Game Design Document
“Music Agency in Video Games”

Group Members (Students)

Jean-Francois Retief
2458318

Malakai Braam
2457821

Abstract

Among the most popular features in video games, customization is one of the most sought-after features. Customization is one of the most clear and easy examples of how to give a player agency in a video game. Quite often, players are given the opportunity to express themselves or to tweak their experience according to their taste, by customising their character, their playstyle or at the very least their settings. However, when it comes to music in most games, players usually get a volume slider. ***This project aims to test how well audiences respond to a music-selector customization menu, which allows them to select what genre of music they want to listen to at certain points (states) in the game***. When the game enter's a specific state (such as combat, exploration, cutscene, etc.) different music tracks will, and depending on what options they selected - a different genre of music will play. This project aims to be a showpiece of how such a system can improve player agency and subsequently their personal experience with the game. Instead of just allowing players to just change the volume or let content creators mute copyrighted music, systems such as this one can be implemented to replace music with other tracks.

1. Introduction

This document aims to encapsulate the vision of the game, as well as document processes and notes during the development of the game. Topics, such as the need for music agency in video games, the planned technologies and methods, a general overview of the game, notes on development of prototypes and the final product, and notes on the research question. For more detailed information on the narrative and quest design of the game, please refer to the *Quest/Narrative Design Document*[1]. For more detailed information on the animation, visuals and sound in the game, please refer to the Animation, Visual and Sound Document[2].

2. The Why: the Need for Music Agency in Video Games

Customization is featured in many games in some form or another.

- Some games allow you to change your character's face and/or body (eye-colour, face shape, body-build, race, voice, etc.)
- Some games allow you to change your character's personality and/or stats (tallents, perks, backstory, etc.)
- Some games allow you to change your character's clothing or equipment (armour, weapons, items, etc.)
- Some games allow you to change various **settings** to customise your experience. These settings include:
    - Accessibility Settings - to help players with disabilities to play the game
    - Difficulty Settings - offers easier or harder experiences
    - HUD Settings (Heads Up Display) - to customise the UI experience of the game
    - Sound Setting - usually only subtitles and various volume settings (voice, sfx, music, etc.)

Customization is a popular feature to have in a video game, because it is an effective method of giving players **agency**. Character creators are a very popular mechanic, since they give the players **choice**, immediately (or soon after) after selecting “new game.” Once the player is established to have agency over the game, or in the game-world, they feel much more immersed in the game.

Now, why Music Agency? Well… look at the list above. Giving players agency and choice within a game is an effective method of immersing the player within the game and its world. Thus one can extrapolate that it would be wise to experiment with any and all systems within the game by adding player choice to it. So why not music customization?

As seen above, most game’s sound settings stop at volume. Some games allow the players to mute copyrighted music - enable “streamer mode” - but there are very few games that allow you to change the music (to different tracks) within the settings menu.

Some limited examples of music agency in video games are as follows:

- The player is able to switch **radio stations** in games like Grand Theft Auto, Saints Row and Fallout.
- Mixtape in some Saints Row games
- In Marvel’s Spider-man 2, the ambient “open-world music” can be set to “ACT 1”, “ACT 2” or “ACT 3” upon completion of the main story.
- Brutal Legend uses a metal inspired soundtrack where players can unlock new songs that will play through gameplay by discovering hidden objects.
- Games such as Journey and Rez make use of music agency by having the player’s progression and actions determine sound effects and music in the game.

3. Technologies and Methodologies that we plan to utilise

3.1. Hardware

- Dell G15 5511 Laptops with an Intel® Core™ i7-11800H (11th Gen) processor and 16GB RAM

3.2. Software

- Unity - Editor Version: 2020.3.30f1 (or later version)
- Source Control (and backups): Github
- Sound and Music: Ableton Live.
- Animation and Asset creation: Blender and Krita
- Communication: Discord, Microsoft Teams, Whatsapp & email
- Documentation: Google Docs

3.3. Technical details on how we plan to implement the music system within the game

➔ Unity has inbuilt GameObjects that are called “Audio Sources” and these can easily be manipulated with code.

Within the code, we will implement a state machine that will track what “scenario” the player is in, and subsequently what TYPE of music to play.

```
public enum State
{
   Walking,
   Running,
   Flying,
   COMBAT,
   WinScreen,
   DeathScreen,
   CUTSCENE
}
```

```
public enum MusicTYPE
{
   Default,
   Western,
   Classical,
   Acoustic,
   AIgenerated,
   Retro,
   AfricanLocal
}
```

Figure 1: Possible code to use (subject to change if better method is found)

Using Unity’s UI toolset, we will create a menu that will control which individual audio source will play.

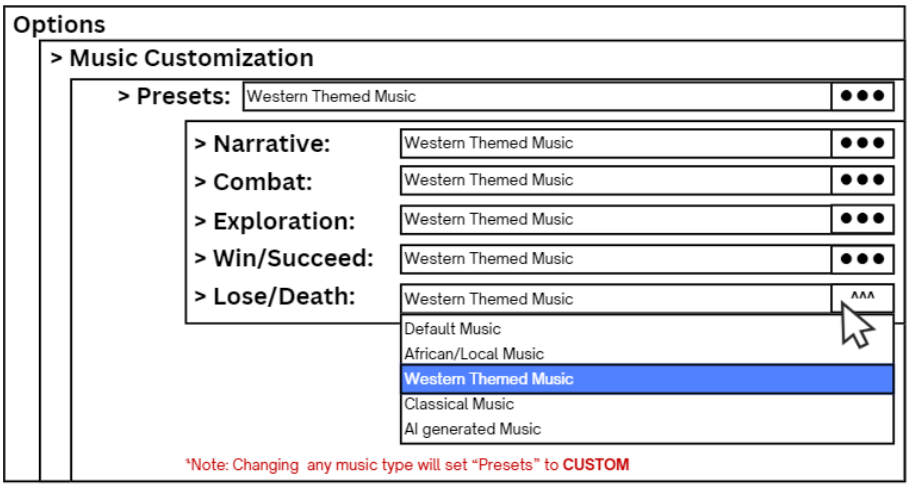


Figure 2 (also appeared in Expression of Interest): Wireframe prototype of Customization Page (in Options Menu, and will be automatically opened throughout the game to encourage players to change these settings (made by Jean-Francois Retief)

