SECV 3123 REAL-TIME COMPUTER GRAPHICS

SECTION 02

FINAL PROJECT REPORT

LEAP MOTION AND HOLOGRAM

DEVELOPERS:

GAN HENG LAI	A21EC0176
LOW CHEW SIM	A21EC3016
YEO CHUN TECK	A21EC0148

LECTURER:

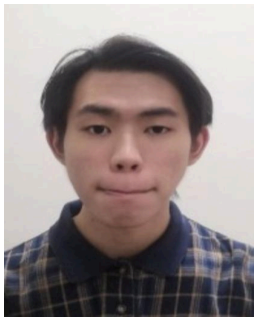
DR AJUNE WANIS BINTI ISMAIL

UNIVERSITI TEKNOLOGI MALAYSIA

TABLE OF CONTENT

1.0 OVERVIEW

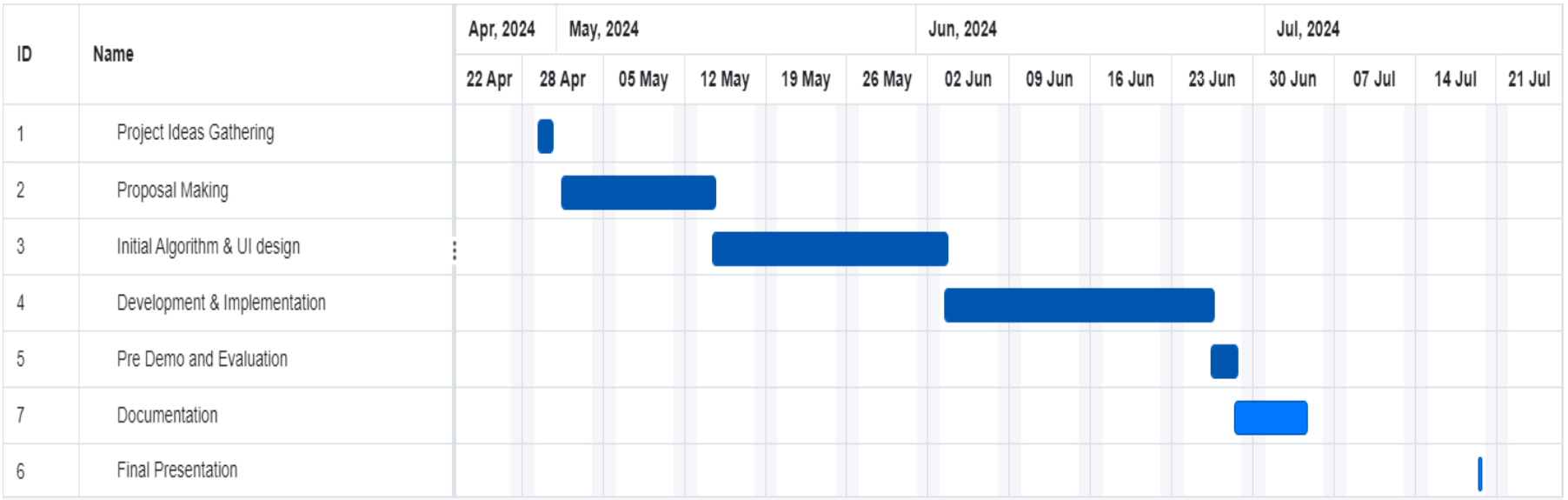
1.1 TEAM MEMBER AND TASK DISTRIBUTION

Team Members	Role and Task Distribution
 Gan Heng Lai (A21EC0176)	• Project Manager • Developer • Software Tester • Documentation
 Low Chew Sim (A21EC3016)	• Developer • Software Tester • Documentation
 Teo Chun Teck (A21EC0148)	• Developer • Software Tester • Documentation

1.2 MEETING SCHEDULE

No.	Time/Date	Location	Content
1	2:00 p.m. Tuesday, 30/4/2024	N28 Student Lounge	Discussed the project proposal: ● Poll and Choose one game from the game ideas discussed ● Task Distribution
2	2:00 p.m. Tuesday, 21/5/2024	VicubeLab	Set up and Test Leap Motion Environment in Unity
3	3:00 p.m. Friday, 31/5/2024	Google Meet	Documented the project design phase and recorded a presentation video for the design phase
4	2:00 p.m. Sunday, 2/6/2024	N28 Student Lounge	Corrected the report of the design phase: ● Addition of Hologram Setup in UI Design ● Addition of the structure of our game
5	1:00 p.m. Wednesday, 12/6/2024	VicubeLab	Developed the program: ● Development of MainMenu and Gameplay Features ● Development of Score Summary Features
6	1:00 p.m. Thursday, 20/6/2024	VicubeLab	Developed the program: ● Improvement of MainMenu Features ● Development of Gameplay Features
7	10:00 a.m. Friday, 21/6/2024	MA1, KTDI	Developed the program: ● Development of Hologram Features
8	1:00 p.m. Tuesday, 25/6/2024	VicubeLab	Completing the game flow: ● Integrating all the features to create a complete game ● Improvement in the features
9	1:00 p.m. Wednesday, 26/6/2024	VicubeLab	- Complete final improvement on the features. - Pre Demo the Game Flow. - Carry out User Testing and Evaluation
10	5 p.m. Wednesday, 3/7/2024	Google Meet	- Completed the final report - Record presentation video

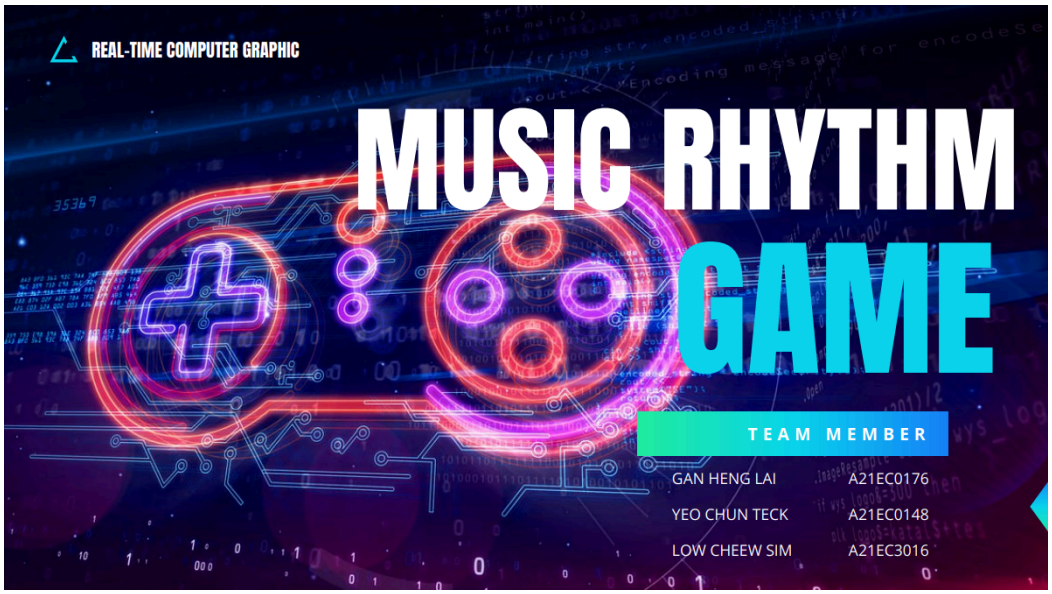
1.3 GANTT CHART



2.0 ABSTRACT

This project is to propose a system that implements the technology of "Leap Motion" and "Hologram". Music Rhythm Game is a game where the player would enjoy music with the rhythm. The concept of the game is to utilize a leap motion controller to track the player's hand movements in real-time accurately. This would enable the player to interact with the game by simply moving their hands in the air, making the game intuitive. Moreover, the game would use hologram technology to project the game environment and music notes in front of the player. This creates a visually stunning and futuristic setting where the players can see the notes coming toward them in 3D space, enhancing the immersive experience. The project has gone through several phases which are Proposal, Design, Development and Implementation, and Testing and Evaluation. During the Proposal phase, this project is proposed by defining the two technologies that are implemented, project description and requirements, planning, and expected output. The study of the two technologies was done during the Design phase and then the technologies were implemented into the system during the Development and Implementation phase. Last but not least, the system is tested in different hardware and evaluated during the Testing and Evaluation phase.

