Faculty of Science and Technology

May 2021

Automatic Level Generation for a VR Rhythm Game

By Dan Udal

DISSERTATION DECLARATION

Retention

Confidentiality

Signed: Dan Udal
Name: Dan Udal
Date: 14/05/21
Programme: BSc GSE

Abstract

Automatic level generation is very rare in rhythm games and is often a concept modded in by fans of a game rather than one created by the developers themselves. This is often due to the nature of their rhythm game where it can be hard to create an algorithm that generates satisfying to play levels. In spite of this players of games often enjoy having access to more levels and while the solutions given by most developers can be effective they are not without their problems. This project seeks to design a system which can automatically generate levels for a simple rhythm game which will be built around it. However the algorithm will not be solely reliant on this rhythm game as it should be applicable to other games as well. By building the algorithm in this way other developers would be able to apply it to their own games in development, which will give them an unlimited set of levels without needing each one to be designed manually.

Contents

1. Introduction

1.1 Overview

Rhythm games have possibly the widest appeal of any game genre as they're able to be extremely simple in concept while providing a deep and fun experience to players of any demographic. The most successful rhythm games, such as guitar (Harmonix, 2005) hero and recently beat saber (Hyperbolic Magnetism, 2018), tend to have simple concepts and mechanics. This is especially so in beat saber where the player only has to slice blocks in the correct direction while avoiding obstacles. Guitar hero has a slightly more focused audience to people who are interested in guitars, which is still an extremely wide audience. This wide appeal has allowed some rhythm games to break into mainstream media, beat saber was played on a national tv program, ant and dec's saturday night takeaway[5]. Due to this appeal rhythm games should strive to make their experiences as simple and elegant as possible in order to make themselves intuitive to use and easy to play.
This project's goal is to create a fun rhythm game to play that implements an automatic level generation algorithm which will allow players to play levels created from any music they like.

1.2 Context

1.2.1 Rationale

Most rhythm games use hand crafted levels with bespoke music for their gameplay which often results in well designed levels which provide a good challenge to players. However an issue with rhythm games built this way can be a limited set of levels available to players. The games can create an enjoyable experience for a long time as players try to get perfect scores on the levels, however once this goal has been achieved there's very little replay value for the game anymore. A way developers have tried to alleviate this is by allowing players to create their own levels with a level editor. This has the advantage of creating more levels than players could ever hope to complete if the game is popular. While this is a good solution it does come with problems. First, the people creating the levels are very rarely versed in good practice for game design. This means that their levels can sometimes be frustrating to play due to the creator not taking good user experience into mind[4]. For a game based on music, such as beat saber, it can also sometimes be difficult to find a good level for a song you enjoy if that song is very obscure. A good solution to all these problems is to have the game itself generate levels automatically. This solution has been implemented by a team of fans of beat saber using a neural network AI to read songs and produce levels[2]. Another system that uses AI was made by Nick

Sypteras[3] to generate levels for the game Osu, a rhythm game for pc where you click on circles in time with music. Due to the nature of AI this method is unpredictable and can be inconsistent in producing fun levels. Because of this, this project will take an algorithmic approach to solving the problem similar to the game music95[1] which is a game that uses an algorithm to generate levels from user provided music.

1.2.2 Scope

The scope of this project is focused on the creation of a VR rhythm game that makes use of algorithmic level generation which can create levels based upon midi files given to the game. The rhythm game for this project will be kept basic as it isn't the primary focus, rather it is the background for the focus of a level generation algorithm.

1.3 Aims and Objectives

1.3.1 Aims

1. To develop a music based rhythm game in VR that uses a 2D coordinate system to define the level
2. To develop a system that can read a music file and convert the notes into 2D coordinates in order to generate levels for a rhythm game