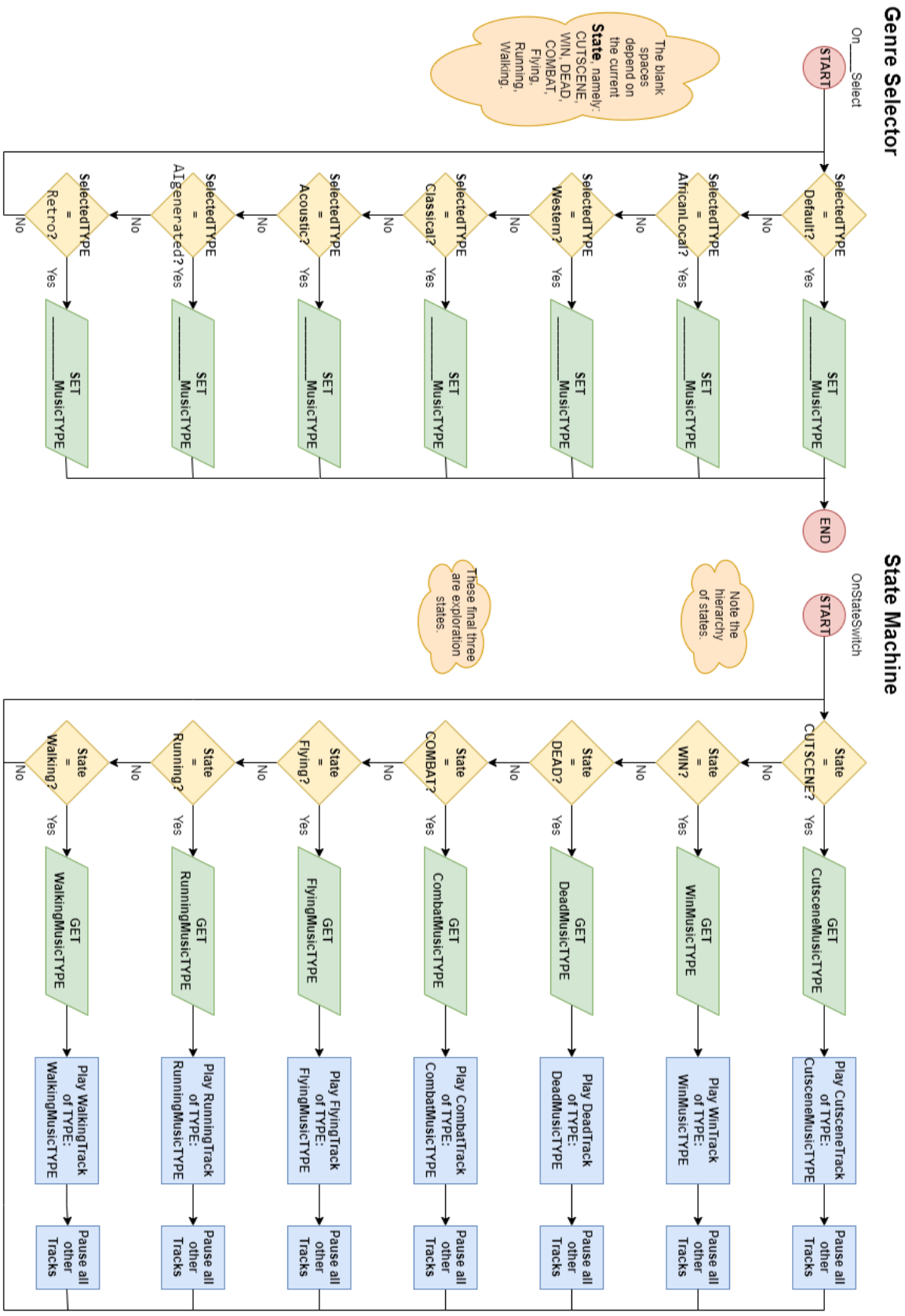


Figure 3: Flowchart of various systems

The currentState will depend on the current context the player finds themselves in, and the musicTYPE will be selected by the player in the music customization menu.

```
//more sources
public AudioSource RetroCombatMusic;
//more sources
if(currentState == State.COMBAT
      && selectedTYPE == MusicTYPE.Retro)
{
   RetroCombatMusic.Play(0);
}
else { RetroCombatMusic.Pause(); }
```

Figure 3: Possible code to use (subject to change if better method is found)

➔ To create the music we will feed through live recordings of instruments such as guitar and or keyboard through to Ableton as well as use the sample sound instruments that Ableton provides as well.

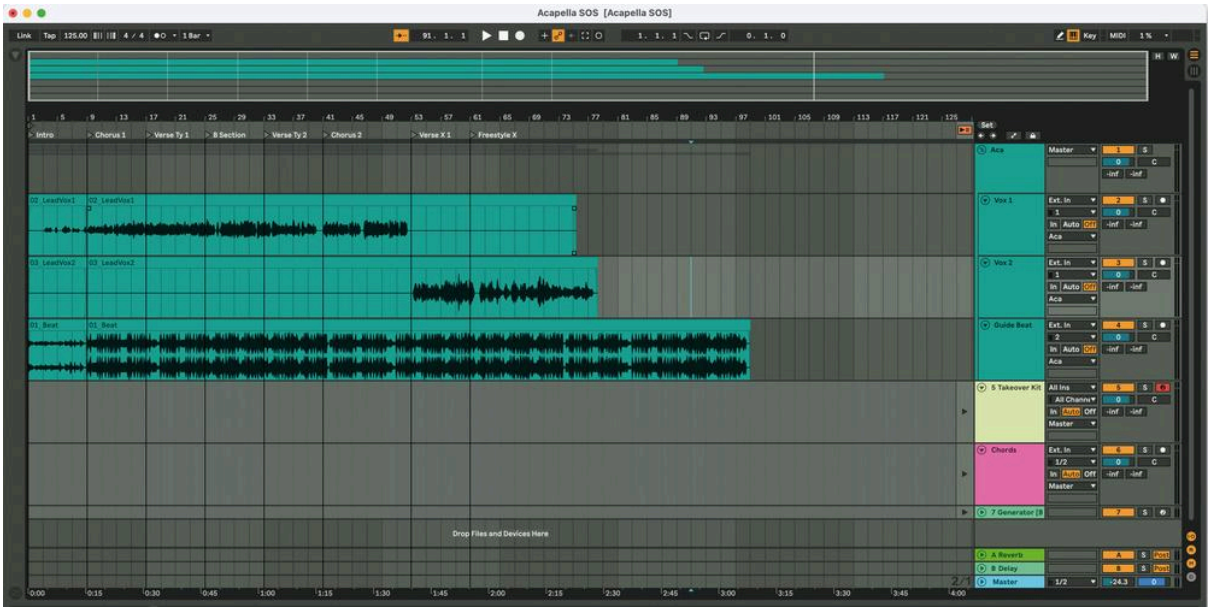


Figure 4: Example image of Ableton project layout.

There will be an array of songs that will be created to accommodate a variety of genres that will be no longer than a minute each in order for them to be looped through playing the game.

For the AI generated music, either we will prompt AI for chord progressions, a bpm (beats per minute) and melody for the song or make use of a music generation website such as Mubert or Soundraw to create an entirely unique song with a prompt of style.

4. Game Overview

Game Title

4.1. Genre & Subgenres

This game is an action-adventure open world game. When not on a mission, the player will freely explore a small town. The world is filled with main storyline quests, side quests, enemies and collectibles.

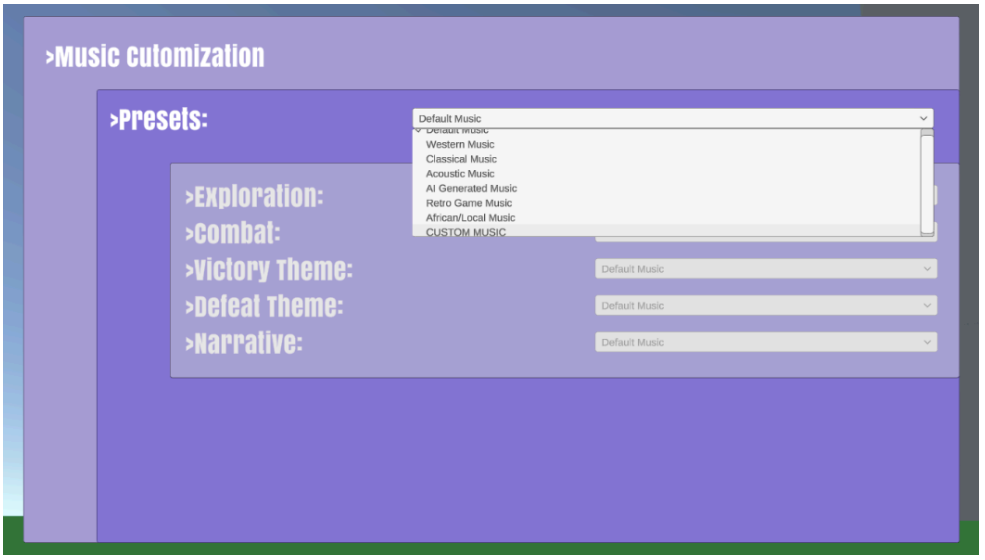
4.2. Our Plan / Hypothesis / Design Goals