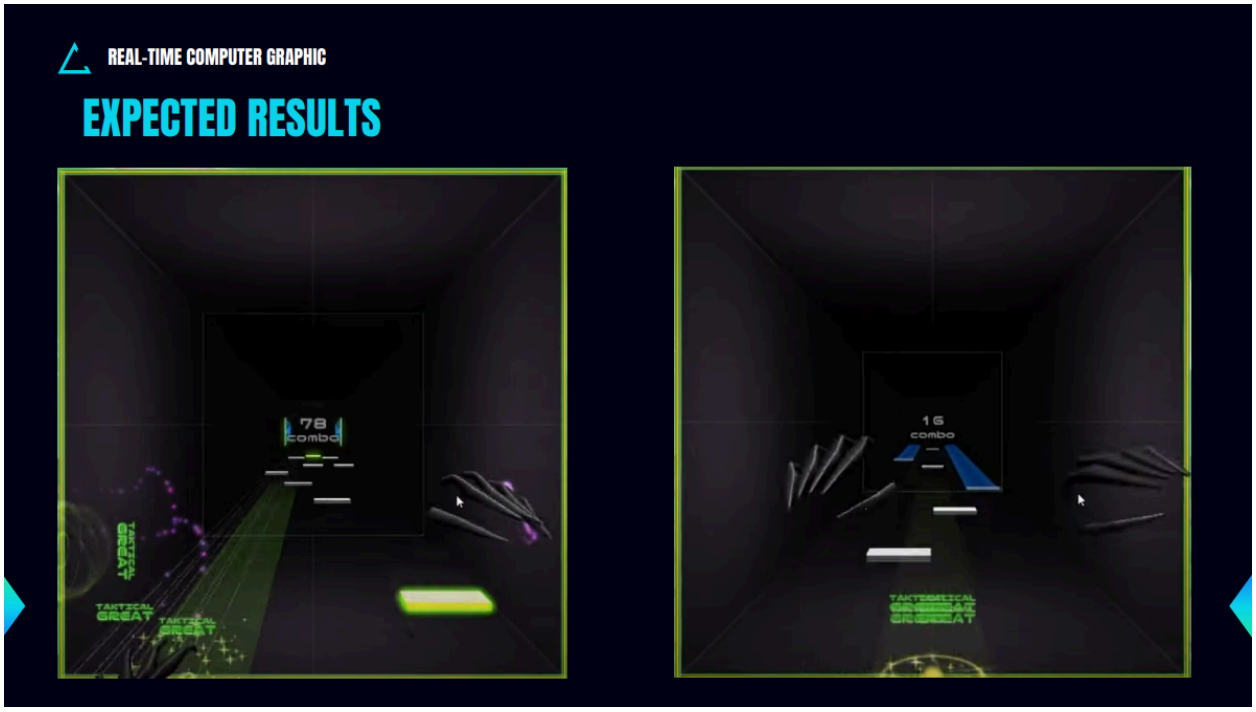
3.0 PROPOSAL



3.1 INTRODUCTION

3.2 STORYTELLING AND IDEAS

3.3 EXPECTED RESULT



4.0 INITIAL ALGORITHM AND DESIGN

4.1 INTRODUCTION

This section will consist of a detailed description or ideas of our project. It contains the initial algorithm in our project, the structure of the game, the hologram setup as well as the UI design on every page of our game.

4.2 INITIAL ALGORITHM

Figure 4.1 shows the initial algorithm of our project which was a Music Rhythm Game with Leap Motion and Holograms.

```
BEGIN Game
    INITIALIZE the Leap Motion device.
    DISPLAY the Main Menu.

    WHILE the game is running:
        IF the player is in the Main Menu:
            DISPLAY the Song Selection Menu.

            WHILE the player is in the Song Selection Menu:
                CAPTURE the hand gesture from the Leap Motion device.
            IF the player swipes left:
                MOVE to the previous song in the list.
            ELSE IF the player swipes right:
                MOVE to the next song in the list.
            ELSE IF the player swipes up:
                SELECT the highlighted song and START song playback.

        IF a song is selected and playing:
            DISPLAY the Gameplay Screen.

            WHILE the song is playing:
                RETRIEVE the upcoming notes.

                IF the note reaches the score mark line:
                    IF the note is single:
                        IF the player hit the note:
                            IF the timing of the gesture is accurate:
                                DISPLAY "Perfect" feedback.
                                INCREASE the player's score.
                            ELSE: DISPLAY "Miss" feedback.

                    IF the notes is continuous:
                        IF the player maintains the hold gesture:
                            CONTINUE to increase the player's score.
                        ELSE: DISPLAY "Miss" feedback.
                              STOP increasing the player's score.

                ELSE IF the note passes and no action:
                    DISPLAY "Miss" feedback.

                IF hand click upright button OR space bar clicked
                    POP menu
                    IF Resume button clicked
                        CONTINUE gameplay
                    ELSE IF Quit button clicked
                        RETURN Main Menu

        ONCE the song ends:
            DISPLAY total score and missed notes.
            WAIT for player input to proceed.
            RESET the game to the Main Menu state.

    END WHILE
END Game
```

Figure 4.1 Initial Algorithm

4.2.1 MAIN MENU

The main menu will be presented when the music game is launched. The player interacts with the menu using hand gestures that are detected by the Leap Motion controller. The player can swipe left to go to the previous song or swipe to the right to move to the next song. Lastly, the player can select the song by grabbing their hand and it will start the initiation of the music game.

4.2.2 GAMEPLAY

After that, the music game will start when the player has selected and confirmed the song that they prefer by grabbing their hand. The rhythmic notes begin to flow towards a score mark line on the screen. The player must interact with these notes using precise hand movements. Moreover, the player should hit the note as the note approaches the score mark line. If the timing is precise, the game acknowledges a “Perfect” hit and increases score points; if the timing is too early, the game acknowledges a “Fail” hit and no points are given; if the timing is off a bit, the game acknowledges a “Good” hit and increase the scores points less; if the player misses the timing, the game records a “Miss” and no points are given. During the gameplay, there is an additional UI element that the player can interact with. By moving the hand to interact with this element, the player can pause the game. The clicking triggers the appearance of a pause menu to give the player the option to either continue playing or go back to the main menu.

4.2.3 END OF SONG SUMMARY

This summary provides an overview of the player's performance during the song. At the end of the song, the game displays a summary screen showing the player's total score and the number of missed notes. Then, the player has the option to await input in order to either restart the game or navigate back to the main menu and choose another song.

4.3 MUSIC RHYTHM GAME DESIGN

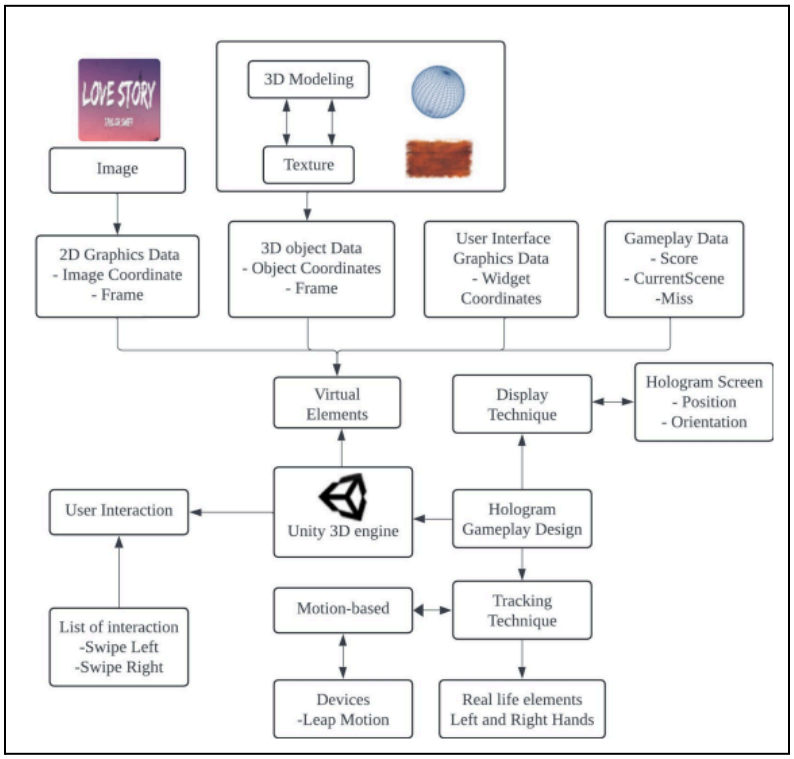


Figure 4.2 Structure of Music Rhythm Game

Based on Figure 4.2 above, it shows the structure of our game project which is a music rhythm game. The game will develop using the Unity engine and Leap Motion will be the device to track the hand motion of the player in order to interact with the game. There are some elements like menu buttons, hand gestures, and music notes that require the player to interact using their hand. The Z-hologram box will be used as the environment to display game output in the hologram

