FMOD for loosely coupled architecture.

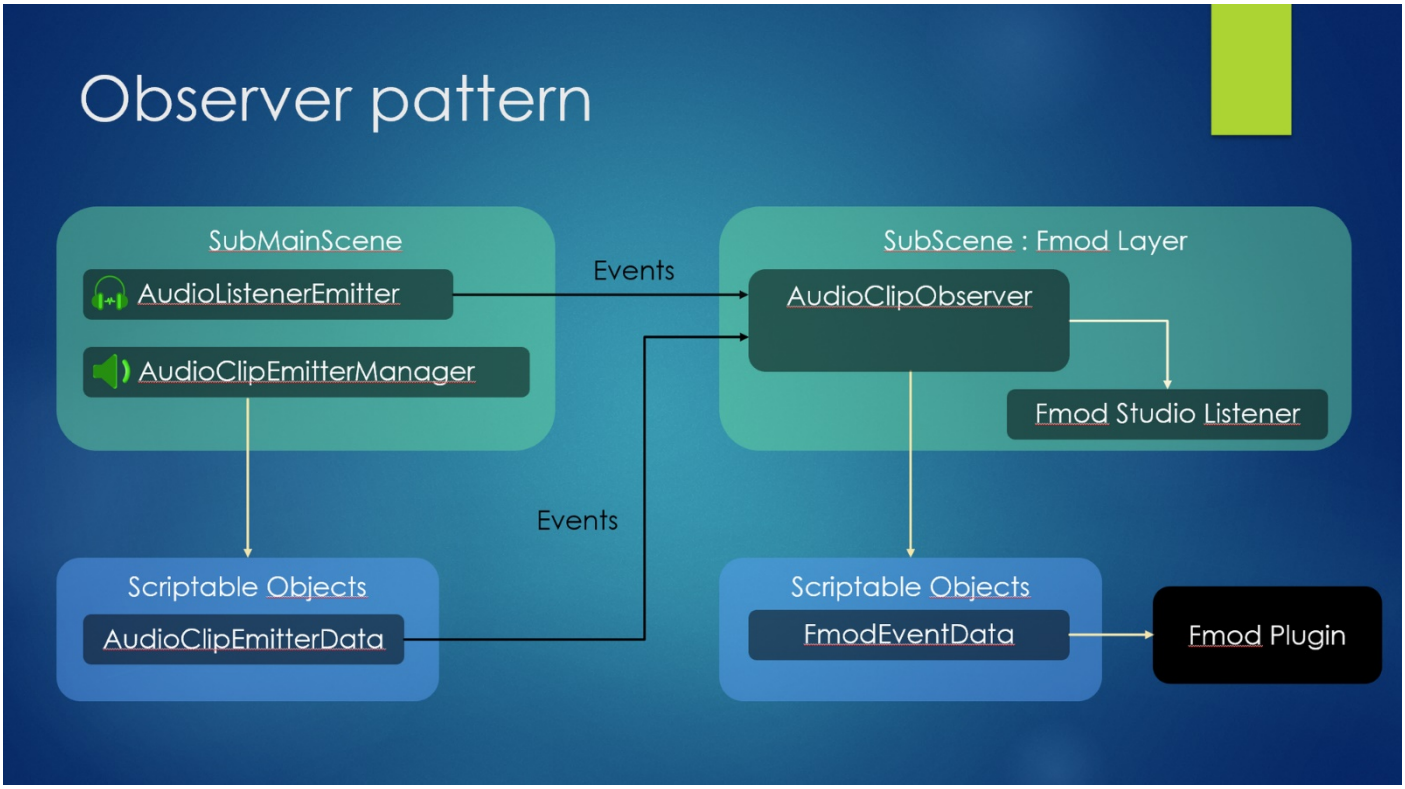
Decoupling allows game development without requiring prior installation of an audio engine. With the rise of remote work, this will significantly ease your work as a developer as you can work in parallel with the dedicated sound team. They can manage the audio aspect in parallel without waiting for your validation for integration.

Author :
Aude Valfroy

Table of Contents :

Introduction to the extension:

1. Overview of how it works:



2. Some explanations:

In a decoupled architecture project or with multiple sub-scenes, where each sub-scene is dedicated to a specific functionality. The main idea is to have a sub-scene dedicated only to sound (*whether you use FMOD, WWISE or any other sound engine*). This "layer" (*sub-scene*) is loaded from the launch of the game in order to constantly provide audio to other sub-scenes of your game that depend on it.

A dedicated event serves as a means of communication between the rest of the game and this sound sub-scene.

Decoupling allows game development without requiring prior installation of an audio engine. With the rise of remote work, this will significantly ease your work as a developer as you can work in parallel with the dedicated sound team. They can manage the audio aspect in parallel without waiting for your validation for integration.

This extension includes **two distinct parts**:

1. The first part is dedicated to the functioning of the scene dedicated to the sound design of your project (*this is where FMOD or WWISE comes in*).

2. The second part serves as a communication link between the sound scene and the rest of the game.

The scene called **FMODLayer** will allow the use of **specific components** to place sounds in the scene. However, these components will not be related to the audio engine like FMOD's Event Emitter.

The components to be used are as follows:

- **AudioClipEmitterManager**: This component should be attached to all elements that need to produce sound in your scene (it replaces the FMOD Event Emitter).
- **AudioListenerEmitter**: This component should be attached to the main camera of your project or the game character to hear the sounds you integrate.

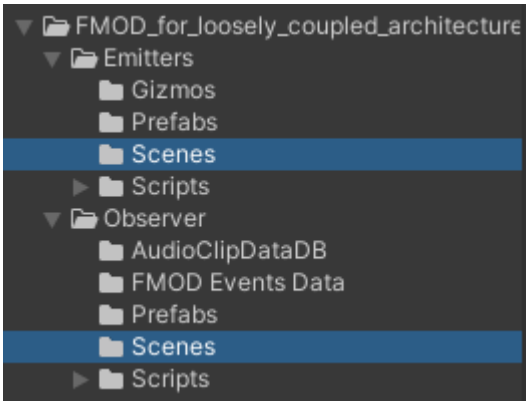
These two components will be used to send event messages to the **FMODLayer**, which will interpret and execute them.

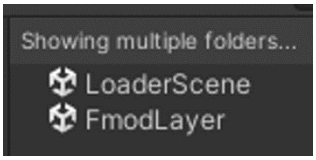
Configuring your project to integrate the extension:

Once you have downloaded and imported the extension, you should have a folder named **FMOD_for_lossely_coupled_architecture**.

1. Integrate the 2 scenes into your project:

You need to integrate these two **sub-scenes** into your project:

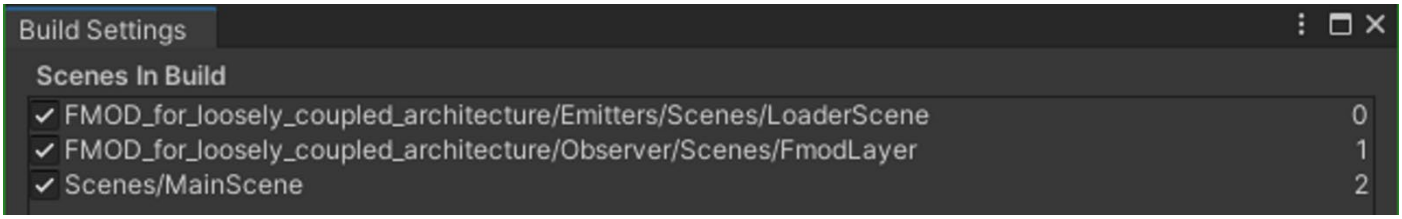




LoaderScene: This scene will handle the launching and management of other scenes in addition to the audio scene. (**FMODLayer**).

FMODLayer: Create the database (*related to the FMOD Studio bank, the list of events, buses, and VCAs*). Its role is to execute commands like the famous **PlayOneShot** from FMOD via scripts; we will not use **FMOD Event Emitters**. Therefore, we will enter a list of scriptable objects that we have created (*the database*).

Add **LoaderScene** and **FMODLayer** in the **Build Settings** options, in this order:

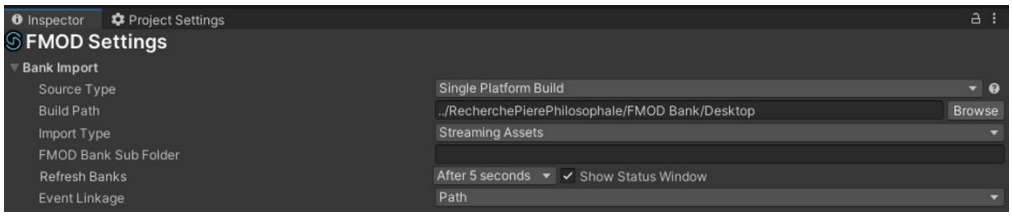


Make sure **LoaderScene** is at the top of the list as it will load the other scenes in the project.

2. FMOD Plugin Configuration:

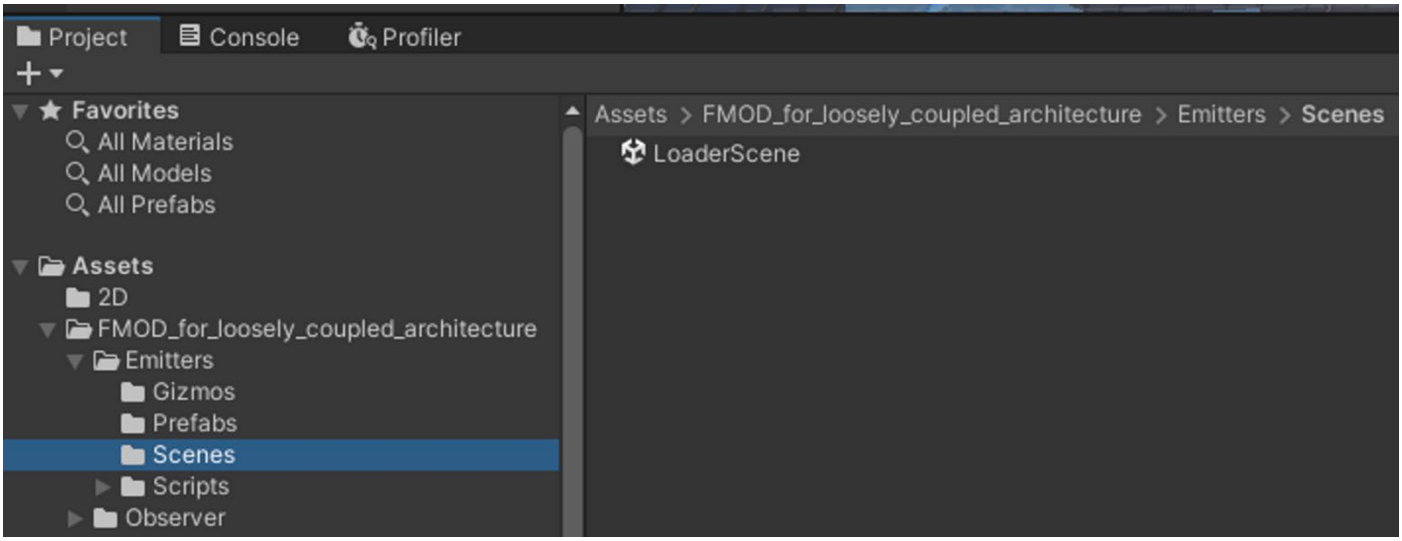
The FMOD plugin in your project should be configured with the **Single Platform Build** or **Multiple Platform Build** option. We use the plugin cache to create our database.
In Streaming mode, the cache will not be created.
In a team workflow, this solution is highly recommended.

