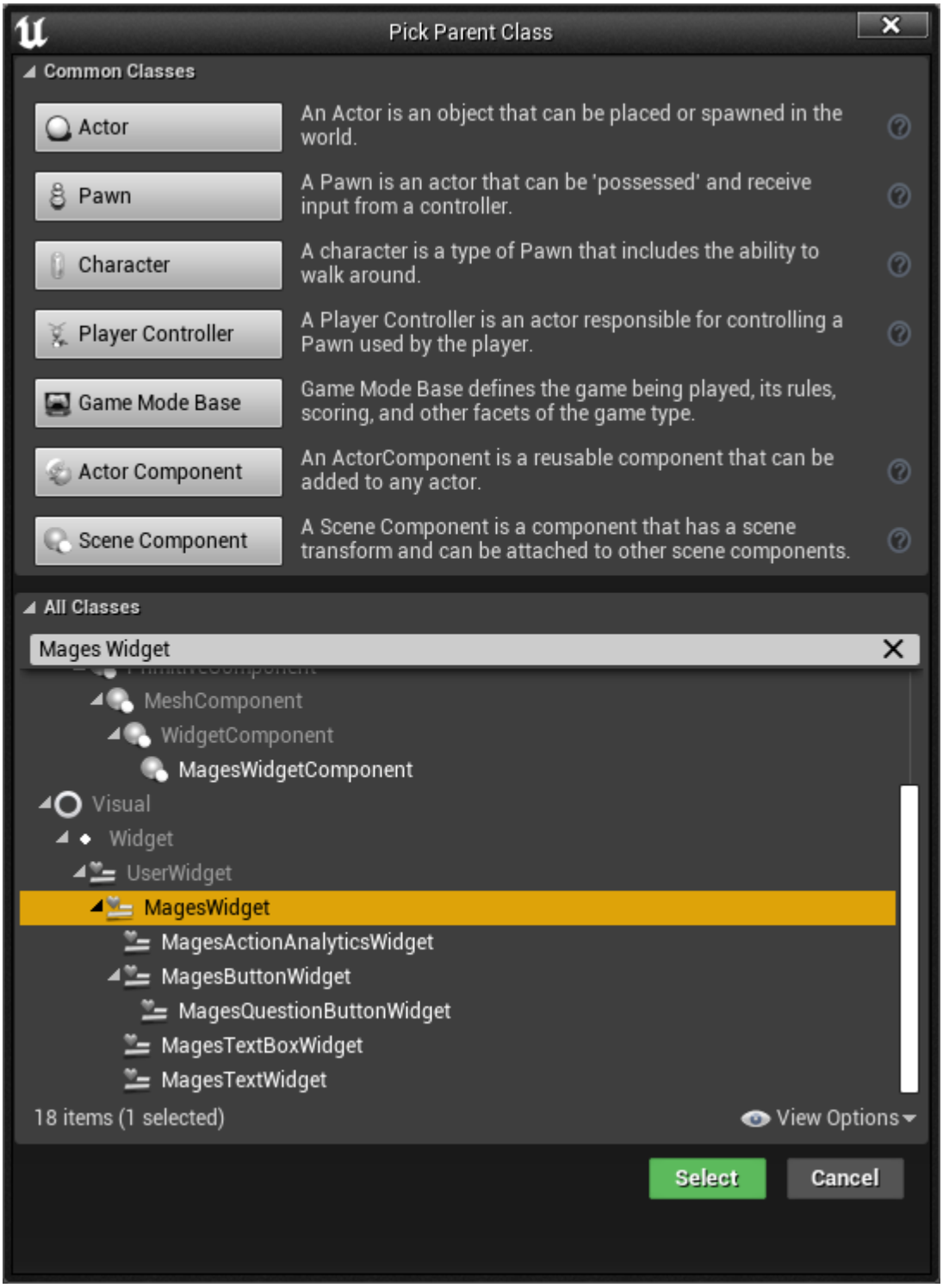
11.7 UI

11.7.1 Creating a Widget

UMG Designer

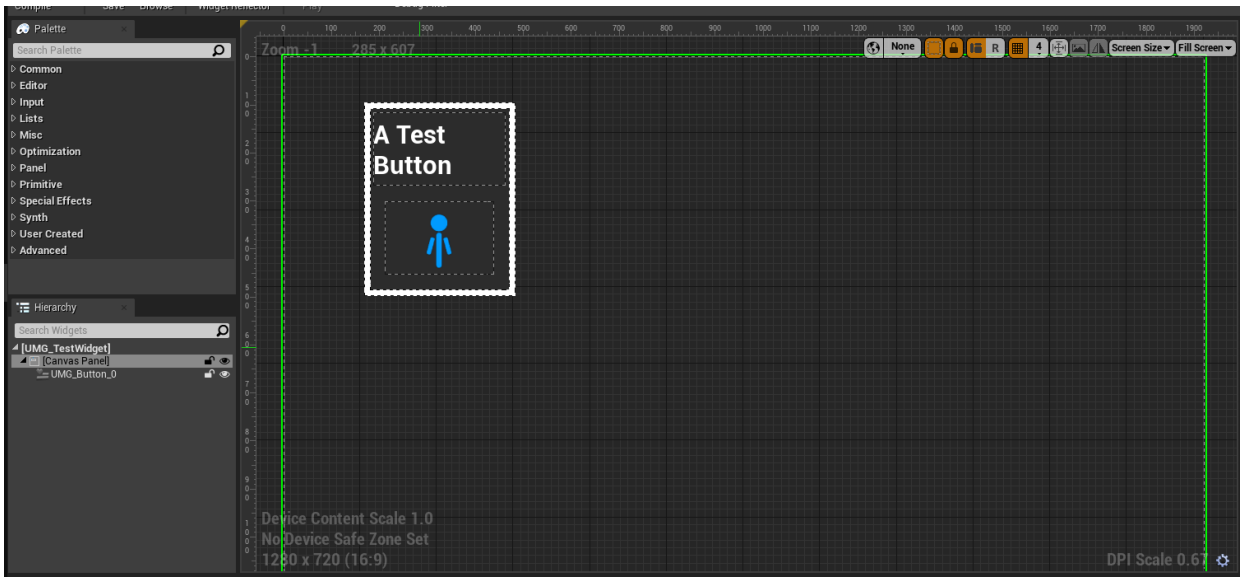
In this tutorial, we'll guide you through the process of creating a Mages Widget, and using it inside a level. We'll create a simple widget which will consist of a button and a text block. Pressing the button will change the text block.

First, create a new blueprint deriving from the Mages Widget class:

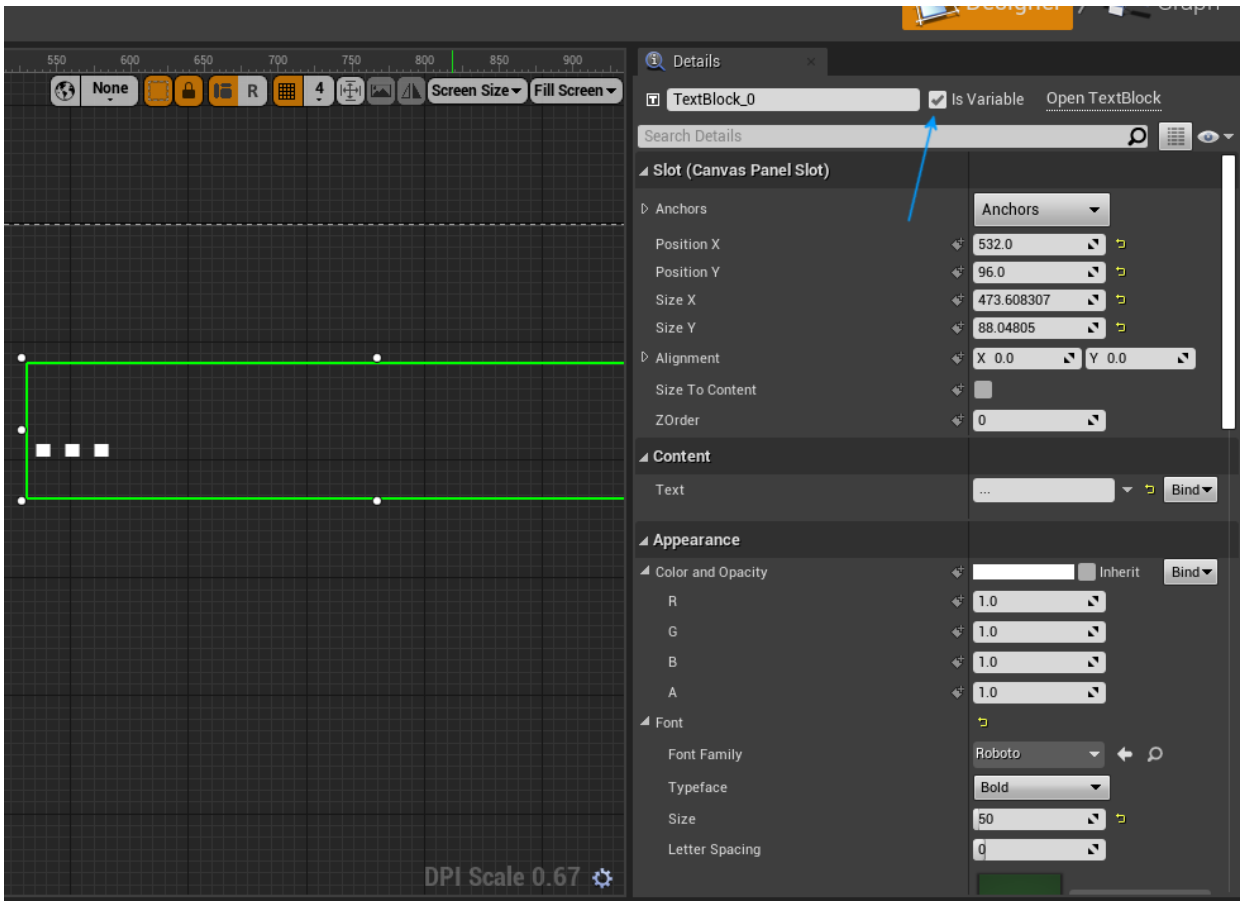


Add a canvas panel to hold our widgets, and a `UMG Button` widget under it:

Note: UMG Button is a template that encapsulates the functionality and design of all of the buttons used in our applications. You can create your own template button for use inside of your interfaces by creating a subclass of the `Mages Button Widget`.

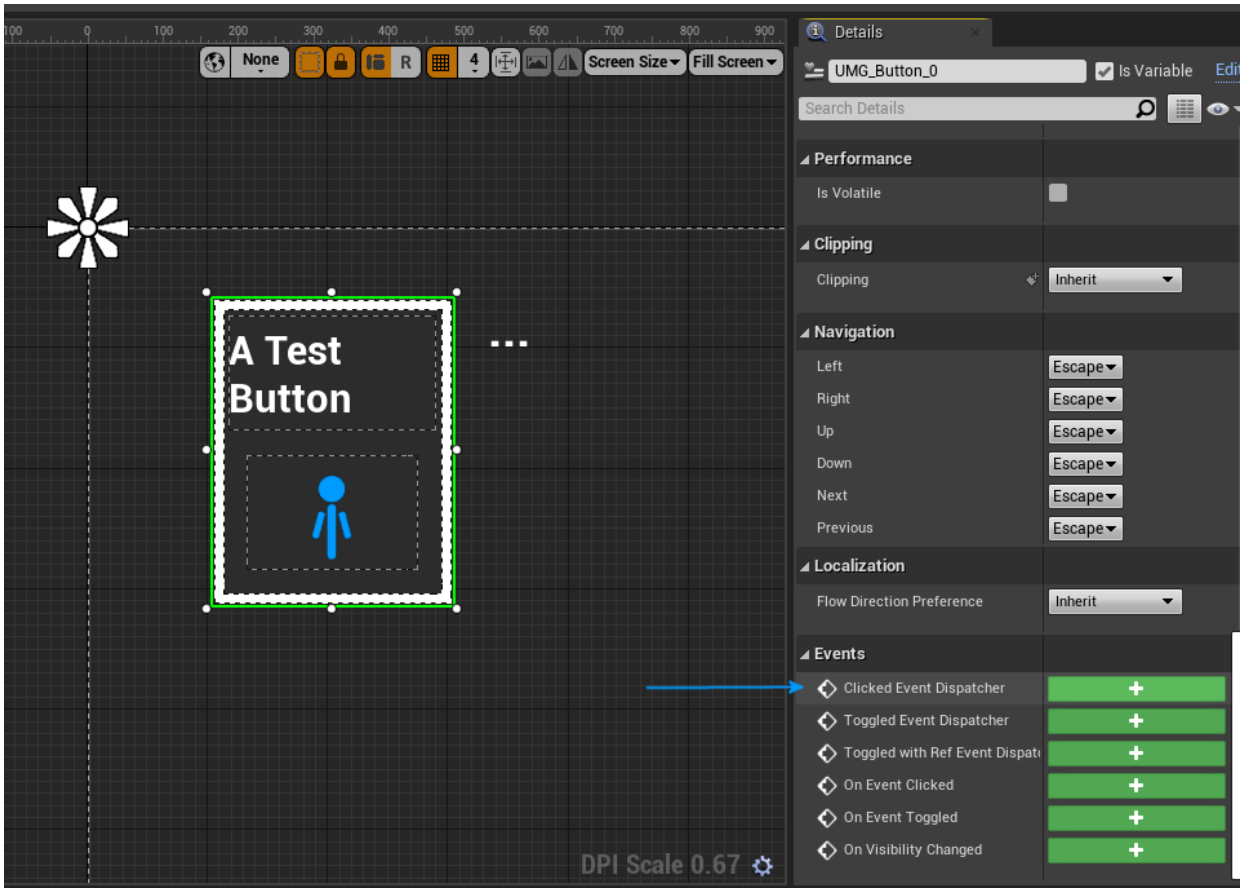


We'll also add a text block and mark it as a variable so that we can change it later:

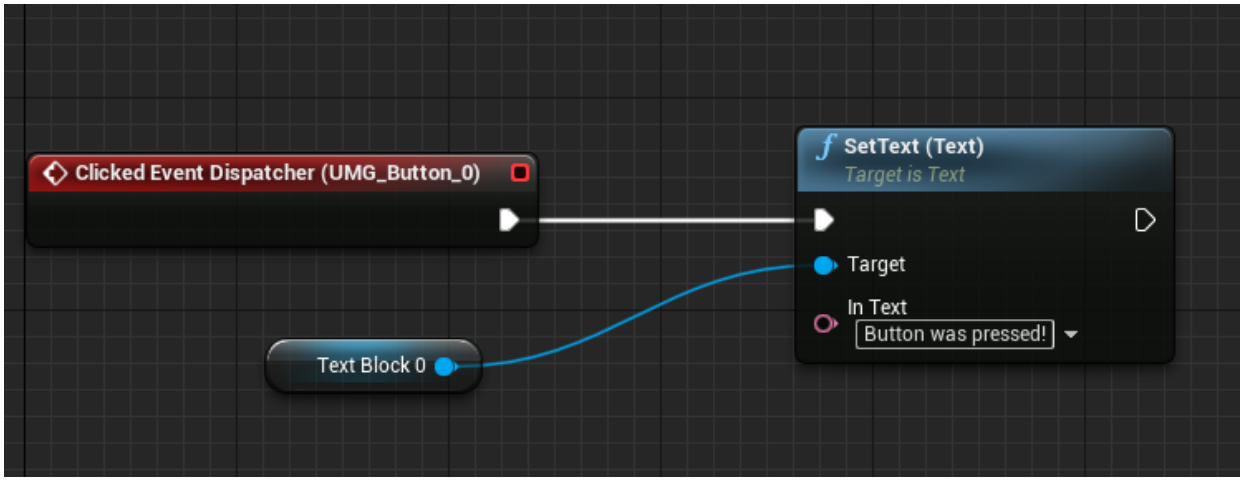


UMG Graph

Click on the button widget, and add an event for the `Clicked Event Dispatcher`:

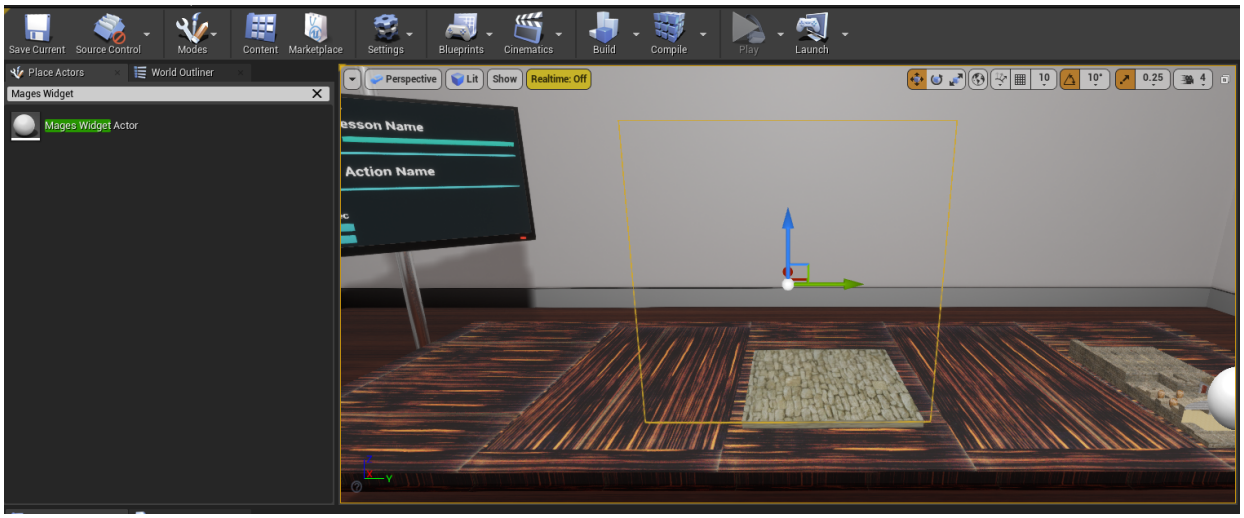


Here's the implemented event for the functionality we want:



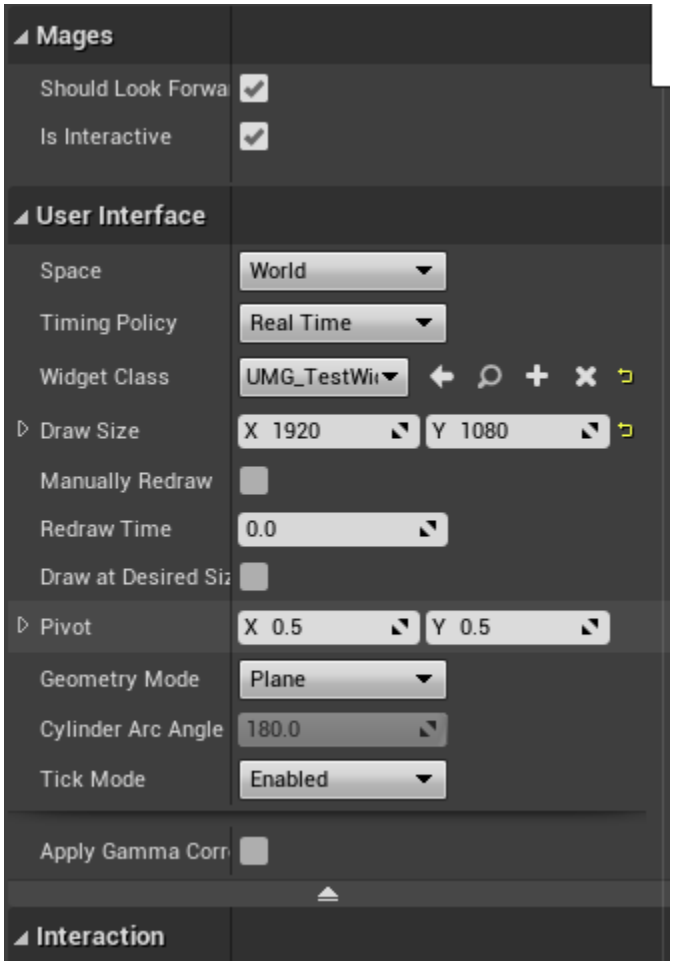
Adding it to the Level

Inside the level, add a new `Mages Widget Actor` from the "Place Actors" panel.



Note: The Mages Widget Actor, is a simple actor that wraps a `Mages Widget Component`. If you need a blueprint that contains widgets with different 3D transforms, you can use the component instead of the actor.

Afterwards, configure the actor to display our widget with the proper size.



Done! You can now preview the level in VR and see that it's working.

11.8 Soft Bodies

The MAGES SDK supports real time soft bodies as a part of its standard features

---

Note: Soft bodies are used to simulate soft and deformable surfaces like the human skin, liquids, internal organs, cloths and any other deformable surfaces.

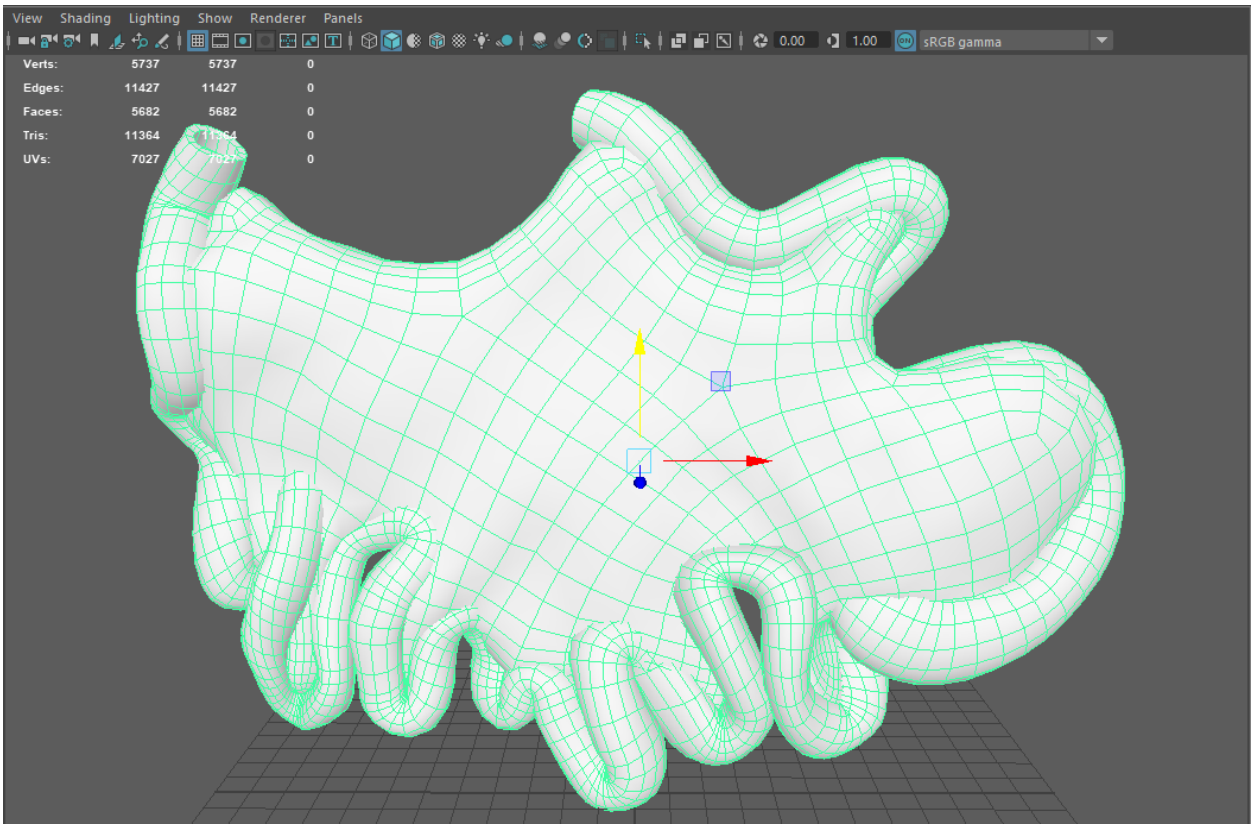
In this tutorial we will demonstrate how to setup the soft body behaviour on a human bowel model (small intestine).

Tutorial overview:

1. Exporting the model from Maya
2. Soft body setup
3. General discusson

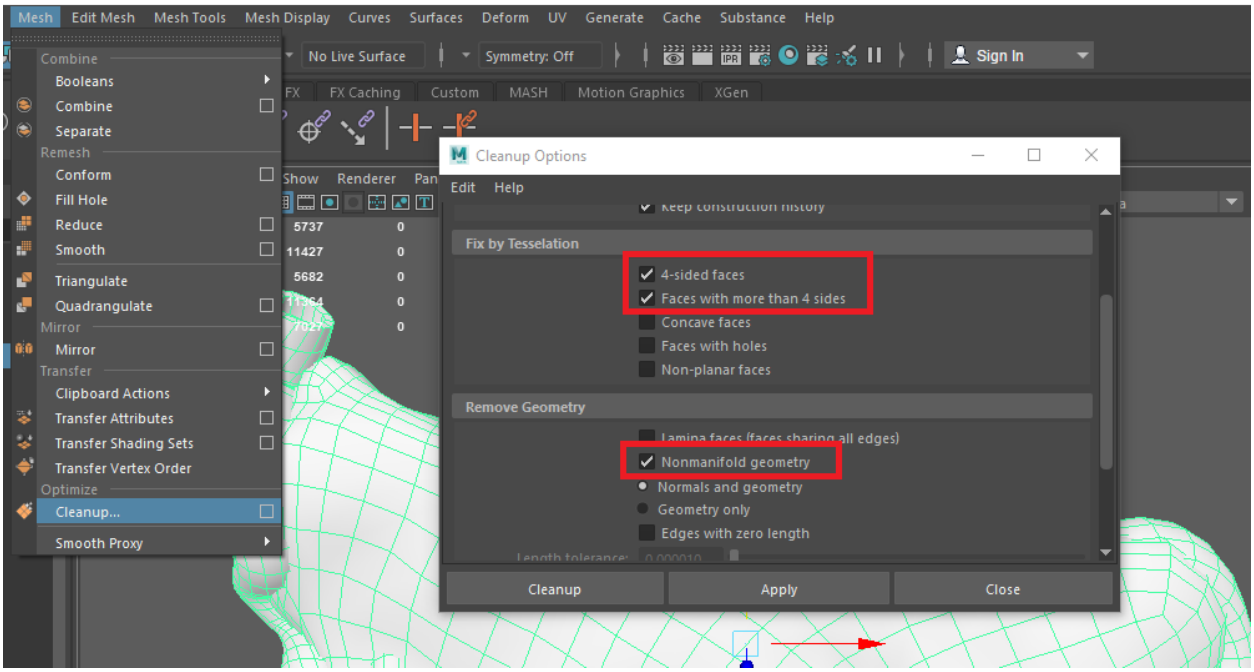
11.8.1 Exporting the model from Maya

The image below shows the bowel model in Maya.



This model has 5700 vertices. The number of vertices is really important in our soft body system both for the real-time simulation updates as well as the pre-load configuration and caching preprocessing time. Both operations (pre-processing and real-time updates) all depend heavily on the number of vertices under processing

Make sure to do a cleanup before exporting the FBX. In the cleanup window select the **4-sided faces**, **Faces with more than 4 slides** and the **Nonmanifold geometry**.

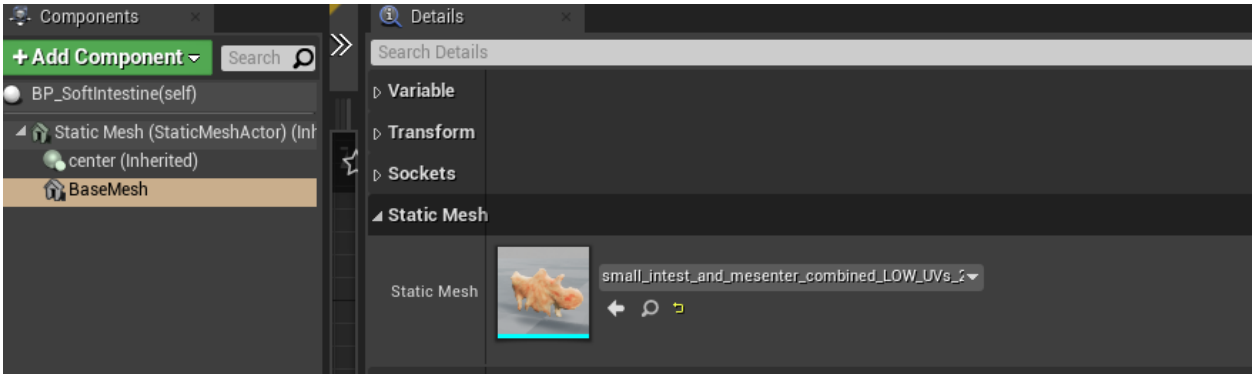


Then export the model using the FBX format.

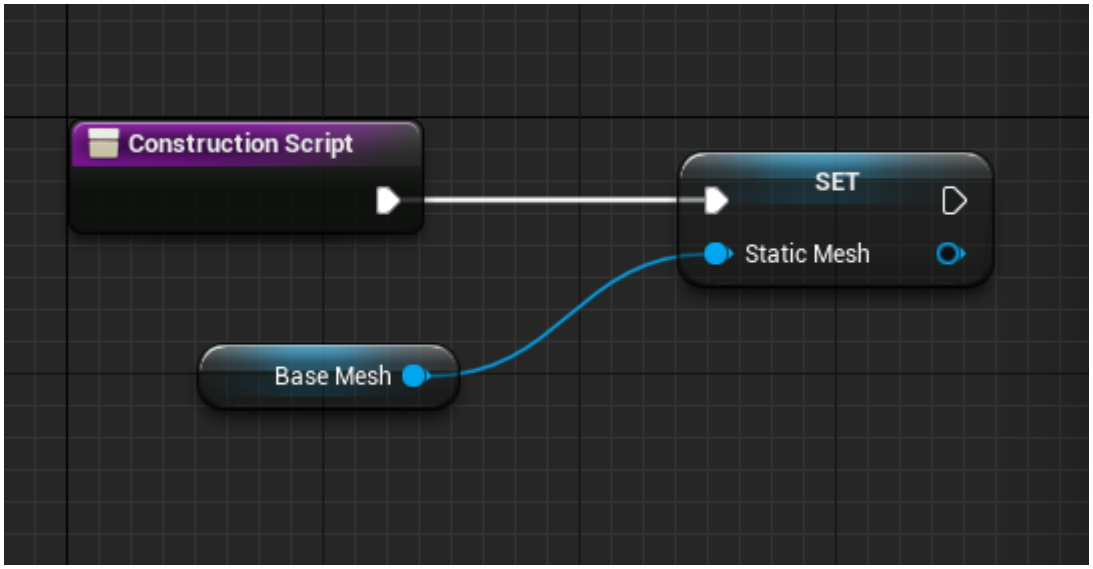
11.8.2 Softbody setup

The first part is to create the Blueprint Actor that will serve have the softbody properties. Right click in your content browser, select blueprint class and make sure it inherits from the CreateDeformMesh class.

Once you have selected which static mesh, you want to deform, add a static component to the new Blueprint and specify the 3D model that will be soft. For this example we will be using the intestine 3D model.



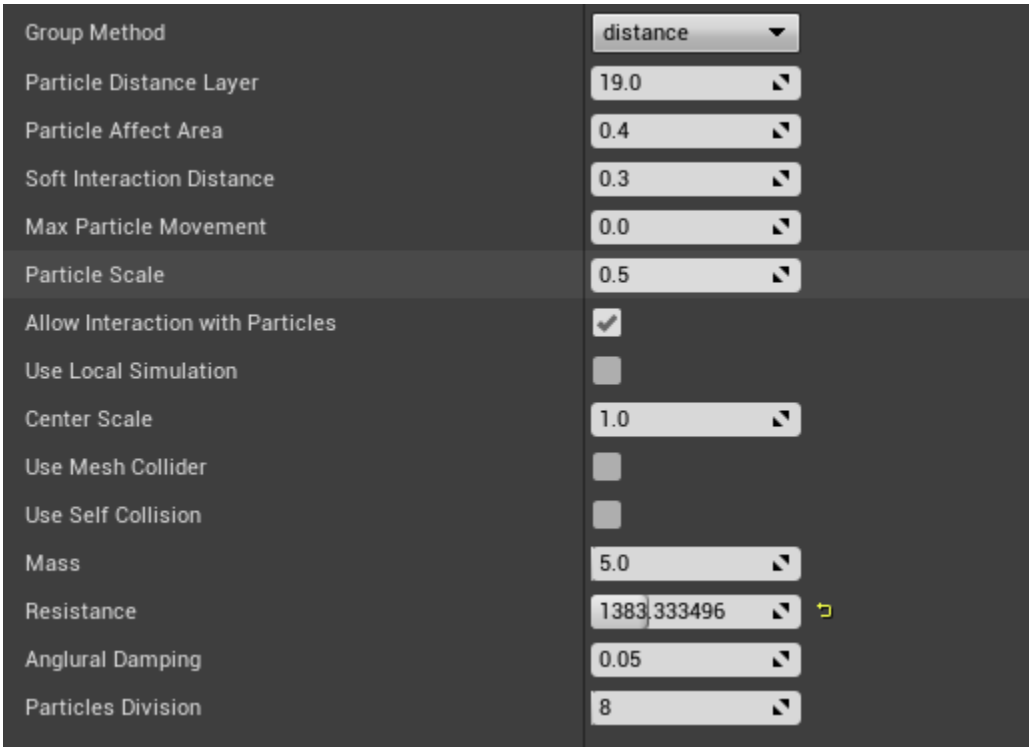
In order for the softbody algorithm to be initialized we need also set the reference of the static mesh from the CreateDeformClass Actor. So, from the construction script of the blueprint we will set the Static Mesh variable. In this case the BaseMesh is the static mesh component added to our blueprint.



Configuring the SoftBody

In order to be able to simulate different types of softbodies MAGES provides 3 types of algorithms, as well as many configurable variables that developers can use to specify how the mesh will deform. On startup the CreateDeformMesh class will generate *particles* attached to vertices of the mesh which will be interactable and influence the mesh, deforming through physics. How many particles are generated, how dense they are, as well as how they influence the vertices is specified by the configuration of the *CreateDeformMesh* class.

These are the settings we used for the intestine model:



- The *group method* specifies how the particles will be grouped under each interactable item. Usually, the best results are achieve based on **Distance_with_soft_interaction** as it combines the two previous options.
- The mesh property refers to the geometry of each particle. In this case we are using cubes.
- The **Particle Distance Layer** refers to the distance between the particles. The higher the number the less particles will be generated.
- The same principle applies for the **Soft Interaction Distance**.
- *Resistance* and *mass* refers to how much each particle will influence the vertices to return back to their equilibrium.

As you can see in, the following picture, from the above configuration the following particles are generated in the scene.

The above example can be found in the blueprint BP_SoftIntestine in the folder *MAGES_SDK Content/MAGES/SDKAddons/SoftBodies*

11.8.3 General discussion

Soft bodies introduce a powerful way to simulate deformable surfaces. The MAGES SDK offers a plug-and-play solution for particle-based soft bodies in Unreal, ideal for medical or any other use.

It is important to consider that real-time soft body simulation uses advanced physics algorithms to calculate the object's movement per vertex, paying respect to its original shape. This is why you need to be extra careful with the script configuration as well as the model's topology for best results.

11.9 Multiplayer

Applications built with MAGES™ SDK are multiplayer/network ready, meaning that with a few more actions needed by the developers, multiple users can cooperate and complete these simulations together online [M_1].

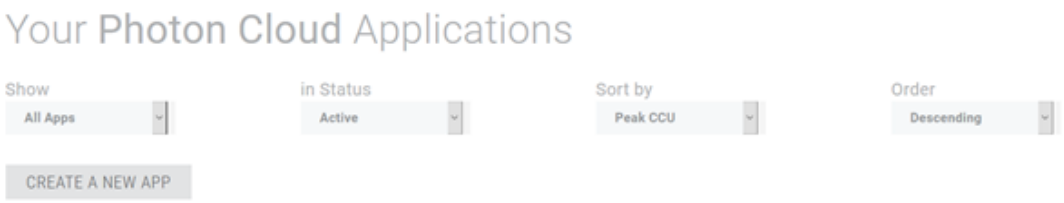
In this section we will go through the details that developers need to check to ensure that their application is multiplayer/network ready, as well as a tutorial on how a user can create and/or join multiplayer sessions within a MAGES™ application.

11.9.1 Networking Developer Guidlines

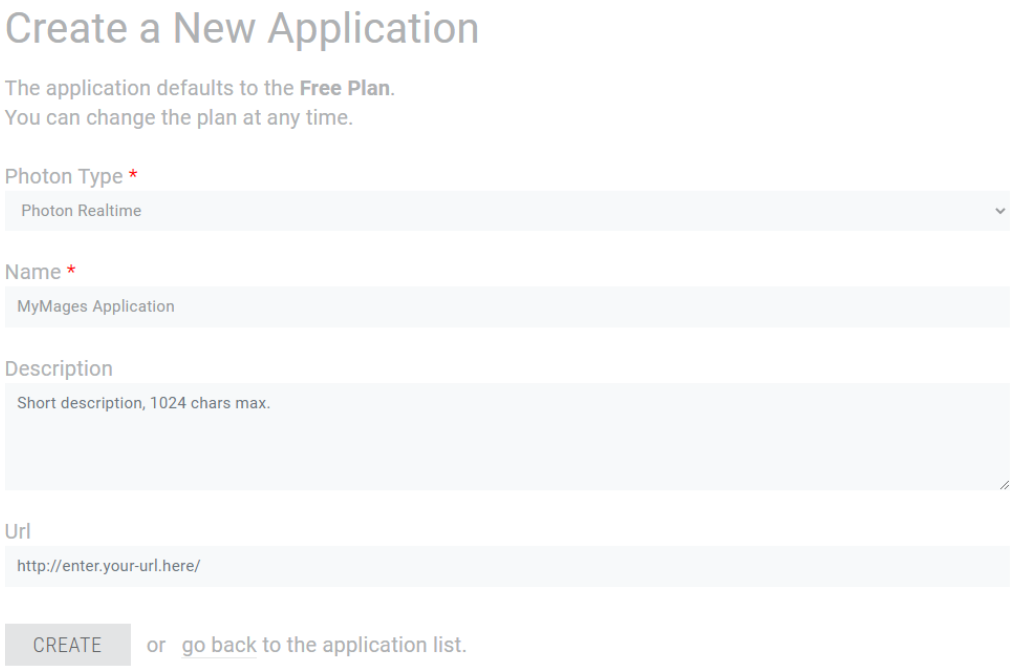
MAGES™ SDK supports Photon networking as the default networking API. Photon SDK is already installed in the MAGES plugin.