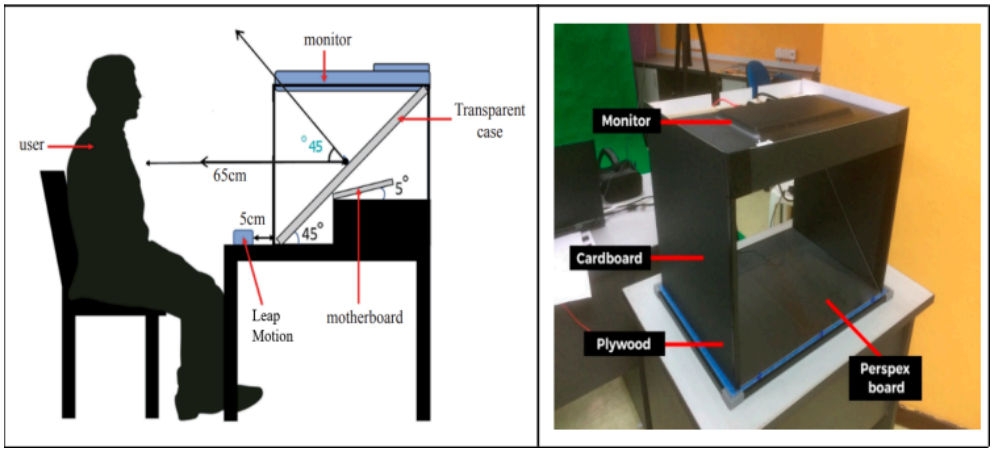
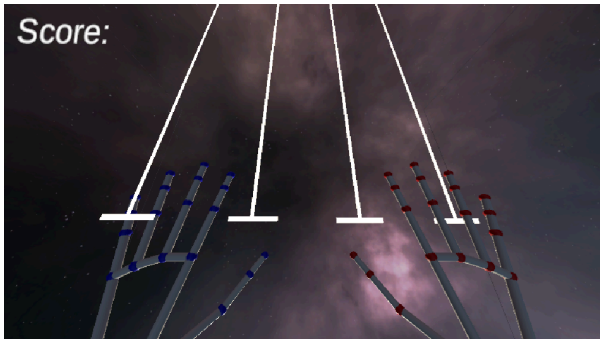


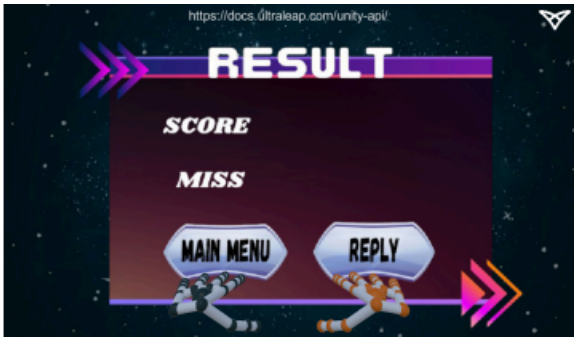
Figure 4.3 Hologram Setup

Based on Figure 4.3, it shows the hologram setup in our project. This is the design of the gameplay environment. The player is required to sit on the chair and interact with the Leap Motion device placed on the table. The scene of the game will be presented in a Z-hologram box. The output of the game will be displayed on the monitor above in flipped display orientation first and then reflected to the perspex board in order to show the hologram output.

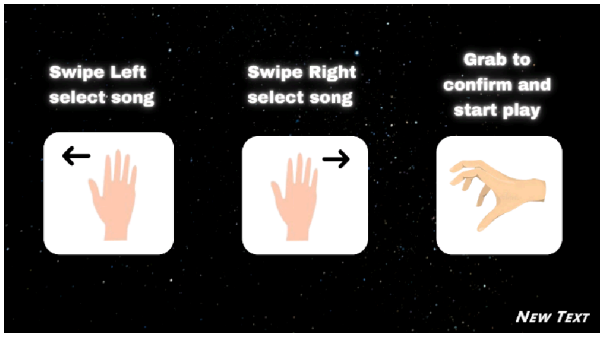
UI Design of Main Menu



UI Design of Game Play



UI Design of Scoring Board



UI Design of Guidelines

5.0 DEVELOPMENT AND IMPLEMENTATION

5.1 INTRODUCTION

This music game is a game that requires us to connect with the Leap Motion and Holograms devices. Hence, the interaction that will be integrated with this game is using hand gestures such as swiping left and right to select multiple songs from the main menu, hitting the rhythm notes when the player starts to play the game and it also will show the points that the player scored when they successfully hit the notes.

5.2 TECHNIQUES AND IMPLEMENTATION

5.2.1 MAIN MENU

In the Main Menu, we have developed it to swipe left and right by using the hand gesture that is detected with the Leap Motion controller. We have developed the hand pose which is ‘Open Palm Hand’ to swipe left and swipe right to select multiple songs. Moreover, it also contains a user menu in the selection that enables the player to select and read the instructions of the game. Furthermore, we have implemented a 'Grab Hand' pose to allow the player to confirm and start playing the game. Initially, we developed a swipe-up gesture for this interaction. However, we are unable to effectively demonstrate it in the game due to sensitivity issues with the swipe-up hand gesture. Thus, we decided to use the “Grab Hand” gesture instead of the swipe-up gesture.

```
foreach (Hand hand in frame.Hands)
{
    // Get swipe direction and velocity
    Vector3 swipeDirection = hand.PalmVelocity.normalized;
    float swipeVelocity = hand.PalmVelocity.magnitude;

    if (Mathf.Abs(swipeDirection.x) > Mathf.Abs(swipeDirection.y)) // Horizontal swipe
    {
        // Adjust the scrolling speed based on hand velocity
        float scrollSpeed = swipeVelocity * scrollSpeedFactor;

        if (hand.IsLeft) // Left hand
        {
            if (swipeDirection.x < 0) // Swipe Right
            {
                scroll_pos += scrollSpeed; // Move left
            }
            else // Swipe Left
            {
                scroll_pos -= scrollSpeed; // Move right
            }
        }
        else // Right hand
        {
            if (swipeDirection.x > 0) // Swipe Left
            {
                scroll_pos -= scrollSpeed; // Move left
            }
            else // Swipe Right
            {
                scroll_pos += scrollSpeed; // Move right
            }
        }
    }

    if (hand.GrabStrength > 0.9f) // Grab pose detected
    {
        grabDetected = true;
    }
}
```

Figure 5.1 Update() method from SwipeMenu.cs

The above snippet shows the script that we wrote in our project to swipe left and right as well as grab to choose and start to play the game. In addition, the below Figure 5.1 shows the Main Menu that we successfully implemented.

Figure 5.2 Main Menu (Songs Selection)

5.2.2 GAMEPLAY

In Gameplay, there are four lanes that comprise their own moving notes. In every lane, there are a Miss Effect game object, a PerfectEffect game object, a FailEffect game object, and a GoodEffect game object at the score mark line. Every note in a lane has the same Miss Line Mark. The Score Text is the UI game object that displays the current score of the player. We have developed the rhythmic notes mechanism, which defines the movement of the notes from their own original positions to their own miss line mark as their destination during the gameplay, and the game pauses their interaction with other game objects.

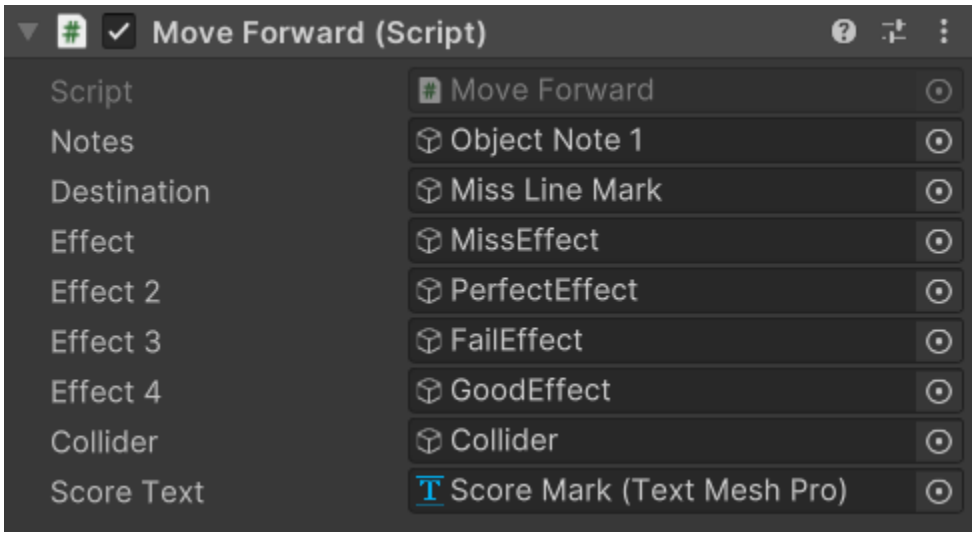


Figure 5.3 Variable Initialization from Move Forward.cs Component of A Note in The First Lane

```
if (!isPaused && Notes.activeSelf)
{
    Notes.transform.position = Vector3.MoveTowards(Notes.transform.position, Destination.transform.position, speed * Time.deltaTime);

    float distanceToDestination = Vector3.Distance(Notes.transform.position, Destination.transform.position);

    if (distanceToDestination < 0.001f)
    {
        Notes.SetActive(false);
        Effect.SetActive(true);
        miss += 1;
        Invoke("ActiveObject", 0.3f);
    }

    if (distanceToDestination > 2f)
    {
        notesMeshRenderer.enabled = false;
    }
    else
    {
        notesMeshRenderer.enabled = true;
    }
}
```

Figure 5.4 Update() method from Move Forward.cs

The above snippet shows the script that we wrote in our project to define the movement of the notes. Every note will have this object of the Moveforward class as one of its c# script components.