1.3.2 Objectives

1. Research background for the project
    a. Research the structure of midi files
    b. Look into how to load midi files into unity in a way that allows them to be used mathematically
    c. Research tools available for creating VR games in unity
    d. Develop an equation for converting midi notes into a 2D coordinate system
2. Build the VR environment for the rhythm game
    a. Install any tools for creating a VR game
    b. Use their documentation to implement the tools in unity
3. Implement the loading of midi files for unity
    a. Implement a system for loading midi files found during research
    b. Convert midi notes into a polar 3D coordinate system where r will be a constant (r, θ, φ) and the other 2 coordinates will be angles determined by the note
    c. Use these coordinates to generate levels for the rhythm game
4. Implement the basic mechanics for the rhythm game

    a. Create the block spawning algorithm
    b. Build a player object which has health and a collider
    c. Create weapons the player can use to destroy blocks

5. Build a simple menu system for selecting songs
    a. Create an algorithm that spawns buttons dynamically based on the number of songs the user has in their file
    b. Code an onClick function that starts the song a button relates to
    c. Build a way for the user to click buttons within the VR environment

1.4 Methodology

1.4.1 Background

The background research for this project will be mostly focused on ways midi files can be used in mathematical models as well as how VR games are made within unity. It will also look into games with a similar system such as those described above

1.4.2 Design

The design of this project will be predicated on creating a fun experience for a very wide audience, which should be the primary goal of any game. The design will also focus on making the level creation system as intuitive as possible for end users as not everyone understands technology well enough to use complex tools, like the ones that exist in other games.

1.4.3 Implementation

The implementation of this game will start by creating the most basic systems to meet the design plan given and then gradually expand to cover more ideas as time allows. This will ensure the game is playable as soon as possible while leaving room for improvement should time allow for it.

1.4.4 Analysis

The analysis of this project will be performed using specific critical criteria. The aim of this analysis will be to determine where the game still has room for improvement and the weaknesses of the original idea compared to alternative solutions to the problem given.

2. Background

2.1 Inspirations and Comparisons

The main inspiration for this project was Beat Saber. That game has overwhelmingly positive reviews and is considered by many to be a flawless game as well as being extremely fun to play. This project began with some ideas of Beat Saber, such as blocks spawning in time with music and the player needing to destroy them and dodge obstacles, then built upon them. Other inspirations for this project include Osu (Dan Herbert, 2007) which is a rhythm game for pc and other projects for automatic level generation such as beat sage[2].

2.1.1 Beat Saber

Beat Saber is a VR rhythm game released in 2018. In this game the player holds 2 lightsabers which they use to slice blocks in the correct direction, which spawn to reach the player in time with the music. In the levels the player also has to avoid slicing bombs with their swords and move out the way of red walls. The game is extremely simple at its core with very few mechanics people need to understand in order to play it. This makes the game extremely fun to play for many different people, however the game only comes with a limited number of premade levels. The creators of the game attempted to solve this by releasing a level editor that anyone is able to use but, as seen in figure 1 below, the program is fairly complex and not accessible to the majority of players.

2.1.2 Beat Sage

A solution the Beat Saber community have tried to provide is an automatic level generation system in beat sage. This system makes use of a non deterministic AI which they trained to generate levels. This has produced fairly good results but it has the problem of being very unpredictable. This means it doesn’t always produce good levels and there can be significant variance in the style of levels it creates.