(*For more information on using these parameters, please refer to the FMOD documentation available at this address ⇒* https://www.fmod.com/docs/2.00/unity/user-guide.html#accessing-your-fmod-studio-content)

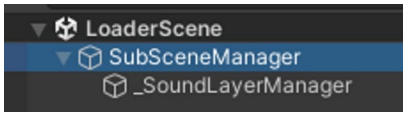


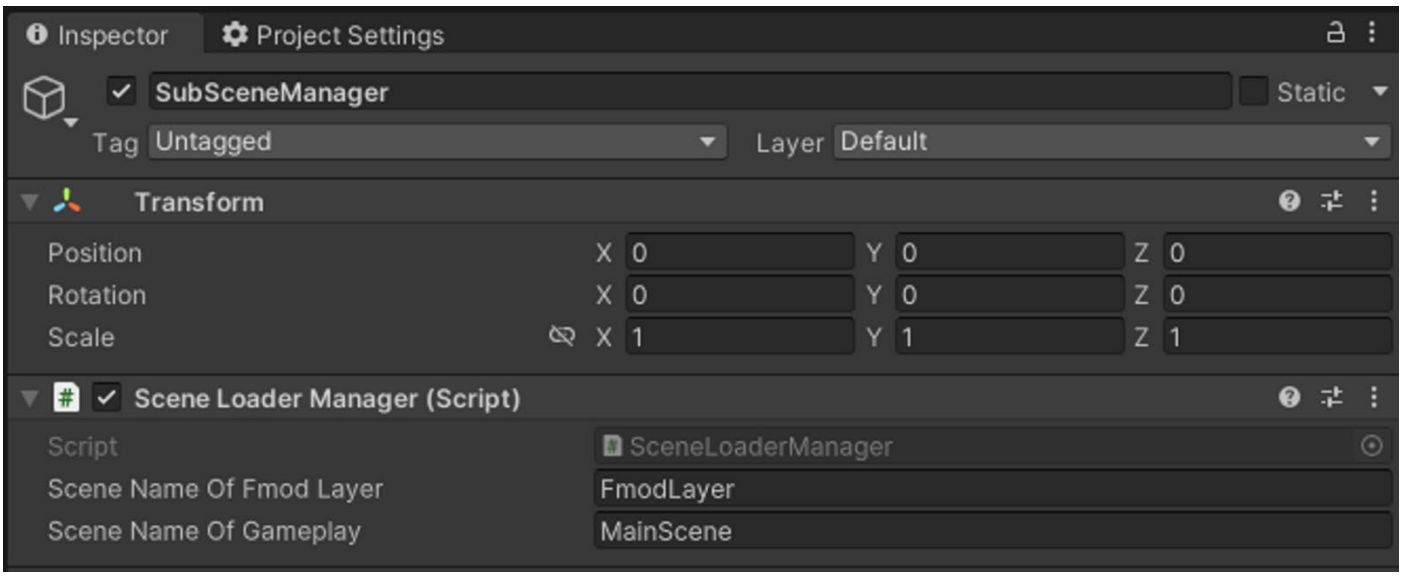
3. Scene Configuration:

Load the **LoaderScene** scene:



Once loaded, select the GameComponent **SubSceneManager** in the **Hierarchy** section:



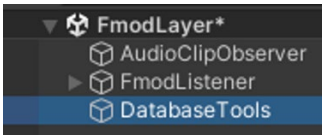


In the **Inspector** section**:**

- In the **Scene Name Of FMOD Layer** line, enter the name of the **FMODLayer scene**, which contains the **AudioClipEmitter** and allows you to hear the audio.
- The **Scene Name Of Gameplay** line should be filled with the name of the **main scene** of the project (*Gameplay section*).

Then, load the **FMODLayer** scene**:**

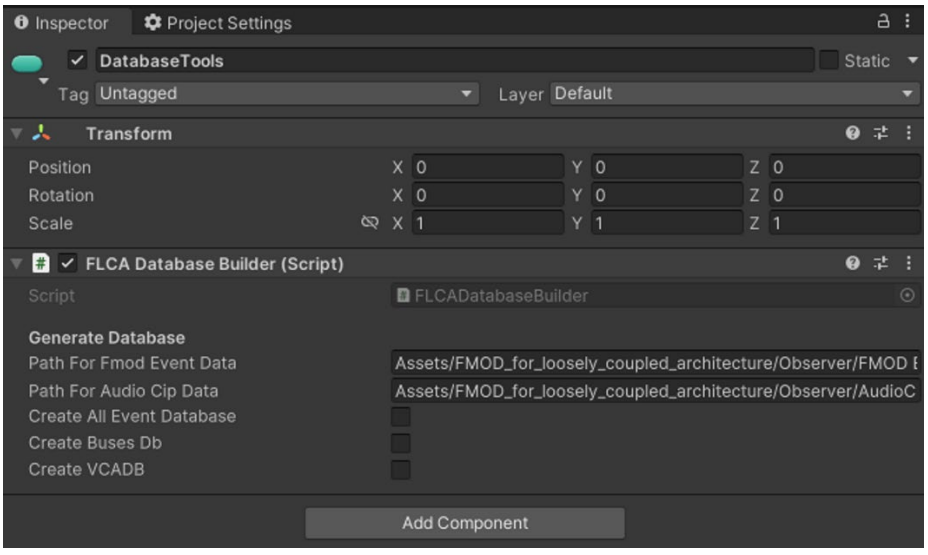
Select the **DatabaseTools** in the scene**:**



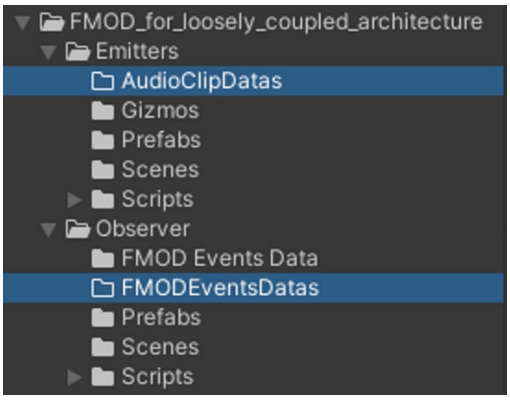
In the **Inspector** section, enter the following in the corresponding lines:

- **Path For FMOD Event Data** ⇒ he folder where the database related to FMOD calls (Sound Designer side) will be located.
  Default Folder Location:
  Assets/FMOD_for_loosely_coupled_architecture/Observer/FMODEventsData

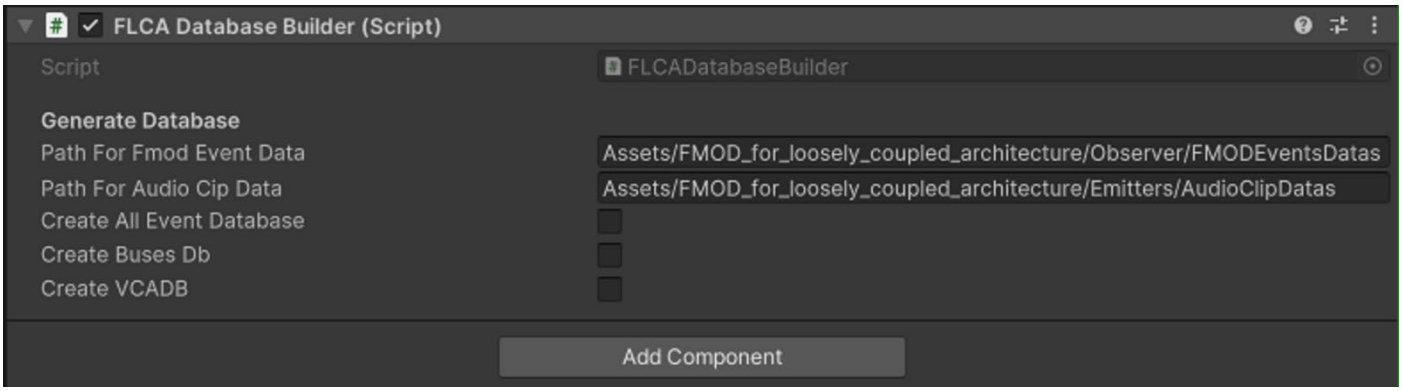
- **Path For Audio Clip Data** ⇒ The folder where the database related to **AudioClipEmitterData**, which are the elements placed in the scene, will be located. (*Developer/Audio Programmer side*)
  Default Folder Location:
  Assets/FMOD_for_loosely_coupled_architecture/Emitters/AudioClipData



In our example, we will create two folders. One folder for the developer side (*AudioClipData*) and another folder for the Sound Designer side (*FMODEventsData*).



So, we will provide the file path for these two folders in the appropriate location:

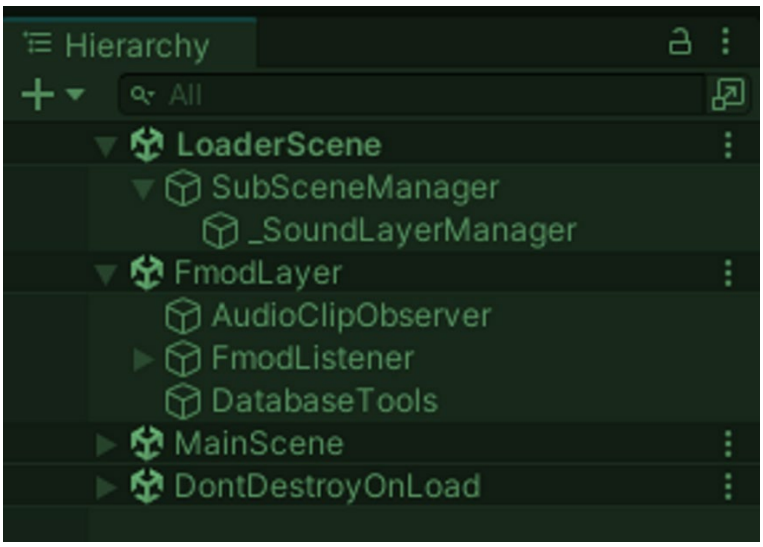


Once these two paths are entered, save your changes.

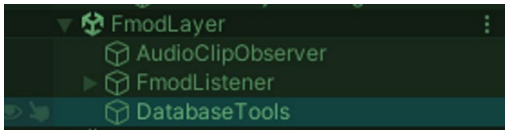
Create the sound database for your project:

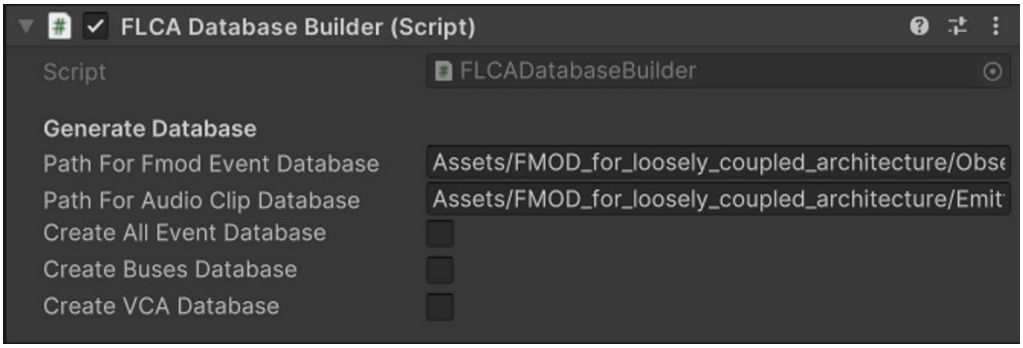
1. Create the database:

Load the **LoaderScene** scene and then run a **PlayRuntime** in the **Unity editor**. You should get a scene with several sub-scenes:



Select the GameComponent **DatabaseTools** in the **FMODLayer** sub-scene. You should see the following menu on the **Inspector**:





Now, check the corresponding boxes for the **Database** you want to create:

1. **Create All Event Database:** This will create scriptable objects for all the **FMOD events** and **snapshots**, as well as their associated elements. (*Please refer to the documentation on FMOD Studio 2.02 for more information on snapshots)* ⇒ *https://www.fmod.com/docs/2.02/studio/mixing.html#snapshots-and-the-tracks-view*)

2. **Create Buses Database**: This will create a **list of buses** that you have created in **FMOD Studio**. If you prefer to directly manipulate a bus instead of a VCA in your project, you can do so.

