

Onward Custom Content Guide

Mar 2025 - Patch 2.0 - Guide/Package Version: 15.0 - SDK Increment 15

Disclaimer

- All information in this guide is up to date as of **Onward 2.0**, March 12th, 2025 and is subject to change. Changes and updates will be noted in the 'Change log' section below.
- Updates to the custom content package files may cause inconsistencies with this guide
- If you encounter any issues with this guide or the custom content package files, please contact us using the [Onward Discord channel](#).

Changelog

- 1/18/2023 Version 9.3 released. Changed upload method to upload more successfully more often. Fixed incorrect custom content link on page 7 and 10. Added Table of Contents for easier browsing, especially of new sections.
- 6/27/2023 Version 11.0 released. Fixed the ammo boxes Upgraded unity to 2019.4.40f1. Guide rewritten to improve readability and clarity.
- 8/15/2023 Version 12.0 released. Upgraded unity to 2021.3.23f1. Fixed the box gizmo matrices. Added Fire Fight support. Updated Lightbake guide. Updated set up for the new Unity version. Updated Quest guidelines now that Quest 1 is not supported. Added custom tent visuals section. Removed Max LOD on Quest (PC and Quest are equal now). Usage of the built in skybox is now recommended for Quest maps. Added [troubleshooting section](#) (it will grow over time).
- 8/31/2023 version 12.1 released. Added far clip plane settings. Added new skybox shaders.
- 10/25/2023 version 13 released. New AI workflows. Hunt and Evac are no longer tied together. Night vision LUTs. Cleaned up old/out of date error checks. Shader clean up pass: replaced the old HLSL PreProcessing with the SG one. Replaced the skybox shaders with one, deleted the others. Updated all other shaders, added per texture tiling and offset for multiple maps.
- 11/16/2023 version 13.1 released. Improved saving and loading of AI entities.
- 1/31/2024 Version 14 released. Updated shaders.
- 3/12/2025 Version 15 released. Added support for 2.0 features: Retrieval Mode, ladders, ladder perches, lightning storms, and flyovers. Incorporated community guide suggestions. Fixed audio reverb zones (see AreaReverb section for how to enable fix).

Requirements

-
- Unity 2021.3.23f1 & 60-100GB free storage
 - [Onward Custom Content Package](#)
 - An Oculus Account ([Publishing](#))
 - Own Onward on that account.
-

Table of Contents

Onward Custom Content Guide.....	1
Disclaimer.....	1
Changelog.....	1
Requirements.....	2
Table of Contents.....	2
Content submission rules.....	4
Technical Guidelines.....	4
Content Guidelines.....	4
Setting up Unity.....	6
Starting your first project.....	11
Updating an existing project.....	11
Creating a custom map.....	11
Custom Tent Visuals.....	14
Setting up game modes.....	14
Set objectives & spawn points.....	16
Setting up the Escort game mode.....	19
Setting up the Assault game mode.....	20
Setting up Fire Fight.....	21
Setting up the Co-op game modes (Hunt, Evac, and Retrieval).....	25
AI_Navigation Set Up.....	26
Navigation Mesh Baker.....	26
Navigation Map Generator.....	28
Waypoints.....	31
AI Spawners.....	34
Hunt.....	36
Evac.....	39

Retrieval.....	43
Setting up the Social game modes (OITC, Gun Game, Spec Ops).....	45
Gun Game & One In the Chamber.....	46
Spec Ops.....	46
Additional gameplay components.....	48
Set Physic Material.....	48
Elevators.....	49
Ladders.....	49
Ammo Boxes.....	51
PostProcessing Profile.....	53
OnwardAudioSource.....	54
AreaReverb.....	55
AreaLadderPerch.....	56
FlyoverSystem.....	56
LightningSystem.....	58
Check For Errors.....	61
Restricted Components.....	61
Testing your maps.....	63
PC Testing.....	63
Quest testing.....	64
Performance Considerations.....	66
PC performance considerations.....	66
Quest performance considerations.....	67
Best practices in development.....	68
Publishing.....	69
Conclusion.....	72
Appendix A: Updating your project.....	73
SDK 9 Upgrade - Onward v1.9.....	73
SDK 13 Upgrade - Onward 1.13.....	73
Appendix B: Claiming map ownership (update 1.9 onwards).....	75
Appendix C: Optimization tools & tricks for Quest.....	78
OVRMetricsTool.....	78
Optimize Window.....	79
Appendix D: Troubleshooting.....	79
Textures are Pixelated on the Quest.....	79
The skybox gets cut off.....	79

Content submission rules

Below are technical and content guidelines your maps must adhere to. Not following these guidelines will result in your upload being denied. In addition to these rules, please review the [Community Guidelines](#). You are responsible for the testing and verification of your own map content submissions as well as curating an acceptable user experience.

Submission of custom content for play in Onward is a free service we offer to enrich the experience for players, it is not a right. Please follow these rules and you will be on your way to share your creations with thousands of Onward players. Your content/maps may be removed from our servers at any time post-approval if Downpour Interactive deems it has violated the rules for content submission.

By submitting your map(s) to the Onward Workshop, you agree to the [Custom Content End User Agreement](#) terms.

Technical Guidelines

- Your map materials must use only the PreProcessing (prefixed; PP_xyz..) shaders included with the Onward Custom Content SDK v1.12.0+. If your map currently does not contain only these shaders, that's alright. You can use our Shader Conversion steps and tools to assist you in converting your shaders. It is a simple process that can be carried out in a few minutes to an hour, depending on the complexity of your custom content scene.
- You must lightmap your map. Use [this lightmapping document](#) as a guide when you are baking lightmaps in your map.
- Your map must support the FreeRoam game mode and at least one (1) other game mode.
- PC and Quest share LODs now. Before Quest would strip LOD0 and use LOD1. Now Quest uses LOD0.
- The built in Unity sky box is recommended for both PC and Quest. You no longer need to use the old sky box mesh on Quest.

Content Guidelines

- You may not use any copyrighted / unlicensed assets in your map. For example, you may not directly port de_dust2, or Blood Gulch etc.). Inspired homages of maps from other games and universes must be carried out without infringing source material.
- You must refrain from using terminology in your map descriptions that likens or compares it to a 1:1 copy of an existing map in another game or universe.

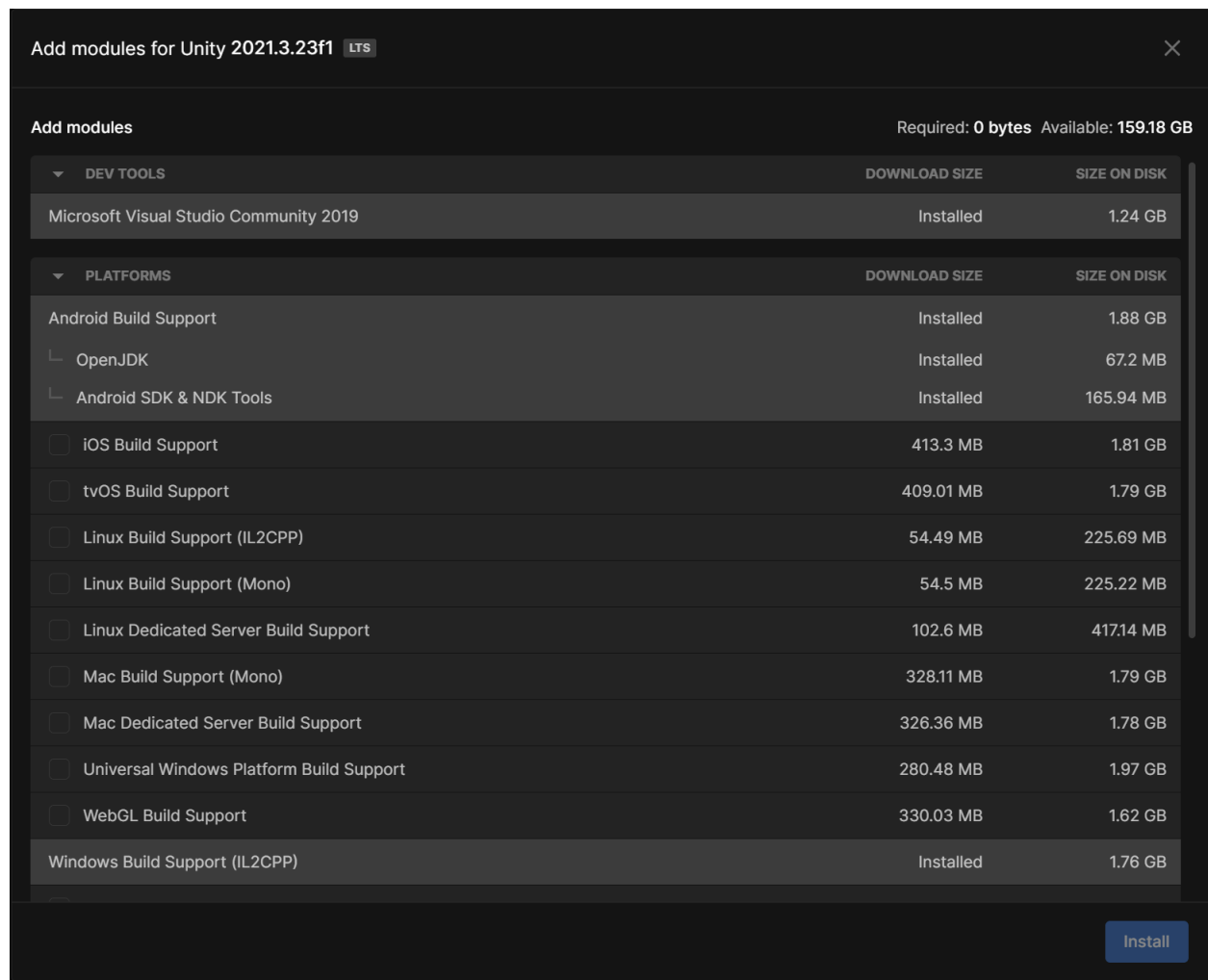
- “A recreation of...” is not allowed.
 - “Inspired by...” is allowed.
- You may not place vulgar, inappropriate, racist, obscene or otherwise unpalatable or illicit content in your map.
- You may not use the full name, likeness or representation of any person (alive or dead) in your scenes.
- Do not put any content in your map that could be deemed hazardous or otherwise induce bodily harm to a player. This includes but is not limited to: extremely loud sounds, flashing/strobing visual aspects, nausea-inducing mechanics or otherwise hazardous elements.

Setting up Unity

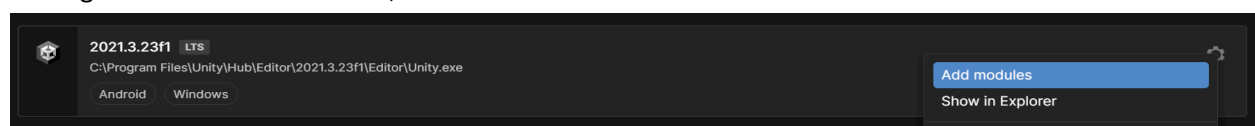
Ensure that you have enough disk space to store the custom project and associated data generated during map creation (>60GB)

Download and install **Unity 2021.3.23f1**

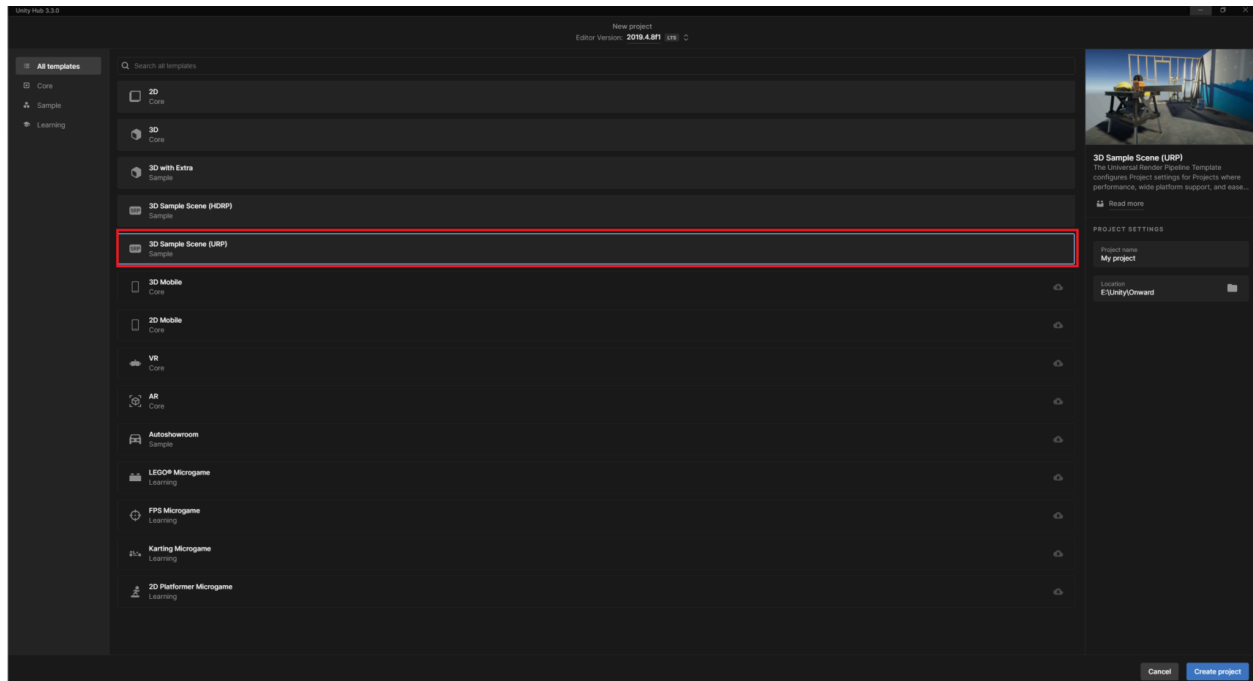
Go to [the Unity website](#), Select Unity 2021.x and locate and download Unity 2021.3.23f1 with Unity Hub, install Android Build Support when installing.



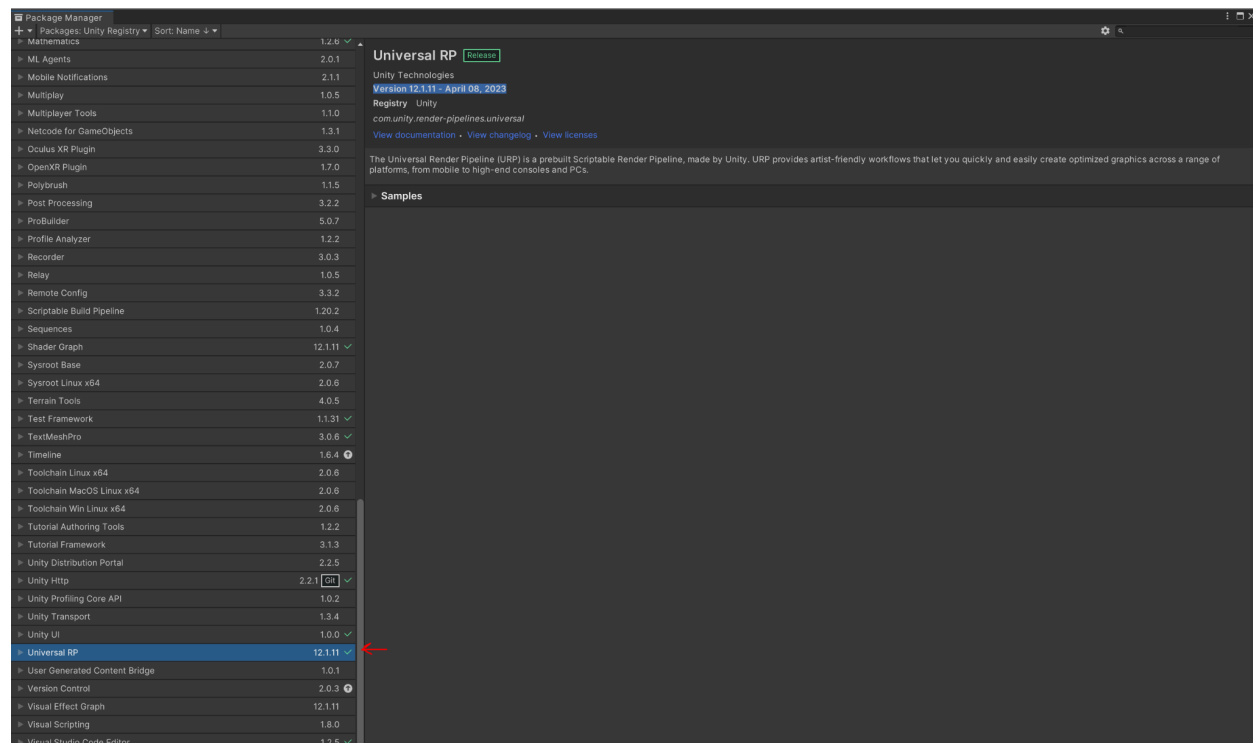
Note: You can install these tools later if you installed Unity with the Hub option. If you did not install Unity with Hub, you will need to do so in order to continue. To add these packages Go to Installs, click the cog icon next to 2021.3.23f1, and then click Add modules. Add the modules shown above.



Make a new project. Select the **3D Sample Scene (URP)**. This will take care of installing URP to our project for us, which is the pipeline used by Onward.

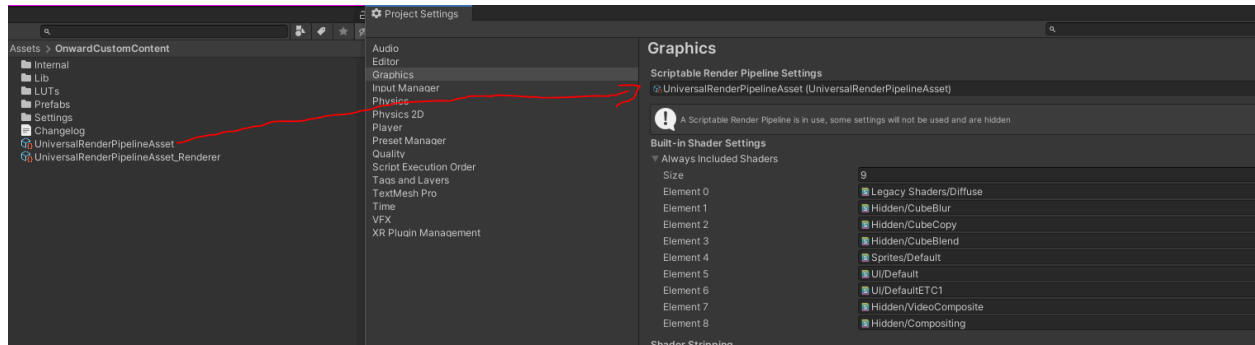


Check that it installed URP Version 12.1.11. You can check the version and change it if need through the package manager.

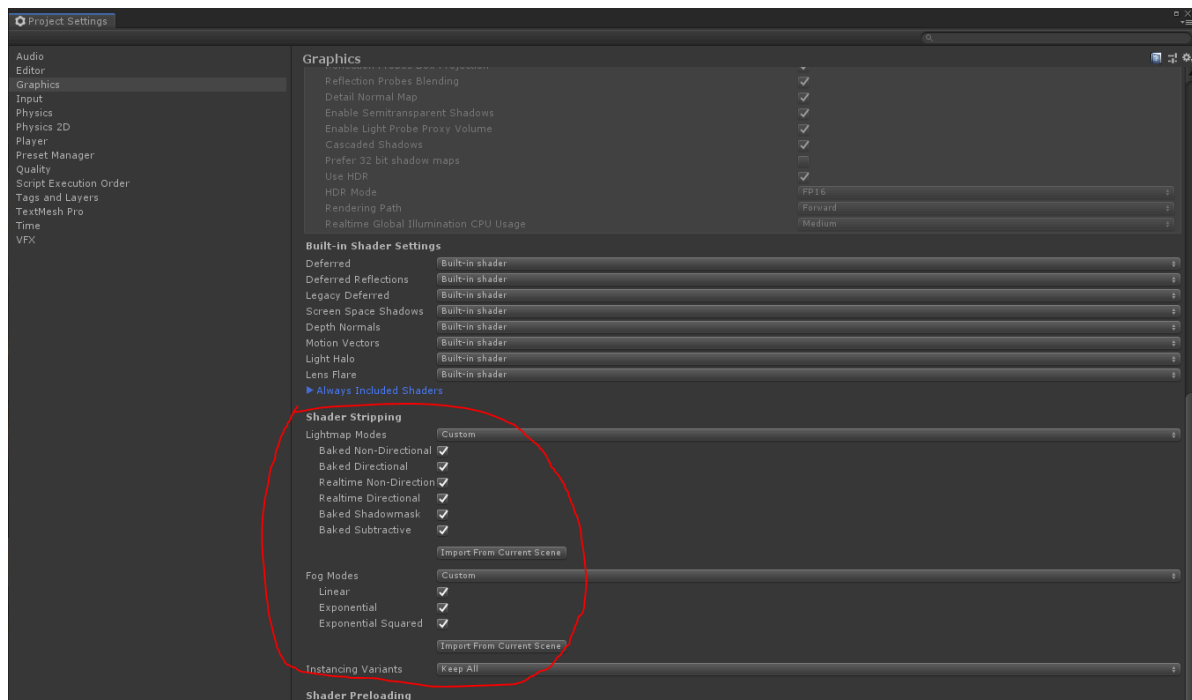


Download and add the custom content package to your Assets folder (drag and drop the package into your Unity window to import it)

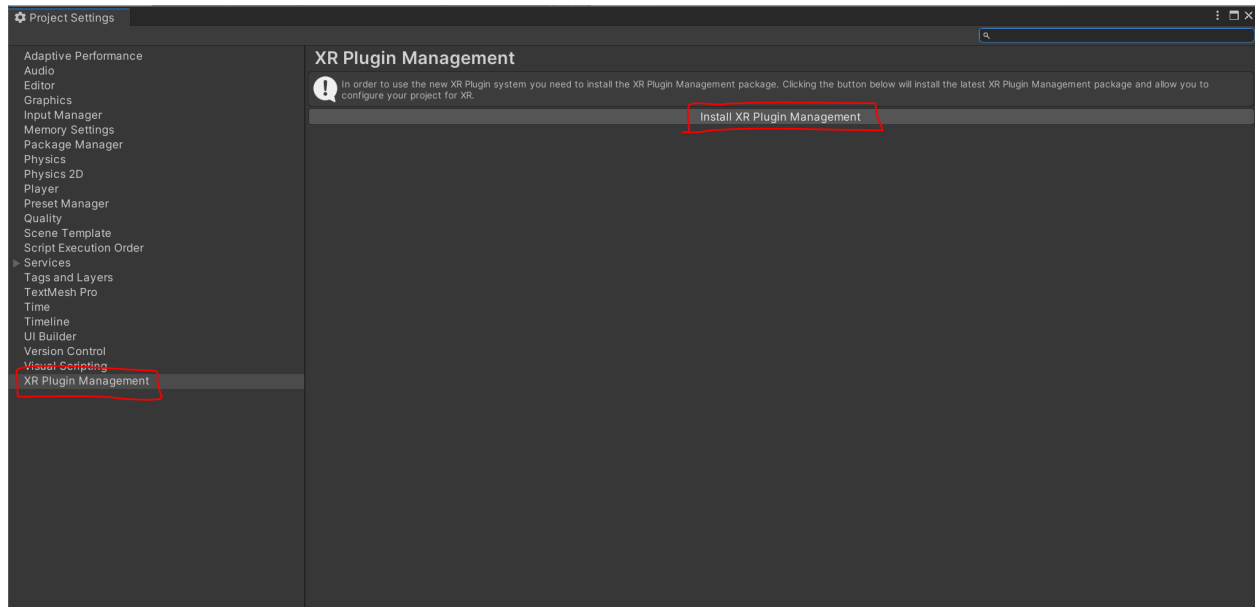
Now, Open your **Project Settings** window under **Edit > Project Settings** and navigate to the **Graphics** tab. Once there, locate the **UniversalRenderPipelineAsset** file located in the root folder of **Assets > OnwardCustomContent** and drag it into the **Scriptable Render Pipeline Settings** field of the Graphics tab in the project settings window.



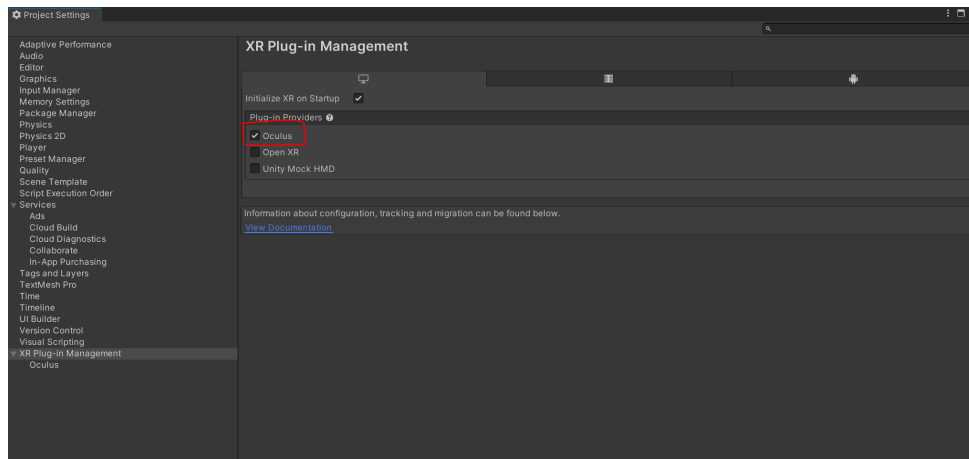
Go to **Edit > Project Settings > Graphics tab** scroll down to the “**Shader Stripping**” section. Ensure all checkboxes are checked and both fields for **Lightmap Modes** and **Fog Modes** are set to ‘**Custom**’. The screenshot below shows the correct settings.



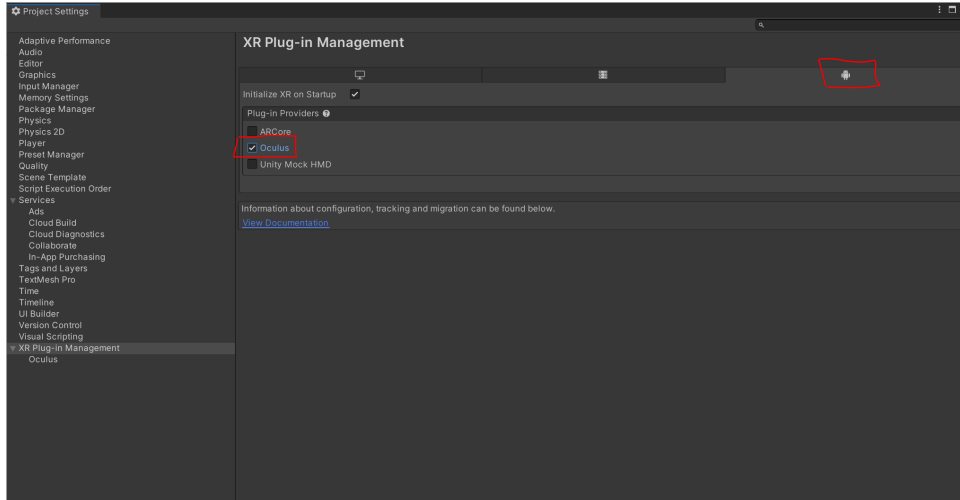
Then, navigate to the **XR Plugin Management** tab. Click the button to **Install the XR PLUGIN Management** Expand.



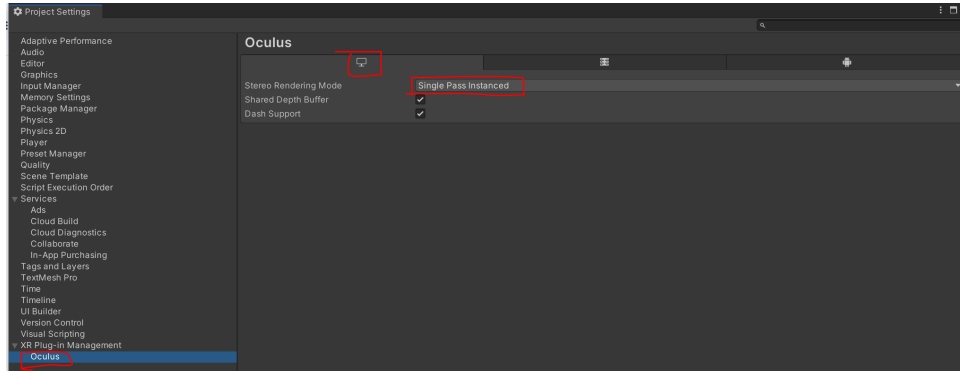
Once the installation is complete, select PC and select **Oculus**. It will go through another small installation step.



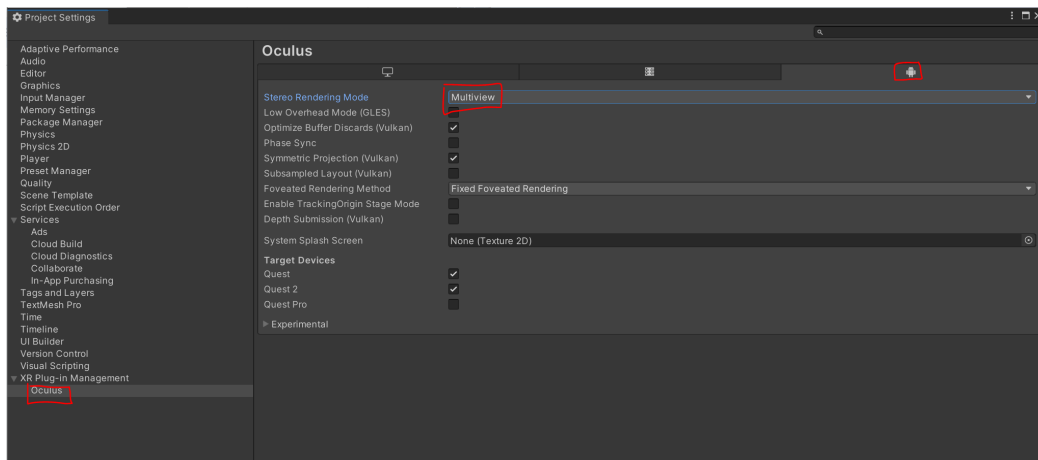
Now select the Android tab and again select Oculus.



Next select Oculus on the left and ensure PC is set to Single Pass Instanced.



Finally select the Android tab and make sure that the render mode is set to Multiview.



Starting your first project

Locate the unpacked **Custom Content Template Project** in your Assets and open **ExampleScene**. You can use this scene as an experimentation area and working example for custom content *game mode* setups in Onward.

