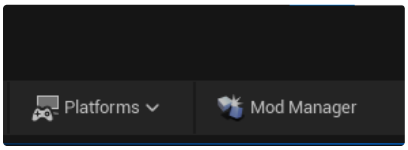
Mod Mission Setup

In this page you can find step by step instructions for how to setup your own mission. We'll start with the broader side of things and go more in depth as we go along.

Set up a Mod

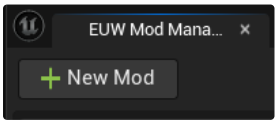
Creating the Mod

At the top of the screen, click on **Mod manager**.



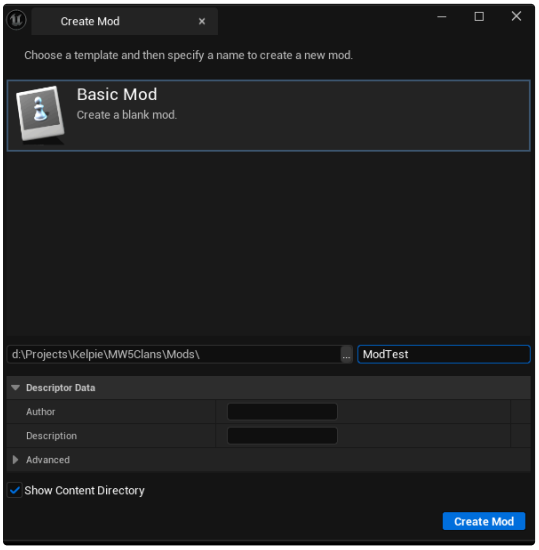
Mod Manager button

Click on **New Mod**.



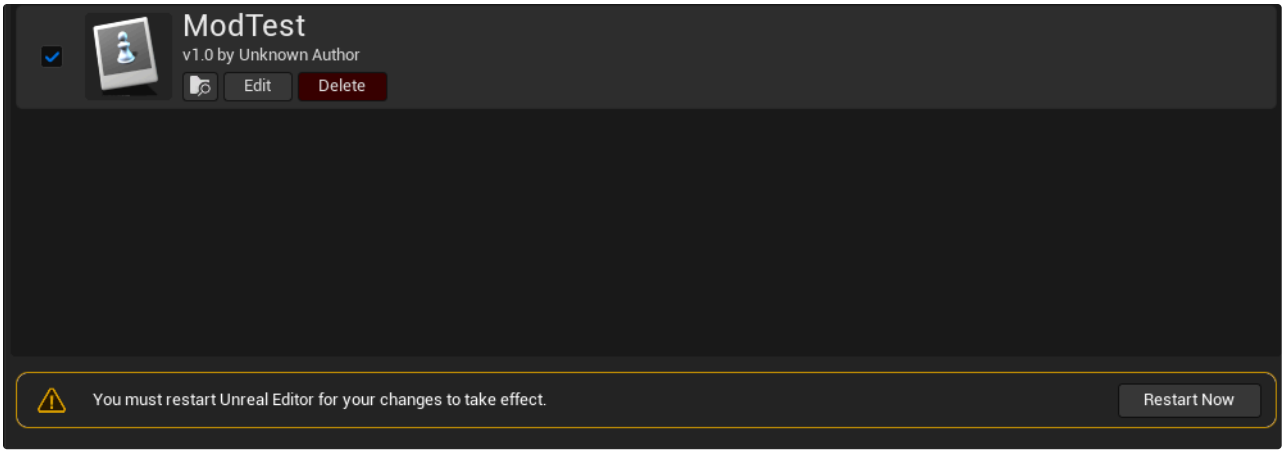
New Mod button

A screen will pop up and it'll ask you where to save the Mission. Keep the folder as is and give it a suitable name. Click on **Basic Mod** and **Create Mod** should become clickable.



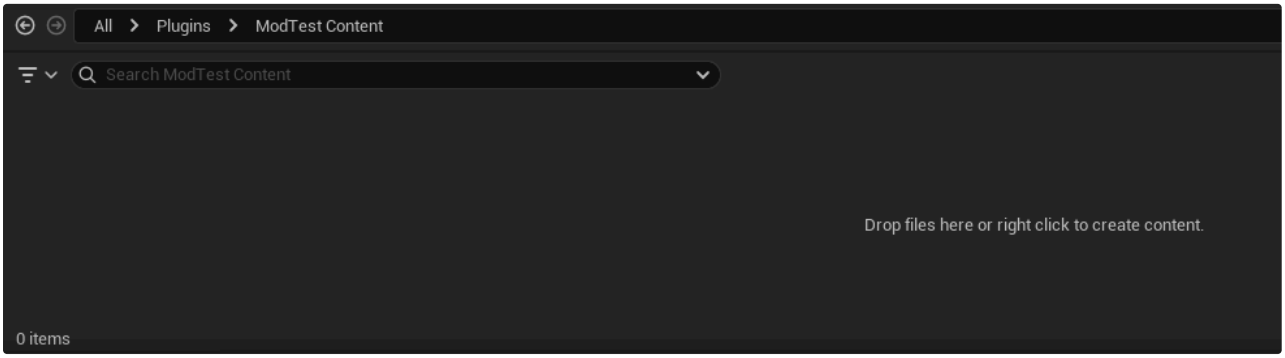
Create Mod screen

Click on Create Mod and it'll ask you to **restart** Unreal.



Screen after creating Mod

Now you should have a **Content** folder in the **Plugins** folder that has the same name as what you named your Mod.

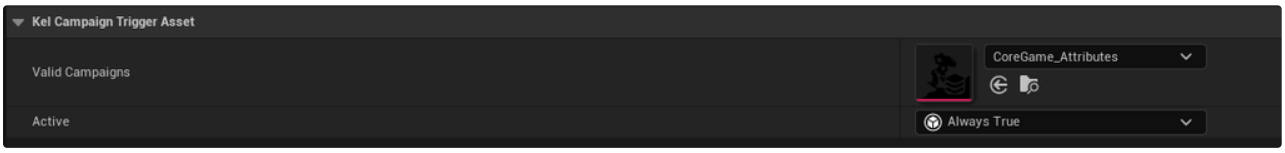


Mod folder

Set up the Holo Table

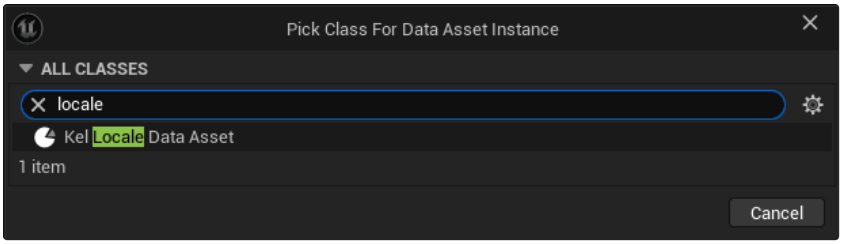
Create a **KelCampaignTrigger Asset**. That can be found under Kelpie. Open it up, change **Valid Campaigns** to **CoreGame_Attributes** and **Active** to **Always True**.

KelCampaignTrigger Asset

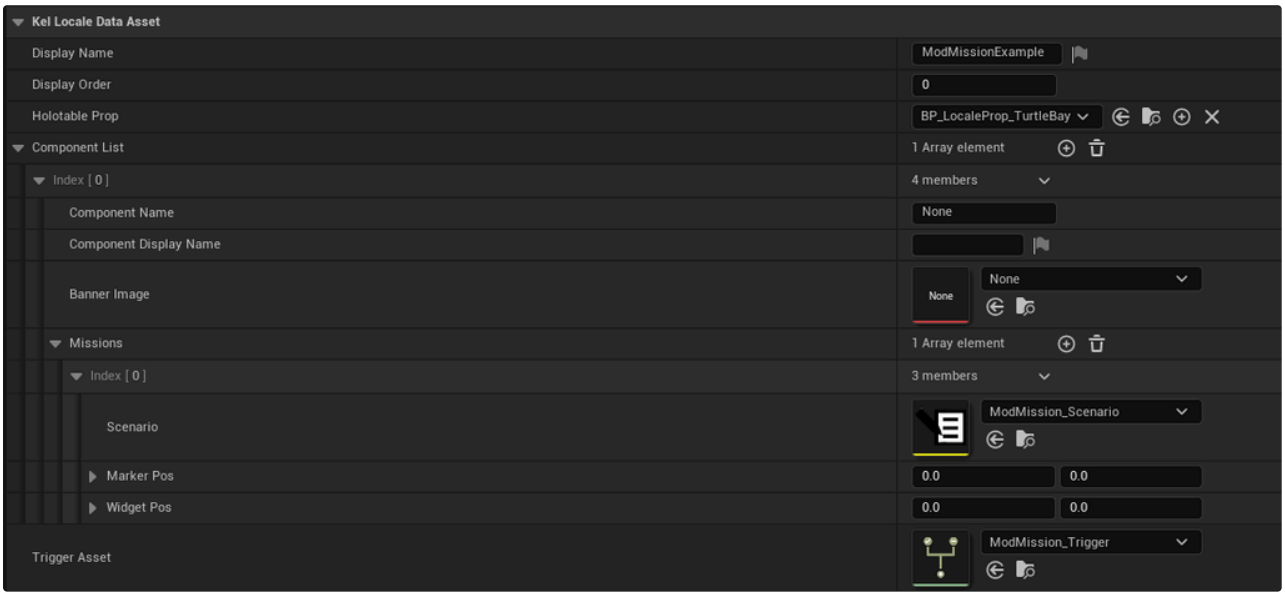


KelCampaignTrigger settings

Create a **DataTable Asset** as a **KelLocaleDataAsset**. Add a **Display Name, Holotable Prop, Component Name, Scenario** (Will be created later, see 'Make Mission Playable in Campaign') and **Trigger Asset**. Component Name should be **Planet** to make it appear.



Kel Locale Data Asset

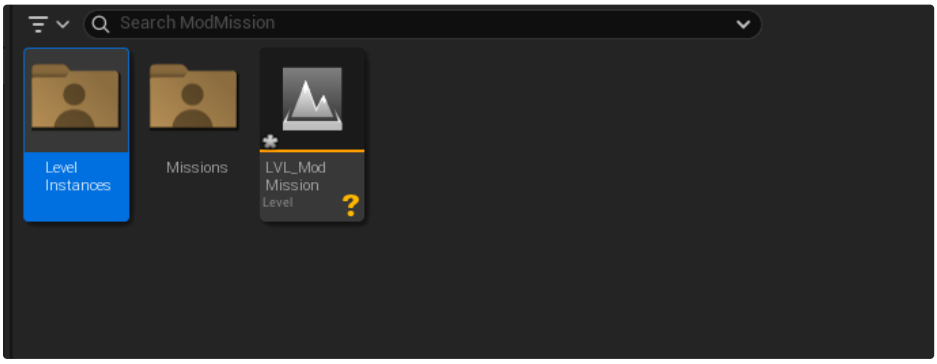


Kel Locale Data Asset settings

Creating a level

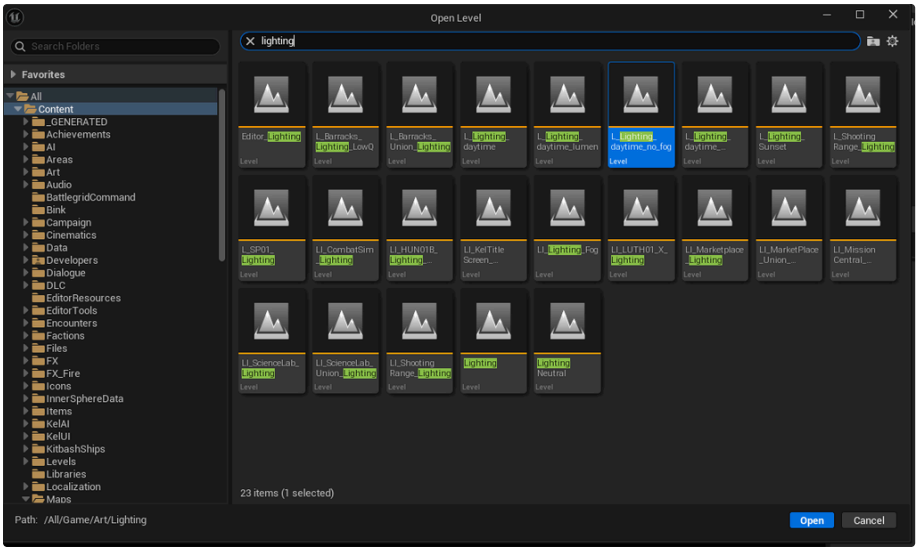
I'll explain here how to setup your level before actually going in and creating a playable mission.

I would start by creating a simple **Folder structure** that start with Missions or whatever name you prefer in case you want to create multiple levels to play. In there create a folder with the actual name of your mission. That's going to be the folder that we're working in. Right click anywhere in that folder and create your level. Create two folders, one for your sublevels and one for your mission files.



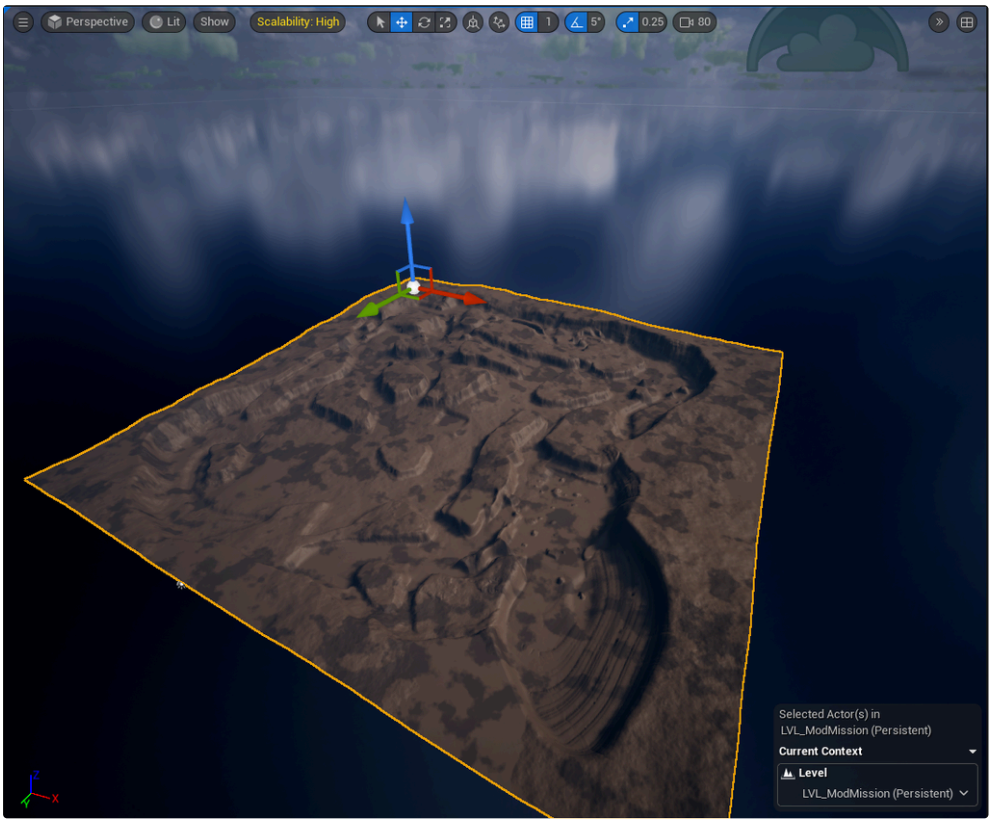
Simple setup with a folder for sublevels and a folder with your mission files

For now add an existing sublevel for lighting, so we can see things. I've added **L_Lighting_daytime_no_fog**.



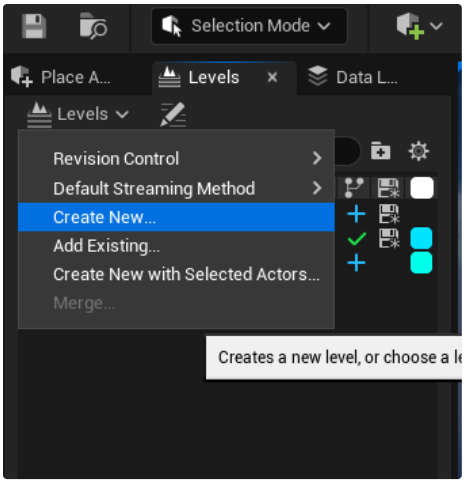
Add a lighting sublevel

Add a landscape with a **heightmap** that you can sculpt in Unreal or import. Landscape resolution size is 4081 x 4081. Used the material **LandscapeTest_MTL** from the project to see it properly.



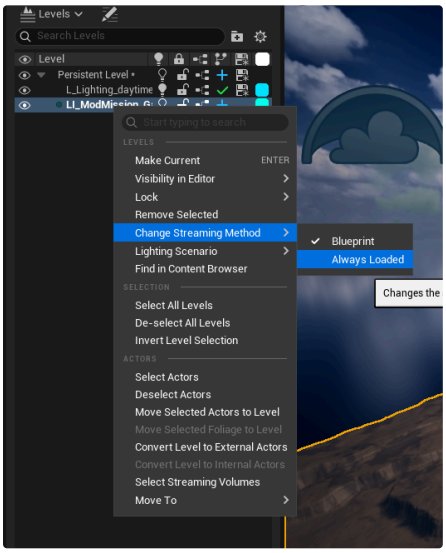
Your editor should look something like this

Create a new **Sublevel** and name it as you see fit.



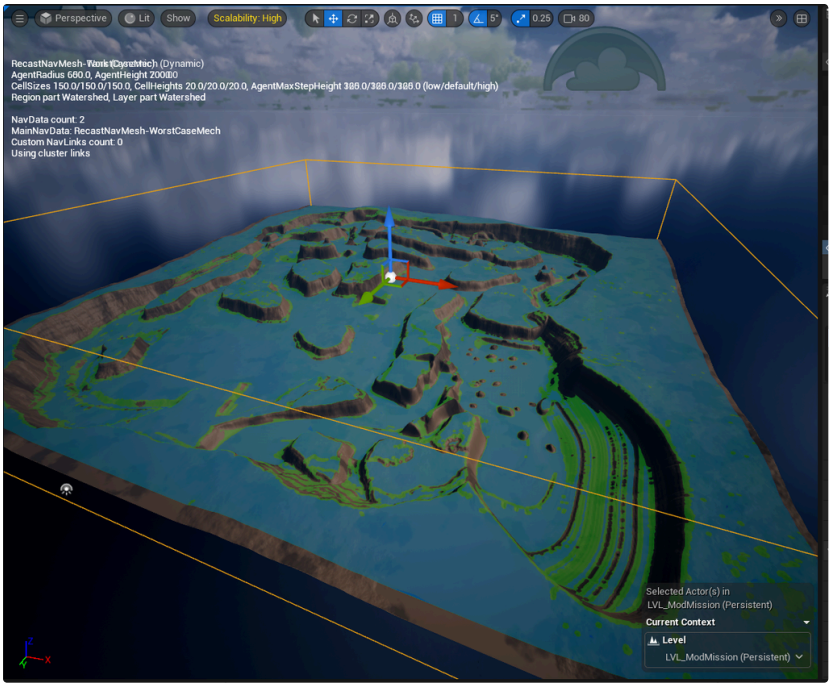
Can find it here

Make sure that the Sublevels are set to **Always Loaded**, so it's visible when you play the game.



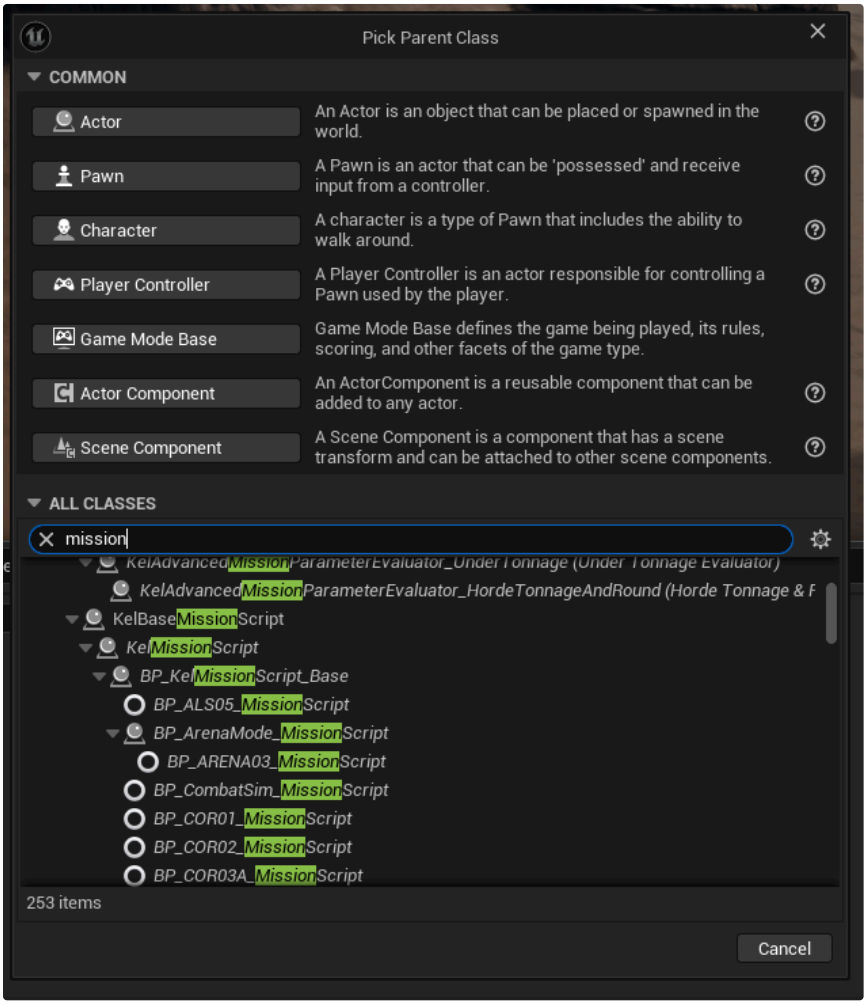
Only visible in game when loaded

Add a **NavMeshBoundsVolume** roughly the size of your map, so AI can move around.



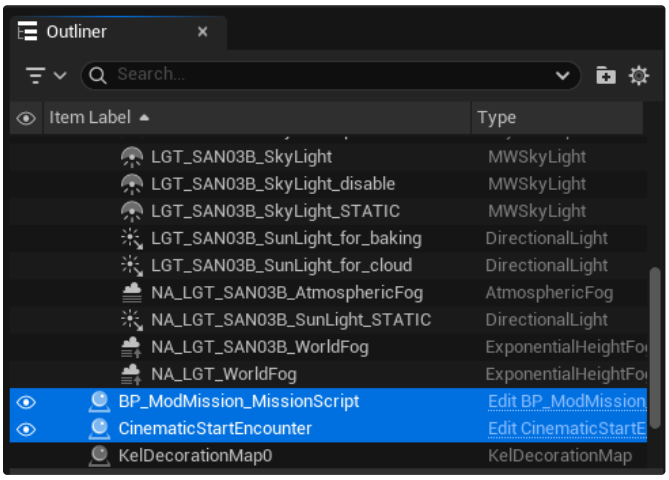
Press P to see NavMesh when build

In your Missions folder, create a **BP_KelMissionScript_Base** and name it according to your mission.



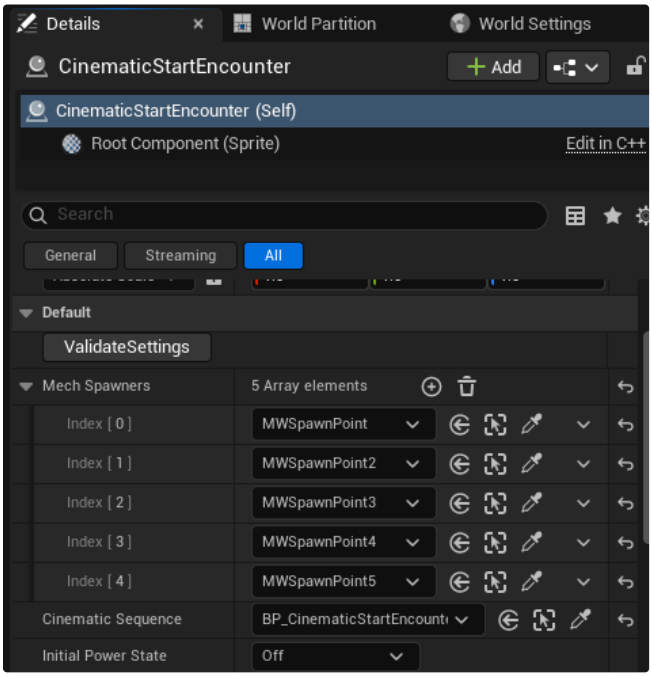
BP_KelMissionScript_Base is used to build your mission logic

After having created your MissionScript, drag it into your **Gameplay level**. Also drag in a **CinematicStartEncounter**.



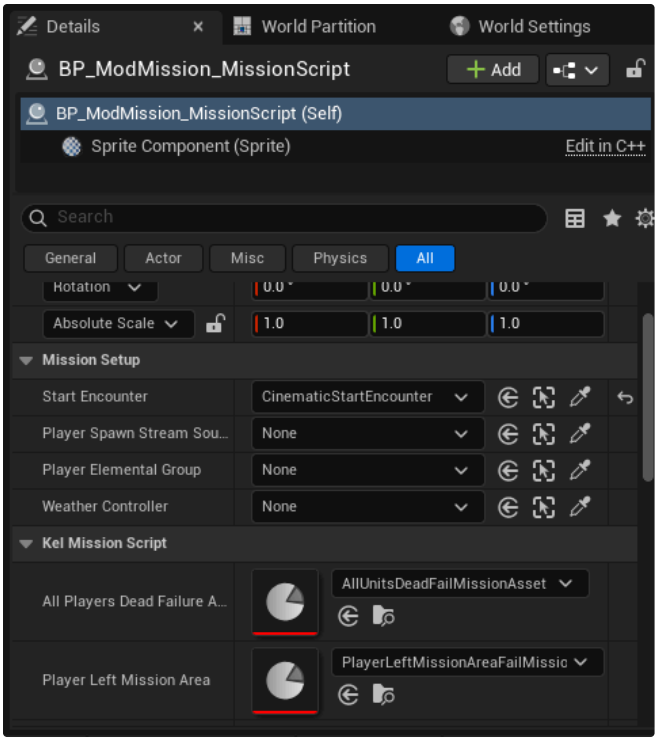
Both in Gameplay Sublevel

Drag in 5 **MWSpawnPoints** and a **BP_CinematicStartEncounterSequence**. Assign them in your **CinematicStartEncounter**.