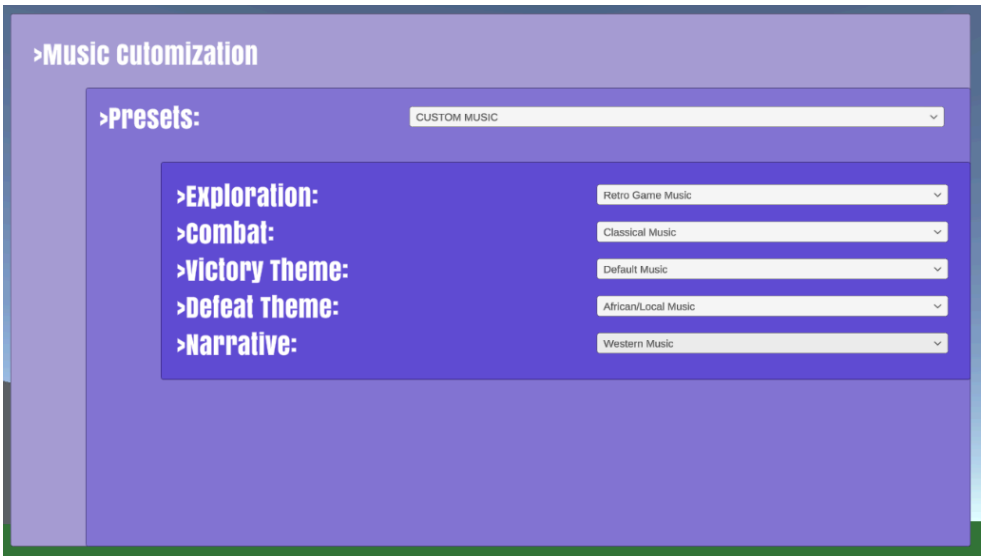
We wanted to make a small open-world game, since most of our favourite games are in this genre, and we've yet to create a game in this genre in our careers as Game Design Students. In the narrative, we integrate the question of "How empathy impacts the world around us."

Our plan is to create the music system first, then build the open-world game while testing the music system's interactions with the game-state machine during development. We will split the workload into two rolls, the game developer / coder and the game artist / asset creator. I.e. One of us will be responsible for the game systems, balance and logic, while the other will be responsible for visuals, sound, music and animation.

5. Early Prototypes

Music Customization Menu





Figures 5 & 6: Image of in-game Music customization menu

The underlying code for this menu, as well as the music customization system and its connection to the game's state management system is very similar to the planned code in section *3.3. Technical details on how we plan to implement the music system within the game*.

The current version of this UI is sufficient for now, but might change in future if player feedback deems it necessary.

State Machine

```
if (CutscenePanel.active == true)
{
    currentState = State.CUTSCENE;
}
else if (DeathScreenPanel.active == true)
{
    currentState = State.DeathScreen;
}
else if (WinScreenPanel.active == true)
{
    currentState = State.WinScreen;
}
else if(inCombat)
{
    currentState = State.COMBAT;
}
else if(playerMovementScript.inFlight == true)
{
    currentState = State.Flying;
}
else if (Input.GetKey(KeyCode.LeftShift))
{
    currentState = State.Running;
}
else //Default state
{
    currentState = State.Walking;
}
```

```
public MainQuest003 mainQuest003;
public MainQuest004 mainQuest004;
public MainQuest005 mainQuest005;

[Header("Side Quest Scripts")]
public SideQuest001 sideQuest001;
public SideQuest002 sideQuest002;
public SideQuest003 sideQuest003;
```

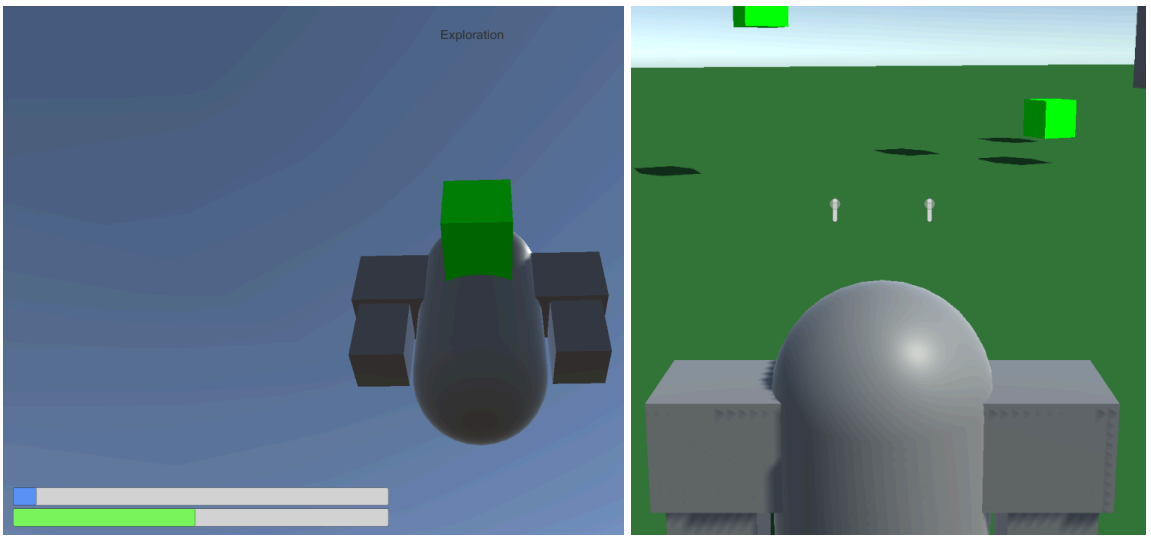
```
SpeedControl();

if(!grounded && Input.GetKeyDown(KeyCode.Space))
{
    inFlight = !inFlight;
    if(inFlight && jumpClickCounter < 2)
    {
        Jump();
    }
}

if (inFlight)
{
    rb.useGravity = false;
    flightTimer--;
    FlightParticleSystem.SetActive(true);
}
else
{
    rb.useGravity = true;
    if(flightTimer <= flightMaxTime*60)
    {
        flightTimer += 0.5f;
    }
    FlightParticleSystem.SetActive(false);
}

if(flightTimer <= 0)
{
    inFlight = false;
}
```

```
Unity Message | 0 references
private void OnTriggerStay(Collider other)
{
    if (other.tag == "EnemyTrigger")
    {
        stateMachineScript.inCombat = true;
    }
    else
    {
        stateMachineScript.inCombat = false;
    }

}

Unity Message | 0 references
private void OnTriggerExit(Collider other)
{
    if (other.tag == "EnemyTrigger")
    {
        stateMachineScript.inCombat = false;
    }

}
```

Figures 7, 8, 9 & 10: Screenshots of various scripts handling states

We decided to keep the State Machine itself as simple as possible and subsequently it was decided to have a separate and dedicated Quest Handler. Within each quest, all of these states are possible, while one can image the quest handler as a second state machine, with states: mainQuest1, mainQuest2, sideQuest4, etc.