3. **Create VCA Database**: This will create a **list of VCAs**, which are often used for volume control in project options.

When you check the **Create All Event Database** box, Unity may freeze for a moment while it creates all the elements. Depending on the number of events in FMOD Studio and the power of your computer, this process may take some time. (*On average, it can take between 30 seconds and 2 minutes*)

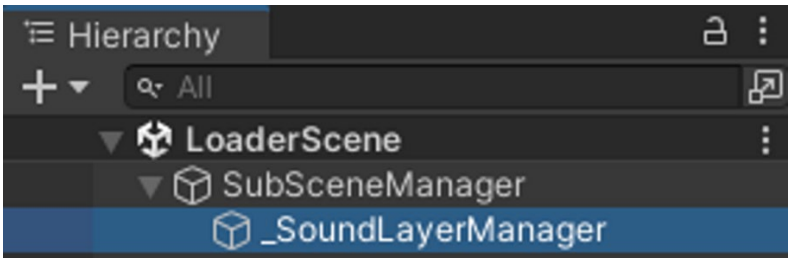
The checkbox will remain unchecked after the process is complete, which is normal!

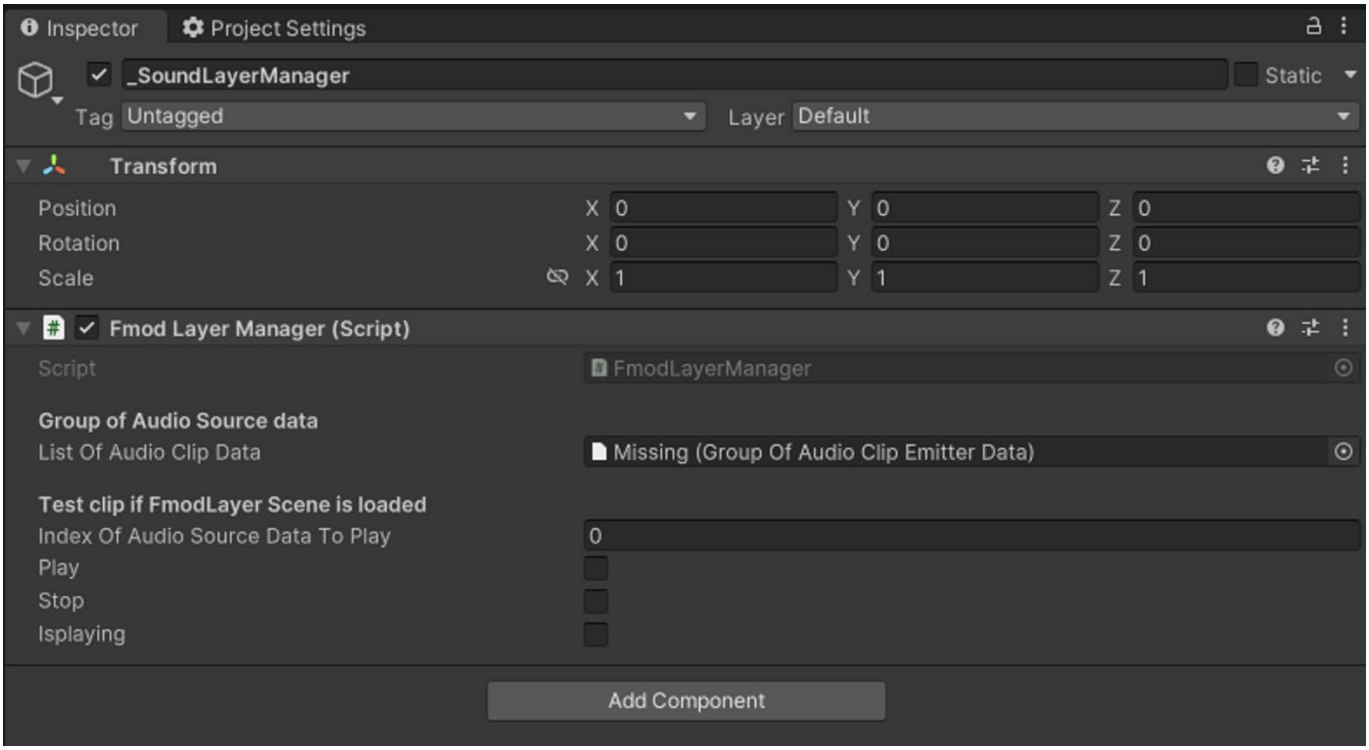
Once the database is created, you can stop the **Play Runtime** mode in the Unity editor.

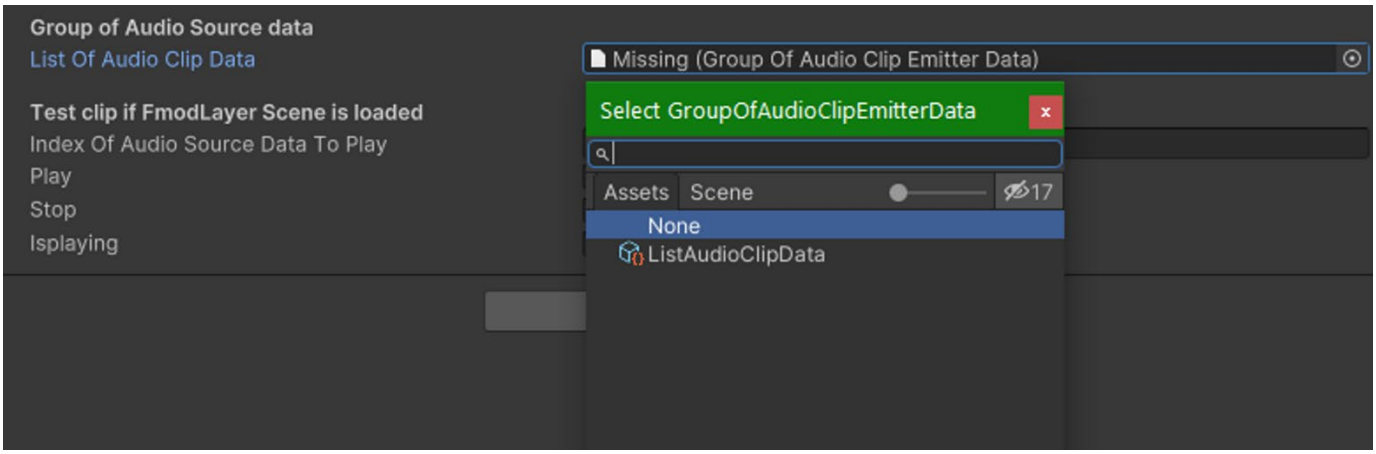
2. Setting up the scenes:

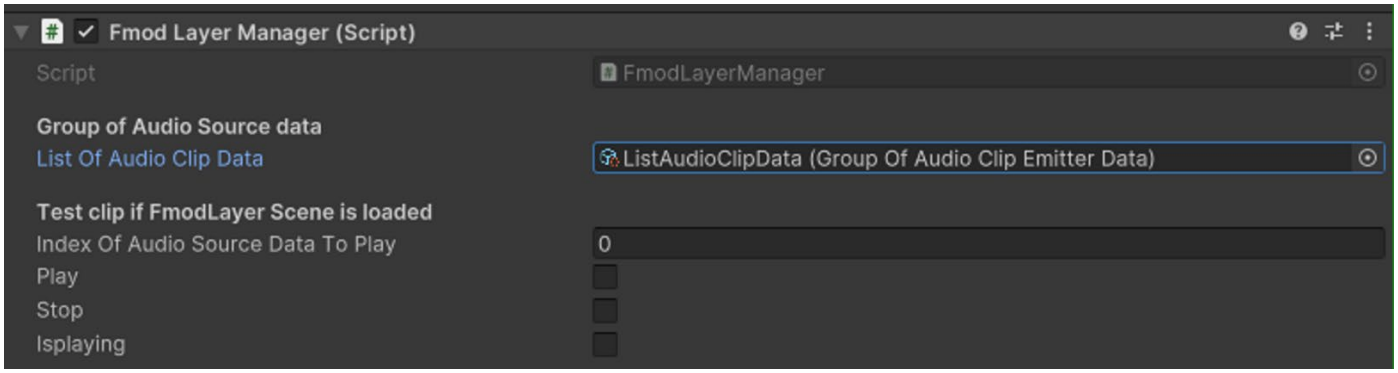
The next step will involve filling in the created database in the appropriate locations within our scenes to establish the connections between our two scenes, **FMODLayer** and **LoaderScene**.

1. Load the **LoaderScene** again (*if not already done*), then select the **_SoundLayerManager** to display the relevant parameters in the **inspector**. In the **List of Audio Clip Data** line, choose the newly created database. There is no possibility of making a mistake.





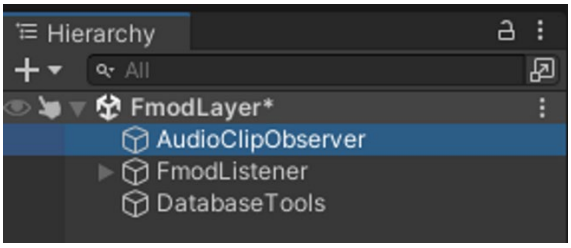


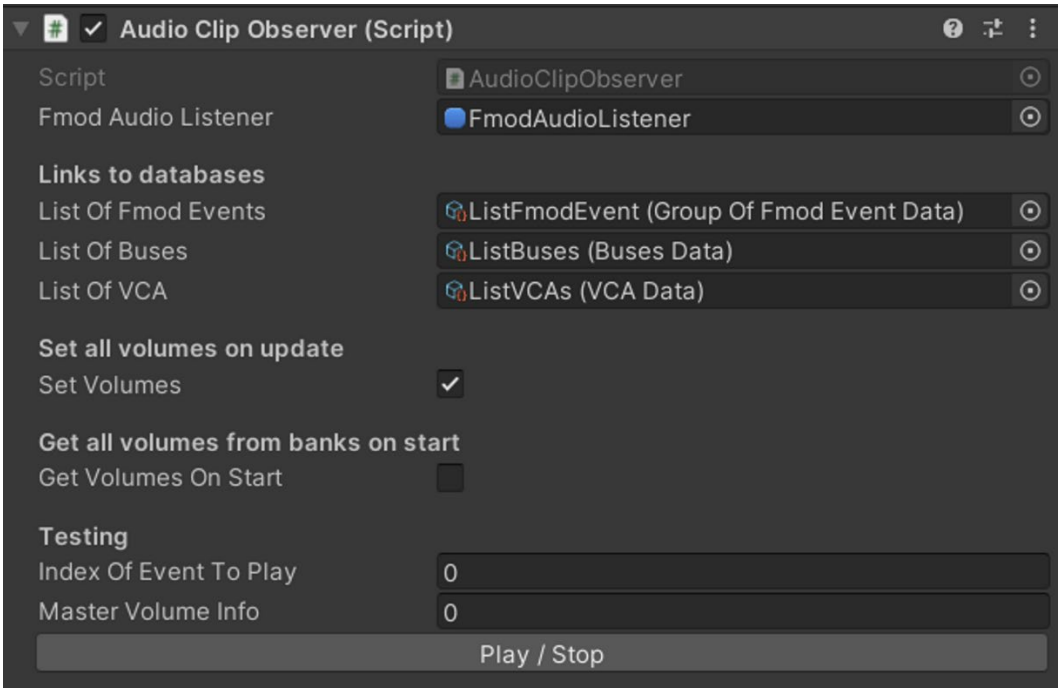


Save the scene once the changes have been applied.

2. Next, load the **FMODLayer** scene, then select the **AudioClipObserver**. In the **inspector** panel, you will need to provide information for three elements:

- **List Of Fmod Events**
- **List Of Buses**
- **List Of VCA**





Remember, save your changes.

You have finished configuring the two main scenes that allow you to listen to and control the audio extension for managing the audio in your project.

3. Additional information!

Attention: Please note that these steps will need to be repeated if you delete the database from the two folders or any part of it. This applies to situations such as making certain changes or renaming your elements.

