# CMP407 – Audio Programming

Presentation – Ollie Hall

# Audio techniques used in this project:

- ✓ Use of audio that you've recorded and edited yourself in an appropriate context.
- ✓ At least three basic audio effects in an appropriate game context.
- ✓ Use of compressed audio file formats or network streaming of compressed audio.
- ✓ Integration with audio middleware or audio facilities within a game engine.
- ✓ Use of digital music (sampled or sequenced) that reacts dynamically to game events.
- ✓ Use of spatial audio techniques to reproduce real-world environments.
- ✓ Use of dynamic audio techniques for sound effects or music.

# The Application

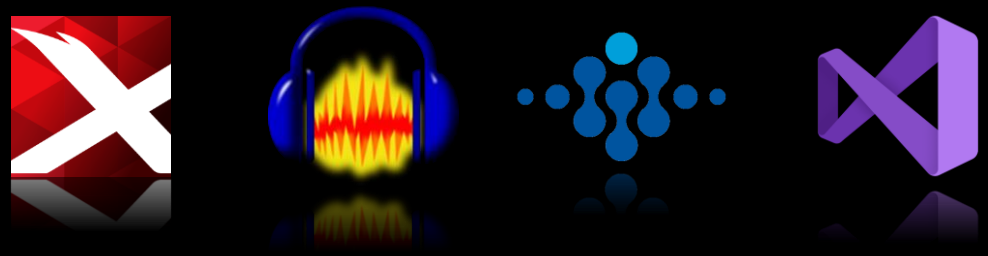
- Created in Unity, using Wwise as middleware.
- A level built as an extension to a game I made in a game jam.
- Fellow student Ben is working on another level for the same game.
- All audio implementation is my own.
- A couple of sounds *may* be shared within both of our projects:
  - Footstep audio
  - Elevator audio
  - Light switching sound audio
  - Object pick-up audio
- The player finds themselves accidentally discovering a secret experimentation chamber, and has to solve puzzles to escape as they are tormented by the chamber's odd caretaker.

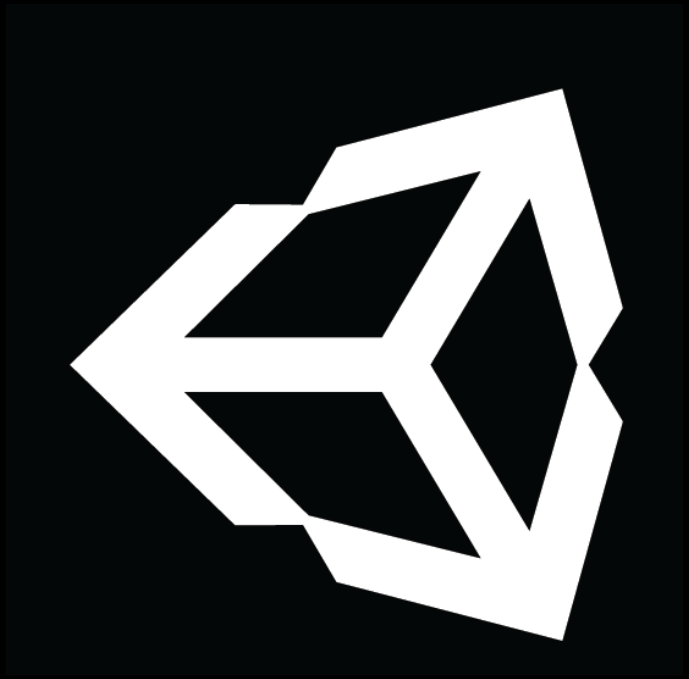
# Use of audio that you've recorded and edited yourself in an appropriate context.

-Recordings of my voice. Pitched down using Audacity and a basic high-pass filter and distortion applied through my C++ application.

- Piece of music created using Mixcraft 9 Pro Studio and a MIDI keyboard, used within a Blend Container in Wwise (more on this later).

- Recording of slow breathing, low-pass filter applied through my C++ application and pitched down in Wwise.





# Integration with audio middleware or audio facilities within a game engine.

- Wwise is used as middleware for my submission.
- The event-based nature of Wwise makes it very useful when interfacing with Unity and Unity's GameEvents.
- I decided to use Wwise only and not Unity's in-built audio handling system as it would create unnecessary complications between Wwise and Unity – however, Unity provides visualizations of attenuation and positioning that you cannot use with Wwise.