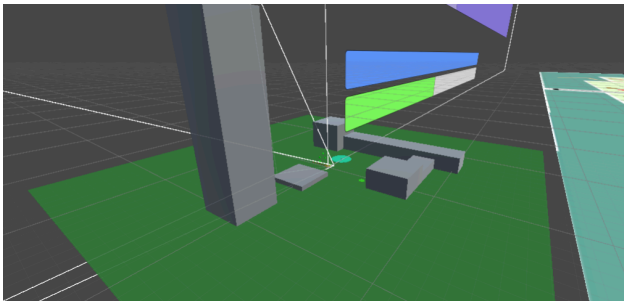
HUD



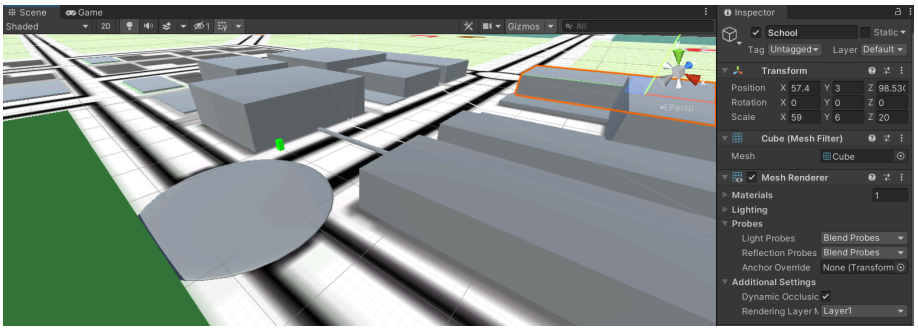
Figures 11 & 12: Screenshots showcasing early versions of HUD elements

In the above figures, one can see the green health bar (displaying the player character's current health), the blue flight gauge (displaying the player's current "fuel" level) and two crosshairs that display the trajectory of the two hand lasers (ranged attack).
These are simplistic prototype versions and are most likely going to change in the future. The crosshairs, especially, will need to be moved when the final player character model is imported to the unity project.

Dev Areas and Map



In order to test various systems, a simple greybox environment was created for testing purposes. The 2D map image was also imported to allow the team to agree on a general idea of world scale of the game's various buildings, roads, and other geometry.



Figures 13 & 14: Screenshot of dev area and imported map used to get measurement

Combat: Player

Figures 15 & 16: Images displaying the ranged and melee attacks, respectively

In this early prototype, before the final visuals are added, the actual triggers that are used to detect collisions are used as visual aids to see the range of each attack.

Figure 17: Inspector view of combat script attached to the playerCharacter gameObject

The player health, max health, melee damage and ranged damage are all public fields.

This was done to customise the different values seamlessly during future development, but also allow other scripts to change these values.

Thus, in future versions, scripts can be added that, for example: increase the player's max health, give the player an immediate health boost (health pack), upgrade their attack damage, etc.

Combat: Enemies

Enemy movement is handled by Unity's built in nav-mesh system, as seen below. The player object is the navigation destination, if the distance between the player and the enemy is below a certain threshold.

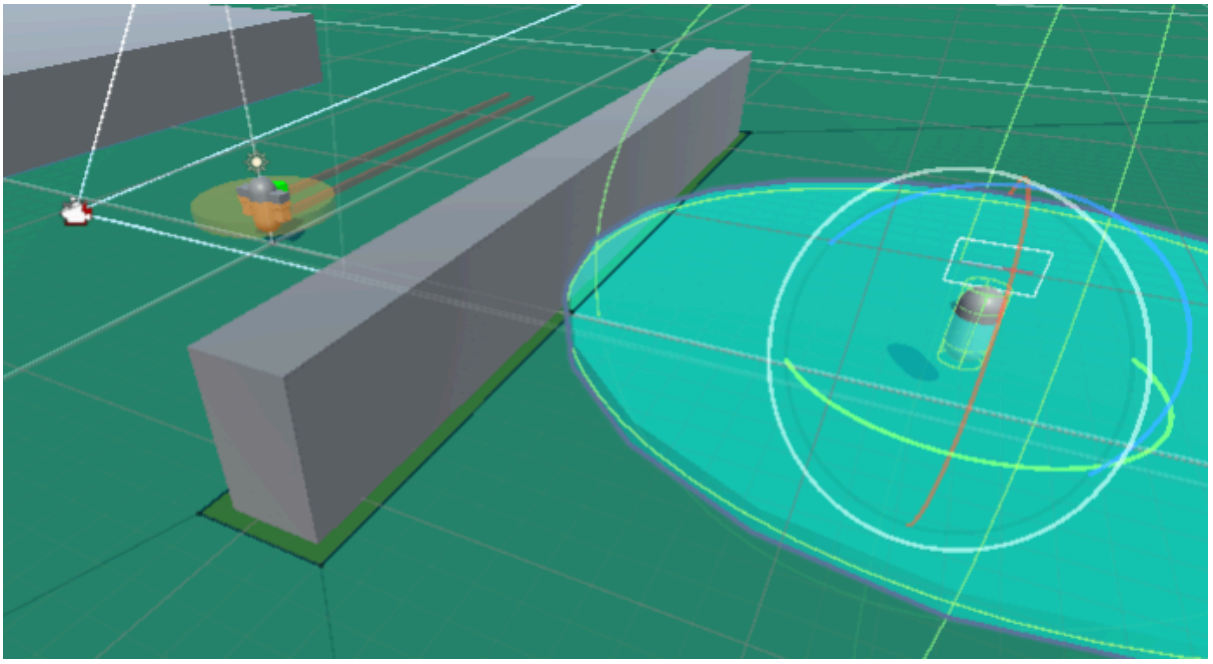


Figure 18: NavMesh, player, navigation obstacle (cube in centre) as well as an enemy (Nav Mesh Agent) and the CombatState trigger (blue) surrounding it.

The enemy health bar is not on the HUD, instead it is in worldspace, however it always points to the camera. The LookAt function allows the player to always see the enemies health.

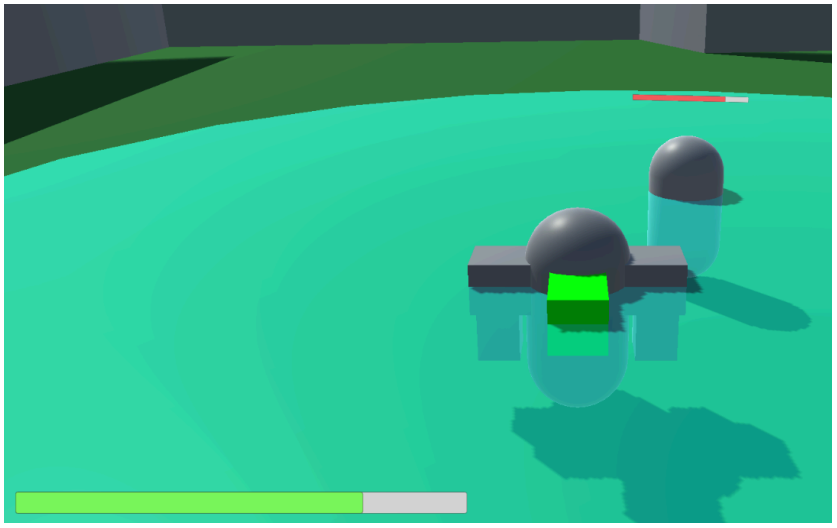
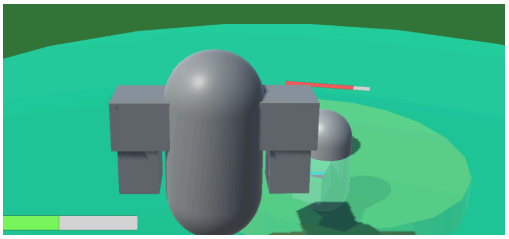


Figure 19: Enemy health bar

At the time of writing this, Enemies have two attacks: melee and ranged (projectiles).

