Game audio implementation in Unreal Engine 5

Luca Leopardi
Student number: 16080A

Università Statale degli Studi di Milano
Final project submission for course: *Sound in Interaction*
February 2025

Contents

1 Introduction

This paper illustrates the implementation of audio for a videogame developed in Unreal Engine 5[1], using its in-engine tools for sound design and integration. Cropout[2], a sample project offered by UE5 developer Epic Games for educational purposes, has been chosen as a template to expand upon. The audio solutions originally included have been removed, while the rest of the project has been left as-is, so as to be able to work on a clean canvas for audio while still having to interface with other game systems, and allowing for the presentation of the final result in its full context.

Cropout is a RTS (Real-Time Strategy) game set on a small island. The player has a top-down point of view and can zoom or pan the camera around. Villagers walk around the map and can be instructed to collect resources or build structures. The player wins when they have enough resources to build the Unreal Monument, or loses if they run out of food for their villagers.

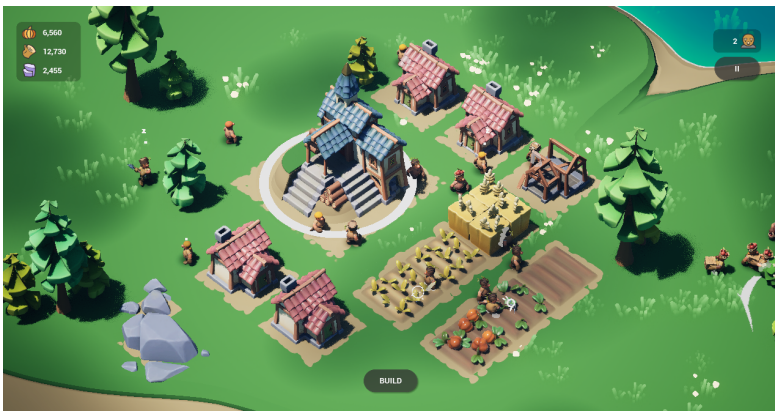

Figure 1: Screenshot of Cropout

At time of writing, the final project source is available on GitHub at: sound-in-interaction.

Note: All files and assets used belong to Epic Games and the original developers of Cropout[2]. The only exceptions are the components whose implementation is described in this paper, which were developed by the author, and the additional audio samples whose respective owners are credited in the asset files themselves.

1.1 Goals

The goal of this project is to fully implement audio in Cropout, to a level comparable to that of a finished game. This means adding features for:

- Sound effects for in-game events;
- Ambient sound effects;
- Believable and functional sound attenuation;
- User-customizable audio settings.

Since the focus of this project is implementation and not design, the original music and sound files have been carried over from the original example project. Additional pre-recorded sound samples have also been added.

1.2 Tools

All audio systems have been developed in-engine in UE5, in particular using Blueprints (Unreal's visual scripting language), MetaSounds[3] (Epic's latest audio system) the Audio Modulation[4] plug-in. Additional free-to-use sound samples have been sourced from SoundDino[5], Freesound[6], Pixabay[7] and Videvo[8].

2 Audio in Unreal Engine 5

Available audio solutions in UE5 and this project's approach

In Unreal Engine 5 sound is handled by the Audio Engine[9] and Audio Mixer[10] modules. The former manages assets, logic and gameplay interfacing, while the latter handles final mixing and rendering. Both expose functionalities to developers via APIs, for use in code or Blueprint.

2.1 Sound sources

Audio sources both inside the game world (SFX, ambient) and outside of it (music, UI) are represented as Sound Sources.

Sound Cues[11] are Sound Source objects that can play audio wave assets, and are bound to game actors or triggered through Blueprints and code. They are mostly a legacy feature, superseded in UE5 by MetaSounds.

MetaSounds[3] is a complete Digital Signal Processing system that offers a graph-based interface to procedurally generate audio or manipulate and play wave assets, and the resulting object functions as a Sound Source. The use of templates and composition is supported.

2.2 Listeners and Attenuation

All localized Sound Sources reference an active Audio Listener in the game scene for spatial processing. The default Listener is placed on the Camera object, but it can be reassigned. Attenuation objects[12] define how the relative position and distance between Source and Listener affects the audio per each Sound Source. Effects include volume reduction, spatialization (panning or binaural), occlusion and more.