# Integration with audio middleware or audio facilities within a game engine.

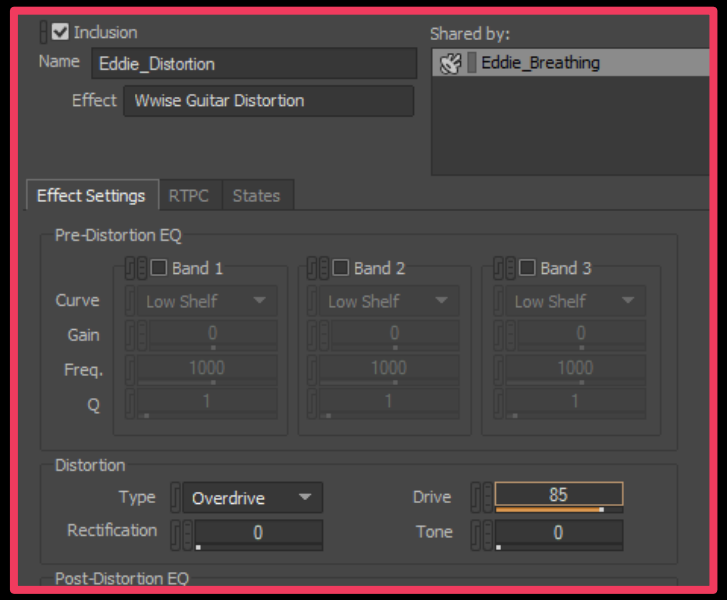
```
Unity Script | 0 references
public class InvokeWWiseEvent : MonoBehaviour
{
    [SerializeField] AK.Wwise.Event WWiseEvent = new AK.Wwise.Event();

    0 references
    public void CallEvent()
    {
        WWiseEvent.Post(gameObject);
    }
}
```

A simple script that can be attached to an object which calls a specified event.

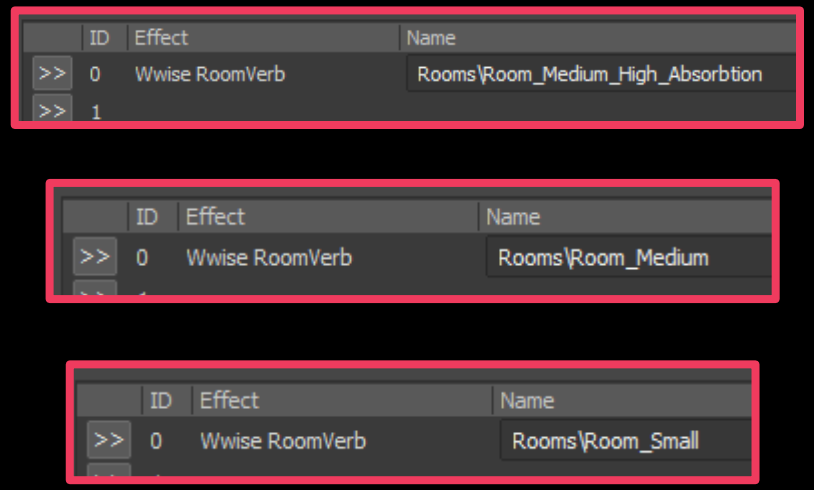
The public function CallEvent() can be called from any event within the game, allowing Wwise events to be played in tandem with the level's event system.

Apart from this script, there are many other uses of Wwise events in code, and other functions such as setting RTCP values.

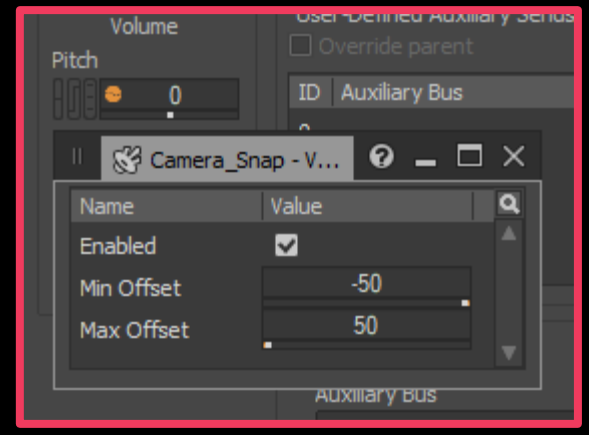
# At least three basic audio effects in an appropriate game context.




A Wwise distortion effect with custom settings to distort a breathing sound effect to sound less human.