The melee attack, similar to the player melee event, is handled by a trigger. Once the player enters the melee radius, they take damage and get knocked back.



```
private void OnTriggerEnter(Collider other)
{
    if (other.tag == "EnemyMelee")
    {
        playerHealth -= enemyMeleeDamage;
        Vector3 direction = transform.position - other.transform.position;
        Rigidbody rb = gameObject.GetComponent<Rigidbody>();
        rb.AddForce(direction.normalized * knockbackMultiplier);
        rb.AddForce(Vector3.up * knockbackMultiplier);
    }

    if (other.tag == "EnemyProjectile")
```

Figures 20 & 21: The player experiencing melee damage and knockback and the underlying code

As for the ranged attack, unlike the player's lazer attack, enemies will shoot projectiles. A projectile will, when it hits the player object, damage the player and disappear.

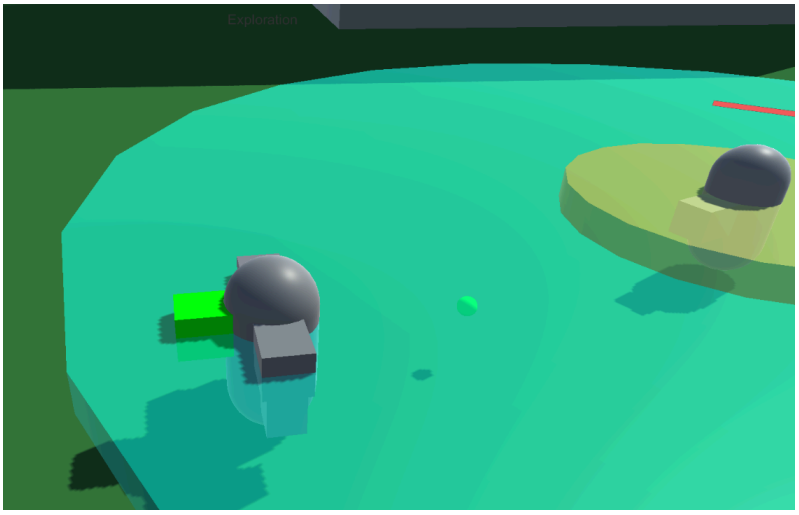


Figure 22: Projectile flying towards player

Level/World Design

To start off, we created a 2D map in draw.io while planning out the setting.

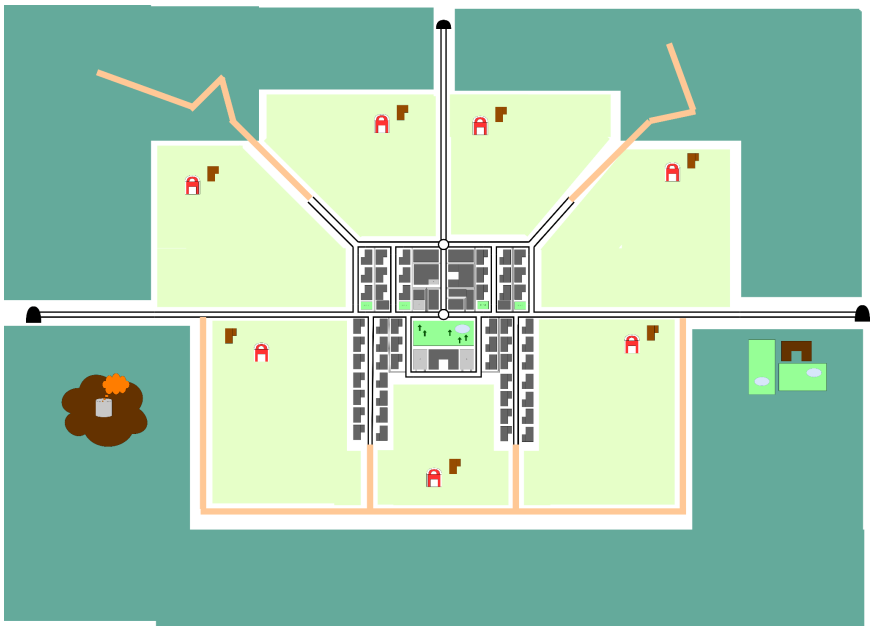
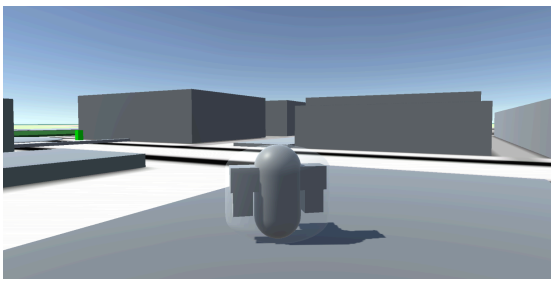
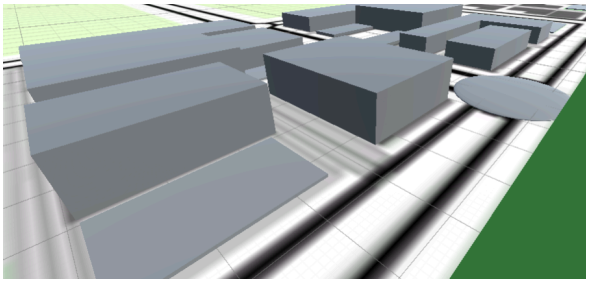


Figure 23: 2D Map Prototype

Then we imported the map-image into unity and started to put grey-boxes onto the map, to get a feel for the 3D environment. This was done to plan out the measurements of each building, as well as giving us a to-scale test-scene for quest design (see Quest document for more info[1]).





Figures 24 & 25: Grey-boxed Unity scene, used for planning out the measurements of each building

When it comes to early versions of the quests, we implemented invisible walls, triggers and enemy prefabs in the greybox scene, along with test videos instead of cutscenes and text-based dialogue (instead of recorded voice lines)

