

H04

FYP Final Report

**Computer Music Game:
ZAGREC**

by

Kong Shuk Kwan, Lam Kin Ming, Ng Fiona K K and Tam Wing Kit

H04

Advised by

Prof. Andrew B. HORNER

Submitted in partial fulfilment

of the requirements for CPEG4901

in the

Department of Computer Science and Engineering

The Hong Kong University of Science and Technology

2021-2022

Date of submission: April 20, 2022

Abstract

Game industry has been well-developed in terms of genres, graphics, and playing modes. A variety of video game genres are going viral, ranging from sandbox to role-playing, from shooters to sports, they are always built with great graphics and intriguing features. If we look deeper into the mainstream market, however, music games are always built-in 2D with static playing modes. We could scarcely see a 3D environment implemented in a music game, not to mention a free-roaming one. In view of this, we have developed **ZAGREC** - a brand new computer music game combining a 3D free-roaming environment and numerous rhythm song stages. We aim to break the stereotype and make a breakthrough in the music game industry.

In this report, we will give an overview of our game and walk you through the whole journey from brainstorming and ideation to the actual implementation and integration using the game engine - **Unity**.

Table of Contents

Abstract	2
1 Introduction	5
1.1 Overview	5
1.2 Objectives	7
2 Methodology	8
2.1 3D Gaming world construction	8
2.2 3D Game mechanics	16
2.3 Rhythm song stages	24
2.4 Game settings	37
2.5 Integration and Testing	46
2.6 Usability Testing	54
2.7 UI/UX Improvements	57
2.8 Evaluation	65
3 Discussion	66
3.1 Customized settings	66
3.2 Difficulties encountered	66
3.3 Future Directions	68
4 Conclusion	69
5 References	70
6. Appendix A - Literature Survey	71
6.1 The market value of the game industry	71
6.2 Determinants of game enjoyment	72
6.3 Automated beat mapping technology	73
6.4 Review on other similar music games	74
7. Appendix B - Meeting Minutes	78
7.1 Minutes of the 1st Project Meeting	78
7.2 Minutes of the 2nd Project Meeting	82
7.3 Minutes of the 3rd Project Meeting	91
7.4 Minutes of the 4th Project Meeting	93
7.5 Minutes of the 5th Project Meeting	94
7.6 Minutes of the 6th Project Meeting	95
7.7 Minutes of the 7th Project Meeting	99
7.8 Minutes of the 8th Project Meeting	100

7.9 Minutes of the 9th Project Meeting	101
7.10 Minutes of the 10th Project Meeting	102
8 Appendix C - Project Planning	106
8.1 Distribution of Work	106
8.2 GANTT Chart	107
9 Appendix D - Required Hardware and Software	108
9.1 Hardware	108
9.2 Software	108

1 Introduction

1.1 Overview

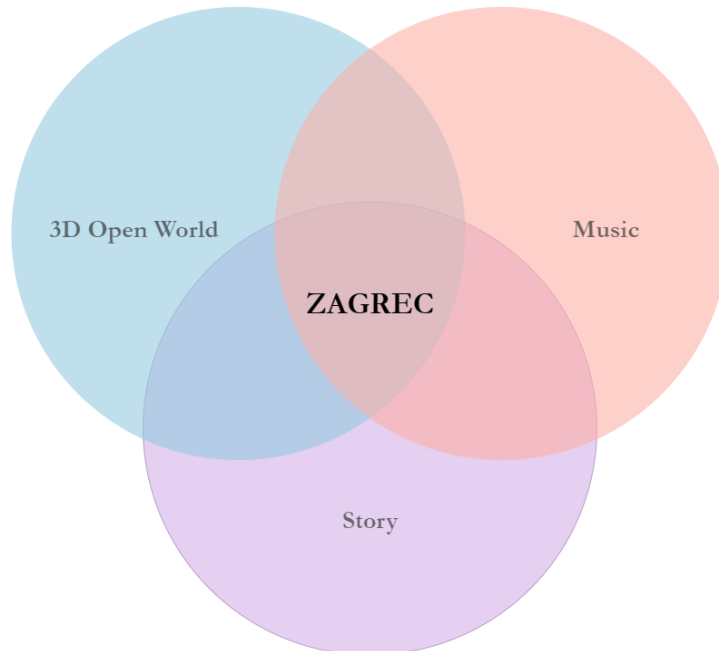


Diagram 1.1: ZAGREC overview

ZAGREC is a 3D open-world game that consists of the main storyline with numerous stages. It is a third-person single-player game in which the player will see the backside of the characters. A mythical story will be told along with the game's progress. The player is expected to search for clues and elements that are crucial to reaching the goal.

The motivation for building **ZAGREC** mostly comes from other games on the market. There are countless excellent 3D open-world video games with breathtaking storylines, such as Grand Theft Auto (GTA), NieR, and Yakuza. In the meantime, there are also some spectacular and famous music games out there like Cytus, Deemo, and our childhood memory - Taiko Drum Master. The game design of different games is always diverse, from pixel art graphics to realistic 3D world, from single-player mode to multiplayer mode; Yet, all of them have some unique features or highlights that make them well-known and go viral. We are then inspired to build a game that has our own uniqueness with a reflective storyline. Most of the popular music games on the market are static instead of interactive and dynamic. To make a breakthrough in music games, we would like to build **ZAGREC** by merging the music elements into a 3D open world, with an impressive storyline. Building a free-roaming virtual world not only provides a more interactive and mesmerizing game experience to players but also allows them to explore every little detail of the game, even the "Easter eggs" - hidden messages of the backstory.

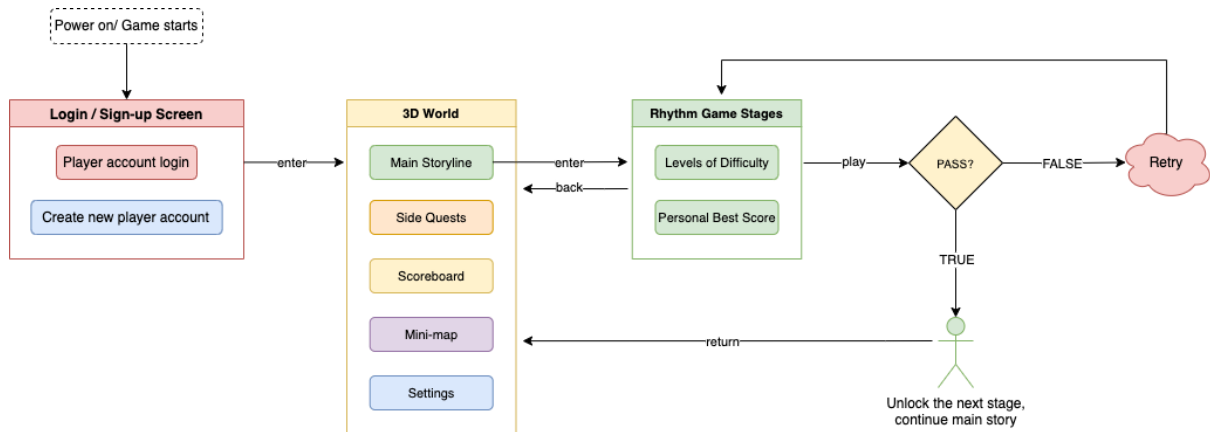


Diagram 1.2: ZAGREC basic game flow

ZAGREC is an integration of a 3D free-roaming environment and 2D rhythm game stages, where players have to pass every rhythm game stage to unlock the next stage and continue the main storyline. The 3D world is where the main storyline is being told, and it also consists of numerous functions and buttons, including a mini-map that helps players navigate in the gaming environment, a ranking board to show the ranking of all players, as well as the settings button that allows players to adjust the game settings and exit the game. Players will also complete all side quests in the 3D world. In this report, we will focus more on explaining the four main parts of **ZAGREC**: 3D free-roaming environment, 3D modelling, rhythm game stages, and the integration of all elements.

1.2 Objectives

The main goal of this project is to design and develop an open-world music game, which aims to alter the static impression of conventional music games that are built with linear and structured playing modes. During the project development, we will focus on the following:

1. To provide a user-friendly game interface

The game will be designed to be nonlinear which integrates with a harmonic colour scheme to display all the 3D game objects in the virtual world.

2. To develop rhythm game stages with high accuracy

Beat mapping is an essential technique to build **ZAGREC**. Since every song has its own map of notes, there is always a unique pattern of player actions in different songs. We will develop the song stages with high accuracy on the rhythm and beats so as to provide a more immersive and satisfying game experience to players.

3. To develop intuitive player controls

The control mechanism will be implemented by the Unity Input System package that uses any kind of input device, so players can then control their character with a keyboard and mouse easily.

4. To build a database

The database will be built to store and manage the player information and the corresponding game data such as the score or grade they obtained in every stage, as well as all their assets in the game. The database allows players to load and continue the game at the point they stopped last time, all the game progress will be saved and players do not have to start a new game every single time as long as they log in to the same account.

2 Methodology

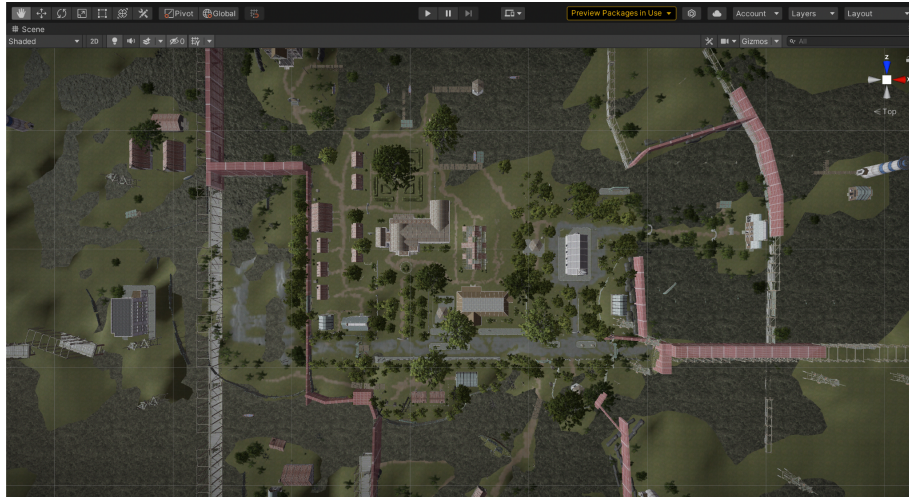
2.1 3D Gaming world construction

- **Main story**

The timeline of the game will be set at the doom days, where the environment would be an abandoned city. The main characters include 1 artificial intelligence robot (AI robot) and 2 human characters, in which the AI robot will be in the role of guidance, leading the 2 human characters to figure out the path and ways to strengthen themselves throughout the journey. The player will be one of the human characters, and the goal of the journey will be to find out how the world ended and try to revive the world. There will be some remaining data or clues saved in the AI robot, which are relevant or crucial to reaching the goal. The story chapter design with respective themes is as follows:

Chapter	Title	Chapter Description
Prologue	Eschatos	Briefly introduce the background of the game, including only one tutorial song stage in this chapter.
1	Prometheus	Tell the story of why the characters are awakened and how it starts.
2	Clotho	This is the beginning of the main storyline, which explains human activities that happened in that place
3	Lachesis	The middle part of the story, which explains more about human culture and how it leads to the end of the world.
4	Eos	This is the turning point of the plot, where some hope emerged during the journey, the main characters found out that music can help to purify this ruined world.
5	Elysian	The characters follow the light and start reviving the world. They have found a little piece of fine land that is naturally recovered. They want to expand the land and accelerate the purification.
6	Zagreus	When the world has reached a suitable living condition, there shall be human beings or living creatures, which achieved the main goal - reawakening the world.

- **3D world environment**



Diagrams 2.1.1 & 2.1.2: 3D world scenes in ZAGREC

Players would enter the 3D world right after login. The world is full of post-apocalyptic elements to make it more like the world in doom days, while the whole environment will be gradually revived and energized in the later story chapters. Some sound effects like water and breeze have also been added to the scene, as well as the adjustment of shading, light, and shadow of the 3D game objects.

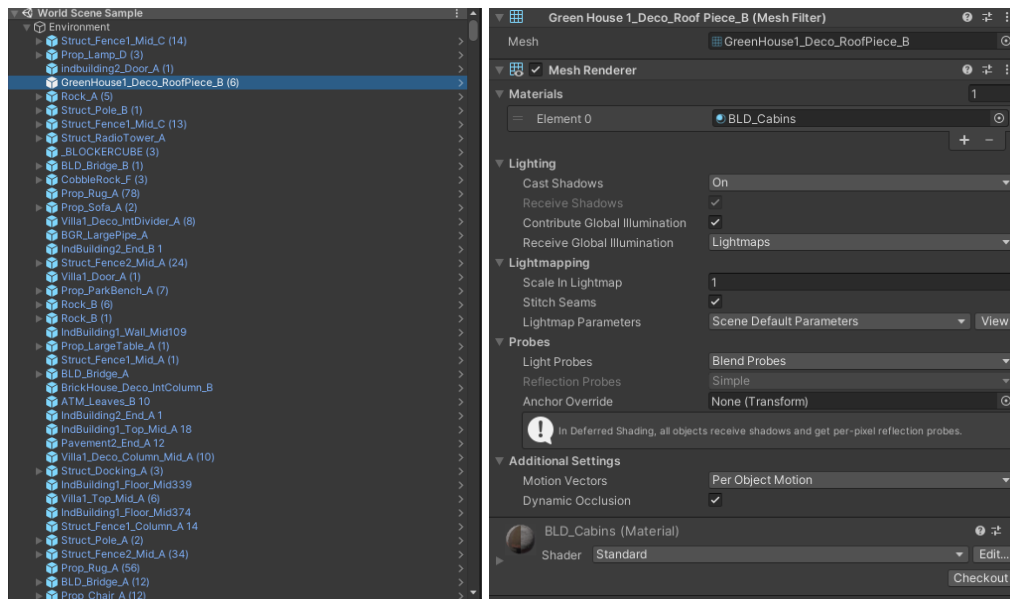


Diagram 2.1.3 & 2.1.4: 3D game objects and inspector in Unity

By using a **Mesh Renderer** component in **Unity**, which takes the geometry from the mesh filter and renders it at the position defined by the game object's transform, we have created and rendered different 3D game objects and placed them into our 3D world. The 3D world scene consists of hundreds of 3D game objects, with different materials and parameters for lighting and shadowing. We hope to find out the most natural and clear materials to be put on the 3D objects to present an immersive gaming experience to players.

- **Game graphics, lighting, and rendering**

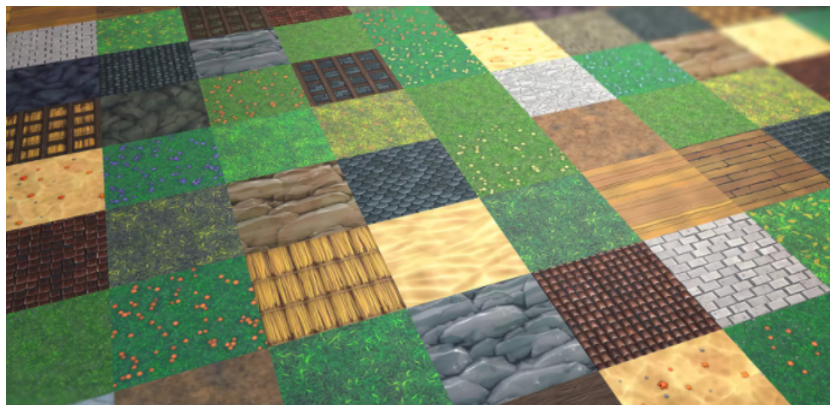


Diagram 2.1.5: Texture pack from Unity Asset Store [1]

We have fine-tuned some 3D models in terms of their texture, lighting, and shadowing. Some graphical art mods from the **Unity Asset Store** will be applied to obtain higher-quality graphics with fine details, ranging from the texture of a tiny object to the whole structure of the roads and buildings [1].

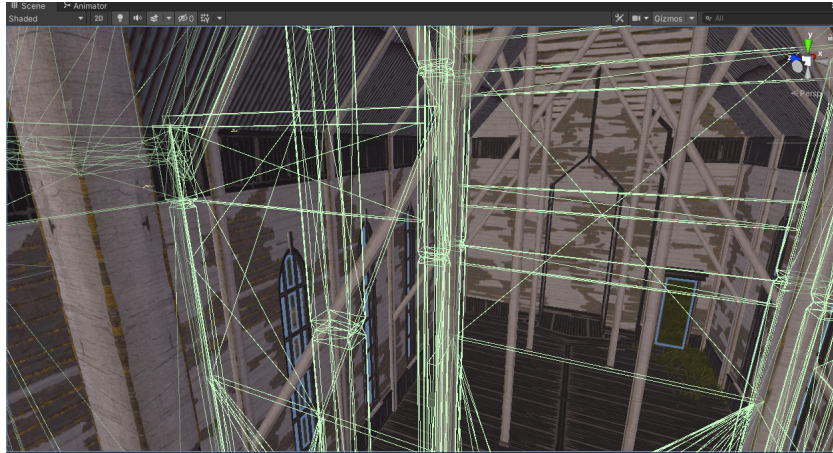


Diagram 2.1.6: Game building graphic mesh

For every building we created, mesh rendering is attached to the structure of each piece of a game object. The mesh will be affected by the lights to ray-cast and receive shadows, and projectors so that it can be drawn for all cameras or just for some specific camera.



Diagram 2.1.7: Game light and shadow

To achieve better lighting and shadowing performance, we have applied one of the computer graphics rendering algorithms - **ray casting**. Similar to ray tracing, ray casting will intersect the rays with every object in the scene and colour the pixel according to the object with the closest intersection. We also applied the global illumination technique, which is used to describe a range of techniques and mathematical models that attempt to simulate the complex behaviour of light as it bounces and interacts with the world. So we can simulate the world to be more realistic by updating complex scene lighting interactively.

- **Camera effect**

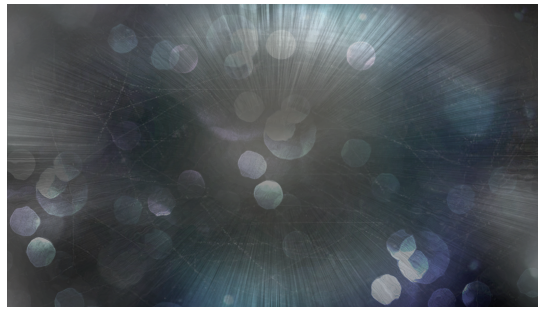


Diagram 2.1.8: Dirty Lens Flare Texture from Unity Store

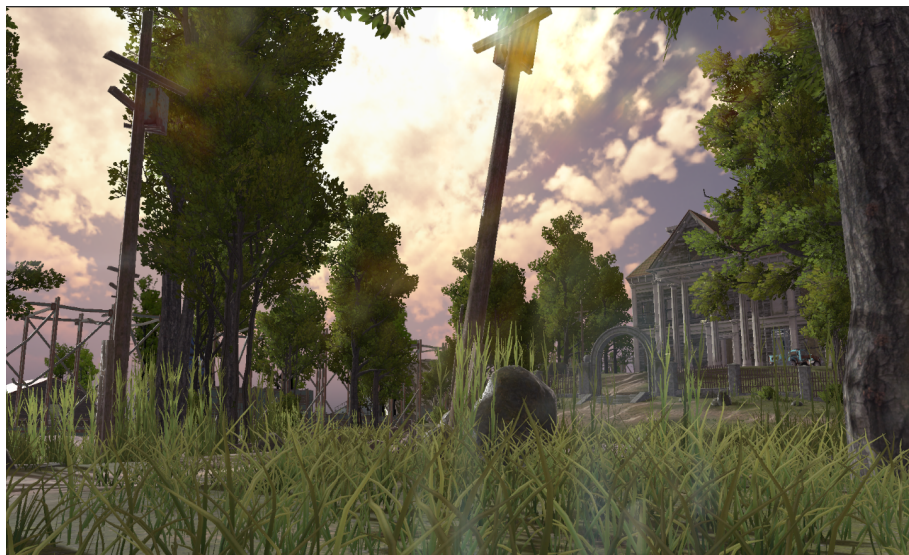
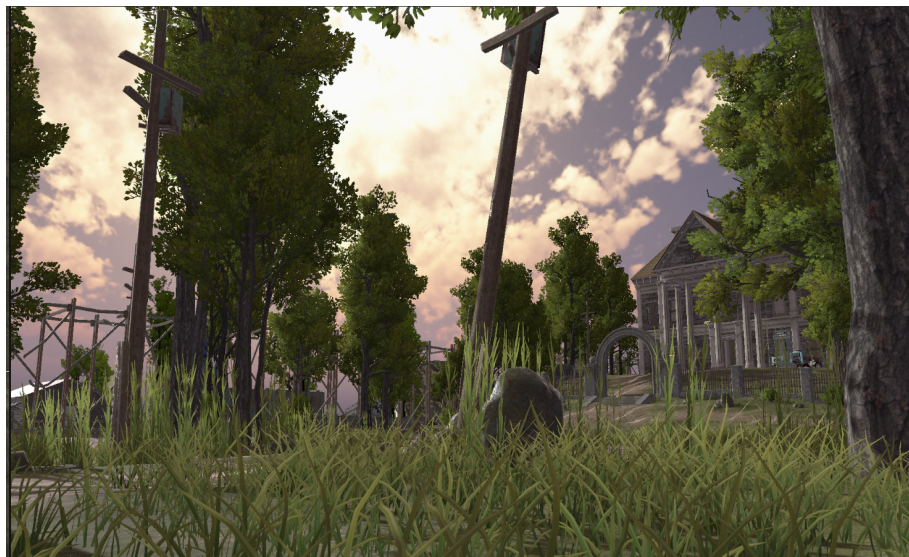


Diagram 2.1.9 & 2.1.10: Dirty Lens Flare disabled vs enabled

To create an overground world and fit the post-war atmosphere, we want to create some effects such as light flare, dirty lense to the third-person camera. By Unity **Graphic.Blit()** and **OnRenderImage()**, we are able to apply some custom blur, dirty lens flare, and shader with the camera's final image after a camera has finished rendering.

- **Game characters**

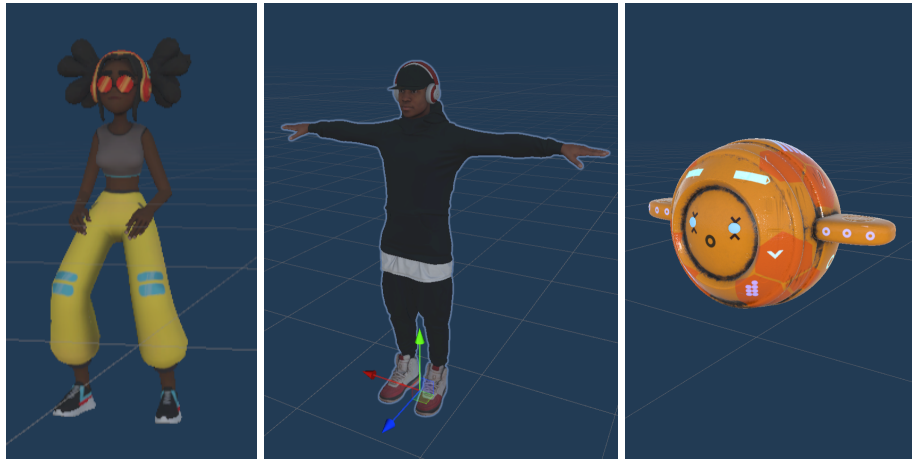


Diagram 2.1.11 & 2.1.12 & 2.1.13: 3D character models

We implemented the characters with some music elements like headphones to fit in the theme and make it more relevant to the crucial part of our game - music. We implemented the AI robot with flying movement, and it will always be following the main character (player). To accomplish this automatic movement, we have applied an AI agent to these characters, which will be further explained in a later part.



Diagram 2.1.14 & 2.1.15 & 2.1.16: 3D NPC and Machine models

To handle game events like triggering song stage and side quests, we designed and added some **non-player characters (NPC)**, including a song stage music album, animals like a wolfdog, a cube bot, a human-like bot, vending machine, and stores to fit in the world. They are rendered and placed in different places in the world scene, so players need to find them to trigger corresponding events.

