Rash Paul Wadhwa

WAD23211049  UCB, Birmingham

Programming Principles

[Task 01 1](#_Toc182817588)

[1.1 1](#_Toc182817589)

[1.2 6](#_Toc182817590)

[Task 02 9](#_Toc182817591)

[2.1 9](#_Toc182817592)

[2.2 10](#_Toc182817593)

[Task 03 12](#_Toc182817594)

[3.1 12](#_Toc182817595)

[Task 04 17](#_Toc182817596)

[4.1 17](#_Toc182817597)

[5.1 19](#_Toc182817598)

[5.2 19](#_Toc182817599)

[6.1 22](#_Toc182817600)

[6.2 23](#_Toc182817601)

[6.3 24](#_Toc182817602)

[Task 07 26](#_Toc182817603)

[5.2 26](#_Toc182817604)

[Task 08 26](#_Toc182817605)

[7.1 26](#_Toc182817606)

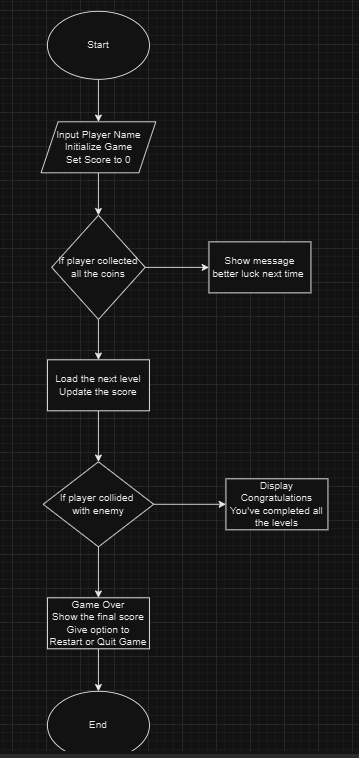
[7.2 29](#_Toc182817607)

[References: 31](#_Toc182817608)

# Task 01

## 1.1

Produce detailed design documentation for planning the game and its constituent code that meets user requirements.



Planning the game:

1. Set up the scenes:

Set up the layout of the scene, create player, enemies, objects, Obstacles etc.

Organize scenes, scripts, materials, images, prefabricated, and text files into folders to proceed.

1. Initialize Game:

Load the player, levels, and enemies.

Set initial score to 0, define the health.

Load UI elements for score display, player name, health, and Game Over screen.

1. Define Player properties:

Such as position, lives, and controls (e.g. move left/right, jump)

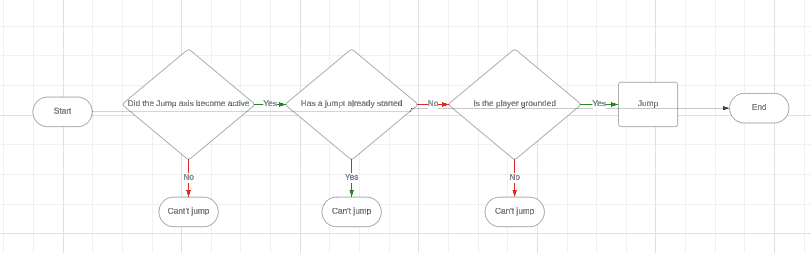
1. Define Object properties:

Objects that change colour when hit.

Coins: collectible items that increase the score. ￼

Enemies: objects that cause a ‘Game Over’ upon collision with the player.

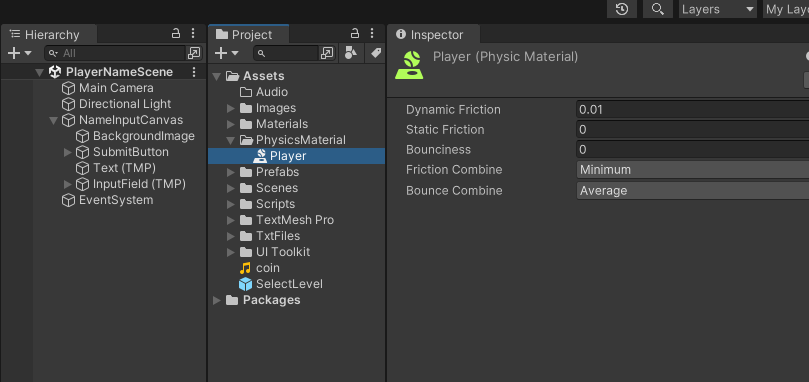
1. Check player input based on movement. Apply gravity to simulate jumping and landing.
2. Player – Object interactions such as if the player collides with a coin, destroys the coin from the scene. Increase player’s score and update the score display. Change the object’s colour if player hits a changeable object.
3. If the player collides with an enemy, display a ‘Game Over’ message. Give option to the player to restart or quit the game.
4. If player completes all levels, display ‘Congratulations’ screen.



I wanted to show through the above diagram that how I tackled the ‘player grounded’ issue. I should only be allowed to jump when I am grounded. For that, I have created a function that checks if the player is grounded and returns true if that is the case.

* I used the method to check whether any of the bottom corners of the player is on top of a collider (an object that produces collision, such as the floor or any object with a collider component).
* Get the player's collider, so I can get its size. I get that collider object in Start, following the same approach I took with the rigid body.
* With the size of the player, I get the positions of all 4 bottom corners.
* I sent a ray cast from these points, pointing downwards. A ray cast is a line that finds colliders. It will return true if it finds a collider such as the floor.
* This way I found that any of the corners is grounded, I will say that the player is grounded.

Now, I had to deal with, what happens if a player jumps against a wall and on pushing towards that direction. The player will get stuck on the wall. This is caused by ‘friction’ I disabled this so that the player does not rotate by creating a physics material. Physics materials allow us to give our game object custom physical properties such as friction, and bounciness, and it will behave when colliding with other objects. In my asset folder, I have created a subfolder called Physics Material, and inside of that folder I created a physics material and named it Player.



I have given this new Properties as shown in the above screenshot, so that the player does not have any friction. I have selected minimum in friction combined. This means that no matter what the friction is of the colliding object, the minimum value will be used for physics calculations.

Pseudocode for the multi-level platform game:

Initialize Game

Initialize Score = 0

For each level in the game:

Load level assets such as coins, platform, colour changeable objects,

While player is alive and not at ‘Game Over’:

Get player input

Update player position based on input

For each coin in level:

If player collides with coin

Remove coin from the scene

Increment score

Update score display

For each changeable object in level:

If player hits changeable object:

Change object colour.

For each enemy in level:

If player collides with enemy:

Display ‘Game Over’ screen

Restart or Quit game option

Check if level completion condition is met

If true:

Load next level

Reset player position

If player completes all levels:

Display ‘Congratulations’ screen.

Purpose of the Unified Modelling Language (UML)

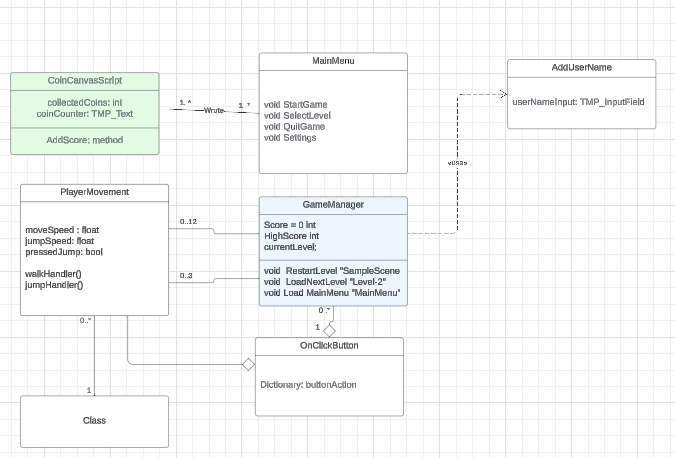
Unified Modelling Language (UML) diagrams are a powerful tool for visualizing and understanding the structure and behaviour of software systems, including using object-oriented principles and in-game development.

