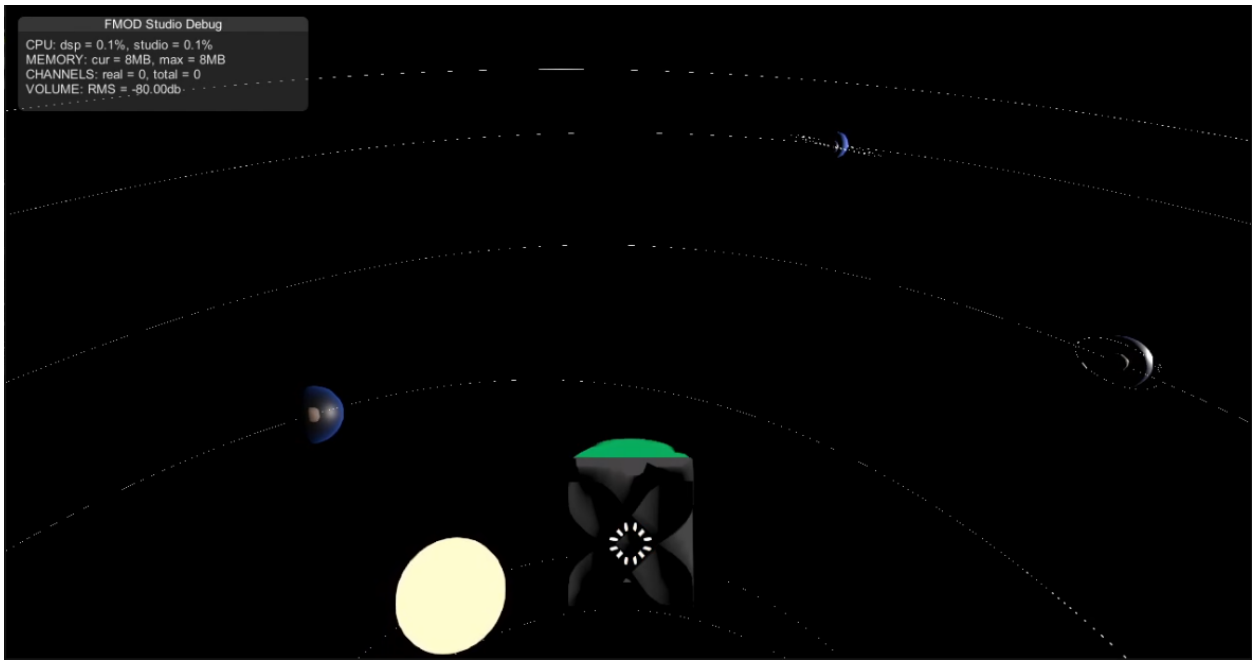
Sound Design Using FMOD for Unity

Sounds of Space

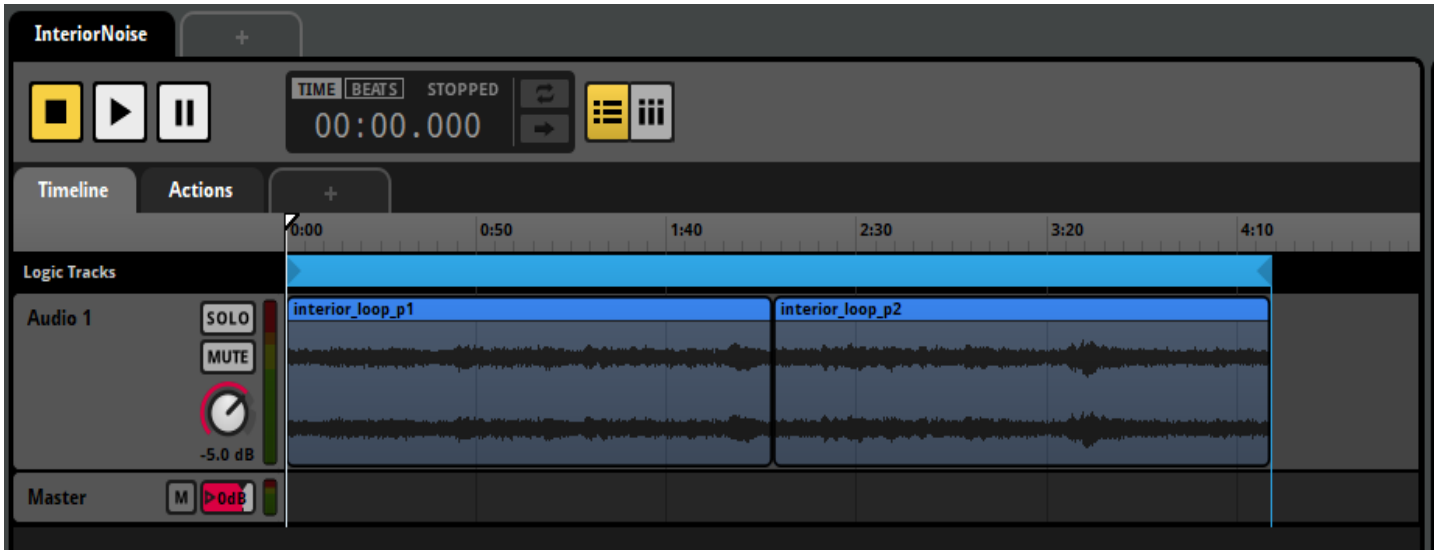
Sounds of Space is a procedurally generated space exploration game where the player searches for beacons that emit sound. The player spawns in their ship in a solar system where they are asked to collect all nearby beacons. Upon collecting these beacons, the player can warp to a new procedurally generated system.