First, you need to create an account at Photon.

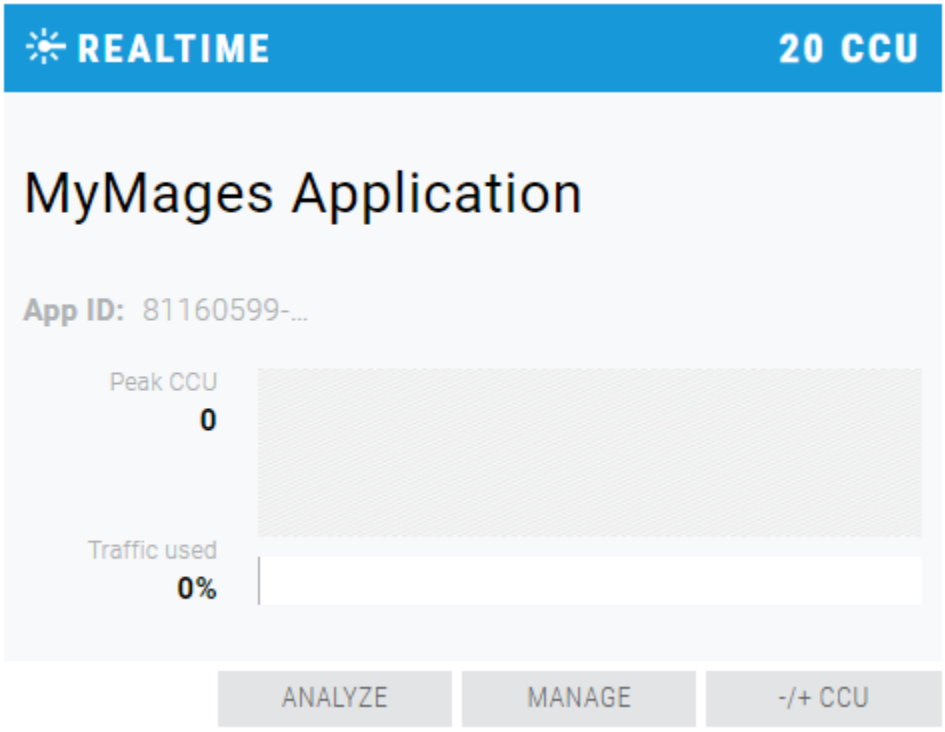
After successful registration, go to the Photon Dashboard and click the *CREATE A NEW APP* button.



Select Photon Realtime from the Photon type dropdown, give a name to your application and click the *CREATE* button.

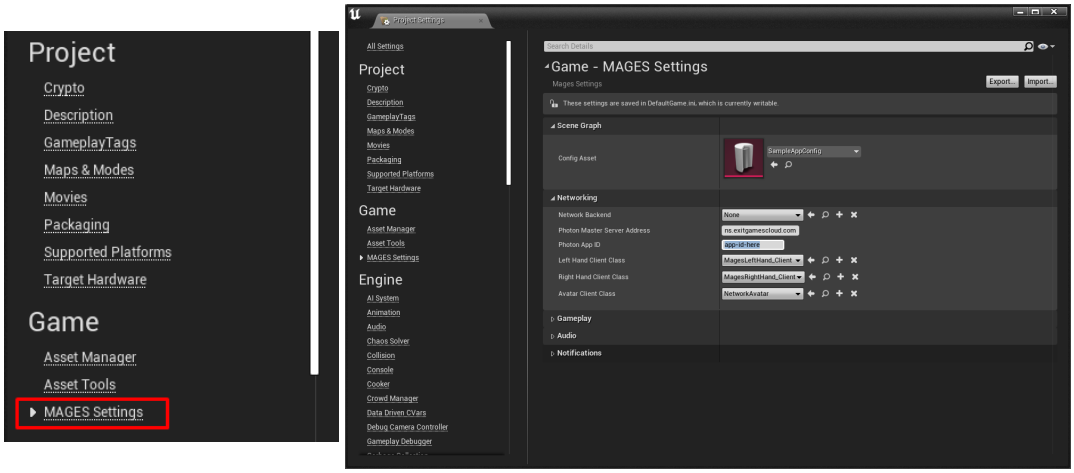


Navigate back to the main dashboard page and you will see your new RealTime application. You need to copy the App ID.



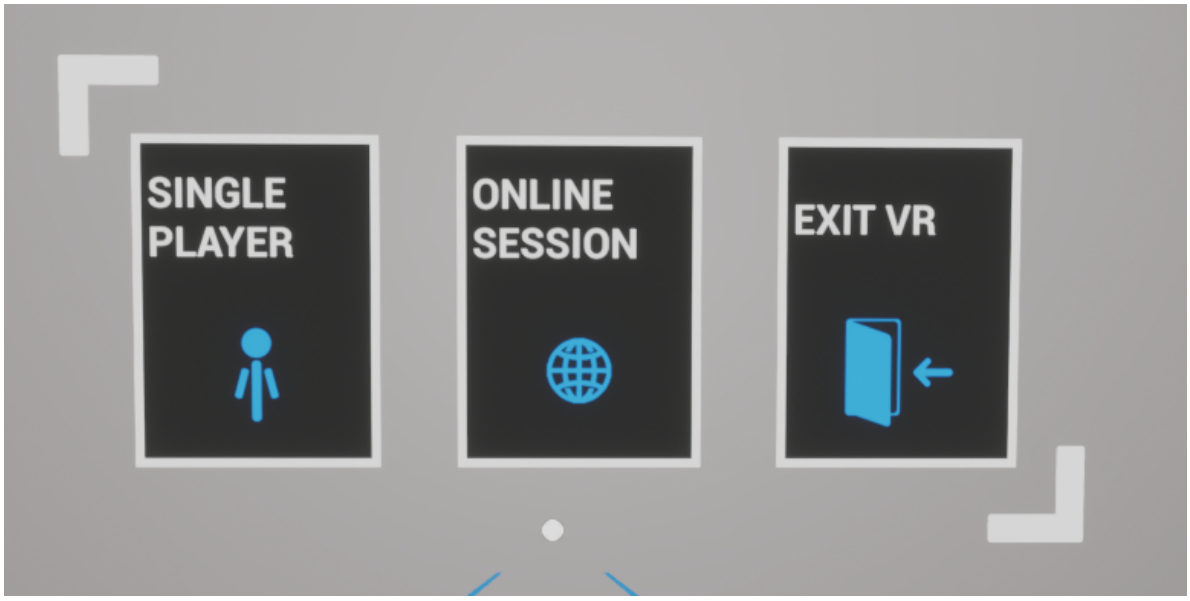
Note: Click the code next to the App ID to reveal the full key.

Open the Unreal project and in the project's settings; select the `MAGES Settings` section on the left and paste the Photon App ID on the `Photon App ID` field.



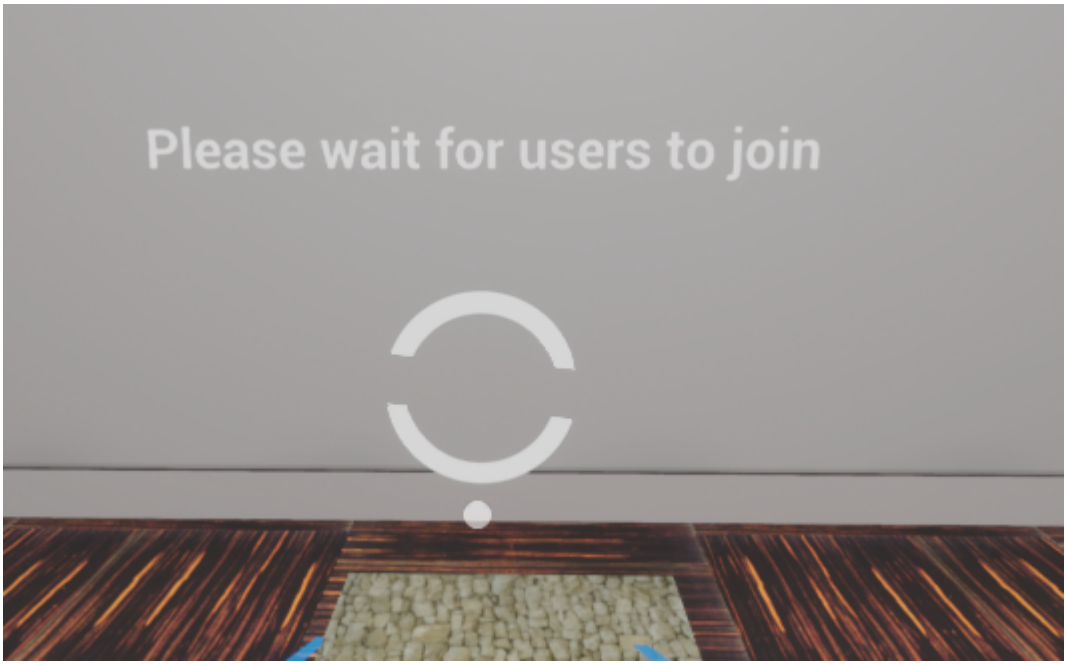
11.9.2 Host Online Session

1. Once you start the simulation, you will be greeted with two options. Starting the simulation in Single Player, or go to Online Sessions.



2. Select the Online Sessions. There you will be able to create a new session and wait for other to join, or join an existing session that is demonstrated on the Sessions board.

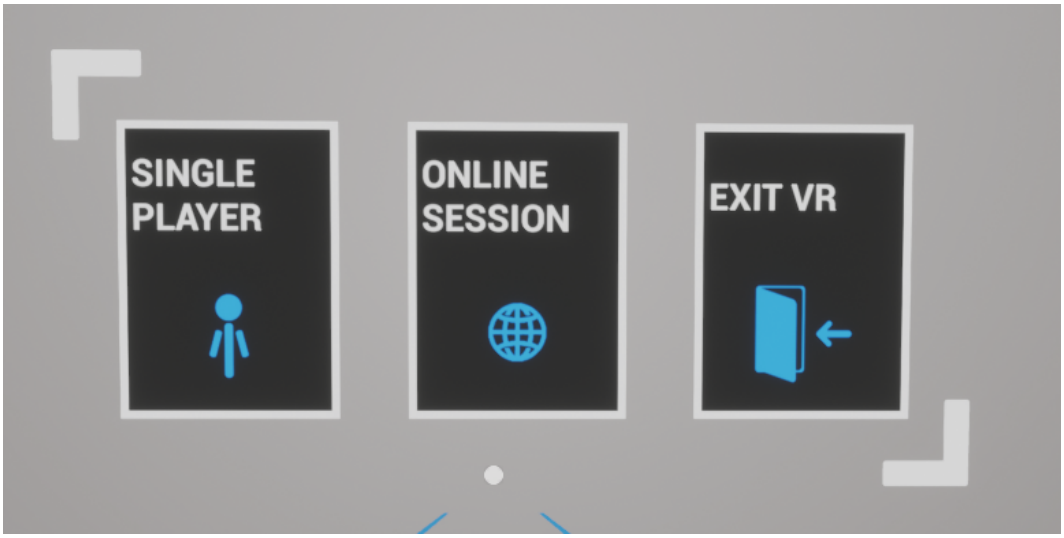
3. To Create a session, select the Create New Session button.
4. Wait for others to join. If at least one more user has joined, you will be able to start your online session.



5. If you want to go back, select Exit VR to quit.

11.9.3 Join Online Session

1. Once you start the simulation, you will be greeted with two options. Starting the simulation in Single Player, or go to Online Sessions.



2. Select the Online Sessions. There you will be able to create a new session and wait for other to join, or join an existing session that is demonstrated on the Sessions board.

3. Select an available session from the Sessions board, and select Join Session. Each available session will demonstrate all connected users.

4. If you want to go back, select Exit VR to quit.

References

CHAPTER

TWELVE

VIDEO TUTORIALS

12.1 Video Tutorials

Note: Comming Soon...

CHAPTER

THIRTEEN

CLASS REFERENCE

13.1 MAGES_SDK Class Reference

Reference and Index:

13.1.1 Global Namespace

namespace Collections

namespace Diagnostics

namespace DotNETCommon

namespace ELicenseType

enum ELicenseType::Type

```
#include <OAuthClient.h>

enum Type

    None      = -1,
    Unlimited = 1,
    Free      = 2,
    Demo      = 10,
    Developer = 100,
;
namespace ELicenseType

// enums

enum Type;

 // namespace ELicenseType
```

namespace ESyncTransformFlags

```
namespace ESyncTransformFlags

// typedefs

typedef uint8 Type;

// enums

enum

    None     = 1,
    Rotation = 2,
    Position = 4,
;

 // namespace ESyncTransformFlags
```

namespace ExitGames

namespace ExitGames::Common

template class ExitGames::Common::JVector

```
namespace Common

// classes

template <typename EType>
class JVector;

 // namespace Common
```

namespace ExitGames::LoadBalancing

```
namespace ExitGames

// namespaces

namespace ExitGames::Common;
namespace ExitGames::LoadBalancing;

 // namespace ExitGames
```

namespace IO

namespace MagesMath

struct MagesMath::FPointNormalPlane

Overview

```
#include <MagesMathUtility.h>

struct FPointNormalPlane

    // fields

    FVector N;
    float D;

    // construction

    FPointNormalPlane();
    FPointNormalPlane(FVector N, float D);
    FPointNormalPlane(FVector P1, FVector P2, FVector P3);

    // methods

    FORCEINLINE FVector GetPoint() const;
;
```

Detailed Documentation

Construction

`FPointNormalPlane(FVector N, float D)`

Construct Plane from Point-Normal

`FPointNormalPlane(FVector P1, FVector P2, FVector P3)`

Construct Plane from 3 Points

Methods

`FORCEINLINE FVector GetPoint() const`

Get a point on the plane; *not* randomly, just get a point

struct MagesMath::FQuadruplet

```
#include <MagesMathUtility.h>

struct FQuadruplet

    // fields

    int32 A;
    int32 B1;
    int32 B2;
    int32 C;
;
```

struct MagesMath::FTriangle

```
#include <MagesMathUtility.h>

struct FTriangle

    // fields

    FVector A;
    FVector B;
    FVector C;
;
```

struct MagesMath::FTriplet

```
#include <MagesMathUtility.h>

struct FTriplet

    // fields

    int32 A;
    int32 B;
    int32 C;
;
```

Overview

```
namespace MagesMath

// structs

struct FPointNormalPlane;
struct FQuadruplet;
struct FTriangle;
struct FTriplet;
```

```
// global functions

int FindRoots(float K, float M, float N, float Out[2]);

static FORCEINLINE FPointNormalPlane PlanePerpendicularToPlane(
    const FPointNormalPlane& TearPlane,
    FVector A,
    FVector B
    );

static FORCEINLINE FVector LineIntersectPlane(
    FVector Point0,
    FVector Point1,
    const FPointNormalPlane& Plane
    );

static FORCEINLINE float PointIntersectPlane(
    FVector Point,
    const FPointNormalPlane& Plane
    );

static FORCEINLINE bool SameSide(FVector P1, FVector P2, FVector A, FVector␣
↪B);

static FORCEINLINE bool IsPointInsideTriangle(
    FVector Point,
    FTriangle& Triangle
    );

static FORCEINLINE FVector FindMidpoint(FVector A, FVector B);
static FORCEINLINE int Orient3D(FVector A, FVector B, FVector C, FVector D);

static FORCEINLINE FVector RotatePointAroundPivot(
    FVector Point,
    FVector Pivot,
    FQuat Rotation
    );

static FORCEINLINE FVector ProjectPointOntoPlane(
    FVector PlaneNormal,
    FVector PlanePoint,
    FVector Point
    );

static FORCEINLINE FVector2D SortSwizzleVector(FVector2D V);

static FORCEINLINE FTriplet FindNewFace(
    TArray<int32>& triangles,
    int32 i,
    int32 cid0,
    int32 cid1,
    int32 id0,
    int32 id1
    );
```

```
static FORCEINLINE bool IsPointWithinShapeComponent(
    FVector Point,
    UShapeComponent* Shape,
    const FTransform& Transform = FTransform::Identity
    );

static FORCEINLINE FVector LineIntersectPlane(
    const FLineSegment& Segment,
    const FPointNormalPlane& Plane
    );

static FORCEINLINE void DrawDebugWireTriangle(
    const UWorld* World,
    FVector A,
    FVector B,
    FVector C,
    FColor Color,
    bool Persistent = false,
    float Duration = 0.f
    );

static FORCEINLINE FVector4 BoundedPlaneToWorldSpaceCartesian(
    UBoundedPlane* plane,
    FTransform Transform
    );

static FORCEINLINE FVector4 PointNormalToCartesian(
    FVector point,
    FVector normal
    );

 // namespace MagesMath
```

Detailed Documentation

Global Functions

```
int FindRoots(float K, float M, float N, float Out[2])
```

Finds the roots of a 2nd degree equation with K, M, N as the parameters

```
static FORCEINLINE FPointNormalPlane PlanePerpendicularToPlane(
    const FPointNormalPlane& TearPlane,
    FVector A,
    FVector B
    )
```

Find a new plane perpendicular to the tear_plane and that passes from v1, v2

```
static FORCEINLINE FVector LineIntersectPlane(
    FVector Point0,
    FVector Point1,
    const FPointNormalPlane& Plane
    )
```

Find intersection of a line segment and a plane

```
static FORCEINLINE float PointIntersectPlane(
    FVector Point,
    const FPointNormalPlane& Plane
    )
```

Returns the side of the up-projected point with respect to the plane: +,- or 0 = on the plane

```
static FORCEINLINE bool IsPointInsideTriangle(
    FVector Point,
    FTriangle& Triangle
    )
```

Test Point inside Triangle

```
static FORCEINLINE FVector FindMidpoint(FVector A, FVector B)
```

Midpoint of A and B

```
static FORCEINLINE int Orient3D(FVector A, FVector B, FVector C, FVector D)
```

Returns the orientation of the tetrahedron defined by a,b,c,d

```
static FORCEINLINE FVector ProjectPointOntoPlane(
    FVector PlaneNormal,
    FVector PlanePoint,
    FVector Point
    )
```

Project point onto a plane with the specified normal and contained point

```
static FORCEINLINE bool IsPointWithinShapeComponent(
    FVector Point,
    UShapeComponent* Shape,
    const FTransform& Transform = FTransform::Identity
    )
```

Returns true when the point is inside the shape

namespace MagesNetworkStatusCode

```
namespace MagesNetworkStatusCode

// enums

enum

    ConnectionSuccessful = 255,
    Invalid              = 0,
    ConnectionFailed     = -1,
;

 // namespace MagesNetworkStatusCode
```

namespace System

namespace UnrealBuildTool

enum EActionType

```
#include <MagesSceneGraph.h>

enum EActionType

    UMETA =(DisplayName = "Crucial"),
;
```

enum EAmbientAudioType

```
#include <MagesSettings.h>

enum EAmbientAudioType

    Music,
    Noise,
;
```

enum EAnalyticsColliderBehavior

```
#include <MagesAnalyticsAsset.h>

enum EAnalyticsColliderBehavior

    AvoidObjects,
    MustHitObjects,
    StayWhileInteracting,
;
```

enum EAnalyticsErrorType

```
#include <MagesAnalyticsAsset.h>

enum EAnalyticsErrorType

    Warning,
    Error,
    CriticalError,
;
```

enum EAnalyticsFactorImportance

```
#include <MagesAnalyticsAsset.h>

enum EAnalyticsFactorImportance

    VeryLittle,
    Little,
    Neutral,
    Big,
    VeryBig,
;
```

enum EAudioClipType

```
#include <MagesSettings.h>

enum EAudioClipType

    ActionPerform,
    ActionUndo,
    Notification,
    Error,
;
```

enum ECollisionType

```
#include <AvoidObjectFactor.h>

enum ECollisionType

    HitError,
    StayInCollider,
    HitBeforePerform,
    HitWaitForTime,
;
```

enum EControllerDOF

```
#include <MagesControllerClass.h>

enum EControllerDOF

    TwoDOF,
    ThreeDOF,
    SixDOF,
;
```

enum EControllerTypes

```
#include <MagesControllerClass.h>

enum EControllerTypes

    HTCViveController,
    OculusTouchController,
    WindowsMixedRealityController,
    NoController,
;
```

enum EDifficulty

```
#include <UserAccountManager.h>

enum EDifficulty

    Easy,
    Medium,
    Hard,
;
```

enum EErrorType

```
#include <ScoringFactor.h>

enum EErrorType

    Warning,
    Error,
    CriticalError,
;
```

enum EFactorImportance

```
#include <ScoringFactor.h>

enum EFactorImportance

    VeryLittle,
    Little,
    Neutral,
    Big,
    VeryBig,
    Trivial,
;
```

enum EHandState

```
#include <MagesHand.h>

enum EHandState

    Uninitialized,
    Idle,
    GripDownNotInteracting,
    GripDownInteracting,
    GripToggleOnNotInteracting,
    GripToggleOnInteracting,
    GripToggleOff,
;
```

enum EInteractionStyle

```
#include <MagesHand.h>

enum EInteractionStyle

    UMETA =(DisplayName = "Hold"),
;
```

enum ELoginStatus

```
#include <AuthenticationHandler.h>

enum ELoginStatus

    Trying             = 0,
    InternalFailure    = 1,
    InvalidLicense     = 2,
    Success            = 3,
    InvalidCredentials = 4,
    MissingCredentials = 5,
;
```

enum EMagesButtonInteractionMethod

Overview

```
#include <MagesButtonWidget.h>

enum EMagesButtonInteractionMethod

    Default,
    TapAndRepeat,
;
```

Detailed Documentation

Enum Values

`Default`

Activated when the user lets go of the button on top of the widget bounds

`TapAndRepeat`

Activated when the user presses the button. Will repeat after a set interval

enum EMagesButtons

```
#include <MagesButtonsHelper.h>

enum EMagesButtons

    System,
    ApplicationMenu,
    Grip,
    DPad_Left,
    DPad_Up,
    DPad_Right,
    DPad_Down,
    A_Button,
    B_Button,
    X_Button,
    Y_Button,
    Axis0,
    Axis1,
    Axis2,
    Axis3,
    Axis4,
    Touchpad,
    Trigger,
    Back,
    Stick,
;
```

enum EMagesControllerButtons

```
#include <MagesControllerClass.h>

enum EMagesControllerButtons

    TriggerButton,
    GripButton,
    MenuButton,
    A,
    B,
    X,
    Y,
```

```
    ThumbStick,
;
```

enum EMagesDeformableMeshType

```
#include <MagesDeformationComponent.h>

enum EMagesDeformableMeshType

    StaticMesh,
    SkeletalMesh,
;
```

enum EMagesSDKIntegrations

```
#include <MagesPlayer.h>

enum EMagesSDKIntegrations

    None,
    FallbackNonVR,
    Ovid,
;
```

enum EMagesSyncTransformMode

```
#include <MagesNetTransform.h>

enum EMagesSyncTransformMode

    None,
    Position,
    Rotation,
    All,
;
```

enum ENetVarType

```
#include <MagesNetwork.h>

enum ENetVarType

    Integer,
    Floating_Point,
    Boolean,
    String,
;
```

enum EOperationDifficulty

```
#include <MagesSceneGraph.h>

enum EOperationDifficulty

    UMETA =(DisplayName = "Hard"),
;
```

enum EOvidVRHand

```
#include <MagesControllerClass.h>

enum EOvidVRHand

    Left  = 0,
    Right = 1,
;
```

enum EOwnershipOption

```
#include <MagesView.h>

enum EOwnershipOption

    Fixed,
    Takeover,
    Request,
;
```

enum EScoringMethod

```
#include <ScoringFactor.h>

enum EScoringMethod

    Partial,
    Exact,
;
```

enum ESendMethod

```
#include <MagesNetwork.h>

enum ESendMethod

    Reliable,
    Unreliable,
;
```

enum EUIType

```
#include <UIManagement.h>

enum EUIType

    Notification,
    Warning,
    Error,
    CriticalError,
;
```

enum EVisibilityLevel

```
#include <MagesHand.h>

enum EVisibilityLevel

    Invisible = 0,
    Ghost     = 70,
    Visible   = 100,
;
```

enum InheritTransformFrom

```
#include <BasePrototype.h>

enum InheritTransformFrom

    UMETA =(DisplayName = "none"),
;
```

enum NetKeyCode

```
#include <NetMessageClass.h>

enum NetKeyCode

    RequestAuthority            =100,
    Perform,
    Undo,
    SyncOperationState,
    FinilizePrefab,
    changeAlternativePathCustom,
    OperationDiff,
    ClientNumber,
    ClientMode,
    ObjectDestroy,
    stageNodeInteraction,
    StopJigNotifier,
    NextAidLine,
    PerformCombinedAction,
```

```
    SyncProperties,
    Instantiate,
    SerializeViewBatch,
    SelectTool,
    DeselectTool,
    SetPath,
    SyncQuestionButton,
;
```

enum OnPrefabDetachFeature

```
#include <InteractablePrefabConstructor.h>

enum OnPrefabDetachFeature

    UMETA =(DisplayName = "Generic"),
;
```

enum ParticleGroupMehod

```
#include <CreateDeformMesh.h>

enum ParticleGroupMehod

    distance                        = 0 UMETA(DisplayName = "distance"),
    closest_point                   = 1  UMETA(DisplayName = "closest_point"),
    distance_with_soft_interaction = 2     UMETA(DisplayName =␣
↪"distance_with_soft_interaction"),
;
```

enum PrefabActionOnPerform

```
#include <GenericPrefabConstructor.h>

enum PrefabActionOnPerform

;
```

enum PrefabInteractableType

```
#include <InteractablePrefabConstructor.h>

enum PrefabInteractableType

;
```

enum PrefabType

```
#include <GenericPrefabConstructor.h>

enum PrefabType

    UMETA =(DisplayName = "Destroy"),
;
```

enum PumpMode

```
#include <PumpPrefabConstructor.h>

enum PumpMode

    FullPump,
    HalfPump,
;
```

enum ToolFlashType

```
#include <GestureHands.h>

enum ToolFlashType

    Error,
    VisualAid,
;
```

enum ToolGrabbingType

```
#include <GestureHands.h>

enum ToolGrabbingType

    Pinch,
    Grab,
;
```

enum ToolRotationAxis

```
#include <GestureHands.h>

enum ToolRotationAxis

    forward,
    backward,
    left,
    right,
    up,
    down,
```

```
    none,
;
```

enum ToolTriggerButton

```
#include <GestureHands.h>

enum ToolTriggerButton

    Grip,
    Trigger,
;
```

enum UseColliderTrigger

```
#include <UseColliderPrefabConstructor.h>

enum UseColliderTrigger

    simple,
    usewithtool,
    hit,
;
```

struct FActionAnalyticsData

```
#include <ActionAnalyticsData.h>

struct FActionAnalyticsData

    // methods

    GENERATED_USTRUCT_BODY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
;
```

struct FActionGroup

```
#include <MagesLiveScenegraphSubsystem.h>

struct FActionGroup

    // fields

    bool IsParallel = false;
    TArray<FMagesActionNode*> Actions;
;
```

struct FActionSummary

```
#include <UIManagement.h>

struct FActionSummary

    // fields

    float ActionTime;
    int Score;
    int Errors;
;
```

struct FAnalyticsErrorData

```
#include <MagesAnalyticsAsset.h>

struct FAnalyticsErrorData

    // methods

    GENERATED_USTRUCT_BODY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
;
```

struct FAnalyticsTimeData

```
#include <MagesAnalyticsAsset.h>

struct FAnalyticsTimeData

    // methods

    GENERATED_USTRUCT_BODY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
;
```

struct FAnimationGroup

```
#include <AnimationAction.h>

struct FAnimationGroup

    // fields

    FSpawnActorDesc GrabbableActor;
    TEnumAsByte<InheritTransformFrom> InheritGrabbable;
;
```

struct FAudioAsset

```
#include <MagesSettings.h>

struct FAudioAsset

    // fields

    FSoftObjectPath Asset;
    float Volume = 1.0f;
;
```

struct FAvoidObjectsData

```
#include <ActionAnalyticsData.h>

struct FAvoidObjectsData

    // methods

    GENERATED_USTRUCT_BODY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
```

```
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
;
```

struct FDrillCategorizeFacesDesc

```
struct FDrillCategorizeFacesDesc

    // fields

    FLineSegment Line;
    float Radius;
    TArray<int32>& Inside;
    TArray<int32>& Outside;
    TArray<int32>& Boundary;
;
```

struct FDrillEvalIntersectionDesc

```
struct FDrillEvalIntersectionDesc

    // fields

    FLineSegment Line;
    float Radius;
    int32 I;
    int32 J;
    int32& Count;
    TArray<FVector2D>& VChecked;
    TArray<FQuadruplet>& VRelation;
    TArray<FVector>& Intersections;
    TArray<int32>& AffectedIds;
;
```

struct FDrillSplitCertainFacesDesc

```
struct FDrillSplitCertainFacesDesc

    // fields

    FVector A;
    FVector B;
    float Radius;
    TArray<int32>& AffectedIds;
    TArray<int32>& NewAffectedIds;
    TArray<int32>& NewIdsInside;
;
```

struct FDuo

```
struct FDuo

    // fields

    int32 A;
    int32 B;
;
```

struct FErrorsStayData

```
#include <ActionAnalyticsData.h>

struct FErrorsStayData

    // methods

    GENERATED_USTRUCT_BODY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
;
```

struct FGenerateTearSegmentTrianglesDesc

```
struct FGenerateTearSegmentTrianglesDesc

    // fields

    int32 IndexI1;
    int32 IndexI2;
    int32 IndexMiddle0;
    int32 IndexMiddle1;
;
```

struct FHoloGroup

```
#include <BasePrototype.h>

struct FHoloGroup

    // fields

    FSpawnActorDesc Hologram;
    FString holoPath;
    AActor* holoParent;
    AActor* holoObject;
;
```

struct FIActionGroup

```
#include <ParallelAction.h>

struct FIActionGroup

    // fields

    int numberOfPath;
    IIAction* iAction;
;
```

struct FInsertGroup

```
#include <InsertAction.h>

struct FInsertGroup

    // fields

    FSpawnActorDesc InsertActorDesc;
    FSpawnActorDesc FinalActorDesc;
    InheritTransformFrom inheritGrabbable;
    InheritTransformFrom inheritFinal;
;
```

struct FLineSegment

```
#include <MagesMathUtility.h>

struct FLineSegment

    // fields

    FVector Start;
    FVector End;

    // methods
```

```
    FORCEINLINE FVector GetVector() const;
;
```

struct FMagesActionAnalyticsEntry

```
#include <AnalyticsManager.h>

struct FMagesActionAnalyticsEntry

    // fields

    TMap<int, TArray<UScoringFactor*>> ComponentsReference;
    TMap<int, TArray<UScoringFactor*>> CustomScoringFactors;
    TMap<int, TArray<FScoringFactorData>> CurrentActionStats;
    TMap<FString, float> Results;
    TMap<FString, int> Errors;
    float Score;

    // methods

    FORCEINLINE void Clear();
;
```

struct FMagesActionPath

```
#include <MagesLiveScenegraphSubsystem.h>

struct FMagesActionPath

    // fields

    TArray<FMagesActionPathEntry> Entries;
;
```

struct FMagesActionPathEntry

```
#include <MagesLiveScenegraphSubsystem.h>

struct FMagesActionPathEntry

    // fields

    FString ActionName;

    // construction

    FMagesActionPathEntry(FString ActionName);
;
```

struct FMagesPrepareTearResult

```
#include <MagesTearComponent.h>

struct FMagesPrepareTearResult

    // fields

    FPointNormalPlane TearPlane;
    FPointNormalPlane PerpendicularPlane1;
    FPointNormalPlane PerpendicularPlane2;
    FVector I1;
    FVector I2;
    int Face_I1 = 0;
    int Face_I2 = 0;
    bool Found0 = false;
    bool Found1 = false;
;
```

struct FMagesUserEventAssetEntry

```
#include <MagesUserEventAsset.h>

struct FMagesUserEventAssetEntry

    // fields

    FString EventName;
    TSubclassOf<UObject> EventDataObjectClass;
    TArray<FOnMagesUserEvent> EventDelegates;
;
```

struct FNamedTimeProfileContext

```
#include <Profiling.h>

struct FNamedTimeProfileContext: public FTimeProfileContext

    // fields

    const TCHAR* Name;

    // construction

    FNamedTimeProfileContext(const TCHAR* Name);
;
```

Inherited Members

```
public:
    // fields

    double DiffSeconds;
    double DiffMilliseconds;
    uint64 DiffCycles;

    // methods

    FORCEINLINE void Begin(void);
    FORCEINLINE void End(void);
```

struct FOrientationContext

```
#include <MagesGameplayUtility.h>

struct FOrientationContext

    // fields

    FQuat Orientation;
    FVector2D MovementContext;
;
```

struct FPostCheckoutProduct

```
#include <PostCheckoutProduct.h>

struct FPostCheckoutProduct

    // fields

    FString ProductName;
;
```

struct FPumpGroup

```
#include <PumpAction.h>

struct FPumpGroup

    // fields

    FSpawnActorDesc Actor;
    FORamaVR performActionFunction;
;
```

struct FQuestionData

```
#include <ActionAnalyticsData.h>

struct FQuestionData

    // methods

    GENERATED_USTRUCT_BODY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
;
```

struct FQuestionOptionData

Overview

```
#include <QuestionOptionData.h>

struct FQuestionOptionData

    // fields

    bool isCorrect;
    FString optionText;
    FString orderOfAnswer;
    FString questionMessageUser;
;
```

Detailed Documentation

Fields

`bool isCorrect`

Set to true if is the correct option.

`FString optionText`

Set option's text.

`FString orderOfAnswer`

Set the order of the answer with a number e.x (1). Make Sure that the variable 'Answers With Order' is true.