- **Map and Icons**








Diagram 2.1.17 & 2.1.18: Minimap & full map in **ZAGREC**

A map of the 3D environment will be added to provide assistance for navigation in the gaming environment so that players could reach the NPCs or buildings they have to. Players will also find it easier to complete quests with the help of a map. The mini-map will be put on the top right corner of the game interface, while the full map will be opened by a button. To build these two maps, we assigned a camera following the main character and displayed the view from the top on the raw image at the corner. By using the **Sprite Renderer** component in **Unity**, we also created symbols in hidden layers that help to render 2D graphic objects, and render the 2D map in a 3D world.



Diagram 2.1.19: Icons on the map

To indicate the player and navigate to the destination inside the game, the icon symbols are added to follow the above important objects such as NPCs from different rendering layers. This can be shown in the texture that captures the raw image from the 3D world and cannot be seen from the main camera's rendering layer. Moreover, the border of the minimap is modified to a rotational combination of borders. The Minimap border rotates to give a direction when the player moves the camera around. The following table explains all icons used in the map:

Icon	Refers to
	Song stage (for entering song stage)
	Ranking machine (for viewing all player rankings)
	Sidequest
	Health booster (for refilling a small amount of Zagro)
	Store (for refilling a large amount of Zagro)

- **Player view**



Diagram 2.1.20: 3D world player view

The complete gaming view in the 3D world looks like this. There is a **player profile** (which includes an avatar icon, player name, and energy level - Zagro) on the top left corner, a **minimap** on the top right corner to help the player navigate in the scene, and two **shortcut help icons** for opening settings menu and a full map of the scene. The player will be the female character, while there is also a male friend character and a flying AI robot following the player. All of them will be involved in any conversation or dialogues with different NPCs.

2.2 3D Game mechanics

- **Scriptable object**

A scriptable object is a data container that can be used to save a large amount of data, independent from class instances. One of the main use cases for scriptable objects is to reduce our project’s memory usage by avoiding copies of values. In **ZAGREC**, we have constructed three main kinds of scriptable objects, the **game event object**, **dialogue event object**, and **loading object**.

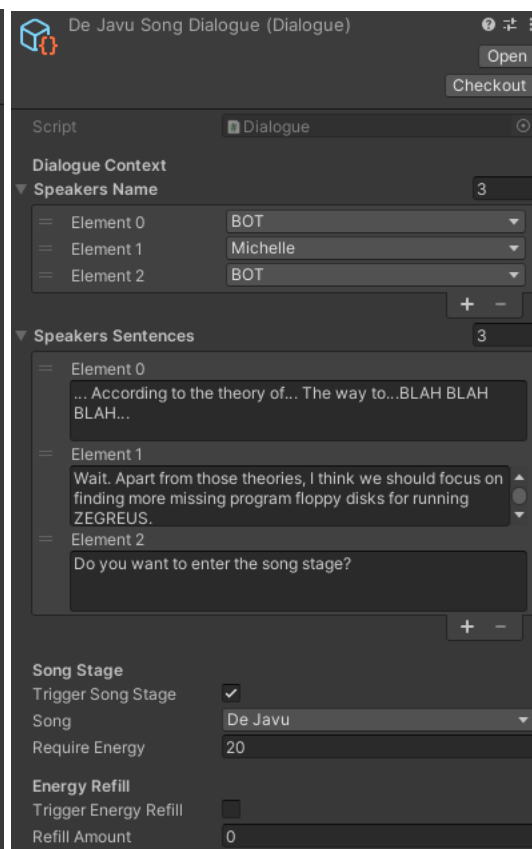
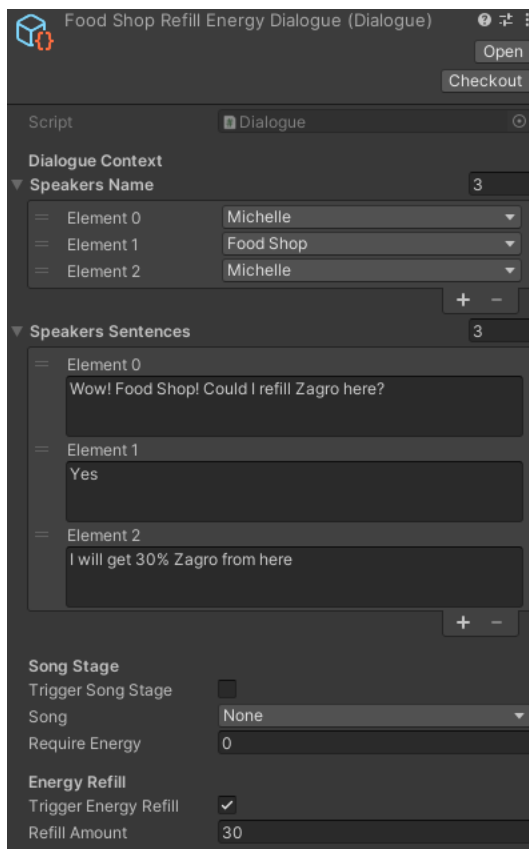
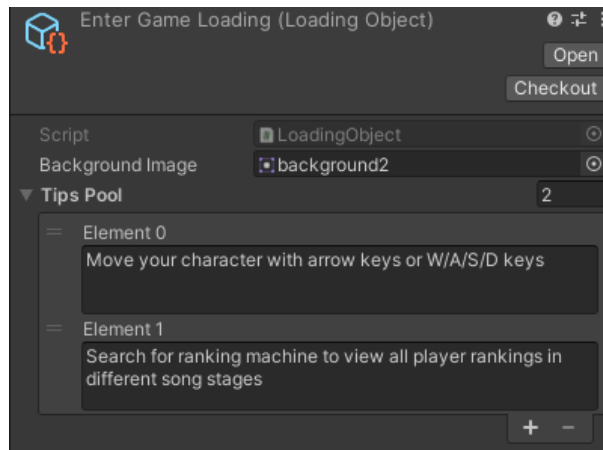


Diagram 2.2.1 & 2.2.2 & 2.2.3: Scriptable objects and inspector in Unity

For each kind of scriptable object, the data can be customized for different usage. A **loading object** contains a background image and gaming tips, which will be used in the loading scene where the data has been serialized and read in run time, fitting for various loading screens. A **dialogue event object** has contained basically the dialogue content and speaker names. In the game, there are many NPCs that players can start a conversation with, so we customized and separated the dialogue event into two categories, one for triggering the song stage and another one for triggering energy refilling. Therefore, we can easily customize different scriptable objects for providing data that is needed to trigger various game events.

- **Game event listener**

To handle idle animation and look at us when starting a conversation with a non-player character (NPC) and entering song stages, we designed a **game event listener** that listens to all game events. Once a player moves the character to get inside of a certain radius or discover new objects or press a button in front of the NPC, then the event trigger receives that event. It will call related functions in the order they were provided or invoke the **scriptable objects** that contain the relevant data. Therefore, the game scene will be switched to the next playing scene or triggered by different events.

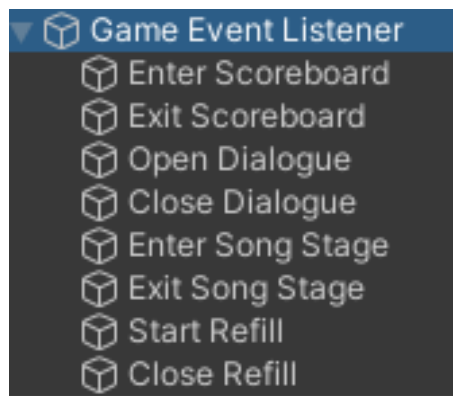


Diagram 2.2.4: Game event listener hierarchy

Based on what we created in the 3D world, we need to connect those objects to certain results when they are triggered. By creating an event listener inside the game at the top level of the process, apart from the animation of the object, all the other events including physics events, GUI events, and game events in Unity are going to communicate with the listener.

```

public class GameEventListener : MonoBehaviour {

    [Header("Events")]
    [SerializeField]
    private GameEvent gameEvent;

    [SerializeField]
    private UnityEvent unityEvent;

    Event function
    private void Awake() => gameEvent.RegisterListener(this);

    Event function
    private void OnDestroy() => gameEvent.DeregisterListener(this);

    1 usage 1 override
    public virtual void RaiseEvent() => unityEvent.Invoke();
}

public class GameEvent : ScriptableObject {

    private readonly HashSet<GameEventListener> _eventListeners = new HashSet<GameEventListener>();

    1 usage
    public void Invoke() {
        foreach (var listener in _eventListeners) {
            listener.RaiseEvent();
        }
    }

    1 usage
    public void RegisterListener(GameEventListener gameEventListener)
    => _eventListeners.Add(gameEventListener);

    1 usage
    public void DeregisterListener(GameEventListener gameEventListener)
    => _eventListeners.Remove(gameEventListener);
}

```

Diagram 2.2.5 & 2.2.6: C# Script for game event listener and game event

The game event will be firstly registered to the listener, and get ready to be called. Once the game event is invoked by the player, the corresponding event will then be raised in the game. For example, when the player gets close to an NPC, the event for opening dialogue will be triggered immediately, so the conversation with that NPC will be started. Besides, the game event can also switch scenes with a loading screen in between and enter a song stage.

- **Game Agent (AI)**

To make the behaviour of our game NPCs more intelligent, we have applied the **navigation mesh agent** in the game environment so that NPCs can automatically move between goals with steering and obstacle avoidance automatics.

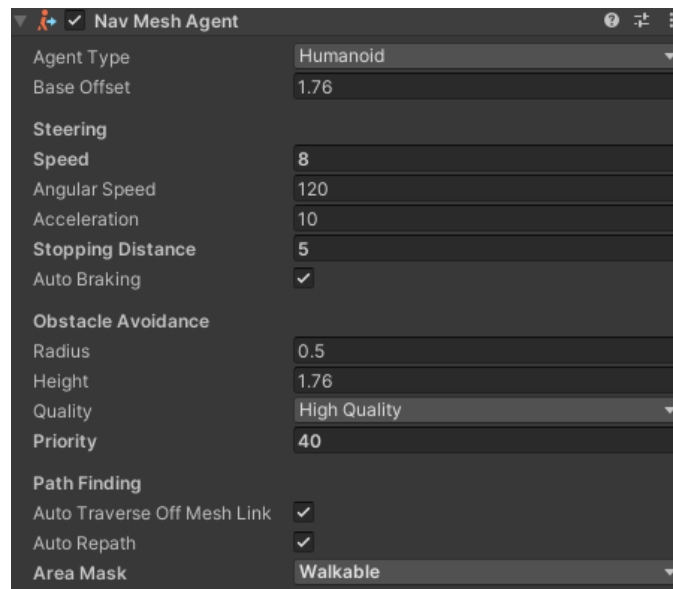


Diagram 2.2.7: Navigation mesh agent in Unity

The waypoint mechanics in the pathfinding method will also be using a navigation mesh rather than the predefined straight lines between points, as it will make the NPC movement more dynamic and natural. The blue areas generated in the world scene mean that those polygons are where the AI agent is allowed to walk, as these areas are marked as walkable for the NPCs.

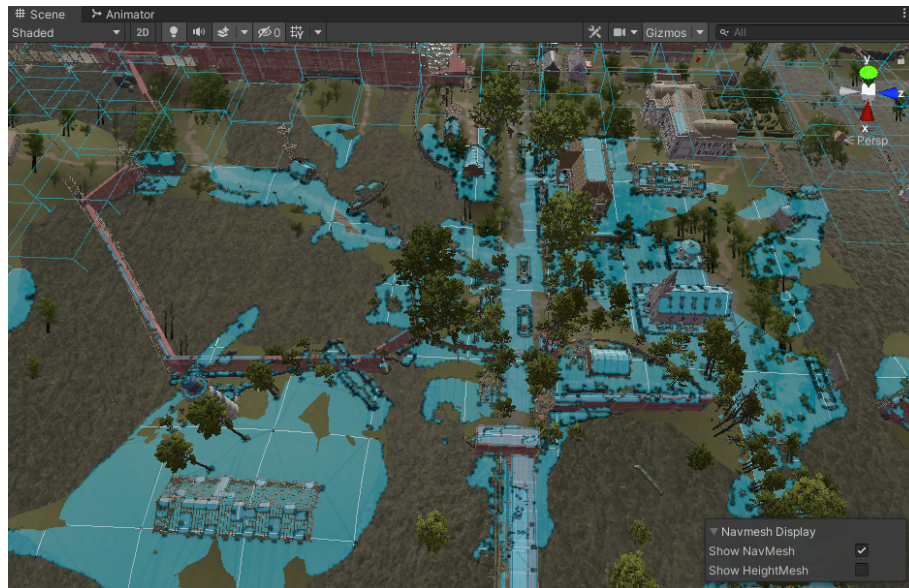


Diagram 2.2.8: Navigation meshes (blue areas) game environment

Using the navigation mesh as the waypoint system, we have developed the AI algorithms that command the NPCs to find the path between their current location and destination. We also designed two main parts of the AI algorithms, **Pursuit** and **Wander**.

```

Event function
private void Update() {

    _agent.SetDestination(isAutoMove ? _randomPosition : followTarget.position);

    if (_isAnimatedFollower) {
        _npcAnimationManager.Move(_agent.remainingDistance > _agent.stoppingDistance
            ? _agent.desiredVelocity : Vector3.zero);
    }
}

```

Diagram 2.2.9: C# Script for NPC auto movement

Pursuit is a behaviour in which NPCs will have the agent smartly follow the player's position. And we applied this algorithm to the follower character in the game.

Wander is a behaviour in which NPCs will have the agent freely walk around the walkable environment with smooth randomness. And we applied this algorithm to the automatic movement of NPCs in the game.

- **Dialogue system**

In the scripting process, there are mainly 4 parts that we considered: plot outline, detailed characterization of characters, main plot, and chapter settings: the general plot of each chapter, and the fate of the characters.

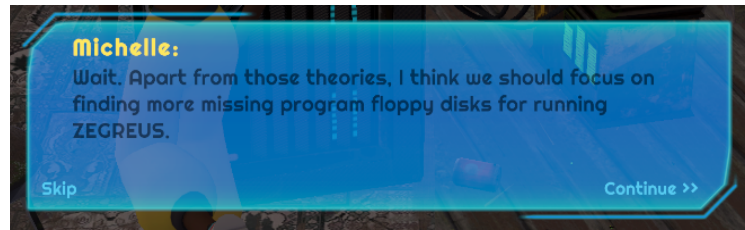


Diagram 2.2.10: A pop-up dialogue box in **ZAGREC**

Dialogues between the player and the NPCs are designed based on the story created for this game. Characters have distinctive characteristics for easy choice of words and clear verbal expression of the characters' habits and personalities. The length of the dialogue scenes is kept short for not to make our players feel bored.

```
public void StartDialogue(Dialogue dialogue) {

    DialogueCamera.Instance.TriggerDialogueCamera();

    _triggerSongStage = dialogue.triggerSongStage;
    _songName = dialogue.song.ToString();
    _requireEnergy = dialogue.requireEnergy * -1;

    _triggerEnergyRefill = dialogue.triggerEnergyRefill;
    _refillAmount = dialogue.refillAmount;

    animator.SetBool(id: IsOpen, value: true);
    _speakers.Clear();
    _sentences.Clear();

    speakerText.text = dialogue.speakersName[0] + ":";

    foreach (var speakerName in dialogue.speakersName) {
        _speakers.Enqueue(item: speakerName.ToString());
    }

    foreach (var sentenceString in dialogue.speakersSentences) {
        _sentences.Enqueue(sentence);
    }

    BeginNextSentence();
}
```

Diagram 2.2.11: C# Script for dialogue manager

To implement the dialogue system, we started by creating a **dialogue manager** that handles all dialogue events. When the **game event listener** receives a signal from the player that triggered the dialogues, it will raise the dialogue event, encapsulating with data from **dialogue scriptable objects**. This encapsulated dialogic object will then be passed to the **dialogue manager** for de-encapsulation so the dialogic data can be processed by the manager. After finishing enqueueing the speakers' sentences and names, the in-game conversation will then be triggered.

- **Energy level (Zagro)**

In **ZAGREC**, one of the add-on features is the energy level - **Zagro**.



Diagram 2.2.12: Energy bar in **ZAGREC**

Each song stage requires a different amount of **Zagro**, and players need to fill their **Zagro** when it is not enough for them to continue a song stage.

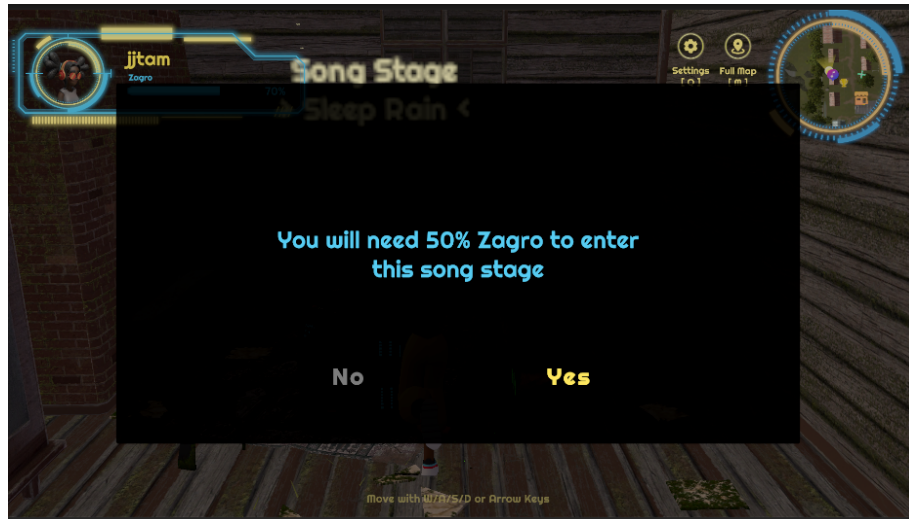


Diagram 2.2.13: Energy requirement for song stage

They can refill their **Zagro** in various machines, including food shops, vaccine stations, vending machines, street food vendors, etc. Players can search for the respective facilities by looking for a **Health Booster** (+) or a **Store** (🏆) on the map.

The main idea of implementing an energy level to **ZAGREC** is to prevent players from getting addicted to the game and hence suffering from any potential health issues. We hope to provide a healthy gaming atmosphere and encourage all players to take a pause from time to time during gameplay.

- **Side quests**

Some side quests are designed and implemented in **ZAGREC**, which we hope to make the quests more hilarious and interesting but are also related to music to some degree. For example, we might ask the player to reach an NPC who plays a piece of instrumental music and guess what instrument it is, in a multiple-choice format. This could be achieved by using the **LMMS** (Linux MultiMedia Studio) techniques. We will create a new channel for the short rhythm and select the instrument for the rhythm. The rhythm can be composed by the corresponding keys of the instrument and output as MIDI, .ogg, .wav, or mp3 files. There are also some other side quests that are not that related to music, but players will find it fun to play with it and gain some crucial information about the gaming environment and clues to reviving the whole city.

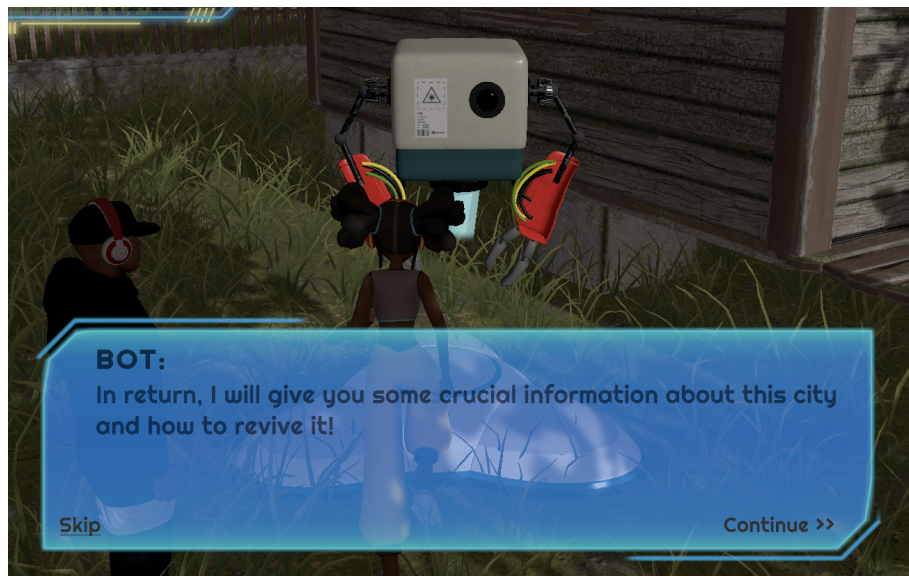


Diagram 2.2.14: An example of side quests in **ZAGREC**

For example, the complete conversation with the bot shown above is as follows:

Speaker	Content
Bot	Hey... hey! Can you help me?
Player	Oh! What happened?
Bot	One part of my body was stolen by a Wolfdog... but I have been here for too long and my battery cannot sustain to let me find it. Can you help me find the Wolfdog and return that part of the hardware to me?
Bot	In return, I will give you some crucial information about this city and how to revive it!
Player	Sure! I will help you to find it.
James	Let's go ~ Woohoo!



Diagram 2.2.15: Searching for the Wolfdog to complete side quest

As all the living NPCs (including animals and robots) can wander around the place randomly, players have to really walk around to search for them (like playing hide and seek)! This boosts some more excitement in the game.

2.3 Rhythm song stages

- **Song list for chapters**

We have discussed and come up with a song list. We will follow the list to generate song charts and implement them in **ZAGREC** according to different chapters.

Chapter	Songs	Description (if any)
Prologue	<ul style="list-style-type: none"> • Undertale OST: Once upon a time (Remix) 	Tutorial song
1	<ul style="list-style-type: none"> • Awaken • Neo Tokyo - Cyberpunk Mix (Sandman - Ignite) 	Quiet in the beginning, then progressively becoming louder - Sci-fi, energetic music
2	<ul style="list-style-type: none"> • Alice's Theme • Nier Copied City • Cytus Symphony LVBNR5 Weiß 	- Rhythmic, nervous, wonder, unknown
3	<ul style="list-style-type: none"> • Nier • 【Ado】Usseewa • 【Ado】ボッカデラベリタ • Biotonic Cytus 	Showing the seamy side of humanity, feeling hopeless - dark music, minor, heavy metal
4	<ul style="list-style-type: none"> • light and shadow • Hopes and Dreams • De Javu (Uki Violetta) • Sleeping rain song (piano piece) • Grand Escape (RADWIMPS) 	- Peaceful, feel like raining
5	<ul style="list-style-type: none"> • Sky Isle of Dawn: Dawn Flight 2 • Peaceful Piano Royalty Free Calm Relaxing Background Music No Copyright 	- Even more peaceful music
6	<ul style="list-style-type: none"> • Legends Never Die 	- Motivational, exciting music

- **Rhythm game playing view**



Diagram 2.3.1: 2D rhythm game playing scene

The rhythm game stage has been developed in a 2D workspace that contains 4 tracks (corresponding to **A/S/J/K** keys in default) and a baseline on a 2D surface. Each of these 2D graphics is rendered by the UI Sprite Renderer component with different colour materials in Unity, it controls how the 2D graphic object visually appears in a 2D scene with custom colours. With the use of **UI Sprite Renderer**, we can easily change any interface of the 2D scene with another graphic.