According to this script in normal gameplay, the notes will move from their predefined positions to the score mark line with predefined gameplay speed in the interval where they are active. During the game pause, the statement under the if condition will not be executed. Hence, the notes will stay in their previous position until the gameplay continues from its pause.

The notes will interact with the miss line mark based on the distance between them and with the hand during the interval trigger. The above script shows one interaction between the notes and the miss line mark. When the notes approach closely to the score mark line, a “Miss” effect will pop out for a while and then disappear. The number of “Miss” counts will be accumulated.

```
void OnTriggerEnter(Collider other)
{
    float distanceToDestination = Vector3.Distance(Notes.transform.position, Destination.transform.position);

    Notes.SetActive(false);

    if (distanceToDestination > 0.35f)
    {
        Effect3.SetActive(true);
        miss += 1;
        Invoke("ActiveObject3", 0.3f);
    }
    else if (distanceToDestination > 0.25f && distanceToDestination <= 0.35f)
    {
        Effect4.SetActive(true);
        AddFiveScore();
        Invoke("ActiveObject4", 0.3f);
    }
    else if (distanceToDestination > 0.11f && distanceToDestination <= 0.25f)
    {
        Effect2.SetActive(true);
        AddTenScore();
        Invoke("ActiveObject2", 0.3f);
    }
    else
    {
        Effect4.SetActive(true);
        AddFiveScore();
        Invoke("ActiveObject4", 0.3f);
    }
}
```

Figure 5.5 OnTriggerEnter() method from Move Forward.cs

The above snippet shows the script that we wrote in our project to define the interaction of the notes when it is on the trigger with the hand before the notes arrive at the position of the score mark line, which is another game object. When the notes are triggered by hand, the AddFiveScore() method or AddTenScore() method will be executed based on the distance between the notes and the score mark line. These methods are working like the interaction we mentioned in the 4.2.2 section. Figure 5.6, Figure 5.7, and Figure 5.8 show the “Perfect” Hit, “Good” Hit, “Fail” Hit, and “Miss Hit” effect activations respectively.

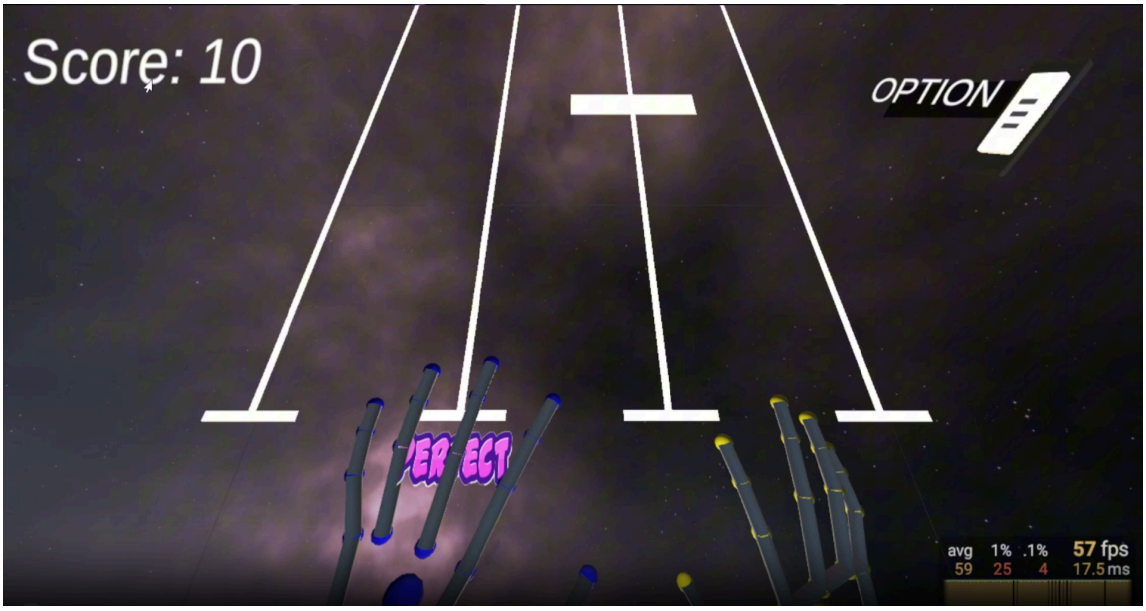


Figure 5.6 Perfect Hit Activation (Gameplay)

Figure 5.7 Good Hit Activation (Gameplay)

Figure 5.8 Fail Hit and Miss Hit Activation (Gameplay)

We have also developed the game pause feature which is embedded in the option button of the gameplay screen.

```
public void ToggleTrigger()
{
    if (pauseController != null)
    {
        pauseController.TogglePause();
        gamePause.TogglePause();

        foreach (MoveForward moveForward in moveForwardInstances)
        {
            moveForward.TogglePause();
        }
    }
}

0 references
void OnTriggerEnter(Collider other)
{
    ToggleTrigger();
    GetComponent<BoxCollider>().enabled = false;
}
```

Figure 5.9 ToggleTrigger() and OnTriggerEnter() methods from HandHit.cs

```
public void TogglePause()
{
    isPaused = !isPaused;

    if (isPaused)
    {
        audioSource.Pause();
    }
    else
    {
        audioSource.UnPause();
    }
}
```