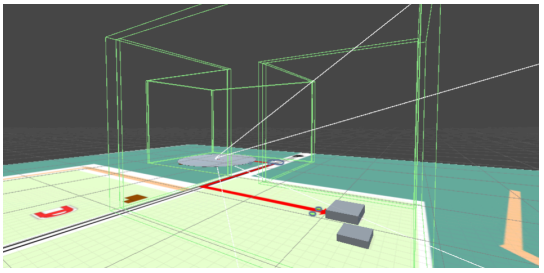
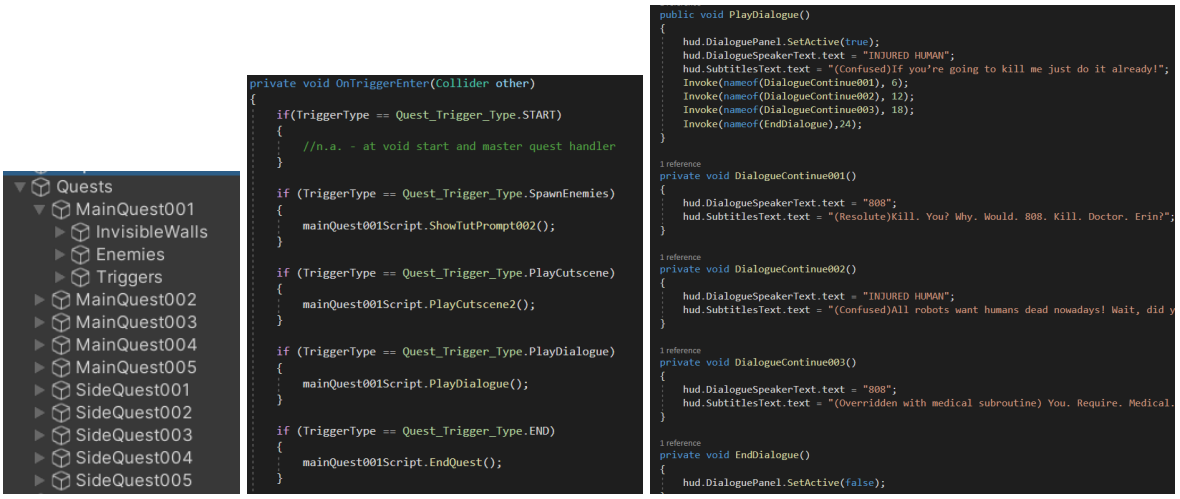


Figure 26: Invisible walls around MainQuest 1

Simple and reusable code was used for each quest. The quest handling is separate from the main state machine. Early testing indicated that the state machine works automatically, even in context of a quest.



Figures 27, 28 & 29: Main Quest 1's objects and scripts

In the figures below, see early versions of the Quest-based UI. This includes the quest title, quest objective, tutorial hints, speaker name and spoken dialogue text fields (all of which change at certain points in the game).

Figures 30 & 31: Quest description and dialogue box

Kate uses a script, similar to the enemy movement script, to move. The main difference is that the navigation destinations are set points in quest 1, rather than the player's transform position.

Each quest, while sharing a similar script template (MainQuest00XScript, MQ1TriggerScript and MasterQuestHandlerScript) each quest can have wildly different scripts within that template.

When Importing world assets, we slowly placed them using the above shown map in figure 23 as a guide.

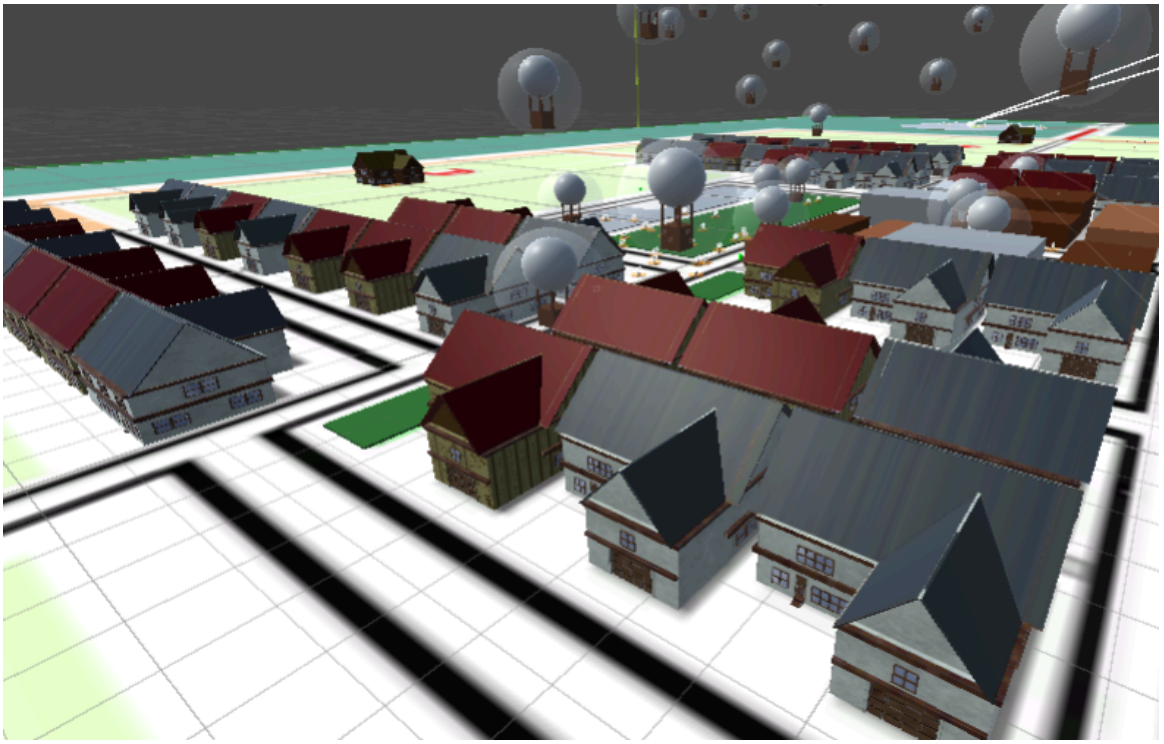
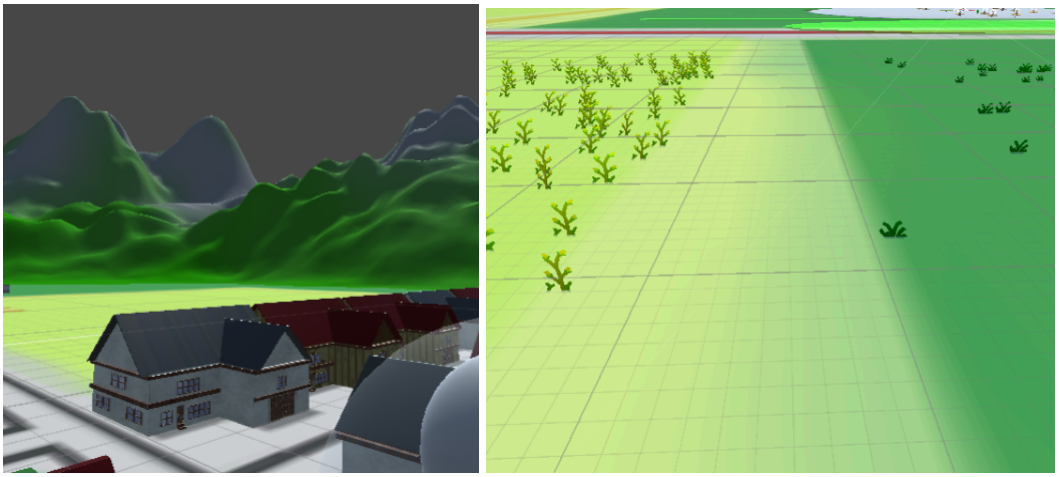


Figure 32: WIP game world

As for the terrain of the game, Unity's in-built terrain tool allowed us to easily place grass, crops and mountains without heavily impacting performance.



Figures 33 & 34: Images of mountains, crops and grass, respectively

Game Menu

Due to player feedback on the prototype of the game, the extra controls like “M: for music menu, and “C”/”E” for switching cameras where too much to keep track of, and thus they were changed. Camera switching is now dependent on the combat state, and getting to the music menu can now be facilitated in the in-game menu.