UML diagrams provide a clear representation of classes, their attributes, and methods. This helps in understanding the relationships between classes, such as inheritance, association, and aggregation.

UML diagrams can be used to model level layout, including the placement of objects, obstacles, and enemies.

Relationships:

1. Game Manager composes level: The Game Manager manages the current level.
2. Levels: each level has different layout, collectibles, platforms, objects hit and obstacles.
3. Player interacts with Coin, Enemy, Platform. Player interacts with these objects as they collect, collide, or navigate through levels.
4. Main Menu UI communicates with Game Manager. Main Menu UI displays player name, score updates, Game Over messages and level completion based on the conditions met.

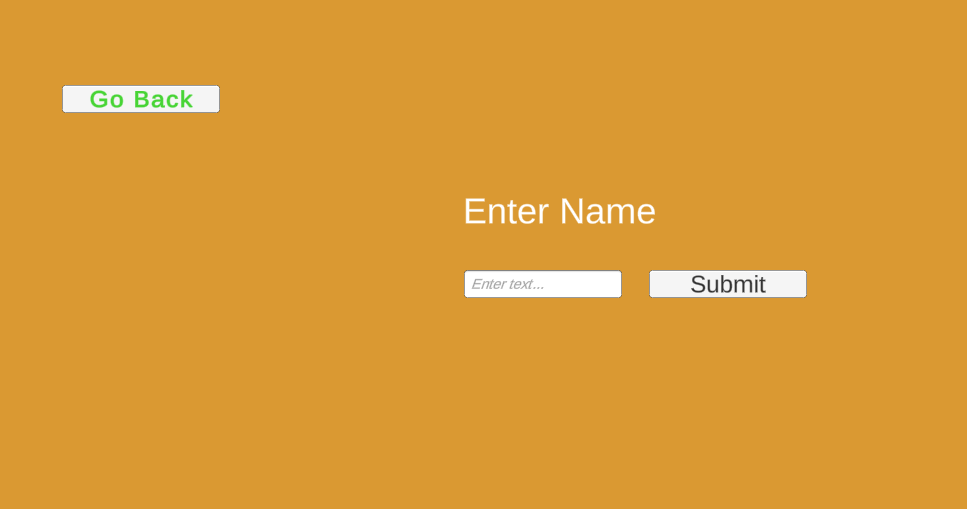


## 1.2

Check the game design documents, to make sure they meet the games requirements.

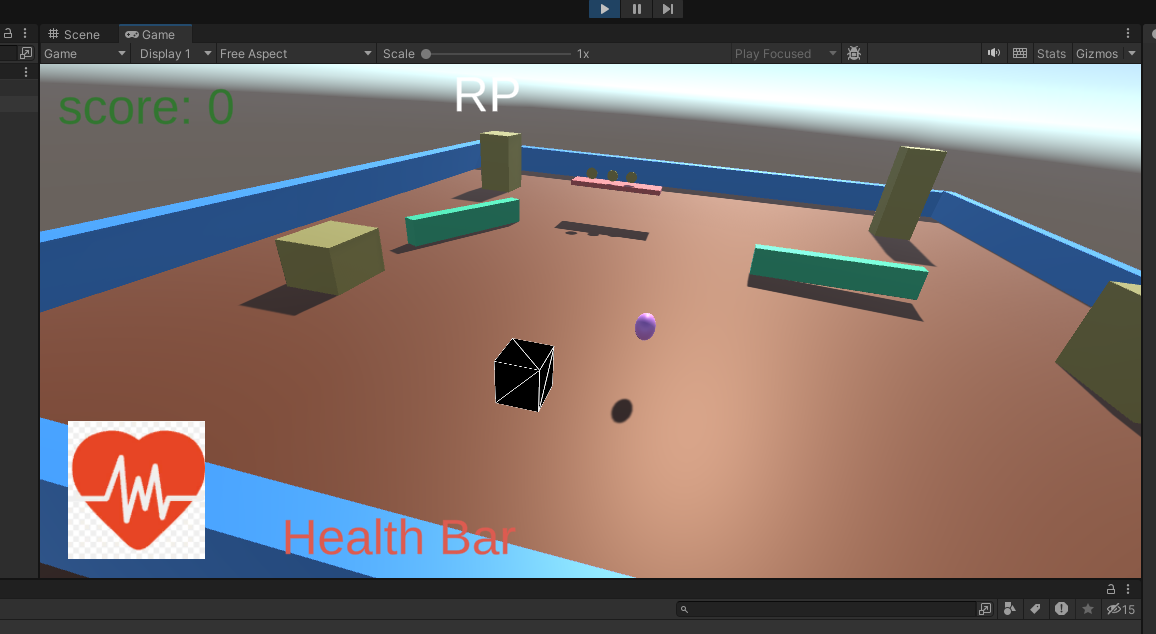
I have checked my game design documents to make sure that they meet the games requirements. I have created a game with total of 3 levels, named Sample Scene, level 2 and Game Over. My game has a Main Menu and the ability to change the settings such as camera sensitivity or any other settings in the inspector. I have created a Go Back button to get back to the main menu and attached the screenshots of all the menu and scenes UI. I have added the functionality to enter the player's name and submit button as per the game requirement. Player name will be stored in a text file, updated, and displayed on play mode. I have made the variable names by using common programming conventions and commented the code as I went along to make the code readable and tried to avoid the duplication. I have followed the Object-Oriented Programming Principles wherever I could.

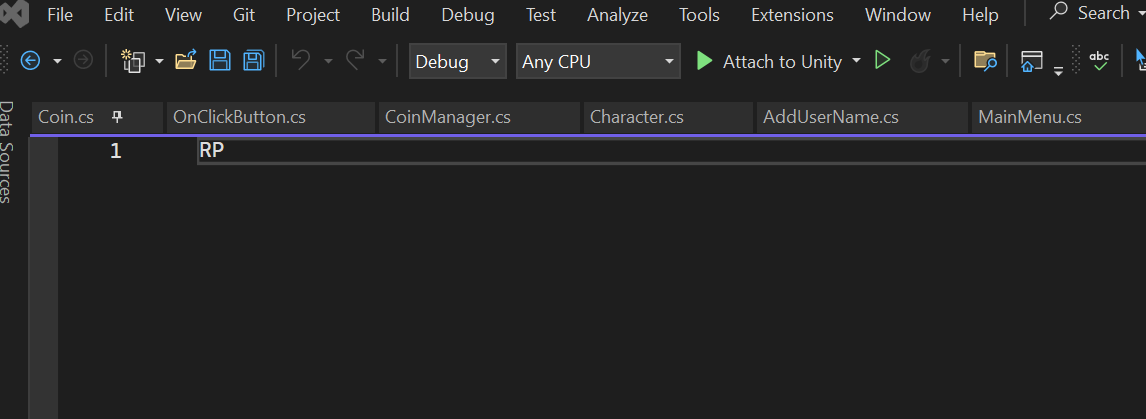












# Task 02

## 

## 2.1

Create the functionality of the game utilising common coding practices including variable naming, commented code and indentations.

I have carefully created the functionality of the game utilising common coding practices including variable naming, commented code to make the code easier to read and debug errors and indentations. I have added few screenshots to show this. I have used clear, descriptive names for variables that convey their purpose such as player Score, walk speed, jump speed. I have included comments to describe the purpose of code sections, functions, and classes. Comments helps the explain the logic, making it easier for others to understand and troubleshoot. I followed the consistent indentation which visually separates code blocks such as conditional, loops and functions. This improves readability, which is crucial for debugging and collaborating with others.

As I have learned these practices to improve code quality, readability, and maintainability.



## 2.2

Create code in an appropriately modular way utilising classes and functions.