➤ IMPORTANT

Rename your Example Scene and save it as a separate file. This will avoid any issues if the ExampleScene file is updated in the future.

➤ IMPORTANT

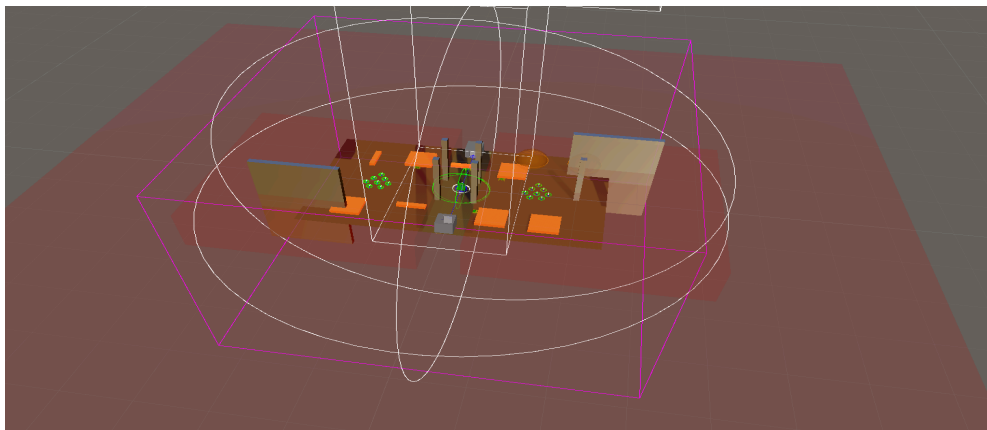
It is recommended to build your own map set-up by following the steps in this document so you have a full understanding of each component and how it is used. You can still use the Example Scene as a reference of what a valid set-up looks like.

Updating an existing project

1. Make sure to backup your project before updating.
2. Confirm that you have renamed any test scene and that it is no longer named **ExampleScene**.
3. Drag the new custom content package into your Assets folder inside the editor and accept the import of new and changed files.

Creating a custom map

After loading the **ExampleScene**, you will see a simple map, as shown in the screenshot below.

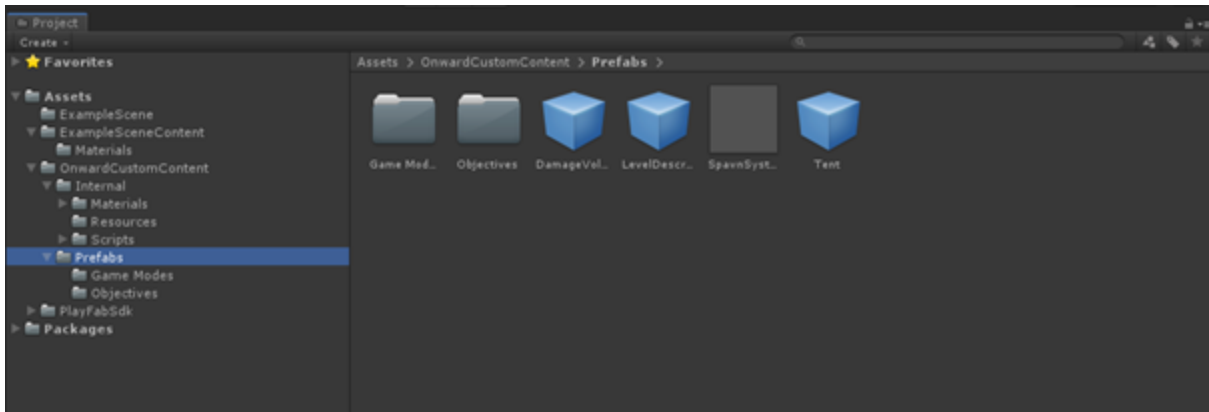


➤ **IMPORTANT**

Make sure you have already renamed this scene and saved it as a separate file. If you have not, do so now. This will avoid any issues if the **ExampleScene** file is updated in the future.

This scene is a valid map for all game modes, which can be used as a starting point to experiment with your own custom map.

All Unity prefabs for this project are located in **Assets > OnwardCustomContent > Prefabs**



➤ **IMPORTANT**

Do not modify the existing prefabs. **Changing these prefabs will cause errors** in your build. You may make modifications to the prefabs within the scene hierarchy instead.

Now, we will begin to set up these game objects in a blank scene (or your own pre-existing map) to show you how they work and to allow you to build scenes from scratch. You can also follow along and inspect the game objects in the Example Scene to gain an understanding of how they work.

You can always look to the original Example Scene for a reference on a working setup of a given game mode or feature.

To begin, every scene needs the following prefabs, you can find them in **OnwardCustomContent/Prefabs**:

- Tent
- MapQuad
- LevelDescriptor

Place these three Prefabs into your scene.

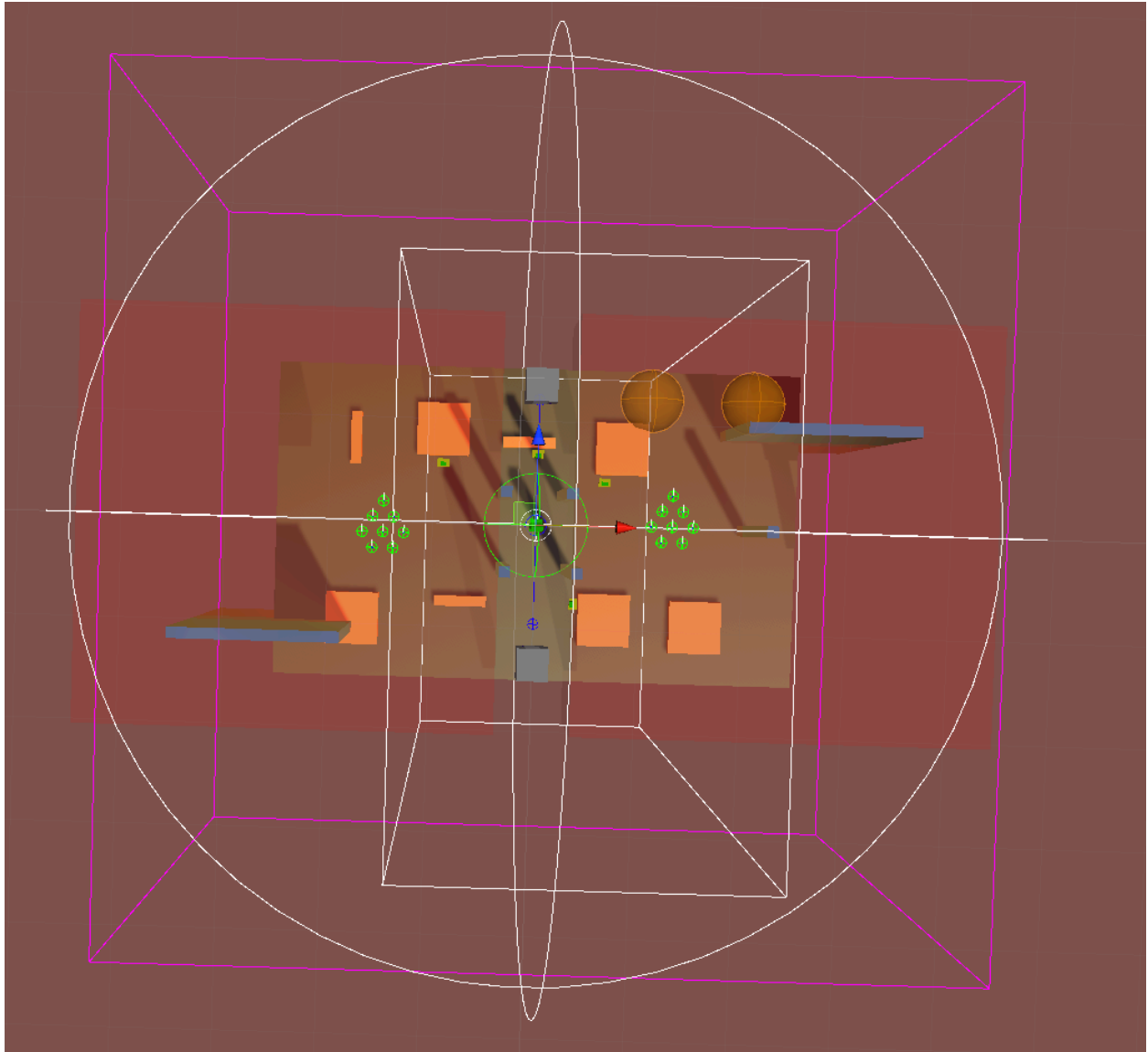
➤ **IMPORTANT**

Ensure the **Tent Prefab** is moved to be situated well below the play area of the map, or players may see

your level geometry through the tent's windows.

➤ **IMPORTANT**

Ensure the **MapQuad Prefab** is appropriately-sized over the map, encapsulating its playable-area within the magenta-pink wireframe box that is MapQuad's bounds. Use the size values in the inspector to shape the MapQuad to fit your map. Center the MapQuad over your map as shown in the picture below.

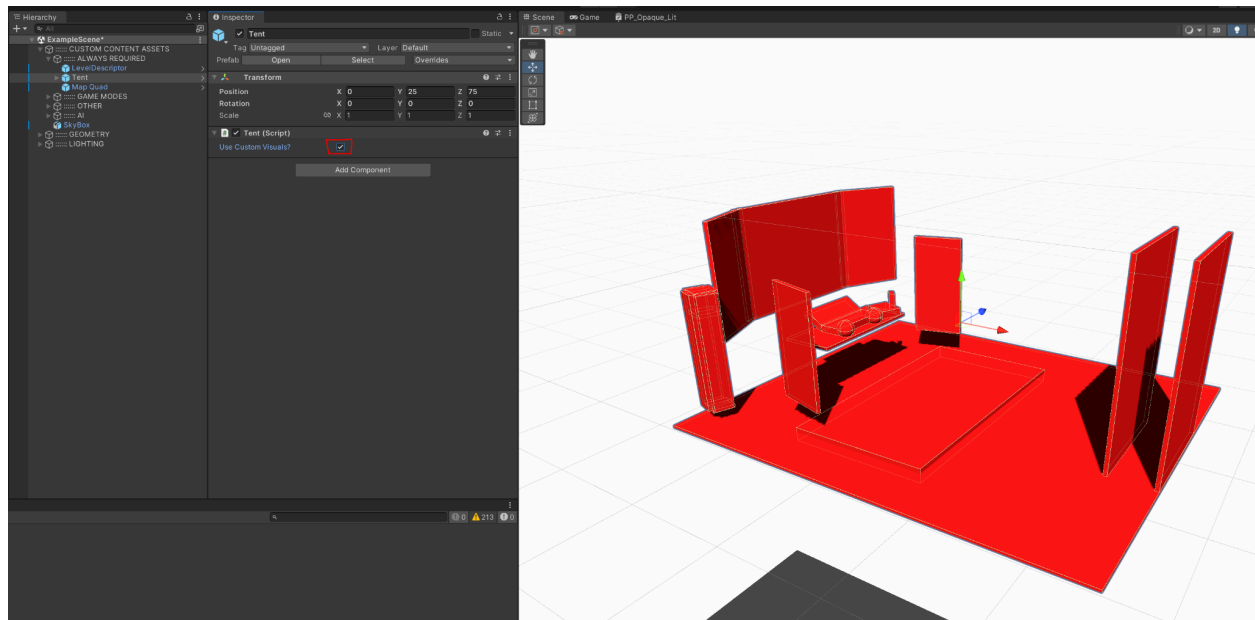


The LevelDescriptor's position does not matter, it will contain information about your map for later use.

Now that you have essential gameplay objects in your map and have set them up properly, you can move onto adding Game Modes to your map.

Custom Tent Visuals

By default the tent will be created in game for you at the tent position in your map. You can set it to not load visuals and build your own tent if you want. To do so check the **Use Custom Visuals** box. This will display a red framework of the tent to make it easier to place assets.



Setting up game modes

In order to support a game mode, the Prefab for that game mode must be added to your custom map scene. We will be using the Uplink game mode for this first example, but you can add more/other game modes by following the same steps. Note that some game modes have additional steps beyond Uplink, those will be detailed below this initial setup.

➤ IMPORTANT

Remember, every map is required to support the Free Roam game mode and at least one other mode.

➤ IMPORTANT

If you are starting from scratch with a blank scene, create some geometry to serve as the 'ground' in your scene.

The **Example Scene** is designed to support all game modes and thus has pre-existing setups, but we will be replicating a new **Uplink** game mode in your own scene to start off.

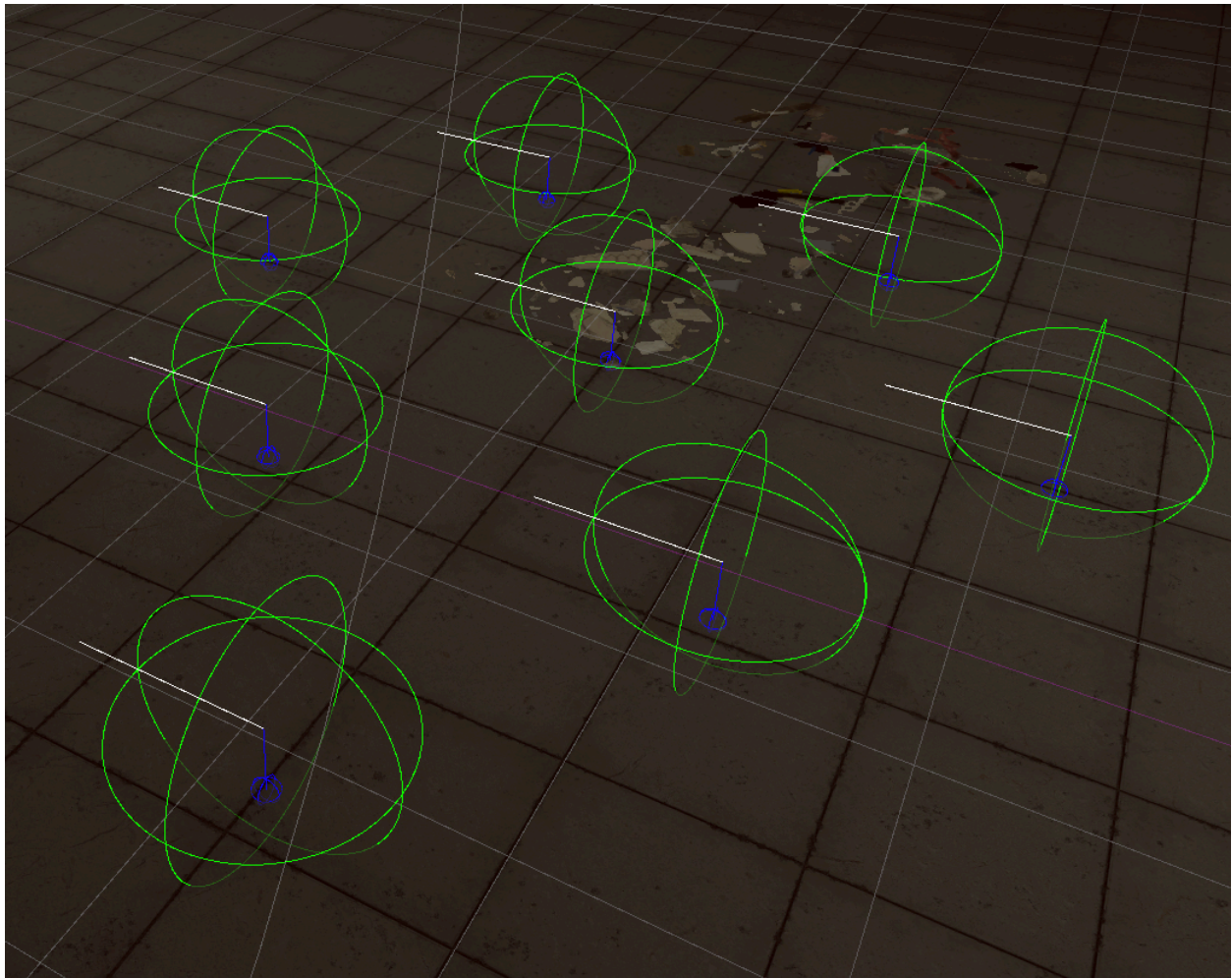
To replicate its setup, begin by opening ***Assets > OnwardCustomContent > Prefabs > Game Modes*** and add **GM_Uplink** and the **GM_FreeRoam** prefabs to your scene. Game Mode prefabs will always start with 'GM_'.

Set objectives & spawn points

Once the game modes have been added, you will need to create *objectives* and *spawn points*. You can find the objective prefabs in **Assets > OnwardCustomContent > Prefabs > Objectives**. In this case, drag the **Obj_Uplink** prefab into your scene. (*Objective prefabs will always start with 'Obj_'*) Select the **Obj_Uplink** prefab in your scene hierarchy so you can see the information below in the inspector window.

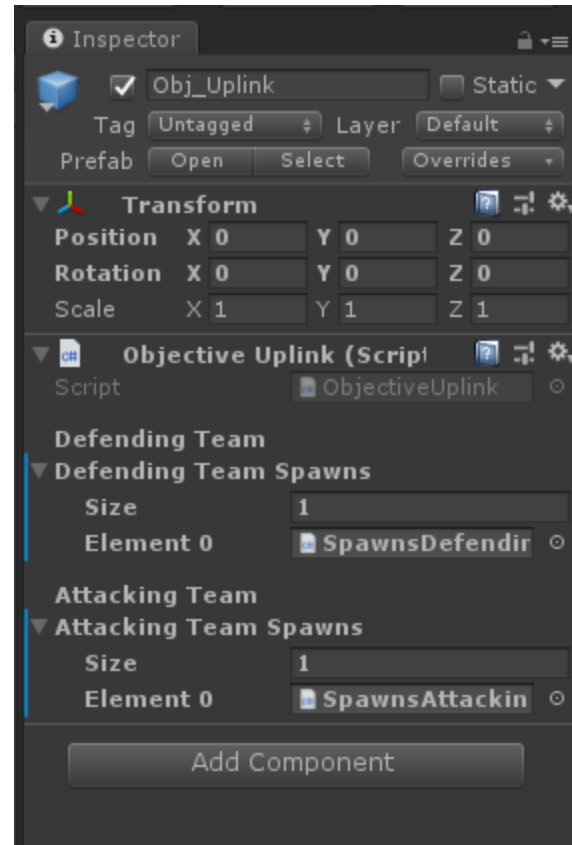


Now it's time to add spawn points. Open **Assets > OnwardCustomContent > Prefabs** and drag two **SpawnSystem** prefabs into your scene. Rename each prefab (F2, or through the inspector window) to **SpawnsAttacking** and **SpawnsDefending**. These are example names, you can rename them to whatever you wish.



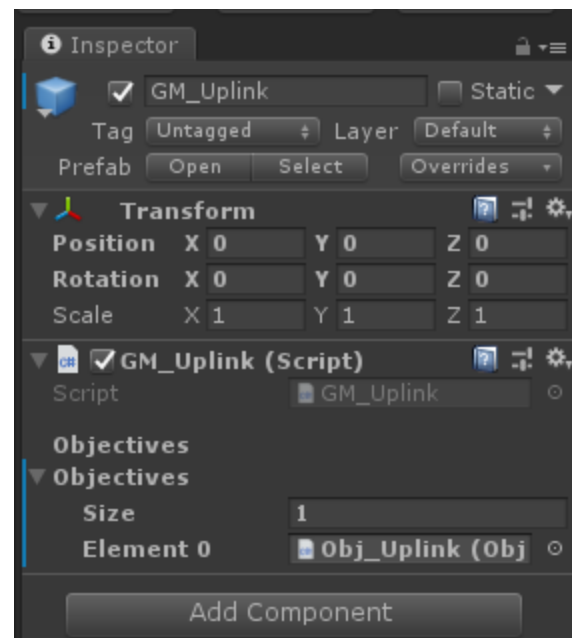
SpawnSystem prefab in scene.

Go back to the inspector window for the **Obj_Uplink** prefab and set the **Defending and Attacking Team sizes** fields to **1**, and then drag and drop the **SpawnSystem** prefabs that you have named **SpawnsAttacking** and **SpawnsDefending** in their respective fields for **attackers** and **defenders**. Your settings on the **Obj_Uplink** should now look like this



Correct spawns (you can have more than 1 spawn for Attackers to vary the gameplay)

The last item in this step is to make sure that the **GM_Uplink** prefab is referencing the matching **Obj_** prefab. Select **GM_Uplink** from the hierarchy to open it in the inspector. Increase the Size of the Objectives field to **one (1)**. Drag the **Obj_Uplink** prefab into the field of the **Objectives** dropdown. Your settings should match the image to the right



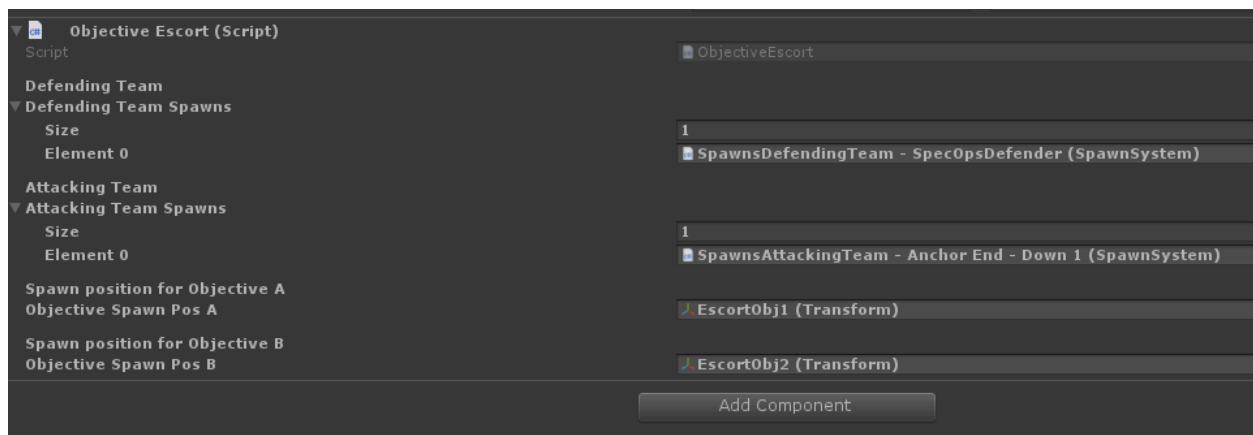
If you want to create an Uplink game with a single objective, that's all you need to do! Your Uplink game mode will now work. Refer to **Section 7: Testing for testing information**. You can add more objectives and game modes using the steps above, just add more Obj_Uplink prefabs (that are set up with different spawns) to the GM_Uplink prefab.

Be aware that game modes can share spawn points, and you can also modify individual spawn points within the SpawnSystem object in your hierarchy.

We recommend that you place a **DamageVolume** prefab below the map and scale it to cover the entire footprint of your map. If a player somehow falls through the map, this will prevent them from falling endlessly. This has already been done in the ExampleScene, and you can use this as a reference for your own maps. You can find **DamageVolume** in **Assets > OnwardCustomContent > Prefabs**.

Setting up the Escort game mode

Escort is a game mode in which MARSOC has to escort 1 member of their team (The VIP) to one of two extraction points- marked with red smoke grenades. For this mode use **GM_Escort** and **Obj_Escort** prefabs. This mode allows you to define 2 different objective placements as dictated in the screenshot below:



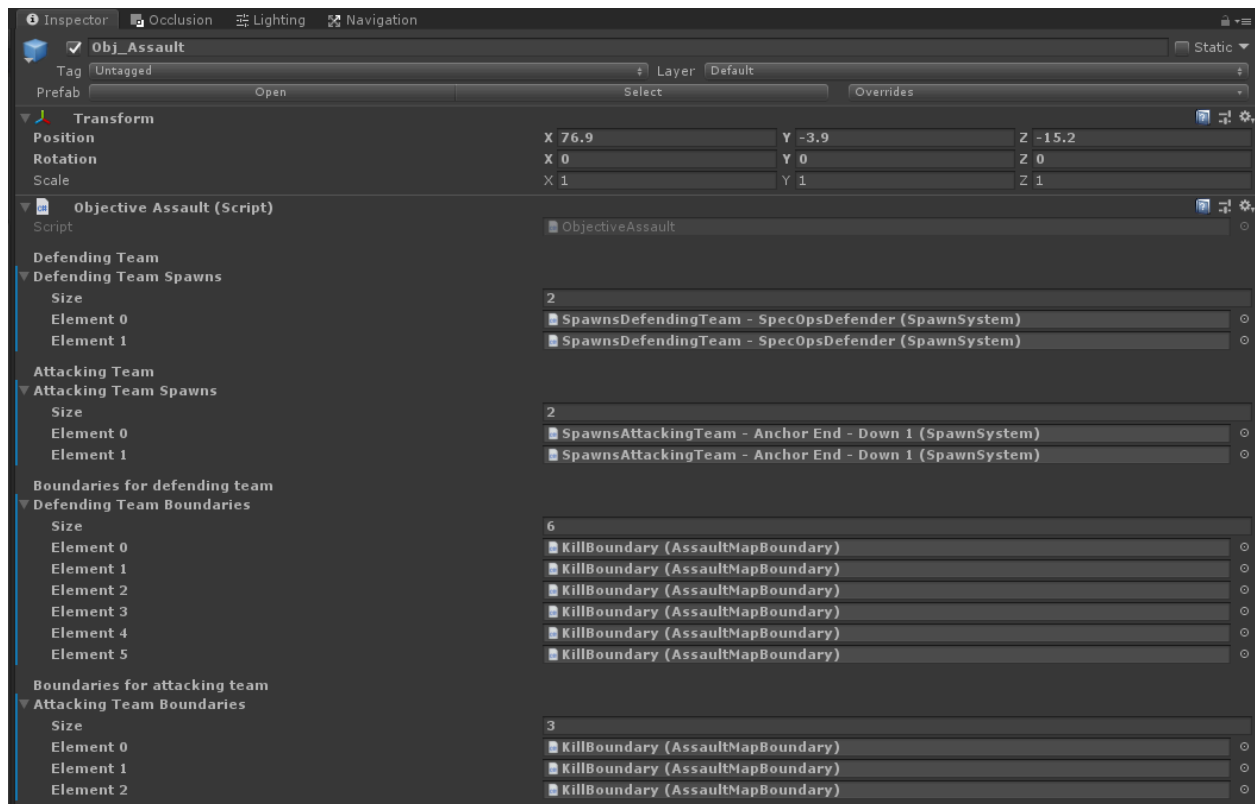
When you place the **Obj_Escort** prefab you will have two objective children that are created that you must place in unique locations.

Take time to consider the implications of two objectives to reach and how that might influence your gameplay.

Setting up the Assault game mode

Assault is a game mode which has MARSOC and Volk fighting over an objective with respawns for both sides. For this mode use the **GM_Assault** and **Obj_Assault** prefabs. To prevent spawn camping, additional no-go gameplay zones can be defined using the **AssaultMapBoundary** prefab. Place and scale these prefabs to fit around zones the opposing teams should not be able to encroach on (such as player spawns), then associate them with the **Obj_Assault** prefab. You can associate just a single zone or multiple to each team's limitations (as is done in this screenshot).

Place 2 SpawnSystem for each team, to allow them to choose between A and B spawn points in-game.



Boundaries for **ATTACKERS** will penalize **ATTACKERS** from entering the zone.

Boundaries for **DEFENDERS** will penalize **DEFENDERS** from entering the zone.

You can also rename your **AssaultMapBoundary** prefabs to help distinguish them from each other.

Setting up Fire Fight

Fire Fight has each team split into two smaller fire teams. You can have the teams spawn anywhere you want. Best practices are to spawn the teams in positions where they won't see the other teams fire team for 15 or more seconds. The mode will randomly select from a spawn set, randomly select an evac point, and randomly select a hard drive spawn location. Try to add a lot to provide new experiences for the players.

Most official maps have:

10-14 or more spawn sets

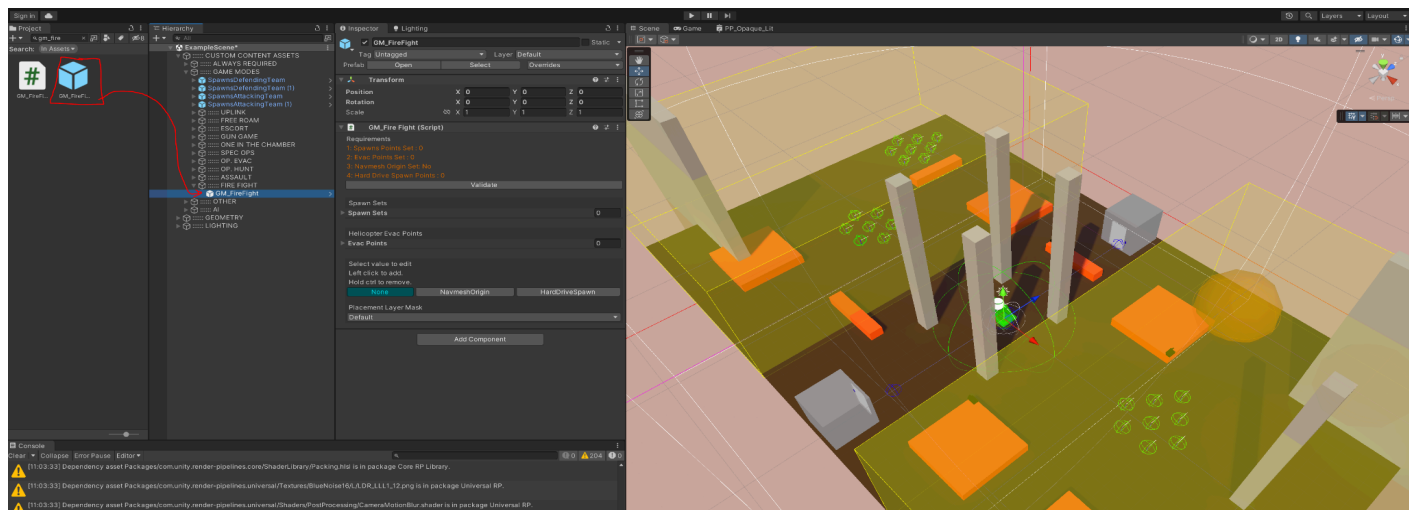
6-8 evac points

50-150 hard drive spawns

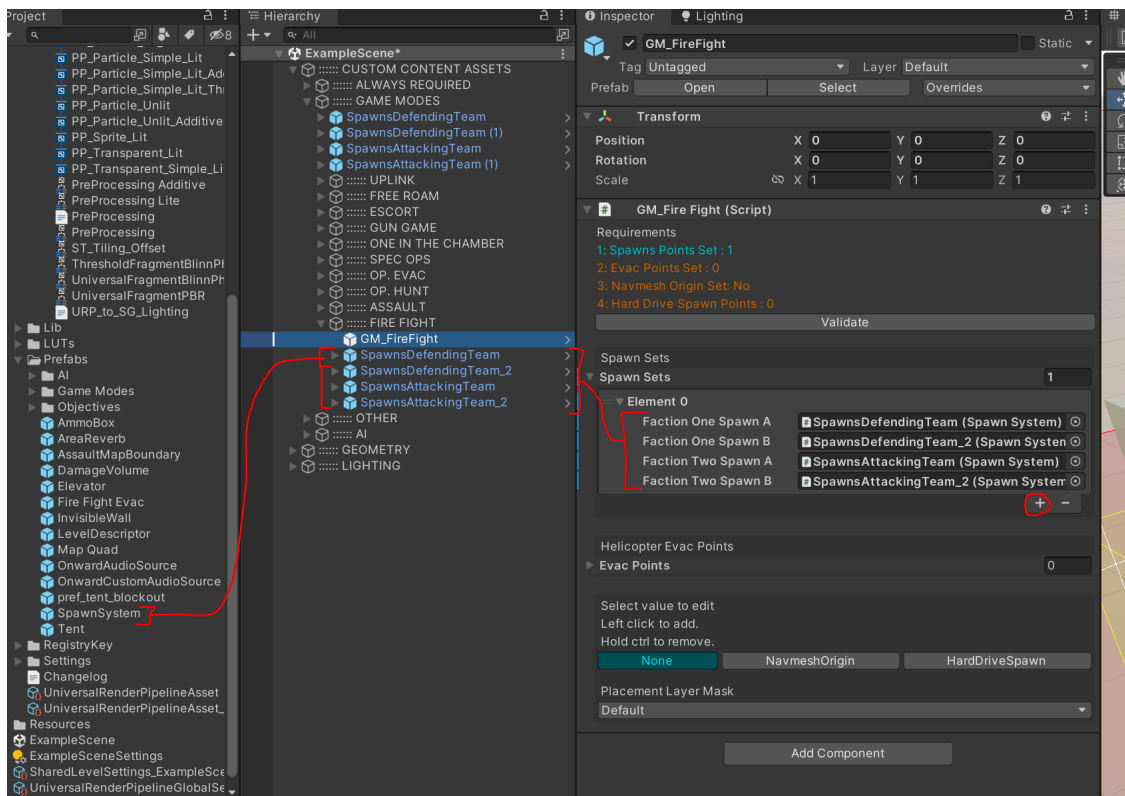
These are just what we found to work well, but feel free to play around and see what works for your map!