Figure X: Early version Runtime menu of the game

6. Final Game

World

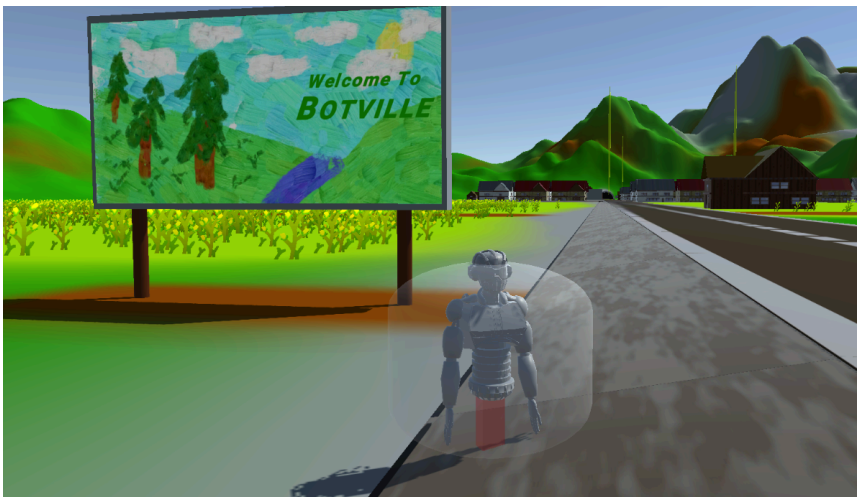


Figure X: Welcome sign

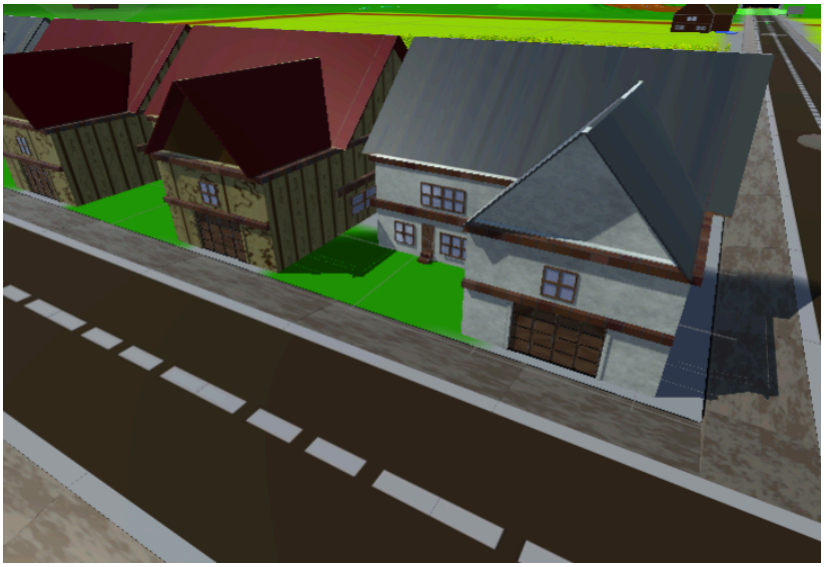


Figure X: Residential Street

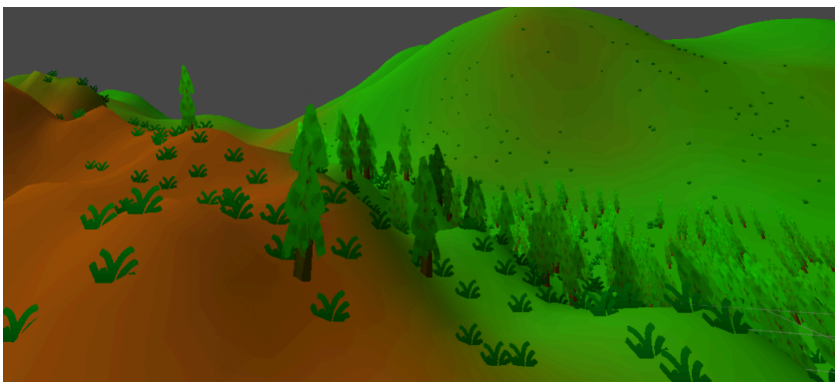


Figure X: Terrain (Woods)

[insert images of town hall]
Figure X: Town Hall

[insert image of Main Street]
Figure X: Main Street

[insert image of Farmhouse and Barn]
Figure X: Farmhouse and Barn

[insert image of Mansion]
Figure X: Mansion

[insert image of Map view, for comparison with original 2D map images]
Figure X: Top down view of Map

When all buildings are added, insert more images of 3D environment to show off how the game looks (for final November submission)

Narrative

Cutscenes

They are just videos created from still frames with posed characters and environments within blender. The videos are directly imported to the game. This method is relatively easy, but the trade-off is the increased file size of the game.

[insert image of Cutscene 1]
Figure X: First Cutscene

[insert image of Last Cutscene]
Figure X: Last Cutscene

Dialogue during gameplay

The story also unfolds during gameplay, where the audio and subtitles of the dialogue is played while the player is control of a playable character

[insert image of Dialogue instance]
Figure X: Dialogue instance during gameplay

UI

The music menu was the first to be created, and has remained relatively unchanged throughout development. The main menu has slowly gotten more buttons as features like saving and collectables were added.

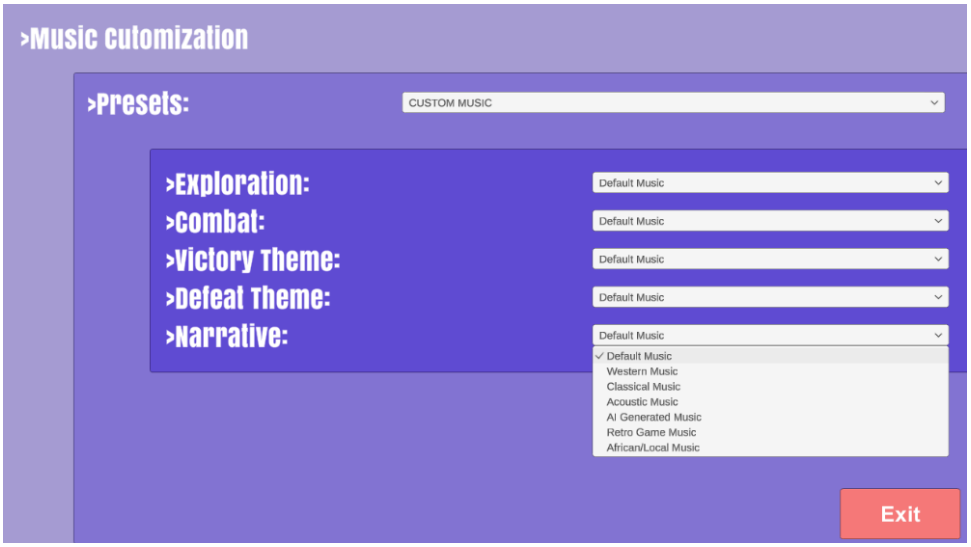


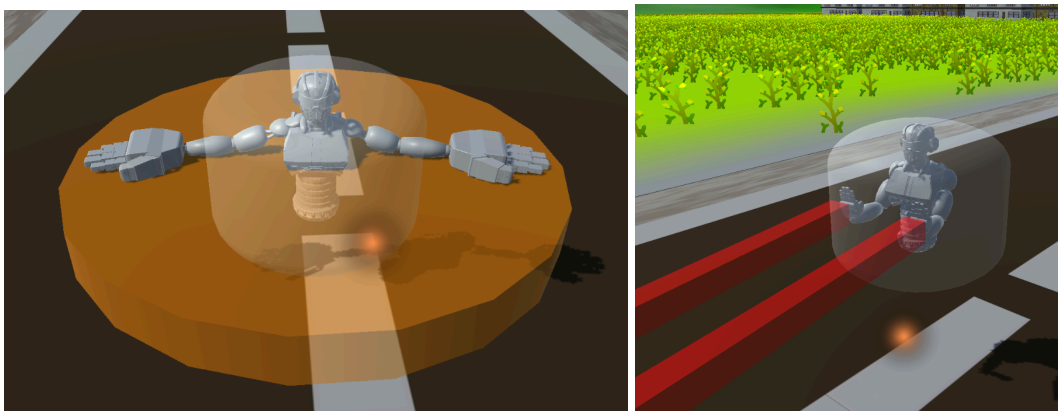
Figure X: Music Menu

The main menu's design remains simple and sleek.