I have created classes and functions using the correct naming conventions which makes easier to connect the dots and easy to debug the errors. I also converted the if / else statements into methods in On Click Button script to make the code easy to read and can be recalled whenever the relevant button presses.

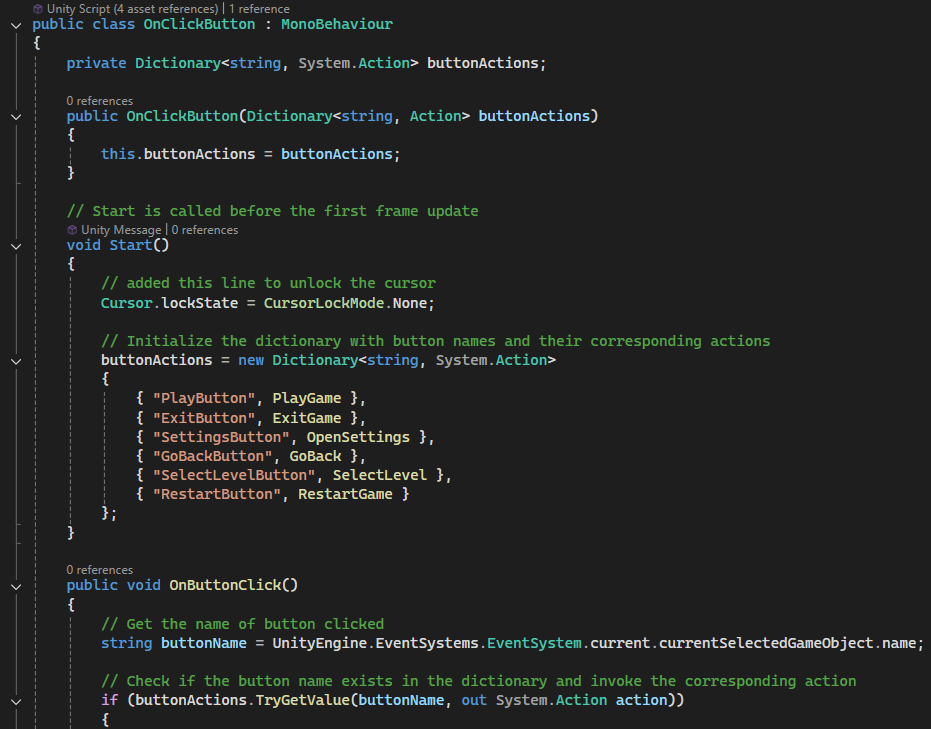
For example, The Main Menu script purpose is to handle the main menu interactions such as starting the game, opening the settings menu, selecting the levels, and exiting the game.

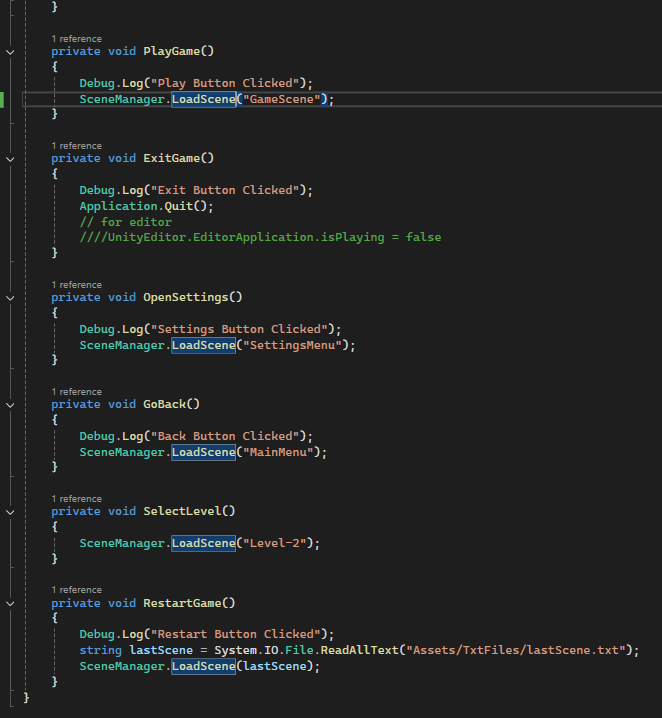
Secondly, the Player Movement script manages the player’s movements, jump and interaction within the game world.

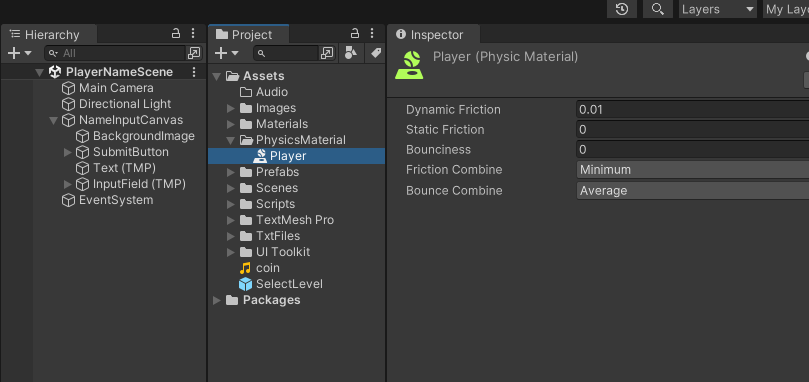
Thirdly, the Add Username allows the player to change settings such as character name and Settings script allows the player to change the control schemes.

Finally, the Select Level and Game Over manages the transitions, restarting levels, and returning to the main menu.

With regards to the functions / methods, in my On Click Button script I have created a dictionary which initialize with button names and their corresponding actions. This checks if the button name exists in the dictionary and invoke the corresponding action. For example, Play Button, Exit Button, Settings Button, Select Level, Go Back Button, and Restart Button. I have added the screenshots for your reference.







# Task 03

## 3.1

Create code for the game that utilises the following:

* Appropriate variable names, types, and sizes.  
  public int coin Collected = 0;

public float rotate Speed = 100f;

// [Serialize Field] makes the variable visible in the Unity editor

[SerializeField] float move Speed = 8f;

[SerializeField] float jump Speed = 10f;

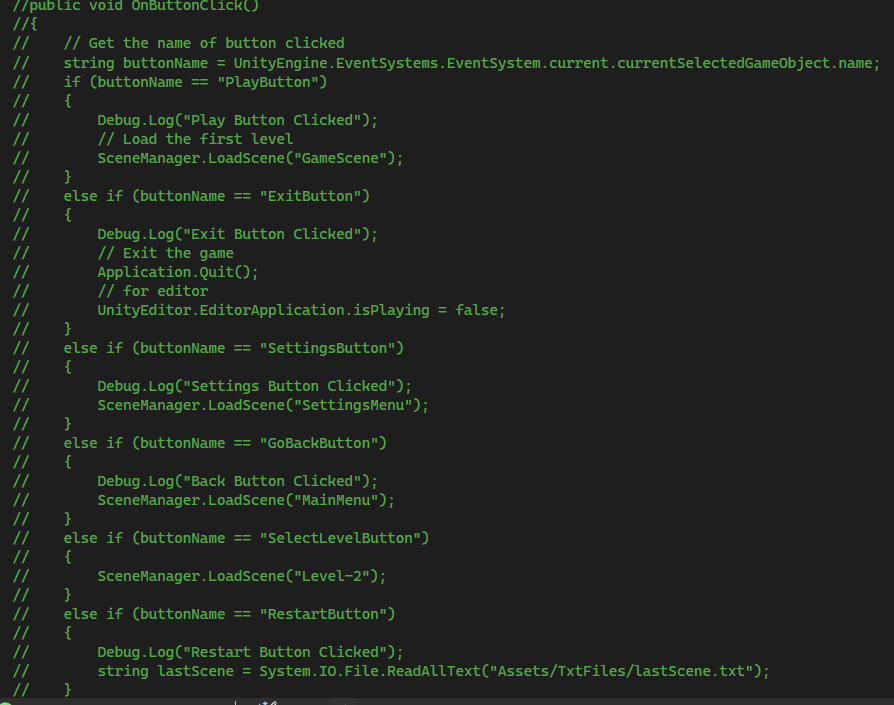
bool pressed Jump = false;