"RoomVerb" effects added to multiple auxiliary buses in Wwise. These create an artificial reverb effect for any object within the reverb zone that emits sound.




For many of the sound effects that are reused, I have applied pitch randomization so the sound is slightly different every time without being too noticeable. This makes it feel more natural.

# At least three basic audio effects in an appropriate game context.

```
void Distortion(std::vector<float>& audioData, float threshold)
{
    for (vector<float>::iterator it = audioData.begin(); it != audioData.end(); ++it)
    {
        if (*it > threshold || *it < -threshold)
        {
            *it = fabs(fabs(fmod(*it - threshold, threshold * 4)) - threshold * 2) - threshold;
        }
    }
}
```

I used a basic distortion offline effect to provide a very slight distortion to my voice lines so it sounds like it's coming through an old television

```
float Filter::Filter_LowPass(float input)
{
    float output;
    output = prev + ((input - prev) * coeff);

    prev = output;

    return output;
}

float Filter::Filter_HighPass(float input)
{
    return(input - Filter_LowPass(input));
}
```

I used a basic offline high-pass filter for the voice lines coming through the television, and a low-pass filter for a slow breathing sound effect to remove the higher range and make it sound inhuman.

# Use of compressed audio file formats or network streaming of compressed audio.

The Wwise middleware is used to handle audio file compression for this project.

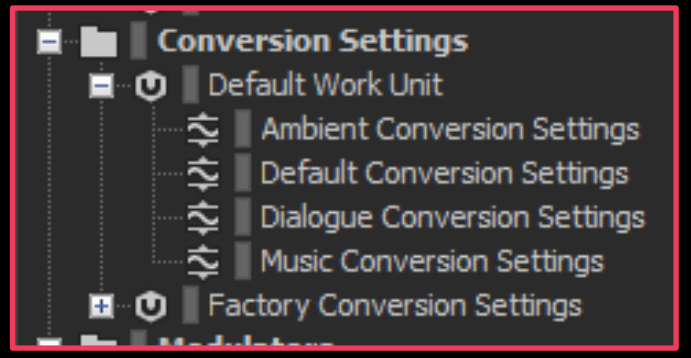
The level has sound effects for a range of gameplay elements and objects.

Most sound effects are under a second long and range from ~20kb to ~750kb. Due to the small size and frequent use, these effects are not compressed, with Wwise handling them as PCM formats. This means they require very little CPU processing power and RAM allocation as they do not need to be decompressed.

Other sounds within the game such as the music, dialogue and ambient audio are compressed with Opus, a lossy audio coding format.

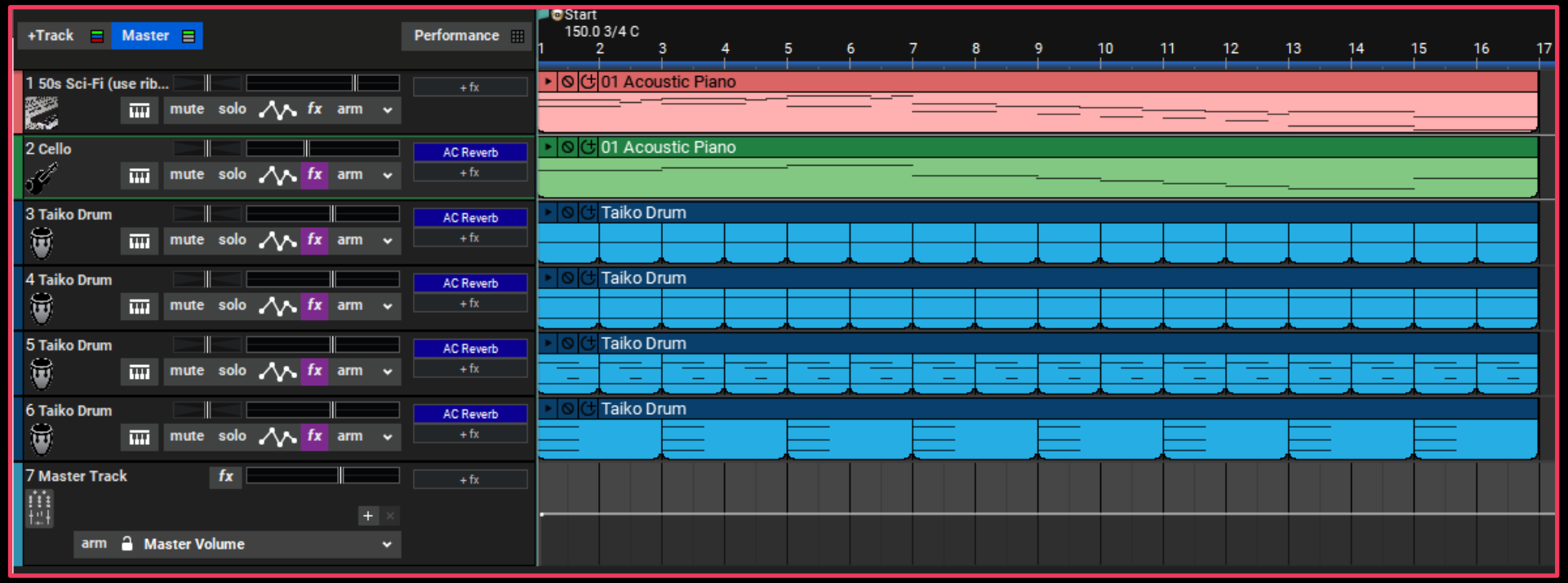
Due to the less frequent loading requirements of these audio files, and larger length and file size, compressing them allows for a reduction in project size without the drawbacks of being too CPU intensive (as compressed files need to be decompressed when they are played).