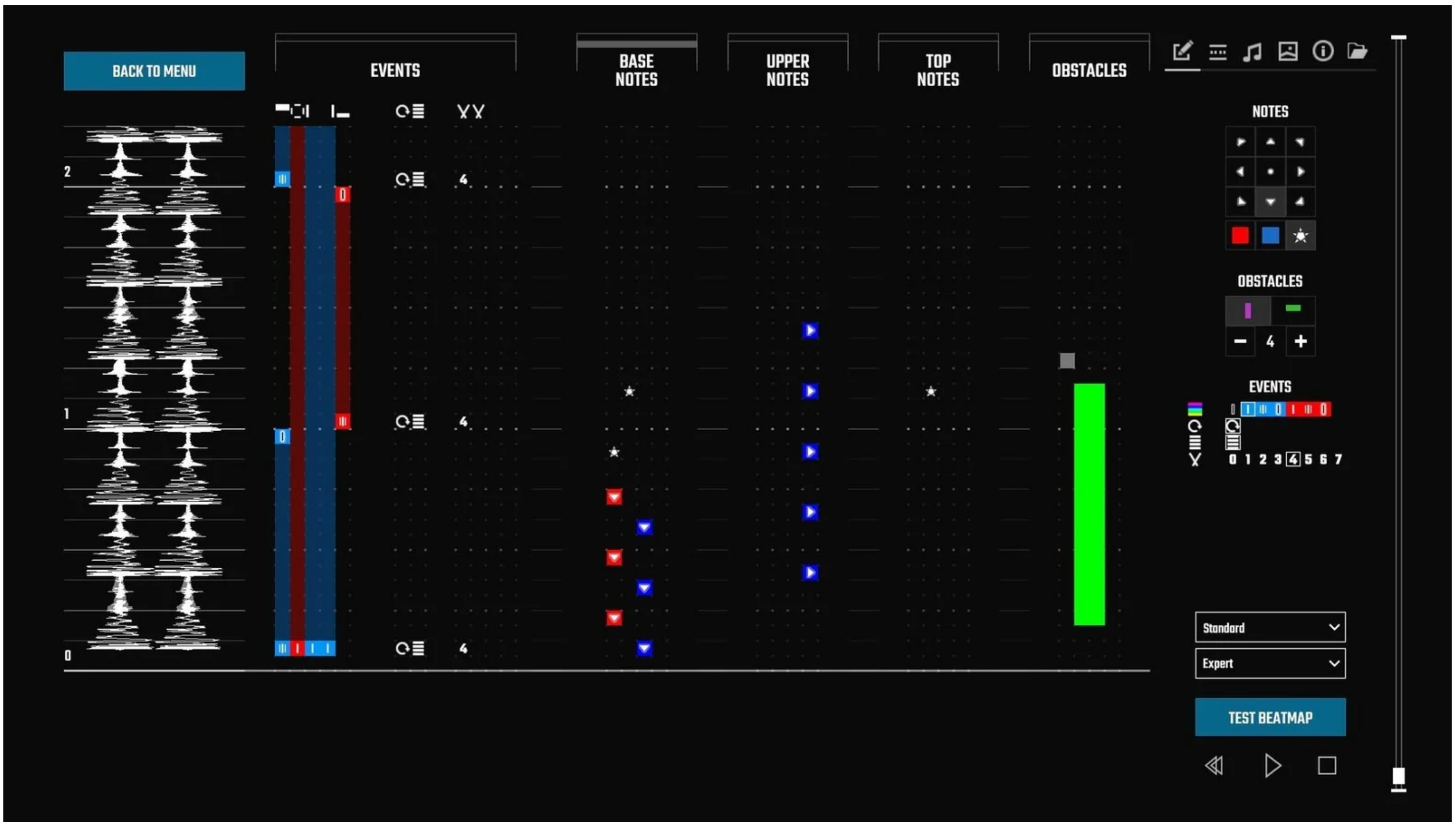

Figure 1

2.1.3 Osu!

Osu is a pc game similar to the arcade game dance dance revolution. In it you have to click on circles in time with the music as well as follow some moving circles with your mouse. This game like beat saber allows players to create their own levels, however levels in osu are more complex because the number of places a circle can be are much higher due to the game being in continuous 2D space. This makes it much more difficult to create an automatic level generation system for Osu similar to beat sage for beat saber. An attempt was made in the past, however that project has been abandoned and the website taken down so it’s now impossible to see how that system might have worked. On the blog post explaining the system, the creator stated that it used a non-deterministic AI system, the same approach taken by beat sage and as such it will have had similar problems. These problems would have been exacerbated by the additional complexity of Osu

2.2 Research

2.2.1 Midi Files

Upon researching the structure of midi files it has been observed that they are made up of several types of bytes. They begin with header bytes which define how the file is played, then they are followed by bytes that define each note in the song. Each note requires multiple bytes to define as they have many properties, such as the note to be played, the octave and the velocity or hardness of the note. The complexity of midi files made it clear that creating a midi reader would be an arduous task outside the scope of this project. As such research was made into a prebuilt library solution. The one found is named drywetmidi (Melanchall, 2017) and is written in c#, which allows it to be used within unity. This library allows the reading of individual notes within midi files without the need to recoding complex algorithms to decode the files.