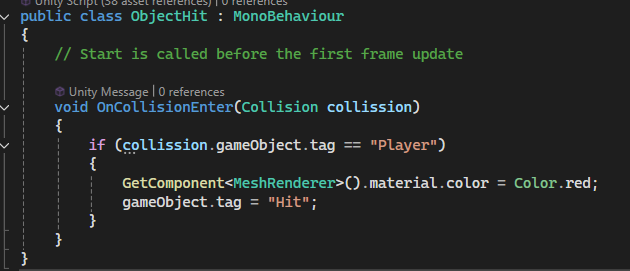
Collider Coll;

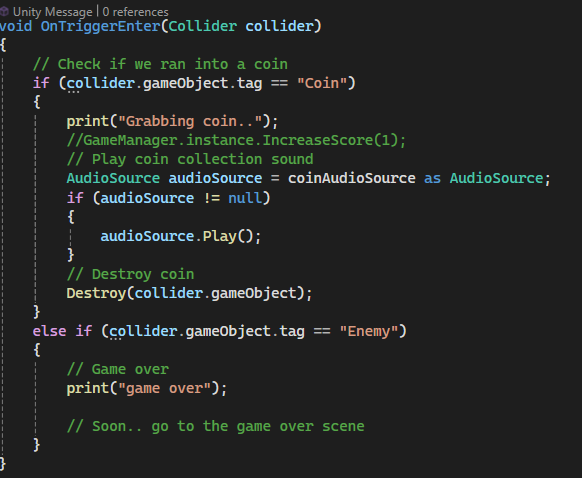
Rigid body rb;

* If/, else if and else statements

I have added 3 screenshots to show, how I used if, else if and else statement in my scripts. The first one which is commented out, was in On Click Button script which I have converted into methods after taking feedback from my teacher. Another screenshot where I have used this if, else if and else statements in On Trigger Enter, and On Collision Enter.

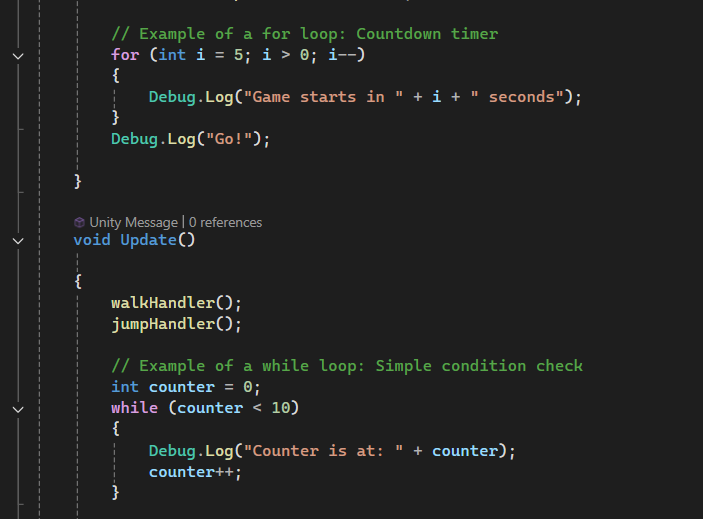






* While and for loops

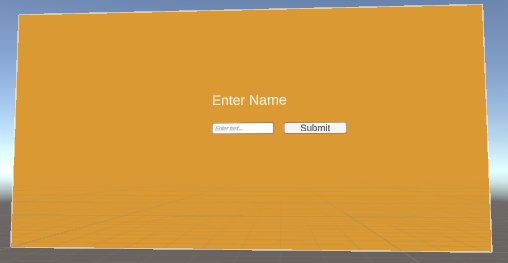
I have used this for loop example to countdown the timer, where initializing the variable, setting the condition and decrement it. I have learned that for loops are best fit for iterating purposes.

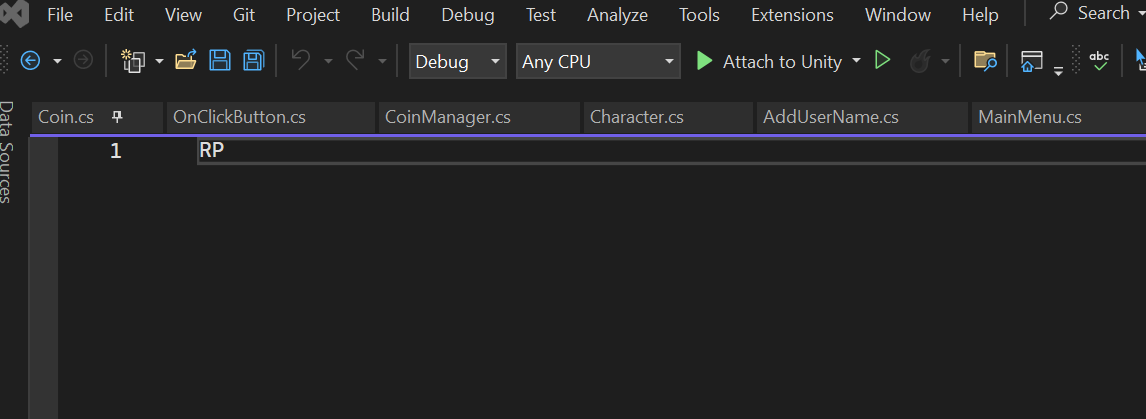


In the above screenshot, I have used the while loop to check a simple condition, where I set the variable named counter to 0 and checking for the condition that if it is less than 10, show message on the screen and finally the increment.

* User data storage (i.e. usernames and high scores)

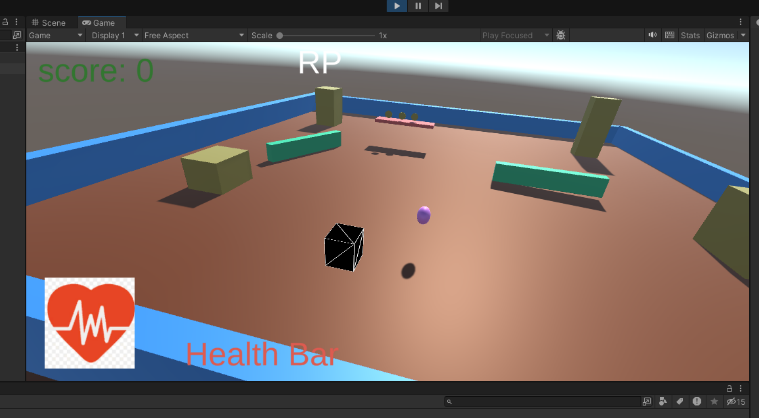
I have created a folder named TXT Files to store the usernames and display the high scores. This will retain the high scores and player name and update on play mode.

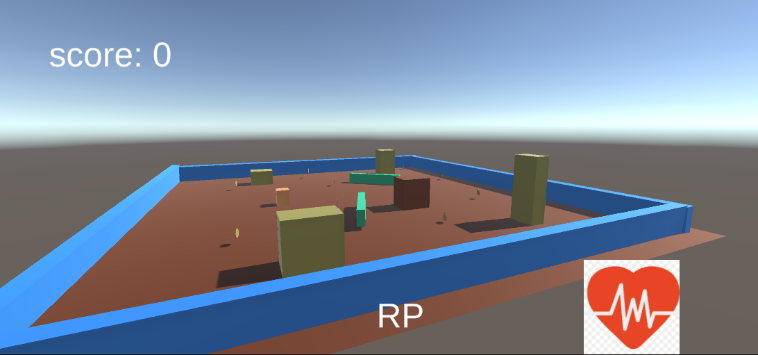




* UI elements such as dynamic text and images such as high score, character health etc.