`FString questionMessageUser`

Set option's user name. Only if you want the name to appear at the beginning of the text.

struct FQuestionScoringData

```
#include <MagesAnalyticsAsset.h>

struct FQuestionScoringData

    // methods

    GENERATED_USTRUCT_BODY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
;
```

struct FRaiseEventBatch

```
#include <MagesNetworkClasses.h>

struct FRaiseEventBatch

    // fields

    uint8 Group;
    bool Reliable;
;
```

struct FRecalculateNormalsDesc

```
struct FRecalculateNormalsDesc

    // fields

    UMagesDeformableMeshData* Data;
    float Angle;
;
```

struct FRecalculateNormalsVertexEntry

```
struct FRecalculateNormalsVertexEntry

    // fields

    int32 TriangleIndex;
    int32 VertexIndex;
;
```

struct FRecalculateNormalsVertexKey

Recalculation for Normals, Tangents Code Adapted from: https://gist.github.com/unitycoder/81888c54f87b56113f17a5c8eb6bb32b

```
struct FRecalculateNormalsVertexKey

    // fields

    int64 X;
    int64 Y;
    int64 Z;
    static constexpr int32 Tolerance = 100000;
    static constexpr int64 FNV32Init = 0x811c9dc5;
    static constexpr int64 FNV32Prime = 0x01000193;

    // construction

    FRecalculateNormalsVertexKey(FVector Position);
;
```

struct FRemoveGroup

```
#include <RemoveAction.h>

struct FRemoveGroup

    // fields

    FSpawnActorDesc Grabbable;
    FSpawnActorDesc Tool;
;
```

struct FReplicaCacheEntry

```
#include <ReplicaCache.h>

struct FReplicaCacheEntry

    // fields

    class AActor* Instance;
    FString BlueprintPath;
    int ViewID;
;
```

struct FRetriangulateInitialTearPointsDesc

```
struct FRetriangulateInitialTearPointsDesc

    // fields

    TArray<int32>* NewTriangles;
    TArray<FVector>* NewVertices;
    TArray<FTriplet>* VRelation;
    TArray<int32>* FacesToBeRemoved;
    TArray<int32>* OuterTearPoints;
    TArray<FVector>* OuterTearVertices;
    FPointNormalPlane Plane;
    int32 Down;
    int32 Up;
    int32* L;
;
```

struct FRoomInfo

```
#include <PhotonLBClient.h>

struct FRoomInfo

    // fields

    FString Name;
    int PlayerCount;
;
```

struct FScoringFactorData

```
#include <AnalyticsObjects.h>

struct FScoringFactorData

    // methods

    GENERATED_BODY();
    UPROPERTY(BlueprintReadWrite, Category = "Mages");
    UPROPERTY(BlueprintReadWrite, Category = "Mages");
    UPROPERTY(BlueprintReadWrite, Category = "Mages");
    UPROPERTY(BlueprintReadWrite, Category = "Mages");
    UPROPERTY(BlueprintReadWrite, Category = "Mages");
    UPROPERTY(BlueprintReadWrite, Category = "Mages");
;
```

struct FScoringFactorRef

```
#include <ActionProperties.h>

struct FScoringFactorRef

    // fields

    TArray<UScoringFactor*> ScoringFactors;
;
```

struct FSerializationMessageInfo

```
#include <MagesNetworkClasses.h>

struct FSerializationMessageInfo

    // fields

    int Timestamp;
    int NumObjectUpdates;
;
```

struct FSerializeViewBatch

```
#include <MagesNetworkClasses.h>

struct FSerializeViewBatch

    // fields

    FRaiseEventBatch Batch;
    int Offset;
    uint32 NumObjectUpdates;
    TArray<uint8> Updates;
    static const int ObjectsInOneUpdate = 20;

    // construction

    FSerializeViewBatch(FRaiseEventBatch NewBatch, int NewOffset);

    // methods

    void Add(const TArray<uint8>& ObjectUpdateData);
    void Clear();
;
```

struct FSpawnActorDesc

```
#include <BasePrototype.h>

struct FSpawnActorDesc

    // fields

    TSubclassOf<AActor> Class = nullptr;
    AActor* AttachActor = nullptr;
    USceneComponent* AttachParent = nullptr;
    FName SocketName = NAME_None;
    AActor* Instance = nullptr;
;
```

struct FTimeData

```
#include <ActionAnalyticsData.h>

struct FTimeData

    // methods

    GENERATED_USTRUCT_BODY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
;
```

struct FTimeProfileContext

```
#include <Profiling.h>

struct FTimeProfileContext

    // fields

    double DiffSeconds;
    double DiffMilliseconds;
    uint64 DiffCycles;

    // methods

    FORCEINLINE void Begin(void);
    FORCEINLINE void End(void);
;

// direct descendants

struct FNamedTimeProfileContext;
```

struct FUserCredentials

```
#include <CredentialsManager.h>

struct FUserCredentials

    // fields

    FString Username;
    FString Password;
    bool WereLoaded;
;
```

struct FVelocityData

```
#include <ActionAnalyticsData.h>

struct FVelocityData

    // methods

    GENERATED_USTRUCT_BODY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
;
```

struct FVelocityScoringData

```
#include <MagesAnalyticsAsset.h>

struct FVelocityScoringData

    // methods

    GENERATED_USTRUCT_BODY();
    UPROPERTY();
    virtual UPROPERTY() = 0;
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
    UPROPERTY();
;
```

struct LocalPlayer

```
#include <LoadBalancingListener.h>

struct LocalPlayer

    // fields

    int x;
    int y;
    int color;
    unsigned long lastUpdateTime;
;
```

class AAction

```
#include <Action.h>

class AAction: public AActor

public:
    // methods

    virtual void Tick(float DeltaTime);
;
```

class AActorNode

```
#include <ActorNode.h>

class AActorNode: public AActor

public:
    // methods

    void AddChild(AActor* Child);
    int GetChildIndex(AActor* Child);
    int GetChildCount();
    AActor* GetChild(int I);
    bool Remove(int Index);
    bool SetNodeIndex(int NodeIndex, int TargetIndex);
;
```

class AAnalyticsExporter

```
#include <AnalyticsExporter.h>

class AAnalyticsExporter: public AActor

public:
    // methods

    void Init();
    virtual void Tick(float DeltaTime);
    void ExportAnalytics();
    void Upload();
;
```

class AAnalyticsManager

```
#include <AnalyticsManager.h>

class AAnalyticsManager: public AActor

public:
    // fields

    int CurrentSubActionInitialized =-1;
    TMap<FString, float> Results;
    TMap<FString, int> Errors;
    TMap<int, TArray<UScoringFactor*>> ComponentsReference;
    TMap<int, TArray<FScoringFactorData>> currentActionStats;
    float CurrentScore;
    int LastIdx =0;
    UActionProperties* actionProperties;
    TMap<int, TArray<UScoringFactor*>> CustomScoringFactors;

    // methods

    float GetWeightFromEnum(EFactorImportance enumWeight);
    virtual void Tick(float DeltaTime);
    void InitializeAction(class ABasePrototype* ActionInstance, const␣
↪FString& Name);

    void Perform(
        const FString& ActionName,
        ABasePrototype* Action,
        bool skipped = false
        );

    void SubPerform(int Index);
    void Undo(ABasePrototype* Action);
    void DebugLog();

    void AddScoringFactor(
        TSubclassOf<class UScoringFactor> ScoringFactor,
```

```
        int SubAction = 0
        );
;
```

class AAnimationAction

```
#include <AnimationAction.h>

class AAnimationAction: public ABasePrototype

public:
    // methods

    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();

    void SetAnimationPrefab(
        TSubclassOf<AActor> Blueprint,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachComponent = nullptr,
        FName SocketName = NAME_None
        );
;

// direct descendants

class ABPAnimationAction;
```

Inherited Members

```
public:
    // classes

    class InstrumentTransforms;

    // fields

    TArray<FORamaVR> afterSpawnFunctions;
    TMap<AActor*, UAnimSequence*> CharacterAnimationsMap;
    TMap<AActor*, UAnimSequence*> CharacterUndoAnimations;
    TArray<AActor*> BoundActorCharacterAnimated;
    USceneComponent* _rootComp;
    int OverrideSetPath = -1;
    int AlternativePath = -1;
    FString ActionName = TEXT("");

    // methods

    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
```

```
    void SetActionNode();
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
    virtual void DifficultyRestrictions() = 0;
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    virtual void InitializeHolograms();
    void SetAfterSpawn(FORamaVR afterSpawnFunction);
    void SetUndoAction(FORamaVR undoActionFunction);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void BindCharacterAnimationToPerform(
        TSubclassOf<AActor> Actor,
        UAnimSequence* seq
        );

    void BindCharacterAnimationOnUndo(TSubclassOf<AActor> Actor,␣
↪UAnimSequence* seq);
    virtual void DifficultyRestrictions();
    virtual void ActionCall();
    void SetEventListener(FString _event);

    void SetHologramObject(
        TSubclassOf<AActor> Class,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachParent = nullptr,
        FName SocketName = NAME_None
        );

    virtual void SetNextModule(FORamaVR action);
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet);
    FString GetClassName() const;
```

class ABPAnimationAction

```
#include <BPAnimationAction.h>

class ABPAnimationAction: public AAnimationAction

public:
    // methods

    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    void InitializeBP();
    void PerformBP();
    void UndoBP();
;
```

Inherited Members

```
public:
    // classes

    class InstrumentTransforms;

    // fields

    TArray<FORamaVR> afterSpawnFunctions;
    TMap<AActor*, UAnimSequence*> CharacterAnimationsMap;
    TMap<AActor*, UAnimSequence*> CharacterUndoAnimations;
    TArray<AActor*> BoundActorCharacterAnimated;
    USceneComponent* _rootComp;
    int OverrideSetPath = -1;
    int AlternativePath = -1;
    FString ActionName = TEXT("");

    // methods

    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
    void SetActionNode();
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
    virtual void DifficultyRestrictions() = 0;
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
```

```
    virtual void InitializeHolograms();
    void SetAfterSpawn(FORamaVR afterSpawnFunction);
    void SetUndoAction(FORamaVR undoActionFunction);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void BindCharacterAnimationToPerform(
        TSubclassOf<AActor> Actor,
        UAnimSequence* seq
        );

    void BindCharacterAnimationOnUndo(TSubclassOf<AActor> Actor,␣
↪UAnimSequence* seq);
    virtual void DifficultyRestrictions();
    virtual void ActionCall();
    void SetEventListener(FString _event);

    void SetHologramObject(
        TSubclassOf<AActor> Class,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachParent = nullptr,
        FName SocketName = NAME_None
        );

    virtual void SetNextModule(FORamaVR action);
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet);
    FString GetClassName() const;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();

    void SetAnimationPrefab(
        TSubclassOf<AActor> Blueprint,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachComponent = nullptr,
        FName SocketName = NAME_None
        );
```

class ABPBasePrototype

```
#include <BPBasePrototype.h>

class ABPBasePrototype: public ABasePrototype

public:
    // methods

    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
```

```
    void InitializeBP();
    void PerformBP();
    void UndoBP();
;
```

Inherited Members

```
public:
    // classes

    class InstrumentTransforms;

    // fields

    TArray<FORamaVR> afterSpawnFunctions;
    TMap<AActor*, UAnimSequence*> CharacterAnimationsMap;
    TMap<AActor*, UAnimSequence*> CharacterUndoAnimations;
    TArray<AActor*> BoundActorCharacterAnimated;
    USceneComponent* _rootComp;
    int OverrideSetPath = -1;
    int AlternativePath = -1;
    FString ActionName = TEXT("");

    // methods

    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
    void SetActionNode();
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
    virtual void DifficultyRestrictions() = 0;
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    virtual void InitializeHolograms();
    void SetAfterSpawn(FORamaVR afterSpawnFunction);
    void SetUndoAction(FORamaVR undoActionFunction);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void BindCharacterAnimationToPerform(
        TSubclassOf<AActor> Actor,
        UAnimSequence* seq
```

```
        );

    void BindCharacterAnimationOnUndo(TSubclassOf<AActor> Actor,␣
↪UAnimSequence* seq);
    virtual void DifficultyRestrictions();
    virtual void ActionCall();
    void SetEventListener(FString _event);

    void SetHologramObject(
        TSubclassOf<AActor> Class,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachParent = nullptr,
        FName SocketName = NAME_None
        );

    virtual void SetNextModule(FORamaVR action);
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet);
    FString GetClassName() const;
```

class ABPCombinedAction

```
#include <BPCombinedAction.h>

class ABPCombinedAction: public ACombinedAction

public:
    // methods

    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    void InitializeBP();
    void PerformBP();
    void UndoBP();
;
```

Inherited Members

```
public:
    // classes

    class InstrumentTransforms;

    // fields

    TArray<FORamaVR> afterSpawnFunctions;
    TMap<AActor*, UAnimSequence*> CharacterAnimationsMap;
    TMap<AActor*, UAnimSequence*> CharacterUndoAnimations;
    TArray<AActor*> BoundActorCharacterAnimated;
    USceneComponent* _rootComp;
    int OverrideSetPath = -1;
    int AlternativePath = -1;
```

```
    FString ActionName = TEXT("");

    // methods

    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
    void SetActionNode();
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
    virtual void DifficultyRestrictions() = 0;
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    virtual void InitializeHolograms();
    void SetAfterSpawn(FORamaVR afterSpawnFunction);
    void SetUndoAction(FORamaVR undoActionFunction);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void BindCharacterAnimationToPerform(
        TSubclassOf<AActor> Actor,
        UAnimSequence* seq
        );

    void BindCharacterAnimationOnUndo(TSubclassOf<AActor> Actor,␣
↪UAnimSequence* seq);
    virtual void DifficultyRestrictions();
    virtual void ActionCall();
    void SetEventListener(FString _event);

    void SetHologramObject(
        TSubclassOf<AActor> Class,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachParent = nullptr,
        FName SocketName = NAME_None
        );

    virtual void SetNextModule(FORamaVR action);
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet);
    FString GetClassName() const;
    virtual void SetNextModule(FORamaVR action);
    void InsertIActions(TArray<ABasePrototype*> iActions);
    void InsertIActions(TArray<IIAction*> iActions);
    UFUNCTION();
```

```
    UFUNCTION();
    UFUNCTION();
    UFUNCTION();
    void SetEventListenerCombined(FString _event);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void SetUndoAction(FORamaVR undoActionFunction);
    virtual void DifficultyRestrictions();
    int GetCurrentSubActionName();
    int GetNumberOfSubActions();
```

class ABPInsertAction

```
#include <BPInsertAction.h>

class ABPInsertAction: public AInsertAction

public:
    // methods

    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    void InitializeBP();
    void PerformBP();
    void UndoBP();
;
```

Inherited Members

```
public:
    // classes

    class InstrumentTransforms;

    // fields

    TArray<FORamaVR> afterSpawnFunctions;
    TMap<AActor*, UAnimSequence*> CharacterAnimationsMap;
    TMap<AActor*, UAnimSequence*> CharacterUndoAnimations;
    TArray<AActor*> BoundActorCharacterAnimated;
    USceneComponent* _rootComp;
    int OverrideSetPath = -1;
    int AlternativePath = -1;
    FString ActionName = TEXT("");

    // methods
```

```
    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
    void SetActionNode();
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
    virtual void DifficultyRestrictions() = 0;
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    virtual void InitializeHolograms();
    void SetAfterSpawn(FORamaVR afterSpawnFunction);
    void SetUndoAction(FORamaVR undoActionFunction);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void BindCharacterAnimationToPerform(
        TSubclassOf<AActor> Actor,
        UAnimSequence* seq
        );

    void BindCharacterAnimationOnUndo(TSubclassOf<AActor> Actor,␣
↪UAnimSequence* seq);
    virtual void DifficultyRestrictions();
    virtual void ActionCall();
    void SetEventListener(FString _event);

    void SetHologramObject(
        TSubclassOf<AActor> Class,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachParent = nullptr,
        FName SocketName = NAME_None
        );

    virtual void SetNextModule(FORamaVR action);
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet);
    FString GetClassName() const;
    virtual void Initialize();
    virtual void Undo();
    virtual void Perform();

    void SetInsertPrefab(
        TSubclassOf<AActor> InsertBlueprint,
        TSubclassOf<AActor> FinalBlueprint,
        AActor* InsertBlueprintAttachActor = nullptr,
```

```
        USceneComponent* InsertBlueprintAttachComponent = nullptr,
        FName InsertBlueprintSocketName = NAME_None,
        AActor* FinalBlueprintAttachActor = nullptr,
        USceneComponent* FinalBlueprintAttachComponent = nullptr,
        FName FinalBlueprintSocketName = NAME_None
        );
```

class ABPParallelAction

```
#include <BPParallelAction.h>

class ABPParallelAction: public AParallelAction

public:
    // methods

    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    void InitializeBP();
    void PerformBP();
    void UndoBP();
;
```

Inherited Members

```
public:
    // classes

    class InstrumentTransforms;

    // fields

    TArray<FORamaVR> afterSpawnFunctions;
    TMap<AActor*, UAnimSequence*> CharacterAnimationsMap;
    TMap<AActor*, UAnimSequence*> CharacterUndoAnimations;
    TArray<AActor*> BoundActorCharacterAnimated;
    USceneComponent* _rootComp;
    int OverrideSetPath = -1;
    int AlternativePath = -1;
    FString ActionName = TEXT("");
    TMap<int, FIActionGroup> iActionMap;

    // methods

    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
    void SetActionNode();
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
```

```
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
    virtual void DifficultyRestrictions() = 0;
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    virtual void InitializeHolograms();
    void SetAfterSpawn(FORamaVR afterSpawnFunction);
    void SetUndoAction(FORamaVR undoActionFunction);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void BindCharacterAnimationToPerform(
        TSubclassOf<AActor> Actor,
        UAnimSequence* seq
        );

    void BindCharacterAnimationOnUndo(TSubclassOf<AActor> Actor,␣
↪UAnimSequence* seq);
    virtual void DifficultyRestrictions();
    virtual void ActionCall();
    void SetEventListener(FString _event);

    void SetHologramObject(
        TSubclassOf<AActor> Class,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachParent = nullptr,
        FName SocketName = NAME_None
        );

    virtual void SetNextModule(FORamaVR action);
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet);
    FString GetClassName() const;
    UPROPERTY();
    void InsertActionToMap(int numberOfPath, ABasePrototype* iAction);
    void InsertIActionToMap(int numberOfPath, IIAction* iAction);
    UFUNCTION(BlueprintCallable, Category = "Mages|Actions");
    UFUNCTION();
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    virtual void DifficultyRestrictions();
    void OnSetPath(int PathNum);
```

class ABPPumpAction

```
#include <BPPumpAction.h>

class ABPPumpAction: public APumpAction

public:
    // methods

    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    void InitializeBP();
    void PerformBP();
    void UndoBP();
;
```

Inherited Members

```
public:
    // classes

    class InstrumentTransforms;

    // fields

    TArray<FORamaVR> afterSpawnFunctions;
    TMap<AActor*, UAnimSequence*> CharacterAnimationsMap;
    TMap<AActor*, UAnimSequence*> CharacterUndoAnimations;
    TArray<AActor*> BoundActorCharacterAnimated;
    USceneComponent* _rootComp;
    int OverrideSetPath = -1;
    int AlternativePath = -1;
    FString ActionName = TEXT("");

    // methods

    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
    void SetActionNode();
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
    virtual void DifficultyRestrictions() = 0;
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
```

```
    virtual void InitializeHolograms();
    void SetAfterSpawn(FORamaVR afterSpawnFunction);
    void SetUndoAction(FORamaVR undoActionFunction);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void BindCharacterAnimationToPerform(
        TSubclassOf<AActor> Actor,
        UAnimSequence* seq
        );

    void BindCharacterAnimationOnUndo(TSubclassOf<AActor> Actor,␣
↪UAnimSequence* seq);
    virtual void DifficultyRestrictions();
    virtual void ActionCall();
    void SetEventListener(FString _event);

    void SetHologramObject(
        TSubclassOf<AActor> Class,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachParent = nullptr,
        FName SocketName = NAME_None
        );

    virtual void SetNextModule(FORamaVR action);
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet);
    FString GetClassName() const;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();

    void SetPumpPrefab(
        TSubclassOf<AActor> Blueprint,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachComponent = nullptr,
        FName SocketName = NAME_None
        );

    void SetWaitForAllPumps(bool wait);
```

class ABPQuestionAction

```
#include <BPQuestionAction.h>

class ABPQuestionAction: public AQuestionAction

public:
    // methods

    virtual void Initialize();
```

```
    virtual void Perform();
    virtual void Undo();
    void InitializeBP();
    void PerformBP();
    void UndoBP();
;
```

Inherited Members

```
public:
    // classes

    class InstrumentTransforms;

    // fields

    TArray<FORamaVR> afterSpawnFunctions;
    TMap<AActor*, UAnimSequence*> CharacterAnimationsMap;
    TMap<AActor*, UAnimSequence*> CharacterUndoAnimations;
    TArray<AActor*> BoundActorCharacterAnimated;
    USceneComponent* _rootComp;
    int OverrideSetPath = -1;
    int AlternativePath = -1;
    FString ActionName = TEXT("");
    bool ready;

    // methods

    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
    void SetActionNode();
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
    virtual void DifficultyRestrictions() = 0;
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    virtual void InitializeHolograms();
    void SetAfterSpawn(FORamaVR afterSpawnFunction);
    void SetUndoAction(FORamaVR undoActionFunction);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );
```

```
    void BindCharacterAnimationToPerform(
        TSubclassOf<AActor> Actor,
        UAnimSequence* seq
        );

    void BindCharacterAnimationOnUndo(TSubclassOf<AActor> Actor,␣
↪UAnimSequence* seq);
    virtual void DifficultyRestrictions();
    virtual void ActionCall();
    void SetEventListener(FString _event);

    void SetHologramObject(
        TSubclassOf<AActor> Class,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachParent = nullptr,
        FName SocketName = NAME_None
        );

    virtual void SetNextModule(FORamaVR action);
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet);
    FString GetClassName() const;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    void SetQuestionPrefab(TSubclassOf<AActor> Class);
```

class ABPRemoveAction

```
#include <BPRemoveAction.h>

class ABPRemoveAction: public ARemoveAction

public:
    // methods

    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    void InitializeBP();
    void PerformBP();
    void UndoBP();
;
```

Inherited Members

```
public:
    // classes

    class InstrumentTransforms;

    // fields

    TArray<FORamaVR> afterSpawnFunctions;
    TMap<AActor*, UAnimSequence*> CharacterAnimationsMap;
    TMap<AActor*, UAnimSequence*> CharacterUndoAnimations;
    TArray<AActor*> BoundActorCharacterAnimated;
    USceneComponent* _rootComp;
    int OverrideSetPath = -1;
    int AlternativePath = -1;
    FString ActionName = TEXT("");

    // methods

    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
    void SetActionNode();
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
    virtual void DifficultyRestrictions() = 0;
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    virtual void InitializeHolograms();
    void SetAfterSpawn(FORamaVR afterSpawnFunction);
    void SetUndoAction(FORamaVR undoActionFunction);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void BindCharacterAnimationToPerform(
        TSubclassOf<AActor> Actor,
        UAnimSequence* seq
        );

    void BindCharacterAnimationOnUndo(TSubclassOf<AActor> Actor,␣
↪UAnimSequence* seq);
    virtual void DifficultyRestrictions();
    virtual void ActionCall();
```

```
    void SetEventListener(FString _event);

    void SetHologramObject(
        TSubclassOf<AActor> Class,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachParent = nullptr,
        FName SocketName = NAME_None
        );

    virtual void SetNextModule(FORamaVR action);
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet);
    FString GetClassName() const;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();

    void SetRemovePrefab(
        TSubclassOf<AActor> RemoveBlueprint,
        TSubclassOf<AActor> ToolBlueprint,
        AActor* RemoveAttachActor = nullptr,
        USceneComponent* RemoveAttachComponent = nullptr,
        FName RemoveSocketName = NAME_None,
        AActor* ToolAttachActor = nullptr,
        USceneComponent* ToolAttachComponent = nullptr,
        FName ToolSocketName = NAME_None
        );

    FRemoveGroup SetRemovePrefabLong(
        FString grabbablePrefabPath,
        FString removePrefabPath,
        AActor* removePrefabParent = nullptr,
        FString _toolPrefabPath = "",
        InheritTransformFrom _inheritTransformGrabbale = none,
        InheritTransformFrom _inheritTransformRemove = none
        );
```

class ABPUseAction

```
#include <BPUseAction.h>

class ABPUseAction: public AUseAction

public:
    // methods

    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    void InitializeBP();
    void PerformBP();
    void UndoBP();
;
```

Inherited Members

```
public:
    // classes

    class InstrumentTransforms;

    // fields

    TArray<FORamaVR> afterSpawnFunctions;
    TMap<AActor*, UAnimSequence*> CharacterAnimationsMap;
    TMap<AActor*, UAnimSequence*> CharacterUndoAnimations;
    TArray<AActor*> BoundActorCharacterAnimated;
    USceneComponent* _rootComp;
    int OverrideSetPath = -1;
    int AlternativePath = -1;
    FString ActionName = TEXT("");

    // methods

    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
    void SetActionNode();
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
    virtual void DifficultyRestrictions() = 0;
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    virtual void InitializeHolograms();
    void SetAfterSpawn(FORamaVR afterSpawnFunction);
    void SetUndoAction(FORamaVR undoActionFunction);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void BindCharacterAnimationToPerform(
        TSubclassOf<AActor> Actor,
        UAnimSequence* seq
        );

    void BindCharacterAnimationOnUndo(TSubclassOf<AActor> Actor,␣
↪UAnimSequence* seq);
    virtual void DifficultyRestrictions();
    virtual void ActionCall();
```

```
    void SetEventListener(FString _event);

    void SetHologramObject(
        TSubclassOf<AActor> Class,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachParent = nullptr,
        FName SocketName = NAME_None
        );

    virtual void SetNextModule(FORamaVR action);
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet);
    FString GetClassName() const;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();

    void SetUsePrefab(
        TSubclassOf<AActor> UseBlueprint,
        TSubclassOf<AActor> ColliderBlueprint,
        bool UsePrefabAlreadyExists = false,
        AActor* UseAttachActor = nullptr,
        USceneComponent* UseAttachComponent = nullptr,
        FName UseSocketName = NAME_None,
        AActor* ColliderAttachActor = nullptr,
        USceneComponent* ColliderAttachComponent = nullptr,
        FName ColliderSocketName = NAME_None
        );
```

class ABasePrototype

class ABasePrototype::InstrumentTransforms

class ABasePrototype::InstrumentTransforms::CustomTransform

```
#include <BasePrototype.h>

class CustomTransform

public:
    // fields

    FVector position;
    FQuat rotation;
;
#include <BasePrototype.h>

class InstrumentTransforms

public:
    // classes

    class CustomTransform;
```

```
    // fields

    static CustomTransform grabbablePrefab;
    static CustomTransform finalPrefab;
    static CustomTransform removePrefab;

    // methods

    static void CopyTransformFrom(AActor* from, InheritTransformFrom to);
;
#include <BasePrototype.h>

class ABasePrototype:
    public AActor,
    public IIAction

public:
    // classes

    class InstrumentTransforms;

    // fields

    TArray<FORamaVR> afterSpawnFunctions;
    TMap<AActor*, UAnimSequence*> CharacterAnimationsMap;
    TMap<AActor*, UAnimSequence*> CharacterUndoAnimations;
    TArray<AActor*> BoundActorCharacterAnimated;
    USceneComponent* _rootComp;
    int OverrideSetPath = -1;
    int AlternativePath = -1;
    FString ActionName = TEXT("");

    // methods

    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    virtual void InitializeHolograms();
    void SetAfterSpawn(FORamaVR afterSpawnFunction);
    void SetUndoAction(FORamaVR undoActionFunction);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void BindCharacterAnimationToPerform(
        TSubclassOf<AActor> Actor,
        UAnimSequence* seq
        );

    void BindCharacterAnimationOnUndo(TSubclassOf<AActor> Actor,␣
```

```
↪UAnimSequence* seq);
    virtual void DifficultyRestrictions();
    virtual void ActionCall();
    void SetEventListener(FString _event);

    void SetHologramObject(
        TSubclassOf<AActor> Class,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachParent = nullptr,
        FName SocketName = NAME_None
        );

    virtual void SetNextModule(FORamaVR action);
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet);
    FString GetClassName() const;
;

// direct descendants

class AAnimationAction;
class ABPBasePrototype;
class ACombinedAction;
class AInsertAction;
class AParallelAction;
class APumpAction;
class AQuestionAction;
class ARemoveAction;
class AUseAction;
```

Inherited Members

```
public:
    // methods

    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
    void SetActionNode();
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
    virtual void DifficultyRestrictions() = 0;
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
```

class ACameraRigInputController

struct ACameraRigInputController::ButtonState

```
struct ButtonState

    // fields

    float Axis;
    bool Down;
;
```

struct ACameraRigInputController::HapticEffectState

```
struct HapticEffectState

    // fields

    float TimeElapsed = 0.f;
    float TimeSet = 0.f;
    bool IsPlaying = false;
;
```

Overview

```
#include <CameraRigInputController.h>

class ACameraRigInputController:
    public APawn,
    public IDeviceControllerInterface

public:
    // structs

    struct ButtonState;
    struct HapticEffectState;

    // fields

    float Speed = 1.2f;
    UCameraComponent* CameraHead;

    // methods

    virtual void Tick(float DeltaTime);
    virtual void SetupPlayerInputComponent(class UInputComponent*␣
↪InputComponent);
    bool GetTriggerPressed_Implementation(const EOvidVRHand& HandType);
    bool GetGripPressed_Implementation(const EOvidVRHand& HandType);
    bool GetIsGrabbed_Implementation(const EOvidVRHand& HandType);
    float GetGripStrength_Implementation(const EOvidVRHand& HandType);
    float GetPinchStrength_Implementation(const EOvidVRHand& HandType);
```

```
    bool IsMoving_Implementation(const EOvidVRHand& HandType);

    void PlayHapticPulse_Implementation(
        const EOvidVRHand& HandType,
        float Strength,
        float Frequency,
        float Duration
        );
;
```

Inherited Members

```
public:
    // methods

    bool GetTriggerPressed(const EOvidVRHand& HandType);
    bool GetGripPressed(const EOvidVRHand& HandType);
    bool GetIsGrabbed(const EOvidVRHand& HandType);
    float GetGripStrength(const EOvidVRHand& HandType);
    float GetPinchStrength(const EOvidVRHand& HandType);
    bool IsMoving(const EOvidVRHand& HandType);

    void PlayHapticPulse(
        const EOvidVRHand& HandType,
        float Strength,
        float Frequency = 0.5f,
        float Duration = 0.7f
        );
```

Detailed Documentation

Fields

`float Speed = 1.2f`

Set Translation Speed.

`UCameraComponent* CameraHead`

Set Camera head for translation to be affected by it's forward vector Camera Rig > Camera (head) > Camera (eye)

class ACombinedAction

```
#include <CombinedAction.h>

class ACombinedAction: public ABasePrototype

public:
    // methods

    virtual void SetNextModule(FORamaVR action);
    void InsertIActions(TArray<ABasePrototype*> iActions);
```

```
    void InsertIActions(TArray<IIAction*> iActions);
    UFUNCTION();
    UFUNCTION();
    UFUNCTION();
    UFUNCTION();
    void SetEventListenerCombined(FString _event);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void SetUndoAction(FORamaVR undoActionFunction);
    virtual void DifficultyRestrictions();
    int GetCurrentSubActionName();
    int GetNumberOfSubActions();
;

// direct descendants

class ABPCombinedAction;
```

Inherited Members

```
public:
    // classes

    class InstrumentTransforms;

    // fields

    TArray<FORamaVR> afterSpawnFunctions;
    TMap<AActor*, UAnimSequence*> CharacterAnimationsMap;
    TMap<AActor*, UAnimSequence*> CharacterUndoAnimations;
    TArray<AActor*> BoundActorCharacterAnimated;
    USceneComponent* _rootComp;
    int OverrideSetPath = -1;
    int AlternativePath = -1;
    FString ActionName = TEXT("");

    // methods

    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
    void SetActionNode();
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
    virtual void DifficultyRestrictions() = 0;
```

```
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    virtual void InitializeHolograms();
    void SetAfterSpawn(FORamaVR afterSpawnFunction);
    void SetUndoAction(FORamaVR undoActionFunction);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void BindCharacterAnimationToPerform(
        TSubclassOf<AActor> Actor,
        UAnimSequence* seq
        );

    void BindCharacterAnimationOnUndo(TSubclassOf<AActor> Actor,␣
↪UAnimSequence* seq);
    virtual void DifficultyRestrictions();
    virtual void ActionCall();
    void SetEventListener(FString _event);

    void SetHologramObject(
        TSubclassOf<AActor> Class,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachParent = nullptr,
        FName SocketName = NAME_None
        );

    virtual void SetNextModule(FORamaVR action);
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet);
    FString GetClassName() const;
```

class ACreateDeformMesh

```
#include <CreateDeformMesh.h>

class ACreateDeformMesh: public AActor

public:
    // fields

    UStaticMeshComponent* StaticMesh = nullptr;
    FName CollisionProfileName = FName(TEXT("SoftBody"));
    UProceduralMeshComponent* center = nullptr;
    UStaticMesh* DeformableMesh = nullptr;
    USkeletalMeshComponent* SkeletalMesh = nullptr;
    ParticleGroupMehod groupMethod;
    bool bEnableDebug = false;
    TArray<ASoftParticleHelper*> particlesList;
```

```
    TArray<ASoftParticleHelper*> particlesArray;
    TArray<TArray<ASoftParticleHelper*>> particlesRotationMaping;
    TArray<FVector> TempVertices;
    TArray<FVector> Vertices;

    // methods

    virtual void Tick(float DeltaTime);
;
```

class AEventManager

class AEventManager::InvokeData

```
#include <EventManager.h>

class InvokeData

public:
    // fields

    FString keyBind;
    TArray<FString> functionNamesFromlisteners;

    // construction

    InvokeData(FString _key);

    // methods

    void AddFunctionNameFromListener(FString _funcName);
;
#include <EventManager.h>

class AEventManager: public AActor

public:
    // classes

    class InvokeData;

    // methods

    void Init();
    void Listening(FString eventName, FEventDelegate listener);
    void StopListeningAll(FString eventName);
    void TriggerEvent(FString eventName, float delaySec = 0.f);
;
```

class AInsertAction

```
#include <InsertAction.h>

class AInsertAction: public ABasePrototype

public:
    // methods

    virtual void Initialize();
    virtual void Undo();
    virtual void Perform();

    void SetInsertPrefab(
        TSubclassOf<AActor> InsertBlueprint,
        TSubclassOf<AActor> FinalBlueprint,
        AActor* InsertBlueprintAttachActor = nullptr,
        USceneComponent* InsertBlueprintAttachComponent = nullptr,
        FName InsertBlueprintSocketName = NAME_None,
        AActor* FinalBlueprintAttachActor = nullptr,
        USceneComponent* FinalBlueprintAttachComponent = nullptr,
        FName FinalBlueprintSocketName = NAME_None
        );
;

// direct descendants

class ABPInsertAction;
```

Inherited Members

```
public:
    // classes

    class InstrumentTransforms;

    // fields

    TArray<FORamaVR> afterSpawnFunctions;
    TMap<AActor*, UAnimSequence*> CharacterAnimationsMap;
    TMap<AActor*, UAnimSequence*> CharacterUndoAnimations;
    TArray<AActor*> BoundActorCharacterAnimated;
    USceneComponent* _rootComp;
    int OverrideSetPath = -1;
    int AlternativePath = -1;
    FString ActionName = TEXT("");

    // methods

    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
    void SetActionNode();
```

```
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
    virtual void DifficultyRestrictions() = 0;
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    virtual void InitializeHolograms();
    void SetAfterSpawn(FORamaVR afterSpawnFunction);
    void SetUndoAction(FORamaVR undoActionFunction);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void BindCharacterAnimationToPerform(
        TSubclassOf<AActor> Actor,
        UAnimSequence* seq
        );

    void BindCharacterAnimationOnUndo(TSubclassOf<AActor> Actor,␣
↪UAnimSequence* seq);
    virtual void DifficultyRestrictions();
    virtual void ActionCall();
    void SetEventListener(FString _event);

    void SetHologramObject(
        TSubclassOf<AActor> Class,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachParent = nullptr,
        FName SocketName = NAME_None
        );

    virtual void SetNextModule(FORamaVR action);
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet);
    FString GetClassName() const;
```

class AJSONParser

```
#include <JSONParser.h>

class AJSONParser: public AActor

public:
    // fields

    int32 genderIdx;
```

```
    int32 suitIdx;
    int32 skinIdx;
    AUserAccountManager* user;

    // methods

    int32 GetgenderIdx();
    int32 GetsuitIdx();
    int32 GetskinIdx();
    int CheckJson();
    void CreateJson(FString JsonFilePath);
    FString getFilePath();
    void ExistedJson(FString JsonFilePath);
    virtual void Tick(float DeltaTime);
;
```

class ALesson

```
#include <Lesson.h>

class ALesson: public AActor

public:
    // fields

    bool accessLesson;

    // methods

    void SetLessonName(FString lesName);
    FString GetLessonName();
    AStage* GetCurrentStage();
    bool Perform();
    bool Undo();
    int32 GetStageID();
    int32 GetActionID();
    void SetCurrentStage(int numOfStage);
    AStage* GetStage(int StageID);
    int GetStageCount() const;
    int GetStageIndex(AStage* Node);
    bool SetStageIndex(int NodeIndex, int TargetIndex);
    bool Remove(int Index);
    void AddStage(AStage* Stage);
    virtual void EndPlay(const EEndPlayReason::Type EndPlayReason);
    virtual void Tick(float DeltaTime);
;
```

class AMagesController

```
#include <MagesController.h>

class AMagesController: public AActor

public:
    // fields

    class UMagesControllerClass* ControllerClass;

    // methods

    virtual void Tick(float DeltaTime);
;
```

class AMagesPlayer

```
#include <MagesPlayer.h>

class AMagesPlayer: public AActor

public:
    // fields

    EInteractionStyle InteractionStyle;
    EMagesSDKIntegrations CurrentIntegrationType;
    bool VibrateOnHover = true;
    bool PhysicalHands;
    float MagesPhysXExpectedDeltaTime = 0.0111f;
    const float MagesPhysXVersion = 1.19f;
    bool MakeControllerInvisibleOnInteraction = false;
    bool AutomaticallySetControllerTransparency = true;
    int VelocityHistorySteps = 3;
    TArray<UMagesHand*> Hands;

    // methods

    UPROPERTY();
    UPROPERTY();
    void RegisterHand(UMagesHand* Hand);
    UMagesHand* GetHand(UPrimitiveComponent* Collider);
    UFUNCTION(BlueprintCallable, Category = "ORamaVR");
    virtual void Tick(float DeltaSeconds);
    static void DeregisterInteractable(UMagesInteractable* interactable);
;
```

class AMagesSceneGraph

```
#include <MagesSceneGraph.h>

class AMagesSceneGraph: public AActor

public:
    // fields

    EOperationDifficulty Difficulty;

    // methods

    virtual void BeginPlay();
;
```

class AMagesWidgetActor

Wrapper class for MagesWidgetComponent

```
#include <MagesWidgetActor.h>

class AMagesWidgetActor: public AActor

public:
    // methods

    class UMagesWidgetComponent* GetMagesWidgetComponent() const;
;
```

class AMarker

```
#include <Marker.h>

class AMarker: public AActor

public:
    // methods

    virtual void Tick(float DeltaTime);
;
```

class AOperationAnalytics

```
#include <OperationAnalytics.h>

class AOperationAnalytics: public AActor

public:
    // methods

    virtual void Tick(float DeltaTime);
```

```
    void OperationFinished();
;
```

class AParallelAction

```
#include <ParallelAction.h>

class AParallelAction: public ABasePrototype

public:
    // fields

    TMap<int, FIActionGroup> iActionMap;

    // methods

    UPROPERTY();
    void InsertActionToMap(int numberOfPath, ABasePrototype* iAction);
    void InsertIActionToMap(int numberOfPath, IIAction* iAction);
    UFUNCTION(BlueprintCallable, Category = "Mages|Actions");
    UFUNCTION();
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    virtual void DifficultyRestrictions();
    void OnSetPath(int PathNum);
;

// direct descendants

class ABPParallelAction;
```

