Can the use of more musical features lead to more immersive rhythm games experiences?

Jonathan Byrne
BYR19702034

19702034@students.lincoln.ac.uk

School of Computer Science

College of Science

University of Lincoln

Submitted in partial fulfilment of the requirements for the
Degree of MComp Games Computing

Supervisor: Dr. Olivier Szymanezyk

May 2024

Acknowledgements

Firstly, I want to thank both James Brown and Olivier Szymanezyk for their supervision and advice regarding this project. I also wish to thank my friend James Gibson for his advice concerning the creation of the game, together with the general feedback he has provided. I also wish to thank all those who volunteered to playtest the game enabling me to refine it. I also wish to thank my parents and brother for the support and advice they have provided throughout my university experience, together with all those I have met throughout the university for helping to make my time at Lincoln one of my most treasured experiences.

Abstract

In this project, a rhythm game has been created which implements new musical features such as key and volume alongside the existing application of tempo in rhythm games as it is believed that the use of key and volume could lead to more immersive experiences than with the use of just tempo. This is evidenced by literature that confirms a link to both key and emotion, with emotion being a component of immersion. Volume is linked to intensity, with intense experiences being linked to more exciting games and therefore should make the game more immersive as a consequence. The development of both a control and experimental game is discussed in detail in the design and methodology and implementation sections with the key findings discussed in detail in the results section. In summary, it can be determined that the experimental game did facilitate a higher level of immersion when compared to the control game, with a mean immersion score of 132.5 compared to 121.9, this being attributed to the colour change feature derived from using key.

However, these results should be treated with caution as there may be reasons why these differences appeared, some participants may previously have been more exposed to rhythm games. A new test should therefore be created to determine if this prior experience in rhythm games is a significant factor.

Keywords: Immersion, Rhythm Games, Tempo, Loudness, Volume, Key, Emotion, Colour, Difficulty, Flow

The full project can be found here: https://github.com/JonBYR/Year4Project

Final Word Count: 21296

Table of Contents

List of Figures

List of Tables

Chapter 1 - Introduction

This section, together with sections 1.1.2 and 1.1.3, have been taken from the project proposal stage (Byrne, 2023).

The aim of this thesis was to determine to what extent different musical features in rhythm games could be used to enhance player immersion as currently rhythm games only incorporating the use of tempo during live gameplay. Other song features, such as music key, have not been fully exploited. Hein (2014, 102) states "Most mainstream commercial music games center around rhythm, rather than pitch, timbre, or other aspects of music". Therefore, the use of these additional features could be incorporated into the genre potentially providing more immersive experiences.

The aim was to explore new mechanics by integrating musical features alongside tempo. This differed from existing games as it was planned that these musical features manipulate the game during play. The user should perceive that the additional music features are changing their experience. An investigation was carried out to measure if there was a positive impact on immersion when compared to existing rhythm games. The aim and objectives are stated in section 1.1.1 and 1.1.2 respectively.

Music in video games is itself integral to the experience, with Munday (2007, 3) defining video game music as "sounds and silences generated by the game software which …. contribute to creating the phenomenon of the gameworld". Therefore, music is a contributing factor to immersion in games. However, the rhythm game alters this definition. Aspects of the music, i.e. tempo, immerse the player by generating the game mechanics which can often be very simple, offering as Pichlmair and Kayali (2007,

1-3) describe “a single repetitive interactive task for the user” where “the story and setting as well as the displayed on-screen action are crucial for the gameplay experience.”, meaning the music still immerses the user in the virtual world.

Rhythm games use music in combination with visuals to generate an immersive experience. The PS1 game Vib-Ribbon (NanaOn-Sha, 1999) has, as Pichlmair and Kayali describe, very simple gameplay requiring one/two button(s) to be pressed at the correct moment in time with the music. Similarly, Crypt of the NecroDancer (Brace Yourself Games, 2015) is a dungeon crawler game where all actions such as movement and attacks are only executed if the player presses the button in time with the music. In both games the difficulty is affected purely by tempo. Vib-Ribbon’s protagonist moves quicker to a faster tempo song, giving the user less time to react to upcoming obstacles. Similarly, Crypt’s protagonist has a smaller window of opportunity to perform an action on beat with a faster song. Audiosurf (Dylan Fitterer, 2008) further extends the concept of rhythm games. Its levels are based on music self-selected by the user who has two objectives, move left and right to avoid grey blocks and to collect coloured blocks. These tasks alter in difficulty as the tempo of the song fluctuates. There is also a visual difference during gameplay, with the colours of the level adapting to the song, allowing each level to provide a unique experience as dynamic colour scheming is introduced.

There are, however, exceptions to the rhythm games listed above. Soundfall (Drastic Games, 2022) provides similar gameplay to Crypt, but its levels are procedurally generated using features of the music self-selected by the user, i.e. music genre, to determine the type of level and volume to determine the number of enemy encounters (Cooper, 2023). As these features are used in the internal workings of the software it is

likely they are not noticed by the player, whereas the use of tempo is something the player perceives as influencing the game as each level is played.

The dissertation is outlined below: firstly, existing literature was reviewed and the way it informed the aims and objectives of the project are discussed. A requirement analysis expands on these, outlining the requirements for the project. Then the design methodology of the project is discussed, describing how the project was managed, the risks associated, the methodology undertaken, the design and tools used and finally how the project was tested. Following this the implementation of the project at each phase is demonstrated, followed by a discussion of the results and a conclusion evaluating the answers to the research question.

1.1 - Aims & Objectives

1.1.1 - Aim

- Investigate key and volume of music in a rhythm game to enhance player immersion.

To investigate the aim a 2D game was created that utilised a Spotify API to extract features of a song, such as volume, tempo and key, to affect the game's parameters. Two different games were created, one only using tempo, as in existing games, and the other also incorporating key and volume. The immersion levels for both games were then measured using an immersion questionnaire. The objectives derived from this are shown below.

1.1.2 – Objectives

Objective Number	Objective
1	Decide on five different songs that are musically distinct from each other, aim to be achieved by 5th November 2023.
2	Formulate a game idea by examining five rhythm games and research and establish the perceived game loop for this project together with a design document of 500 words, aim to be achieved by 19th November 2023.
3	Analyse Spotify API information and perform MDA analysis of 500

	words to establish what mechanics can be created from this information and their perceived effect on the user, aim to be achieved by 3rd December 2023.
4	Identify assets that can be used in a dungeon crawler game as outlined in the design document, aim to be achieved by 10th December 2023.
5	Create code that utilises the Spotify API to retrieve audio information from the five different songs, aim to be achieved by 17th December 2023.
6	Link the Spotify API code to Unity so that a user can choose from a selection of five songs, aim to be achieved by 24th December 2023.
7	Using game loop and MDA analysis, create a game that utilises Spotify information, aim to be achieved by 3rd March 2024.
8	Create a second game that uses only tempo as a game mechanic, aim to be achieved by 24th March 2024.

9	Establish an immersion questionnaire for both games to be completed by players, aim to be achieved by 31st March 2024.
10	Test both games using observational data from twenty participants and analyse feedback to determine how immersive both experiences are by 28th April 2024.

Table 1: The Objectives for the project

1.2 - Project Implementation

The following suggestions for ways to use musical features to enhance immersion were implemented. Firstly, music key influences the colours of the game due to existing links to how key and colour coincide, achieved using the bloom effect in Unity via Castel's model, further discussed in the literature review. Volume/loudness was used to determine the types of enemies spawned in each level of the game. It was theorised that this would be better perceived by players than the existing implementation in Soundfall for increased immersion. Finally, a discussion on how a beat detection algorithm was created, this being used to capture whether players are on beat, is provided together with the way Spotify's API was used to retrieve song information and its interaction with Unity.

1.3 - Contributions

This paper makes the following contributions:

- Research indicates that numerous existing rhythm games use only tempo as a core mechanical feature, however literature indicates that key and volume are two further ways to heighten the immersive experience. This contribution is shown in sections 2.2-2.6.
- Two games were designed, developed and implemented based on literature linking key to colour, emotion and, therefore, immersion, with volume linked to intensity and immersion. These findings from literature are implemented in an experimental game, with a control game using only tempo as a variable for immersion. Implementation is shown in section 5, with both games shown in Figures 1 and 2.
- Both games were tested by 20 participants, 10 for the experimental and 10 for the control, with a new questionnaire being introduced for this study blending the IEQ, GEQ with its own questions. Testing is discussed in section 4.5.
- This data was used to compare both games, with the experimental game showing a higher mean immersion, tested to be significant, as discussed in section 6.2.
- The conclusion summarises that whilst there is greater immersion, there are still areas for improvement or investigation, the full conclusion provided in section 7.
- A significant finding is that the use of colour in rhythm games may not enhance immersion through emotional changes, but instead immersion and engagement are enhanced through visual interest, with justification provided in 6.3.3.

Figure 1: The control game for the song Reckoner

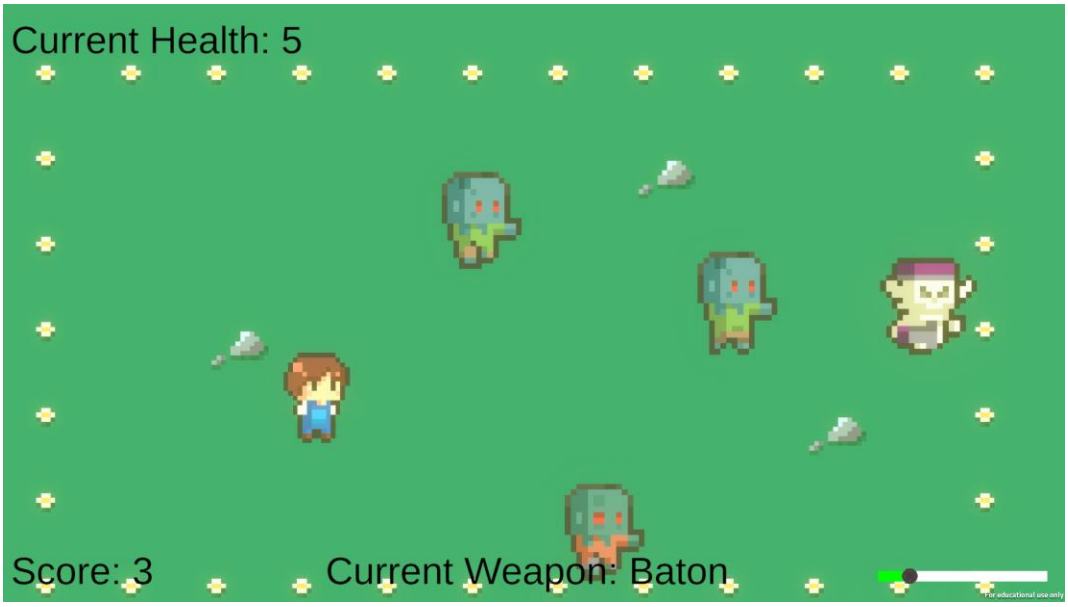


Figure 2: The experimental game for the song Reckoner. As can be seen, there is now a colour change from the bloom effect, showing a yellow hue. Stronger enemies are also present

Chapter 2 - Literature Review

2.1 - Player Immersion in Rhythm Games

Sections 2.1 to 2.4 have been adapted from the interim report (Byrne, 2024).

There have already been studies surrounding player immersion in rhythm games. Kagan (2020) analyses how Crypt of the NecroDancer immerses players, noting how all player actions are linked to the beat of a song. Kagan explains how, in later levels, enemy patterns become more complex and that, in response, the tempo of the music slows down to enable players to adapt to these changes. This draws from game flow, defined by Sweetser and Wyeth (2005, 7) as "The rate at which players experience new challenges and details can be paced to maintain appropriate levels of challenge and tension throughout the game". It should be noted that this concept applies to more linear games, where new challenges are introduced as the player progresses. However, most rhythm games, are also linear. As Tsujino (2019, 2) notes "Players have to have quick decisions and actions for the faster song, it is thus considered that the faster song is more difficult", meaning tempo controls difficulty and thus flow. Furthermore, Kagan (2020, 4) explains how the substages of each level in Crypt have a linear increase in tempo to match the linearity of flow occurring once a player becomes familiar with each level. Zheng (2022) further explored this concept by developing a rhythm fighting game that used different tempos to measure how the general flow was affected. Zheng found that for certain characteristics of game flow, such as perceived control, a slower tempo was better received by players, whilst other concepts, such as balance of challenge and skill, were not affected by tempo. However, this was related to a style of video game different to Crypt. It could be that certain game genres better suit changing tempos to maintain a sense of flow and that therefore a suitable game genre should be incorporated to analyse the research question.

2.2 - Musical feature exploration

Most rhythm games, and the relevant research, focus mainly on the use of tempo as a gameplay mechanic. However, other aspects of music which could further increase a player's immersion such as the musical key of a song or volume, have not yet been explored.

People associate types of music with emotions. Studies (Lindborg and Friberg, 2015, 1, Barbiere et al., 2007, 1) showed that "happy music was associated with yellow, music expressing anger with large red colour patches, and sad music with smaller patches towards dark blue" (Lindborg and Friberg, 2015, 1) and "Brighter colours such as yellow, red, green, and blue were usually assigned to the happy songs and grey was usually assigned to the sad songs" (Barbiere et al., 2007, 1). It is clear certain colours can be assigned to certain kinds of music to invoke emotions in players and heighten their immersion. Joosten et al. (2010, 5) used different colours in a video game to examine emotional response and found "that red elicited a highly aroused, negative emotional response, and yellow elicited a positive emotional response". Zammitto (2005, 11) also notes how colours help for "setting a mood of the world the player would immerse in". Therefore, colours help to set an emotional response in the player and are utilised in games to allow immersion in a certain feeling. According to Zammitto (2005, 12) that immersive feeling could be "depression, sadness, solitude, loneliness, distance".

Ciuha et al. (2010, 3) attempts to visualise the colour of particular chords, aka keys, via a colour wheel, shown in Figure 3. Whilst they admit that no studies have been undertaken using this wheel, a similar model, developed by Louis Bertrand Castel, shown in Figure 4, was used by Fonteles et al. (2013) for their particle system. This attempted to use music visualisation as it "establishes different colours for each one of twelve musical existent notes" (Fonteles et al., 2013, 3). This was received positively by "general audience, music students, and music experts" (Fonteles et al.,

2013, 7), suggesting that Castel's model is a better model to use due to experimental backing.

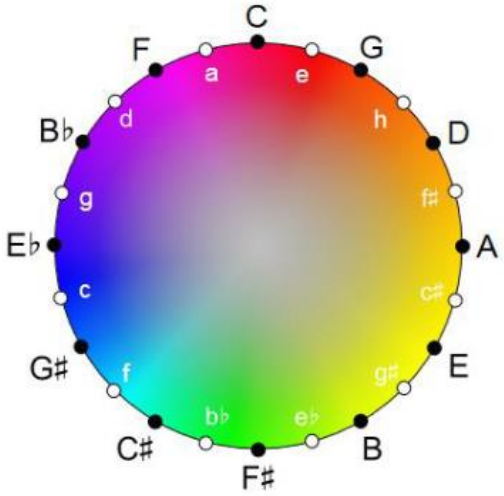


Figure 3: Colour wheel of keys (Ciuha et al., 2010, 2)

Figure 4: Castel's colour model (Fonteles et al., 2013, 3)

As with music key, there are studies which investigate the relationship between volume and emotion. Sloboda and Juslin (2001, 26) state that "loud fast music shares features with events of high energy, and so suggests a high energy emotion such as excitement". Similarly, Blesser (2007, 3) describes how "an aural space loud music is often experienced as "exciting" because loudness represents intense activity". Furthermore, the volume of a song also changes people's behaviour when participating in activities such as exercise. Edworthy and Waring (2006, 9) found that "fast, loud music is particularly effective in increasing speed of running" when compared to slower, quieter music, with Welch and Fremaux (2017, 5-6) explaining this behaviour as being "linked to adrenaline, in the colloquial sense of an internal sense of arousal", meaning the body is more "alert and active". Relating these

findings to video games, Chanel et al. (2008, 4) report how players experienced an increase in heart rate as difficulty increased. In their later study this is further explained as an increase in a player's arousal (Chanel et al., 2011, 10). These reactions to difficulty are similar to a person's reaction to loud music. The implication, therefore, is that the difficulty of the game should be adapted accordingly. If a song has louder music, difficulty should increase to induce the intensity and arousal the user expects. Plans and Morelli (2012, 4-6) also explained that they procedurally generated music that would "celebrate flow by rewarding the user with obviously joyous-sounding music", meaning that if a player is within the flow state described by Chanel et al., the music can be louder to invoke a feeling of excitement as Sloboda and Juslin state. Plans and Morelli admit however, that the userbase this music was tested on was inconclusive, due to a small sample size and limited responses for the questionnaire. Adapting the difficulty needs to be carefully considered. Chanel et al. (2008, 1-2) state that, while increasing difficulty can lead to a positive player experience when "skills of an individual meets the challenge of a task", if difficulty is increased too quickly it can cause anxiety, therefore, this must be taken into consideration when making both games.

2.3 - Colours on Immersion

It is envisioned that the visual component will change depending on the type of song used. Aesthetic features such as visuals (colours) and sound coexist when influencing immersion as they both contribute to "enhance the emotions elicited from the play, support the theme of the game" (Tanskanen, 2018, 33). It is established that music key affects colour and that colour affects emotions. The experimental game plans to use music key to change the overall colour of the game using Castel's model as it is believed this will provide a more immersive player experience by influencing mood. This will be achieved by colour grading which has already successfully been utilised in games to provide differing effects. Kauranen (2023, 30-31) describes how colour grading used grey colours "to give certain

games a more realistic feeling" or use a "green cast…enhancing this ominous feeling". Therefore, this technique performs the immersive qualities outlined by Zammitto, to set an emotional response in the player for greater immersion. This response is provided by the song key via Castel's model for the experimental game.

2.4 - Measurement of Immersion

Features of the game will be generated based on song choice. Studies have hypothesized that immersion increases should self-selected music be used. Fierro (2012, 32) notes that "Most probably, if people could choose music after their own taste in the game, the level of immersion of this people could be still higher". Cassidy and MacDonald (2009, 18) explored this concept by allowing users to self-select music in a driving game and comparing performance and player experience against music Cassidy and MacDonald had selected. It was found that both metrics were best for self-selected music, further reinforcing Fierro's hypothesis. Since self-selected music has been proven to create a positive effect during gameplay, it is possible its use may create bias in players when testing their immersion. Thus, the use of music will be restricted to a pre-set selection of songs.

Objectives 9 and 10 state that testing data will need to be gathered from users, presented in two ways, measuring immersion via a questionnaire and establishing observational data to determine player engagement. However, both objectives are currently too vague to truly gauge how to properly measure immersion. Jennett et al. (2008, 9) defined different ways to effectively determine immersion. For instance, they measured the number of times a user's eyes changed what they were fixated on, with an immersive task having less fixations as the player is more engrossed in the task, with the inverse true for non-immersive tasks. They linked these results with an immersion questionnaire to further explain these fixations. The questionnaire was derived from two prior studies. Firstly, Agarwal and Karahanna (2000, 10) who propose five dimensions for user engagement and secondly Brown

and Cairns' (2004, 1), which interviewed players to determine what aspects of games lead to more immersive experiences. These two papers will be studied further to aid the construction of the immersion questionnaire in objective 9. It should, however, be noted that Agarwal and Karahanna's study relates to general software experiences and not just games. However, some concepts outlined will still be relevant. It is also possible that certain aspects mentioned in Brown and Cairns' (2004, 3) study may not be applicable to the rhythm game created. One aspect is that, to achieve total immersion, a player needs to feel empathy, to "feel attached to a main character or team". However, this aspect is more applicable in role playing games "where the gamers assumed a character" and where detailed stories and characters emerge, something that is not deemed necessary for a rhythm game. Other aspects such as atmosphere, where "graphics, plot and sounds combine" are more important for immersive rhythm game experiences. Grimshaw et al. (2008, 2-3) used questionnaire data combined with other objective observation data, such as galvanic skin response, to evaluate the effect of sound on immersion. Whilst all four studies show effective ways to quantify immersion, it should be noted that they are all older studies. As games have evolved, the ways in which they are immersive may also have changed, making certain aspects of the questionnaires used in these studies outdated. Therefore, whilst the use a subjective questionnaire can help to explain results from observational data, more research should be done to establish if these studies are still relevant for today's gameplay.

2.5 - Musical Features and Game Design

An important objective was to analyse five existing rhythm games to help formulate the design document. These games are mentioned briefly in the introduction: Crypt of the NecroDancer, Vib-Ribbon, Soundfall and Audiosurf. A final game, Rhythm Sprout (SURT, 2023) was also analysed. Each game is summarised below.

2.5.1 - Crypt of the NecroDancer

Crypt is a roguelike dungeon crawler, where the player moves their character across a procedurally generated dungeon to the beat of the song, using the arrow keys. Every player turn is achieved via the rhythm, meaning when pressing the arrow keys, a player either moves, attacks an enemy or uses an item. Enemies also move to the rhythm, with each enemy having its own unique movement pattern. The player's goal is to exit the dungeon, usually via the stairs.

2.5.2 - Vib-Ribbon

Vib-Ribbon involves a player automatically moving across a level, avoiding oncoming obstacles utilising different button presses. The objective of the game is to survive until the song ends. Players can make a set number of mistakes before getting a game over. However, if a player does well, they gain more life and, therefore, more chances. Levels are unique depending on the song with the speed in which a player has time to react being determined by the song's tempo.

2.5.3 - Audiosurf

Audiosurf is a puzzle racer where the player aims to collect blocks of the same colour as they automatically move along a highway. The speed at which the player

moves is dependent on the song chosen with the game's colour also adapting according to the chosen song.

2.5.4 - Soundfall

Soundfall is also a rhythm-based dungeon crawler. Whilst players are able to move independent–of the music, game actions such as attacking and dashing must be completed on beat to execute correctly. Levels are procedurally generated based on music features including using volume to increase/decrease enemy encounter rate.

2.5.5 - Rhythm Sprout

Rhythm Sprout is a rhythm action game where the protagonist moves from A to B by pressing the left and right arrows in time with the music, attacking an enemy in the same way when prompted and using the spacebar to dodge attacks. Failing to dodge results in a loss of life. The speed at which these inputs must be completed is dictated by the tempo of the song.

2.5.6 - Games Analysis

It can be determined that all five games possess a central mechanic, this being game difficulty. The faster the tempo in each level, the less time players have to react. Therefore, it can be determined that tempo is of singular importance in rhythm games. Most rhythm games tend to have a player "on-rails", where the player moves automatically, reacting to oncoming obstacles. Crypt, however, allows players to move or perform an action only when on beat with the song. Rhythm games have differing goals, these being survival either until the song ends or until a certain exit point is reached. Whilst the introduction stated that rhythm games only make use of

tempo for game mechanics, it is now acknowledged that games, such as Audiosurf and Soundfall, also utilise other features. Audiosurf aims to change the colour of the level based on the given song, whilst Soundfall uses volume to vary enemy encounters when generating a level. However, it should be noted that these rhythm games are exceptions to the rule. Whilst Audiosurf does use colour, it is not known if this is the result of a feature other than tempo. Furthermore, the use of volume/genre are aspects that are internal workings of Soundfall, possibly not noticeable to the player. As there is currently little research supporting the hypothesis that the use of more musical features leads to greater immersion, the game and questionnaire should be designed to determine if this is the case.

2.6 - Literature Conclusion

Studying the available literature helped to inform the design document for objective 2. Whilst existing research has linked tempo with immersion in rhythm games, there is a gap in research regarding the other musical features previously mentioned. The research question therefore aims to determine if incorporating these other features will help to provide more immersive experiences. There is currently little research into how music key affects player immersion. However, from the literature analysed, it can be inferred that key can affect player mood as mention is made of key being linked to specific colours, with these in turn being linked to specific emotions. Thus, key can be used in conjunction with colour grading to immerse the player emotionally. The study by Zhang et al. (2017, 1) states that part of immersion is emotional immersion, where “the user feels emotionally aroused and absorbed”. Therefore, it can be assumed that affecting a player’s emotional state by combining key, colour theory and colour grading in the experimental game will influence a player’s overall immersion. This will be examined when testing the experimental game to see if this hypothesis aligns with the research aim. Again, there is little research on the effects of volume and immersion. However, it has been stated that

volume changes a person's state of mind by making environments more intense and exciting. Therefore, it can be inferred that immersion will increase by establishing a more intense scenario in the experimental game determined by the intensity (volume) of the music. Difficulty and intensity are linked and Ermi and Mäyrä, (2005, 4) notes that difficulty causes greater immersion as there is "increasing demand on working memory" of the player, therefore facilitating an intense and difficult environment will deliver greater immersion. If both the musical features, together with the way in which they are incorporated into the game, have a positive immersive effect, these techniques can be applied to the rhythm genre in the future.

Chapter 3 - Requirements Analysis

Please note that parts of this section refer to comments made in the proposal document (Byrne, 2023) and interim (Byrne, 2024).

There are a set number of functional and non-functional requirements that this project must achieve, outlined in the aims, objectives and literature. The first non-functional requirement was the creation of the game design document (GDD) outlined in objective 2. This was informed by the literature review conducted in both the proposal and interim stages and includes details on how tempo, music key and volume are utilised.

It is important to create game design documents as they provide a guide to refer back to throughout the whole development process (Almeida and da Silva, 2013, 2). Whilst there are no common standards detailing how these are formatted (Dormans, 2012, 59) they are a good way of outlining the initial ideas for a game's development, although it is likely there will be changes which deviate from the initial design document over time (Dormans, 2012, 60).

The static nature of the design document means it can be unproductive to reflect changes in the design (Almeida and da Silva, 2013, 2). Therefore, even though this design document was created to outline the core goals and mechanics for the game, attributes in the final product have deviated from the initial document.

A game loop which helps "identify the core mechanics and to think about how other activities can support it or allow for variety" (Guardiola, 2016, 6) was also planned. This helps visualise how the player interacts with the game mechanics outlined in the design document. objective 2 describes how five rhythm games were analysed to help facilitate the design document. This was done in the form of an MDA (Mechanics, Dynamics and Aesthetics) analysis, which as Hunicke et al. (2004, 2-5) states, helps "consider both the designer and player perspectives" and helps "reason explicitly about particular design goals, and to anticipate how

changes will impact … the resulting designs/implementations". Therefore, the requirement for objective 2 is to:

- Create a design document and game loop informed by an MDA analysis and existing research

A functional requirement for this project will be to enable the user to pick one of five different songs generating a different type of level depending on its musical features. The literature review outlines how tempo has already been used in rhythm games as a metric of difficulty. However, it is believed that the inclusion of new musical features in the game such as music key and volume will lead to a more immersive experience. This will be achieved by relating music key to colour and volume to intensity. Therefore, the game must be able to adapt its parameters depending on the song chosen, using Spotify's API to extract song information.

API stands for Application Programming Interface with Lindman et al. (2018, 1) describing the benefits thus: "provide a useful interface for service provision, customer access, and third-party development." The API was used to retrieve the required information tempo, music key and volume for each song using a Spotify query called Get Track's Audio Analysis. The song information is returned as a JSON file, which is stored in the game. Information from this file must be parsed into the game to adapt the game parameters, depending on the song chosen. Whilst any programming language can be used to access APIs, including object orientated languages such as Java and C# (Nguyen et al., 2017, 1), Python was selected due to its use as a primary backend in development (Goel, 2021, 5). A suitable game engine also needed to be selected to parse in the JSON files. The two major games engines used in the industry are Unity and Unreal. Although other game engines exist, for those who are "looking to get into game development for the medium or long run Unity or Unreal Engine are the recommended choices" (Andrade, 2015, 6). Unity has "a simpler user interface…its installation does not require high-performance hardware" (Christopoulou and Xinogalos, 2017, 14) whilst Unreal is much more

resource intensive and while “its graphics are remarkable” (Christopoulou and Xinogalos, 2017, 14), the proposal planned to create a 2D game with pixel graphics, meaning a high-performance game engine was not required. Therefore, a second requirement was to:

- Create a game that can parse through JSON objects and manipulate the game’s parameters from the JSON file

Another functional requirement was to create a game where the player can easily infer that different musical features are influencing the level. As already discussed, a player can infer that the game’s difficulty is adjusted by the tempo of the game, as this mechanic has a known effect on difficulty. However, other features proposed, such as music key, provide a visual effect. Therefore, the game must be able to adequately communicate these visual changes to the player when changing the game’s parameters. One study has already discussed developing a particle system using Castel’s colour model and, as Unity has a particle system of its own (Unity, undated), this was a potential avenue to explore for visual effects. Another requirement therefore, is to:

- Create a game that provides feedback to the player on how the musical features affect the game experience

The last functional requirement(s) relate to objective 7, the creation of an experimental game. The literature review states how tempo can be used as a game mechanic and outlines that music key and volume may enhance immersion. These requirements were unknown until the GDD was created, outlining the core mechanics and ideas for the game that were not known at the proposal stage. The requirements for the experimental game were therefore:

- Creation of beat detection algorithm
- Use of this algorithm to allow player movement/attacks

- Use of this algorithm to allow enemy movement
- Spawning of different enemies using time signature
- Weapon durability, weapon functionality and weapon spawning
- Use of json file to increase/decrease enemy pool

The beat detection algorithm detects when players/enemies are “on beat” and are able to move. The weapon durability, functionality and spawning all refer to mechanics described in the design document. The final requirement relates to the discussion surrounding how musical features affect gameplay, with louder songs causing stronger enemies to spawn. objective 8 describes the creation of a control game. Whilst similar to the experimental game, only a song’s tempo is used, as in existing rhythm games. These two games will then be tested and their immersive qualities evaluated to ascertain if the research question has proven correct.

A final non-functional requirement relates to how this project will be tested. The literature review discussed how more research was required to inform the questionnaire. Further research discovered that there are two questionnaires that utilise player immersion, these being the Immersive Experience Questionnaire (IEQ) (Jennett et al., 2009, 19-20) and the Game Experience Questionnaire (GEQ) (Brockmyer et al., 2009, 4). As Nordin et al. (2014, 6) notes, the IEQ has been used in many studies, i.e. a study comparing the immersion of a game with/without music (Sanders and Cairns, 2010, 4) and provides a greater range of accuracy by using positive and negative statements (Nordin et al., 2014, 6). The GEQ measures player engagement, incorporating immersion and flow as components. Brown and Cairns (2004, 2) mention engagement as important in facilitating immersion. The IEQ should therefore be adapted to include further questions from the GEQ relating to engagement, to better understand how immersive experiences are created in the game. However, Denisova et al. (2016, 4) notes that some questions in the GEQ are not applicable to all genres, i.e. for a rhythm game, the GEQ question “I feel scared”. Also, some questions in the GEQ are too vague i.e. “I feel different”. This question

could be adapted to relate to the experimental game, for instance, did a user feel their emotions changed during the game. Therefore, a final requirement is to:

- Create a questionnaire that adapts both the IEQ and GEQ to relate to the experimental game to measure player immersion.

Chapter 4 - Design & Methodology

4.1 - Project Plan

A Gantt chart in Figure 5, taken from the project proposal, outlining when each objective should be achieved and the estimated time each would take was created.

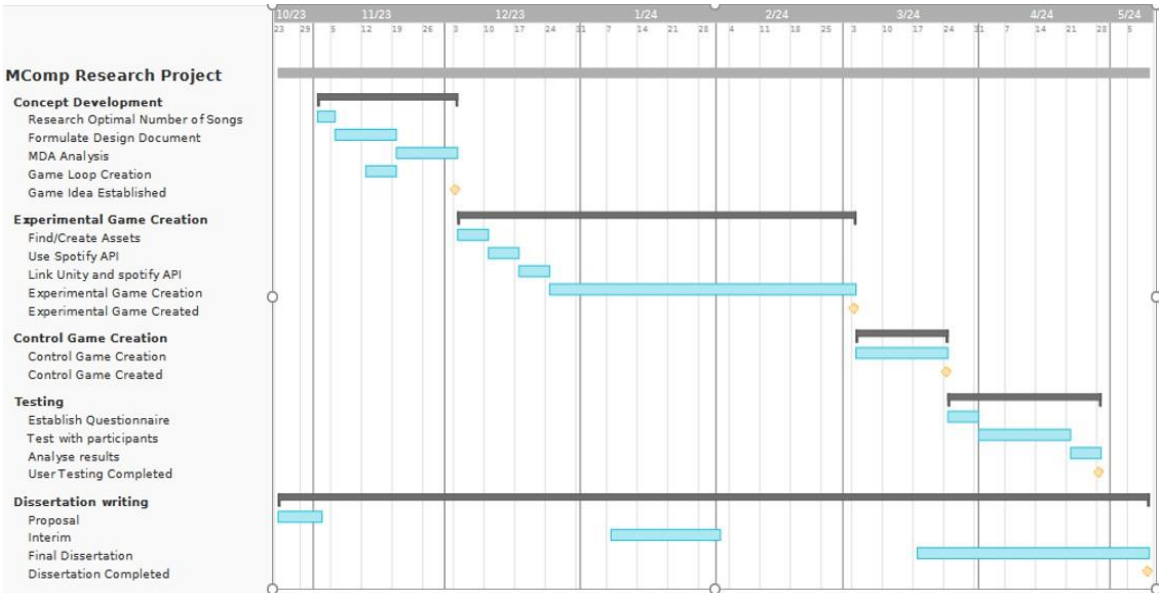


Figure 5: The finalised Gantt Chart, showing all stages of development (Byrne, 2023)

Certain objectives in the Gantt chart have changed. The Concept Development phase originally stated that an optimal number of songs would be researched. However, this conflicted with objectives stating five songs were to be used. Therefore, musically distinct songs were researched, while the find/create assets now relate to identifying a suitable asset pack for the game (Byrne, 2024). It should be noted that Gantt charts normally visualise the waterfall software methodology (Karlesky and Vander Voord, 2008, 1). However, a pure waterfall model is not advised in the industry due to games needing multiple iterations (Kanode and Haddad, 2009, 1) with agile methodologies such as SCRUM being implemented

instead. Agile practices have been adopted 10% more often in games compared to waterfall with advantages including designers not being required to wait for a particular feature implementation (Kristiadi et al., 2019, 3). Agile is specifically favoured due to its flexibility in handling changeable game design processes and allowing developers more influence on the design (McKenzie et al., 2021, 5).

SCRUM can be split into three phases: the pregame, game development and postgame phases (Kristiadi et al., 2019, 3), represented in the Gantt chart as the Concept Development (pregame), the Experimental and Control Game Creation (game development) and finally the Testing (postgame) phases. Furthermore, the use of SCRUM is an iterative model, where clients (i.e. playtesters) review each stage of the game build so that changes can be implemented in the next phases of development (Kortmann and Harteveld, 2009, 5). SCRUM consists of multiple sprints implementing aspects of the backlog, these being the features required for the game, with any changes needed added to this backlog for the next sprint (Schwaber, 1997, 14-15). Kristiadi et al. (2019, 5) recommend following the three phases outlined above, in conjunction with the game design document as it "reveals the goals and also all things that will be needed in both planning and development later" meaning that the backlog for SCRUM is first defined in the Concept Development document. SCRUM was therefore determined to be the optimal methodology for this project being common in the industry and beneficial to the game design process.

SCRUM is attributed to be the main workflow for the creation of game mechanics and features (McKenzie et al., 2021, 5), whilst other agile practices such as Kanban are better employed for asset creation, such as art or levels which are "predictable and steady" (McKenzie et al., 2021, 5) compared to game mechanics. However, as a pre-existing asset pack is used, the need for Kanban to develop art assets is cut out completely, therefore only SCRUM is required for the overall project as only game

mechanics needed development. The results of each iteration of SCRUM for the Experimental Game Creation is shown in Chapter 5.

4.2 - Risk Analysis

Risk	Likelihood	Impact	Mitigation
Client secret is exposed	High	Client secret together with client ID are needed to access the Spotify API. However, if client secret is exposed then there is the potential that outside users could attempt to access the Spotify API pretending to be the user and so manipulate the account's access to the API. Since it is planned to use GitHub, which is a public	Keep the GitHub repo private so that no other GitHub users can access the repo or use other, less public, file storages like OneDrive.

		software, there is a potential the client secret variable could be viewed by others. This is more of a concern for the researcher who will need to keep themselves protected from potential access violations	
Performance issues from music features	High	Spotify's API returns a JSON file that is broken into different segments. This means there is the potential for tempo shifts, key shifts etc. between segments. An update check for any changes between segments will	As HDRP is more memory intensive URP should be used instead. If changing colour via script for post-processing is too intensive for each segment, then the average key for the song, provided by Spotify, should be used instead,

		be required together with a response if a change is detected. Since it is planned that key will affect colour, this will involve post processing in Unity using either URP or HDRP which could be memory intensive to manipulate in script.	which should still invoke an emotional response for immersion measurement.

Potential disparities in skill	High	As mentioned in the literature review, flow and immersion tend to be linked, usually when the skill level of the player matches the difficulty of the game. In rhythm games this has traditionally been achieved by gradually introducing songs of higher tempo thus creating increased difficulty. As this project aims to use different songs, there is the potential that the differing tempos may alienate users	Ensure that there are different songs to match the skill level of different kinds of players, from less to more experienced. This could mean starting with a slower tempo song and gradually transitioning to faster tempo songs.

		unfamiliar with rhythm games.	
Potential sickness in participants	High	Since the game is using colour grading with the idea that the colours will change as the key of the song shifts, there is the potential that sickness, such as epilepsy, may be induced in certain participants. (Parra et al. 2007, 1-2)	Ensure that the researcher is present in the test space and able to stop the experiment should a participant start to feel unwell. Ensure this is highlighted as a potential problem to any interested participants. The colour shift feature could be

			optional for those who are uncomfortable with this setting as its removal will not affect gameplay.
Poor performance data/immersion questionnaire	Medium	There is the potential that the questionnaire does not contain sufficient information or pose enough questions relating to user experience and so participants do not provide sufficient data for proper evaluation. Similarly, the	Look at existing literature surrounding how immersion is measured and use this literature to create a similar immersion questionnaire

		data used to evaluate player performance may not prove to be relevant to the study of immersion. Therefore, there is the potential that testing has been completed but no usable data is returned.	
Disconnect between music and game	Low	Spotify's API returns a JSON file containing averages of a song's tempo, key etc. for both the song as well as for segments of the song. Using the average for the song could potentially create problems should there be a particular segment of the	Rather than using the average for the song as a whole, use the segment information instead and ensure that the game will update correctly as soon as there are any differences between segments.

		song that is faster than the average tempo of the song. This therefore disrupts the sense of flow a player will experience and, consequently, negatively impact immersion due to the link between the two. The experimental game also uses key and volume and therefore these features are also variable throughout the song.	

Table 2: Risk Analysis Table (Byrne, 2023)

4.3 - Design

4.3.1 - Concept Development Phase

It was decided to model the experimental game on an existing rhythm game analysed in the literature review. Crypt of the NecroDancer was chosen due to its easily understandable game loop and the ease of finding assets suited to its genre, with asset justification discussed in section 4.3.2. It was also discovered that other genres of games, i.e. the fighting game developed by Zheng, worked better for slower tempos. As the songs from objective 1 had similar tempos to those in Crypt, it was felt that this genre was the most accommodating for the research question. The goal has been amended so that the player survives until the song ends, similar to existing rhythm games allowing the player time to notice if the experimental musical features are affecting immersion. The creation of the game itself is discussed in Chapter 5.

It was important to develop both an MDA analysis, game loop and GDD for the Concept Development Phase, with the justifications for these outlined in Chapter 3 together with an explanation on why the GDD did not change after the interim. The GDD details ways in which key and volume are used, derived from the literature review. Key visually changes the colours of the game due to colour influencing emotional responses, whilst volume causes harder enemies to spawn when louder songs are chosen facilitating a more intense environment, with rationale for these features discussed in Chapter 2. The results from this phase are shown in Appendices A-C.

Fullerton (2014, 197) states the importance of prototyping to test the feasibility of the game idea and make improvements, similar to how SCRUM can incorporate changes to the game build. Prototyping is specifically concerned with the fundamental mechanics, for instance moving to the song's tempo. Prototypes are

either physical or digital, with physical prototyping typically created from “paper, cardboard and household objects” focusing purely on gameplay rather than technological aspects like code and, as it is cheap, allowing for multiple iterations (Fullerton, 2014, 197-198). However physical prototypes have downsides. They are only suited to certain styles of games, namely turn based systems (Neil, 2012, 4-5). Also, the core loop of the game relies on beat detection for mechanics such as player movement and, therefore, requires that the developer is able to detect beats. Generally, it is difficult for people to detect rhythm and therefore when they are “on beat” (Povel, 1984, 5) so it is felt that this concept is better realised in code. Fullerton (2014, 236) recommends creating digital prototypes for concepts not easily modelled physically, still allowing for testing core mechanics, alongside modelling aesthetics such as sound effects or art (Fullerton, 2014, 238). Therefore, the implementation of the game in Chapter 5 shows the iterative process of digital prototyping, testing code to generate beat detection and aesthetic ways to demonstrate the music key, using a SCRUM development process to test each game concept.

4.3.2 - Game Assets

An asset pack was sourced from the Unity Asset Store, as outlined in the interim report, to create a dungeon crawler rhythm game, established from the Concept Development stage. The pack is included in the references section of the dissertation and was chosen as, whilst designed for a game called Vampire Survivors (Poncle, 2021) it was felt the style of Vampire Survivors was similar to Crypt, both being 2D, pixel art games with roguelike elements. It was important to ensure that the assets chosen could be used to create a similar feeling game to Crypt. Both developed games incorporate similar gameplay to Crypt, i.e. players only moving or attacking whilst on beat and as the controls emulate Crypt, players who have already experienced good flow in Crypt, i.e. the mastery of the controls in the game (Cowley et al., 2008, 14) will readily take to both games. As “games can be a self-

referential media" (Cowley et al., 2008, 15), meaning that both games are referencing Crypt's prior game mechanics, the player themselves will be "familiar with the genre and its conventions" (Cowley et al., 2008, 15) allowing for the easing in of gameplay.

As a roguelike game was being created, it was important to incorporate certain concepts that feature heavily in the genre, including: loss of all progress on death, turn based gameplay in a grid-based structure, an inventory system with only a single character to control and allowing the player to discover how new items work (Izgi, 2018, 13), to further familiarise players about the game's genre. Some of these are subverted by Crypt, for instance the turn-based system usually means that once a player moves, the enemies move, however in Crypt movement is solely dictated by a player moving in time with the beat, with enemies moving to a set number of beats (with separate attack patterns).

4.3.3 - Front and Back End in Games Design

Back-end relates to the use of tools such as Visual Studio 2022, with Koleva et al. (2015, 4) describing the back-end as "creating the underlying resources", such resources including the development of the beat detection algorithm described in Chapter 5. The Spotify API used to influence the game parameters is itself connected to a backend system when called (Biehl, 2016, 199). Front-end development normally relates to web development, with many libraries and toolsets utilised for this purpose (Dinh and Wang, 2020, 2) however Koleva et al. (2015, 4) describes front-end as "coding in the interactive elements" relating to UI development (Xing et al., 2019, 2). Games UI includes utilising components such as Canvas in Unity. This contains information such as text describing the game state or current weapon, similar to how information heavy games such as Massive Multiplayer Online (MMOs) have text panels that contain game instructions (Llanos and Jørgensen, 2011, 2). However, other information can be conveyed as juicy

feedback or greater polish, based on game actions, discussed in greater depth in Chapter 5.5, with the use of such feedback causing a higher sense of immersion (Llanos and Jørgensen, 2011, 2). Juicy feedback are general features in the games industry and are incorporated here to keep a player engaged in both games and prevent boredom in flow.

4.4 - Toolsets and Machine Environments

To create the artefact the following toolsets have been used: Unity version 2022.3.1, Python and C# with Visual Studio 2022, Spotify's API, Chocolatey and ffmpeg and finally GitHub Desktop for version control. Unity and Spotify's API have already been discussed in Chapter 3, with C# being utilised as the default language used by Unity. A further discussion is shown below on other software used in the project.

4.4.1 - Chocolatey and ffmpeg

As the Spotify API only returns song information and not a file to play the song, the researcher's own iTunes library was utilised to provide the audio files required for the game. However, an issue was discovered using this method. Song files in iTunes exist in an .m4a format, but Unity does not accept this audio format, instead using formats such as .wav or .mp3 (Unity, undated). Therefore, software to convert the .m4a file into a .wav file was required. Wav was preferred to mp3 as "acoustic research should ideally be made in a lossless format" (Fuchs and Maxwell, 2016, 1) as provided by .wav. Whilst the research is not acoustic based, audio is a requirement for the game and therefore it was felt that the best quality audio should be implemented. The software used was ffmpeg (2000), as audio conversion can occur in a single line in command line and has been used in existing softwares such as VLC media player and browsers such as Google Chrome for audio playback (Hock and Lingxia, 2014). Installation was achieved using Chocolatey, (2011)

advertised as a package manager used by Windows that also runs on command line with its use shown in Figure 6. However, in this case the command line is used to install packages like ffmpeg. Figure 7 showcases the process of using ffmpeg.

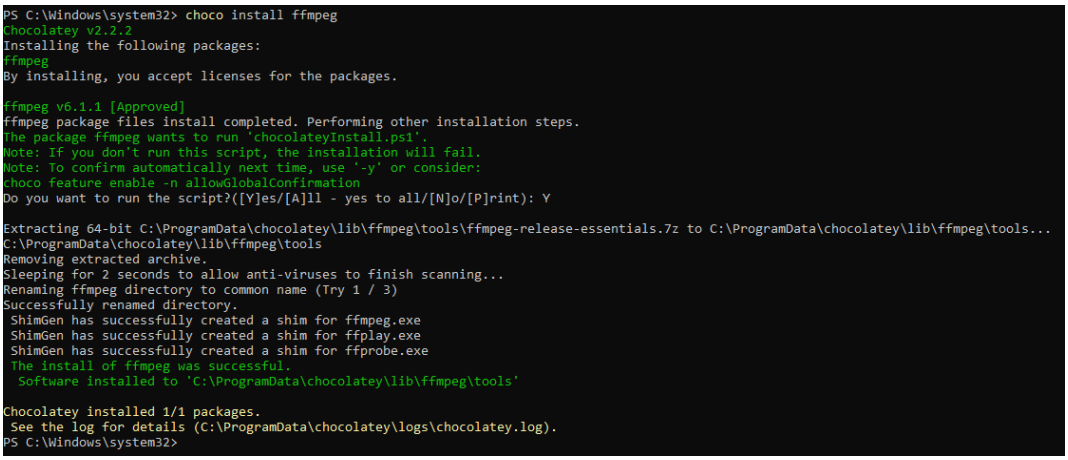


Figure 6: The use of Chocolatey to download ffmpeg

Figure 7: The use of ffmpeg to convert a .mp4 to a .wav

4.4.2 - Version Control

Version control is required in the software engineering process as it helps to manage and organise the files and revisions that occur throughout the software development process. It prevents potential risks when saving copies of code such as erasing the wrong file (Zolkifli et al., 2018, 2) which could occur when uploading to cloud storage i.e. OneDrive. Additionally, version control software has been implemented in project-based learning to good effect, offering greater professional skills, and cooperation with others (Milentijevic et al., 2008, 8, Glassy, 2006, 6). The most popular version control system is Git (Hultstrand and Olofsson, 2015, 8) which exists either as a command line or with a UI in GitHub Desktop. Both versions work by uploading and updating code to an individual repository. Hultstrand and

Olofsson (2015, 29-31) found that novice users of Git preferred to have the GUI, such as GitHub Desktop. It is also utilised by more experienced users, albeit less frequently and in combination with the command line. There are several issues with Command Line Interfaces (CLI) as described by Sampath et al. (2021, 5-6). CLI is difficult to navigate with tasks such as scrolling being considered painful by developers. CLI manuals are also not easy to read as users cannot skip to headings or skip sections as with a front-end webpage. Therefore, due to the less accessible nature of a CLI, the GUI Git offers was adopted as it is considered more user friendly.

4.5 - Testing

In Chapter 3 it was determined that a suitable questionnaire was required that adequately encapsulated if an experience was immersive for players. This questionnaire was adapted from the IEQ, which has been incorporated in many studies analysing immersion, with certain questions adapted from the GEQ. Furthermore, as this study is evaluating whether the use of new musical features affects player immersion, new questions have been formulated specifically to evaluate these features, i.e. the questions “Did you ever notice any differences during gameplay?” and “Did the game feel intense?” as the results from these questions determine if the player detected that volume was contributing to gameplay. Furthermore, the question “Did you feel emotionally different when playing the game?” establishes whether the use of different colours relating to key was recognised by the player, eliciting a positive response if true. Finally, the questionnaire includes a space for detailed participant comments allowing them to expand on their experience. This “allows respondents whose feelings and thoughts are aroused by the questions to express themselves in their own way” (Phellas et al., 2011, 13) and should provide more detail why players were/were not immersed. The questionnaire is provided in Appendix D.

The majority of this data is quantitive, using likert scales between 1-5 in order to establish the user's immersion. Nordin et al. (2014, 6) explains that the IEQ is computed thus "The overall score is composed of the summary of the results from the positive questions, as well as the inverted results of the negative ones". Taking the Thompson et al. (2012, 3) study as an example, the mean is calculated by adding the results from all questions (inverting negative questions) to get the total immersion and dividing this by the total number of participants.

The hypothesis that "players should be more immersed in the experimental game than the control game" will be verified by first detecting the difference between the mean values for both games. To determine if there is general significance between the two groups, a t-test will be used using the difference of the experimental and control means (Smucker et al., 2007, 1-2). Significance therefore determines that the results are not down to chance or error and thus having "inherent noise" (Smucker et al., 2007, 1) but instead indicates that there is a link. Note that the t-test assumes data exists in a normal distribution and is independent from one another. A normal distribution can be observed by mapping a histogram of the data. Again, taking Thompson et al. (2012, 4) as an example, total immersion score is mapped on x and y maps the frequency of a particular ordinal value, shown in Figure 8.

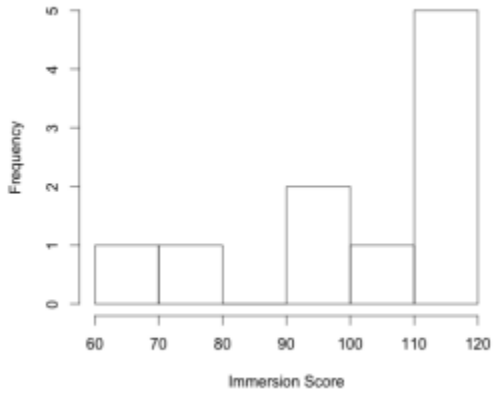


Figure 8: The histogram created by Thompson et al. (2012, 4)

Should a normal distribution not be observed, as in Thompson et al.'s (2012) study, a different significance test, such as a Mann Whitney U test which assumes no normality is required. Therefore, the significance test will not be officially adopted until it can be adequately assumed that both sets of data are operating under normal distributions.

In the objectives it was established that observational data should be used as well as the IEQ to help determine immersion, being achieved in the literature review via examples such as galvanic skin response**.** However, the use of such equipment is normally reserved for studies relating to nervous activity, for instance, lie detection (Sharma et al., 2016, 1). Instead, metrics related to the game, such as the number of times a beat is successfully hit, the number of times a player dies in the game and the player's final score will be measured. Although these metrics relate more to how "easy" the player finds the game, it should be noted that these can be used to explain immersion, a higher score meaning greater flow due to mastering the game mechanics (Sweetser and Wyeth, 2005, 9). Furthermore Jennett et al. (2008, 13) mentions that a game that is "too difficult…leads to too much anxiety" and is less immersive, further collaborated by Qin et al. (2010, 1) who discuss that immersion

is lessened if game difficulty changes are too slow or too fast or if difficulty decreases then increases. Therefore, if a player is dying or missing beats too often the game's difficulty has progressed too fast. Likewise, if the player does not die as often then the game difficulty is not increasing fast enough leading the player to the boredom state of flow. A mean score for each song in both the experimental and control games will be calculated from all participant data.

4.5.1 - Procedure

20 participants were recruited for this study, the same number as Thompson et al's (2012, 3) immersion study. 10 played the experimental game whilst 10 played the control game. Participants were sat in one of the university laboratories and assigned a PC with either the control or experimental game loaded, with no knowledge as to which version was played. A brief explanation of how the game is played was provided. So that all songs are evaluated for their tempo, key and volume, the player was asked to play each song for at least 2 minutes, with the escape key being used to exit the song. After each song was played, the participant was then asked to complete the questionnaire in Appendix D. Participants were recruited from the university by advertising the study in various game lectures and workshops. A participant information sheet, reference number and consent form detailing all information required for participation was provided. The reference number for each participant was recorded on each questionnaire allowing complete anonymity. No sensitive information was stored, however contact details for the researcher were provided in case any participant decided to withdraw from the study prior to May 9th. Observation metrics were also recorded for each participant during each level. After all tests were completed, the mean score for both the immersion questionnaires together with observational data was calculated. The set up for this experiment is shown in Figure 9.

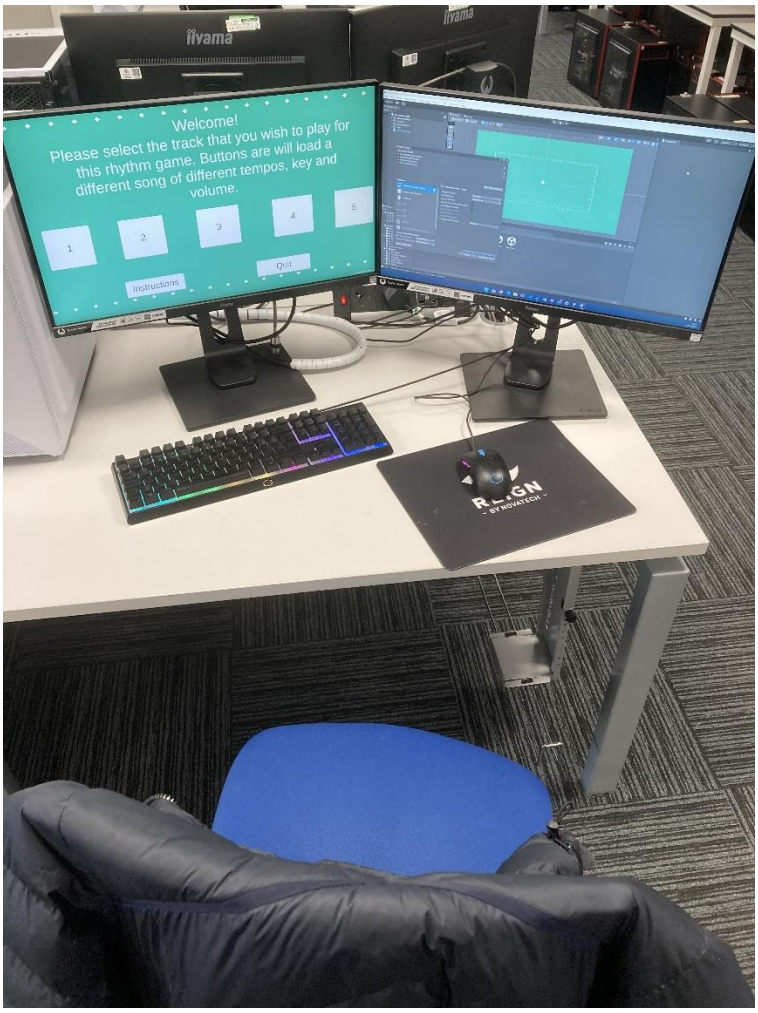


Figure 9: The workspace for participant testing, showing the control game on the left monitor screen for players to interact with

Chapter 5 - Implementation

This chapter reviews how objectives 5 and 6 were implemented, discussing the utilisation of Spotify's API via Python and how it was linked to Unity, how the GDD, game loop and MDA Analysis, formed from objectives 2 and 3, were utilised to create the experimental game for objective 7. Also included is the implementation of the beat detection algorithm for tempo, present both in the control game for objective 8 and the experimental game. Details of how music key and volume were implemented are also outlined. Any further changes to the game are discussed in the playtesting and polishing section. These changes relate to player feedback utilised to refine the game before official testing began. Table 3 summaries the subsections. Flow is mentioned throughout this Chapter, being a core game design concept that, when executed successfully, helps facilitate immersion (Sweetser and Wyeth, 2005, 10). Poor game flow leads to less immersive experiences. Therefore, the flow between the control and experimental games should be similar as a controlled variable. All experimental game requirements follow an ABC sprint (Alpha, Beta, Completion) for SCRUM, described by Kristiadi et al. (2019, 4), the alpha stage implementing the basic functionality from the backlog, the beta stage implementing the feature complete game from the alpha with "final assets" and the completion stage removing bugs and adding polish. The alpha and beta stage are implemented simultaneously in sections 5.1-5.3.4 as these sections implement the basic functionality alongside the final game features. The completion stage is incorporated in 5.3.5, with the backlog for this created from playtesting. Any full code not disclosed in the appendices can be found in the GitHub repository in the abstract.

Subsection	Summary
5.1 Implementation of Spotify API	A discussion of how a Python program was created to obtain the song information required for this project.
5.2 Spotify and Unity	A discussion of how the Spotify API was incorporated with Unity for this project.
5.3.1 Beat Detection Algorithm	A discussion of the beat detection algorithm, which is used to determine when a player is “on beat” with the tempo of the song and the different variants to attempt this.
5.3.2 Key and Colour Implementation	A discussion of how colour and key linked together and the various ways to visualise this in Unity, i.e. via post processing, a particle system, and finally, background colour.
5.3.3 Volume and enemies	A discussion of how enemies are spawned using volume in the game together with how enemies are attacked.
5.3.4 JSON files and Unity	As Spotify’s API returns a JSON file this subsection describes how the JSON

	files were parsed through to the game and how this information was utilised.
5.3.5 Polishing and Playtesting	This section relates to any polishing features used in the game together with the ways playtesting feedback was incorporated. This section discusses concepts such as “juiciness” as well as game flow.

Table 3: Subsections of Chapter 5

5.1 - Implementation of Spotify API

When developing both games, an important requirement was to successfully develop Python code to access the Spotify API, returning relevant song information for the game. When developing the API, it is important to first authenticate that the user accessing the API has authorisation. Unauthorised API accesses can lead to cyber-attacks, such as DoS (Denial of Service) attacks, when the API receives a sudden influx of traffic (Madden, 2020, 3). It also prevents spoofing by ensuring that the person performing the operation is the actual user themselves (Madden, 2020, 9). API authorisation works as follows. The Python code requests access using a client ID and client secret, where client ID refers to the unique identifier of the app (Spotify, undated) and client secret refers to the key that authorises API calls (Spotify, undated). It is vital that this key is kept hidden. Lu (2014, 2) explains that, if an unauthorized user gets a hold of this key, they have free reign to perform API actions and it’s likely that DoS attacks will occur. However, as mentioned in the risk analysis, the Python project is currently set to private to prevent this. Once the client ID and secret have been sent to Spotify, Spotify returns an access token (Spotify, undated), allowing the Python

program to send requests (queries) to the API. The code which handles authorisation and returns an access token is shown in Figure 10, whilst the workflow for authentication is shown in Figure 11.

```
def get_token():
    auth_string = client_id + ":" + client_secret #authorization string needs to be this format
    auth_bytes = auth_string.encode("utf-8")
    auth_base64 = str(base64.b64encode(auth_bytes), "utf-8") #returns base_64 object as string
    url = "https://accounts.spotify.com/api/token" #url that auth_string will be sent to
    headers = {
        "Authorization": "Basic " + auth_base64,
        "Content-Type": "application/x-www-form-urlencoded"
    }
    data = {"grant_type": "client_credentials"}
    result = post(url, headers=headers, data=data) #returned as json
    json_result = json.loads(result.content) #convert json to dictionary
    token = json_result["access_token"] #get access token from json object
    return token
```

Figure 10: Python code to create an access token that will be used by the client credentials workflow

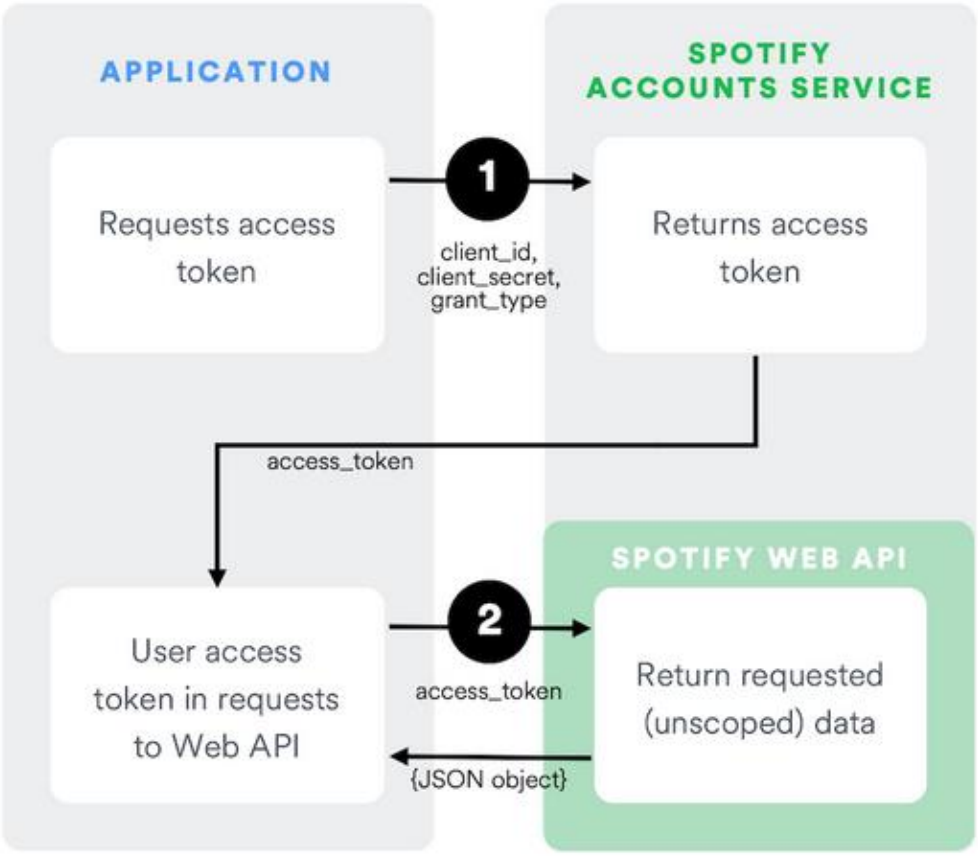


Figure 11: Spotify Client Credentials Workflow, used to generate the JSON files for the project (Spotify, undated)

Note that an access token for Spotify only lasts for an hour (Spotify, undated) and is defined as "a system where a physical object (token) is used to access some digital information stored outside the object" (Holmquist et al., 1999, 5). In this case the digital information is the access to song information. In order to generate the access token, the request to Spotify must be encoded in the form "application/x-www-form-urlencoded" and the data itself is sent to the url https://accounts.spotify.com/api/token. The request for the access token must be formed as follows: it must include the token url, a header object containing the type of authorisation, the encoded form and finally the grant type, relating to the client credentials workflow.

After the access token is generated, it is then used to access information from Spotify. This was tested using two API queries described in Figure 12.

```python
def get_auth_header(token):
    return {"Authorization": "Bearer " + token}
def artist_search(token, artist_name):
    url = "https://api.spotify.com/v1/search" #api used to search for artist (or tracks)
    headers = get_auth_header(token)
    query = f"?q={artist_name}&type=artist&limit=1" #use &type=artist,track to search for artist and a track, limit = 1 will only get most popular artist
    query_url = url + query #fstrings are string formating
    result = get(query_url, headers=headers)
    json_result = json.loads(result.content)["artists"]["items"]
    if len(json_result) == 0:
        return None
    return json_result[0]

def get_songs_by_artist(token, artist_id):
    url = f"https://api.spotify.com/v1/artists/{artist_id}/top-tracks?country=UK" #finds top tracks by this artist
    headers = get_auth_header(token)
    result = get(url, headers=headers)
    json_result = json.loads(result.content)["tracks"]
    return json_result

token = get_token()
artist_name = "ACDC"
result = artist_search(token, artist_name)
artist_id = result["id"]
songs = get_songs_by_artist(token, artist_id)
print(songs)
```

Figure 12: Code to find artist and get top songs by artist

The access token creates a new set of headers, which are utilised to authorise access to Spotify when sending the constructed query. The function artist_search tries to find the particular artist specified by the artist_name string, in this case AC/DC, and returns the artist ID. This is returned as the query obtaining an artist's top tracks requires the artist ID. When searching for an artist, a new url is constructed containing both the url for searching and a formatted string containing the artist to search for, with a limit of how many artists to return, in this case 1. This returns a JSON file that is converted to a dictionary using json.loads() and shortened to only contain the item's header, as this contains the information relating to the artist themselves, including the artist ID. When this JSON object is returned, the artist ID is stored individually by returning and storing the value from the 'id' dictionary key. This ID is used in a separate function that constructs a query to search for the artist's top tracks, resulting in the output in Figure 13.

Figure 13: The returned songs JSON file for the artists AC/DC

Having confirmed that the code works, it can now be adapted for the purposes of the project: to extract song information for tempo, key, volume to utilise in both the experimental and control games, shown in Figure 14. Most of the code is unchanged, however the query for searching is now adapted to search for a specific track. A further query uses the track ID to generate a JSON file containing the audio features described above (Spotify, undated).

```
def get_track_id(token, artist, song):
    url = "https://api.spotify.com/v1/search" #api used to search for artist (or tracks)
    headers = get_auth_header(token)
    query = f"?q={artist}, {song}&type=artist,track&limit=1" #use &type=artist,track to search for artist and a track, limit = 1 will only get most popular song
    query_url = url + query #fstrings are string formating to create a query in the spotify api, ? needed at start of query string
    result = get(query_url, headers=headers) #format the information is returned as
    json_result = json.loads(result.content)["tracks"]["items"] #converts this into a dictionary
    if len(json_result) == 0:
        return None
    return json_result[0] #returns the id of the track (first element of the dictionary file)
def get_track_information(token, id):
    url = f"https://api.spotify.com/v1/audio-analysis/{id}"
    headers = get_auth_header(token)
    result = get(url, headers=headers)
    json_result = json.loads(result.content)
    return json_result #returns all information from get_track_audio_analysis as a json file
token = get_token()
result_track = get_track_id(token,artist_name,artist_song)
song_id = result_track["id"] #dictionary is returned and id is stored from the id key field
track_info = get_track_information(token, song_id)
with open('musicdata.json', 'w') as exportFile: #writes all the track_information returned from spotify into a json file using the dump method
    json.dump(track_info, exportFile)
```

Figure 14: The code for get_track_id and get_track_information

Note that artist_name and artist_song are user inputs and that the search query looks for the specific song relating to the specific artist, as the song name alone is insufficient.

Also note that rather than printing the dictionary of the JSON file, a new JSON object is created using json.dump(). Although the file is called “musicdata.json”, the file name will be rewritten as song_name.json after creation when using the

files in the experimental game, preventing the same file being overwritten. Admittedly the file name could instead have been formatted by creating a string variable that compounded the strings “artist_song” and “.json”, with this passed to open() to store the information, although for 5 songs manually changing the file name had negligible efficiency penalties. The information generated for the song Reckoner by Radiohead is shown in Figure 15.

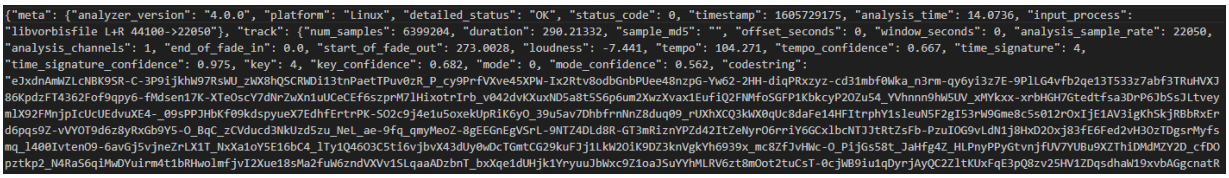

{"meta": {"analyzer_version": "4.0.0", "platform": "Linux", "detailed_status": "OK", "status_code": 0, "timestamp": 1605729175, "analysis_time": 14.0736, "input_process": "libvorbisfile L+R 44100->22050"}, "track": {"num_samples": 6399204, "duration": 290.21332, "sample_md5": "", "offset_seconds": 0, "window_seconds": 0, "analysis_sample_rate": 22050, "analysis_channels": 1, "end_of_fade_in": 0.0, "start_of_fade_out": 273.0028, "loudness": -7.441, "tempo": 104.271, "tempo_confidence": 0.667, "time_signature": 4, "time_signature_confidence": 0.975, "key": 4, "key_confidence": 0.682, "mode": 0, "mode_confidence": 0.562, "codestring": "eJxdnAmWZLcNBK9SR-C-3P9ijkhW97RsWU_zWX8hQSCRWDi13tnPaetTPuv0zR_P_cy9PrfVXve45XPW-Ix2Rtv8odbGnbPUee48nzpG-Yw62-2HH-diqPRxzyz-cd31mbf0Wka_n3rm-qy6yi3z7E-9PlLG4vfb2qe13T533z7abf3TRuHVXJ86KpdzFT4362Fof9qpy6-fMdsen17K-XTeOscY7dNrZwXn1uUCeCEf6szprM7lHixotrIrb_v042dvKXuxND5a8t5S6p6um2XwzXvax1EufiQ2FNMfoSGFP1KbkcyP2OZu54_YVhnnn9hW5UV_xMYkxx-xrbHGH7Gtedtfsa3DrP6JbSsJLtveymlX92FMnjpIcUcUEdvuXE4-_09sPPJHbKf09kdspyueX7EdhfErtrPK-SO2c9j4e1u5oxekUpRiK6yO_39u5av7DhbfrnNnZ8duq09_rUXhXCQ3kWX0qUc8daFe14HFItrphY1sleuN5F2gI53rW9Gme8c5s012rOxIjE1AV3igKhSkjRBbRxErd6pqs9Z-vVYOT9d6z8yRxGb9Y5-O_BqC_zCVducd3NkUzd5zu_NeL_ae-9fq_qmyMeoZ-8gEEGnEgVSrL-9NTZ4DLd8R-GT3mRiznYPZd42ItZeNyrO6rriY6GCxlbcNTJTtRtZsFb-PzuIOG9vLdN1j8HxD2Oxj83fE6Fed2vH3OzTDgsrMyfsmq_l400IvtenO9-6avGj5vjneZrLX1T_NxXa1oY5E16bC4_lTy1Q46O3C5ti6vjbvX43dUy0wDcTGmtCG29kuFJj1LkW2OiK9DZ3knVgkYh6939x_mc8ZfJvHWc-O_PijGs58t_JaHfg4Z_HLPnyPPyGtvnjfUV7YUBu9XZThiDMdMZY2D_cfDOpztkp2_N4RaS6qiMwDYuirm4t1bRHwolmfjvI2Xue18sMa2fuW6zndVXVv1SLqaaADzbnT_bxXqe1dUHjk1YryuuJbWxc9Z1oaJSuYYhMLRV6zt8mOot2tuCsT-0cjwB9iu1qDyrjAyQC2Z1tKUxFqE3pQ8zv25HV1ZDqsdhaW19xvbAGgcnatR

Figure 15: The track information for Reckoner by Radiohead, including loudness and tempo

5.2 - Spotify and Unity

Early in development, an attempt was made to link the Spotify API code developed to Unity, utilising the InputField UI component to input the song and artist the player wished to listen to. These variables are passed through to the Python code and uses the get track audio analysis query, returning a JSON file to be accessed by Unity which has built in tools for parsing JSON (Bueno, 2017, 31).

A package called Python Scripting (Unity, undated) was utilised for this functionality. Rather than using a .env file as in the Python implementation, the client ID and secret were instead placed in the Python file itself, although this does mean the client secret is shown more prominently. To load in the artist and song required to generate the track information, the artist and song names are passed as a JSON object, which is loaded as a Python dictionary and stored in the relevant variables. However, whilst the two InputField variables are able to be converted into a JSON object using the Unity function JsonUtility.ToJson, there were issues

accessing the Python file in Unity. Firstly, the requests library that handles API queries, needed installation in the Python code. When installing the requests module for use with Python only, the package ran with no issues. However, as the requirement is for this module to run in Unity, the requests package must be installed in a different location to allow Unity access. A requirements.txt file using the command line function "pip freeze > requirements.txt", must be generated containing the external libraries/packages that the Python project requires. Unity needs this .txt file for the Python Scripting package and it must be stored in the ProjectSettings folder. Although this method should allow the Python code to run successfully when playing the game, the decision was taken to abandon this method of generating the JSON files.

This was initially due to confusion as to why the Python file was not running in conjunction with Unity. When attempting the generation of the JSON file, an error occurred stating that the requests module could not be found. However, when using the Python file outside of Unity, the code ran successfully. It was not discovered why this initial process had failed until later. As the literature review had also found that the use of custom music could lead to bias in how a player may feel immersed, it was decided to limit the user to a selection of pre-selected songs, with the JSON files for these being generated using the pre-existing Python code running independently from Unity. The created JSON objects are transported to Unity and assigned to a button UI component, which parse the JSON file into Unity adjusting the parameters of the game.

5.3 - Experimental Game Development

5.3.1 - Beat Detection Algorithm

The key aspect for both games was to implement a beat detection algorithm to allow users to move within the tempo of a song, this serving as one of the most crucial mechanics in Crypt. Rhythm games work by ensuring “a player … translate visual and auditory cues into actions and perform them at appropriate time and in rhythm” (Lin et al., 2011, 2). To be in time with the rhythm means being within the song tempo, defined as the beats per minute (BPM) (Schacher, 2007, 1). To ensure that the player is in time with the tempo at every second, the BPM must be converted into beats per second. As the reaction time for auditory stimuli is around 284 milliseconds (Shelton and Kumar, 2010, 1) there needs to be some leeway allowed for the player in case they move too early/late. As most of the game mechanics incorporate the song’s features, it was decided to use a singleton, defined such that: only one instance of this class exists at any time, that the instance of the class is reusable and they are usually static and public, allowing all game objects to access the class (Stencel and Węgrzynowicz, 2008, 5). The initial creation of the singleton, defined as GameManager, is shown in Figure 16.

```
// Update is called once per frame
Unity Message | 0 references
void Update()
{
    timer += Time.deltaTime;
    if(Input.anyKeyDown)
    {
        if(timer < 0.1 || secPerBeat - margin <= timer) //making sure that player is within beat
        {
            Debug.Log("Within Beat");
            timer = 0;
        }
    }
    if(timer >= secPerBeat)
    {
        Debug.Log("Missed Beat");
        timer = 0;
    }
}
```

Figure 16: The first implementation of the GameManager singleton

A timer has been created, incremented by Time.deltaTime, indicating the interval in seconds from the last frame to the current frame (Unity, undated). The secPerBeat variable relates to the beats per second and the update loop checks to ensure that, if the player is slightly too late (less than 0.1 meaning that they just missed the beat), or hit the beat exactly/slightly before, when a key is pressed, then the player is "on beat". Otherwise, the player has missed the beat. The main issue when using Update, is that it is non-deterministic, meaning that the number of times Update is called per second is not always consistent; it could be called 60 times one second and 62 times the next (Aversa and Dickinson, 2019, 32). This could lead to performance issues when using Time.deltaTime as a valid input could be misread as false. Also, depending on the optimization of the project, i.e. if there is any memory overhead, this can directly affect the performance of Update (Nusrat et al., 2021, 7-8) and therefore affect the beat algorithm's performance. As the FixedUpdate() method in Unity has a fixed value of 20 milliseconds, or 50 updates a second and never varies (Dickinson, 2017, 156) the decision was taken to rewrite the singleton class to adapt to FixedUpdate().

The second attempt at the beat detection algorithm was developed in conjunction with the player and enemy movement scripts, with functionality adapted from an

existing GitHub repository (JJMaslen, 2024), again utilising the FixedUpdate loop. However, as FixedUpdate is set to 50 frames, rather than the theoretical 60 frames of Update, the beats per second must be converted for FixedUpdate(), shown in Figure 17.

```
public double bpmConversion(double bpm)
{
    double fixedUpdateBpm = 60 / bpm; //
    return fixedUpdateBpm;
}
int beat = (int)(bpmConversion(songBpm) * 50);
```

Figure 17: The code to convert the bpm for FixedUpdate() using the function bpmConversion()

The full FixedUpdate() code is shown in Figure 18.

```
void FixedUpdate()
{
    timer++;
    int beat = (int)(bpmConversion(songBpm) * 50); //converts beat to bpm
    int count = timer % beat; //check current time between beats
    if (count >= beat - margin) //if between beats (or slightly before)
    {
        if (onBeat == false)
        {
            onBeat = true;
            if(beatDone == false) beatDone = true;
        }
        onBeat = true;
    }
    else if (count <= margin) //accounts for lateness
    {
        onBeat = true;
    }
    else
    {
        if(beatDone == true)
        {
            beatDone = false;
            foreach(int id in enemyObjects.Keys.ToList()) //once beat is completed let enemies move again
            {
                enemyObjects[id] = true;
            }
        }
        onBeat = false;
    }
}
```

Figure 18: The FixedUpdate() code for the second beat detection implementation

A timer variable increases by 1 for every call to FixedUpdate() with a count variable used to check if the player is exactly on beat. If the modulus operator returns 0, then the player is confirmed to be in time with the music. If the modulus returns a value that is slightly above 0 (late input) or slightly below the value of the beat variable (early input), these are also counted as acceptable inputs. Otherwise, the player cannot move as they are not within beat. A new variable, onBeat, has been created to notify all game objects when they can move. As the singleton is static, its variables are constant when two different game objects access the onBeat variable and therefore notifies when the player/enemy can move, as shown in Figure 19.

```
void Update()
{

    if (Input.anyKeyDown)
    {
        if (man.onBeat == true) //code for movement taken from this tutorial: https://www.youtube.com/watch?v=mbzXIOKZurA
        {
            moving = true;
            if (Input.GetAxisRaw("Horizontal") == 1f || Input.GetAxisRaw("Horizontal") == -1f)
            {
                direction = Input.GetAxisRaw("Horizontal");
                horizontal = true;
                if (!Physics2D.OverlapCircle(transform.position += new Vector3(Input.GetAxisRaw("Horizontal") * rawDist, 0, 0), 0.2f, walls)) transform.position += new Vector3(Ir
                //this code will check to make sure that if, when moving, the player hits a collider, the player no longer moves
            }
            else if (Input.GetAxisRaw("Vertical") == 1f || Input.GetAxisRaw("Vertical") == -1f)
            {
                direction = Input.GetAxisRaw("Vertical");
                horizontal = false;
                if (!Physics2D.OverlapCircle(transform.position += new Vector3(0, Input.GetAxisRaw("Vertical") * rawDist, 0), 0.2f, walls)) transform.position += new Vector3(0f,
            }
            Debug.Log("BPM matched!"); //singleton used to access global information. If the user is pressing within beat, user can move
            score++;
            scoreText.text = "Score: " + score;
            //moving = false;
        }
        else
        {
            moving = false;
            Debug.Log("BPM failed!");
        }
    }
    if (man.onBeat == false) moving = false;
```

Figure 19: The use of onBeat for player movement, when onBeat is true the player is able to move

As an enemy could move at any point when onBeat was equal to true, the beatDone variable was created informing enemies they can only move once a full beat had elapsed. However, issues became apparent during playtesting. The game appeared to reject valid inputs that were in rhythm, causing beat detection to feel random. Therefore, a core aspect of flow “the ability to exercise a sense of control over actions” (Sweetser and Wyeth, 2005, 3) was not being realised. Johnson (2015, 7-8) notes that in the game Steel Battalion: Heavy Armor (FromSoftware, 2012) players often had inputs that did not perform their expected action, leading to a lack

of control which renders overcoming challenges as unrewarding as it involves overcoming a physical challenge from controls and not an in-game challenge. If the player only feels challenged from aspects outside the game, as in the current implementation, this means they are focusing on aspects outside the game and are not immersed. Therefore, the decision was taken to switch back to Update, as the previous concerns outlined above were considered negligible. It is highly unlikely that there will be significant memory overhead as the creation of the 2D game is not memory intensive and, whilst the number of loops in Update is not always 60, the lack of overhead means the performance of Update should be reliable enough for this to not be a concern. Furthermore, as the same computer is planned to be used during testing, the rendering and processing power should be consistent for each participant.

```
void Update()
{
    dTimer += Time.deltaTime;
    double bpm = bpmConversion(songBpm);
    if(dTimer >= bpm - dMargin || dTimer < 0.0+dMargin)
    {
        onBeat = true;
        if(beatDone == false) beatDone = true;
    }
    else
    {
        if(beatDone == true)
        {
            beatDone = false;
            foreach(int id in enemyObjects.Keys.ToList()) //once beat is completed let enemies move again
            {
                enemyObjects[id] = true;
            }
        }
        onBeat = false;
    }
    if(dTimer >= bpm) dTimer = 0;
}
```

Figure 20: The beat detection algorithm configured to the Update() loop

In Figure 20, the code reverts back to using Time.deltaTime as in Figure 16. A double is used to check whether or not the player is within beats per second. As the value returned from bpmConversion() is a double, there was a possibility that the earlier iteration may have lost precision when converting to an integer (Lindholm et al., 2013). Furthermore, rather than checking if, for every call, a value divides

cleanly with no remainder to determine if a player is exactly on beat, the Update loop instead determines that a beat cycle is performed if the timer value exceeds the beats per second.

However, despite these modifications, the same issue still occurred. As with the FixedUpdate() loop, the player does not feel in flow as it was hard to determine when they could move. It then became apparent that the issue was not because the player was not in time with the music, but because there was no visual cue. As already discussed by Lin et al., a rhythm game works by utilising a combination of visual and audio cues at an appropriate time, that being the period in which they are "on beat". Currently the player is relying solely on an audio cue, to detect when they are "on beat". However, as previously discussed, the audio reaction time is 284 milliseconds, meaning a player may always be slightly off beat when detecting this audio cue. For instance, for a 120 BPM song, the beats per second corresponds to 0.5. Should the reaction time in seconds be used for a margin, the player could be "on beat" in the following conditions: Player input < 0.284 or Player input > 0.216. Both these conditions overlap, meaning players can always move, meaning the margin must be less than the audio reaction time. It was therefore decided that a visual cue was needed to help the player sync to the BPM, which exists in Crypt. Furthermore, Bégel et al. (2017, 4) noted that, in 27 rhythm games analysed, "most of the games…consists in reacting to visual stimulations" implying that visual communication is a standard in rhythm games. A final version of the beat detection algorithm is shown in Figure 21.

```
Unity Message | 0 references
void Update()
{
    dTimer += Time.deltaTime;
    s.value = (float)dTimer;
    double bpm = bpmConversion(songBpm);
    if(dTimer >= bpm - dMargin || dTimer < 0.0+dMargin)
    {
        onBeat = true;
        s.gameObject.transform.Find("Fill Area").Find("Fill").GetComponent<Image>().color = green;
        if(beatDone == false) beatDone = true;
    }
    else
    {
        if(beatDone == true)
        {
            beatDone = false;
            foreach(int id in enemyObjects.Keys.ToList()) //once beat is completed let enemies move again
            {
                enemyObjects[id] = true;
            }
        }
        s.gameObject.transform.Find("Fill Area").Find("Fill").GetComponent<Image>().color = red;
        onBeat = false;
    }
    if (dTimer >= bpm) { dTimer = 0; s.value = 0f; }
}
```

Figure 21: The completed beat detection algorithm in Update()

A new game object has been assigned in GameManager which is a slider UI component, shown in Figure 22. If the player can move, this slider component is green, otherwise it is red. The colours were selected for the following reasons. Firstly, red is a colour with normally a negative association (Gil and Bigot, 2016, 1), i.e. danger (Pravossoudovitch et al., 2014, 2) whilst green is normally more positive, i.e. safety (Mammarella et al., 2016, 2). Therefore, applying these colour associations to the slider should help a player to understand when they can move, ideally easing players into the rhythm cycle. Costello (2018) defines rhythm as "something that patterns the flow of experience…to cycles of prediction and anticipation" meaning that as the player gains experience in matching their inputs to the visual cue, they will begin to become in tune with the rhythm no longer relying on the visual cue.

Figure 22: The slider for visual cues, showing green to inform the player they can move

More visual feedback for the player was required however. Figure 19 illustrates that, on the successful execution of a valid press, the user's score increases. This type of visual feedback is a motivator in rhythm games (Charbonneau et al., 2009, 3) and encourages players to stay in rhythm. However, there was no visual feedback to communicate to players that they had failed an input. Swink (2008, 119-124) states that an input signal needs to be processed by the game and provide some form of instantaneous feedback, otherwise the game feels unresponsive. As there is an instantaneous response on successful movement, the same was required to indicate failure to indicate the input was processed. The solution was to apply a screen shake effect for missed beats. The code for this effect, is shown in Figures 23-24. With this inclusion the beat detection algorithm was considered complete.

```
public class MissedBeat : MonoBehaviour
{
    1 reference
    public IEnumerator Shake(float duration, float magnitude)
    {
        Vector3 originalPos = transform.localPosition; //code taken from this video https://www.youtube.com/watch?v=9A9yj8KnM8c
        float elapsed = 0f;
        while(elapsed < duration)
        {
            float x = Random.Range(-1f, 1f) * magnitude;
            float y = Random.Range(-1f, 1f) * magnitude;
            transform.localPosition = new Vector3(x, y, originalPos.z);
            elapsed += Time.deltaTime;
            yield return null;
        }
        transform.localPosition = originalPos; //return to original position after coroutine has finished
    }
}
```

Figure 23: The script that controls the camera shake for missed beats

```
if (Input.anyKeyDown)
{
    if (man.onBeat == true) //code for movement taken from this tutorial: https://www.youtube.com/watch?v=mbzXIOKZurA
    {
        moving = true;
        if (Input.GetAxisRaw("Horizontal") == 1f || Input.GetAxisRaw("Horizontal") == -1f)
        {
            direction = Input.GetAxisRaw("Horizontal");
            horizontal = true;
            if (!Physics2D.OverlapCircle(transform.position += new Vector3(Input.GetAxisRaw("Horizontal") * rawDist, 0, 0), 0.2f, walls)) transform.position += new Vector3(Inp
            //this code will check to make sure that if, when moving, the player hits a collider, the player no longer moves
        }
        else if (Input.GetAxisRaw("Vertical") == 1f || Input.GetAxisRaw("Vertical") == -1f)
        {
            direction = Input.GetAxisRaw("Vertical");
            horizontal = false;
            if (!Physics2D.OverlapCircle(transform.position += new Vector3(0, Input.GetAxisRaw("Vertical") * rawDist, 0), 0.2f, walls)) transform.position += new Vector3(0f, 1
        }
        Debug.Log("BPM matched!"); //singleton used to access global information. If the user is pressing within beat, user can move
        score++;
        scoreText.text = "Score: " + score;
        //moving = false;
    }
    else
    {
        moving = false;
        StartCoroutine(mbeat.Shake(.15f, .4f));
        Debug.Log("BPM failed!");
    }
}
if (man.onBeat == false) moving = false;
```

Figure 24: The call of this camera shake script in player movement, when the player is not on beat

5.3.2 - Key and Colour Implementation

As discussed in the literature review, a change in key would be utilised to change the colour of the game, using colour grading to help elicit an emotional response in players. Unity has different post processing effects for colour grading, some of which were attempted achieving differing results. To set the colour used for the game, it was important to define a dictionary in a script, shown in Figure 25, that corresponds to each colour as defined by Castel's model in Figure 4.

```
public Dictionary<int, Color> keyColourMap = new Dictionary<int, Color> {
    {0, new Color(0,0,255) }, //blue
    {1, new Color(13,152,186) }, //blue green
    {2, new Color(0,255,0)}, //green
    {3, new Color(128,128,0)}, //olive green
    {4, new Color(255,255,0) }, //yellow
    {5, new Color(255,174,66) }, //yellow orange
    {6, new Color(255,165,0) }, //orange
    {7, new Color(255,0,0) }, //red
    {8, new Color(215,0,64) }, //carmine
    {9, new Color(143,0,255) }, //violet
    {10, new Color(138,43,226) }, //blue violet
    {11, new Color(75,0,130) } //indigo blue
};
```

Figure 25: Key to colour dictionary defined by Castel's model, each colour is shown by each comment

Each key in the dictionary corresponds to the music key, stored in the GameManager singleton. The first attempt at using colour was the bloom post processing effect in Unity. Bloom "produces fringes of light extending from the borders of bright areas in an image" (Unity, undated) and has a tint component for different colours. The effect of bloom on the game is shown in Figure 26.

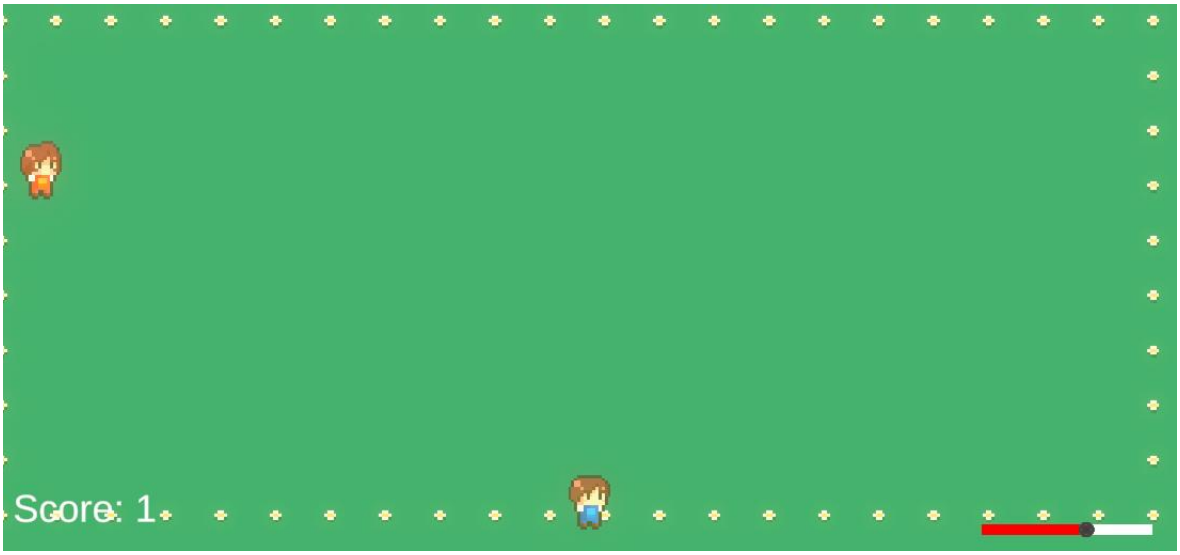


Figure 26: The use of Unity's bloom effect for a key of 4

The bloom effect for the colour violet is shown in Figure 27 alongside the code used to generate bloom.

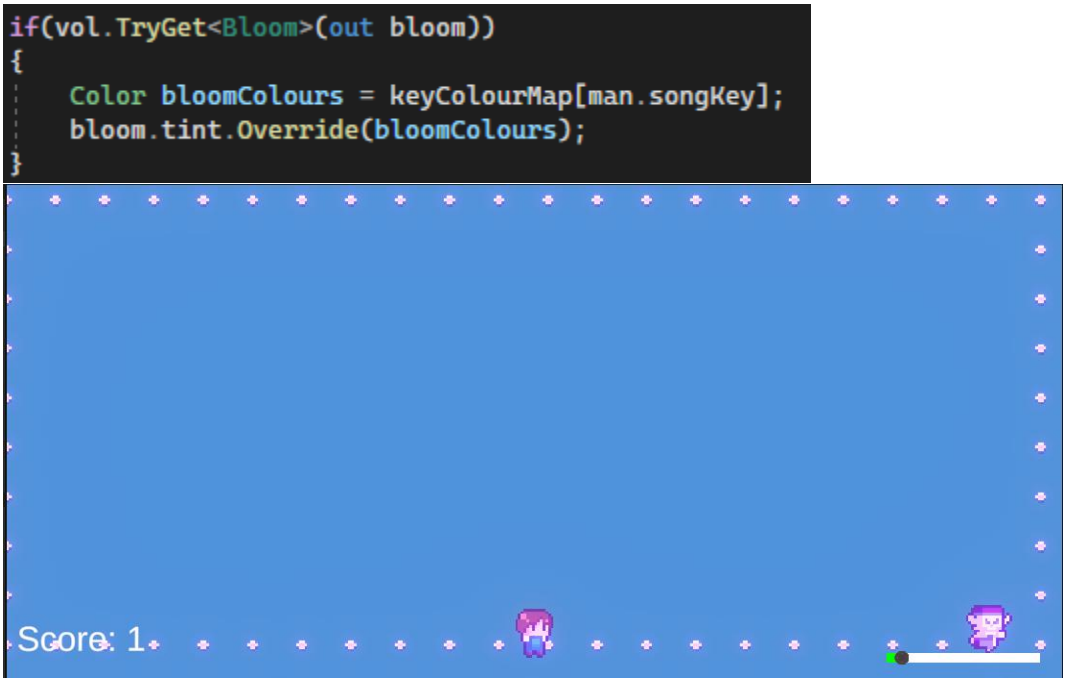


Figure 27: The code for bloom along with the effect of bloom for a key of 9

The main observations from using bloom are, whilst the tint offered a subtle colour effect, the colour of the background of the game in Figure 27 turned blue, rather than the usual green. This was a concern as, rather than the player receiving an emotional response from the song key, any emotional response could be attributed to the change in background colour. Therefore, a different effect was explored.

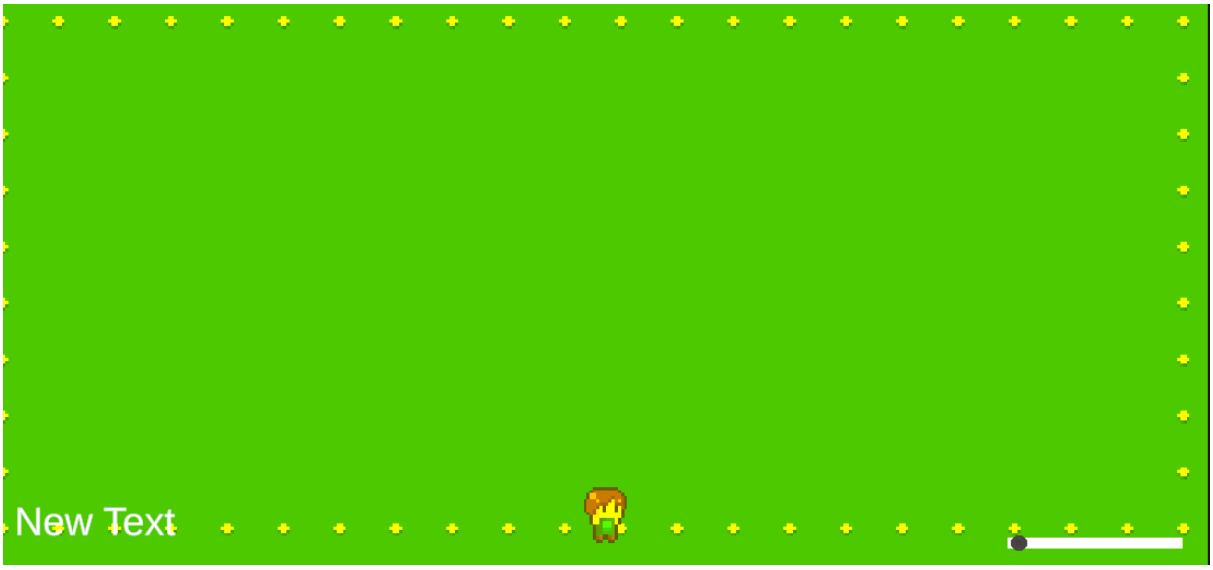
The second post processing effect tried was channel mixer which influences the components of each colour on the overall mix of the channel. For instance, if the influence of green increases in the red channel, shown in Figure 28, all aspects of the game scene that are green will have a redder hue (Unity, undated). The mixer is split into three components, the red, green and blue channels.

Figure 28: The red component of the channel mixer

However, as the channel mixer has a value between -200 to 200, it meant that the RGB component of each colour needed to be converted into a number the channel mixer could accept. The calculation is shown in Figure 29 and the resulting effects in Figure 30.

```
if(vol.TryGet<ChannelMixer>(out mixer))
{
    Color mixerColours = keyColourMap[man.songKey];
    mixer.redOutRedIn.Override(mixerColours.r * (200.0f / 255.0f)); //gets red channel of channel mixer and scales it between 0-200
    mixer.greenOutGreenIn.Override(mixerColours.g * (200.0f / 255.0f)); //gets green channel of channel mixer
    mixer.blueOutBlueIn.Override(mixerColours.b * (200.0f / 255.0f)); //gets blue channel of channel mixer
    Debug.Log(mixerColours.r);
}
```

Figure 29: The calculations required to convert RGB values for the channel mixer



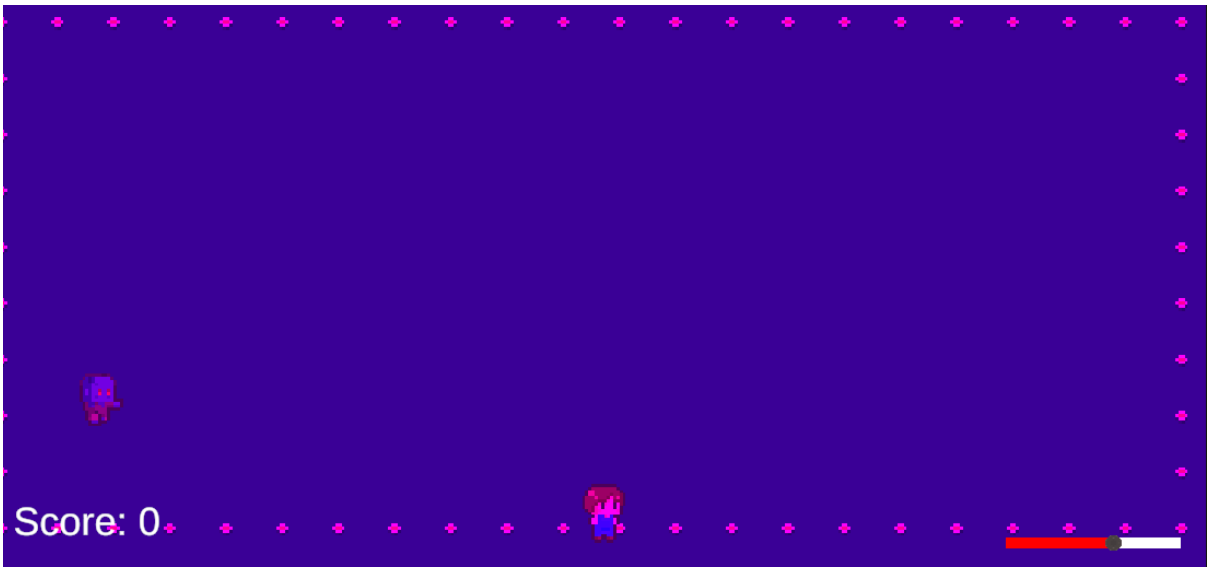


Figure 30: The result of using channel mixer for keys 4 and 9

Whilst Figure 30 does show both yellow and violet colours as expected, there are still issues with the visual output. Yellow can be seen slightly in the flowers for key 4, however the predominant effect of the mixer causes the background to become a fluorescent green. This means the same immersion issues as outlined for bloom remain. In fact, for a key of 0, a blue colour, the effect was worse as the game was completely enveloped in blue, making it difficult to see either the player or enemies in gameplay, as shown in Figure 31. One question in the IEQ related to players liking the visuals, as if players in the experimental game preferred the visuals (due to the use of different colours within the game) it contributes to greater immersion overall (Bowman and McMahan, 2007, 3). Therefore, this poor visual output negatively effects the immersive qualities of the experimental game.

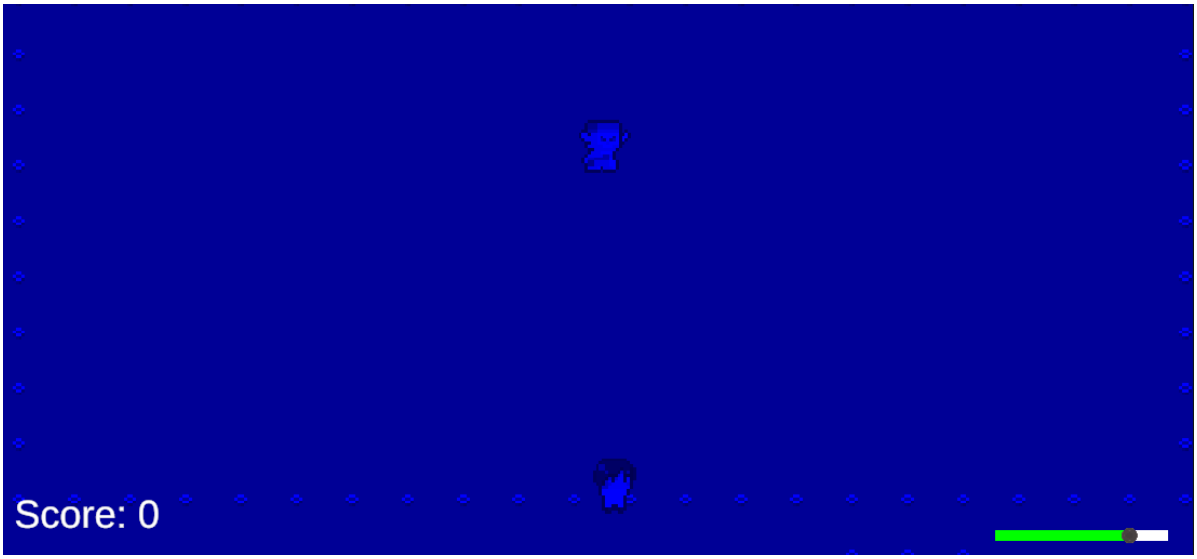


Figure 31: The result of using channel mixer on key 0

Neither post processing effect was working as intended, with the background being predominantly affected. It was therefore decided that a different approach to map colour was needed. Since Fonteles et al. (2013) had already used Castel's colour system in a successful particle system implementation, it was decided to incorporate Unity's own particle system utilising Castel's model, with the particles being emitted from the player as they are always present on the screen. The particle system utilises the same dictionary as the post processing implementation, illustrated in Figure 32, with the resulting screen also shown.

```
public Dictionary<int, Color> keyColourMap = new Dictionary<int, Color> {
    {0, new Color(0,0,255) }, //blue
    {1, new Color(13,152,186) }, //blue green
    {2, new Color(0,255,0)}, //green
    {3, new Color(128,128,0)}, //olive green
    {4, new Color(255,255,0) }, //yellow
    {5, new Color(255,174,66) }, //yellow orange
    {6, new Color(255,165,0) }, //orange
    {7, new Color(255,0,0) }, //red
    {8, new Color(215,0,64) }, //carmine
    {9, new Color(143,0,255) }, //violet
    {10, new Color(138,43,226) }, //blue violet
    {11, new Color(75,0,130) } //indigo blue
};
public ParticleSystem system;
// Start is called before the first frame update
Unity Message | 0 references
void Start()
{
    Color particleColours = keyColourMap[man.songKey];
    ParticleSystem.MainModule ma = system.main;
    ma.startColor = particleColours;
}
```

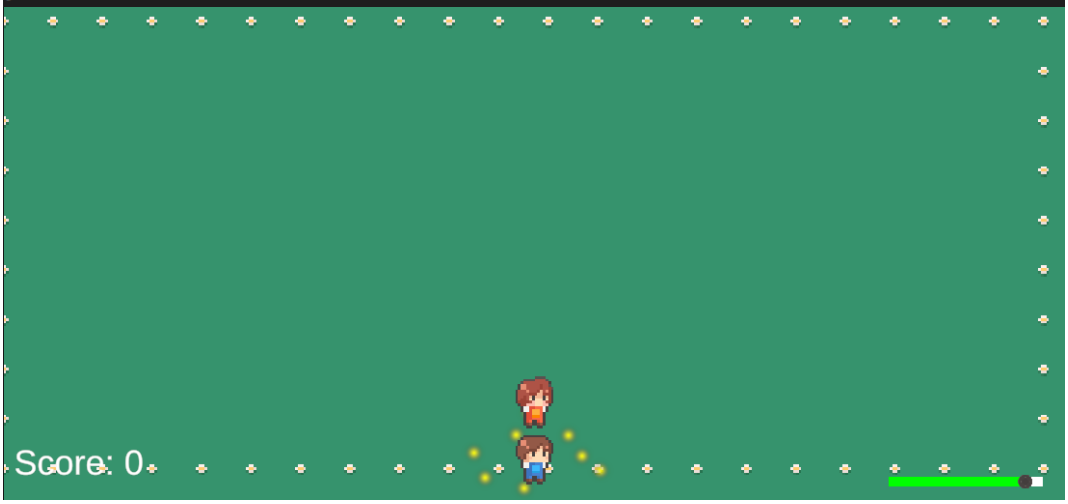


Figure 32: The use of a particle system to display colour, alongside the code required

It was not clear if the particle system was the best mapping to use for key. Particle systems in games are usually reserved for special effects, i.e. fire, smoke, explosions (Hastings et al., 2008, 1) and are usually triggered as a visual response to a game action, i.e. dust when a foot hits the ground (Swink, 2008, 152) conveying impact or motion (Swink, 2008, 159-160). However here, the particle system is triggered

automatically, regardless of player action. It has no effect on gameplay, nor does it convey information about game actions. When El-Nasr and Yan (2006, 4) describe player visual attention in 3D games, they note that certain visual effects, i.e. a bright object to indicate an item to improve player health, catches the player's attention. Since the current visual effect leads to no significant difference in gameplay or offers helpful feedback such as hinting at a useful item, it is likely that the player will lose interest in this effect over time as it represents nothing to the player (Swink, 2008, 171) rather than identify that it reflects the song key.

Consideration was given to removing the background colour entirely, instead using the background colour component of the camera as the post processing effects Unity offered were being adversely affected. Whilst the literature review stated that colour grading was an effective way to influence player emotions, the stated aim of the study is to attempt to see if a colour change, derived from key, leads to enhanced immersion and this does not need to come from colour grading. It was theorised that loading a level with the exact colour relating to the key should be striking and noticeable enough to enable the player to feel an emotion instigated by the colour change and not influenced by any external factors. The resulting script and effect are illustrated in Figure 33. With this feature implemented, the key and colour implementation aspect of the project was considered complete.

```
public Dictionary<int, Color32> keyColourMap = new Dictionary<int, Color32> {
    {0, new Color32(0,0,255, 255) }, //blue
    {1, new Color32(13,152,186, 255) }, //blue green
    {2, new Color32(0,255,0, 255)}, //green
    {3, new Color32(186,184,108, 255)}, //olive green
    {4, new Color32(255,255,0, 255) }, //yellow
    {5, new Color32(255,174,66, 255) }, //yellow orange
    {6, new Color32(255,165,0, 255) }, //orange
    {7, new Color32(255,0,0, 255) }, //red
    {8, new Color32(215,0,64, 255) }, //carmine
    {9, new Color32(143,0,255, 255) }, //violet
    {10, new Color32(138,43,226, 255) }, //blue violet
    {11, new Color32(75,0,130, 255) } //indigo blue
}; //Unity Colours are accepted as float values, so Color32 is required source
// Start is called before the first frame update
Unity Message | 0 references
void Start()
{
    man = GameManager.Instance;
    cam = Camera.main;
    cam.backgroundColor = keyColourMap[man.songKey];
    Debug.Log(cam.backgroundColor);
}
```

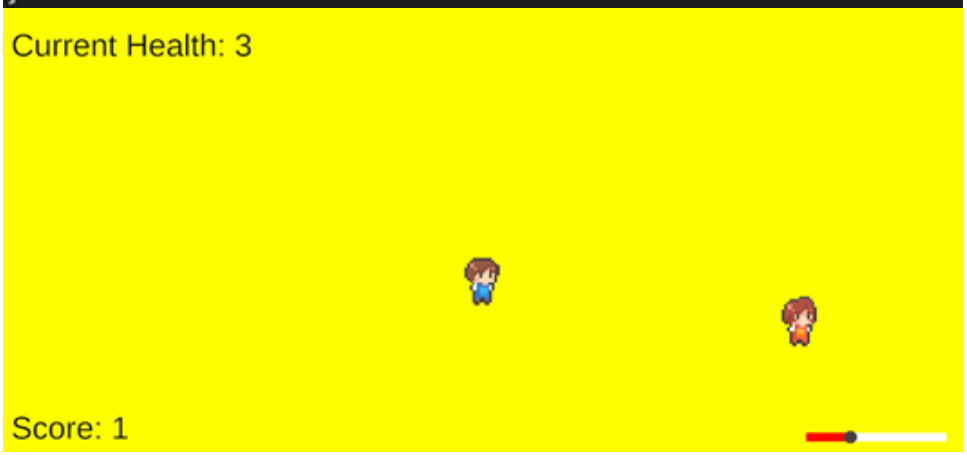


Figure 33: The script for changing the camera colour and the resulting game scene.

Note that some colour values have changed, as the previous RGB values for colours such as olive green were erroneous and needed amending. Also, Color32 has been used instead of Color as it is less memory intensive (Unity, 2015).

5.3.3 - Volume and enemies

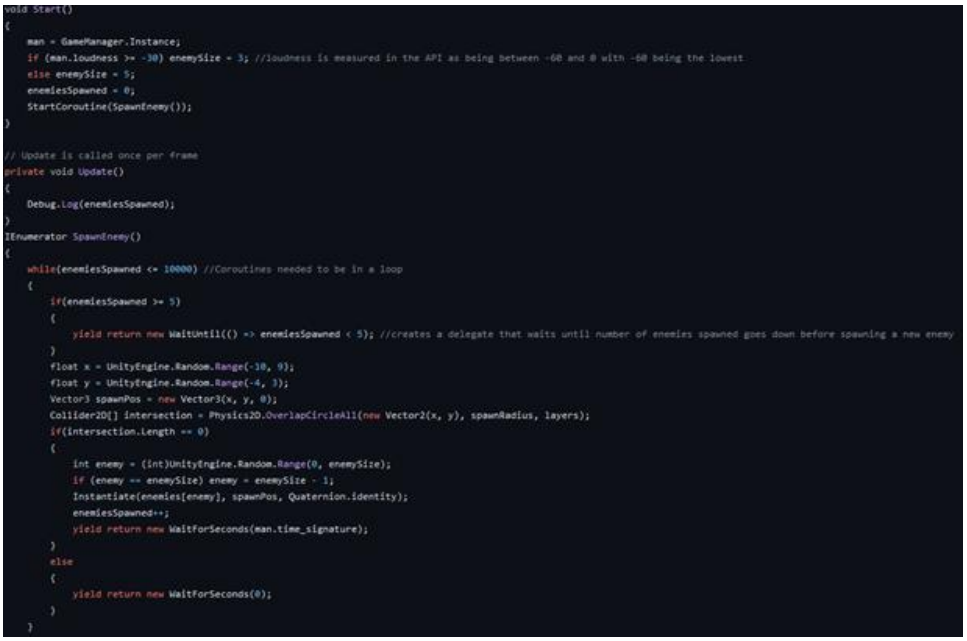
The literature review discussed how there tended to be a greater intensity for louder songs and thus that the game should reflect this. Difficulty was adapted as greater difficulty leads to more intense gaming experiences. Good game flow is achieved by offering players appropriate challenges, i.e. to surpass challenging opponents (Sweetser and Wyeth, 2005, 6) and so more challenging enemies were added to elicit this desired intensity for greater immersion. 5 enemies were designed for the game with 2 being reserved for louder songs. The code for all 5 enemies can be found in appendices E-I, however a summary of the 3 enemies common to all songs is shown below:

- Mimic: Moves in the opposite direction to player movement
- Zombie: If the horizontal distance to the player is less than the vertical distance, move horizontally, else move vertically
- Skeleton: Moves in a random direction

For louder songs:

- Strong zombie: Works similarly to a regular zombie, however has a larger attack radius
- Mage: Moves towards the player and shoots a projectile if the player is in range

Once each enemy was created, a script was required to spawn them, shown in Figure 34, based on the loudness property and time signature field in the singleton informed from Spotify.



```
void Start()
{
    man = GameManager.Instance;
    if (man.loudness >= -30) enemySize = 3; //loudness is measured in the API as being between -60 and 0 with -60 being the lowest
    else enemySize = 5;
    enemiesSpawned = 0;
    StartCoroutine(SpawnEnemy());
}

// Update is called once per frame
private void Update()
{
    Debug.Log(enemiesSpawned);
}
IEnumerator SpawnEnemy()
{
    while(enemiesSpawned <= 10000) //Coroutines needed to be in a loop
    {
        if(enemiesSpawned >= 5)
        {
            yield return new WaitUntil(() => enemiesSpawned < 5); //creates a delegate that waits until number of enemies spawned goes down before spawning a new enemy
        }
        float x = UnityEngine.Random.Range(-10, 9);
        float y = UnityEngine.Random.Range(-4, 3);
        Vector3 spawnPos = new Vector3(x, y, 0);
        Collider2D[] intersection = Physics2D.OverlapCircleAll(new Vector2(x, y), spawnRadius, layers);
        if(intersection.Length == 0)
        {
            int enemy = (int)UnityEngine.Random.Range(0, enemySize);
            if (enemy == enemySize) enemy = enemySize - 1;
            Instantiate(enemies[enemy], spawnPos, Quaternion.identity);
            enemiesSpawned++;
            yield return new WaitForSeconds(man.time_signature);
        }
        else
        {
            yield return new WaitForSeconds(0);
        }
    }
```

Figure 33: The code for spawning enemies

Spotify measures loudness between 0 and -60 Db, from quiet to loud, however it had initially been assumed that louder songs were below -30D, so for any song where this is not true, only the 3 easier enemies spawn, otherwise the spawn pool includes all enemies. Enemies are spawned in each level at a random empty location, with spawn rate dependant on the song's time signature. So that the player does not get overwhelmed by enemies and reach the anxiety zone of flow, the maximum number of spawnable enemies for the level is 5 with the coroutine only running if the number of enemies in the level does not exceed 5, otherwise it waits until the condition is true i.e. as soon as an enemy is killed, a new enemy will spawn.

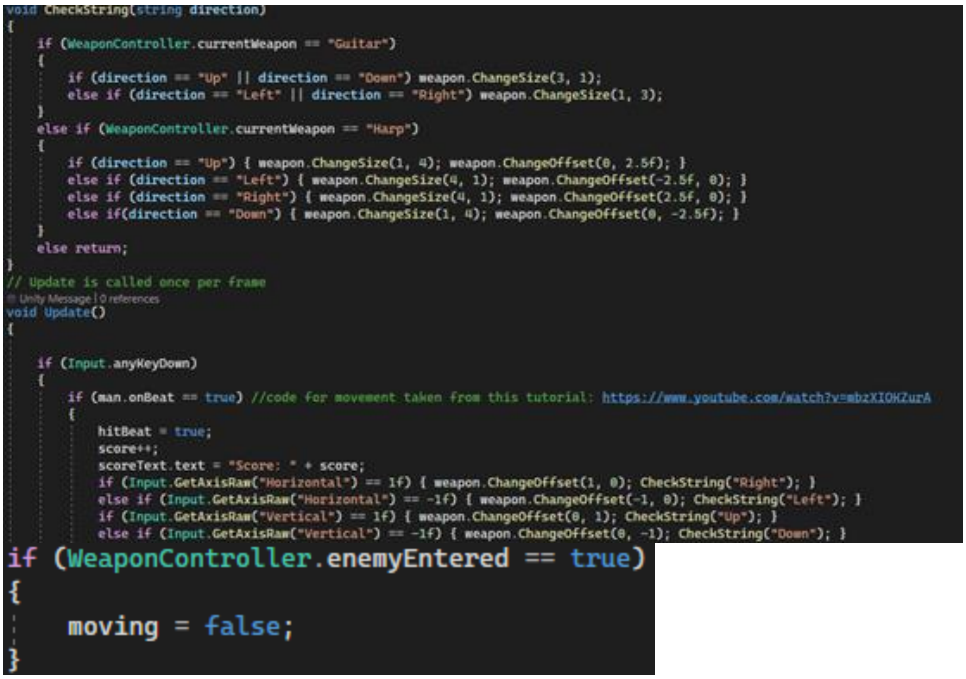
As a roguelike game is being created, the following elements are required. Firstly, the creation of turn based combat and movement, dictated by the rhythm. Secondly, a character with an inventory system to manage, achieved by adding durability to each weapon, meaning there is a requirement for players to manage weapon durability. Part of roguelike design is allowing players themselves to discover the functionality of different weapons, with each weapon demonstrating a different

attack style to satisfy this curiosity. Weapons spawn after durability reaches zero. It was initially planned to use the BoxCollider2D component as a child of the player, using a static variable to check the player's current weapon, adapting the strike zone depending on where the player moved. The relevant code is shown in Figures 35-37.

```
public class WeaponController : MonoBehaviour
{
    private BoxCollider2D boxCollider;
    public static int durability = 10;
    public static bool enemyEntered = false;
    public static string currentWeapon = "Baton";
    public GameObject player;
    private Rigidbody2D playerRb;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        boxCollider = this.GetComponent<BoxCollider2D>();
        boxCollider.offset = new Vector2(1, 0); //sets the collider for the weapon to be one square to the right
        playerRb = player.GetComponent<Rigidbody2D>();
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        if(currentWeapon == "Null")
        {
            ChangeOffset(0, 0);
            ChangeSize(0, 0);
        }
    }
    9 references
    public void ChangeOffset(float x, float y)
    {
        boxCollider.offset = new Vector2(x, y);
    }
    7 references
    public void ChangeSize(float x, float y)
    {
        boxCollider.size = new Vector2(x, y);
    }
}
```

Figure 34: The weapon controller script handling different weapon functionality



```
void CheckString(string direction)
{
    if (WeaponController.currentWeapon == "Guitar")
    {
        if (direction == "Up" || direction == "Down") weapon.ChangeSize(3, 1);
        else if (direction == "Left" || direction == "Right") weapon.ChangeSize(1, 3);
    }
    else if (WeaponController.currentWeapon == "Harp")
    {
        if (direction == "Up") { weapon.ChangeSize(1, 4); weapon.ChangeOffset(0, 2.5f); }
        else if (direction == "Left") { weapon.ChangeSize(4, 1); weapon.ChangeOffset(-2.5f, 0); }
        else if (direction == "Right") { weapon.ChangeSize(4, 1); weapon.ChangeOffset(2.5f, 0); }
        else if(direction == "Down") { weapon.ChangeSize(1, 4); weapon.ChangeOffset(0, -2.5f); }
    }
    else return;
}
// Update is called once per frame
Unity Message | 0 references
void Update()
{

    if (Input.anyKeyDown)
    {
        if (man.onBeat == true) //code for movement taken from this tutorial: https://www.youtube.com/watch?v=mbzXIOHZurA
        {
            hitBeat = true;
            score++;
            scoreText.text = "Score: " + score;
            if (Input.GetAxisRaw("Horizontal") == 1f) { weapon.ChangeOffset(1, 0); CheckString("Right"); }
            else if (Input.GetAxisRaw("Horizontal") == -1f) { weapon.ChangeOffset(-1, 0); CheckString("Left"); }
            if (Input.GetAxisRaw("Vertical") == 1f) { weapon.ChangeOffset(0, 1); CheckString("Up"); }
            else if (Input.GetAxisRaw("Vertical") == -1f) { weapon.ChangeOffset(0, -1); CheckString("Down"); }
```

```
if (WeaponController.enemyEntered == true)
{
    moving = false;
}
```

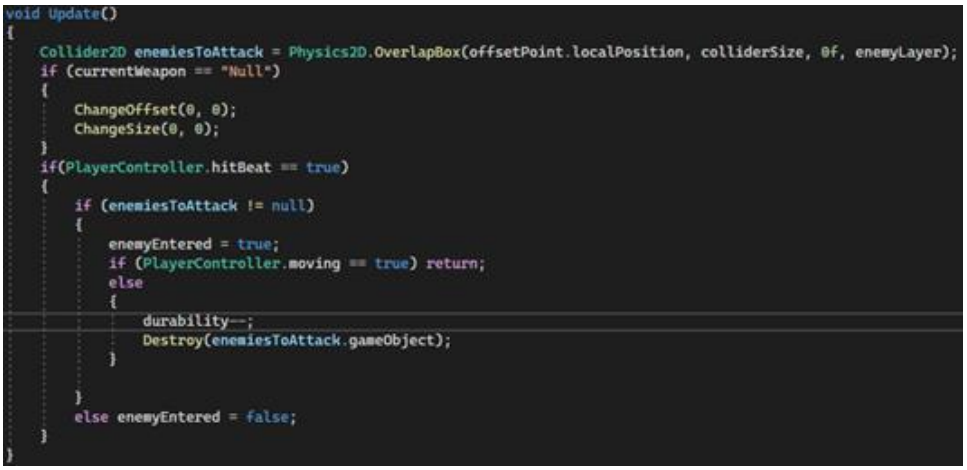
Figure 35: The PlayerController code that adapts the weapon zone depending on movement and aims to stop player movement is an enemy is within range

```
private void OnTriggerStay2D(Collider2D collision)
{
    if (collision.tag == "WeaponZone")
    {
        WeaponController.enemyEntered = true;
        if (PlayerController.hitBeat == true && WeaponController.enemyEntered == true)
        {
            Debug.Log("This is called");
            WeaponController.durability--;
            Debug.Log(WeaponController.durability);
            if (WeaponController.durability <= 0)
            {
                WeaponController.currentWeapon = "Null";
            }
            WeaponController.enemyEntered = false;
            Destroy(this.gameObject);
        }
    }
}
```

Figure 36: The trigger function in Enemy that allows weapons to attack enemies

However, it was found that this method violated a principal component of roguelike combat, that a player should not be able to move and attack simultaneously (Izgi, 2018, 13). This was due to a trigger in the enemy script, allowing weapons to attack enemies. OnTriggerStay2D is called on every frame, causing an instantaneous

response once the player had both moved and entered an enemy collider, allowing both actions to occur at the same time. Rather than creating a BoxCollider2D on the child of the player, this component was removed and a Physics.OverlapBoxAll used instead. This checks if any enemies were present inside the box and, if so, allows the player to attack. This rendered the use of the trigger function in the Enemy script obsolete. The revised WeaponController script is shown below.



```
void Update()
{
    Collider2D enemiesToAttack = Physics2D.OverlapBox(offsetPoint.localPosition, colliderSize, 0f, enemyLayer);
    if (currentWeapon == "Null")
    {
        ChangeOffset(0, 0);
        ChangeSize(0, 0);
    }
    if(PlayerController.hitBeat == true)
    {
        if (enemiesToAttack != null)
        {
            enemyEntered = true;
            if (PlayerController.moving == true) return;
            else
            {
                durability--;
                Destroy(enemiesToAttack.gameObject);
            }

        }
        else enemyEntered = false;
    }
}
```

Figure 37: Revised Weapon Controller, using the Physics2D method

Once the box collider is drawn, a check is completed to detect if the player is currently moving. If true, the enemy should not be attacked, otherwise, any enemies in the box are attacked and the durability of the weapon decreases. The game now had functioning combat and variants of difficulty via enemy spawning. A further observation is that, enemies should not move on every instance that the GameManager is “on beat”. Currently enemies move on every instance where this is true. However, in Crypt, most enemies tend to have an interval of beats between moving. Therefore, a further condition for an enemy is required, that being to ensure that the enemy waits a set number of beats before moving, shown in the Figure 39 Zombie script.

```
beatCounter++;
float positiveHorizontal = horizontalDistance;
float positiveVertical = verticalDistance;
positiveHorizontal = Mathf.Abs(positiveHorizontal);
positiveVertical = Mathf.Abs(positiveVertical);
if(beatCounter >= beatThreshold)
{
    if (positiveHorizontal < 1f && positiveVertical < 1f)
    {
        moveDirection = new Vector3(0, -rawDist, 0);
    }
```

Figure 38: Limiting zombie movement in the ZombieController script

When onBeat is true a variable called beatCounter increases, with the enemy moving if beatCounter passes a threshold, this being the number of beats required to pass before an enemy can move again, with the counter resetting once true.

Finally, there needed to be a way for enemies to attack the player, shown in Figure 40, achieved by assigning a separate child object to the player, a circle trigger collider that damages the player when entered by enemies. To ensure that the player does not get repeatedly attacked, the enemy cannot attack again for three seconds. The same screen shake effect as shown in Figure 23 is used to communicate to the player that they have been attacked.

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (this.gameObject.tag == "TriggerZone") return;

    if (collision.gameObject.tag == "Enemy")
    {
        enemyAttack = true;
        PlayerHealth.TakeDamage();
        StartCoroutine(mBeat.Shake(.15f, .4f));
        if (collision.gameObject.name.Contains("Mage"))
        {
            mageController = collision.gameObject.GetComponent<MageController>();
            mageController.enabled = false;
        }
        else if(collision.gameObject.name.Contains("Zombie"))
        {
            zombieController = collision.gameObject.GetComponent<ZombieController>();
            zombieController.enabled = false;
        }
        else if(collision.gameObject.name.Contains("Mimic"))
        {
            mimicController = collision.gameObject.GetComponent<MimicController>();
            mimicController.enabled = false;
        }
        else if (collision.gameObject.name.Contains("Skeleton"))
        {
            skeletonController = collision.gameObject.GetComponent<SkeletonController>();
            skeletonController.enabled = false;
        }
    }
}
private void OnTriggerExit2D(Collider2D collision)
{
    if (this.gameObject.tag == "TriggerZone") return;
    if (collision.gameObject.tag == "Enemy")
    {
        enemyAttack = false;
        if (collision.gameObject.name.Contains("Mage"))
        {
            Invoke("StartMage", 3f);
        }
        else if (collision.gameObject.name.Contains("Zombie"))
        {
            Invoke("StartZombie", 3f);
        }
        else if (collision.gameObject.name.Contains("Mimic"))
        {
            Invoke("StartMimic", 3f);
        }
        else if (collision.gameObject.name.Contains("Skeleton"))
        {
            Invoke("StartSkeleton", 3f);
        }
    }
}
0 references
void StartMage()
{
    mageController.enabled = true;
}
```

Figure 39: The code for enemy attacks, should the trigger collider on players be entered by an enemy

Note there is a static function called PlayerHealth.TakeDamage(), which removes health from the player on enemy attacks. Now that all the baseline game features had been implemented, it was time to incorporate the JSON files into the game.

5.3.4 - JSON files and Unity

Objective 1 stated that 5 songs were picked that were musically distinct from each other, meaning that the tempo, volume and key differed. Table 4 outlines the 5 songs chosen.

Song	Artist	Tempo	Volume	Key	Citation
Reckoner (Song 1)	Radiohead	104.271	-7.441	4	(Radiohead, 2007)
Twilight (Song 2)	Electric Light Orchestra	139.588	-10.876	0	(Electric Light Orchestra, 1981)
Little Dark Age (Song 3)	MGMT	97.512	-6.156	6	(MGMT, 2017)
Shimmy (Song 4)	System of a Down	118.97	-3.556	2	(System of a Down, 2001)
She's Lost Control (Song 5)	Joy Division	144,246	-7.63	11	(Joy Division, 1979)

Table 4: Songs used in this project

Each JSON file was generated independently using the Python script developed in section 5.1, with the files generated integrated into the Unity project. However, JSON files are structured as follows, as “dictionaries…of key-value pairs” (Bourhis et al., 2017,1). The JSON file currently returned has the following key-value pairs. Firstly, a key of “meta” which is associated with various meta data such as the platform used to read the data relating to the track (Spotify, undated). Then a further key, that being “track”, containing the audio data required for the project (Spotify, undated). Further keys exist, including “bars”, “beats”, all containing different elements relating to the song. The function JsonUtility.FromJson() is required to load this data in Unity, taking the fields from the JSON file (these being the keys) and assigning them to variables in a serializable object. However, the following issue was found. As each field in the JSON file was effectively a nested object, this also requires the serializable object in Unity to have objects nested inside, an example of which is shown in Figure 41.

```
[System.Serializable]
1 reference
public class Meta
{
    //information relating to meta
}
[System.Serializable]
1 reference
public class Track
{
    //information relating to track
}
[System.Serializable]
0 references
public class JsonObject
{
    Meta m;
    Track t;
}
```

Figure 40: The initial implementation of JSON files in Unity

This started to become complex as there were now multiple keys that linked to different objects within the JSON file. However, as the risk analysis identified that, if there were any performance issues, the average key could be used it was decided to adapt the JSON file returned, with most information except for “track” not needed for the project. Therefore, the only object that needed to be serialized in the JSON file was “track”. A further consideration was that there was less risk of players feeling unwell should the background colour continually change by transitioning between keys. Keeping to an average tempo allows a consistent difficulty throughout the song meaning that new players do not feel out of sync by continually adjusting to different tempos, negating the risk of alienating new players. Furthermore, analysing the Spotify data, any differences in tempo throughout each segment were minor and would therefore be negligible to the player as shown in Table 5 for Reckoner, where tempo tended to stay at roughly 104 BPM. Therefore, the Python code was updated in Figure 42.

Section	Tempo
1	104.333
2	104.314
3	104.227
4	104.313
5	104.319
6	103.237
7	104.494

Table 5: The differing tempos for the song Reckoner

```
track = track_info["track"] #stores all informat
with open('musicdata.json', 'w') as exportFile:
    json.dump(track, exportFile)
```

Figure 41: The updated Python code, only exporting the "track" JSON key

This Python code now only contains information for the track object that is passed through to Unity. The updated file is shown below for Reckoner.

Figure 42: The updated JSON file for Reckoner, which no longer has the meta key

Now that the JSON file loads the relevant information for the game, all that was required was to parse these variables into Unity and into the GameManager

singleton, so that they could be used in game. In order to allow the player to choose which song they wished to play, Figure 44 shows a new scene with 5 buttons each corresponding to a different song.

Figure 43: The Music Selection scene, containing five buttons that load a level of a specific song attached on the button

The following script, CanvasSelection, assigned information from the JSON file as it contained a System.Serializable class called track, with variables corresponding to variables in the JSON object, shown in Figure 45.

```
public class Track // https://stackoverflow.com/questions/45215320/how-to-take-a-particular-field-from-json-data-in-unity
{
    public int num_samples;
    public double duration;
    public string sample_md5;
    public int offset_seconds;
    public int window_seconds;
    public int analysis_sample_rate;
    public int analysis_channels;
    public double end_of_fade_in;
    public double starta_of_fade_out;
    public float loudness;
    public float tempo;
    public double tempo_confidence;
    public int time_signature;
    public double time_signature_confidence;
    public int key;
    public double key_confidence;
    public int mode;
    public double mode_confidence;
    public string codestring;
    public double code_version;
    public string echoprintstring;
    public double echoprint_version;
    public string synchstring;
    public double synch_version;
    public string rhythmstring;
    public double rhythm_version;
}
```

Figure 44: The Track object in CanvasSelection containing information fields found in the JSON file

Once created, the JSON file was loaded in using the aforementioned method. This method returns an object created from the JSON file, with the variables of this object assigned to the corresponding values in the singleton. The related song also needed to be loaded into the game, created in a separate method, with both methods executed when the related button is clicked. Both methods are shown in Figure 46.

```
public void MusicLoader(TextAsset pathName)
{
    m = JsonUtility.FromJson<Track>(pathName.text);
    Debug.Log(m.loudness);
    man.loudness = m.loudness;
    man.songBpm = m.tempo;
    man.songKey = m.key;
    man.time_signature = m.time_signature;
    man.duration = (float)m.duration;
    man.setTimer();
    man.counter = 0;
    double health = HealthSetter(m.duration);
    PlayerHealth.setHealth((int)health);
    WeaponController.durability = 5;
    SceneManager.LoadScene("GameScene");
}
public void AudioLoader(AudioClip a)
{
    man.music.clip = a;
    man.startMusic();
}
```

Figure 45: The MusicLoader and AudioLouder methods, which pass information to the singleton and load the song relating to the JSON file respectively

As a singleton exists as a static object, its variables remain accessible between scenes allowing the JSON information to be carried over between scenes. With the JSON files successfully parsed, the next step was polishing and playtesting the game.

5.3.5 - Polishing and Playtesting

After the songs had been added to the game, a modification for enemy spawning was required. Currently, if the volume of a song was less than -30 dB, then the size of the enemy pool increased to include more difficult enemies. However, none of the songs fell below -30 dB. Furthermore, it was realised that the initial

understanding of this parameter was incorrect. When measuring loudness in decibels, the scale is normally positive i.e. “threshold for human hearing is set at 0 dB; painful sound is 140 dB” (Knauert et al., 2016, 1). However, Spotify’s loudness is measured from -60 to 0. In section 5.3.3 the assumption was that Spotify measured volume backwards. However, further research discovered that negative dB values in fact indicate “a decrease in … loudness” (Sziklai et al., 2011, 3). Therefore, the code now be needs to check if the song has greater than the average loudness of all 5 chosen songs. This was calculated to be -7.1318 dB.

It was also felt that the game needed more “juiciness” defined as “one player action triggers multiple non-functional reactions in the game” in the form of excessive positive feedback (Hicks et al., 2019, 1). Adding juiciness to a game heightens both player experience and motivation, providing the level of juiciness isn’t extreme (Kao, 2020, 1) as this can cause irritation for being overly excessive (Kao, 2020, 7). In order to facilitate the positive feedback that juiciness allows, a particle effect was added when an enemy is killed by the player. This helps to inform the player that they have performed a positive action and gives intrinsic motivation to continue attacking enemies. It was felt that the prior lack of juiciness may have caused the game to feel boring as there was less motivation to attack enemies, meaning lower immersion. This is because the player is no longer focusing on the main goal, to survive, instead focusing on other goals, potentially outside of the game due to boredom removing them from the game experience (Bench and Lench, 2013, 1). The code for this particle system and the resulting effect are shown in Figure 47.

```
Unity Message | 0 references
public void OnDestroy()
{
    Instantiate(enemyParticles, transform.position, transform.rotation);
    GameManager.Instance.UnregisterEnemy(GetInstanceID());
}
```

Figure 46: The particle effect after enemy death and the code to trigger this

Note that the particle code is assigned to the parent class Enemy and inherited by each enemy. As the OnDestroy() method is the same for each enemy it is more efficient for this to be inherited. Whilst the use of this particle system leads to all particle systems being played when transitioning between scenes, as all enemy game objects have been destroyed, this effect unintentionally allows the game to feel "juicy" as it provides feedback that the level is over (Kao, 2020, 3).

Playtesting is important and normally happens throughout the design process (Fullerton, 2014, 272). Initial playtesting was implemented by the game designer to ensure that the core mechanics were established and that any problems with the core design were resolved (Fullerton, 2014, 272-273), with the results of this shown throughout section 5.3. Now that this had been completed it was important to playtest the game with both the supervisor and associates of the designer, to ensure that "they have enough information to begin playing" (Fullerton, 2014, 273) meaning that they are able to understand how the game is to be played. A summary of the feedback from this stage is outlined below, adding new items to the SCRUM backlog:

- Players were unsure when to attack
- Players did not understand the difference between weapons

- Players were initially unsure how to play the game
- The margin of error felt too unforgiving
- There was no penalty for failing a beat
- There was no gameplay reward for killing an enemy
- It was very hard to game over

Most of these critiques stem from poor flow and could cause frustration in players, leading them to quit the game (Ašeriškis and Damaševičius, 2017, 3) with this negative experience causing a biased perspective when filling the IEQ.

Games must support player skill development and mastery which involves players being taught to play the game (Sweetser and Wyeth, 2005, 5). As the player is confused as to how the game itself is played, i.e. when to attack and the differences between weapons, then this flow criteria is not being achieved. Furthermore, the margin of error was unforgiving, meaning the skill level of the player did not match the game difficulty and thus led to anxiety (Sweetser and Wyeth, 2005, 6).

While juiciness was added to the enemy for intrinsic motivation, there was no gameplay reward from this event, as rewards promote a positive game experience and affect performance by giving better motivation (Denisova and Cook, 2019, 2). Conversely, receiving no penalty for failing a beat, led to little motivation and challenge for the player. Players also found the game too easy, therefore increasing boredom levels in the game (Sweetser and Wyeth, 2005, 6). The lack of game sound effects when attacking enemies also made the game feel less polished. Swink (2008, 160-161) defines sound effects as a polish metric, for instance to indicate the impact of two objects. Sound effects also greatly increase immersion (Sweetser and Johnson, 2004, 5). Therefore, sound effects were added once an enemy was defeated, to generate a better feeling game and experience, with the sound varying dependant on the weapon used, representing different object interactions. The

margin of error was also increased to be more forgiving and so prevent player anxiety.

In order to provide the player with more motivation to both attack enemies and to successfully execute a beat, the score doubles once an enemy is defeated with the score decreasing on failure, replicating similar features in existing rhythm games. An increase in score acts as a reward mechanism for doing well (Chen et al., 2016, 2) whilst scores are often reduced on beat mistakes (Chen et al., 2016, 1). A tutorial screen was included, describing how the game is played to make it more understandable. The screen is toggled by the use of a button meaning experienced players can bypass this screen. Whilst this violates the procedure of flow, that "learning the game should not be boring, but be part of the fun" (Sweetser and Wyeth, 2005, 5), as information relating to the game is displayed in a similar way for existing rhythm games published on itch.io (Zong and Shannon, 2023) this was deemed acceptable. There was also no clear indication of the difficulty level of each song despite the songs having varying tempos. Therefore, the instruction screen was amended to include this information enabling players to pick songs that matched their skill level. This was needed as the risk analysis stated players new to rhythm games may need to begin with a slower song before proceeding to faster songs.

An arrow has been added to the screen to indicate where a player can attack after moving their character, shown in Figure 47. This was as a result of feedback indicating that users felt they had no way of knowing if their actions had any effect on the game state and thus led to frustration (Atanasov, 2013, 20). The code for displaying the arrow is shown in Figure 48.

```
void RenderArrow()
{
    renderArrow = !renderArrow;
}
1 reference
IEnumerator ArrowShowcase()
{
    while(true)
    {
        if (currentWeapon == "Null") arrow.enabled = false;
        else
        {
            arrow.transform.position = offsetPoint.localPosition;
            arrow.transform.rotation = arrowRotation;
            arrow.enabled = true;
            Debug.Log("Calling Coroutine");
            RenderArrow();
        }
        yield return new WaitForSeconds(1.0f); //this gets called to wait a second before derending the arrow
        arrow.enabled = false;
        yield return new WaitUntil(() => PlayerController.hitBeat == true); //waits until the player hits a beat before re rendering the arrow
    }
}
```

Figure 47: Arrow display code that is used to inform players the direction of attack

This coroutine waits until the player has hit a beat successfully, at which point an arrow is rendered on the local position of the weapon zone child object in player. The arrow is displayed only when the player currently has a weapon and will only be displayed for a second. In order to give players more context about how each weapon is utilised, more text was added to the canvas so that the player received appropriate feedback and information. This better facilitates the discoverability aspect of roguelikes as the player is provided with information that encourages them to experiment with the weapon. The updated PlayerController code is shown in Figure 49.

```
2 references
void DisplayInfo()
{
    infoText.enabled = true;
    if (WeaponController.currentWeapon == "Baton") infoText.text = "Default weapon. Can only attack in front of you";
    else if (WeaponController.currentWeapon == "Guitar") infoText.text = "Wider width of attack!";
    else if (WeaponController.currentWeapon == "Harp") infoText.text = "Wider height of attack";
    Invoke("RemoveText", 3f);
}
0 references
void RemoveText()
{
    infoText.enabled = false;
}
```

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if(collision.tag == "Weapon" && this.gameObject.name == "Player")
    {
        Debug.Log("Collected by" + this.gameObject.name);
        if (collision.name.Contains("Baton")) WeaponController.currentWeapon = "Baton";
        else if (collision.name.Contains("Guitar")) WeaponController.currentWeapon = "Guitar";
        else if (collision.name.Contains("Harp")) WeaponController.currentWeapon = "Harp";
        DisplayInfo();
        Debug.Log(WeaponController.currentWeapon);
        Destroy(collision.gameObject);
        WeaponController.durability = 5;
        WeaponSpawner.spawnWeapon = false;
    }
}
```

Figure 48: The code to display weapon information in PlayerController

Originally, when a player was attacked by an enemy, a camera shake effect was played similar to when a player missed a beat. Ideally feedback should act as communication "between player and game… so that players can understand and act accordingly in any situation" (Atanasov, 2013, 24). However, players may now be confused and uncertain as to their next action as the feedback could relate to two different scenarios: a missed input or enemy attack. To remove this ambiguity, separate feedback was created for an enemy attack, achieved by applying a film grain filter via post processing (Unity, undated) which now allows players to have better knowledge of their status enabling better flow (Sweetser and Wyeth, 2005, 6).

Consideration was also given to whether the current implementation of colour was noticeable to players. In the current state, only a single flat background colour was being displayed, initially due to existing post processing effects manipulating the green background. It is known that flat colours in a game can work well with certain aesthetics, for example they work for a more cartoonish game than a realistic game (Cheng and Cairns, 2005, 2) and therefore works within the context of the created game. However, from the literature it was also known how post processing effects can influence the immersion of the player. It was therefore decided to retry bloom

and evaluate its effect. Figure 50 below shows the resulting effects from all 5 key to colour mappings in the game for yellow, blue, orange, green and indigo blue.

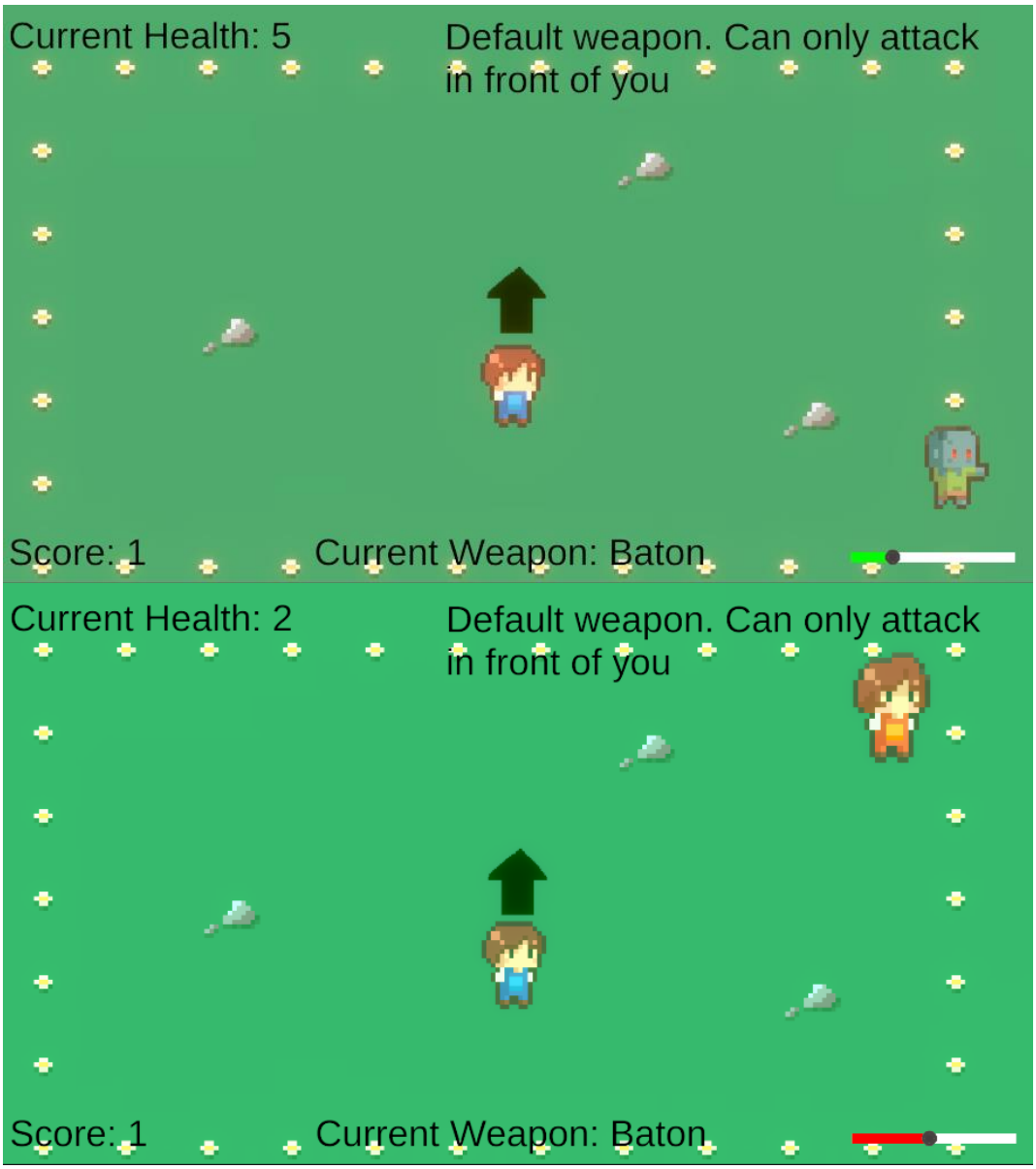

Current Health: 5
Default weapon. Can only attack in front of you
Score: 1
Current Weapon: Baton
Current Health: 2
Default weapon. Can only attack in front of you
Score: 1
Current Weapon: Baton

Figure 49: The results of bloom for all five colours of yellow, blue, orange, green and indigo blue

It was initially felt that bloom's effect was too subtle, so may not be perceived by players. In certain cases, this caused green background was changed to blue, with this scenario occurring when colours are "colder", i.e. blue and indigo blue. Each game object received a tint of each respective colour in Figure 50. When the supervisor was asked if it was clear that each colour was seen, the supervisor responded positively. It was therefore decided to revert to using bloom as it was felt that the provision of bloom elicits better immersion, corroborated by existing literature. The use of post processing could also allow for a more detailed background, as shown by tiles containing rocks and flowers in Figure 50. Schwartz (2006, 2) found that for existing games such as Shenmue (Sega-AM2, 1999) players found the game more realistic if there were detailed environments, regardless of how fantastical they seemed. Since a flat background colour offers no detail, it also goes against the metaphor metric of treatment with Swink (2008, 171) describing specifically that "the cohesive whole formed by visual art, visual effects, sound effect…and music" is lessened. Whilst the sound effects or character art provide detail for the game, the background itself is simple and abstract, causing some

"dissonance in the player" (Swink, 2008, 172) as one aspect of the treatment contradicts all other aspects. Therefore, it was felt that overall bloom was better than using a flat colour to represent the key.

With the results of playtesting completed, the game was ready for official testing.

Chapter 6 - Results & Discussion

6.1 - Control Game Creation

Certain requirements for the experimental game were also applicable to the control game. Although the control game required JSON objects to be parsed through, only tempo was required for use in the beat detection algorithm for player/enemy movement. Other game aspects preserved included the use of weapon durability with enemy spawning being based on a constant value of four, rather than time signature. As the existing songs used had a time signature of four anyway, this means the control and experimental games will always have the same enemy spawning rate. Furthermore, key and volume, the two differential features that are measured in the experimental game, are removed from the control game to determine if tempo alone leads to a less immersive experience. This means that the post processing effect of bloom is also removed, so there is no longer a colour shift to affect the player's emotions. The enemy pool is limited to the three default enemies reserved in the experimental game for quieter songs. This determines if the intensity was still the same in the control game despite there being no influence from volume with no harder enemies being spawned. A screenshot of the control game is shown in Figure 51, which shows gameplay without the colour change feature.

Figure 50: The control game showing song 2 during gameplay – the player has moved upwards and aims to attack the zombie enemy

6.2 - Immersion Questionnaire Results

The IEQ results are shown in Appendix J from the testing procedure described in 4.5.1. No participants withdrew from the study. Observational data was recorded for all participants, however the first three control observations had to be discarded as the tester did not accurately record the number of missed beats during play leading to inaccurate data. All other aspects of the procedure were followed correctly, meaning all IEQ scores could be used. The score for each IEQ was calculated using Nordin et al. (2014)'s method alongside the existing immersion study Thompson et al. (2012). In Thompson et al's. (2012, 3) study they specify that questions "6, 8, 9, 10, 18 and 20 (are) marked negatively" therefore the corresponding questions in this study are also marked negatively. An extra question "Did you ever want to stop playing the game at any point?" was added to this study and also marked negatively. This questionnaire also facilitated the last requirement in Chapter 3, as certain GEQ questions were

adapted to fit this IEQ, as discussed in 4.5. The final four questions were created to determine if key and volume were having an effect, i.e. a higher score when asking if the game was intense in the control game meant that volume's use in the experimental game provided a negligible effect.

The general hypothesis was as follows: "Players should be more immersed in the experimental game than the control game" with the use of key and volume being perceived as enhancing immersion in either an emotional or exciting way.

The overall mean immersion score for the experimental game was 132.5 and 121.9 for the control game, while standard deviation was 14.3 (3sf) for experimental and 8.39 (3sf) for the control. This therefore would seem to imply that the research question was met: the more musical features used, the greater the effect of immersion on the game itself. Figure 52 shows the results for both the control and experimental games in box plots.

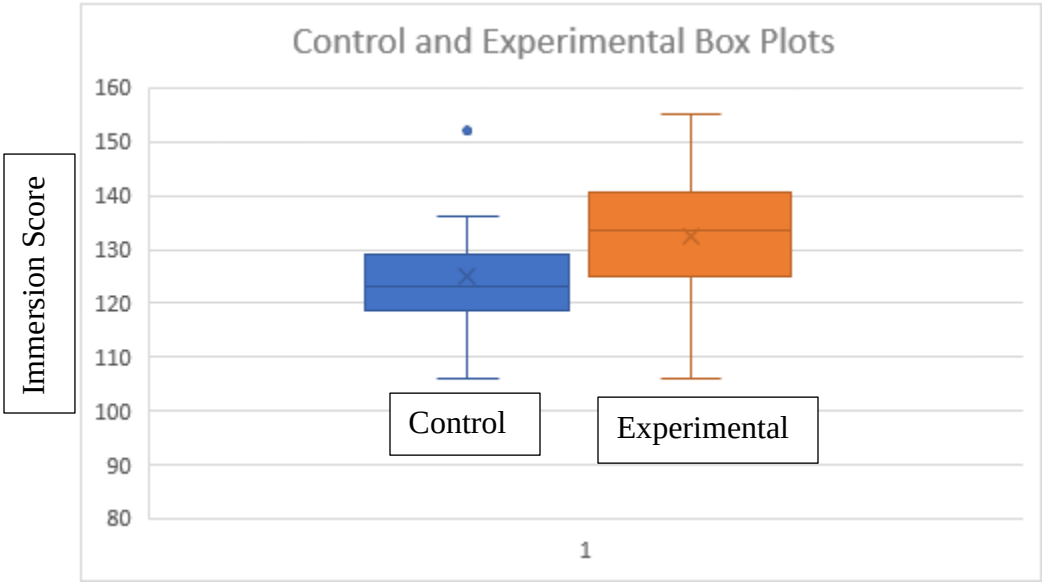


Figure 51: The box plot for the control and experimental games, in blue and orange respectively

For the control game, one value, 152, was an outlier and therefore excluded when calculating the mean. Outliers were determined if they are greater than the third quartile, so for the control game 127 + (1.5 x interquartile range), with the IQR being 9.5, and 152 greater than 141.25.

The outlier participant's comments surrounding the control game was that they had found the game challenging due to their lack of rhythm game experience, confirmed by certain results in their questionnaire, such as finding the game particularly intense, not finding the game easy and feeling they performed poorly in the game overall. They also wanted to win the game and felt in suspense at the results of the game. Lazzaro (2009) discuss how one aspect of play experience can be hard fun, "where the frustration of the attempt is compensated by the feelings of accomplishment and mastery from overcoming obstacles". Games considered hard fun include Dark Souls (FromSoftware, 2011), notorious for its difficulty, but maintains flow by creating "goals and motivation" stemming from the desire not to lose progress (Guzsvinecz, 2023, 5-16). This user could therefore be used to playing games that are "hard fun" and therefore, in spite of finding the game challenging, is never demotivated as they have an intrinsic wish to master the game's difficulty, whereas other players may be more disheartened as they are not used to this gameplay style.

A histogram was created for both games, in Figures 53 and 54 respectively, to determine if the data fits a normal distribution in both cases.

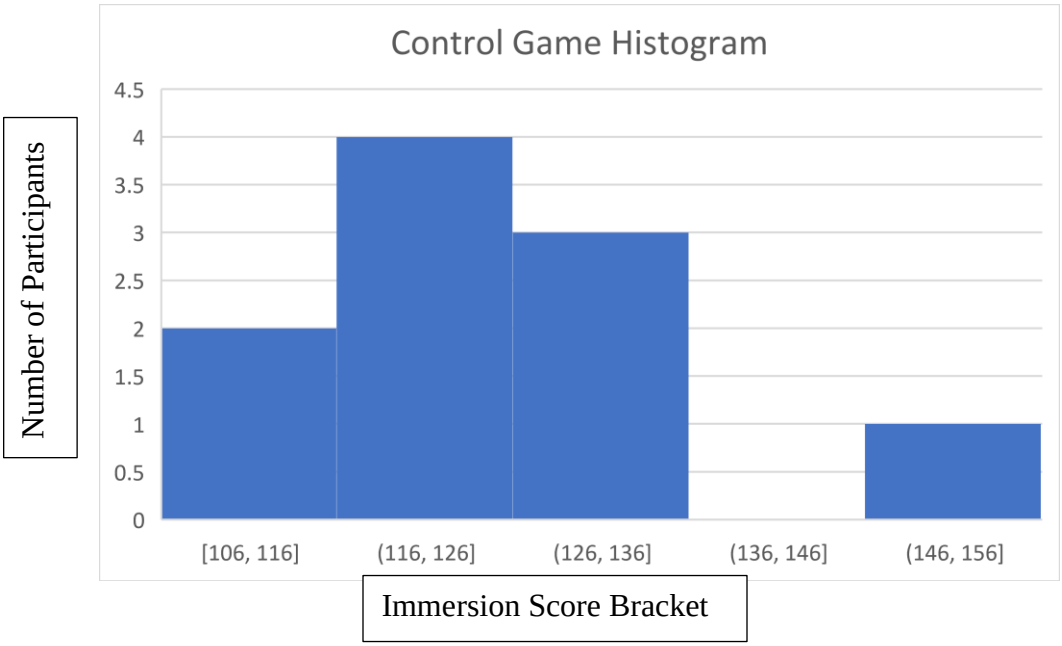


Figure 52: The histogram for the control game

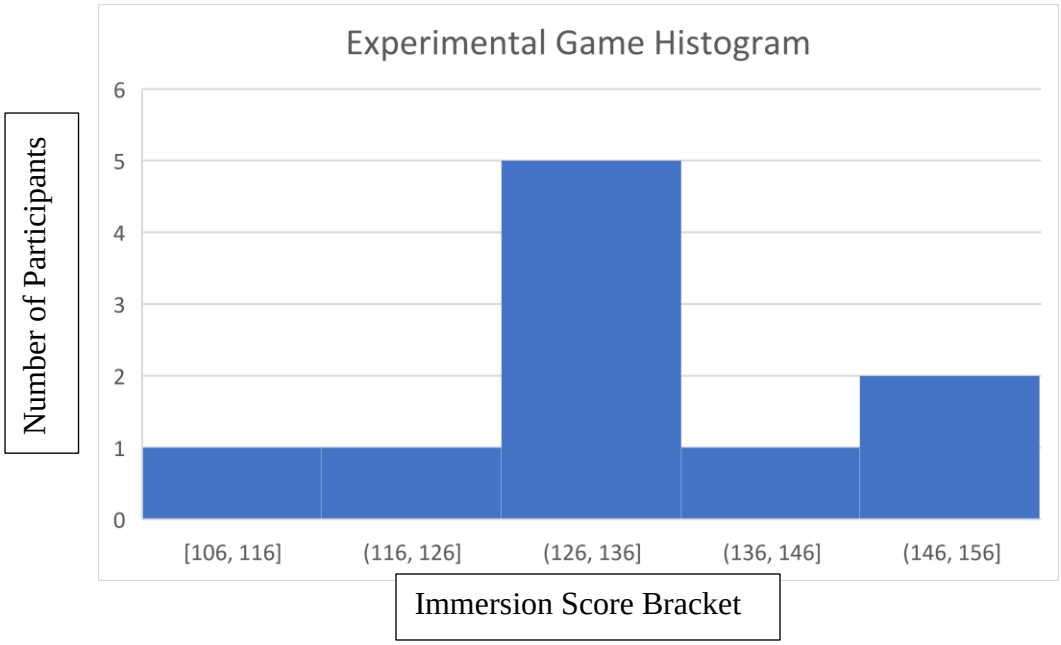


Figure 53: The Experimental Game Histogram

Thompson et al. (2012, 3-4) warned that their study did not produce a normal distribution. To verify a normal distribution for these results, at least 95% of the values must lie plus/minus two standard deviations away from the mean. This was important as, depending on if the data is parametric or non-parametric, the

type of significance test used is affected. For the control game, that range is 105.12 and 138.68, which fits all values bar the outlier. For the experimental game, this lies between 103.9 and 161.1. Furthermore, both histograms were not skewed and had not heavily deviated from a bell shape (Sainani, 2012, 1) further reinforcing this was normal data. Therefore, the results have a normal distribution and significance testing can occur using a right tailed independent samples t test, as discussed in 4.5, to determine if there is a significance between the two groups. When calculating this p value, the null hypothesis which "assumes no difference between the two means…the recorded difference is not significant" (Mindrila and Balentyne, 2013, 4) is rejected only if p is less than 0.05, as this p value represents the tail end of the normal distribution (Mindrila and Balentyne, 2013, 7), the result is therefore unlikely due to chance as these values do not represent 95% of the populace.

The way to calculate the t value differs depending whether the variances of the two populations (standard deviation squared) are equal. An F-ratio test is used (Snedecor and Cochran, 1983) to check this, which has a null hypothesis that both variances are the same, but is rejected if $F = \sigma 1 / \sigma 2$, where $\sigma 1/\sigma 2$ is the ratio of the respective variances, is greater than the F-ratio table value from the degrees of freedom of the experimental and control games, which are 10-1 and 9-1 respectively. Using this formula, F = 204.49/70.3921 and therefore F = 2.91 (3sf), with the F-Table value being 3.39 at row 9, column 8 in Appendix K (University of Sussex, 2005). Therefore, the calculated F value is not greater than the F-Table value at 0.05 significance so there is insufficient evidence to reject the null hypothesis. The t test formula used is shown in Figure 55.

$$t = \frac{\text{difference of group averages}}{\text{standard error of difference}}$$

Figure 54: The two sample t test formula (JMP Statistical Discovery, undated)

The standard error of difference is calculated by the pooled standard deviation, shown in Figure 56 squared, which is then square rooted.

$$s_p^2 = \frac{((n_1-1)s_1^2)+((n_2-1)s_2^2)}{n_1+n_2-2}$$

Figure 55: The pooled standard deviation squared (JMP Statistical Discovery, undated)

Where n1 and n2 are the number of participants in the experimental and control group respectively, and s1 and s2 consequently being the standard deviations from their respective experiments. The pooled standard deviation was calculated as 11.9 (3sf). Standard error of difference required this value, multiplied by the square root of 1/n1 + 1/n2 (JMP Statistical Discovery, undated), leading to a final value of 5.47 (3sf). Difference of group averages was calculated as 132.5 – 121.9, equalling 10.6. Therefore t = 10.6/5.47 and overall t = 1.94 (3sf).

The t value itself needed to be determined using a t-table at 0.05 significance level and a degrees of freedom value equal to (n1 + n2 -2) (JMP Statistical Discovery, undated) which was 17. Utilising a website that calculates the t value at 0.05 significance (ttable.org, undated), the t value was 1.7396. As the calculated t value is greater than this value, the null hypothesis can be rejected, meaning that there is significant difference between the two means calculated and there is evidence that immersion does increase using more musical features. It is likely this result occurred as, the larger the sample size, the greater the t value. As t grows there is greater likelihood that the null hypothesis is rejected, as the growth of t causes the p value to “converges to zero” (Rouder et al., 2009, 3). Furthermore, “increasing the sample size will, on average, result in a gain of

evidence against the null" (Rouder et al., 2009, 2) meaning that if more participants were included in the study, a larger t value occurs to reject null.

6.3 - Discussion of Findings

Whilst the use of significance testing does support the notion that immersion increases in the experimental game, Rouder et al. (2009, 2) sounds a word of caution: "significance tests tend to overstate the evidence" against the null hypothesis. This could therefore imply that this significance test does not truly support the research question. Other potential concerns are outlined below:

6.3.1 - The use of the slider

One common comment in both games was regarding the slider. This related to how its incorporation in the bottom right screen was distracting. For instance, E2 (Experimental 2) noted that the slider in the corner meant they had to switch their focus from the character to the UI whenever going off tempo, and also felt it was not immediately clear to understand, despite the use of green and red colours. E4 felt a similar way and suggested a better way of communicating when on beat, such as "splitting into 4 to better differentiate beats of each song". E3, E10 and C4 also stated that they felt that even though the slider was indicating they were off beat, they felt the opposite, although this could mean that the margin of error needed re-calibration. C6 commented that even with the slider, they found the game difficult, stating that in existing rhythm games, the visual cue was much more apparent than in this game. In Crypt of the NecroDancer (shown in Figure 57), the visual cue is displayed in the middle of the screen, with "bars" showing the player the time until the next beat. C1 stated this gives better feedback to the player as it can determine if they were too early or late on each beat. This might have caused some negative effects on immersion for each participant due to diminishing effects on flow:

Sweetser and Wyeth (2005, 6) mention that “players should receive immediate feedback on their actions” and the fact players were not informed how early/late they were meant they lacked crucial information to help them “progress toward their goals”. However, when looking at questions 18 and 22 which evaluated if the player felt like giving up (anxiety state of flow) or if they were performing well (in flow), the overall results in both experimental and control were 4.1 and 3.6 (using inverted values) for Q18 and 3.4 and 2.6 for Q22. This implies that, even if the slider was not fully effective at communicating information, players felt mostly in flow in the experimental game. This was surprising to the researcher considering that there were more dangerous enemies involved in the experimental game and distraction from the slider, could cause anxiety as players had less control on moving their character away from these enemies (Sweetser and Wyeth, 2005, 5). C2 mentioned that while feedback was not always clear, they felt the core loop itself was engaging enough to overlook this and perhaps this view resonated more in the experimental group.

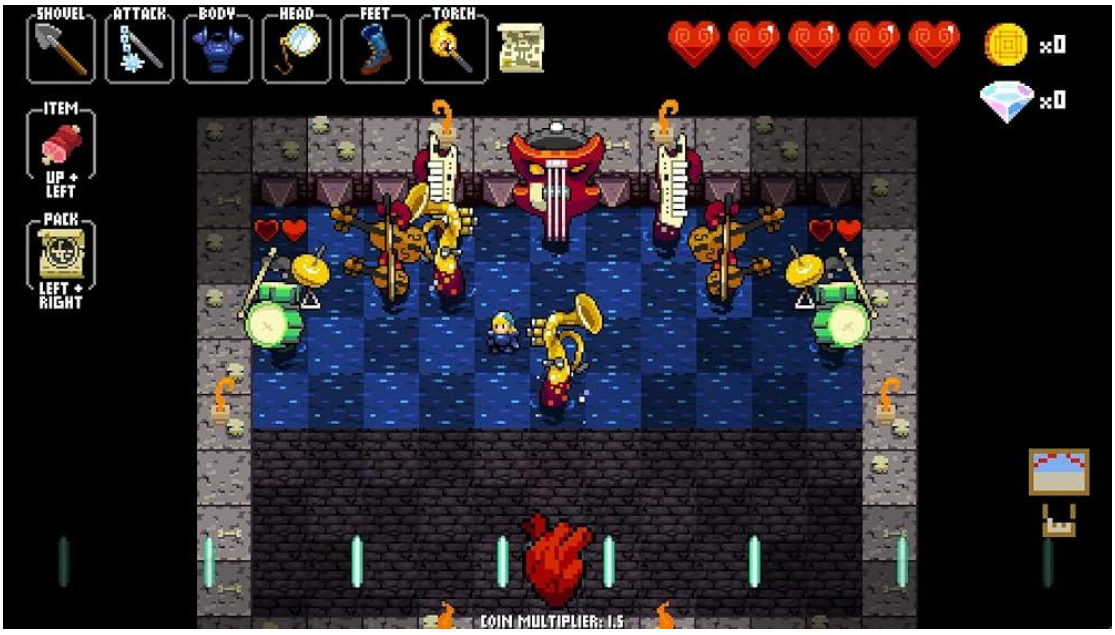


Figure 56: The visual cue in Crypt of the NecroDancer (Brace Yourself Games, 2015)

To improve the slider system, consideration should be given to where the player looks during play. Crypt centres the visual cue in the bottom centre of the screen, rather than the bottom right. Caroux et al. (2011, 1) mention that for game information i.e. score or, in this game, the slider, the UI information should be located, but not overlapping with, the direction where the player is anticipated to move. Furthermore, it is recommended that designers place information sources closely together if they contribute to the same goal (Caroux et al., 2011, 13). Since the slider contributes to the player goal by informing them when to move, the slider should be closer to the player, addressing the complaints regarding how players needed to shift their attention between the game and the visual cue. This can be achieved by displaying the slider directly above, and moving with, the player meaning the player focuses purely on the game area rather than breaking up their play to observe the slider.

E2 further mentioned that they found the overall use of the UI was too distracting, with information such as player health, weapon durability again drawing attention

away from the game. Caroux et al. (2015, 5) mention that players had better flow when “less information is displayed on…screen”. Therefore, the UI information could instead be translated to better polish metrics as described by Swink (2008, 160) such as sound effects, where an impact sound could be played as a weapon breaks, to provide a better perception of object interaction. This could also address other issues players highlighted, E8 stated that they focused so much on being on beat that they did not notice their weapon broke, whereas a sound effect would make this immediately clear.

6.3.2 - Player Performance

Appendices L-P showcase player performance for each song in the control/experimental stage, indicating that the performance results vary wildly, regardless of the control or experimental game. However, it can be determined that people generally performed the best in Song 3 and the worst in Song 1. This can probably be attributed to a “sudden start” as E10 describes with E10 further suggesting the creation of a tutorial for new players. As Sweetser and Wyeth (2005, 7) outline, it is important to teach players how to play the game before starting. Therefore the “sudden start” may have adversely affected enjoyment. As Song 3 had the slowest tempo of all the songs, it may have been easier for new players to grasp the game mechanics, as illustrated by Crypt starting at lower tempos to introduce its gameplay. Song 3 also had a larger margin of error than all the other songs, compensating for its slower tempo. This could also have made the song easier to navigate.

Some players stated that they felt they performed better in certain songs. E6 found Twilight easier, having heard the song before, a view shared by C7, who stated that they enjoyed playing Twilight’s level as they enjoyed listening to the song. Whilst songs were picked with the aim of ensuring that no bias was derived from listening to self-selected music, with Fierro (2012) noting that this creates higher immersion

levels, it is impossible to control if a song is liked by the participant, or been heard before. This could have positively influenced the high immersion score for E6 at 152. Another comment relating to the scoring system is that when players were able to attack enemies their score was doubled. E3 tended to get very high scores and mentioned that this exponential score made them feel good and felt more in flow due to being rewarded for good performance (Sweetser and Wyeth, 2005, 5). This is further reinforced by the experimental players feeling that, on average, they performed better in the game and could have contributed further to immersion.

Finally, certain players commented that they felt some songs were unsuitable for rhythm games. E4, for instance believed that electronic or dance music was best suited to the genre whilst E5 felt that Song 4 was not a good fit for the gameplay. E8, however, mentioned that songs 3 and 5 were the most engaging due to the music, E4 also performed the best in these songs. This suggests that the genre of music used in the rhythm game may impact player performance and therefore, immersion. Table 6 demonstrates the mean missed beats in the control and experimental games.

Song	Control Mean	Experimental Mean
Reckoner (Song 1)	36	40
Twilight (Song 2)	30.6	32.5
Little Dark Age (Song 3)	18.6	19
Shimmy (Song 4)	37	38.6
She's Lost Control (Song 5)	29	31.5

Table 6: The mean number of missed beats

Overall, Song 3 evokes the best performance, followed by songs 5, 2, 4 then 1, with this trend occurring in both games. This may mean that Song 3 is the best suited to the rhythm genre due to players consistently achieving the best performance. Alternatively, it could also be that different music genres may suit different levels of challenge, as discussed by Aslinger (2009, 1) genre diversity can "satisfy inexperienced, intermediate and advanced players" due to genres having "varying beats per minute". Perhaps, Song 3 recorded the lowest number of errors as it was best suited for more inexperienced players, allowing them to ease themselves into the flow state. There is the potential that, for certain songs, players cannot easily determine where the tempo lies. Large and Palmer (2002, 5) mention that is can be difficult to determine "which tones or features belong to the same … part" which could make it difficult for a player to focus on tempo during play, as this feature could be hard to determine across different parts of the music. More research should be undertaken to determine the effect of music genre on player performance in rhythm games to establish whether there is a definitive trend.

6.3.3 - The use of key and volume

This section examines key and volume's effect at enhancing immersion in closer detail, to explain the results of the significance test. One question devised for this study was "Did you enjoy the visuals of the game?" this establishing if the use of key to colour was more visually interesting to the player. Another question was "Did you feel emotionally different when playing the game?" establishing if the player derived an emotional response from the colours of the game. Finally, the player is asked "Did the game feel intense?" to determine if the loudness of the game caused any effect on game intensity. A summary of the mean scores for these three questions: Q36, Q32 and Q35 respectively are shown in Table 7.

Question	Control	Experimental
Q36	3.6	3.9
Q32	2.9	2.9
Q35	3.4	3.3

Table 7: The means for Q36, Q32 and Q35

The following can be determined: whilst overall players preferred the visuals, this did not translate to a change in emotional response. Furthermore, the intensity of the experience was marginally better in the control game, despite more challenging enemies being spawned in the experimental levels. Q17 which refers to how challenging the game was to play, was analysed resulting in values of 4.3 and 4.5 respectively for the control and experimental game. The fact that intensity is more prevalent in the control game, but reversed when analysing how challenging the game was, is contradictory. The literature review discussed that intensity results from challenge i,e when more difficult enemies spawn. Furthermore, E3 mentions how these stronger enemies generated more

challenge, as projectiles were harder to dodge in certain songs. As the spawning rate of the enemies is random, it is not always guaranteed that the stronger enemies will spawn, therefore a player may have been faced with 'easier' enemies when playing in the experimental game, potentially reducing the intensity. Sweetser and Wyeth (2005, 7) mention that challenge should increase as the player becomes close to mastery, so it is probable that a higher spawn rate of these tougher enemies as the player starts performing well facilitates the desired intensity. E6 and E7 both mentioned the random enemy spawning, enquiring about its functionality and, on learning that the loudness value was used, stated that this was not obvious, similar to how Soundfall's use of loudness was not believed to be perceptible to the player.

Regarding visuals and key, there is evidence that the visuals in the experimental game were preferred over those in the control game. This is further reinforced by Q34 "Did you notice any differences in gameplay?" as the control game scored 2.9 on average compared to 3.2. Since volume was not well perceived, it is believed that the only difference in gameplay mentioned in Q34 must come from key, as colour grading can be more noticeable to players as illustrated in other studies such as Misztal et al. (2020, 4) who used colour grading to better convey a sense of stress. Indeed, colour grading is known to have greater immersive effects and this was reflected in overall immersion in the experimental game. However, it is not clear if the visuals themselves link to emotional response, as originally envisioned in the literature review. E7, for instance, stated that they enjoyed the visuals as they reminded them of "old school games they played as a kid". This means, however, that the visuals were enjoyed more due to the assets rather than the colour grading. E4 and E6 did notice that the colours changed, responding positively to this. However, they were not aware that this resulted from key. Therefore, the emotional level may not have been derived from the key, but rather from other factors with Ravaja et al. (2006, 7) mentioning that positive emotions could be elicited from rewarding events, i.e.

E3 mentioning how the score doubling for killing enemies made them ‘feel good’. It could be that colour and emotion should be viewed separately in this study. Furthermore, it is not believed the emotional response is linked to player perception of the intensity of the experience: Cabanac (2002, 5) classifies that an intense experience “may be related to sensation…without being emotional” with Q35 referring to Chanel et al. (2008)’s findings, indicating if intensity and arousal were more prevalent in the experimental game.

It is clear that the use of colour does lead to a greater immersive effect in rhythm games. Players generally liked the change in colours even if this did not necessarily lead to an emotional shift with Kauranen (2023, 31) discussing the use of colour grading within a narrative frame, such as the use of green being used to make the game feel more ominous. This stems from the film industry, as colour grading is used to “support the mood of the story” (Higgins, 2003, 12). However, the game created has no story, relying instead on the player goal of achieving a high score, rather than to progress a narrative. Narrative, when combined with other aspects of the game world, contributes to the immersive experience (Brown and Cairns, 2004, 3) and perhaps the colour grading could have enhanced immersion if it had been combined with a plot. Narrative tends to not be the focus of rhythm games as it is the gameplay itself that feeds the player’s goals for progression (Song et al., 2019, 5). Jennett et al. (2008, 8) state that immersion could instead have been enhanced as there was a “change in visual attention”: the difference in colour through each level. Miller (2013, 3) discusses how Audiosurf, described in detail in 2.5.4, is able to demonstrate the musical concept of tempo in an engaging way, with part of this being via the use of colour. It may therefore be, that the use of colour in rhythm games may not enhance immersion through emotional changes, but instead enhance immersion and engagement through visual interest.

6.4 - Summary of Objectives and Requirements

A brief summary of the objectives and requirements facilitation is described in this section. 6.1 describes how the control game for objective 8 was created, removing features from the experimental game in order to investigate the project aim. This experimental game was created for objective 7, facilitating the following requirements: to parse JSON objects to give tempo data for the beat detection algorithm, key to provide feedback on how musical features affect the game via colour changes and volume to create variety in the enemies that spawn. Further requirements for weapon spawning, durability, enemy movement and player movement/attacks and enemy spawning are discussed in detail in Chapter 5, which also incorporated objectives 4 and 5 and outlined how the experimental game was refined from playtesting. The non-function requirements to create a design document and game loop are shown in Appendices A and B with the MDA analysis in Appendix C, fulfilling objectives 2 and 3. 5 musically distinct songs were picked for objective 1, as shown in Table 4 and the requirement to create an IEQ is shown in Appendix D. The initial requirement stated this was for the experimental game. However, it became apparent that this questionnaire should apply to both games to determine if the control game performs better/worse than the experimental game regarding aspects that the experimental game should enhance i.e. intensity. Objectives 9 and 10 were completed as twenty participants took part in the experiment, completing the IEQ and having observational data recorded, following the procedure set in 4.5.1.

Chapter 7 - Conclusion

As mentioned in the contributions, an elegant solution has been created for the research question. This aim stems from a lack of research regarding features outside of tempo being utilised in rhythm games. Justifications are outlined in the literature, mainly that key relates to colour and colour relates to emotion, with volume providing more intense experiences, this research being incorporated into the game design document for the experimental game developed for this project. Key changes the colours of the game via colour grading as this was established to be an effective way to evoke emotional responses in games, whilst more difficult enemies are present in louder songs to provide more intense experiences. The aim was to give a novel solution to the problem, as whilst colours may be used in existing rhythm games, as far as can be determined this study is the first to establish directly a link between using key to influence colour and its effects on immersion in rhythm games. Similarly, while Soundfall may use volume, it was argued that this may not be perceivable by players so a new way of implementing volume was required to investigate its immersive effects.

Overall, the results outlined in Chapter 6 do achieve the aim and research question, that additional musical features can facilitate more immersive experiences in rhythm games. Players particularly enjoyed the use of key to change colour in the game, although admittedly this was not because they noticed that key was being used to generate this effect, whilst the use of volume for generating intensity was not readily noticeable. As discussed in 6.3.4, all objectives and requirements have been achieved with the use of playtesting being incorporated into the project in 5.3.5 due to its use in game design. Useful feedback regarding game feel and juiciness was incorporated to generate good flow and engagement. Playtesting helped to inform game aspects that were initially missed, such as the provision of better goals and rewards for positive actions in game. SCRUM as a methodology was also achieved. Whilst each

sprint, using the ABC method, was implemented successfully and allowed for playtesting to add more features for a better game, it could have been deployed more successfully. The games themselves, were generally enjoyed by all participants, however there were some negative comments, particularly relating to the slider, which should be taken into consideration for refinement.

Whilst the evidence from testing does support the research question by rejecting the null hypothesis, there are certain limitations to the study itself. As discussed in 6.3, colour did facilitate greater immersion, but is not clear whether this was due to key facilitating an emotional response, or key facilitating a more visually stimulating environment. There is evidence to support how changes in visuals facilitate increased immersion, meaning that the use of key should not be disregarded for more immersive environments. Future work could be undertaken to establish how to create engaging narratives in rhythm games, using key-colour theory to provoke more emotional response as colour grading is better at eliciting emotions in a narrative context. There is potential interest in seeing if there is a way to blend the existing hypothesis for key, colour, emotion with narrative in rhythm games as existing rhythm games tend to not focus on narrative. The risk analysis mentions players could feel unwell if colours shifted between segments. Whilst this risk was erased by using average key, the question could be asked, was there no emotional difference derived from the use of colour since this did not change to reflect the differences in keys between song segments. This study should be undertaken again using the different key segments with analysis carried out to ascertain if this causes a greater immersive and emotional effect in rhythm games compared to using the average key.

Another limitation to this study relates to the use of loudness in the game. Initially it was believed that the value Spotify returned in the JSON file was the song loudness. However, this may not be the case. In music mixing, there is something called a “loudness war”, defined as an “increase in the loudness of

recorded music, particularly on Compact Discs…apply dynamics compression and limiting in an attempt to make their recording louder than those of their competitors" (Vickers, 2010, 1). This means that more recent recordings may appear 'louder' when compared to recordings from decades ago. The music for this study has been taken from a variety of eras, from 1979 to 2017. It is not fully conclusive; however, which song is the loudest as this is subjective. Some participants in the experimental game commenting that they felt Twilight was the loudest song, even though it was the quietest overall. This could account for why it seemed there was no difference in overall intensity between both games. Whilst the loudest song should generate the highest intensity condition, the player may feel the louder, intense song is more applicable to a different level. This could imply that loudness is not a reliable musical feature to incorporate in rhythm games, at least in relation to more intense experiences. An error in the code for enemy spawning was also discovered after testing. Chapter 5 outlines how any values greater than the average loudness should spawn the more difficult enemies, this being the case for the songs Shimmy and Little Dark Age. The enemy pool range had mistakenly been set to the inverse condition. However, due to the subjective nature of how a player perceives loudness, the study was not adversely affected.

One final limitation is that the test size is quite small and that there could be unforeseen factors influencing the results obtained. Thompson et al. (2012, 4) mentions the possibility that gender could have played a role in how immersed a player was. Alternatively, if players had played a game previously, this could also influence immersion. Whilst these factors were determined as not being significant to Thompson et al's study, these factors could be significant for the control or experimental games. Some players stated that they had no prior experience playing rhythm games and this could have influenced their responses to questions regarding how easy they found the game. It could also explain why the intensity was slightly higher in the control game if more inexperienced

players were included in this group. Therefore, going forward, questions regarding a player's experience in rhythm games should be included to determine whether this causes a significant effect on results. A larger sample size could also be utilised, although as mentioned in 6.2, this does tend to mean the null hypothesis is more likely to be rejected.

Whilst not a limitation of the study, as this issue was prevalent in both games, playcentric design should have been utilised when creating the game, defined by Fullerton (2014, 11) as "involving the player in your design process from conception to completion" and whilst one component, playtesting, was incorporated, this was at the end of the implementation stage. When designing goals or features the designer should get "inside the heads of players, not focusing on the features of the game as you intend to design it" (Fullerton, 2014, 12). Whilst it is true that the designer was able to implement a visual cue during self-playtesting, players were not involved at this stage. Their involvement could, perhaps, have influenced the designer to develop a better visual cue, together with other game features, such as the "sudden start" that players felt. Whilst initially deemed satisfactory, on reflection, the comments from players insinuated that this could have violated the concept of having the challenge of the game matching the skill level for new players (Sweetser and Wyeth, 2005, 5) as players may feel the game has an immediate difficulty spike. Involving players during implementation could have led to the design of a better system for easing players into the game resulting in better flow. Furthermore, when using SCRUM, a general principle is that at each step of the sprint a testable product is created, i.e. an alpha version of the project (Schild et al., 2010, 3) with potential new features/bug fixes/levels being added based on feedback from the testing (Schild et al., 2010, 3, Kristiadi et al., 2019, 5), meaning that playcentric design blends with SCRUM. Therefore, improvements to this project, should be carried out in conjunction with players to improve aspects of the game i.e. the slider.

This dissertation finishes by identifying areas for future work incorporating results from Chapter 6. Firstly, there is a belief that key, colour and emotion are linked and thus can lead to more immersive experiences. However, it is also believed that this is not as easily determinable in a rhythm game, with colour grading most effective in narrative contexts, not common in this genre. This highlights two avenues for research. Firstly, is it possible to have a compelling and engaging narrative in a rhythm game? Since it is discussed how colour grading affects emotion during narrative, the key-colour theory could be incorporated when pursuing this research question. Secondly, how powerful is the colour grading effect in game narratives? It was discovered that players found certain genres of music not a good fit for the game, reflected by some songs, namely Reckoner and Shimmy, achieving very poor results. Therefore, a further research question could be: Does the use of certain musical genres effect gameplay performance in rhythm games? A final question relates to loudness. In this project and the game Soundfall, loudness has been utilised in an attempt to affect game difficulty. However, it is currently unclear if this feature adequately affected player perceptions of both the experimental game and Soundfall. Further research should be conducted into the effect of volume to determine if there is a way to perceive this, incorporating different mechanics to measure intensity. Furthermore, this study could be repeated, factoring in variables such as gender or rhythm game experience to ascertain if these variables cause a significant effect on results. If this is not the case, then it further supports the results found in this study. Finally, the study could also be recreated to examine the effect of using the different keys in each song compared to the average key, to determine if this creates any emotional and immersive difference.

References

Agarwal, R. and Karahanna, E., 2000. Time flies when you're having fun: Cognitive absorption and beliefs about information technology usage. MIS quarterly, pp.665-694. [accessed 24 January 2024].

Almeida, M.S.O. and Da Silva, F.S.C., 2013. A systematic review of game design methods and tools. In Entertainment Computing–ICEC 2013: 12th International Conference, ICEC 2013, São Paulo, Brazil, October 16-18, 2013. Proceedings 12 (pp. 17-29). Springer Berlin Heidelberg. [accessed 24 March 2024].

Andrade, A., 2015. Game engines: a survey. *EAI Endorsed Transactions on Serious Games*, *2*(6). [accessed 18 October 2023].

Ašeriškis, D. and Damaševičius, R., 2017. Computational evaluation of effects of motivation reinforcement on player retention. Journal of Universal Computer Science, 23(5), pp.432-453. [accessed 30 April 2024].

Aslinger, B., 2009. Genre in Genre: The Role of Music in Music Games. In DiGRA Conference. [accessed 27 April 2024].

Atanasov, S., 2013. Juiciness: Exploring and designing around experience of feedback in video games. [accessed 27 March 2024].

Aversa, D. and Dickinson, C., 2019. Unity Game Optimization: Enhance and extend the performance of all aspects of your Unity games. Packt Publishing Ltd. [accessed 25 March 2024].

Barbiere, J.M., Vidal, A. and Zellner, D.A., 2007. The color of music: Corresponence through emotion. Empirical studies of the arts, 25(2), pp.193-208. [accessed 23 October 2023].

Bégel, V., Di Loreto, I., Seilles, A. and Dalla Bella, S., 2017. Music games: potential application and considerations for rhythmic training. Frontiers in human neuroscience, 11, p.273. [accessed 26 March 2024].

Bench, S.W. and Lench, H.C., 2013. On the function of boredom. Behavioral sciences, 3(3), pp.459-472. [accessed 30 April 2024].

Biehl, M., 2016. RESTful Api Design (Vol. 3). API-University Press. [accessed 1 April 2024].

Blesser, B., 2007. The seductive (yet destructive) appeal of loud music. Retrieved August, 1, p.2012. [accessed 24 January 2024].

Bourhis, P., Reutter, J.L., Suárez, F. and Vrgoč, D., 2017, May. JSON: data model, query languages and schema specification. In Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems (pp. 123-135). [accessed 26 March 2024].

Bowman, D.A. and McMahan, R.P., 2007. Virtual reality: how much immersion is enough?. Computer, 40(7), pp.36-43. [accessed 30 April 2024].

Brace Yourself Games (2015) Crypt of the NecroDancer [game]. Vancouver: Klei Entertainment. Available from https://store.steampowered.com/app/247080/Crypt_of_the_NecroDancer/ [accessed 18 October 2023].

Brockmyer, J.H., Fox, C.M., Curtiss, K.A., McBroom, E., Burkhart, K.M. and Pidruzny, J.N., 2009. The development of the Game Engagement Questionnaire: A measure of engagement in video game-playing. Journal of experimental social psychology, 45(4), pp.624-634. [accessed 6th February 2024].

Brown, E. and Cairns, P., 2004, April. A grounded investigation of game immersion. In CHI'04 extended abstracts on Human factors in computing systems (pp. 1297-1300). [accessed 24 January 2024].

Bueno, J., 2017. Development of Unity 3D Module For REST API Integration: Unity 3D and REST API Technology. [accessed 28 March 2024].

Byrne, J. (2023) ‘Can the use of more musical features in live gameplay lead to more immersive rhythm games?’ CMP9056M: MComp Research Project. University of Lincoln. Unpublished assignment. [accessed 28 April 2024].

Byrne, J. (2024) ‘Project Interim Report: Can the use of more musical features in live gameplay lead to more immersive rhythm games?’ CMP9056M: MComp Research Project. University of Lincoln. Unpublished assignment. [accessed 28 April 2024].

Cabanac, M., 2002. What is emotion?. Behavioural processes, 60(2), pp.69-83. [accessed 28 April 2024].

Caroux, L., Isbister, K., Le Bigot, L. and Vibert, N., 2015. Player–video game interaction: A systematic review of current concepts. Computers in human behavior, 48, pp.366-381. [accessed 27 April 2024].

Caroux, L., Le Bigot, L. and Vibert, N., 2011. Maximizing players' anticipation by applying the proximity-compatibility principle to the design of video games. Human Factors, 53(2), pp.103-117. [accessed 27 April 2024].

Cassidy, G. and MacDonald, R., 2009. The effects of music choice on task performance: A study of the impact of self-selected and experimenter-selected music on driving game performance and experience. Musicae Scientiae, 13(2), pp.357-386. [accessed 22 October 2023].

Chanel, G., Rebetez, C., Bétrancourt, M. and Pun, T., 2008, October. Boredom, engagement and anxiety as indicators for adaptation to difficulty in games. In Proceedings of the 12th international conference on Entertainment and media in the ubiquitous era (pp. 13-17). [accessed 30 January 2024].

Chanel, G., Rebetez, C., Bétrancourt, M. and Pun, T., 2011. Emotion assessment from physiological signals for adaptation of game difficulty. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 41(6), pp.1052-1063. [accessed 30 January 2024].

Charbonneau, E., Miller, A., Wingrave, C. and LaViola Jr, J.J., 2009, August. Understanding visual interfaces for the next generation of dance-based rhythm video games. In Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games (pp. 119-126). [accessed 26 March 2024].

Chen, C.H. and Lo, C.S., 2016, November. The development of a music rhythm game with a higher level of playability. In 2016 International Conference on Advanced Materials for Science and Engineering (ICAMSE) (pp. 132-135). IEEE. [accessed 27 March 2024].

Cheng, K. and Cairns, P.A., 2005, April. Behaviour, realism and immersion in games. In CHI'05 extended abstracts on Human factors in computing systems (pp. 1272-1275). [accessed 15 April 2024].

Christopoulou, E. and Xinogalos, S., 2017. Overview and comparative analysis of game engines for desktop and mobile devices. [accessed 18 October 2024].

Chocolatey Software (2011) Chocolatey [software]. Available from https://chocolatey.org/ [accessed 10 February 2024].

Ciuha, P., Klemenc, B. and Solina, F., 2010, October. Visualization of concurrent tones in music with colours. In Proceedings of the 18th ACM international conference on Multimedia (pp. 1677-1680). [accessed 23 October 2023].

Cooper, N. (2023) Hitting the Right Notes: Tying Gameplay to Music in Soundfall. In: GDC 2023, 20-24 March, San Francisco. Available from https://gdcvault.com/play/1029186/Independent-Games-Summit-Hitting-the [accessed 29 October 2023].

Costello, B.M., 2018. The rhythm of game interactions: player experience and rhythm in minecraft and don't starve. Games and Culture, 13(8), pp.807-824. [accessed 26 March 2024].

Cowley, B., Charles, D., Black, M. and Hickey, R., 2008. Toward an understanding of flow in video games. Computers in Entertainment (CIE), 6(2), pp.1-27. [accessed 25 March 2024].

Denisova, A. and Cook, E., 2019, October. Power-ups in digital games: The rewarding effect of phantom game elements on player experience. In Proceedings of the Annual Symposium on computer-human interaction in Play (pp. 161-168). [accessed 27 March 2024].

Denisova, A., Nordin, A.I. and Cairns, P., 2016, October. The convergence of player experience questionnaires. In Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play (pp. 33-37). [accessed 6th February 2024].

Dickinson, C., 2017. Unity 2017 Game Optimization: Optimize All Aspects of Unity Performance. Packt Publishing Ltd. [accessed 25 March 2024].

Dinh, D. and Wang, Z., 2020. Modern front-end web development: how libraries and frameworks transform everything. [accessed 1 April 2024].

Dormans, J., 2012. Engineering emergence: applied theory for game design. [accessed 24 March 2024].

Drastic Games (2022) Soundfall [game]. Raleigh: Noodlecake. Available from https://store.steampowered.com/app/1608700/Soundfall/ [accessed 29 October 2023].

Dylan Fitterer (2008) AudioSurf [game]. Dylan Fitterer. Available from https://store.steampowered.com/app/12900/AudioSurf/ [accessed 18 October 2023].

Edworthy, J. and Waring, H., 2006. The effects of music tempo and loudness level on treadmill exercise. Ergonomics, 49(15), pp.1597-1610. [accessed 24 January 2024].

El-Nasr, M.S. and Yan, S., 2006, June. Visual attention in 3D video games. In Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology (pp. 22-es). [accessed 26 March 2024].

Electric Light Orchestra (1981) Twilight [download]. 3 mins. 42 secs. Time. Munich: Jet. [accessed 10 March 2024].

Ermi, L. and Mäyrä, F., 2005, June. Fundamental components of the gameplay experience: Analysing immersion. In *DiGRA Conference* (pp. 7-8). [accessed 4 May 2024].

Ffmpeg Team (2000) ffmpeg. [software]. Available from: https://ffmpeg.org/ [accessed 10 February 2024].

Fierro, A., 2012. Musical rhythm: How musical rhythm in a serious game can increase the immersion and how the immersion can encourage the rehabilitation process. [accessed 22 October 2023].

Floraphonic (undated) Punch 6 [download]. 1 secs. Available from https://pixabay.com/sound-effects/punch-6-166699/ [accessed 7 May 2024].

Fonteles, J.H., Rodrigues, M.A.F. and Basso, V.E.D., 2013. Creating and evaluating a particle system for music visualization. Journal of Visual Languages & Computing, 24(6), pp.472-482. [accessed 23 October 2023].

FromSoftware (2011) Dark Souls [game]. Tokyo: Namco Bandai Games [accessed 26 April 2024].

FromSoftware (2012) Steel Battalion: Heavy Armor [game]. Tokyo: Capcom [accessed 7 May 2024].

Fuchs, R. and Maxwell, O., 2016. The effects of mp3 compression on acoustic measurements of fundamental frequency and pitch range. In Speech prosody (Vol. 2016, pp. 523-527). [accessed 1 April 2024].

Fullerton, T., 2014. Game design workshop: a playcentric approach to creating innovative games. CRC press. [accessed 27 March 2024].

Gil, S. and Le Bigot, L., 2016. Colour and emotion: children also associate red with negative valence. Developmental science, 19(6), pp.1087-1094. [accessed 26 March 2024].

Glassy, L., 2006. Using version control to observe student software development processes. Journal of Computing Sciences in Colleges, 21(3), pp.99-106. [accessed 1 April 2024].

Goel, A., 2021. Best Programming Language to Learn in 2020 (for Job & Future). hackr. io, 11(05). [accessed 1 April 2024].

Goldmetal (2023) Undead Survivor Assets Pack [asset]. Unity Asset Store. Available from https://assetstore.unity.com/packages/2d/undead-survivor-assets-pack-238068 [accessed 8 December 2023].

Grimshaw, M., Lindley, C. and Nacke, L., 2008. Sound and immersion in the first-person shooter: Mixed measurement of the player's sonic experience. In Audio Mostly-a conference on interaction with sound. www. audiomostly. Com. [accessed 24 January 2024].

Guardiola, E., 2016, November. The gameplay loop: a player activity model for game design and analysis. In *Proceedings of the 13th International Conference on Advances in Computer Entertainment Technology* (pp. 1-7). [accessed 6 May 2024].

Guzsvinecz, T., 2023. The correlation between positive reviews, playtime, design and game mechanics in souls-like role-playing video games. Multimedia Tools and Applications, 82(3), pp.4641-4670. [accessed 26 April 2024].

Hastings, E.J., Guha, R.K. and Stanley, K.O., 2008. Interactive evolution of particle systems for computer graphics and animation. IEEE Transactions on Evolutionary Computation, 13(2), pp.418-432. [accessed 26 March 2024].

Hein, E., 2014. Music games in education. Learning, Education and Games, p.93. [accessed 29 October 2023].

Hicks, K., Gerling, K., Dickinson, P. and Vanden Abeele, V., 2019, October. Juicy game design: Understanding the impact of visual embellishments on player experience. In Proceedings of the annual symposium on computer-human interaction in play (pp. 185-197). [accessed 27 March 2024].

Higgins, S., 2003. A new colour consciousness: Colour in the digital age. Convergence, 9(4), pp.60-76. [accessed 28 April 2024].

Hock, K.S. and Lingxia, L.I., 2014. Automated processing of massive audio/video content using FFmpeg. Code4Lib Journal, (23). [accessed 1 April 2024].

Holmquist, L.E., Redström, J. and Ljungstrand, P., 1999. Token-based access to digital information. In Handheld and Ubiquitous Computing: First International Symposium, HUC'99 Karlsruhe, Germany, September 27–29, 1999 Proceedings 1 (pp. 234-245). Springer Berlin Heidelberg. [accessed 25 March 2024].

Hultstrand, S. and Olofsson, R., 2015. Git-cli or gui: Which is most widely used and why?. [accessed 1 April 2024].

Hunicke, R., LeBlanc, M. and Zubek, R., 2004, July. MDA: A formal approach to game design and game research. In *Proceedings of the AAAI Workshop on Challenges in Game AI* (Vol. 4, No. 1, p. 1722). [accessed 6 May 2024].

Izgi, E., 2018. Framework for Roguelike Video Games Development. [accessed 25 March 2024].

Jennett, C., Cox, A.L., Cairns, P., Dhoparee, S., Epps, A., Tijs, T. and Walton, A., 2008. Measuring and defining the experience of immersion in games. International journal of human-computer studies, 66(9), pp.641-661. [accessed 6th February 2024].

JJMaslen (2024) Walk With Rhythm [game]. Lincoln: Avaliable from https://github.com/JJMaslen/WalkWithRhythm [accessed 25th January 2024].

JMP Statistical Discovery (undated) The Two-Sample t-Test JMP Statistical Discovery. Available from https://www.jmp.com/en_gb/statistics-knowledge-portal/t-test/two-sample-t-test.html [accessed 26 April 2024].

Johnson, D., 2015. Animated frustration or the ambivalence of player agency. *Games and Culture*, *10*(6), pp.593-612. [accessed 7 May 2024].

Joosten, E., Van Lankveld, G. and Spronck, P., 2010. Colors and emotions in video games. In 11th International Conference on Intelligent Games and Simulation GAME-ON (pp. 61-65). sn. [accessed 23 October 2023].

Joy Division (1979) She's Lost Control [download]. 3 mins. 57 secs. Unknown Pleasures. Stockport: Factory. [accessed 10 March 2024].

Kagan, B., 2020. Slave to the Rhythm: Examining immersive experiences through the interplay of music and gameplay in "Crypt of The Necrodancer. [accessed 19 October 2023].

Kanode, C.M. and Haddad, H.M., 2009, April. Software engineering challenges in game development. In 2009 Sixth International Conference on Information Technology: New Generations (pp. 260-265). IEEE. [accessed 29 March 2024].

Kao, D., 2020. The effects of juiciness in an action RPG. Entertainment Computing, 34, p.100359. [accessed 27 March 2024].

Karlesky, M. and Vander Voord, M., 2008. Agile project management. ESC, 247(267), p.4. [accessed 29 March 2024].

Kauranen, A., 2023. How colours guide the player in video games. [accessed 23 January 2024].

Knauert, M., Jeon, S., Murphy, T.E., Yaggi, H.K., Pisani, M.A. and Redeker, N.S., 2016. Comparing average levels and peak occurrence of overnight sound in the medical intensive care unit on A-weighted and C-weighted decibel scales. Journal of critical care, 36, pp.1-7. [accessed 27 March 2024].

Koleva, B., Tolmie, P., Brundell, P., Benford, S. and Rennick Egglestone, S., 2015, October. From front-end to back-end and everything in-between: work practice in game development. In Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play (pp. 141-150). [accessed 1 April 2024].

Kortmann, R. and Harteveld, C., 2009. Agile game development: lessons learned from software engineering. In Learn to game, game to learn; the 40th conference ISAGA. [accessed 29 March 2024].

Kristiadi, D.P., Sudarto, F., Sugiarto, D., Sambera, R., Warnars, H.L.H.S. and Hashimoto, K., 2019, November. Game development with scrum methodology. In 2019 International Congress on Applied Information Technology (AIT) (pp. 1-6). IEEE. [accessed 29 March 2024].

Large, E.W. and Palmer, C., 2002. Perceiving temporal regularity in music. Cognitive science, 26(1), pp.1-37. [accessed 27 April 2024].

Lazzaro, N., 2009. Why we play: affect and the fun of games. Human-computer interaction: Designing for diverse users and domains, 155, pp.679-700. [accessed 26 April 2024].

Lin, R.M., Ho, H.C. and Chen, K.T., 2011, November. Bot detection in rhythm games: a physiological approach. In Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology (pp. 1-8). [accessed 25 March 2024].

Lindborg, P. and Friberg, A.K., 2015. Colour association with music is mediated by emotion: Evidence from an experiment using a CIE Lab interface and interviews. PloS one, 10(12), p.e0144013. [accessed 23 October 2023].

Lindholm, T., Yellin, F., Bracha, G. and Buckley, A., 2013. *The Java virtual machine specification*. Addison-wesley. [accessed 3 May 2024].

Lindman, J., Horkoff, J., Hammouda, I. and Knauss, E., 2018. Emerging perspectives of application programming interface strategy: A framework to respond to business concerns. IEEE software, 37(2), pp.52-59. [accessed 18 October 2023].

Llanos, S.C. and Jørgensen, K., 2011, September. Do players prefer integrated user interfaces? A qualitative study of game UI design issues. In Proceedings of DiGRA 2011 Conference: Think Design Play (pp. 1-12). [accessed 1 April 2024].

Lu, H.K., 2014, June. Keeping your API keys in a safe. In 2014 IEEE 7th International Conference on Cloud Computing (pp. 962-965). IEEE. [accessed 25 March 2024].

Madden, N., 2020. API security in action. Simon and Schuster. [accessed 25 March 2024].

Mammarella, N., Di Domenico, A., Palumbo, R. and Fairfield, B., 2016. When green is positive and red is negative: Aging and the influence of color on emotional memories. Psychology and aging, 31(8), p.914. [accessed 26 March 2024].

McKenzie, T., Morales-Trujillo, M., Lukosch, S. and Hoermann, S., 2021, May. Is agile not agile enough? A study on how agile is applied and misapplied in the video game development industry. In 2021 IEEE/ACM Joint 15th International Conference on Software and System Processes (ICSSP) and 16th ACM/IEEE International Conference on Global Software Engineering (ICGSE) (pp. 94-105). IEEE. [accessed 1 April 2024].

MGMT (2017) Little Dark Age [download]. 4 mins 59 secs. Little Dark Age. Columbia. [accessed 10 March 2024].

Milentijevic, I., Ciric, V. and Vojinovic, O., 2008. Version control in project-based learning. Computers & Education, 50(4), pp.1331-1338. [accessed 1 April 2024].

Miller, B.J., 2013. Music learning through video games and apps: Guitar Hero, Rock Band, amplitude, frequency, and Rocksmith, and bandfuse, and bit. Trip complete, and audiosurf, and beat hazard, and biophilia. American Music, 31(4), pp.511-514. [accessed 28 April 2024].

Mindrila, D., Balentyne, P. and from a Table, S., 2013. Tests of Significance. The Basic Practice of Statistics, pp.2-12. [accessed 26 April 2024].

Misztal, S., Carbonell, G., Zander, L. and Schild, J., 2020, September. Intensifying Stress Perception Using Visual Effects in VR Games. In Proceedings of the 15th International Conference on the Foundations of Digital Games (pp. 1-4). [accessed 27 April 2024].

Mixkit (undated) Bass guitar single note [download]. 6 secs. Available from https://mixkit.co/free-sound-effects/bass/ [accessed 7 May 2024].

Munday, R., 2007. Music in video games. Music, sound and multimedia: From the live to the virtual, pp.51-67. [accessed 18 October 2023].

NanaOn-Sha (1999) Vib-Ribbon [game]. Tokyo: Sony Computer Entertainment. [accessed 18 October 2023].

Neil, K., 2012. Game design tools: Time to evaluate. Proceedings of 2012 DiGRA Nordic. [accessed 29 March 2024].

Nguyen, T.D., Nguyen, A.T., Phan, H.D. and Nguyen, T.N., 2017, May. Exploring API embedding for API usages and applications. In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE) (pp. 438-449). IEEE. [accessed 1 April 2024].

Nordin, A.I., Denisova, A. and Cairns, P., 2014. Too many questionnaires: measuring player experience whilst playing digital games. [accessed 6th February 2024].

Nusrat, F., Hassan, F., Zhong, H. and Wang, X., 2021, May. How developers optimize virtual reality applications: A study of optimization commits in open source unity projects. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE) (pp. 473-485). IEEE. [accessed 25 March 2024].

Parra, J., Lopes da Silva, F.H., Stroink, H. and Kalitzin, S., 2007. Is colour modulation an independent factor in human visual photosensitivity?. *Brain*, *130*(6), pp.1679-1689. [accessed 3 May 2024].

Phellas, C.N., Bloch, A. and Seale, C., 2011. Structured methods: interviews, questionnaires and observation. Researching society and culture, 3(1), pp.23-32. [accessed 6th February 2024].

Pichlmair, M. and Kayali, F., 2007, September. Levels of Sound: On the Principles of Interactivity in Music Video Games. In DiGRA Conference. [accessed 18 October 2023].

Pixabay (undated) snapping ruler vibration [download]. 8 secs. Available from https://pixabay.com/sound-effects/snapping-ruler-vibration-107426/ [accessed 7 May 2024].

Plans, D. and Morelli, D., 2012. Experience-driven procedural music generation for games. IEEE Transactions on Computational Intelligence and AI in Games, 4(3), pp.192-198. [accessed 30 January 2024].

Poncle (2022) Vampire Survivors [game]. Italy: Poncle Available from https://store.steampowered.com/app/1794680/Vampire_Survivors/ [accessed 25 March 2024].

Povel, D.J., 1984. A theoretical framework for rhythm perception. Psychological research, 45(4), pp.315-337. [accessed 29 March 2024].

Pravossoudovitch, K., Cury, F., Young, S.G. and Elliot, A.J., 2014. Is red the colour of danger? Testing an implicit red–danger association. Ergonomics, 57(4), pp.503-510. [accessed 26 March 2024].

Qin, H., Rau, P.L.P. and Salvendy, G., 2010. Effects of different scenarios of game difficulty on player immersion. Interacting with computers, 22(3), pp.230-239. [accessed 2 April 2024].

Radiohead (2007) Reckoner [download]. 4 mins. 50 secs. In Rainbows. XL. [accessed 10 February 2024].

Ravaja, N., Saari, T., Salminen, M., Laarni, J. and Kallinen, K., 2006. Phasic emotional reactions to video game events: A psychophysiological investigation. Media psychology, 8(4), pp.343-367. [accessed 27 April 2024].

Rouder, J.N., Speckman, P.L., Sun, D., Morey, R.D. and Iverson, G., 2009. Bayesian t tests for accepting and rejecting the null hypothesis. Psychonomic bulletin & review, 16, pp.225-237. [accessed 26 April 2024].

Sainani, K.L., 2012. Dealing with non-normal data. Pm&r, 4(12), pp.1001-1005. [accessed 2 May 2024].

Sampath, H., Merrick, A. and Macvean, A., 2021, May. Accessibility of command line interfaces. In Proceedings of the 2021 CHI conference on human factors in computing systems (pp. 1-10). [accessed 1 April 2024].

Sanders, T. and Cairns, P., 2010. Time perception, immersion and music in videogames. Proceedings of HCI 2010 24, pp.160-167. [accessed 6th February 2024].

Schacher, J.C. and Neukom, M., 2007. Where's the beat? Tools for Dynamic Tempo calculations. In ICMC. [accessed 25 March 2024].

Schild, J., Walter, R. and Masuch, M., 2010, June. ABC-Sprints: adapting Scrum to academic game development courses. In Proceedings of the fifth international conference on the foundations of digital games (pp. 187-194). [accessed 28 April 2024].

Schwaber, K., 1997. Scrum development process. In Business Object Design and Implementation: OOPSLA'95 Workshop Proceedings 16 October 1995, Austin, Texas (pp. 117-134). Springer London. [accessed 29 March 2024].

Schwartz, L., 2006. Fantasy, realism, and the other in recent video games. Space and culture, 9(3), pp.313-325. [accessed 15 April 2024].

Sega AM2 (1999) Shenmue [game]. Tokyo: Sega [accessed 15 April 2024].

Sharma, M., Kacker, S. and Sharma, M., 2016. A brief introduction and review on galvanic skin response. Int J Med Res Prof, 2(6), pp.13-17. [accessed 2 April 2024].

Shelton, J. and Kumar, G.P., 2010. Comparison between auditory and visual simple reaction times. Neuroscience and medicine, 1(01), pp.30-32. [accessed 25 March 2024].

Sloboda, J.A. and Juslin, P.N., 2001. Psychological perspectives on music and emotion. Music and emotion: Theory and research, pp.71-104. [accessed 24 January 2024].

Smucker, M.D., Allan, J. and Carterette, B., 2007, November. A comparison of statistical significance tests for information retrieval evaluation. In Proceedings of the sixteenth ACM conference on Conference on information and knowledge management (pp. 623-632). [accessed 1 April 2024].

Snedecor, G.W. and Cochran, W.G., 1989. Statistical Methods, eight edition. Iowa state University press, Ames, Iowa, 1191(2). [accessed 26 April 2024].

Song, D.H., Kim, K.B. and Lee, J.H., 2019. Analysis and evaluation of mobile rhythm games: game structure and playability. International Journal of Electrical and Computer Engineering, 9(6), p.5263. [accessed 28 April 2024].

Spotify (undated) Access Token. Stockholm: Spotify. Available from https://developer.spotify.com/documentation/web-api/concepts/access-token [accessed 25 March 2024].

Spotify (undated) Apps. Stockholm: Spotify. Available from https://developer.spotify.com/documentation/web-api/concepts/apps [accessed 25 March 2024].

Spotify (undated) Client Credentials Flow. Stockholm: Spotify Available from https://developer.spotify.com/documentation/web-api/tutorials/client-credentials-flow [accessed 25 March 2024].

Spotify (undated) Get Artist's Top Tracks. Stockholm: Spotify. Available from https://developer.spotify.com/documentation/web-api/reference/get-an-artists-top-tracks [accessed 25 March 2024].

Spotify (undated) Get Track's Audio Analysis. Stockholm: Spotify. Available from https://developer.spotify.com/documentation/web-api/reference/get-audio-analysis [accessed 25 March 2024].

Spotify (undated) Search for Item. Stockholm: Spotify. Available from https://developer.spotify.com/documentation/web-api/reference/search [accessed 25 March 2024].

Stencel, K. and Węgrzynowicz, P., 2008, November. Implementation variants of the singleton design pattern. In OTM Confederated International Conferences" On the Move to Meaningful Internet Systems" (pp. 396-406). Berlin, Heidelberg: Springer Berlin Heidelberg. [accessed 25 March 2024].

SURT (2023) Rhythm Sprout: Sick Beats & Bad Sweets. [game]. Available from https://store.steampowered.com/app/1475840/Rhythm_Sprout_Sick_Beats__Bad_Sweets/ [accessed 29 March 2024].

Sweetser, P. and Johnson, D., 2004, September. Player-centered game environments: Assessing player opinions, experiences, and issues. In International

Conference on Entertainment Computing (pp. 321-332). Berlin, Heidelberg: Springer Berlin Heidelberg. [accessed 30 April 2024].

Sweetser, P. and Wyeth, P., 2005. GameFlow: a model for evaluating player enjoyment in games. Computers in Entertainment (CIE), 3(3), pp.3-3. [accessed 19 October 2023].

Swink, S., 2008. Game feel: a game designer's guide to virtual sensation. CRC press. [accessed 26 March 2024].

System of a Down (2001) Shimmy [download]. 1 min. 51 secs. Toxicity. Hollywood: Columbia. [accessed 10 March 2024].

Sziklai, I., Szilvássy, J. and Szilvássy, Z., 2011. Tinnitus control by dopamine agonist pramipexole in presbycusis patients: A randomized, placebo-controlled, double-blind study. The Laryngoscope, 121(4), pp.888-893. [accessed 27 March 2024].

Tanskanen, S., 2018. Player immersion in video game: Designing an immersive game project. [accessed 23 January 2024].

Thompson, M., Nordin, A.I. and Cairns, P., 2012, September. Effect of Touch–Screen Size on Game Immersion. In The 26th BCS Conference on Human Computer Interaction. BCS Learning & Development. [accessed 1 April 2024].

Tsujino, Y., Yamanishi, R. and Yamashita, Y., 2019, August. Characteristics study of dance-charts on rhythm-based video games. In 2019 IEEE Conference on Games (CoG) (pp. 1-4). IEEE. [accessed 22 October 2023].

Ttable.org (undated) T Table Available from https://www.ttable.org/ [accessed 26 April 2024].

Unity (2015) Why when I change the color of text from script, it becomes White? San Francisco: Unity. Available from https://discussions.unity.com/t/why-when-i-change-the-color-of-text-from-script-it-becomes-white/128250 [accessed 5 March 2024].

Unity (undated) Audio Files San Francisco: Unity. Available from https://docs.unity3d.com/Manual/AudioFiles.html [accessed 1 April 2024].

Unity (undated) Bloom. San Francisco: Unity. Available from https://docs.unity3d.com/560/Documentation/Manual/PostProcessing-Bloom.html [accessed 26 March 2024].

Unity (undated) Channel Mixer. San Francisco: Unity. Available from https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@7.1/manual/Post-Processing-Channel-Mixer.html [accessed 26 March 2024].

Unity (undated) Film Grain. San Francisco: Unity. Available from https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@7.1/manual/Post-Processing-Film-Grain.html [accessed 27 March 2024].

Unity (undated) JsonUtility.FromJson. San Francisco: Unity. Available from https://docs.unity3d.com/ScriptReference/JsonUtility.FromJson.html [accessed 26 March 2024].

Unity (undated) ParticleSystem. San Francisco: Unity Available from https://docs.unity3d.com/ScriptReference/ParticleSystem.html [accessed 24 March 2024].

Unity (undated) Python Scripting. San Francisco: Unity. Available from https://docs.unity3d.com/Packages/com.unity.scripting.python@6.0/manual/index.html [accessed 25 March 2024].

Unity (undated) Time.deltaTime. San Francisco: Unity. Available from https://docs.unity3d.com/ScriptReference/Time-deltaTime.html [accessed 25 March 2024].

University of Sussex (2005) Table of critical values for the F distribution (for use with ANOVA) Falmer: University of Sussex. Available from https://users.sussex.ac.uk/~grahamh/RM1web/F-ratio%20table%202005.pdf [accessed 26 April 2024].

Vickers, E., 2010, November. The loudness war: Background, speculation, and recommendations. In Audio Engineering Society Convention 129. Audio Engineering Society. [accessed 28 April 2024].

Welch, D. and Fremaux, G., 2017. Why do people like loud sound? A qualitative study. International journal of environmental research and public health, 14(8), p.908. [accessed 30 January 2024].

Xing, Y., Huang, J. and Lai, Y., 2019, February. Research and analysis of the front-end frameworks and libraries in e-business development. In Proceedings of the 2019 11th International Conference on Computer and Automation Engineering (pp. 68-72). [accessed 1 April 2024].

Zammitto, V., 2005, June. The Expressions of Colours. In DiGRA Conference. [accessed 23 October 2023].

Zhang, C., Perkis, A. and Arndt, S., 2017, May. Spatial immersion versus emotional immersion, which is more immersive?. In 2017 Ninth International Conference on Quality of Multimedia Experience (QoMEX) (pp. 1-6). IEEE. [accessed 30 January 2024].

Zheng, H., 2022. Effects of Music Tempo on Flow in Rhythm-Fighting Game (Doctoral dissertation, Northeastern University). [accessed 19 October 2023].

Zolkifli, N.N., Ngah, A. and Deraman, A., 2018. Version control system: A review. Procedia Computer Science, 135, pp.408-415. [accessed 1 April 2024].

Zong and Shannon (2023) Rhythm Hell. [game]. Available from https://rhythmhell.itch.io/rhythm-hell [accessed 27 March 2024].

Appendix A – Game Design Document

Design Document

The general idea for the game will be to develop a rhythm game that co-ops the dungeon crawler style of gameplay. A similar example of this would be Crypt of the NecroDancer. Like Crypt, there should be different enemies to vary the gameplay by having different attack and movement styles. Unlike Crypt, this game will test the player's ability on how long they can survive in the singular dungeon, which will be based on how long the song plays for. Like Crypt however, the way in which the player moves and attacks enemies will all be done in time with the rhythm of the song.

Using audio features

Looking at Spotify's get track's audio analysis function the following song information will be used in the following ways:
https://developer.spotify.com/documentation/web-api/reference/get-audio-analysis
Duration: This will be how long the level lasts overall as it is the songs length
End_of_fade_in/start_of_fade_out: As these relate to when a song will fade in and out there is a possibility the same effect could be done to the game itself i.e have the game fade in from black, slowly having the light radius of the game increase as the song starts to fade in and do the opposite as the song fades out.
Tempo: This refers to the average tempo of the song, however in order to ensure the players actions are mapped accurately to the rhythm a different algorithm should be employed instead, such as the one in soundfall
Time_signature: This could be used for the frequency of enemy spawns
Key: This will affect the colour grading of the game as different keys correspond to different colours

There are different sections to every song, defined by Spotify as "large variations in rhythm or timbre e.g chorus, verse etc" These contain different values for the values defined above and will be used to vary the game overtime.

Segments are defined by Spotify as having "a roughly consistent sound throughout its duration" and have pitches and timbre which will be explored later.

Enemies

As the game will be based on crypt, there will be similar enemies to Crypt as well. They can however all be killed in one hit. As stated the time signature will be used to spawn enemies, however enemies will not spawn near the player, nor will they spawn near other enemies. Loudness (which represents a more intense song) will be used, with louder songs introducing stronger enemies in the spawn pool.

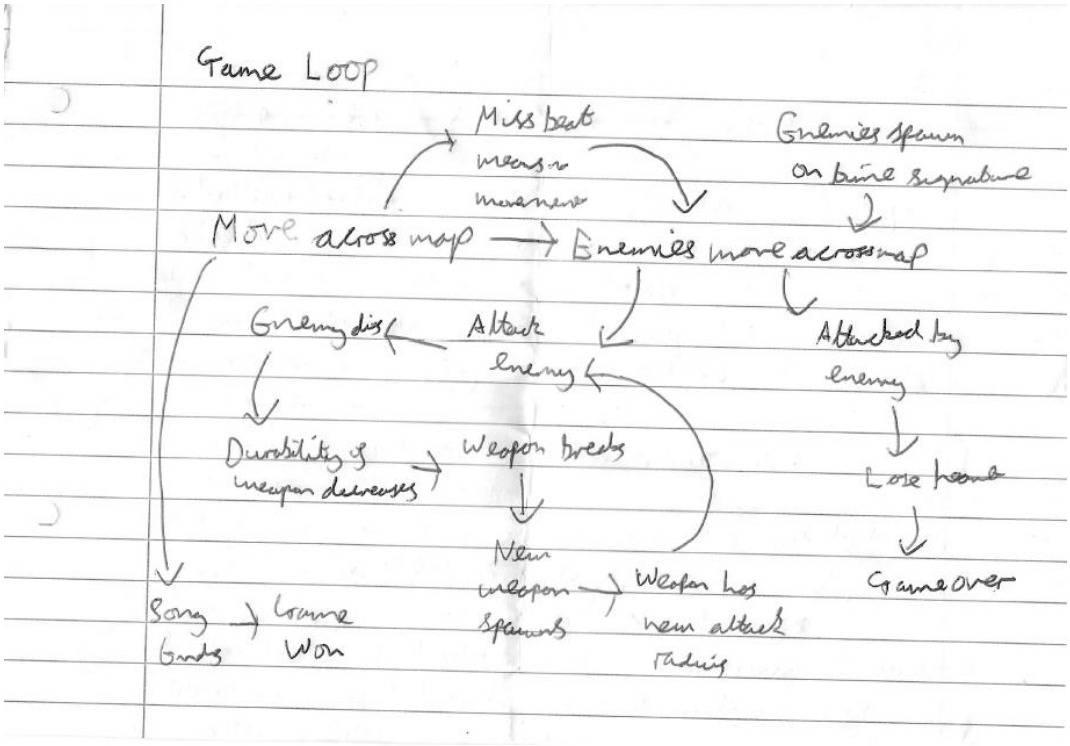
Skeleton: Moves one space every beat in the direction of the player
Bat: Moves to a random space every beat
Golem: Moves one space every beat and does splash damage on movement
Mage: Fires a projectile in horizontal/vertical direction of player
Mimic (Different enemy): Moves in the opposing direction of the player

Weapons

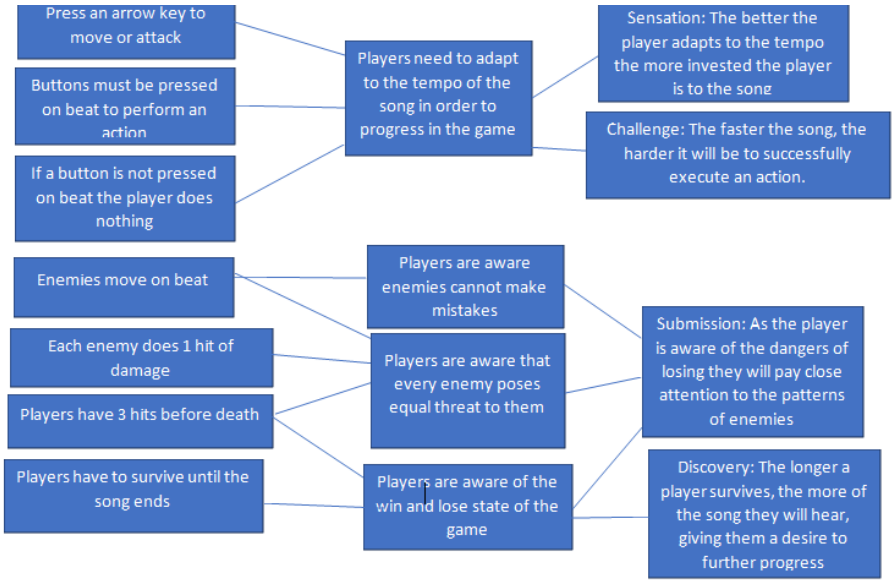
Player starts with a basic dagger however other weapons can spawn, such as a bow which can shoot horizontally or vertically up to 4 tiles, a spear to attack up to two tiles away and a broadsword which can attack within a 3 tile wide radius. (weapon pool affect by instrumentalness)

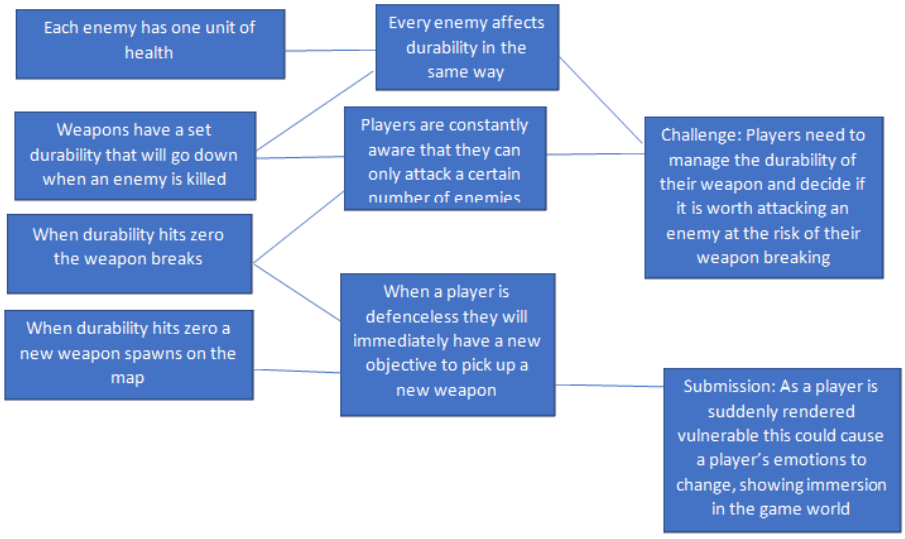
So that the player will continually have new weapons to use, weapons will have durability that will go down after use. Once a weapon breaks a new one will automatically spawn that the player then moves towards

Appendix B – Core Game Loop

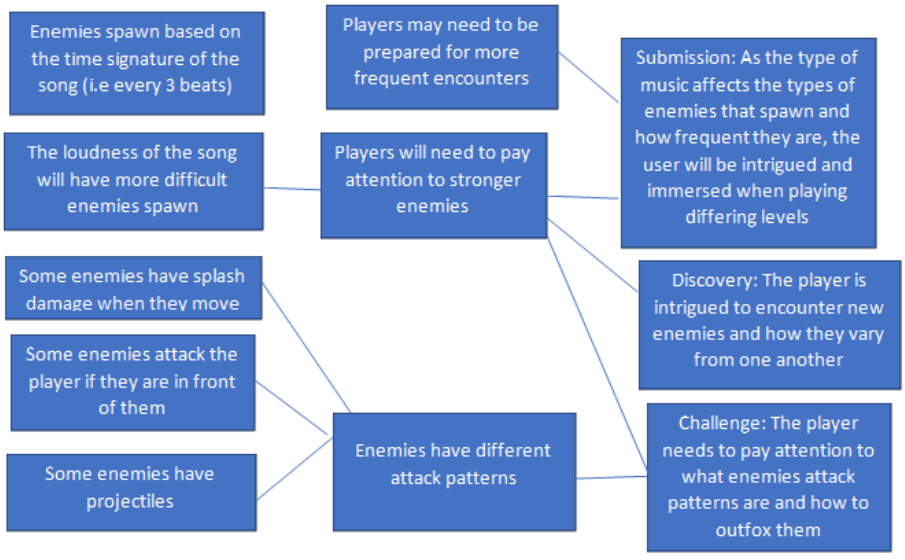


Appendix C – MDA Analysis






A player can attack an enemy if the enemy is within the weapon radius
Players spawn with a dagger that attacks an enemy directly in front
Spears/bows attack enemies horizontally/vertically in a 2/4 tile spike
Broadswords attack enemies in a 3 tile radius
As the player has just spawned, they will have a basic weapon to ease them into the level
A players offensive strategy will change depending on their current weapon
Discovery: The player will want to explore and see what other weapons are spawned in the game
The instrumentalness of the music will affect what weapons can appear
Different songs will mean a greater variety of weapons
Discovery: As this weapon is the basic starting weapon for the game, the player will expect that there are stronger weapons in the game and wish to find and use them
Challenge: As a player may have to adapt their approach to the game on the fly, they will also have to learn new ways to preserve weapon durability and enemy dispatchment
Fantasy: Weapons will be themed on musical instruments, giving players more info on the game world
Challenge: As there are more weapons in the level, there is more chance players will have weapons they may be unfamiliar with, meaning players will have to master more weapons


Enemies spawn based on the time signature of the song (i.e every 3 beats)
The loudness of the song will have more difficult enemies spawn
Some enemies have splash damage when they move
Some enemies attack the player if they are in front of them
Some enemies have projectiles
Players may need to be prepared for more frequent encounters
Players will need to pay attention to stronger enemies
Enemies have different attack patterns
Submission: As the type of music affects the types of enemies that spawn and how frequent they are, the user will be intrigued and immersed when playing differing levels
Discovery: The player is intrigued to encounter new enemies and how they vary from one another
Challenge: The player needs to pay attention to what enemies attack patterns are and how to outfox them

Appendix D - IEQ

Please answer the following questions by circling the relevant number. In particular, remember that these questions are asking you about how you felt at the end of the game.

1. To what extent did the game hold your attention?

Not at all 1 2 3 4 5 A lot

2. To what extent did you feel you were focused on the game?

Not at all 1 2 3 4 5 A lot

3. How much effort did you put into playing the game?

Very little 1 2 3 4 5 A lot

4. Did you feel that you were trying your best?

Not at all 1 2 3 4 5 Very much so

5. To what extent did you lose track of time?

Not at all 1 2 3 4 5 A lot

6. To what extent did you feel consciously aware of being in the real world whilst playing?

Not at all 1 2 3 4 5 Very much so

7. To what extent did you forget about your everyday concerns?

Not at all 1 2 3 4 5 A lot

8. To what extent were you aware of yourself in your surroundings?

Not at all 1 2 3 4 5 Very aware

9. To what extent did you notice events taking place around you?

Not at all 1 2 3 4 5 A lot

10. Did you feel the urge at any point to stop playing and see what was happening around you?

Not at all 1 2 3 4 5 Very much so

11. To what extent did you feel that you were interacting with the game environment?

Not at all 1 2 3 4 5 Very much so

12. To what extent did you feel as though you were separated from your real-world environment?

Not at all 1 2 3 4 5 Very much so

13. To what extent did you feel that the game was something you were experiencing, rather than something you were just doing?

Not at all 1 2 3 4 5 Very much so

14. To what extent was your sense of being in the game environment stronger than your sense of being in the real world?

Not at all 1 2 3 4 5 Very much so

15. At any point did you find yourself become so involved that you were unaware you were even using controls?

Not at all 1 2 3 4 5 Very much so

16. To what extent did you feel as though you were moving through the game according to your own will?

Not at all 1 2 3 4 5 Very much so

17. To what extent did you find the game challenging?

Not at all 1 2 3 4 5 Very difficult

18. Were there any times during the game in which you just wanted to give up?

Not at all 1 2 3 4 5 A lot

19. To what extent did you feel motivated while playing?

Not at all 1 2 3 4 5 A lot

20. To what extent did you find the game easy?

Not at all 1 2 3 4 5 Very much so

21. To what extent did you feel like you were making progress towards the end of the game?

Not at all 1 2 3 4 5 A lot

22. How well do you think you performed in the game?

Very poor 1 2 3 4 5 Very well

23. To what extent did you feel emotionally attached to the game?

Not at all 1 2 3 4 5 Very much so

24. To what extent were you interested in seeing how the game's events would progress?

Not at all 1 2 3 4 5 A lot

25. How much did you want to ''win'' the game?

Not at all 1 2 3 4 5 Very much so

26. Were you in suspense about whether or not you would win or lose the game?

Not at all 1 2 3 4 5 Very much so

27. At any point did you find yourself become so involved that you wanted to speak to the game directly?

Not at all 1 2 3 4 5 Very much so

28. To what extent did you enjoy the graphics and the imagery?

Not at all 1 2 3 4 5 A lot

29. How much would you say you enjoyed playing the game?

Not at all 1 2 3 4 5 A lot

30. When interrupted, were you disappointed that the game was over?

Not at all 1 2 3 4 5 Very much so

31. Would you like to play the game again?

Definitely not 1 2 3 4 5 Definitely yes

32. Did you feel emotionally different when playing the game?

Definitely not 1 2 3 4 5 Definitely yes

33. Did you ever want to stop playing the game at any point?

Definitely not 1 2 3 4 5 Definitely yes

34. Did you ever notice any differences during gameplay?

Definitely not 1 2 3 4 5 Definitely yes

35. Did the game feel intense?

Definitely not 1 2 3 4 5 Definitely yes

36. Did you enjoy the visuals of the game?

Definitely not 1 2 3 4 5 Definitely yes

For any of the questions above, please write a comment below detailing why you felt this was the case.

Appendix E – Parent Enemy Class

```
Unity Script | 4 references
public class Enemy : MonoBehaviour
{

    //private GameManager man;
    public Vector2 attackSize;
    public Transform enemyTransform;
    public LayerMask playerMask;
    private MissedBeat mBeat;
    public bool canMove;
    public float rawDist;
    public ParticleSystem enemyParticles;
    protected int currentScale;
    protected Vector3 big = new Vector3(1.2f, 1.2f, 1.0f);
    protected Vector3 small = new Vector3(1.0f,1.0f,1.0f);
    public BoxCollider2D enemyCollider;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        currentScale = 0;
        big = new Vector3(1.2f, 1.2f, 1);
        small = new Vector3(1, 1, 1);
        enemyCollider.enabled = false;
        Invoke("StartColliding", 1f);
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        GameObject d = GameManager.Instance.defaultObject;
        if(d != null )
        {
            this.gameObject.transform.localScale = d.transform.localScale;
        }
    }
    0 references
    void StartColliding()
    {
        enemyCollider.enabled = true;
    }
    Unity Message | 0 references
    public void OnDestroy()
    {
        Instantiate(enemyParticles, transform.position, transform.rotation);
        GameManager.Instance.UnregisterEnemy(GetInstanceID());
    }
}
```

Appendix F – Zombie Enemy Script

```
public class ZombieController : Enemy
{
    private GameManager man;
    private PlayerController player;
    private GameObject p;
    private Rigidbody2D playerRb;
    public LayerMask walls;
    private int beatCounter;
    public int beatThreshold;
    public BoxCollider2D box;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        p = GameObject.Find("Player");
        //man = GameObject.Find("GameManager").GetComponent<GameManager>();
        man = GameManager.Instance; //gets the singleton
        man.RegisterEnemy(GetInstanceID()); //gets the unique identifier for enemy
        player = GameObject.Find("Player").GetComponent<PlayerController>();
        box.enabled = true;
    }
    // Update is called once per frame
    Unity Message | 0 references
    void FixedUpdate()
    {
        if(man.onBeat == true && man.returnEnemyMove(GetInstanceID())) //enemy assumes perfect movement
        {
            man.setEnemyMove(GetInstanceID());
            /*
            man.setEnemyMove(GetInstanceID()); //this will set the enemy to be false after beat is called so the enemy doesn't continually move
            if (man.GetEnemySize() == false)
            {
                this.gameObject.transform.localScale = new Vector3(1.2f, 1.2f, 1f);
            }
            else if (man.GetEnemySize() == true) this.gameObject.transform.localScale = new Vector3(1.0f, 1.0f, 1.0f);
            */
            float horizontalDistance = p.transform.position.x - transform.position.x;
            float verticalDistance = p.transform.position.y - transform.position.y;
            Vector3 moveDirection = Vector3.zero;
            beatCounter++;
            float positiveHorizontal = horizontalDistance;
            float positiveVertical = verticalDistance;
            positiveHorizontal = Mathf.Abs(positiveHorizontal);
            positiveVertical = Mathf.Abs(positiveVertical);
```

```
            if(beatCounter >= beatThreshold)
            {
                if (positiveHorizontal < 1f && positiveVertical < 1f)
                {
                    moveDirection = new Vector3(0, -rawDist, 0);
                }
                else if ((positiveHorizontal < positiveVertical || positiveVertical < 1f) && positiveHorizontal >= 1f) //if there is
                {
                    if(horizontalDistance < 0)
                    {
                        moveDirection = new Vector3(-rawDist, 0, 0);
                    }
                    else
                    {
                        moveDirection = new Vector3(rawDist, 0, 0);
                    }
                }
                else if ((positiveHorizontal > positiveVertical || positiveHorizontal < 1f) && positiveVertical >= 1f)
                {
                    if(verticalDistance < 0)
                    {
                        moveDirection = new Vector3(0, -rawDist, 0);
                    }
                    else
                    {
                        moveDirection = new Vector3(0, rawDist, 0);
                    }
                }
                if (!Physics2D.OverlapCircle(transform.position += moveDirection, 0.2f, walls)) transform.position += moveDirection;
                beatCounter = 0;
            }
        }
    }
    0 references
    public void TurnBox()
    {
        box.enabled = !box.enabled;
    }
}
```

Appendix G – Mimic Controller Script

```
public class MimicController : Enemy
{
    /// <summary>
    /// private GameManager man;
    /// </summary>
    private GameManager man;
    private PlayerController player;
    public LayerMask walls;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        man = GameManager.Instance;
        man.RegisterEnemy(GetInstanceID());
        player = GameObject.Find("Player").GetComponent<PlayerController>();
    }
    Unity Message | 0 references
    private void Update()
    {
        GameObject d = GameManager.Instance.defaultObject;
        if (d != null)
        {
            this.gameObject.transform.localScale = d.transform.localScale;
        }
    }
    // Update is called once per frame
    Unity Message | 0 references
    void FixedUpdate()
    {
        if(PlayerController.moving == true)
        {
            if (player.horizontal)
            {
                if (!Physics2D.OverlapCircle(transform.position += new Vector3(-player.direction * rawDist, 0, 0), 0.2f, walls)) transform.position += new Vector3(-player.direction * rawDist, 0, 0);
            }
            else if (!player.horizontal)
            {
                if (!Physics2D.OverlapCircle(transform.position += new Vector3(0, -player.direction * rawDist, 0), 0.2f, walls)) transform.position += new Vector3(0f, -player.direction * rawDist, 0f);
            }
        }
    }
}
```

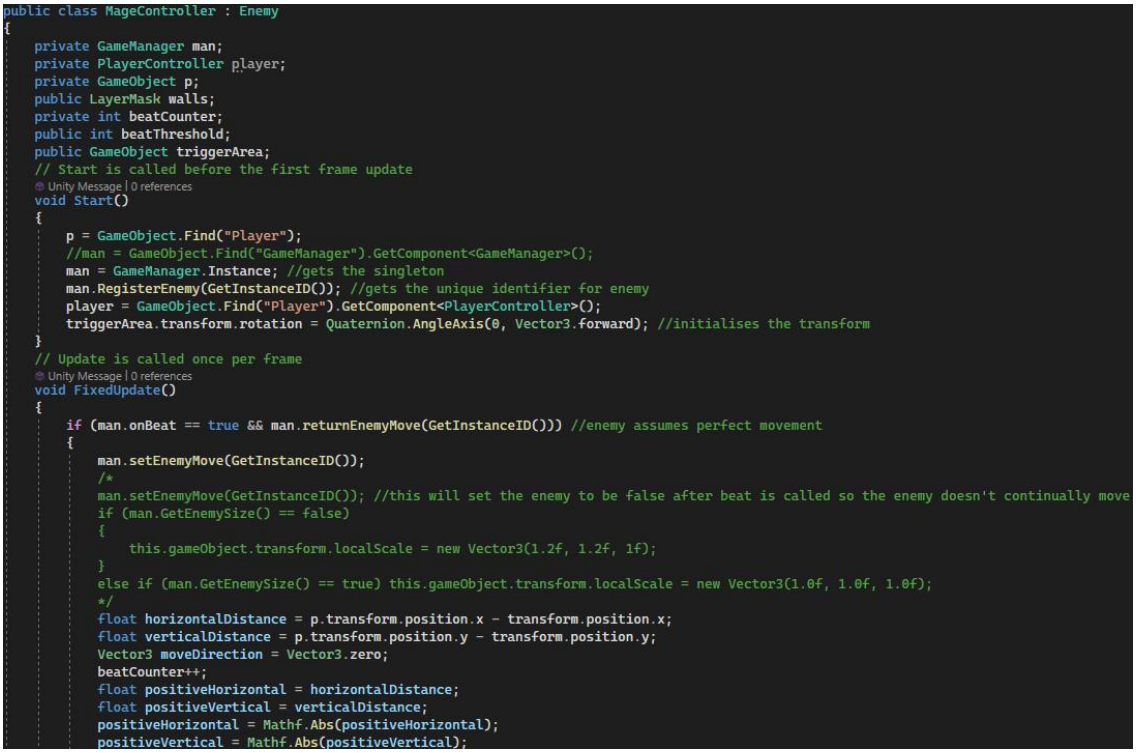
Appendix H – Skeleton Controller Script

```
public class SkeletonController : Enemy
{
    private GameManager man;
    public LayerMask walls;
    private int beatCounter;
    public int beatThreshold;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        man = GameManager.Instance; //gets the singleton
        man.RegisterEnemy(GetInstanceID()); //gets the unique identifier for enemy
    }
    // Update is called once per frame
    Unity Message | 0 references
    void FixedUpdate()
    {
        if(man.onBeat == true && man.returnEnemyMove(GetInstanceID())) //checks that the enemy is within the beat and that the enemy is allowed to move
        {

            man.setEnemyMove(GetInstanceID()); //this will set the enemy to be false after beat is called so the enemy doesn't continually move
            Vector3 moveDirecion = Vector3.zero;
            beatCounter++;
            if(beatCounter >= beatThreshold)
            {
                int randomDirection = (int)Random.Range(0, 3);
                if (randomDirection == 0)
                {
                    moveDirecion = new Vector3(rawDist, 0, 0);
                }
                else if(randomDirection == 1)
                {
                    moveDirecion = new Vector3(0, rawDist, 0);
                }
                else if(randomDirection == 2)
                {
                    moveDirecion = new Vector3(-rawDist, 0, 0);
                }
                else if(randomDirection == 3)
                {
                    moveDirecion = new Vector3(0, -rawDist, 0);
                }
                if (!Physics2D.OverlapCircle(transform.position += moveDirecion, 0.2f, walls)) transform.position += moveDirecion;
                beatCounter = 0;
            }
        }
    }
}
```

Appendix I – Mage Controller Script and Bullet Scripts

Mage



```
public class MageController : Enemy
{
    private GameManager man;
    private PlayerController player;
    private GameObject p;
    public LayerMask walls;
    private int beatCounter;
    public int beatThreshold;
    public GameObject triggerArea;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        p = GameObject.Find("Player");
        //man = GameObject.Find("GameManager").GetComponent<GameManager>();
        man = GameManager.Instance; //gets the singleton
        man.RegisterEnemy(GetInstanceID()); //gets the unique identifier for enemy
        player = GameObject.Find("Player").GetComponent<PlayerController>();
        triggerArea.transform.rotation = Quaternion.AngleAxis(0, Vector3.forward); //initialises the transform
    }
    // Update is called once per frame
    Unity Message | 0 references
    void FixedUpdate()
    {
        if (man.onBeat == true && man.returnEnemyMove(GetInstanceID())) //enemy assumes perfect movement
        {
            man.setEnemyMove(GetInstanceID());
            /*
            man.setEnemyMove(GetInstanceID()); //this will set the enemy to be false after beat is called so the enemy doesn't continually move
            if (man.GetEnemySize() == false)
            {
                this.gameObject.transform.localScale = new Vector3(1.2f, 1.2f, 1f);
            }
            else if (man.GetEnemySize() == true) this.gameObject.transform.localScale = new Vector3(1.0f, 1.0f, 1.0f);
            */
            float horizontalDistance = p.transform.position.x - transform.position.x;
            float verticalDistance = p.transform.position.y - transform.position.y;
            Vector3 moveDirection = Vector3.zero;
            beatCounter++;
            float positiveHorizontal = horizontalDistance;
            float positiveVertical = verticalDistance;
            positiveHorizontal = Mathf.Abs(positiveHorizontal);
            positiveVertical = Mathf.Abs(positiveVertical);
            if (beatCounter >= beatThreshold)
            {
                if (positiveHorizontal < 1f && positiveVertical < 1f)
                {
                    moveDirection = new Vector3(0, -rawDist, 0);
                    triggerArea.transform.rotation = Quaternion.AngleAxis(180, Vector3.forward);
                }
                else if ((positiveHorizontal < positiveVertical || positiveVertical < 1f) && positiveHorizontal >= 1f) //if there is a negligable distance the AI should move vertically
                {
                    if (horizontalDistance < 0)
                    {
                        moveDirection = new Vector3(-rawDist, 0, 0);
                        triggerArea.transform.rotation = Quaternion.AngleAxis(90, Vector3.forward);
                    }
                    else
                    {
                        moveDirection = new Vector3(rawDist, 0, 0);
                        triggerArea.transform.rotation = Quaternion.AngleAxis(-90, Vector3.forward);
                    }
                }
                else if ((positiveHorizontal > positiveVertical || positiveHorizontal < 1f) && positiveVertical >= 1f)
                {
                    if (verticalDistance < 0)
                    {
                        moveDirection = new Vector3(0, -rawDist, 0);
                        triggerArea.transform.rotation = Quaternion.AngleAxis(180, Vector3.forward);
                    }
                    else
                    {
                        moveDirection = new Vector3(0, rawDist, 0);
                        triggerArea.transform.rotation = Quaternion.AngleAxis(0, Vector3.forward);
                    }
                }
                if (!Physics2D.OverlapCircle(transform.position += moveDirection, 0.2f, walls)) transform.position += moveDirection;
                beatCounter = 0;
            }
        }
    }
}
```

Bullet

```
public class BulletController : MonoBehaviour
{
    public float speed = 5f;
    private Transform playerPos;
    bool negative = false;

    Unity Message | 0 references
    private void Awake()
    {
        playerPos = GameObject.Find("Player").GetComponent<Transform>();
        float horizontal = playerPos.position.x - this.transform.position.x;
        float vertical = playerPos.position.y - this.transform.position.y;
        if (Mathf.Abs(horizontal) < Mathf.Abs(vertical)) //horizontal should be of negligable distance
        {
            if (vertical < 0) negative = true;
            else negative = false;
        }
        else if (Mathf.Abs(horizontal) > Mathf.Abs(vertical))
        {
            if(horizontal < 0)
            {
                negative = true;
            }
            else
            {
                negative = false;
            }
        }
        if(negative && (Mathf.Abs(horizontal) < Mathf.Abs(vertical)))
        {
            this.GetComponent<Rigidbody2D>().velocity = new Vector2(0, -speed);
        }
        else if(negative && (Mathf.Abs(horizontal) > Mathf.Abs(vertical)))
        {
            this.GetComponent<Rigidbody2D>().velocity = new Vector2(-speed, 0);
        }
        else if(!negative && (Mathf.Abs(horizontal) < Mathf.Abs(vertical)))
        {
            this.GetComponent<Rigidbody2D>().velocity = new Vector2(0, speed);
        }
        else if (!negative && (Mathf.Abs(horizontal) > Mathf.Abs(vertical)))
        {
            this.GetComponent<Rigidbody2D>().velocity = new Vector2(speed, 0);
        }
    }
```

```
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.tag == "TriggerZone" || collision.gameObject.name.Contains("Mage"))
        {
            Debug.Log("Called");
            return;
        }
        else if (collision.gameObject.tag == "Player")
        {
            PlayerHealth.TakeDamage();
            Destroy(this.gameObject);
        }
    }
}
```

Appendix J – Immersion Results for Control and Experimental Games

Control

Question	1	2	3	4	5	6	7	8	9	10
1	4	4	5	3	4	5	5	4	5	4
2	4	5	4	5	4	5	4	4	5	5
3	4	3	4	4	5	5	4	5	4	4
4	5	4	5	4	5	5	5	5	5	3
5	3	3	4	5	4	3	4	4	5	4
6 (Inverted)	2	4	1	4	4	3	4	3	3	4
7	2	5	4	5	4	3	4	3	3	5
8 (Inverted)	2	5	2	3	4	4	3	2	3	4
9 (Inverted)	2	3	4	5	4	4	4	4	4	5
10 (Inverted)	2	5	4	5	5	2	5	4	4	4
11	4	2	4	5	5	2	3	3	4	3
12	3	4	2	2	4	4	2	4	3	1
13	3	3	3	2	4	2	4	4	4	2
14	3	4	4	2	4	4	3	4	4	2
15	4	3	3	2	5	2	2	3	2	2
16	4	2	4	4	4	2	3	3	3	2
17	4	3	4	5	5	5	5	4	4	4
18 (Inverted)	3	5	4	3	5	2	4	3	2	5
19	3	3	4	2	4	3	3	4	4	4
20 (Inverted)	4	3	3	5	5	4	4	4	3	5
21	4	4	3	2	5	3	2	3	3	3
22	3	4	3	2	2	1	4	2	2	3
23	3	3	2	1	4	1	2	3	2	2
24	3	2	3	2	5	1	3	2	4	5
25	5	3	2	4	5	4	4	3	5	4
26	3	2	3	5	3	2	3	3	4	2
27	2	2	3	2	4	1	1	3	5	1
28	3	3	5	4	5	3	2	4	4	3
29	3	4	5	3	4	3	3	5	5	4
30	3	3	4	4	5	3	2	3	4	2
31	4	4	4	3	1	3	4	3	5	3
32	3	4	4	1	5	1	2	3	4	2
33 (Inverted)	3	5	4	4	4	3	4	3	3	5
34	2	3	3	2	4	2	4	2	4	3
35	3	4	3	4	4	3	3	4	4	2
36	3	4	4	4	4	3	2	4	4	4
Total	**115**	**127**	**127**	**122**	**152**	**106**	**120**	**124**	**136**	**120**

Experimental

Question	1	2	3	4	5	6	7	8	9	10
1	5	4	4	5	5	5	4	5	5	5
2	4	4	5	4	5	5	4	4	5	5
3	4	5	4	4	5	5	5	5	5	5
4	5	5	4	5	5	4	5	4	5	5
5	3	3	3	3	4	5	4	4	5	4
6 (Inverted)	2	2	3	3	4	5	2	2	5	2
7	2	3	5	4	5	4	5	3	1	4
8 (Inverted)	2	2	3	4	5	4	4	2	4	1
9 (Inverted)	4	5	4	4	5	5	5	1	5	2
10 (Inverted)	5	5	5	5	5	5	5	5	5	4
11	4	4	2	3	5	5	5	4	2	4
12	2	4	4	3	5	4	4	3	4	3
13	2	1	4	4	3	4	4	4	2	4
14	2	2	4	2	4	4	4	2	4	3
15	1	1	2	1	3	4	3	2	2	3
16	2	2	4	4	4	4	5	2	3	4
17	5	4	5	4	5	4	5	5	4	4
18 (Inverted)	4	3	4	4	4	5	3	5	5	4
19	5	3	3	3	4	4	3	4	4	5
20 (Inverted)	4	4	3	3	4	5	5	2	4	4
21	4	2	2	4	5	4	5	3	3	5
22	2	4	5	2	3	3	3	4	2	4
23	4	1	4	2	4	3	2	3	1	3
24	4	3	2	4	4	4	4	3	3	5
25	4	5	5	4	5	5	5	5	5	4
26	2	1	4	4	4	4	1	4	5	3
27	1	1	2	3	4	2	1	3	2	3
28	5	2	5	3	4	4	4	5	3	5
29	5	3	4	2	4	4	5	5	4	5
30	5	1	4	2	5	4	1	5	2	3
31	4	3	5	4	5	5	3	5	3	4
32	4	4	4	2	4	3	1	3	1	3
33 (Inverted)	2	4	3	5	4	5	3	5	4	5
34	4	3	4	2	2	4	3	4	4	2
35	5	1	2	2	4	4	4	4	4	3
36	5	2	4	2	5	4	5	4	3	5
Total	**127**	**106**	**134**	**119**	**155**	**152**	**134**	**133**	**128**	**137**

Appendix K – Critical values in F Table (University of Sussex, 2005)

Critical values of F for the 0.05 significance level:

	1	2	3	4	5	6	7	8	9	10
1	161.45	199.50	215.71	224.58	230.16	233.99	236.77	238.88	240.54	241.88
2	18.51	19.00	19.16	19.25	19.30	19.33	19.35	19.37	19.39	19.40
3	10.13	9.55	9.28	9.12	9.01	8.94	8.89	8.85	8.81	8.79
4	7.71	6.94	6.59	6.39	6.26	6.16	6.09	6.04	6.00	5.96
5	6.61	5.79	5.41	5.19	5.05	4.95	4.88	4.82	4.77	4.74
6	5.99	5.14	4.76	4.53	4.39	4.28	4.21	4.15	4.10	4.06
7	5.59	4.74	4.35	4.12	3.97	3.87	3.79	3.73	3.68	3.64
8	5.32	4.46	4.07	3.84	3.69	3.58	3.50	3.44	3.39	3.35
9	5.12	4.26	3.86	3.63	3.48	3.37	3.29	3.23	3.18	3.14
10	4.97	4.10	3.71	3.48	3.33	3.22	3.14	3.07	3.02	2.98

Appendix L – Reckoner Results Table

Participant	Control Score	Control Deaths	Control Misses	Experimental Score	Experimental Deaths	Experimental Misses
1	N/A	N/A	N/A	8	2	53
2	N/A	N/A	N/A	11051	2	37
3	N/A	N/A	N/A	8	2	42
4	893	2	38	20	3	29
5	142	2	24	35	3	93
6	22	3	36	19	3	49
7	1480	2	43	8	3	34
8	5	1	30	684	3	18
9	1519	2	39	6346	2	13
10	559	2	42	12	2	32

Appendix M – Twilight Results Table

Participant	Control Score	Control Deaths	Control Misses	Experimental Score	Experimental Deaths	Experimental Misses
1	N/A	N/A	N/A	20	3	50
2	N/A	N/A	N/A	144	2	30
3	N/A	N/A	N/A	100285	2	34
4	481000	1	45	45	3	30
5	11	1	7	1470	2	24
6	24	3	32	197	3	35
7	79992648	1	24	142	3	38
8	5055	1	11	14768646	2	34
9	222	1	48	328349	1	29
10	4281	2	47	1961	2	21

Appendix N - Little Dark Age Results Table

Participant	Control Score	Control Deaths	Control Misses	Experimental Score	Experimental Deaths	Experimental Misses
1	N/A	N/A	N/A	37	1	40
2	N/A	N/A	N/A	620000	1	30
3	N/A	N/A	N/A	8958182	1	12
4	5155681	1	28	25468669	1	29
5	484	2	12	2155206	1	25
6	9617	1	5	176021052	0	12
7	16976	2	6	1248360	0	20
8	2308	1	10	27360179	0	12
9	191	2	59	19296586	1	2
10	116710	0	10	200700	0	8

Appendix O – Shimmy Results Table

Participant	Control Score	Control Deaths	Control Misses	Experimental Score	Experimental Deaths	Experimental Misses
1	N/A	N/A	N/A	12	2	50
2	N/A	N/A	N/A	145716	1	35
3	N/A	N/A	N/A	308574	1	18
4	15	1	58	728	2	32
5	7	2	52	9	2	75
6	110	2	29	260	2	36
7	2387205	1	20	24	2	52
8	78	0	17	2213	2	18
9	300	2	33	1704	1	20
10	1	1	50	1	1	50

Appendix P – She's Lost Control Results Table

Participant	Control Score	Control Deaths	Control Misses	Experimental Score	Experimental Deaths	Experimental Misses
1	N/A	N/A	N/A	1900	2	70
2	N/A	N/A	N/A	550	2	27
3	N/A	N/A	N/A	14847163	1	28
4	6519	2	41	704	3	14
5	5512	1	30	18326	1	42
6	300	1	42	55	3	33
7	2837828	1	15	272	1	44
8	430	1	12	1811520484	0	27
9	29147	1	33	13885505	0	20
10	286887	1	30	1439	2	10