Step by step guide:

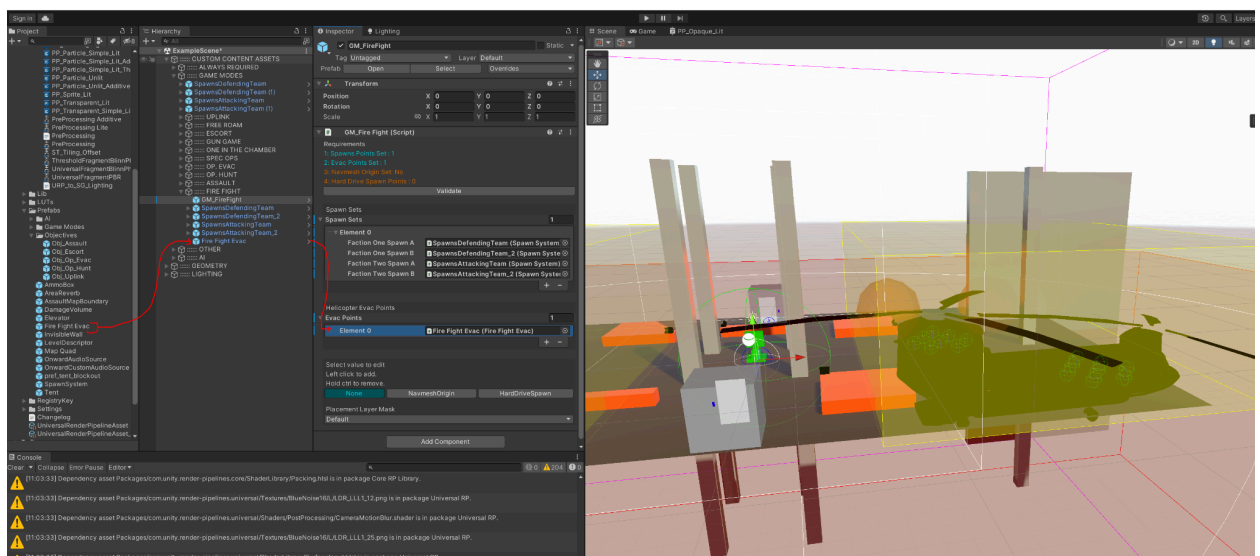
1. Add the GM_FireFight object to the scene.



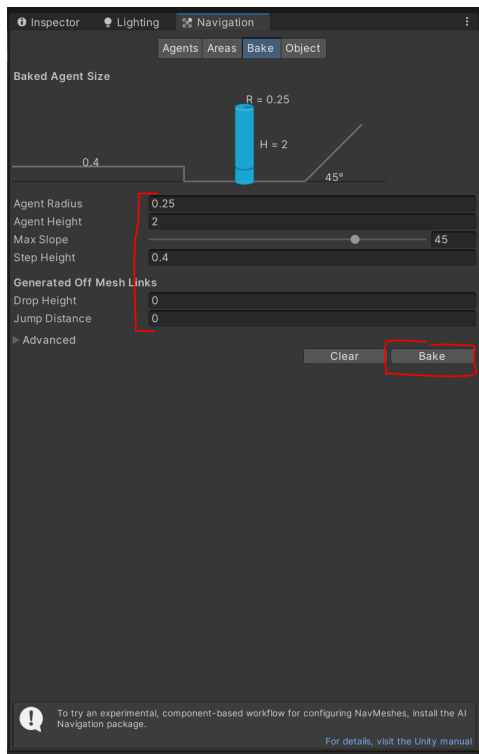
2. Add a spawn set to the Spawn Sets list on the GM_FireFight game object. Add four SpawnSystem prefabs to the scene. Each faction needs two sets of spawns. Assign the spawn systems to the Faction spawns inside the spawn set. You can have as many spawn sets as you want.



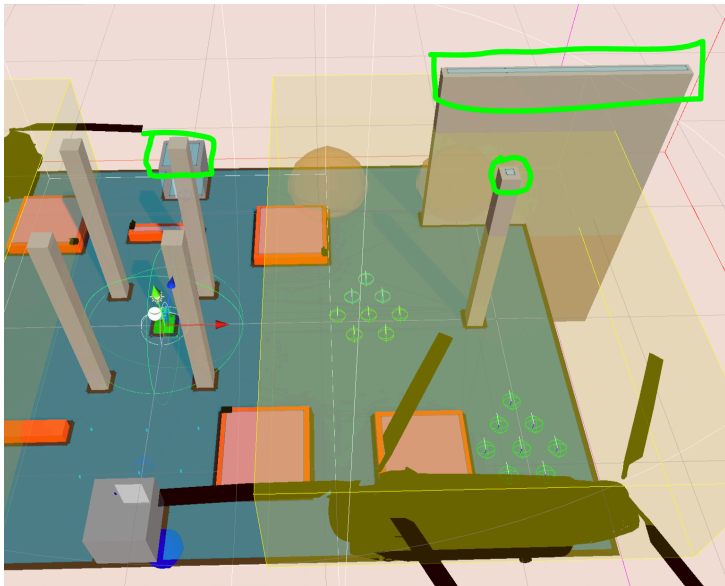
3. Add Evac points. Use the Fire Fight Evac prefab. Add the prefabs to the Evac Points list on the GM_FireFight prefab in the scene. Ensure the ramp is on the ground. You can add as many evac points as you want.



4. Fire fight requires a navigation mesh. If your map is already set up for the AI modes you should be good. If not, open the Navigation window (Window<AI<Navigation). Check the settings match these in the image, and click bake.

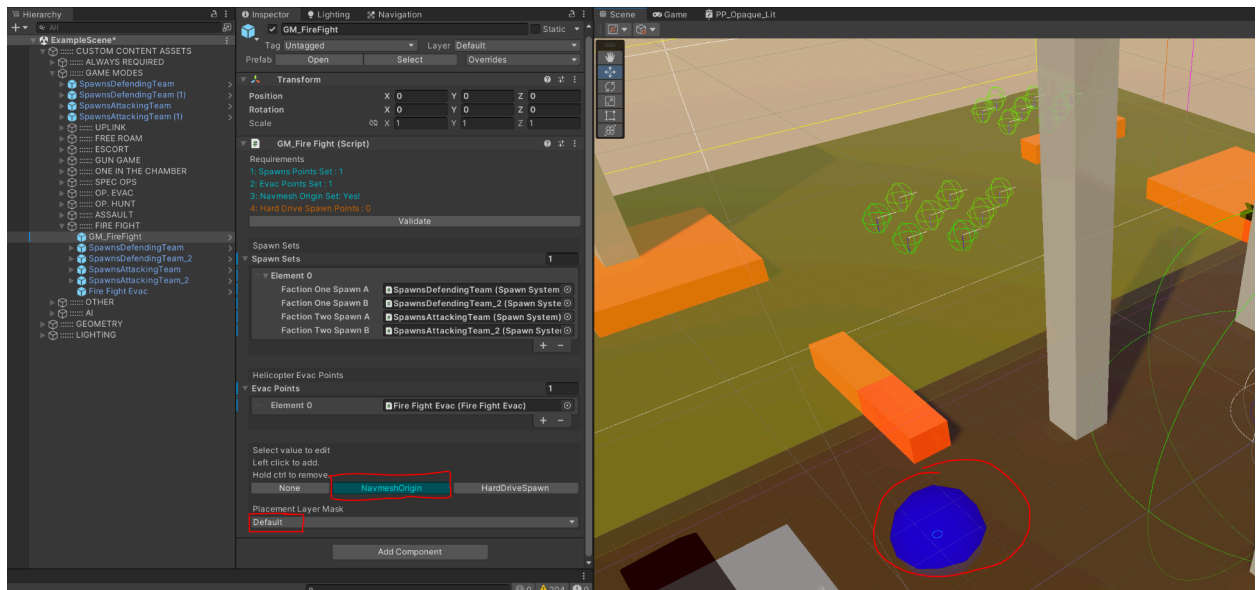


NOTE: You do not need to worry about separated navmesh areas (areas in green circles). The Navmesh Origin in the next step takes care of them for you.

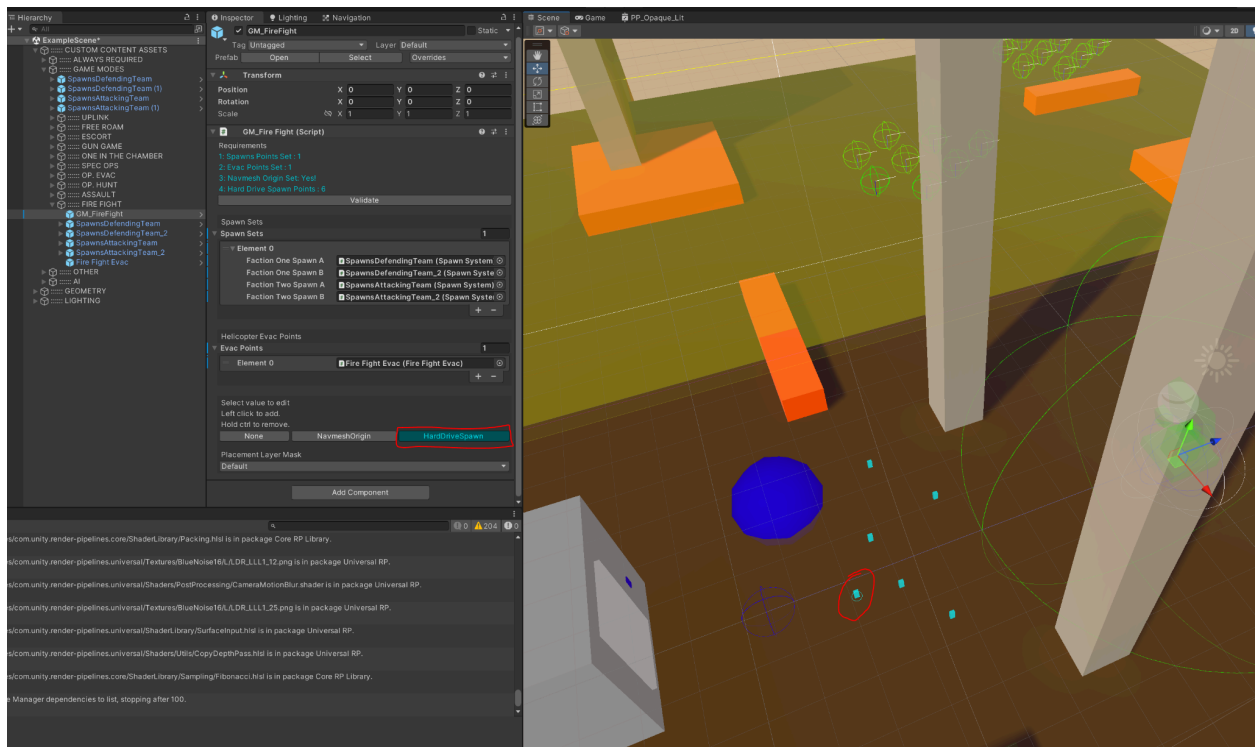


5. Next add a Navmesh Origin to the scene. Click NavmeshOrigin, check that the Placement Layer Masks are set up to something (Default is a good one to start with). Then just left click in the

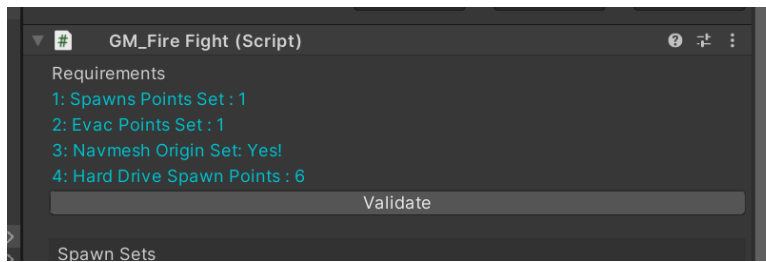
scene. You'll see a blue sphere in the scene.



- Now add hard drive spawns to the scene. Click HardDriveSpawn, then click to place. Ctrl+click will remove the spawn. You need at least 1 but you can place as many as you want. Remember to place the spawns where a player could reach it.



7. Check that all of the requirements at the top are finished. They will be orange if you still need to complete a step.



Setting up the Co-op game modes (Hunt, Evac, and Retrieval)

Onward has three co-op modes; Hunt, Evac, and Retrieval.

Hunt spawns a player-determined number of bots into the level and the players goal is to eliminate them. Players can select between 4 and 512 bots. You can set up spawn points to decide where the bots spawn and waypoints for them to follow. Bots will spawn randomly from the spawn points you set up.

Evac requires the players to survive until a helicopter arrives to rescue them. Players spawn at the start of a round and then spawns bots over time. Lower difficulties spawn fewer bots, higher difficulty spawns more bots. Bots will seek out and attack the player. If the helicopter has arrived then bots have a chance to defend the evac point.

Retrieval is similar to the PvP mode Fire Fight, it requires the players to locate a hard drive that will spawn randomly in the map. Bots will spawn randomly around the map like in Hunt (they will not respawn like in Evac). However, like in Evac a helicopter will land and the player must extract with the hard drive. Note that Retrieval reuses objectives from Hunt and Evac, so it is easier to set up but a little unique. We'll go over it below.

Co-op modes are set up using a few prefabs:

AI_Navigation - contains important navigation information

AI_WaypointSet - defines patrol paths for the bots

AI_Spawnpoint - determines where and what types of bots spawn

GM_Op_Hunt - contains all of the hunt objectives

Obj_Op_Hunt - defines one hunt objective including player spawns, bot spawns, and waypoint sets

GM_Op_Evac - contains all of the evac objectives

Obj_Op_Evac - defines one evac object including player spawns, bot spawns, and evac locations

GM_Op_Retrieval - contains all of the retrieval related objectives

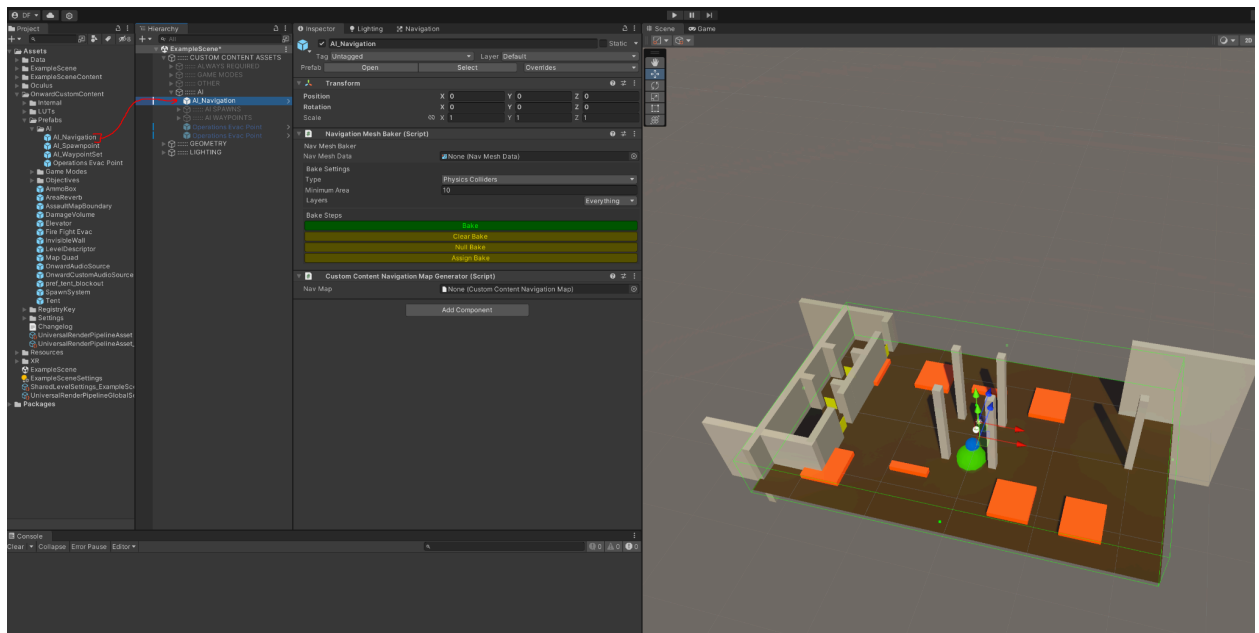
Operations Evac Point - determines where and how and evac helicopter will spawn

AI_Navigation Set Up

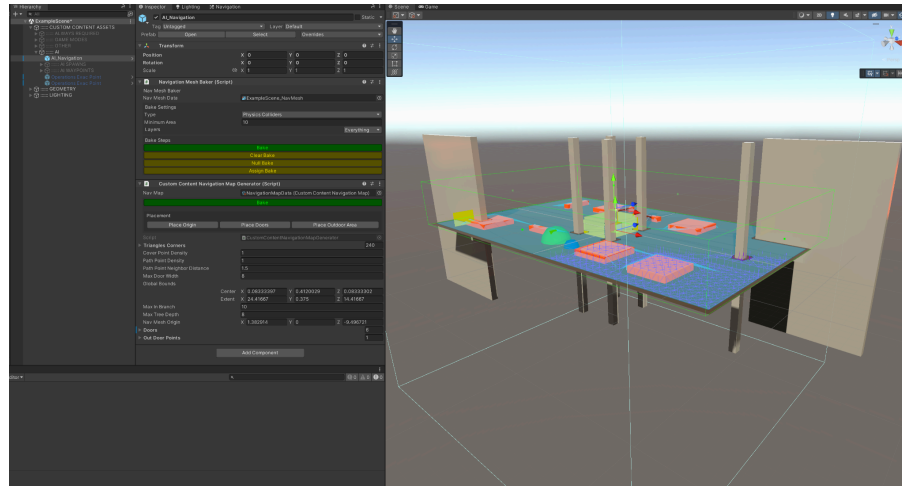
Navigation Mesh Baker

The navigation mesh baker is just a wrapper for Unity's normal nav mesh with a couple additional things like the ability to specify an area to bake. This must be done before moving to the Navigation Map Generator section.

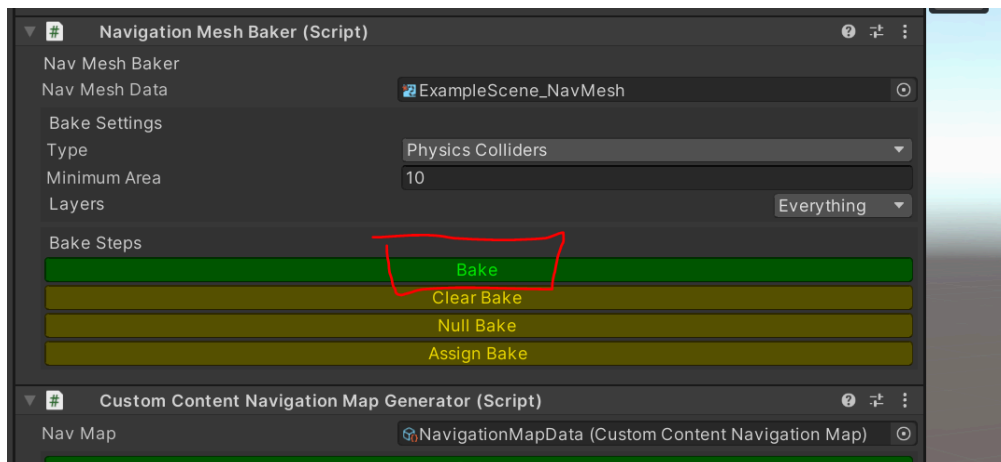
1. Drag the AI_Navigation prefab into the scene.



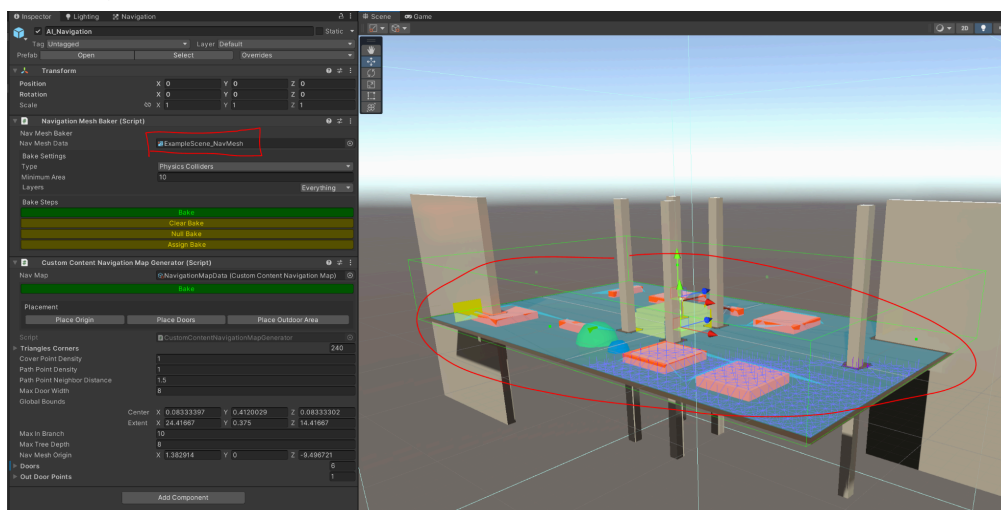
2. First use the bounds gizmos to cover the playable area of your level. The bounds are the green wire box and you can adjust the size using the small green squares at the center of each face.



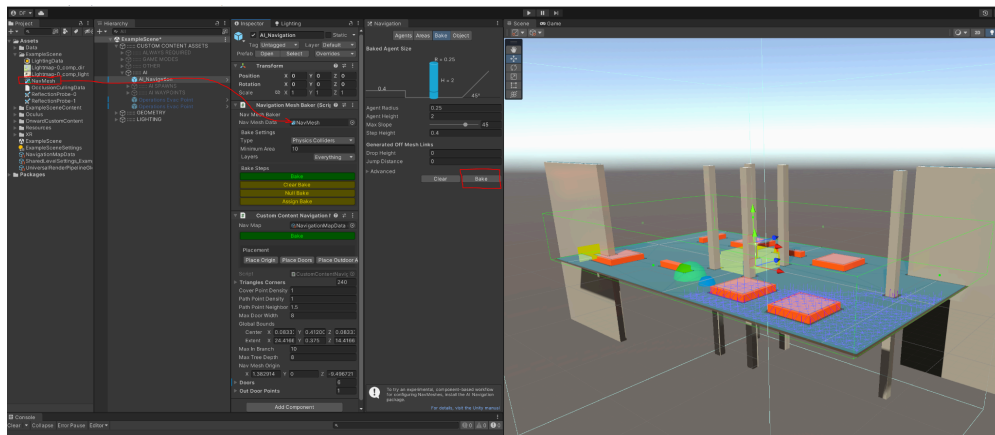
3. Click Bake



4. Make sure you can see a Nav Mesh Data object and the nav mesh in the scene (the blue stuff on the floor).



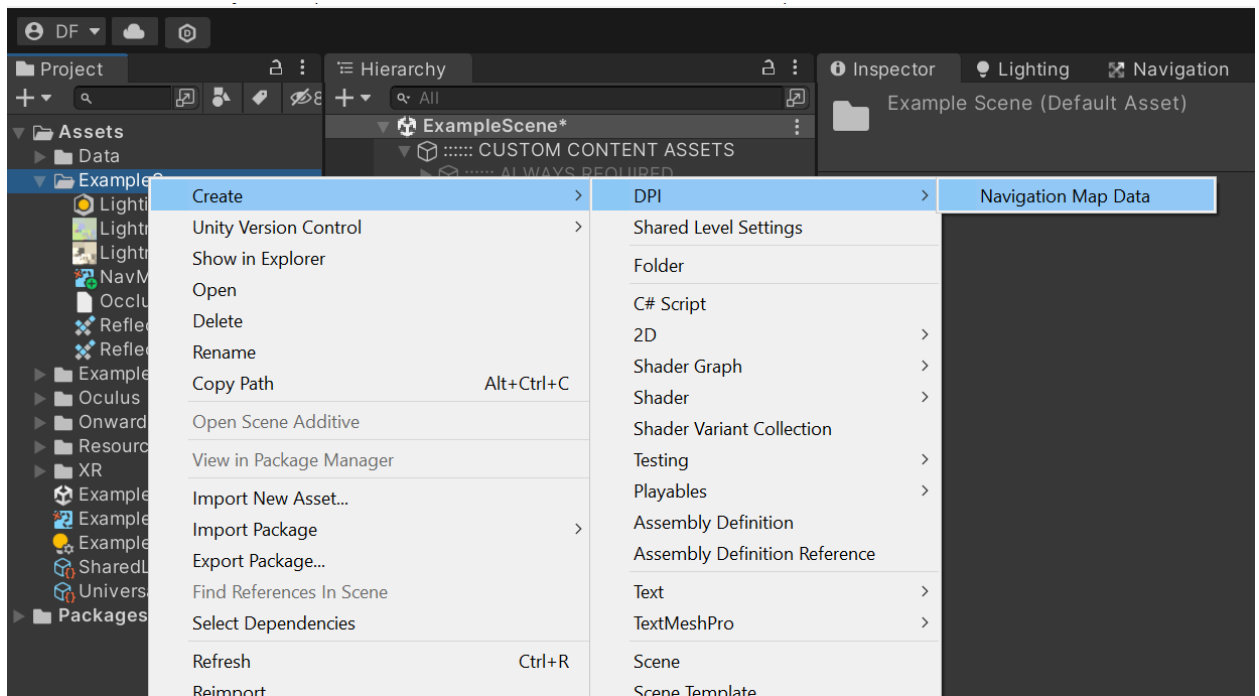
Alternatively if you need to customize the setting of the nav mesh, you can just use the built in baking system. You must then manually assign the Nav Mesh Data to the Navigation Mesh Baker.



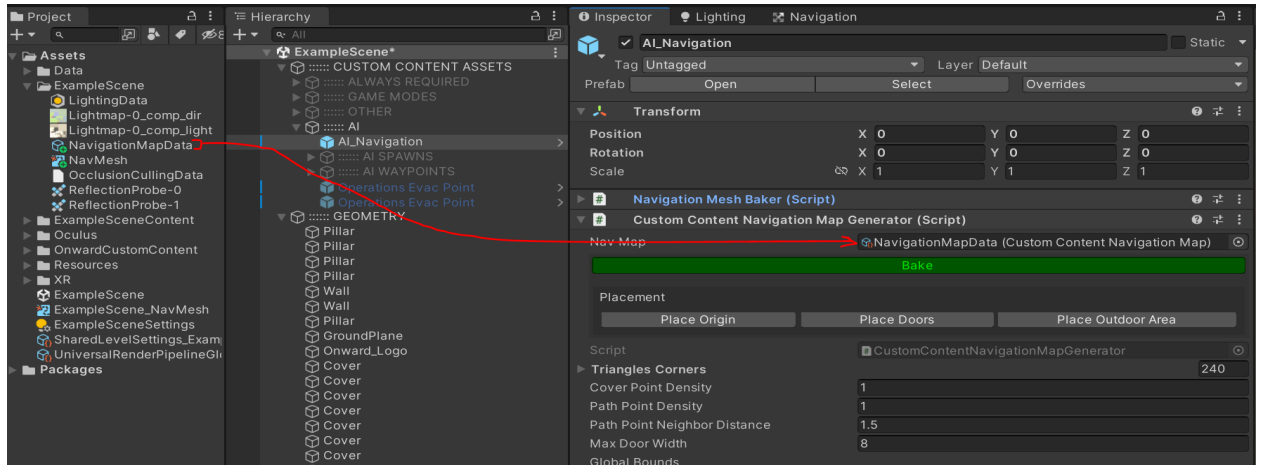
Navigation Map Generator

The navigation map generator helps the bots make decisions during runtime. It's an improved version of the old nav node system.