Figure 5.10 TogglePause() from PauseController.cs

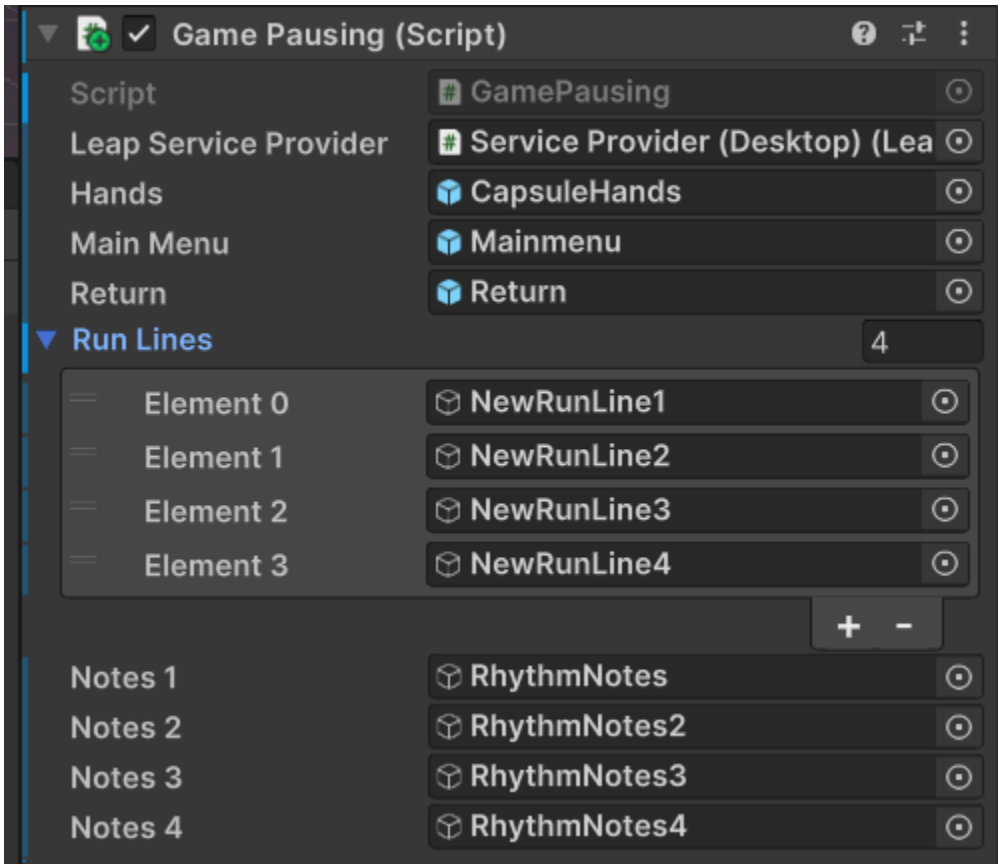


Figure 5.11 Variable Initialization of GamePausing.cs

Based on Figure 5.11, in the GamePausing Component of the music game object, Run Lines are the lanes game object in the gameplay, while Notes1, Notes2, Notes3, and Notes4 are the group of notes in every lane.

```
public void TogglePause()
{
    isPaused = !isPaused;

    if (isPaused)
    {
        foreach (GameObject Runline in RunLines)
        {
            MainMenu.SetActive(true);
            Return.SetActive(true);
        }

        SetNotesBoxCollidersActive(false);
    }
    else
    {
        foreach (GameObject Runline in RunLines)
        {
            MainMenu.SetActive(false);
            Return.SetActive(false);
        }

        SetNotesBoxCollidersActive(true);
    }
}

2 references
private void SetNotesBoxCollidersActive(bool isActive)
{
    GameObject[] notes = { Notes1, Notes2, Notes3, Notes4 };

    foreach (GameObject note in notes)
    {
        if (note != null)
        {
            BoxCollider[] colliders = note.GetComponentsInChildren<BoxCollider>();
            foreach (BoxCollider collider in colliders)
            {
                collider.enabled = isActive;
            }
        }
    }
}
```

Figure 5.12 TogglePause() and SetNotesBoxColliderActive() from GamePausing.cs

The above snippet shows the scripts that we wrote in our project to define the interaction of the options button when it is on the trigger by hand. The “Option” Button game object has an object of HandHit class as one of its components.

According to Figure 5.13, when the “Option” button is clicked by the hands, it will call ToggleTrigger() methods from the PauseController object and GamePausing object which are of the components of other game objects. The ToggleTrigger() method from the PauseController object pauses the music. The ToggleTrigger() method from the GamePausing object activates the MainMenu button and Return button and deactivates the collider of every note so that the hand will not trigger the notes during the game pause. In the meanwhile, the collider of the “Option” button will also be disabled so that the player will not be able to interact with the “Option” button after he calls out the “Option Menu”. Figure 5.13 shows the OptionMenu during game pause.

Figure 5.13 OptionMenu During Game Pause (Gameplay)

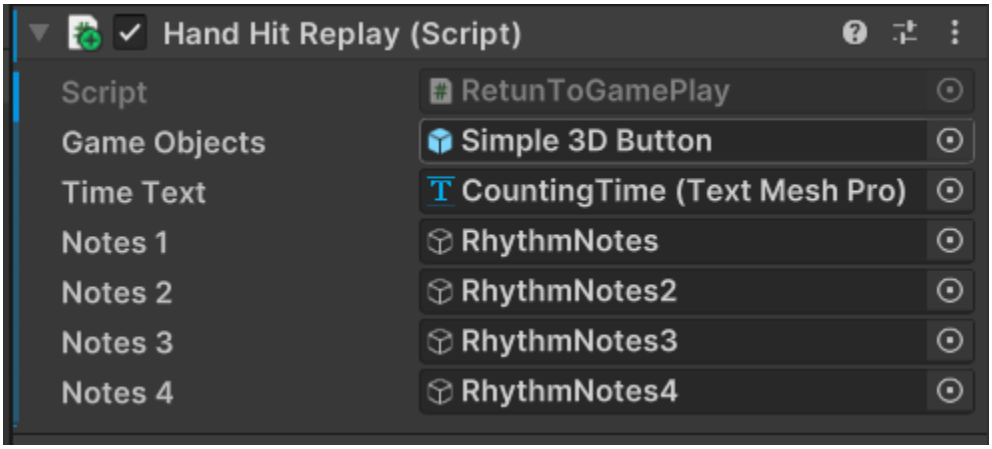


Figure 5.14 Variable Initialization of ReturnToGame.cs

Based on Figure 5.14, in the HandHitReplay component of the Return button, the GameObjects is the Option Button game object; the Time Text is the UI game object that counts down the time when the player continues the game from the pause; Notes1, Notes2, Notes3, and Notes4 are a group of notes in different lanes.

```
public void ToggleTrigger()
{
    if (pauseController != null && !isCountingDown)
    {
        isCountingDown = true;
        gamePause.TogglePause();
        SetNotesBoxCollidersActive(false);
        InvokeRepeating("CountDown", 1f, 1f);
    }
}

0 references
private void CountDown()
{
    timeText.text = remainingTime.ToString();
    remainingTime--;

    if (remainingTime < 0)
    {
        CancelInvoke("CountDown");
        remainingTime = 3f;
        timeText.text = "";
        isCountingDown = false;
        pauseController.TogglePause();
        MoveNotes();
        SetNotesBoxCollidersActive(true);
    }
}

1 reference
private void MoveNotes()
{
    foreach (MoveForward moveForward in moveForwardInstances)
    {
        moveForward.TogglePause();
    }
}

0 references
void OnTriggerEnter(Collider other)    Remove this unused meth
{

    ToggleTrigger();
    gameObjects.GetComponent<BoxCollider>().enabled = true;
}
```

Figure 5.15 ToggleTrigger(), Countdown(), MoveNotes() and OnTriggerEnter() from ReturnToGameplay.cs

```
private void SetNotesBoxCollidersActive(bool isActive)
{
    GameObject[] notes = { Notes1, Notes2, Notes3, Notes4 };

    foreach (GameObject note in notes)
    {
        if (note != null)
        {
            BoxCollider[] colliders = note.GetComponentsInChildren<BoxCollider>();
            foreach (BoxCollider collider in colliders)
            {
                collider.enabled = isActive;
            }
        }
    }
}
```

Figure 5.16 SetNotesBoxColliderActive() and MoveNotes() from ReturnToGameplay.cs

Both of the above snippets show the scripts that we wrote in our project to define the interaction of the Return button when it is on the trigger by hand. The Return Button game object has an object of ReturnToGameplay class as one of its C# components.

According to these scripts, when the "Return" Button is clicked by hand, it will call ToggleTrigger() methods from the PauseController object and GamePausing object which are the components of other game objects. Then, the main menu button and Return button will be inactive. There will be a remaining time of 3 seconds before returning back to the game. After 3 seconds, the gameplay will continue and the collider of the Option button will be enabled again so that the player can interact with the Option button after the gameplay continues. Figure 5.17 shows the countdown of 3 seconds after clicking the Return button in the OptionMenu.

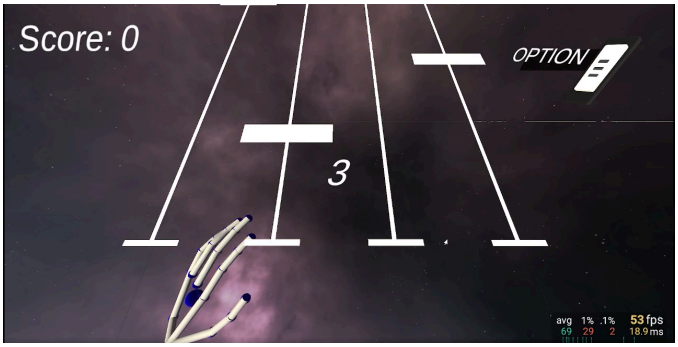


Figure 5.17 Countdown of 3 seconds After Clicking The Return Button (Gameplay)

```
void OnTriggerEnter(Collider other)
{
    SceneManager.LoadScene("SwipeMenu");
}
```

Figure 5.18 OnTriggerEnter() from Back2Menu.cs

The above snippets show the script that we wrote in our project to define the interaction of the MainMenu button when it is on the trigger with hand. The MainMenu Button game object has an object of the "Back2Menu" class as one of its components. When the hand clicks the MainMenu Button, the player will move to the MainMenu.