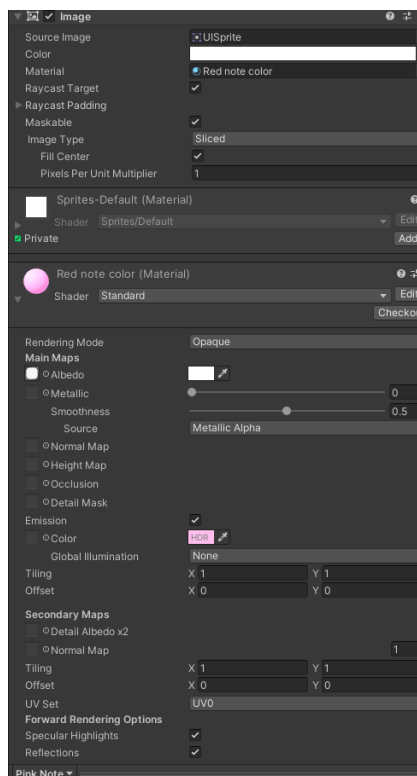


Diagram 2.3.2: Pink note UI rendering

The music note is spawned on the top of each lane during the music game playing and moved from the vanishing point to the baseline, so players need to press the correct keys before the note exceeds the baseline. Players are able to get different scores (Perfect/Great/Good/Bad) based on their performance, which is calculated by our music note scoring algorithms. They will get a health deduction if they Miss any note, the game will also be lost after the health goes down to zero.

PCTyx is able to provide the CSV, text, and JSON files for the beat mapping generation[3]. Yet, it takes time for us to read and interpret the text in the file through C# algorithms for each page of the notes and the corresponding song bpm, speed of judgment line movement, and note positions. This may increase the running time on interpreting the beat map during the gameplay. On the other hand, **Unity** allows the use of the **MIDI library**, and editing MIDI is a convenient way to compose the beat map. Yet the MIDI file is not able to store the positions of the pop-out notes, so we decided to have a different playing mode such that it is easier to be interpreted by **Unity** and further coding.

- **2.5D rhythm game playing scene design**

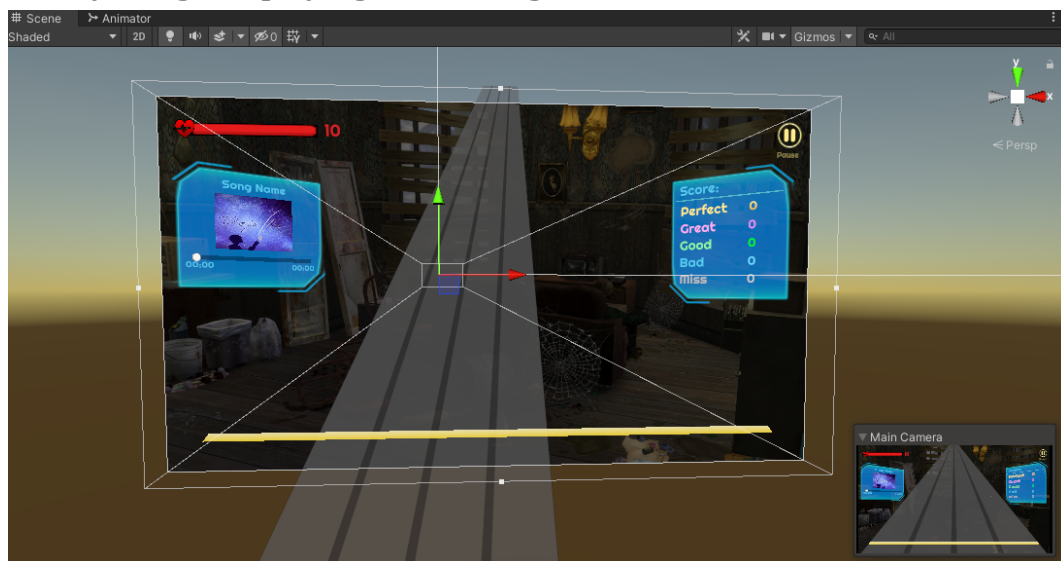


Diagram 2.3.3: Development view of 2.5D song stage

We have successfully visualized the MIDI best map in the game in a vertical falling 2D layout. However, the notes are falling too fast towards the judgment line for the players to react. This may make the rhythm game too difficult to play. In this case, we are making the rhythm game playing scene in 2.5D instead of 2D to increase the reaction time for the players.

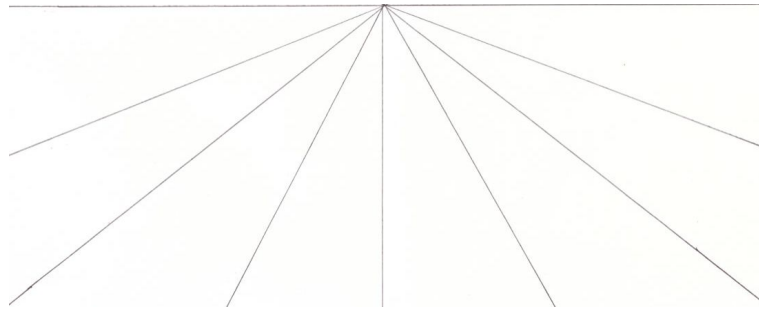


Diagram 2.3.4: Sample of perspective lines

As the notes will be travelling in perspective lines, moving from the vanishing point to the nearest point. Therefore, the notes will be seen much earlier before they reach the judgment line. A 2.5D playing scene will increase the reaction time for players and reduce the difficulty of the game. This can be achieved by adjusting the X-Y-Z coordination of the notes generating scene.

- **MIDI files and beat map generation**

For our rhythm game stage, one MIDI file is read to the game for producing a beat map for one song. MIDI file is an excellent way to store a sequence of timed events, whatever they may represent. These timed events can be the triggering of certain hardware-specific samples that can be played in music. Indeed, it can be also used to sequence the events of a rhythm-based game, which provides abilities to read music tempo maps, frequencies, pitches notes, and chords for our song stage.

Having the pitches representing the positions of the notes, our rhythm game is like going through the notes in vertical instead of horizontal layout in the MIDI editor.

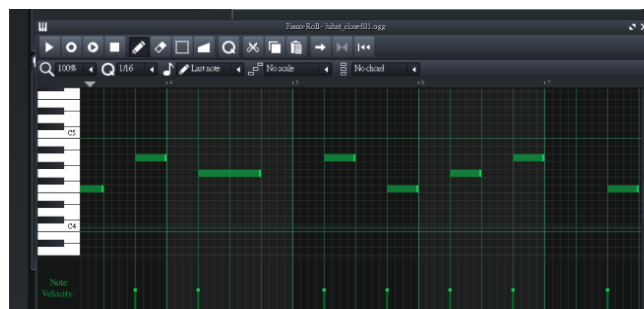


Diagram 2.3.5 & 2.3.6: The LMMS application interface

The MIDI file used in the rhythm stage is created by LMMS, which is a digital audio workstation application program [4]. By adding the new instrument track and the sample song as a new track to the project, we can compose the beat map while listening to the sample song. In our case, we use 4 pitches of the Hi-hat to represent the 4 linear positions that the notes travel. If more note positions are needed, we can simply choose another instrument that has a wider pitch range for composing. During the composition, only the R/T/Y/U keys are needed to record the hi-hat notes on the LMMS project. Manual corrections are done by clicking on certain notes after the first composition.

- **Music note generation**

After creating the MIDI file, we have employed the DryWetMIDI .NET library to work with MIDI data [5]. This library allows us to read standard MIDI files, and manage high-level objects like musical notes by coding. The data in a MIDI file contains important information about the musical elements, which helps to build music notes from the beat map in the game.

```
using Melanchall.DryWetMidi.Core;
using Melanchall.DryWetMidi.Interaction;

private void ReadFromFile() {
    _midiFile = MidiFile.Read(filePath: Application.dataPath + "/My Audio/Midis/" + midiFileName);
    GetDataFromMidi();
}
```

Diagram 2.3.7 & 2.3.8: C# Script for reading MIDI file

GetDataFromMidi() function has been written to get the music notes from the MIDI file, the note array is an array that keeps the entire position-in-beats of the notes in the song. We can then set the time stamp of each note to our tracks in the 2D rhythm song stage, so music notes can be generated according to the playing time.

```
private void GetDataFromMidi() {
    var notes:ICollection<Note> = _midiFile.GetNotes();
    var noteArray = new Note[notes.Count];
    notes.CopyTo(noteArray, arrayIndex: 0);

    foreach (var noteTrack in noteTracks) {
        noteTrack.SetTimeStamps(noteArray);
    }
}
```

Diagram 2.3.9: C# Script for getting MIDI data

We followed the song position in each of the tracks to determine when a note should be spawned. The music note is spawned using our time spawning algorithm. We first check if there are no notes left in the time stamps queue ($\text{spawnIndex} < \text{timeStamps.Count}$). If there are notes not yet spawned, we then check with the entire song time to see if the song reaches the beat where the next note should be spawned. If it is true, we spawn the note and increment spawnIndex so that the algorithm keeps track of the next note to spawn.

```

if (_spawnIndex < timeStamps.Count) {
    if (SongStageManager.Instance.GetAudioSourceTime() >= timeStamps[_spawnIndex] - SongStageManager.Instance.noteTime) {
        var musicNote:GameObject = Instantiate(noteObject, transform);
        musicNote.SetActive(true);
        musicNotes.Add(item: musicNote.GetComponent<MusicNote>());
        musicNote.GetComponent<MusicNote>().assignedTime = (float)timeStamps[_spawnIndex]; // set the note position, so t
        _spawnIndex++;
    }
}

```

Diagram 2.3.10: C# Script for spawning music notes

- **Music note movement**

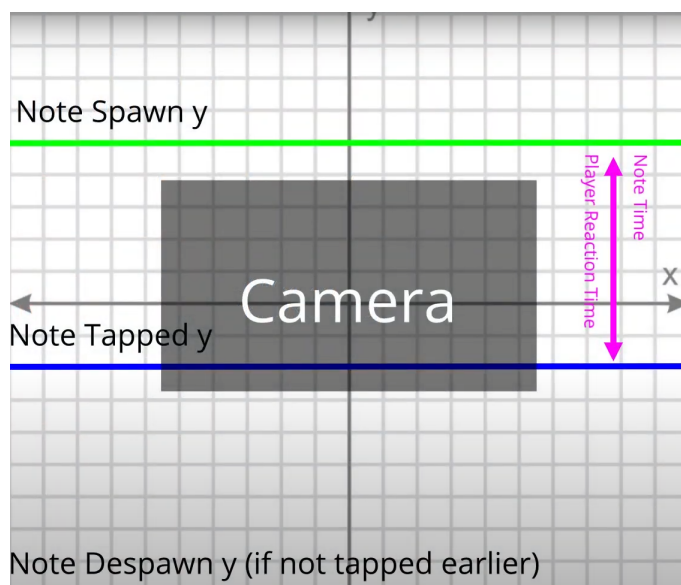


Diagram 2.3.11: 2D note coordinate surface

The first version of the rhythm game stage has been developed in a 2D surface along with the X-axis and Y-axis. An area of the camera is set in the centre, which represents the player's view in the game while the X-axis represents the width of our 2D playing scene, but it is trivial in this movement calculation. The Y-axis in this 2D surface acts as more important to note spawn and de-spawn. Note Spawn Y (Green line) is where the music note begins to spawn, the Note Tapped Y (Blue line) is where we put the baseline and the note can be tapped. Note time (Pink line) represents how much time the note is on the screen, so the player has their reaction time to tap it before the note goes down and de-spawns.

```

double timeSinceInstantiated = SongStageManager.Instance.GetAudioSourceTime() - _timeInstantiated;
float t = (float) (timeSinceInstantiated / (SongStageManager.Instance.noteTime * 2));

if (t > 1) {
    Destroy(gameObject);
}
else {
    transform.localPosition = (Vector3) Vector2.Lerp(a: Vector2.up * SongStageManager.Instance.noteSpawnY,
        b: Vector2.up * SongStageManager.Instance.NoteDespawnY, t);
    // GetComponent<Image>().enabled = true;
}

```

Diagram 2.3.12: C# Script for moving music notes

After designing the X and Y coordinates, we developed the movement of notes we spawned according to the song. We first need to calculate the elapsed time of the song, so we can have a time parameter (t) to determine whether to destroy the note or move the position according to the **Spawn Y** and **De-spawn Y-axis** by using the Linear Interpolation function in Unity.

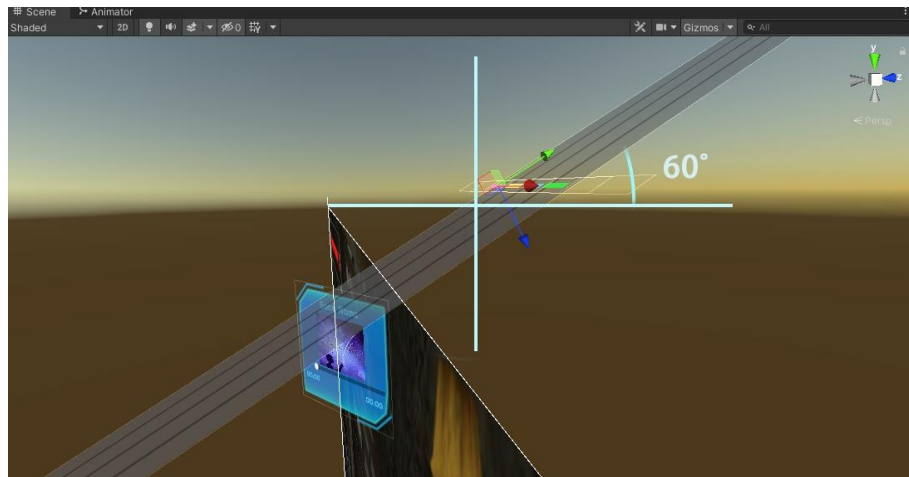


Diagram 2.3.13: Angle adjustment to 60 degrees

The second version of the song stage is updated to a 2.5D playing scene, so we made an angle adjustment to the line panel of 4 tracks, the angle has been set to 60 degrees for each line, and notes spawner so that the notes can be generated according to this degree of angle. From the player's gaming view, this adjustment of the angle of notes will make the player see the note will be moving from the vanishing point to the baseline at the bottom.

- **Music audio controller**

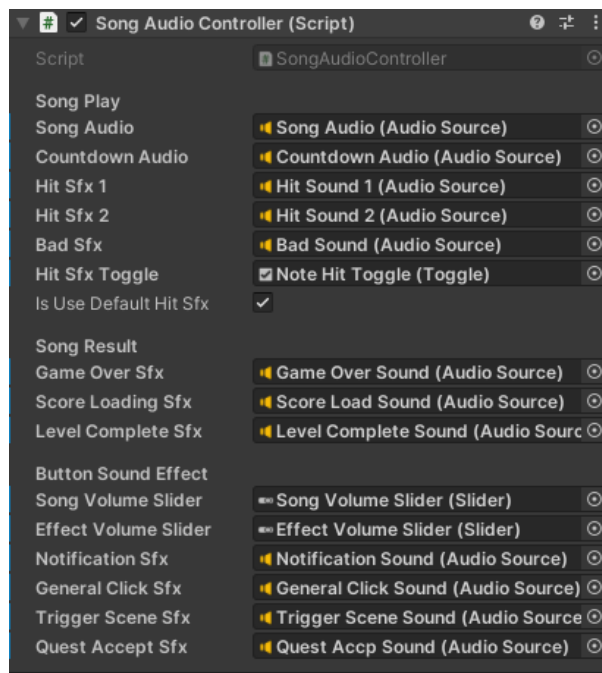


Diagram 2.3.14: Song audio controller in Unity

To control all the audio sources in the song stage, a song audio controller has been built. It can manage all the sounds the player can hear from the song stage during the playing mode. Changing the volume and toggling the hit sound effect are the player preferences in the game.

```

1 usage
private void StartSong() {
    if (_songStarted) { // for resume song playing
        _songAudio.UnPause();
    }
    else {
        _songStarted = true;
        _ready = true;
        _songAudio.Play();
    }
}

```

Diagram 2.3.15: C# Script for audio controller

The **StartSong()** function has been written to start the audio playing. It helps us to better manage song playing status in the game, like song ready, song pause, and song start.

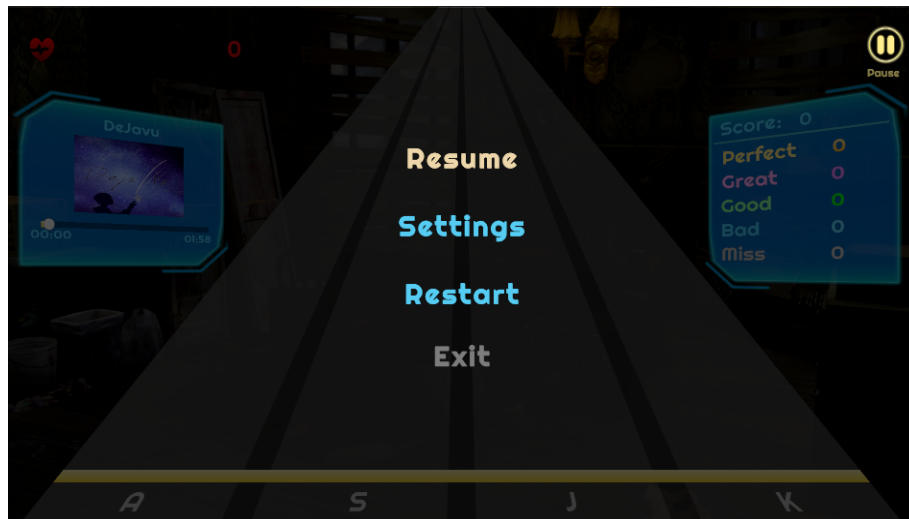


Diagram 2.3.16: Pause screen in **ZAGREC** song stage

With the music audio controller, the player can press the right top button to pause, resume, and restart their rhythm song stage at any time they like.

- **Music note tapping and scoring algorithms**

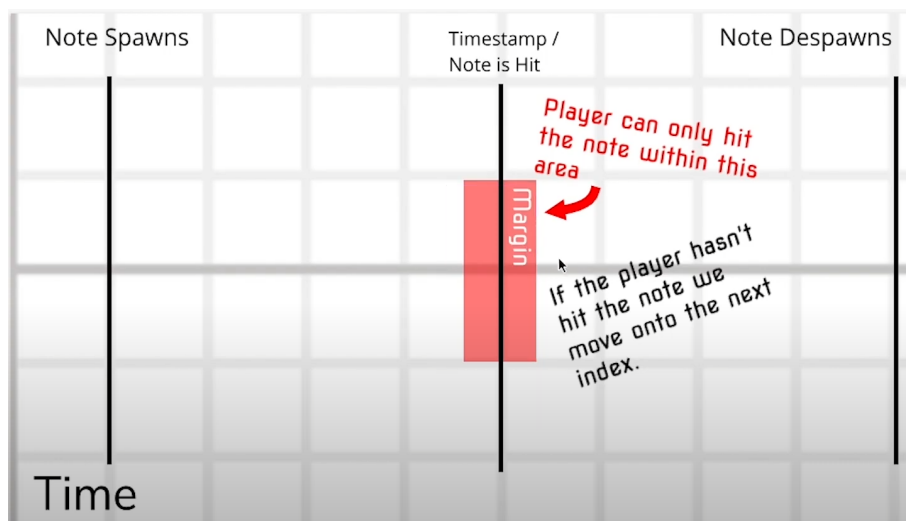


Diagram 2.3.17: Player input margin design

To get the more accurate player inputs for note score calculation, we designed the **tapping algorithm**. In the rhythm game stage, players need to press the correct buttons before the note surpasses the baseline in order to get their score. Therefore, we designed an area of margin error for calculating the player input accuracy. A keyboard input manager can be deployed at this margin, so we can capture and receive what keys the player pressed, and these keys will trigger different events. If a player does not press any key, the input manager will not be triggered, which means the player has missed a note. Otherwise, the player's

input can be immediately received by us with a small input latency and a high accuracy.

To deal with this margin of error, we designed a **scoring algorithm** that calculates the note score during the song stage. The key factor or parameter should be how accurate the players tap every note in the margin of error during the song stage. In other words, the reaction time for the player to tap the note before it has been missed. Therefore, to design the algorithm, we need to calculate the value of the player's reaction time.

We firstly assigned different values to the margin of errors that will be used to calculate the accuracy of players' tapped notes:

Note Score	Margin Error
PERFECT	≤ 0.025
GREAT	≤ 0.065
GOOD	≥ 0.065
BAD	≥ 0.1

Having designed the margin error for the music notes, we developed the algorithm to compare the tapped note absolute time differences. We will have to get the absolute time difference of a note, measured by the time stamp of a note minus the audio song playing time (noteTime - songTime). Therefore, we can use it as the player input reaction time for the note tapping action.

```
double inputReactionTime = Math.Abs(audioTime - timeStamp); // song time - button pressed time
```

Diagram 2.3.18: C# Scripts for player reaction time calculation

After calculating the player input reaction time, we can compare it to the border error in order to get the note score. If players haven't missed the note, they would enter the note hitting statement, passing their reaction time to the **NoteHit()** function to further calculate their scores.

```

StartCoroutine(routine: ActiveForSecond(lineEffect, second: 0.2f));

if (inputReactionTime < inputMargin) {
    NoteHit(inputReactionTime);
    Debug.Log(message: $"Hit on {_inputIndex} note");
    Destroy(musicNotes[_inputIndex].gameObject);
    _inputIndex++;
}
else {
    NoteBad();
    Debug.Log(message: $"Hit inaccurate on {_inputIndex} note with {inputReactionTime} delay");
}
}
else if (timeStamp + inputMargin <= audioTime) { // too late to tap the note
    NoteMiss();
    _inputIndex++;
    Debug.Log(message: $"Missed {_inputIndex} note");
}
}

```

Diagram 2.3.19: C# Scripts for player input calculation

In the **NoteHit()** function, we get the player reaction time parameter to compare the values of the margin of errors. Finally, we can accurately calculate the player score for each note they have hit, and invoke the corresponding visual and sound effects.

```

Frequently called 2 usages
private void NoteHit(double reactionTime) {
    var position:Vector3 = effectPosition.transform.position;
    Instantiate(pressedEffect, position, Quaternion.identity);

    Debug.Log(message: $"reactionTime = {reactionTime}");

    if (reactionTime <= 0.025) { // Perfect
        Instantiate(perfectEffect, position, Quaternion.identity);
        SongScoreManager.Instance.NotePerfectHit();
    }
    else if (reactionTime <= 0.065) { // Great
        Instantiate(greatEffect, position, Quaternion.identity);
        SongScoreManager.Instance.NoteGreatHit();
    }
    else { // Good
        Instantiate(goodEffect, position, Quaternion.identity);
        SongScoreManager.Instance.NoteGoodHit();
    }
}
}

```

Diagram 2.3.20: C# Scripts for note tapping calculation

- **Score grading scheme**

To develop our music note scoring system, we have designed a grading scheme for all songs, the score includes 90% of the "Judgment Points" and 10% of the "Combo Points", and the Combo Points are calculated progressively, COMBO interruptions have an obvious impact on it, BAD and MISS will interrupt the max COMBO number, and affect the total score.



Diagram 2.3.21: Song stage results screen in **ZAGREC**

Technical Point (TP) reflects the accuracy of strikes and is an important measure of a player's strength. TP has nothing to do with COMBO number, PERFECT count is the only factor affecting TP value. TP is calculated by having a PERFECT count divided by a total number of notes.

When the player completes the stage, a corresponding grade will be assigned according to the score obtained. The player must attain 70% to clear the stage. Required scores for obtaining different grades are as follows:

Grade Assigned	TP
S	$\geq 90\%$
A	$80\% \leq x < 90\%$
B	$70\% \leq x < 80\%$
C	$x < 70\%$

After calculating the full mark for the playing song by the formula ($\text{totalNotes} * \text{scorePerPerfectNote} + \text{totalNotes} * \text{comboBonus}$), we finally can compute the player final score by comparing their hit percentage with TP, and complete the grading progress with showing the grade image.

```
private int CalFullScore() {
    int totalNotes = _perfect + _great + _good + _miss;
    int fullScore = totalNotes * _scorePerPerfectNote + totalNotes * (comboCoEfficient * _scoreForCombo);
    return fullScore;
}
```

```
int fullScore = CalFullScore();  
int hitPercent = finalScore/fullScore;  
if (hitPercent >= 0.90f) {  
    sRank.SetActive(true);  
}  
else if (hitPercent >= 0.80f) {  
    aRank.SetActive(true);  
}  
else if (hitPercent >= 0.70f) {  
    bRank.SetActive(true);  
}  
else {  
    cRank.SetActive(true);  
}
```

Diagram 2.3.22 & 2.3.23: C# Scripts for final score calculations

2.4 Game settings

- Main camera



Diagram 2.4.1: Main Camera viewing in Unity

We adjusted the parameters for camera settings to make it a better angle for players to view. We tested different angles and found the best camera settings to use in **ZAGREC**. **Cinemachine** is applied to override the original camera inside **Unity** as **CinemachineBrain** which works with a branch of virtual cameras called **CinemachineCamera** [6].