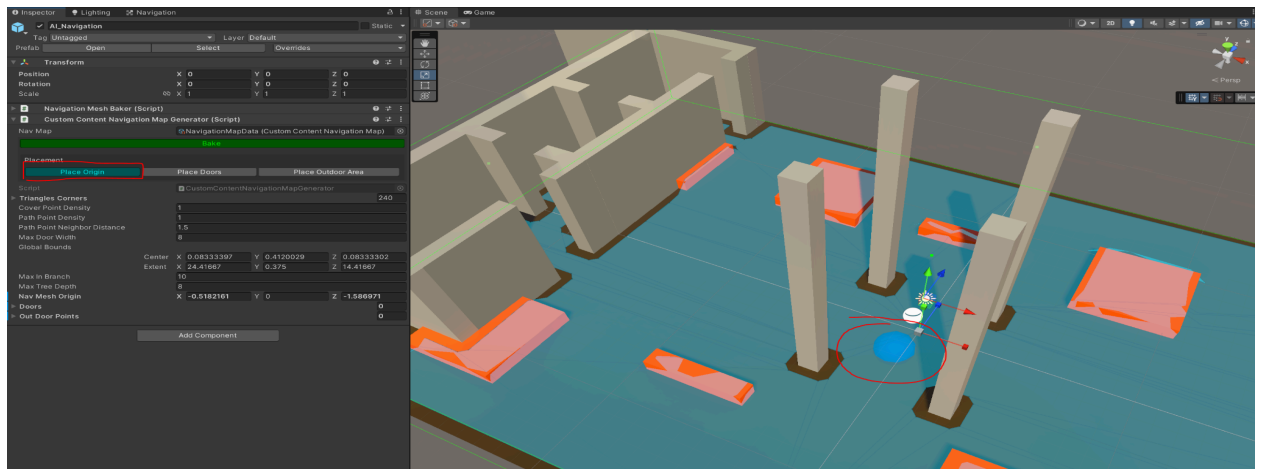
1. Create a Navigation Map data object. Right click a folder, go to Create < DPI < Navigation Map Data.



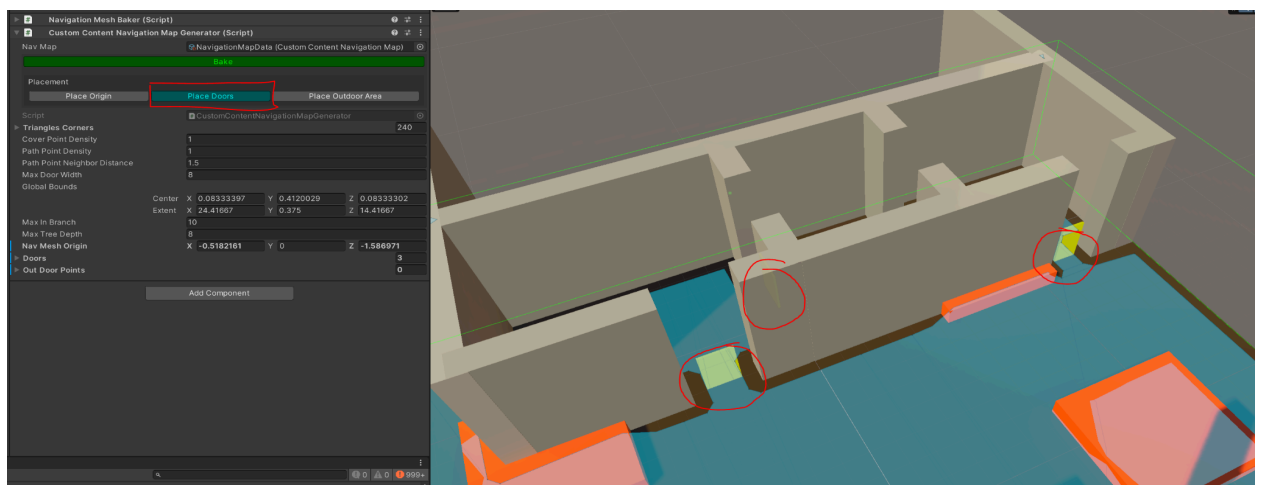
2. Name the data whatever you want and then drag it into the Nav Map Field in the Navigation Map Generator.



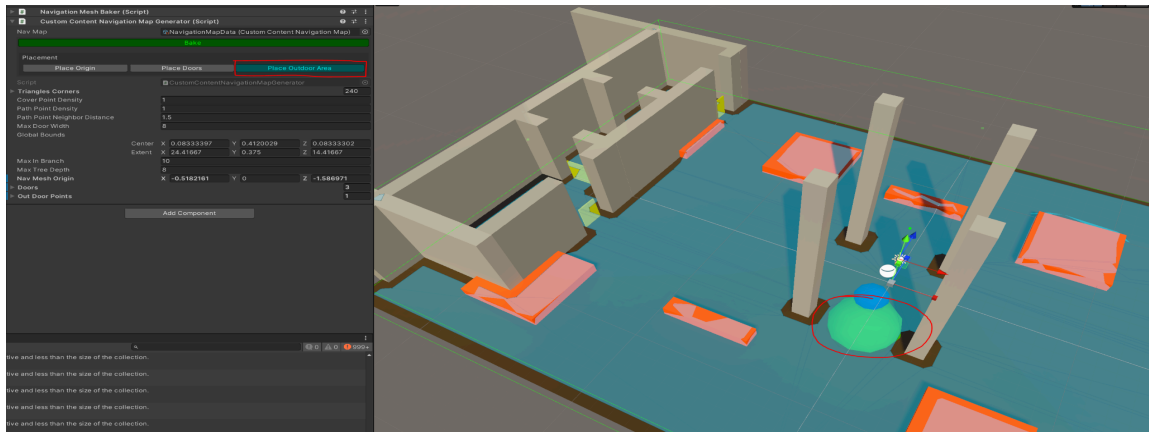
- Next Select “Place Origin” and click a point on the navigation map. Try to place the point outside and as close to the center of the map as possible. You will see a blue sphere when it’s placed.



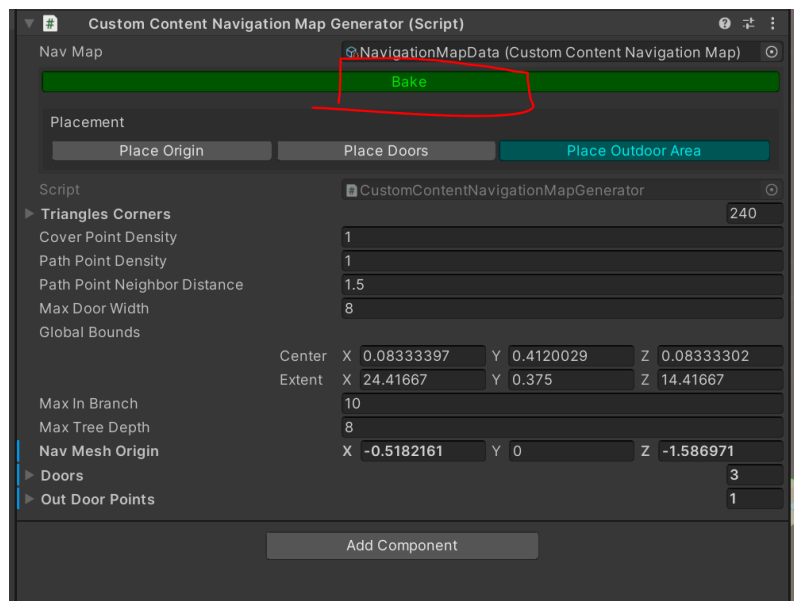
- Next select “Place Doors”. Then left click to place a doorway. It’s best to place doors in actual doorways but they can be used to divide up larger spaces. (if needed increase the “Max Door Width”)



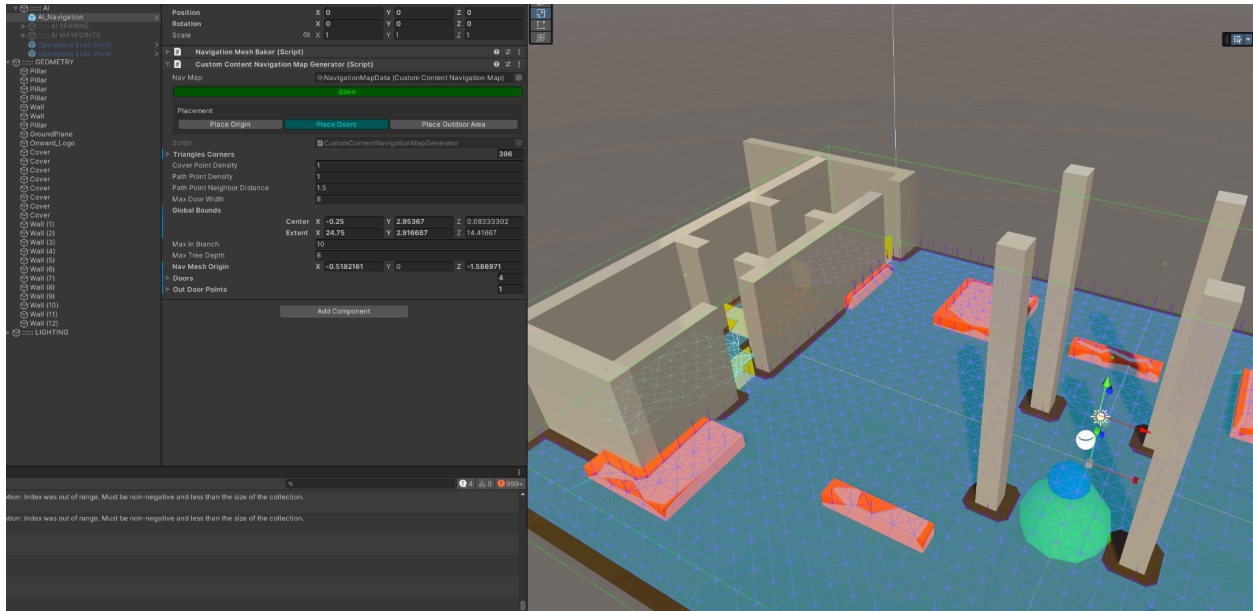
5. Next select “Place Outdoor Area”. Left click to place an outdoor space. Even indoor only maps need an “Outdoor” space so just pick a large room. You’ll see a green sphere when it’s placed.



6. Finally click “Bake”. Large maps can take a few minutes. Small maps a few seconds.



When the Bake is finished you will see a bunch of little lines on the map. Rooms will be different colors.

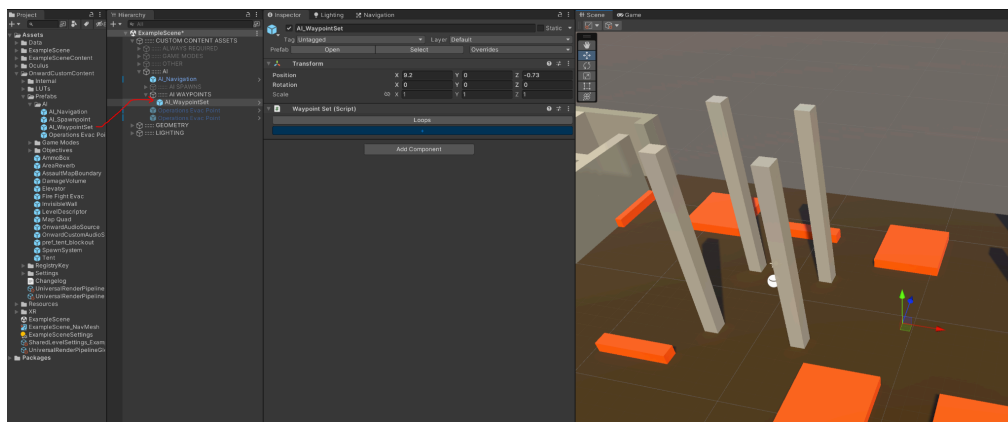


That's it for the AI navigation set up. Make sure you save your scene. If you are upgrading a map from a previous version of the SDK you can delete all the old prefabs.

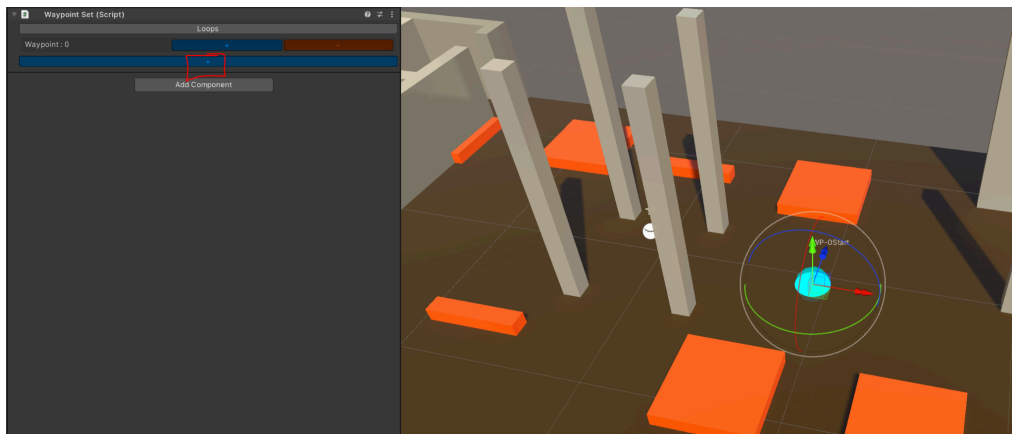
Waypoints

You can now define paths or waypoint sets for the AI to patrol along. Waypoints only affect Hunt. Waypoints can be used for just one objective or shared between objectives.

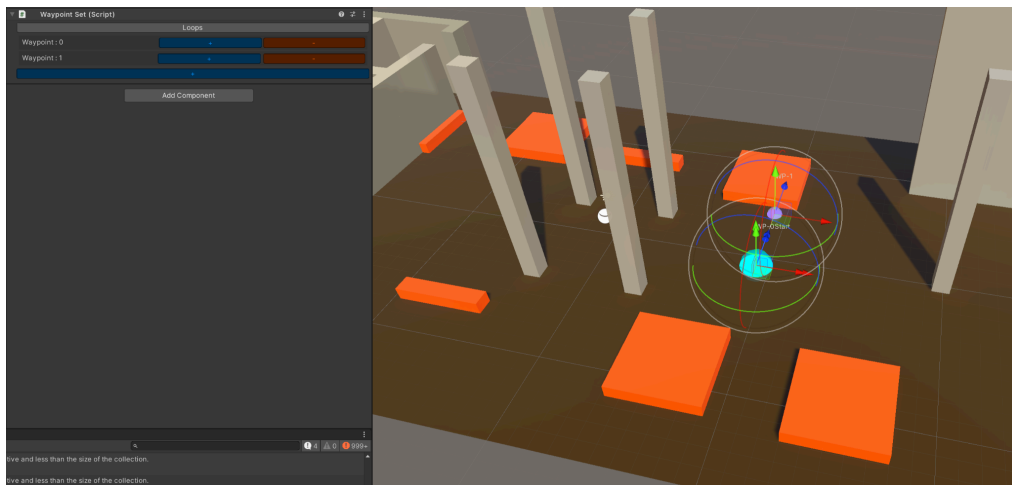
1. Drag the waypoint set prefab into the scene.



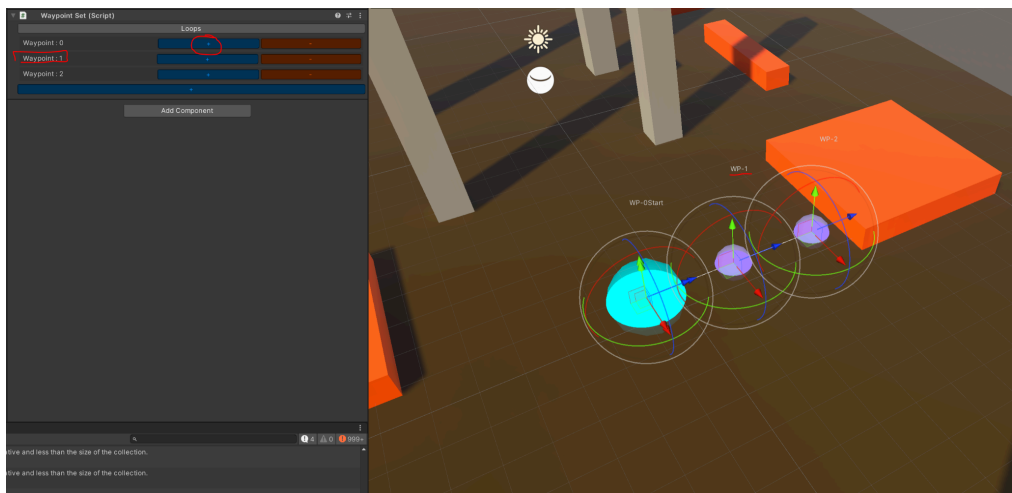
- Click the “+” button to add a point. This first point is the starting point.



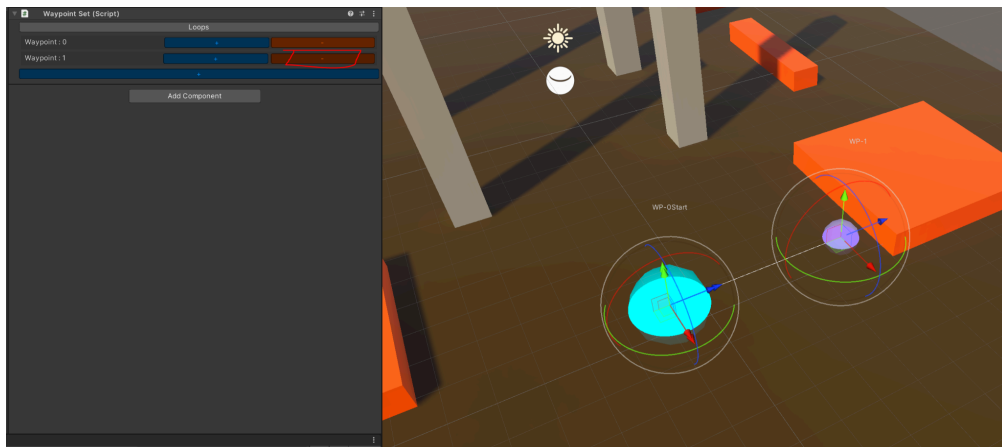
- Press the “+” button again to add a second point. The start is light blue, the waypoints are purple.



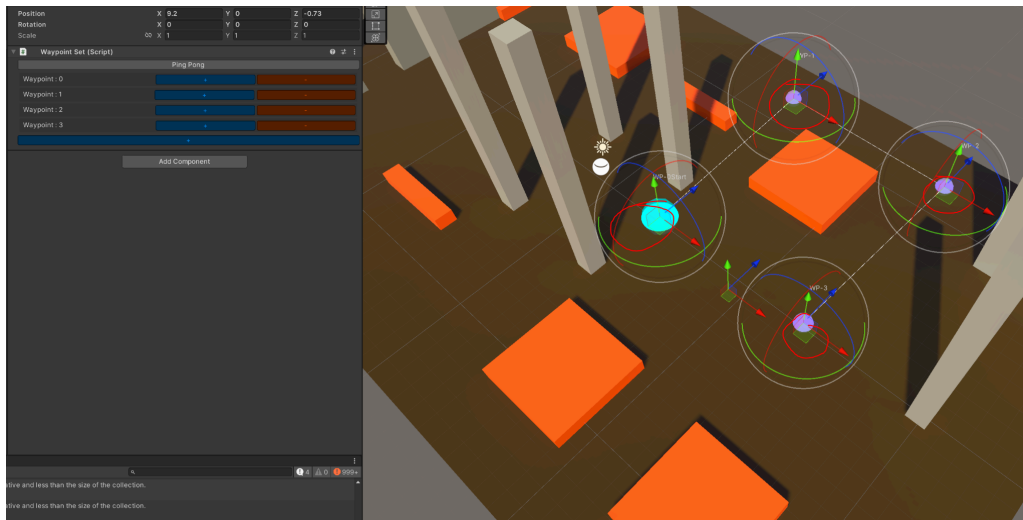
- Clicking the “+” next to each way point will add a point in between that point and the next.



5. Clicking the “-” button deletes that point.

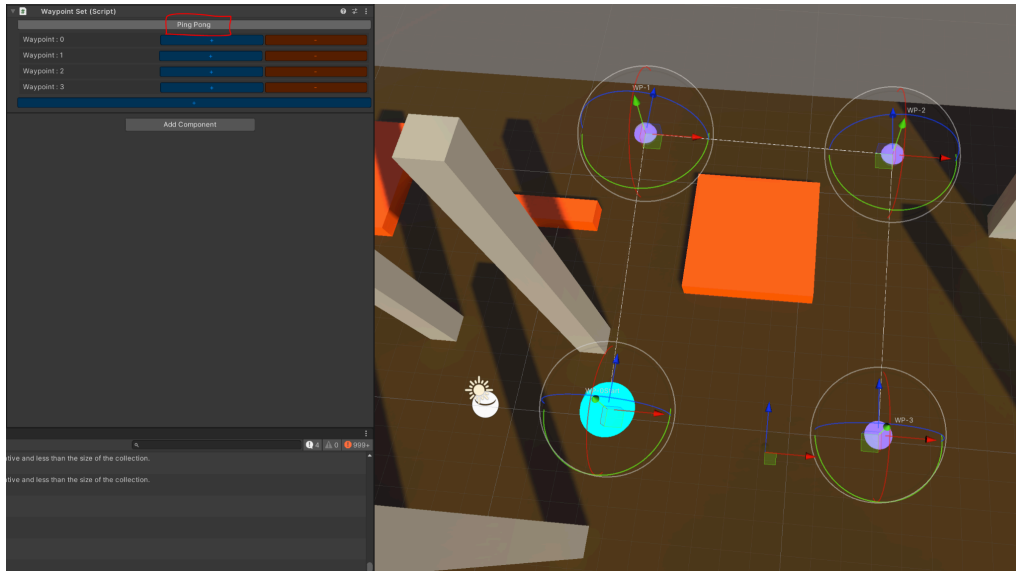


6. Use the transform gizmos to position the waypoints. Waypoints must be placed near a navmesh. There's a validation step during the build process that will snap them to nearby navmeshes for you so you don't need to be perfect, within a 5 meters is fine.

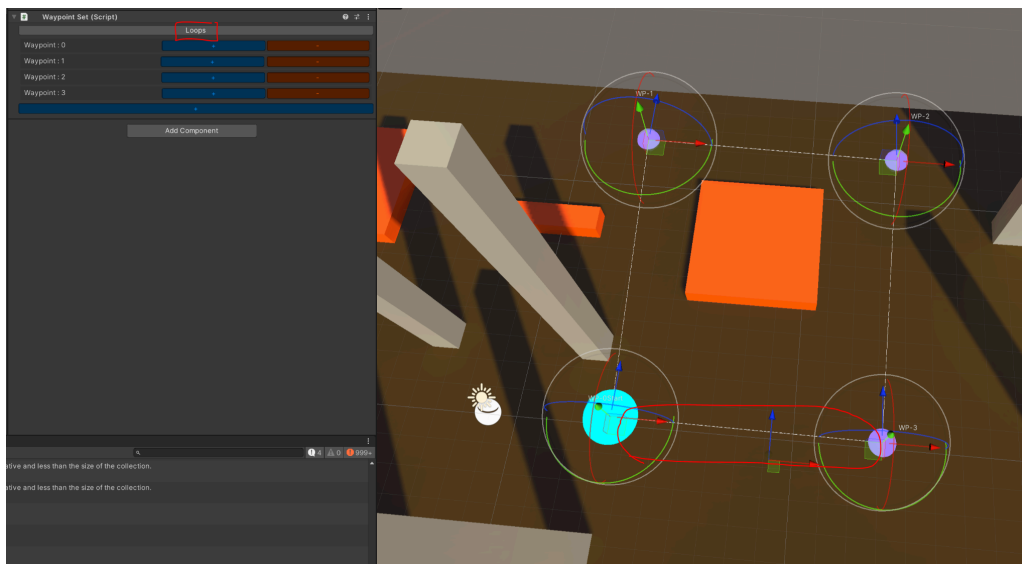


7. You can swap between a Looping or Ping Pong path by clicking the button at the top. This will cause the bots to either; Ping Pong - reverse direction when reaching the start and end, Looping - go to the start point when reaching the end. You can see this in the path too.

Ping Pong



Looping

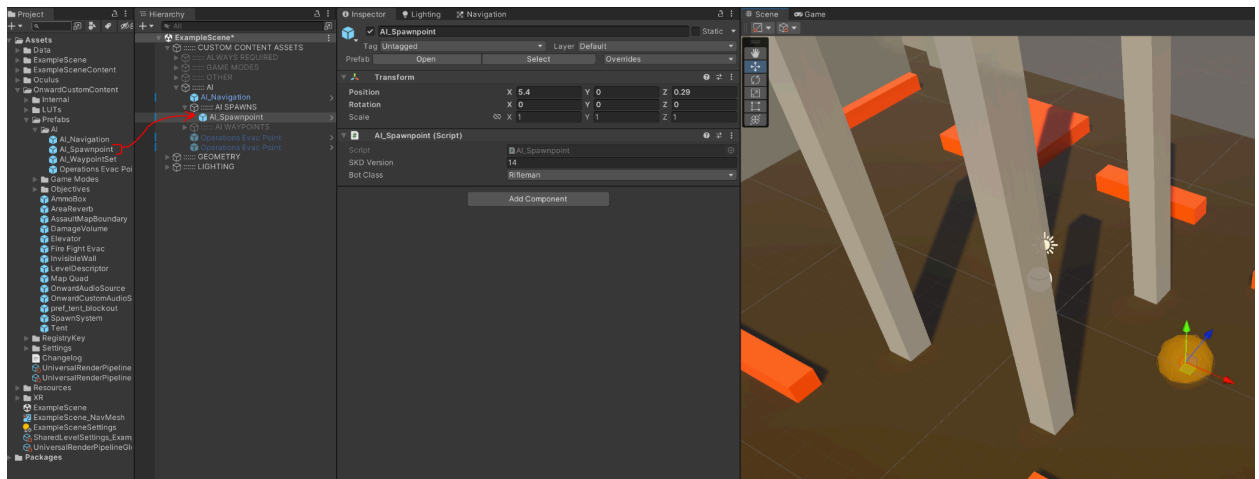


Note: The line between waypoints can go through things, the line only signifies the order the bots will visit each waypoint - the bots will still pathfind from waypoint to waypoint. So for example if in the image above a line went through the orange square, the bots would still walk around it.

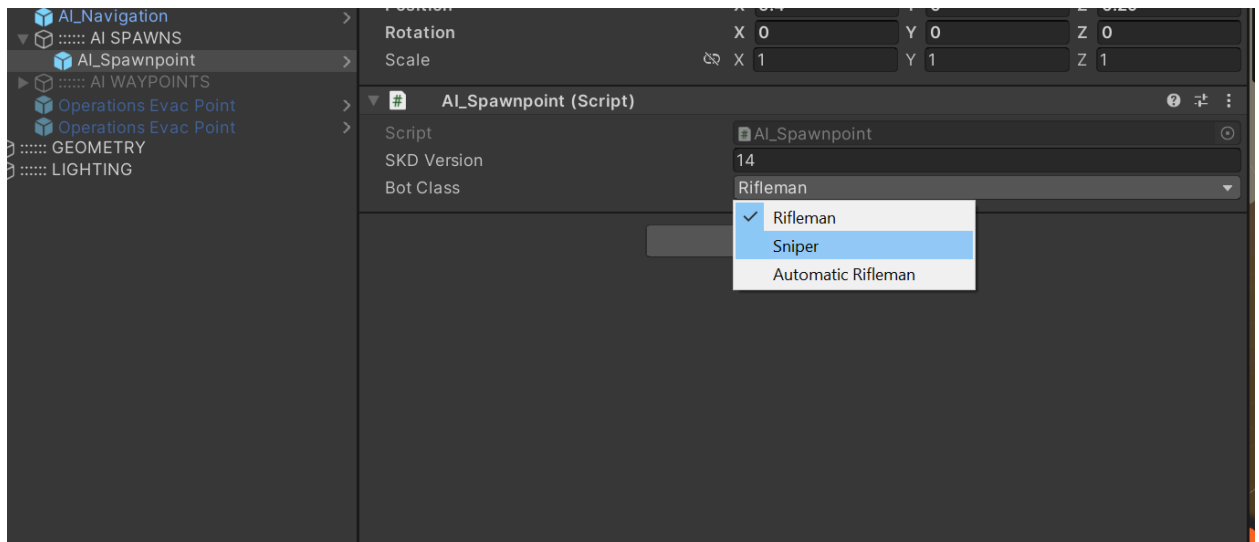
AI Spawners

Next, you will add some AI Spawners to the level.

1. Drag the AI_Spawner prefab into the scene. Spawn Points must be placed near a navmesh. There's a validation step during the build process that will snap them to nearby navmeshes for you so you don't need to be perfect, within a 5 meters is fine.



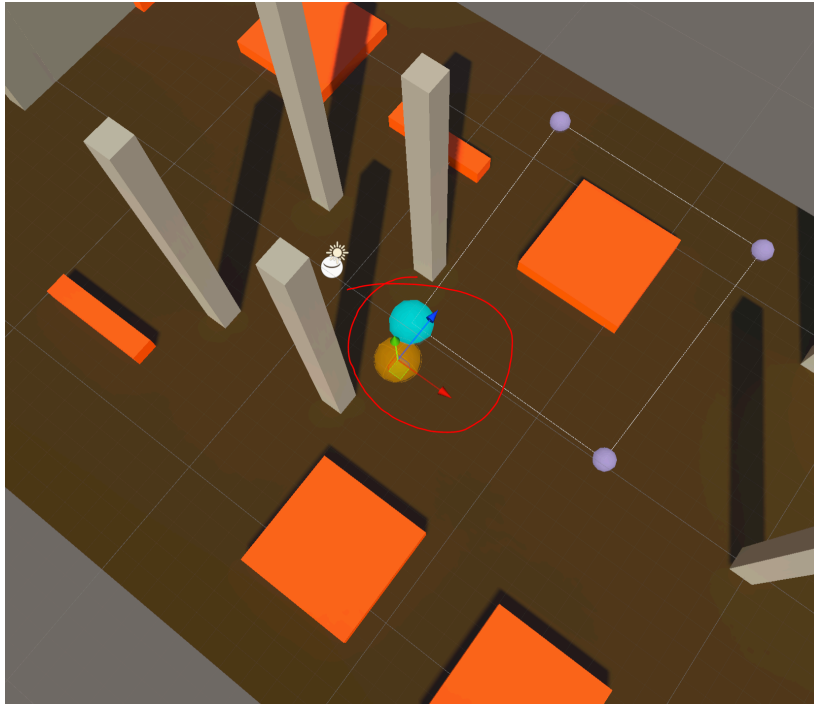
2. Here you can select from 3 options; Rifleman, Sniper, and Automatic Rifleman. (NOTE: Only the rifleman will spawn in the game for now. Other classes are coming in the future and we want to future proof the SDK a bit 😊)



You need at least 32 spawn points in the level. Spawn points can be shared between Hunt and Evac, or you can define spawn points per objective. If you are upgrading from an older SDK the important thing to note is that the spawn points are individual now, the squad spawning is gone to provide more flexibility in bot placement.