2.3 Mixing and Modulation

Audio from Sound Sources is routed through SubMixes and eventually sent to the Audio Mixer for rendering. By default all Sources send to a base Master SubMix. Parameters that alter the signal at mixing stage can be set in multiple ways.

Sound Classes[13] allow to define a set of static parameters to apply to Sound Sources or Mixes. A Class can inherit from another to create a hierarchy. They are a legacy feature, to be replaced by Modulation.

The Modulation plug-in[4] introduces Control Buses, connections from a Sound Source or Mix to Modulation Parameters, which can be grouped and modified in Control Bus Mixes or by code at runtime.

2.4 Chosen approach

For the purposes of this project, the latest features have been preferred over legacy options. Sound Sources have been modelled using MetaSounds, and sound mixing has been achieved via the Modulation plug-in.

Given the top-down POV of the player in a management game like Cropout, traditional positioning of the Audio Listener at the Camera position is not ideal: when it is far away from the map, realistic attenuation renders sound effects inaudible. For this reason, the Listener has been placed along the Camera-map segment, to be moved by only a fraction (0.15) of the Camera's zoom. This way sounds are still spatialized with regard to Camera position, but the impact of vertical distance is greatly diminished.

3 Audio control

Mixing and in-game audio settings

As a basis for the rest of the project, a simple system for audio dataflow and control was developed.

3.1 Routing and mixing

Given the relatively simple nature of the game - featuring a couple dozen of concurrent sound emitters at most - the base Master Submix was deemed sufficient for audio routing. It is assigned to all Sound Sources by default, unless the developer specifies otherwise.

Using the Modulation plug-in, Control Buses were created to control the volume Parameter of different Sound Source categories: Music, SFX, Ambient, UI and Main. Sources belonging to the same group would subscribe to the same bus, while the Main Volume Bus was bound to the output of the Master SubMix.

3.2 User settings

To allow the user to modify Modulation Parameter values and store them in a persistent way, a Control Bus Mix was created.

The Control Bus Mix groups Parameters and allows to save and load values for them from file. Sliders were added in the game's settings menu, with the value of each bound to a Parameter's. On a slider's interaction end, the updated Control Bus Mix is saved to file. The Mix is loaded from file and activated at the game's launch.

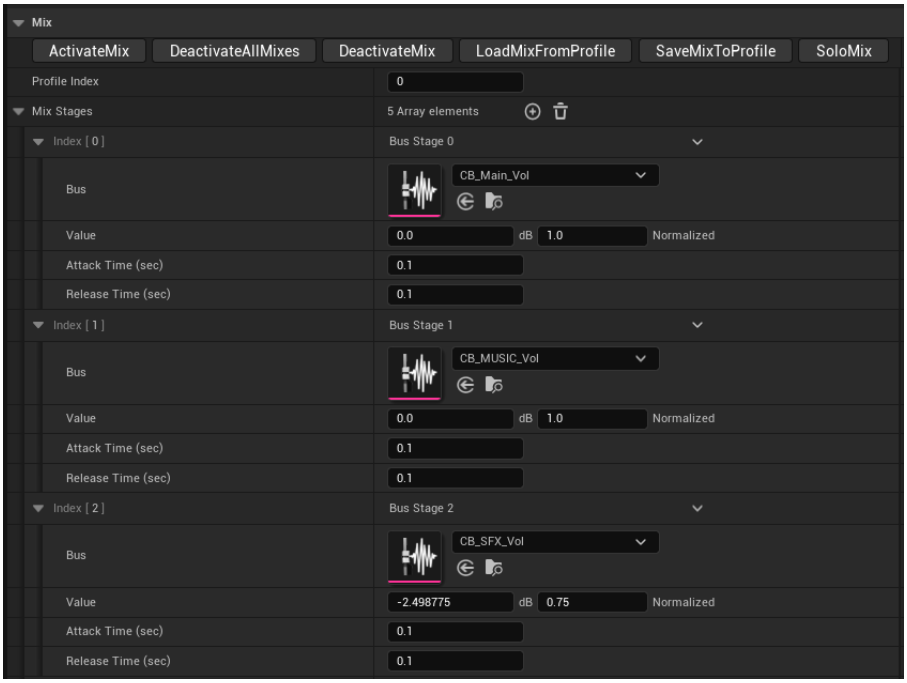

Figure 2: ControlBusMix Object

4 Sound effects

Implementation of in-game and UI sound effects

Playback of sound effects for in-game audio events was handled with a single MetaSound Source template, modified to fit each specific use case.