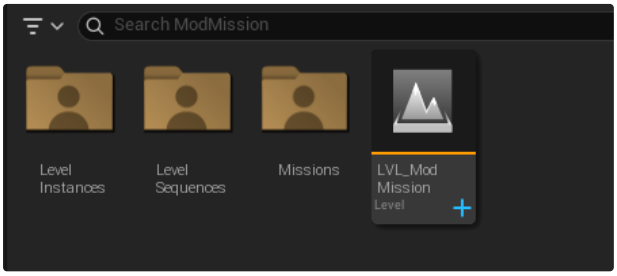
Player squad spawn points and blueprint for cinematic mission intro

Assign the **CinematicStartEncounter** to your MissionScript.

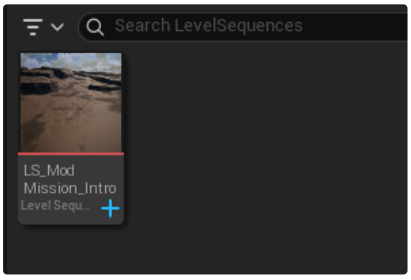


Now the Missionscript tells Players to spawn

Create a new folder for your **Level Sequences**. This project will need a sequence at the start of a mission to allow the mechs to walk around. Create a new Level Sequence and add a Fade track of 1 second. You can create any intro you like later by changing that sequence.

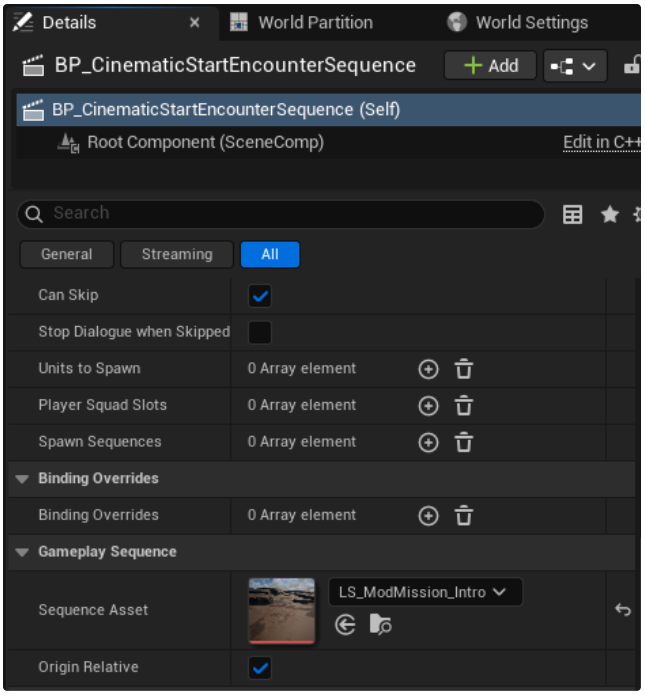


New LevelSequences folder



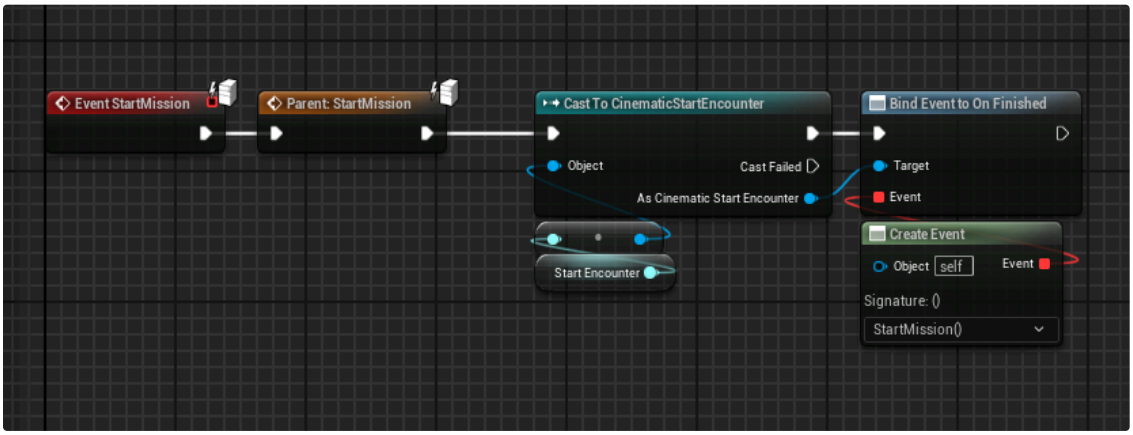
Intro Sequence

Add that Intro Sequence to your **BP_CinematicStartEncounterSequence**.



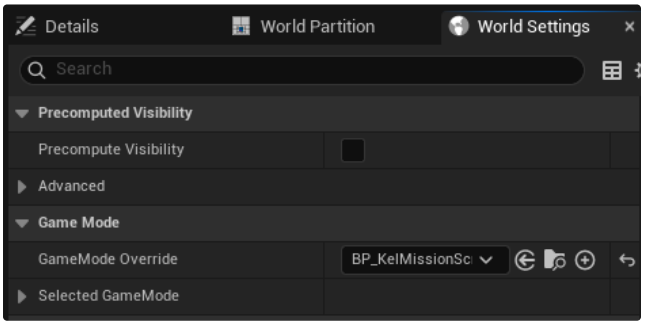
Assign it to the Sequence Asset in BP_CinematicStartEncounterSequence

Open your MissionScript and add this to play your Intro Sequence.



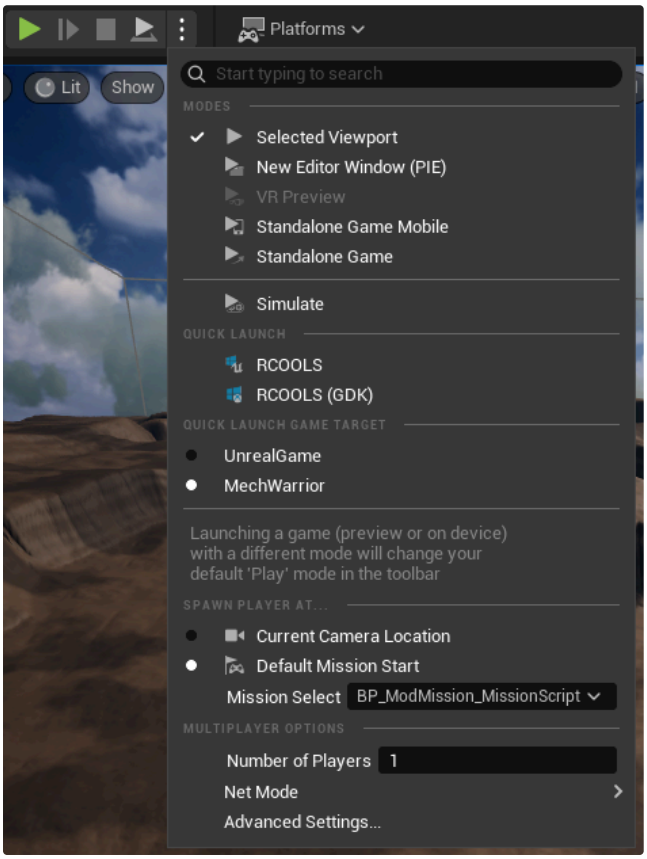
Start mission setup

To setup the Editor to test what you've created, we'll have to **Override the GameMode** in the **World Settings** to **BP_KelMissionScriptTestMode**.



Change GameMode to play in Editor

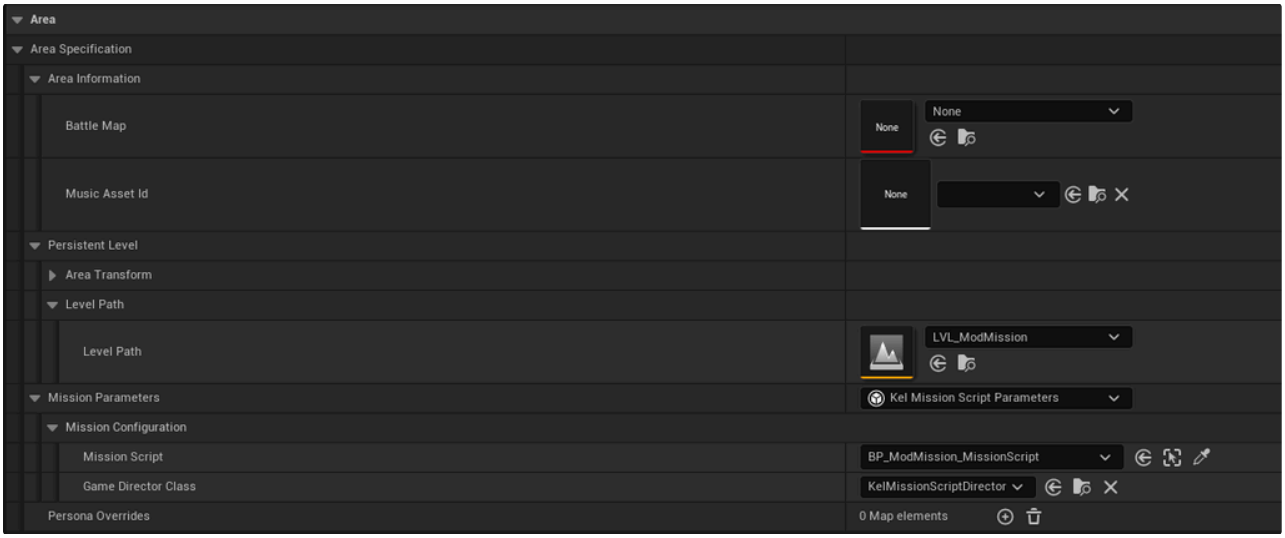
One final step before playing the mission. Make sure that the **MissionScript** and the **Default Mission Start** are selected here.



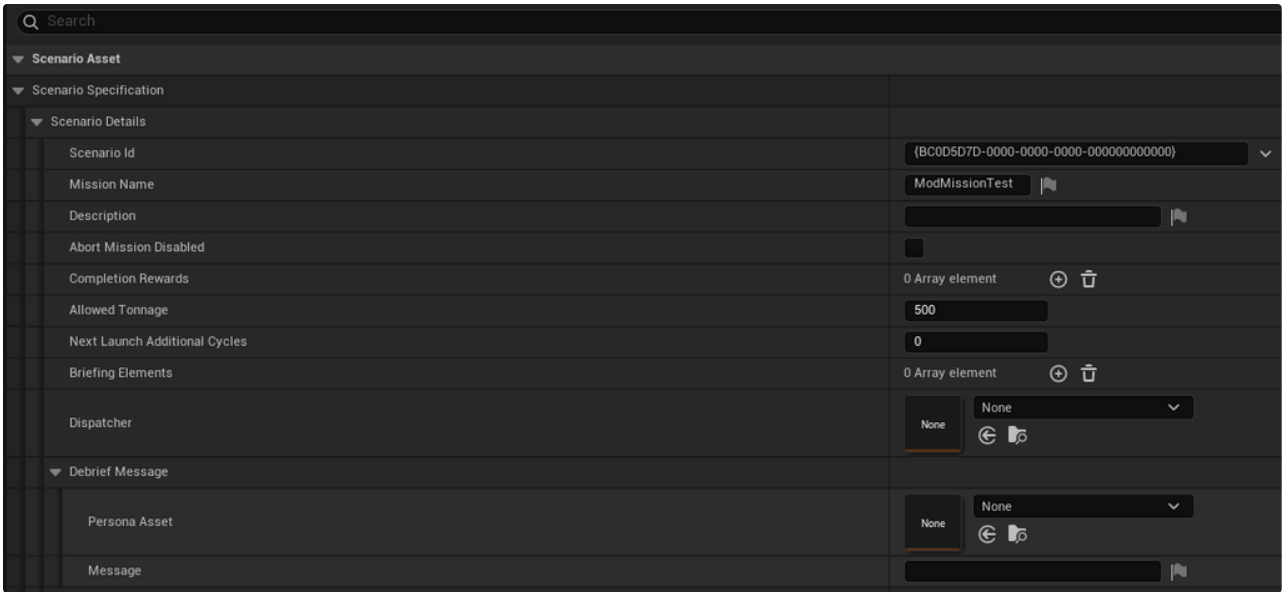
Select the MissionScript to tell the game to use your logic.

Make Mission playable in Campaign

Create an **MW.AreaSpecification Asset**. Level Path will be your **Persistent Level**. Mission Script will be the **MissionScript** you created.



Create an **MW.ScenarioSpecification** Asset. Generate a **Scenario Id**, add a **Mission Name** and set **Allowed Tonnage** to **500** or lower depending on what weight Mechs are allowed in your mission.



Scenario setup

Set Area Spec Asset to the **MW.AreaSpecification Asset** that you created and the Trigger Condition to the **KelCampaignTrigger Asset** you created.

Scenario setup with MW.AreaSpecification and KelCampaignTrigger

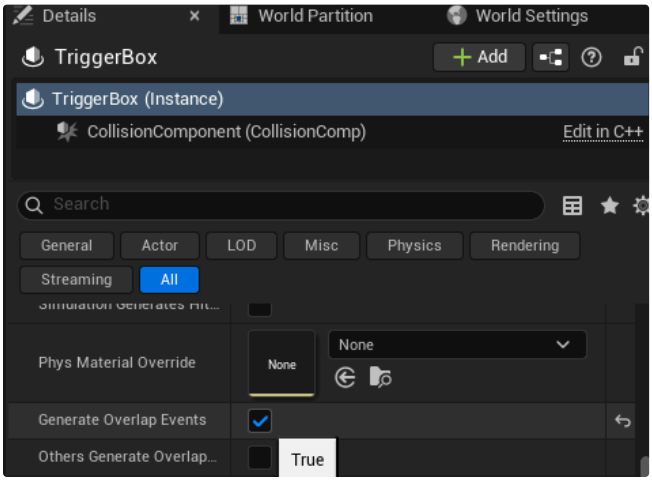
Creating a mission

Objectives

Objective Waypoint

Drag in a **trigger**, make it quite big and make sure that **Generate Overlap Events** is true.

Trigger in Gameplay level

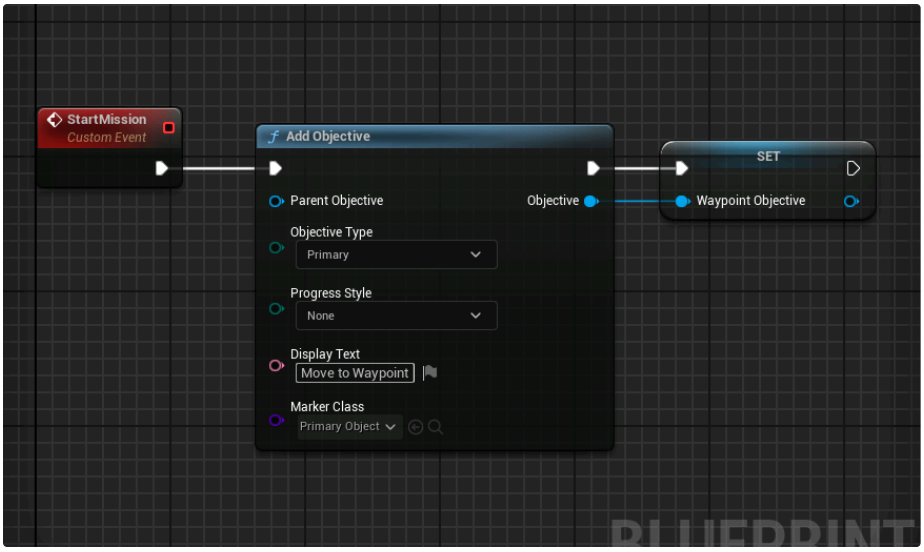


Generate Overlap Events is true

Open the MissionScript and create a new variable of type **TriggerBase**. Make that a **Soft Object Reference**.

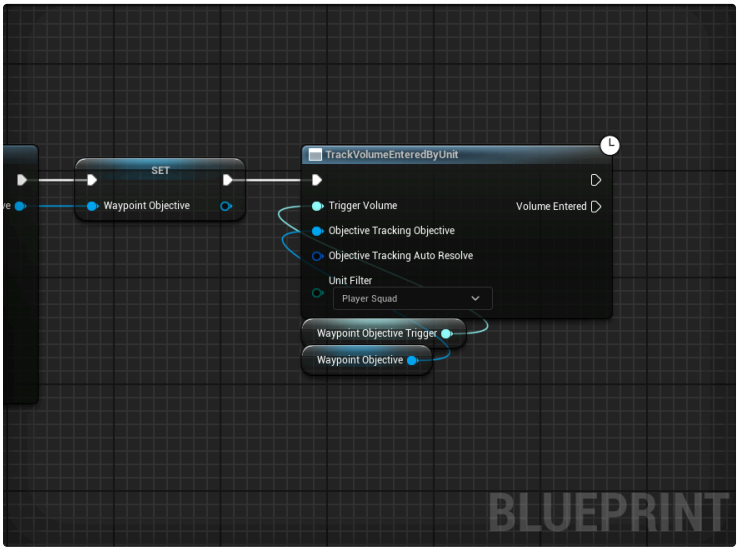
Trigger Base Soft Object Reference

From the event where you start your mission, add a node for **Add Objective**. Give it a **Display Text**(Will display objective text on left hand side of the screen) and a **Marker Class**(Marker that appears in world). Good practice is to create a variable for that to use it later if needed, but not obligatory.



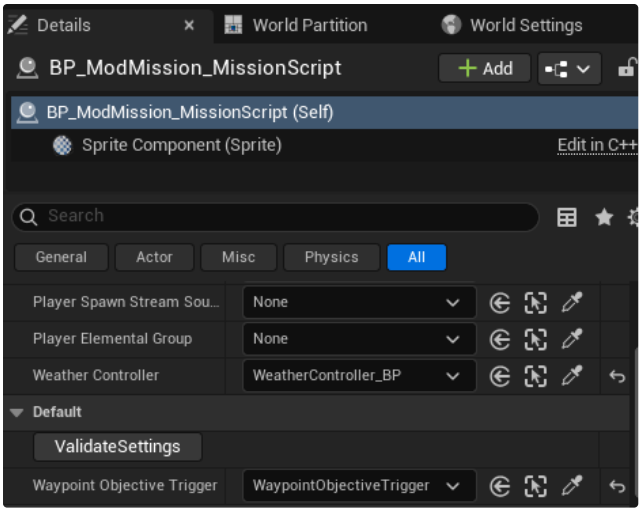
Add Objective

To make it appear on the trigger and resolve when the trigger is hit, we need to add a **TrackVolumeEnteredByUnit** node. Plug in your trigger and objective variable to that node and it should look like this. You can add more logic that fires after that node or when the trigger is hit by what is specified in the unit filter.



TrackVolumeEnteredByUnit

Assign the trigger to your **TriggerBase** variable in your MissionScript.

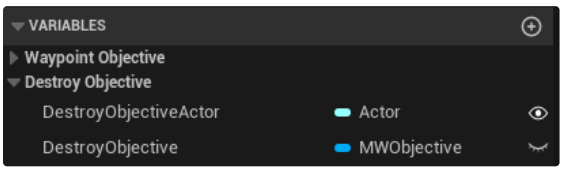


Trigger assigned to TriggerBase variable

Destroy Objective

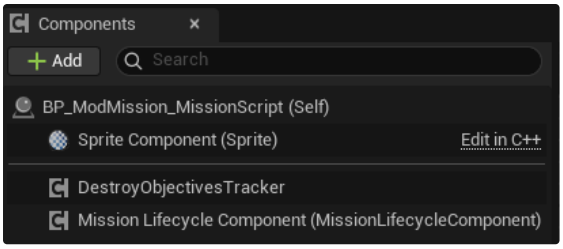
Drag in an object with a **Destructible Component**. In this case, I've used **UTL_Industrial_Military_SatelliteDish**.

In the MissionScript, create variables for your destructible actor and your objective. The destructible actor will be of type **Actor** and a **Soft Object Reference**. The Objective variable will be of type **MWObjective** and an **Object Reference**.

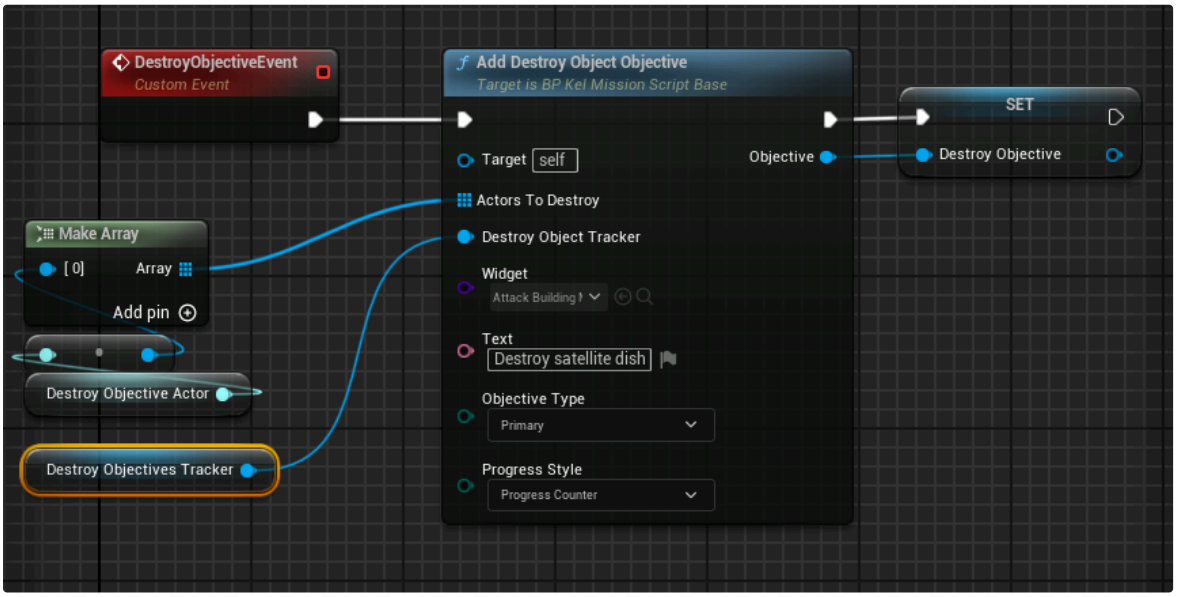


Destroy Objective variables

In the Components tab, create a **DestroyObjectivesTracker**. Drag that into the Event Graph, get the **Objectives to Detroy** variable from the Destroy Objectives tracker and add the Actor you want to destroy to that array.

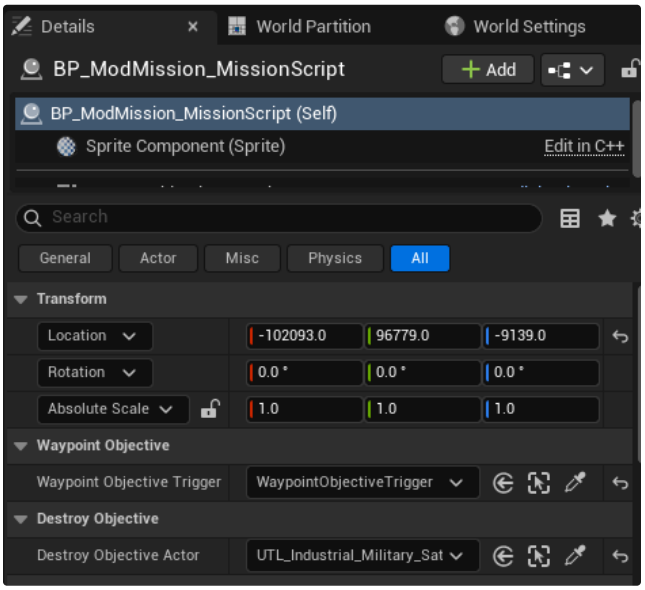


Create an **Add Destroy Object Objective** node, fill in the **Display Text** and the **Marker Class**. Plug in the variables for the Actors to Destroy and Destroy Object Tracker. You could change the Progress Style to Progress Counter if you want it to count for example, but not necessary.



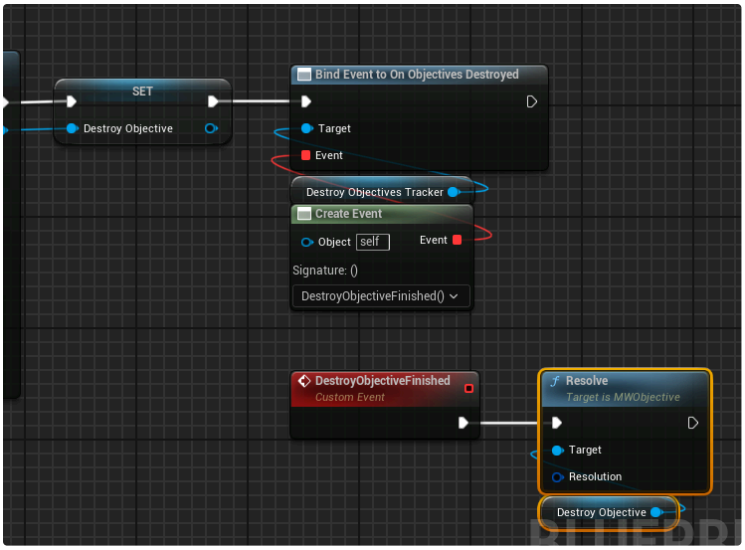
Setup for tracking if an object is destroyed

Add the Destructible actor to your DestroyObjectiveActor variable in the MissionScript.