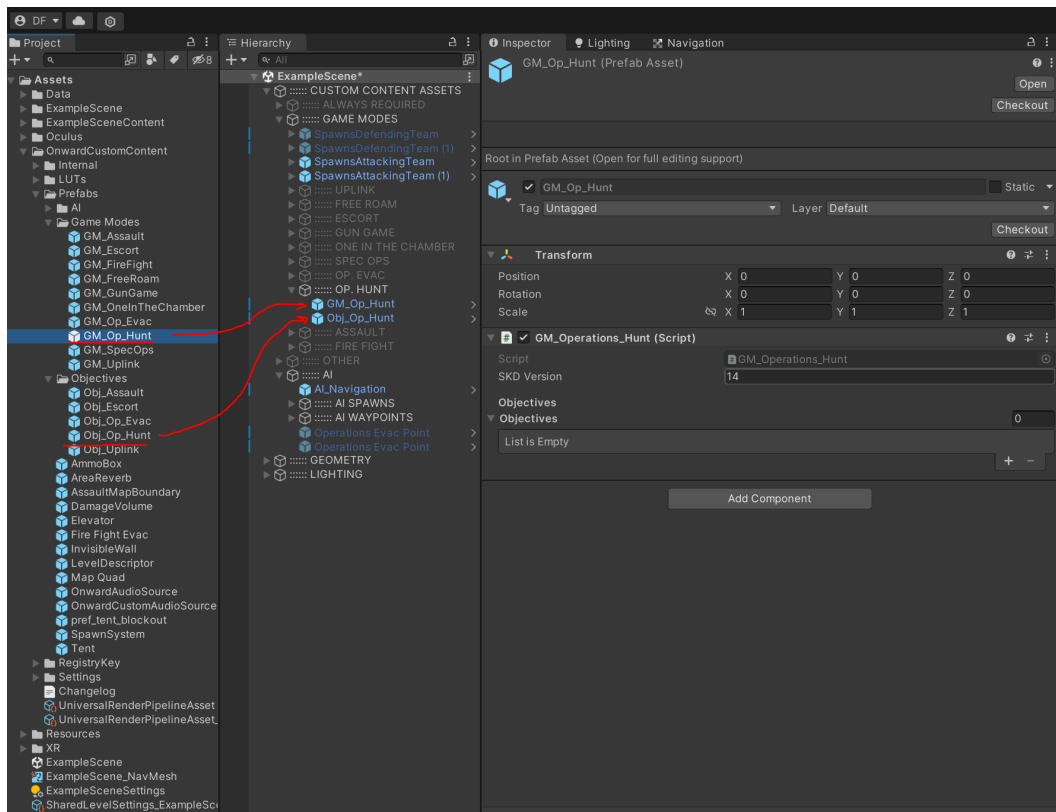
To have a bot use a Waypoint set you just need to place the spawn point near the start point of the waypoint, within 5 meters should work. Note, the spawn points are randomly chosen so there may not always be a bot on a patrol path.

Bots not placed near a Waypoint set will use some default idle behaviors. They will mostly just wander around their spawn area. More behaviors will come in future updates.

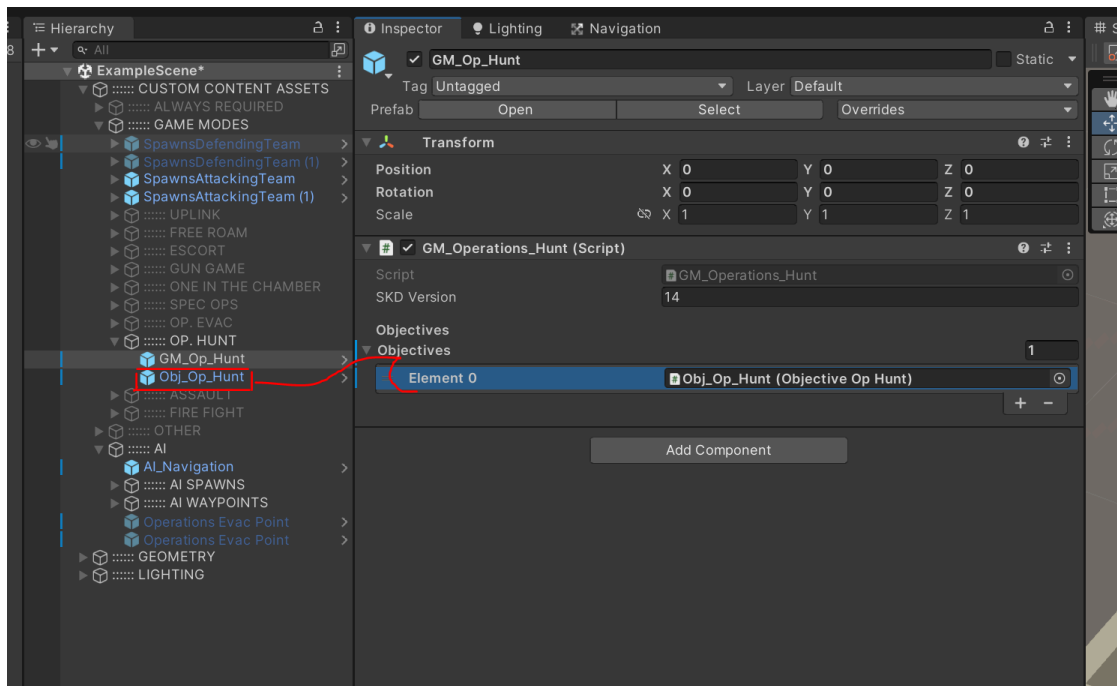


Hunt

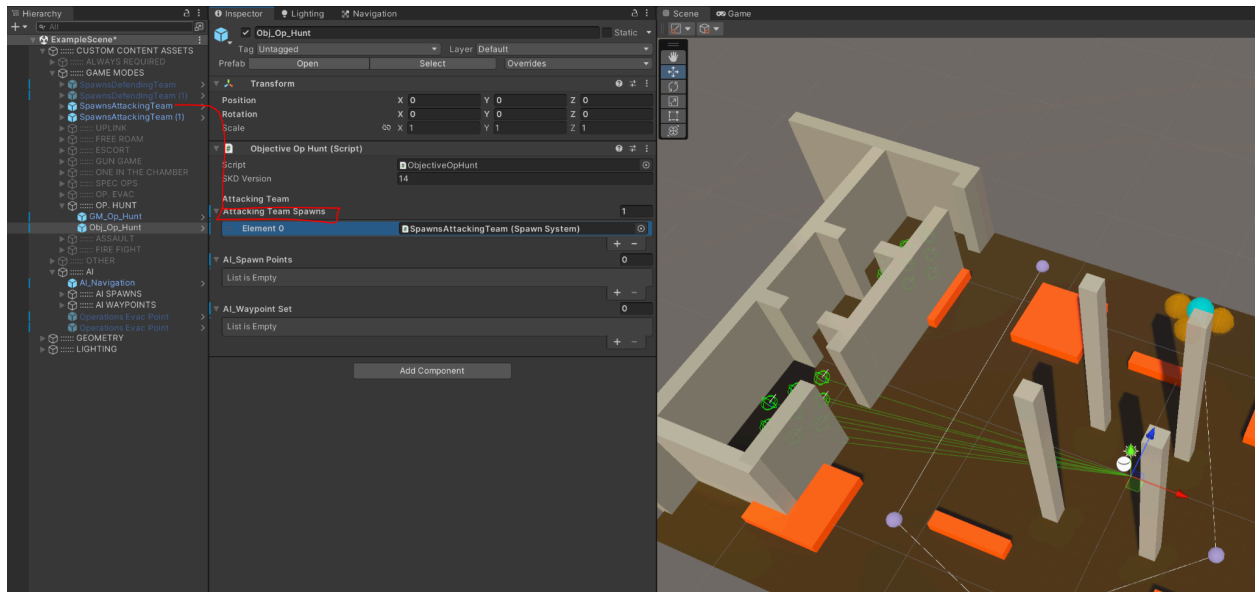
3. Add a GM_Op_Hunt prefab and a Obj_Op_Hunt prefab to the scene.



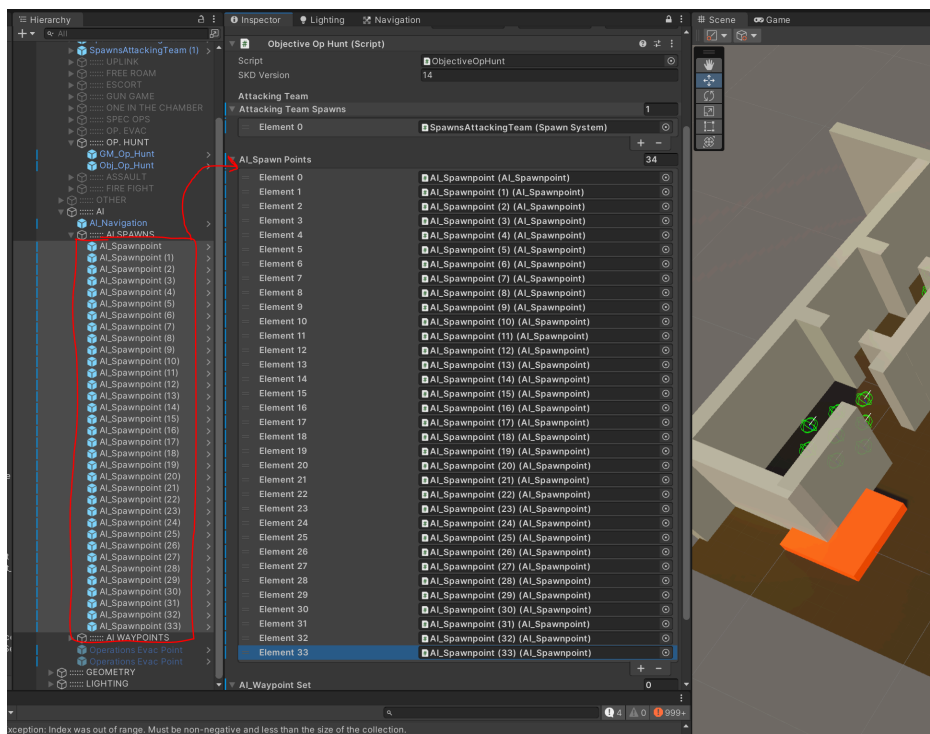
4. Add the Obj_Op_Hunt prefab to the Objectives list on the GM_Op_Hunt object.



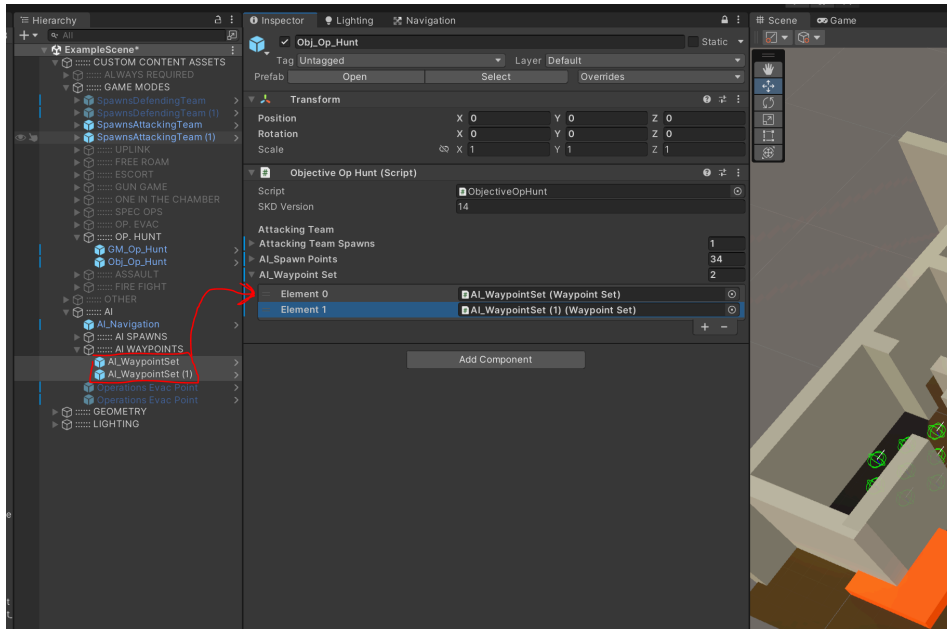
- Assign a spawn set for the Attacking team. This is where the players will spawn.



- Assign at least 32 AI spawn points to the AI_Spawn Points list.



7. Finally, assign the waypoints for the objective.

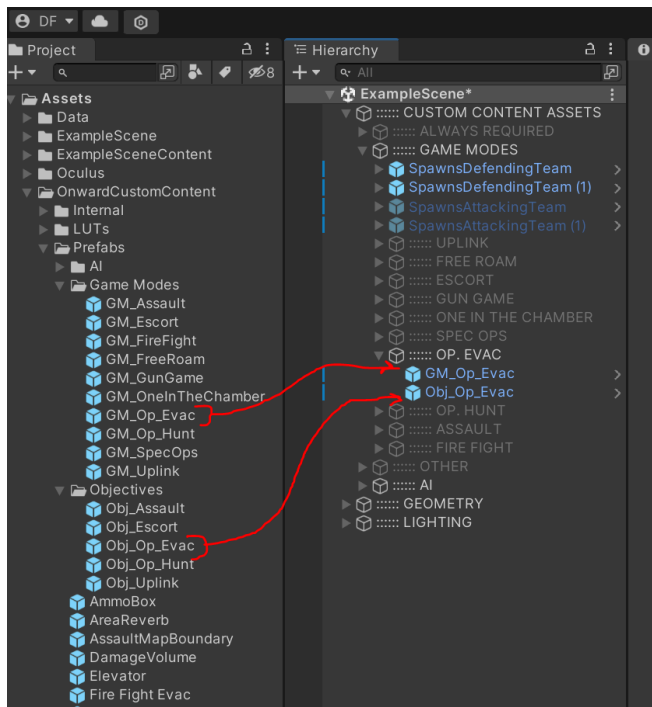


That's it. You now have one Hunt objective set up. You can have as many hunt objectives as you want. Just remember to add them to the GM_Op_Hunt objective list.

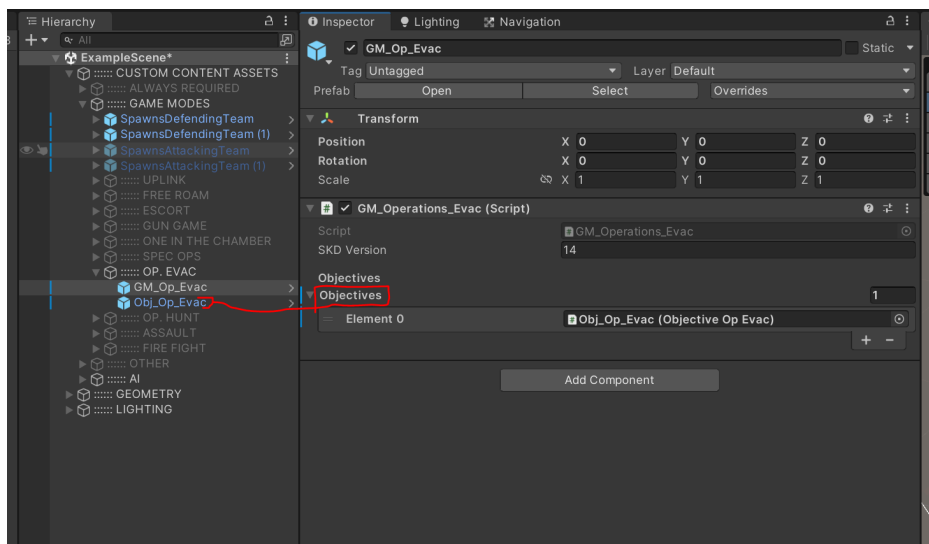
Now that Spawnpoints are not shared across all hunt spawns and you have the ability to place waypoints for each objective, feel free to mix and match AI spawn points and waypoints to come up with some basic scenarios for the players.

Evac

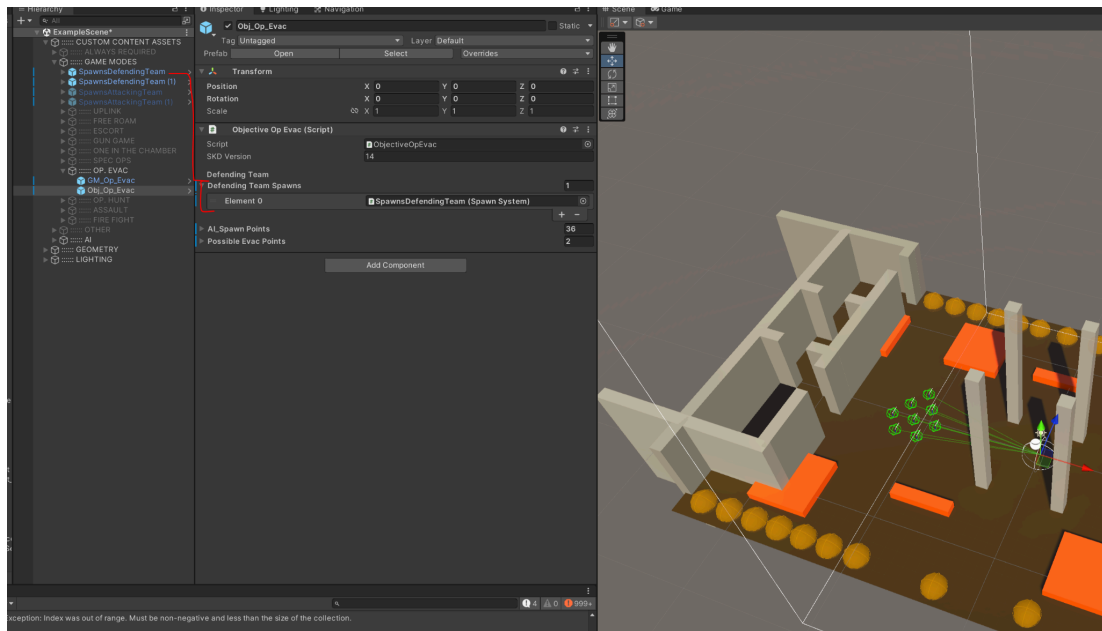
1. Add a GM_Op_Evac prefab and a Obj_Op_Evac prefab into the scene.



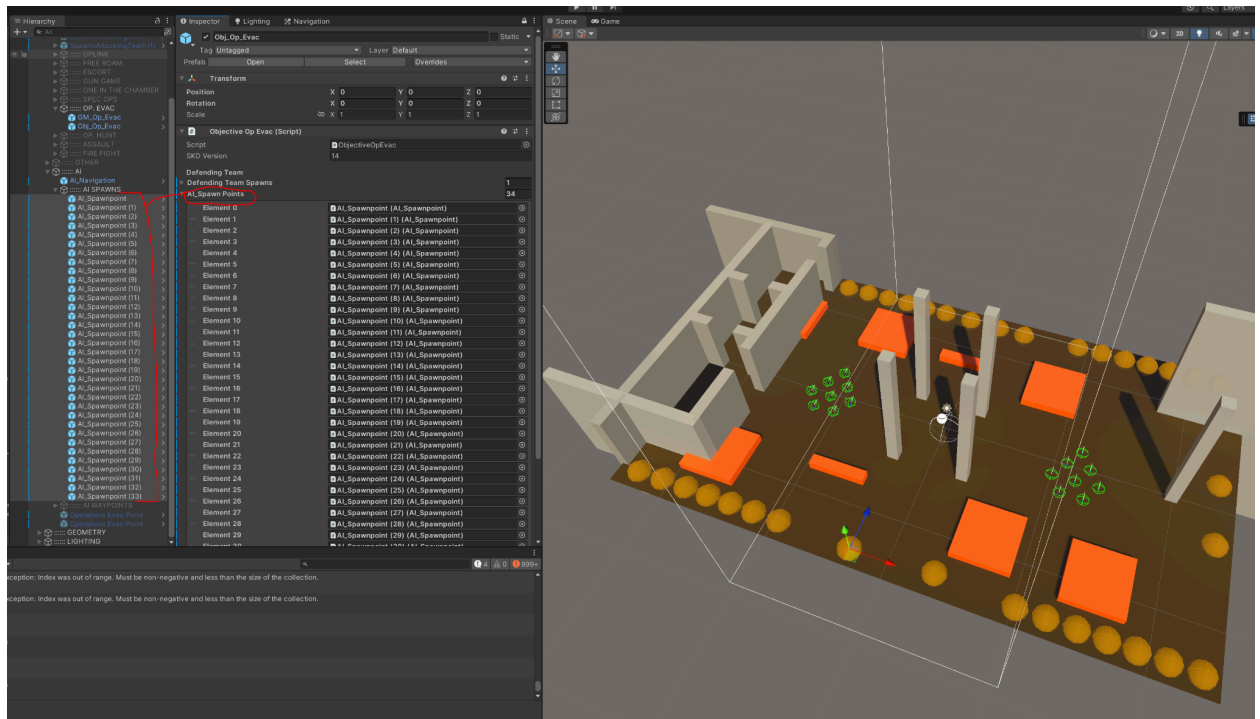
2. Add the Obj_Op_Evac item to the Objectives list in the GM_Op_Evac.



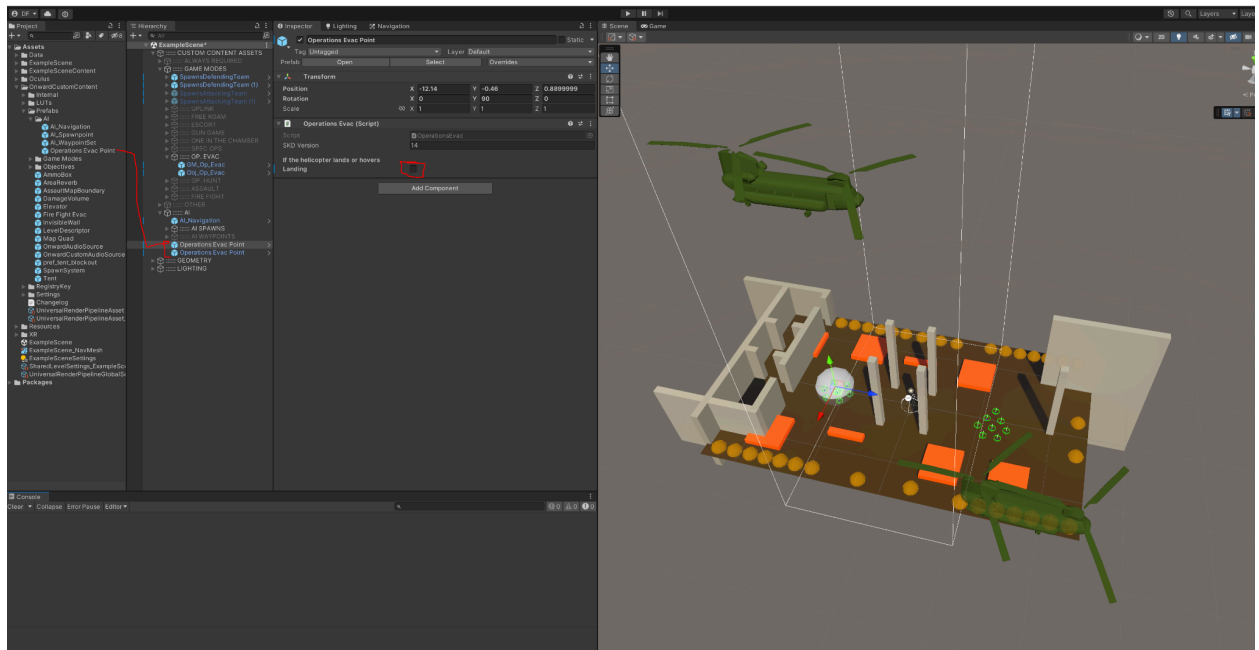
3. Add a Spawn System to the Defending Team Spawns on Obj_Op_Evac. This is where the players will spawn.



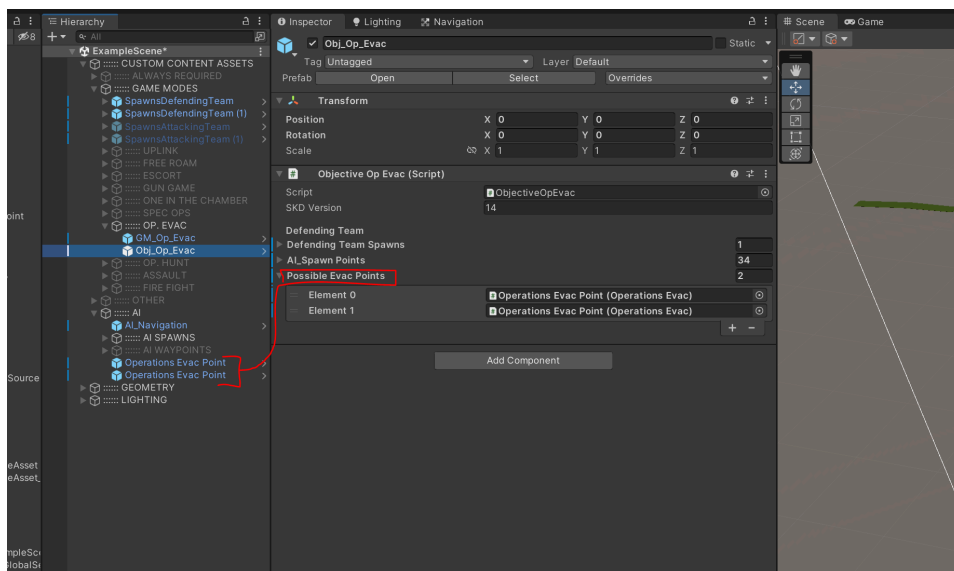
4. Add AI spawn points to the AI_SpawnPoints list. Best practice is to have them spawns on the outer edges of the map since the Bots spawn over time.



5. Add one or more Operations Evac Prefabs to the scene. You can have the helicopters land or hover.



6. Add the Operations Evac Prefabs to the Possible Evac Points list on the Objective. The Evac will be randomly selected so you can add as many as you want. You must have at least one.

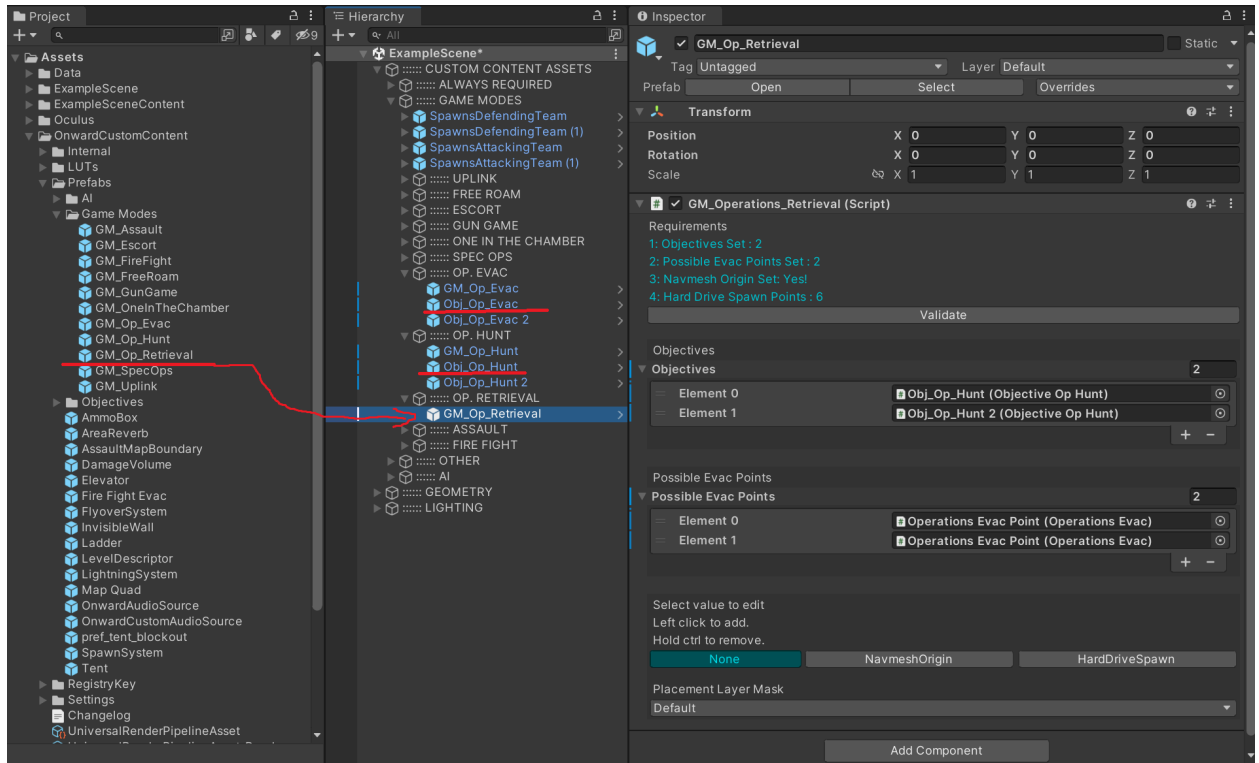


That's it for Evac. Evac doesn't use waypoints, the bots will assault the players position, or defend the helicopter landing point. You can add additional Evac Objectives to provide different player spawns, bot spawns, and evac points. Objectives will be chosen randomly.