5.2.3 END OF SONG SUMMARY

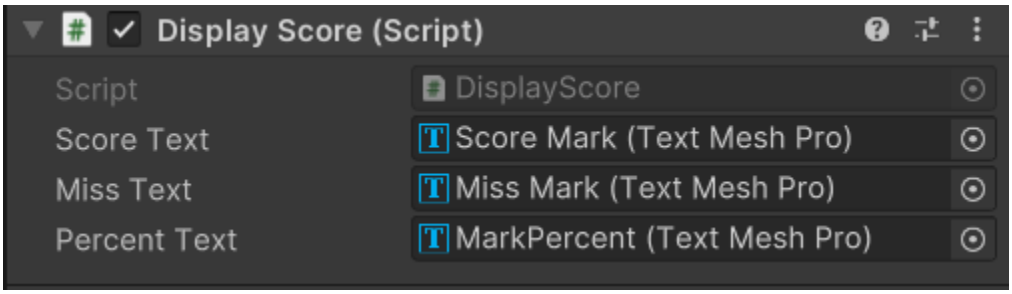


Figure 5.19 Variable Initialization of DisplayScore.cs

In the Score Summary Scene, there are three text UI game objects that display score, miss, and percent respectively.

```
void Start()
{
    int totalScore = PlayerPrefs.GetInt("TotalScore", 0);
    int totalMiss = PlayerPrefs.GetInt("TotalMiss", 0);

    string previousSceneName = PlayerPrefs.GetString("PreviousSceneName", "");

    if (previousSceneName == "GamePlay1")
    {
        totalMark = 1260;
    }
    else if (previousSceneName == "GamePlay2")
    {
        totalMark = 3060;
    }
    else if (previousSceneName == "GamePlay3")
    {
        totalMark = 2050;
    }

    percent = totalScore / totalMark * 100;

    scoreText.text = totalScore.ToString();
    missText.text = totalMiss.ToString();
    percentText.text = percent.ToString("F2") + "%";
}
```

Figure 5.20 Start() from DisplayScore.cs

```
void OnTriggerEnter(Collider other)    Remove this unused method parameter 'other'.
{
    string previousSceneName = PlayerPrefs.GetString("PreviousSceneName", "");

    if (!string.IsNullOrEmpty(previousSceneName))
    {
        SceneManager.LoadScene(previousSceneName);
    }
}
```

Figure 5.21 OnTriggerEnter() from Replay.cs

Figure 5.18, Figure 5.20, and Figure 5.21 show the scripts that we wrote in our project to define the display of the marks and interactions with buttons in the Score Summary. The scores, miss hits, and percentages from the recent gameplay will be displayed on the Score Summary UI. The player could replay the game if the Replay button is clicked or return to MainMenu if the MainMenu button is clicked.

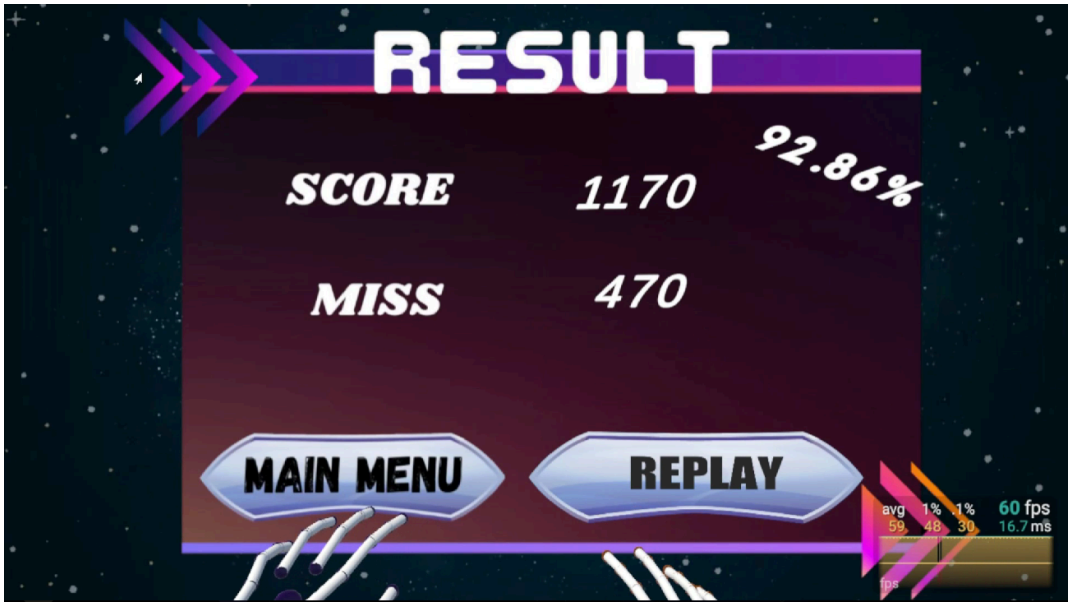


Figure 5.22 Score Summary (End of Song Summary)

5.2.4 HOLOGRAM DISPLAY

The projection of the gameplay screen onto the hologram display will be inverted. We overcame this issue by adjusting the y-axis rotation angle or z-axis rotation angle in the transform components of every game object and the rect transform components of every UI game object so that all of them could be flipped.

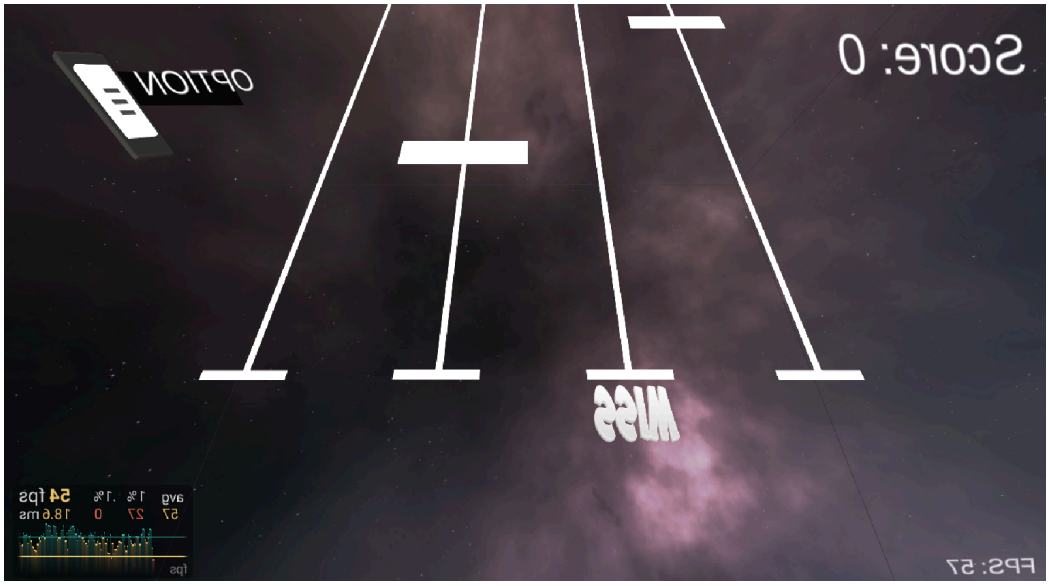


Figure 5.23 Gameplay Screen in the PC (Gameplay)

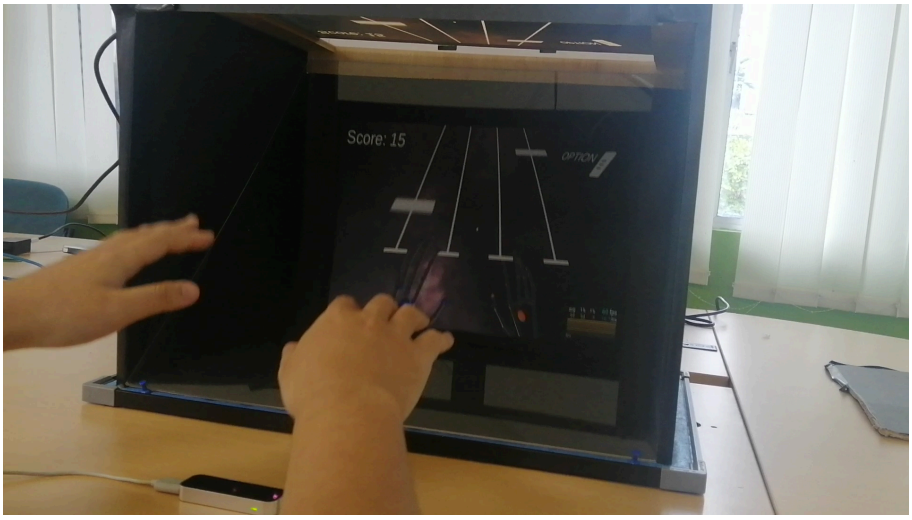


Figure 5.24 Gameplay Screen in the Hologram (Gameplay)

6.0 TESTING AND EVALUATION

6.1 REQUIREMENTS AND SPECIFICATION

In this section, we will discuss three different specifications of computers that we use to develop this project and present the results. Shown below is the table for three different specifications of computers. This testing and specification has been carried out by using three different types of hardware and software.