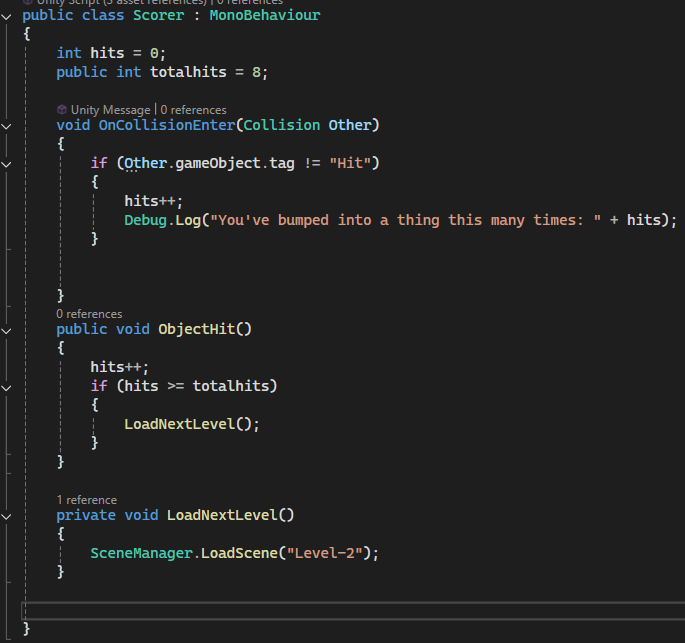
In the below screenshot, it shows the initial score and character name and health, which changes as the game goes along.





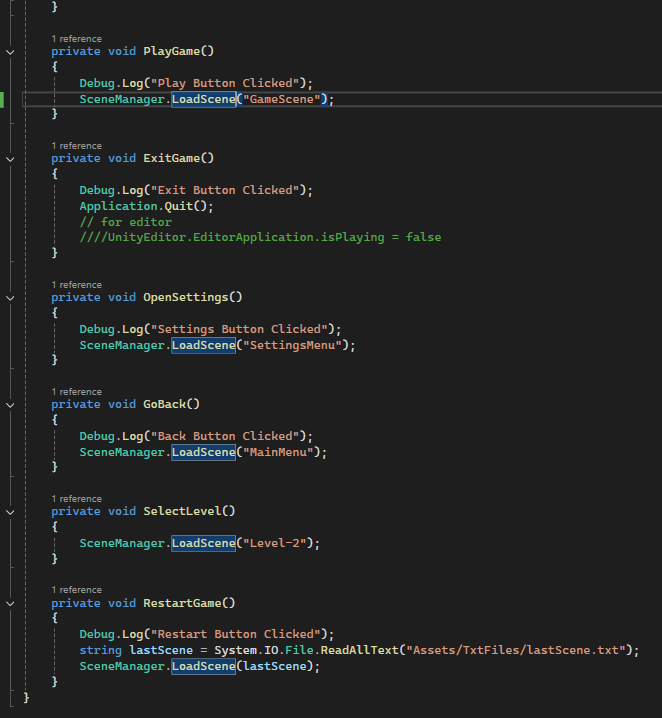
* Mathematical calculations for scores (Arithmetical Operations)

I have used <, ++, --, and => in for and while loops. I also used this to count the Hits in Scorer script which is attached to sample scene. I have attached the for / while loop example on page 13 for your reference.



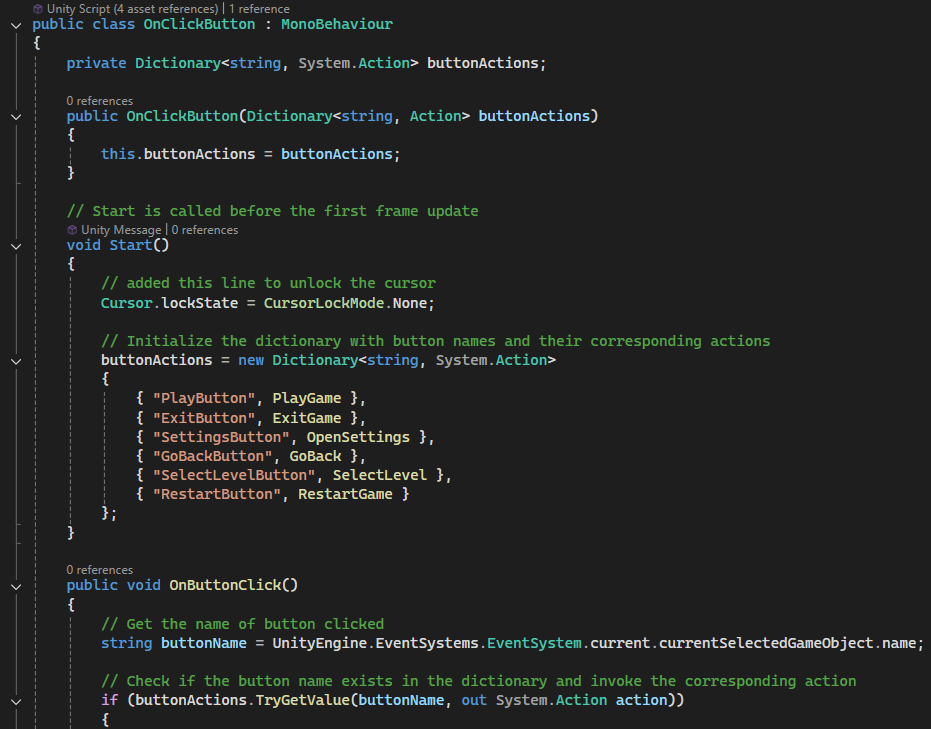
* Functions/ Methods

The below screenshot is an example of a few methods I created which initialize with button names and their corresponding actions. This checks if the button name exists in the dictionary and invokes the corresponding action.

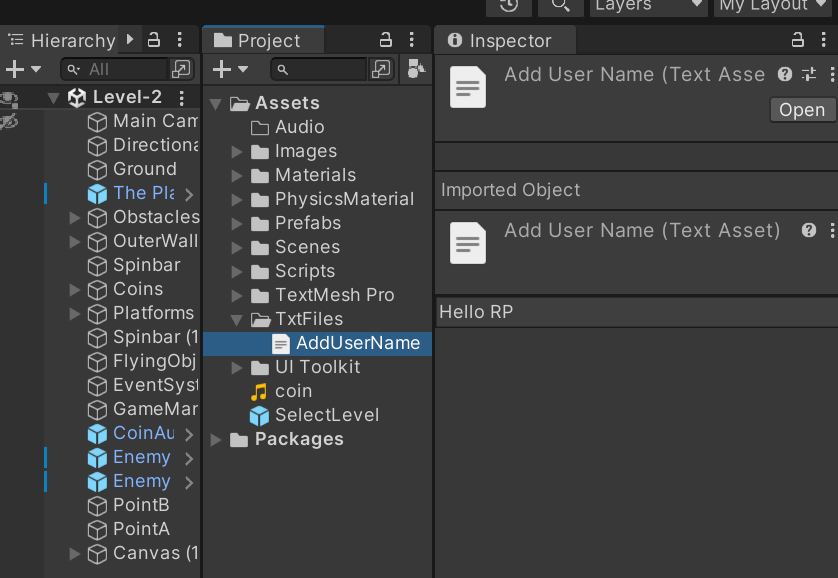


* Using appropriate data structure for storing & accessing data

In this game I have used the dictionary to store data in key-value pairs, which allows fast lookups by key. They are useful for storing and retrieving data where each item has a unique identifier. For example, checking for existed buttons to invoke the corresponding actions.