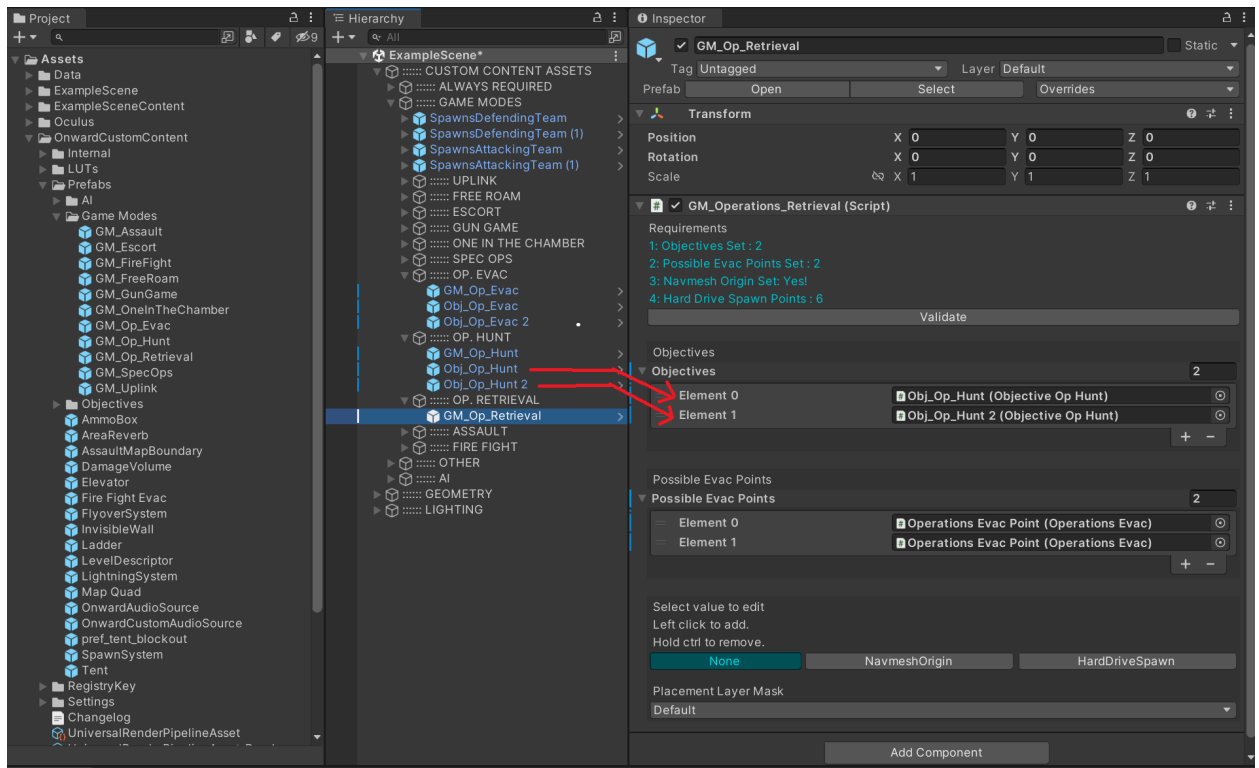
Retrieval

Before we begin, I want to note that any time I reference `Obj_Op_Hunt` or `Obj_Op_Evac`, those can be reused from the two previous sections. Retrieval can utilize the same objectives from Hunt and Evac. So in those sections you'll either need to reuse one that already exists in the scene, or you will need to refer to the previous two sections to set those objectives up.

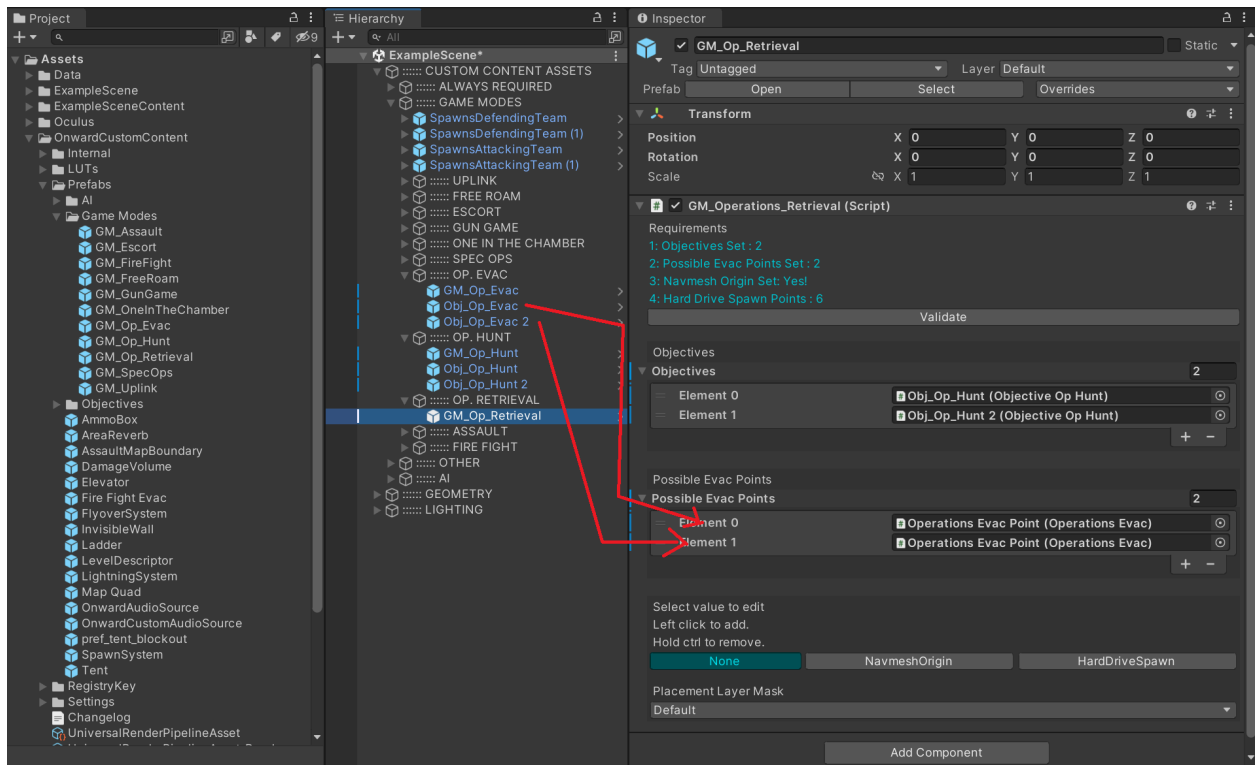
1. Add a `GM_Op_Retrieval` prefab and at least one `Obj_Op_Hunt` and `Obj_Op_Evac` prefab to the scene. Remember these can be reused, or set up according to the previous sections.



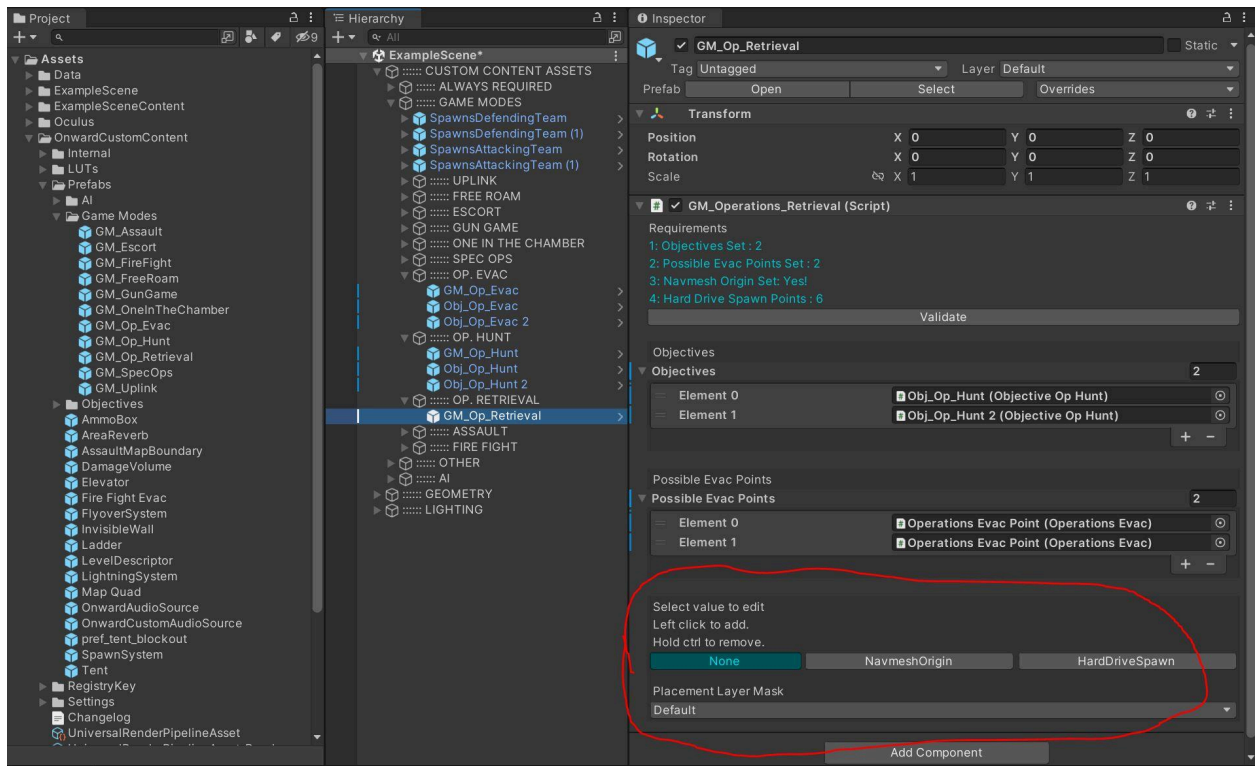
2. Add the `Obj_Op_Hunt` prefab(s) to the Objectives list on the `GM_Op_Retrieval` object.



3. Add the `Obj_Op_Evac` prefab(s) to the Possible Evac Points list on the `GM_Op_Retrieval` object.



4. The last few things you need to do are in the bottom section of GM_Op_Retrieval here, you need to set up a NavMeshOrigin, set up hard drive spawns, and validate. This process is exactly the same as the last three steps in the [Fire Fight](#) section, so just follow those steps.



That's it. You now have Operations Retrieval mode set up. You can add as many hunt objectives and evac points as you like. Note that you don't have to use the same ones you used when setting up Hunt and Evac, you can make new ones too.

Setting up the Social game modes (OITC, Gun Game, Spec Ops)

In Gun Game, players must fight to be the first to achieve a kill with a list of weapons, receiving a new one each time they score a kill.

In One in the Chamber, players must fight to be the last one standing, their only weapon being a bolt-action rifle that contains a single round. You can find additional rounds throughout the game mode, but if you fail to kill on the first shot, you'll need to resort to melee tactics.

In Spec Ops, a MARSOC squad with flashlight-equipped pistols and limited ammo tries to survive a team of Volk assassins armed with combat knives and night vision until the MARSOC can escape after a set time period. A dense fog covers the battlefield, limiting MARSOC visibility. The Volk assassins have ditched all unnecessary equipment, greatly increasing their mobility and stealth. Ammo Boxes are scattered throughout the map- enticing each side with the tools to survive, or assassinate. The MARSOC

will find extra magazines for their sidearms and the Volk will have a chance to find a flashbang that can give them the edge to close the gap. *Who will survive?*

➤ IMPORTANT

Social Modes are different from other **GM_** Prefabs because they do not require an **Obj_** Prefab to link into the **GM_** Prefab. They work just by adding a **GM_OneInTheChamber**, **GM_GunGame**, or **GM_SpecOps** Prefab into the scene and associating **SpawnSystem** Prefabs with them.

Gun Game & One In the Chamber

Both of these game modes use a single **SpawnSystem** Prefab that is associated into the respective **GM_GunGame** and **GM_OneInTheChamber** Prefabs. You can use the same **SpawnSystem** for both Game Modes, or individualize them.

Initially, the **SpawnSystem** will have eight spawn locations arranged next to each-other when you place this prefab. For these two *Free For All* modes, you will be moving each individual spawn to a unique location. If you desire more than eight spawn points, you can duplicate any of the **SpawnPoint** children under the **SpawnSystem** parent and continue to place them in new locations until you are satisfied with the amount of spawns.

Once your **SpawnSystem** is linked to the **GM_GunGame** and **GM_OneInTheChamber** prefabs and you have moved the individual *SpawnPoints* in the **SpawnSystem** to unique locations (and possibly added additional ones through duplication), your setup is complete!

Spec Ops

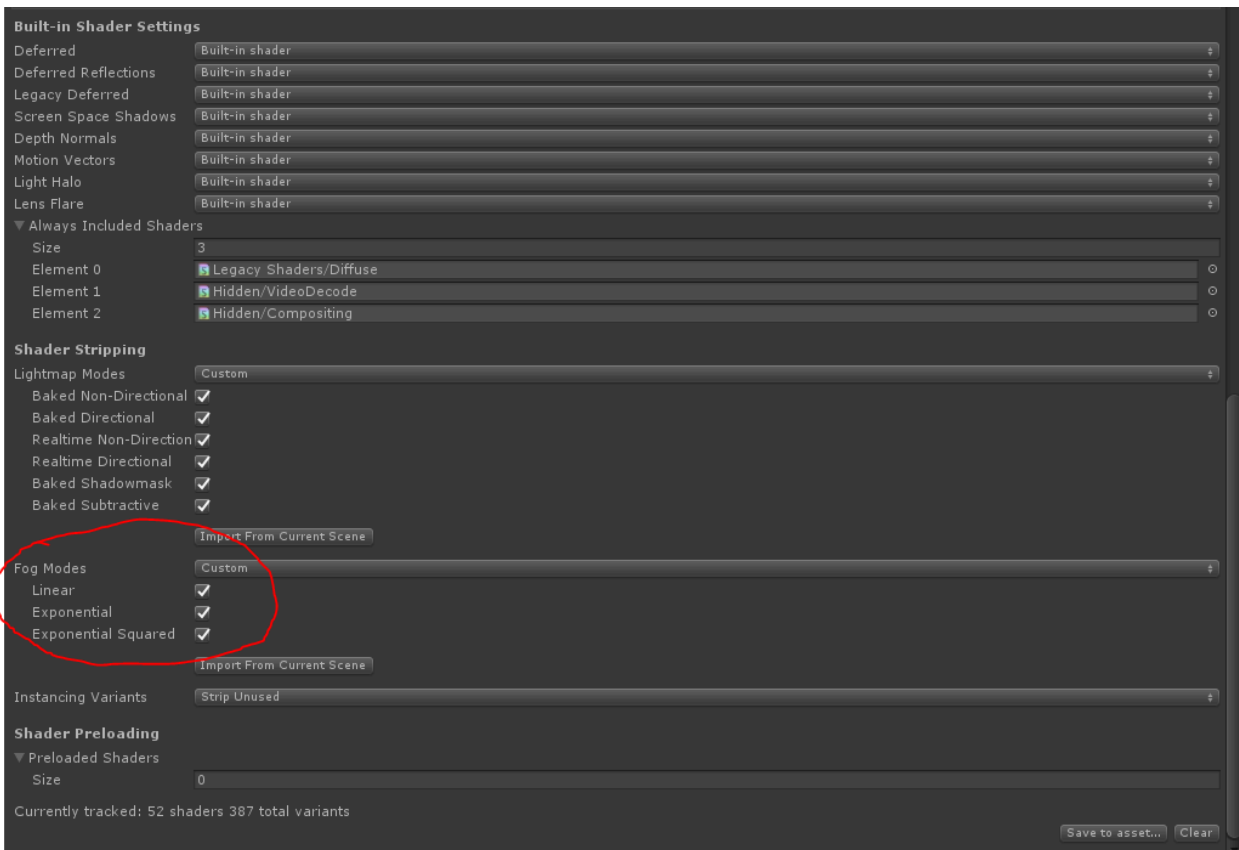
This mode makes use of **SpawnSystems** for attackers (Volk) and defenders (MARSOC) as the regular non-FFA game modes do.

Place **Spawn System** prefabs for both Attackers and Defenders in the fields of the **GM_SpecOps** prefab. Once you've moved your spawns into unique locations, your setup is complete. Note the section below on fog settings if you have not already verified your settings before.

➤ IMPORTANT

If your map supports Spec Ops please double-check that your **Shader Stripping / Fog modes are set to Custom and all checkboxes are checked**. These checkboxes can be found by going to **Edit > Project**

Settings > Graphics tab scroll down to “**Shader Stripping**” section. Ensure all checkboxes are checked and both fields for Lightmap Modes and Fog Modes are set to ‘Custom’



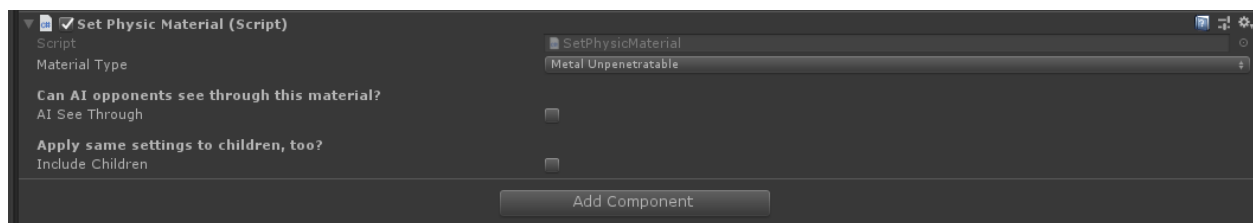
Additional gameplay components

There are additional scripts, prefabs and settings available in the Onward Custom Content bundle that will allow you to further upgrade the quality and gameplay of your maps. These components and their use are covered in this section

Note: You can find scripts by searching for them by name in the “**Add Component**” dropdown of the inspector

Set Physic Material

Adding the **Set Physic Material script** to any GameObject with a **collider** will allow you to choose a **physical material** to add to that collider. This will decide what type of **impact effect and penetration** a given surface has.



There are a few different physical materials available but the general overview is as follows:

- *Concrete* - **Impenetrable**
Glass - **Penetrable**, reduces damage slightly
- *Glass Unpenetrable* - **Impenetrable**
GlassVfx - **Penetrable**, does not reduce damage
- *Metal* - **Penetrable**, reduces damage
Metal Unpenetrable - **Impenetrable**
Wood - **Penetrable**, reduces damage
Wood Unpenetrable - **Impenetrable**
- *Dirt* - **Impenetrable**
- *Grass* - **Impenetrable**
- *Cloth* - **Penetrable**, reduces damage slightly
- *Cloth Unpenetrable* - **Impenetrable**
- *ClothVfx* - **Penetrable**, does not reduce damage
- *Water* - **Penetrable**, reduces damage
- *Snow* - **Penetrable**, reduces damage
- *SnowUnpenetrable* - **Impenetrable**
- *Ice* - **Penetrable**, reduces damage
- *IceUnpenetrable* - **Impenetrable**

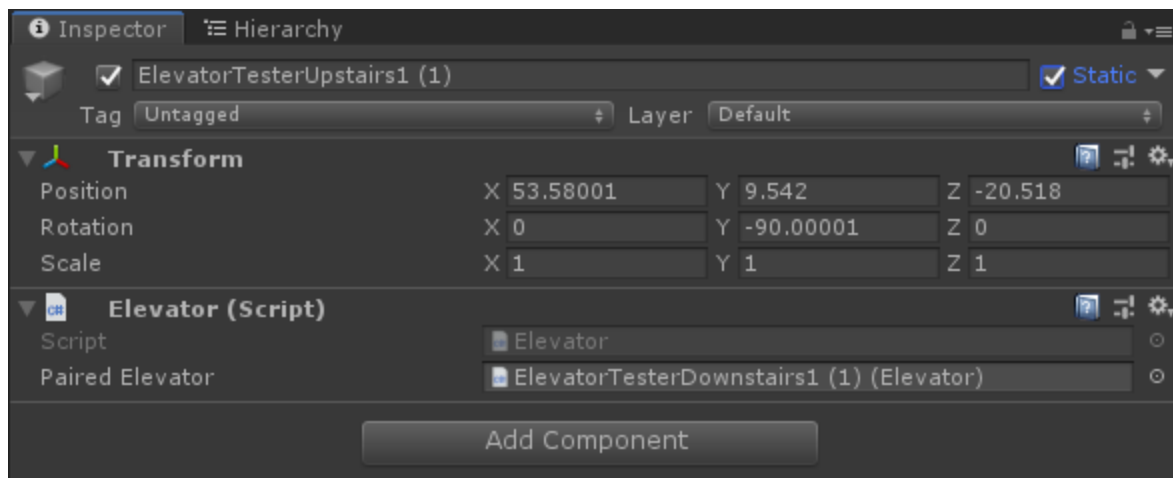
There is a checkbox called '**AI See Through**' for allowing AI to see through a material as well on a given object using Set Physic Material, use this for Glass or other see-through objects such as chain-link fences.

➤ IMPORTANT

If you have no SetPhysicMaterial, the materials of all objects with colliders will default to Concrete - Impenetrable.

Elevators

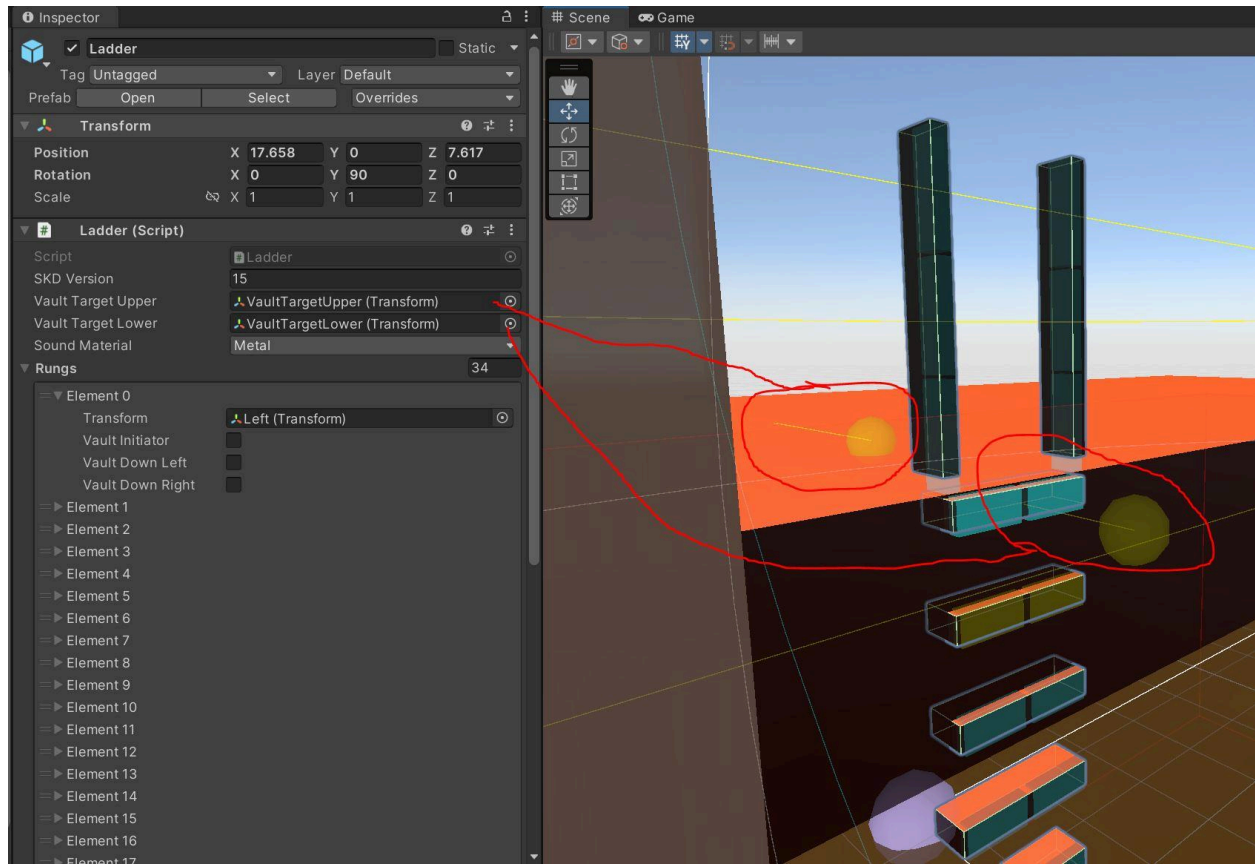
The **Elevator Prefab** is a functional elevator that players can use to send themselves or equipment/items to a paired elevator. Two Elevator Prefabs will be able to be linked together by dropping each respective Elevator Prefab instance into the field of the linked elevator instance. (Elevator A reference goes in Elevator B's field, and vice-versa.) The screenshot below shows a proper elevator set-up. Elevators will start named as 'Elevator' but you should rename them to suit your organizational needs.



Note: *ElevatorTesterUpstairs1* has the connection to *ElevatorTesterDownstairs1* (its destination elevator), *ElevatorTesterDownstairs1* has a setup that is a mirrored version (reverse) of this.

Ladders

The **Ladder Prefab** is a functional ladder that players can climb up and down. You will need to create a list of rungs and set up various references in order for the ladder to function. We will go over those below.



The first thing you probably want to do is set up your vault targets. When a player reaches the top of a ladder and pulls up, we teleport them to `VaultTargetUpper`. If they are at the top of a ladder and pull down, we teleport them to `VaultTargetLower`. These are transforms, represented by yellow spheres. You want `VaultTargetUpper` to be on the platform above the ladder, facing forward. This is where the player will stand after vaulting up. You want `VaultTargetLower` to be facing the ladder and aligned with the top rung. When the player vaults down they will be placed here with their hand attached to these top rungs.

Next you want to set the material of your ladder, this is only used for sounds that play as the player climbs. Your ladder can be metal or wood.

Last, and most importantly, we have to set up the list of rungs. There can be as many or as little rungs as you like, and you are quite free with how you want to place them. For example, placing them horizontally would create “monkey bars”. We generally recommend that for each visible ladder rung you split it into two rungs as shown in the image above, so the player can put two hands on the bar.

Rungs are simply transforms representing boxes. The position and rotation of the transform are the position and rotation of the box, and the scale of the transform is the size of the box. In the ladder prefab rungs are organized into vault rungs and non-vault rungs but this isn't totally necessary, it's just for organizational purposes. Creating a rung is as follows:

- Add a rung element to the rungs list.

- Create a new game object / transform, assign this to “Transform” in the new rung element.
- Position, rotate, and scale this transform as desired, this orients the box shown in the ladder’s gizmo for the rung.
- Lastly, there are three flags for each rung. Most rungs will not have these checked, but we’ll go over them below:
 - **Vault Initiator:** Check this if you want to be able to pull up to vault up to VaultTargetUpper, or pull down to vault down to VaultTargetLower.
 - **Vault Down Left / Right:** In the image above you’ll notice the second rungs from the top are yellow. This is the Vault Down Left and Right rung. These are the rungs we attach your hands to if you vault down, using either your left or right hand respectively. We use the second rung from the top, because the top rung is a vault initiator (which is why it is a bit brighter), so that may make things weird.

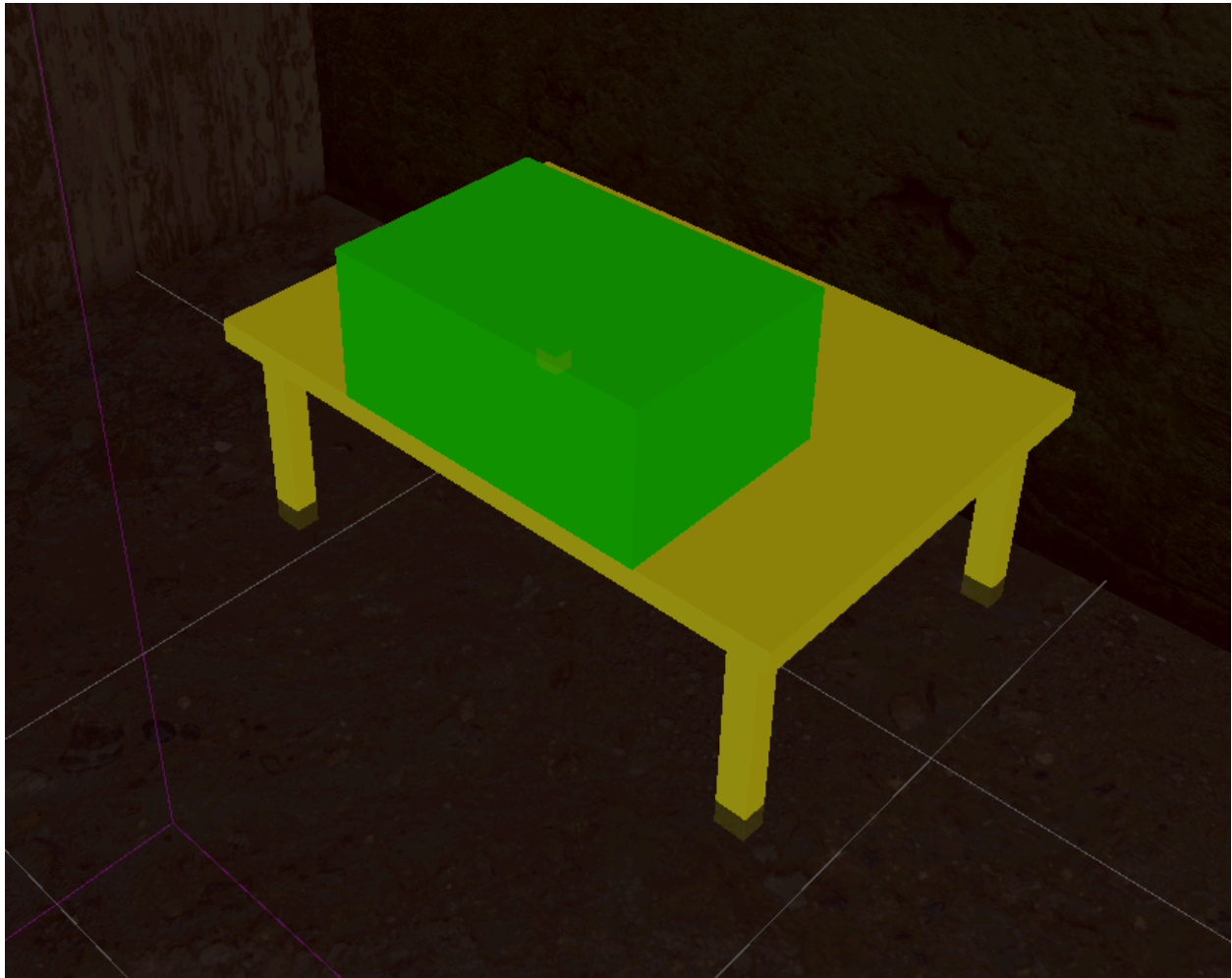
➤ IMPORTANT

The direction between the VaultTargetUpper and VaultTargetLower is important because when you vault, this direction is used to define which way is up and which way is down.

Note: You will also notice all of our ladders have “extensions” on the top, vertical ones. These are our vault down initiator rungs. We found through playtesting that extending them higher like this feels better for players, so they don’t have to crouch down to climb down the ladder.

Ammo Boxes

The ‘**Ammobox**’ Prefab is a cache of ammunition and other supplies that can be found across the map in these game modes: **Hunt, Evac, One in the Chamber, Spec Ops**. Its contents will change depending on the game mode that utilizes them.