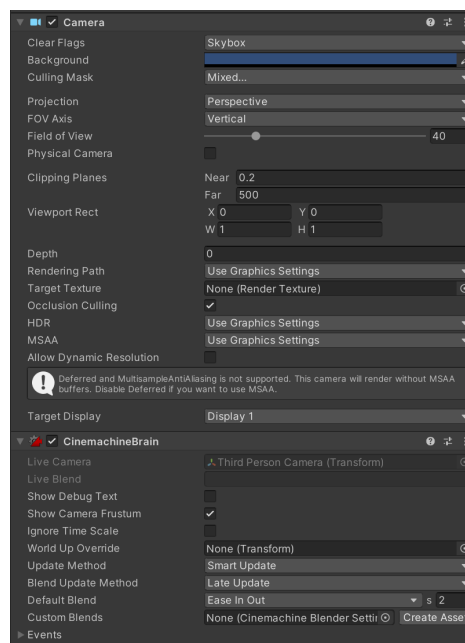


Diagram 2.4.2: Camera inspector in Unity

With the priority property, camera switching becomes simpler. On top of this, we have implemented a **CameraManager** to execute switching and custom blending based on their priority at different aspects.

- **Third-person camera**



Diagram 2.4.3: Third-person Camera viewing in Unity

Third-person camera is the players' usual view in-game that has been implemented with **CinemachineCamera** mentioned above in the Main Camera section. It has been set to follow the player's character at a certain angle. To avoid solid shaking when a character is moving, damping is added to the tolerance of the camera distance from the player. When a player is running out of the damping range, the camera starts blending to follow the player and blend back to the settled distance when the player stopped running. This makes **ZAGREC** a third-person player game.

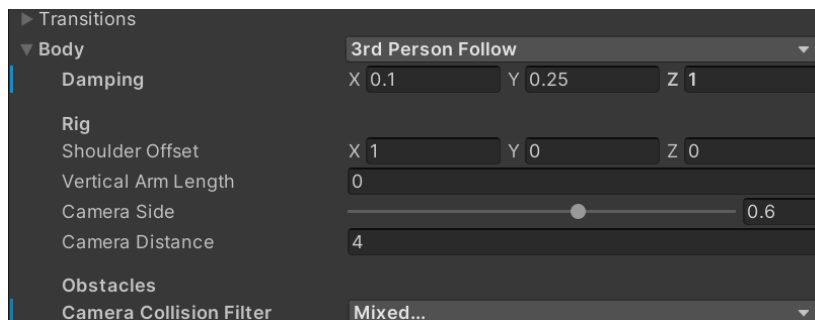


Diagram 2.4.4: Third-person Camera inspector in Unity

- **Map camera**

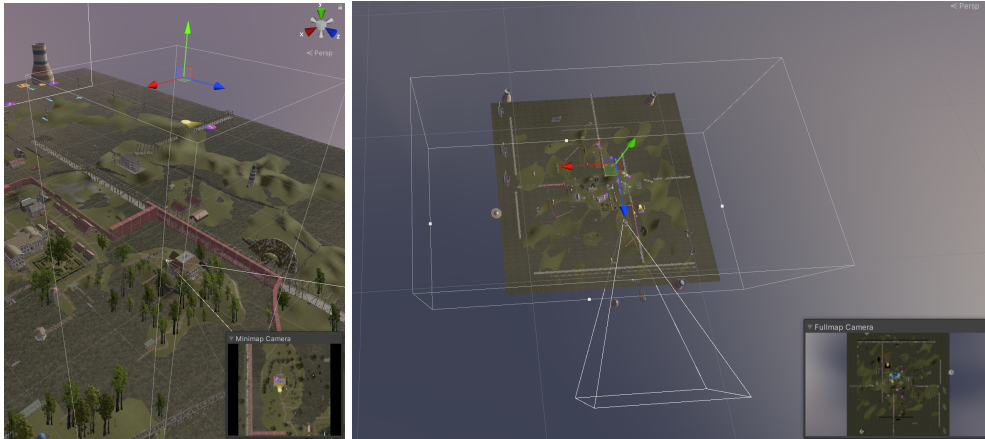


Diagram 2.4.5 & 2.4.6: minimap camera, full map camera in Unity

Utilizing the **Camera** in **Unity**, we can show the entire map area and in an orthographic projection. Different from the player's perspective projection, it gives a rectangle projection from the player's location on the upper view. The images captured from the cameras are then output and rendered as a live texture on the mini-map and full map.

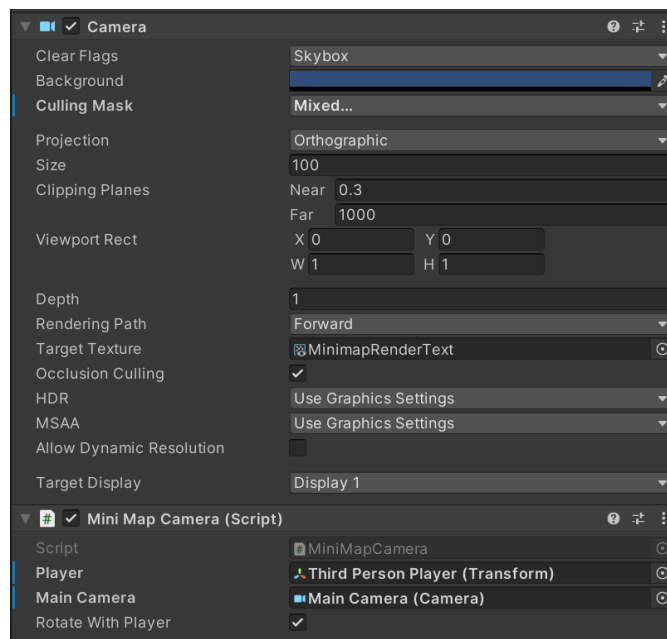


Diagram 2.4.7: Minimap Camera inspector in Unity

To give some rotation to the min map when a player is moving around, the minimap camera is implemented to follow and transform with the player's perspective and movement at a nearer angle.

- **Third-person player control**

To achieve the third-person control blending with the third-person camera, we have developed a third-person controller to manipulate all the player controls.

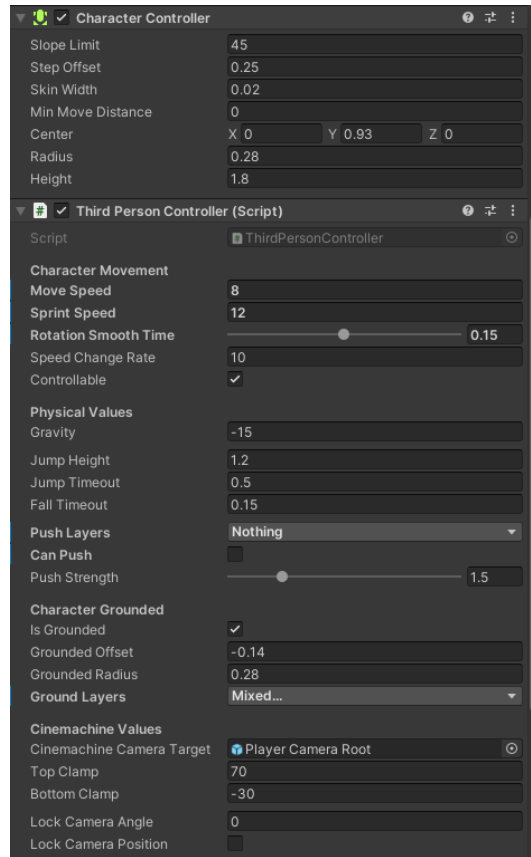


Diagram 2.4.8: Third-person controller inspector in Unity

The third-person controller will manage and override the **Character Controller** in **Unity**, which directly manages the third-person gaming values for a character, including character movement, animations, physics, and **Cinemachine** values.

```

_hasAnimator = TryGetComponent(out _animator);
_controller = GetComponent<CharacterController>();
_input = GetComponent<PlayerInputSystem>();

AssignAnimationByIds();

// reset our timeouts on start
_jumpTimeoutDelta = jumpTimeout;
_fallTimeoutDelta = fallTimeout;

// check player position prefab
if (!PlayerPrefs.HasKey("PlayerPosX") && !PlayerPrefs.HasKey("PlayerPosY") && !PlayerPrefs.HasKey("PlayerPosZ")) {
    PlayerPrefs.SetFloat("PlayerPosX", 379f);
    PlayerPrefs.SetFloat("PlayerPosY", 6.5f);
    PlayerPrefs.SetFloat("PlayerPosZ", 700f);
}
LoadPosition();
    
```

Diagram 2.4.9: C# script for third-person controller

The code here showed the starting function for the third-person controller, which assigns, loads, and gets the required components, objects, and data that are

needed in the late update method. Therefore, the third-person controller will dynamically update the character's parameters per frame during the gameplay.

- **Player input system**

To collaborate with the third-person control, we have developed the player input system to handle the player input values from the keyboard and mouse signals. So, the player can control the character by using the input system.

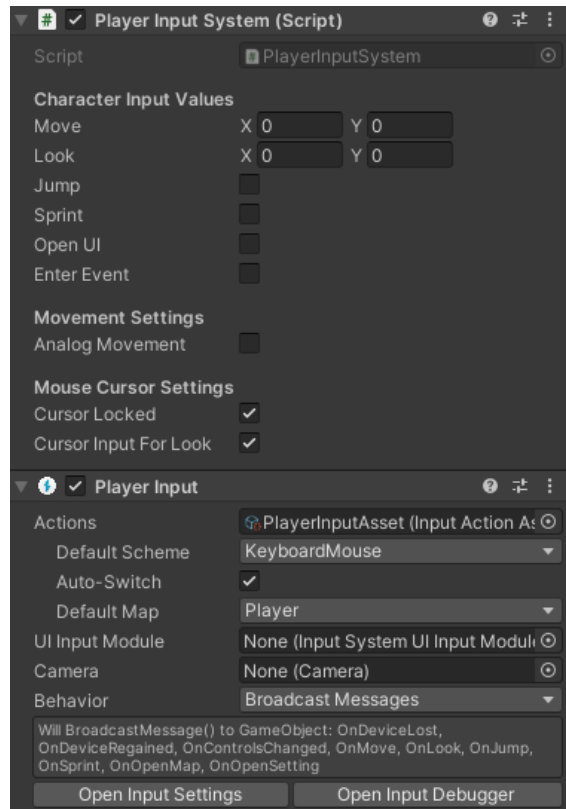


Diagram 2.4.10: Player Input System inspector

With the player input system, we can easily customize our key of control for the player and their input actions to notify the input system of Unity.

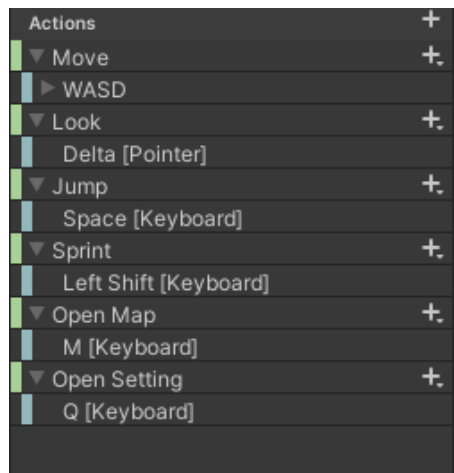


Diagram 2.4.11: Player actions asset in Unity

Therefore, with the input system, players are able to move the character around in the 3D world with W/A/S/D keys or arrow keys, and adjust the camera view angle by moving the mouse. In the rhythm song stage, the default playing keys for the 4 tracks are A/S/J/K, while players can also customize the playing keys to their preferred ones by pausing the game and changing in **Control Settings**. The changes will be saved in the database and applied in all other song stages.

- **Character movement**

With the player control and input system, we can control our character to move around by using different keys. Additional to simple position movement, we also applied the animation to the character in order to make movements to be more human-like.

```

Vector3 targetDirection = Quaternion.Euler(x:0.0f, y:_targetRotation, z:0.0f) * Vector3.forward;

// move the player
_controller.Move(motion:targetDirection.normalized * (_speed * Time.deltaTime) + new Vector3(x:0.0f, y:_verticalVelocity, z:0.0f) * Time.deltaTime);

// update animator if using character
if (_hasAnimator) {
    _animator.SetFloat(_animIDSpeed, _animationBlend);
    _animator.SetFloat(_animIDMotionSpeed, inputMagnitude);
}
    
```

Diagram 2.4.12: C# script for character move

We have used the **Animator** provided by **Unity** to customize and simulate the character movement for each motion.

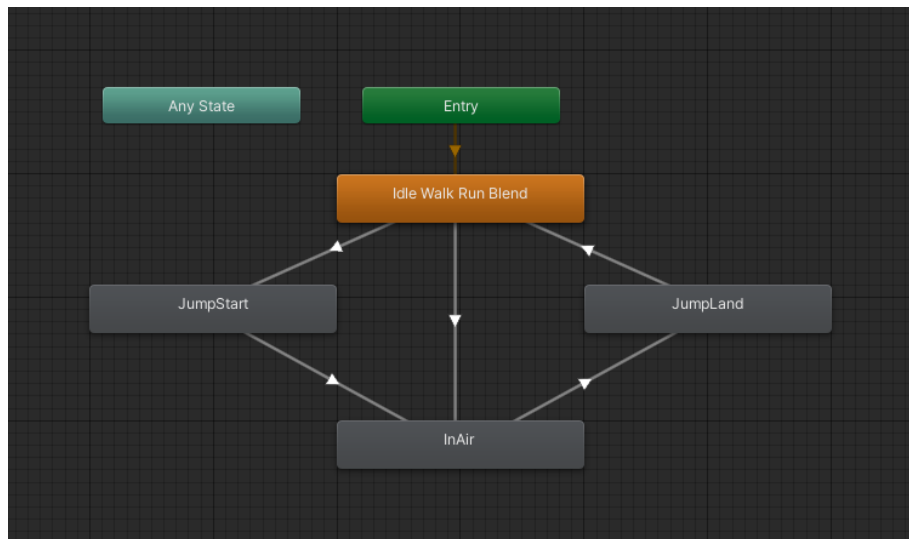


Diagram 2.4.13: Character animator in Unity

The animator contains different action animations of the character, including **Jump**, **Run**, **Walk**, and **Idle**, and they will be connected with each other animation. We aim to ensure the natural movements of the characters, so we added different conditions for the animation transition along with a white arrow.

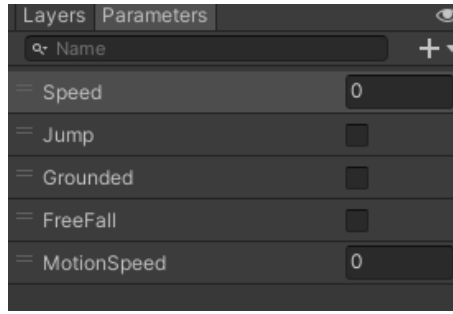


Diagram 2.4.14: Animator parameters in Unity,

To make our transition more smoothly, we added serial parameters for the transition of animation. For instance, when a character is running, the transition value for **Speed** is overwritten by the **MotionSpeed**, and the animation is notified and changed simultaneously to **Run** instead of **Idle**. We hope to make the human characters more human-like and natural to provide a more immersive gaming experience to players.

- **Audio settings**

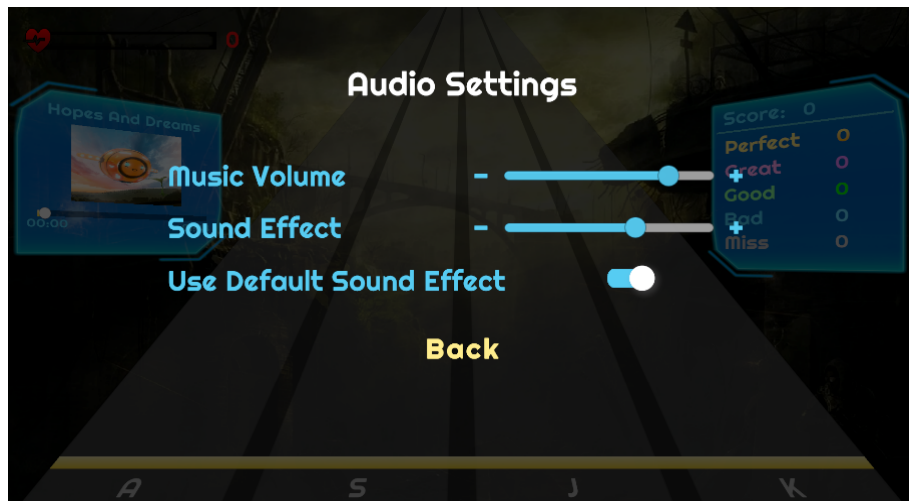


Diagram 2.4.15 & 2.4.16: Audio settings in song stage and 3D game

We have developed our own audio settings for players in-game, in which the audio settings are controlled and managed by the **Music audio controller**. Our players can open the UI to set their preferences of the music volume, effects, and so on.

- **Customized control settings**

For customizing the key of a track in the song stage, we built a **Track Key Manager**. The **track key manager** stores the key code of each note track, managing the dropdown for each track to save or change the key for the player. Therefore, the player can change the key value of each note track during the song stage playing, for example, the key of track 1 from **A** to **B**.

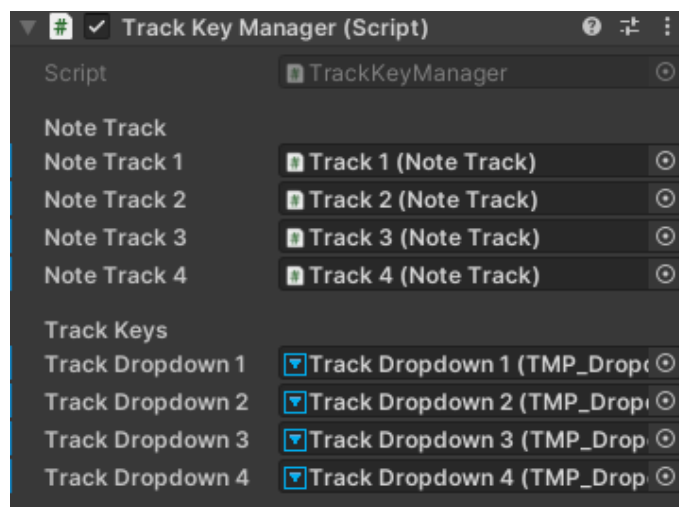


Diagram 2.4.17: Track key manager inspector

- **Hidden surface removal by Object Space Algorithms**

For each original game object in the scene, we have to first compute the part that is not occluded by any other objects, then draw. Unlike reality, there is no such automatic elimination that takes place when objects are projected onto the screen coordinate system in computer graphics generation.

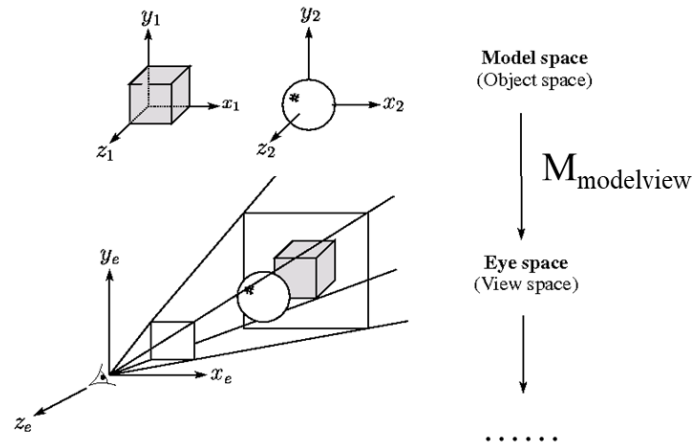


Diagram 2.4.18: Visualization of Object Space Algorithm

The object space algorithm will therefore be applied to determine the visibility of the object parts by comparing each object with all other parts of objects within the scene. The visible, invisible, or hardly visible surface could be determined after comparison [2].

2.5 Integration and Testing

- **Login and registration system**

A login screen is designed and implemented for the login and registration system, which will be the first screen showing up after the game starts.

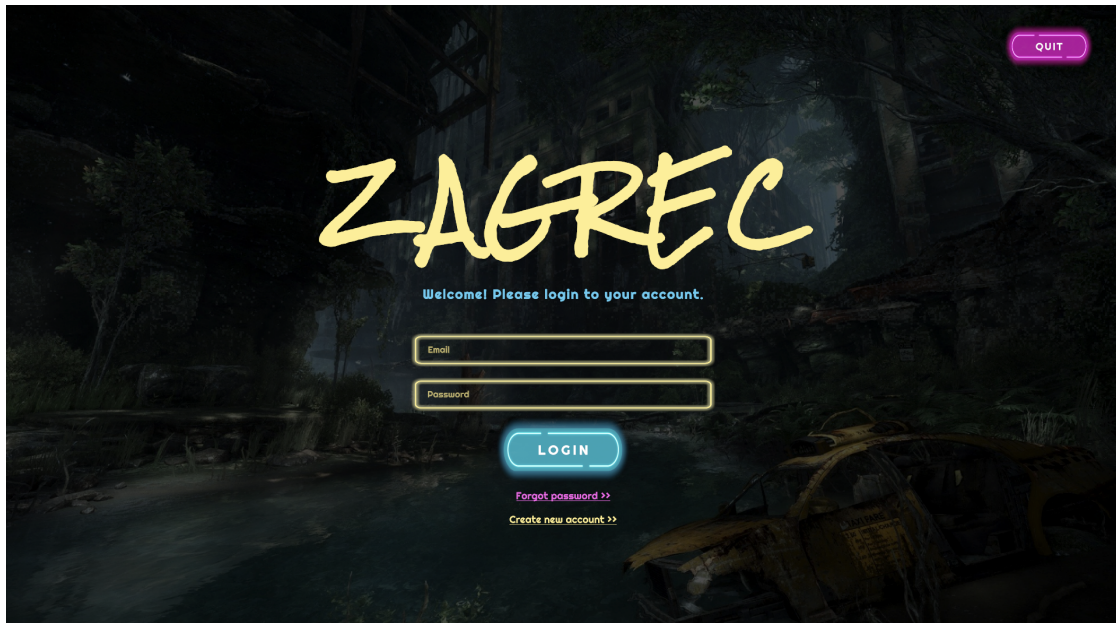


Diagram 2.5.1: ZAGREC login screen

After building the login and registration system with the backend services of **Firestore Authentication**, we have tried to register different user accounts and tested the whole login and registration process, as well as the authentication rules and functions of the database. The login system ensures players can only access their own data with unique access identifications or credentials.