2.2.2 VR Solutions

VR is an immensely complex medium to create a game in. As such it is unfeasible to code a VR interfacing system from scratch, which would be far outside the scope of this project. Instead for this project a unity plugin simply called SteamVR has been used, which includes prefabs that interface with VR inputs automatically and allow for all events to be coded as listeners to certain actions by the player.

3. Development Report

3.1 Design

3.1.1 The Rhythm Game

This project aims to create an experience that is similar to Beat Saber in feel so the core design will take inspiration from that. In the game the player will have 2 weapons they can use. A sword that destroys blocks it strikes and a gun that can destroy blocks by shooting them with bullets. Due to midi files having up to 16 channels playing at once there will be far more blocks than the player can reasonably destroy. Because of this the goal of the game will be to evade blocks primarily and destroying them will be a method of making the evasion easier. The blocks will fire from a fixed distance away and reach the player in a fixed amount of time. By doing this the blocks will always reach the player on the beat of the music if the timer for the song starts several seconds in advance.

The blocks will start tracking towards the player 1 second before their striking time and have a dynamic speed that makes them collide at the correct time. soon after their time to strike blocks will be deleted. The goal of the game will be to survive until the end

The player will have a set amount of hp at the start of a song and will recover some hp whenever they destroy a block with one of their weapons. This means that the player will only lose a level if they get hit by multiple blocks in a row and will be able to heal by putting themselves in slightly more danger than just by dodging.

The gun that the player has access to will work on an ammunition system to balance it against being able to destroy blocks at range, if the player has good aim in the moment. Once all the ammo has been used up the gun will require a couple seconds to reload which will be done by pressing another button on the controller. The player will be able to reload even if the gun isn't empty so that they can refill their ammo during a lull in the song.

3.1.2 VR UX and UI Design

This game will be primarily developed for the vive VR headset as portability is outside the scope of this project. All VR inputs will be handled by the steamVR unity plugin which gives names to all the input types on a vive controller and provides functions to add listeners to those inputs. SteamVR also provides scripts for use with UI, specifically laser pointers which can send events to UI buttons the same as with a mouse click. These pointers will be mapped to the triggers on the vive controllers allowing the player to select their song using these pointers.

The UI in this game will be very minimalistic as in a VR game the screen needs to be kept as clear as possible in order to avoid disorienting the player. The indicator for the gun's ammunition will be put on the gun itself so that the player can move it into their view when they want to check their ammo count. Their health will be a hidden value from the player as it often is with rhythm games. This will put more pressure on the player when they get hit once and give the illusion of difficulty even if the player isn't in any danger of losing the song. This way all players will feel a certain amount of challenge from the game and will never be able to feel completely safe from the challenge.

With VR games motion sickness is a very real concern for users, so with this project the user will be kept stationary except for when they actually move their body. This will minimise the risk of motion sickness for players as their eyes will never see movement except when their actual body moves adding to the wide appeal intended for this game.

3.1.3 Midi File Design

The drywetmidi library that is being used in this project allows singular notes to be extracted from midi files into note objects, which have multiple properties. The properties that will be used from the notes are the octave of the note and the specific note itself. 60 represents middle C. These notes will be mapped onto angles between 0-360 for the horizontal angle, and 0-90 for vertical angles. These will be used to define the position of blocks using 3D polar coordinates where r will be constant for all the blocks. The mapping will be done using modified sigmoid functions because the original sigmoid function maps any value between 0 and 1. With some modification the function can be made to map x = 0 to y => 0. Using these modified functions the range of notes and octaves can be mapped between the angles needed in order to fit the set angles.

After running the values extracted from notes through the sigmoid functions they will be used to spawn blocks for the player to avoid by using the algorithm for converting polar coordinates to the cartesian coordinate system. The blocks will be moved directly towards the location of the player when they first spawn. They will follow this straight path for 4 seconds. Once the blocks reach the last second of their travel time they will start changing direction towards the player with a slow rotation speed. This rotational inertia will make it possible for the player to avoid the blocks. The speed of the blocks will be dynamically adjusted in order to ensure the blocks reach the player at the correct time.