Ambient effects in the level are compressed with a lower quality than other audio files, due to them being less audible and a high file size.



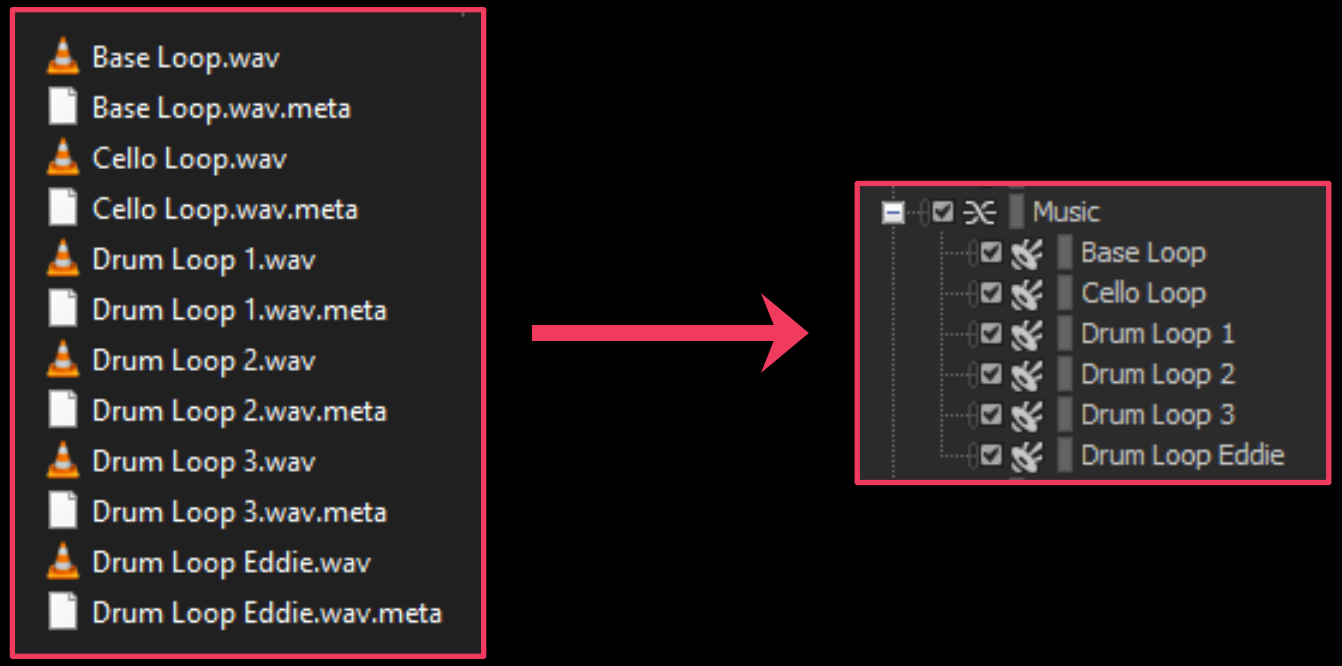

The current conversion settings. Ambient, Dialogue and Music settings are Opus, and Default is PCM