Assign object to DestroyObjectiveActor variable in MissionScript

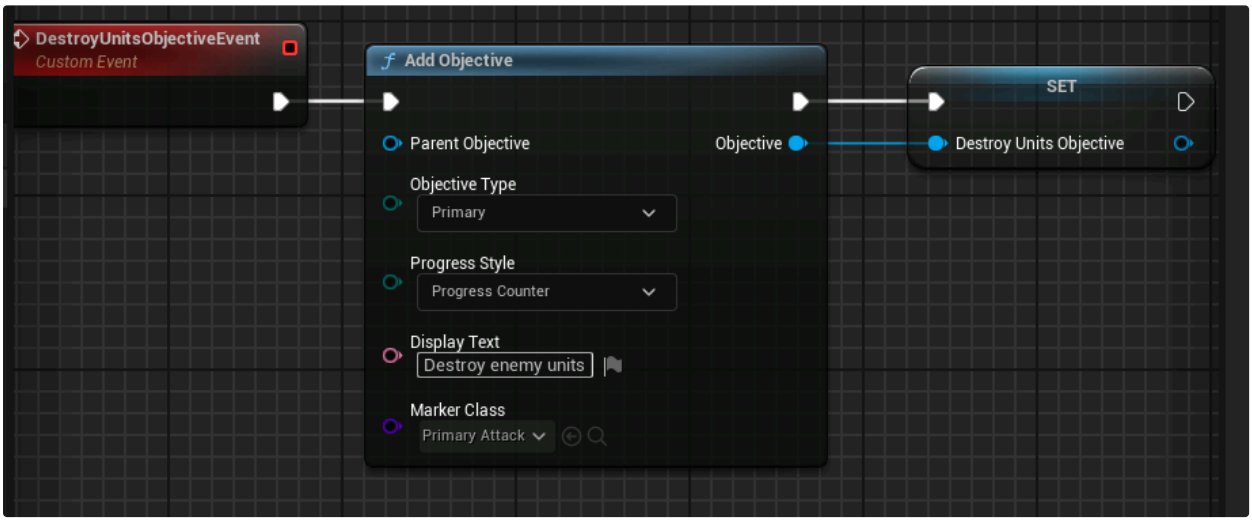
Bind and event to **OnObjectivesDestroyed** to Resolve the objective and add more things after the objective is finished.



Objective doesn't resolve automatically

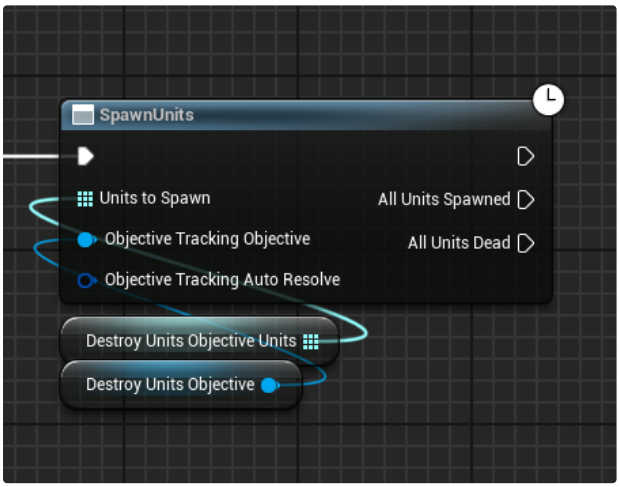
Destroy Units Objective

Create an **Add Objective** node and fill in the **Display Text**. Everything else is optional, depending on what you want to do.



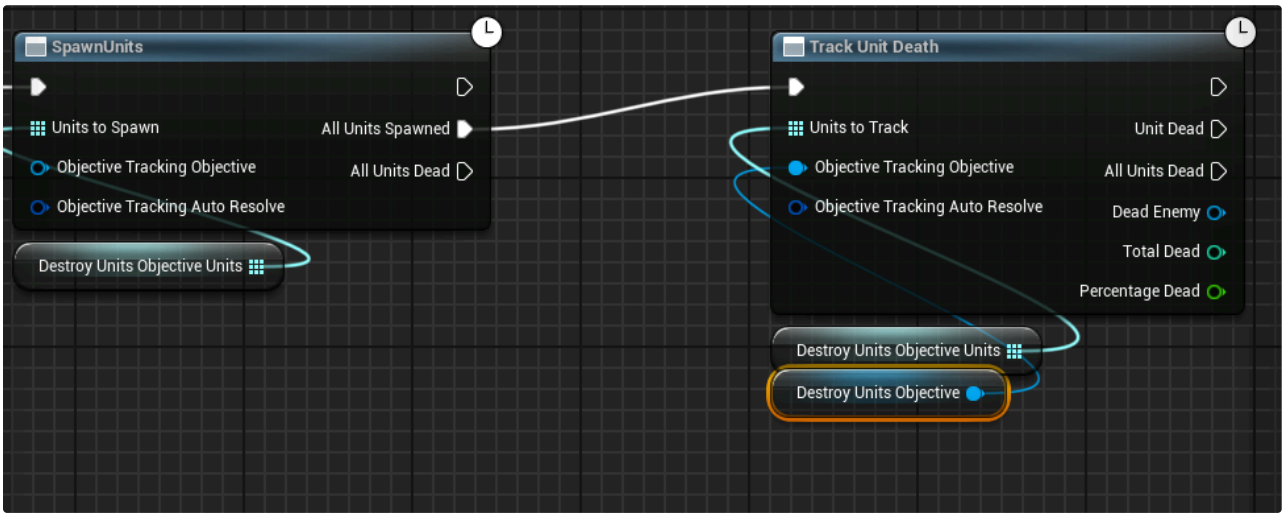
Add Objective node for tracking dead units counter

Plug in the **MWObjective** variable to the SpawnUnits node and it should track the units spawned.



SpawnUnits node can track the destroyed units

Can also track the objective like this if you decide to Track the objective later, rather than immediately at spawn.



Track Destroy Units Objective after units have spawned

Scan Objective

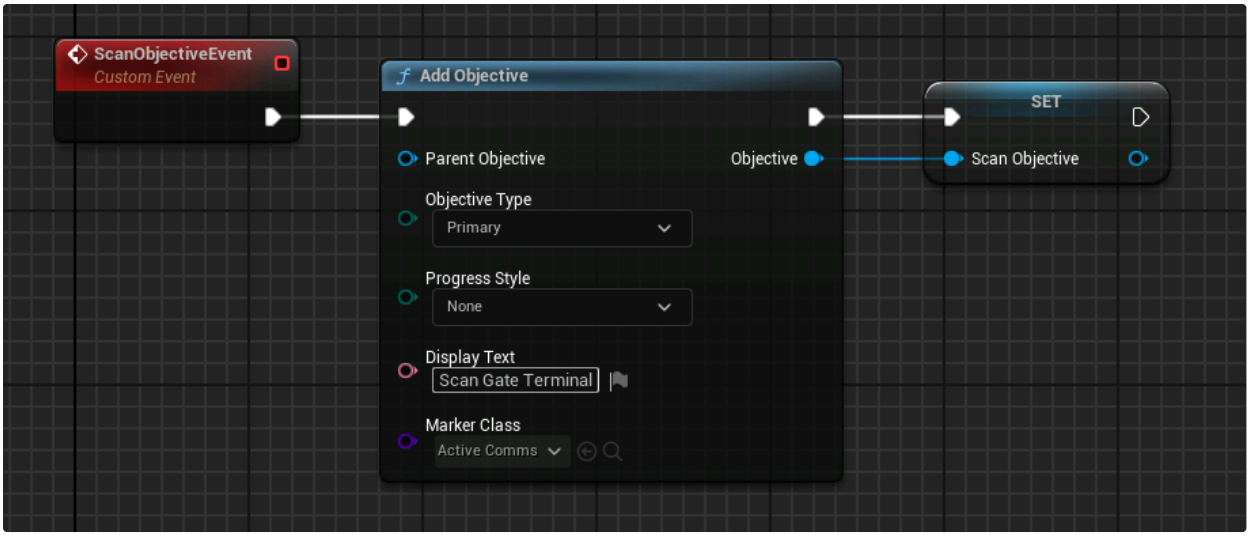
Drag an actor with a **KelScannableComponent** in your level. In this example **BP_GateTerminal**

Actor with KelScannableComponent

Create variables in your MissionScript for the Actor to scan and the scan objective.

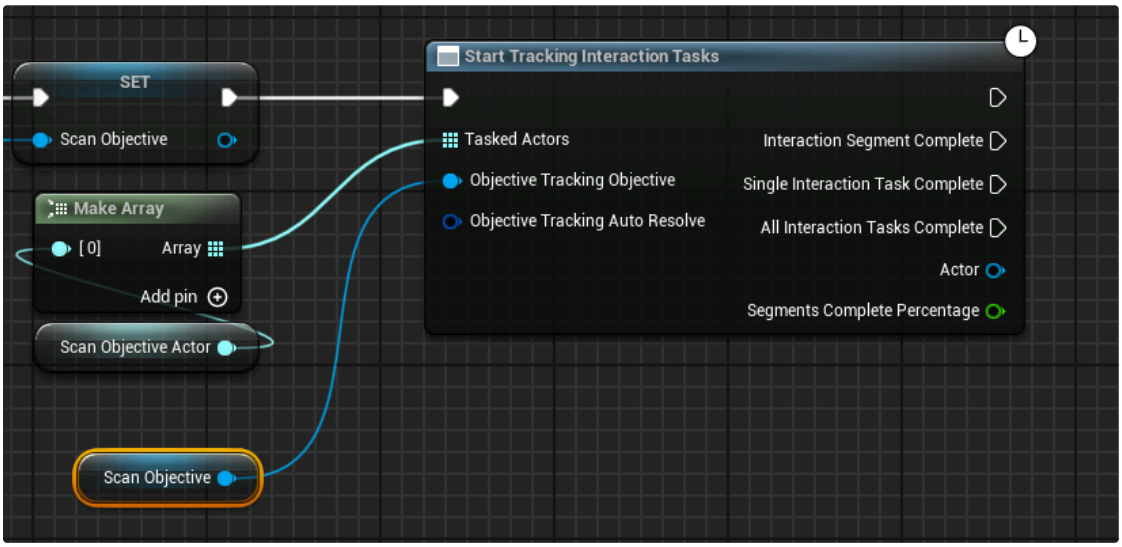
Variables for scan objective

Create an **Add Objective** node and set a **Display Text**. Everything else is optional.



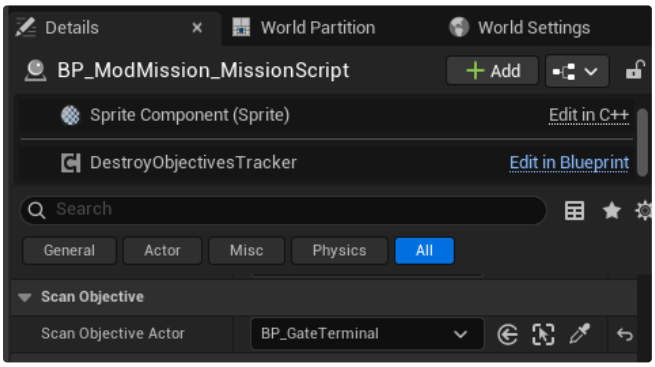
Add Objective node also used for Scan objectives

Add a **StartTrackingInteractionTasks** node and plug in the ScanObjectiveActor and the ScanObjective.



The scanning is tracked and objective completes when scanned

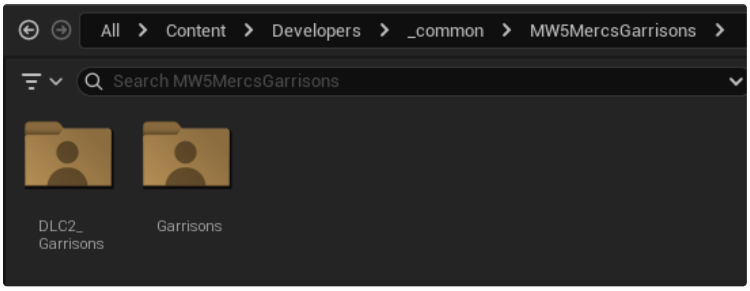
Assign the Actor to scan to the ScanObjectiveActor variable in your MissionScript.



Actor to scan assigned to MissionScript

Destructible zone Objective

Before actually creating the objective, you need something to destroy. We can add a **Garrison** from MW5 Mercenaries in our level. They can be found in this folder.

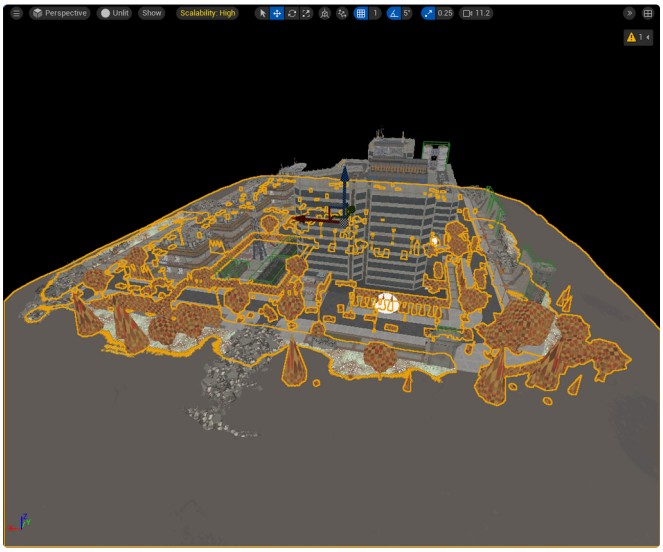


Garrisons can be found here

I recommend **copying** the Garrison that you like in your LevelInstance folder.

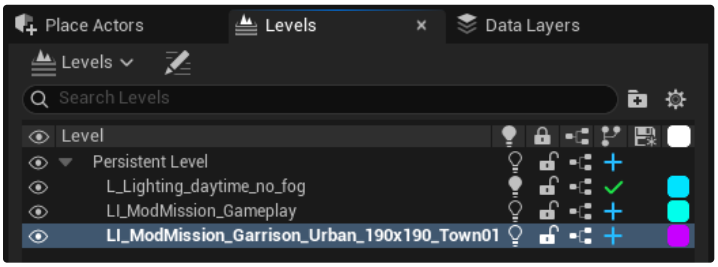
Garrison level copied

Open the level and **delete** the **foliage** and **landscape**. These actors aren't used in clans, so won't work.

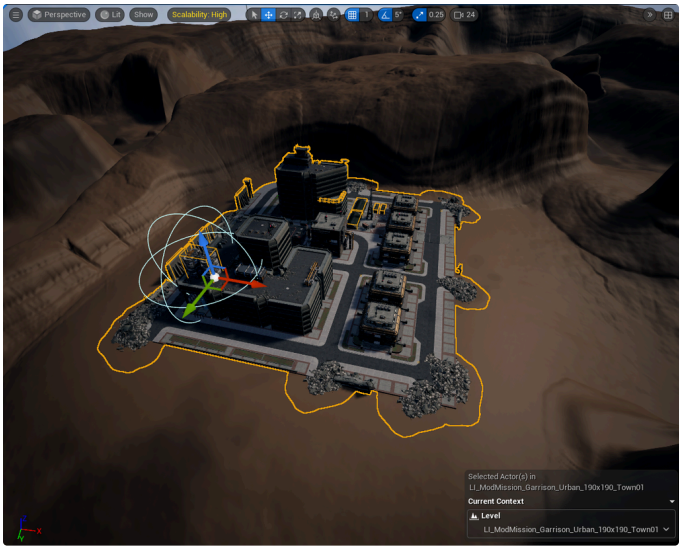


Selected Foliage and Landscape actors

Add it to your level as a **sublevel** and move it where you want it to be.



Sublevel added

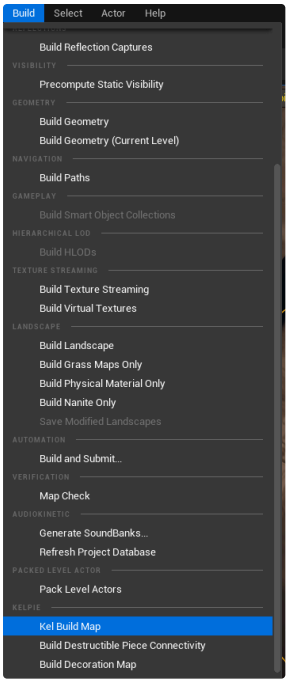


Garrison in level

Add a **BP_KelMissionObjectiveDestructibleZone** to your level and put it on your Garrison. Change the **radius** to make it big enough.

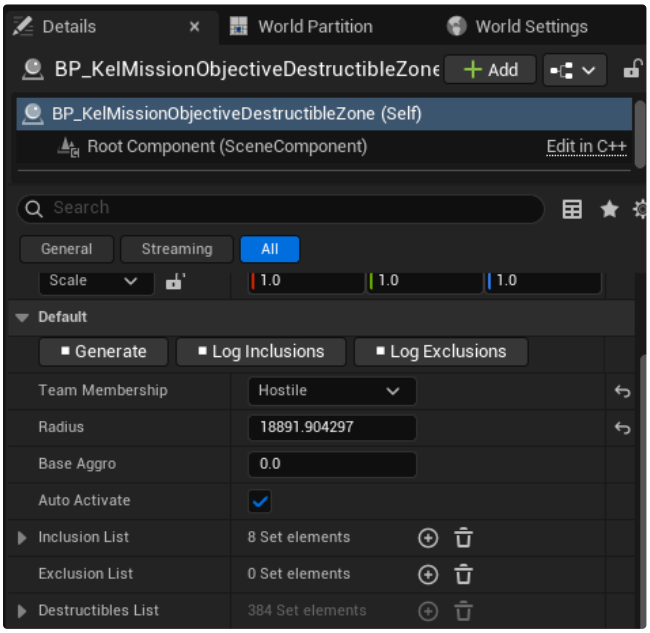
Destructible zon placed around Garrison

Go to Build and select **Kel Build Map**. This will build your destructibles and navigation at the same time.



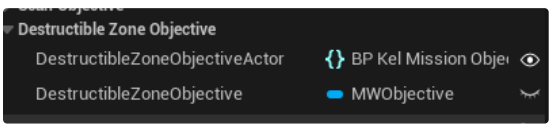
Kel Build Map can be found in Build

Select your **BP_KelMissionObjectiveDestructibleZone** and click on **Generate** to make sure that it registers the destruction. Also set it to hostile if you want friendly mechs to target the buildings.



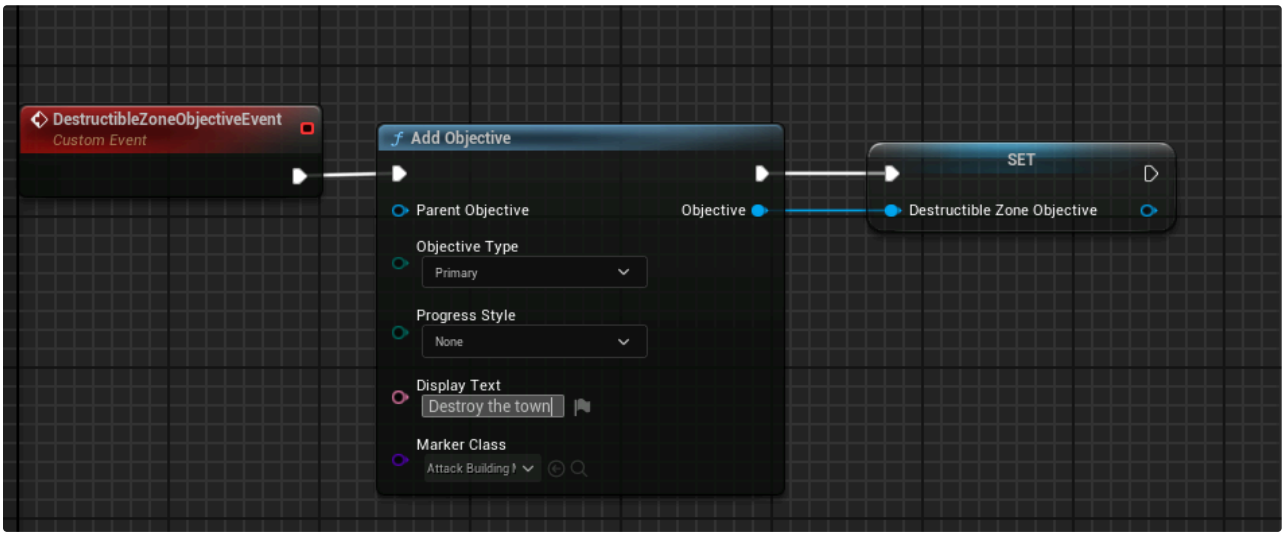
BP_KelMissionObjectiveDestructibleZone settings

Create a **SoftObjectReference** variable for your Destructible zone of type **BP_KelMissionObjectiveDestructibleZone**. This time make it a set. Also a variable for your MWObjective.



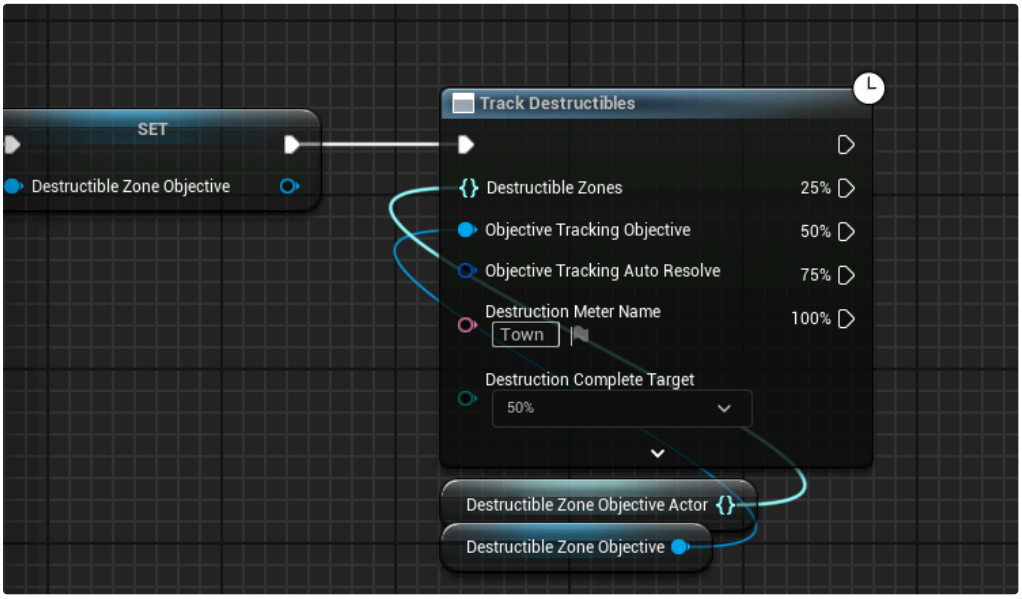
Destructible zone objective variables

Create a new **Add Objective** and fill in the **Display Text**.



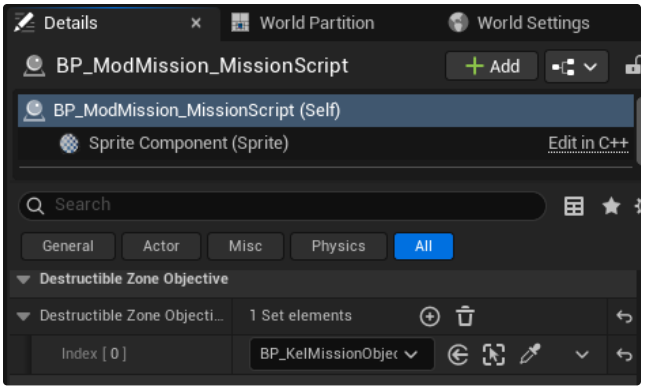
Objective for Destructible zone

Add a **Track Destructibles** node, plug in your Destructible zone and objective variables. Give the **Destruction Meter** a name to tell players what they're destroying and set a **Destruction Complete Target** of when the objective completes. As you can see in the image, you can fire logic after certain percentages of destruction.



Destructible zone objective

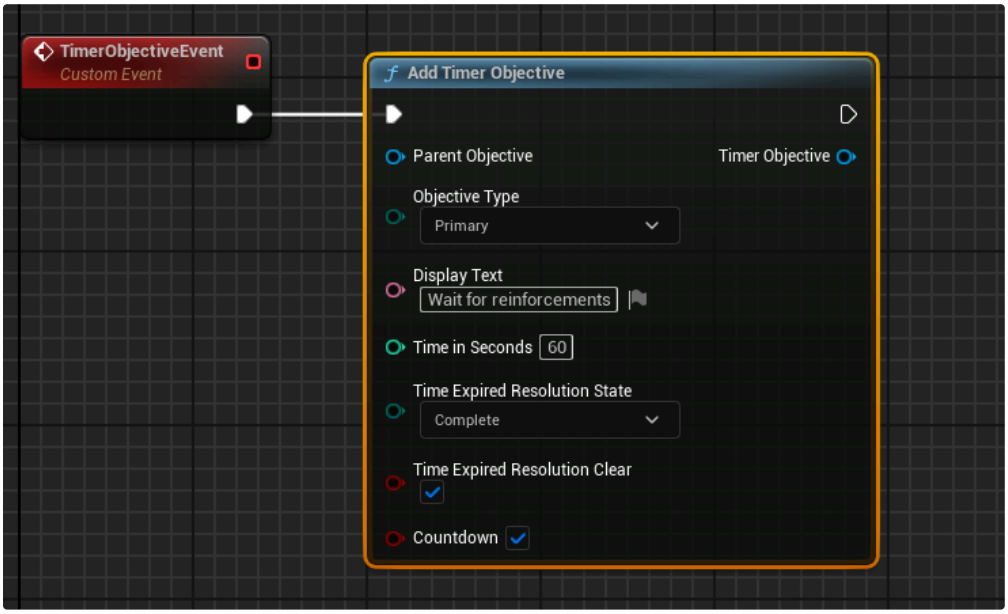
Assign the BP_KelMissionObjectiveDestructibleZone to your MissionScript.