I also want to show via the below screenshot that how I organized the files in separate folders to retrieve and store the data.



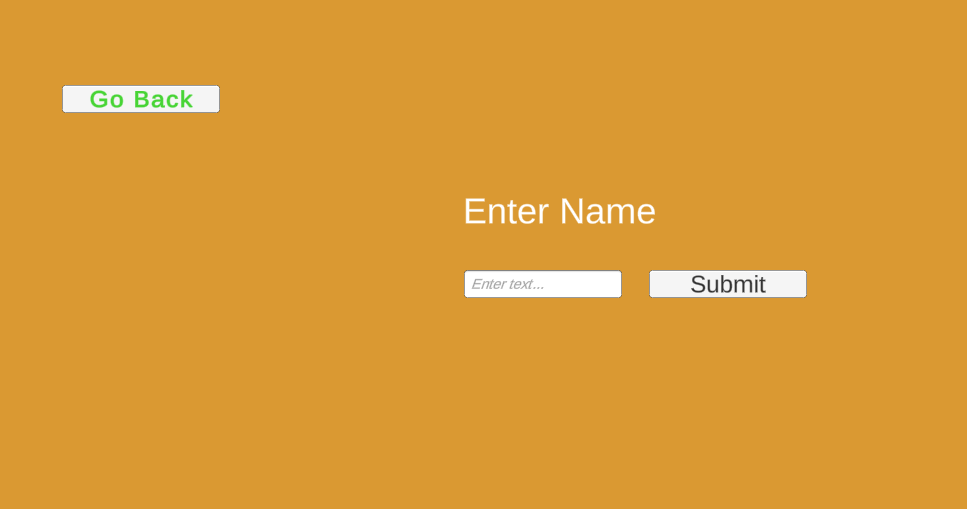
# Task 04

## 4.1

Create a functional game that is user friendly and accessible for both inputs and  
outputs via the following:

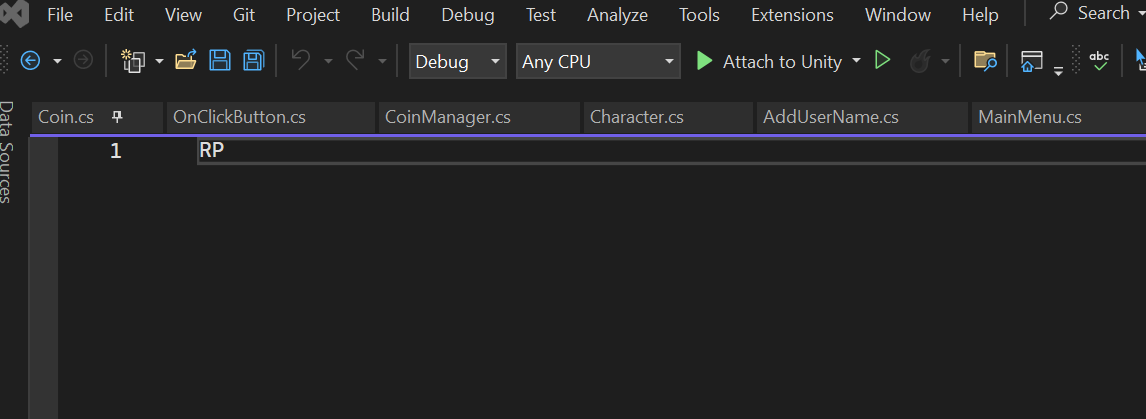
1. Appropriate user prompts for inputs (i.e. controls and username...etc)

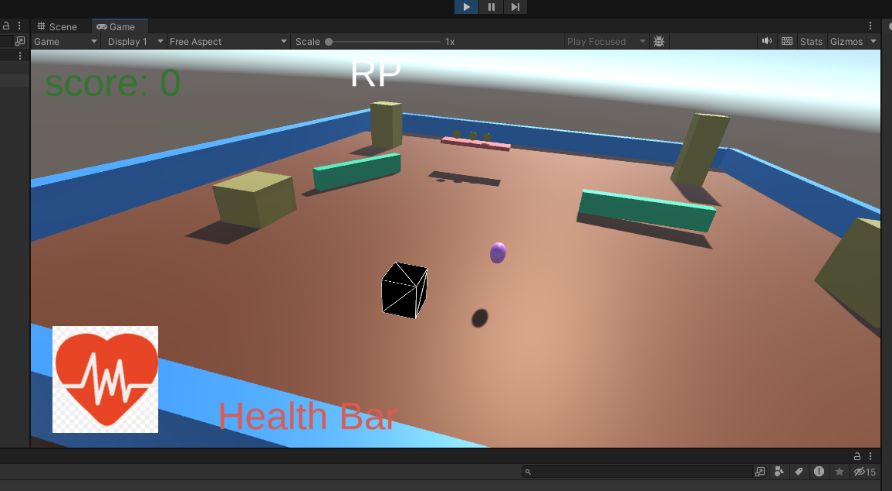
The below screenshot is a proof of user input, where user can Enter Name and submit it which will be stored in text file format and display in play mode. Also, player can use this Go Back button to go back to Main Menu.



1. Formats inputs appropriately for output.

The below screenshots are proof of storing the player's name and displaying while on play mode.





1. The above meets the game’s user requirements

I can justify that my game design and development of this game align with the goal of creating an engaging, interactive experience for players. Using popular game mechanics, the game is set in a visually appealing world where player navigates multiple levels, collect coins, hit objects which changes colour on hit, and faces a few challenges as the game goes along.

The game’s structure encourages skill building by introducing few obstacles and enemies to proceed to next level, and if collided with enemies, the message prompt ‘Game Over’ and offer the player to either restart or quit the game.

This is a simple game where players must navigate carefully to avoid game-over conditions, while collectible coins and hitting objects provide reward incentives and maintain interest. The ability to interact with game environment (e.g. changing object colours) adds a creative element, allowing players to feel more connected to the game world. I consistently used the aesthetics features such as font and colour throughout the UI canvas to give a bright and entertaining environment to users.

This is my first game, and I tried to utilize the object-oriented programming (OOP) principles, such as inheritance, encapsulation, and polymorphism, making future game enhancements more manageable. Additionally, common coding practices such as clear variable naming, code commenting, and organized indentation create a readable and maintainable codebase.

Overall, I have learned and tried this game combines entertainment with an approachable learning curve, serving an ideal platform for myself who is trying to understand structured game design and programming and for casual players.

Task 05

## 5.1

Use Visual Studio/Visual Studio Code to create & edit the game’s source code  
using C# and Unity Engine.

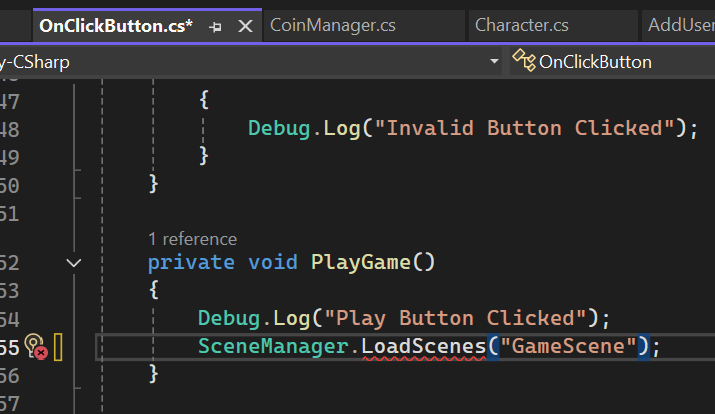
I have used the Visual Studio to create and edit game’s source code using C# and Unity Engine. This was a wonderful experience to learn and develop my knowledge as this IDE has lot to offer with regards to debugging, IntelliSense and Unity API Referencing under the Help menu which I found very useful tool as I hover onto some error and need a clarification to deal with it, just go to help and Unity API referencing this took me to the Unity document directly. I have attached the link on page 22 under the references.