Table 6.1 Hardware Specification

Hardware			
	Laptop 1	Laptop 2	Laptop 3
System Model	Lenovo IdeaPad 5 Pro	HP VICTUS 16	ACER NITRO5 N20C
Central Processing Unit (CPU)	AMD Ryzen 7 5800U with Radeon Graphics	AMD Ryzen 5 7640Hs/ Radeon 760M Graphics	AMD Ryzen 5 5600H Radeon Graphics GHz
Random Access Memory (RAM)	16.00 GB	16.00 GB	8.00 GB
Graphics Processing Unity (GPU)	NVIDIA GeForce MX450	NVIDIA GeForce RTX 4050	NVIDIA GTX 1060
System Type	64-bit operating system, x64-based processor	64-bit operating system, x64-based processor	64-bit operating system, x64-based processor

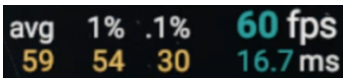
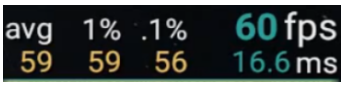
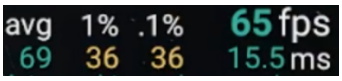
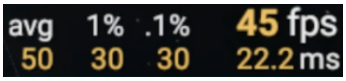
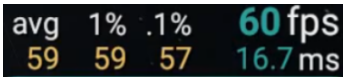
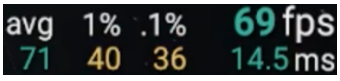
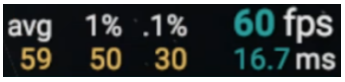
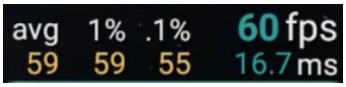
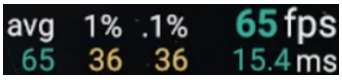
Table 6.2 Software Specification

Software			
	Laptop 1	Laptop2	Laptop 3
Operating System	Window 11	Window 11	Window 10
Unity Version	2022.3.23f1	2022.3.27f1	2022.3.5f1
Integrated Development Environment (IDE)	Microsoft Visual Studio Community 2022 17.7.6	Microsoft Visual Studio Community 2022 17.10.1	Microsoft Visual Studio Code

6.2 GAME PERFORMANCE

Table 3.1 and Figure 6.1 shows the frame-per-second performance measured from 3 different devices. The rank from highest to lowest is Laptop 3 > Laptop 2 > Laptop 1. This ranking indicates that Device 3 has the highest FPS, Device 2 has a medium FPS, and Device 1 has the lowest FPS. This may be due to the difference in the hardware level of the laptop especially the Graphics Processing Unity where Laptop 3 owns the highest level while Laptop 1 owns the lowest.

Table 6.3 Performance Comparison Between Devices

Game Song Scene	**Laptop 1**	**Laptop 2**	**Laptop 3**
1			
2			
3			

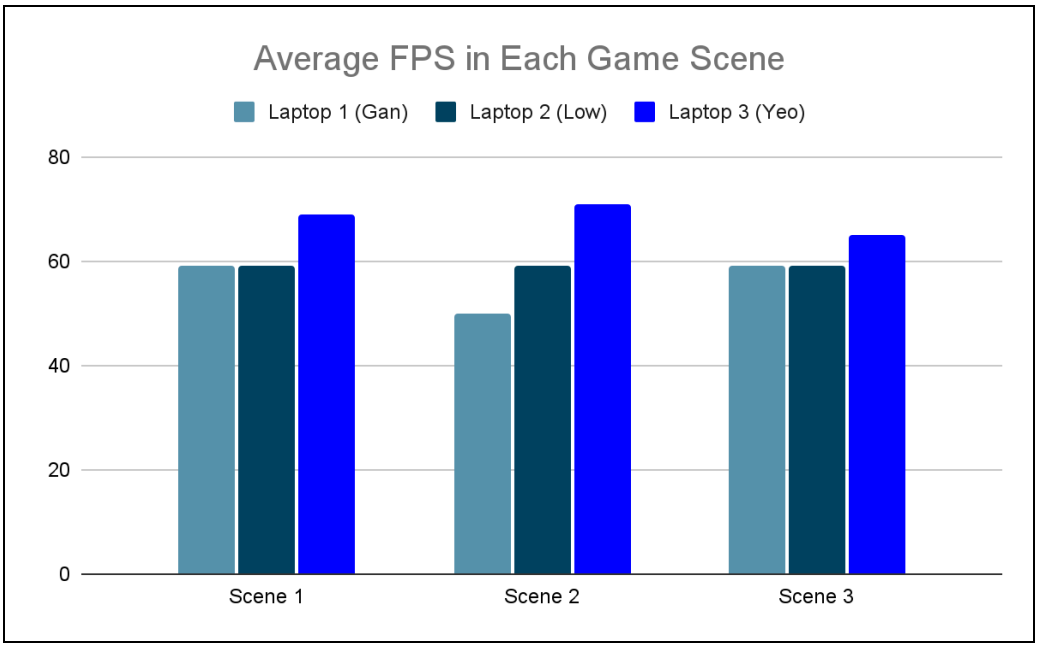


Figure 6.1 Bar Chart of Average FPS in Game Scene

6.3 USER TESTING AND RESULT

After successfully developing the music game, we have created a usability feedback form that contains 15 questions and it is divided into 3 sections which are Leap Motion, Holograms and User Experience (UX) for the users to fill up. The feedback form is created because it can validate design and development choices, ensuring that the product meets user needs and expectations.

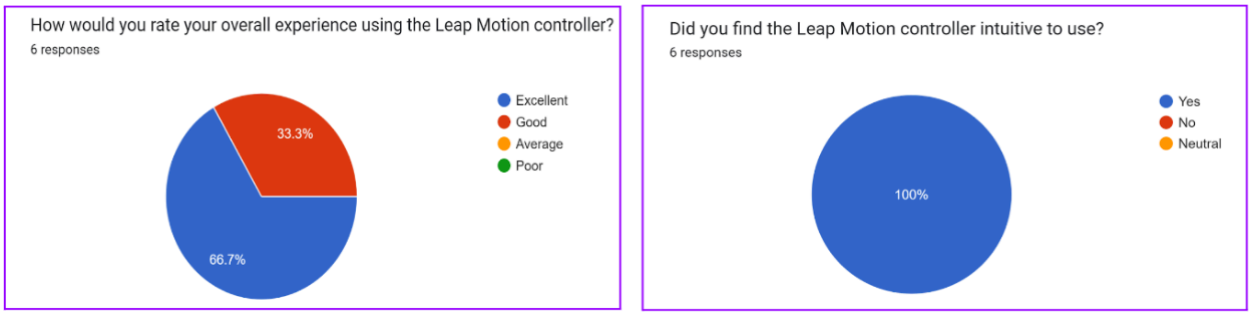


Figure 6.2 Evaluation of Leap Motion in Performance Part 1

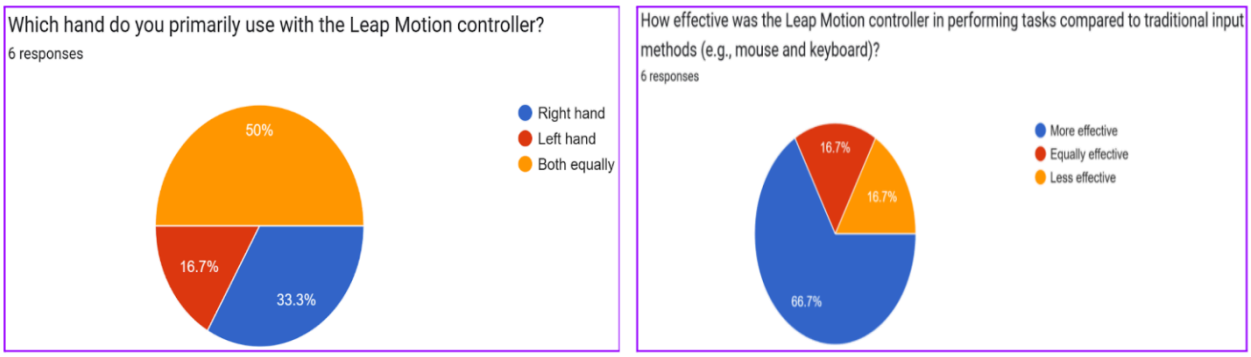


Figure 6.3 Evaluation of Leap Motion in Performance Part 2

Based on Figure 6.1, Leap Motion achieves good performance in its accuracy, responsiveness, and ease of use. The majority of the participants gave positive feedback on implementing the Leap Motion and primarily used both hands with the Leap Motion controller. However, 16.7% (1 out of 6) considered that the Leap Motion is not effective in performing tasks compared to traditional input methods. He noted occasional issues with tracking, particularly when his hands moved out of the sensor‘s range.