Figure X: Main Menu

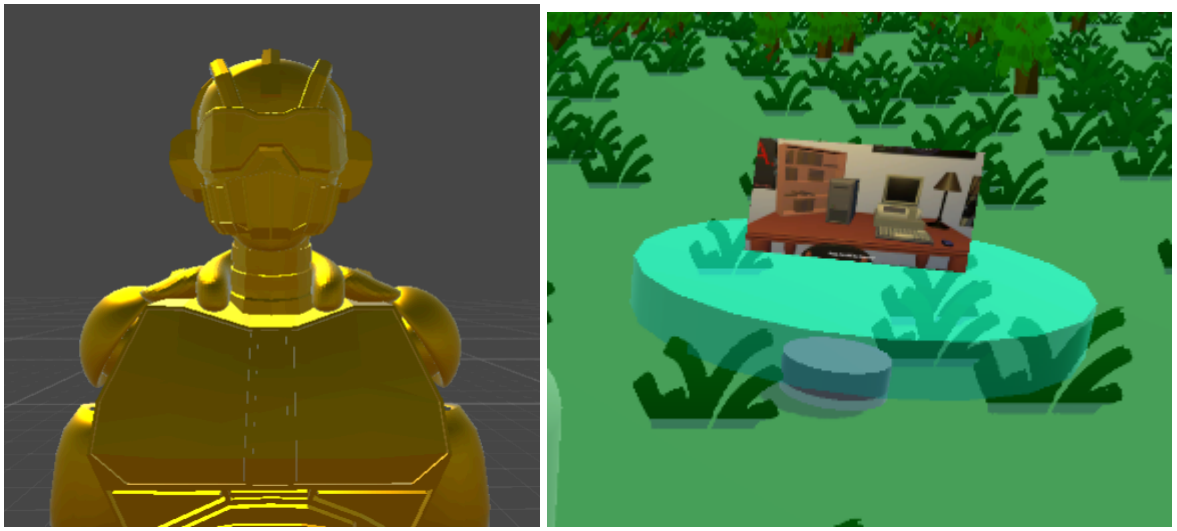
Combat

Combat interactions consist of a spin-attack (melee/CQC) and a double-lazer-blast (ranged). The camera and the player character's relation to the camera also changes when close enough to enemies - to make combat encounters more readable.



Figures X & Y: Melee and Ranged attack, respectively [replace in final submission]

Collectables



Figures X & Y: Reward for collecting all the collectibles and a screenshot of a collectable in the game world, respectively

The collectables are all references to other games we've worked on during our time at Wits. For example: the Games & AI game we made this semester, as well as our Game Design 4 game last semester.

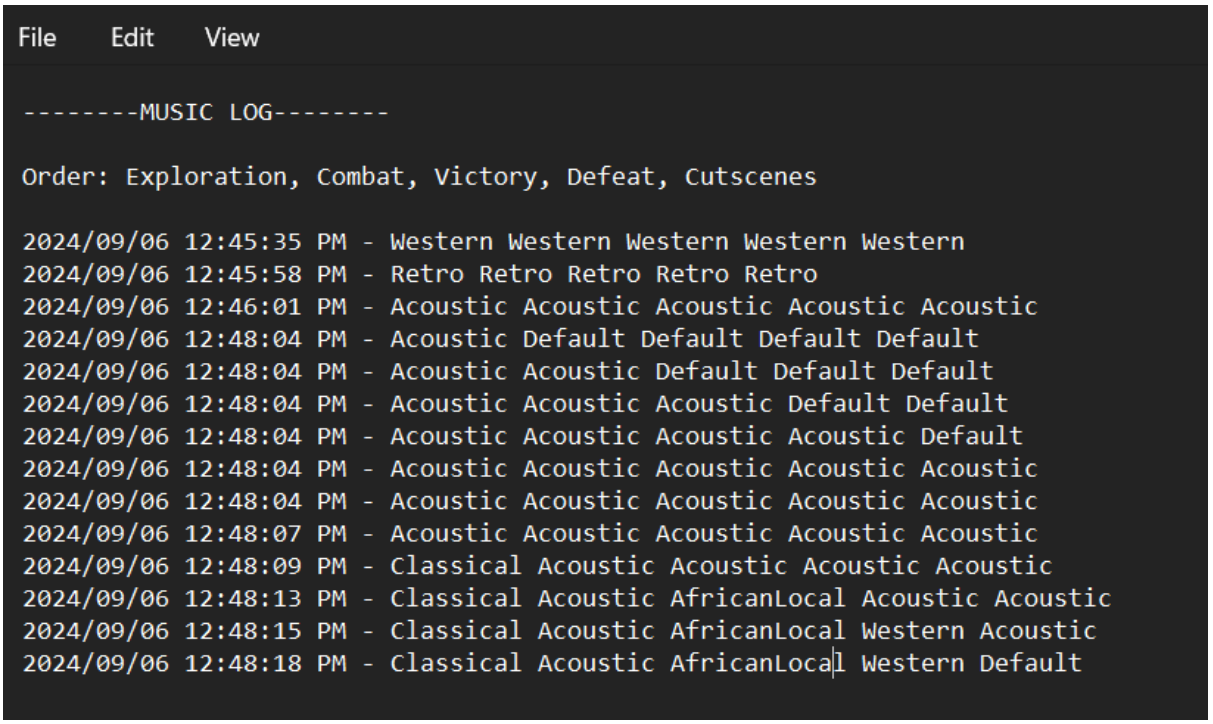
Figure X: Collectable menu screen

7. Research Notes

There are various methodologies that we will use to get data from players.

Firstly, we can det qualitative data from players via a questionnaire that we provide them during playtests. To see the actual questions, please refer to the Playtest Questionnaire Template Document[3].

Secondly, we collect quantitative data from players via a created **text file** that records the times when they switch their music genres and to what.



Figures X: Screenshot of DADP4MusicGame_Build002\ DAProject_MusicGame_Data\StreamingAssets\Music_Logs\MusicData.txt

References

[1] J-F. Retief, M. Braam, (2024), ‘Quest/Narrative Design Document “Music Agency in Video Games”’, Unpublished Internal Development Document

[2] J-F. Retief, M. Braam, (2024), ‘Animation, Visual and Sound Document “Music Agency in Video Games”’, Unpublished Internal Development Document

[3] J-F. Retief, M. Braam, (2024), ‘PlayTest Questionnaire “Music Agency in Video Games”’, Unpublished Internal Development Document