A procedurally generated solar system in *Sounds of Space*.

Choosing an Audio Engine

As sound guides the player to their primary objective in *Sounds of Space*, it was important to create a flexible set of sound events for the player's ship. These sounds need to be playable simultaneously with each other and the sound of the beacons, and some must be loopable. This is why FMOD was selected as the primary sound design tool for this project. FMOD has a large number of built in audio effects, and allows for the creation of loopable audio regions. Additionally, the FMOD editor requires no code to create an event, instead using a familiar DAW layout.



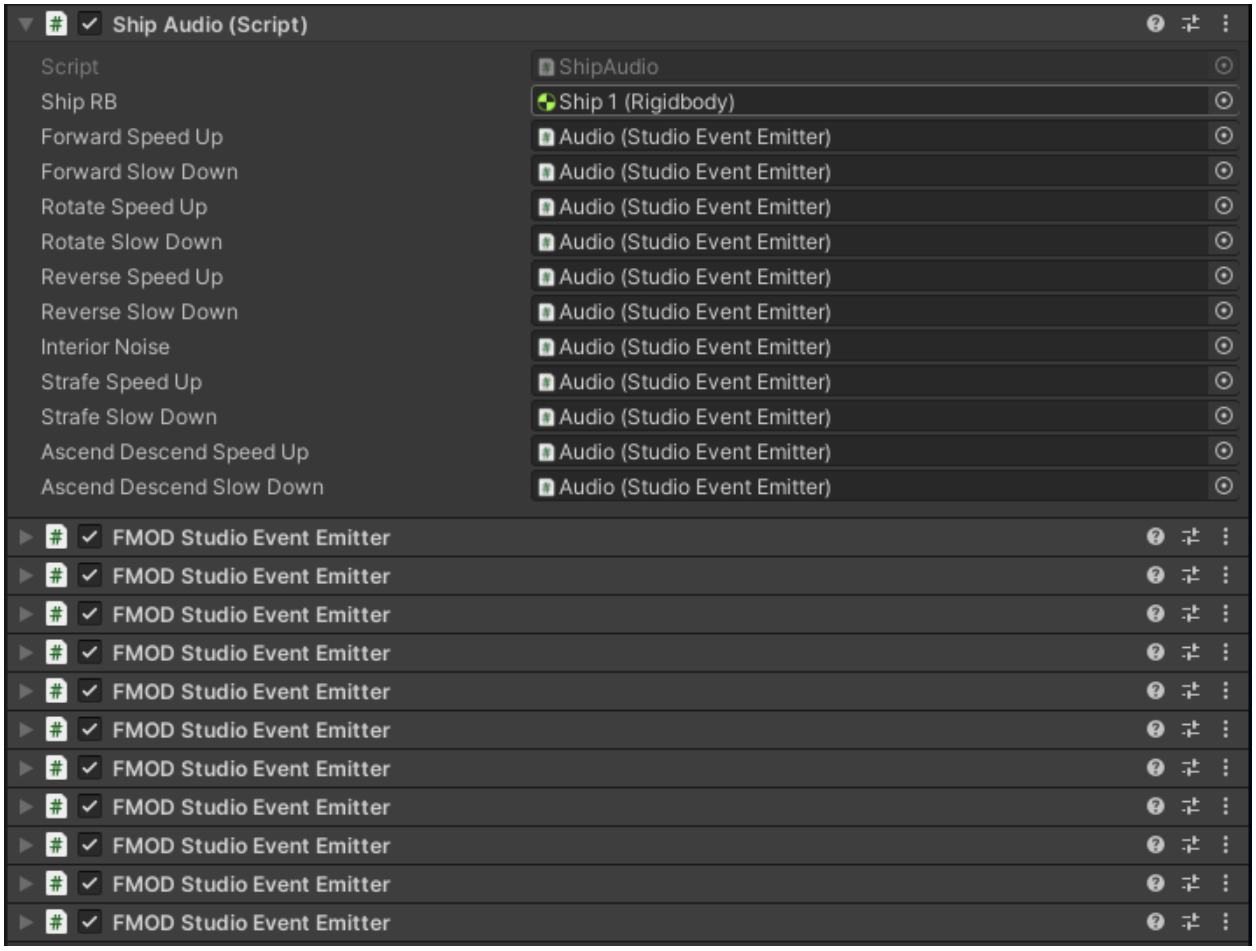
An FMOD event that uses a looping region (the top light blue bar indicates the looped area).