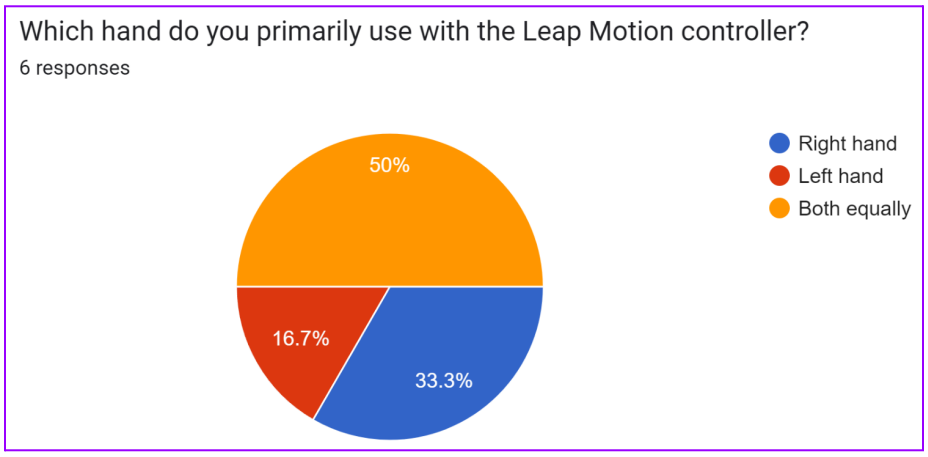


Figure 6.4 Preference in Using Leap Motion Controllers

From Figure 6.2, it appears that there is an equal preference for using either one hand or both hands with the Leap Motion controller among users surveyed. The users who prefer using one hand stated that the usage of the Leap Motion controller in interaction was like moving the mouse cursor on a PC, allowing many interactions to be done easily with one hand. Conversely, the users who prefer using both hands praised the challenging nature of the game design, finding it engaging and rewarding to use both hands to achieve higher scores.

Figure 6.5 Evaluation of Leap Motion

Figure 6.2 shows the responses in the aspect of Holograms. Based on the feedback of the users, all of the users have excellent experiences playing music games with Holograms. Moreover, the majority of the users are enjoyable and immersed as well as intuitively interacting with holograms during play. Finally, there are 4 users choosing 4 scales in the responses to "How Holograms affect your sense of timing and rhythm in the game". This is because they found the hologram to be a valuable aid in maintaining rhythm and timing.

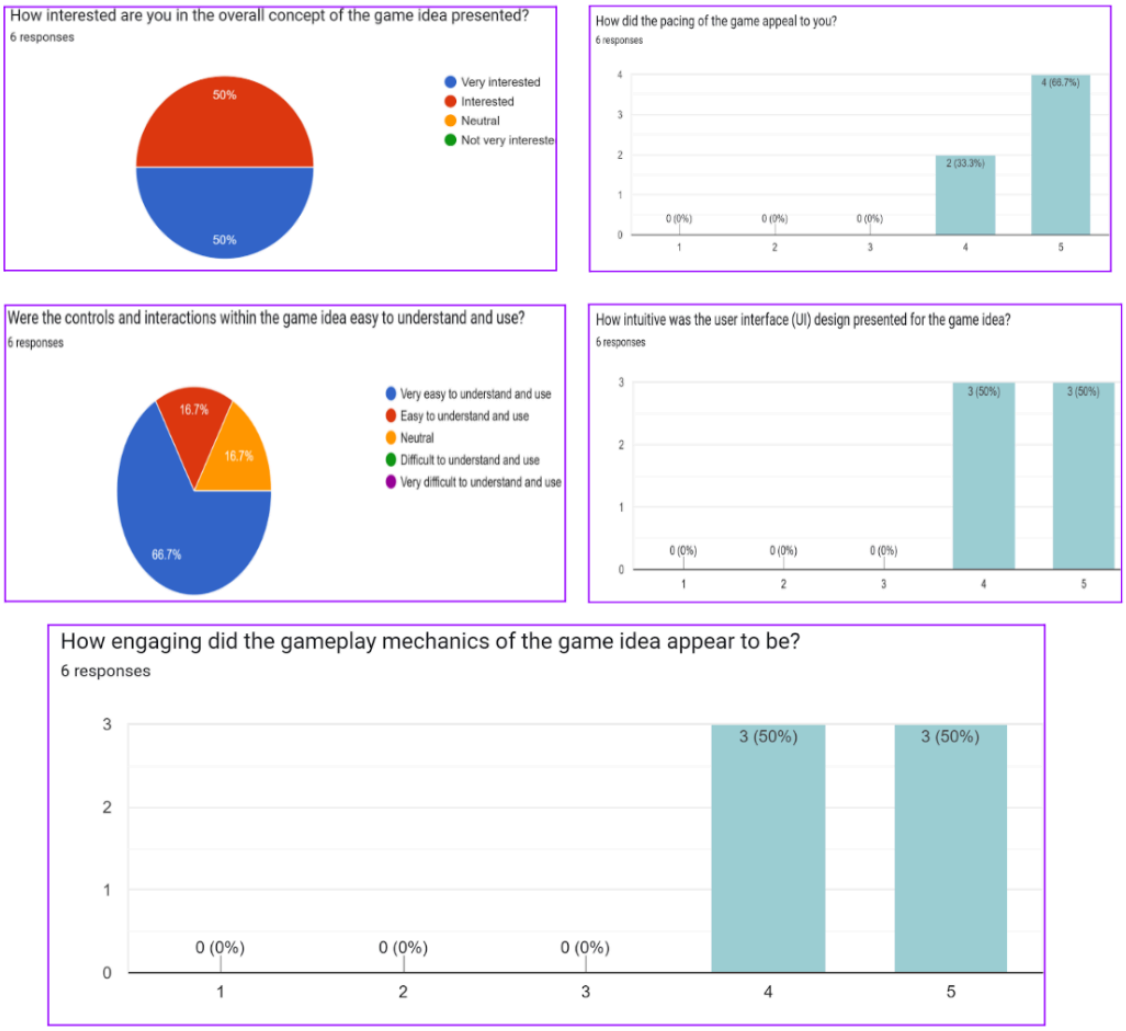


Figure 6.6 Evaluation of User Experience (UX)

Figure 6.3 shows the responses in the aspect of user experience (UX). Based on the feedback of the users, the majority of the users are interested in the game idea, and it is also very easy to understand as well as use the game while they are playing the music game. Additionally, the users have a higher intuitiveness of the user interface presented for game ideas. This is because they had no trouble understanding the purpose of each screen and menu. The gameplay mechanism of the game appears to be highly engaging, with an equal distribution of users rating it as a 4 or a 5 on a scale of engagement. This indicates that all users surveyed found the gameplay mechanism to be engaging, with half considering it very engaging (rating 4) and the other half finding it extremely engaging (rating 5). This positive feedback highlights the effectiveness of our game's core mechanics in capturing and maintaining player interest. Lastly, there are 4 user responses highly pacing of the game that appeals to them are very appealing.

7.0 CONCLUSION

To sum up, we successfully finished our job in the allotted time. An important turning point in the history of entertainment and technology is represented by this endeavor. This creative effort has shown how motion-sensing technology and immersive holographic projections may work together to produce a musical experience that is both entertaining and participatory. Leap Motion was integrated during the development phase to enable accurate hand tracking, which allowed users to engage with the music in a natural and dynamic way. Holograms were incorporated to offer an eye-catching visual element and create a lively, rhythmic atmosphere for the user. This project showed a high degree of originality and technical proficiency in addition to overcoming a variety of technical obstacles.

The accomplishment of this project not only establishes a new benchmark for what can be accomplished with the combination of motion-sensing technology and holographic projections but also shows promise for future advancements in interactive music experiences. This achievement presents great prospects for the future and opens the door to additional research and innovation in the field of interactive entertainment.

8.0 REFERENCES

simishu. (2017, November 24). *Takt-Rhythm Demo Gameplay (Digital Games Expo 2017)* [Video]. YouTube. https://www.youtube.com/watch?v=maYM_3FB3Fk

3DBuzz. (2014, September 16). *Modern GUI development in Unity 4.6 - #3: The Canvas* [Video]. YouTube. https://www.youtube.com/watch?v=at6gUB_-HII

The Unity Dude. (2023, August 31). *Infinite looping Scroll View / Scroll Rect (Unity Beginner Tutorial)* [Video]. YouTube. https://www.youtube.com/watch?v=r8Z-gPcWlgg

Get started with our plugins for XR developers - Ultraleap documentation. (n.d.). https://docs.ultraleap.com/