3.1.4 System Design

This game will have 2 scenes in unity. A scene for the main menu and a scene for the gameplay. The game will begin on the main menu and switch to the song scene when one of the songs is chosen by pressing a button. Within the song scene there will be a game controller script that handles a global timer, spawns blocks and plays the music. This will be the primary script for controlling variables for the scene itself. Variables relating to the player and the player's actions will be coded with a player controller script. Each object in the scene that has functionality will have its own script controlling its behaviour while the game controller script will control their interactions with each other and the scene itself.

3.1.5 User Experience Design

Elements that contribute towards a good or bad user experience are often extremely subtle and can be very wide in scope. This means when there is a good user experience the decisions towards that end almost always go unnoticed by the uninitiated. On the other end of the spectrum a bad user experience is extremely noticeable by everyone but people who aren't game devs sometimes can't explain why a game feels bad to play. For this reason when developing a game these subtle decisions need to be taken into consideration even if they're unlikely to be appreciated by the majority of a game's audience.

The first thing that was considered regarding user experience is the size and shape of buttons. The size of a button in a certain dimension determines how quickly a user can click it from the related direction. For example a thin button can be slow to click if your cursor is to the side of it. Keeping this in mind all UI elements in this game were designed such that buttons are sufficiently large to be easy to click with the laser pointers.

Another consideration is nested menus. Sometimes these can be necessary to organise a menu in a game, such as having item buttons nested behind an inventory button. This can be a useful tool if there are many buttons to press that can be sorted into categories but caution must be taken against over using this tool. If menus are overly nested they can become frustrating and slow to navigate for players and if the organisation isn't simple and logical then that can frustrate end users further as they struggle to find the button they want. In this project there will be only a single un-nested menu that users need to use so it will be easy to navigate.

A common problem with VR games is the lack of resistance you feel when objects make contact inside games. This is because people expect, when they touch a solid object, to feel the object push back against them due to a reaction force. This is near impossible to recreate in VR so developers have to find creative solutions around this problem. Beat Saber's approach and the one that has been used in this project is to remove the expectation of physical push back. By making the weapons seem like they cut through objects like nothing is there the player doesn't have their immersion broken by the lack of solidness in the game.

3.2 Scripting

3.2.1 Game Controller

The game controller script is the script at the top of the hierarchy that controls the scene and interactions between other scripts. It has a float timer that increases by delta time once every tick in update. This timer is used to determine when to spawn blocks and begins at -6. This is because blocks start moving 5 seconds before impact so they need to start spawning 5 seconds before the song begins. The timer spawns them 6 seconds in advance in order to leave processing time for when they're initiated. This way there is a one second grace period to spawn blocks before they need to start moving.

The game controller is also where the midi file is loaded and blocks are spawned. In order to load a midi file the GC takes the name of the wav file sent to it when the scene loaded and adds the suffix .mid in order to load in a midi file with the same name. This has the result that the midi and wav files for songs need to have the same name. The GC spawns blocks based on the time they're set to play by using a foreach loop that goes through an ICollection of notes that was loaded in using drywetmidi in the start function. The algorithm for spawning blocks can be seen below

```
foreach(o in notes)
```

```
  If (o.time - 6 <= timer)
    spawnBlock(o)
    notesToRemove.Add(o)

foreach(o in notesToRemove)
  notes.Remove(o)
```

spawnBlock() is a function in the projectile spawner script which creates a block using the note o as the location. The second foreach loop is necessary due to it being impossible to remove elements from an ICollection during a foreach loop that uses it as index. Instead a list must be used to save all the notes that need to be removed and then they can be removed during a second loop.

The GC also controls the audio source in the scene. When the song scene is loaded the GC is passed an audio clip which it gives to the audio source. It then tells the source to start playing once the timer reaches 0 from -6. This will make it start playing when the blocks first reach the player. When the scene is closed everything in it will be reset so the audio clip in the source will default to null, where it can be given a new audio clip again.

The final thing the GC does is end the song scene when the song ends or when the player runs out of hp. It does this by first loading the menu scene with LoadSceneAsync. Then it unloads the song scene using UnloadSceneAsync.