4.1 SFX MetaSound Source template

MetaSounds allows for the use of templates: from a MetaSound Source a Preset can be created, which is a non-modifiable copy where only the input variables can be changed. The template made for SFX consists of a few simple nodes:

- **Random Wave Getter:** on MetaSound play, forwards a random sound wave asset from the input array, avoiding repeat selections on next play.
- **Random Pitch Shift**: forwards a random float value to be used as pitch shift.
- **Wave Player**: plays the selected sound wave, applying time-stretching pitch shift by the input value. The output is forwarded to the Master SubMix by default.

To the preset are also applied effects shared by all instances in its SFX category:

- **Volume Modulation:** binding to SFX Volume Control Bus.
- **Concurrency:** specifies the maximum number of concurrent instances of this MetaSound, and the policy for their culling. Since game SFX are often of short duration and hence rarely overlap, a maximum number of 12 has been chosen, with the rule to stop the quietest one if this limit is exceeded.
- **Attenuation:** set to decrease volume logarithmically with the distance between Source and Listener. Panning spatialization is also applied.

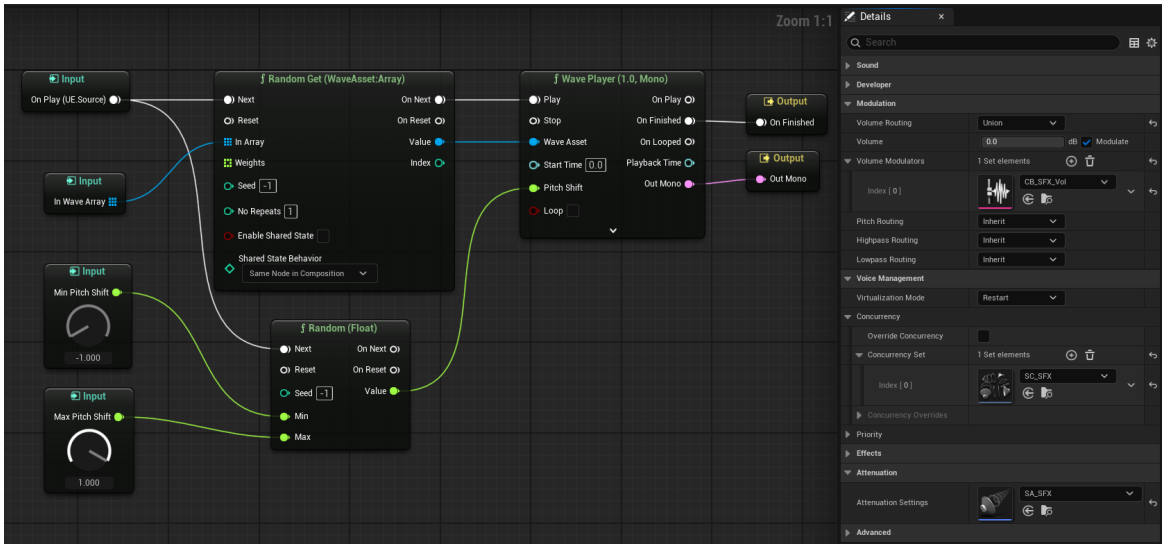

Figure 3: SFX MetaSound template

4.2 Sound Effects

Starting from the SFX template, Presets have been created for each sound effect in the game. For each, an array of wave assets has been selected and the values of pitch shift adjusted to produce varied but realistic resutls. The resulting MetaSounds have been set to trigger with corresponding animation key-frames, like when a villager's axe impacts wood or its feet impact the ground.