Inherited Members

```
public:
    // classes

    class InstrumentTransforms;

    // fields

    TArray<FORamaVR> afterSpawnFunctions;
    TMap<AActor*, UAnimSequence*> CharacterAnimationsMap;
    TMap<AActor*, UAnimSequence*> CharacterUndoAnimations;
    TArray<AActor*> BoundActorCharacterAnimated;
    USceneComponent* _rootComp;
    int OverrideSetPath = -1;
    int AlternativePath = -1;
    FString ActionName = TEXT("");

    // methods
```

```
    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
    void SetActionNode();
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
    virtual void DifficultyRestrictions() = 0;
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    virtual void InitializeHolograms();
    void SetAfterSpawn(FORamaVR afterSpawnFunction);
    void SetUndoAction(FORamaVR undoActionFunction);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void BindCharacterAnimationToPerform(
        TSubclassOf<AActor> Actor,
        UAnimSequence* seq
        );

    void BindCharacterAnimationOnUndo(TSubclassOf<AActor> Actor,␣
↪UAnimSequence* seq);
    virtual void DifficultyRestrictions();
    virtual void ActionCall();
    void SetEventListener(FString _event);

    void SetHologramObject(
        TSubclassOf<AActor> Class,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachParent = nullptr,
        FName SocketName = NAME_None
        );

    virtual void SetNextModule(FORamaVR action);
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet);
    FString GetClassName() const;
```

class APhotonLBClient

```
#include <PhotonLBClient.h>

class APhotonLBClient:
    public AActor,
    public BaseView

public:
    // fields

    FString serverAddress;
    FString AppID;
    FString appVersion;

    // methods

    virtual void initPlayers();
    void removePlayer(int32 playerNr);
    void setupScene(int32 gridSize);

    virtual void updateState(
        int state,
        const ExitGames::Common::JString& stateStr,
        const ExitGames::Common::JString& joinedRoomName
        );

    void updateState(
        int32 state,
        const FString& stateStr,
        const FString& joinedRoomName
        );

    virtual void addPlayer(
        int playerNr,
        const ExitGames::Common::JString& playerName,
        bool local
        );

    void addPlayer(int32 playerNr, const FString& playerName, bool local);
    virtual void updateRoomList(const ExitGames::Common::JVector<ExitGames::Common::JStrin
↪roomNames);
    void updateRoomList(const TArray<FString>& roomNames);
    LoadBalancingListener* GetLoadBalancingListener();
    void Connect();
    void CreateRoom();
    void Leave();
    void JoinRoom(FString gameId);
    TArray<FRoomInfo> GetRoomList();
    void SetLocalPlayerPos(int32 x, int32 y);
    void SetAutomove(bool automove);
    void SetUseGroups(bool useGroups);
    bool IsInLobby();
    bool IsInRoom();
```

```
;
```

Inherited Members

```
public:
    // methods

    virtual void updateState(
        int state,
        const ExitGames::Common::JString& stateStr,
        const ExitGames::Common::JString& joinedRoomName
        ) = 0;

    virtual void initPlayers(void);

    virtual void addPlayer(
        int playerNr,
        const ExitGames::Common::JString& playerName,
        bool local
        ) = 0;

    virtual void removePlayer(int playerNr) = 0;
    virtual void updateRoomList(const ExitGames::Common::JVector<ExitGames::Common::JStrin
↪roomNames) = 0;
```

class APickUpTransform

```
#include <PickUpTransform.h>

class APickUpTransform: public AActor

public:
    // fields

    USceneComponent* _rootComp;

    // methods

    virtual void Tick(float DeltaTime);
;
```

class AProceduralActorComponent

```
#include <ProceduralActorComponent.h>

class AProceduralActorComponent: public AActor

public:
    // fields

    UStaticMeshComponent* StaticMeshActor = nullptr;
    UProceduralMeshComponent* ProceduralMeshActor = nullptr;
```

```
    // methods

    virtual void Tick(float DeltaTime);
;
```

class APumpAction

```
#include <PumpAction.h>

class APumpAction: public ABasePrototype

public:
    // methods

    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();

    void SetPumpPrefab(
        TSubclassOf<AActor> Blueprint,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachComponent = nullptr,
        FName SocketName = NAME_None
        );

    void SetWaitForAllPumps(bool wait);
;

// direct descendants

class ABPPumpAction;
```

Inherited Members

```
public:
    // classes

    class InstrumentTransforms;

    // fields

    TArray<FORamaVR> afterSpawnFunctions;
    TMap<AActor*, UAnimSequence*> CharacterAnimationsMap;
    TMap<AActor*, UAnimSequence*> CharacterUndoAnimations;
    TArray<AActor*> BoundActorCharacterAnimated;
    USceneComponent* _rootComp;
    int OverrideSetPath = -1;
    int AlternativePath = -1;
    FString ActionName = TEXT("");

    // methods
```

```
    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
    void SetActionNode();
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
    virtual void DifficultyRestrictions() = 0;
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    virtual void InitializeHolograms();
    void SetAfterSpawn(FORamaVR afterSpawnFunction);
    void SetUndoAction(FORamaVR undoActionFunction);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void BindCharacterAnimationToPerform(
        TSubclassOf<AActor> Actor,
        UAnimSequence* seq
        );

    void BindCharacterAnimationOnUndo(TSubclassOf<AActor> Actor,␣
↪UAnimSequence* seq);
    virtual void DifficultyRestrictions();
    virtual void ActionCall();
    void SetEventListener(FString _event);

    void SetHologramObject(
        TSubclassOf<AActor> Class,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachParent = nullptr,
        FName SocketName = NAME_None
        );

    virtual void SetNextModule(FORamaVR action);
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet);
    FString GetClassName() const;
```

class AQuestionAction

```
#include <QuestionAction.h>

class AQuestionAction: public ABasePrototype

public:
    // fields

    bool ready;

    // methods

    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    void SetQuestionPrefab(TSubclassOf<AActor> Class);
;

// direct descendants

class ABPQuestionAction;
```

Inherited Members

```
public:
    // classes

    class InstrumentTransforms;

    // fields

    TArray<FORamaVR> afterSpawnFunctions;
    TMap<AActor*, UAnimSequence*> CharacterAnimationsMap;
    TMap<AActor*, UAnimSequence*> CharacterUndoAnimations;
    TArray<AActor*> BoundActorCharacterAnimated;
    USceneComponent* _rootComp;
    int OverrideSetPath = -1;
    int AlternativePath = -1;
    FString ActionName = TEXT("");

    // methods

    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
    void SetActionNode();
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
```

```
    virtual void DifficultyRestrictions() = 0;
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    virtual void InitializeHolograms();
    void SetAfterSpawn(FORamaVR afterSpawnFunction);
    void SetUndoAction(FORamaVR undoActionFunction);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void BindCharacterAnimationToPerform(
        TSubclassOf<AActor> Actor,
        UAnimSequence* seq
        );

    void BindCharacterAnimationOnUndo(TSubclassOf<AActor> Actor,␣
↪UAnimSequence* seq);
    virtual void DifficultyRestrictions();
    virtual void ActionCall();
    void SetEventListener(FString _event);

    void SetHologramObject(
        TSubclassOf<AActor> Class,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachParent = nullptr,
        FName SocketName = NAME_None
        );

    virtual void SetNextModule(FORamaVR action);
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet);
    FString GetClassName() const;
```

class ARemoveAction

```
#include <RemoveAction.h>

class ARemoveAction: public ABasePrototype

public:
    // methods

    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();

    void SetRemovePrefab(
        TSubclassOf<AActor> RemoveBlueprint,
        TSubclassOf<AActor> ToolBlueprint,
```

```
        AActor* RemoveAttachActor = nullptr,
        USceneComponent* RemoveAttachComponent = nullptr,
        FName RemoveSocketName = NAME_None,
        AActor* ToolAttachActor = nullptr,
        USceneComponent* ToolAttachComponent = nullptr,
        FName ToolSocketName = NAME_None
        );

    FRemoveGroup SetRemovePrefabLong(
        FString grabbablePrefabPath,
        FString removePrefabPath,
        AActor* removePrefabParent = nullptr,
        FString _toolPrefabPath = "",
        InheritTransformFrom _inheritTransformGrabbale = none,
        InheritTransformFrom _inheritTransformRemove = none
        );
;

// direct descendants

class ABPRemoveAction;
```

Inherited Members

```
public:
    // classes

    class InstrumentTransforms;

    // fields

    TArray<FORamaVR> afterSpawnFunctions;
    TMap<AActor*, UAnimSequence*> CharacterAnimationsMap;
    TMap<AActor*, UAnimSequence*> CharacterUndoAnimations;
    TArray<AActor*> BoundActorCharacterAnimated;
    USceneComponent* _rootComp;
    int OverrideSetPath = -1;
    int AlternativePath = -1;
    FString ActionName = TEXT("");

    // methods

    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
    void SetActionNode();
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
    virtual void DifficultyRestrictions() = 0;
```

```
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    virtual void InitializeHolograms();
    void SetAfterSpawn(FORamaVR afterSpawnFunction);
    void SetUndoAction(FORamaVR undoActionFunction);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void BindCharacterAnimationToPerform(
        TSubclassOf<AActor> Actor,
        UAnimSequence* seq
        );

    void BindCharacterAnimationOnUndo(TSubclassOf<AActor> Actor,␣
↪UAnimSequence* seq);
    virtual void DifficultyRestrictions();
    virtual void ActionCall();
    void SetEventListener(FString _event);

    void SetHologramObject(
        TSubclassOf<AActor> Class,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachParent = nullptr,
        FName SocketName = NAME_None
        );

    virtual void SetNextModule(FORamaVR action);
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet);
    FString GetClassName() const;
```

class ARigidbodyAnimationController

```
#include <RigidbodyAnimationController.h>

class ARigidbodyAnimationController: public AActor

public:
    // methods

    virtual void Tick(float DeltaTime);
    virtual void AddRigidBodyAnimation(RigidBodyAnimationBase* r);

    bool DualQuatCast(
        float& QW,
        float& QX,
        float& QY,
        float& QZ,
```

```
        float& VX,
        float& VY,
        float& VZ,
        float& dqwReal,
        float& dqxReal,
        float& dqyReal,
        float& dqzReal,
        float& dqwDual,
        float& dqxDual,
        float& dqyDual,
        float& dqzDual
        );

    bool DualQuatLerp(
        float& dqwStartReal,
        float& dqxStartReal,
        float& dqyStartReal,
        float& dqzStartReal,
        float& dqwStartDual,
        float& dqxStartDual,
        float& dqyStartDual,
        float& dqzStartDual,
        float& dqwEndReal,
        float& dqxEndReal,
        float& dqyEndReal,
        float& dqzEndReal,
        float& dqwEndDual,
        float& dqxEndDual,
        float& dqyEndDual,
        float& dqzEndDual,
        float& dqwInterpReal,
        float& dqxInterpReal,
        float& dqyInterpReal,
        float& dqzInterpReal,
        float& dqwInterpDual,
        float& dqxInterpDual,
        float& dqyInterpDual,
        float& dqzInterpDual,
        float factor
        );

    bool QuatVecCast(
        float& dqwReal,
        float& dqxReal,
        float& dqyReal,
        float& dqzReal,
        float& dqwDual,
        float& dqxDual,
        float& dqyDual,
        float& dqzDual,
        float& QW,
        float& QX,
        float& QY,
        float& QZ,
```

```
        float& VX,
        float& VY,
        float& VZ
        );

    TArray<RigidBodyAnimationBase*> GetAnimations() const;
;
```

class ASoftParticleHelper

```
#include <SoftParticleHelper.h>

class ASoftParticleHelper: public AActor

public:
    // fields

    TMap<int32, float> affectedVertices;
    TSet<FString> currentAffectedParfticleIDs;
    ACreateDeformMesh* meshContainer;
    float resistance;
    float maxMovement;
    FVector lastposition;
    FVector initposition;
    TArray<FVector> meshVertices;
    TArray<TPair<int32, float>> affectedVerticesArray;
    TArray<TPair<ASoftParticleHelper*, float>> affectedParticlesArray;
    TMap<ASoftParticleHelper*, float> affectedParticles;
    ACreateDeformMesh* deformCreator;
    UMagesInteractableItem* InteractableItemInstance = nullptr;
    UShapeComponent* Shape;

    // methods

    void Initialize(TArray<FVector>& Vertices);
    void SetDeformCreator(ACreateDeformMesh* cdm);
    void AddAffectedVertex(int id, float affect);
    void ChangeAffectedVertex(int id, float affect);
    void AddAffectedParticle(ASoftParticleHelper* _ph, float _distPersent);
    void UpdateParticle();
    void UpdateExternalVelocities(float DeltaTime);
    void UpdateParticleVertices();

    void MoveToStartingPos(
        FVector Pos,
        FRotator Rot,
        bool bApplyToOtherParticles = true,
        float AffectValue = 1.0f
        );

    void AddExternalVelocity(FVector Vel, float AffectValue = .5f);
    void RecenterParticle();
    void RecenterParticle(TArray<FVector> from_vertices);
```

```
    FTransform getCenterPoint();
    void setCenterPoint(FTransform value);
    void setBaseVertexID(int value);
    void Solve(FVector Movement);
    FORCEINLINE bool operator == (const ASoftParticleHelper& Other) const;
;
```

class AStage

```
#include <Stage.h>

class AStage: public AActor

public:
    // methods

    void SetStageName(FString stName);
    FString GetStageName();
    bool Perform();
    bool Undo();
    int32 GetActionID();
    void SetCurrentAction(int actionID);
    AActor* GetAction(const int ActionID);
    int GetActionCount() const;
    int GetActionIndex(AActor* Node);
    bool SetActionIndex(int NodeIndex, int TargetIndex);
    bool Remove(int Index);
    void AddAction(AActor* ActionActor);
    virtual void EndPlay(const EEndPlayReason::Type EndPlayReason);
    virtual void Tick(float DeltaTime);
;
```

class AUIExtraExpNotification

```
#include <UIExtraExpNotification.h>

class AUIExtraExpNotification: public AActor

public:
    // methods

    void SetUpExtraExplanationNotification(
        const FString& _displayMessage,
        USceneComponent* _endSpherePos,
        bool _followConstantly,
        float _scaleMul
        );

    virtual void Tick(float DeltaTime);
    UPROPERTY(EditAnywhere, Category = "Mages");
;
```

class AUINotification

Overview

```
#include <UINotification.h>

class AUINotification: public AActor

public:
    // fields

    bool isActive;
    bool isDynamic;
    float CurrentLifetime;
    bool IsLifetimeApplied;
    float notificationLifetime = 8.0f;
    float CurrentFadeoutTime;
    bool IsDead;
    float timerCheck;
    float intervalDuration;
    float timerLerp = 0.f;
    float lerpDuration = 2.0f;
    float DistanceTheshold = 26.f;
    bool IsRelocating;
    FVector uiCurrPos;
    FVector headCurrPos;
    FQuat uiCurrRot;
    FQuat headCurrRot;
    EUIType UIType;

    // methods

    virtual void Tick(float DeltaTime);
    virtual void EndPlay(const EEndPlayReason::Type EndPlayReason);
    UFUNCTION() const;
    UFUNCTION();
    UFUNCTION();
    void SetType(EUIType Type);
    UFUNCTION();
    UPROPERTY();
;
```

Detailed Documentation

Fields

```
float DistanceTheshold = 26.f
```

Minimum distance between UI location and head location until FollowUser applies

class AUseAction

```
#include <UseAction.h>

class AUseAction: public ABasePrototype

public:
    // methods

    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();

    void SetUsePrefab(
        TSubclassOf<AActor> UseBlueprint,
        TSubclassOf<AActor> ColliderBlueprint,
        bool UsePrefabAlreadyExists = false,
        AActor* UseAttachActor = nullptr,
        USceneComponent* UseAttachComponent = nullptr,
        FName UseSocketName = NAME_None,
        AActor* ColliderAttachActor = nullptr,
        USceneComponent* ColliderAttachComponent = nullptr,
        FName ColliderSocketName = NAME_None
        );
;

// direct descendants

class ABPUseAction;
```

Inherited Members

```
public:
    // classes

    class InstrumentTransforms;

    // fields

    TArray<FORamaVR> afterSpawnFunctions;
    TMap<AActor*, UAnimSequence*> CharacterAnimationsMap;
    TMap<AActor*, UAnimSequence*> CharacterUndoAnimations;
    TArray<AActor*> BoundActorCharacterAnimated;
    USceneComponent* _rootComp;
    int OverrideSetPath = -1;
    int AlternativePath = -1;
    FString ActionName = TEXT("");

    // methods

    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
```

```
    void SetActionNode();
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
    virtual void DifficultyRestrictions() = 0;
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
    virtual void Initialize();
    virtual void Perform();
    virtual void Undo();
    virtual void InitializeHolograms();
    void SetAfterSpawn(FORamaVR afterSpawnFunction);
    void SetUndoAction(FORamaVR undoActionFunction);

    void SetPerformAction(
        FORamaVR performActionFunction,
        int numOfTriggerToPlay = -1
        );

    void BindCharacterAnimationToPerform(
        TSubclassOf<AActor> Actor,
        UAnimSequence* seq
        );

    void BindCharacterAnimationOnUndo(TSubclassOf<AActor> Actor,␣
↪UAnimSequence* seq);
    virtual void DifficultyRestrictions();
    virtual void ActionCall();
    void SetEventListener(FString _event);

    void SetHologramObject(
        TSubclassOf<AActor> Class,
        AActor* AttachActor = nullptr,
        USceneComponent* AttachParent = nullptr,
        FName SocketName = NAME_None
        );

    virtual void SetNextModule(FORamaVR action);
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet);
    FString GetClassName() const;
```

class AUserAccountManager

```
#include <UserAccountManager.h>

class AUserAccountManager: public AActor

public:
    // methods

    void InitializeUserAccountManager(
        ApplicationUser user,
        FString in_user_token = ""
        );

    void SetUsername(FString username);
    void SetPassword(FString in_password);
    void SetOperation(FString operation);
    void SetSession(int in_session);
    void SetDifficulty(EDifficulty difficulty);
    FString GetUserToken() const;
    FString GetUsername() const;
    FString GetOperation() const;
    FString GetPassword() const;
    int GetSession();
    EDifficulty GetDifficulty() const;
    FString GetFirstName() const;
    FString GetLastName() const;
    FString GetCountry() const;
    FString GetUserId() const;
    FString GetTenantsOrg() const;
;
```

class AUserPathTracer

```
#include <UserPathTracer.h>

class AUserPathTracer: public AActor

public:
    // methods

    virtual void Tick(float DeltaTime);
    TMap<FString, FUserAction> GetUserActions();
    void NewActionStart(class ABasePrototype* ActionInstance, const FString&␣
↪Name);

    void ActionFinished(
        const FString& Name,
        bool skipped,
        float score,
        TMap<int, TArray<FScoringFactorData>> ActionStats
        );
;
```

class AnalyticsRuntimeImporter

```
#include <AnalyticsRuntimeImporter.h>

class AnalyticsRuntimeImporter

public:
    // methods

    static TMap<int, FScoringFactorRef> ImportAnalytics(
        UMagesAnalyticsAsset* Asset,
        AActor* ActionActor
        );
;
```

class ApplicationUser

```
#include <ApplicationUser.h>

class ApplicationUser

public:
    // fields

    FString Id;
    FString UserName;
    FString Email;
    FString FirstName;
    FString LastName;
    FString Country;
    Tenant tenant;
;
```

class BaseView

```
#include <BaseView.h>

class BaseView

public:
    // methods

    virtual void updateState(
        int state,
        const ExitGames::Common::JString& stateStr,
        const ExitGames::Common::JString& joinedRoomName
        ) = 0;

    virtual void initPlayers(void);

    virtual void addPlayer(
        int playerNr,
```

```
        const ExitGames::Common::JString& playerName,
        bool local
        ) = 0;

    virtual void removePlayer(int playerNr) = 0;
    virtual void updateRoomList(const ExitGames::Common::JVector<ExitGames::Common::JStrin
↪roomNames) = 0;
;

// direct descendants

class APhotonLBClient;
```

class ClientConfiguration

```
#include <ClientConfiguration.h>

class ClientConfiguration

public:
    // fields

    FString ClientId;
    FString ClientSecret;
    FString AllowedScopes;
;
```

class DeveloperCredentials

```
#include <DeveloperCredentials.h>

class DeveloperCredentials

public:
    // fields

    FString Username;
    FString Password;
;
```

class FCurrentRequest

enum FCurrentRequest::FRequestVerb

```
#include <HttpRequestHelper.h>

enum FRequestVerb

    POST,
    GET,
    PUT,
;
```

```
#include <HttpRequestHelper.h>

class FCurrentRequest

public:
    // enums

    enum FRequestVerb;

    // fields

    FRequestVerb requestVerb;
    FString requestUrl;
    TMap<FString, FString> Headers;
    TMap<FString, FString> URLParams;

    // methods

    void Init(bool _isURLEncoded = true);
    void SendWebRequest();
    FHttpRequestRef GetRequestPointer();
;
```

class FDeformableHelper

```
#include <DeformableHelper.h>

class FDeformableHelper

public:
    // methods

    static bool IsInside(UStaticMeshComponent* test, FVector point);

    static FORCEINLINE FVector RotatePointAroundPivot(
        FVector Point,
        FVector Pivot,
        FQuat Angle
        );

    static FORCEINLINE FVector RotatePointAroundPivot(
        FVector Point,
        FVector Pivot,
        FRotator Angle
        );
;
```

class FMAGES_SDKModule

Overview

```
#include <MAGES_SDK.h>

class FMAGES_SDKModule: public IModuleInterface

public:
    // methods

    virtual void StartupModule();
    virtual void ShutdownModule();
;
```

Detailed Documentation

Methods

```
virtual void StartupModule()
```

IModuleInterface implementation

class FMagesActionNode

Overview

```
#include <MagesLiveScenegraphSubsystem.h>

class FMagesActionNode: public FPendingLatentAction

public:
    // fields

    bool IsCompleted;
    int32 OutputLink;
    FWeakObjectPtr CallbackTarget;
    FName ExecutionFunction;
    ABasePrototype* Action;
    FString ActionName;
    class UMagesLiveScenegraphSubsystem* ScenegraphSubsystem;
    UMagesAnalyticsAsset* Analytics;
    int32 ActionId;
    bool WasRecentlyTriggered;

    // construction

    FMagesActionNode(
        const FLatentActionInfo& LatentInfo,
        ABasePrototype* Action,
        FString ActionName,
        class UMagesLiveScenegraphSubsystem* ScenegraphSubsystem,
```

```
        UMagesAnalyticsAsset* Analytics,
        int32 ActionId
        );

    // methods

    virtual void UpdateOperation(FLatentResponse& Response);
;
```

Detailed Documentation

Fields

```
bool WasRecentlyTriggered
```

Set to true to trigger this action

class FMagesInteractables

```
#include <MagesInteractables.h>

class FMagesInteractables

public:
    // methods

    static void Register(
        UMagesInteractable* Interactable,
        TArray<UPrimitiveComponent*> Colliders
        );

    static void Initialize();
    static void Deregister(UMagesInteractable* Interactable);
    static UMagesInteractable* GetInteractable(UPrimitiveComponent* Collider);
    static bool IsInitialized();
;
```

class FMagesNetStream

```
#include <MagesNetworkClasses.h>

class FMagesNetStream

public:
    // fields

    bool IsWriting;
    TArray<uint8>* WriteData;

    // construction

    FMagesNetStream(TArray<uint8>* Data, bool IsWriteStream);
```

```
    // methods

    void SendNext(FVector Vector);
    void SendNext(FQuat Quaternion);
    void SendNext(uint8 Byte);
    void SendNext(uint8 Value, int Count);
    void SendNextRange(uint8* Values, int Count);
    void ReadNext(FVector& Vector);
    void ReadNext(FQuat& Quaternion);
    void ReadNext(uint8& Byte);
    void ReadNextRange(uint8* Dest, int Count);
    void ResetPayloadCount();
    uint32 GetPayloadCount();
    uint32 GetPayloadSize();
    uint32 GetReadOffset();
;
```

class FReplicaCache

Local cache of all actors spawned and replicated. Creator actors are responsible for replicated their actors on newly joined players

```
#include <ReplicaCache.h>

class FReplicaCache

public:
    // methods

    void AddEntry(class AActor* Instance, FString BlueprintPath, int ViewID);
    void RemoveEntry(int ViewID);
    void SendEntries(int TargetActorNumber, class LoadBalancingListener*␣
↪Listener);
;
```

class FUserAction

```
#include <AnalyticsObjects.h>

class FUserAction

public:
    // fields

    FString Name;
    int Multiplier;
    int Score;
    bool Skipped;
    TArray<FScoringFactorData> ScoringFactorsResults;
    float Time;
    int Errors;
    int Crits;
```

```
    int Warnings;

    // construction

    FUserAction();

    FUserAction(
        FString Name,
        bool Skipped,
        int Score,
        float Time,
        int Errors,
        int Crits,
        int Warnings,
        int Multiplier
        );
;
```

class IDeviceControllerInterface

```
#include <DeviceControllerInterface.h>

class IDeviceControllerInterface

public:
    // methods

    bool GetTriggerPressed(const EOvidVRHand& HandType);
    bool GetGripPressed(const EOvidVRHand& HandType);
    bool GetIsGrabbed(const EOvidVRHand& HandType);
    float GetGripStrength(const EOvidVRHand& HandType);
    float GetPinchStrength(const EOvidVRHand& HandType);
    bool IsMoving(const EOvidVRHand& HandType);

    void PlayHapticPulse(
        const EOvidVRHand& HandType,
        float Strength,
        float Frequency = 0.5f,
        float Duration = 0.7f
        );
;

// direct descendants

class ACameraRigInputController;
```

class IIAction

```
#include <IAction.h>

class IIAction

public:
    // methods

    FString GetActionName();
    void SetActioName();
    AAction* GetActionNode();
    void SetActionNode();
    int32 GetAlternativePath();
    void SetAlternativePath();
    virtual void Perform() = 0;
    virtual void Undo() = 0;
    virtual void Initialize() = 0;
    virtual void InitializeHolograms() = 0;
    virtual void DifficultyRestrictions() = 0;
    virtual void SetNextModule(FORamaVR action) = 0;
    virtual void SetNextModulePath(FORamaVRSetPath Action, int pathToSet) = 0;
;

// direct descendants

class ABasePrototype;
```

class IMagesInputDevice

```
#include <MagesInputDevice.h>

class IMagesInputDevice

public:
    // fields

    bool IsCurrentlyTracked;

    // methods

    virtual void Initialize(UMagesHand* Hand) = 0;
    virtual TArray<UShapeComponent> SetupDefaultPhysicalColliders(FTransform␣
↪ModelParent) = 0;
    virtual bool ReadyToInitialize() = 0;
    virtual TArray<UShapeComponent*> SetupDefaultColliders() = 0;
    virtual FString GetDeviceName() = 0;

    virtual void TriggerHapticPulse(
        uint8 durationMicroSec,
        EMagesButtons Button = Touchpad
        ) = 0;
```

```
    virtual AActor* SetupDefaultRenderModel() = 0;
    virtual float GetAxis1D(EMagesButtons button) = 0;
    virtual FVector2D GetAxis2D(EMagesButtons button) = 0;
    virtual bool GetPressDown(EMagesButtons button) = 0;
    virtual bool GetPressUp(EMagesButtons button) = 0;
    virtual bool GetPress(EMagesButtons button) = 0;
    virtual bool GetTouchDown(EMagesButtons button) = 0;
    virtual bool GetTouchUp(EMagesButtons button) = 0;
    virtual bool GetTouch(EMagesButtons button) = 0;
    virtual bool GetNearTouchDown(EMagesButtons button) = 0;
    virtual bool GetNearTouchUp(EMagesButtons button) = 0;
    virtual bool GetNearTouch(EMagesButtons button) = 0;
;
```

class IMagesNetObservable

```
#include <MagesNetObservable.h>

class IMagesNetObservable

public:
    // methods

    virtual void OnSerializeView(
        FMagesNetStream* Stream,
        const FSerializationMessageInfo* MessageInfo
        ) = 0;
;

// direct descendants

class UMagesSyncTransform;
```

class IMagesNetTransform

```
#include <MagesNetTransform.h>

class IMagesNetTransform

public:
    // methods

    virtual void Initialize() = 0;
    virtual void Tick() = 0;
    virtual void ChangeSendRate(int SendRate) = 0;
    virtual void ChangeSyncMode(EMagesSyncTransformMode InSyncTransformMode)␣
 ↪= 0;
;

// direct descendants

class UMagesSyncTransform;
```

class IMagesNetworkMessage

```
#include <MagesNetworkBackend.h>

class IMagesNetworkMessage

public:
    // fields

    uint32 EventCode;

    // methods

    virtual void Put(const FString& Key, const FString& Value) = 0;
    virtual void Put(const FString& Key, int32 Value) = 0;
    virtual int GetInt(const FString& Key) = 0;
    virtual FString GetString(const FString& Key) = 0;
;
```

class LoadBalancingListener

```
#include <LoadBalancingListener.h>

class LoadBalancingListener: public Listener

public:
    // construction

    LoadBalancingListener(BaseView* pView, UMagesNetwork* Network);

    // methods

    void setLBC(ExitGames::LoadBalancing::Client* pLbc);
    ExitGames::LoadBalancing::Client* getLBC();
    void connect(const ExitGames::Common::JString& userName);
    void setUseGroups(bool value);
    bool getUseGroups(void);
    void updateGroups(void);
    void setSendGroup(nByte value);
    void service(void);
    void createRoom(void);
    void setAutomove(bool a);
    bool getAutomove(void);
    void changeRandomColor(void);
    void nextGridSize(void);
    bool setLocalPlayerPos(int x, int y);
    void moveLocalPlayer(void);
;
```

class MAGES_SDK

```
class MAGES_SDK: public ModuleRules

public:
    // methods

    MAGES_SDK(ReadOnlyTargetRules Target);
    bool LoadPhoton(ReadOnlyTargetRules Target);
;
```

class NetMessageClass

```
#include <NetMessageClass.h>

class NetMessageClass

public:
    // fields

    NetKeyCode messageCode;
    FString toolName = "";
    bool isActive;
    int LessonID =-1;
    int StageID =-1;
    int ActionID =-1;
    FString NetIDGameobject = "";
    int NetID =-1;
    int ClientNO =-1;
    FString CustomStringData = "";
    int ParallelPath =-9999;
    int ButtonID = -1;

    // construction

    NetMessageClass(NetKeyCode k);

    NetMessageClass(
        NetKeyCode k,
        int32 lessonID,
        int32 stageID,
        int32 actionID,
        int ParallelPath = 0
        );

    NetMessageClass(NetKeyCode k, int netID, FString ObjectName, int␣
↪RandomSeed);
    NetMessageClass(NetKeyCode k, int netID);
    NetMessageClass(NetKeyCode k, int netID, int ButtonID, bool bIsToggled);
    NetMessageClass(NetKeyCode k, FString data);
    NetMessageClass(NetKeyCode k, FString data, int32 ParallelPath);

    // methods
```

```
    ExitGames::Common::Hashtable ToHashTable();
    void FromHashTable(ExitGames::Common::Hashtable* Hash);
;
```

class PeerStatesStrChecker

```
class PeerStatesStrChecker

public:
    // construction

    PeerStatesStrChecker(void);
;
```

class PhotonEventHandler

class PostRefreshLicense

```
#include <PostRefreshLicense.h>

class PostRefreshLicense

public:
    // fields

    FString UserId;
    FString ProductName;
;
```

class RigidBodyAnimationBase

```
#include <RigidBodyAnimationBase.h>

class RigidBodyAnimationBase

public:
    // fields

    USceneComponent* TransformObj;
    float Speed;
    UWorld* World;

    // methods

    virtual void UpdateRigidBodyAnimation() = 0;
    virtual void Stop() = 0;
    virtual void Start() = 0;
    virtual bool IsFinished() = 0;
;

// direct descendants
```

```
class RigidBodyMoveAndRotateDualQuat;
class RigidBodyMoveDualQuat;
```

class RigidBodyMoveAndRotateDualQuat

```
#include <RigidBodyMoveAndRotateDualQuat.h>

class RigidBodyMoveAndRotateDualQuat: public RigidBodyAnimationBase

public:
    // fields

    FVector PosTarget;
    FQuat RotTarget;
    bool bForceStart;

    // construction

    RigidBodyMoveAndRotateDualQuat(
        FVector TargetPos,
        FQuat TargetRot,
        USceneComponent* Target,
        float AnimationTime,
        bool ForceStart = true
        );

    // methods

    USceneComponent* GetTransformObj() const;
    void SetTransformObj(USceneComponent* InTransformObj);
    float GetSpeed();
    void SetSpeed(float InSpeed);
    virtual bool IsFinished();
    virtual void Start();
    virtual void Stop();
    virtual void UpdateRigidBodyAnimation();
    void Restart(FVector InTargetPos, FQuat InTargetRot);
;
```

Inherited Members

```
public:
    // fields

    USceneComponent* TransformObj;
    float Speed;
    UWorld* World;

    // methods

    virtual void UpdateRigidBodyAnimation() = 0;
```

```
    virtual void Stop() = 0;
    virtual void Start() = 0;
    virtual bool IsFinished() = 0;
```

class RigidBodyMoveDualQuat

```
#include <RigidBodyMoveDualQuat.h>

class RigidBodyMoveDualQuat: private RigidBodyAnimationBase

public:
    // fields

    USceneComponent* Target;
    bool bForceStart;

    // construction

    RigidBodyMoveDualQuat(
        USceneComponent* Target,
        USceneComponent* CurrentComponent,
        float Speed,
        bool ForceStart = true
        );

    // methods

    USceneComponent* GetTransformObj() const;
    void SetTransformObj(USceneComponent* InTransformObj);
    float GetSpeed();
    void SetSpeed(float InSpeed);
    virtual bool IsFinished();
    virtual void Start();
    virtual void Stop();
    virtual void UpdateRigidBodyAnimation();
;
```

Inherited Members

```
public:
    // fields

    USceneComponent* TransformObj;
    float Speed;
    UWorld* World;

    // methods

    virtual void UpdateRigidBodyAnimation() = 0;
    virtual void Stop() = 0;
    virtual void Start() = 0;
    virtual bool IsFinished() = 0;
```

class Tenant

```
#include <Tenant.h>

class Tenant

public:
    // fields

    FString TenantId;
    FString OrganizationName;
    FString Country;
;
```

class UAssetsImporter

```
#include <AssetsImporter.h>

class UAssetsImporter: public UGameInstanceSubsystem

public:
    // methods

    virtual void Initialize(FSubsystemCollectionBase& Collection);

    AActor* Spawn(
        TSubclassOf<AActor> Class,
        AActor* AttachActor,
        USceneComponent* AttachParent,
        FName SocketName = NAME_None
        );

    void InitActor(
        AActor* Instance,
        UClass* Class,
        USceneComponent* Parent,
        FName SocketName,
        bool IsReplicated,
        bool IsLocalCreator
        );

    UBlueprintGeneratedClass* GetBlueprintClassAtPath(FString Path);
    static TArray<FSoftObjectPath> LoadLessonBlueprints(const FString& Path);
;
```

class UAuthenticationHandler

```
#include <AuthenticationHandler.h>

class UAuthenticationHandler: public UActorComponent

public:
    // fields

    static bool hasLic = false;
    FString ProductName;
    FIsUserLoginSuccessfull IsUserLoginSuccessfullTEST;
    FOnUserLoginResponse OnUserLoginResponse;
    bool Has_License;
    FString ConnectionStatus;

    // methods

    virtual void TickComponent(
        float DeltaTime,
        ELevelTick TickType,
        FActorComponentTickFunction* ThisTickFunction
        );

    void LoginDeveloper(FMagesAuthenticationCallback Callback);
    bool LoginUser(FString _username, FString _password);

    bool LoginUserWithoutSSO(
        ClientConfiguration* client,
        FString identityUrl,
        FString username,
        FString password,
        FString product,
        FMagesAuthenticationCallback _result
        );

    void _IsLoginSuccess(bool _arg);
    void RefreshUser();
    void RefreshDeveloper();
;
```

class UAvoidObjectFactor

```
#include <AvoidObjectFactor.h>

class UAvoidObjectFactor: public UScoringFactor

public:
    // fields

    FAnalyticsErrorData ErrorData;
    ECollisionType TypeCollision;
    bool ErrorWithTime;
```

```
    float ErrorTimer;

    // methods

    float GetTimer() const;
    void SetTimer(float Timer);
    void ResetTimer();
    void AddInteractableEvents(AActor* actor, UErrorCollision*␣
↪ErrorCollision);
    void OnInteract(UErrorCollision* scriptEC);
;
```

Inherited Members

```
public:
    // fields

    EFactorImportance SFactorImportance;

    // methods

    void Initialize();
    void Perform(bool bSkipped, float& Result);
    void Undo();
    FScoringFactorData GetReadableData();
    void LogError();
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadOnly, Category = "Mages|Analytics");
```

class UCountDownFactor

```
#include <CountDownFactor.h>

class UCountDownFactor: public UScoringFactor

public:
    // methods

    virtual void Initialize_Implementation();
    virtual void Perform_Implementation(bool bSkipped, float& result);
    virtual FScoringFactorData GetReadableData_Implementation();
    virtual void Undo_Implementation();
    UPROPERTY();
    UPROPERTY();
;
```