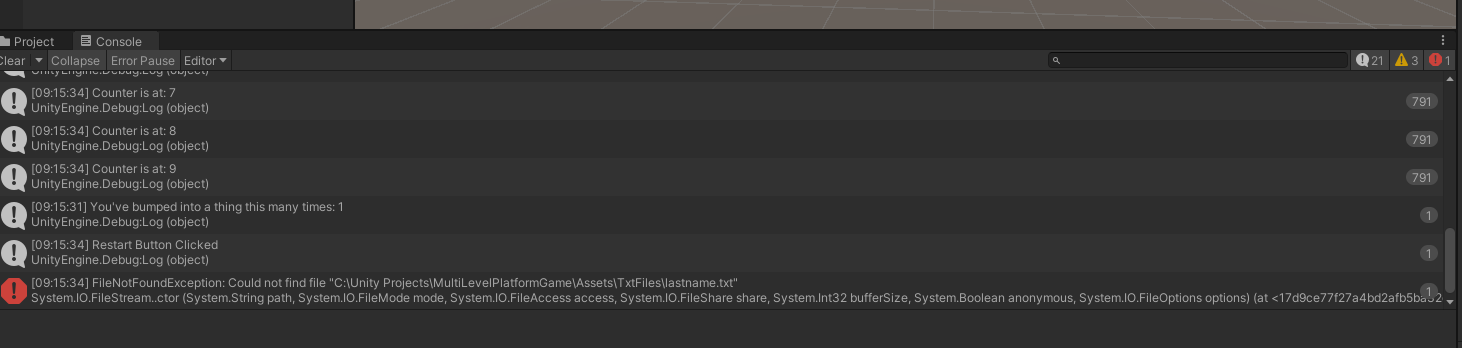
## 5.2

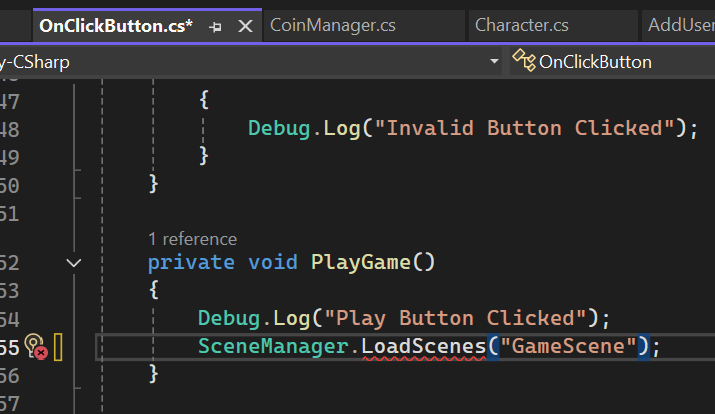
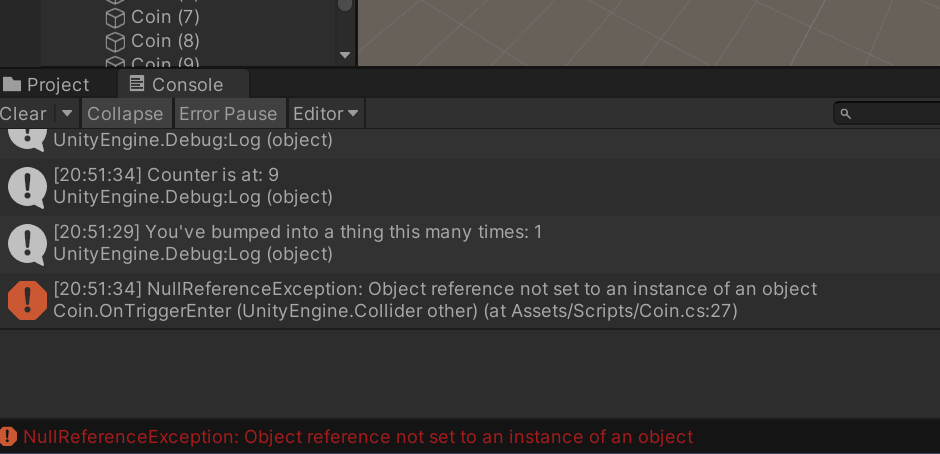
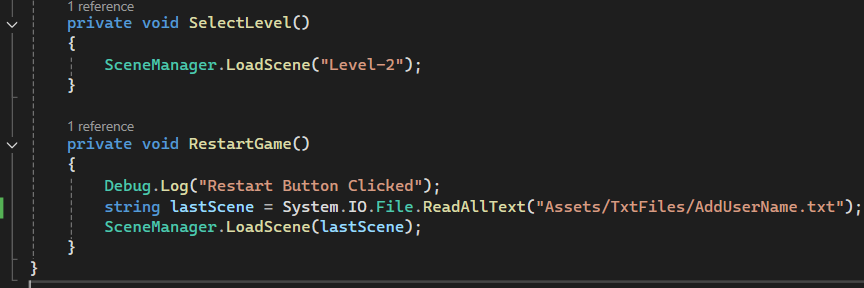
Create detailed testing plans and evidence any bug fixing using a Unity debugger.

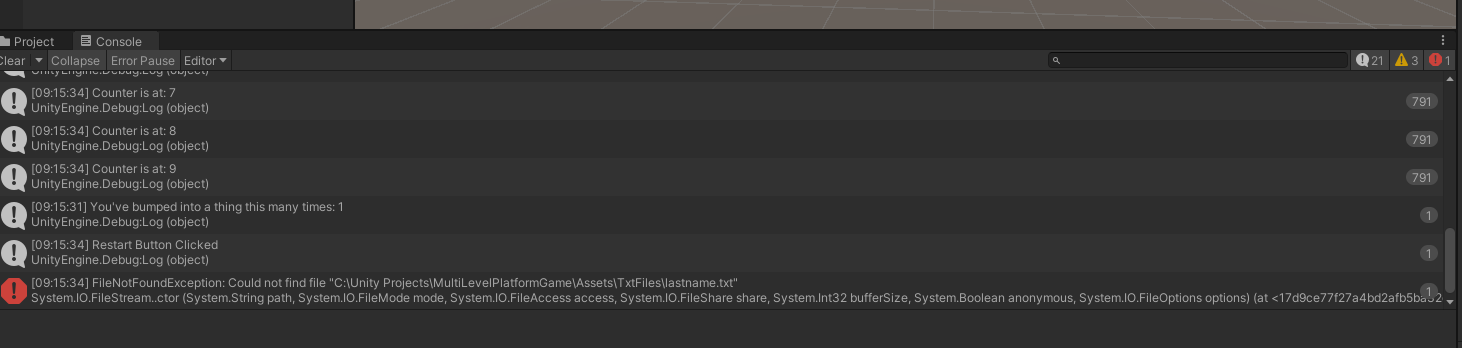
I aim to ensure that the game is functional, free of critical bugs, and provides the expected player experience. I ensured that all core mechanics such as movement, jump, coin collection, score updates, message appearing up on screen, enemy collision, and level transitions.

