University of Portsmouth

School of Creative Technologies

Final year Project undertaken in partial fulfilment of the requirements for the BSc (Honours) in Music Computing.

The Music Box: An Audio-based Video game

By

Benjamin Peter Jones

UP818988

Supervisor: Stephen Pearse

Project Unity: CT6CTPRO

May 2019

Project Type: **Artefact**

Abstract

The Music Box is an audio-based video game that minimises visual graphics to measure how effective 3D audio is to communicate the location, orientation and setting of a game world. The project was created using the Unity Game Engine with Wwise audio middleware integration to handle the audio. The aim of the project is to understand the importance of audio in a 3D game world environment split into the three areas of; location, orientation and setting. The development process was carried out alongside a user testing group that offered feedback to meet the aims of the project with a secondary focus on enjoyable gameplay. The findings were that precise location and orientation were difficult to communicate through the use of only sound design but, general locations of game objects were easily found by following the source of the audio. The project also found that creating a setting is completely achievable through the use of everyday listening techniques and applying this to design effective audio. This creates a picture in a player's mind about how the game environment looks despite the lack of visual graphics.

The Music Box: An Audio-based Video game

By

Benjamin Peter Jones

Project Unit: CT6CTPRO

Supervisor: Stephen Pearse

I hereby declare that this dissertation is substantially my own work

I consent to my dissertation in this attributed format (not anonymous), subject to final approval by the Board of Examiners, being made available electronically in the Library Dissertation Repository and/or Department/School digital repositories. Dissertations will be kept for a maximum of ten years.

Yes	BPJ	No		*Please tick/Initial as appropriate*

Table of Contents

Chapter 1: Introduction

The Music Box is an aurally focused interactive puzzle game that explores new ways of implementing communication through sound design in a 3D environment. The supporting documentation explores how to create effective 3D audio and how it is applied to the artefact. Information and everyday listening are studied as secondary subjects to get a better understanding of how to apply these in the video-games industry, as sound in a video-game is simply nothing more than gameplay feedback to the player (Beauchemin S., 2019). Gameplay feedback is a form of direct communication between the player and the game world. Dynamic audio complicates the traditional diegetic-nondiegetic division of film sound (Collins K., 2008) leading to many different uses for game audio using sound effects or music to communicate many different attributes of the game such as; setting, danger, location and emotion.

Digital Signal Processing (DSP) is one of the main application areas that are processed on graphical processing units. The Fast Fourier Transform (FFT) often forms a core part of those processing algorithms and very computationally expensive (TELFOR, 2015). But as hardware becomes more powerful and available, FFT and other DSP algorithms become easier and faster to handle leading to more creative and real audio-based solutions which, *The Music Box* explores by using audio zones and using vector mathematics to change real-time game parameters.

The aim of the project is to understand the importance of audio in a 3D game world environment by limiting the visual aspects of the game so that visual elements are only essential if there are no audio solutions available. Since the sense of touch is unavailable when playing a video game, it is the main focus to fill this lack of information with audio-visual elements. The aim of the project is split into three areas; the location, orientation and setting of the game. As the project is minimising visual elements, aural communication is the vital focus to described to the player how to navigate *The Music Box*.

Chapter 2: Literature Review

2.1 Short history of 3D Game Audio Communication

As the project is focused on first-person perspective games, looking into the history of game audio is a good start for the project to be successful. The most famous first-person shooter that pioneered the genre was DOOM (id Software, 1993). It utilised very little locational sound design and was heavily focused on music whereas Duke Nukem 3D (FormGen, 1996), added in noticeable stereo audio where game objects seemed to act as emitters of audio with relatively realistic and useful panning information.

The Playstation 2 (Sony, 2000) and Xbox (Microsoft, 2001) era which, was the year 2000 to 2006, added a huge opportunity for developers to take advantage of using 3D audio as a communication method. Bethesda Softworks developed The Elder Scrolls III: Morrowind (Bethesda Softworks, 2002) which, included a lot of 3D diegetic audio to communicate to the player what is going on in their surroundings. This was particularly effective when enemies were offscreen because the player could hear when there is danger about. Another great example would be Unreal Tournament (GT Interactive, 1999). This first-person shooter utilises 3D audio in a great way to communicate to the player where enemies are. As an arena-based first-person shooter, Unreal Tournament had all the players as emitters from their guns and the character's voice. As the game is incredibly fast-paced, it was very useful to be able to hear where there is gunfire and enemy players delivering their catchphrases or calling out if they're low on health.

As games developed into the next generation of consoles, the Playstation 3 (Sony, 2006) and Xbox 360 (Microsoft, 2005), first-person shooters reiterated their previous generation's counterparts into popular competitive online games, for example, the Call of Duty series. Call of Duty: Modern Warfare 2 (Activision, 2009) had a perk system in the online game which, changed the behaviour or grant positive effects to the player. The 3D audio became very useful because players could tell the distance and direction of a gunshot or footsteps to a good degree of accuracy. The developers included perks and weapon attachments that deal with transforming the communication of audio. The perk Ninja Pro removed all

footsteps that the player emitted allowing the player to sneak around easily. The perk Sitrep Pro increased enemy footsteps by a factor of four and reduced teammates' footsteps by a factor of four. Also, almost all of the weapons had the ability to attach a silencer at the expense of reduced damage. The competitive nature of the previous titles (Modern Warfare (Activision, 2007) and World at War (Activision, 2008)) pushed the developers to focus more on the 3D spatial audio to aid the player in finding out where there are enemies to track down.

The modern era of games hasn't changed a huge amount in terms of audio design but the hardware available has made the audio capabilities much more sophisticated creating much more accurate positioning within a 3D environment. Hunt: Showdown (Crytek, early access 2017) is a first-person shooter game that takes advantage of this sophisticated 3D audio and uses Wwise to handle the audio in CryEngine.

Hunt: Showdown's sound design is described as more than half of the overall focus on design with the intention to encourage the player to listen to the world and what the game world tells the player (Magnus Larbrant: Creative Director for Hunt: Showdown, 2017). Whatever sound plays in the game has a real 3D world connection and it actually means something (Florian Fusslin: Audio Director for Hunt: Showdown, 2017). Players in Hunt: Showdown have been recorded to deliberately make as much noise as possible in order to draw other teams towards them to ambush and hunt them back (Chris Auty: Level Design Director for Hunt: Showdown, 2017). Florian Fusslin also talks about how they used audio to describe the setting of the game as hot or humid and essentially, in his words, the game "stinks" to create a believable Louisiana Swamp.

This evidence shows the fundamental importance of sound design in first-person perspective games as a clear motive to listen to the game world to be successful in the game's objectives.

2.2 Everyday Listening and Audio Information

Information theory (Shannon & Weaver, 1949; Pierce, 1980) is a good example of theory consistent with the view of information as stuff (Gaver, 1988) that is “a difference that makes a difference” (Bateson, 1972). Gaver also goes on to describe that “[information] implies a need for a memory” to integrate the difference in attributes between the information received and the current information known of the source.

Gaver uses the example of a car driving behind a person in a narrow alley to describe the portrayal of information in everyday listening in his document “*What in the World do we Hear?: An Ecological Approach to Auditory Event Perception*”. He describes that the listener will hear a large and powerful engine, it’s coming from behind and that there is reverb from walls displaying that the listener is in a narrow alleyway. He uses this as his example of Everyday Listening rather than Musical Listening which, would assume the listener is focusing on what sound it specifically makes rather than the information it conveys.