Inherited Members

```
public:
    // fields

    EFactorImportance SFactorImportance;

    // methods

    void Initialize();
    void Perform(bool bSkipped, float& Result);
    void Undo();
    FScoringFactorData GetReadableData();
    void LogError();
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadOnly, Category = "Mages|Analytics");
```

class UCredentialsManager

```
#include <CredentialsManager.h>

class UCredentialsManager: public UObject

public:
    // methods

    static FUserCredentials LoadCredentials();
    static void SaveCredentials(FString Username, FString Password);
    static FString EncryptString(const FString& StringParam);
    static FString DecryptString(const FString& String);
;
```

class UDeviceControllerInterface

```
#include <DeviceControllerInterface.h>

class UDeviceControllerInterface: public UInterface

;
```

class UIAction

```
#include <IAction.h>

class UIAction: public UInterface

;
```

class UMagesActionAnalyticsListEntry

```
#include <MagesActionAnalyticsListEntry.h>

class UMagesActionAnalyticsListEntry: public UObject

public:
    // methods

    UPROPERTY(BlueprintReadOnly, Category = "Mages");
    UPROPERTY(BlueprintReadOnly, Category = "Mages");
    UPROPERTY(BlueprintReadOnly, Category = "Mages");
    UPROPERTY(BlueprintReadOnly, Category = "Mages");
;
```

class UMagesActionAnalyticsWidget

```
#include <MagesActionAnalyticsWidget.h>

class UMagesActionAnalyticsWidget: public UMagesWidget

public:
    // methods

    void AddEntry(UMagesActionAnalyticsListEntry* NewEntry);
    virtual void NativeOnCreated();
;
```

Inherited Members

```
public:
    // methods

    virtual void SetupWidget(
        FOnRequestDestroySelf RequestDestroySelfDelegate,
        FOnRequestOwnerRef RequestOwnerRef
        );

    UMagesWidgetComponent* RequestOwner();
    void RequestDestroySelf(bool ForceNoAnim);
    void SetInteractive(bool NewInteractive);
    void OnCreated();
    void OnInteractiveChanged(bool NewInteractive);
    FORCEINLINE bool IsInteractive();
```

```
    virtual void NativeOnInteractiveChanged(bool bIsInteractive);
    virtual void NativeOnCreated();
```

class UMagesAnalyticsAsset

```
#include <MagesAnalyticsAsset.h>

class UMagesAnalyticsAsset: public UObject

public:
    // fields

    int Multiplier = 1;
    FAnalyticsTimeData Time;
    bool QuestionEnabled = false;
    FQuestionScoringData Question;
    bool ErrorCollidersEnabled = false;
    TArray<FAnalyticsErrorData> Errors;
    bool VelocityEnabled = false;
    FVelocityScoringData Velocity;
;
```

class UMagesAudioSubsystem

Overview

```
#include <MagesAudioSubsystem.h>

class UMagesAudioSubsystem: public UGameInstanceSubsystem

public:
    // fields

    class AAmbientSound* AmbientSoundActor;
    class UAudioComponent* AudioClipComponent;

    // methods

    virtual void Initialize(FSubsystemCollectionBase& Collection);
    void PlayAmbient(const EAmbientAudioType Type);
    void StopAmbient();
    void PlayClip(const EAudioClipType Type);

    void PlayClipOnComponent(
        const EAudioClipType Type,
        UAudioComponent* AudioComponent
        );
;
```

Detailed Documentation

Methods

```
void PlayClipOnComponent(
    const EAudioClipType Type,
    UAudioComponent* AudioComponent
    )
```

Play Audio clip on the provided component

class UMagesButtonInput

```
#include <MagesButtonInput.h>

class UMagesButtonInput: public UObject

public:
    // methods

    bool GetPressDown();
    bool GetIsPressed();
    bool GetPressUp();
    bool GetTouchDown();
    bool GetTouchUp();
    bool GetIsTouched();
    bool GetNearTouchDown();
    bool GetNearTouchUp();
    bool GetIsNearTouched();
    FVector2D Axis();
    float SingleAxis();
    void FrameReset(UMagesDevice* inputDevice, EMagesButtons Button);
;
```

class UMagesButtonWidget

Overview

```
#include <MagesButtonWidget.h>

class UMagesButtonWidget: public UMagesWidget

public:
    // fields

    bool isToggleMode = false;
    EMagesButtonInteractionMethod ButtonInteractionMethod =␣
↪EMagesButtonInteractionMethod::Default;
    float RepeatInitialDelay = 0.5f;
    float RepeatDelay = 0.1f;
    bool IsTextVisibleOnInit = true;
    FOnMagesButtonClicked OnEventClicked;
```

```
    FOnMagesButtonToggled OnEventToggled;

    // methods

    void SetText(const FString& text);
    void SetTextVisible(bool Value);
    void OnClicked();
    bool GetIsToggleMode();
    void SetToggled(bool Toggled);
    bool GetToggled();
    bool IsButtonHovered();
    virtual void NativeOnInteractiveChanged(bool NewInteractive);
;

// direct descendants

class UMagesQuestionButtonWidget;
```

Inherited Members

```
public:
    // methods

    virtual void SetupWidget(
        FOnRequestDestroySelf RequestDestroySelfDelegate,
        FOnRequestOwnerRef RequestOwnerRef
        );

    UMagesWidgetComponent* RequestOwner();
    void RequestDestroySelf(bool ForceNoAnim);
    void SetInteractive(bool NewInteractive);
    void OnCreated();
    void OnInteractiveChanged(bool NewInteractive);
    FORCEINLINE bool IsInteractive();
    virtual void NativeOnInteractiveChanged(bool bIsInteractive);
    virtual void NativeOnCreated();
```

Detailed Documentation

Fields

`bool isToggleMode = false`

Button is Toggled or not

`bool IsTextVisibleOnInit = true`

Determined if text block will be visible on initialization

class UMagesButtonsHelper

```
#include <MagesButtonsHelper.h>

class UMagesButtonsHelper

public:
    // methods

    TArray<EMagesButtons> GetArray();
;
```

class UMagesConfig

```
#include <MagesConfig.h>

class UMagesConfig: public UDataAsset

public:
    // fields

    TSubclassOf<AMagesSceneGraph> SceneGraphClass;
    UMagesUserEventAsset* NetworkUserEventAsset;
    FDirectoryPath AnalyticsLocalPath;
    FString Login_URL;
    FString UploadAnalytics_URL;
    FString ProductCode;
    FString ClientID;
    FString ClientSecret;
    FString ProjectKey;

    // methods

    void Validate(TArray<FText>& ValidationError);
;
```

class UMagesControllerClass

```
#include <MagesControllerClass.h>

class UMagesControllerClass: public UActorComponent

public:
    // fields

    EControllerTypes ControllerType = OculusTouchController;
    TScriptInterface<IDeviceControllerInterface> CurrentController;
    float strength;
    float float _freq = 0.5f;
    float float float _duration = 0.7f);

    // methods
```

```
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Mages");
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Mages");
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Mages");
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Mages");
    UFUNCTION(BlueprintCallable, Category = "Mages|Controllers") const;
    UFUNCTION(BlueprintCallable, Category = "Mages|Controllers") const;
    UFUNCTION(BlueprintCallable, Category = "Mages|Controllers");
    UFUNCTION(BlueprintCallable, Category = "Mages|Controllers");
    UFUNCTION(BlueprintCallable, Category = "Mages|Controllers");
    UFUNCTION(BlueprintCallable, Category = "Mages|Controllers");
;
```

class UMagesDeformableMeshData

Overview

```
#include <MagesDeformationComponent.h>

class UMagesDeformableMeshData: public UObject

public:
    // fields

    EMagesDeformableMeshType Type;
    int32 SectionIndex;
    TArray<FVector> Positions;
    TArray<FColor> Colors;
    TArray<FVector> Normals;
    TArray<FVector2D> UVs;
    TArray<FProcMeshTangent> Tangents;
    TArray<int32> Triangles;
;
```

Detailed Documentation

Fields

`EMagesDeformableMeshType` Type

The source mesh type

`int32` SectionIndex

Section Index

class UMagesDevice

```
#include <MagesDevice.h>

class UMagesDevice: public UActorComponent

public:
    // fields

    bool IsCurrentlyTracked;

    // methods

    virtual void TickComponent(
        float DeltaTime,
        ELevelTick TickType,
        FActorComponentTickFunction* ThisTickFunction
        );

    virtual void Initialize(UMagesHand* hand);
    virtual TArray<UShapeComponent> SetupDefaultPhysicalColliders(FTransform␣
↪ModelParent);
    virtual bool ReadyToInitialize();
    virtual TArray<UShapeComponent*> SetupDefaultColliders();
    virtual FString GetDeviceName();

    virtual void TriggerHapticPulse(
        uint8 durationMicroSec,
        EMagesButtons Button = Touchpad
        );

    virtual AActor* SetupDefaultRenderModel();
    virtual float GetAxis1D(EMagesButtons button);
    virtual FVector2D GetAxis2D(EMagesButtons button);
    virtual bool GetPressDown(EMagesButtons button);
    virtual bool GetPressUp(EMagesButtons button);
    virtual bool GetPress(EMagesButtons button);
    virtual bool GetTouchDown(EMagesButtons button);
    virtual bool GetTouchUp(EMagesButtons button);
    virtual bool GetTouch(EMagesButtons button);
    virtual bool GetNearTouchDown(EMagesButtons button);
    virtual bool GetNearTouchUp(EMagesButtons button);
    virtual bool GetNearTouch(EMagesButtons button);
;
```

class UMagesGameplayUtility

Generic gameplay (math focused) utility functions

```
#include <MagesGameplayUtility.h>

class UMagesGameplayUtility: public UBlueprintFunctionLibrary

;
```

class UMagesInputDevice

```
#include <MagesInputDevice.h>

class UMagesInputDevice: public UInterface

;
```

class UMagesInstance

```
#include <MagesInstance.h>

class UMagesInstance: public UGameInstance

public:
    // methods

    class AAnalyticsManager* AnalyticsManager();
    class UAssetsImporter* AssetsImporter();
    class AUserAccountManager* UserAccountManager();
    class AUserPathTracer* UserPathTracer();
    class AAnalyticsExporter* AnalyticsExporter();
    class AOperationAnalytics* OperationAnalytics();
    class UMagesControllerClass* MagesControllerClass();
    class ARigidbodyAnimationController* RigidbodyAnimationController();
    class UUIManagement* UIManagement();
    class UMagesNetwork* MagesNetwork();
    class UMagesLiveScenegraphSubsystem* SceneGraph();
    void RegisterNetworkInstance(UMagesNetwork* Network);
    void RegisterAuthHandlerInstance(class UAuthenticationHandler* Handler);
    class UAuthenticationHandler* AuthenticationHandler();
    void MAnalytics();
    void Initialize();
;
```

class UMagesLiveScenegraphSubsystem

Overview

```
#include <MagesLiveScenegraphSubsystem.h>

class UMagesLiveScenegraphSubsystem: public UGameInstanceSubsystem

public:
    // fields

    FString CurrentLesson = "";
    int32 CurrentStageIndex = -1;
    FString LastActionName;

    // methods

    virtual void Initialize(FSubsystemCollectionBase& Collection);

    void SetupActionAnalytics(
        class ABasePrototype* Action,
        class UMagesAnalyticsAsset* Asset
        ) const;

    void CreateAction(
        const UObject* WorldContextObject,
        struct FLatentActionInfo LatentInfo,
        FString ActionName,
        TSubclassOf<ABasePrototype> ActionClass,
        UMagesAnalyticsAsset* Analytics = nullptr
        );

    void SkipAction();
    bool UndoAction();
    void Perform(ABasePrototype* ActionToPerform);
    class AMagesSceneGraph* GetGraph();
    class UMagesConfig* GetConfig();
    ABasePrototype* GetCurrentAction();
    FString GetCurrentActionName();
    void BindEvent(FString EventName, FSceneGraphEventDelegate Delegate);
    void InvokeEvent(FString EventName);
    void UnbindEvent(FString EventName);
    void AddActionAfterPerform(FSceneGraphChangeDelegate Delegate);
    void AddActionOnPerform(FSceneGraphChangeDelegate Delegate);
    void AddActionAfterUndo(FSceneGraphChangeDelegate Delegate);
    void AddActionOnUndo(FSceneGraphChangeDelegate Delegate);
    bool IsPlaying() const;
    TArray<FActionGroup> GetActionGroups();
    FMagesActionNode* FindAction(FString ActionName);
    void PerformByServer();
    void UndoByServer();
    static TArray<FString> GetActionGroupNames(FActionGroup Group);
;
```

Detailed Documentation

Methods

```
void SetupActionAnalytics(
    class ABasePrototype* Action,
    class UMagesAnalyticsAsset* Asset
    ) const
```

SceneGraph Modification

```
void CreateAction(
    const UObject* WorldContextObject,
    struct FLatentActionInfo LatentInfo,
    FString ActionName,
    TSubclassOf<ABasePrototype> ActionClass,
    UMagesAnalyticsAsset* Analytics = nullptr
    )
```

Create a new action

```
ABasePrototype* GetCurrentAction()
```

Returns current action actor instance

```
FString GetCurrentActionName()
```

Returns current action name (defined inside SceneGraph Object

```
void BindEvent(FString EventName, FSceneGraphEventDelegate Delegate)
```

SceneGraph Event Handling Bind an event

class UMagesNetObservable

```
#include <MagesNetObservable.h>

class UMagesNetObservable: public UInterface

;
```

class UMagesNetTransform

```
#include <MagesNetTransform.h>

class UMagesNetTransform: public UInterface

;
```

class UMagesNetwork

Overview

```
#include <MagesNetwork.h>

class UMagesNetwork: public UObject

public:
    // fields

    float NetworkedActorDestructionTimeout = 0.5f;
    FRandomStream RandomStream;
    bool IsClient = false;
    static const int MAX_VIEW_IDS = 100;

    // methods

    UPROPERTY(BlueprintAssignable, Category = "Mages|Networking");
    UPROPERTY(BlueprintAssignable, Category = "Mages|Networking");
    UPROPERTY(BlueprintAssignable, Category = "Mages|Networking");
    UPROPERTY(BlueprintReadOnly, Category = "Mages|Networking");
    UPROPERTY(BlueprintReadOnly, Category = "Mages|Networking");
    void Initialize();
    void OnInit(LoadBalancingListener* CurrentListener);
    void Tick(float DeltaTime);
    UMagesView* GetView(int ID);
    int GetLocalPlayerNumber();
    AActor* Instantiate(TSubclassOf<AActor> Blueprint, AActor* Parent =␣
↪nullptr);
    int GetNumPlayersInRoom();
    void RegisterUserEvent(FString EventName, FOnMagesUserEvent Callback);
    void RaiseUserEvent(FString EventName, UObject* EventData);
    bool RegisterView(UMagesView* View, int ViewID);
    void DeregisterView(UMagesView* View, int ViewID);
    void ReplicateActor(AActor* ActorInstance, AActor* ParentActor);
    void RaiseTransferAuthorityEvent(int viewID, int newPlayerID);
    void RaiseDestroyEvent(int viewID, int creatorID);
    void RaiseEvent(NetMessageClass msg, bool SentToAll = false);
    int GetServerTimestamp();
    int AllocateViewID(int OwnerID);

    void RemoveInstantiatedActor(
        AActor* actorToRemove,
        bool LocalOnly,
        int viewID,
        int CreatorID
        );
;
```

Detailed Documentation

Fields

`float NetworkedActorDestructionTimeout = 0.5f`

Sets the time to wait until an actor can be safely destroyed

`FRandomStream RandomStream`

A Synchronized RNG

`static const int MAX_VIEW_IDS = 100`

Internal

Methods

`UPROPERTY(BlueprintAssignable, Category = "Mages|Networking")`

Events Called when a player (including the local one) has entered the room

`UPROPERTY(BlueprintAssignable, Category = "Mages|Networking")`

Called on any event

`UPROPERTY(BlueprintAssignable, Category = "Mages|Networking")`

Called when the local player has created a room

`UPROPERTY(BlueprintReadOnly, Category = "Mages|Networking")`

Flags True if the local player is the host (or server)

`UPROPERTY(BlueprintReadOnly, Category = "Mages|Networking")`

True if the local player is in a Coop session

`void Initialize()`

Initialization

UMagesView`* GetView(int ID)`

Networking Returns the Mages View Component associated with this ID

`int GetLocalPlayerNumber()`

Returns the local player's actor number

`AActor* Instantiate(TSubclassOf<AActor> Blueprint, AActor* Parent = nullptr)`

Instantiate a blueprint for all connected players

`int GetNumPlayersInRoom()`

Returns the total amount of players in the current room (including the local player) or -1 otherwise

`void RegisterUserEvent(FString EventName, FOnMagesUserEvent Callback)`

Create a custom event type with a callback

`void RaiseUserEvent(FString EventName, UObject* EventData)`

Call a custom event on the network

`int AllocateViewID(int OwnerID)`

Returns 0 on failure, as it is an invalid ViewID (since no player can have 0 as their actor number)

class UMagesNetworkBackend

```
#include <MagesNetworkBackend.h>

class UMagesNetworkBackend: public UObject

public:
    // fields

    FMagesNetDelegate ConnectionResultDelegate;

    // methods

    virtual void InitializeNetwork();
    virtual void Tick(float DeltaTime);
    virtual IMagesNetworkMessage* CreateMessage(uint32 EventCode);
;
```

class UMagesQuestionButtonWidget

Overview

```
#include <MagesQuestionButtonWidget.h>

class UMagesQuestionButtonWidget: public UMagesButtonWidget

public:
    // fields

    int NetworkID;

    // methods

    void ShowQuestionCorrectness(bool IsCorrect);
;
```

Inherited Members

```
public:
    // fields

    bool isToggleMode = false;
    EMagesButtonInteractionMethod ButtonInteractionMethod =␣
↪EMagesButtonInteractionMethod::Default;
    float RepeatInitialDelay = 0.5f;
    float RepeatDelay = 0.1f;
    bool IsTextVisibleOnInit = true;
    FOnMagesButtonClicked OnEventClicked;
    FOnMagesButtonToggled OnEventToggled;
```

```
    // methods

    virtual void SetupWidget(
        FOnRequestDestroySelf RequestDestroySelfDelegate,
        FOnRequestOwnerRef RequestOwnerRef
        );

    UMagesWidgetComponent* RequestOwner();
    void RequestDestroySelf(bool ForceNoAnim);
    void SetInteractive(bool NewInteractive);
    void OnCreated();
    void OnInteractiveChanged(bool NewInteractive);
    FORCEINLINE bool IsInteractive();
    virtual void NativeOnInteractiveChanged(bool bIsInteractive);
    virtual void NativeOnCreated();
    void SetText(const FString& text);
    void SetTextVisible(bool Value);
    void OnClicked();
    bool GetIsToggleMode();
    void SetToggled(bool Toggled);
    bool GetToggled();
    bool IsButtonHovered();
    virtual void NativeOnInteractiveChanged(bool NewInteractive);
```

Detailed Documentation

Fields

```
int NetworkID
```

For Question button synchronization; Manually set by QuestionPrefabConstructor

class UMagesScrollBox

```
#include <MagesScrollBox.h>

class UMagesScrollBox: public UScrollBox

public:
    // methods

    void ScrollByPage(int Number);
;
```

class UMagesSyncTransform

Overview

```
#include <MagesSyncTransform.h>

class UMagesSyncTransform:
    public UActorComponent,
    public IMagesNetTransform,
    public IMagesNetObservable

public:
    // fields

    int SendTimesPerSecond = 15;
    float MovementThreshold = 0.05f;
    float RotationThreshold = 0.1f;
    EMagesSyncTransformMode SyncTransformMode = EMagesSyncTransformMode::All;
    TArray<FString> SynchronizedComponentNames;
    bool ShouldSynchronizeActor = true;

    // methods

    virtual void Initialize();
    virtual void Tick();
    virtual void ChangeSendRate(int SendRate);
    virtual void ChangeSyncMode(EMagesSyncTransformMode SyncTransformMode);

    virtual void OnSerializeView(
        FMagesNetStream* Stream,
        const FSerializationMessageInfo* MessageInfo
        );

    void BeginDestroy();
;
```

Inherited Members

```
public:
    // methods

    virtual void Initialize() = 0;
    virtual void Tick() = 0;
    virtual void ChangeSendRate(int SendRate) = 0;
    virtual void ChangeSyncMode(EMagesSyncTransformMode InSyncTransformMode)␣
↪= 0;

    virtual void OnSerializeView(
        FMagesNetStream* Stream,
        const FSerializationMessageInfo* MessageInfo
        ) = 0;
```

Detailed Documentation

Fields

```
TArray<FString> SynchronizedComponentNames
```

The components whose transforms will be synchronized. Obviously, they can only be sub-classes of Scene Component

class UMagesTextBoxWidget

```
#include <MagesTextBoxWidget.h>

class UMagesTextBoxWidget: public UMagesWidget

public:
    // fields

    UEditableTextBox* TextBox;

    // methods

    FString GetText() const;
;
```

Inherited Members

```
public:
    // methods

    virtual void SetupWidget(
        FOnRequestDestroySelf RequestDestroySelfDelegate,
        FOnRequestOwnerRef RequestOwnerRef
        );

    UMagesWidgetComponent* RequestOwner();
    void RequestDestroySelf(bool ForceNoAnim);
    void SetInteractive(bool NewInteractive);
    void OnCreated();
    void OnInteractiveChanged(bool NewInteractive);
    FORCEINLINE bool IsInteractive();
    virtual void NativeOnInteractiveChanged(bool bIsInteractive);
    virtual void NativeOnCreated();
```

class UMagesTextWidget

```
#include <MagesTextWidget.h>

class UMagesTextWidget: public UMagesWidget

public:
    // methods

    void SetText(const FString& text);
;
```

Inherited Members

```
public:
    // methods

    virtual void SetupWidget(
        FOnRequestDestroySelf RequestDestroySelfDelegate,
        FOnRequestOwnerRef RequestOwnerRef
        );

    UMagesWidgetComponent* RequestOwner();
    void RequestDestroySelf(bool ForceNoAnim);
    void SetInteractive(bool NewInteractive);
    void OnCreated();
    void OnInteractiveChanged(bool NewInteractive);
    FORCEINLINE bool IsInteractive();
    virtual void NativeOnInteractiveChanged(bool bIsInteractive);
    virtual void NativeOnCreated();
```

class UMagesUserEventAsset

```
#include <MagesUserEventAsset.h>

class UMagesUserEventAsset: public UDataAsset

public:
    // fields

    TArray<FMagesUserEventAssetEntry> Events;

    // methods

    bool Validate();

    FMagesUserEventAssetEntry* FindEntry(
        const FString& Query,
        uint32* OutCode = nullptr
        );

    FMagesUserEventAssetEntry* FindEntry(uint8 EventCode);
```

```
;
```

class UMagesView

```
#include <MagesView.h>

class UMagesView: public UActorComponent

public:
    // fields

    int ViewID;
    bool HasAuthority;
    EOwnershipOption OwnershipTransferType = EOwnershipOption::Takeover;
    bool IsInitialized =false;
    bool WasBound = false;

    // methods

    void Initialize(int ID);
    void BeginPlay();
    void EndPlay(const EEndPlayReason::Type EndPlayReason);

    void SerializeView(
        FMagesNetStream* Stream,
        const FSerializationMessageInfo& MessageInfo
        );

    void DeserializeView(
        FMagesNetStream* Stream,
        const FSerializationMessageInfo& MessageInfo
        );

    int GetID();
    int GetViewCreator();
    bool IsMine();
    void AddObservedComponent(UObject* Object);
    void TransferOwnership(int NewOwnerID);
    void SetNewOwner(int newOwnerID);
;
```

class UMagesWidget

Overview

```
#include <MagesWidget.h>

class UMagesWidget: public UUserWidget

public:
    // methods
```

```
    virtual void SetupWidget(
        FOnRequestDestroySelf RequestDestroySelfDelegate,
        FOnRequestOwnerRef RequestOwnerRef
        );

    UMagesWidgetComponent* RequestOwner();
    void RequestDestroySelf(bool ForceNoAnim);
    void SetInteractive(bool NewInteractive);
    void OnCreated();
    void OnInteractiveChanged(bool NewInteractive);
    FORCEINLINE bool IsInteractive();
    virtual void NativeOnInteractiveChanged(bool bIsInteractive);
    virtual void NativeOnCreated();
;

// direct descendants

class UMagesActionAnalyticsWidget;
class UMagesButtonWidget;
class UMagesTextBoxWidget;
class UMagesTextWidget;
```

Detailed Documentation

Methods

`UMagesWidgetComponent* RequestOwner()`

Request to get a reference to the owning Mages Widget Actor

`void RequestDestroySelf(bool ForceNoAnim)`

Request to be destroyed by the owning Mages Widget Actor

`void OnCreated()`

Called when created at game time by the Mages Widget Actor

class UOrama_Util

Overview

```
#include <Orama_Util.h>

class UOrama_Util: public UObject

public:
    // methods

    static void MagesDestroyActor(AActor* ToDestroy);
    static UStaticMesh* FindAsset(FString Name);
    static UMagesInstance* MagesInstance(const UObject* WorldContextObject);
    static class UMagesControllerClass* GetMagesControllerClass(const␣
↪UObject* WorldContextObject);
```

```
    static void ThrowBlueprintError(
        const UObject* WorldContextObject,
        FString ErrorString
        );

    static void ThrowBlueprintWarning(
        const UObject* WorldContextObject,
        FString WarnString
        );

    static AActor* FindBlueprint(FString Name, UWorld* World);
    static void PrintMessage(FString Message);
    static float AngleBetweenVectors(const FVector& A, const FVector& B);
    static AActor* FindActor(const UWorld* World, const FString& Query);
    static TArray<AActor*> FindAllActors(const UWorld* World, const FString&␣
↪Query);
    static FString EvalPath(FString Path);

    static void SetActorCollisionEnabled(
        AActor* Actor,
        ECollisionEnabled::Type Type
        );

    static AActor* FindActorByClass(const UWorld* World, UClass* ActorClass);

    static UActorComponent* FindComponentByName(
        AActor* Actor,
        FString ComponentName,
        TSubclassOf<UActorComponent> ComponentSubclass =␣
↪UActorComponent::StaticClass()
        );

    static bool BlueprintClassHasComponent(
        class UBlueprintGeneratedClass* GenClass,
        UClass* ComponentClass
        );

    static UActorComponent* FindDefaultComponentByClass(
        const TSubclassOf<AActor>& ActorClass,
        const TSubclassOf<UActorComponent>& ComponentClass
        );

    static FString RemoveConsecutiveSeparators(const FString& Path);
    static TArray<UMagesInstance*> GetAllMagesInstances();
;
```

Detailed Documentation

Methods

```
static FString RemoveConsecutiveSeparators(const FString& Path)
```

Workaround for FSoftObjectPath::TryLoad crashing when dealing with multiple consecutive separators

class UQuestionScoringFactor

```
#include <QuestionScoringFactor.h>

class UQuestionScoringFactor: public UScoringFactor

public:
    // fields

    FQuestionScoringData Data;
    FString BlueprintPath;

    // methods

    virtual void BeginPlay();
    virtual void Initialize_Implementation();
    virtual void Perform_Implementation(bool bSkipped, float& result);
    virtual void Undo_Implementation();
    virtual FScoringFactorData GetReadableData_Implementation();
;
```

Inherited Members

```
public:
    // fields

    EFactorImportance SFactorImportance;

    // methods

    void Initialize();
    void Perform(bool bSkipped, float& Result);
    void Undo();
    FScoringFactorData GetReadableData();
    void LogError();
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadOnly, Category = "Mages|Analytics");
```

class URemoveWithToolConstructor

```
#include <RemoveAction.h>

class URemoveWithToolConstructor: public UActorComponent

public:
    // methods

    virtual void BeginPlay();
    void Construct();

    void BeginOverlap(
        UPrimitiveComponent* OverlappedComp,
        AActor* OtherActor,
        UPrimitiveComponent* OtherComp,
        int32 OtherBodyIndex,
        bool bFromSweep,
        const FHitResult& SweepResult
        );

    void EndOverlap(
        UPrimitiveComponent* OverlappedComp,
        AActor* OtherActor,
        UPrimitiveComponent* OtherComp,
        int32 OtherBodyIndex
        );
;
```

class UScoringFactor

```
#include <ScoringFactor.h>

class UScoringFactor: public UActorComponent

public:
    // fields

    EFactorImportance SFactorImportance;

    // methods

    void Initialize();
    void Perform(bool bSkipped, float& Result);
    void Undo();
    FScoringFactorData GetReadableData();
    void LogError();
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadOnly, Category = "Mages|Analytics");
;
```

```
// direct descendants

class UAvoidObjectFactor;
class UCountDownFactor;
class UQuestionScoringFactor;
class UVelocityScoringFactor;
```

class UToolNetSync

Overview

```
#include <ToolNetSync.h>

class UToolNetSync: public UActorComponent

public:
    // fields

    bool Activated;
    bool IsToolActive = false;
    float LastSentTime = -100.0f;
    float LastSentDeactiveTime = -100.0f;
    AActor* Tool;
    UGestureHands* GetstureHands;

    // methods

    void ActivateTool(int ViewID);
    void DeActivateTool(int ViewID);
    virtual void BeginPlay();
    void ChangeActive(bool Value);
    void ChangeActiveTool(bool Value, int ViewID);
;
```

Detailed Documentation

Fields

```
bool Activated
```

Activated Tool by network

Methods

`void ActivateTool(int ViewID)`

Activate Tool network callback. This function is set by networkManager

`void DeActivateTool(int ViewID)`

Deativate Tool network callback. This function is set by networkManager

class UUIManagement

```
#include <UIManagement.h>

class UUIManagement: public UObject

public:
    // fields

    EUIType _type;
    FString _displayMessage;
    FString float _lifeTime;
    USceneComponent* _endSpherePos;
    USceneComponent bool _followConstantly;
    USceneComponent bool float _scaleMul = 1.f);
    bool _makeCameraParent;
    bool float _distance = 200.f);
    bool _followLeftHand;
    bool float offsetY = 12.f);
    bool ForceNoAnim = false);

    // methods

    void OnInit();
    UFUNCTION(BlueprintCallable, Category = "Mages|Widgets");
    UFUNCTION(BlueprintCallable, Category = "Mages|Widgets");
    UFUNCTION(BlueprintCallable, Category = "Mages|Widgets");
    UFUNCTION(BlueprintCallable, Category = "Mages|Widgets");
    UFUNCTION(BlueprintCallable, Category = "Mages|Widgets");
    UFUNCTION(BlueprintCallable, Category = "Mages|Widgets");
    UFUNCTION(BlueprintCallable, Category = "Mages|Widgets");
    UFUNCTION(BlueprintCallable, Category = "Mages|Widgets");
    UFUNCTION(BlueprintCallable, Category = "Mages|Widgets");
    UFUNCTION(BlueprintCallable, Category = "Mages|Widgets");
    UFUNCTION(BlueprintCallable, Category = "Mages|Widgets");
    UFUNCTION(BlueprintCallable, Category = "Mages|Widgets");
    UFUNCTION(BlueprintCallable, Category = "Mages|Widgets");
    UFUNCTION(BlueprintCallable, Category = "Mages|Widgets") const;
    void UpdateSpawnedUICount(bool Spawned);
    void SpawnNextUINotificationFromQueue();
;
```

class UUserCredentialsSaveGame

```
#include <UserCredentialsSaveGame.h>

class UUserCredentialsSaveGame: public USaveGame

public:
    // fields

    FString EncryptedUsername;
    FString EncryptedPassword;
;
```

class UVelocityScoringFactor

```
#include <VelocityScoringFactor.h>

class UVelocityScoringFactor: public UScoringFactor

public:
    // fields

    FVelocityScoringData Data;
    FString BlueprintPath;
    float maxVelocity;

    // methods

    virtual void BeginPlay();
    virtual void Initialize_Implementation();
    virtual void Perform_Implementation(bool bSkipped, float& result);
    virtual void Undo_Implementation();
    virtual FScoringFactorData GetReadableData_Implementation();
;
```

Inherited Members

```
public:
    // fields

    EFactorImportance SFactorImportance;

    // methods

    void Initialize();
    void Perform(bool bSkipped, float& Result);
    void Undo();
    FScoringFactorData GetReadableData();
    void LogError();
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
```

```
    UPROPERTY(BlueprintReadWrite, Category = "Mages|Analytics");
    UPROPERTY(BlueprintReadOnly, Category = "Mages|Analytics");
```

class UserAccessToken

```
#include <UserAccessToken.h>

class UserAccessToken

public:
    // fields

    FString access_token;
;
```