# Use of digital music (sampled or sequenced) that reacts dynamically to game events.



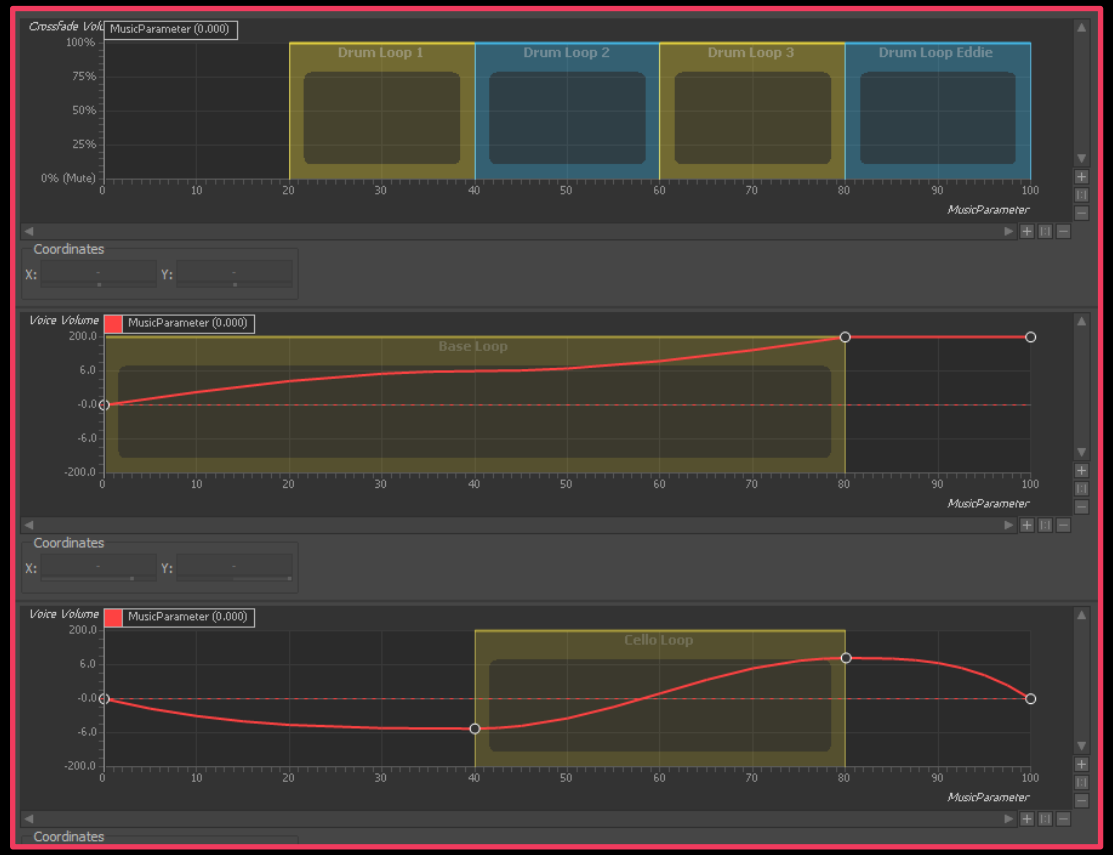

A mix was created in Mixcraft 9 Pro Studio, and a MIDI keyboard was used to assist creation.

# Use of digital music (sampled or sequenced) that reacts dynamically to game events.




Each track within the mix was separately exported as a .wav file. These separate tracks were then imported in Wwise/Unity.

# Use of digital music (sampled or sequenced) that reacts dynamically to game events.




The audio files were put in a blend container. This contains three blend tracks:

- The base music loop
- The drum loop
- The cello loop

These blend tracks are linked to a parameter I've suitably named "MusicParameter".

Depending on the value of MusicParameter (0-100), the drum track changes, and volume of the base track and cello track changes.

The base loop (second blend track) uses an inverted s-curve for the volume so it remains steady in the middle before it increases again.

This allows for **dynamic music** to enhance the intensity of the experience based on the situation.