Attention²: **Any renaming of events, parameters, buses, or VCAs will require manually deleting the contents of the two folders containing the database in order to recreate it properly. It's important to manage your architecture carefully from the beginning to handle such situations effectively.**

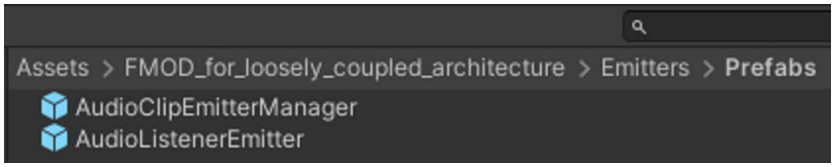
(*An improvement to avoid manual deletion is currently being developed to address this aspect.*)

Concrete use of the extension:

1. How to use AudioClipEmitterManager:

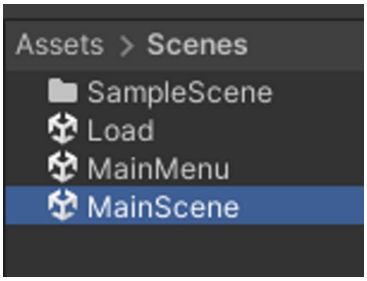
If you are already familiar with the **FMOD plugin**, you are already familiar with its usage through the **FMOD Event Emitter** component. Whenever we need to trigger a sound in the scene, we typically use the **FMOD Event Emitter** on a GameObject in the scene.

You will replace the usage of **FMOD Event Emitter** with the **AudioClipEmitterManager** (*either as a GameComponent with the prefab or as a regular component, depending on the triggering method you want to achieve*):



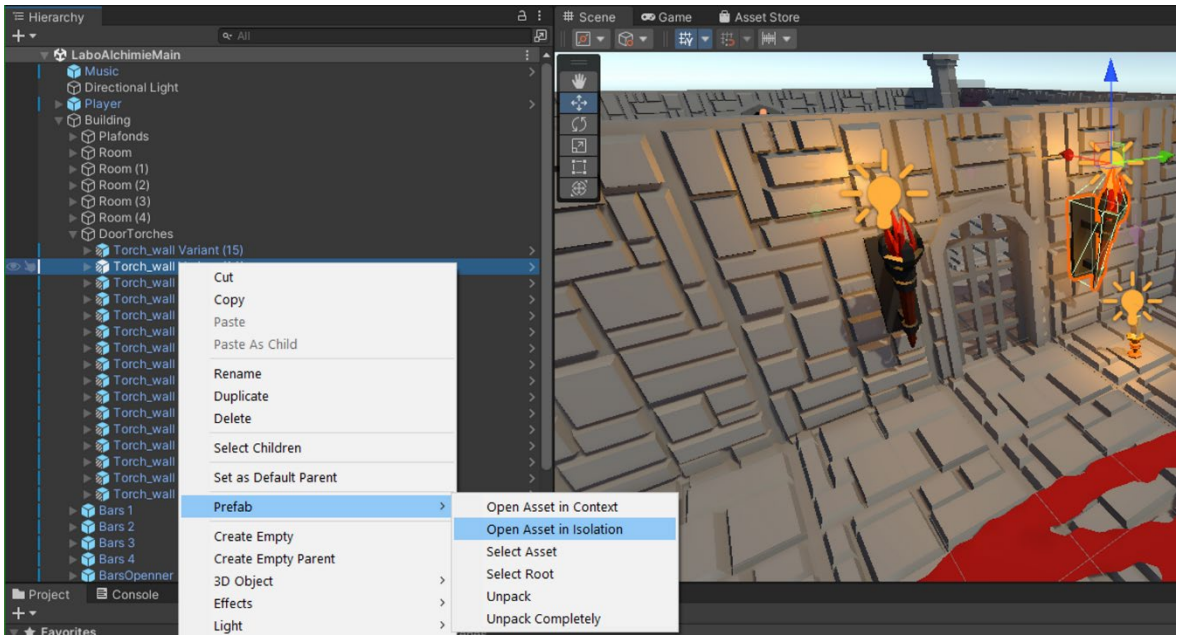
In our example, we will place our **AudioClipEmitterManager** on the elements that we want to add sound to, in this case, **the torches** in this scene:

To be able to place our audio in our game scene, we will load the **main scene**:

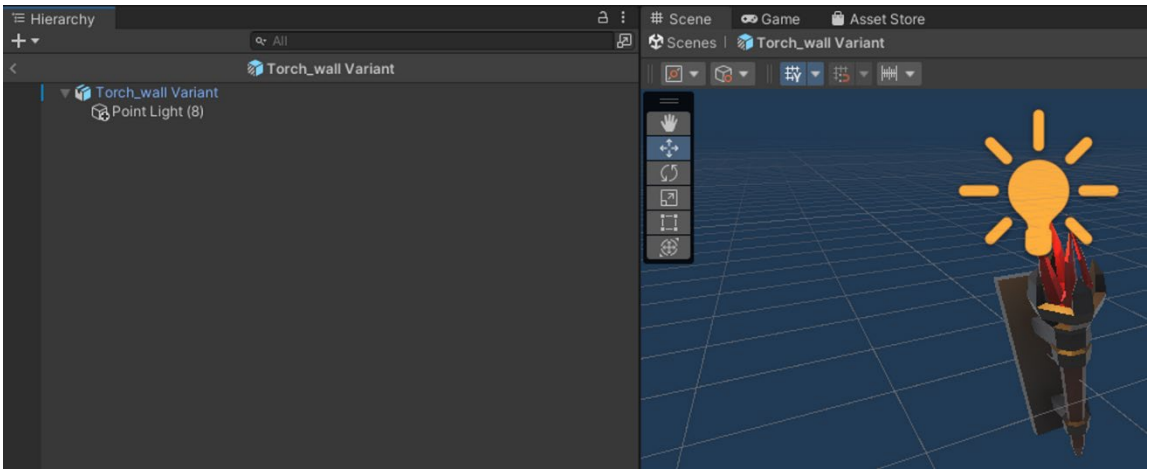


Here, my main scene is called **MainScene**, which is the scene where the entire project is located.

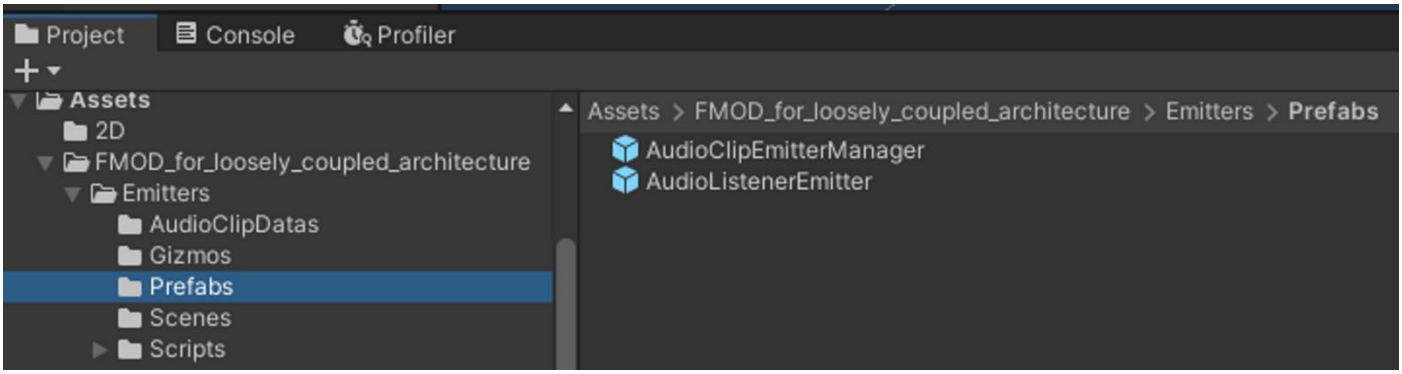
By selecting one of the **torches on the wall**, I choose to open the **parent prefab** associated with it so that my modifications apply to **all the torches** in the scene.

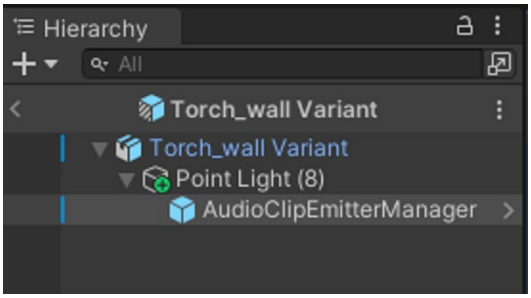


In the torches **prefab**:

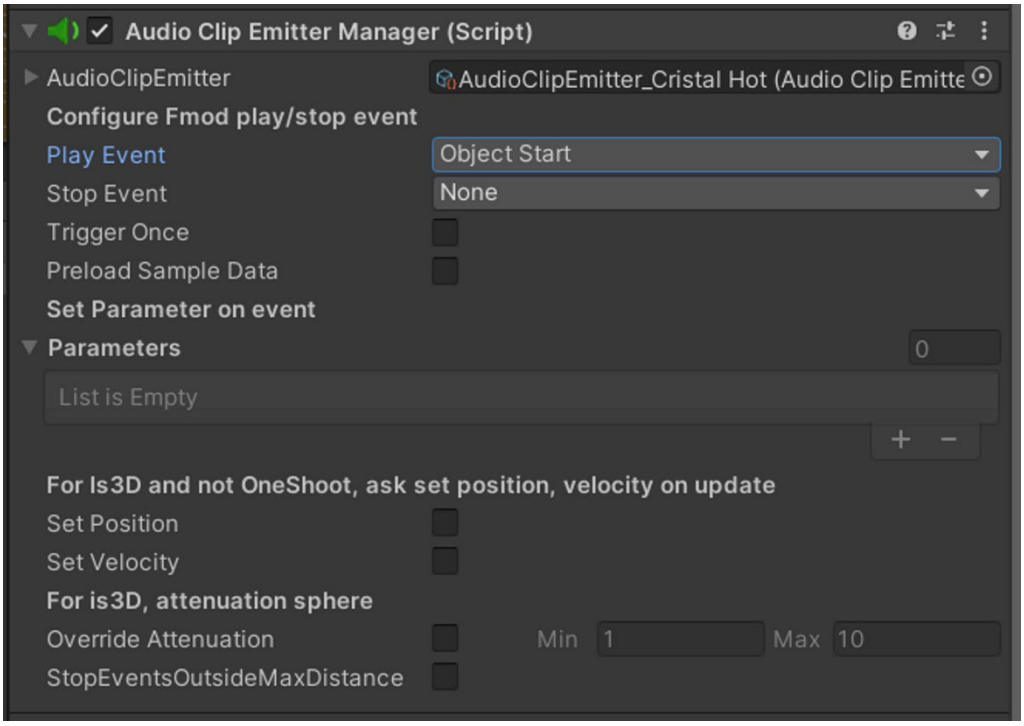


I place my **AudioClipEmitterManager**:



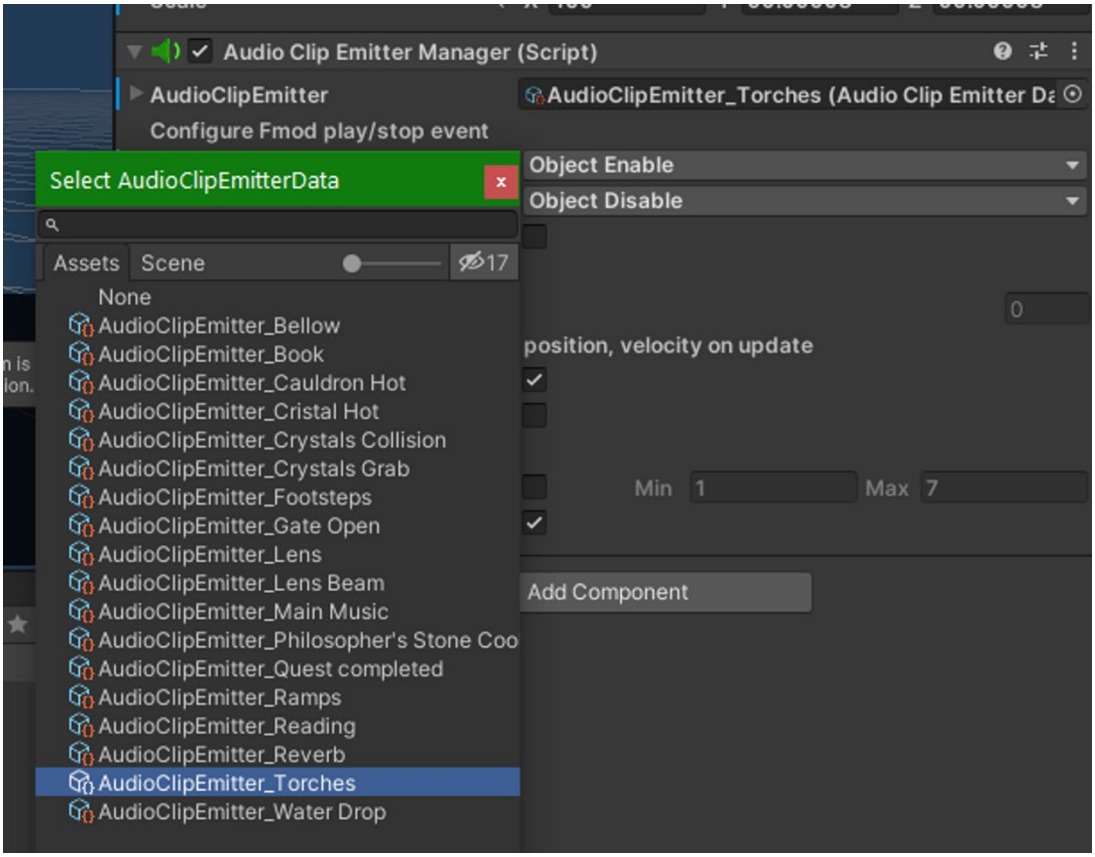


Next, I select the **AudioClipEmitterManager** GameComponent to fill in some parameters in the **inspector**.

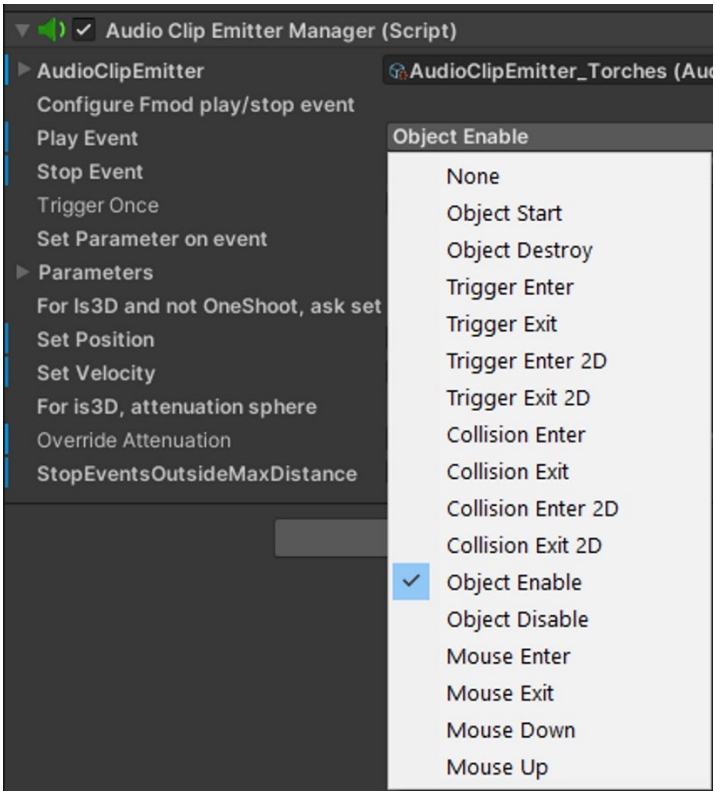


We have a similar setup to an **FMOD Event Emitter**, which we can configure as usual.

Then we provide the **Torches event:**

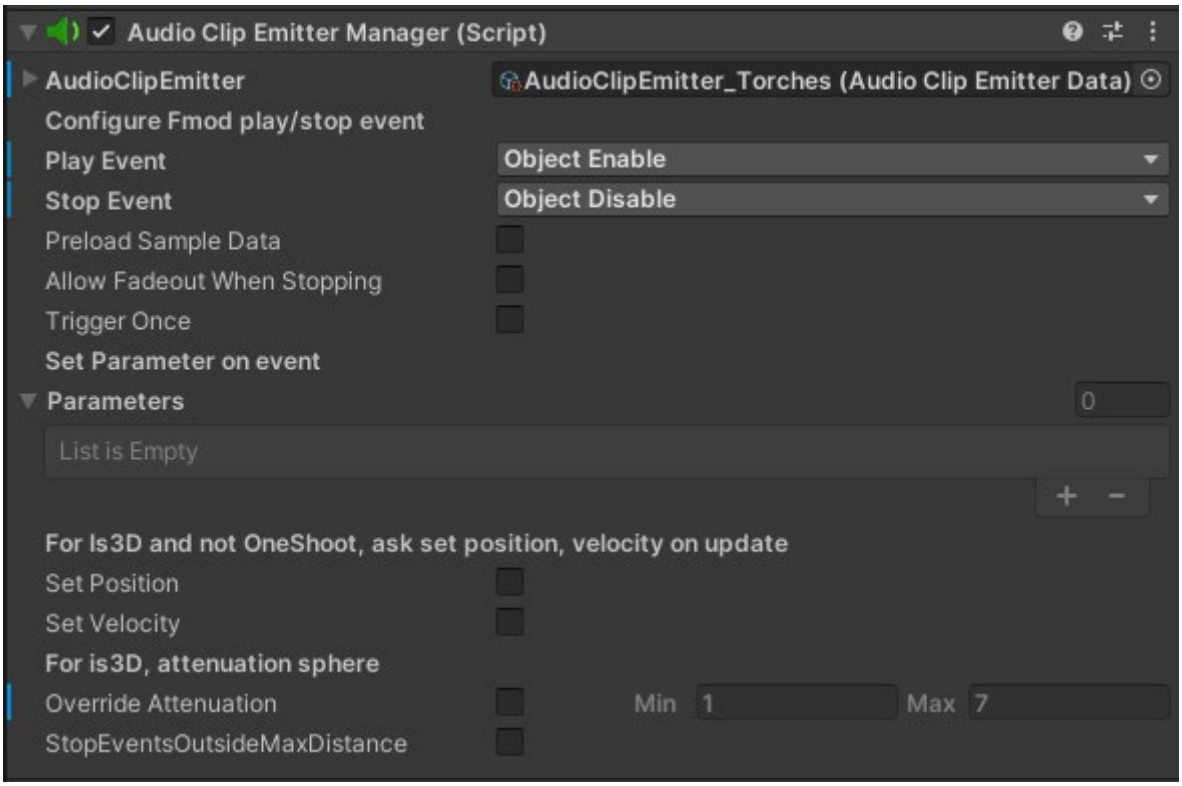


Next, just like with a typical **FMOD Event Emitter**, we choose the options that interest us based on the **triggering method** we are looking for:

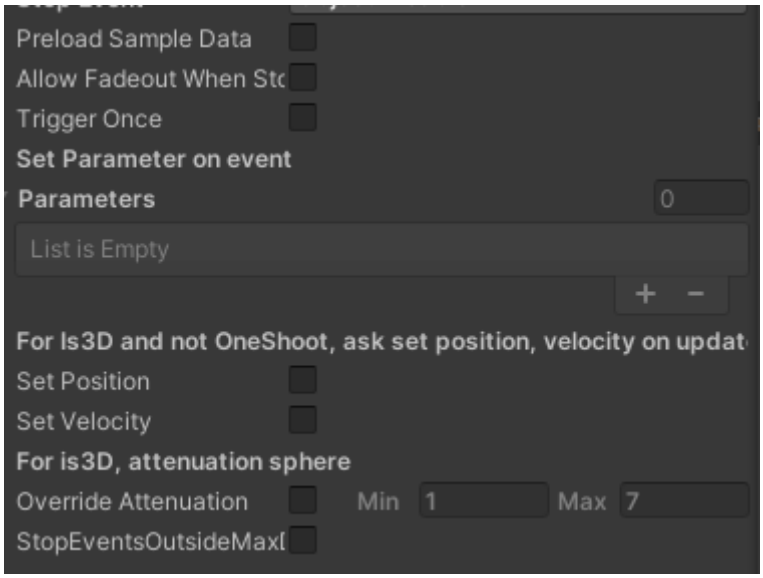


The **AudioClipEmitterManager** is connected as a **child** of the **Point Light** GameComponent. We want the audio to be heard only when the light (*Point Light*) is turned on. The status of the object in the inspector will determine whether the light is on or off.

So, I choose **Object Enable** as the trigger parameter and **Object Disable** as the stop parameter. This means that the sound will play when the torches is active (*Object Enable*), and if I interact with it and turn it off, the sound will stop playing (*Object Disable*) at the same time.



The checkboxes below are related to various useful options depending on the nature of the sound you're integrating:



- **Preload Sample Data**: option has the same functionality as the option with the same name in the **FMOD plugin settings**. Here, you can **enable it individually** for each audio clip instead of applying it to all the audio clips collectively.

- **Allow Fadeout When Stopping:** Allows enabling the ability to consider the ADSHR effect created in FMOD Studio for the selected event. If it is not present, it has no effect.

- **Trigger Once**: option allows you to play the event only once per scene.

- **Set Position**: If checkbox is checked, the position of the sound source will **follow the object** to which it is attached. However, it is not recommended to check this option for **fixed elements** like torches.

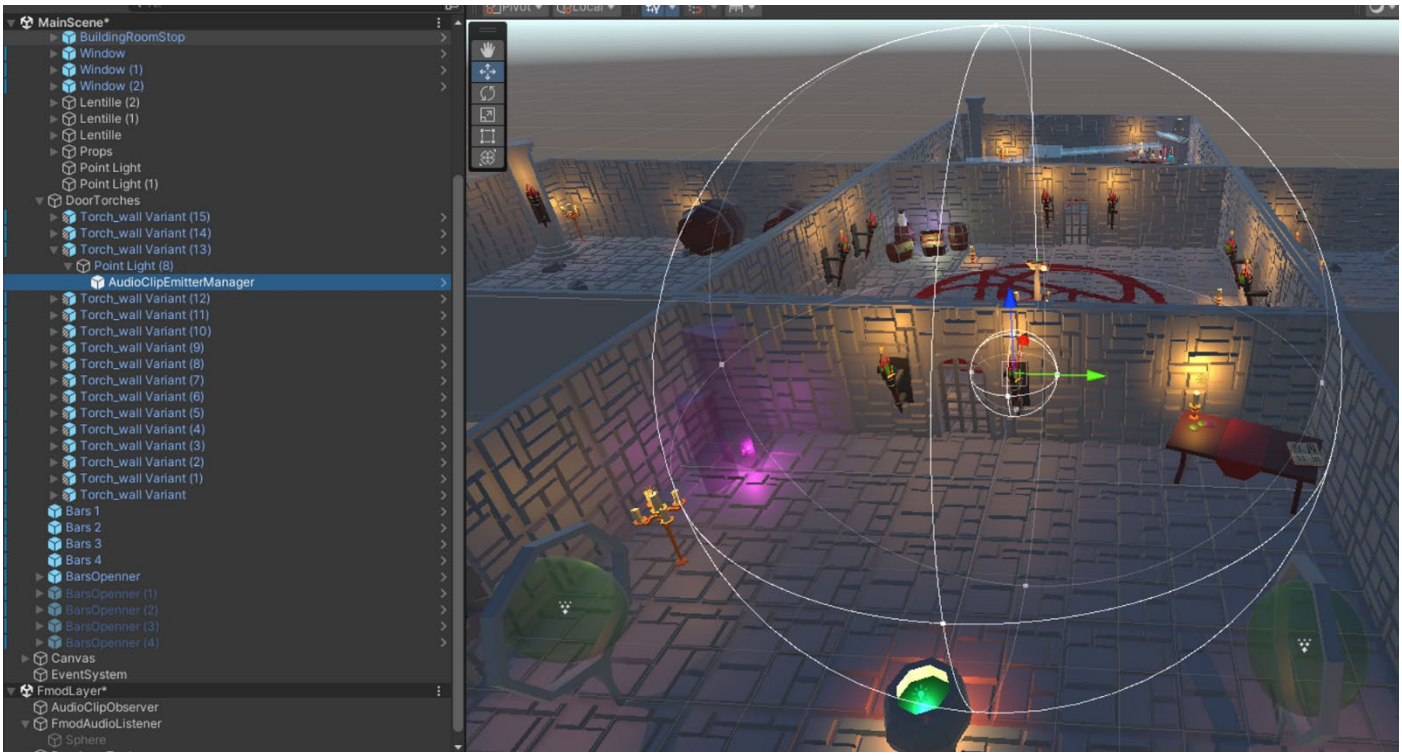
- **Set Velocity**: checkbox is checked, it sends the object's velocity information to the FMOD event. This feature requires a **Rigidbody component** as the **parent** to work properly.
- Use case: Player Footsteps.