Overview

```
// namespaces

namespace Collections;
namespace Diagnostics;
namespace DotNETCommon;
namespace ELicenseType;
namespace ESyncTransformFlags;
namespace ExitGames;
    namespace ExitGames::Common;
    namespace ExitGames::LoadBalancing;
namespace IO;
namespace MagesMath;
namespace MagesNetworkStatusCode;
namespace System;
namespace UnrealBuildTool;

// typedefs

typedef int EMagesNetworkStatusCode;

// enums

enum EActionType;
enum EAmbientAudioType;
enum EAnalyticsColliderBehavior;
enum EAnalyticsErrorType;
enum EAnalyticsFactorImportance;
enum EAudioClipType;
enum ECollisionType;
enum EControllerDOF;
enum EControllerTypes;
enum EDifficulty;
enum EErrorType;
enum EFactorImportance;
enum EHandState;
enum EInteractionStyle;
```

```
enum ELoginStatus;
enum EMagesButtonInteractionMethod;
enum EMagesButtons;
enum EMagesControllerButtons;
enum EMagesDeformableMeshType;
enum EMagesSDKIntegrations;
enum EMagesSyncTransformMode;
enum ENetVarType;
enum EOperationDifficulty;
enum EOvidVRHand;
enum EOwnershipOption;
enum EScoringMethod;
enum ESendMethod;
enum EUIType;
enum EVisibilityLevel;
enum InheritTransformFrom;
enum NetKeyCode;
enum OnPrefabDetachFeature;
enum ParticleGroupMehod;
enum PrefabActionOnPerform;
enum PrefabInteractableType;
enum PrefabType;
enum PumpMode;
enum ToolFlashType;
enum ToolGrabbingType;
enum ToolRotationAxis;
enum ToolTriggerButton;
enum UseColliderTrigger;

// structs

struct FActionAnalyticsData;
struct FActionGroup;
struct FActionSummary;
struct FAnalyticsErrorData;
struct FAnalyticsTimeData;
struct FAnimationGroup;
struct FAudioAsset;
struct FAvoidObjectsData;
struct FDrillCategorizeFacesDesc;
struct FDrillEvalIntersectionDesc;
struct FDrillSplitCertainFacesDesc;
struct FDuo;
struct FErrorsStayData;
struct FGenerateTearSegmentTrianglesDesc;
struct FHoloGroup;
struct FIActionGroup;
struct FInsertGroup;
struct FLineSegment;
struct FMagesActionAnalyticsEntry;
struct FMagesActionPath;
struct FMagesActionPathEntry;
struct FMagesPrepareTearResult;
struct FMagesUserEventAssetEntry;
```

```
struct FNamedTimeProfileContext;
struct FOrientationContext;
struct FPostCheckoutProduct;
struct FPumpGroup;
struct FQuestionData;
struct FQuestionOptionData;
struct FQuestionScoringData;
struct FRaiseEventBatch;
struct FRecalculateNormalsDesc;
struct FRecalculateNormalsVertexEntry;
struct FRecalculateNormalsVertexKey;
struct FRemoveGroup;
struct FReplicaCacheEntry;
struct FRetriangulateInitialTearPointsDesc;
struct FRoomInfo;
struct FScoringFactorData;
struct FScoringFactorRef;
struct FSerializationMessageInfo;
struct FSerializeViewBatch;
struct FSpawnActorDesc;
struct FTimeData;
struct FTimeProfileContext;
struct FUserCredentials;
struct FVelocityData;
struct FVelocityScoringData;
struct LocalPlayer;

// classes

class AAction;
class AActorNode;
class AAnalyticsExporter;
class AAnalyticsManager;
class AAnimationAction;
class ABPAnimationAction;
class ABPBasePrototype;
class ABPCombinedAction;
class ABPInsertAction;
class ABPParallelAction;
class ABPPumpAction;
class ABPQuestionAction;
class ABPRemoveAction;
class ABPUseAction;
class ABasePrototype;
class ACameraRigInputController;
class ACombinedAction;
class ACreateDeformMesh;
class AEventManager;
class AInsertAction;
class AJSONParser;
class ALesson;
class AMagesController;
class AMagesPlayer;
class AMagesSceneGraph;
```

```
class AMagesWidgetActor;
class AMarker;
class AOperationAnalytics;
class AParallelAction;
class APhotonLBClient;
class APickUpTransform;
class AProceduralActorComponent;
class APumpAction;
class AQuestionAction;
class ARemoveAction;
class ARigidbodyAnimationController;
class ASoftParticleHelper;
class AStage;
class AUIExtraExpNotification;
class AUINotification;
class AUseAction;
class AUserAccountManager;
class AUserPathTracer;
class AnalyticsRuntimeImporter;
class ApplicationUser;
class BaseView;
class ClientConfiguration;
class DeveloperCredentials;
class FCurrentRequest;
class FDeformableHelper;
class FMAGES_SDKModule;
class FMagesActionNode;
class FMagesInteractables;
class FMagesNetStream;
class FReplicaCache;
class FUserAction;
class IDeviceControllerInterface;
class IIAction;
class IMagesInputDevice;
class IMagesNetObservable;
class IMagesNetTransform;
class IMagesNetworkMessage;
class LoadBalancingListener;
class MAGES_SDK;
class NetMessageClass;
class PeerStatesStrChecker;
class PhotonEventHandler;
class PostRefreshLicense;
class RigidBodyAnimationBase;
class RigidBodyMoveAndRotateDualQuat;
class RigidBodyMoveDualQuat;
class Tenant;
class UAssetsImporter;
class UAuthenticationHandler;
class UAvoidObjectFactor;
class UCountDownFactor;
class UCredentialsManager;
class UDeviceControllerInterface;
class UIAction;
```

```
class UMagesActionAnalyticsListEntry;
class UMagesActionAnalyticsWidget;
class UMagesAnalyticsAsset;
class UMagesAudioSubsystem;
class UMagesButtonInput;
class UMagesButtonWidget;
class UMagesButtonsHelper;
class UMagesConfig;
class UMagesControllerClass;
class UMagesDeformableMeshData;
class UMagesDevice;
class UMagesGameplayUtility;
class UMagesInputDevice;
class UMagesInstance;
class UMagesLiveScenegraphSubsystem;
class UMagesNetObservable;
class UMagesNetTransform;
class UMagesNetwork;
class UMagesNetworkBackend;
class UMagesQuestionButtonWidget;
class UMagesScrollBox;
class UMagesSyncTransform;
class UMagesTextBoxWidget;
class UMagesTextWidget;
class UMagesUserEventAsset;
class UMagesView;
class UMagesWidget;
class UOrama_Util;
class UQuestionScoringFactor;
class URemoveWithToolConstructor;
class UScoringFactor;
class UToolNetSync;
class UUIManagement;
class UUserCredentialsSaveGame;
class UVelocityScoringFactor;
class UserAccessToken;

// global variables

static const TCHAR* Importances[] =    TEXT("VeryLittle"),     TEXT("Little"),
↪     TEXT("Neutral"),    TEXT("Big"),  ;
static const TCHAR* ErrorTypeStrings[] =   TEXT("Warning"),    TEXT("Error"),␣
↪ TEXT("Critical Error"), ;
static const TCHAR* ColliderBehaviorStrings[] =    TEXT("Avoid Objects"), ␣
↪TEXT("Must Hit Objects"),   TEXT("Stay While Interacting"), ;
static uint8 USER_CREDENTIALS_KEY[FAES::FAESKey::KeySize] =      0x16,0x23,
↪0x45,0x4a,0xba,0x96,0x7c,0x16,0x43,0x85,0xa0,0xcf,0x69,0x0b,0xe1,0x5f,   ␣
↪0x97,0x64,0x39,0x22,0xde,0x33,0xd9,0x47,0x5c,0xa9,0x6b,0xc8,0xfb,0x42,0x31,
↪0x26  ;
const ExitGames::Common::JString PeerStatesStr[] =   L"Uninitialized",  ␣
↪L"PeerCreated",     L"ConnectingToNameserver",  L"ConnectedToNameserver",
↪   L"DisconnectingFromNameserver",     L"Connecting",  L"Connected",
↪   L"WaitingForCustomAuthenticationNextStepCall",  L"Authenticated",
↪   L"JoinedLobby",     L"DisconnectingFromMasterserver", ␣
```

```
↪ L"ConnectingToGameserver",  L"ConnectedToGameserver",
↪L"AuthenticatedOnGameServer",   L"Joining",     L"Joined",
↪L"Leaving",     L"Left",    L"DisconnectingFromGameserver",
↪L"ConnectingToMasterserver",    L"ConnectedComingFromGameserver",
↪L"AuthenticatedComingFromGameserver",   L"Disconnecting",   L"Disconnected"
↪;
class PeerStatesStrChecker checker;
static float QUESTION_ANIMATION_TIMES[(int) UQuestionPrefabConstructor::AnimationState::Cou
↪=                                    0.6f,                                    0.
↪35f,                           0.5f,     -1.f,
↪   0.6f,                                          -1.f, ;

// global functions

static void CategorizeFaces(
    UMagesDeformableMeshData* MeshData,
    FDrillCategorizeFacesDesc* Desc
    );

static bool FindAffectedIds(
    UMagesDeformableMeshData* MeshData,
    FLineSegment Line,
    const TArray<int32>& IdsInside,
    TArray<int32>& OutAffectedIds,
    UMagesDrillComponent* This
    );

static void EvalAllIntersections(
    UMagesDeformableMeshData* MeshData,
    FDrillEvalIntersectionDesc* Desc
    );

static void EvalIntersection(
    UMagesDeformableMeshData* MeshData,
    FDrillEvalIntersectionDesc* Desc
    );

static void TriangulateModel(
    UMagesDeformableMeshData* MeshData,
    TArray<int32>& AffectedIds,
    TArray<FQuadruplet>& VRelation,
    TArray<int32>& Inside,
    TArray<int32>& IdsToBeDeleted,
    TArray<FTriplet>& AllNewFaces
    );

static void SplitCertainFaces(
    UMagesDeformableMeshData* MeshData,
    FDrillSplitCertainFacesDesc* Desc
    );

static int32 IsWithinDrill(
    const FLineSegment& DrillLine,
    float DrillRadius,
```

```
    FVector Point
    );

static void DisableShadows(AActor* actor);
void AppendToFile(FString path, FString content);
FString DoubleDigit(FString str);
FString DifficultyToString(EDifficulty diff);
const TCHAR* FactorImportanceToString(const EAnalyticsFactorImportance&␣
↪Importance);
const TCHAR* ErrorTypeToString(const EAnalyticsErrorType& Type);
const TCHAR* ColliderBehaviorToString(const EAnalyticsColliderBehavior& Type);

static void CopyMaterialAttributes(
    UPrimitiveComponent* From,
    UPrimitiveComponent* To
    );

static void InternalRecalculateNormals(FRecalculateNormalsDesc* Desc);
static void InternalEnsureMatchAttributes(UMagesDeformableMeshData* Data);

static void CreateMeshSectionsForPredicates(
    UProceduralMeshComponent* Mesh,
    UMagesDeformableMeshData* Data,
    const TArray<UShapeComponent*>& Predicates,
    TArray<UMagesDeformableMeshData*>& MeshDatas,
    bool EnableDebug
    );

static FORCEINLINE bool operator == (
    const FRecalculateNormalsVertexKey& A,
    const FRecalculateNormalsVertexKey& B
    );

static FORCEINLINE bool operator != (
    const FRecalculateNormalsVertexKey& A,
    const FRecalculateNormalsVertexKey& B
    );

static FORCEINLINE uint32 GetTypeHash(const FRecalculateNormalsVertexKey&␣
↪VertexKey);

template <typename T>
TArray<T> GetUniqueArray(const TArray<T>& InArray);

static TArray<FVector> GetUniqueVertices(
    const TArray<int32>& InBetween,
    const TArray<int32>& UniqueInBetween,
    const TArray<FVector>& InBetweenVertices
    );

static FVector GetFaceType(
    FVector F0,
    FVector F1,
    FVector F2,
```

```
    FPointNormalPlane Plane
    );

static int ClassifyConflictingType(FVector FaceType);

static void GenerateTearSegmentTriangles(
    FGenerateTearSegmentTrianglesDesc* Desc,
    TArray<int32>& NewTriangles,
    int32 F0,
    int32 F1,
    int32 F2
    );

static void DebugDrawTriangle(
    UWorld* World,
    FVector A,
    FVector B,
    FVector C,
    FColor Color
    );

static int32 ClosestIntersectionPointToTearPoint(
    TArray<int32>& Triangles,
    TArray<FVector>& Vertices,
    TArray<int32>& FinalPointsTriangles,
    int32 Face
    );

static int randomColor(int from = 0, int to = 256);

static void TranslateObject(
    ExitGames::Common::Hashtable& Hashtable,
    UObject* Object,
    UClass* ObjectClass,
    bool ShouldDecode
    );

static FORCEINLINE FSoftClassPath SoftClassPathFromUClass(UClass* Class);
static FORCEINLINE UClass* ClassFromSoftClassPath(FSoftObjectPath& SoftClass);

template <typename T>
static void TranslatePrimitiveValue(
    ExitGames::Common::Hashtable& Hashtable,
    UObject* Object,
    const ExitGames::Common::JString& Name,
    FProperty* Property,
    bool ShouldDecode
    );

static bool TryTranslateString(
    ExitGames::Common::Hashtable& Hashtable,
    UObject* Object,
    FProperty* Property,
    const ExitGames::Common::JString& Name,
```

```
    bool ShouldDecode
    );

static bool TryTranslateBool(
    ExitGames::Common::Hashtable& Hashtable,
    UObject* Object,
    FProperty* Property,
    const ExitGames::Common::JString& Name,
    bool ShouldDecode
    );

static bool TryTranslateNumber(
    ExitGames::Common::Hashtable& Hashtable,
    UObject* Object,
    FProperty* Property,
    const ExitGames::Common::JString& Name,
    bool ShouldDecode
    );

static void _ValidateFilePath(
    TArray<FText>& ValidationError,
    const FFilePath& Path,
    bool bCanBeEmpty,
    const char* Name
    );

static bool PackagedValidateKey(const TArray<FString> Components);
static FString EncryptDecryptKey(FString text, FString key);
static int32 DecodeKey(FString Key);

void SetWidgetInteractionComponentEnabled(
    UWidgetInteractionComponent* Component,
    bool enabled
    );

static UMagesWidgetComponent* SpawnWidgetComponent(
    AActor* ActorToAttach,
    USceneComponent* OptRoot,
    TSubclassOf<UMagesWidgetComponent> Class
    );

static AActor* BindActorReplica(UWorld* World, UClass* ActorClass);
static FString GetFullBlueprintPath(FString BasePath, FString BlueprintPath);

static FVector MoveTowards(
    FVector current,
    FVector target,
    float maxDistanceDelta
    );

static void ClampLocation(
    FVector& V,
    FVector Origin,
    float DistX,
```

```
    float DistY,
    float DistZ
    );

const MAGES_SDK_API TCHAR* FactorImportanceToString(const␣
↪EAnalyticsFactorImportance& Importance);
const MAGES_SDK_API TCHAR* ErrorTypeToString(const EAnalyticsErrorType& Type);
const MAGES_SDK_API TCHAR* ColliderBehaviorToString(const␣
↪EAnalyticsColliderBehavior& Type);

DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(
    FORamaAnimator,
    FString,
    AnimationName
    );

DECLARE_DYNAMIC_DELEGATE(FEventDelegate);
DECLARE_DYNAMIC_DELEGATE(FORamaVR);
DECLARE_DYNAMIC_DELEGATE_OneParam(FORamaVRSetPath, int, numberOfPath);

DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(
    FIsUserLoginSuccessfull,
    bool,
    _RESULT
    );

DECLARE_EVENT_OneParam(FIsUserLoginSuccessfull, IsLoginSuccessTEST, bool);

DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(
    FOnUserLoginResponse,
    ELoginStatus,
    Status
    );

DECLARE_DELEGATE_OneParam(FMagesAuthenticationCallback, ELoginStatus);
DECLARE_DELEGATE_OneParam(FMagesAuthClientCallback, FString);
DECLARE_DELEGATE_OneParam(FMagesAuthClientLicenseCallback,␣
↪ELicenseType::Type);
DECLARE_DELEGATE_OneParam(FMagesAuthClientAppUserCallback, ApplicationUser*);
DECLARE_DYNAMIC_DELEGATE(FHandInteractDelegate);
DECLARE_DYNAMIC_DELEGATE(FInteractactionDelegate);
DECLARE_DELEGATE(FInteractactionStaticDelegate);

DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(
    FOnEvent,
    int,
    PlayerNumber,
    uint8,
    EventCode
    );

DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(
    FOnPlayerEnteredRoom,
    int,
```

```
    PlayerNumber
    );

DECLARE_DYNAMIC_MULTICAST_DELEGATE(FOnCreatedRoom);
DECLARE_MULTICAST_DELEGATE_OneParam(FMagesNetDelegate, int);

FORCEINLINE bool operator == (
    const FRaiseEventBatch& Left,
    const FRaiseEventBatch& Right
    );

FORCEINLINE uint32 GetTypeHash(const FRaiseEventBatch& Batch);

DECLARE_DYNAMIC_DELEGATE_TwoParams(
    FOnMagesUserEvent,
    int,
    Sender,
    UObject*,
    EventData
    );

FORCEINLINE bool operator == (
    const FReplicaCacheEntry& A,
    const FReplicaCacheEntry& B
    );

FORCEINLINE uint32 GetTypeHash(const FReplicaCacheEntry& Entry);

DECLARE_DYNAMIC_DELEGATE_TwoParams(
    FOnMagesScenegraphEvent,
    int,
    Sender,
    FString,
    EventData
    );

DECLARE_DYNAMIC_DELEGATE(FSceneGraphEventDelegate);

DECLARE_DYNAMIC_DELEGATE_OneParam(
    FSceneGraphChangeDelegate,
    ABasePrototype*,
    Action
    );

DECLARE_DELEGATE_OneParam(
    FAuthenticationDelegate,
    class UAuthenticationHandler*
    );

DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(
    FOnMagesButtonClicked,
    class UMagesButtonWidget*,
    ButtonWidget
    );
```

```
DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(
    FOnMagesButtonToggled,
    class UMagesButtonWidget*,
    ButtonWidget,
    bool,
    bIsToggled
    );

DECLARE_DYNAMIC_DELEGATE_OneParam(FOnRequestDestroySelf, bool, ForceNoAnim);

DECLARE_DYNAMIC_DELEGATE_RetVal(
    class UMagesWidgetComponent*,
    FOnRequestOwnerRef
    );

DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(
    FOramaOnAnswerSubmission,
    TArray<FString>,
    GivenAsnwers,
    TArray<FString>,
    CorrectAnswers
    );

DECLARE_DYNAMIC_DELEGATE_OneParam(FOnStayTimeDelegate, float, timeValue);
FORCEINLINE uint32 GetTypeHash(const ASoftParticleHelper& Helper);

// macros

#define GET_SUBSYSTEM(T)
#define LOCTEXT_NAMESPACE
#define MAGES_DESTROY(Actor)
#define MAGES_VALID_ACTOR(Actor)
#define VALIDATE_FILE_PATH(ValidationError, Path, bCanBeEmpty)
```

Detailed Documentation

Global Functions

```
static void CategorizeFaces(
    UMagesDeformableMeshData* MeshData,
    FDrillCategorizeFacesDesc* Desc
    )
```

Categorize faces by their orientation with respect to the drill circle

```
static bool FindAffectedIds(
    UMagesDeformableMeshData* MeshData,
    FLineSegment Line,
    const TArray<int32>& IdsInside,
    TArray<int32>& OutAffectedIds,
    UMagesDrillComponent* This
    )
```

Find all the affected ids on the drill circle excluding the ones one the opposite side of the model

```
static void EvalAllIntersections(
    UMagesDeformableMeshData* MeshData,
    FDrillEvalIntersectionDesc* Desc
    )
```

Stores the intersection point of each edge of the mesh

```
static void EvalIntersection(
    UMagesDeformableMeshData* MeshData,
    FDrillEvalIntersectionDesc* Desc
    )
```

Stores the intersection point of the edge defined by v[i] and v[j] by the drill

```
static void TriangulateModel(
    UMagesDeformableMeshData* MeshData,
    TArray<int32>& AffectedIds,
    TArray<FQuadruplet>& VRelation,
    TArray<int32>& Inside,
    TArray<int32>& IdsToBeDeleted,
    TArray<FTriplet>& AllNewFaces
    )
```

Re triangulates the model to include the new vertices and "remove" the ones inside the drill circle

```
static void SplitCertainFaces(
    UMagesDeformableMeshData* MeshData,
    FDrillSplitCertainFacesDesc* Desc
    )
```

Split faces for affected vertices

```
static int32 IsWithinDrill(
    const FLineSegment& DrillLine,
    float DrillRadius,
    FVector Point
    )
```

Get where the point is situated with respect to the drill sector Return Value: 1 - Inside of the sector -1 - Outside of the sector 0 - On top of the sector (inside the sector boundary)

```
static void InternalEnsureMatchAttributes(UMagesDeformableMeshData* Data)
```

Ensures that UVs, normals, and tangents match up to the total number of vertices.

```
static void CreateMeshSectionsForPredicates(
    UProceduralMeshComponent* Mesh,
    UMagesDeformableMeshData* Data,
    const TArray<UShapeComponent*>& Predicates,
    TArray<UMagesDeformableMeshData*>& MeshDatas,
    bool EnableDebug
    )
```

Creates multiple mesh section depending on the predicates array, which should be populated (or not) properly before the call to Initialize()

Todo : Each section can only have a single material, can we provide an easy way to merge them?

```
static int ClassifyConflictingType(FVector FaceType)
```

Given the tuple corresponding the vertices of a face f, it returns the conflict type that corresponds to this face.

```
DECLARE_DYNAMIC_DELEGATE_TwoParams(
    FOnMagesScenegraphEvent,
    int,
    Sender,
    FString,
    EventData
    )
```

Required path definitions for scenegraph data sheets

```
DECLARE_DYNAMIC_DELEGATE_OneParam(FOnStayTimeDelegate, float, timeValue)
```

TODO: offsetTimer

Macros

```
#define GET_SUBSYSTEM(T)
```

C++ Only Shorthands

Global Namespace
genindex

CHAPTER

FOURTEEN

CLOUD SERVICES

In this section we are going to provide a brief overview of the cloud services –sample projects that are included in the **MAGES SDK version 3.2.** moving forward.

Further, we will be discussing the overall architecture, the connection and exchange of information between the services, and how we can efficiently enable you to securely authenticate and manage your users' licenses.

Subsequently, for each service we will provide a separate section that covers at least the following points:

- Configuration & runtime behavior
- Customization
- Build & deploy!

Note: At the time being, ORamaVR web services strongly depend on Microsoft Azure to operate.

Migration to other Cloud providers is feasible but requires extra work and customization on your behalf. Support for other providers is not in place yet.

14.1 Preliminary

Now let's assume you have built a high-quality VR application with Unity based on MAGES SDK that you have released for the whole world to see.

And now assume that random-*bob* is your typical end-user, your teensy tiny happy user.

But the thing is you don't really know *bob*, do you? At least not before he **authenticates** and presents himself.

In addition, you wish to grant access and present to *bob* his **analytic insights** that you have been gathering all this time with so much zealous and sweat.

But putting in place every small bit of this functionality, let alone making analytics look cool, takes time, resources, and highly-payed engineering work.

Besides, you don't really find it an interesting investment, and you wish there was a way to skip all this boilerplate and dive right into the creative part, the part you love the most. Developing VR content, placing all your mojo-juice exactly right where it matters the most.

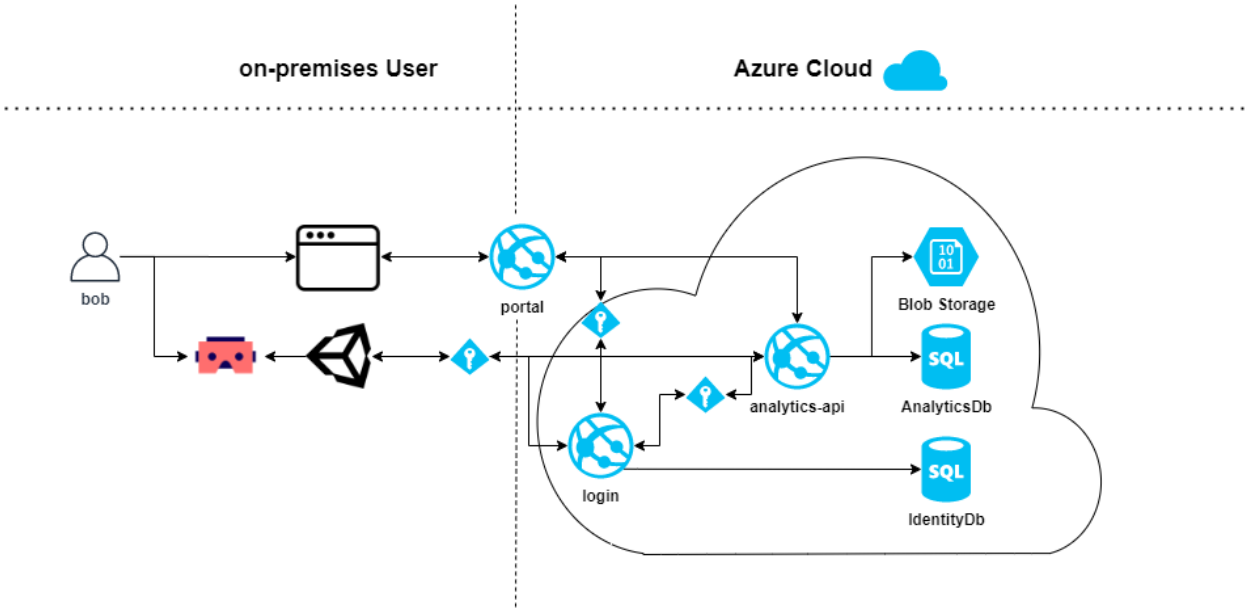
Appealling, huh?

We got you covered! Keep reading.

14.2 Architectural Overview

Before jumping to the details, let us first outline a deployment viewpoint of a VR application built with the MAGES SDK alongside with provided ORamaVR web services to get a clear picture.

Take a good look in the figure below.



14.2.1 User Perspective

On the left-side we depict a common deployment scenario from the perspective of the user w.r.t. your applications & services. Basically, this boils down to two things:

1. An HMD through which users access your VR applications
2. A typical web browser for access to user analytics

On the right-side we exhibit how web-services are deployed and exchange information on the Cloud. Namely, we present the following:

1. Web Portal
2. Login service
3. Analytics API

Combining these perspectives at a high-level overview there are two flows of information we are concerned with.

1. User authentication through the VR module and streaming of data (i.e., analytics) to the Cloud for further use.
2. Enable user to view the collection of gathered analytics at the online portal.

For 1., the VR module connects to the Login service which acts as a central authority and access controller (notice how every information flow goes through Login) that is responsible for user authentication and authorization based on the selected flow (e.g., resource owner password, SSO, etc.).

Additionally, the VR module connects to the Analytics API and uploads analytics at the end of the user session.

Regarding 2., the user gains access to the Web Portal through a regular browser and by authenticating via the Login service. Through the Web Portal, the user is able to view the analytics collected from each product and on a per session-basis.

From the scenario above we can deduce the following information regarding the web services:

Service	Description
Web Portal	Single Page Application (SPA) accessible through the browser. Responsible for displaying user analytics.
Login Service	Central identity management. Responsible for user authentication and authorization.
Analytics API	RESTful API. Responsible for processing, storing, and delivery of streaming data regarding user analytics.

14.2.2 Administrator Perspective

From an administrators perspective, there is a lot more going on.

For instance, admins need a way to let users register, or register them on-demand.

Another example is that admins need to be able to create product entities and associate these products with a group of users. In other words, hand-out licenses to users (e.g., userA is licensed to use productA, productB, etc.).

For these scenarios and more, the provided web services are able to cover your use-cases.

In detail and complementary to the table above:

Service	Description
Web Portal	Responsible for displaying user and product/license management.
Login Service	REST API, responsible for user management, authentication, and authorization.

14.2.3 Supervisor Perspective

Now lets assume that a group of users belong to the same organization, e.g., ORamaVR.

Lets say for example that we want a **supervisor** that belongs to the organization to be able to track user's progress, or even examine their analytics and conduct statistical evaluations.

You guessed it right, this is also part of the provided web services, with the following scheme:

Service	Description
Web Portal	Responsible for user management within a particular organization. Track user progress.
Login Service	REST API, responsible for user management within an organization.
Analytics API	REST API, responsible for granting access to supervisors.

14.2.4 What about the storage?

Now that you have a firm idea of what does what and how everything is connected, you might be wondering about what happens with all the data.

As presented in the figure above, you can see there are two types of storage systems our web services utilize:

1. Relational Databases (SQL)
2. Azure Storage Blobs

Both Login and Analytics API maintain a distinct Azure SQL database each for keeping relational data (e.g., Users, Products, SessionSummary, etc.).

Analytics API on the other hand, utilizes Azure Blob Storage for storing and processing streaming data from user sessions (i.e., analytics data).

While keeping relational data on a SQL database sounds reasonal, Azure Blob Storage is also a sound design decision for streaming data and large blob files.

Quoting from Microsoft Azure documentation:

- Azure Blobs Allows unstructured data to be stored and accessed at a massive scale in block blobs.
- Also supports Azure Data Lake Storage Gen2 for enterprise big data analytics solutions.
- You want your application to support streaming and random access scenarios.
- You want to be able to access application data from anywhere.
- You want to build an enterprise data lake on Azure and perform big data analytics.

14.2.5 Connecting the Dots

Communication between the services and VR modules takes place in an asynchronous HTTPS manner.

No information is exchanged in any other way. The same principle holds for the storage systems.

For instance, the Analytics API often requests user information for functioning properly, but it never directly gains access to the Identity database to retrieve this information.

By doing so, you might be wondering that we have added an unecessary communication overhead but in fact what we have done is beneficial to the overal architecture.

In detail, we have reduced the complexity of our system by providing single-points of responsibility amongst our services or in other words, seperation of concerns.

14.2.6 Further Reading

After this initial introduction you should have a brief understanding of the web services provided in MAGES SDK.

Now you can navigate to each separate section for the details and the practical stuff!

Login Service

Introduction

The Login service is a fundamental pilar that complements MAGES SDK.

As briefly described in the introductory section, all traffic from and to the VR module goes through this service.

Namely, the core functionalities that Login provides are the following:

- User management
- User authentication & authorization
- Product management & licensing system
- SingleSignOn (SSO) capabilities

Essentially, Login is an identity provider and access controller.

In this section we are going to explore how you can utilize the service for your needs, starting with the basics.

First, we are going to discuss how Login is packaged into a sample application, and proceed to outline the technical details (e.g., frameworks, depending third-party libraries, etc.).

Then, we will proceed to show how you can configure the Login service, run it, and deploy it on the Azure infrastructure (Cloud provider).

Finally, we will explain how you can customize it to serve your needs.

Note: If you wish to directly run the service without going over the details and explanations scroll down to Development Environment.

Sample App

The Login service is packaged as a Visual Studio 2019 project.

In this manner, we enable developers to directly configure and customize the codebase to tailor their needs.

Besides customization to the maximum, our solution is fully transparent as to what it does under the hood with the organization acquiring it.

Furthermore, it is easier for customers to explore alternative ways of solving the problems, and perhaps extending it to fit Cloud providers of their choice.

Requirements

At its core, Login service is a full-stack .NET Core 3.1 MVC web application including a RESTful API for HTTP calls.

Therefore, it is required that you download the latest .NET Core 3.1 SDK, and update your Visual Studio to 2019 version.

Additionally, the service itself has a series of 3rd party open-source dependencies that you can install through the NuGet Package Manager through Visual Studio.

Package	Description
IdentityServer4	IdentityServer4 is responsible for OpenIDConnect and OAuth2 support, as well as identity management.
MailKit	MailKit is responsible for the EmailService. In other words, you can configure it to send emails to your clients such as Account Confirmation, or Reset Password functionalities.
EF Core	EF Core is Microsoft's object-relational mapper, particularly useful to work with databases and treat them as .NET Objects.
ASP.NET Core Identity	ASP.NET Core Identity is Microsoft's user management package that enables user management, role, claims, tokens and everything user-related as well as a lot of out-of-the-box functionalities.

Note: You don't have to manually download or install required NuGet packages. Visual Studio handles this for you.

In any case, you can follow one of the methods below:

1. Right click on the *Solution* in the *Solution Explorer* and select **Restore NuGet Packages**
2. Right click on the *Solution* in the *Solution Explorer* and select **Manage NuGet Packages for Solution...**. Then, you can install them one-by-one via GUI.

Basic Configuration

Now let's dive into the basics.

In case you haven't already, proceed to open the Visual Studio project and load the NuGet packages.

Open the Startup.cs file, in the root project folder.

Startup.cs

Startup is perhaps the most fundamental class in an ASP.NET Core project.

In this file we configure all services via Dependency Injection (DI) that will be used at runtime.

Also, in Startup we provide configurations for 3rd party packages, routing, middlewares, authentication, etc.

Read more about App startup.

Constructor

As with any C# class, Startup starts with the constructor. Take a look at the one below:

```
public Startup(IWebHostEnvironment environment, IConfiguration configuration)
{
    var builder = new ConfigurationBuilder()
        .SetBasePath(environment.ContentRootPath)
        .AddJsonFile("appsettings.json", optional: true, reloadOnChange:␣
↪true)
        .AddJsonFile("appsettings.Development.json", optional: true);
    builder.AddEnvironmentVariables();

    Environment = environment;
    Configuration = builder.Build();
}
```

In the above snippet, all we do is specify a few of the basics regarding the environment and configuration.

Based on the `ASPNETCORE_ENVIRONMENT` environment variable and whether its value is `Development` or `Production` a different `appSettings.json` loads.

Note: The `appSettings.json` is responsible for runtime configurations and we explore it further down.

Configure Services

In the Configure Services function we provide the specifics for all services and the connection blocks for external ones.

More importantly, in this function we declare all services that can be injected (DI) to other modules (i.e., Controllers, Services, etc.) at run-time.

Read more about DI in ASP.NET Core here.

```
1  public void ConfigureServices(IServiceCollection services)
2  {
3      string connectionString = Configuration.GetConnectionString("Identity");
4      var migrationsAssembly = typeof(Startup).GetTypeInfo().Assembly.
   ↪GetName().Name;
5
6      services.AddControllersWithViews();
7
8      services.AddDbContext<ApplicationDbContext>(options =>
9          options.UseSqlServer(connectionString));
10
11     services.AddIdentity<ApplicationUser, IdentityRole>()
12         .AddEntityFrameworkStores<ApplicationDbContext>()
13         .AddDefaultTokenProviders();
14
15     services.Configure<IdentityOptions>(options =>
16     {
17         options.Lockout.MaxFailedAccessAttempts = 20;
18         // Password Settings.
19         options.Password.RequireNonAlphanumeric = false;
```

(continues on next page)

(continued from previous page)

```
        options.Password.RequireDigit = false;
        options.Password.RequireUppercase = false;
        // Email Settings.
        options.User.RequireUniqueEmail = true;
        // SignIn Settings.
        options.SignIn.RequireConfirmedEmail = true;
    });

    services.Configure<CookiePolicyOptions>(options =>
    {
        options.MinimumSameSitePolicy = SameSiteMode.Unspecified;
        options.Secure = CookieSecurePolicy.SameAsRequest;
    });

    var builder = services.AddIdentityServer(options =>
    {
        options.Events.RaiseErrorEvents = true;
        options.Events.RaiseInformationEvents = true;
        options.Events.RaiseFailureEvents = true;
        options.Events.RaiseSuccessEvents = true;

        // see https://identityserver4.readthedocs.io/en/latest/topics/
↪resources.html
        options.EmitStaticAudienceClaim = true;
    })
        .AddConfigurationStore(options =>
        {
            options.ConfigureDbContext = b =>
            {
                b.UseSqlServer(connectionString, m =>
                m.MigrationsAssembly(migrationsAssembly));
            };
        })

        .AddOperationalStore(options =>
        {
            options.ConfigureDbContext = b =>
            {
                b.UseSqlServer(connectionString, m =>
                m.MigrationsAssembly(migrationsAssembly));
            };
        })
        .AddAspNetIdentity<ApplicationUser>();

    builder.AddDeveloperSigningCredential();

    services.AddAuthentication()
        .AddGoogle(options =>
        {
            options.SignInScheme = IdentityServerConstants.
↪ExternalCookieAuthenticationScheme;
            options.Scope.Add("email");
            options.ClientId = "Your Client Id";
            options.ClientSecret = "Your Secret";
        });

    services.AddLocalApiAuthentication();
```

(continues on next page)

(continued from previous page)

```
75
76      services.AddAuthorization(options =>
77      {
78          options.AddPolicy("Admin", policy =>
79          {
80              policy.RequireAuthenticatedUser();
81              policy.RequireClaim("role", "Admin");
82          });
83          options.AddPolicy("Supervisor", policy => {
84              policy.RequireAuthenticatedUser();
85              policy.RequireClaim("role", "Supervisor");
86          });
87          options.AddPolicy("User", policy =>
88          {
89              policy.RequireAuthenticatedUser();
90              policy.RequireClaim("role", "User");
91          });
92      });
93
94      services.AddHostedService<BackgroundCheckoutService>();
95      services.AddScoped<IProfileService, ProfileService>();
96      services.AddScoped<LicenseValidationService>();
97      services.AddSingleton<IEmailConfiguration>(Configuration.GetSection(
    ↪"EmailConfiguration").Get<EmailConfiguration>());
98      services.AddTransient<IEmailService, EmailService>();
99
100     services.AddMvc(option => option.EnableEndpointRouting = false)
101         .SetCompatibilityVersion(CompatibilityVersion.Version_3_0)
102         .AddNewtonsoftJson(opt => opt.SerializerSettings.
    ↪ReferenceLoopHandling = ReferenceLoopHandling.Ignore);
103 }
```

Let's take a look at the important bits on a configuration-basis.

DbConfiguration

Between `lines 8-9` we specify the Database Context (i.e., ApplicationDbContext) that points to our SqlServer. For this we need the connection string, whether it is a local database or a database on Azure, that we obtain from the AppSettings.json file in `line 3`

We use the same database for the Configuration and Operational store required by IdentityServer4 as shown between `lines 44-60`.

Identity & IdentityServer4

For user management we will be using ASP.NET Identity which provides out-of-the-box a user management store, a signin manager, role-based authorization, and more.

First, we proceed to include and configure ASP.NET Identity to our project in `lines 11-13`.

Then, we need to specify the user class model that represents our final users. This is the `ApplicationUser` model included under the `Models/` directory. See how `ApplicationUser` extends the original `IdentityUser` and introduces more properties that we want to keep track of.

For instance, the Country of origin for the user, and the `Tenant` (i.e., Organization) the user belongs to.

In addition, we have to specify the database that will store our users. In this case, this the `ApplicationDbContext` SQL database we configured in the previous paragraph.

Further down, between `lines 15 and 26` we specify some Identity options for user management.

For example, we specify that each user should have a unique email address (prevents duplication of user accounts on an email basis), and that we do not require any special characters in the password (**consider altering this to prevent users from using weak passwords**).

In lines `lines 34-61` we proceed to add IdentityServer4 to our services. There we specify the Configuration and Operational store, which is the same database in practice, and more importantly we specify that IdentityServer4 will support out-of-the-box `AspNetIdentity` and the `ApplicationUser` we declared earlier on.

Role-based Authorization

Role-based authorization is an important part of user management.

Essentially, we provide different level of access scopes to different users based on their associated roles. `AspNetIdentity` supports role-based authorization out-of-the-box.

That being said, in `lines 76-92` we define three different policies.

1. Admin
2. Supervisor
3. User

Role-based access works in an inclusive and not in an exclusive manner.

In other words, an Admin has also the roles of Supervisor and User. A supervisor, has also the role of User. And finally, the User only contains User.

This means that Admins can access all available functionality – scope-wise, whilst the other two are restricted.

For example, deleting a user through an API call is only available if the user who invokes the API call has the role of an Admin. Therefore, this functionality is Unauthorized (401) for Supervisors and Users.

SingleSignOn (SSO)

As an example for SSO support, we provide the snippet between `lines 67-72`.

There you need to fill in the `ClientId` and `ClientSecret` that you obtained from Google, to enable access to users to your service through their Google accounts.

Note: There is also support for other services, but you will have to configure that manually.

Read more about it at Google.

Services

Finally, we have to declare our own custom services in a similar manner to provide DI support at runtime. In this manner we also configure the *lifetime* per service.

In particular, between `lines 94-98` we configure the services found under `/Services` directory.

Pay attention to how services have different *lifetimes* depending on the declaration.

For instance, the EmailConfiguration is declared as a Singleton and the *lifetime* is the same as the application and wherever it is injected, e.g., in some controller, the same object is injected.

On the other hand, the EmailService itself is declared as a transient. This means that wherever EmailService is injected throughout an HTTP request, EmailService will be reinstantiated for this exact declaration. Finally, scoped services live throughtout a single HTTP request.

More information for service lifetimes.

Configure

The Configure method specifies the request pipeline, in other words, the middlewares involved whenever a request reaches the application.

In the snippet below we specify a typical Configure method:

```
public void Configure(IApplicationBuilder app)
{
    app.UseCookiePolicy(new CookiePolicyOptions {
        MinimumSameSitePolicy = SameSiteMode.None
    });

    app.UseCors(options =>
    {
        options.AllowAnyOrigin();
        options.AllowAnyHeader();
        options.AllowAnyMethod();
    });

    if (Environment.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
    }

    app.UseStaticFiles();

    app.UseRouting();
    app.UseIdentityServer();
    app.UseAuthorization();
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapDefaultControllerRoute();
    });
}
```

The code in the snippet is pretty much self-explanatory, we the only difference that the order middlewares are specified in the request pipeline **matters**.

For instance, `app.UseEndpoints` function is typically the last one to be declared after authorization, routing, and the rest of the middlewares have been set to ensure traffic goes to specified controllers.

Read more about the Configure method.

MVC

MVC ASP.NET Core applications are based on the Model-View-Controller pattern, which is perhaps the easiest and most fundamental pattern involving UI, data-layers, and business logic under one unified application.

The Login service effectively utilizes this pattern, and specifically for login flows and operations that have to be done **on-site**.

For example, the common Login page where users log into the service is part of the MVC with the structure:

1. Models are under `Controllers/Account/*`
2. Controller is under `Controllers/Account/AccountController.cs`
3. Views are under `Views/Account/*`

You can observe that the AccountController supports more than one functionality, and as the name suggests, operations that naturally are associated with Users and their accounts.

For instance, through the `AccountController` we support the following functionalities:

- User Login
- User Registration
- Forgot Password
- Reset Password
- etc.

While each of these operations are bundled for efficiency under a single Controller, almost everyone necessitates a different View. The same holds true for the Models.

Note: The `AccountsController` inherits from the base class `Controller`.