Integrating FMOD with Unity

The ship in *Sounds of Space* requires 11 audio events: speed up and slow down events for forward, reverse, rotate in place, ascend/descend, and strafe, with a single event for the interior cabin noise. Each of these events first gets built in the FMOD application, where each part of its audio is positioned in the timeline, and any initial effects are applied. The dB levels of the different parts of each audio event are balanced against themselves, but not against other events, as this is easier to do in the engine.

An FMOD event with two audio tracks and a looped region. This event also uses a limiter and 3D spatialization audio effect.

After events are created in the FMOD application there are two techniques for utilizing them in Unity, the engine we used for this project. FMOD events can either be created in code, or attached to an FMOD StudioEventEmitter script, which is built into the FMOD for Unity library. This project uses this helper script, as it simplifies the management of each event at runtime, and does the initial setup of building the audio event object itself. However, creating events in your own code can be useful if you want to generate a dynamic quantity of certain events. Regardless of which method is used, FMOD will play all of its events in a separate audio pool from the Unity built in audio, which is used for other parts of this project.



The ship audio script setup used in *Sounds of Space*. Each FMOD Studio Event Emitter is connected to an event created in the FMOD application.

Challenges and Experience

After adding these events to Unity, the primary challenge of this ship audio task became clear: all ship audio events must be playable simultaneously, and opposite events (Ex. forward slow down and forward speed up) must smoothly transition between each other. FMOD has a built-in way to create this behavior, by fading in and out audio events, but in testing it was found that simulating a fade in/out behavior in code was the most straightforward solution. This project uses two coroutines for fading an FMOD event in or out. This allows many fade in and outs to occur at once, which is needed as the ship has up to 10 potential fades that could run simultaneously.

```
107
108     IEnumerator FadeIn(FMODUnity.StudioEventEmitter emitter, float seconds){
109         emitter.Play();
110         emitter.EventInstance.setVolume(0f);
111         float time = 0;
112         while (time < seconds)
113         {
114             emitter.EventInstance.setVolume(Mathf.Lerp(0, 1, time / seconds));
115             time += Time.deltaTime;
116             yield return null;
117         }
118         emitter.EventInstance.setVolume(1f);
119     }
120
121     IEnumerator FadeOut(FMODUnity.StudioEventEmitter emitter, float seconds){
122         float startVol;
123         emitter.EventInstance.getVolume(out startVol);
124         float time = 0;
125         while (time < seconds){
126             emitter.EventInstance.setVolume(Mathf.Lerp(startVol, 0, time / seconds));
127             time += Time.deltaTime;
128             yield return null;
129         }
130         emitter.EventInstance.setVolume(0f);
131         emitter.Stop();
132     }
133 }
134
```

The coroutines used to fade FMOD events in and out.

This project provided a significant amount of experience working with FMOD and connecting FMOD events to a Unity project. It also made clear the importance of identifying not only a functional implementation of a game feature, but one that is efficient and intuitive.