4.3 User Interface

For user interface interaction, a MetaSound simply plays a sound wave with no attenuation or pitch-shifting, with the addition of a low-pass filter to give a more neutral sound. Hovering and clicking SFX are played from Blueprints when the corresponding UI events trigger. Volume modulation is routed to the UI Volume Parameter.

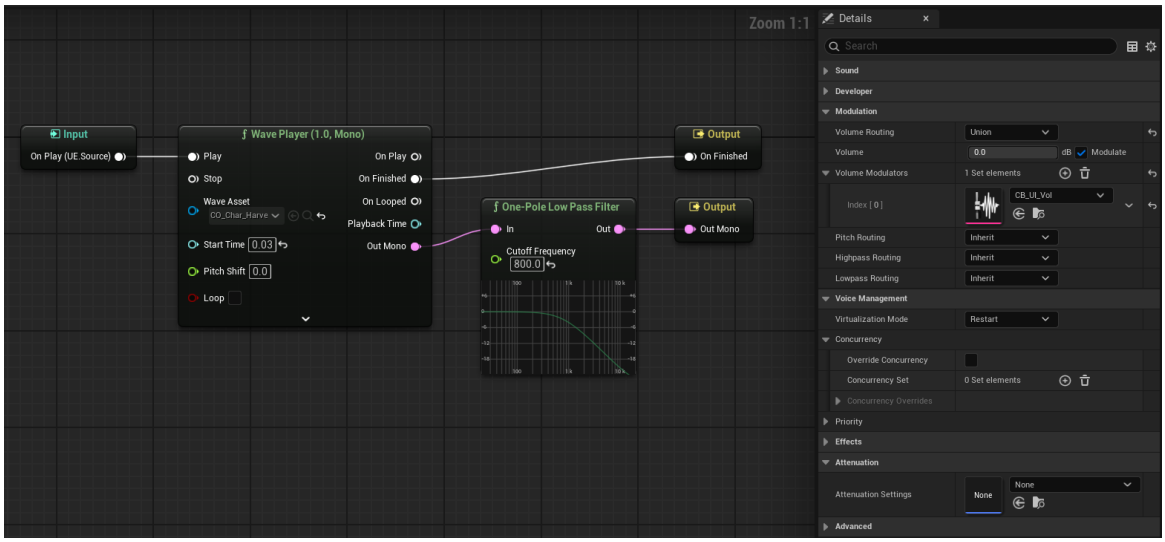

Figure 4: UI SFX MetaSound

5 Ambient sounds

Implementation of environment and background sound effects

Sound effects were put in the ambient category if they are not caused by or important to the player. Their goal is to enchance the immersion in the game world. Since ambient sounds and often playing for long periods of time in the background, a more complex implementation was developed to allow for looping while minimizing repetition.

5.1 Ambient MetaSound Source template

Like for primary sound effects, a template MetaSound was built to utilize the same component for different instances. Volume Modulation is bound to the Ambience Volume Control Bus Concurrency and Attenuation are set here to apply to all Sources deriving from it.

- **Volume Modulation:** binding to Ambient Volume Control Bus.
- **Concurrency:** set to cap the number of concurrent active instances at 16, with the rule to stop the quietest one if this limit is exceeded.
- **Attenuation:** logarithmic decay and spatialization with respect to the Listener's positions set as for primary SFX Attenuation, but specific Ambient sounds overwrite this for different results.

The Ambient MetaSound allows for continuous play of random sound waves from an array, giving options for fade-in/fade-out timings and random start time, pitch-shift, and delay at each loop.

The first step in the graph is similar to that of the SFX MetaSound, with the random selection of a sound wave from an input array. The following steps are implemented using MetaSound Patches to keep the main graph manageable.

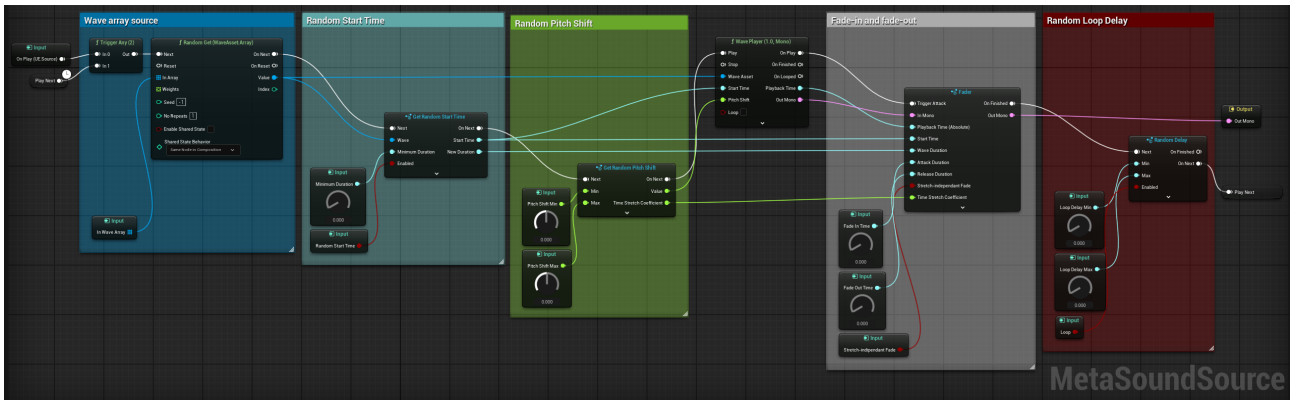

Figure 5: Ambient SFX MetaSound template

5.2 Patch: Random Start Time

The `Get Random Start Time` patch receives in input a sound wave asset and a minimum desired duration, and outputs a Time value for the random Start point and one for the new duration of the track. It does so by selecting a random value between 0.0 and the maximum remaining duration accounting for the given minimum duration. If the latter exceeds the track duration, it outputs 0.0.

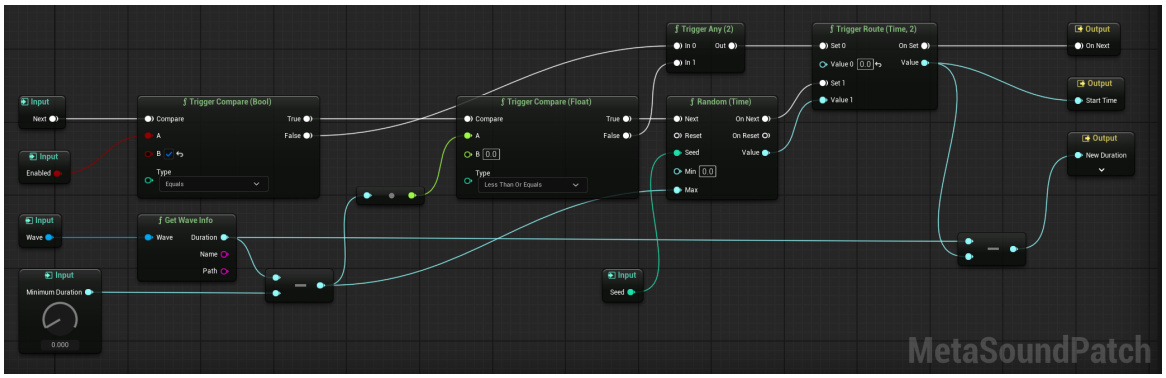

Figure 6: Random Start Time MetaSound Patch

5.3 Patch: Random Pitch Shift

The `Get Random Pitch Shift` patch simply outputs a random float value between the given minimum and maximum, but also gives the time-stretch coefficient that will result from the time-stretching pitch-shift implementation that MetaSounds employs, for later use in the `Fader` patch. It is calculated as the coefficient in the formula $t_{new} = t_{old} * 2^{-PitchShift/12}$.

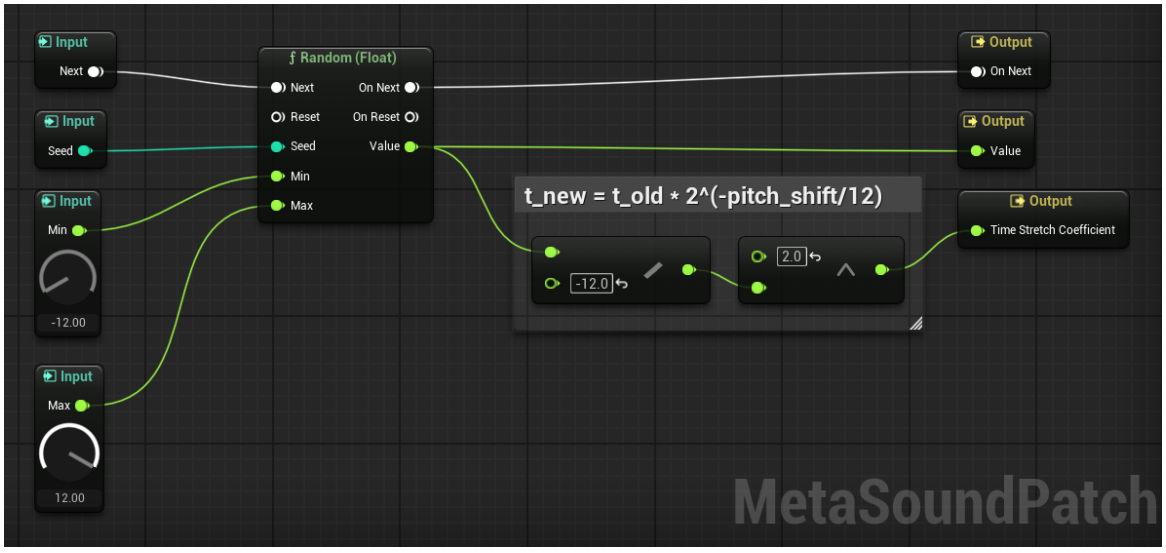

Figure 7: Random Pitch-Shift MetaSound Patch

5.4 Patch: Fader

The `Fader` patch takes as input an audio signal and multiplies it with an envelope with the given fade-in and fade-out durations. These can be absolute timings or scaled by the same time-stretching coefficient introduced by pitch-shifting.

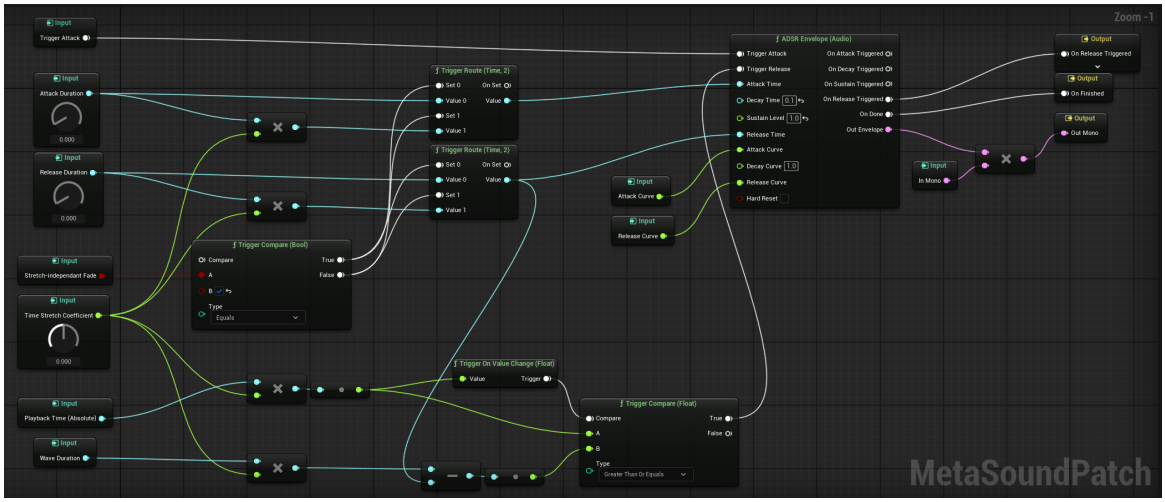

Figure 8: Fader MetaSound Patch

5.5 Patch: Random Delay

The `Random Delay` patch simply takes in a trigger signal and, if enabled, delays it by a random Time between the given minimum and maximum.

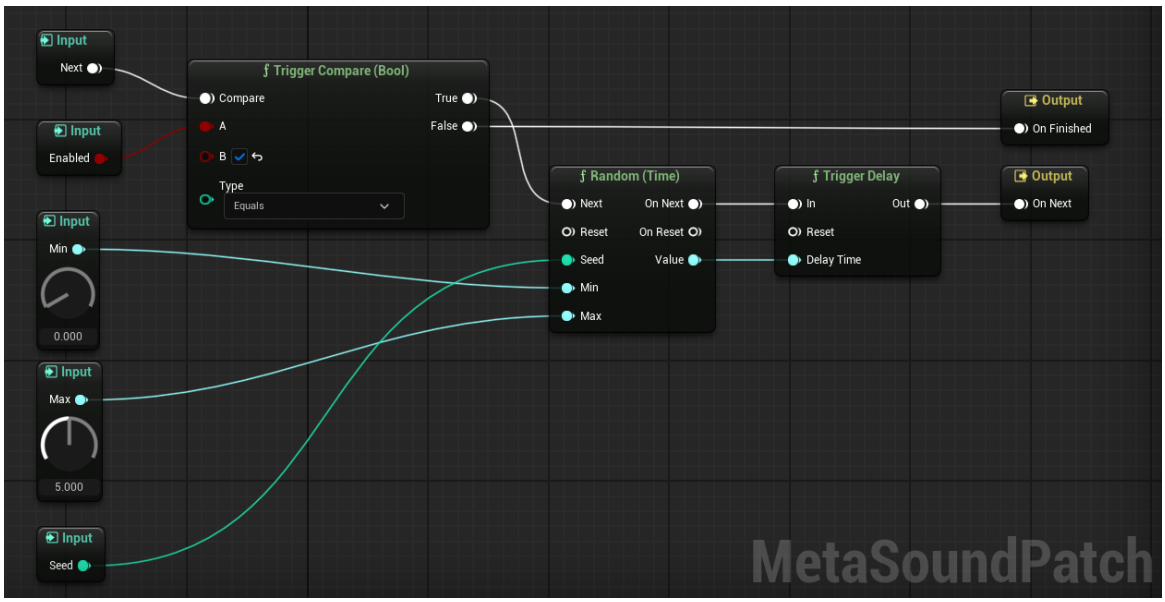

Figure 9: Random Delay MetaSound Patch

5.6 Ambient Sound Sources

MetaSound Presets made from the template but with different inputs have been used to different effects in Ambient Sound Sources:

- **Seagulls:** a MetaSound instance is spawned attached to the particles of the `NS BirdFlying` particle effect. Since they are numerous, the delay between the play loop is increased. Pitch-shift and start time randomization is also more pronounced to reduce the sense of repetition.
- **Waves:** multiple Ambient Sound Sources have been placed on the map around the island. Since the sound is supposed to be constant, loop delay is reduced.
- **Wind:** a single MetaSound is attached to the emitter of the `NS WindSwish` particle effect, with Attenuation set so that only vertical distance to it affects audio spatialization.

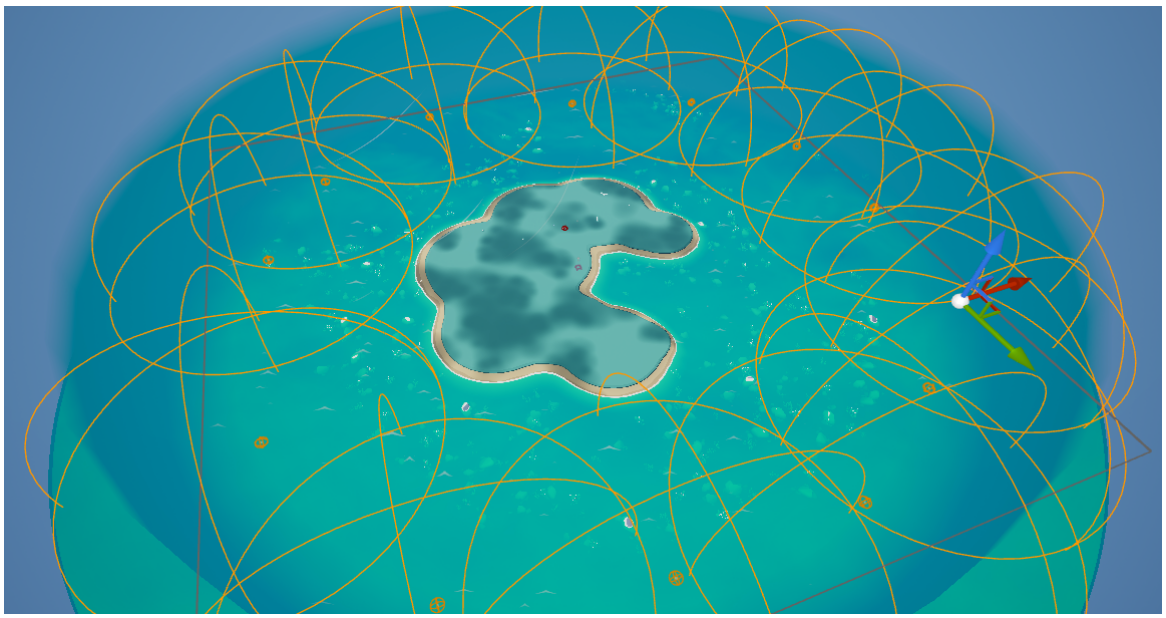

Figure 10: Placement of Ambient Sound Sources for sea SFX

With the addition of ambient sounds, which are often long and overlap, the number of active audio sources at any given time could grow rapidly. To avoid Concurrency triggering and culling sound, all Sources are set to deactivate when the Listener is outside of their Attenuation limit - meaning they are not audible - and to restart computation when they return within range.

6 Conclusion

The end result completes the game with a convincing sound experience, fitting with its RTS-management genre. The adjustments to the Audio Listener are functional to its Camera system, and careful evaluation of the Attenuation parameters create subtle "sound layers" that follow the flow of the game: villagers SFX are most audible and distinguishable when the player has zoomed in and is directly managing their units, and become softer and softer as they zoom out, leaving space to the sounds of wind and seagulls flying by. The addition of ambient sounds greatly adds to the immersion and cozy aesthetic of the game.

Figure 11: A Debug View of the game showing Attenuation volumes and active sounds

As for the implementation, the MetaSounds and systems developed are easily adaptable and scalable to different projects, and have been realized following the latest UE5 workflows recommended by Epic Games.

References

[1] Unreal engine 5 official website. URL https://www.unrealengine.com/en-US/unreal-engine-5.

[2] Croput sample project presentation webpage. URL https://www.unrealengine.com/en-US/blog/cropout-casual-rts-game-sample-project.

[3] *MetaSounds Official Documentation*. URL https://dev.epicgames.com/documentation/en-us/unreal-engine/metasounds-in-unreal-engine.

[4] *Audio Modulation Official Documentation*. URL https://dev.epicgames.com/documentation/en-us/unreal-engine/audio-modulation-overview-in-unreal-engine.

[5] SoundDino official website, . URL https://sounddino.com/.

[6] Freesound official website. URL https://freesound.org/.

[7] Pixabay official website. URL https://pixabay.com/.

[8] Videvo official website. URL https://www.videvo.net/.

[9] *Audio Engine Official Documentation*, . URL https://dev.epicgames.com/documentation/en-us/unreal-engine/audio-engine-overview-in-unreal-engine.

[10] *Audio Mixer Official Documentation*, . URL https://dev.epicgames.com/documentation/en-us/unreal-engine/audio-mixer-overview-in-unreal-engine.

[11] *Sound Cues Official Documentation*, . URL https://dev.epicgames.com/documentation/en-us/unreal-engine/sound-cues-in-unreal-engine.

[12] *Sound Attenuation Official Documentation*. URL https://dev.epicgames.com/documentation/en-us/unreal-engine/sound-attenuation-in-unreal-engine.

[13] *Sound Classes Official Documentation*, . URL https://dev.epicgames.com/documentation/en-us/unreal-engine/sound-classes-in-unreal-engine.