- **Override Attenuation**: allows you to apply a **custom attenuation sphere** with different minimum and maximum values.

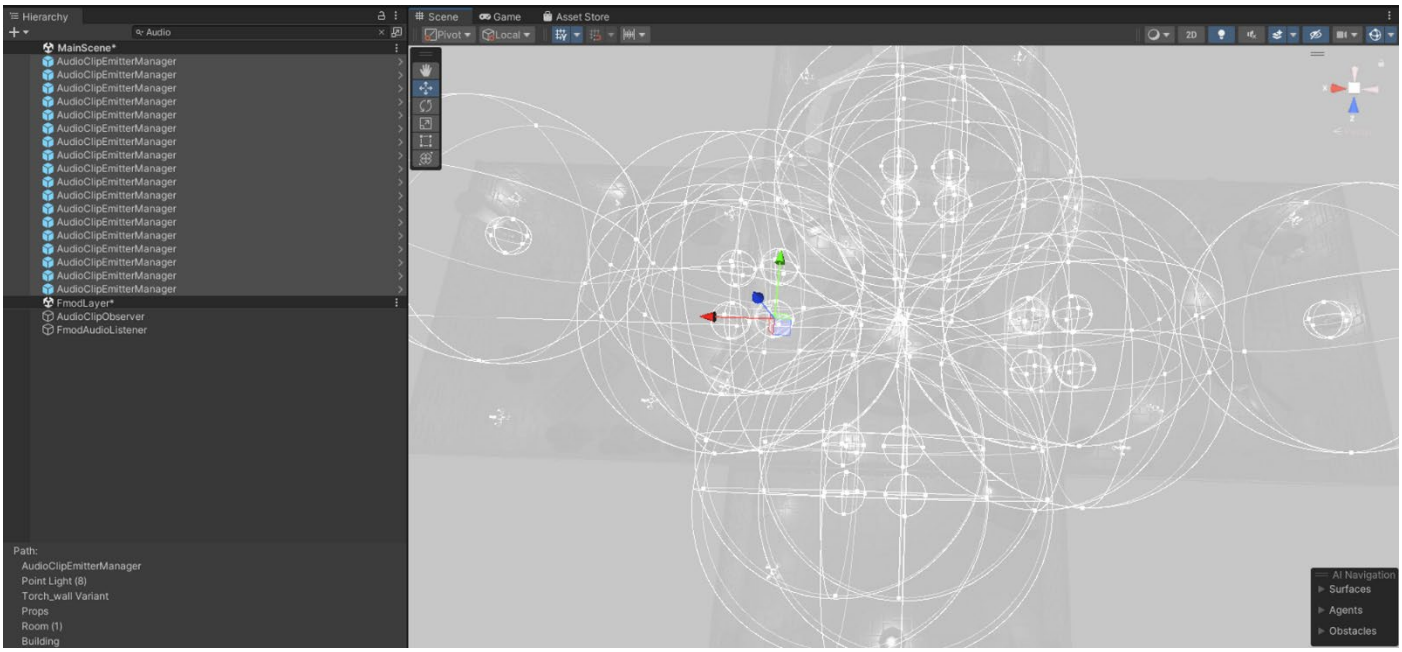
- **StopEventsOutdieMaxDistance**: allows you to stop playing the sound when you move outside the attenuation sphere (*maximum distance value*).

2. Attenuation Sphere:

To view the **attenuation sphere** of your elements in the scene, you must have the selected **AudioClipEmitterManager**:

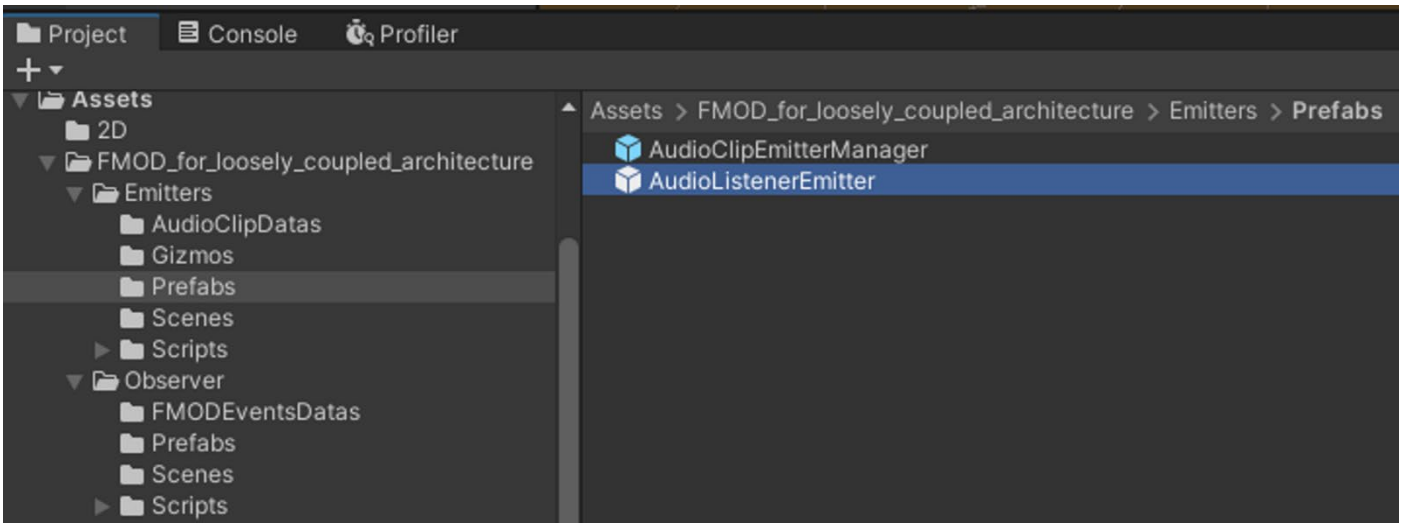


A quick search of my audio sources in the scene, a grouped selection, and I can see all my attenuation spheres:

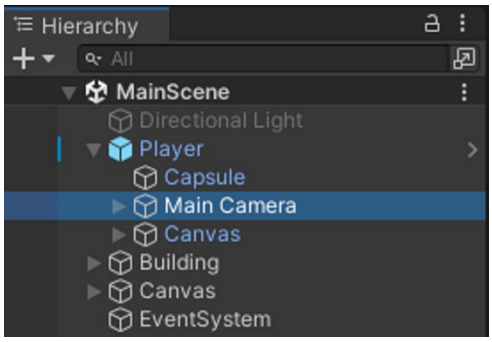


3. Don’t forget AudioListenerEmitter:

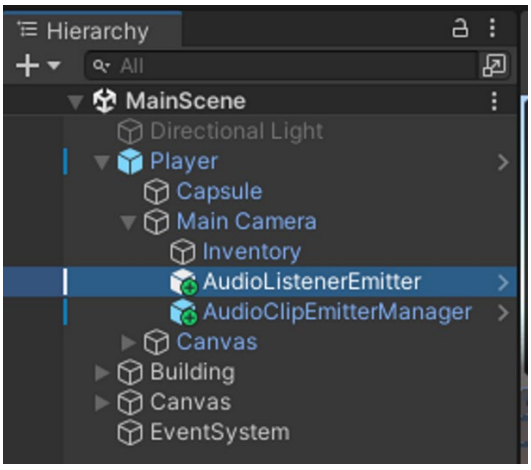
To be able to listen to your sounds, you'll need **ears** in your scene. That's where the **AudioEmitterListener** prefab comes into play.



In most projects, you need to place the **AudioListenerEmitter** prefab on the **MainCamera** GameObject. (*Depending on the style of your game, be cautious with top-down shooters, for example, as they may require different panning management.)*
Load the **main scene** and locate the position of the **main camera**:



Drag and drop the **AudioListenerEmitter** prefab to the desired location:

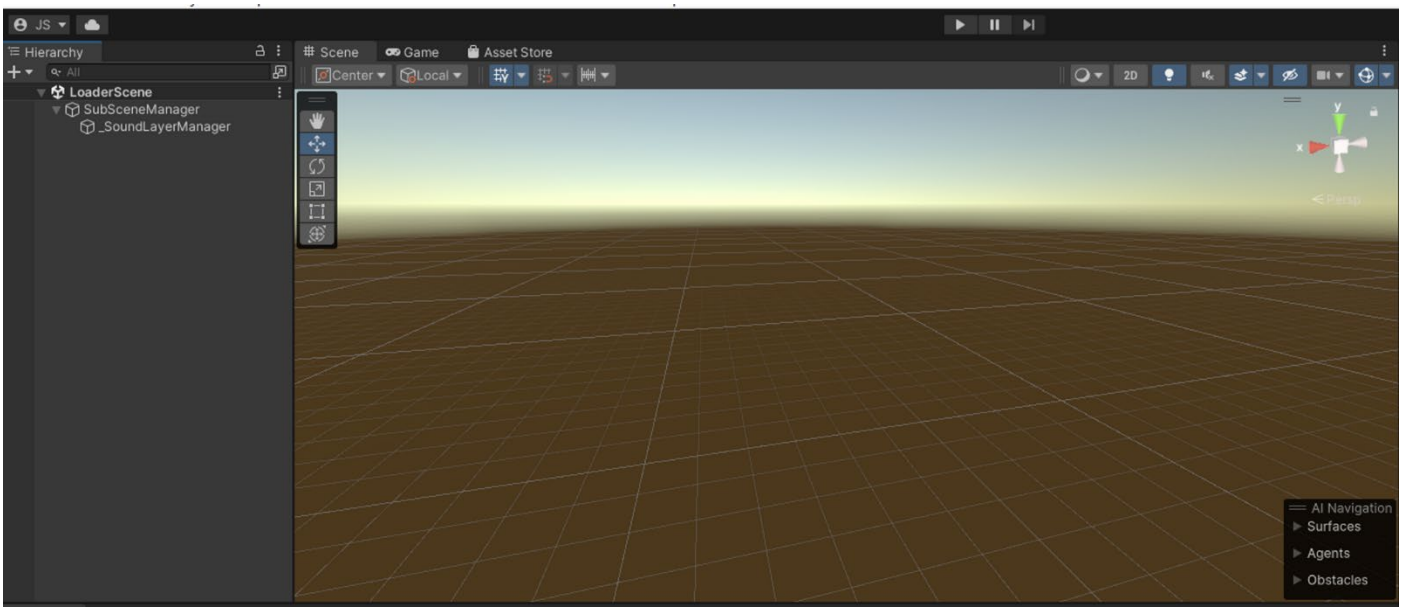


You don’t need to do anything in the **inspector**.

Save Scene!

Everything is set up, let's start listening!