Chapter 3: Methodology

3.1 Software Used and Justification

3.1.1 Unity

The project was made in the Unity Game Engine (Version 2018.2.0f2). The decision to do so was purely based on personal experience within the software. The other candidates for the game engine would have been Unreal Engine or CryEngine which, have been praised as better game engines for graphics as CryEngine outshines Unity's graphical capabilities and are equivalent to Unreal's (ThinkWik, 2018). But as the project is a blindfolded game, graphics are only taken into consideration when considering user interface elements.

Unity is simpler to use because there are fewer tools than Unreal or CryEngine but more importantly, Unity uses C# for its programming language which is easier to use and learn. Unreal uses C++ which, is more powerful but harder to learn (Raka Mahesa, 2018). CryEngine wasn't considered for the project due to the availability of the software in the current working conditions.

For the context of the project, using Unity is the best choice. The software is easy to use and is relatively basic in the available tools but with access to them through the Unity Store if there is a need for more tools. The main development of the project was done using code and Wwise as there was no cinematics and minimal animations. Having an easy to use and basic game engine compared to Unreal Engine makes Unity the ideal choice. Unreal Engine most likely would have slowed down development time for the same end result.

3.1.2 Wwise

The project uses Wwise (Version 2017.2.6) as its audio middleware. Similar to the game engine decision, the choice was based on personal experience but, the choice was done so willingly because Wwise works incredibly well to exceed expectations for what the project needed. Wwise is middleware that has to be integrated so that it creates soundbanks for

the game in the same way that FMOD does but, the User Interface between the two is largely different.

Wwise is designed to be unique as a piece of sound integration software whereas FMOD, on the other hand, is designed like a DAW (Digital Audio Workstation). This does mean that Wwise will potentially take a long time to learn if you're a music composer or sound designer but, Wwise becomes very powerful when the user learns how to use it well. This is better for use by an audio programmer or game audio developer as a game is not a linear piece of media (Mongeau, 2016). Understanding how to manipulate game audio using Wwise's integration becomes very rewarding as the amount of control over audio rarely meets any technical limitations.

Wwise makes the best candidate for the audio middleware chosen to drive the audio of the project. The project is an audio-based game so, the audio needs to be exceptionally effective at communicating audio information very well. The increased control over something simple such as footsteps randomly changing can really place the player in the game world. This is very easy to implement in Wwise by using random containers which then each sound effect can still have its own individual editing in Wwise and their own game parameters controlling attributes on the audio file.

3.1.3 Audacity

Audacity was used to cut and edit all the raw sound files into processed individual audio files. Audacity has a multitude of effects that the user can apply to an audio file or to a part of an audio file. More than one audio file can be imported and mixed separately from the other. The most important processing that Audacity has for the project is the noise reduction effect. The user can define a noise profile which Audacity seeks out within the audio and reduces the amplitude of the noise profile dependant on the user's specification. This was incredibly important for the project because each audio file needed to be clear to sound exactly like what it is attempting to portray. This is especially the case when it came to the project's ambient sound effects as a lot of the recording could not have been done in an isolated area.

Normalisation, fade in and fade out processes were also applied to the audio files to make the audio files sound cleaner and have smoother transitions. Normalisation almost always brought up the volume of the background noise so, noise reduction was very important for all audio files that needed normalisation.

Logic Pro X was also the available piece of software that had all the desired effects that the project needed to process and edit the audio files. Logic Pro X was not needed though because it is too complex for editing and processing single audio files despite normalisation, fade in, fade out and noise reduction is available in the software. Its main use is a digital audio workstation (DAW), and therefore has a lot of unnecessary functions for editing just audio files. It has much better use at arranging multiple audio files that can be used for ambience sound effects. For example, in the context of the project, if multiple bird tweets were recorded individually instead of general garden ambience, the user could design garden ambience by piecing together all the bird tweets and mixing them separately. There is a larger amount of control but it is ultimately unnecessary because the end result would be similar.

3.1.4 Other

Three pieces of other software were used to aid the project. These include Alchemy and Logic Pro X to create the non-diegetic task correct and incorrect audio and Adobe Illustrator that was used to create the images of the map.

3.2 Agile and Waterfall Methodology

The methodology for the project was largely using waterfall. Waterfall methodology is generally the methodology used for solo developer projects because it employs a sequential design, build, test and deploy process (Glowtouch, Unknown date). Agile methodology was implemented during the user testing phase when constructive feedback came from the users and which prompted the start of ongoing testing throughout development. After the user testing, it was documented what was wrong with the project and what feedback needed to be addressed (this can be found in Appendix 3).

The sprint after user testing referred back to the feedback (Appendix 3) and the original design document (Appendix 1). A huge downside to the project was the lack of structure that ended up in the confusion of which methodology to use. In hindsight, the project should have decided on which methodology to proceed with from the design stage. The decision should have been Agile so that there was a community around the development of the game. Community feedback is common in game development and games that use community feedback become tailored towards the audience of the game.

3.3 User Testing

User testing was done with the user who could then offer written and verbal feedback. Appendix 2 is a google forms document of two user testing stages. There should be a third and final one to test the final iteration of the game but, poor time management and technical challenges led to the final user testing to not being carried out.

Doing user testing with the user was a positive experience because it highlighted the need for many elements that are in the game. Watching a user attempt to complete the game led to additional features such as the map and the audio panning of the wall rub. This was because the users had to be told how to complete certain points in the game who would then offer feedback on how to tackle the issue or make the game easier but still challenging.

Chapter 4: Design:

A brief description of the game would be an audio-focused game that minimises visual elements where applicable.

4.1 Key Features

Originally, the game idea was a text-based adventure on the surface but within a blind 3D environment. The development changed to test how much a first-person perspective game needs visual information to be able to communicate the gameplay process to the player by getting rid of visual information entirely. The key features needed to achieve this are:

- Believable audio to create a game world.
- A task system for gameplay.
- Minimal UI to fill in for the sense of touch.

4.2 Sound Design

To describe the setting to the player, sound design is the only way to do so. Referring to the Design Document (Appendix 1), the sound design described is realistic and/or hyper-realistic. Using the sound design purely as a means of locational data communication between the game world and the player. Describing the world to the player in the game turned out to be rather simplistic by taking real-world recording and isolating them within the game.

4.2.1 Ambient Sounds

Using code, the audio sources for the traffic ambience and garden (bird song) ambience follow the player because they use Wwise's spatialization. Without the sound source following the same axis as the player, the sound source becomes an object within the scene rather than being an ambient sound effect. These two ambient sound effects describe locational data about the current setting and therefore create a general position of the

player. The player can still hear both sound effects while within the house but faintly. This is important because the player can follow the ambience to position themselves at the front garden or in the back garden.

4.2.2 Footsteps

As the main source of the setting description, footsteps are very important for the game. In the first iteration of the game from the feedback gathered from user testing seen in Appendix 2, a user explained that they were expecting to hear different types of footsteps depending on different surfaces. The footsteps are used to explain which part of the game world that they are in which, ultimately produced five different types of footsteps with eight variations of each type.

The spotting sheet within the Design Document (Appendix 1) shows the sounds recorded to create different footsteps. Varying from using plastic bags to a bathroom scale while emulating the walking motion (heel first then toes) with a trainer on each different surface. Some footsteps were created by layering different footsteps on top of each other such as using the wet footsteps sound and tile footsteps sound to create wet tile footsteps with some tweaking of the volume. The same technique was done with the stair's footsteps and the grass footsteps.

4.2.3 Task Sounds and Miscellaneous

As with the rest of the game, the task sounds also have a very obvious sound design. They're all designed to copy their real-life audio and, in most cases, it is recorded using the real-life audio counterpart. Namely, the plughole, taps running, wall rubbing, indoor and outdoor hitting and the clock ticking.

4.2.4 Lack of Visual Elements

As the game is designed to be heavily focused on aural feedback, the lack of visual elements is a must. Working with the user group, the game received small UI designs to fulfil the key feature; minimal UI to fill in for the sense of touch.

Fig 1. The UI object for hover over interaction.

The sense of touch is impossible to convey in a video without visual elements. Standard controllers such as keyboard and mouse reduce complex embodied responses to a simpler and more easily managed relationship between vision and touch (Eugénie Shinkle, 2008). Describing touch through sound design is much more complicated because the locational data that audio carries is not specific enough, as shown through the user testing. A lot of the users from the user testing found it very difficult to pinpoint the clock on the crosshair in the second iteration of the game. None of the users got past that specific task.

The solution to this was to create a UI object that explains that there is an object to interact within front of the player as seen in fig 1. Inspiration was taken from *The Elder Scrolls IV: Oblivion*.

Creating a map for the game was also crucial for the player to complete the tasks in the game. The main issue was the second and third task (the generator and the bath) because these sounds were many walls apart from where the player was to go to them. If the player were to follow the noise to the generator (despite there being audio occlusion through the wall), there was very little information that told the player to go through the back door into the garden to reach the generator. Adding the map was a fair justification that helps the gameplay become easier but still relatively challenging.

Chapter 5 Implementation:

5.1 How the software was used for implementation

Audacity and Logic Pro X are an honourable mention as secondary pieces of software used as mentioned in chapter 3.1.4. Audacity was used to cut and edit the raw audio files into their respective counterparts. For example, in Appendix 1, the file "STE-004.wav" is a collection of footsteps made by emulating walking on carpet with a trainer. The file was processed by putting the whole file into Audacity, cutting each hit and exporting them to create individual audio files for a footstep.

Audacity has a multitude of effects that a developer can apply to edit a sound. In the case of the project, the audio files were edited to clean up the audio into better versions of themselves. This was done by normalising to -1db and applying a tiny amount of fade in and fade out at the start and end to stop audio glitches and smoothen out the transitions between audio files. Audacity has an effect called noise reduction which was incredibly useful for the project. Having no way to isolate certain sounds, such as the bird song and traffic ambience, noise reduction can eliminate any unwanted noise in an audio file. To do this, highlight the noise that needs to be eliminated and click on noise reduction to create a noise profile. When the noise reduction is applied, Audacity looks for the noise profile and reduces it to the user's defined specification. Too much reduction creates audio glitches in the form of frequency sweeps in the desired audio and too little will leave too much noise in the audio file. Trial and error were done to find the optimal specifications.

Logic Pro X was simply used for Alchemy, a very powerful piece of synthesis software. A pre-set was used to create the task completion and wrong non-diegetic audio. Logic Pro X was then used to record and bounce down the audio into a .wav file.

Adobe Illustrator was used to create the onscreen maps by changing the background colour to black and creating the map using the line tool and text tool. To export, it was simply screenshotted using MacOS' screenshot shortcut (Shif+CMD+4).

5.1.1 Unity Implementation

Unity's design using game objects and materials is very user-friendly. This is especially useful for a developer who is primarily an audio programmer and designer. 3D objects were added into the game scene by simply through the process of right-clicking->3D Object->… This can be done in the scene view or in the hierarchy window. The project makes use of a lot of 3D objects and was done by adding cubes, transforming them and then duplicating them, renaming in the hierarchy window to their respective positions, for example, *Garden Fence North*. The floor objects were done by adding a plane in the 3D object selection. Materials are created in a similar way by right-clicking and clicking add new material. Materials were used in the project to structure the game world and only used simple colours. If everything were the same colour, it would be impossible to determine the difference between game objects which, would make it very difficult to design the game flow even though the game is an audio-based game.

When the game world is constructed, tags were added to objects if need be. Tags are the most useful part when it comes to interaction between game objects when programming the game using UnityScript. Objects and their components can be found within scripts by using *GameObject.FindWithTag()* and *GameObject.GetComponent<>().* Get component was used to refer to the Player object's CharacterController. The CharacterController is a simple way of controlling a game object that doesn't need to use physically realistic controls. This is ideal for the project because the movement only needs to be very basic. Using the CharacterController, the player's movement was calculated on a per-frame basis in the CharacterController's.*Move()* function which, takes a Vector3 as an argument (in this context, a Vector3 is a position and direction within a 3D environment).

The CharacterController is used in calculating when to play the footsteps of the player. The CharacterController stores the current velocity of a game object. By using this information, the Footsteps function is only carried out when the velocity of the player game object is above zero. Looking into the Footsteps function, a footstep (or wall hit) is played every time the variable *stepCycle* reaches or is above the variable *stepCycleThreshold.* And by making a simple counter using *stepCycle += Time.deltaTime* the footstep audio is played based on

time rather than frames. In the context of the game, a footstep is played every 0.55 seconds. This can be found and set in the inspector view of the Player game object.

To create the audio zones, a cube is added into the scene then fills the area where the audio is going to potentially be changed. These cubes fill in the areas of the game world such as the garden or the living room. Taking off the mesh renderer on these objects and making the collider a trigger, the player can walk through the object and trigger three functions: *OnTriggerEnter, OnTriggerExit* and *OnTriggerStay.* Those three functions are used in code to compare tags of objects to see if they're entering, exiting or staying in the other object's collider. In the context of the project:

OnTriggerEnter: Used to determine which footsteps audio and which wall hit audio to play.
OnTriggerExit: Used to reset the state of hit wall audio to off.
OnTriggerStay: Used to set the state of the Wall Hit to play the wall rub and where all the panning calculations take place for the wall rub.

5.1.2 Wwise Implementation

As mentioned in chapter 3.1.2, Wwise is the chosen middleware to implement the game's audio. Wwise supplies its own library (the Audio Kinetic library, Audio Kinetic being the creators of Wwise) of code to connect the Unity Game engine to the soundbank that Wwise generates which, can be read by the compiler when writing the code for scripts in Unity. The game uses three functions from the Audio Kinetic library; *PostEvent*, *SetState* and *SetRTPCValue*. The most important and most used being the Post Event function which, is as follows:

AkSoundEngine.PostEvent("*Name of Wwise event*", *Emitter*);

This function plays the audio corresponding to the user-defined event that was created in Wwise. The emitter is the game object that the developer would like the audio to be emitted from, making use of 3D spatialization (see chapter 5.1.2.1 below) unless the audio is 2D stereo. In the case of 2D stereo, any game object can be the emitter as it will make no

difference, but it is good practice to use *gameObject* which, is a reference to the game object that the script is attached. For example, in the game's context, the arguments for the footsteps event are "*FootstepsEvent"* and *gameObject* ultimately meaning that the footsteps audio get emitted from the player though, in 2D stereo.

Within Wwise, there is a large amount of control for the developer ranging from playing a one-shot audio file to complete control over interactive musical compositions. The most important aspect of Wwise for the project being the random container because of the large amount of variations in Footsteps and wall hits. The random container is one that contains more than one audio file. Triggering the event attached with the random container will choose a random audio file within the container each time the event is posted. Very useful to reduce the effect of repetition on a sound effect such as footsteps. Footsteps in daily life have slight variations even when walking on the same surface therefore, using a random container and recording multiple footsteps is a very good solution at emulating real-life footsteps.

There are other sounds that do not play just once. In the general settings of a Wwise *Sound SFX*, there is a checkbox called *Loop*. The generator, the clock ticking and bath taps loop their audio in this way infinitely. This is why it's incredibly important to have a game sync state that controls the volume of the tasks' audio. For example, the flow of the generator is to play the turn on audio once, loop the generator audio, and then reduce the generator audio to inaudible and play the generator off audio once when the player interacts with it. In the general settings tab, the initial delay setting is used to play the generator in sequence. Both generator turn on and generator on events are posted at the same time but the delay lets the generator turn on audio finish before playing the generator on audio.

5.1.2.1 Spatialization in Wwise

To aid in the creation of believable audio, the game uses Wwise's attenuation editor in the positioning tab. The spatialization is the most important part to communicate the location of the game objects to the player. Wwise has a very easy to use the spatialization system

with its own inbuilt attenuation preview that was used in the game to create a general amount of spatialization within Wwise and then tweaked after user testing.

Each object has its own 3D spatialization apart from the footsteps, wall rub and door opening which, are all played in 2D stereo. The decision to have those sounds in 2D stereo is because they're too close to the audio listener for any spatialization to be heard. The game could have spatialization on the footsteps, but it would have taken too much time to create an audio source for each foot and animate them for there to be any substantial effect on gameplay or immersion. This is especially taken into consideration because the game is blind, meaning that the player's imagination will fill in the lack of visuals of the player character moving.

The sounds using 3D spatialization were carefully tailored using the attenuation editor to create hyper-realistic audio which, communicates the location of the game objects within the game world. The most "hyper-realistic" piece of audio in the game is the clock ticking. It was recorded by pressing the recorder up to a clock as close as possible to the ticking sound source, the ticking was completely inaudible unless the listener was within centimetres of the source. The clock in the game world becomes an obvious and audible source when within the living room. Using Wwise, the spatialization controls the amount of Volume and high and low pass filter dependant on the distance between the listener and the source.

For more specific location of audio sources, cone attenuation has been used when 3D spatialization has been implemented. Cone attenuation adds extra control to the developer to change Volume and filter values when the listener is facing away from the sound source. By increasing the hipass and lopass filter values within the cone attenuation editor, the game achieves a simple emulation of the outer ear because the outer ear performs a significant transformation of an incoming sound front aiding in localisation (Batteau, 1967). The cone attenuation pushes the player to find the sound sources and seek them out which, are essential to complete the tasks. And in the case of the ambient sounds, the garden and traffic, the player is pushed away from the ambient sound sources because they need to explore in between them. This is particularly effective with the traffic as the player is aurally

pushed directly towards the front door and then met with the visual UI interaction to open the front door.

5.1.2.2 Use of States and RTPC in Wwise and Unity

Wwise and Unity can talk to each other using real-time parameter control (RTPC). It is relatively simple to set up in terms of creating an RTPC in Wwise and implementing it into UnityScript. In Wwise, under the game syncs tab, three states and one game parameter have been made for the project. The states determine the volume of the respective parameter. For example, the *Carpet* state in *FootstepsSwitch* sets the volume of the carpet footsteps to +/-0db and reduces the volume of all other footsteps so they're inaudible.

The *TaskController* state does as it is named. It controls the volume amount of the task game objects. Task 1 is clock on, task 2 is generator on and task 3 is the bath on. The other two game objects are inaudible when it's not their respective task. This is to not confuse the player with multiple task audio sources going on so that the task at hand can be focused on by the player especially as the clock ticking noise is relatively quiet. The code to implement the state change is very simple and works effectively:

AkSoundEngine.SetState("*Name of state group*", "*Name of state to change to*");

This would usually be implemented in an if statement that needs a certain condition such as clicking on the clock in this context. But, there is the initial set state in the *Start()* function to start the clock ticking at the start of the game. The other set state functions occur when their respective task is to change; Generator to turn on and clock to turn off when the clock is interacted with, bath to start running and the generator to turn off when the generator is interacted with, player to turn off taps and pull plug and music box turn on when the bath is interacted with.

The game parameter *WallPan* is used to set the panning of the wall rub audio. Calculated every frame in code, the panning helps position the angle of the player against the wall so they can determine whether they are walking into the wall or away from the wall. It has a

range of -180 to 180 where, -180, 180 and 0 are dead centre, 90 is full left and -90 is full right. The code is:

AkSoundEngine.SetRTPCValue("*Name of game parameter*", *value to set*);

The game parameter is used on the wall rub sound SFX under the Audio tab where a user-defined graph is made which, has set the values to those stated above.

5.2 Code of Interest

For a small project, there is relatively a large amount of code that takes advantage of Unity's C# library; UnityScript. Naturally, when creating a 3D environment within a video game, the game uses a lot of code using Vector3 mathematics to calculate direction and positions of game objects. The simplest form of this being in the traffic ambience (TrafficAmbience.cs) and garden ambience (GardenAmbience.cs) scripts that allow the respective sound source to follow the same x-axis as the player (line 19). In doing so, the sound source stays in reference to the spatialization that you would hear in real life. As mentioned in chapter 4.2.1.1, if the sound source was to not follow the player and be static, it becomes an object within the scene rather than acting as an ambient sound effect.

If statements are a fundamental part of a software's logic. They allow the flow of the program to be changed, and so they allow algorithms and more interesting code (Alex Allain, 2019). Switch cases act similar to if statements but with better clarity of reading and speeding up the compiler when there is multiway branching (Unknown, Unknown). The player's audio response to wall and door objects is handled in the player controller in the *OnTriggerEnter* function. Simply by creating a switch case that checks the collider's tag, the state of the footsteps (FootstepsSwitch) and hitting doors (HitWall) can be changed depending on the audio zone that the player is currently entering. The audio zones are created by objects that don't have meshes and use their colliders as triggers.

5.2.1 Object Interaction code using Raycast

To create the object interaction code, a simple system using Raycast was put in place. A Raycast in this context is essentially a drawing a line between two vector positions which stores information of the first object that the ray hits. Using the stored information, checks can be made using if statements and the objects tag in Unity.

The main object interaction script (objectDetectionScript.cs) was attached to the image of the handover (Fig 1) because *CrossFadeAlpha(…)* and *CrossFadeColor(…)* was needed for gameplay purposes. *CrossFadeAlpha* (lines 33 and 46) was used to make the image appear and disappear depending on whether the player is hovering over an interactable object. *CrossFadeColor* (lines 36 and 40) was used to change the colour of the image to grey (the new colour in line 23) from white (the original colour of the object in line 22) when the player clicks the left mouse button when hovering over an interactable object to indicate that they're interacting with said object.

Using an accessor of the mouse down Boolean and a reference to the object interaction script, the player controller (PlayerController.cs) can set conditions depending on which object is being interacted with using the raycast system. Checking the tag of the object that the raycast is hitting, the player controller sets conditions to true and sends Booleans to the respective object script so that the Wwise can handle the audio events that are being posted on those objects. For example, at line 147 in PlayerController.cs, the if statements check if the raycast is hitting the Generator object and if the mouse down Boolean from the object detection script is true. When these two conditions are met, the player now possesses the bedroom door key. In terms of the game, this means that the generator turns off, a pick-up key sound is played, and the bath taps audio starts playing pushing the player to investigate the bathroom. All the tasks and audio are done in this way.

5.2.2 Audio panning calculation using Raycast and Physics.OverlapSphere

The most complex piece of code in the project spanning over fifty lines of code to determine the panning amount for the wall rub audio. **Please refer to the PlayerController.cs script from lines 263 to 320.** A brief explanation of the result would be the amount of panning calculated from the angle between the centre of the player to the closest point of the wall and the ear of the player to where it hits on the wall.

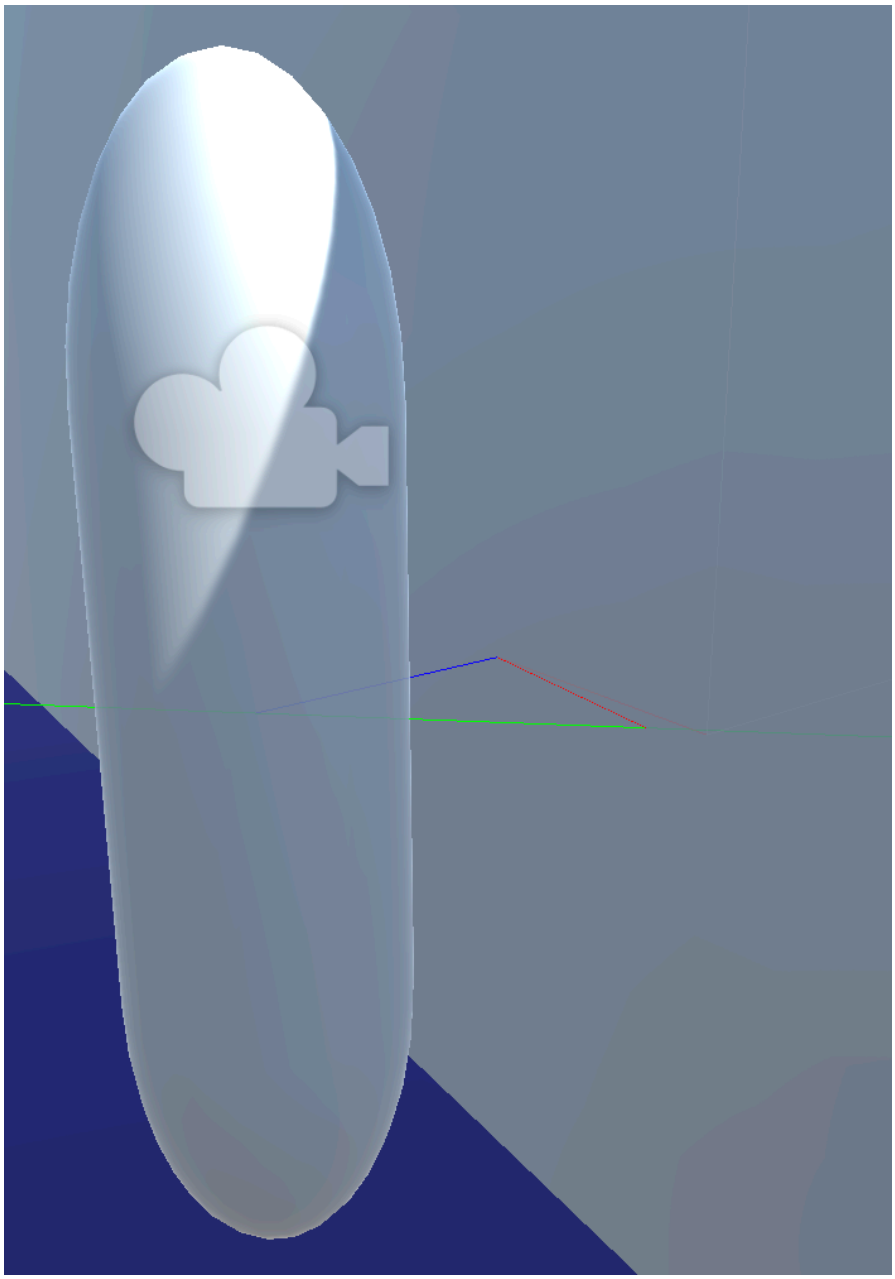

Fig 2. Showing the *Debug.DrawRay* raycasts for the panning system

Blue ray = The centre of the player to the closest point on the wall.

Green ray = The centre of the player to the direction of where the ear would be facing (directly right in this case).

Red ray = The point where the blue ray hits the wall to the point where the green ray hits the wall (not used in the equation but for display and debugging purposes).

Using unity's library for vector equations, the angle between the blue ray and green ray can be done with the function Vector3.SignedAngle(*Green ray, Blue ray, Vector3.up*) in lines 307 and 308. But first, the rays have to be calculated and created before being able to put them into the angle equation.

For the green rays, there are two rays because there is one for each ear, they are calculated by creating a new Vector3 which has the direction and magnitude of five on the x-axis (or minus five for left) by simply using the code in lines 266 and 267. The Vector3 is then transformed into local space using the *transform.TransformDirection(…)* function in lines 269 & 270. Transforming the Vector3 into local space is crucial because the rays then follow the rotation and position of the player game object. Now that the Vector3 position from the ears is calculated, the ray can be created by specifying an origin and direction using the player's position as the origin and the previously calculated Vector3 for the ears as the direction.

Using *Physics.OverlapSphere* in line 297, the closest position to the wall can be calculated which, is needed to create the blue ray. *Physics.OverlapSphere* (line 297) is an array of all the colliders inside a sphere at an origin of a given radius. In this case, the sphere has a radius of five and its centre is the player's position. Using a for loop (line 299), the code searches through all the colliders within the overlap sphere and checks if the tag is *WallObject,* the tag that all the walls have. A Vector3 (*wallsClosestPos*) can then store the position of the closest point on the current wall's collider in the for loop which, will be used to draw the blue ray. The blue ray's origin being the player's position and the direction being the difference between the wall's closest position and the player's position.

To then calculate the angle between the blue and green ray, as mentioned above, the *Vector3.SignedAngle* function is used. The arguments that this function takes is the vector **from** the angular difference is measured, the vector **to** which the angular difference is measured, and the axis that the other vectors are rotated (Unity Technologies, 2019). As mentioned above, in lines 307 and 308, the three arguments that project puts into the signed angle function to calculate the angle is: the direction between the hit of the green ray and the player's position; the direction between the wall's closest position and the player's positions; and the final argument being the rotation around the y-axis. The result is a floating-point number between the value of -180 and 180.

The two floating-point numbers that have just been calculated (named *leftAngle* and *rightAngle*) are used to set an RTPC in Wwise called *WallPan* (as mentioned in chapter 5.1.2.2). The angle represents the amount of pan in the left or right ear. 180, -180 and 0 are centre, 90 is full left and -90 is full right. The user-defined graph is what determines these values and are updated in real-time hence the use of real-time parameter control (RTPC). As the value is calculated and set in every frame, the transition of panning from the player turning away from and into a wall is very effective at communicating the rotation of the player allowing the player to easily work out the direction they need to face to walk away from a wall into the middle of the room.

Chapter 6 Evaluation:

6.1 How the game performs

- The game performs very well with a constant frame rate.
- There is no code that slows down the game.
- The lack of complex graphical rendering is a large factor that helps with performance.
- There are no audio glitches in the final iteration of the game.

Overall, the performance of the game is a success and is polished in terms of the implemented material does what it is intended to do.

6.1.1 Previously Fixed Issues

The main issue that the game had was that the audio was glitching with the front door and pulling plug audio. This was because the audio event was posted in the update function. The Boolean for checking that the mouse was clicked was true for more than one frame. This means that the audio was being posted for every frame that the mouse was clicked for resulting in an unbearable audio glitch. The fix to this problem was to the include a Boolean flip in the code so that the event is only posted once. The best example of this is the pulling plug audio code in lines 161 and 165 of PlayerController.cs**.** Creating a Boolean named *playOncePlug* and initialising it to true, an if statements uses that Boolean as a condition and instantly sets the Boolean to false so that the audio event is only posted once.

Another issue that the game had was the panning for the wall rub. As chapter 5.2.2 states, the code used for the panning was very complex and faced many different iterations until a solution was found. The issue was working out how to calculate angles using Vector3 mathematics. The first working iteration had an inconsistent range of values depending on where the player was within the game world.

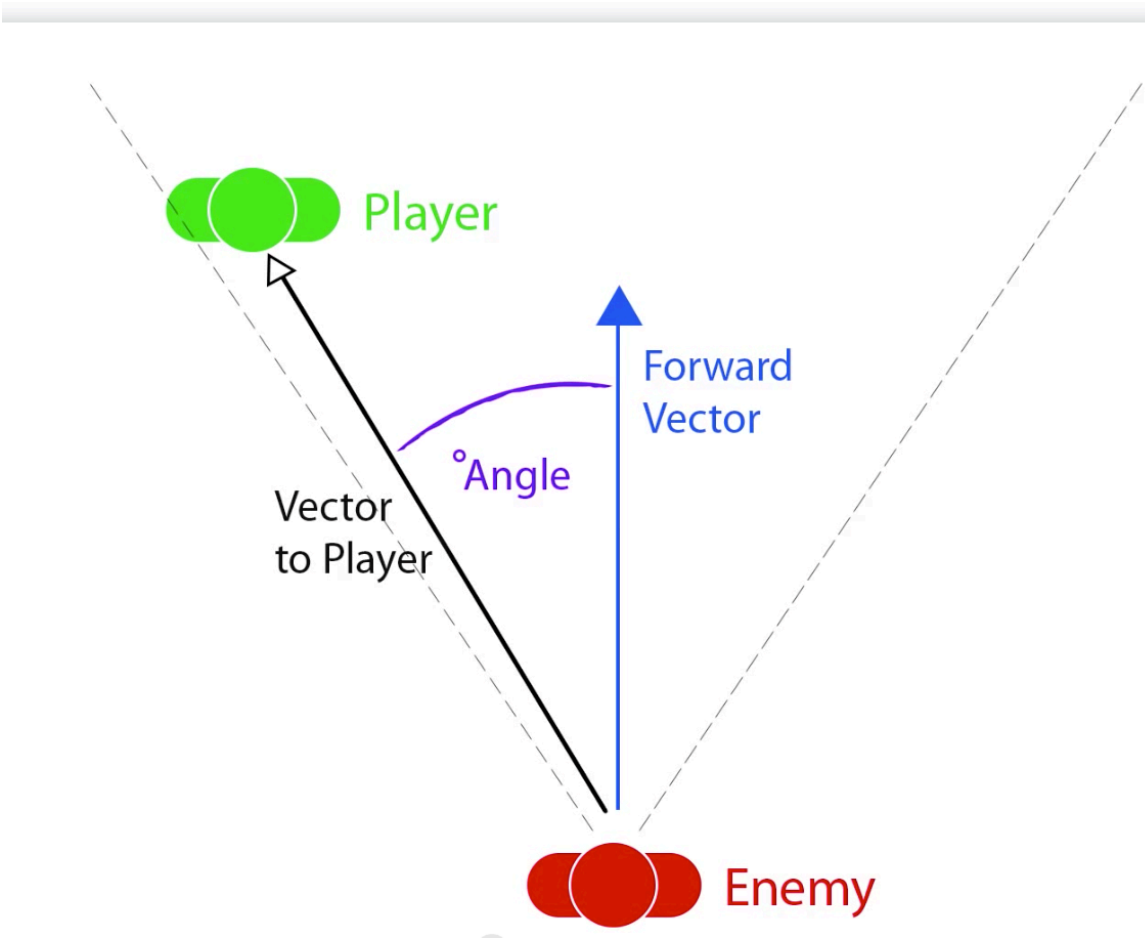

Fig 3. Image to visualise an angle using vectors

Fig 3 shows how the Unity game engine works out the angle between two Vector3's in its signed angle function (*Vector3.SignedAngle(Vector3 from angle, Vector3 to angle, Vector3 to rotate)*). The code was not working because the function was not being used correctly. Fig 3 described how to use the function by visually describing that the Vector3s in the equation were directions and not positions. By measuring the difference between two vectors (by simply taking one vector away from the other in the code) the direction vectors are created which, made the function output the correct value.

6.2 User Experience and Feedback

User experience is generally not well received in terms of enjoyable gameplay but, the game's audio was the standout success as it was described as believable and communicative by all users. The front garden area was particularly well received because it pushed the player to enter the front door, working exactly as it intended to do.

The gameplay, on the other hand, is not well received was because the game is frustrating to play because there is a no initially obvious communication of the player's orientation. One user mentioned that the sensitivity was too high and struggled to pinpoint sound sources because of this. The user was aware of the location of the game objects but was confused by the speed of orientation.

The only issue with the user testing though was that there wasn't really much time given to each user to sit down and focus properly on the game. Given time, the user potentially would have worked out how the game works. There was no testing done to see how steep the learning curve was for the game in its current state.

6.2.1 Previously added features

Through user testing, many gameplay features were added that were not previously planned. The main feature being the different floor surfaces in certain areas. The stairs were the largest issue because there was no way to tell whether the player was climbing a flight of stairs or not. Adding in creaks to footsteps on the stairs and different floor surfaces between the ground floor and first-floor hallway was the way to communicate that the player is in different elevated areas of the house.

The brushing up against wall audio and audio for the player walking into all types of doors was suggested through user testing. There was originally no information given to the player when they were up against a wall or hitting a door. Even with the map, the player still had little idea of position within the game world, the player would otherwise have to walk

around aimlessly until the interaction UI appeared. This was solved through audio and not easily through visuals which, is the main focus of the project.

6.2.2 Current Gameplay issues

Orientation is still the largest issue in the game. Despite there being a map and wall rubbing audio, there is no way to tell which direction the player is facing in relation to the 3D world environment. A suggestion was to implement a ping system that played a high-pitched ping sound at the most northern part of the game. The player could rotate the camera to the ping and then see which way they are facing by checking the map.

Currently, there is no menus or end game condition. The focus moved towards creating believable 3D audio and audio systems so, it is not detrimental to the project. The game would have been much more enjoyable to play if these were added because it becomes more like a game rather than a testing application.

Chapter 7 Conclusion:

7.1 Criteria Met

In chapter 4.1 the key features are listed as; Believable audio to create a game world, a task system for gameplay, minimal UI to fill in for the sense of touch. The key features are all met in the project where the first and third features are completed to a high standard.

The task system was lacking and could have done with some improvements such as adding the final task, the music box, as the end game condition. The obstruction and occlusion values that Wwise uses that are set in the code for the Bath still need some refining. In an audio game where there are no 3D visuals, the elevation is difficult to gauge because the player walks underneath the taps running audio to go back into the living room. Many users were confused because they thought that the bath was in front of them rather than above.

Other than the task system, the project succeeded in creating a believable and interesting solution to communicating locations of objects in 3D audio which, in turn, built upon skills and knowledge of 3D audio design and manipulation. These skills will be used in future projects within the games industry.

7.2 Critical Reflection

There were still many processes that need to be implemented to make an enjoyable gameplay experience which, the key features in chapter 4.1 set out. The best idea to improve the gameplay would be to take inspiration from text-based adventure games and applying the ideas to a 3D world environment.

If the project was to be created again, the ideal gameplay would be a player within a 3D world environment with no visuals but UI and running commentary of what the player is doing within the 3D world, very similar to text-based adventure games such as Colossal Cave Adventure (William Crowther, 1977). The 3D audio would direct the player to certain areas in the game while the running commentary would confirm to the player what was

happening in the game as the player had to complete audio-based puzzles. The gameplay was largely the main downside to project.

However, the main aim of the project was not to create an enjoyable gameplay experience but to effectively communicate the location, orientation and setting of *The Music Box*. As the evaluation of the user testing showed in chapter 6.2.2 orientation and location were still relatively difficult to understand by the user group. Orientation was the main issue because there was no obvious way of telling the player which way they were facing in the game. This may have been due to the sensitivity of the player look rotation as one user mentioned in chapter 6.2. The description of the setting of the game was a success because the users all believed that the audio implemented was believable in the sense of everyday listening. They knew exactly what they were hearing and the general whereabouts of the audio sources.

Chapter 8 Appendices:

1 - Dissertation Design Document

Dissertation design/to-do list

The Game

Setting: Set in a house where the player has to traverse blindly and complete a set of tasks to reach the attic and turn on a music box. Each noise/sound for each task will turn off when the respective task is complete.

Software: Unity and Wwise. Wwise spatial audio.

Front garden:
The front garden is where the player will spawn. There will be traffic instantly behind the player to force the player forward. Pavement in front of door and grass to the sides. Locked gate on left side.

Living room:
The living room will have a clock and TV in to indicate it's a living area. Player is able to turn TV off to hear the other sounds clearer. No need for other furniture because lack of visuals.

Back garden:
Bird noises, wind on tree leaves, and grass footsteps to indicate outside. Concrete slabs first to indicate the backdoor. Generator in corner that will hold the key to the second floor.

Second floor: Two rooms; bedroom and bathroom. Bedroom will have carpet and bathroom will have tiled floor (and reverb to compensate). Bath with dripping tap in bathroom. Bedroom will be empty, just the carpet and reverb to characterise it.

Attic: Simply has a music box playing in a box. Go to box and take it box. Use of occlusion for the box.

Tasks:

1. Open front door (into living room)
    - Animation for door opening.
    - Unlock door sounds and door swing noise.
2. Find key for back door in clock (clock turns off)
    - Opening clock face sound.
3. Open back door (into back garden)
    - Unlock back door sound and swing.
    - Animation for back door opening.
4. Find key for second floor (engine noise)
    - Engine noise from back garden.
    - Turn off engine sound.
5. Open second floor (into bedroom with bathroom)
    - Animation for door opening.
    - Unlock door sounds door swing noise.
6. Find key for attic in bath (water dripping)
    - Pull plug noise.
7. Open attic (into attic)
    - Open attic door noise.
    - And Animation.
8. Follow music box noise to open music box
    - Occlusion through wall for music.

Audio needed (Diegetic):

1. Footsteps – (Pavement, grass, carpet, floorboards, tiles)
2. Opening door
3. Clock ticking
4. Water dripping into water
5. Generator engine noise

6. Music box music
7. Car/traffic noises
8. Bird noises
9. Tree leaves in wind noises
10. TV noises

Non-Diegetic:

1. Interaction sounds
2. Task complete musical motif

Spotting “Sheet”:

Folder 07 K26T

STE-000.wav = Clock ticking - DONE

STE-001.wav = Trainer on carpet (footsteps)

STE-002.wav = Trainer on bathroom scales (tiled footsteps)

STE-003.wav = Walking with trainers on floorboards (Experimental) - DONE

STE-004.wav = Emulating walking on carpet - DONE

STE-005.wav = Emulating walking on tiles - DONE

STE-006.wav = back garden ambience without shield – half done

STE-007.wav = Back garden ambience with shield - DONE

STE-010.wav = traffic ambience with shield - DONE

STE-011.wav = Car engine 1.6 with shield mid gain (generator noise) – Not using

STE-012.wav = Car engine 1.6 with shield high gain (generator noise) - DONE

SET-013.wav = Open door handle noise (unsure, too much reverb) - DONE

STE-014.wav = Taps running water (for bathroom) - DONE

STE-015.wav = Plughole - DONE

STE-017.wav = open bedroom door – Need to come back to

STE-018.wav = Emulating grass by walking on plastic bags – DONE

STE-019.wav = Emulating walking on water for bathroom using trainer and kitchen sink basin – DONE

R05_0002.wav = Squeaky sofa to emulate creaky stairs. - DONE

R05_0003.wav = Rubbing clothes against walls. - DONE

R05_0004.wav = Walking into indoor door. - DONE

R05_0005.wav = Walking into front door. - DONE

R05_0006.wav = Playing with cardboard packet to emulate opening clock. - DONE

R05_0007.wav = Hitting key against glass pot, pick up key sounds. - DONE

R05_0008.wav = Turning car engine on/off. - DONE

Extra notes:

Grass footsteps, I layered on the grass emulation and the carpet emulation on top. Slightly reduced volume of grass emulation. Grass emulation was done by walking my trainer on plastic bags.

2 - User Testing Forms and Responses

The Music Box User Testing V1

3 responses

To what extent were the sounds in the game world believable?

3 responses

All sounds were believable except the door and bath tub plug

It was easy to understand what everything was supposed to be.

Yes

To what extent did the sounds explain exactly what was happening within the game world?

3 responses

Fairly well, effective use of diegesis

It was easy to understand where you were in the game world. was confusing with the bath I thought i done something wrong .

The obvious sounds were very believable, some of the synthesised sounds were a bit confusing

Were there any sounds that you expected to hear but didn't? List them and explain.

3 responses

Footsteps on different textures, TV sound

TV, clock from further away.

not nessessarily sounds, but attentuation could be used

Is there any additional functionality that the sounds should have? List them and explain.

3 responses

Bathtub filter in the living room, filter on clock as amplitude is constant

Audio to explain when the player has completed at task correctly.

see above, possibly some variation in the footsteps

Was there a believable sense of distance between the sound sources and the player? List them and explain.

3 responses

Yes, the distance from the traffic outside and clock were accurate

The bath was too loud.

yes, the traffic to start works very well

Any other comments?