ID	識別資訊提供者	建立日期 ↓	登入日期	使用者 UID
raytuin0523@gmail.com	✉	2022年4月1...	2022年4月1...	miS1ixjGCUNIVZafk0fesbONc1A3
fionakng121@gmail.com	✉	2022年4月8...	2022年4月1...	s5s47mAzoubEEeDz6nEmIzqvLFO2
skkongaa@connect.ust.hk	✉	2022年1月1...	2022年4月1...	JOMFwbN1fCY7Kp7rnaHl6MNdM...
3rdbrian@gmail.com	✉	2022年1月7...	2022年1月7...	0QXxv3MTZGWESyg3uFeucDTbT...
jedtam520@gmail.com	✉	2021年12月...	2022年4月1...	frch7KK6DEhTWIGadQKZFWHDp4...
kongskwan@gmail.com	✉	2021年10月...	2022年4月1...	sl8itT3licMEpg3J5R1fuy6x4Qk1
zhanzhang46@gmail.com	✉	2021年10月...	2022年4月1...	GBdBxnNUP2ZndXeAWITKb4D9qll2

Diagram 2.5.2: ZAGREC accounts in Firestore

● **Firestore database setup**

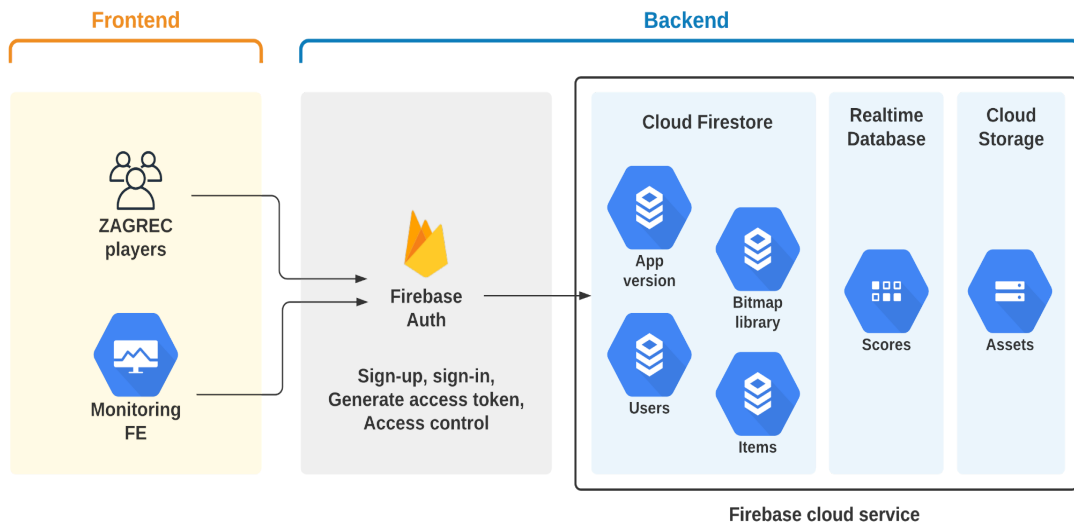


Diagram 2.5.3: Basic setup of ZAGREC database and backend infrastructure

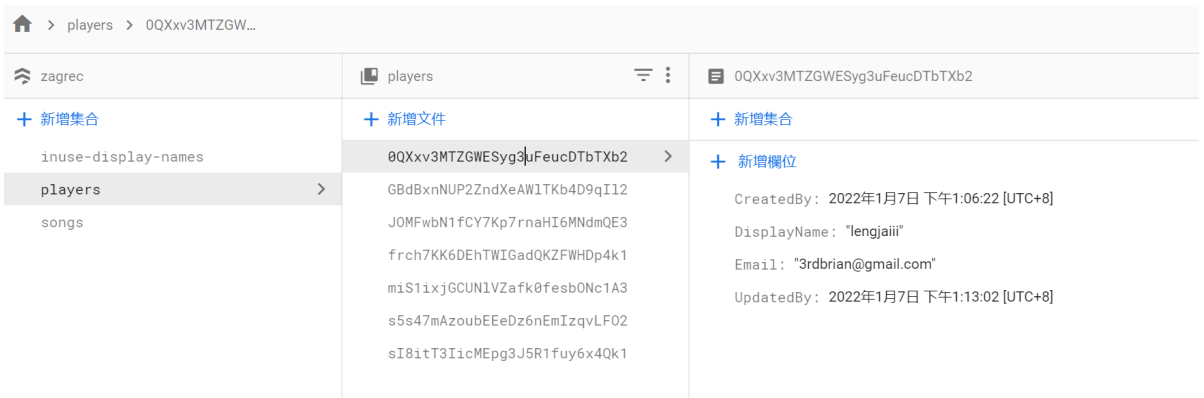


Diagram 2.5.4: Player's data in **Firestore**

A **NoSQL cloud database** has been used to store and sync data for the game. To test the data writing and reading capabilities of the Firestore database, a set of player data such as display name, score, and email have been created. The player's data can be updated to and fetched from the database regardless of network latency or Internet connectivity.

- **Loading screen**

We have developed different loading screens interpolated between scenes. A loading screen acts like an intermediate between game scenes, which can provide some important information about the upcoming stage or scene, and convey game-related tips. It also keeps the player entertained in other ways while the game objects are initializing and the game-play world is loading.



Diagram 2.5.5: Loading screen in **ZAGREC**

The loading screen is built with 2D graphic objects rendered by the **Sprite Renderer** in **Unity**, while the loading bar is presenting the scene loading operation. In the backend, we first have to load the scene that needs to be switched. Having this loading operation not yet finished, we interpolated the loading objects to it, so the loading screen can be updated for every frame with the loading operation progress.

```
while (!LoadingOperation.isDone) {
    float progress = Mathf.Clamp01(LoadingOperation.progress/0.9f);
    _progressSlider.value = progress;
    _progressText.text = $"Now Loading... {progress * 100f}%";

    Debug.Log(message: $"tips: {_tipsText.text}. progress: {progress}");

    yield return null;
}
```

Diagram 2.5.6: C# Script for loading screen implementation

Once the loading progress is finished, the player will then enter another scene.

- **Password reset**

Players can reset their password for **ZAGREC** whenever they forget or want to change a new password. Clicking the “**Forgot password**” button on the login screen will direct us to this screen.

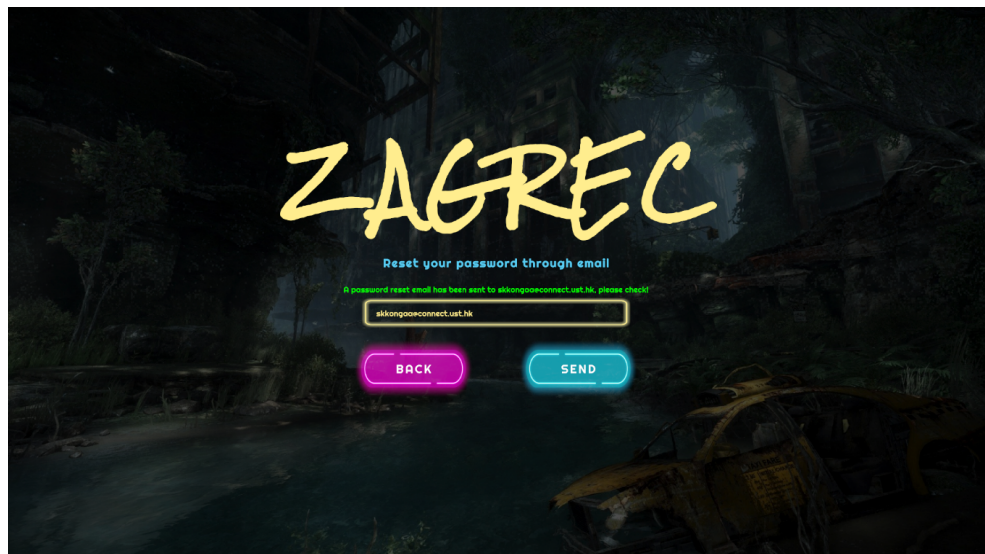
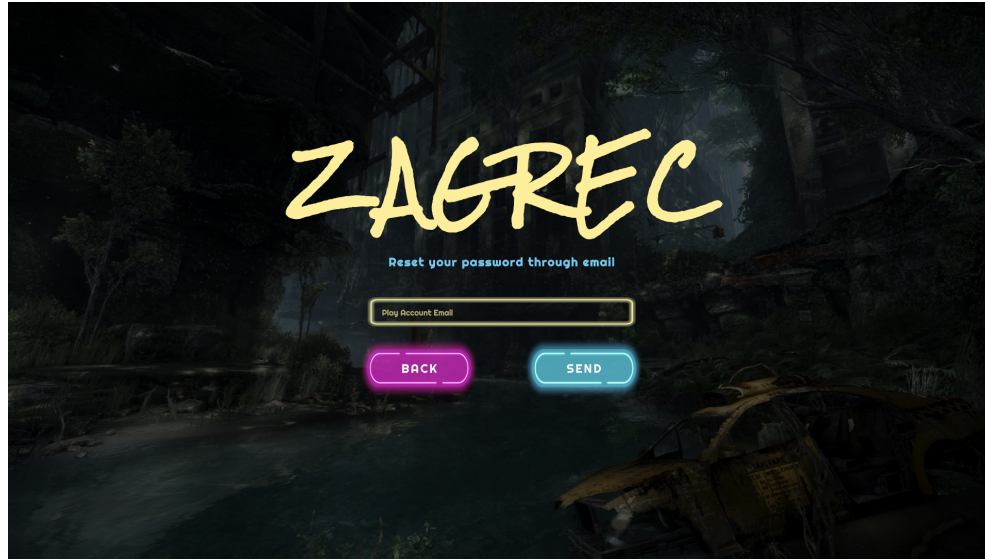


Diagram 2.5.7 & 2.5.8: Password reset screens in **ZAGREC**

An email will then be sent and players can then reset their password through the link provided in the email.

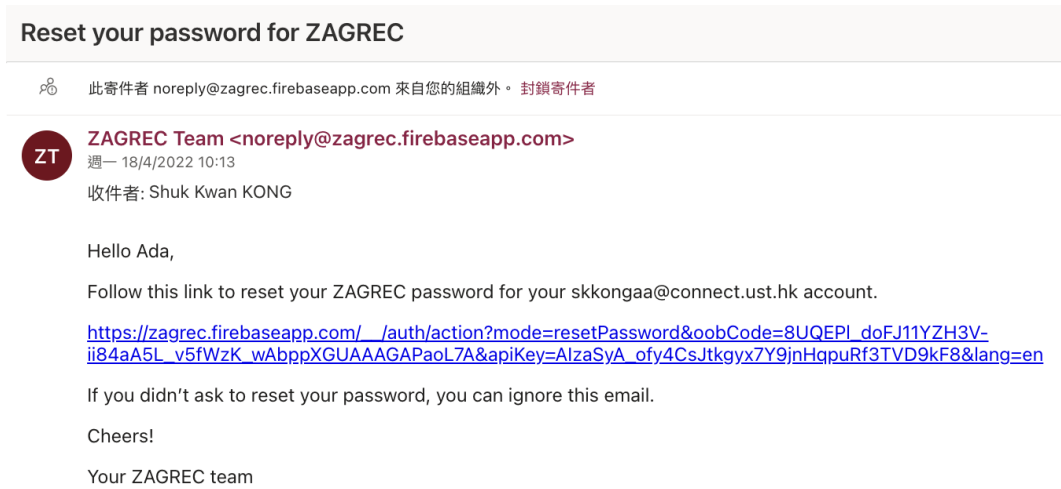


Diagram 2.5.9: Receiving a password reset email

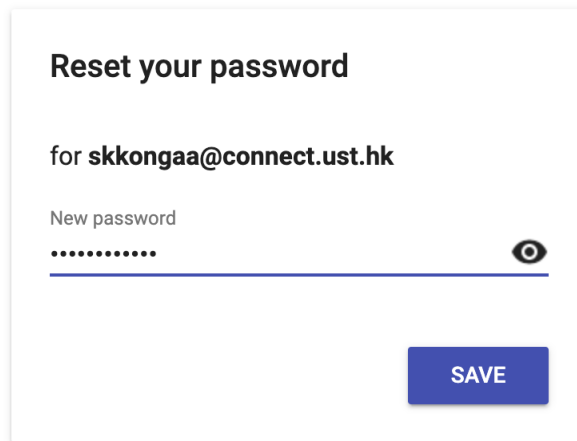


Diagram 2.5.10: Reset password after clicking on the link

We have tested this system by sending a password reset email to our email address and trying to login with the new password. Therefore, if the player forgets their account password which requires logging in to the game, they may use the forget password function to request from our system. And our **Firestore** backend server will then send a password reset email to their registered email address.

Players can press the link once they receive the email, and a pop-up screen will be shown asking for a new password for the game account. Lastly, players have to click the **"SAVE"** button to complete the entire process.

- **Firestore security rules**

We have set up the Firestore security rules that help to secure our backend data, and manage infrastructure, server-side authentication, and authorization code. Security rules provide access control and data validation in a simple yet expressive format, which builds user-based and role-based access systems that keep our players' data safe.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    // Allow only authenticated content owners access
    // Make sure the uid of the requesting user matches name of the user document.
    // The wildcard expression {userId} makes the userId variable available in rules.
    match /inuse-display-names/{displayNameId} {
      allow read: if true;
      allow update, delete, create: if request.auth.uid != null;
    }
    match /players/{userId} {
      allow update, delete: if request.auth != null && request.auth.uid == userId;
      allow read, create: if request.auth != null && request.auth.uid != null;
    }
    match /songs/{songId} {
      allow read, update, create: if request.auth != null && request.auth.uid != null;
    }
    match /songs/{songId}/scores/{userId} {
      allow update: if request.auth != null && request.auth.uid == userId;
      allow read, create: if request.auth != null && request.auth.uid != null;
    }
  }
}
```

Diagram 2.5.11: Firestore security rules

The rules have been tested with **Firestore** Authentication and **Cloud Firestore** services, so we can define what data our players can access for that by matching a pattern against database paths, and then applying custom conditions to allow access to data at those paths.

- **Ranking board**



Diagram 2.5.12 & 2.5.13: Ranking machine and ranking board

A ranking board is implemented and integrated into the 3D world scene, where players can walk to any ranking machine to view the ranking for every song. We

have tested the real-time ranking of all players in **ZAGREC** and it can be correctly shown on the board.

```

Frequently called 1 usage
public async Task<Dictionary<object, object>> GetAllScoresByASongAsync(string songName) {

    Dictionary<object, object> playerWithScore = new Dictionary<object, object>();

    // List<ScoreModel> scoreModels = new List<ScoreModel>();

    Query songNameQuery = _firestore.Collection(SongsDataPath).WhereEqualTo("SongName", songName);
    await songNameQuery.GetSnapshotAsync().ContinueWith(async task => {
        QuerySnapshot songNameSnapshot = task.Result;

        foreach (DocumentSnapshot documentSnapshot in songNameSnapshot.Documents) {
            Query getAllScoresQuery = documentSnapshot.Reference.Collection(ScoresDataPath).OrderByDescending("Score");

            await getAllScoresQuery.GetSnapshotAsync().ContinueWith(continuationAction: getScoreTask => {

                QuerySnapshot allScoresSnapshot = getScoreTask.Result;

                foreach (DocumentSnapshot scoreSnapshot in allScoresSnapshot.Documents) {
                    Dictionary<string, object> playerDictionary = scoreSnapshot.ToDictionary();
                    playerWithScore.Add(GetPlayerByIdAsync(scoreSnapshot.Id).Result.DisplayName, playerDictionary["Score"]);
                    // ScoreModel song = scoreSnapshot.ConvertTo<ScoreModel>();
                    // scoreModels.Add(song);
                }
            }, TaskContinuationOptions.AttachedToParent); //Task
        }
    }); //Task<Task>