Load the **LoaderScene** and then start the **PlayRuntime**. You will find all the scenes and sub-scenes with your sounds:

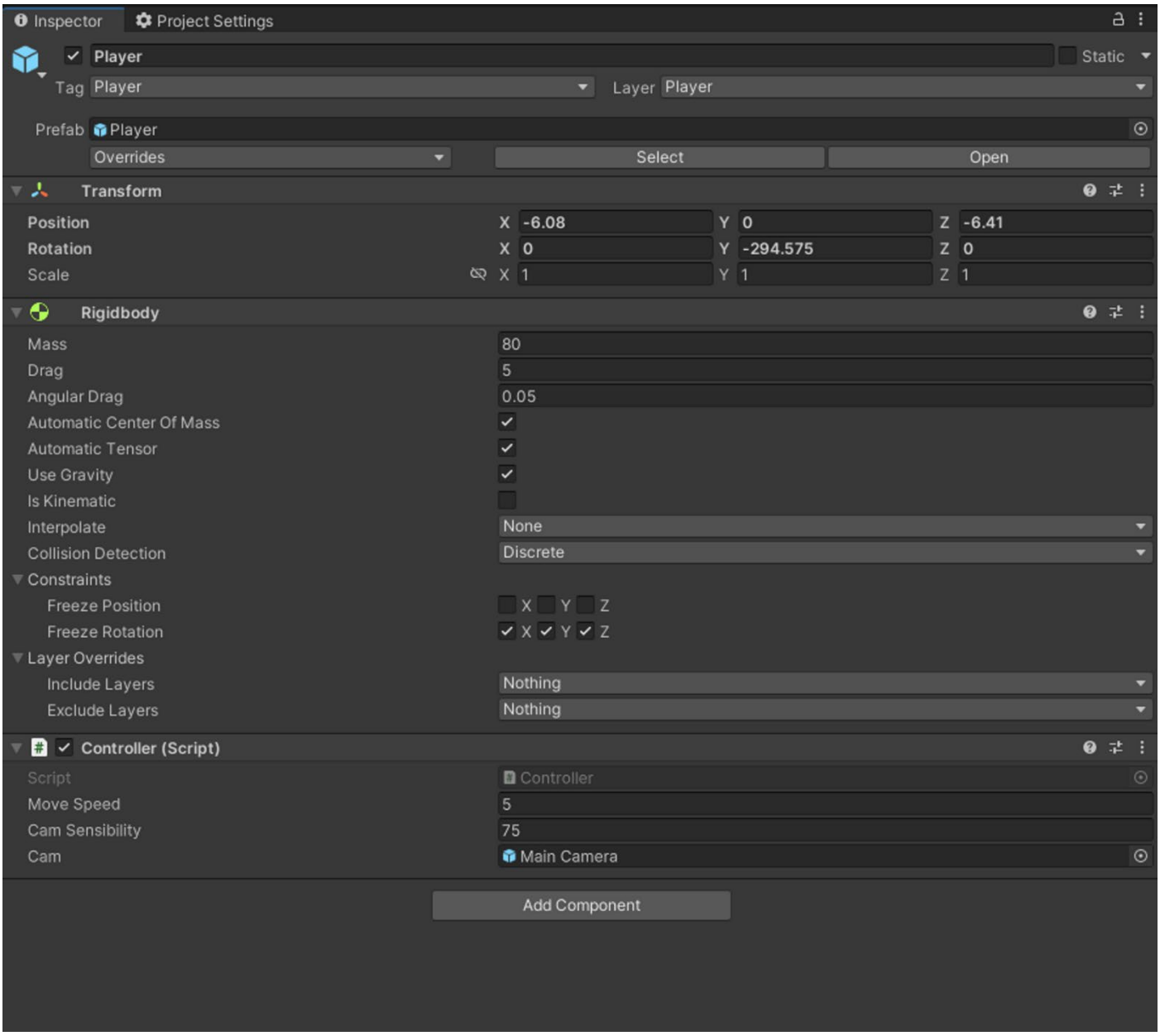


Of course, if you're a Sound Designer and you need to place your sounds in the main scene, to avoid the hassle of loading scenes every time, you can simply drag and drop the **LoaderScene** and **FmodLayer** as **sub-scenes**. This allows you to work quickly and efficiently.

Here are a few practical use cases

1. Varying a "Built-in" parameter of FMOD Studio:

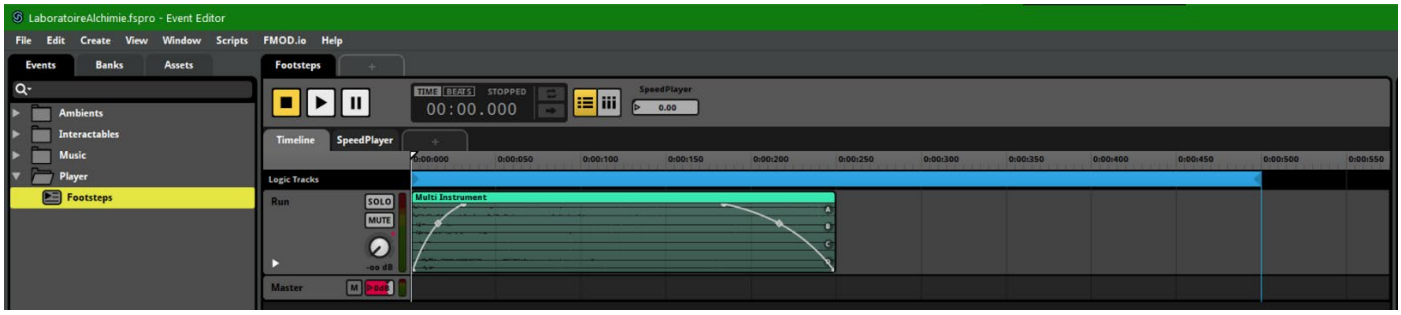
In our example, we have a GameComponent **Player** that has a **RigidBody.**



The maximum movement speed of the character in this project is set to a value of 5.

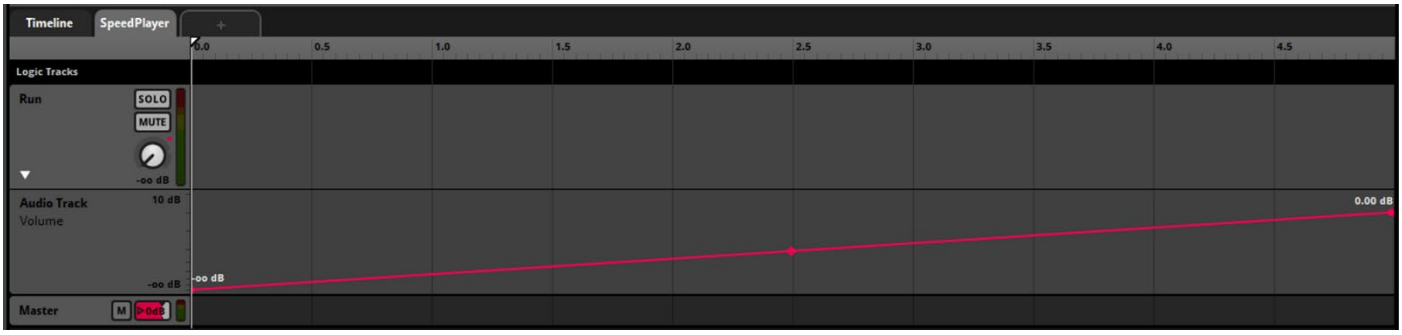
I will use this information to create my **parameter** in **FMOD Studio**:

An **event** named **Footsteps** has been created:

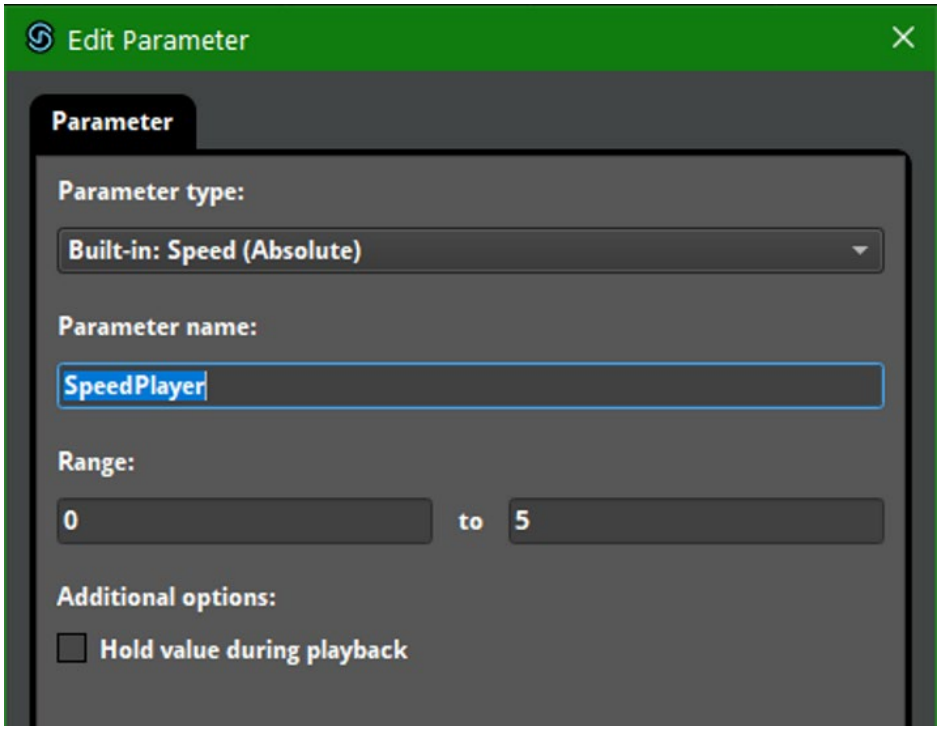


There is a loop that will play upon game loading, and we will vary the volume based on the **player's speed**.

A **parameter** named **SpeedPlayer** in the **Built-in** format will affect the volume in real-time.



Here's how the **SpeedPlayer parameter** is configured:

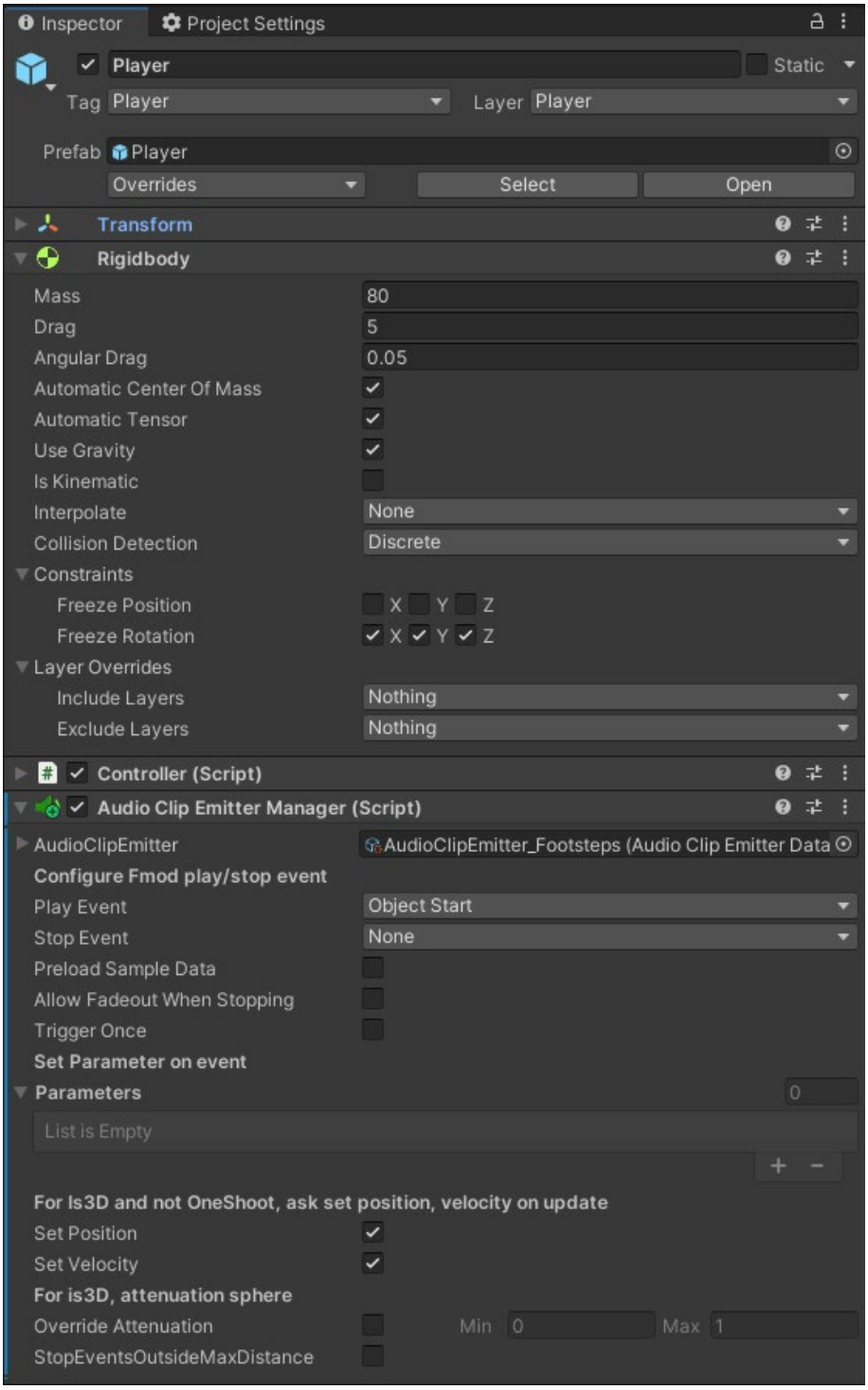


We provide it with a minimum and maximum value to be consistent with the configured in-game speed variable.

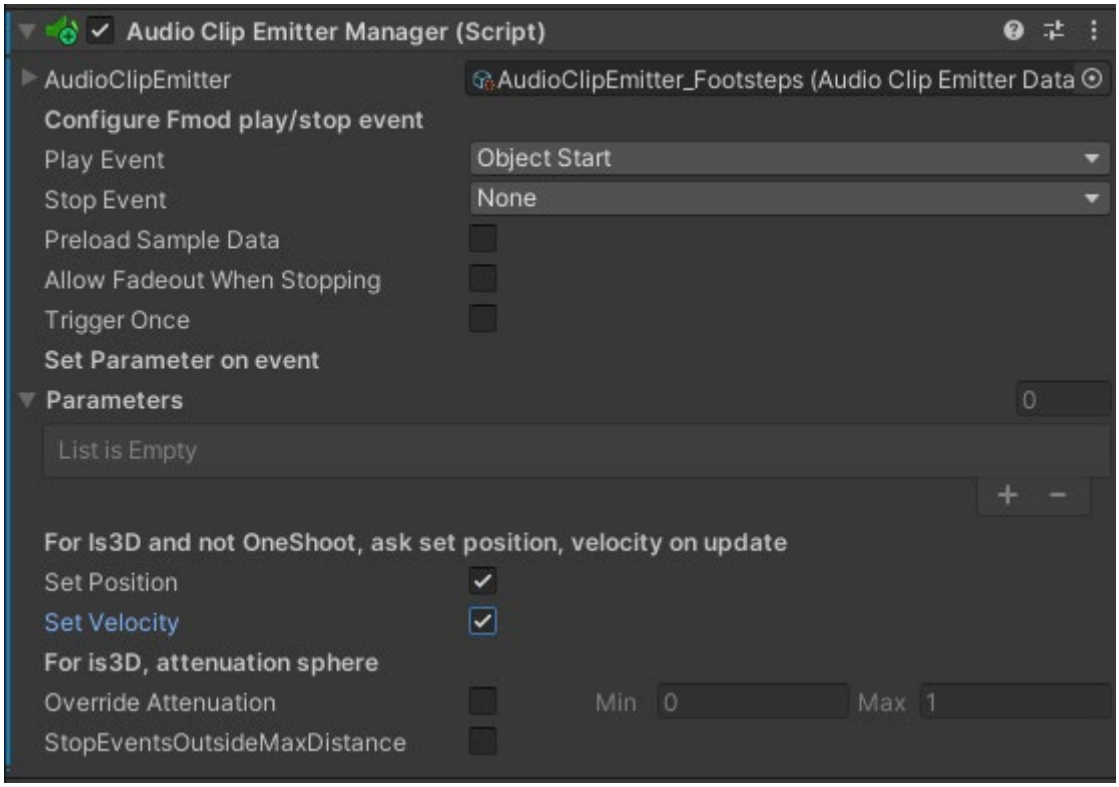
Now that the parameter and event are created, I will proceed to build my **FMOD Studio project** to update the banks that are called in the project, and then I will return to Unity to connect this event to my player.

Additional information : Regarding the topic of built-in parameters, I refer you to the FMOD documentation at the following address ⇒ https://www.fmod.com/docs/2.00/studio/parameters-reference.html#built-in-parameters
You will find a comprehensive overview of the functionality of FMOD Studio's built-in parameters.

Back in **Unity**, we will add a component to the **Player**, at the same level as the **RigidBody** component, which is necessary for the proper functioning of our event that needs to retrieve information from it in order to vary our **SpeedPlayer parameter**.



Here's how to configure our **AudioClipEmitterManager** component to make everything work:

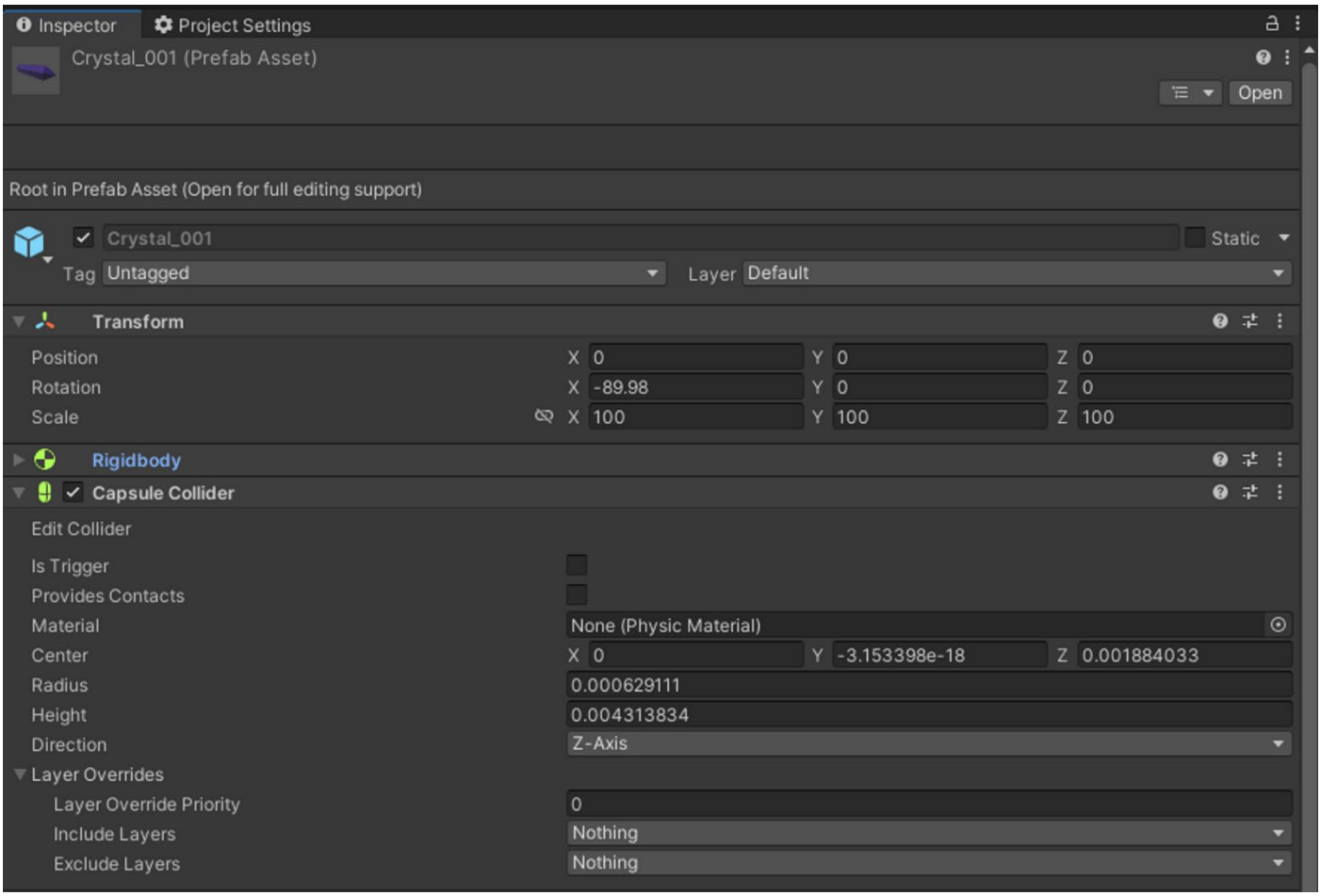


Having created an event with an infinite loop, I have chosen to start this event at the game's startup. To ensure that the parameter receives real-time updates from the **player's movement speed**, I need to have the **Set Velocity** checkbox checked.

Additional information: You can use the same approach on any **moving element** in your scene that has a **RigidBody** to create a **Doppler effect** between the player (*who is listening to the sound*) and the object that will play a sound and is moving closer or farther away from the player.

2. Play sound with Collider:

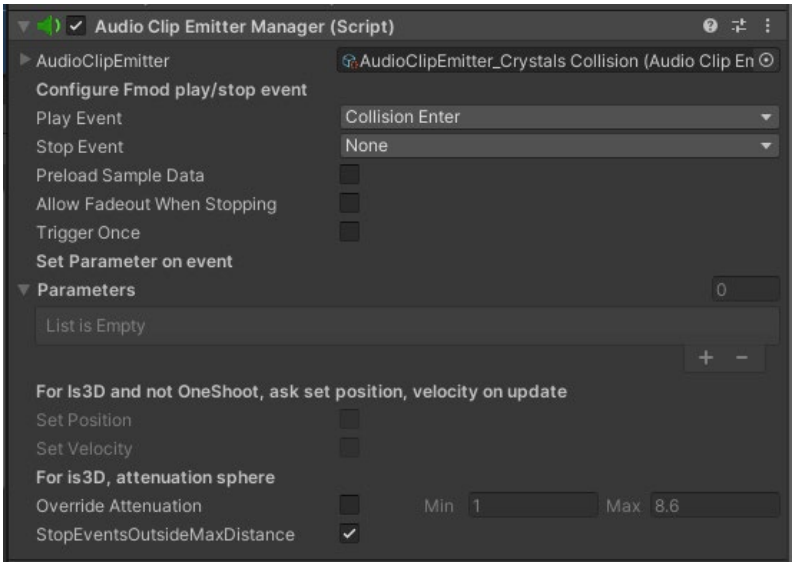
In our example project, we will now focus on the crystals. Each crystal in the game has a **Capsule Collider** component, which is important for the following steps to work.



I want to make a slightly crystalline impact sound play when my **Player collides** with these different crystals (a total of 500 different crystals). To do this, I have a simple event in **FMOD Studio** with my sound asset.

On the crystal, you will need to provide this **event** in the **AudioClipEmitterManager** component.

We should have the following:



I choose to fill in only the Play Event field to determine when the sound will be played. I don't want the sound to stop playing when I'm no longer colliding with the crystal, so I won't provide anything for the Stop Event field. The maximum number of simultaneous sound instances will be defined directly in the event created in FMOD Studio using the **Max Instances parameter**, which I will also set on a bus dedicated to SFX.

Here is what these parameters look like in **FMOD Studio**: