CMP407 Audio Programming Report

Harry Findlay

2102552

# Using your project as an example of past work

Check this box if you’re happy for your project to be used as an example of past work on the module for future students of CMP407

# Video Evidence

# Foundational Requirements

## Sound File Playback

My project contains various looping sounds and audio queues throughout my dynamic soundscape alongside a single one-shot sound, acting as a door entry bell. The looping sounds consist of crickets when outside the house, and when inside the house, the player is exposed to:

* Television static
* Phone ringing
* Lights buzzing
* Dripping tap
* Oven sizzling

I have various sounds within range of one and other, allowing for concurrent playback. This was done deliberately as to help emphasise the audio/gameplay mechanic implemented as the main part of this project. Such sounds include the buzzing lights, the phone ringing and the television static alongside the dripping tap and another light, and the dripping rap alongside the oven audio.

## Spatial Localisation

Throughout the project, Occlusion was utilised to help localise audio to certain areas or rooms. Occlusion was a very important part of my project as the project consisted of a house with an interior and exterior which contained interior and exterior based audio. Continuing, the interior also consisted of two rooms, a ‘living room’ and a ‘kitchen’, again, both with their own audio. I originally opted to use the steam audio extension however, after much research and trial and error, I opted to use the built in Unreal Engine 5 systems as these were sufficient and also helped lower the project size. Additionally, Unreal’s Occlusion was sufficient in getting the task done effectively. The best example is when you enter one room from another and the audio queues shift – this also works when entering or exiting the house.

As mentioned, Unreal Engine 5’s built in Occlusion and sound localisation systems were utilised. I ensured that all audio sources were strategically placed throughout the scene to really make use of, enable, and disable the localisation features that Unreal has to offer. Additionally, audio pans correctly with the camera and makes use of both headphones/earphones that the user may be using, helping the player locate the source of a sound. Of a sound is to the player’s left, the sound will be observed from the left headphone/earphone.

## Attenuation

Attenuation was utilised thoroughly in the creation of the project and, again, Unreal Engine 5’s built-in attenuation features were utilised. Unreal’s attenuation is a very easy-to-use and effective feature as it allows for tweaking and fine-tuning where the used needs. One feature I make good use of was the inner and outer radius’ (or alternative depending on shape of attenuation used). This allowed me to really control where each of the sounds where at their loudest as well as allowed me to control the area of the audio drop off offered by the attenuation.

## Reverb

The project makes use of three reverb zones: one for the house exterior, one for the main room of the house, and a third for the house kitchen.

These were set up as ‘Audio Gameplay Volume’ which is another of Unreal Engine 5’s built in functionalities. These were used to help separate ‘audio zones’ for each key section of the world.

## File Compression Formats

Unreal Engine 5 has a multitude of audio format options to make use of when using audio files. The options available consisted of: Bink Audio, PCM, Opus, Platform-Specific and RAD Audio. Bink is the default audio format assigned to audio files that are added into Unreal Engine. This is due to Bink Audio’s generally good and quick to implement nature that contains a decent compression to sound ratio.

However, after further research, I opted to use the following instead:

* For my looping sounds, the ‘Opus’ audio format was utilised. This is due to its effective compression ratio which allows for the preservation of the audio’s quality, and still has slight compression. Ambient looping audio make use of ‘Bink Audio’ for at instead, as this is less demanding in the CPU and was of less importance/priority as this was external to the house.
* Single-shot sounds make use of the PCM audio format. This is due to it being very resource intensive allowing for the preservation of the sound’s quality, however, as this sound was only utilised once per trigger, this was not much of an issue as the audio was not looped and only triggered once per collision with the front door.

# Sound Files & Mix Quality

I made use of many of Unreal’s started content, audio included. I did, however, source a few audio files from FreeSound.com. When using sounds that were sourced to an external source from Unreal Engine, I ensure to investigate the sound in Audacity beforehand. Firstly, I normalized all of them to -6.0dB to ensure the volume levels of the audio file were consistent, optimised and balanced. I also maintained general maintenance of the audio in ways such as compression and clipping of un-needed silent stretches. Continuing, the audio was then imported into Unreal where it was attached to the audio source’s game object and the volume (min and max) was adjust to fir the scene’s needs.

# Code Focus

The main code focus of the project is the ‘key audio focus tool’. This mechanic divides sounds into two categories: Ambient Audio and Focused Audio. The mechanic starts by continuously checking the distance between the player and the item(s) of interest and if this distance is found to be within the distance check range – the item of interest’s audio source is focused on and all ambient (and other fixed sounds) are muted. This mechanic was inspired by the gameplay mechanic in Assassin’s Creed where the player needs to eavesdrop on NPC conversations.

This mechanic was achieved by making use of Unreal Engine’s Blueprints systems alongside a Sound Mixer and two main Sound Classes. This system also implemented a custom tag-like system, similar to Unity’s system, to help identify the Focused Audio in-range.

Let’s speak about how the mechanic functions:

Firstly, a distance check is utilised to check whether the game object of the focused sound is within a valid range of the player. If this distance is found to be valid, the blueprints of the fixed game object communicate with the Sound Mixer which in turn, sets the AmbientSoundClass’ volume and pitch to 0. Extending form this, a string is then consulted – this string value acting as a tag for the current focused audio’s game object. The focused game objects with non-matching tags are then muted as well.

If the player is found to the have exited the valid focus distance check, the same process is applied however, the AmbientSoundClass’ volume and pitch are set back to 1.

This is a feature that I am particularly proud of, as it pulled me from my comfort zone – having very little prior audio experience.

# References

<https://freesound.org/people/truflabart/sounds/125875/>

<https://freesound.org/people/golovlev.sound/sounds/390193/>

<https://freesound.org/people/zebragrrl/sounds/632227/>

*No generative AI was used.*

# Rubric

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Application Functionality** | **Code Quality** | **Practical Audio Knowledge** | **Written Audio Knowledge** | **Reflection/Evaluation** |
| **A** |  |  |  |  | X |
| **B** | X | X | X | X |  |
| **C** |  |  |  |  |  |
| **D** |  |  |  |  |  |
| **F** |  |  |  |  |  |