3.2.2 SteamVR Scripting

The steamVR plugin for unity has its own system for scripting actions for VR. With this plugin the code is event driven with everything being triggered by listeners to the controllers or any other input source possible such as a treadmill. The way this is doen is by first generating all the inputs you want for your game using the steamVR input screen under window in the unity editor. By saving those inputs steamVR generates them inside its scripts allowing them to be used in the code like any other variable. It provides a default input set which provides inputs for the trigger and grip buttons on controllers. These are the only button inputs that have been used for this project as the game is very simple.

In order to add a listener to an input in steamVR the action has to be contained within a function. Then the input and its source are coded in as shown here
```
SteamVR_Action_Boolean someAction = SteamVR_Actions._default.GrabPinch
someAction[SteamVR_Input_Sources.source].trigger += someFunction
```
As can be seen here the specific inputs can have multiple sources, such as left hand or right hand. This source is specified within the square brackets. The trigger is the type of event you want to add a listener to such as onStateDown or onStateChange.

Then a function is added to the action in a list of functions it will trigger upon receiving that event.

3.2.3 Gun Shoot

The gun in this game is tracked onto the left hand controller in the camera rig. This means it will follow the left controller when someone uses it in VR. it has been coded to fire whenever the player presses the trigger on the left controller. This has been done with a shoot function that initialises a bullet on the trigger
fire[SteamVR_Input_Sources.LeftHand].onStateDown
This means that upon the trigger having its state go from not pressed to pressed a bullet will be spawned at the location of the gun.

The gun also has an ammo variable that starts at 6 and reduces by 1 every time the gun fires. When the ammo reaches 0 the player can no longer shoot the gun. In order to reload a coroutine has been coded that causes a delay of 2 seconds before returning the ammo count to 6. This has been done through a coroutine in order to create a delay that doesn’t freeze the game as coroutines run in parallel to the rest of the game.

The reload function has been set to trigger when the grip button is pressed down on the left hand controller using the same method as seen above. This gives the player an intuitive button to reload their gun.

3.2.4 Main Menu

The main menu script’s purpose is to load all the wavs contained in the songs folder of the game and spawn a button for each one. It does this by first using Resources.LoadAll() from unity’s resource system. This loads all files in the songs folder that fit into an audioClip type. Then a for loop is used to loop through each wav in the list returned by LoadAll and a button is created in each iteration with a y value determined by (3.33 - i * 1.35). This was determined through some testing. 3.33 is the y value of where the first button needs to spawn in order to be at the very top of the screen while 1.35 is the vertical size of each button. This will cause each button to spawn directly below the last with no gap in between starting from the very top of the screen. Each button is also given a reference to the wav file it’s representing so that the button can then pass the wav to the game controller in the song scene when pressed

3.2.5 UI Select

The UI control in this game has been done using the laser pointer script provided by the steamVR plugin. This creates a laser coming out of the controllers which can be used to press buttons. When the trigger on a controller is pressed it sends a pointerClick event to the scripts which then calls all the functions in a list of listeners.

The pointerClick event also passes through the functions a target variable which determines the object in the scene that the laser is currently pointing at. This means that when the pointer is clicked a function can be used to call the onClick function of any button the laser is currently pointing at. Functions can be added to this list of listeners like any other listener in steamVR
pointer.PointerClick += onClick

3.2.6 Projectile Spawner

This is the script that contains the spawnBlock function described before. It takes in 2 floats that are the note number and its octave, and another float for the time that the block needs to reach the player. It uses 2 sigmoid functions to transform the note and octave into 3D spatial coordinates that it uses to determine the position of the blocks it spawns. The first sigmoid function is for the octave and is as follows
(100 / (1 + Mathf.Exp(0.07f * (32 - octave)))) - 10
Mathf.Exp is the unity function that provides e^x which is necessary for the sigmoid function. The number on the top of the function determines the maxima of the function along with the number added at the end. The maxima of the sigmoid function is the number on top added to the number at the end, in this case that is 100-10=90. This means that the octave will be mapped between 0 and 90. 32 - octave is the x value of the function. The octave is being subtracted from 32 because 32 is roughly the midpoint of what the octave can be so the middle octave will be in the middle of the sigmoid function, at roughly 45 degrees. The 0.07 float is the gradient of the function and it’s there to give a reasonable slope to the function as it was multiplied by 100 on top. Without the 0.07 the function would go from 0 to 90 between a few numbers for octave meaning blocks would spawn either at the bottom or the very top always.