Destructible zone assigned in MissionScript

Timer Objective

In the MissionScript, create an **Add Timer Objective** node. Fill in the **Display Text**, **Time in Seconds** and **Time Expired Resolution**.



Timer objective setup

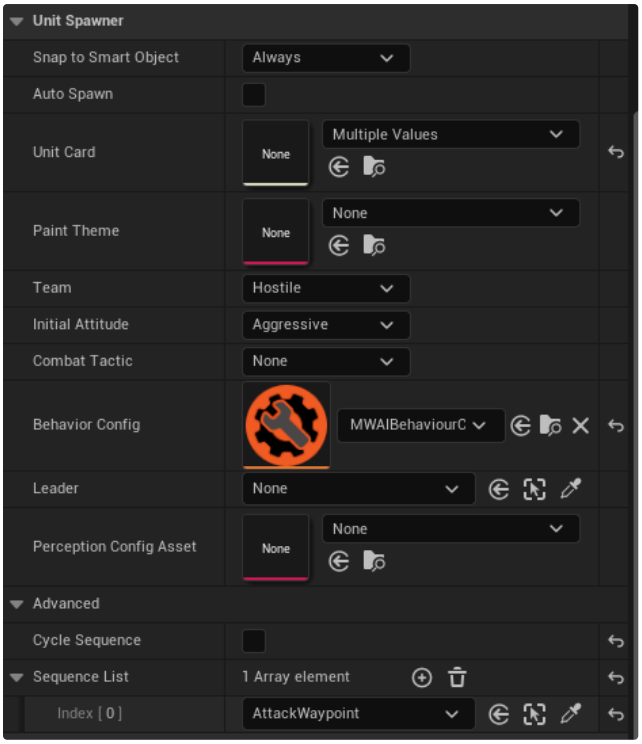
Spawning AI

Spawn enemies by placing **Unitcards** in your level. You can tell them to **AutoSpawn**, but I recommend telling them to spawn through the **MissionScript**. That way they can spawn in when you need them. I recommend after spawning a unit to have them walk to a **Waypoint** and then an **AttackWaypoint** to help them get out of their spawn location. Waypoints are explained in the Waypoint section.

Mech UnitCards

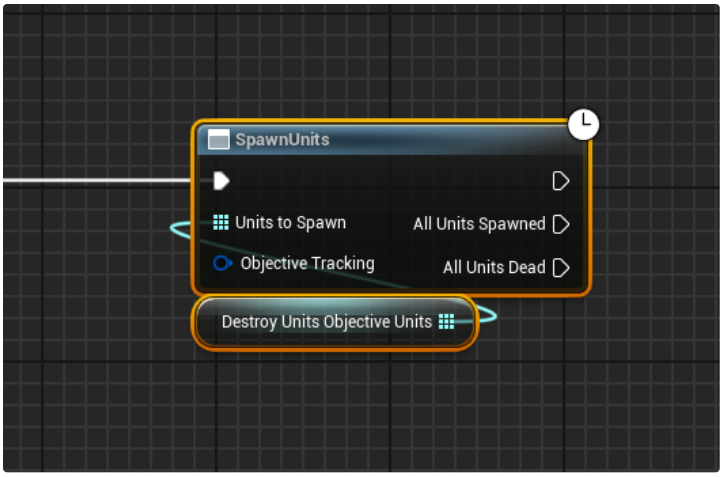
Fill in the details for the Unit behaviour. Most important are the **Team**, **Attitude**, **Behavior Config** and **Sequence List**.

- Team (Hostile, Neutral or Friendly)
- Attitude (Passive or Aggressive behavior)
- Behavior Config (Difficulty)
- Sequence List (Movement and Targeting)



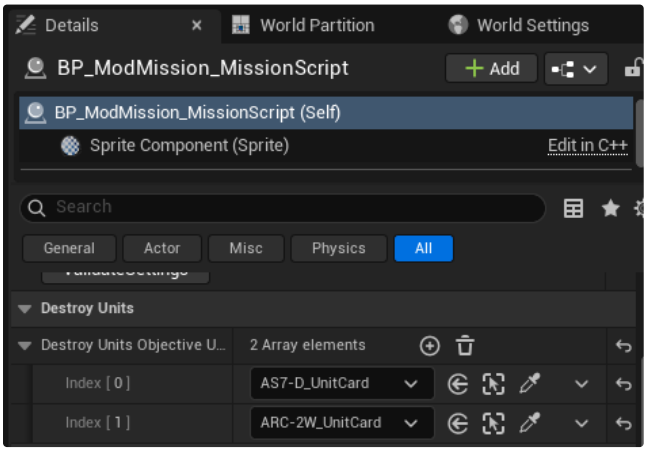
UnitCard settings

Create a variable of type KelUnitSpawner as a SoftObjectReference. Use a **SpawnUnits** node in MissionScript to spawn them.



SpawnUnits node

Assign the units to your KelUnitSpawner variable in your MissionScript.

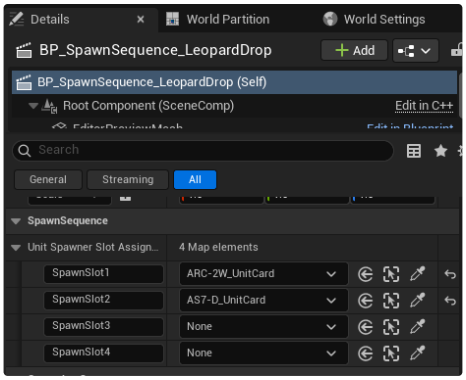


Array of UnitCards

There's a couple of locations that enemies can spawn from.

- Hide them (Use explanation above)
- Dropship:
  2 Dropships that work exactly the same way **BP_SpawnSequence_LeopardDrop**(Inner Sphere Mechs Max 4) and **BP_SpawnSequence_BroadswordDrop**(Clan Mechs Max 5). Assign the Unitcards that you want to drop out of the ship and follow the instructions above.

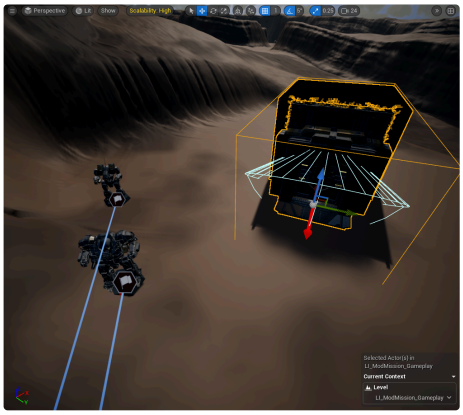
Both available dropships to spawn



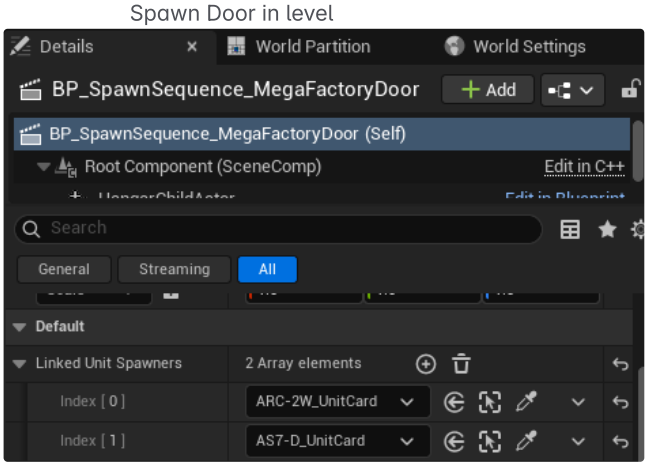
Assign Mechs to the dropship

- Spawn Door:

  Drag a **BP_SpawnSequence_MegaFactoryDoor** in your level and assign the Unitcards that you want to spawn.



Spawn Door in level

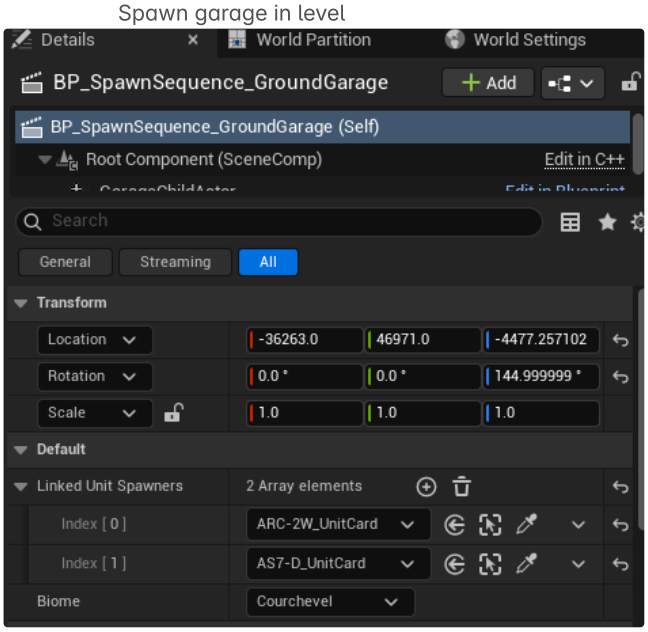


Assigned Unitcards to Spawn Door

- Spawn Garages:

  Drag a **BP_SpawnSequence_GroundGarage** into your level and assign the Unitcards that you want to spawn.

Spawn garage in level

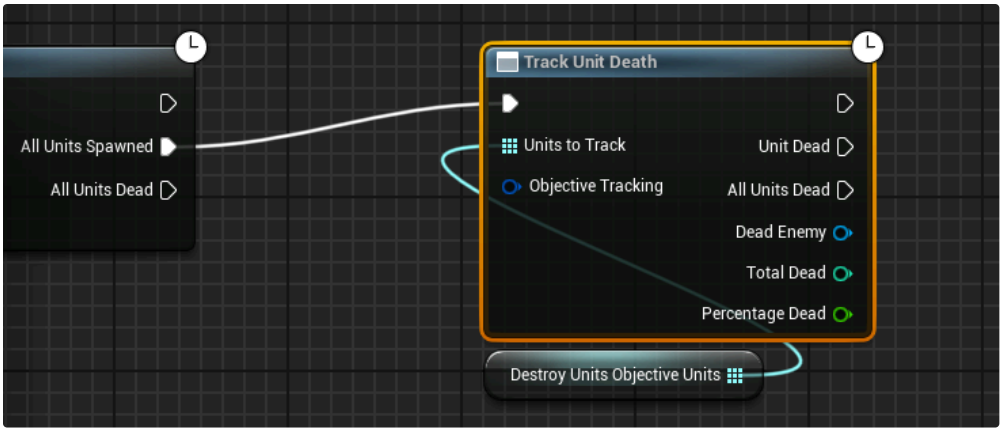


Assigned Unitcards to Spawn Garage

Track Unit Death/Damage

You can track the units death after they've been spawned already with **Track Unit Death**.

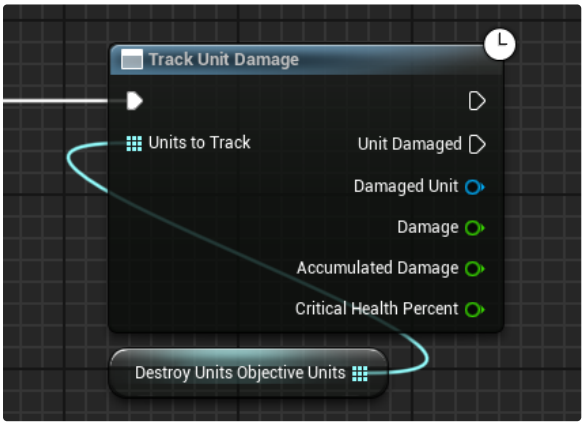
- Pass through (Can continue the fire logic after firing this node)
- Unit Death (Can fire logic each time a unit dies that's tracked)
- All Units Dead (Can fire logic when all tracked units are dead)
- Dead Enemy (Retrieves the actor that died)
- Total Dead (Counter for amount of Units Died)
- Percentage Dead (Percentage dead of the total tracked Units)



Track Unit Death node

Can also track units damage after they've spawned with **Track Unit Damage**.

- Pass through (Can continue the fire logic after firing this node)
- Unit Damage (Can fire logic each time a unit that's tracked receives damage)
- Damaged Unit (The tracked unit that received damage)
- Damage (Damage received)
- Accumulated Damage (Total damage received)
- Critical Health Percent (Unit at critical health)

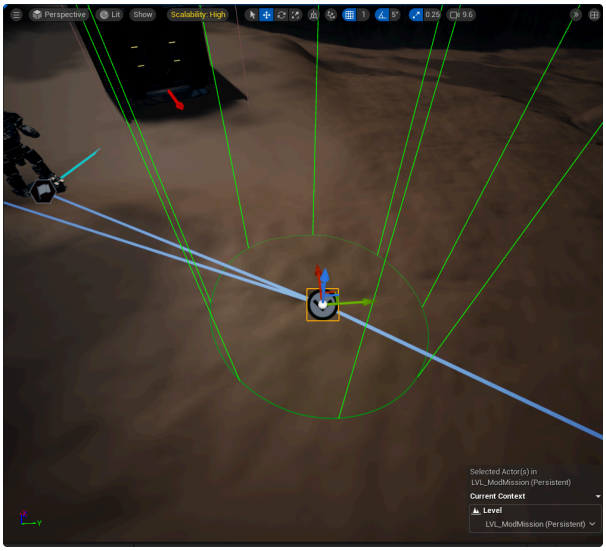


Waypoints

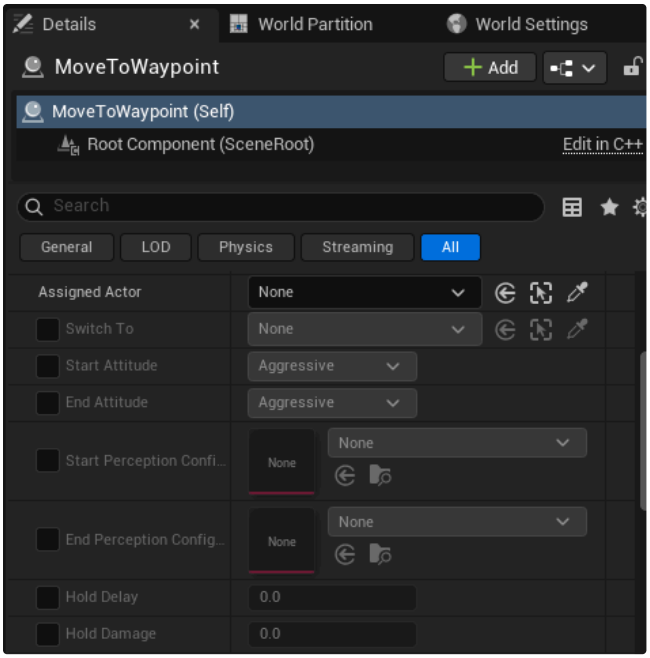
MoveToWaypoint

The name kind of explains it. It's a waypoint that the unit moves to. Can do a couple of things and I'll got through each one of them.

- Assigned Actor (Not used with this Waypoint)
- Switch To (UnitCards have usually have Waypoints in Sequence, this allows the unit to move to a Waypoint that's not in that Sequence)
- Start Attitude (This is the Units Attitude when it is moving to this Waypoint)
- End Attitude (Attitude when the Waypoint is reached)
- Start Perception Config (Not used)
- End Perception Config (Not used)
- Hold Delay (Amount of seconds before going to the next Waypoint 0.0 = infinite)
- Hold Damage (Amount of damage before moving to the next Waypoint)



MoveToWaypoint

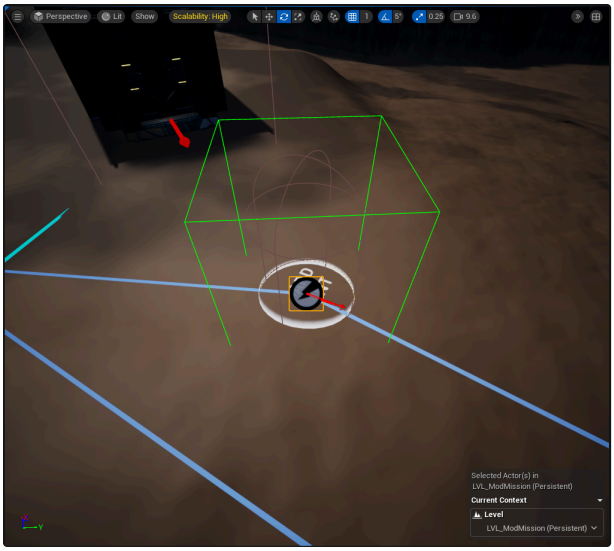


MoveToWaypointSettings

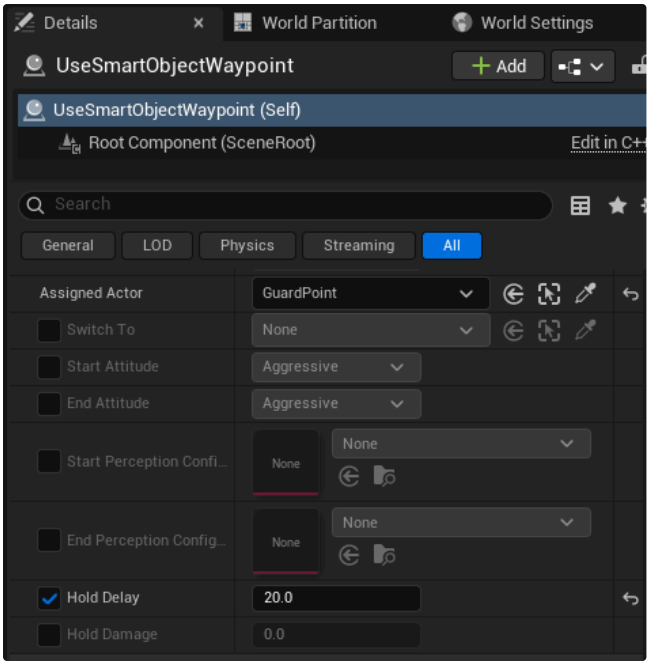
SmartObjectWaypoint

This Waypoint can do things to make the unit feel more **smart**. In my example, I'll use the **GuardPoint**, but I'll go through all of them. Same settings as a regular Waypoint, but now we can use **Assigned Actor**.

- Assigned Actor (Assign a Smart Actor to have the unit act different from Perceiving and attacking)
- Switch To (UnitCards have usually have Waypoints in Sequence, this allows the unit to move to a Waypoint that's not in that Sequence)
- Start Attitude (This is the Units Attitude when it is moving to this Waypoint)
- End Attitude (Attitude when the Waypoint is reached)
- Start Perception Config (Not used)
- End Perception Config (Not used)
- Hold Delay (Amount of seconds before going to the next Waypoint 0.0 = infinite)
- Hold Damage (Amount of damage before moving to the next Waypoint)



SmartObjectWaypoint



SmartObjectWaypoint used with a GuardPoint

The Smart Actors have different uses.

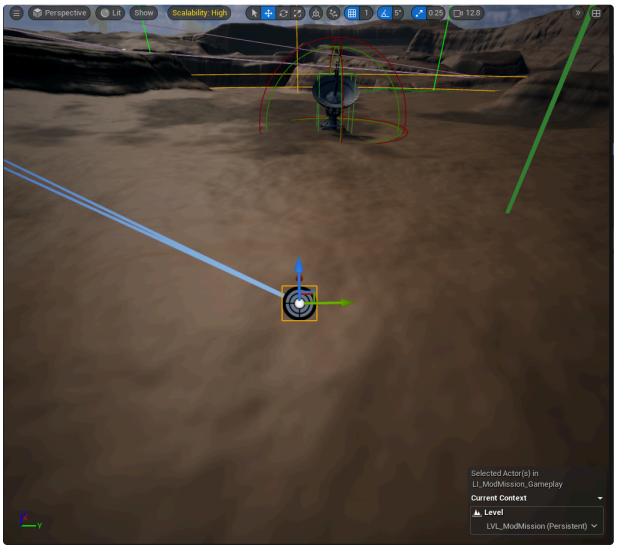
- GuardPoint (Has the unit wait for the Hold time specified and look left and right)
- ParkingSpot (Unit can't move from here unless told otherwise)
- Melee Target (Will melee targets within range of this Waypoint)
- LandingPad (Only for VTOLS and works similar to the ParkingSpot)
- LookAtPlayer (Unit will focus attention on the Player)

AttackWaypoint

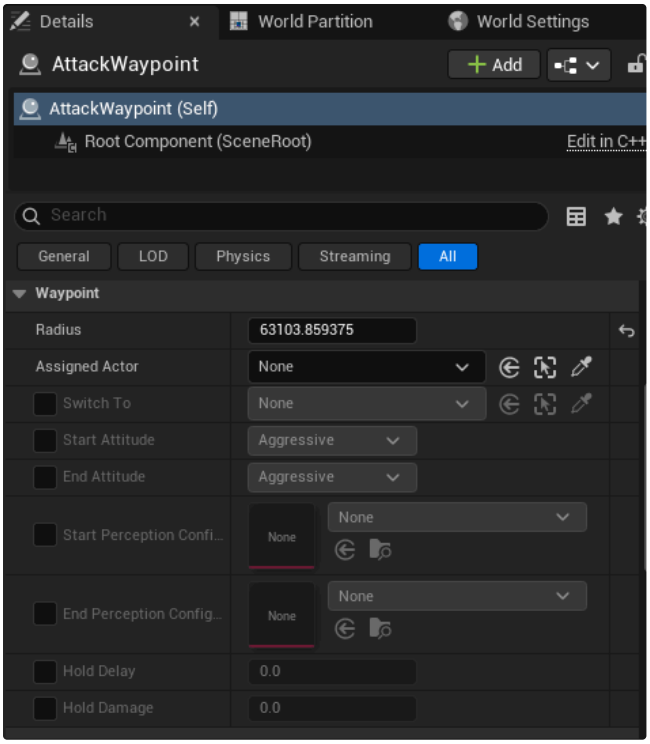
You want to use this one on all your AI that needs to attack. Anything that gets in the radius of this Waypoint will be selected as a **target**. This prevents AI from just standing around. Some extra functionality has been added to the **AttackWaypoint**.

- Assigned Actor (Not used with this Waypoint)
- Switch To (UnitCards have usually have Waypoints in Sequence, this allows the unit to move to a Waypoint that's not in that Sequence)

- Start Attitude (This is the Units Attitude when it is moving to this Waypoint)
- End Attitude (Attitude when the Waypoint is reached)
- Start Perception Config (Not used)
- End Perception Config (Not used)
- Hold Delay (Amount of seconds before going to the next Waypoint 0.0 = infinite)
- Hold Damage (Amount of damage before moving to the next Waypoint)



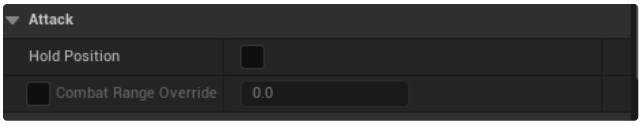
AttackWaypoint



Same options to change as other Waypoints

The new functionality gives a little bit more freedom in attack behavior.

- Hold Position (Will stay in the same location when Waypoint has been reached)
- Combat Range Override (Usually AI uses the radius from the AttackWaypoint as combat range, but this overrides that with this range instead)



Extra functionality AttackWaypoint

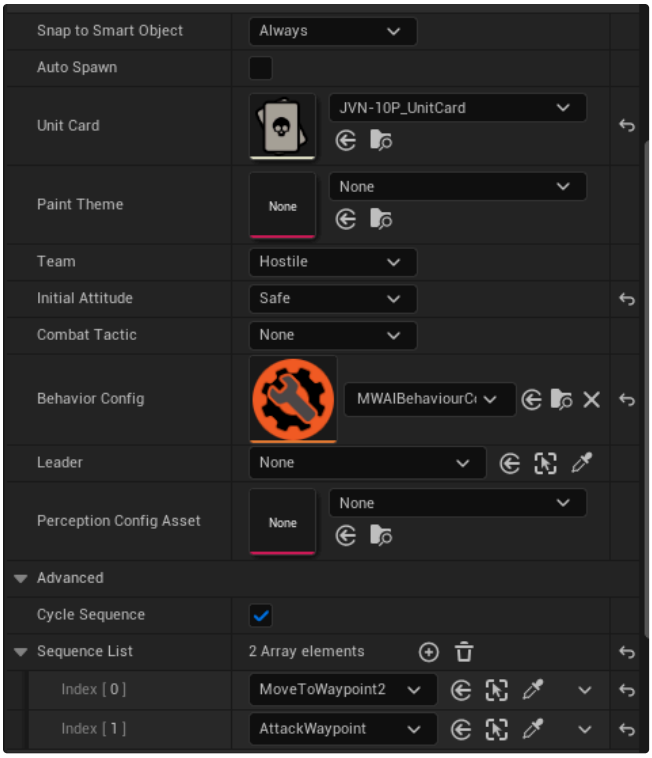
JumpJet Links

Some mechs can use **JumpJets**. We can force them to use those to jump over obstacles or jump down ledges.

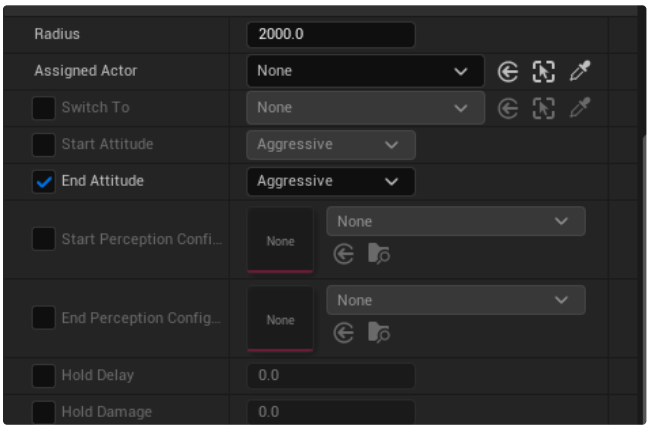
Drag in a **UnitCard** with **JumpJets** and set it up to spawn when you need it.