➤ **IMPORTANT**

You are required to place at least four (4) Ammobox throughout your map if your map has a Hunt/Evac, One in the Chamber or SpecOps game mode. You may add more AmmoBoxes as desired.

The ammo boxes contain the following items in these specific game modes:

- **Hunt/Evac**
The Ammo box spawns equipment such as magazines for weapons that a player has equipped, or grenades and syringes to support them in battle.
- **One in the Chamber**
The Ammo box spawns a magazine containing one round for a bolt-action rifle.
- **Spec Ops**
The Ammo box contains different tools for MARSOC and Volk. They are as follows: Pistol magazines for MARSOC. Combat knives and a rare chance to spawn a single flashbang for VOLK.

PostProcessing Profile

Onward uses post processing to further boost the quality of the image delivered to a player during gameplay and stylistically distinguish each map. There are several different profiles available to choose from for day and night maps of varying locales. Experiment with the different choices for your use-case and determine which you like the best for your map. Alternatively, you can have no PostProcessing.

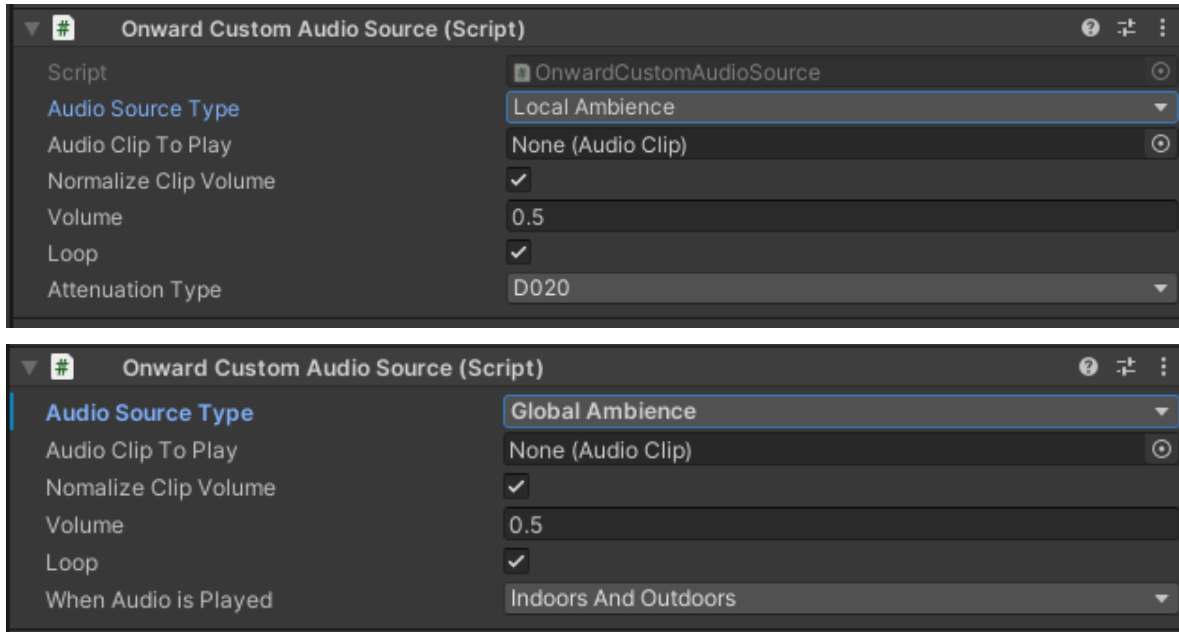
The screenshot shows the 'PostProcessing Profile' settings in the Onward map editor. The 'Title' field is empty. Below it is an 'Outlet' button. The 'Description' field contains the text: 'Radio chatter indicates this shuttered mall in rural America has been occupied by a Volk cell. MARSOC has just entered it's dilapidated halls on a search and destroy mission.' Below the description are four settings, each with a dropdown menu and a reset icon: 'Map enviroment:' set to 'Urban', 'Map size:' set to 'Medium', 'Time of day:' set to 'Day', and 'Post processing profile:' set to 'Urban Day 2'. The 'Post processing profile:' dropdown is circled in red.

Note: you will only be able to see a PostProcessing setting change when you load into your map.

OnwardCustomAudioSource

OnwardCustomAudioSource is a prefab that allows you to play your own audio files in the game. It is recommended to not use too many as there are potential memory usage issues since the audio files are fully uncompressed during runtime, and the cpu usage will be higher than the game's built in audio.

- **Audio Source Type** refers to how the audio is heard. Local Ambience will play from a 3D spatialized position. Global Ambience will play throughout the entire scene as if you are in the center of it.
- **Audio Clip to Play** is your own audio file you wish to play. Any audio type that is supported by Unity is allowed (.ogg, .wav, .mp3, .aiff/.aif, .mod, .it, .s3m, .xm). It is recommended to use .ogg since the compressed format will reduce the download size of the map.
- **Normalize Clip Volume** normalizes the volume of the audio clip. This will essentially make the 0 to 1 volume setting consistent for all audio files.
- **Volume** controls how loud the audio provided is.
- **Loop** continues to loop the audio clip after it reaches the end.
- **Attenuation Type (Local)** is the falloff distance for the audio source in meters.
- **When Audio is Played (Global)** determines if the audio source ducks when inside or outside

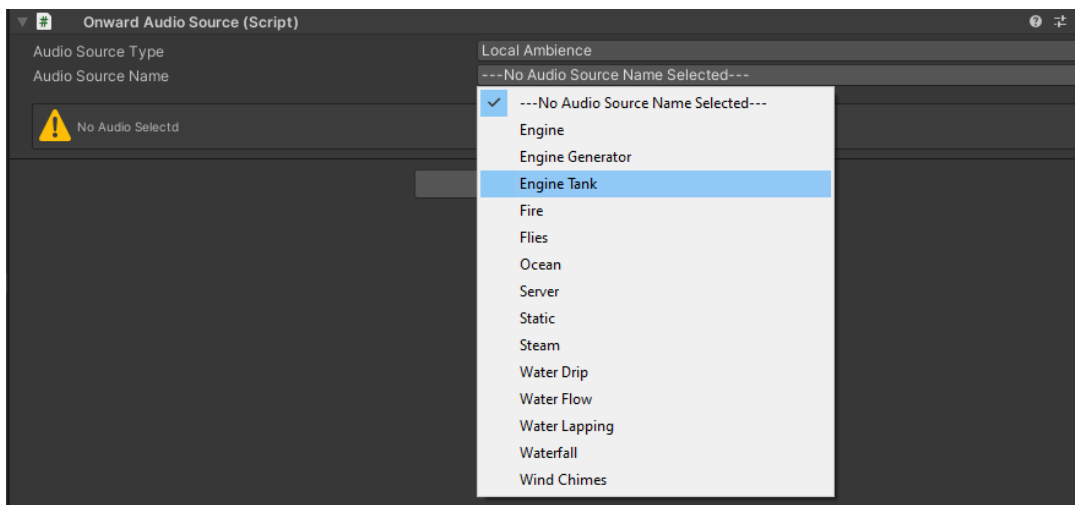


OnwardAudioSource

OnwardAudioSource is a prefab that allows you to select audio used in the game and play it. Some good uses for it are to break up the silence in your level by adding ambience to it.

- **Audio Source Type** refers to how the audio is heard. Local Ambience will play from a 3D spatialized position. Global Ambience will play throughout the entire scene as if you are in the center of it.
- **Audio Source Name** is the type of effect you wish to play.

Use Onward Audio Source to give some character and ambience to your custom scene.

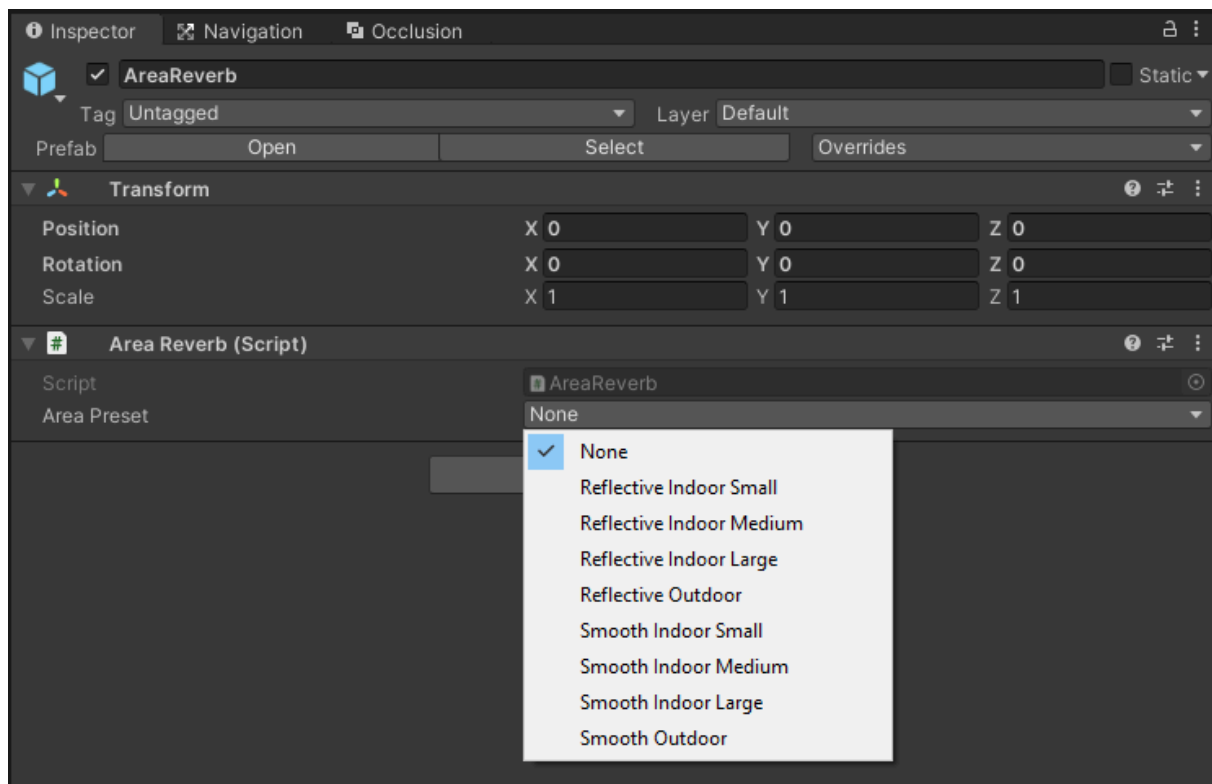


AreaReverb

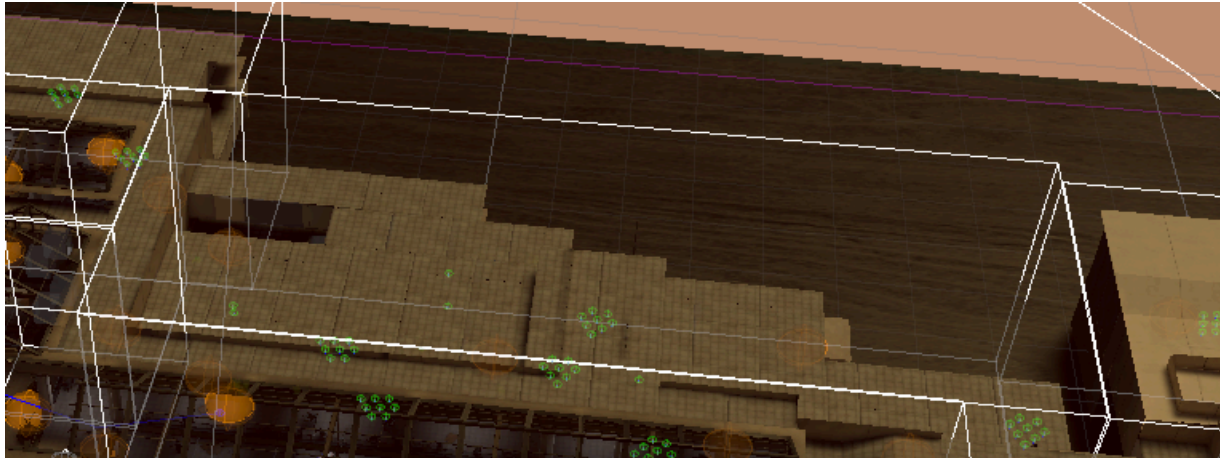
AreaReverb is a prefab that allows you to add different spatial effects to areas allowing for more realistic sound reverberations. Some examples are the Hallway preset for tight hallways, or the Auditorium preset for large open interior areas. Experiment with your local map versions to try different sound reverberations out and see what fits for you.

> IMPORTANT

In order for AreaReverb to function you must go to your LevelDescriptor and check the box in your SharedSettings in the Audio section called “Enable Reverb Injection”. Without this box checked, AreaReverb zones will not actually do anything to the audio in the enclosed area. With the box checked, they will work, but loading your level will be a little slower. We know this is weird, but this needed to be implemented in this way.



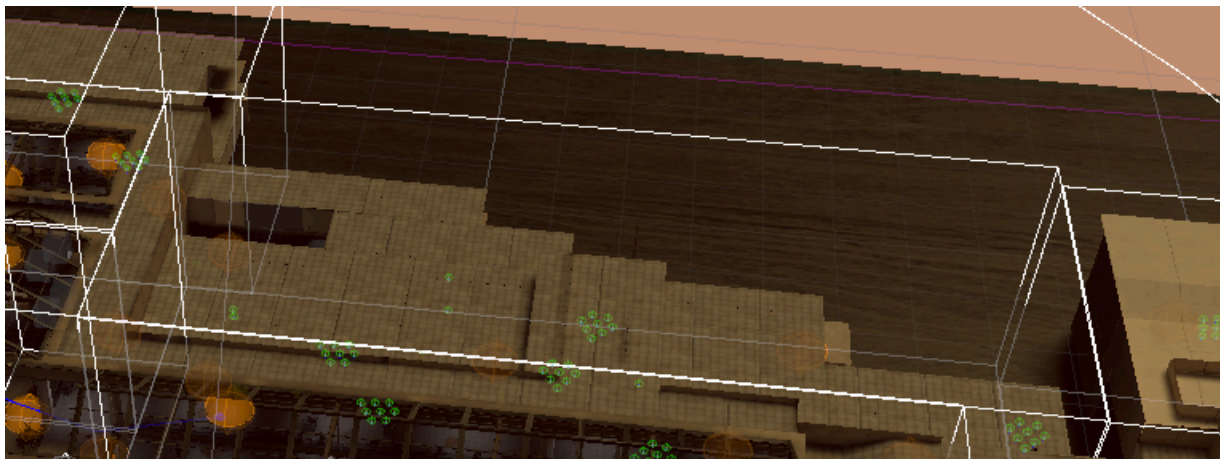
Place the AreaReverb prefab into your scene, then scale it to enclose the area you wish to have reverberations within and set the Area Preset to your desired effect.



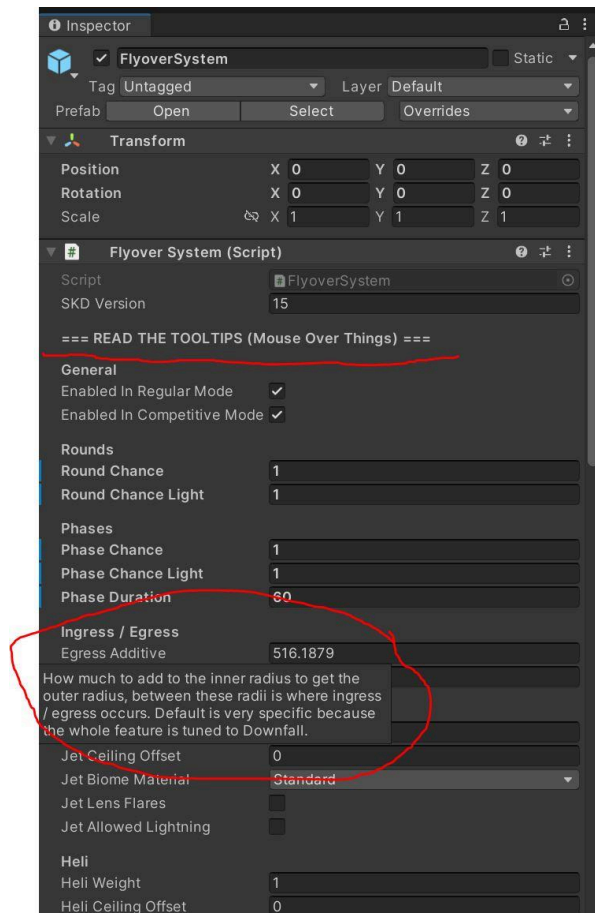
AreaLadderPerch

Unfortunately in Onward, AI cannot climb ladders, leading them to bunch up around the base of the ladder trying to get to the player. AreaLadderPerch is a prefab that allows you to specify the area at the top of the ladder. If you flag that area in this way, the AI will not attempt to approach players at the top of the ladder. Instead, they will prioritize taking cover and backing away, giving them a better chance to have line of sight on the player and attack them.

Placing a ladder perch is exactly the same as placing an AreaReverb, except you use the AreaLadderPerch prefab. Place the AreaLadderPerch prefab into your scene, then scale it to enclose the area you wish to flag as the upper area the ladder leads to.



FlyoverSystem



Flyovers in Onward have default behavior, by dragging the FlyoverSystem prefab into your scene, you will override this behavior. FlyoverSystem has a ton of options, and we are not going to go over all of them here, but each option has a tooltip, so we recommend reading the tooltips.

Instead of going over every one of these options we're going to go over general concepts, and how the tools work.

When a flyover happens, it happens at the top of the MapQuad, where the image is taken for the drawing board. We use the size of the MapQuad to determine an inner radius. The flyover will fly from one point on this circle to another in a straight line.

We expand this inner radius to get an outer radius, forming a donut shape. Between the inner and outer radius is where flyovers "ingress" and "egress". Since their paths are random, this zone is where they can, for example, pitch up to avoid a mountain as they fly away.

Whether a flyover should divert in this way and how it should do so is something defined by "avoidance arcs". Imagine the inner and outer radii as a pizza, an avoidance arc would define one slice of that pizza and say "there's a mountain here, pitch up".

Towards the bottom of the options you will find "path overrides". This lets you specify a number of **specific paths** for the flyovers to choose instead of generating random ones. This is used by default for boats, since they are not flying and need to avoid things like the shoreline. But you could also use it for flying vehicles if you want.

At the bottom you will see your tools, there are checkboxes here to show various gizmos. You want to set the "Tool Flyover Type" to whatever you are working with, Jets, Helicopters, or Boats, before you start working. This is because these different vehicles may have different flight levels based on the options you set above.

Show Ingress Egress Radii: does exactly what it says. The inner radius will be cyan, the outer radius will be yellow. This is useful to have in conjunction with the level cutter while you set up avoidance arcs.

Show Level Cutter: displays a black plane at the same level as your flight level across the whole level. This is useful because anywhere geometry pokes through this plane between the inner and outer radius is probably where you need to set up an avoidance arc.

Show Path Overrides: This will just show the paths that are set up in path overrides. It's useful when working with boats for example.

Show Avoidance Gauge: Checking this will show a purple tool that helps with setting up avoidance arcs, a lot. Let's say you have a mountain poking up through the level cutter, turn on the avoidance gauge, and make a pizza slice with the gauge using the "gauge start" and "gauge end" values. Now adjust "gauge vertical" to avoid the mountain, you will see the pizza slice pitch up.

> IMPORTANT

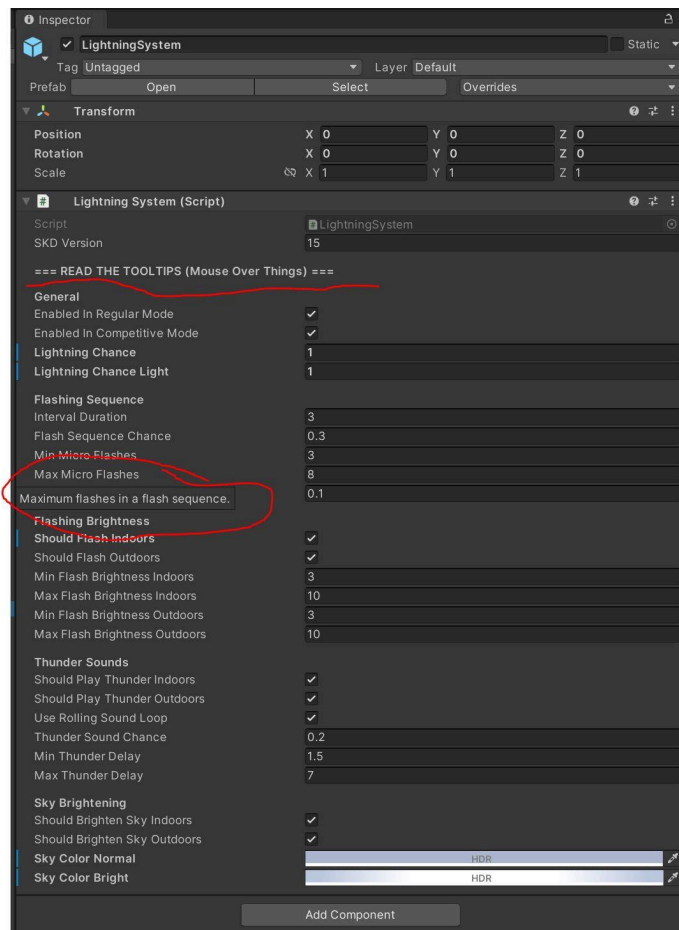
Note the values from your avoidance gauge, and we'll go over how to plug them into an avoidance arc now. Go back up to the avoidance section and add an arc.



Take the avoidance gauge start and avoidance gauge end values and plug them into start angle and end angle respectively. Now an avoidance arc applies to all three vehicles, so you need to find the one you were adjusting here, and plug the gauge vertical value into the offset's X axis there. Yes the X axis is vertical, don't ask me. The other axes are also available here in case you want to yaw to avoid a building or something. If you check "forbidden" for a particular vehicle, that means flyovers that generate for this pizza slice, for that vehicle, will be considered invalid and they will not proceed with the flyover at all.

Note On Testing: Flyovers are random, and that makes them hard to test. If you want to test them more easily, set your phase chance to 1, and your phase duration to something low like 30 seconds. Then you can hop into a non-round based mode like Free Roam, and you'll have a guaranteed flyover every 30 seconds. You can tone down the settings after you get all your avoidance arcs and path overrides and stuff set up.

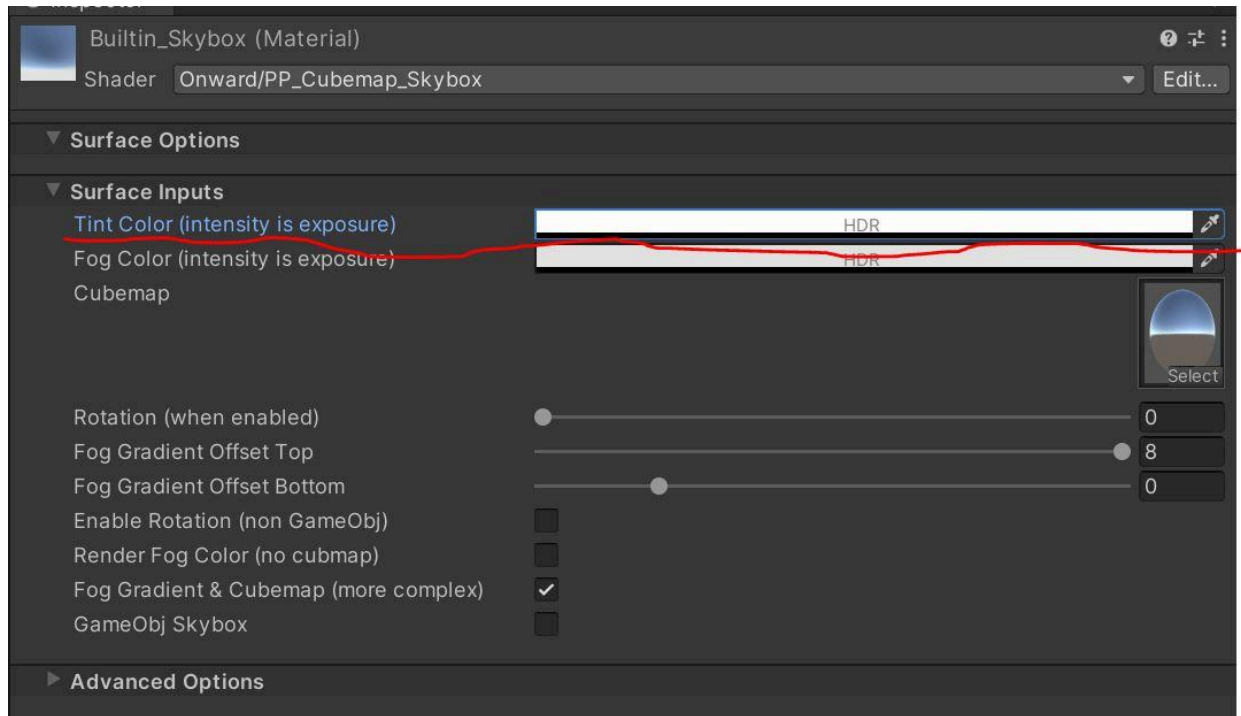
LightningSystem



This is going to read very similarly to the FlyoverSystem section, but lightning storms are simpler to work with.

Lightning Storms in Onward have default behavior, by dragging the LightningSystem prefab into your scene, you will override this behavior. LightningSystem has a ton of options, and we are not going to go over all of them here, but each option has a tooltip, so we recommend reading the tooltips.

LightningSystem for the most part is self explanatory, the tooltips explain most things. The only thing I would say you absolutely need to modify is those two sky colors at the end. Everything else is more or less optional. The Sky Color Normal can be copied directly from your scene's skybox material here:



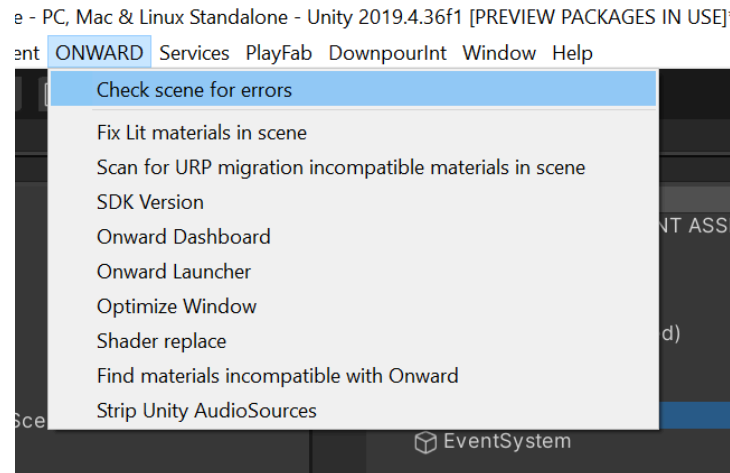
Sky Color Bright should be the same color with some additional intensity, that's the color that is used when the lighting flashes to tint the sky.

All of the other settings can be tweaked to taste, and let you adjust how often the lighting flashes, how the sounds play, and so forth. We recommend only using lightning on night maps. It uses the same effect as digital night vision to flash the level geometry, so on day maps it may look weird. But it's up to you!

Note On Testing: Lightning Storms are random, and that makes them hard to test. If you want to test them more easily, set your lightning chance to 1. Then you will have guaranteed lightning storms. Just make sure to tone it down when you're done adjusting how they behave. (Or don't, it's up to you!)

Check For Errors

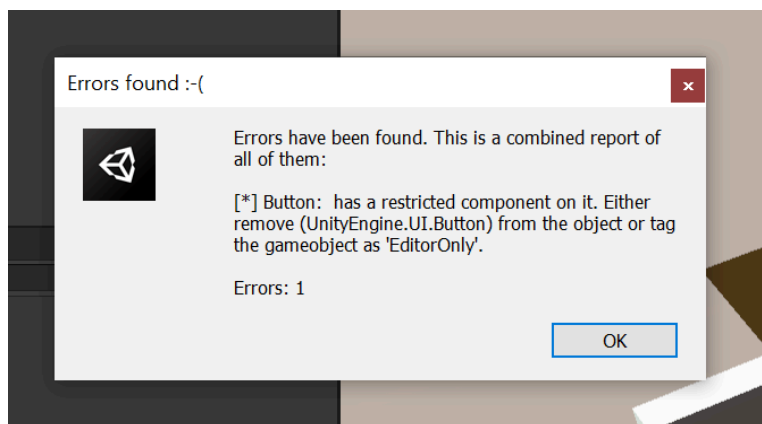
Click **ONWARD > Check scene for errors**. If there are any errors they will be listed in a dialogue box.



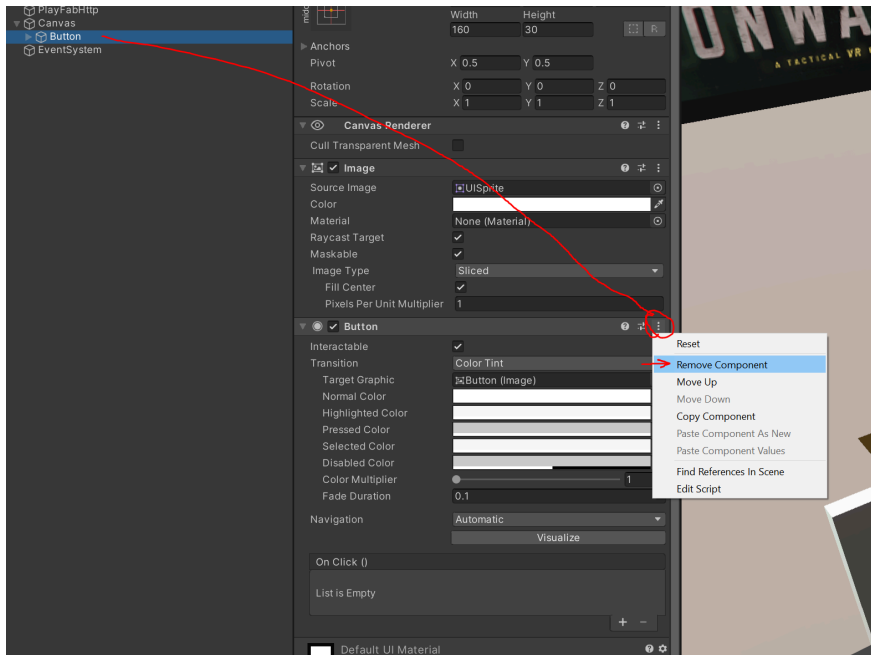
Restricted Components

Restricted components are things that may not be compatible with Onward. If you see an error message mentioning a restricted component there are a couple ways to fix it.