# Use of digital music (sampled or sequenced) that reacts dynamically to game events.

The “MusicParameter” value of the blend container can be changed from within a script in Unity with this line:

```
AkSoundEngine.SetRTPCValue(Parameter Name, Value, Current GameObject);
```

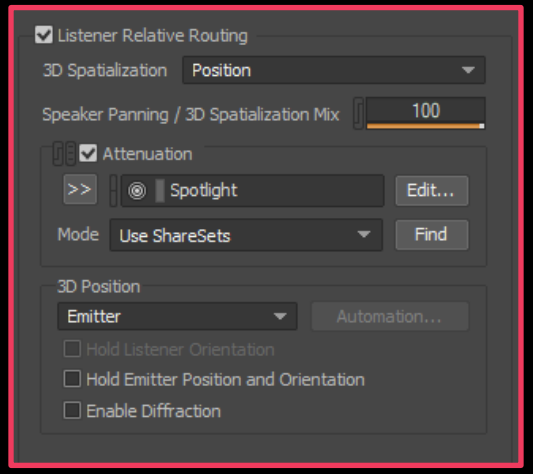
As the game loop is not yet finished, the dynamic music is not yet implemented as intended for the finished level.

# Use of spatial audio techniques to reproduce real-world environments.

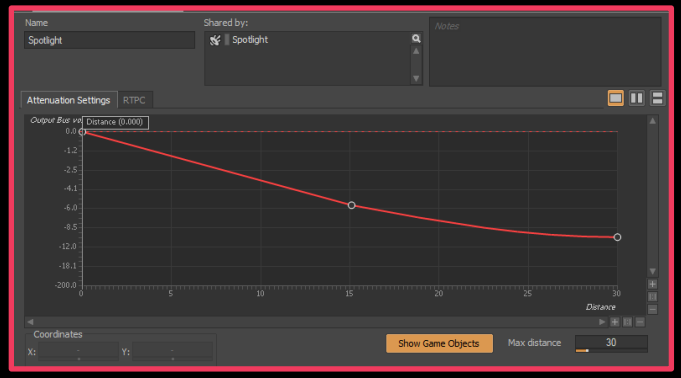
For the mood of my game, I considered it very important that almost every sound the player hears seems as though exists within the space.

Certain moments in the level use spatialization to create an artificial binaural effect, with the player hearing breathing behind them, moving to the left and right.

I used Wwise for audio spatialization. It has extensive options for customising aspects of spatialization such as attenuation, orientation and even diffraction.




The Positioning tab in Wwise. I have enabled position-based 3D spatialization and set up attenuation for my spotlight sound effect.




An example of attenuation in my level

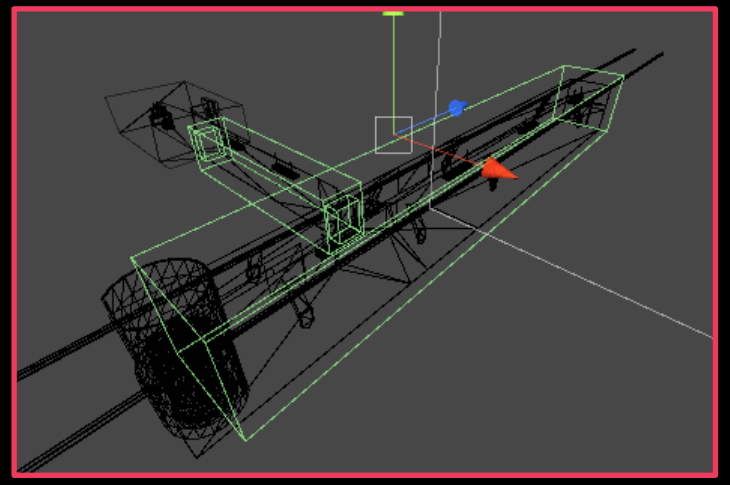
# Use of spatial audio techniques to reproduce real-world environments.

I set up Reverb Zones within my level. Any object within these zones which have a trigger collider and a Rigidbody can interact with these zones.

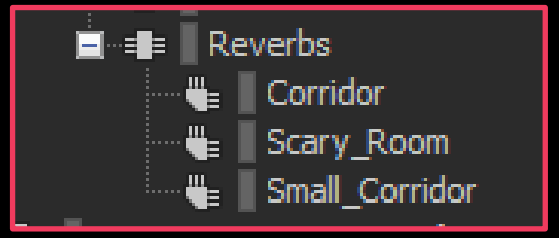
When these objects emit a sound within the reverb zone, the audio is modified to provide a reverb effect that matches the Wwise settings for that zone. This allows me to provide different reverberation effects for different spaces, and even use reverb zones for dramatic effect.

The component AkEnvironment is applied to the box colliders that are the “reverb zones”. This component is linked to the Auxiliary Bus for that area.

The Auxiliary Bus holds the reverb effect for that reverb zone but can contain many other effects ranging from distortion to compression and even pitch-shifting.