Controller

Routing

Routing for an MVC Controller, unless explicitly specified, inherits the controller's name.

For example, if your service is running at `http://localhost:5002`, then `AccountController` routes at:

- `http://localhost:5002/Account`

In addition, routing to a specified Action (i.e., public Controller functions that represent HTTP requests), unless explicitly specified, is as follows:

- `http://localhost:5002/Account/{ActionName}`

For instance, to navigate to the Login page you need to enter the following endpoint at your browser:

- `http://localhost:5002/Account/Login`

HTTP Methods

Moreover, HTTP verbs are defined as attribute on top of Actions.

As an example, take the Login Action below:

```
[HttpGet]
public async Task<IActionResult> Login(string returnUrl)
{
    var vm = await BuildLoginViewModelAsync(returnUrl);
    if (vm.IsExternalLoginOnly)
    {
        return RedirectToAction("Challenge", "External", new { scheme = vm.
↪ExternalLoginScheme, returnUrl });
    }

    return View(vm);
}
```

See that the specified Action is declared as a GET method, and will return its associated View (i.e., Login.cshtml).

On the other hand, we can also proceed to specify the POST Login method in a similar manner:

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Login(LoginInputModel model, string button)
{
    // Some code that actually handles the LoginInputModel and logs in a␣
↪User.
}
```

Note: Notice that both Actions have the same name and return types, only Action parameters change.

Therefore, a typical flow would normally start from the GET Login at the browser, where the Login screen is returned to the User, and finish when the user enters his credentials at the displayed form, which will be posted at the POST Login Action.

Authorization

Level of access can be declared for the whole Controller, or can be added on a per-action basis. Typically you can mix and match policies inside the Controller, according to your needs.

In the `AccountController` above, there is no globally defined level of access. Therefore, Actions such as Login and Register inherit by default the `[AllowAnonymous]` attribute and can be accessed through unauthenticated Users, which makes sense.

On the other hand, Actions as is the ChangePassword are allowed only for logged in Users who have at least the role `"User"` in their claims.

See the example below:

```
[HttpGet]
[Route("account/password")]
[Authorize(Roles = "User")]
public IActionResult ChangePassword(string returnUrl = null)
{
    return View();
}
```

Note: Observe also how this particular Action has a different routing defined and is accessible at `/account/password`.

Model & View

In the code snippet above that involves the POST login Action, notice how the specified input model is provided as a parameter.

Further, take a look at the associated login View below:

```
@model LoginViewModel

<div class="login-page">
    <!-- rest of the elements -->

    <!-- FORM -->
    <form asp-route="Login">
        <input type="hidden" asp-for="ReturnUrl" />
        <div class="form-group">
            <label asp-for="Username"></label>
            <input class="form-control" placeholder="Username" asp-for=
↪"Username" autofocus>
        </div>
        <div class="form-group">
            <label asp-for="Password"></label>
            <input type="password" class="form-control" placeholder="Password
↪" asp-for="Password" autocomplete="off">
        </div>
        @if (Model.AllowRememberLogin)
        {
            <div class="form-group">
                <div class="form-check">
                    <input class="form-check-input" asp-for="RememberLogin">
                    <label class="form-check-label" asp-for="RememberLogin">
                        Remember My Login
                    </label>
                </div>
            </div>
        }
        <div class="row mt-4">
            <div class="col-md-12 mb-3 text-center">
                <button class="google-button" name="button" value="login">
                    <span class="google-button__icon">
                        <img src="~/VR.png" class="oramavr-button-icon" />
                    </span>
                    <span class="google-button__text oramavr-button">Sign in␣
↪with ORamaVR</span>
```

(continues on next page)

(continued from previous page)

```
                </button>
            </div>
        </div>
    </form>
</div>
```

Let's focus on the important bits.

First, notice how the linked model `LoginViewModel` is declared on the top of the page.

Second, notice how the form is associated with the `asp-route="Login"` action.

Finally, each form-control input is associated with the appropriate property of our data model.

API Controllers

On the other hand, we have the `ApiControllers` to provide an HTTP RESTful API for User and Product management through the Portal.

ApiControllers are under `/Controllers/API/*` directory.

The main difference is of course that an `ApiController` is not associated with any View, and does not serve HTML pages to users, but rather HTTP Responses.

In terms of code, here is a following example from the `UsersController`:

```
[Route("api/[controller]")]
[ApiController]
[Authorize(IdentityServerConstants.LocalApi.AuthenticationScheme)]
public class UsersController : ControllerBase
{
    private readonly ApplicationDbContext _context;
    private readonly UserManager<ApplicationUser> _userManager;
    private readonly RoleManager<IdentityRole> _roleManager;


    public UsersController(
        ApplicationDbContext context,
        UserManager<ApplicationUser> userManager,
        RoleManager<IdentityRole> roleManager)
    {
        _context = context;
        _userManager = userManager;
        _roleManager = roleManager;
    }

    // ------------------------------------------------------------ GET ------
↪-------------------------------------------------------
    [HttpGet]
    [Authorize(Policy = "Admin")]
    public async Task<IActionResult> Get()
    {
        var users = await _userManager.Users.ToListAsync();
        if (users == null)
        {
            return NotFound();
        }
```

(continues on next page)

(continued from previous page)

```
        return Ok(users);
    }

    // Rest is omitted for simplicity
}
```

Compared to MVC

The first few obvious differences regarding `UsersController` are the following:

- Routing is explicitly specified at `/api/Users` to differentiate from MVC Controllers.
- The `[ApiController]` attribute is utilized that provides certain features.
- And the `[Authorize(IdentityServerConstants.LocalApi.AuthenticationScheme)]` is also utilized to ensure Clients who make API calls have the registered scoped.
- The controller itself inherits from `ControllerBase` this time.

Apart from those differences, let's take a look at the first GET functionality which returns all Users to the caller.

First, in this controller you can make sense of the DI pattern we discussed earlier.

In detail observe how :code`ApplicationDbContext, UserManager, RoleManager` are injected into the Constructor of the UsersController for use throughout the application.

Then, take a look at the specific GET call.

In this case, we have specifically declared that this is an HTTP GET call through the use of the `[HttpGet]` attribute.

Moreover, we have restricted access to Admins only through the `[Authorize(Policy = "Admin"]` attribute.

Note: We have not defined any explicit routing for this method. Therefore, requests fired at `http://{service}/api/Users` will invoke this method.

If the User is not authorized to do so, an HTTP 401 Unauthorized response will return to the client.

What it does

Essentially, the `Get()` call querys the database and returns a list of all `ApplicationUser` users. In this case, we could have also used the `ApplicationDbContext => _context` to query the database, but we prefer to utilize the `UserManager` API provided by `AspNet.Identity` for user related operations.

Now, if users are null, an HTTP 404 NotFound response is sent back to the client.

In all other cases, an HTTP 200 Ok response with the list of users is returned.

Note: You can access and actively test all available API calls through Postman Client. But remember to authorize yourself first.

Database Migrations

Introduction

We utilize Entity Framework Core (EF Core) as an object-relational mapper (O/RM).

This enables us to work directly with .NET objects that relate directly to our database tables and the underlying schema.

EF Core has two ways of managing the database schema.

In our case we utilize the code-first approach and work with Data Migrations to keep the database up-to-date.

Simply speaking, if we add a new property to the `ApplicationUser` we can add a new migration to that will update the database table accordingly.

The Login service has already certain migrations for your needs that correspond to all models found in the database.

Current database models correspond to the ones under `Models/*` directory and have certain attributes that define relationships between other objects.

In addition, certain relationships have to be defined at the `ApplicationDbContext` with the `OnModelCreating` method, as attributes do not cover more complex relationships.

Add New Migration

You can add a new migration through the NuGet Package Manager console simply as follows:

```
Add-Migration {MigrationName} -Context ApplicationDbContext -OutputDir "/
↪Migrations/ApplicationDb/"
```

Moreover, if the above command builds successfully, you can proceed to update the database with the following command:

```
Update-Database
```

You can also skip this step and read the next section.

Existing Data and Applying Migrations

For convience and ease of use, we provide certain data that will be applied the first time you run the Login Servive.

The code will also ensure that the database is created and all migrations are applied.

This code lies onto the `Program.cs` file under the root project directory.

Particularly, this code is inside the `Main` method and will be invoked as soon as you start running the service and access one of the endpoints or do an API call.

Responsible for this is the helper class is `SeedDataHelper` under `Helpers/*` directory.

The helper class will make sure to add certain dummy users and organizations, as well as a few products and link them with the tenants.

More importantly, it will create all necessary Clients, ApiResources and Scopes required for IdentityServer4 and the services (AnalyticsAPI, Portal, VR module) to communicate.

The initial data for IdentityServer4 resources are under `Config.cs`.

> **Danger:** It is strongly suggested that you alter the `ClientSecrets` for each `Client`, especially in Production database.

Getting Ready for Development

Assuming you read the previous sections and obtained a basic understanding of how everything works, in this section we will provide all necessary configurations you have to follow before you start play-testing around the Login service.

1. Local SQL Database

First you need to create a local SQL Database.

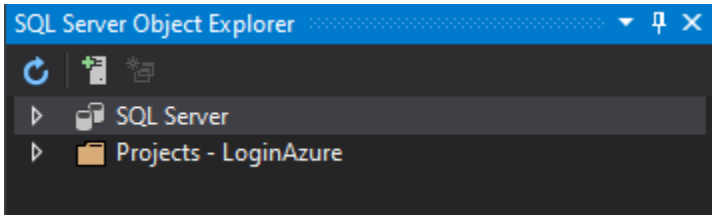
The easiest way to do so is with SQL Server Express LocalDB. There are two options to install LocalDB onto your machine.

1. One is to download and install separately from the link above.
2. Through Visual Studio Installer as described in Installation Media.

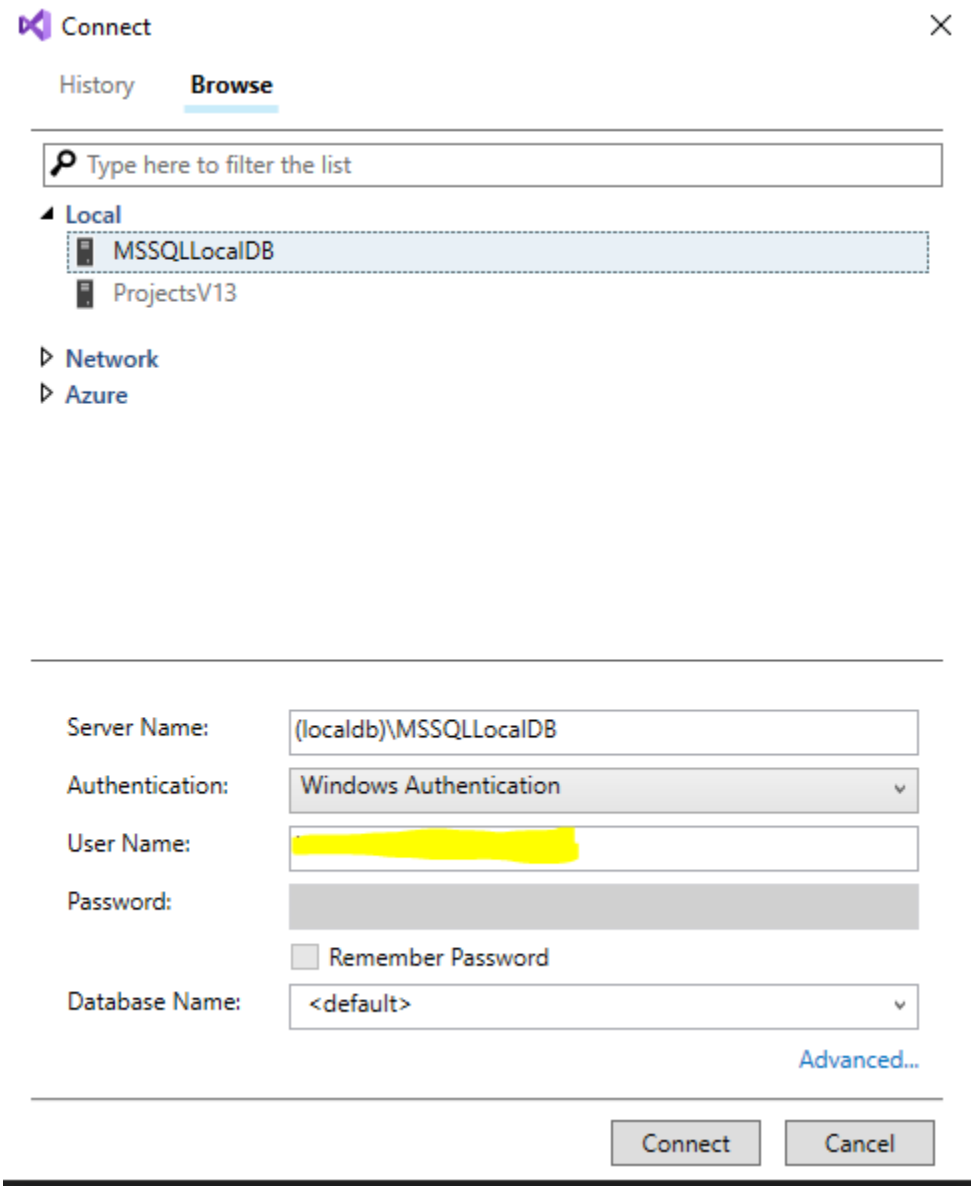
Then, you can manage the SQL Connection to the LocalDB through Visual Studio 2019.

To do so, click on the top toolbar **View** and then click on **Sql Server Object Explorer**.

This will open a panel as follows:

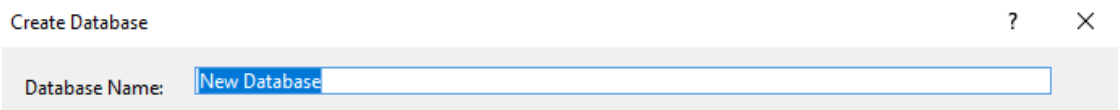


On the top bar click on the **Add SQL Server** button and add select from the **Local** list the LocalDB SQL Server as in the figure below:



Click connect and proceed to expand the **SQL Server** tree structure on the SQL Server Object Explorer as follows:

Right click on the Databases icon and select the **Add New Database** option.



Give a descriptive name for your Database (something like *IdentityDb*) and click *Ok*.

Now you have created your Database but it is empty. Don't worry we will populate it soon, keep reading!

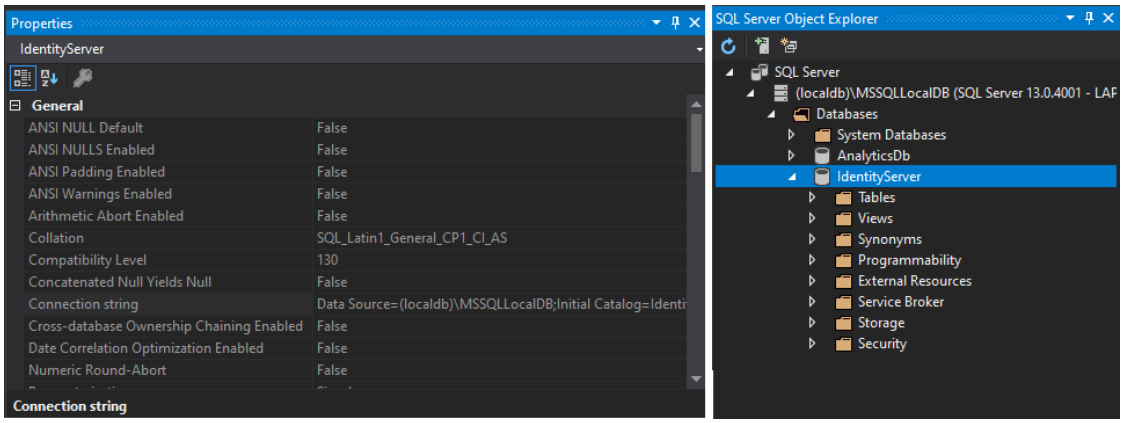
2. App Settings

DB Connection String

After the database is set you need to obtain the connection string.

The easiest to obtain the connection string is through the SQL Server Object Explorer.

Click on the database, and then on the **Properties window** you will find the Connection String as in the figure below.



Proceed to open the file **appSettings.Development.json** and paste it inside the **Connection Strings/Identity** as in the example snippet below:

```
"ConnectionStrings": {
    "Identity": "Server=(localdb)\\mssqllocaldb;Database=IdentityServer;
↪Trusted_Connection=True;MultipleActiveResultSets=true"
}
```

Email Configuration

While in the **appSettings.Development.json** file, proceed to also include the configuration for your EmailService.

This is needed for user account operations that necessitate sending an email to the User.

For instance, *forgot password* functionality.

```
"EmailConfiguration": {
  "SmtpServer": "",
  "SmtpPort": 587,
  "SmtpUsername": "",
  "SmtpPassword": "",
  "PopServer": "",
  "PopPort": 995,
  "PopUsername": "",
  "PopPassword": ""
}
```

If you need to configure extra TLS or other security options, you can do so in the `EmailService.cs` file under `Services/` directory.

Optional Step. You can configure the email from, subject, and content fields in the AccountController and UsersController, where the EmailService is invoked and Emails are sent.

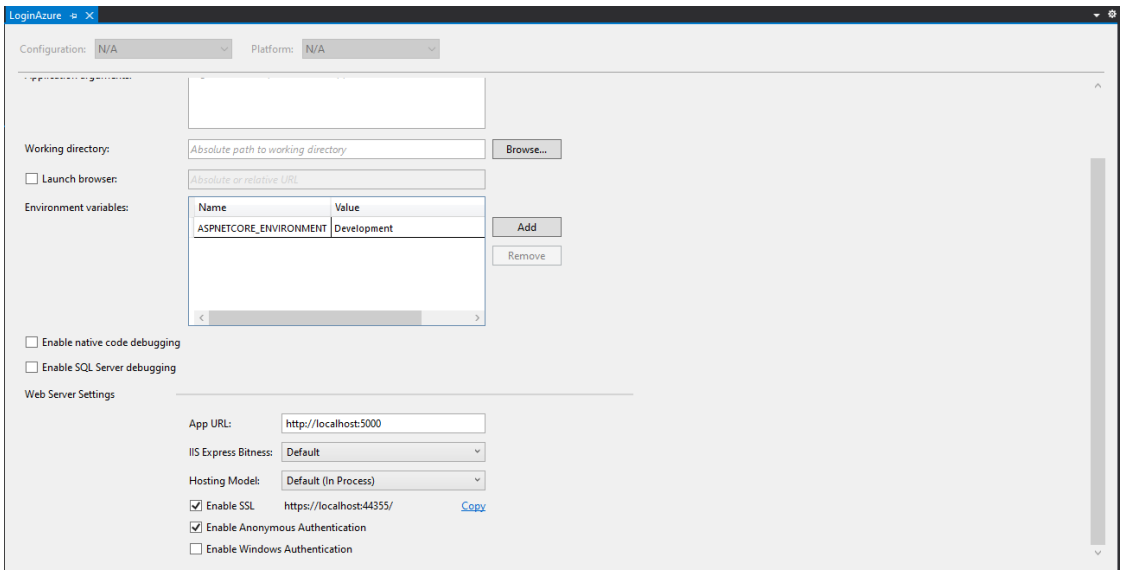
3. (HTTPS) Self-signed Certificate

Now that all configurations are set, we need to set a self-signed developer certificate before running our service.

This is because IdentityServer4 by default prompts to encrypted traffic (HTTPS) and in production won't work otherwise.

So for development purposes, Visual Studio and IIS will generate a self-signed certificate for us.

First, right-click on the Project and click on the *Properties* option.



Scroll down and make sure the **Enable SSL** checkbox is ticked! If it is not, check it.

Note: This is also the default development environment endpoint to access the service `https://localhost:44355`.

When you check it for the first time, a window prompt will appear asking you to create a self-signed certificate that will be stored on your computer.

Click **Agree** on all dialogs, and you are set!

4. Start the Service

Assuming you followed all previous steps successfully and everything is set, you are ready to hit the start button.

Note: Make sure IIS profile is selected and that Debug mode is set on the top bar.

Hitting the **Start** button will fire up Login service at the specified endpoint in Step 3.

```
https://localhost:44355
```

When you point your browser the first time at this URL, it might be a bit slower than expected. This happens mainly due to two reasons:

1. ASP.NET Core slow-start
2. The *SeedDataHelper.cs* script will be invoked and subsequently:

   2.1. Will ensure all Tables are initialized.

   2.2. Migrations are applied.

   2.3. First dummy data will be inserted to the database.

After this cold start you will be navigated to the Login page. Login with one of the predifined users at **SeedDataHelper.cs** and then you can go back to the Home page and see the default page of IdentityServer4 which is useful for debugging user claims, etc.

Note: The Default HomePage of IdentityServer4 is only for the development environment. This page will not be visible when you deploy to Azure.

If everything went fine at this point, you are set. Most likely you won't experience any strange behaviors.

Note: Make sure you test thoroughly all supported functionality (e.g., registration page, forgot password, etc.).

Getting Ready for Production

Assuming you followed all steps in the previous section, and everything is working perfect locally, it's about time to deploy.

Below we outline all necessary steps and precautions before you do so.

> **Warning:** The deployment environment is configured for Azure Cloud provider.

1. App Settings

If you recall from the previous section, we modified the `appSettings.Development.json` file for working locally.

This time, we will have to modify in the same manner `appSettings.json` for use in the deployment.

As before, proceed to insert Database connection string in the `ConnectionStrings: Identity`, this time with the connection string from the live database in Azure.

Note: (Optional), You can download SQL Server Management Studio (SSMS) to actively monitor your databases and input data.

Analytics API

Introduction

The analytics API service of ORamaVR is built using ASP.NET Core 3.1. similarly to the *Login service*.

It serves as a web API and relies on the latest technologies offered by Microsoft (e.g., ASP.NET Core, Azure Blob Storage, etc.).

To leverage easiness of deployment, native support, and robust scaling, our platform relies on the Azure Cloud infrastructure services to operate. Therefore, changes to the existing codebase are easily deployable and tested through the well-known development environments of Microsoft and subsidiary products and frameworks (e.g. Visual Studio, Visual Studio Code, etc.).

All analytics data are saved on our cloud system. This system is based on Microsoft Azure Blob Storage System. The structure along with the files themselves are presented in detail in the following sections.

Purpose

Analytics API is an independent service from Login. This is mainly due to separation of concerns and reducing the overall complexity by decoupling business operations.

Additionally, Analytics API is computationality expensive since it is responsible for storing, processing, and retrieving user's gathered analytics from VR modules. Therefore, decoupling the services reduces the overhead from Login which involves mostly CRUD operations.

Moreover, if there is downtime with Analytics API, this does not prevent user's from accessing your VR modules (i.e., checking out a license).

Omitted information

The project structure of Analytics API is very similar to the *Login service*.

The main difference is that Analytics service is a pure API project, without any associated Views or in other words does not follow the .NET MVC pattern. There are no web pages deployed, only an HTTP API that supports GET, POST, PUT operations, etc.

That being said, certain information about the project structure is omitted, to keep the documentation from bloating. You can read more about ASP.NET Core in the *Login service* documentation.

Connecting To Identity

To ensure users who request data from AnalyticsAPI are authenticated and authorized, AnalyticsAPI delegates Bearer tokens to the LoginService for verification.

This connection is specified at the Startup class. More specifically:

```
JwtSecurityTokenHandler.DefaultInboundClaimTypeMap.Clear();
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, options =>
    {
        options.Authority = IdentityServerUrl;

        options.RequireHttpsMetadata = false;
        options.SaveToken = true;

        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateAudience = false
        };
    });
```

The IdentityServerUrl derives from the `appSettings.*.json`.

Retrieving user data

Moreover, AnalyticsAPI communicates with Login to requester user-related data.

For instance, which products a user/organization owns, or user and organization information.

To do so, we have an `AuthenticationDelegationHandler` class that delegates the JWT Bearer token from the original user request into the subsequent request to the Login service, therefore delegating authorization and making the request on behalf of the user.

Thus, communication is achieved in an HTTP manner (Request/Response).

Azure Blob Storage

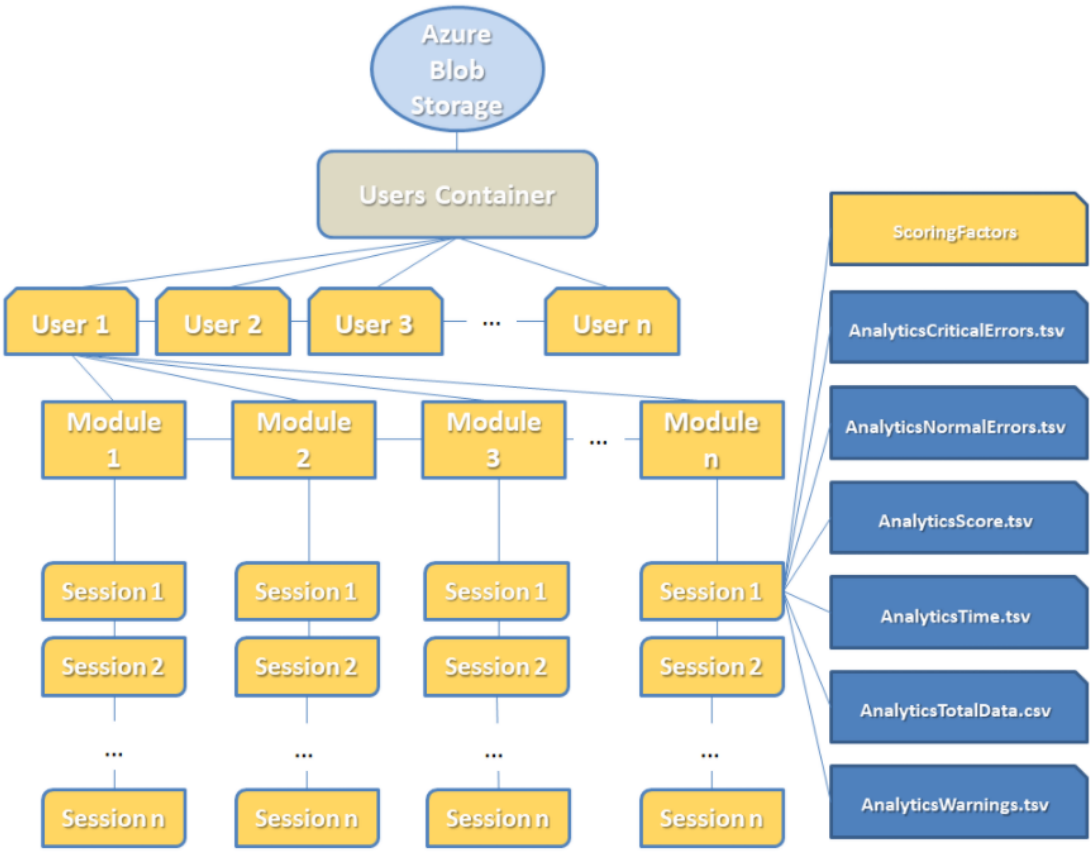
Azure Blob Storage serves as our cloud storage system. Specifically, we utilize Azure Blob Containers to save the user analytics on the cloud, to be able to load them on the analytics platform at a later stage. The users' analytics are uploaded at the end of each operation to the specified container in the form of blobs.

We mainly utilize Block Blobs for the analytics data and Append Blobs for the analytics meta data (i.e. date, average score, time etc.).

Note: Analytics meta data are generated automatically during the Upload request and stored in a "meta-data" folder, in the user's container path. There is one meta data Append Blob per ORamaVR operation.

Storing Data in Azure Containers

The main hierarchy of the data we store in Azure Blob Storage system is visible in the image below.



User related data available on our platform consist of score and performance metrics based on our modules.

Note: Data is stored per user session.

Inside Users container the following structure exists:

User Folders: One folder per user. Each of these folders contains all necessary files of their respective user progress.

Module Folders: Each user folder contains one or more modules folder. Module folders are named after respective module names. Module folders are generated when user runs a module for the first time.

Sessions Folders: These folders are contained inside their respective module folder. Each of the session folders represents a single user session of the module. A session folder is created when the user finishes a complete playthrough of the specific module. Generally, we store the following data for each user:

1. Number of critical errors in each module session and the name of the action, where they occurred.
2. Number of non-critical (or normal) errors in each module session and the name of the

action, where they occurred.

3. The score for each action in each module session.
4. The time that the user needed for each action in each module session, measured in seconds.
5. The total data (all errors, critical errors, warnings, final score) for each module session.

Upload Post Request & Azure Blob System

The parameters of the Upload Post Request and their purposes for the Azure Blob Storage System are the following:

Parameter	Purpose
Username	The username of the current user, specifying the folder name in the storage container root.
Operation	The name of the product/operation that the user played, indicating the folder name where the analytics will be uploaded in the user folder, for this product.
files	The files containing the analytics, which will be placed inside the operation folder (indicated by the operation parameter) in the appropriate folder for the specific session.

SQL Database

Besides Azure blob storage, which is particularly useful for handling loads of data and multitude of files (blobs), we also have a separate database mainly for keeping track of records for how many sessions a user has played, or summaries for user sessions.

Keeping track of these kind of data in a database is very convienient, instead of navigating through Azure blobs and processing lots of data everytime to recover the last user session on a specific product.

Migrations to the database and table schemas are applied in a similar manner as the Login, EF Core, code-first.

So, whenever you start AnalyticsAPI, if there are any pending migrations, they will be applied.

Getting Ready for Development

1. Local SQL Database

As described in the section above, you need to create a local SQL database.

If you don't know how, you can navigate to Login – Local SQL Database to see the steps outlined there.

The recommended name is `AnalyticsDb` but you can alter it to suite your needs.

2. Azure Storage

Unfortunately, at the time being we do not have an alternative solution for Azure Blob Storage. Therefore, even in development environment, you have to create a Storage Account at Azure.

Note: The recommended *Storage Account Type* that supports Blobs is General-Purpose V1 or General-Purpose V2 (V2 is not fully tested yet).

After creating the Storage Account, proceed to create a **private Container** with the name `user-analytics`.

> **Warning:** Do not change the container name.

3. App Settings

Next, we need to configure the appSettings for the development environment.

Open the `appSettings.Development.json` file and you will see similar configurations as in the snippet below:

```
{
    "AppSettings": {
        "LogsStorageConnectionString": "",
        "DataStorageConnectionString": ""
    },
    "IdentityServer": {
        "IdentityUrl": ""
    },
    "ConnectionStrings": {
        "AnalyticsDb": ""
    },
    "Logging": {
        "LogLevel": {
        "Default": "Information",
        "Microsoft": "Warning",
        "Microsoft.Hosting.Lifetime": "Information"
        }
    }
}
```

Proceed to fill in the database connection string as in Login – Local SQL Database in the `ConnectionStrings:AnalyticsDb` section.

Then, fill in the Login (IdentityServer) url in the `IdentityServer:IdentityUrl` section. For instance, `https://localhost:44355`.

Finally, in AppSettings, we set the Azure Storage Container's connection strings.

1. Retrieve the Connection string from the Azure Storage Account and fill it in the `DataStorageConnectionString` section.
2. (Optional) The `LogsStorageConnectionString` is for storing logs.

4. Start the Service

By default, AnalyticsAPI runs on `http://localhost:5002`.

Unlike Login, and for testing purposes, you don't need an TLS/SSL (HTTPS) Certificate.

> **Warning:** A TLS/SSL certificate is necessary in production environments.

If you followed all steps so far, you are ready to hit the Start button.

> **Danger:** Login has to be running in order to validate incoming JWT tokens!

Now you can test all API endpoints.

Note: The easiest way to populate your API with data is by uploading analytics from Unity!

Getting Ready for Production

1. App Settings

If you recall from the previous section, we modified the `appSettings.Development.json` file for working locally.

This time, we will have to modify in the same manner `appSettings.json` for use in the deployment.

As before, proceed to insert Database connection string in the `ConnectionStrings:AnalyticsDb`, this time with the connection string from the live database in Azure.

The Azure blob storage connection strings remain as is.

Finally, make sure you point to the live Login service!

Portal

Introduction

The Web Portal is a Single Page web Application (SPA) which combines all operations from Login and AnalyticsAPI, in an elegant UI.

Based on Authorization levels, the Portal gives Admins a wide overview of their operations regarding Licensing, Products, User Management, etc.

On behalf of Supervisors and Users, the Portal provides an overview of their analytics, scores, sessions etc.

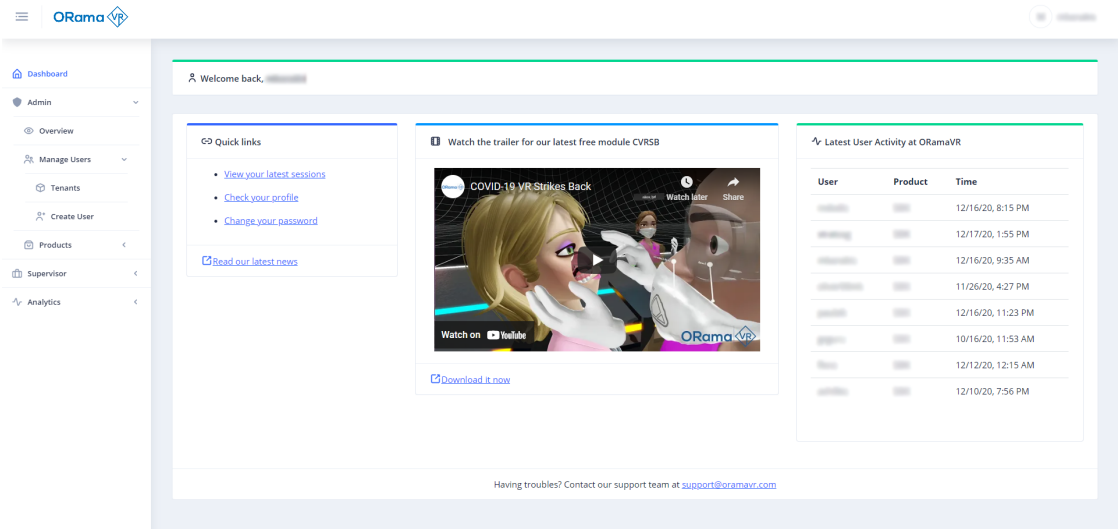
In addition, there are pages that are not restricted on access level, such as the Profile page where users can change their password, modify their name and country, or the Dashboard.

Sample App

Portal comes bundled as an Angular 9+, web application.

In addition, we utilize the open-source NGX-Admin and Nebular UI kit from Akveo, for the default styling and components they provide.

Here is an example of the Portal dashboard:



Getting Ready for Development

1. Angular Requirements

To get up and running, first you need to install Node.js, npm package manager, Angular, and the Angular CLI.

Typically, npm comes installed with Node.js, but for Node.js you have to download it and install it manually.

(Optional) Here is how to get started from the official Angular documentation.

Otherwise, follow the steps below:

1. Proceed to download an LTS version of Node.js.
2. After you finish installing, open up a terminal window and type the following command:

   `npm -v` (This will depict whether you have npm client installed!)
3. Finally, proceed to install Angular CLI with the following command:

   ```
   npm install -g @angular/cli
   ```

Note: We highly recommend Visual Studio Code for development in Angular.

Consider giving it a try!

---

2. Node Modules

After Angular CLI is successfully installed, you can `cd` to the directory the Portal (AnalyticsClient) is located.

There, (at the root folder) you need to type the following command:

```
npm install
```

This will create a folder named `node_modules` where all third party packages and dependencies are stored.

The dependencies are specified inside the file `package.json` in the root directory.

> **Warning:** Do not source control `node_modules` directory! Each developer should have his own local copy of the packages!

3. Environment Variables

Assuming all packages are downloaded correctly, you need to specify certain environment variables.

For the development environment, open the file `environment.ts` under `src/environments`.