    return playerWithScore;
}

```

Diagram 2.5.14: C# script for getting all player's scores from the database

All players' rankings will be calculated by the scores obtained in each rhythm game stage that is also automatically saved to the **Firestore**. By the function **GetAllScoresByASongAsync()**, we can fetch the scores of all players by a song name, and the data can then be updated and displayed on the ranking board in real-time.

- **Rhythm game stage**

After developing the 2.5D rhythm game stage, we start testing the game stage to make sure the music note of the song can be spawned and de-spawned correctly according to the song-playing time.



Diagram 2.5.15: Development layout vs Gameplay interface (current)

The BPM of the MIDI file was previously set as original or half the BPM to create a beat map that matched the song's BPM. The beat map is read from the MIDI file and the music notes are generated according to the notes information in the MIDI. During the game test, the notes (white objects) are successfully generated on the linear plane and fall from top to bottom. The notes will disappear once the player does the click action or exceeds right after the baseline on the bottom.



Diagram 2.5.16: Autoplay toggle

In addition to manually testing on the song stage, we also implemented an auto-play function to test the game progress and function.

2.6 Usability Testing

In order to dig into some possible user pain points and get more useful feedback for potential improvements, we have invited some of our friends to do thorough testing on ZAGREC.

- **Instruction**

The test is performed purely to test whether the design is easy to understand and where we can improve the design layout. Users will be given multiple tasks to perform and they will just have to try to complete the tasks without further guidance. There are no right or wrong answers, we are testing the game but not our users.

- **User 1**

Test Date: 2/4/2022

User Tested: Ray Tuin (a Biotechnology student, who likes to play 3D RPG and multiplayer online games)

Testing Results:

Tasks	Notes	User Feedback
Task 1 Register a new account	User has no problem finding the way to create and verify a new account	It is good to have this clear instruction and intuitive interface to create new account
Task 2 Login and start the game		The “Logout” button can be in another color like red, different from the other two buttons to avoid accidental click
Task 3 You would like to adjust the game settings now		
Task 4 You are happy with the settings now, you may want to open the map and search for a song stage	User has no problem identify the song stage icon in the map, but a bit confused to find the actual location because the song stage may not be on ground floor	
Task 5 Enter a song stage and play	The user has skipped the conversation	It is a bit too long for the dialogue before entering stage

Task 6 Where would you go to refill Zagro?	User walk straight to the vending machine with the Health Booster icon to refill Zagro	
Task 7 Search for Ranking machine to view player rankings for different songs		

- **User 2**

Test Date: 4/4/2022

User Tested: Natalie Wong (a Computer Science student, who likes to play 3D free-roaming RPG and online games)

Testing Results:

Tasks	Notes	User Feedback
Task 1 Register a new account	User has no problem finding the way to create and verify a new account	
Task 2 Login and start the game		
Task 3 You would like to adjust the game settings now		
Task 4 You are happy with the settings now, you may want to open the map and search for a song stage		
Task 5 Enter a song stage and play	User has no problem during the gameplay	It may be better to have custom keys
Task 6 Where would you go to refill Zagro?		
Task 7 Search for Ranking machine to view player rankings for different		

songs		
-------	--	--

- **User 3**

Test Date: 4/4/2022

User Tested: Nicky Wang(an RMBI student, having no rhythm game experience)

Testing Results:

Tasks	Notes	User Feedback
Task 1 Register a new account	User has no problem finding the way to create and verify a new account	
Task 2 Login and start the game		
Task 3 You would like to adjust the game settings now		
Task 4 You are happy with the settings now, you may want to open the map and search for a song stage		
Task 5 Enter a song stage and play	User failed to pass the gameplay	“The notes are on beat yet I am not reacting fast enough to catch them.”
Task 6 Where would you go to refill Zagro?		
Task 7 Search for Ranking machine to view player rankings for different songs		

2.7 UI/UX Improvements

After conducting usability tests, we have improved most of the game features and UI elements in **ZAGREC** to provide a better gaming experience to players:

- **Animation**

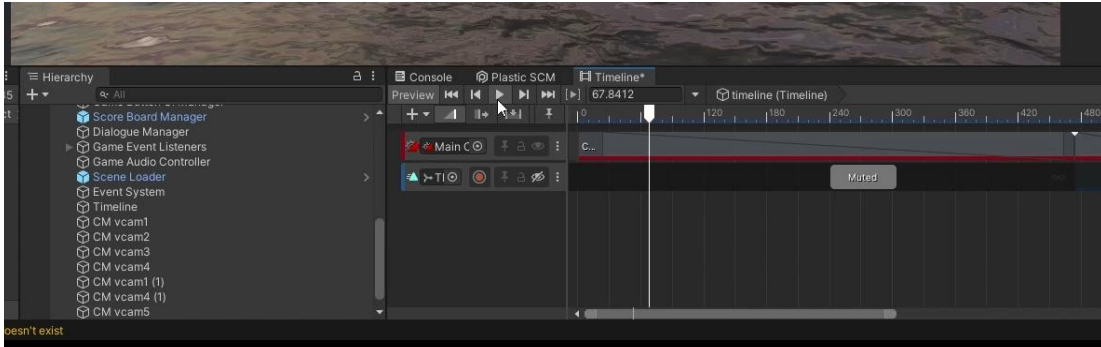


Diagram 2.7.1: Making of animation

To provide clearer storytelling in the game, there are animations for entering each chapter of the story of the game. This is helpful to explain scenes that could not be fully told in the text. **Cinemachine** and **Timeline** in **Unity** are applied to assist with any camera movement designs and animation [6]. We first applied the **CinemachineBrain** plugin to the main camera and added the **CinemachineVirtualCamera** to the scene, then we created an empty object as the playable object for the timeline. Creating a **Cinemachine track** in the **Timeline** allows us to import the **CinemachineVirtualCameras** and virtual camera shots to the **Timeline** for editing the animation. The **CinemachineBrain** has presets for the blending of the virtual camera positions, in which stacking two virtual camera shots in the **Timeline** means to have one shot taken with a smooth camera move between two virtual camera positions. The duration of the camera movement is also adjustable. All the animation clips are recorded through screen capturing for further editing.

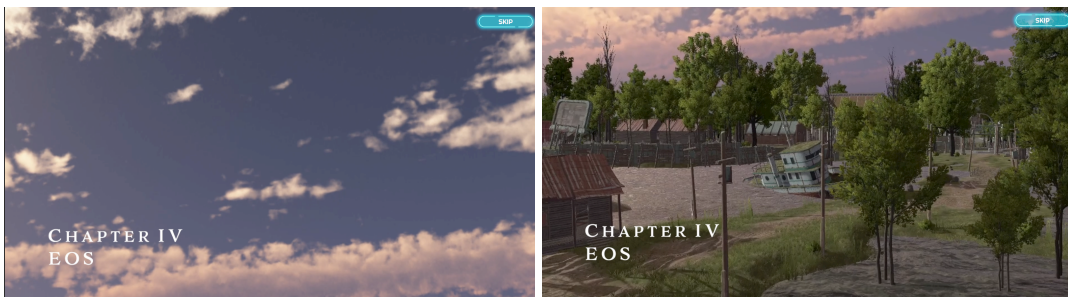
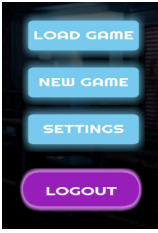
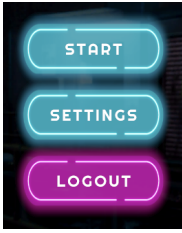
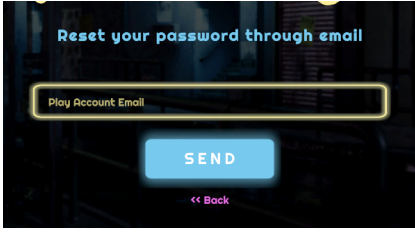
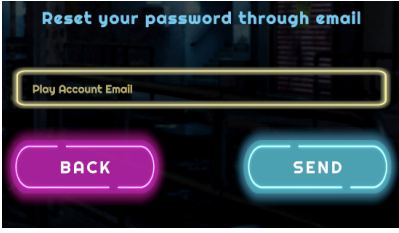


Diagram 2.7.2 & 2.7.3: Intro animation screenshots

A **Skip** button is implemented on the top right corner to allow players to skip the animation.

- **Button Style**

To provide a better outlook of buttons in-game, we have redesigned the buttons to present a clearer text on buttons and meanwhile keep the electronic vibes with neon light effects. The use of primary colour (cyan blue) and secondary colour (magenta) is applied to help users identify different buttons. We tend to use the primary colour for some engaging moves like starting the game, and use the secondary colour for buttons like “Logout”, “Back” and “Cancel”. This improves the user experience as players will tend to click on the buttons with a primary colour and be more aware of the buttons coloured in magenta (or pink), accidental clicks of buttons could be possibly avoided in this way.

Before	Improved
	
	

- **Shortcut help**



Diagram 2.7.4: Buttons with titles and shortcut keys explained

Apart from the aforementioned button style, the shortcut keys in the 3D gaming world are explained with corresponding icons and text for better understanding. Since the mouse is used for adjusting the angle in-game, this helps players get to know how to open the menu for functions like game settings and the full map with shortcut keys on the keyboard. For example, pressing “Q” to open settings and “M” to open the full map:

- **Song Stage**

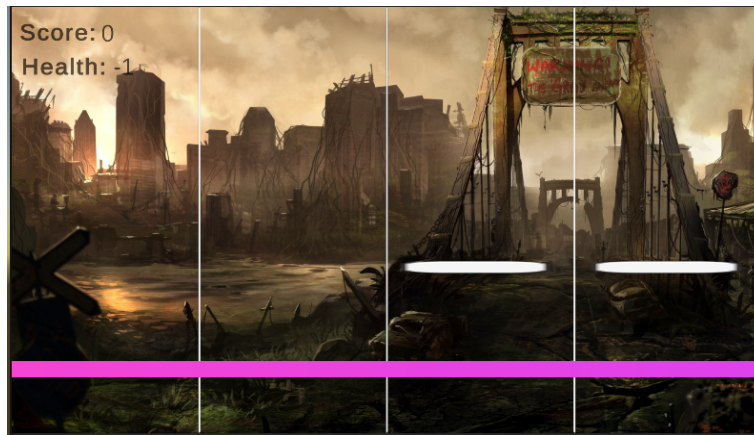


Diagram 2.7.5: First working prototype of 2D rhythm game playing scene

The first working prototype of our rhythm game stage has been developed in a flat 2D workspace that contains 4 tracks (corresponding to A/S/K/L keys in default) and a baseline on a 2D surface. The notes will move from the top direction to the bottom direction.



Diagram 2.7.6: Finalized rhythm game playing scene in **ZAGREC**

After several discussions and adjustments, the rhythm game scene UI is finalized as shown above. Unlike the previous 2D prototype, the current one applies 2.5D perspectives on the four music tracks, the song information board on the left, and the real-time updating scoreboard on the right, where players will feel like the music notes are successively coming to them from a far distance. This will allow players to react faster and ease the difficulty of the game. During gameplay, the clicked track will be brightened to show that the player input has been read and provide a click feedback effect. The background varies in every song stage, subject to the theme or meaning of that song. Song thumbnails are also created for each song for uniqueness.

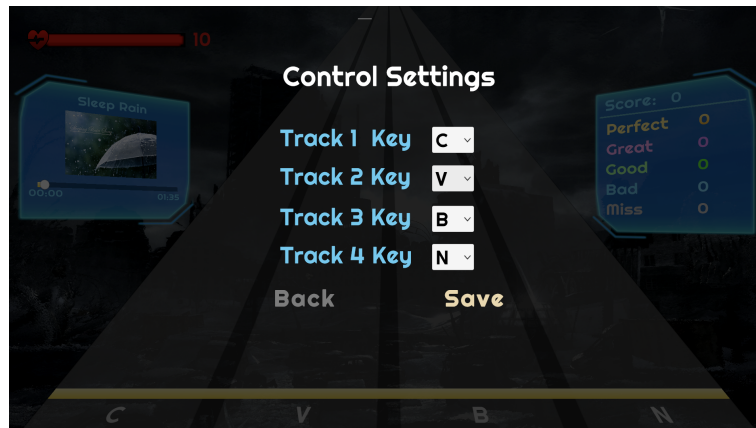


Diagram 2.7.7: Player controls customization

To further improve the gaming experience, note hit sound effects, combo, health bar, and custom key feature are added. Flare effect and coloured grade fonts show up when the player hits the note. There is a combo counter centred above the judgment line which updates itself during the play. If a user makes a bad or miss-hit, the counter counts from zero again. The health bar is located at the top left corner of the scene and will be shortened if the player misses any note. The player can change the key setting by pressing the ESC key to pause the rhythm game and clicking on the setting page. The keys cannot be duplicated and players are allowed to choose distinct keys that they want for the four tracks.

- **Ranking board**

This first prototype of the ranking board has been showing only particular songs.

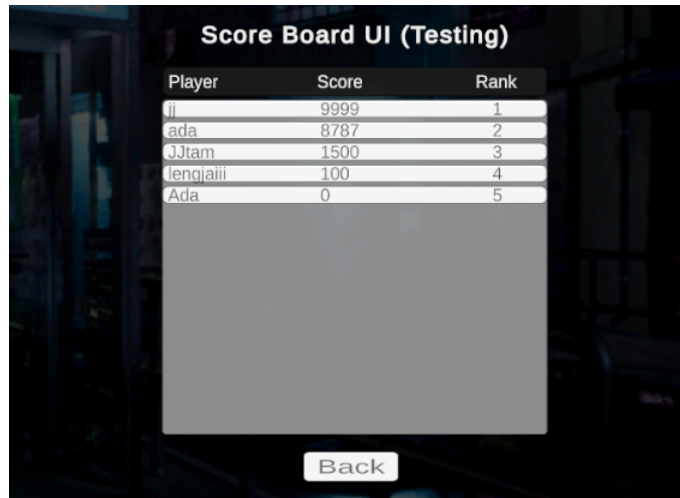


Diagram 2.7.8: The first prototype of the ranking board

It is now improved to a styled retractable ranking board with screen preparation loading and better error handling. The Player's own record and ranking are shown immediately at the bottom for convenience. The ranking board has been updated and improved to different songs, all song scores, and the user's own ranking. A ranking board manager is implemented to manage the related API call from the **FireBase** scores and song list, the construction of UI objects, and error handling as well.

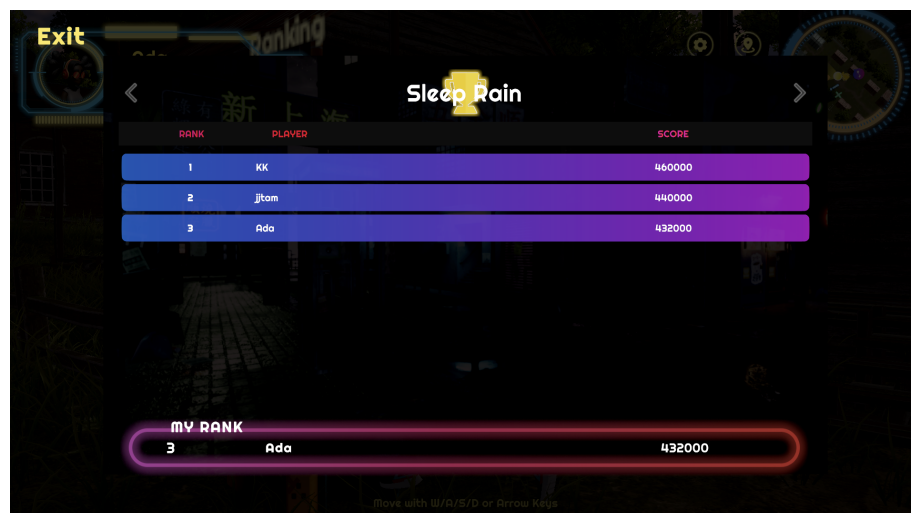


Diagram 2.7.9: Finalized design of scoreboard

- **Customized Player Name**



Diagram 2.7.10 & 2.7.11: Player info

To provide more customization in the game, players are allowed to change their in-game displayed name through **Account Settings**. For example, if a player changes the name from “Ada” to “Bda”, it will be updated and shown in the 3D gaming environment on the top left corner, inside the player profile.



Diagram 2.7.12 & 2.7.13: Player profile shown in 3D world scene

Yet, we will also display this name in the player rankings board as this display name is unique for each player; once the player changes to this name (and is case-sensitive), other players are not allowed to rename themselves to this name anymore. So it provides a unique display on the ranking board without making any confusion.

- **Zoom-in Camera in Dialogues**



Diagram 2.7.14: Camera blends when we start a conversation with NPC

To give a close-up and focus on the dialogue conversation between the player and the target. It is implemented using the **CineMachine Camera**. The camera blends from the player’s angle to a closer perspective when the player has triggered the dialogue and eases out when it is over.

- **Sound Effects (SFX)**

In order to have a more immersive and realistic gaming experience, background music and some sound effects are added to the game scenes and character movements. Having added more sound and visual effects in the song rhythm game may entertain the player, and help them relax and focus on the note they are going to tap. It also helps to increase the playability of our game.

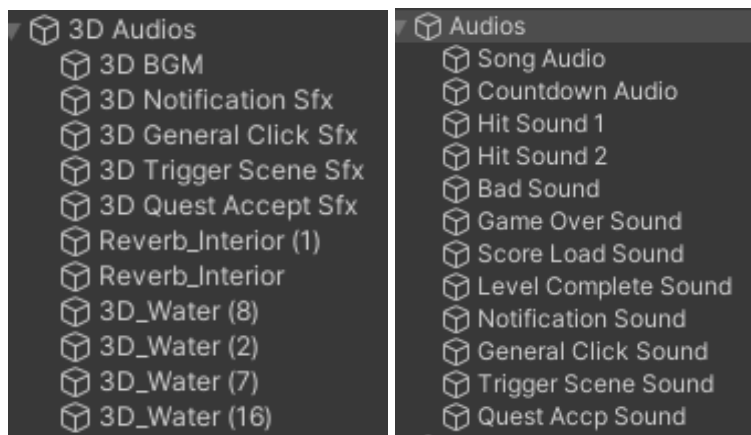


Diagram 2.7.15 & 2.7.16: Hierarchy of 3D world audios and song stage audios

Background music: Soft music is a good choice for the background music of a game. The music pieces are edited to be suitable for playing recursively.

Rhythm gameplay scene: There will be a countdown SFX after the player starts the rhythm gameplay. The chosen SFX for hitting a note is a tambourine and electronic bass drum. Percussion instruments SFX are very suitable for providing a clear beat for the song and to the player. A DJ disk scratch SFX will be triggered if the player missed a note. When it comes to scoring calculation after finishing the song, a countdown typing SFX will be played and a stage clear SFX in a major key will be played afterwards when the scoreboard shows all the numbers.



Diagram 2.7.17: Character walking towards a song stage album

3D world scene: To match the scene we have built, water and wind SFX are added. The characters also have their moves with walking, running, and jumping SFX. Robots have their unique machine SFX. When the player walks nearby the song stage album, the player shall hear the song of the corresponding album. Depending on the direction and distance, the panning and the volume of the song vary.

General: Various click, notification, and UI trigger SFX for different buttons are added to the scenes.

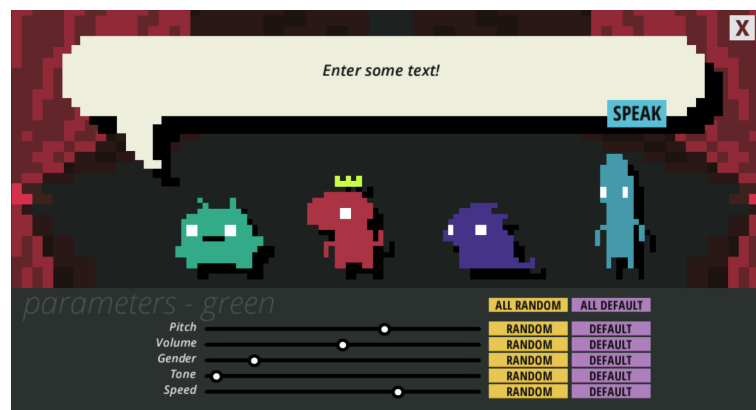


Diagram 2.7.18: Dialogue Engine interface

Animation Voice Acting: To create the random voice for characters, we applied **DialogueEngine** to generate vocal lines for the animation which can change its tone and pauses according to the punctuations we type in [7].

2.8 Evaluation

ZAGREC is developed with a high standard of graphics, animations, and 3D models. The game worked properly with a user-friendly interface.

Following a process of prototyping before actual implementation allowed us to evaluate different parts of the game in each iteration and make the corresponding improvements. For example, we have prototyped the login screen and loading screen with **Figma** and followed the design to implement the whole framework in **Unity**. This has saved much time in our development as everything can be changed easily with design tools like **Figma**, such as the colouring, positioning, size, and shape, but it becomes so hard to edit details like these in **Unity**. Therefore, it is always good to start with a great prototype before we actually implement it. The realistic and natural graphics in-game was created and improved by making every endeavour to adjust the material, light, and shadow for all 3D models.

To further evaluate **ZAGREC**, we should focus on checking whether all the implemented features work as intended, minimizing the bugs and glitches in the game, and making it a fun and enjoyable game. We have received feedback from the testers and made improvements in terms of the user interface and user experience throughout the whole user journey in our game. Here is a more detailed evaluation of our objectives:

1. To provide a user-friendly game interface

We accomplished a user-friendly game interface following some design principles like primary and secondary colours, proximity, and common region. The interface is built with some intuitive UI elements like icons and buttons. Most of our testers went smooth during usability testing.

2. To develop rhythm game stages with high accuracy

To provide an immersive and satisfying game experience to players, we have implemented the rhythm game stages with great accuracy, in which most of the notes are on the beat. This was done by numerous tuning on the speed of gameplay.

3. To develop intuitive player controls

The player controls in our game are explained clearly when they enter the 3D world, where a movement tip in the text is displayed at the bottom of the screen. In the rhythm song stages, players can customize the controls for the four tracks and the respective key will be displayed in text at the bottom of each track, which is intuitive.

4. To build a database

We have successfully integrated our game with a database to store and update various kinds of information needed in our game, including player profiles, scores attained by all players as well as the information of all song stages.

3 Discussion

This project aims to design and implement a 3D open-world music game that aims to alter the static impression of traditional music games built with linear and structured playing modes. We offered a unique and mythical storyline along with an immersive player experience. We have been focusing on providing a user-friendly game interface, developing rhythm game stages with high accuracy, developing intuitive game controls, and building a game database. This part will be talking about some discussions we have gone through regarding various kinds of issues we encountered during development.

3.1 Customized settings

In order to make the game as user-friendly as possible, we all agreed to make customizable game control settings for the player, including sound volume adjustments and keys used in rhythm game stages. Considering the variation of handspan and typing habits of the players, a customizable control is preferred.

3.2 Difficulties encountered

Custom Keycaps

For this part, we need to keep track of each keycode on each line, the stored value of the keycode needs to be ready to change when the player invokes the function during the song playing time. The difficulty is that we need to change the key for the server-side as well so that the player can have the correct key sets to continue the gameplay.

Rhythm game accuracy

During the black-box testing of the first 2.5D tracks, we discovered some problems with the game accuracy. Solutions for the problems were soon carried out for these. The table below explains the problems we encountered in detail:

Problem	Solution
The chart can be played with all 4 keys pressed together every time for preventing the wrong note is clicked	Algorithm modified. If the wrong key is pressed, a bad note is recorded.
Accidental clicks may also count as a MISS note	Give a range of area that starts calculating the reaction time for the notes to prevent counting the accidental/ fun clicks during the interlude
The miss note counting starts before the game for the "Press any key to start the	Algorithm modified. The counting starts only when the song is played.

game”	
Double count MISS note: if a note is missed once and the player presses the note for the second time when the note is still on the screen	* The current algorithm cannot solve this problem of closely packed notes *
long press BAD note → MISS note	* This situation exists if there is a note coming from the same track of the bad note *

Scoring system

After some tests, we found the player can reach S rank too easily so we adjusted the score calculation formula. We have added some calculations to the combo scores.

Original	Improved
Highest combo counting	A variable storing the highest combo number When there is a bad or a miss note, compare the current combo and the variable If the current is larger than var, store current into var.
Only showing the latest number of combo on the result board	During playing we show the current combo number Yet on the result board we will show the highest combo number
Combo score	Maximum combo/ 20 * 8000 + score

3.3 Future Directions

There are some potential advanced features that could be developed in future:

- **Free rhythm game stage**

A place where players can freely choose from all the unlocked songs to play with, without any limitations or pre-requisites to enter.

- **Difficulty levels**

Two or more difficulty levels can be developed, in which the **EASY** mode will be based on the vocal or main theme of the piece for newbies to catch up with the rhythm without focusing on two or more musical elements. Note patterns with less variation will be spawned and shown for a song. **HARD** mode is more likely to be the standard mapping version for the music as its notes are based on the vocal, bass, and drum beats. Note patterns with more variation will be spawned and shown for a song.

- **Variation in playing modes**

The current rhythm game stages only allow clicking, while in the coming days we will implement more variations to the gameplay, such as long-holding the key, involving the spacebar, or even the mouse.

4 Conclusion

To conclude, we are all grateful for the fruitful experience in creating the computer music game - **ZAGREC**. We managed to build it as a 3D open-world rhythm game that consists of the main storyline and some side quests with a lot of fun. The progress we have made is certainly satisfying and makes us excited.

Recalling our game developing journey, all of us started learning everything about game development from scratch as we all have no prior experience in this field. We have been keeping ourselves motivated all the way by our true passion for the field of game development. Undergoing countless trials and errors, we have learned to apply several useful plugins in **Unity** that help to develop a MIDI chart rhythm game and create smooth and storytelling camera moves in a timely manner. Searching for different tutorials and plugins that might be useful as well. Some tutorials might not be applicable as the latest **Unity** version is not compatible with or supports the mentioned methods. Sometimes small features in the game like minimap or custom keys are not as simple as we think, they may even take a longer period of time to develop as they are not just a few lines of code.

There is still room for improvement in **ZAGREC**, such as the accuracy of rhythm song stages and some technical issues like latency. There may probably be a better algorithm design to avoid the double-counting of BAD and MISS notes and the dialogue system can be possibly upgraded to support multiple character conversations. The graphic design of the game may also be improved to have more delicate artwork and assets. The scale of the usability tests can be larger so we can have more user feedback to further improve the overall gaming experience, even the small details matter.

Last but not least, although there is a pandemic that causes a certain extent of inconvenience and embarrassing situations, this project certainly connects all of us through the Internet and makes us help each other when any of us face challenges. Regular official meetings are vitally important but the daily communications about the progress are undeniably the one actually making the whole project done in time. All of us enjoyed walking through every step to make **ZAGREC** alive and we are truly proud to have this as our final year project in **HKUST**.

5 References

[1] Unity Asset Store [online] Available at:

<https://assetstore.unity.com/top-assets/top-paid?q=town%20buildings&orderBy=1>.

[2] Computer Graphics - Hidden Surface Removal [online] Available at:

<https://www.javatpoint.com/computer-graphics-hidden-surface-removal>

[3] XionUzuki, PCtyx [online] Available at:

<https://github.com/XionUzuki/PCtyx>

[4] LMMS - A open-source, digital audio workstation [online] Available at:

<https://lmms.io/>

[5] Unity API - DryWerMIDI [online] Available at:

<https://melanchall.github.io/drywetmidi/>

[6]Unity tool - Cinemachine [online] Available at:

<https://unity.com/unity/features/editor/art-and-design/cinemachine>

[7]Dialogue Generator - Zack Bougucki [online] Available at:

<http://www.zbogucki.com/portfolio/dialogue-generator>

6. Appendix A - Literature Survey

Literature Survey

The information related to building **ZAGREC** in a better way falls into four categories. First, there is an increasing market value of the game industry along with the potential challenges. Then, we will conduct an investigation into the key game features that boost gaming enjoyment. The existing automated beat mapping technology and its limitations will also be covered. In the last part of the literature survey, we will review some popular music games of a similar kind and explain how **ZAGREC** will differ from the other games.

6.1 The market value of the game industry

The computer game industry evolves significantly from a technical novelty to a broad notion, establishing a new category of interactive entertainment, culture, and monetization. Pixelated displays and restricted sound effects have been transformed multidimensionally into lifelike and high-quality video games. In 2020, the gaming business earned globally \$155 billion in revenue, and this sector is anticipated to produce more than \$260 billion in sales by 2025 [1]. Music games, as a new type of gaming, leverage both the game industry and the music industry to create and inspire a new way of gameplays. The new gameplay piques the interest and stimulates the ingenuity of most players, allowing them to improve their engagement and performance in a game.

The computer game industry is recently in the fastest-growing progression. However, computer games are also facing several challenges that the industry needs to address in the future. With the rapid developments and releases of new games, these rises made the game market to be more saturated. Also, mobile gaming has become a new trend to deliver novel experiences to a heavily populated market. As a result, video games struggle to stay competitive. Moreover, a pay-to-win style fascinated players to spend money for in-game purchases, breaking the balance and fairness of the game. This would cause a huge impact on the financial situation of the global gaming industry.

ZAGREC will have great features implemented and players can just strengthen their characters and win without paying. We hope to provide a fun and interactive game experience for the players, where they can grab a fresh look at the music game that is getting dynamic instead of static gameplay.

6.2 Determinants of game enjoyment

Hypothesis	T-Statistic	P-Value	Support
H1: Game Story → Gaming Enjoyment	1.98	<.05	Yes
H2: Game Graphics → Gaming Enjoyment	2.13	<.05	Yes
H3: Game Sound → Gaming Enjoyment	1.36	=.18	No
H4: Game Length → Gaming Enjoyment	2.36	<.05	Yes
H5: Game Control → Gaming Enjoyment	3.33	<.01	Yes
H6: Gaming Enjoyment → Intention to Play Games	4.27	<.01	Yes

Table 1.1: Summary of Hypothesis Test Results [2]

It was investigated that the key game elements related to gaming enjoyment are the **game story, graphics, length, and control** [2].

Similar to movies, popular games always have a story in which players might attach their fantasies and their desire could be fulfilled in the virtual game world. Players are likely to attach feelings to the characters and immerse themselves into the game story. An appealing game story would evoke a sense of curiosity in players and they would then build an in-game character who makes choices in his/her own life in the virtual world, which explains the strong relationship between game story and game enjoyment. On top of that, game graphics are undoubtedly a key to boosting the liveliness of the virtual world. When everything in the game becomes more realistic, from a subtle smile to the blinking of the eyes of the characters, players will find themselves more engaged in the virtual world, and therefore more enjoyable.

Game length is the average amount of time that players spent on achieving the winning situation, in which they successfully clear all the stages and quests in the game. The length is positively related to gaming enjoyment as it determines how detailed the game story could be. If the length is too short, the game story will hardly be told in a satisfactory way. Every game has a unique and ideal length depending on the gameplay and storyline, and that helps the game to be more enjoyable.

To provide a more pleasant game experience, player control should also be smooth and intuitive. Control is essentially linked to game enjoyment as it will affect players' feelings on how smooth and manageable the in-game character is. If the control is hard to understand or is with some delay, players would not enjoy it much as they have to pay attention to the control.

ZAGREC will be following most, if not all, of the aforementioned great features during game development in order to provide a more immersive and enjoyable game experience for players. The virtual world will be in 3D with graphics of good quality, whereas the story will be fascinating and controlled at an ideal length. To provide an intuitive control, **ZAGREC** will be played with controllers, in which players are able to smoothly control the movement of the in-game character with the thumbsticks on the controller.

6.3 Automated beat mapping technology



Diagram 1.2: Taiko no Tatsujin GameInterface [4]

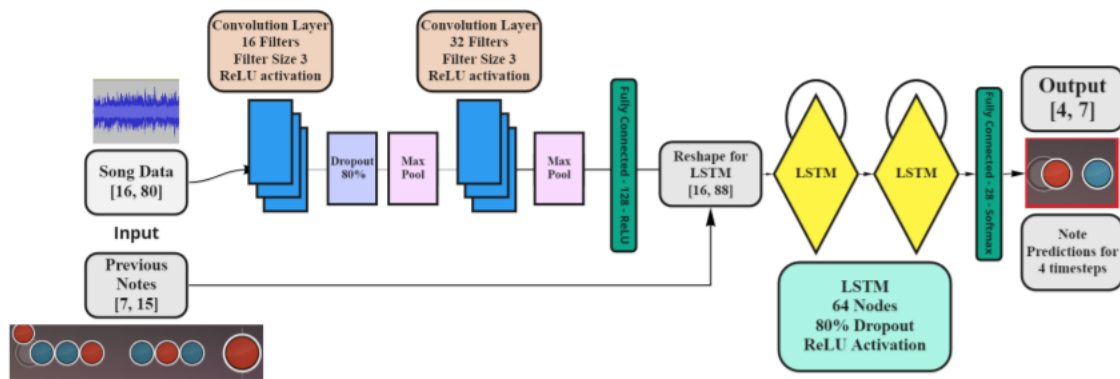




Diagram 1.3: System overview of chart generation pipeline with an LSTM DNN architecture [4]


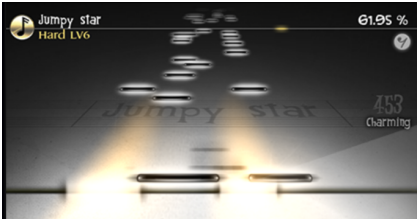
Existing technologies and studies for patterning-focused chart auto-generation are mostly developed by machine learning, like the music game TaikoNation. Applying the machine learning techniques notably helps the horizontal mapping rhythm games, which can be exemplified by the game Taiko no Tatsujin: the notes are moving horizontally and being clicked when they enter the area where player action is needed. To generate charts for such games, rule-based techniques and algorithms are applied to get the exact position where a note should be put. The LSTM DNN architecture, which was trained using large sets of human-authored Taiko no Tatsujin charts, has been applied to automatically generate charts. The note and song data help to predict the following four notes of the chart [4].

Yet, the technology of automatic chart generation by machine learning is considered to be not mature enough due to its limitations. The difficulty of the chart level would hardly be under control, for instance, where the system might simply generate a chart based on the song segments that include all components of the song. In this case, the generated song chart might have some notes packed together, in which the game

interface and player actions would not be that expressive, not to mention how challenging it would be for the beginners. Unless the system will also be trained to distinguish between the components or learn to read the midi file of the song to generate a more rational chart, generating charts of various difficulty levels with human-like patterns by the system still remains puzzling. Moreover, the system is still limited to generating single-direction-moving rhythm game charts, such as the ones for Taiko Master and Sound Voltex. The song charts for notes-popping mode rhythm games like Ctyus and Osu!, manual-made charts are still mainstream to ensure charting accuracy, even the corresponding new charting software (e.g. PCtyx, Cytunity, etc.) are developed by expert players of the game. The song stages in **ZAGREC** will be in notes-popping mode, where we can barely apply the machine learning techniques and algorithm for chart generation. In order to provide a more user-friendly game interface, we will manually do the beat mapping for every song used in the game stage, following the analysis of eye-tracking visualisation [5].

6.4 Review on other similar music games

	1. Music Boy 3D	2. OSU!
User Interface	 <p><i>Diagram 1.4: Music Boy 3D game UI [6]</i></p>	 <p><i>Diagram 1.6: OSU! game UI [7]</i></p>
Platform	Computer	Computer
Game style	Obstacle-dodging	Notes-popping
Game story?	No	No
Input device	<ul style="list-style-type: none"> - Controller - Keyboard - Camera sensor 	<ul style="list-style-type: none"> - Mouse - Keyboard - Touch device - Graphics tablet - Wiimote - Drum controller - Dance pad - Beatmania controller
Custom songs?	Yes	Yes

	3. Cytus, Cytus II	4. Deemo
User Interface	 <p>Diagram 1.7: Cytus UI [8]</p>	 <p>Diagram 1.8: Deemo UI [9]</p>
Platform	Mobile	Mobile
Game style	Notes-popping with a scan line	Drop-based
Game story?	Yes	Yes
Input device	Touch device	Touch device
Custom songs?	No	No

1. Music Boy 3D

The player needs to beat the note and dodge the obstacles while they are running. The game uses an MBOY editor to make rhythm charts, which allows players to create customised tracks and music to play [6].

Maps are randomly generated, so players can have a variety of gaming views and difficulty levels.

2. OSU!

OSU! contains 4 game modes. To link with our project **ZAGREC**, we would focus on the standard model from OSU!. There are 4 types of notes that appear randomly on the screen and click on the note when the surrounding circle has come close to the edge of the note [7].

Players might install extra controllers (e.g. tablet, GAMMON s620, pen) for gameplay to receive more accurate and fast-response clicks as compared with the computer keyboard.

3. Cytus, Cytus II

Cytus is a music game developed by Taiwan's Leia Games. Cytus II continues the gameplay of its predecessor, Cytus, with a dynamic judgment line that scans back and forth [8]. If the scanning line reaches a note, instructions are given according to the type of note.

4. Deemo