A similar idea was used for the note number sigmoid function, just with different numbers to give a slightly different mapping to the function
(410 / (1 + Mathf.Exp(0.07f * (32 - note)))) - 50
As before the maximum of the function is 410 - 50 = 360. This gives a full 360 horizontal area to spawn blocks in. once again the gradient is 0.07 to give a reasonable slope and the note is subtracted from 32 so that it can give negative values to the function.

3.2.7 Asteroid Controller

The asteroid controller determines how the blocks move and uses the timer in the game controller to determine behaviour at any given time. If the time of reaching the player - GC.timer is less than 1 then the block will track to the player with a slow rotation speed. This is when the block has 1 second left to reach the player and changes its behaviour to retarget. If the difference is greater than 1 then a test is given to see if the difference is below 5. If it is, the block will follow a fixed path forward which will be aimed at where the player was at the time the block was created. It will also have a much greater speed during this time.

The speed of the block is dynamically updated every tick to ensure the travel time of the block is always constant. The only way to ensure this when the distance can change is to adjust the speed as time = speed/distance. This formula is used to recalculate the block's speed. The block's first distance is calculated towards a radius around the player which is the point where the block will start tracking towards the player. The speed is much greater in this part of the travel time because speed = distance/time. The block has 4 times the time to travel the distance during this period but it has more than 4 times the distance to travel so the speed is overall higher than in the final period of the travel time.

If the time of impact is 0.5 seconds below the timer of the GC then the block will be destroyed as this means it has overshot and the player avoided a collision with it. It will continue to rotate towards the player for 0.5 seconds which means the player will be able to see the blocks they avoid. After this the block is deleted.

The last section of the asteroid controller is for the collision function. When the block detects a collision with a gameObject that is a trigger it receives a collision function call. When this happens the block checks if the object collided with is tagged as a bullet or a sword. This will destroy the block and heal the player when the player hits the block with the sword or shoots it with their gun.

3.2.8 Minor Scripts

Some of the scripts in this game are very small and simple. For example the bullet controller simply has 1 line of code that translates it forward at a fixed speed. The player controller script tracks the player's health. It has an integer value for hp and will subtract 1 whenever it detects a collision with a block. If the player's health reaches 0 it will end the game with the endGame function in the game controller.

Button press is another small script. It contains the onClick function for the main menu buttons which will load the song scene when pressed and pass into the game controller the wav file that they have stored in their script. The audioClip variable in this script is given to them by the main menu when they're initialised.

3.3 Scene Building

3.3.1 Camera Rig

Using the unity editor 2 scenes have been built for the game to be played in. a main menu scene and a song scene. Both scenes contain the prefab from steamVR called camera rig. This is a prefab that contains a main camera as well as 2 controller objects for the left and right hands. The camera in the camera rig displays directly to the screens in a VR headset giving slightly different views in both eyes so that players see a 3D environment. The camera rig also has scripts attached that move the camera and controllers the same way the player moves in the real world. The controllers are given transforms relative to the camera so their position in the scene doesn't matter. The position of the camera is the default position for the player when they load into the scene. steamVR will take this point to be the relative origin and the camera will move from it the same way the player does.

The default models for the controllers have been changed to the gun and sword used in the song scene rather than the vive controllers they default to. They have had their rotations adjusted such that the controllers are lined up with the handle of the weapon in each hand, this way the gun and sword feel natural to hold and move the way the player would expect.

In the main menu scene the default vive controller models have been kept as they're the commonly accepted default for interacting with menus in VR games. In this game the VR controllers have laser pointers coming out of them that shoot directly out of the top hole in the controllers. They track to the angle of the controllers and are light blue. Upon clicking the trigger they turn green so the player has some feedback to their button presses. Using this pointer the player can use the trigger like left click on a mouse to activate UI buttons and choose a song to play.