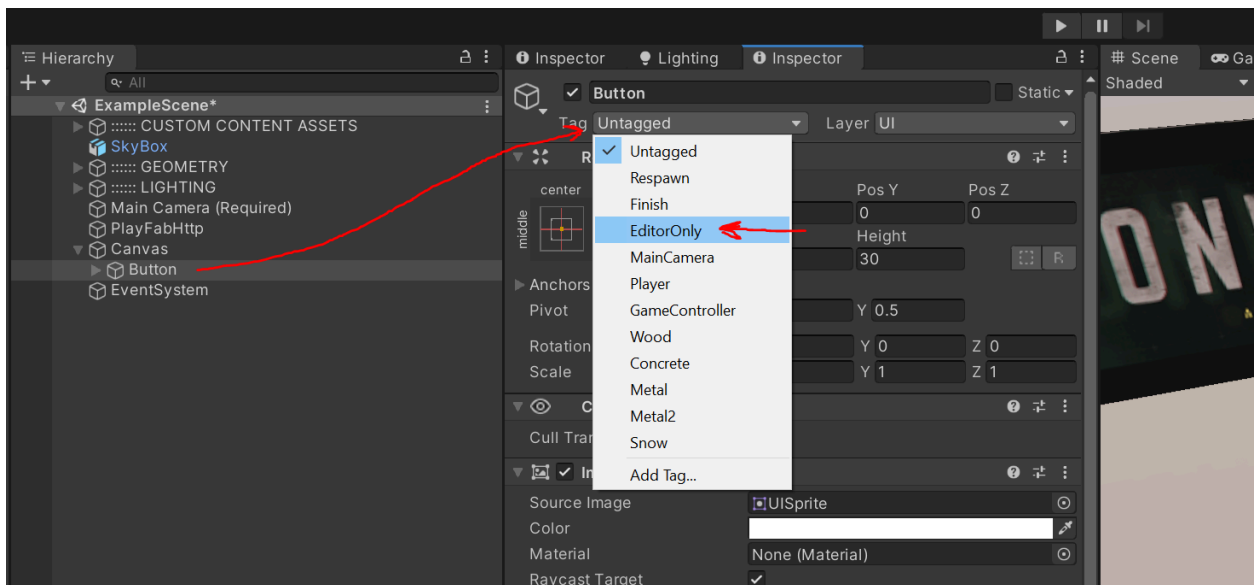
In this example you can see that the Button is a restricted component.



The first way to fix it would be to simply go to the game object holding the component and remove it.



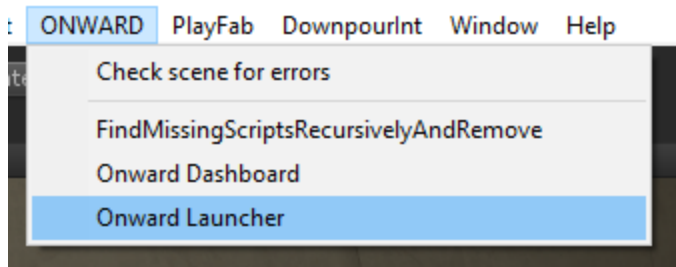
If it is something required at edit time, a terrain tool you are using, or an automated light probe placement script for example. You can tag the object as “Editor Only” This removes that object from the build but lets it stay in the scene file.



Testing your maps

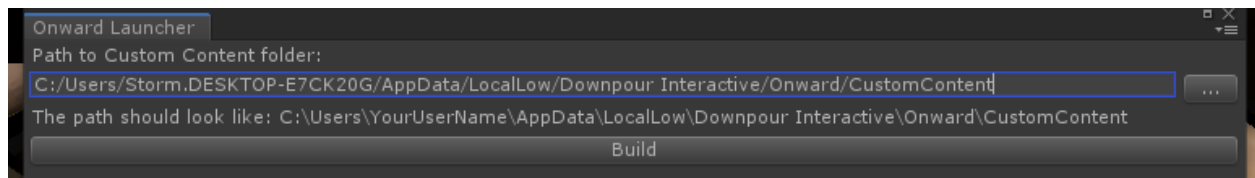
You can test your maps by using a local version and your installed Onward game client to test without uploading. For publishing instructions, refer to the **Publishing** section. For local testing, refer to the instructions immediately below.

You will be able to test your maps without uploading them and waiting for approval. You can use the **Onward** dropdown for this. Select **“Onward Launcher”** and a new window will appear:

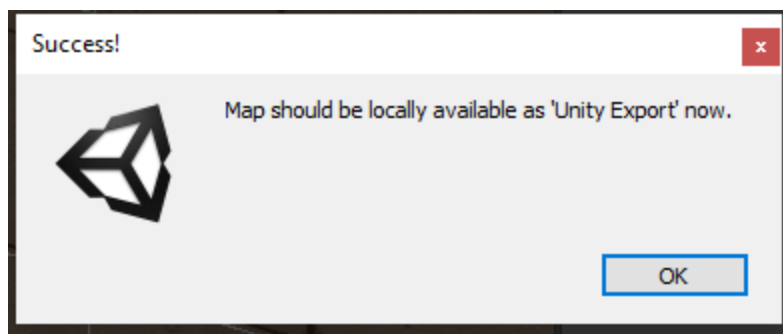


PC Testing

Direct this window to the prompted file path (as shown in the screenshot) and press **“Build”**



A progress bar will appear indicating the packaging of your map. Once it is complete, you will receive a success message.



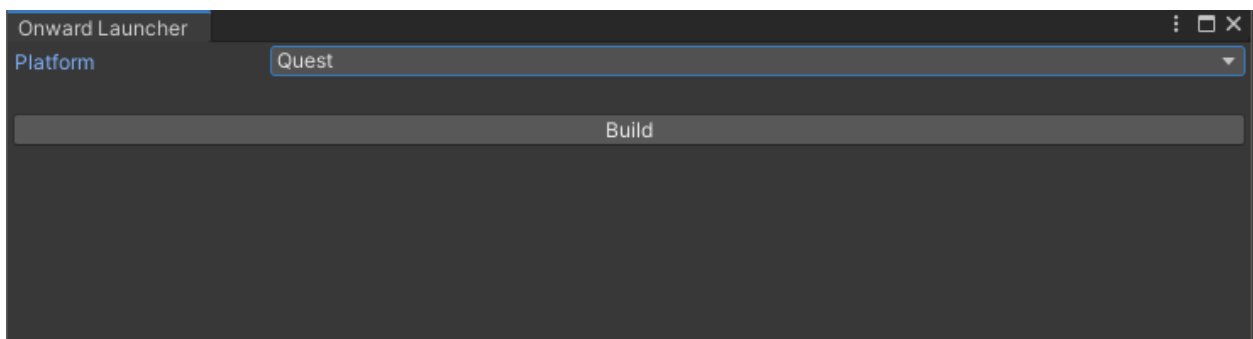
After getting your success message, you can launch a locally-installed version of Onward and browse for your Custom Map using the server creator. It will be labeled as **“Unity Export”**.



Locally-hosted Custom Content in the map selection menu.

Quest testing

Testing for Quest - Set your Platform to Quest and press “Build”. You will be prompted where to save the files. Select a memorable folder location.

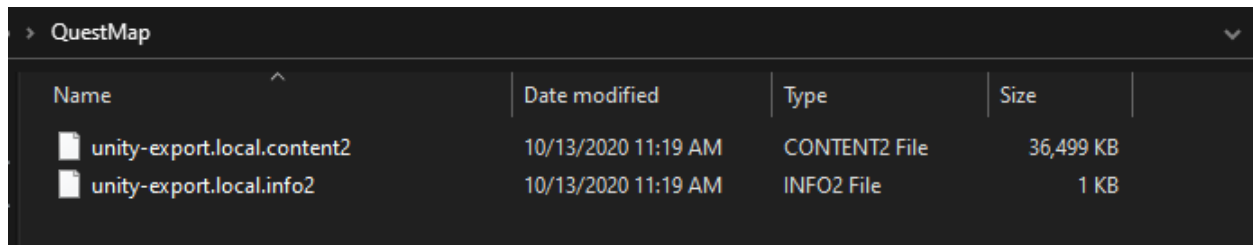


Upon Success you will see a message denoting where it is to be installed.

➤ IMPORTANT

To test on Quest you **MUST manually** add your map locally on the device and you must have Developer Mode enabled on your Quest device.

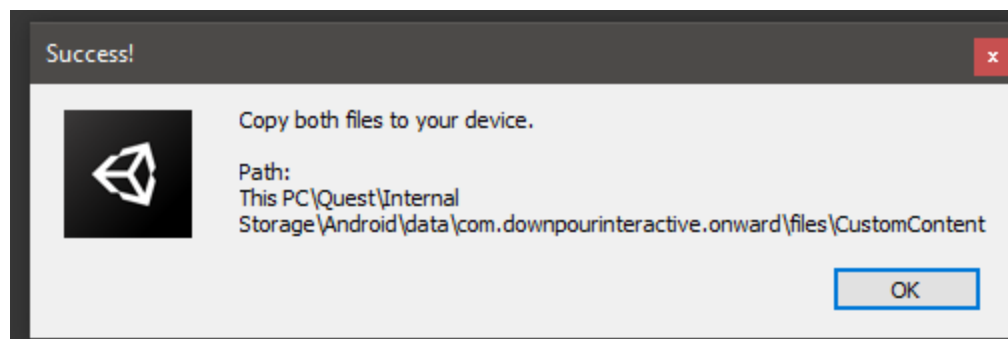
Plug the device into your PC with developer mode active, then browse the file structure and add the files created by the Onward Launcher Builder you saved to a memorable file location earlier to the directory below.



Name	Date modified	Type	Size
unity-export.local.content2	10/13/2020 11:19 AM	CONTENT2 File	36,499 KB
unity-export.local.info2	10/13/2020 11:19 AM	INFO2 File	1 KB

Add these files to

yourdirectory\Quest\InternalStorage\Android\data\com.downpour interactive.onward\files\CustomContent

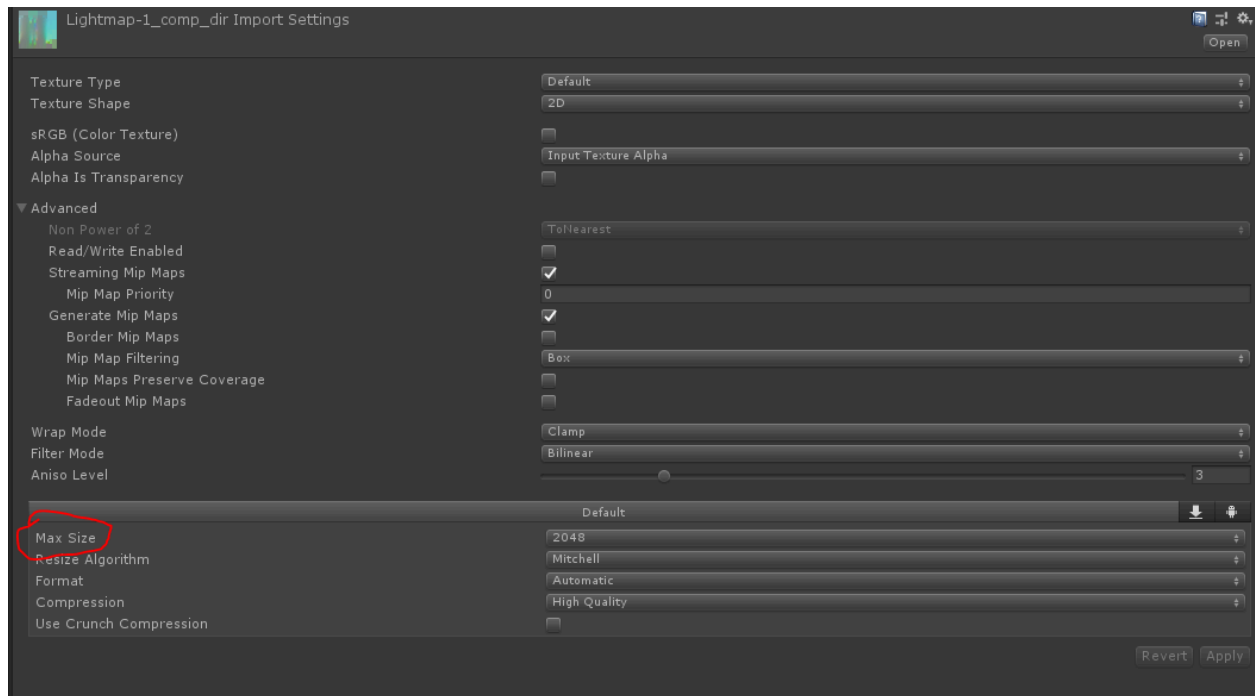


From there, you can launch the map locally on your Quest device when the transfer is complete.

Performance Considerations

PC performance considerations

- Try to not exceed 750,000 vertices and 700 draw calls - the lower, the better.
- Make use of Occlusion Culling, Lightmapping and other performance tools in Unity.
- Resources for these tools and more can be found at Learn | Unity (<https://unity.com/learn>), or on YouTube
- If your submitted map suffers from major performance issues it may be denied.
- You should investigate your textures and scale them to a lower resolution if possible. The default resolution of many textures is in some cases egregious for the level of detail required and will only result in a bloated level size and slow loading times. You can inspect a given texture by clicking on it and choosing its max size. Reduce this from 4k to 2k then “Apply” and observe your texture in the scene. If the quality decrease was acceptable leave it, or even lower it further until it's not acceptable- then raise it up one level from that.



Quest performance considerations

➤ IMPORTANT

Quest scenes must use extremely limited draw calls, on-screen triangle counts and texture memory to perform optimally. Creating a custom map for Quest is intended for advanced map creators. Your submission may be denied if it does not meet performance requirements and causes simulation sickness.

Rough guidelines (the best thing to do is test!):

- ~150 draw calls
- ~250,000 triangles
- ~1GB of texture usage

These limitations account for players, objectives and other gameplay objects for you- so you can develop right to the edge of these specs if you wish.

Quick tips for developing Custom Content for Quest crossplay:

- Usage of occlusion culling is recommended for Quest scenes.
- Utilize clever line of sight to avoid large overview areas that will break your frame limits.
- Share the same shaders as often as possible to let the SRP batcher combine draw calls for you.
- Use smaller resolution texture arrays vs high resolution atlases when possible. This avoids the tiling issues caused by atlasing while increasing the texture look up speeds on the pixel shader.
- Use the frame debugger to check out your draw calls and triangle counts at many given points throughout the scene by placing a camera in your scene and removing it before submission once you are satisfied with your results.

There are many tricks and techniques to make a lean, yet playable level and these are just a few of them. Respect the performance of the Quest 2 platform and you will be rewarded with smooth gameplay and many highly-rated reviews of your custom content map.

Best practices in development

Onward is a game with an established set of rules, based on realistic squad-level combat. You should keep this in mind while designing your maps and objectives to ensure an intuitive and fun gameplay experience for all players.

We highly recommend blocking out your maps with simple 3D shapes and playtesting them before finalizing your map design. This will give you a better understanding of how your map feels, and what sections need to be modified before you start polishing it visually.

Regular playtests are a great way to see how your map evolves. They help you locate areas where players might have difficulty moving through the map, lines of sight that may be too short/long, and spots where lighting issues may occur.

Lastly, getting in touch with other map creators gives you the opportunity to share experiences. You can pick up tips and tricks from more experienced creators, or share your own ideas with others looking for help. Constructive feedback is one of your most useful tools.

Publishing

This is what you've been working for - once you've set up your map, built it in Unity and tested it thoroughly, it's time to publish it!

To upload, modify, or submit your custom map you must have an Oculus account and the oculus client open on your PC.

[Install the Oculus app on your PC | Meta Store](#)

Follow the guide to install the Oculus client and setup or login to your Oculus account. You do not need to set up a headset, but you **MUST** own Onward on your Oculus account.

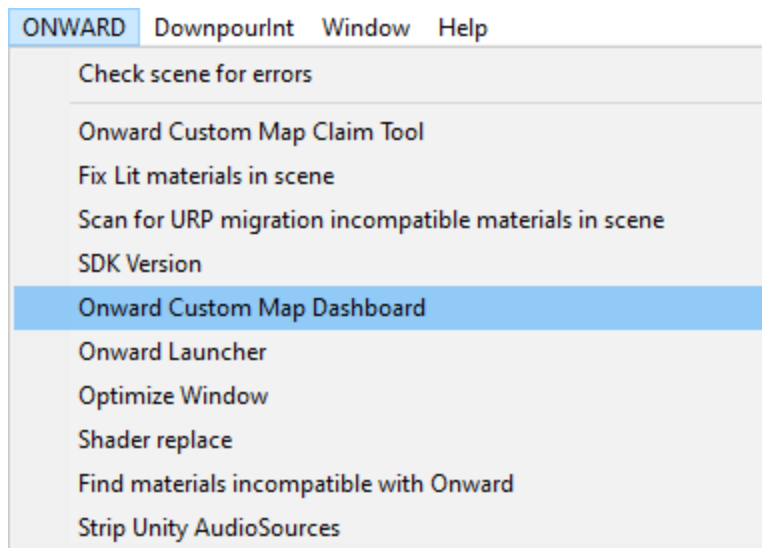
Registry Setup

The Onward UGC package contains a Registry Key Config that you must apply to your PC's Oculus Registry Settings. To do this, follow these steps:

- Find the OnwardUGCRegConfig file at the path: Assets/RegistryKey in your project after importing the UGC SDK
- Simply double click OnwardUGCRegConfig in the project view and you can automatically apply the developer key necessary to work with UGC.

Once the client is installed and your registry is set up, you must restart your PC for Unity to register this change. Once your PC has restarted, open the Oculus client; it is required to be open to work with the custom map dashboard and claim tool.

First, in the Unity top menu, open the **Onward Custom Map Dashboard**.



Take a moment to read and accept the **Terms of Service**.

Make sure you include a clear title and description. We recommend following the suggestions below to ensure that your information displays correctly:

- **Title:** Try not to exceed 24 characters, however 6-18 is ideal
- **Description:** Try not to exceed 500 characters, however 250 is ideal
- **Map Environment:** Change this to match your map
- **Map Size:** Change this to match your map
- **Time of Day:** Change this to match your map.
- **Post Processing Profile:** optional setting, covered in the Additional Gameplay Components section of this document.
- **Thumbnail:** 512px wide by 256px tall.
- **Display Version:** for a finished map, use 1.0

Once you have filled out the information, a **Publish** button will appear. Click it to begin the submission process. Depending on the size of your map, this may take some time. Once it is complete, you will be notified with a **Success** message.

If you receive errors in your console log or any pop-up error message, take time to review the message and determine the issue with your scene, take steps as advised to fix the issue, then try again. The Onward Custom Content bundle will detect errors in your setup automatically and try to help you fix them.

➤ **IMPORTANT**

After your submission is sent, you will need to wait for approval. Approval may take up to 2 weeks. If you are denied, you will be notified. Updates must also be approved. These will be reviewed once per week, same as before. This means that you can submit your updates at any time and they will be reviewed within two weeks of your submission in most cases, be it updates or a new map submission.

Once you've published your submission and it has been approved, you can update it to submit map changes and improvements. The button at the bottom of the **Publish/Update** window will now read **Update**. Click it to submit any revisions. Once your map is available to the public, use local testing to test major changes before deploying them to the live servers. This will help prevent issues for users if you accidentally break some of your map's functionality. Reserve uploading and publishing updates to your map for builds which you are certain function correctly.

Conclusion

Thank you for reading through this guide, and for submitting your custom Onward map(s). We're excited to present them to the community, and we appreciate the time and effort you invested to create new experiences for your fellow players.

If you have any issues submitting your custom content, or if you find any errors in this guide, please let us know via the Onward Discord channel so we can address them. Your feedback will help us provide the best experience for creators and players alike.

Thanks again for your support.

“Onward!”

- Downpour Devs

Appendix A: Updating your project

SDK 9 Upgrade - Onward v1.9

➤ IMPORTANT

THIS SECTION IS ONLY RELEVANT IF YOU ARE UPDATING A MAP CREATED PRIOR TO ONWARD 1.9, IF YOU ARE NOT DOING THAT, SKIP THIS SECTION.

There are a few more steps to update your Custom Map SDK compared to before, because the backend system has changed; but worry not, it is still straightforward. Let's get started!

Upgrading your map consists of 5 steps

1. Delete the PlayfabSDK folder in your project
2. Delete the OnwardCustomContent folder in your project. Note: your scene will appear pink. Ignore this for now.
3. Install the new CustomMapSDK into your project. Your scene should look normal again after it finishes.
4. Install and have the Oculus Client open whenever you plan to modify/submit your map
5. Your registry now needs a little setup, [follow the instructions here](#).

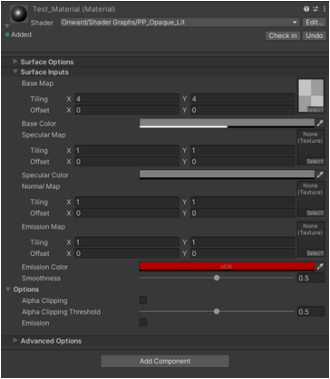
Once done. Restart your PC.

SDK 13 Upgrade - Onward 1.13

➤ IMPORTANT

THIS SECTION IS ONLY RELEVANT IF YOU ARE UPDATING A MAP CREATED PRIOR TO ONWARD 1.13, IF YOU ARE NOT DOING THAT, SKIP THIS SECTION.

- There is now only one sky box shader.
- Material map tiling & offset, the properties are new so any materials that previously used the old single tiling & offset will now use the default value.

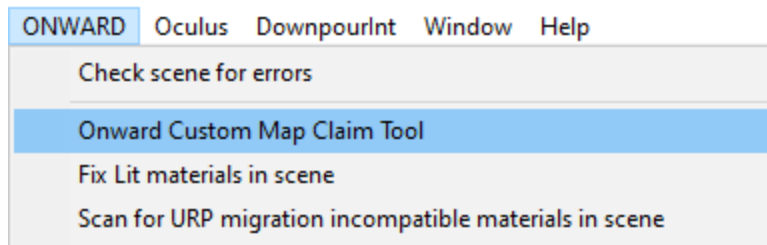


Appendix B: Claiming map ownership (update 1.9 onwards)

Due to the backend change, all maps are now tied to your Oculus ID. This ID has no relation to your Downpour account which has been deprecated. As a one time process, older maps will need to be claimed by their original creators. By doing it this way, current maps can continue to be played while we assign creators to them.

Claiming your map(s) is a simple, one time process! Let's get started. First, make sure to update your custom map SDK to the latest version, as detailed in **Appendix A: Updating your project to work with Onward Workshop**.

Once this is completed, we will need to use the **Onward Custom Map Claim** tool; Open this now.



Once you have the tool open, you will need to fill in some information, including your playfab email address you used prior to 1.9 to submit your custom maps, your Discord username, and last but not least the maps you wish to claim.

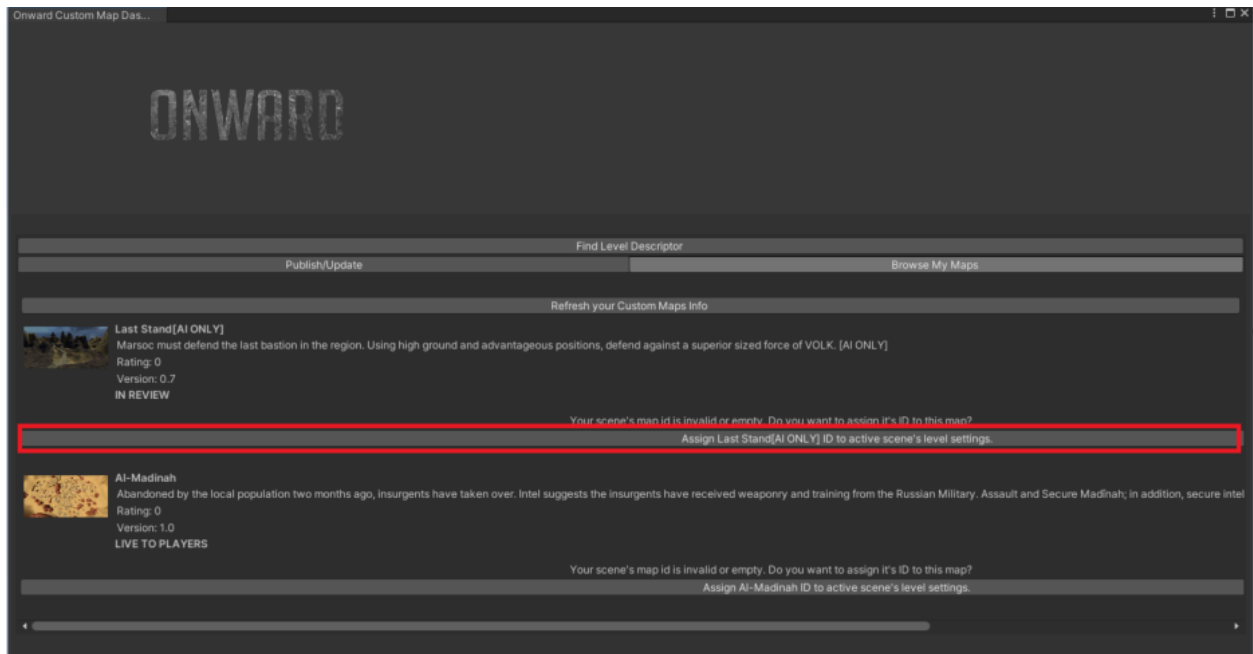
NOTE: Discord account is optional but it is very easy for us to reach out to you with it.

With your claimed maps selected, click **‘Submit your Map Claim’**. You will receive a response from Downpour in a few weeks in Discord or in email once your request has been approved.

When your scene is approved, your map is almost ready! You will need to convert your old Playfab ID to the new ID system. Open your converted scene, then open the **Onward Custom Map Dashboard**. You will see an error at the bottom of the page, this is telling us that the ID is incorrectly set. Let's fix that!

Click **‘Browse My Maps’** at the top right of the dashboard and find your scene. If your scene has not been accepted as claimed yet it will not appear, so be patient until it has been reviewed.

Once it is available, you can finish setting the new ID. Make sure you are on the correct scene before continuing! Click the **Assign** button to the correct map. Once you have done this, your scene will be fully converted to the new system! You will then be able to update your scene whenever you want.



Appendix C: Optimization tools & tricks for Quest

Below are a few tools and features that will help you optimize your maps for Quest if you choose to publish for Quest.

[OVRMetricsTool](#)

This is a handy tool you can sideload on your device that allows you to monitor different performance metrics. Once it is installed, open it from your 'unknown sources' list and a new window will appear with settings for OVRMetrics. You will want to 'enable persistent overlay' and switch it to the 'advanced' view. From there, the main metrics you will want to check out other than FPS are CPU/GPU usage and App T (GPU time per frame in Microseconds).

The most important thing to look at is the overall frame time. Unity will process the game frame on the cpu and then pass it off to the GPU. You may see data like 70% CPU usage and 50% GPU usage and have performance issues. There are a few ways to fix this. These are recommendations so try mixing and matching solutions that fit your scenes requirements.

1. Reducing the draw call count through reducing shader count, mesh combination, or improved culling/LODs will all help on the render thread of the CPU and help the GPU render more efficiently.
2. Reduce texture resolution and use ASTC compression. The CPU must bundle up each draw call and the smaller that bundle is the faster it can send it to the GPU. Small textures and proper compression helps quite a bit. Lower resolution textures also increase the speed of texture lookups so it increases the render speed.
3. Reducing shader complexity will improve the GPU rendering speed. Using the PP shaders provided in the SDK will be the easiest way to achieve this. Vertex shaders are much faster than pixel shaders. If something can operate on a vertex try to do that. Try using look up tables if an intense operation needs to happen often.
4. Transparency is terrible on the tile based renderer used by the Quest. Keep transparency areas small, break it up into multiple meshes, or find an alternative. Sometimes having polygons for leaves instead of using transparency ends up being faster.
5. Use texture arrays when possible. This will require some shader work but it could be worth it.
<https://docs.unity3d.com/Manual/class-Texture2DArray.html>

Onward uses Vulkan and the SRP batcher. The SRP batcher handles a lot of the cpu side optimization for you while Vulkan improves the CPU and GPU rendering efficiency. Find more information here: <https://docs.unity3d.com/Manual/SRPBatcher.html>

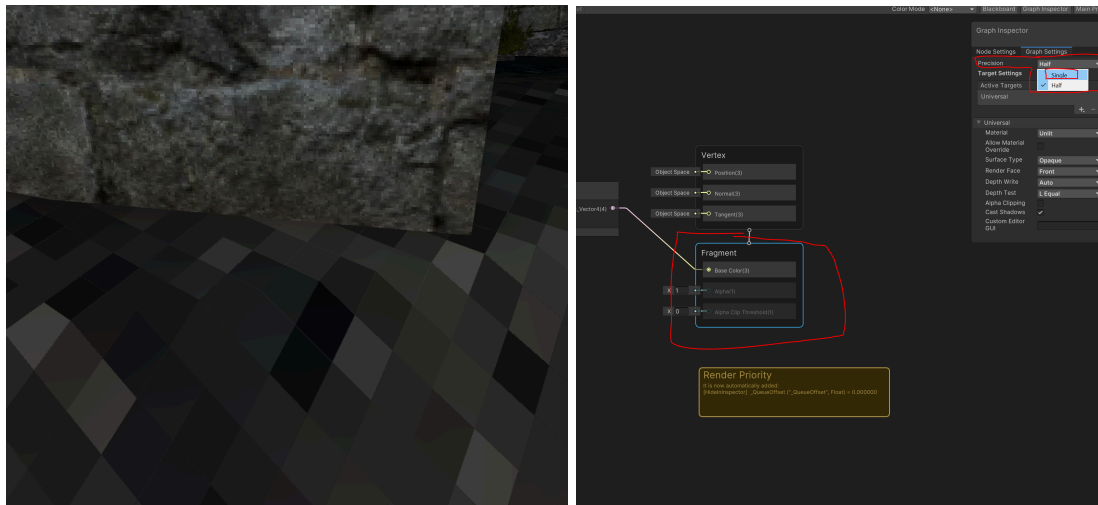
Optimize Window

This is a handy tool we have added to version 2.0 of the custom content package that allows you to see high-triangle count meshes in a list that will help with identifying assets for decimation or further level of detail usage. Please use it as you see fit to help optimize your Quest scenes.

Appendix D: Troubleshooting

Textures are Pixelated on the Quest

Change the shaders precision from Half to Single.



The skybox gets cut off

If you are using the sky box prefab/mesh, swap it out for the built in skybox. Remember to check the “Use Built in Skybox” on the shared level settings object.