3 responses

None

Have audio cues for when doing a task in the wrong order as well as one when completing a task correctly. Maybe have it so the player has to do it in the right order or they get reset?

eee

Music Box User Testing V2

3 responses

Are all the sounds believable? List specifics.

3 responses

Yes. Clock, footsteps and cars believable.

Yes, All.

Yes. Door sound that confirms is good.

Do the sounds give you a sense of direction? List specifics.

3 responses

No. Clock not specific enough.

Could find clock but, could only tell roughly where you are. Can tell whether you're at the back or the front of the house. Don't know where the doors are or where you are on the wall.

Yes the attenuation on the traffic is very good. The clock is difficult to pinpoint.

What did you find easy to work out?

3 responses

The door explained the flow of the game well because it opened another area.

Front area, inside with the carpet and floorboards.

Spatialisation outside the house.

What did you find difficult?

3 responses

Finding the clock. Not knowing the position of the player in terms of whether the player is next to a wall.

Getting lost upstairs because there is no stair sounds. Bathroom is really loud when downstairs.

Finding the clock.

What information, aural or visual, do you still need to be able to progress into the game? List specifics.

3 responses

On-screen instructions for the task. i.e. "Go into the living room". This gives the player an objective.

Hint system so you have a brief flash of where the Player is. Checklist for the items to hit.

Have a negative feedback noise for when the player doesn't click on the next correct task.

Any other comments?

3 responses

Increase the activation size for objects.

Introduction/read me or guide would be vital to improve and understand the gameplay.

None.

3 - User Testing Feedback and planning

User Testing Form Feedback response:

All user testing was done with alongside the user who would play the game until finished in its current state. No leading was done unless the user was obviously stuck for a few minutes or more.

First Iteration, The Music Box V1: 17th April 2019

Brief description of version: The game had all but some appropriate sounds, there was no blind fold on the player, severe lack of functionality for completing tasks.

- All of the users agreed that most of the sounds were believable. The two sounds that stood out as incorrect were the front door opening and pulling the plug out of the bath.

- User three got stuck on going to the back door when clicking the clock, some guidance was given.

- All users were told when they had completed the game because there is no obvious finished state currently.

- Two users specifically stated that the traffic sounds were accurate and believable.

To do:

- Add TV Sounds.
- Add different footstep sounds depending on material/textures.
- Fix audio glitching on front door and pulling plug.
- Add task completion sound or jingles.

- Add pitch randomisation in Wwise for all footsteps.
- Sort out bath taps sounds. Need to work out how to add occlusion so the player can hear that the bath is upstairs.
- Add sounds for stairs.
- Add sounds for when the player is brushing up against a wall or door (communication of when there is a door).

23rd April 2019

Brief Description of Version: Most sounds had been implemented, blindfold is now on for the Player. Almost complete functionality for completing tasks.

- Again, all users agreed that the sounds were believable. One user stated that the sound to progress the game (The non-diegetic sound after the front door) worked well to confirm that the player is progressing.

- The front area was well received because it pushed the player to open the front door. When the game loses linearity is where the issues come.

- All users found difficulty pinpointing the clocks position.

- A couple of users wanted visual information about the objectives within the game.

- A user wanted a sound to explain when the user had not done something correctly.

Fixed from last version:

- Added different footstep sounds for all areas but the stairs.
- Fixed the audio glitches on the front door and pulling plug.
- Added a task completion sound for the front door.
- Tweaked occlusion and obstruction for the generator and bath taps.

To do:

- Add TV sounds.
- Add incorrect task non-diegetic sound.
- Add footsteps audio for going up the stairs.
- Add sounds for when the player is brushing up against the wall and hitting a door.
- Add an objectives UI.
- Add to existing sounds for; picking up keys, opening clock, turning on/off generator.
- Potentially add a map for the game.
- End game condition.

Week beginning 29th April – Informal user testing

A few lecturers tested the game and offered feedback. None of the users reached the garden. One of the users didn't manage to even get past the front door.

A ping system was offered as a suggestion to help with orientation still being an issue. The users suggested that there be a single point on the map that can be pinged so the player can face towards the point so they know their rotation in relation to the game world.

Chapter 9 References:

Shinkle, Eugénie (2008) *Video games, emotions and the six sense.* Accessed from: https://journals.sagepub.com/doi/pdf/10.1177/0163443708096810

Hunt: Showdown (6 Nov 2017). *Hunt: Showdown | Developer Diary | The World Speaks.* Accessed from: https://www.youtube.com/watch?v=oaGlXhNXubs

Gaver, William (1993) *What in the World Do We Hear?: An Ecological Approach to Auditory Event Perception.* Accessed from: http://cspeech.ucd.ie/~fred/research/auditoryeventstructure/docs/gaverWhat.pdf

Gaver, William (1988) *Everday Listening and Auditory Icons.* Accessed from: https://www.researchgate.net/publication/34849868_Everyday_listening_and_auditory_icons

Allain, Alex (n.d.) *If statements in C++.* Accessed from: https://www.cprogramming.com/tutorial/lesson2.html

Switch vs if else (n.d.) Accessed from: https://www.geeksforgeeks.org/switch-vs-else/

Dight W, Batteau (1967) *The Role of the Pinna in Human Localization*. Accessed from: https://pdfs.semanticscholar.org/a4ce/98ea97ade43618bab07cd017ef8f0ac34f5c.pdf

Unity (2019) *Vector3.SignedAngle.* Accessed from: https://docs.unity3d.com/ScriptReference/Vector3.SignedAngle.html

Mahesa, Raka (2018) *Unreal and Unity comparison.* Accessed from: https://hub.packtpub.com/unity-and-unreal-comparison/

Mongeau, Anna (2016) *Audio Middleware Comparison – Wwise/FMOD/Fabric/Unity 5.* Accessed from: https://annesoaudio.com/2016/06/12/audio-middleware-comparion-wwisefmodfabric/

Waterfall vs. Agile: The Good, The Bad and The Misunderstood (n.d.) Accessed from: https://www.glowtouch.com/waterfall-vs-agile-good-bad-misunderstood/

[60FPS] DOOM (1993) LONGPLAY (October 13, 2018) Added by Minase [Online]. Available at https://www.youtube.com/watch?v=iFnOLFd_ByQ [Accessed 7th May 2019]

Duke Nukem 3D Episode 1 Playthrough 100% Secrets (June 12, 2011) Added by ToxicBarrel [Online]. Available at https://www.youtube.com/watch?v=4rSGPiNqbg0 [Accessed 7th May 2019]

Longplay: TES III: Morrowind Part 1 (June 20, 2016) Added by PSeXu [Online]. Available at https://www.youtube.com/watch?v=nQcQlFxM9oA [Accessed 7th May 2019]

Griffo (2015) [Forum] Image accessed from: https://answers.unity.com/questions/914139/vector3angle-2.html

Beauchemin, Stephane (2019) *Game Audio Programming 2.* Florida: Taylor & Francis Group https://ebookcentral.proquest.com/lib/portsmouth-ebooks/reader.action?docID=5504845

Collins, Karen (2008) *Game Sound.* Massachusetts: MIT Press https://ebookcentral.proquest.com/lib/portsmouth-ebooks/reader.action?docID=3338949

Dušan V. Nikolov, Marko J. Mišiü, Member, IEEE, and Milo V. Tomaševiü (2015) *GPU-based Implementation of Reverb Effect*. Accessed from: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7377631