3.3.2 Other Objects

In the main menu scene the only objects inside it besides the camera rig is an empty game object called main menu, the UI canvas and the event system for activating buttons. This empty game object has a component which is the main menu script and determines the default spawn location for buttons within the UI canvas. Within the song scene there is only one gameObject inside it besides the default light and the camera rig with the changed models and that is the audio source which is in the very center of the scene and plays the audio clip given to it by the game controller when activated.

4. Conclusions

4.1 Critical Reflection

While a full rhythm game was initially proposed for this project, one wasn't fully built as due to time constraints and severe hardware limitations a fully featured rhythm was simply not feasible for this project. Instead more focus was put onto the algorithm for generating blocks to move towards the player, which could be applied to other rhythm games as a generic system for generating levels. A major issue that occurred was the attempt to script extremely complex movement for the blocks in the rhythm game that resulted in their speeds either being much too high or the blocks stopping altogether.

Another issue that occurred during this project were problems involving the use of equipment. The complexities of VR made it difficult to play test features for the game properly for a lot of development so if this project were to be repeated again the use of equipment would need to be planned in more detail before the start.

In order to meet the deadlines for this project it has needed to be simplified so blocks now move using the animation system in unity and the block spawning algorithm has been simplified. This ensures that the blocks move towards the player properly.

The primary issue at fault during this project was a lack of organisation and planning for things going wrong. Every issue that occurred made it more difficult to finish the project and so it had to be simplified and unnecessary features cut. Thanks to the simplification of the project it has been finished with an automatic level generation system still intact that uses wavs and midis.

The game created has some models within it but not all objects have custom models made, such as the bullet or blocks. The art for these elements was never received so they couldn’t be implemented. The scenes also have a skybox based on space which gives a pleasant background to the game. The game could be improved further with more custom models and visuals in future.

The game could be improved with the addition of minor sound effects and haptic feedback through the controllers. For example a sound and controller vibration when the gun is fired or a controller vibration when the sword destroys an object.

The algorithm for firing and reloading the gun works with the steamVR inputs, allowing the player to fire the gun with the trigger and reload with the grip button. It has also been given an ammo indicator on the side in the form of blue blocks which turn grey one by one when fired. When the gun reloads they turn blue again one by one over 2 seconds. This allows the player to see at a glance how much ammo they have and if their gun is currently reloading as they don’t have time to read a UI during a fast paced game like this. The bullets are scripted to move forward at a constant speed and destroy themselves if they get too far away from the player. This avoids them stacking up on each other throughout the game and causing lag after too many have been fired.

4.2 Conclusions

Overall this project has completed its core goal of greeting a level generation system for a rhythm game, however the rhythm game that was supposed to be built around it was forced to be cut down somewhat. While the rhythm game itself is basic, this level generation algorithm could be applied to other rhythm games in the future with more polish, allowing them to have an infinite number of levels available to them.

The level generation algorithm is very intuitive for a player to use as they only need to put a midi file and a wav file of the same name into the songs folder for that song to be available to choose from in the game. This achieves the goal of simplicity and usability that all games should work towards and maintains the wide appeal of rhythm games that is their biggest strength.

In the future this rhythm game could be expanded upon given more development time. The movement algorithms for the blocks could be given some slightly random element to make them more unpredictable and their movement overall could be given more complex behaviours so that they track the player better. As stated above, sound effects and haptic feedback could be given to the game in order to give more weight to the objects the player is controlling inside it. The game could also be given a level editor, allowing players to make their own levels in the game that aren't reliant on the algorithm if they want to try designing levels like in a more traditional rhythm game.

This project has been met with many setbacks but in spite of them the core concept of it has endured to completion. This game provides an automatic level generation system that allows users to enjoy an unlimited set of levels that will be made to any music they desire.

References

https://store.steampowered.com/app/1574850/Music95/
https://beatsage.com/
https://www.nicksypteras.com/blog/aisu.html
https://www.youtube.com/watch?v=xTFyPkRgTUY&t=271s&ab_channel=RTGameRTGameVerified
https://www.youtube.com/watch?v=kilWnla1bwk&ab_channel=Ant%26Dec%27sSaturdayNightTakeawayAnt%26Dec%27sSaturdayNightTakeawayVerified
https://github.com/melanchall/drywetmidi