The reverb zones and portals within my level in the Unity editor



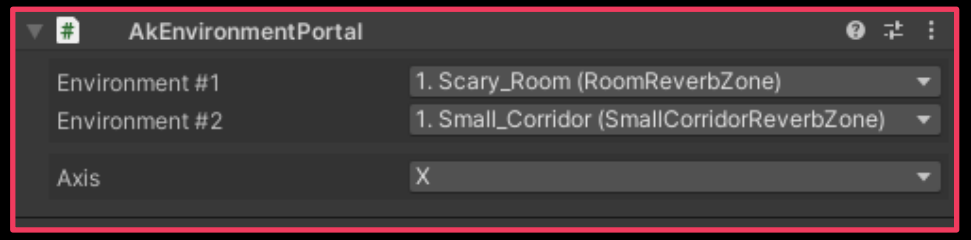

The Auxiliary Buses for my level. Final build may include more.

# Use of spatial audio techniques to reproduce real-world environments.

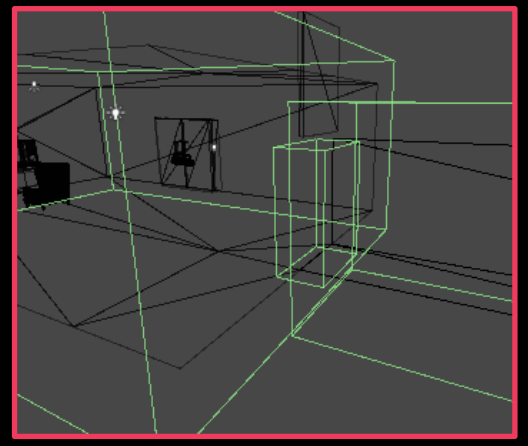
AkEnvironmentPortal is also used within my level between each reverb zone / AkEnvironment.

This allows me to connect the reverb effects in different environments to allow for a smooth, seamless transition between reverb zones.

The AkEnvironmentPortal determines the strength of the effect from each connected AkEnvironment by the distance from the player, so if the player is nearer to the corridor area, the corridor's reverb zone effect will be more prominent.




The AkEnvironmentPortal component within Unity. This one is connecting a small corridor with the escape room which has more reverb.



The portal linking my small corridor and escape room reverb zones.

# Use of dynamic audio techniques for sound effects or music.

The player can interact with a radio within the level.

This radio can be turned on and the frequency can be tuned using two dials.

Certain frequencies can play pieces of music such as Vera Lynn's "We'll Meet Again" or Flanagan and Allen's "Run Rabbit Run"

The player needs to find a specific frequency that contains a clue for the player to escape.

# Use of dynamic audio techniques for sound effects or music.

The radio effect is created through a Blend Container with four different Blend Tracks.

The tracks are linked to a Game Parameter named "*RadioFrequency*". This is set from code within the "*RadioControl*" script in Unity.