Like the other drop-based music games, Deemo is played by clicking on notes as they fall from the top of the screen to reach the judgment line. There are only two types of notes in the game, click and slide [9]. Unlike other drop music, Deemo's notes do not fall in a fixed path and can fall anywhere along the judgement line.

Each time the player finishes playing a song, the tree on the piano will grow according to the completion of the song, which is related to the progress of the storyline.

ZAGREC

Altogether, most of the popular music games on the market are static instead of interactive and dynamic. To make a breakthrough in music games, we would like to build **ZAGREC** by merging the music elements into a 3D open world, with an impressive storyline. Building a free-roaming virtual world not only provides a more interactive and mesmerising game experience to players but also allows them to explore every little detail of the game, even the “Easter eggs” - hidden messages of the backstory.

References

[1] Statista, "Video Game Market Value from 2020 to 2025". [online] Available at: <https://www.statista.com/statistics/292056/video-game-market-value-worldwide/>.

[2] J. Wu, P. Li, and S. Rao, "Why they enjoy virtual game worlds? An empirical investigation," Research Gate, Jan-2008. [Online]. Available: https://www.researchgate.net/publication/228364678_Why_they_enjoy_virtual_game_worlds_An_empirical_investigation.

[3] Taiko no Tatsujin: Drum 'n' Fun!. [online] Available at: <https://www.nintendo.com/games/detail/taiko-no-tatsujin-drum-n-fun-switch/>.

[4] E. Halina and M. Guzdial, "TaikoNation: Patterning-focused Chart Generation for Rhythm Action Games," 2021. [Online] Available at: <https://arxiv.org/pdf/2107.12506.pdf>.

[5] S. Almeida, Ó. Mealha, and A. Veloso, "Video game scenery analysis with eye tracking," Entertainment Computing, 22-Jan-2016. [Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/S1875952116000021>.

[6] Steam Games - Music Boy 3D. [online] Available at: https://store.steampowered.com/app/818620/Music_Boy_3D/.

[7] OSU! Music rhythm game. [online] Available at: <https://osu.ppy.sh/home>.

[8] Cytus | Rayark Inc. [online] Available at: <https://www.rayark.com/en/games/cytus/>.

[9] Deemo | Never Left Without Saying Goodbye [online] Available at: <https://deemo.com/>.

7. Appendix B - Meeting Minutes

7.1 Minutes of the 1st Project Meeting

FYP Group Meeting #1

Date of Meeting: Thursday, 29 July 2021

Time: 8:45 pm - 10:10 pm

Location: Online (Discord channel)

Attendee Names: Ada Kong, Fiona Ng, Gary Lam, Jay Tam

Agenda

1. **Decide which one we are going to build, mobile game or PC game**

	Mobile game	PC game
Graphics	Similar time and effort needed for both	
System Requirements	Consider more to ensure the game is smooth to run, overheating might happen	Consider less, more rooms for building various features in game
Responsive Design	Required	Not required

After consulting Professor Horner and reviewing the pros and cons of building mobile games and PC games, we decided to build a PC game for our FYP.

2. **Game Graphics**

- Feasibility of 2D/3D art
 - Plenty of time is needed to make great 3D art, but we would like to spend more time on building the music scores for the game as well as the main playing mode (music rhythm games) rather than graphics
 - We might go for the 2D art, pixel art in particular
 - Graphics could be improved in late-stage development

3. **Graphical Perspective**

- Third-person view would be implemented because the playing mode does not require a necessary first-person player view, unlike the shooting games

4. **Game Outline**

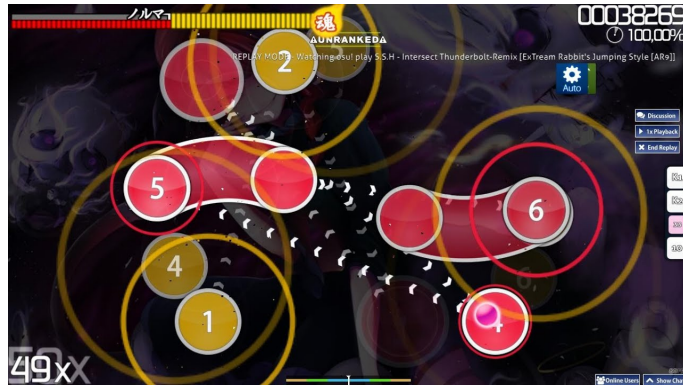
- Game type: story-driven music game
- Single player (main character) with sidekicks (NPC), they would be a music band, and each of them will have a representative colour
- Accuracy score (contribution) of each character will be shown in every stage we play or clear

- The corresponding instrument sound will be called whenever the representative button is clicked
- Rules to clear the stage: 70% or above accuracy
 - might vary among different levels of stage
- Taking reference from Pokemon GO, the user can walk around (with a little map on a corner)
- Story time range: would be wide, probably 1970s until present days
 - The time will change as the character clear and unlock different stages
- Intro animation of the game
 - Black out → spotlight on different characters → zoom in each character → description (background) of each character
- Music features:
 - Music game examples: OSU!, Audition Online, Taiko Drum Master, Cytus, Deemo
 - Implementation of the playing mode like the Audition Online will be easier, but we need to focus more on graphics (see picture below)



- We would instead like to build a game focusing more on the music rhythm as we found that it would be easier and more feasible for us to make great music scores for the game than to make stunning graphics
- Buttons output in our game: mainly pop out
 - referencing Cytus and OSU!





- Hong Kong elements in the game
 - Environment (landmarks or known places in Hong Kong)
 - The higher the player level, the more modern the environment
 - Examples: Big Buddha, Tuen Ma line, HKUST
 - Songs that are well-known among locals (TBC, copyright issues)
 - Cantopop
 - Greensleeves (DSE Listening)
 - Attack on Titan (4 OP)
 - Demon Slayer: Kimetsu no Yaiba (Homura, Gurenge)
- Rewards
 - Coins
 - Usage: buy new instruments, fix current instruments, clothes
 - How to get: clear any stage, finish any battle
 - EXP
 - Depends on accuracy
 - Medals (*advanced feature*)
 - Attain at least 90% accuracy in a stage
 - Collection of medals could be captured
- Control
 - Keyboard
 - Gamepad (*advanced feature*)

Open Actions

Who	Things to be done before next meeting
<p>Ada</p>	<p>*Research on Audition Online (playing mode) **Email Professor Horner Briefly make a UI of some screens Come up a story idea (background + main storyline) Get familiar with Unity</p>
<p>Fiona</p>	<p>*Research on Cytus (playing mode + VOC) & Deemo (playing mode) Get familiar with making music scores for games Come up a story idea (background + main storyline) Get familiar with Unity</p>

Jay	*Research on Taiko Drum Master (playing mode) Come up a story idea (background + main storyline) Get familiar with Unity Keep learning C#
Gary	*Research on OSU! (playing mode + VOC) Come up a story idea (background + main storyline) Get familiar with Unity

* Research on other games:

OSU!, Audition Online, Taiko Drum Master, Cytus, Deemo

- Playing modes (terminology)
- Implementation of music scores
- VOC (Voice of the customer)
 - Reviews on the game design (Pros & Cons)

Linkage to the implementation of our game

** Things to confirm:

- Copyright of songs (No need)
- Selection of songs (Every song)

Next meeting

FYP Group Meeting #2

Date of meeting: Thursday, 12 August 2021

Starting time: 8:30pm

Location: Online (Discord channel)

Agenda:

1. Review our learning progress on Unity, C#, music scores making
2. Report the research findings
3. Review UI
4. Proposal (deadline on 17/9)

7.2 Minutes of the 2nd Project Meeting

FYP Group Meeting #2

Date of Meeting: Thursday, 12 August 2021

Time: 9:45 pm - 11:55 pm

Location: Online (Discord channel)

Attendee Names: Ada Kong, Fiona Ng, Gary Lam, Jay Tam

Agenda

1. Review our learning progress on Unity, C#, music scores making

- **Unity & C#**
 - Unity 3D project, lightmapping
 - Player Control (RigidBody, speed)
 - 4 representative colours → 4 different keys on keyboard
 - "Finish mode" → Spin crazily
 - Graphics
 - Logic

- **Music scores making**
 - **Cytoid Charting**
 - We can refer to Cytoid's VOC.
 - The creator of Cytoid was originally a long-time fan of Raiya from the Cytus 2.0 era. In order to play more home-made Cytus music, he created this kind of "simulator", but unexpectedly more and more players participated in it.

 - **PCtyx**
 - <https://github.com/XionUzuki/PCtyx/releases/tag/v210>
 - User manual: [PCtyx Editor Manual](#)
 - https://blog.teages.xyz/posts-source/newPCtyx/PCtyx_Editor_CN.pdf
 - Supporting audio file type:
 - .ogg (Ogg Vorbis)
 - .m4a (MPEG-4 AAC)
 - .flac (FLAC)
 - .wav (PCM or PCM float)

 - **Cytunity**
 - [Cytoid - Introduction: Cytunity](#)
 - Finding key and BPM: <https://tunebat.com/Analyzer>
 - MP3 to .OGG : <https://audio.online-convert.com/convert-to-ogg>
 - Feedback:
 - PCtyx is more visual and easy to use as it has more visualised buttons and functions.
 - Yet both need preparation on node position design and detail beats time position for easier and efficient charting. Considering midi editor or audacity for catching beats time position.

2. Report the research findings

<p>Ada</p>	<p>Audition Online</p> <ul style="list-style-type: none"> - Implementation of music scores <ul style="list-style-type: none"> - Any kind of music, usually pop songs - Depends on different modes, they will have different music scores <ul style="list-style-type: none"> - For the basic mode, they will just have to mark every 4th beat of the song, and see how accurate players hit the beat - Reviews on the game design (Pros & Cons) <ul style="list-style-type: none"> - Pros <ul style="list-style-type: none"> - The music scores for the game could be fast and easy to build (e.g. the basic mode) - The songs used for the game are usually very popular and trending - Multivarious game modes to play with, players would not get tired or bored with the game easily - Social networking through gameplay, connect players all over the city - The game includes a Fashion Mall that allows players to customise their avatars - Cons <ul style="list-style-type: none"> - Since the game modes are easily implemented and intuitive to play, the graphics instead should be very detailed and appealing (for Audition Online, they make good graphics and all the dancing moves are smooth) <ul style="list-style-type: none"> - Time-consuming - Required advanced skills in computer graphics - Linkage to the implementation of our game <ul style="list-style-type: none"> - Fashion Mall → Instrument Store in our game <ul style="list-style-type: none"> - Take reference from the design of the mall - Visual effect of every bit the player hits
<p>Fiona</p>	<p>Cytus</p> <p>Playing Mode</p> <p>Players need to play the corresponding notes in time with the scanning lines moving up and down in the game interface.</p> <p>There are three types of tap buttons in the game: Click Note, Hold Note, and Drag Note.</p> <p>When the scan line overlays above a certain beat point, it operates according to the following rules</p> <p>Click on the note: When the scan line overlaps the note, click.</p>

Long press on the note: When the scan line overlaps the first dot of the note, press and hold the dot until the end of the note.

Drag the note: when the scan line coincides with the first dot of the note, keep the same speed as the scan line to cross the note track.

There are two types of notes in the game: blue-purple and blue-green, and the notes will show different colours when the scan line moves from top to bottom and from bottom to top.

A colored note indicates that a corresponding operation should be performed when the scan line sweeps over the note. A black ring will appear around the note when the scan line passes by, and the ring will be judged as a colored PERFECT when it is clicked.

The pop-up mode will set the way to affect the appearance of the note.

default mode: The note appears very small and will slowly get bigger as the scan line gets closer.

grouped mode: the notes appear very small, and when the scan line is ready to fold back, the notes on the next screen will get bigger together.

none mode: notes are normal size as soon as they appear, and non-same screen notes are represented as translucent.

click fx setting: when it is set to "on", there will be a 'ta' sound feedback when you click on the note for iOS, and a vibration feedback for Android.

Long press on the note to feedback at the beginning and end, drag the note to feedback at each point.

Difficulty Levels

All tracks in Cytus are divided into Easy and Hard according to their difficulty level, with Hard difficulty being more complex and varied than Easy.

Each score has its own quantified level of difficulty, expressed by the numbers 1 to 9, with 1 being the easiest and 9 being the hardest. Even so, some of the tracks in the game go far beyond the 9 difficulty level and are still represented as 9 difficulty.

Reviews on the game design (Pros & Cons)

- Pros
 - 175ms loose judgement (early no TP concept), even new players can also enjoy the game process, the difficulty is also moderate
 - the music is quite listenable, for a rhythm game, having a good choice of music is a must
 - the way to unlock the hidden song is quite interesting
 - static background, rely less on graphic and animation design.
- Cons
 - the problem of 175ms. Even though the concept of TP was added later, there is still no way to hide the fact that cytus judgments are looser than other rhythm games.

VOC (Voice of Customer)

Nick Tylwalk on toucharcade gave it 4.5 stars, saying that although there are free songs to play, the price of DLC makes the game actually seem like it costs \$22, but it will not quench the enthusiasm of music rhythm game lovers. Blink's NetGreen feels that it is more worth spending money on music games than on card-drawing games. NetGreen of Blink feels that it is more worth spending money on music games than on card draws, and that the songs may be continuously updated.

Pros

- static background
- rich graphics and in game animation
- varies note clicking methods

Cons

- new chapters has to be bought for continuing the story

Deemo**Game Progress**

Each time you finish playing a song, the tree on the piano will grow according to the completion of the song, with All Charming growing the most, followed by Full Combo. In version 1.0, the height of the tree is limited to 20 metres, while in version 2.0, it is 40 metres and the final 100% ladder, which will reveal the ending of Deemo and the character's life after passing the ladder. Version 3.0 adds the paid DLC "Forgotten Hourglass", which allows players to play the game in a round-robin fashion to obtain new tracks and storyline memory photos.

Gameplay

Like other drop-based music games, Deemo is played by clicking on notes as they fall from the top of the screen to reach the judgement line. There are only two types of notes in the game, click and slide. The black notes are clicked notes, which are clicked when they reach the judgement line. The golden notes are sliding notes, which are clicked when they reach the judgement line and slide your finger according to the position and speed of the note until the last note is released. Unlike other drop music, Deemo's notes do not fall in a fixed path, and can fall anywhere along the judgement line. In addition, the width of the notes is inconsistent, ranging in size from half to 1/3 the size of a normal note.

Judgments

Deemo has three types of judgments, Charming Hit, Hit, and Miss, and instead of texting the judgments during gameplay, rainbow or orange (for Charming Hit), green (for Hit), and light blue, purple, or no colour (for Miss) are used to represent accuracy. Also, if the player hits a Charming Hit, the word Charming will be displayed under the Combo count on the right side of the game interface.

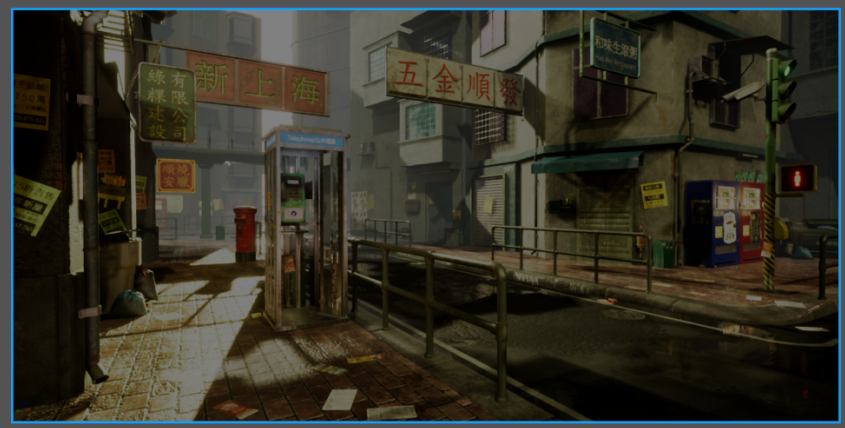
Special Features

	<p>NOTE EFFECT: During the song, black notes will float upward from the judgement line, representing the music surface that the piano is actually playing.</p> <p>TAP TONE: When you press the tap, the corresponding tone will appear, but it will not turn on if the correction value exceeds 0.05.</p>
<p>Gary</p>	<p>OSU!</p> <ul style="list-style-type: none"> - Description <ul style="list-style-type: none"> - <u>summary</u>: - Since OSU contains 4 game modes commonly appearing in the market, for this basic review would focus on the standard mode, which is the most popular game among OSU that could be seen frequently on the internet. <ul style="list-style-type: none"> - characteristic: PC games, online, rank, official beatmap source, - method(standard mode): <ul style="list-style-type: none"> - the 4 types of hit objects appeared randomly on the screen - hit the objects with higher accuracy scores higher, miss would cause damage till 0 and end the game. - Player might install extra controllers (e.g tablet, GAOMON s620, pen) as faster point combining with computer keyboards - simple rules but very challenging - players could create their own skins and rhythm combinations - 4 types of hit objects: <ul style="list-style-type: none"> - HIT circle: hit the circle when outer circle approach - Slider: press and follow the track - Reverse Slider: similar - Spinner: spinner it.... - reflection of difficulty(<u>difficulty index</u>): <ul style="list-style-type: none"> - CS (circle size): < → smaller circle - AR (approach rate): > → faster - OD (overall difficulty): > → higher accuracy is required get scores [300 100 50 miss] - HP (HP drain): > → ^damage/miss - Implementation of music scores <ul style="list-style-type: none"> - Any kind of music that could be downloaded from the official website - uncountable skins for each mode and difficulty from community - Score system is simple: scores are depends on difficulty and mode, basically comba * accuracy(mentioned above)

	<ul style="list-style-type: none"> - Reviews on the game design <ul style="list-style-type: none"> - Pros <ul style="list-style-type: none"> - The selling point of the game is basically the ranking, and difficulty challenges. No storyline and the huge music library allowed players to start and repeat the process with no restrictions. No need to lv up to unlock anything. - The indexes are good references of difficulty that we could consider importing these kinds of features or variables.
Jay	<p>Taiko Drum Master</p> <ul style="list-style-type: none"> - Playing modes (terminology) (from wikipedia) <ul style="list-style-type: none"> - Symbols moving horizontally along a timeline show what to hit and when - The beats represent a strike on either the drum head (red) or rim (blue). Large icons mean a double-stick strike, and the occasional balloon icon requires a rapid series of strikes to clear the staff. - A drum simulating the taiko is played in time with music - Successful play builds up a life metre(HP). If the metre is past a certain point by the end of the song, the song is passed - https://taikotime.blogspot.com/2010/08/advanced-rules.html - Implementation of music scores <ul style="list-style-type: none"> - Source https://ux.getuploader.com/e2351000 - Is based on the music of traditional Japanese festival drums - Some github sources: <ul style="list-style-type: none"> - https://github.com/Formula-9/TJA-Player-Vita - https://github.com/penandlim/TaikoBeats - https://github.com/sssss465/DDR-Processing - https://github.com/FeiZhaixiage/auto-taiko - https://gitlab.com/woodyZootopia/soundgame - Reviews on the game design (Pros & Cons) <ul style="list-style-type: none"> - Pros <ul style="list-style-type: none"> - Very simple, easy to play - Player can have zero skill with music-based games - Cons <ul style="list-style-type: none"> - No freestyling - Hitting two different areas of a drum is a far cry from the wacky analog stick gyrations - The limited selection of inputs/playing songs - And if you play without the drum controller, the game is even more ridiculously simple - Linkage to the implementation of our game <ul style="list-style-type: none"> - It can be one of our game modes/part of the game play, especially for drums playing where the user can play/make drum sounds with the beat/rhythm.

- When it is the drum's turn of the band's show.

3. Review UI





4. Report game story ideas

Ada	https://docs.google.com/document/d/1N-p8aMeNSNbgIN-onJwaBuka9WNeNE70z1YnBfm8wVc/edit?usp=sharing
Fiona	<p>Hypnotist takes a job to help a musician who died in an accident to solve his PTSD and enter his dream world.</p> <p>The world inside the dream is ruled by AI (subconscious mind), and music and art are forbidden in this world because AI cannot understand and produce art-related objects.</p> <p>The main mission is to find out the reason for his PTSD because of the newspaper clippings and the emergence of unexpected events.</p> <p>The protagonist's team is able to treat the problem, the AI is defeated and the cultural desert reappears.</p> <p>The protagonist exits the dream world and the musician is cured of his PTSD.</p>
Gary	<p>The main characters are 4 middle-aged men, previously played in a secondary school band. They haven't continued for a period of time because of busy work, so they tried to regain the happiness of having a band and going for challenges. (~ Pokemon Dojo)</p>
Jay	<p>The post-apocalyptic zombie virus outbreak, music to purify the human</p>

Open Actions

Who	Things to be done before next meeting
Ada	3D Graphics with Unity (make some scenes & characters) Search graphics
Fiona	Search graphics suitable for the story implementation for 2D pixel art Charting 1 minute song Research on merging the charting file with Unity
Jay	Unity Sound, Animation Keep learning C# Search graphics
Gary	Unity C# Notes Search graphics

Next meeting**FYP Group Meeting #3****Date of meeting:** Thursday, 2 September 2021**Starting time:** 2:00pm**Location:** Learning Commons**Agenda:**

1. Review our learning progress on Unity, C#, music scores making
2. Proposal (deadline on 17/9)

7.3 Minutes of the 3rd Project Meeting

FYP Group Meeting #3

Date of Meeting: Thursday, 2 September 2021

Time: 2:00 pm - 6:00 pm

Location: Learning Common

Attendee Names: Ada Kong, Fiona Ng, Gary Lam, Jay Tam

Agenda:

1. Distribution of work for proposal

Session	Person in charge
1.1	Gary
1.2	Jay
1.3	Ada Fiona Jay
2.1.1	Ada
2.1.2	Fiona
2.1.3	Ada
2.1.4	Jay Gary
2.1.5	Ada Jay
2.2.1	Ada Fiona
2.2.2	Fiona
2.2.3	Ada
2.2.4	Gary Jay
2.3	Ada Fiona Jay
2.4	Fiona Gary

2. Details to be added in proposal

- Elaborations on research findings
- Reference links marked in IEEE format
- Sample images
- Confirmed storyline
- Changed art style: from 2D pixel art to 3D model
 - after a few trials on creating graphics in Unity, 3D seemed to be much suitable for the theme of the game

Next meeting

FYP Group Meeting #4

Date of meeting: Thursday, 9 September 2021

Starting time: 2:00pm

Location: Learning Commons

Agenda:

1. Review our learning progress on Unity, C#, music scores making
2. Review Proposal (deadline on 17/9)

7.4 Minutes of the 4th Project Meeting

FYP Group Meeting #4

Date of Meeting: Thursday, 9 September 2021

Time: 2:00 pm - 6:00 pm

Location: LC

Attendee Names: Ada Kong, Fiona Ng, Gary Lam, Jay Tam

Agenda:

1. **Review on proposal progress**
 - a. Design, Implementation, Testing session done
 - b. Working on overview and organising literature survey
 - c. Controlling tool finalised: game console
 - d. Include minutes in the appendix
 - e. Add sample charting video in Appendix B

2. **Review on learning progress**
 - Shared unity project access tutorial by Jay
 - Demo of database and login page
 - Demo of song charts

Next meeting

FYP Group Meeting #5

Date of meeting: Thursday, 23 September 2021

Starting time: TBC

Location: Discord

Agenda:

1. Review our working progress on Unity, graphics, music scores making and database
2. Come up a detailed and specific schedule for each task
3. State the short-term goals for each task

7.5 Minutes of the 5th Project Meeting

FYP Group Meeting #5

Date of meeting: Thursday, 23 September 2021

Starting time: 2:30 pm

Location: Discord

Attendee Names: Ada Kong, Fiona Ng, Gary Lam, Jay Tam

Agenda:

1. Review our working progress on Unity, graphics, music scores making and database
2. Come up with a detailed and specific schedule for each task
3. State the short-term goals for each task

Next meeting

FYP Group Meeting #6

Date of meeting: Thursday, 30 September 2021

Starting time: 2:30

Location: HKUST Library Group Study Room LG4-09

Agenda:

1. Review login and registration system
2. Review song stage interface requirement
3. Discuss story chapter names
4. Discuss song list for different story chapters

7.6 Minutes of the 6th Project Meeting

FYP Group Meeting #6

Date of meeting: Thursday, 30 September 2021

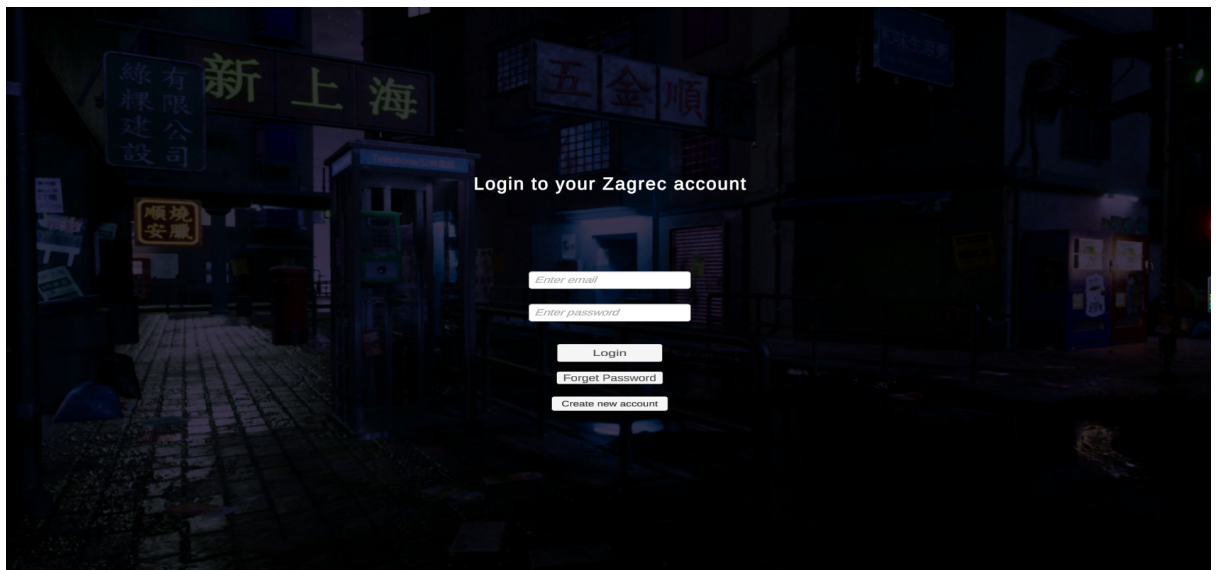
Starting time: 2:30 pm

Location: HKUST Library Group Study Room LG4-09

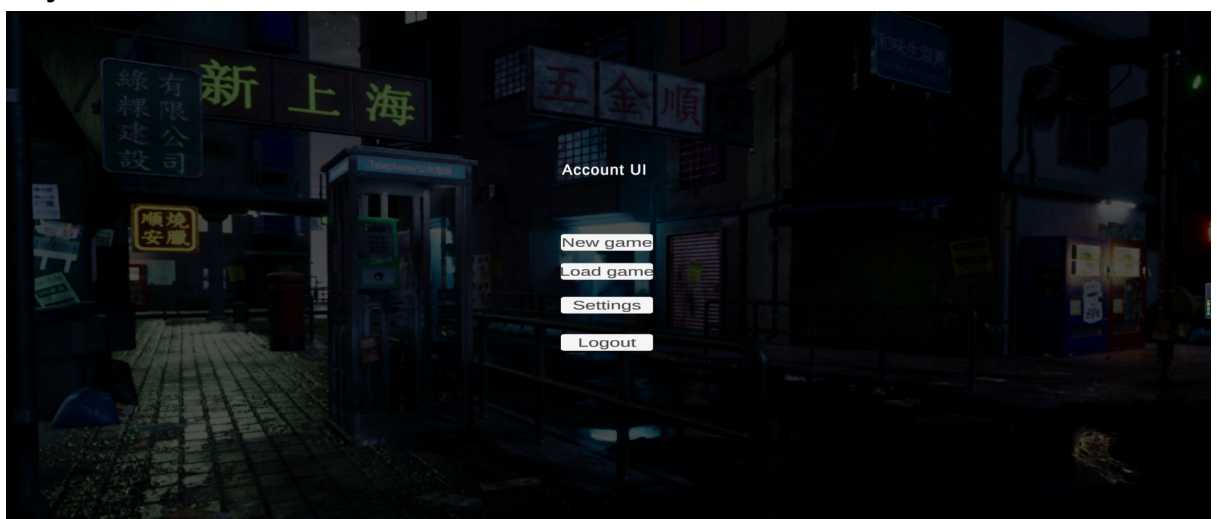
Attendee Names: Ada Kong, Fiona Ng, Gary Lam, Jay Tam

Agenda:

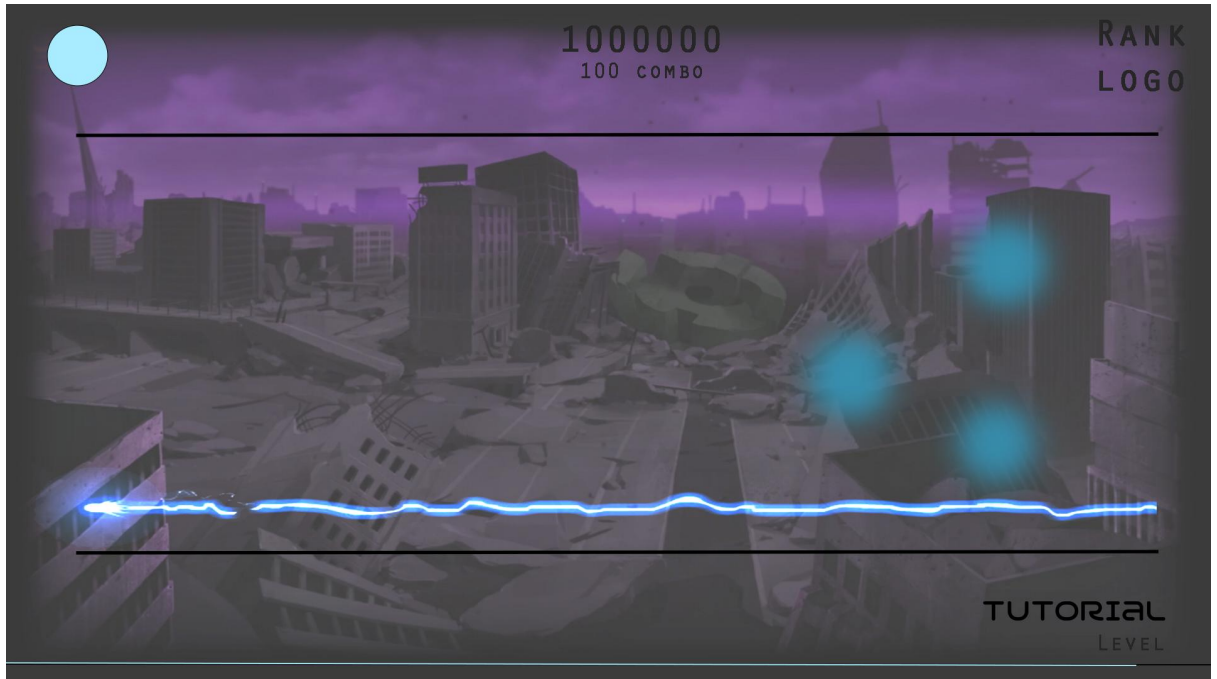
- Review login and registration system



- Player account UI



● Review song stage interface requirement



● Discuss story chapter names

Order	Chapter Title Option	Chapter Description
Prologue	<ul style="list-style-type: none"> ● Eschatos (last, end of world) ● Atropos (end of fate) ● Pandora's Box ● chaos ● Tartarus (hell) 	Briefly introduce the background of the game, including only one tutorial song stage in this chapter.
1	<ul style="list-style-type: none"> ● chaos ● Janus (beginning) ● Prometheus (先知) 	Tell the story of why the characters are awakened and how it starts
2	<ul style="list-style-type: none"> ● Moirae (fate) ● Clotho (beginning, start of fate) ● Gaia (mother earth) 	Beginning of the story - human activities in that place
3	<ul style="list-style-type: none"> ● Lachesis (fate, now) 	The middle part of the story human culture(music)
4	<ul style="list-style-type: none"> ● Eos (Dawn) ● Elpis (Hope) 	There is a little hope during the journey (turning point) found out that music can purify the world

5	<ul style="list-style-type: none"> • Euthenia (繁荣) • Elysian (淨土) • Clotho (beginning of fate) 	Follow the light and start saving the world. They found a little piece of fine land that is naturally recovered. The characters want to expand the land and accelerate the purifying process. (The purifying process is naturally done by the earth)
6	<ul style="list-style-type: none"> • Euthenia (繁荣) • Zagreus (rebirth) • Clotho (beginning of fate) 	When the world condition has reached living standards, there shall be human beings/living creatures. -> rebirth

• Discuss song list for different story chapters

Chapter	Song list (~5 songs per chapter)	Description (if any)
prologue	<ul style="list-style-type: none"> • tutorial x 1 	Tutorial song
1	<ul style="list-style-type: none"> • awaken • Neo Tokyo - Cyberpunk Mix (Sandman - Ignite) 	a bit quiet in the beginning, then progressively becoming louder -sci-fi, energetic
2	<ul style="list-style-type: none"> • Alice's Theme • Nier Copied City • Cytus Symphony 死神來了 	rhythm, nervous, wonder, unknown?
3	<ul style="list-style-type: none"> • Nier? • 【Ado】Usseewa • 【Ado】ポッカデラベリタ • Biotonic cytus 	dark music, 人性醜惡, hopeless, minor, heavy metal
4	<ul style="list-style-type: none"> • light and shadow • Hopes and Dreams 	peaceful, raining
5	<ul style="list-style-type: none"> • Sky 晨島 Dawn Flight 2 • Peaceful Piano Royalty Free Calm Relaxing Background Music No Copyright 	peaceful
6	<ul style="list-style-type: none"> • legends never die 	motivational, exciting

• Possible songs for background music

- Nier - Rebirth and Hope (Login)

- Marin Hoxha VS Vinsmoker - Night Drift
- Undertale series
- Sky series
- self remix preferred, have fl studio crack

Open Actions

Who	Things to be done before next meeting
Ada	- Propose a UX flow - Propose some button styles and game atmosphere for different chapters
Fiona	- Draft organised song list for different story chapter theme - Explore some songs for background music and game start screen
Jay	Try implementing player controls on an object with some animation; Try manipulating player's data from a database in Unity.
Gary	Main mission to the song stage, map generation function

Next meeting

FYP Group Meeting #7

Date of meeting: Thursday, 7 October 2021

Starting time: 8:30 pm

Location: HKUST Computer Barn C

Agenda:

1. Review player control implementation
2. Review UX flow
3. Review button style
4. Review song list
5. Discuss the character settings

7.7 Minutes of the 7th Project Meeting

FYP Group Meeting #7

Date of meeting: Thursday, 7 October 2021

Starting time: 8:30 pm

Location: HKUST Library Group Study Room LG4-09

Attendee Names: Ada Kong, Fiona Ng, Gary Lam, Jay Tam

Agenda:

1. Review player control implementation and the UX flow
2. Confirmed a button theme applying in game
3. Discuss song list
4. Discuss the character setting and implementation of song stage based on research found

Next meeting

FYP Group Meeting #7

Date of meeting: Sunday, 23 January 2022

Starting time: 2:30pm

Location: ZOOM

Agenda:

1. Review player control implementation
2. Review UX flow
3. Review button style
4. Review song list
5. Discuss the character settings

7.8 Minutes of the 8th Project Meeting

FYP Group Meeting #8

Date of meeting: Sunday, 23 January 2022

Starting time: 2:30 pm

Location: Discord

Attendee Names: Ada Kong, Fiona Ng, Gary Lam, Jay Tam

Agenda:

Division of Labour

Who	Things to be done before next meeting
Ada	<ul style="list-style-type: none"> - Design of character (Object, Script, Player control) - Propose Song Stage UI
Fiona	<ul style="list-style-type: none"> - Upload generated song charts to drives - Reach out method to implement the song charts to Unity(with jay) and decide together to use what playing mode for the song stage
Jay	<ul style="list-style-type: none"> - Work with KK to decide what playing mode to use for song stages - Implementation of the song stage (2 screens: 1. before playing the song & 2. during we playing the song)
Gary	<ul style="list-style-type: none"> - Research on mapping system in 3D world and see how to implement in our game - work with Ada to label the building coordinates for mapping system - Implementation of event triggers (3D world -> 2D song stage, click NPC -> take mission, etc.)

Next meeting

FYP Group Meeting #9

Date of meeting: Sunday, 23 February 2022

Starting time: 2:30 pm

Location: Discord

Agenda:

1. Review the work done base on division of labour

7.9 Minutes of the 9th Project Meeting

FYP Group Meeting #9

Date of meeting: Saturday, 5 Feb 2022

Starting time: 9:00 pm

Location: Discord

Attendee Names: Ada Kong, Fiona Ng, Gary Lam, Jay Tam

Agenda: Discuss progress report content and work on that together

Division of Labour

Who	Things to be done before next meeting
Ada	<ol style="list-style-type: none"> 1. 3D character model, then integrate into sample world 2. Build another 3D scene 3. To learn how to connect/switch between two 3D scenes (Need character to trigger) 4. How to persist the character? Make sure a character won't destroy when (3) happens
Fiona	<ol style="list-style-type: none"> 1. Write one more song's MIDI 2. Learn how to implement our storyline to the game (maybe 2D dialog, talking to NPC), this maybe need event trigger
Jay	<ol style="list-style-type: none"> 1. Finish user input system during song stage, need accuracy and playable 2. Scoring system (Perfect = 100 scores), with a note pressed effects, and save to Firebase to our own account 3. Learn how to develop 2.5D game-play, and make sure everything works
Gary	<ol style="list-style-type: none"> 1. Mini Map 2. Event trigger (cooperate with Ada)

Next meeting

FYP Group Meeting #10

Date of meeting: Saturday, 2 April 2022

Starting time: 9:00 pm

Location: Discord

Agenda:

1. Review the work done base on division of labour
2. Testing Evaluation

7.10 Minutes of the 10th Project Meeting

FYP Group Meeting #10

Date of meeting: Saturday, 2 April 2022

Starting time: 9pm

Location: Discord

Agenda

Rhythm Game Test: Tutorial + De Javu

UI suggestions:

Problem	Suggestions
Baseline too thin	Increase thickness
Background doesn't match the perspective layout	change BG
Notes starting line appeared when resolution changed to 4K UHD(3840x2160)	Fixed resolution

Bug Discovered:

Bug	Solution
The chart can be played with all 4 keys pressed together every time for preventing the wrong note is clicked	Algorithm modified
Accidental clicks may also count as a miss note	give a range of area that starts calculating the reaction time for the notes to prevent counting the accidental/ fun clicks during the interlude
The miss note counting starts before the game for the "Press any key to start the game"	Concerning a certain key for starting maybe?
Double count miss note: if a note is missed once and the player presses the note for the second time when the note is still on the screen	count once for a missing note only?

Gaming Algorithm

Original	Improvement
highest combo?	get a var storing the highest combo no when there is a bad or a miss note,

	compare the current combo and the var if current > var, store current into var.
combo showing only current on result board	playing → current result board → highest combo no.
combo score	opt 1: highest combo/ 20 * 8000 + score opt 2: if (highest combo < const) → score remain unchanged else score *= 1.15 if full combo score*1.25

Discussion: Any change on scoring values?