Unit with JumpJet

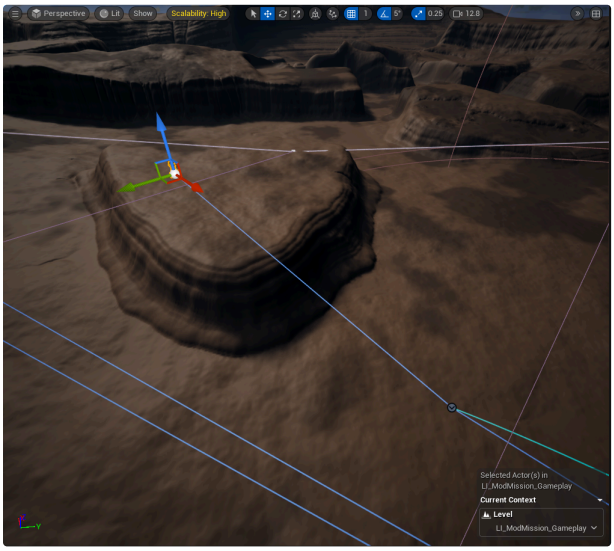
I recommend setting the **Initial Attitude** to either **Safe** or **Passive**, so it won't start attacking and moves away from its path. Give it a **MoveToWaypoint** to tell the unit where it should move to. Give it an **AttackWaypoint** as well to tell it to attack when the destination is reached. Set the **End Attitude** on the MoveToWaypoint to **Aggressive** to have it attack.



Settings for UnitCard



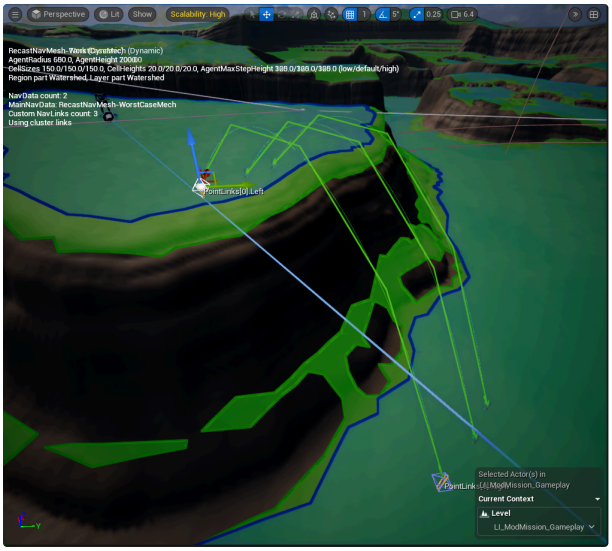
Settings for MoveToWaypoint



Unit path setup to move towards the ledge

Drag a **KelJumpJetNavLinkProxy** into your level. It has a **Left and Right handle** to manipulate. When you drag it in, it should sit quite **flush** with the **Landscape**.

We should be able to click on one of the handles and move it around. Move the handle to the surface that the mech comes from, the ledge in my example. Make sure that it **intersects** with the **landscape mesh**, so it can influence the **NavMesh**. Put it in the **unit's path**, so it will hit it when it moves to the destination. I usually put three **KelJumpJetNavLinkProxy's** there, so it hits it even when the unit's path slightly gets adjusted. Use **KelBuildMap** to build your **navigation** and then the **KelJumpJetNavLinkProxy** should look like this when you **press P**.



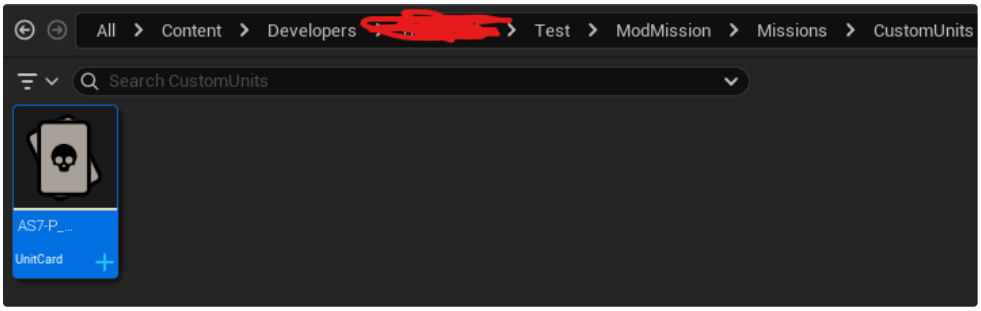
Active KelJumpJetNavLinkProxy

Custom unitcards

I recommend creating a separate folder for your custom units in your Mission folder.

Folder for units specific to this mission

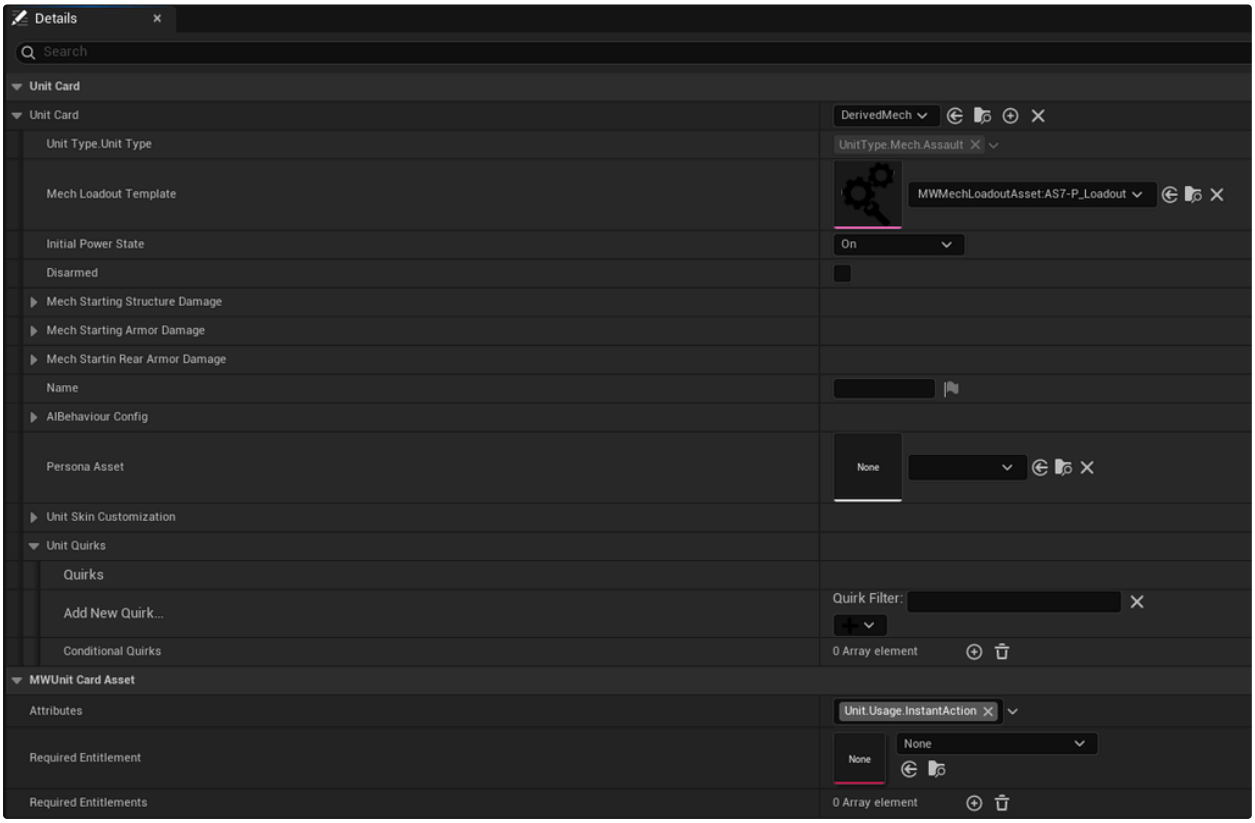
Copy any **Unitcard** that you want to customize into that folder.



Copied unitcard to use as a Boss unit

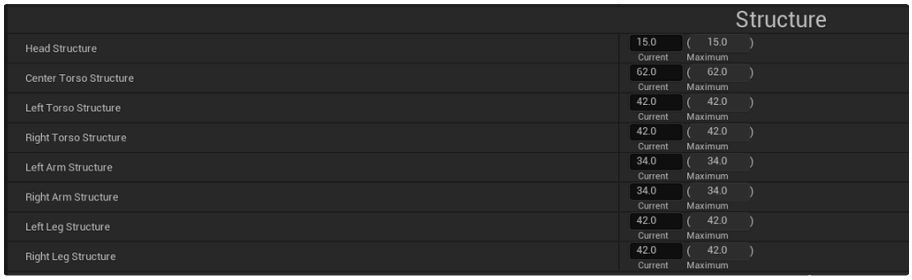
You can **customize** the unit to be what you need for your level. I'll go through any settings you can change. Anything that isn't mentioned, shouldn't be touched.

- Mech Loadout Template (Copy this file and change it to the loadout that you want)
- Mech Starting Structure Damage (Start the Mech with internal structure damage)
- Mech Starting Armor Damage (Start with the Mech having damaged armor)
- Mech Startin Rear Armor Damage (Start with the Mech having rear armore damage)
- AIBehaviour Config (This file can also be copied and adjusted to make it specific to this unit, variables such as health and accuracy are available in there)
- Persona Asset (Pilot that speaks in conversations if that's setup for the character)
- Unit Skin Customization (Can change skin or colors of the Mech)
- Unit Quirks (Quirks can make your unit much stronger or weaker, double the damage it inflicts for example)



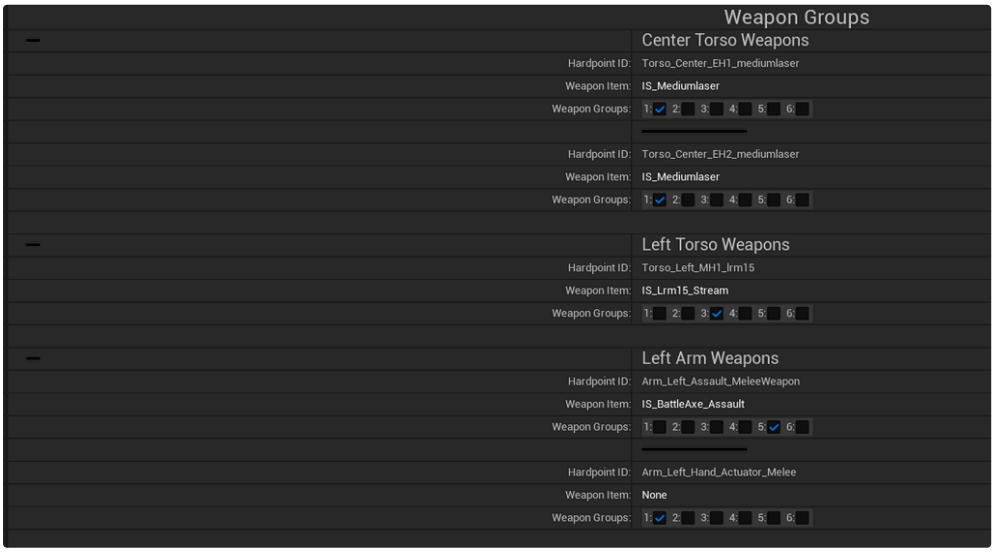
UnitCard settings

In the **Mech Loadout**, there are settings for structure health, armor and weapons.

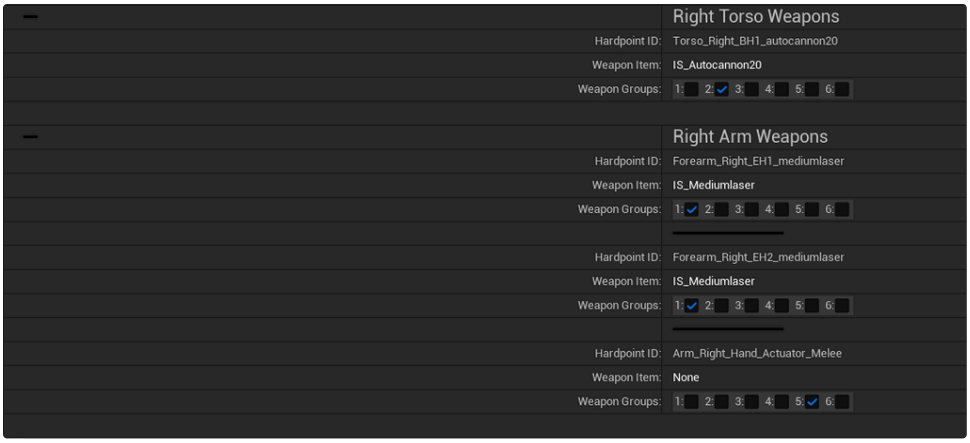


Loadout Structure settings

Loadout Armor settings



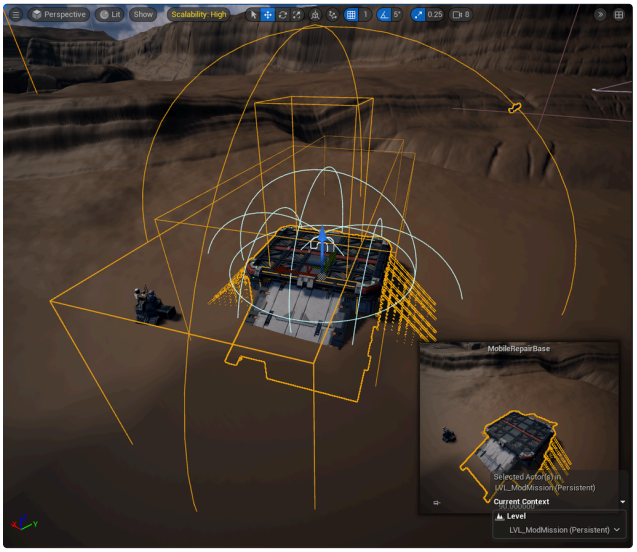
Weapon group settings



More Weapon group settings

Repair bays

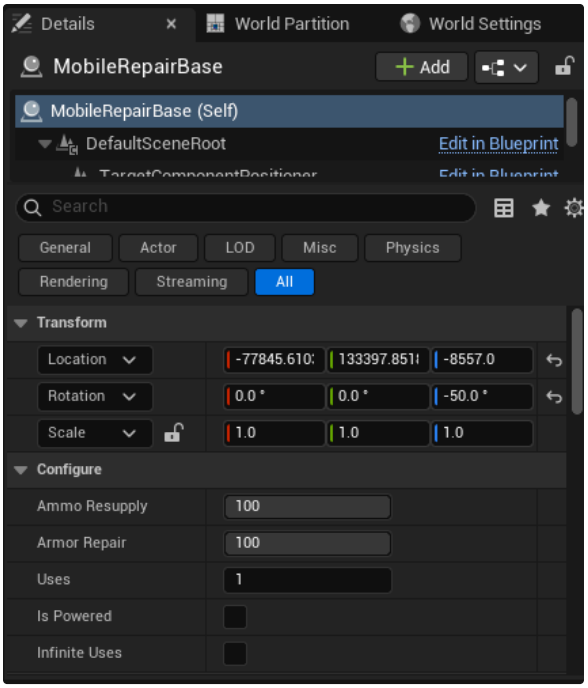
Sometimes we want to repair a mech mid mission. We can do that with a **MobileRepairBase** actor. Drag one into your level and set **Is Powered** to **true** to make it work.



MobileRepairBase

There are some settings you can change to make it function as you like.

- Ammo Resupply (Percentage of ammo resupply)
- Armor Repair (Percentage of armor repair)
- Uses (Number of times this repair bay can be used)
- Is Powered (Active or Inactive state)
- Infinite Uses (Repair bay has inifinite uses or not)

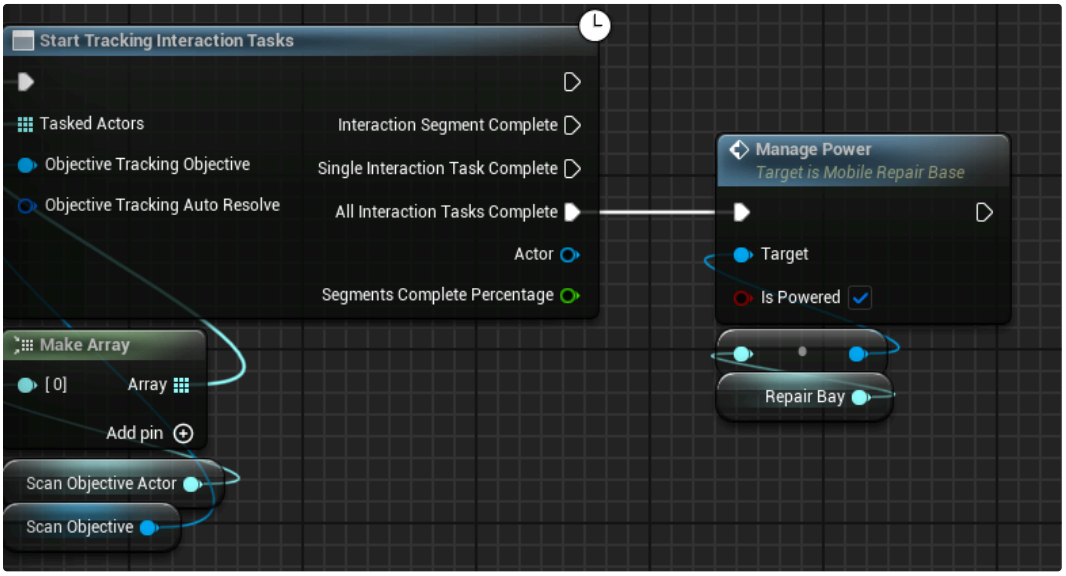


MobileRepairBase settings

You can also change settings in the MissionScript, but all you probably need is know how to activate and deactivate it. Start by creating a variable of type **MobileRepairBase** and make it a **SoftObjectReference**.

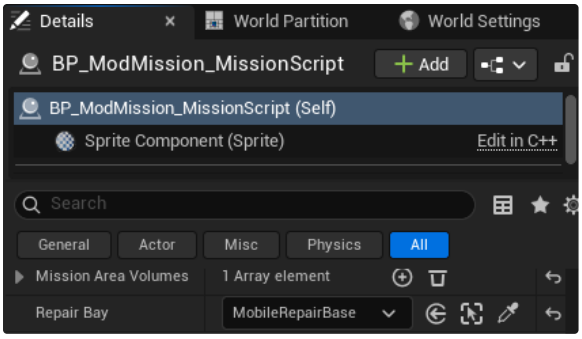
MobileRepairBase variable

From the variable, call **Manage Power** and set **Is Powered** to true or false. Depending if you want to activate or deactivate it. In my example, I scan an actor and it activates the Repair bay.



Scan an actor to activate repair bay

Assign the MobileRepairBase to your MissionScript.

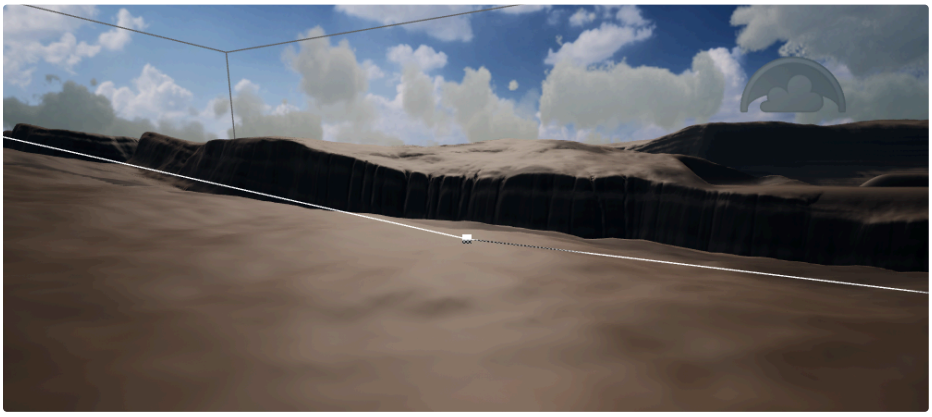


MobileRepairBase assigned to MissionScript

Out of bounds system (OOB)

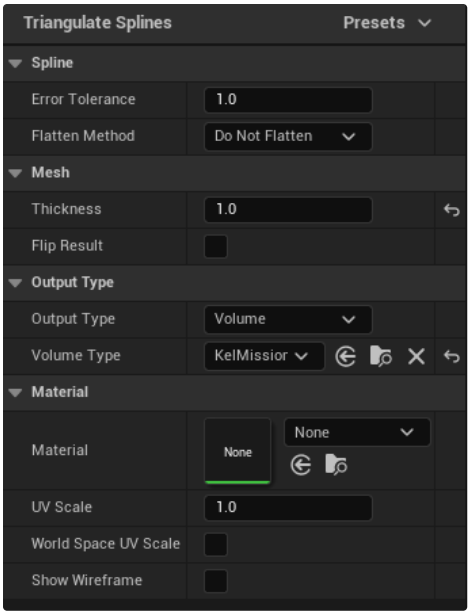
Create a **spline** that covers the whole play area. Make sure that the spline sits **flush** with the **landscape mesh** and is set to **Loop**.

Spline covering gameplay space



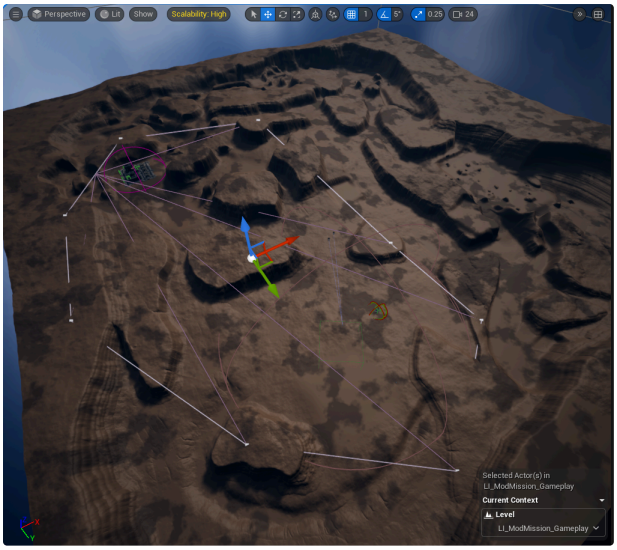
Every spline point is flush against the landscape

While the spline is selected, go to **MeshSplines**. Change the **Thickness** to **1** and **Output Type** to **Volume**. Then **Volume Type** to **KelMissionAreaVolume**.

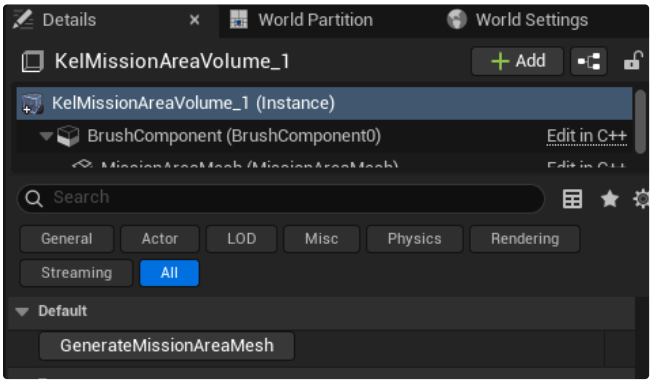


Settings for OOB Mesh

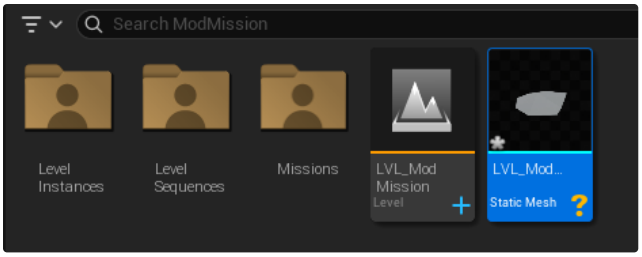
Click accept and it should create the **KelMissionAreaVolume**.



Make sure it's selected in the **Outliner** and click on **GenerateMissionAreaMesh** to create a **Mesh Actor** in the root of the mission folder. **Save** that Mesh.

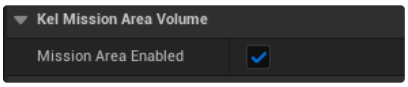


Button to create KelMissionArea Mesh



Mesh will be in your Mission root folder

You can create more than one and activate them when needed to open up more Gameplay space for example. Just make sure that the first active **KelMissionAreaVolume** is **Enabled** and the other ones **Disabled**.

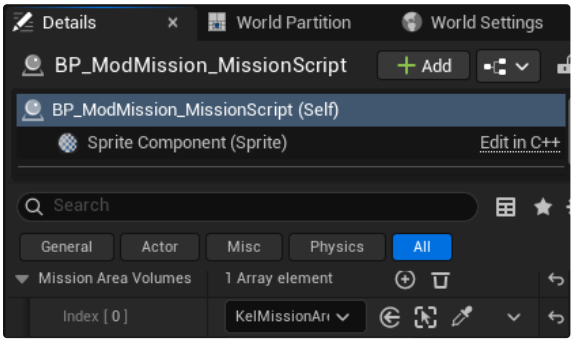


KelMissionAreaVolume Enabled

To change which one is active during runtime, you can Enable and Disable them as needed. One should be active at all times, so enable the new one before you disable the old one with **Enable Mission Area**.

Enabling new MissionAreaVolume

Assign the KelMissionAreaVolume to your MissionScript.