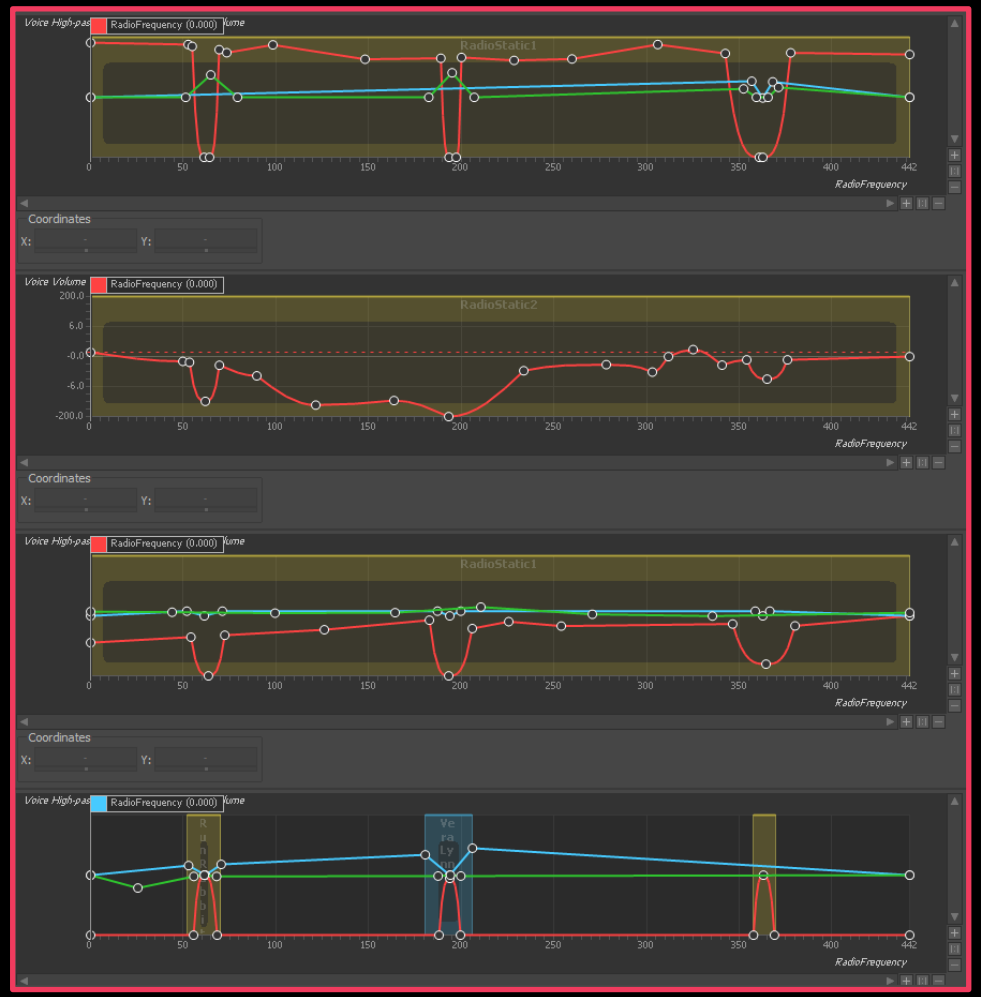
The first blend track is a layer of white noise / static which changes pitch, high pass and volume based on frequency.

The second track is another layer of static, with a different white noise effect.

The third track is a third layer of static, but this layer slightly changes volume, pitch and high pass depending on frequency.

The fourth layer are the "stations", certain songs and the voice lines with the clue are played if the parameter "*RadioFrequency*" is on them.

The volume, high pass and pitch changes based on how finely the player is "tuned in".



# Use of dynamic audio techniques for sound effects or music.

```
Unity Message | 0 references
void Update()
{
    if (selectedKnob == null)
        return;

    if(Input.GetKey(KeyCode.A))
        DecreaseFrequency(selectedKnob);

    if (Input.GetKey(KeyCode.D))
        IncreaseFrequency(selectedKnob);
}
```

```
1 reference
public void IncreaseFrequency(Knob knob)
{
    // Increment selected knob's rotation
    knob.rotation += 1;

    // Cap rotation at 360 degrees
    if (knob.rotation >= 360) knob.rotation = 360;

    RotateKnob(knob);
}

1 reference
public void DecreaseFrequency(Knob knob)
{
    // Decrement selected knob's rotation
    knob.rotation -= 1;

    // Cap rotation at -360 degrees
    if (knob.rotation <= -360) knob.rotation = -360;

    RotateKnob(knob);
}
```

```
1 reference
void SetFrequency()
{
    // Knob 2 changes frequency by a smaller amount, giving it finer control
    frequency = (knob1.rotation / 5) + (knob2.rotation * 0.01f);

    // Ensure the smallest frequency value is 0
    frequencyValue = (int)(frequency * 3) + 226;

    // Set the 3D text to represent the frequency
    SetFrequencyText();

    // Set the blend container RTCP value based on the frequency
    AkSoundEngine.SetRTPCValue(frequencyRTCP, frequencyValue);
}
```

# Audio Credits

Film static 1 – joedeshon
Film static 2 – InspectorJ
Radio power button – LexzachGames
Ceiling Fan switch – InspectorJ
Camera – tmkappelt, kwahmah_02
Air vent – livvy0221
Lock dial spin – egomassive
Lock metal groan – Crinkem
Lock metal clang – jasonLON
Lock metal thud – zaneclampett
Lock unlock – sergiosmarinis
Fluorescent lamp – jacekksiazek

Standing up – kupp2
Sitting down – FreqMan
Sliding door – stereostereo, musicandsound, beerbelly38
Video glitch sound 1 – Free-Rush
Video glitch sound 2 – syn2x-v0
Video glitch sound 3 – tadaizm
Video glitch sound 4 – AmicaSys
Spotlight - Deathscyp
Video background ambience – AniCator
Elevator descend – Robinhood76
Elevator door open - waxsocks