```
export const environment = {
    production: false,
    identityServer: 'https://localhost:44355',
    analyticsAPI: 'http://localhost:5002',
    clientId: '',
    scopes: '',
    responseType: '',
};
```

There you need to place the localhost Login service URL, and the same for the AnalyticsAPI.

Additionally, you need to specify the IdentityServer4 Client required for the Portal to authorize with the Login service.

If you haven't made any changes to the Login service and are running with the default Clients, you can find this there at the `Config.cs`.

In any case, here is the default configuration:

```
export const environment = {
    production: false,
    identityServer: 'https://localhost:44355',
    analyticsAPI: 'http://localhost:5002',
    clientId: 'WebPortal',
    scopes: 'openid profile roles IdentityServerApi',
    responseType: 'code',
};
```

4. Run the Service

With these minimum configuration requirements, we are ready to start the service and work localhost.

From the root directory, type the following command:

```
npm start
```

Note: It might take a while to compile the first-time, but changes you make have hot-reloading, so there shouldn't be long waiting times.

After compilation is finished, point your browser at `localhost:4200` and you are ready to go.

Keep in mind the warning below!

> **Warning:** Login service needs to be also running, since you will redirect to it for Authentication!

5. Linting

By default, and to maintain a uniform coding style, we have linting in our codebase that comes from the Akveo theme.

For instance, to prevent developers from commiting `console.log()` calls by mistake.

This is configured in `tslint.json`. You can either discard it completely, alter the rules to suite your needs, or keep it like this.

If you decide to stick with it, linting will prevent you from pushing to a repository unless all rules match the codebase.

Therefore, before commiting your changes make sure to run:

```
npm run lint:ci
```

This command will scan all TypeScript and SCSS files and will return errors, if any. Fix them, and then you can push ;).

Note: Other similar commands to the one above are under the file `package.json` under the section `scripts`. Make sure to check them out.

Getting Ready for Production

1. Environment Variables

In a similar manner to development environment, now you have to set up the environment variables for production.

For the production environment, open the file `environment.prod.ts` under `src/environments`.

And modify it as in the example below:

```
export const environment = {
    production: true,
    identityServer: 'https://login.yourdomain.com',
    analyticsAPI: 'https://api.yourdomain.com',
    clientId: 'yourclient',
    scopes: 'openid profile roles IdentityServer4',
    responseType: 'code',
};
```

2. Compilation

You can read all about Angular compilation in the official documentation.

In general, you need to compile Angular into an SPA.

To do so, type the command below:

```
ng build --prod
```

The above command will make use of the production environment variables you configured in the previous section, and will produce all the files you need for deployment under the `dist/` directory.

3. Deployment

Angular compiles into a SPA and produces one single index.html.

Because of this, you have plenty of options when it comes to deployment.

You can choose to deploy it with a simple static HTML server.

All you need to do based on the Server you are using is to configure fallback to index.html as described in the link provided.

CHAPTER

FIFTEEN

CHANGELOG

15.1 MAGES™ SDK 4.2.4

- Added vitals URLs and multiplayer analytics URL to mages settings
- Improved analytics and optional actions sync in coop
- Added support for Magic Leap (BETA)
- Added new editors for Softbodies and Cut-Tear-Drill actions
- Updated Deformations Showcase scene
- Added accurate explanation console messages in case of failed login
- Added custom network info UI in MAGES Settings

- Fixed ClientId not been visible in MAGES settings
- Minor changes in custom record and replay scripts in VR Recorder
- VR Recorder now writes one channel of audio signals for remote players to solve the problem with recording distorted (high pitch) sounds
- Fixed desynchronization problem regarding multiplayer sound recording in VR Recorder
- Added comments in VR Recorder and removed unused scripts
- Fixed synchronization of Two-Hand hand poser
- Fixed cut action recorder not working when package is imported to an empty project
- Fixed missing mesh references in Medical Simulation
- Fixed lighting calculation of Softbodies Showcase scene the first time it is loaded
- Fixed error that was caused in action editor, if **Include Hologram** was enabled
- Fixed question action prefabs not properly syncing in coop
- Fixed camera position in Softbodies Showcase scene

- When importing the samples in a new project, the corresponding MagesSettings file may not contain the correct reference to the OperationStartUI
- In medical simulation there is a case where some errors may pop up in the console when the user touches the UIs that appear in the scene

15.2 MAGES™ SDK 4.2.2

- Fixed compile error with multiple versions of Newtonsoft Json importer

15.3 MAGES™ SDK 4.2.1

- Added real time Cut Action prototype [Beta]
- Added new MAGES language translation system
- Added OptionsUi to the MAGESSettings.asset for easy configuration. You can override the UI that pops up when pressing the analog stick
- Added ResetLesson() function in PointSystemManager.cs to override the Lesson name in the gamification monitor and reset the total number of Actions number.

- Improve internal traversal of scenegraph (ScenegraphTraverse)
- Fix bug with analytics panel that was spawning twice at the end of the simulation

15.4 MAGES™ SDK 4.2.0

- Added VAPPS example in Medical Sample
- Added HoloLens2 support
- Added new analytics records for coop sessions. The performer of each action is recorded as well as the time from the start of the session.
- Added synchronization of coop analytics. Currently the score and the performer are synchronized among all users.
- Added beta support for Unity 2021.3

- Fixed issue with visual scripting not saving analytics files

15.5 MAGES™ SDK 4.1.0

- Added Xray, anaesthesia machine and whiteboard on medical sample app.
- Major improvements in CTD algorithms (complexity, running time).
- Remove touching interaction with the UIs (only interaction with raycast is available).
- C++ dlls are no longer there. We integrated them into C# dlls (GAEd, StoryboardDataLicense).
- No need to have the scenegraph xmls in the Documents file for the desktop builds. They are loaded from the project Resources.
- Added teleport feature to move into selected areas in VR.

- Fixed issue with the MAGES settings causing prefabs not to saved properly.

- Fixed issues with the coop room creation.
- Various stability fixes of the VR Recorder.
- Improved names of objects in the scenes.
- Fix coop issues with Optionals and Analytics.
- Rename StartupHelper window.

- In some cases (eg QuestionAction) the Analytics scoring factors are crashing the Action on perform.
- In CTD there are some artifacts left after performing incisions.

15.6 MAGES™ SDK 4.0.2

- Desktop3D camera new UI look. Now you can control the camera by pressing the space button and then select the mode with your mouse.
- Improved MAGESHelper window. The options for networking are separated from the MAGES warnings and marked as info (not warnings since they are optionals).
- Many video tutorials were released.

- Fixed issue with Optionals and Coop.
- Fixed Coop configuration window from the MAGES top bar menu.
- Various fixes on deformable tearing.
- At the MAGESHelper window we fixed the re-import button that was not removed automatically after pressing it.
- On drilling, the hole matches the drill mesh (it was very big previously).

- When you login, the sign in button it is still there. Will be replaced with a logout button in a future version.
- In coop if someone joins after the operation has started, the scenegraph may not sync properly on the client.
- Clients cannot perform the insert plug action (last animation action) on the Cultural Heritage sample app.
- CTD is not tested on co-op.
- On continious tear there are still some artifacts when cutting.
- Tearing can sometimes not work when scalpel touches the mesh but the tear is done when the scalpel is moved a bit.
- On drill, if the model is low poly the holes are not circular.

15.7 MAGES™ SDK 4.0.1

- Major asset cleanup.

- Fixed chair missing material from Cultural Heritage sample app.
- Documentation updates to improve the developer pipeline.
- Download SDK shouldn't lead to the Documentation Page.
- Fixed error with Property _Metallic.
- Fixed Medical Sample App MAGESSettings wrong prefabs.
- Fixed Medical Sample App scalpel wrong placement.
- Fixed Cultural Heritage monitor missing materials
- Duplicate folders in MAGES Deformations. Delete one.
- In SoftbodiesCTD, we should have a Previous Button while we are on our first action.
- Remove some textures from the ActionScripts folder.
- There should not be a previous button on the first Action (bowel)
- Added new easier to use handles on Drill since the old ones were removed when the scripts moved to source.
- Fixed raycast appears to be inside the hand.
- Fixed issue with microphone Usage field still empty in the package causing MacOS Builds to fail.
- Fixed issue with scenegraph the wont load on mac build if we follow standard instructions.
- Fixed scenegraph that wont load on android from mac build.
- Added sound on perform (on Action skip).
- Fixed Cultural Heritage sample app scalpel hand pose.

- Coop configuration is not working.
- The vitals scene cannot open (is readonly).
- Coop clients cannot interact with objects.
- Issues with Optional Actions on Coop (Alternative paths).
- Minir issues on Sample apps.

15.8 MAGES™ SDK 4.0.0

- Added MAGES to the Unity Package manager for easy installation and updated.
- Rename all ovidVR instances with MAGES.
- Added Optional Actions to enable multiple active Actions at the same time.
- Added Freemium Licensing system. Freemium users can develop simulations with up to 20 Actions.
- Added Cultural Heritage, Medical simulation (TKA), Empty project, and Softbodies samples as MAGES examples to download from the package manager.

- Integrated cut, tear and drill algorithms with examples.
- VR Recorder to capture and then replay the training sessions.
- Updated Softbodies deformation algorithms.
- Removed the embedded Photon from the project's assets. Now you need to download the Photon from the Package manager
- MAGES Helper UI to automatically configure the project
- Developer SSO Login with Google.
- Added Auto HandPostures to automatically grab objects without setting a custom pose.
- Added Vitals Manager with realtime patient vitals tracking
- Added VR Annotation to label interactable objects withing the VR. You can take the marker, touch an object and wrtite a title.
- Updated VR Keyboard with different language setups and special characters.
- SceneHandler to automatically switch between scenes by fading in and out.
- Improve the MAGES menu top bar with the latest options and settings.
- Populate the Action Editor from the MAGES top bar menu with all the Action Prototypes. Developers can generate Action scripts using this tool without writing code.
- MAGES_Settings.asset to configure the xmls, api calls, links and other information into a single files
- UILists as a feature to generate MAGES lists for users to select various options in the VR.

- Fixed various issues on Sampleapps

- VR mirror was removed due to malfunction on android devices.
- Optional Actions are not working properly on coop (especially when used as Alternatives).
- Some missing references, materials and mesh references on prefabs due to the MAGES renaming and splitting of SampleApps.
- The photon configuration needs some manual work to attach the PhotonID on the prefabs and change the setting from fixed to takeover.
- Analytic UIs are not properly spawned in some cases.
- If you don't have the IK package, the user names on top of the network avatars is rotated downwards.
- Camera can pass through the ceiling in some Sample apps.
- Users cannot generate new tools using the ToolsEnum.dll. As a bypass we included some dummy tools to use.
- The same applies with the language translator, it is not possible to add new languages and entries.

15.9 MAGES™ SDK 3.3.1

- Added controls UI for Desktop3D camera.
- Exposed “Inactive Tool Layer”, “Active Tool Layer” and “Allow Reset” variables of Tool Constructor for easier modifications.
- Added interactive survey at the end of the scenario (optional).

- Fixed SceneGraph Editor panning not working with trackpad. Now works with Shift + Left Click.

- VR mirror is not working properly due to new Rendering Mode.
- MacOS some libraries may not be compatible due to a new Gatekeeper policy and need manually allow execution.
- Some prefabs in CVRSB co-op are not working properly

15.10 MAGES™ SDK 3.3.0

- Upgraded to Unity3D 2020.3.9f1
- Added XR Plugin Management support
- Added Actions Editor for Insert and User actions.
- Rework MAGES™ menu.
- Reduced significantly *Warnings & Errors* in Unity Editor
- Unified all UIs in a single folder
- Upgraded Hologram shader

- Fixed Use Action ignoring “Prefab Perform Action”.
- Fixed Analytics panel buttons that are not responding every time.
- Fixed multiple debug Errors messages information.

- VR mirror is not working properly due to new Rendering Mode.
- MacOS some libraries may not be compatible due to a new Gatekeeper policy and need manually allow execution.
- Some prefabs in CVRSB co-op are not working properly

15.11 MAGES™ SDK 3.2.1

- We changed the render pipeline to Universal Render Pipeline (URP).
- MacOS support (tested on Catalina). You can deploy a MAGES™ application using your mac using the Desktop3D camera. VR camera for macOS is currently not supported.
- Point&Click camera (experimental) for desktop based MAGES™ applications. You can try our desktop mode with point and click control using your mouse.
- Brand new question mechanic for QuestionActions. We redesigned the QuestionPrefabConstructor with additional features and new animations. Users can configure the question prefab directly from the constructor.
- Skill based analytics (experimental). We introduce skill based analytics in addition to the Action based scoring system.
- Updated VR mirror scripts with URP support.
- Updated CVRSB and MedicalSampleApp to URP.
- Populated the Configuration.cs component with additional public variables to assign product parameters directly from the Unity Editor.

- Fixed login without credentials on builds causing scenegraph crash. In this version developers can create MAGES™ applications for users without a dedicated account.
- Fixed bug Analytics files names created with non Filename Characters

- Analytics from android devices are not visualized/loaded properly on ORamaVR Portal.
- VR mirror is not working properly on Vive Focus Plus headset.
- When answering a question on the updated question prefab, the Action automatically performs.
- In Coop clients can perform the updated question Actions but the question animations and the result's visualization are not shown properly.

15.12 MAGES™ SDK 3.2.0

- In this version we introduce the **2Dof Controller** for seamless testing through Unity Editor without the need for a VR headset.
- A brand new **Medical Sample App** to get you started.
  - Initial scenario of our Total Knee Arthroplasty module
  - The skin incision, drilling, and femoral preparation
  - Bowel with soft-body simulation
  - Anatomically correct model of the patient
  - Added ready to get tools – drill, cauterizer
  - Realistic Operating Room (OR) Model
- A brand new cognitive and psychomotor medical VR sample app (our Covid19 VR Strikes Back: CVRSB training module, including full source code)
  - Training for the proper using of Personal Protective Equipment (PPE)

  - Covid-19 swab testing and interaction with patient
  - Gamified hands disinfection based on WHO
- Cloud services to make User Management, Licensing, and Analytics a smooth experience.
  - Login for User Management and Licensing
  - AnalyticsAPI for streaming data incoming from User sessions
  - Portal for Admins and Users - Charts for Analytics
  - SingleSignOn (SSO) support for user authentication

- We squashed several bugs in this version and reduced significantly the *Warnings & Errors* from the Unity Editor!

Medical Sample App

- The CooP for the Medical Sample App might be unstable for certain actions
- On Operation exit your hands change material to pink (not a feature)
- No room physical boundaries – users can navigate out of the room
- 2Dof Camera is not properly set, needs to be reinstantiated on the Scene from MAGES™ Menu

CVRSB

- The CooP for the CVRSB might be unstable for certain actions
- 2Dof initial Camera position is located in the wrong place
- Instead of 2Dof, the default Camera is SteamVR

15.13 MAGES™ SDK 3.1.0

- **Museum Sample App** that includes all latest features of MAGES™ SDK
  - Training scenario for cultural heritage restoration
  - Added ready to get tools – mallet, scalpel, pliers
  - VR ready mirror
  - Avatar customization
- **Empty Scene** for quick getting started
- MAGES™ Menu with Prefabs initialization for setting up Cameras

General

- Several Warnings on Unity Editor (do not affect the experience)

CHAPTER

SIXTEEN

CHANGELOG

16.1 Unreal MAGES™ SDK 4.0.0

Release version of Unreal M.A.G.E.S. SDK, providing:

- Action Prototypes
- Multiplayer
- Analytics
- Live Scenegraph
- Mages Interaction System
- Two sample apps (Medical & Cultural Inheritance)
- Early access COVID-19 Testing Module
- Cloud services
- Cut Tear & Drill Mesh Deformations
- Softbodies
- Freemium Licensing system. Freemium users can develop simulations with up to 20 Actions.

- Fixed Co-Op crashes in packaged projects
- Fixed multiple consecutive undoes leading to crash
- Question Action is now properly synchronized in Co-Op
- Fixed re-grab on objects while they are still inside the acceptable grab range

- Some UI panels spawn away from the player or overlap with other 3D objects in the scenes.
- Some interactive items keep colliding with the user's hands even after they let go of them.
- CTD Tear produces some artifacts in certain cases.
- VR UI produces reflections in mirror.
- VR Mirror does not work in Mobile VR.
- Networking on Oculus Mobile (Oculus Quest 1 & Oculus Quest 2) crashes

16.2 Unreal MAGES™ SDK 0.9.0

BETA version of Unreal M.A.G.E.S. SDK, providing:

- Action Prototypes
- Multiplayer
- Analytics
- Scenegraph
- Mages Interaction System
- Two sample apps (medical & museum)
- Cloud services
- VR UI templates

- In packaged builds, coop has frequent crashes.
- Fast undoes of actions can lead to crash after a short amount of time, this is observed when using the restart functionality.
- Pump action & Question action are not accurately synced in coop.
- When attempting to re-grab an object the player must reset hovering by distancing his hand away from the interactable and bring it close again.
- Some UI panels spawn away from the player or overlap with other 3D objects in the scenes.

BIBLIOGRAPHY

[C3] George Papagiannakis, Panos Trahanias, Eustathios Kenanidis, and Eleftherios Tsiridis. 2017. Psychomotor Surgical Training in Virtual Reality. Master Case Series & Techniques: Adult Hip (07 2017), 827–830. https://doi.org/10.1007/978-3-319-64177-5_41

[C2] Papagiannakis, G., Zikas, P., Lydatakis, N., Kateros, S., Kentros, M., Geronikolakis, E., Kamarianakis, M., Kartsonaki, I., Evangelou, G., "MAGES 3.0: Tying the knot of medical VR", In ACM SIGGRAPH 2020 Immersive Pavilion (SIGGRAPH '20), Association for Computing Machinery, New York, NY, USA, Article 6, 1–2. https://doi.org/10.1145/3388536.3407888, 2020

[C1] Jessica Hooper, Eleftherios Tsiridis, James E. Feng, Ran Schwarzkopf, Daniel Waren, William J. Long, Lazaros Poultsides, William Macaulay, George Papagiannakis, Eustathios Kenanidis, Eduardo D. Rodriguez, James Slover, Kenneth A. Egol, Donna P. Phillips, Scott Friedlander, and Michael Collins. 2019. Virtual Reality Simulation Facilitates Resident Training in Total Hip Arthroplasty: A Randomized Controlled Trial. The Journal of Arthroplasty (2019). https://doi.org/10.1016/j.arth.2019.04.002

[M1] Kamarianakis, M., Lydatakis, N., & Papagiannakis, G. (2021, September). Never 'Drop the Ball' in the Operating Room: An Efficient Hand-Based VR HMD Controller Interpolation Algorithm, for Collaborative, Networked Virtual Environments. In Computer Graphics International Conference (pp. 694-704). Springer, Cham.

[M_1] Kamarianakis, M., Lydatakis, N., & Papagiannakis, G. (2021, September). Never 'Drop the Ball'in the Operating Room: An Efficient Hand-Based VR HMD Controller Interpolation Algorithm, for Collaborative, Networked Virtual Environments. In Computer Graphics International Conference (pp. 694-704). Springer, Cham.

INDEX

A

AAction
    class, 684
AActorNode
    class, 684
AAnalyticsExporter
    class, 684
AAnalyticsManager
    class, 685
AAnimationAction
    class, 686
ABasePrototype
    class, 704
ABasePrototype::InstrumentTransforms
    class, 704
ABasePrototype::InstrumentTransforms::CustomTransform
    class, 704
ABPAnimationAction
    class, 687
ABPBasePrototype
    class, 689
ABPCombinedAction
    class, 691
ABPInsertAction
    class, 693
ABPParallelAction
    class, 695
ABPPumpAction
    class, 696
ABPQuestionAction
    class, 698
ABPRemoveAction
    class, 700
ABPUseAction
    class, 702
ACameraRigInputController
    class, 706
ACameraRigInputController::ButtonState
    struct, 707
ACameraRigInputController::HapticEffectState
    struct, 707
ACombinedAction
    class, 708
ACreateDeformMesh
    class, 710
Activated
    variable, 774
ActivateTool
    function, 775
AEventManager
    class, 711
AEventManager::InvokeData
    class, 711
AInsertAction
    class, 711
AJSONParser
    class, 713
ALesson
    class, 714
AllocateViewID
    function, 763
AMagesController
    class, 714
AMagesPlayer
    class, 715
AMagesSceneGraph
    class, 715
AMagesWidgetActor
    class, 716
AMarker
    class, 716
AnalyticsRuntimeImporter
    class, 733
AOperationAnalytics
    class, 716
AParallelAction
    class, 717
APhotonLBClient
    class, 718
APickUpTransform
    class, 720
ApplicationUser
    class, 734
AProceduralActorComponent

class, 720
APumpAction
class, 721
AQuestionAction
class, 722
ARemoveAction
class, 724
ARigidbodyAnimationController
class, 726
ASoftParticleHelper
class, 728
AStage
class, 729
AUIExtraExpNotification
class, 729
AUINotification
class, 729
AUseAction
class, 730
AUserAccountManager
class, 732
AUserPathTracer
class, 733

B

BaseView
class, 734
BindEvent
function, 761

C

CameraHead
variable, 708
CategorizeFaces
function, 788
class
AAction, 684
AActorNode, 684
AAnalyticsExporter, 684
AAnalyticsManager, 685
AAnimationAction, 686
ABasePrototype, 704
ABasePrototype::InstrumentTransforms, 704
ABasePrototype::InstrumentTransforms::CustomTransform, 704
ABPAnimationAction, 687
ABPBasePrototype, 689
ABPCombinedAction, 691
ABPInsertAction, 693
ABPParallelAction, 695
ABPPumpAction, 696
ABPQuestionAction, 698
ABPRemoveAction, 700
ABPUseAction, 702
ACameraRigInputController, 706
ACombinedAction, 708
ACreateDeformMesh, 710
AEventManager, 711
AEventManager::InvokeData, 711
AInsertAction, 711
AJSONParser, 713
ALesson, 714
AMagesController, 714
AMagesPlayer, 715
AMagesSceneGraph, 715
AMagesWidgetActor, 716
AMarker, 716
AnalyticsRuntimeImporter, 733
AOperationAnalytics, 716
AParallelAction, 717
APhotonLBClient, 718
APickUpTransform, 720
ApplicationUser, 734
AProceduralActorComponent, 720
APumpAction, 721
AQuestionAction, 722
ARemoveAction, 724
ARigidbodyAnimationController, 726
ASoftParticleHelper, 728
AStage, 729
AUIExtraExpNotification, 729
AUINotification, 729
AUseAction, 730
AUserAccountManager, 732
AUserPathTracer, 733
BaseView, 734
ClientConfiguration, 735
DeveloperCredentials, 735
ExitGames::Common::JVector, 652
FCurrentRequest, 735
FDeformableHelper, 736
FMAGES_SDKModule, 736
FMagesActionNode, 737
FMagesInteractables, 738
FMagesNetStream, 738
FReplicaCache, 739
FUserAction, 739
IDeviceControllerInterface, 740
IIAction, 740
IMagesInputDevice, 741
IMagesNetObservable, 742
IMagesNetTransform, 742
IMagesNetworkMessage, 742
LoadBalancingListener, 743
MAGES_SDK, 743
NetMessageClass, 744
PeerStatesStrChecker, 745

PhotonEventHandler, 745
PostRefreshLicense, 745
RigidBodyAnimationBase, 745
RigidBodyMoveAndRotateDualQuat, 746
RigidBodyMoveDualQuat, 747
Tenant, 747
UAssetsImporter, 748
UAuthenticationHandler, 748
UAvoidObjectFactor, 749
UCountDownFactor, 750
UCredentialsManager, 751
UDeviceControllerInterface, 751
UIAction, 751
UMagesActionAnalyticsListEntry, 752
UMagesActionAnalyticsWidget, 752
UMagesAnalyticsAsset, 753
UMagesAudioSubsystem, 753
UMagesButtonInput, 754
UMagesButtonsHelper, 755
UMagesButtonWidget, 754
UMagesConfig, 756
UMagesControllerClass, 756
UMagesDeformableMeshData, 757
UMagesDevice, 757
UMagesGameplayUtility, 758
UMagesInputDevice, 759
UMagesInstance, 759
UMagesLiveScenegraphSubsystem, 759
UMagesNetObservable, 761
UMagesNetTransform, 761
UMagesNetwork, 761
UMagesNetworkBackend, 764
UMagesQuestionButtonWidget, 764
UMagesScrollBox, 765
UMagesSyncTransform, 765
UMagesTextBoxWidget, 767
UMagesTextWidget, 767
UMagesUserEventAsset, 768
UMagesView, 769
UMagesWidget, 769
UOrama_Util, 770
UQuestionScoringFactor, 772
URemoveWithToolConstructor, 772
UScoringFactor, 773
UserAccessToken, 777
UToolNetSync, 774
UUIManagement, 775
UUserCredentialsSaveGame, 775
UVelocityScoringFactor, 776
ClassifyConflictingType
function, 789
ClientConfiguration
class, 735
Collections
namespace, 651
CreateAction
function, 761
CreateMeshSectionsForPredicates
function, 789

D

DeActivateTool
function, 775
DECLARE_DYNAMIC_DELEGATE_OneParam
function, 790
DECLARE_DYNAMIC_DELEGATE_TwoParams
function, 790
Default
enumvalue, 662
define
GET_SUBSYSTEM, 790
DeveloperCredentials
class, 735
Diagnostics
namespace, 651
DistanceTheshold
variable, 730
DotNETCommon
namespace, 651

E

EActionType
enum, 658
EAmbientAudioType
enum, 658
EAnalyticsColliderBehavior
enum, 658
EAnalyticsErrorType
enum, 658
EAnalyticsFactorImportance
enum, 658
EAudioClipType
enum, 659
ECollisionType
enum, 659
EControllerDOF
enum, 659
EControllerTypes
enum, 659
EDifficulty
enum, 660
EErrorType
enum, 660
EFactorImportance
enum, 660
EHandState
enum, 660
EInteractionStyle

enum, 661
ELicenseType
namespace, 651
ELoginStatus
enum, 661
EMagesButtonInteractionMethod
enum, 661
EMagesButtons
enum, 662
EMagesControllerButtons
enum, 662
EMagesDeformableMeshType
enum, 663
EMagesSDKIntegrations
enum, 663
EMagesSyncTransformMode
enum, 663
ENetVarType
enum, 663
enum
EActionType, 658
EAmbientAudioType, 658
EAnalyticsColliderBehavior, 658
EAnalyticsErrorType, 658
EAnalyticsFactorImportance, 658
EAudioClipType, 659
ECollisionType, 659
EControllerDOF, 659
EControllerTypes, 659
EDifficulty, 660
EErrorType, 660
EFactorImportance, 660
EHandState, 660
EInteractionStyle, 661
ELoginStatus, 661
EMagesButtonInteractionMethod, 661
EMagesButtons, 662
EMagesControllerButtons, 662
EMagesDeformableMeshType, 663
EMagesSDKIntegrations, 663
EMagesSyncTransformMode, 663
ENetVarType, 663
EOperationDifficulty, 663
EOvidVRHand, 664
EOwnershipOption, 664
EScoringMethod, 664
ESendMethod, 664
EUIType, 664
EVisibilityLevel, 665
FRequestVerb, 735
InheritTransformFrom, 665
NetKeyCode, 665
OnPrefabDetachFeature, 666
ParticleGroupMehod, 666
PrefabActionOnPerform, 666
PrefabInteractableType, 666
PrefabType, 666
PumpMode, 667
ToolFlashType, 667
ToolGrabbingType, 667
ToolRotationAxis, 667
ToolTriggerButton, 668
Type, 651
UseColliderTrigger, 668
enumvalue
Default, 662
TapAndRepeat, 662
EOperationDifficulty
enum, 663
EOvidVRHand
enum, 664
EOwnershipOption
enum, 664
EScoringMethod
enum, 664
ESendMethod
enum, 664
ESyncTransformFlags
namespace, 651
EUIType
enum, 664
EvalAllIntersections
function, 789
EvalIntersection
function, 789
EVisibilityLevel
enum, 665
ExitGames
namespace, 652
ExitGames::Common
namespace, 652
ExitGames::Common::JVector
class, 652
ExitGames::LoadBalancing
namespace, 652

F

FActionAnalyticsData
struct, 668
FActionGroup
struct, 668
FActionSummary
struct, 669
FAnalyticsErrorData
struct, 669
FAnalyticsTimeData
struct, 669
FAnimationGroup

struct, 670
FAudioAsset
struct, 670
FAvoidObjectsData
struct, 670
FCurrentRequest
class, 735
FDeformableHelper
class, 736
FDrillCategorizeFacesDesc
struct, 671
FDrillEvalIntersectionDesc
struct, 671
FDrillSplitCertainFacesDesc
struct, 671
FDuo
struct, 671
FErrorsStayData
struct, 672
FGenerateTearSegmentTrianglesDesc
struct, 672
FHoloGroup
struct, 672
FIActionGroup
struct, 673
FindAffectedIds
function, 788
FindMidpoint
function, 657
FindRoots
function, 656
FInsertGroup
struct, 673
FLineSegment
struct, 673
FMAGES_SDKModule
class, 736
FMagesActionAnalyticsEntry
struct, 674
FMagesActionNode
class, 737
FMagesActionPath
struct, 674
FMagesActionPathEntry
struct, 674
FMagesInteractables
class, 738
FMagesNetStream
class, 738
FMagesPrepareTearResult
struct, 674
FMagesUserEventAssetEntry
struct, 675
FNamedTimeProfileContext
struct, 675
FOrientationContext
struct, 676
FPointNormalPlane
function, 653
FPostCheckoutProduct
struct, 676
FPumpGroup
struct, 676
FQuestionData
struct, 676
FQuestionOptionData
struct, 677
FQuestionScoringData
struct, 677
FRaiseEventBatch
struct, 678
FRecalculateNormalsDesc
struct, 678
FRecalculateNormalsVertexEntry
struct, 678
FRecalculateNormalsVertexKey
struct, 678
FRemoveGroup
struct, 679
FReplicaCache
class, 739
FReplicaCacheEntry
struct, 679
FRequestVerb
enum, 735
FRetriangulateInitialTearPointsDesc
struct, 679
FRoomInfo
struct, 680
FScoringFactorData
struct, 680
FScoringFactorRef
struct, 680
FSerializationMessageInfo
struct, 681
FSerializeViewBatch
struct, 681
FSpawnActorDesc
struct, 681
FTimeData
struct, 682
FTimeProfileContext
struct, 682
function
ActivateTool, 775
AllocateViewID, 763
BindEvent, 761
CategorizeFaces, 788

ClassifyConflictingType, 789
CreateAction, 761
CreateMeshSectionsForPredicates, 789
DeActivateTool, 775
DECLARE_DYNAMIC_DELEGATE_OneParam, 790
DECLARE_DYNAMIC_DELEGATE_TwoParams, 790
EvalAllIntersections, 789
EvalIntersection, 789
FindAffectedIds, 788
FindMidpoint, 657
FindRoots, 656
FPointNormalPlane, 653
GetCurrentAction, 761
GetCurrentActionName, 761
GetLocalPlayerNumber, 763
GetNumPlayersInRoom, 763
GetPoint, 653
GetView, 763
Initialize, 763
Instantiate, 763
InternalEnsureMatchAttributes, 789
IsPointInsideTriangle, 657
IsPointWithinShapeComponent, 657
IsWithinDrill, 789
LineIntersectPlane, 656
OnCreated, 770
Orient3D, 657
PlanePerpendicularToPlane, 656
PlayClipOnComponent, 754
PointIntersectPlane, 657
ProjectPointOntoPlane, 657
RaiseUserEvent, 763
RegisterUserEvent, 763
RemoveConsecutiveSeparators, 772
RequestDestroySelf, 770
RequestOwner, 770
SetupActionAnalytics, 761
SplitCertainFaces, 789
StartupModule, 737
TriangulateModel, 789
UPROPERTY, 763
FUserAction
class, 739
FUserCredentials
struct, 682
FVelocityData
struct, 683
FVelocityScoringData
struct, 683

G

GET_SUBSYSTEM
define, 790
GetCurrentAction
function, 761
GetCurrentActionName
function, 761
GetLocalPlayerNumber
function, 763
GetNumPlayersInRoom
function, 763
GetPoint
function, 653
GetView
function, 763
global
namespace, 651

I

IDeviceControllerInterface
class, 740
IIAction
class, 740
IMagesInputDevice
class, 741
IMagesNetObservable
class, 742
IMagesNetTransform
class, 742
IMagesNetworkMessage
class, 742
InheritTransformFrom
enum, 665
Initialize
function, 763
Instantiate
function, 763
InternalEnsureMatchAttributes
function, 789
IO
namespace, 652
isCorrect
variable, 677
IsPointInsideTriangle
function, 657
IsPointWithinShapeComponent
function, 657
IsTextVisibleOnInit
variable, 755
isToggleMode
variable, 755
IsWithinDrill
function, 789

L

LineIntersectPlane

function, 656
LoadBalancingListener
class, 743
LocalPlayer
struct, 683

M

MAGES_SDK
class, 743
MagesMath
namespace, 653
MagesMath::FPointNormalPlane
struct, 653
MagesMath::FQuadruplet
struct, 653
MagesMath::FTriangle
struct, 654
MagesMath::FTriplet
struct, 654
MagesNetworkStatusCode
namespace, 657
MAX_VIEW_IDS
variable, 763

N

namespace
Collections, 651
Diagnostics, 651
DotNETCommon, 651
ELicenseType, 651
ESyncTransformFlags, 651
ExitGames, 652
ExitGames::Common, 652
ExitGames::LoadBalancing, 652
global, 651
IO, 652
MagesMath, 653
MagesNetworkStatusCode, 657
System, 657
UnrealBuildTool, 658
NetKeyCode
enum, 665
NetMessageClass
class, 744
NetworkedActorDestructionTimeout
variable, 763
NetworkID
variable, 765

O

OnCreated
function, 770
OnPrefabDetachFeature
enum, 666
optionText
variable, 677
orderOfAnswer
variable, 677
Orient3D
function, 657

P

ParticleGroupMehod
enum, 666
PeerStatesStrChecker
class, 745
PhotonEventHandler
class, 745
PlanePerpendicularToPlane
function, 656
PlayClipOnComponent
function, 754
PointIntersectPlane
function, 657
PostRefreshLicense
class, 745
PrefabActionOnPerform
enum, 666
PrefabInteractableType
enum, 666
PrefabType
enum, 666
ProjectPointOntoPlane
function, 657
PumpMode
enum, 667

Q

questionMessageUser
variable, 677

R

RaiseUserEvent
function, 763
RandomStream
variable, 763
RegisterUserEvent
function, 763
RemoveConsecutiveSeparators
function, 772
RequestDestroySelf
function, 770
RequestOwner
function, 770
RigidBodyAnimationBase
class, 745
RigidBodyMoveAndRotateDualQuat
class, 746

RigidBodyMoveDualQuat
class, 747

S

SectionIndex
variable, 757
SetupActionAnalytics
function, 761
Speed
variable, 708
SplitCertainFaces
function, 789
StartupModule
function, 737
struct
ACameraRigInputController::ButtonState, 707
ACameraRigInputController::HapticEffectState, 707
FActionAnalyticsData, 668
FActionGroup, 668
FActionSummary, 669
FAnalyticsErrorData, 669
FAnalyticsTimeData, 669
FAnimationGroup, 670
FAudioAsset, 670
FAvoidObjectsData, 670
FDrillCategorizeFacesDesc, 671
FDrillEvalIntersectionDesc, 671
FDrillSplitCertainFacesDesc, 671
FDuo, 671
FErrorsStayData, 672
FGenerateTearSegmentTrianglesDesc, 672
FHoloGroup, 672
FIActionGroup, 673
FInsertGroup, 673
FLineSegment, 673
FMagesActionAnalyticsEntry, 674
FMagesActionPath, 674
FMagesActionPathEntry, 674
FMagesPrepareTearResult, 674
FMagesUserEventAssetEntry, 675
FNamedTimeProfileContext, 675
FOrientationContext, 676
FPostCheckoutProduct, 676
FPumpGroup, 676
FQuestionData, 676
FQuestionOptionData, 677
FQuestionScoringData, 677
FRaiseEventBatch, 678
FRecalculateNormalsDesc, 678
FRecalculateNormalsVertexEntry, 678
FRecalculateNormalsVertexKey, 678
FRemoveGroup, 679
FReplicaCacheEntry, 679
FRetriangulateInitialTearPointsDesc, 679
FRoomInfo, 680
FScoringFactorData, 680
FScoringFactorRef, 680
FSerializationMessageInfo, 681
FSerializeViewBatch, 681
FSpawnActorDesc, 681
FTimeData, 682
FTimeProfileContext, 682
FUserCredentials, 682
FVelocityData, 683
FVelocityScoringData, 683
LocalPlayer, 683
MagesMath::FPointNormalPlane, 653
MagesMath::FQuadruplet, 653
MagesMath::FTriangle, 654
MagesMath::FTriplet, 654
SynchronizedComponentNames
variable, 767
System
namespace, 657

T

TapAndRepeat
enumvalue, 662
Tenant
class, 747
ToolFlashType
enum, 667
ToolGrabbingType
enum, 667
ToolRotationAxis
enum, 667
ToolTriggerButton
enum, 668
TriangulateModel
function, 789
Type
enum, 651
variable, 757

U

UAssetsImporter
class, 748
UAuthenticationHandler
class, 748
UAvoidObjectFactor
class, 749
UCountDownFactor
class, 750
UCredentialsManager

class, 751
UDeviceControllerInterface
class, 751
UIAction
class, 751
UMagesActionAnalyticsListEntry
class, 752
UMagesActionAnalyticsWidget
class, 752
UMagesAnalyticsAsset
class, 753
UMagesAudioSubsystem
class, 753
UMagesButtonInput
class, 754
UMagesButtonsHelper
class, 755
UMagesButtonWidget
class, 754
UMagesConfig
class, 756
UMagesControllerClass
class, 756
UMagesDeformableMeshData
class, 757
UMagesDevice
class, 757
UMagesGameplayUtility
class, 758
UMagesInputDevice
class, 759
UMagesInstance
class, 759
UMagesLiveScenegraphSubsystem
class, 759
UMagesNetObservable
class, 761
UMagesNetTransform
class, 761
UMagesNetwork
class, 761
UMagesNetworkBackend
class, 764
UMagesQuestionButtonWidget
class, 764
UMagesScrollBox
class, 765
UMagesSyncTransform
class, 765
UMagesTextBoxWidget
class, 767
UMagesTextWidget
class, 767
UMagesUserEventAsset
class, 768
UMagesView
class, 769
UMagesWidget
class, 769
UnrealBuildTool
namespace, 658
UOrama_Util
class, 770
UPROPERTY
function, 763
UQuestionScoringFactor
class, 772
URemoveWithToolConstructor
class, 772
UScoringFactor
class, 773
UseColliderTrigger
enum, 668
UserAccessToken
class, 777
UToolNetSync
class, 774
UUIManagement
class, 775
UUserCredentialsSaveGame
class, 775
UVelocityScoringFactor
class, 776

V

variable
Activated, 774
CameraHead, 708
DistanceTheshold, 730
isCorrect, 677
IsTextVisibleOnInit, 755
isToggleMode, 755
MAX_VIEW_IDS, 763
NetworkedActorDestructionTimeout, 763
NetworkID, 765
optionText, 677
orderOfAnswer, 677
questionMessageUser, 677
RandomStream, 763
SectionIndex, 757
Speed, 708
SynchronizedComponentNames, 767
Type, 757
WasRecentlyTriggered, 738

W

WasRecentlyTriggered

variable, 738