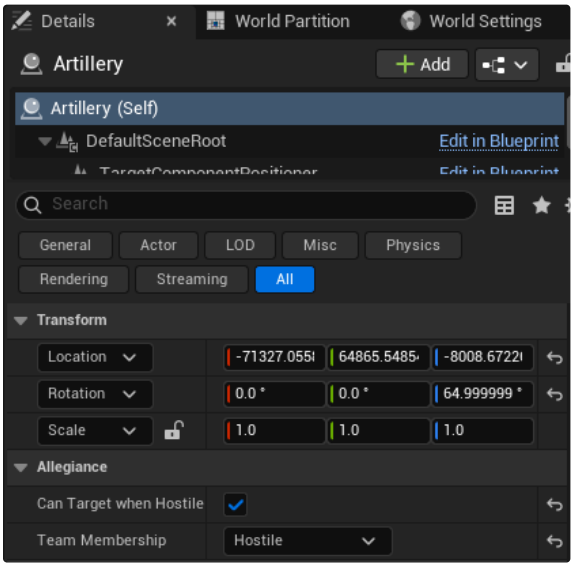


KelMissionAreaVolume assigned to MissionScript

Artillery

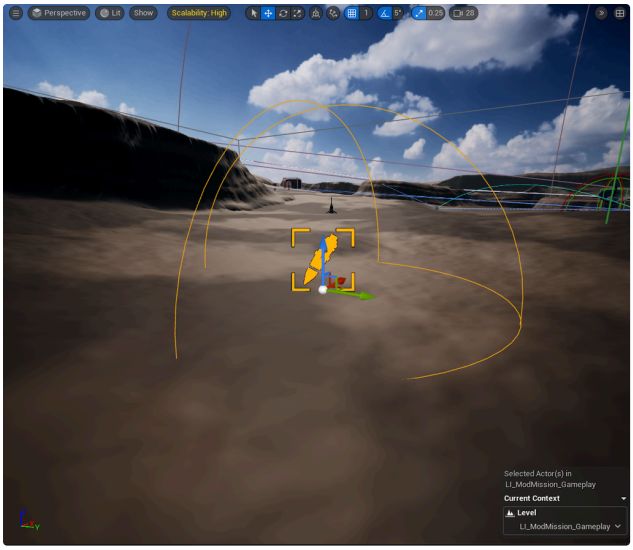
Start by putting an **Artillery** in your level and setting that up. Only settings to worry about are in **Allegiance**. Set it to **Hostile** and set **Can Target** when Hostile to true.

Artillery



Artillery settings

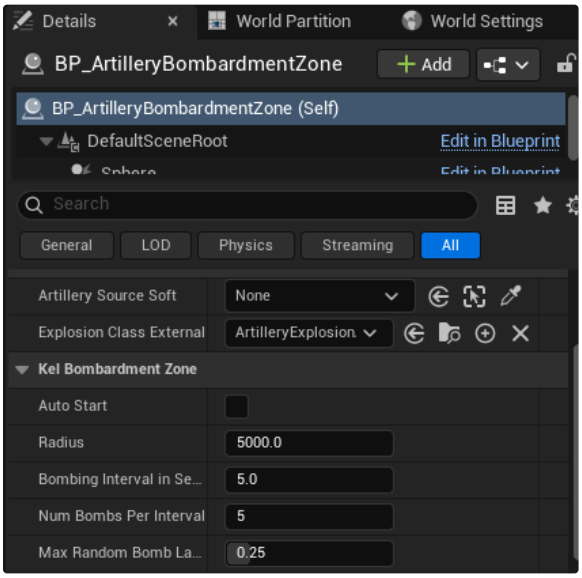
We're going to put in a zone where the **Artillery** will be firing next. Put a **BP_ArtilleryBombardmentZone** in your level where you want it to fire.



BP_ArtilleryBombardmentZone

To make it function, Assign the Artillery in your level to **Artillery Source Soft** and set **Auto Start** to **true**. There are also settings that you can change and you can activate it in script when you need it.

- Artillery Source Soft (Artillery firing, when Artillery is destroyed, firing will stop)
- Explosion Class External (The actual explosion when it hits, don't need to change this)
- Auto Start (Will start firing automatically when mission starts)
- Radius (Radius of where explosions can hit)
- Bombing Interval in Seconds (Amount of seconds between each explosion interval)
- Num Bombs Per Interval (Number of explosions per interval)
- Max Random Bomb Launch Delay (Random delay between each explosion in an interval)

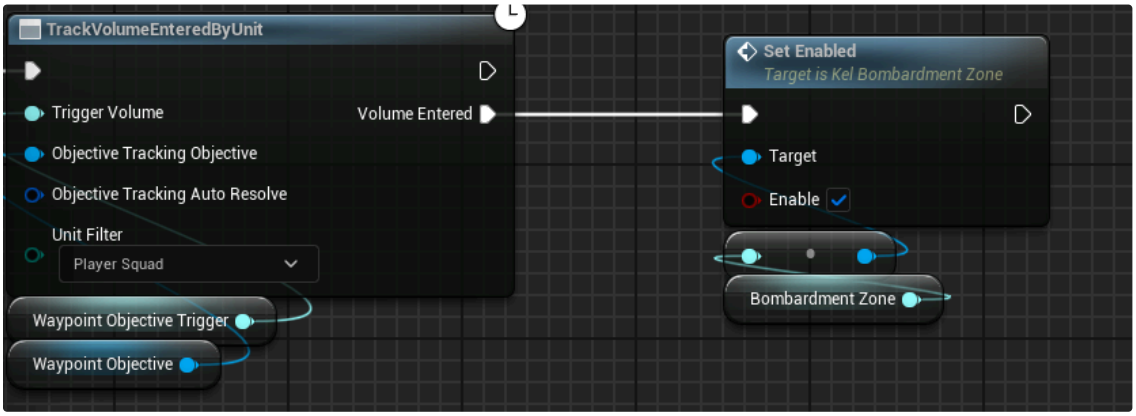


BP_ArtilleryBombardmentZone settings

To activate it in MissionScript, create a variable for the Bombardment zone of type **BP_ArtilleryBombardmentZone** as a **SoftObjectReference**.

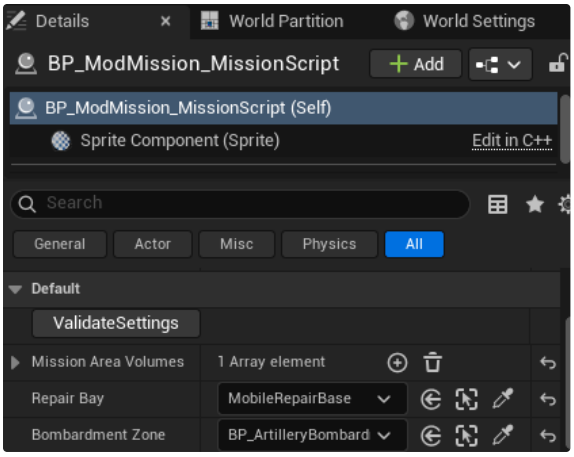
BP_ArtilleryBombardmentZone variable

Drag it into the Event graph and call **Set Enabled** from it.



Enable BP_ArtilleryBombardmentZone in MissionScript

Assign the BP_ArtilleryBombardmentZone to the MissionScript.



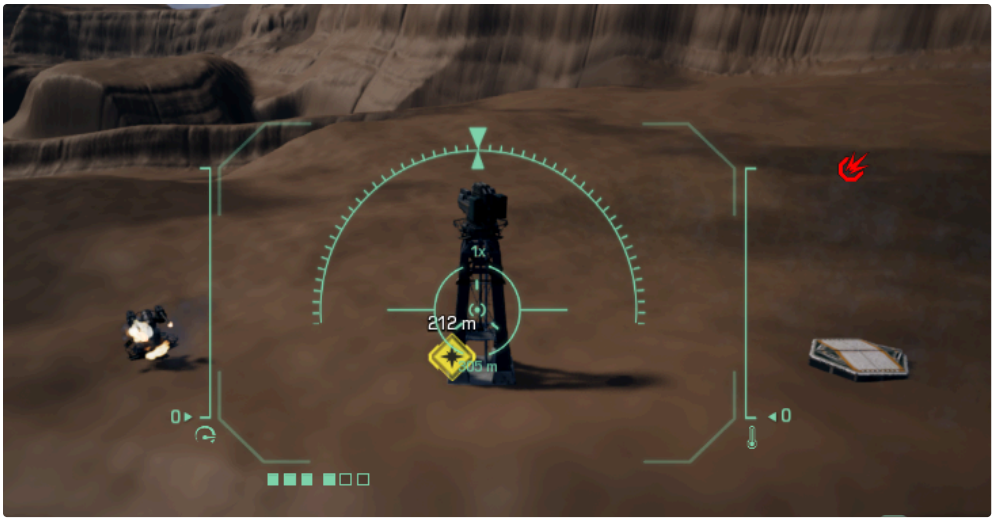
Assigned BP_ArtilleryBombardmentZone to MissionScript

Turrets

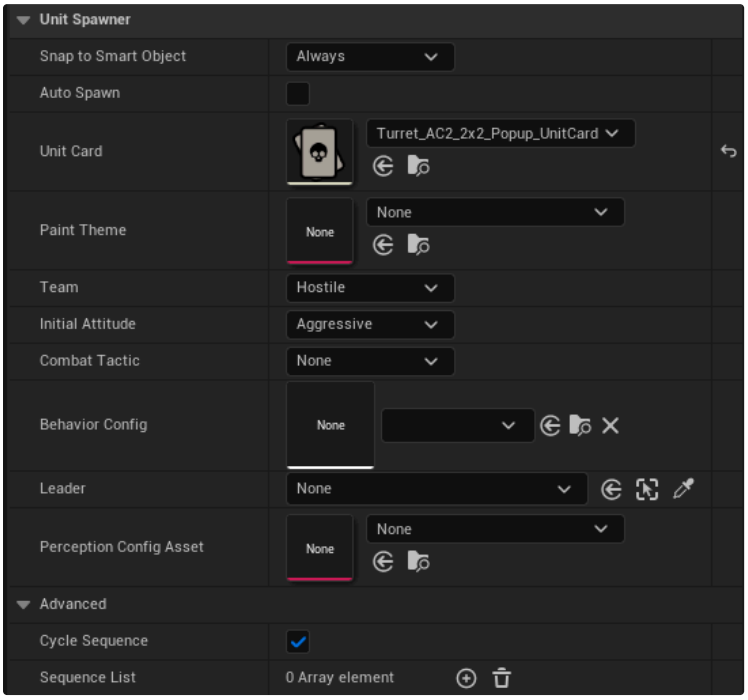
Turret variants

We've got many different types of turrets and I recommend to try them all out to create the gameplay that you like. I'll go through what different setups we have for them.

- Regular Turret (Place in a level and it fires what type of weapon it has)
- Turret Tower (Same as a regular turret, but on a tower)
- Turret Popup (Turrets that can pop out of the ground or walls when a hostile is close enough)



The three different turret variations



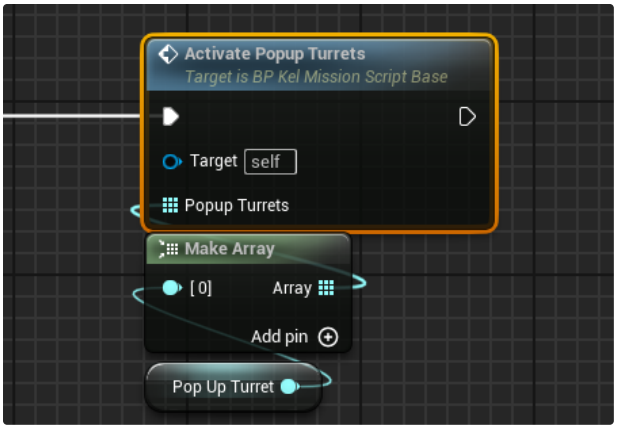
Same as other UnitSpawners

Popup Turrets in MissionScript

Popup Turret Unitcards that are named **Triggered**, can be activated in the MissionScript. Create a variable of type **KelUnitSpawner** and make it a **SoftObjectReference** that references the Popup Turret Triggered Unitcard.

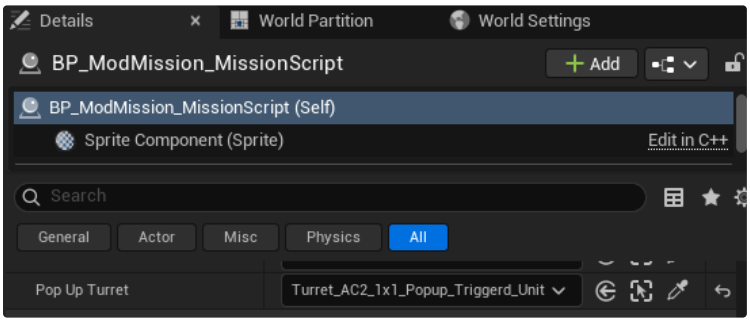
Popup turret triggered unitcard

Call **Activate Popup Turrets** in the Eventgraph



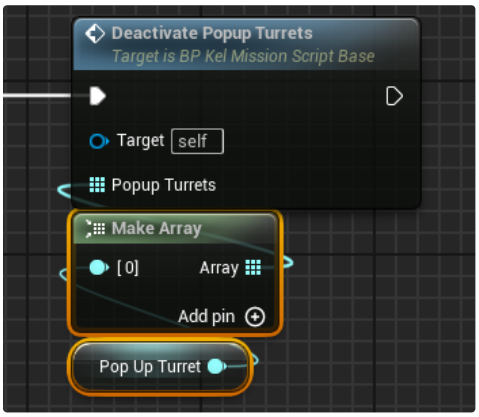
Popup turrets activation in MissionScript

Assign the Popup turret triggered unitcard to the MissionScript.



Popup Turret Triggered unitcard assigned to MissionScript.

You can also deactivate the PopupTurrets by calling **Deactivate Popup Turrets** in the MissionScript.



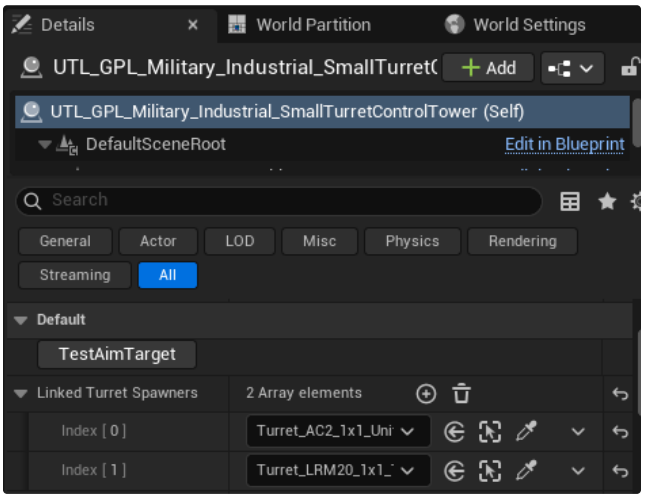
Deactivate Popup Turrets

Capture turrets

Players can turn hostile turrets into friendly turrets. Start by dragging a **UTL_GPL_Military_Industrial_SmallTurretControlTower** in your level. To prevent it from being destroyed, I usually put a **BP_TurretControlTower_Bunker** on top of it.

Turret control tower with bunker

You should have some turrets in your level by now and if not, drag a couple of regular turrets in and spawn them. You can assign them to **Linked Turret Spawners** in **UTL_GPL_Military_Industrial_SmallTurretControlTower**.

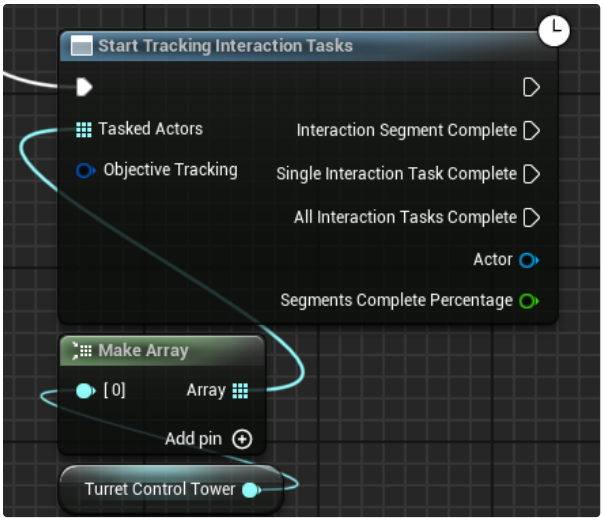


Assigned turrets to UTL_GPL_Military_Industrial_SmallTurretControlTower

Create a variable for the UTL_GPL_Military_Industrial_SmallTurretControlTower in the MissionScript as type **Actor** and make it a **SoftObjectReference**.

Variable for UTL_GPL_Military_Industrial_SmallTurretControlTower

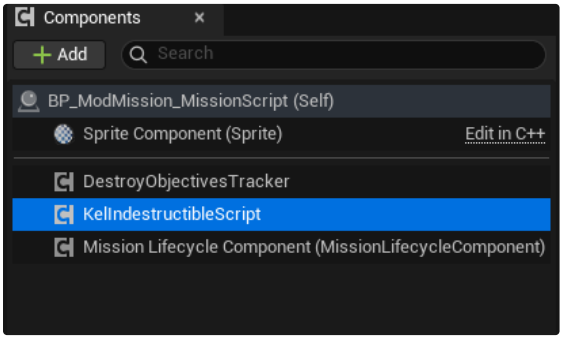
Call **Start Tracking Interaction Tasks** and plug in the Turret control tower to **Tasked Actors**.



Setup for capturing Turrets

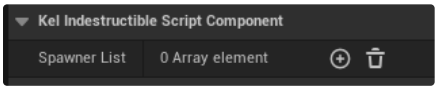
Indestructible units

In the MissionScript, you can make units indestructible with a **KelIndestructibleScript Component**. Create the component.



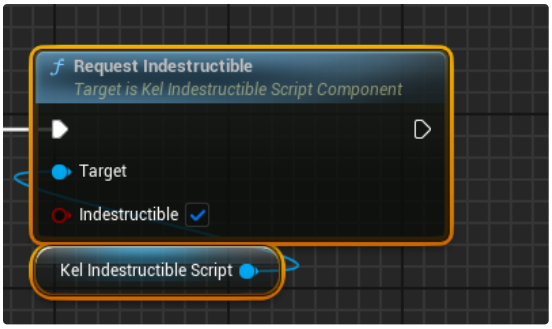
KelIndestructibleScript

In the Details panel, you can tell it which units should be indestructible. Add them to the **Spawner List**.



List to tell what units can be indestructible

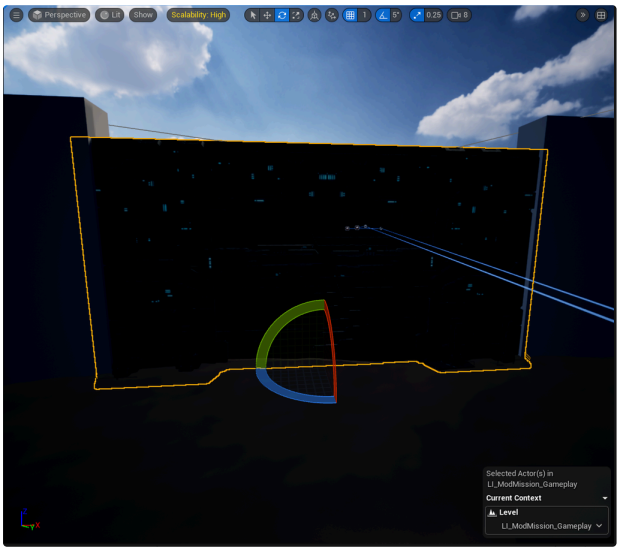
Drag KelIndestructibleScript in the Eventgraph and call **Request Indestructible** from it.



Set to true to make units indestructible

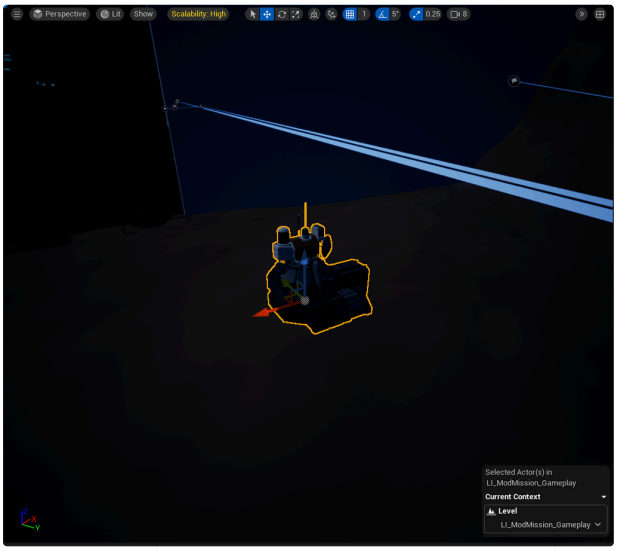
Gates

We can open and close gates. There are a couple to choose from, but I'll be using **BP_Space_Station_Gate** for this example. Drag one into your level. I've also grabbed a cube for each side to block the path and make it actually function as a Gate.

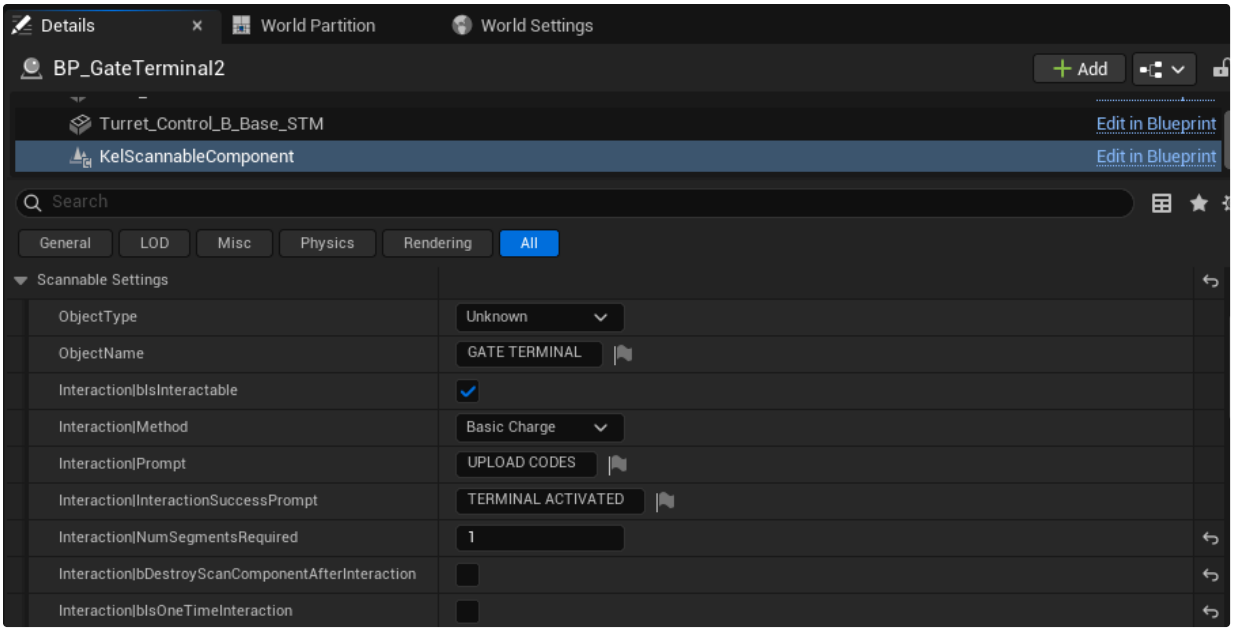


Gate

I've also added a **BP_Gate_Terminal** to scan when opening and closing the Gate. Not necessary to do it this way, but makes sense for the example. I also changed some settings in the terminal to open and close the Gate. In the **KelScannableComponent**, set **Interaction|NumSectionRequired** to 1, so it's easier to use. Set **Interaction|bDestroyScanComponentAfterInteraction** and **Interaction|bIsOneTimeInteraction** to **false**.



Gate Terminal to scan

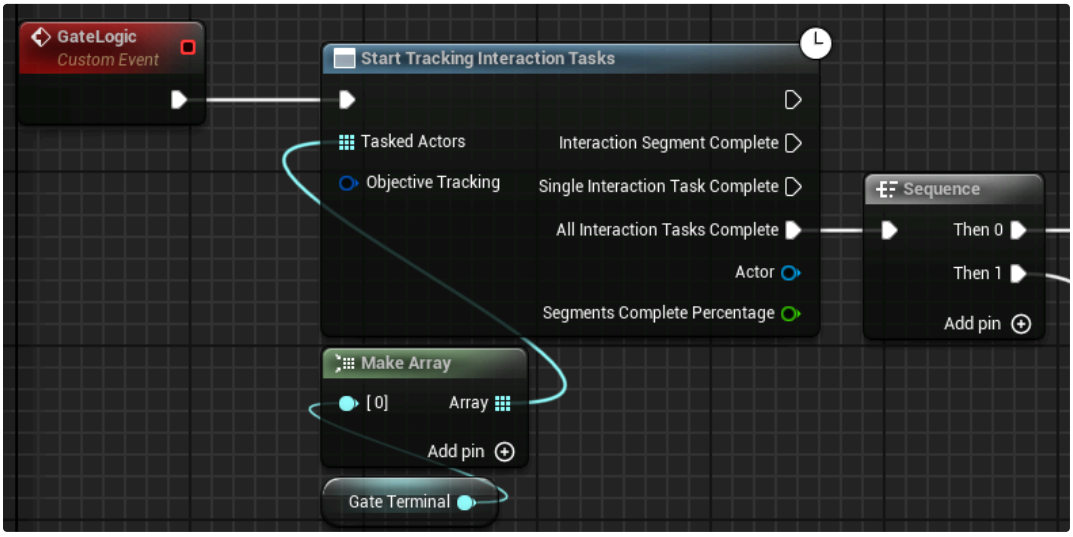


Gate Terminal KelScannableComponent settings for following example

Create two **SoftObjectReference** variables in the MissionScript, one for the Gate Terminal of type **Actor** and one for the Gate of type **BP_Space_Station_Gate**.