```
private int _scorePerNote = 2000;
private int _scorePerGoodNote = 2500;
private int _scorePerGreatNote = 3000;
private int _scorePerPerfectNote = 3500;
```

Assets required:

character 2D → ADA decides the 3D first

dialogue box → PNG



Script draft

Chapter Eos (Dawn)

CHARACTER: Alban, BOT, Letty(player), Sonny

Scene 1:

Song unlock: Light and Shadow, Sleeping Rain Song, Deja vu (unlock after finishing the two in front)

Letty	That will be great, I think we can search for it within a short time without starving that much. At least we get some fresh air and veggies... and no pandemic. :-)
Alban	Alright! MOVE MOVE MOVE! I can't wait to resume normal life!

Scene 3 (after finish all songs)

Letty	One.. Two... OMG, I still can't believe that we are that lucky to find them within such a short time.
Alban	YES PLEAAAASE! BOT! It's your showtime now!
BOT	-- DISK DETECTED --
BOT	--System ZEGREUS activating --
Sonny	I'm kinda nervous...
Alban	Pleaaase... I am PRAYING... PLEASE HAVE IT WORKED
BOT	--System ZEGREUS activated --

TBC

8 Appendix C - Project Planning

8.1 Distribution of Work

Task	Shuk Kwan	Kin Ming	Fiona	Wing Kit
Start the literature survey	○	○	●	○
Analyse the gameplay types	○	○	○	●
Design the game story	●	○	○	○
Design the game logic	○	○	○	●
Design the database system	○	●	○	○
Design the game objects	○	●	○	○
Design the welcoming screen UI	○	○	●	○
Design the mission UI	○	●	○	○
Design the song stage UI	●	○	○	○
Design the 3D world	●	○	○	○
Develop the beat maps	○	○	●	○
Develop the scoring system	○	○	○	●
Develop the database	○	●	○	○
Develop the player control	○	○	○	●
Develop the graphics	●	○	○	○
Develop the visual effects	○	○	●	○
Develop the 3D world	●	○	○	○
Develop the song stage	○	○	○	●
Develop the animation	○	○	●	○
Test the user interface	○	●	○	○
Test the player control	○	○	○	●
Test the 3D world integration	●	○	○	○
Test the database	○	○	○	●
Test the graphics	○	○	●	○
Test the visual effects	●	○	○	○
Test the animation	○	○	●	○
Test the song stage	○	○	○	●
Test the gameplay accuracy	○	●	○	○
Write the reports	●	○	○	○
Prepare for the presentation	●	○	○	○
Design the project poster	○	○	●	○

● Leader ○ Assistant

8.2 GANTT Chart

Task	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr
Do the literature survey	█	█								
Analyse the gameplay types	█	█								
Design the game story	█	█								
Design the game logic			█							
Design the database system			█	█						
Design the game objects			█	█						
Design the welcome screen UI			█	█						
Design the mission UI			█	█						
Design the song stage UI			█	█						
Design the 3D world			█	█	█					
Develop the beat maps			█	█	█					
Develop scoring system				█	█	█				
Develop the database				█	█	█				
Develop the player control				█	█	█				
Develop the graphics				█	█	█	█			
Develop the visual effects					█	█	█			
Develop the 3D world					█	█	█	█	█	
Develop the song stage						█	█	█	█	
Develop the animation							█	█	█	█
Test the user interface							█	█	█	█
Test the player control							█	█	█	█
Test the 3D world integration							█	█	█	█
Test the database							█	█	█	█
Test the graphics							█	█	█	█
Test the visual effects							█	█	█	█
Test the animation							█	█	█	█
Test the song stage							█	█	█	█
Test the gameplay accuracy							█	█	█	█
Write the proposal		█	█							
Write the monthly reports				█	█	█	█			
Write the progress report							█	█		
Write the final report									█	█
Prepare for the presentation										█
Design the project poster										█

9 Appendix D - Required Hardware and Software

9.1 Hardware

Development PC:	PC with MS Windows 10 or later
CPU:	Intel® Core™ i5 or better
RAM:	8 GB or more
GPU:	NVIDIA GeForce GTX 1060 or above
Minimum Display Resolution:	1920 * 1080 with 16-bit colour
Storage (Hard Drives):	20GB or above
Game Console:	Mouse and Keyboard

9.2 Software

Firestore	For building database
C#, C++, Java, Python	Programming languages
Unity	Game engine
Blender	For creating 3D objects and animation
Adobe After Effects	For creating animation and visual effects
Figma	For designing UI Prototype
Linux MultiMedia Studio	For MIDI file and beat mapping
Audacity	For producing in-game audio