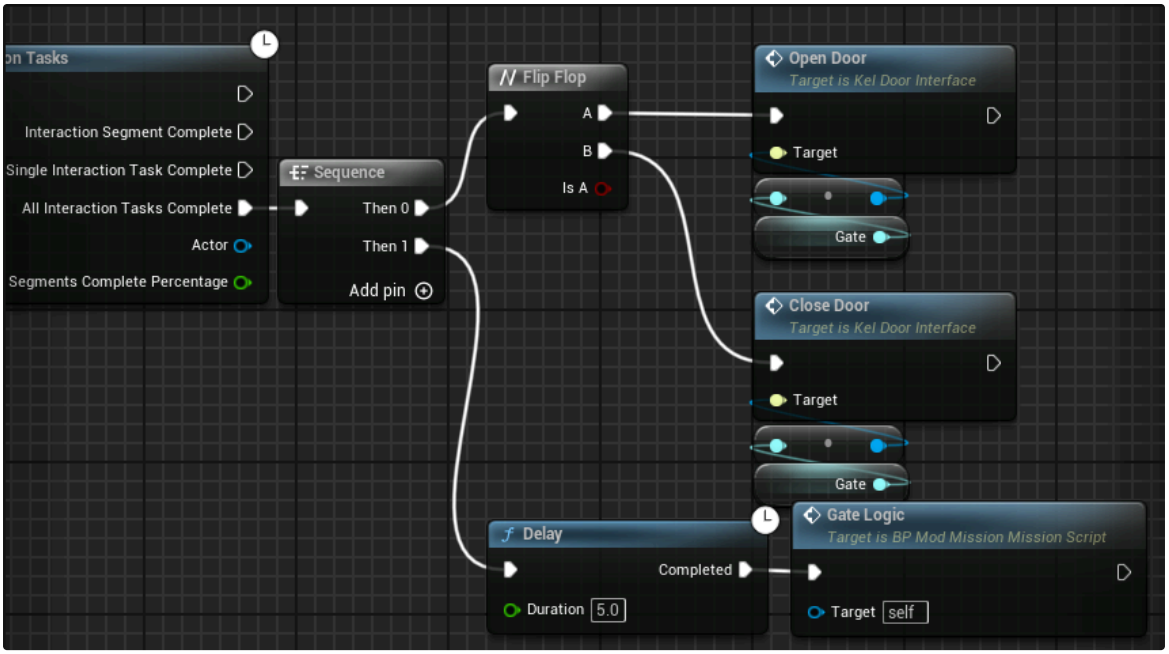
Variables for opening and closing gate example

Create an event and call **Start Tracking Interaction Tasks**. Plug in the Gate Terminal variable.



Interaction setup for scanning Gate Terminal

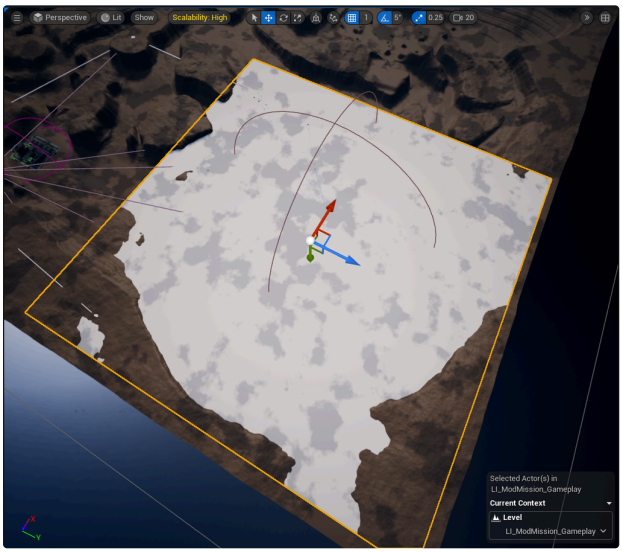
To open and close the Gate, I've added a **FlipFlop** to switch between events. Output A calls **Open Door** from the Gate and Output B calls **Close Door** from the Gate. The sequence is there, so I can reset the Interaction after 5 seconds, giving the Gate time to open and close.



Gate setup to Open and Close after scanning the Terminal

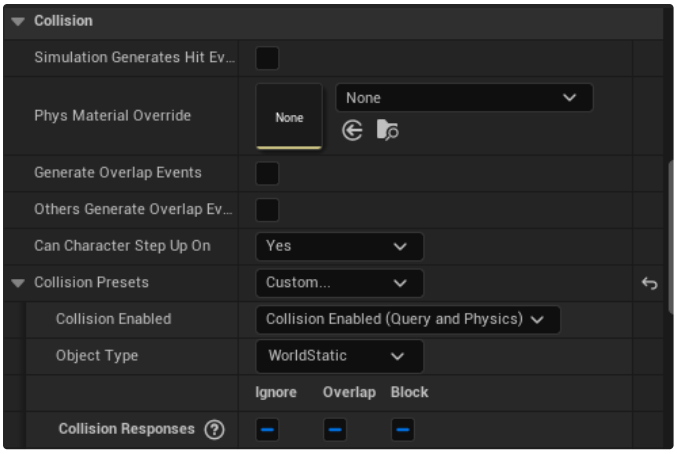
Hiding Mesh from Battlegrid

Some levels might have a roof. Could be a structure or maybe a cave. The roof will prevent the players to issue commands in the **Battlegrid**. We can exclude meshes, so the Battlegrid doesn't try to use it.

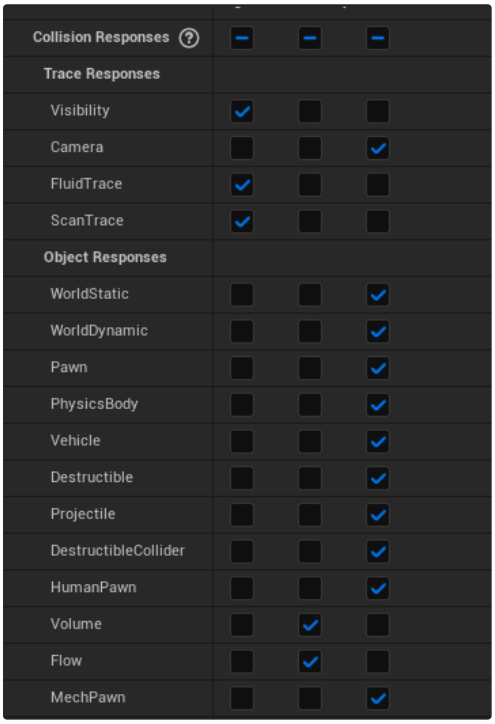


Roof

We have to change some collision settings to exclude it. Set **Collision Presets** to **Custom** and set **Visibility** to **Ignore**.



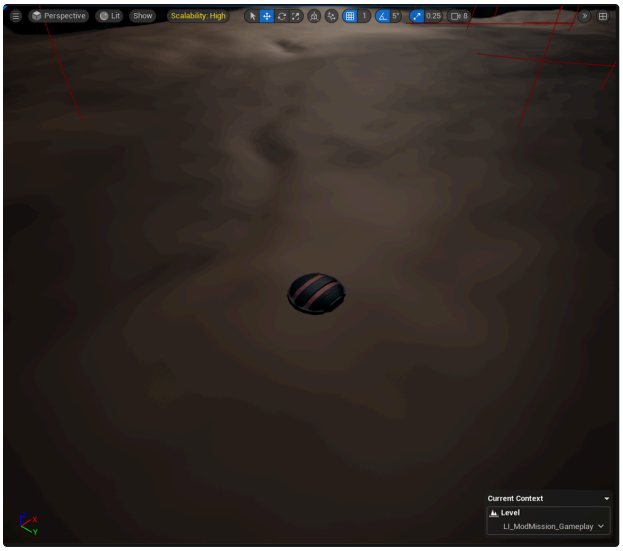
Collision with custom settings



Custom collision settings to exclude mesh from Battlegrid

Proximity Mines

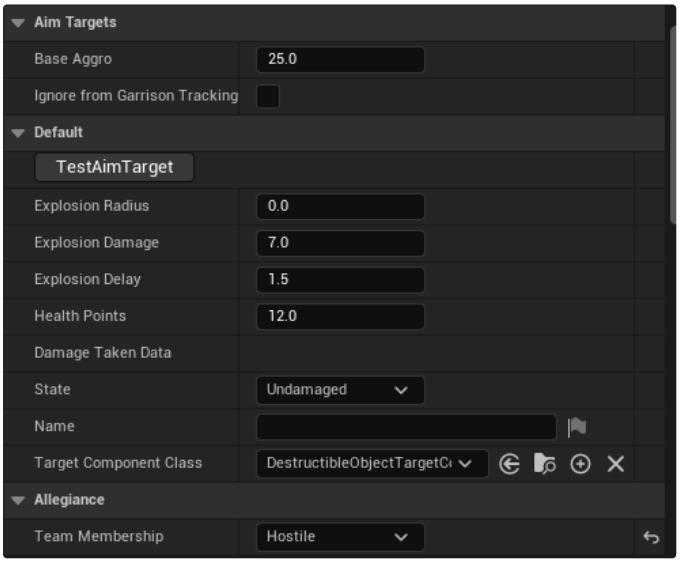
You can place **Proximity Mines** either Friendly or Hostile, depending on the gameplay you want.



Proximity Mine

Here are some settings that you can change and where you can set the team to be Hostile or Friendly. Hostile in my example. The settings that I haven't mentioned here, shouldn't be touched.

- Explosion Radius (Radius of how far the explosion will reach)
- Explosion Damage (How much damage the explosion will inflict)
- Explosion Delay (Amount of settings for the explosion to trigger after mine had been triggered)
- Health Points (Amount of damage required to destroy the mine)
- Team Membership (Set the mine to Hostile or Friendly here)

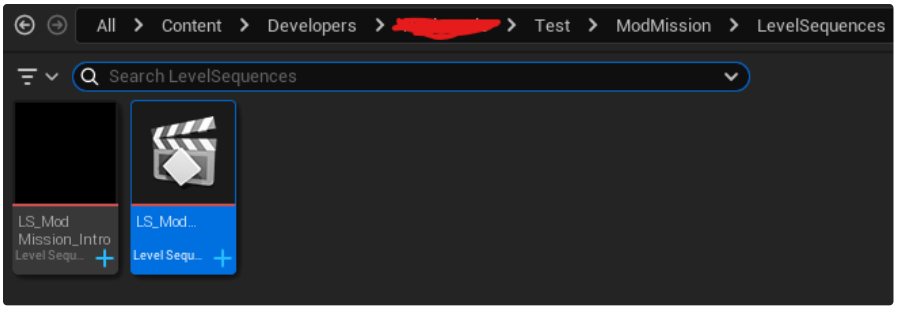


Proximity Mine settings

NIS (Level Sequences)

In game sequence

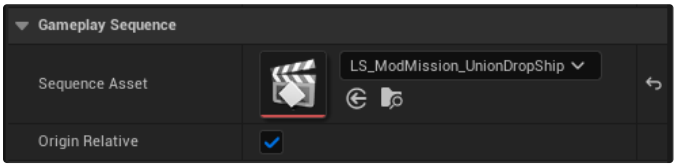
Create a **Level Sequence** and create in there what you would like to happen. I usually create a separate folder for my Level Sequences, so they're easy to find. For this example, I'm just going to copy having a Union ship land.



New Level Sequence in LevelSequence folder

Drag a **KelGameplaySequence** in your level. Select the Level Sequence you created in **Sequence Asset**.

KelGameplaySequence

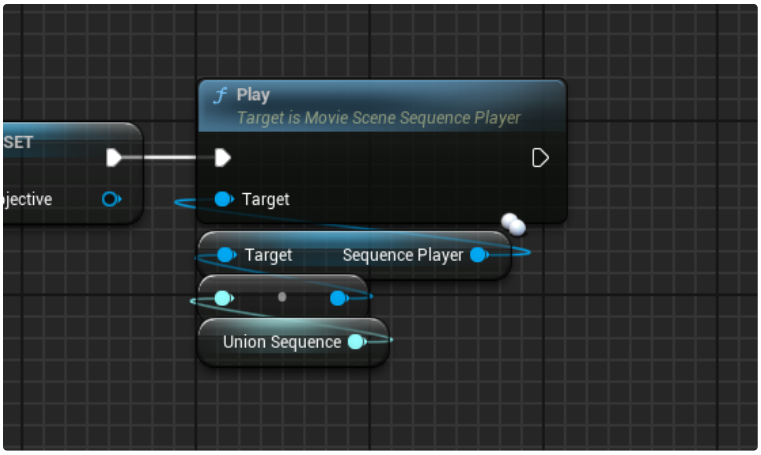


Assigned Level Sequence to Sequence Asset

Create a variable of type **KelGameplaySequence** as a **SoftObjectReference**.

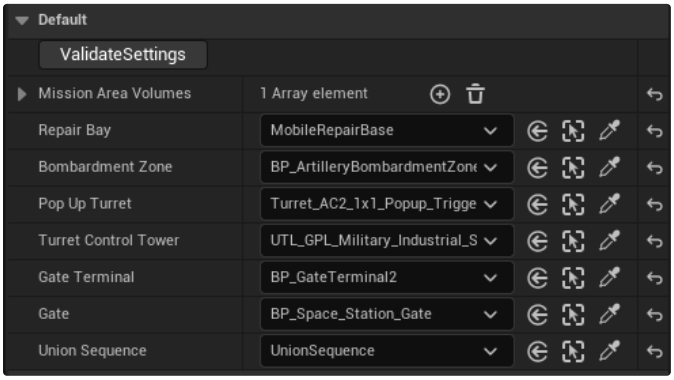
KelGameplaySequence variable

Get the **Sequence Player** from the variable and call **Play**.



Call Play from the Sequence Player

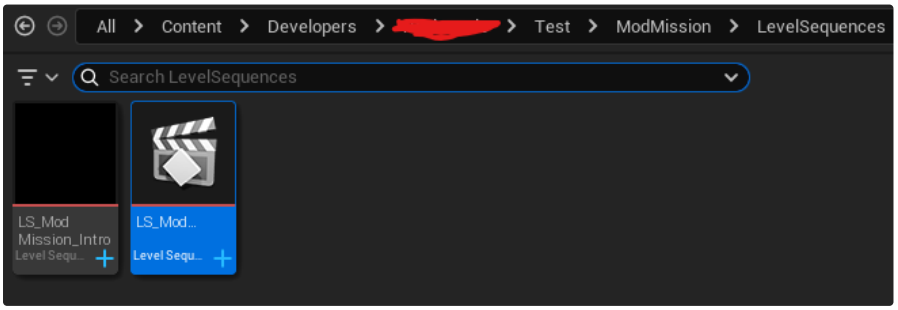
Assign the KelGameplaySequence to the MissionScript.



Assigned the KelGameplaySequence to the MissionScript

Cinematic sequence

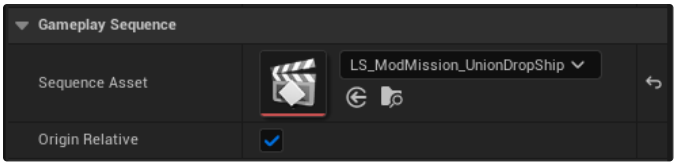
Create a **Level Sequence** and create in there what you would like to happen. I usually create a separate folder for my Level Sequences, so they're easy to find. For this example, I'm just going to copy having a Union ship land.



New Level Sequence in LevelSequence folder

Drag a **BP_CinematicSequence** in your level. Select the Level Sequence you created in **Sequence Asset**.

BP_CinematicSequence

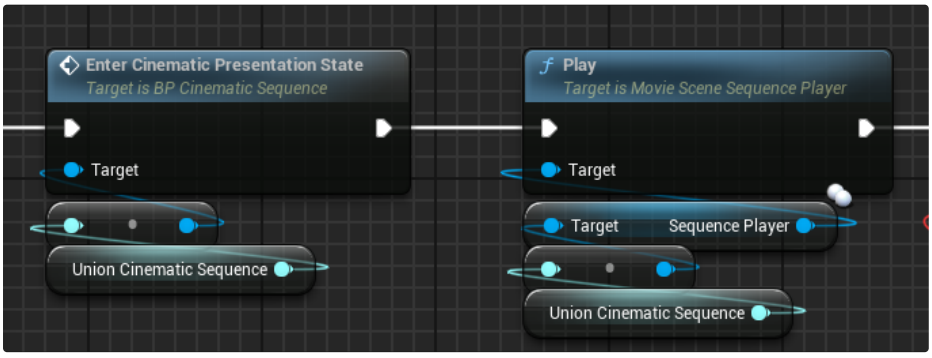


Assigned Level Sequence to Sequence Asset

Create a variable of type **BP_CinematicSequence** as a **SoftObjectReference**.

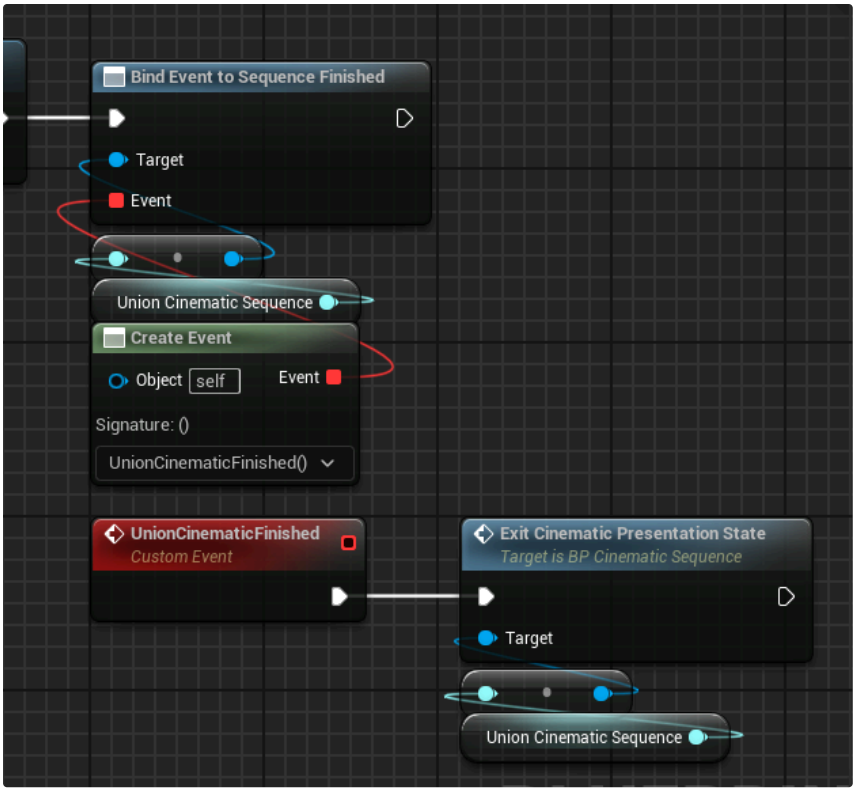
BP_CinematicSequence variable

Call **Enter Cinematic Presentation State** and **Play** from the **BP_CinematicSequence** variable.



Call Enter Cinematic Presentation State and Play

Bind an event to **Sequence Finished** and call **Exit Cinematic Presentation State** to continue gameplay.



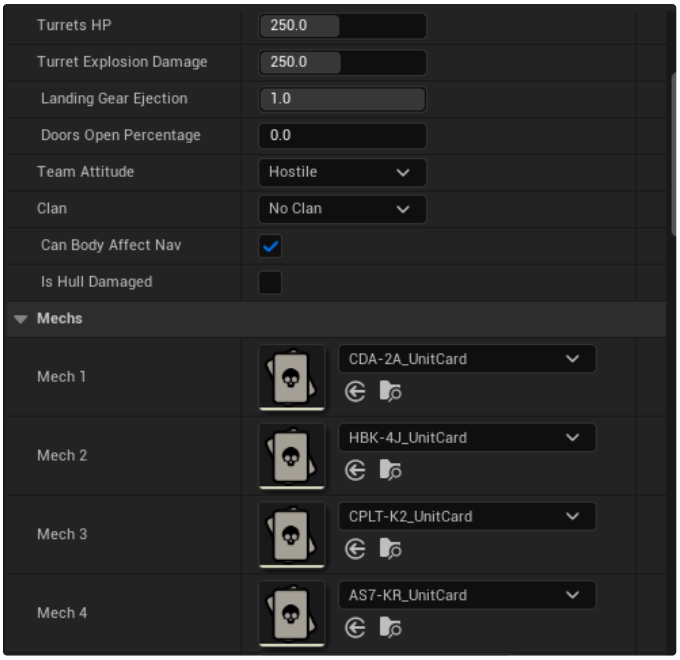
Exit Cinematic Presentation State to continue gameplay

Union ship Setup

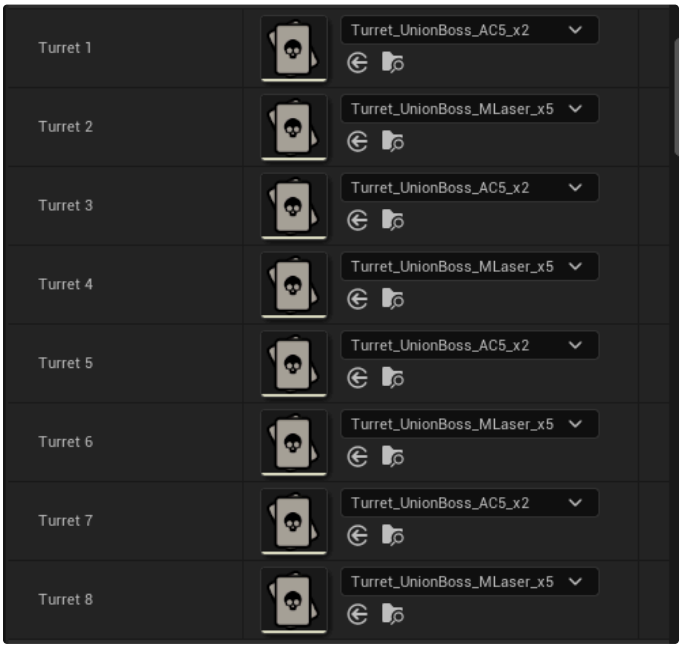
Drag a **BP_UnionBoss** into your level. It has some settings to adjust it to your gameplay. The settings that I won't mention aren't used to setup gameplay.

- Turrets HP (Amount of health for each Turret)
- Turret Explosion Damage (Amount of damage it inflicts to the Union when Turret is destroyed)
- Team Attitude (Change Union to Hostile or Friendly)
- Clan (Change Union Skin)
- Mech 1 to 4 (What Mechs it can spawn)
- Turrets 1 to 8 (Select what UnionBoss Turrets to spawn)

BP_UnionBoss



BP_UnionBoss settings

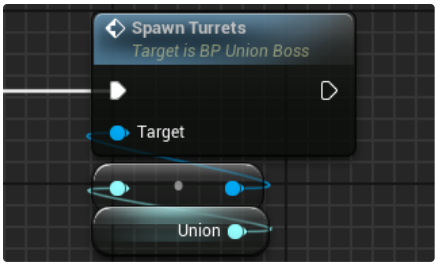


BP_UnionBoss Turrets

Create a **SoftObjectReference** variable of type **BP_UnionBoss**.

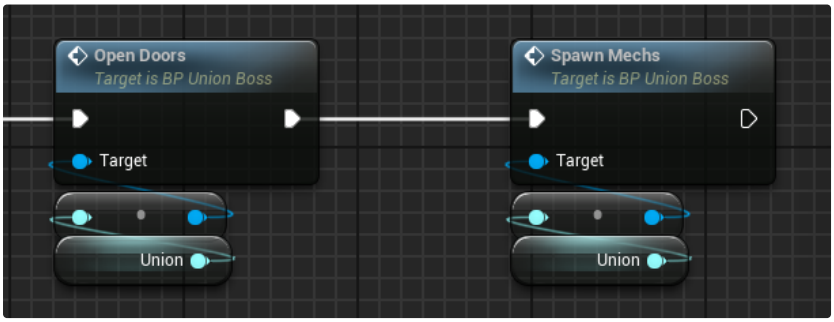
BP_UnionBoss variable

From the BP_UnionBoss variable, call **Spawn Turrets**.



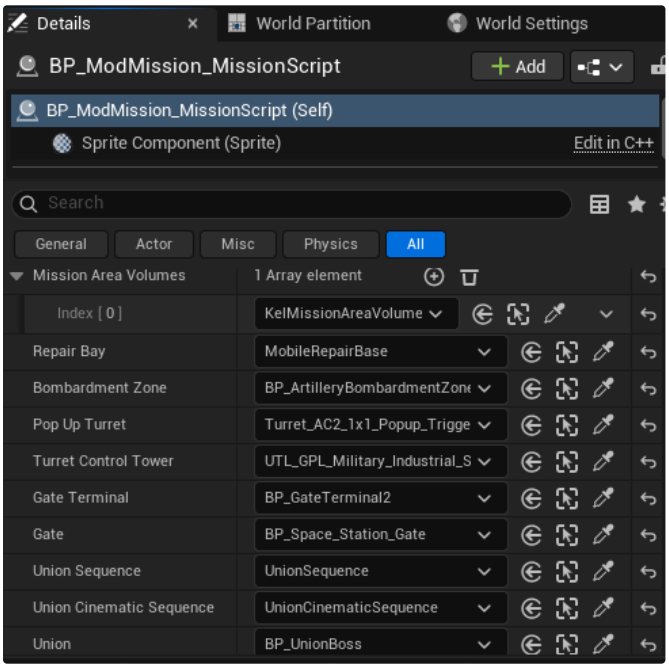
Call Spawn Turrets

You can also spawn Mechs from the Union. Call **Open Doors** and **Spawn Mechs** from the BP_UnionBoss variable.



Call Open Doors and Spawn Mechs

Assign the BP_UnionBoss to the MissionScript.



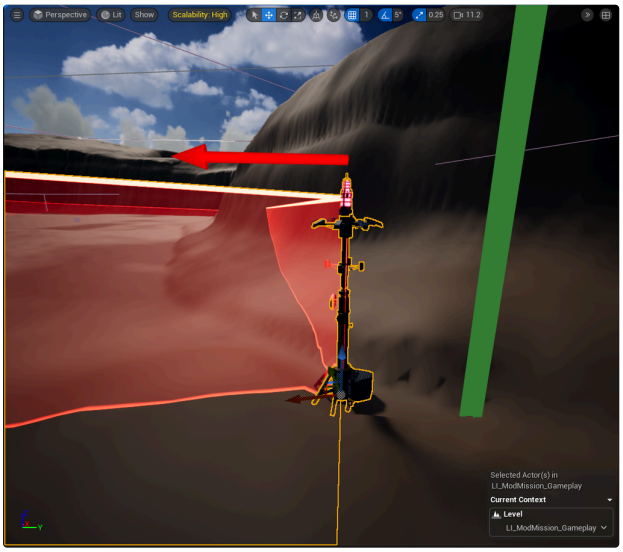
Assigned BP_UnionBoss to the MissionScript

Laser Fence

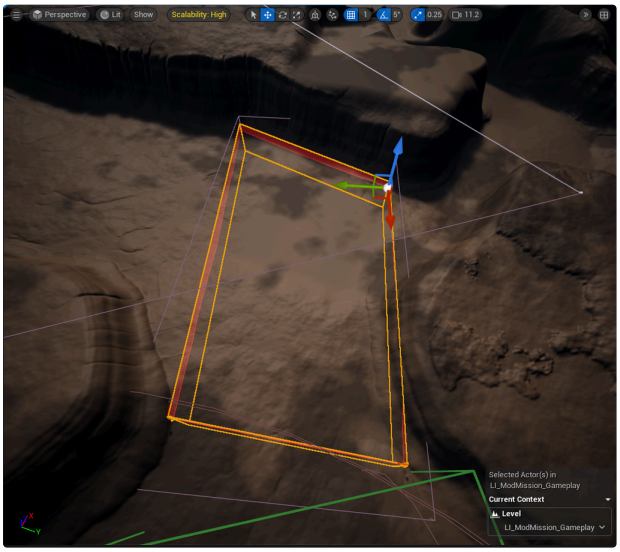
This will allow for triggering an alarm when a fence has been hit. Drag in a **BP_Scannable_TripwireBeacon** and copy that until they form the shape of the area that you want to guard. Then assign the **Connected Beacons** to them in order to form your shape. You can turn **Debug Show Connection** on to see the fence.

- Enabled (Fence is active and can trigger the alarm)
- Connected Beacons (Assign the next BP_Scannable_TripwireBeacon here)
- Regenerate Link (Fence stays active after it's triggered)

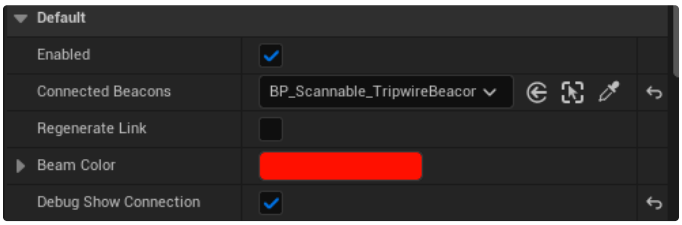
- Beam Color (Color that you want the fence to be)
- Debug Show Connection (Can see fence in editor)



BP_Scannable_TripwireBeacon

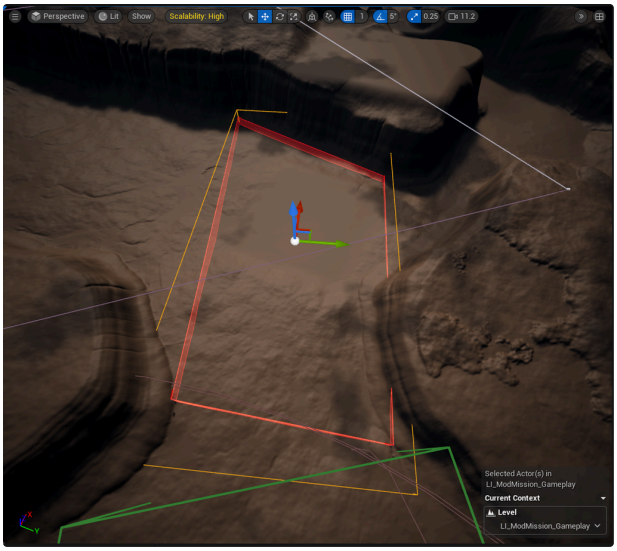


Copied BP_Scannable_TripwireBeacon until it forms the shape that I need

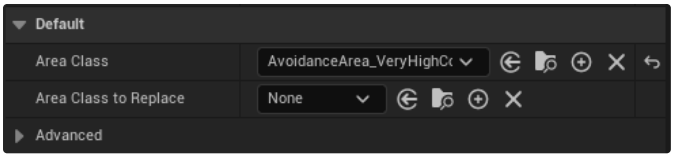


BP_Scannable_TripwireBeacon settings

Put a **NavModifierVolume** around the area that's guarded, so other AI doesn't trigger it. Set Area Class to **AvoidenceArea_VeryHighCost**.

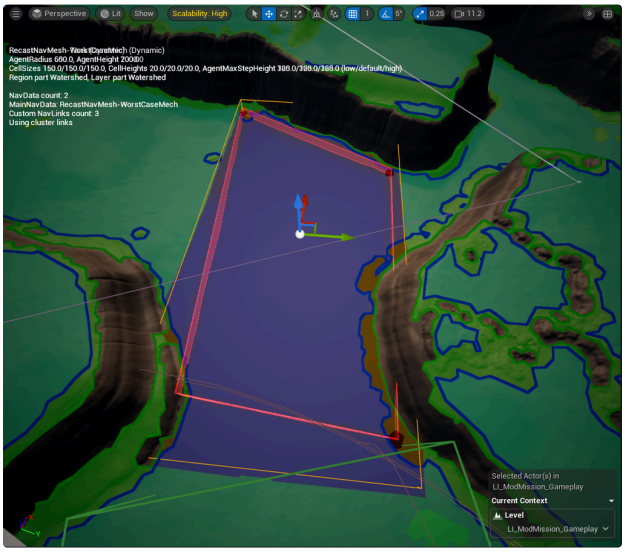


NavModifierVolume



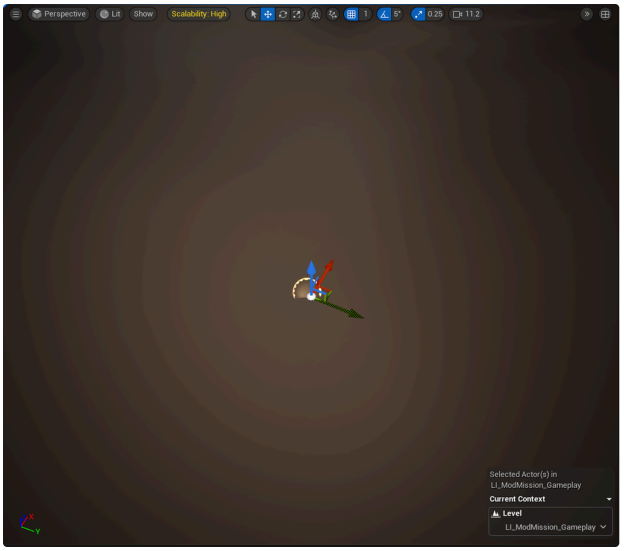
NavModifier settings

KelBuildMap to make sure the navigation will use your **NavModifierVolume**.

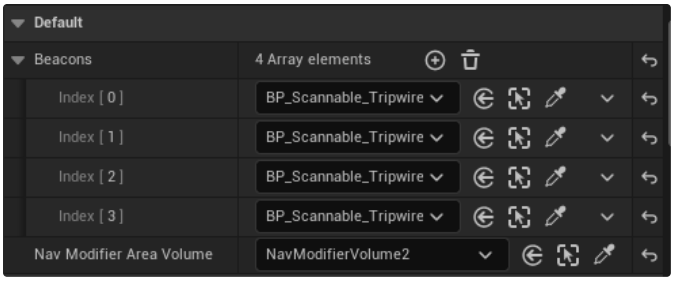


Navigation after KelBuildMap

Drag a **BP_AlarmTrigger_Manager** into your level. Assign the **BP_Scannable_TripwireBeacons** and the **NavModifierVolume**.



BP_AlarmTrigger_Manager

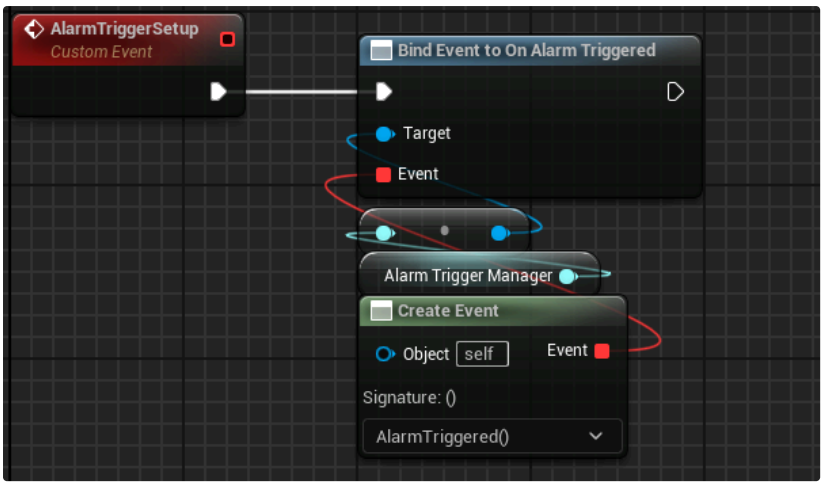


BP_AlarmTrigger_Manager settings

In the MissionScript, create a **SoftObjectReference** variable of type **BP_AlarmTrigger_Manager**.

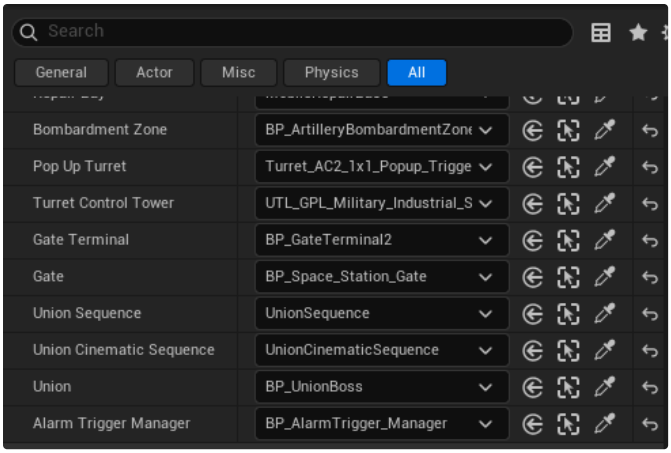
BP_AlarmTrigger_Manager variable

Bind an event to **On Alarm Triggered** and you can fire what logic you want happening after.



Bind event to On Alarm Triggered

Assign the BP_AlarmTrigger_Manager to the MissionScript.



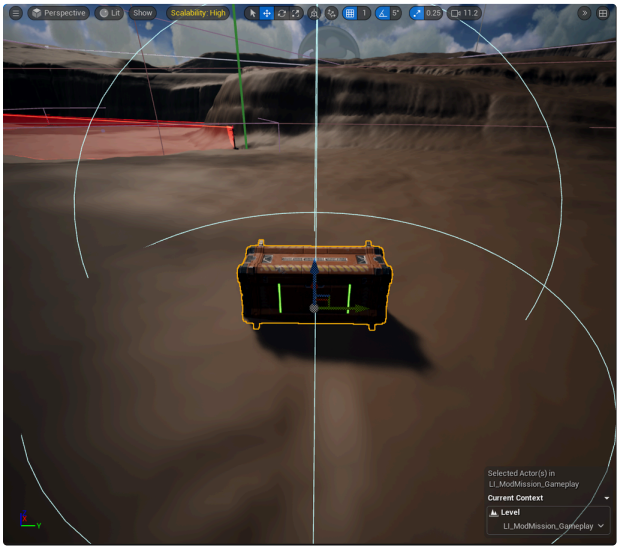
BP_AlarmTrigger_Manager assigned to MissionScript

Ammo/Treasure Crates

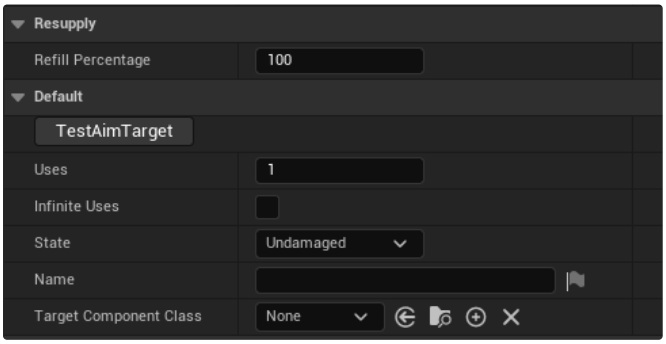
Ammo Crate

An Ammo Crate is called a **Resupply Actor**. Drag it in your level and it works. There are some settings that you can change.

- Refill Percentage (Amount of ammo to replenish)
- Uses (Amount of times this crate can be used)
- Infinite Uses (Check this box for infinite uses)

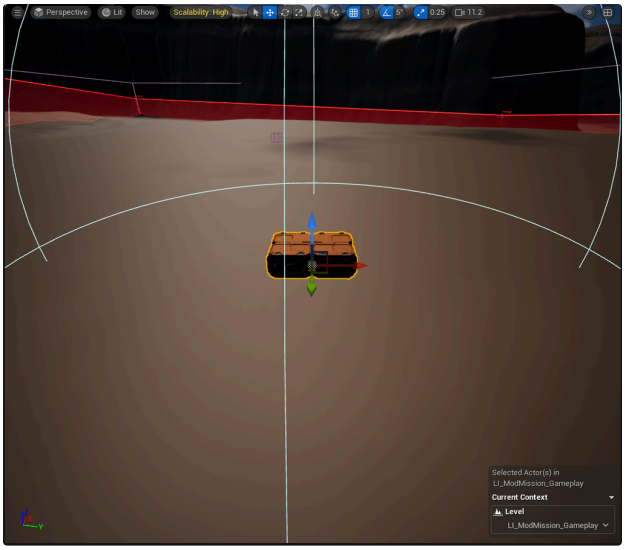


Resupply Actor



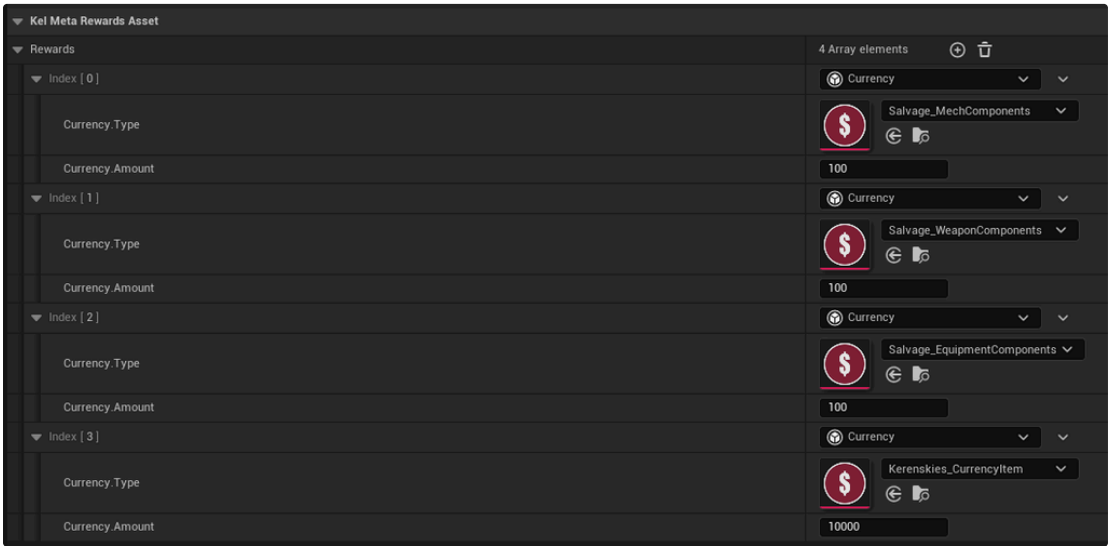
Treasure Crate

Drag a **TreasureActor** in and it should work. A separate actor tells it what treasure it contains.



Treasure Actor

You can create a **KelMetaRewards Asset** in your Mission folder. Inside this asset, you can add rewards for the player to receive when scanning a **TreasureActor**.



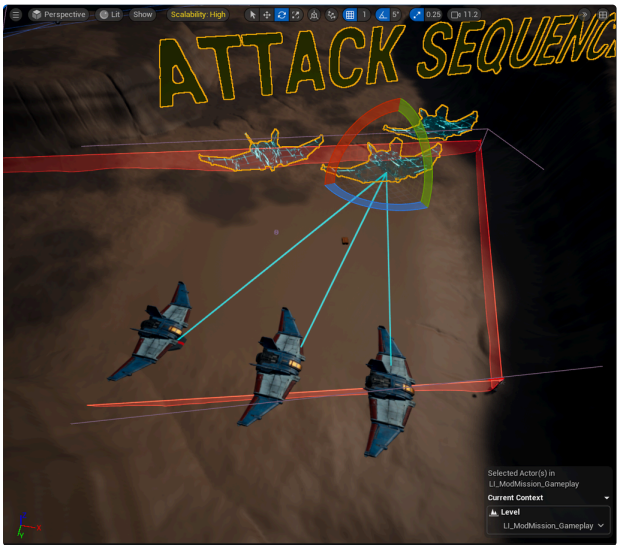
Reward example in KelMetaRewardsAsset

AeroSpaceFighterSequence

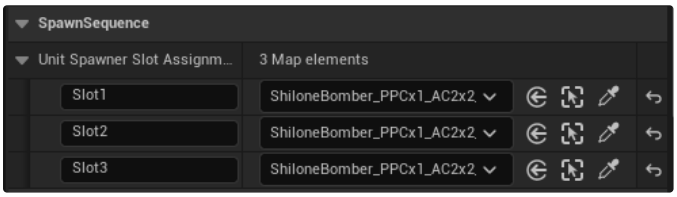
BP_AeroSpaceFighterAttackSequence

You can add some flying **Shilones** or **Jagatai** to the mission. Drag a **BP_AeroSpaceFighterAttackSequence** into your level. Assign **Shilone** or **Jagatai Unitcards** to Slot 1,2 and 3. There's a **Box** component attached to this actor. You can size the box to the size that you need. When a player is inside, it will **Play** the sequence for the Aerospacefighters.

BP_AeroSpaceFighterAttackSequence



Shilone Unitcards assigned



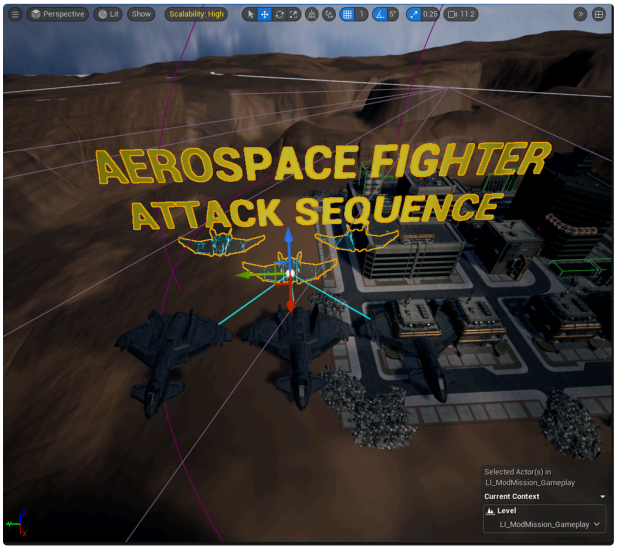
Each Unitcard assigned to a slot

BP_AeroSpaceFighterAttackSequence_Triggerable

Use this AeroSpaceFighter Actor to trigger in the MissionScript when needed. Drag a **BP_AeroSpaceFighterAttackSequence_Triggerable** into your level.

BP_AeroSpaceFighterAttackSequence_Triggerable

Assign **Shilone** or **Jagatai Unitcards** to **BP_AeroSpaceFighterAttackSequence_Triggerable**.



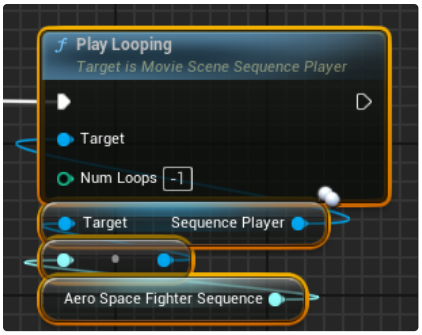
Assigned Unitcards to
BP_AeroSpaceFighterAttackSequence_Triggerable

Assigned Unitcards to a Slot

Create a **SoftObjectReference** variable of type **BP_AeroSpaceFighterAttackSequence_Triggerable** in the MissionScript.

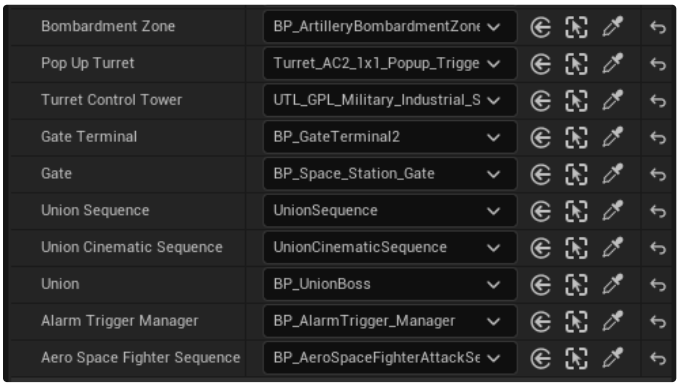
BP_AeroSpaceFighterAttackSequence_Triggerable
variable

Call **Play Looping** from the **Sequence Player** to trigger when you need it.



Call Play Looping from Sequence Player

Assign BP_AeroSpaceFighterAttackSequence_Triggerable to the MissionScript.

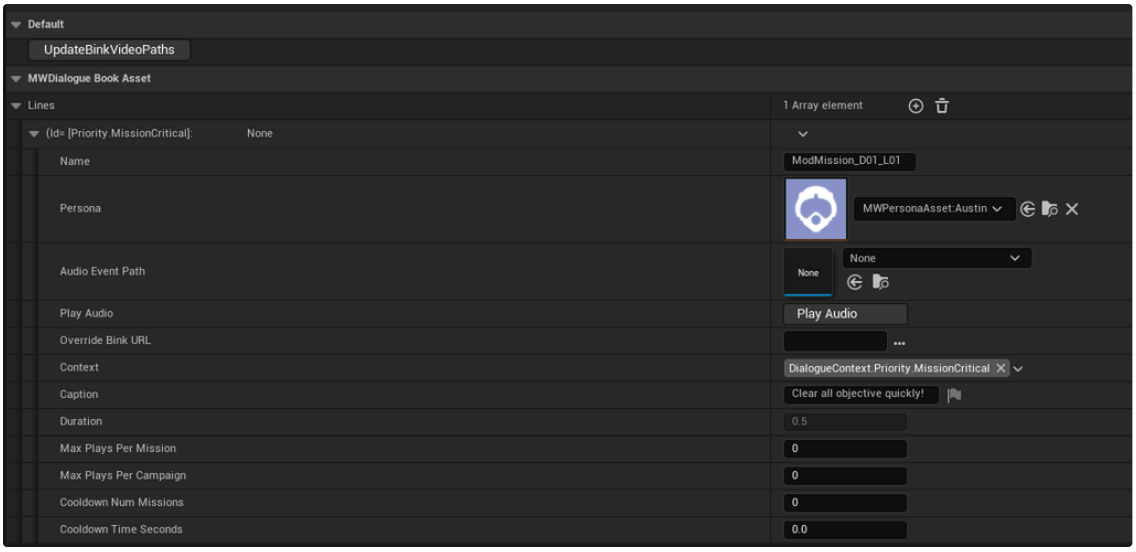


Assigned BP_AeroSpaceFighterAttackSequence_Triggerable to the MissionScript

Dialogue

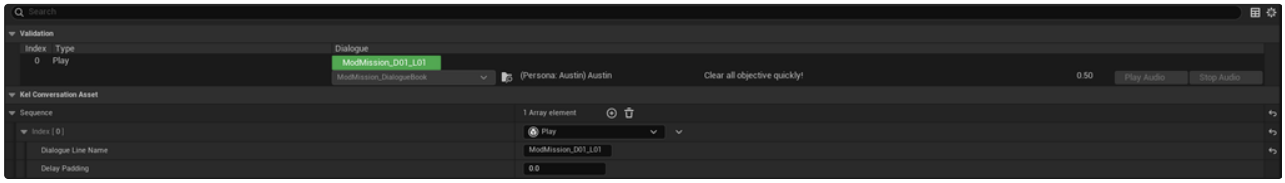
Create a folder for your dialogue files. In that folder, create an **MWDialogueBook Asset**. In this file, you can add your lines. Fill in **Persona** and choose any character that you want to speak. **Caption** to fill in the line that you want the character to say. Give it a **Name**, we'll need this to tell what line we're talking about.

MWDialogueBook Asset



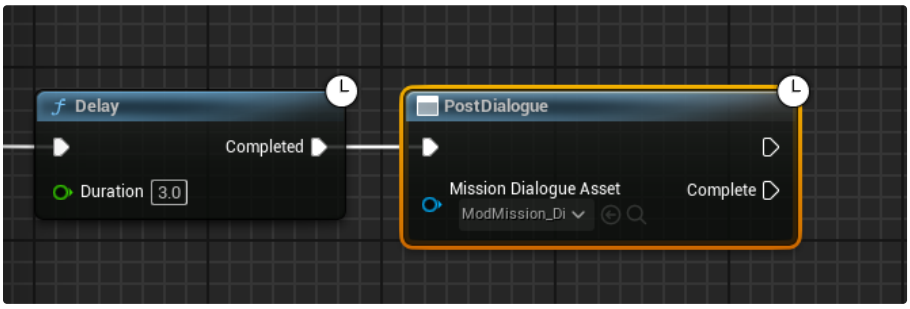
MWDialogueBook settings

Create a **KelConversationAsset** and open it. You'll need one of these for each conversation in your mission. In sequence, tell it to **Play** and add one of the lines from the **MWDialogueBook** to the **Dialogue Line Name**. Continue adding lines to finish the conversation that you want to play.



KelConversationAsset settings

Call **PostDialogue** in the MissionScript. In **Mission Dialogue Asset**, you can choose the **KelConversationAsset** you created. Now it should play the conversation when you need it.



PostDialogue in MissionScript