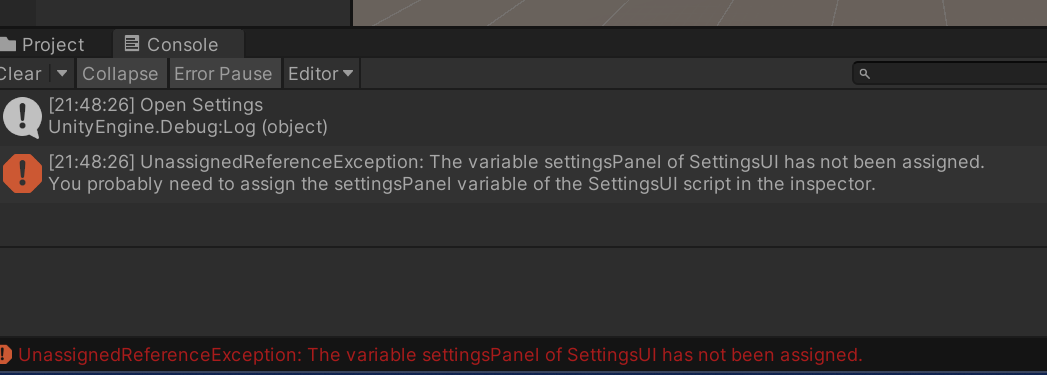
I used Unity Debugger for real-time variable inspection and script debugging.











Task 06

## 6.1

Create detailed testing plans and provide evidence in your report

My objective was from this testing plans to ensure that my game is functional, error free and provides the expected player experience. I ensured that all core mechanics such as movement, jump, coin collection, score updates, message appearing up on screen, enemy collision, and level transitions. I have created a table to show the test I did and the outcome of those test and what action I have taken to fix those bugs.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Test Case  Description | Test Steps | Expected Results | Actual Results | Pass/Fail Status |
| 1 | Test coin collection | Played and collected to ensure score updated | Score increments by 1, coin disappears. |  | Pass |
| 2 | Messages appear on start of the screen and disappear after few seconds. | Played the game to ensure message appears up | Should show up in starting of the game |  | Pass |
| 3 | Ensuring buttons working on click | Checked every button by clicking | Should click |  | Passed after fixing error |
| 4 | Checked for Text reference is in place | Played to see if any error occurs | Should take text reference |  | Passed after fixing error |
| 5 | Test settings, menu functionality | Change a setting like player name in the settings menu | Settings updates correctly and persists |  | Pass |
| 6 | Ensuring error free run after building up |  | Should run after building |  | Passed after fixing error |
| 7 |  |  |  |  |  |

## 6.2

Utilise effective testing data and test cases that are documented and evaluated  
thoroughly.

Well-crafted testing data and test cases are crucial to ensure the quality and reliability of software applications. They help identify defects, improve performance, and enhance user experience.

Test 1: I wanted to make sure that score updates as player collect coins, but it was not happening at first. To do that I created a script attached directly to canvas and found that the code line to update the score was in start method rather than in update method, sooner i fixed that it worked.

Test 2: I showed the game to my nine-year-old daughter and asked her to play. The first question, she asked me what button I should press daddy which made me think about target audience and I decided to put a message on screen as the game starts, telling the user what button needed to press to play the game. To that I created a script named Message Controller, attached it to Canvas, assigned a variable timer, and used an if else statement ensure message disappear after few seconds.

Test 3: Ensured all the defined buttons in Main Menu UI and other UI works as expected. I found play button stopped working and to fix that I found a spelling error in script which I fixed, and it worked as i wanted.

Test 4: The TMP Text in Coin Canvas Script was not updating. To fix that error I checked in inspector the object reference was not assigned. It worked after I assigned the reference in empty field.

Test 5: Checked that player name updates and showed while playing, after entering new name in settings menu.

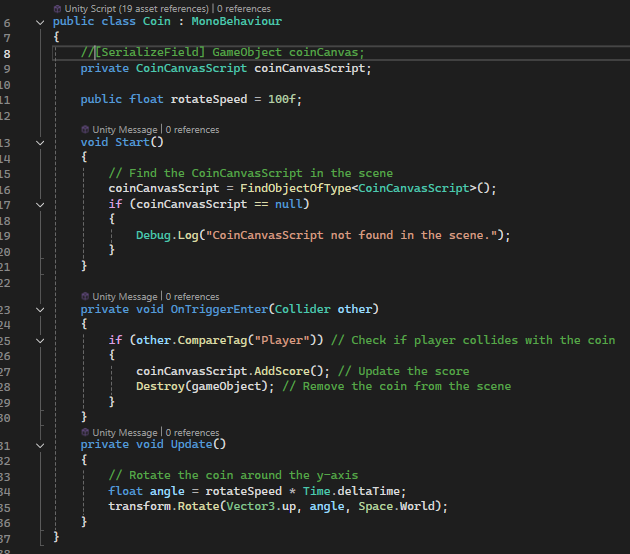
Test 6: I wanted to ensure that before submitting the assignment, it should run error free so after building the scene I ran it and error appeared up saying Text file not found, check the path. I created a new folder, named assets, dragged the text file into that folder and it worked.

## 6.3

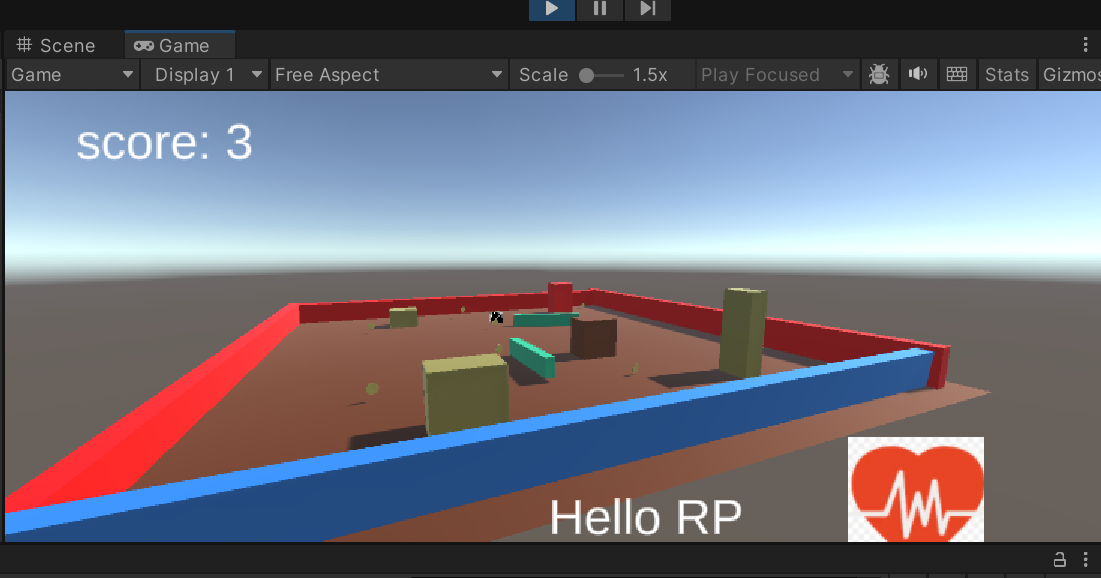
Resolve any bugs in the game that were encountered to fix them. Ensure to  
evidence them in your report.

Here is a list of bugs I encountered, and the evidence added to show that how I fixed those.

1. Coins were not rotating in sample scene.
2. Player was going out of the boundary walls
3. Dropping object was falling too soon
4. Flying object was pushing the player out of the walls
5. Jump was not working properly
6. Score was not updating
7. There was no message on screen to tell what keys need to use to play the game.
8. Script corrected



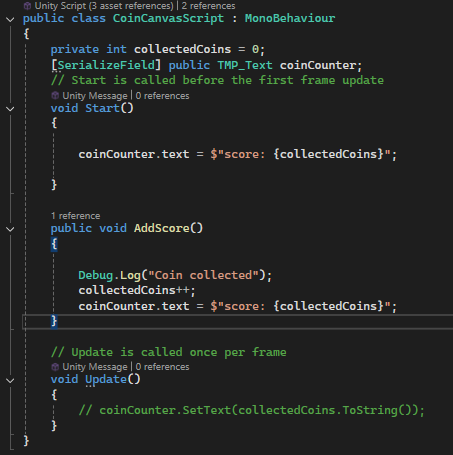
1. Layout fixed 3. changed the time for dropping object



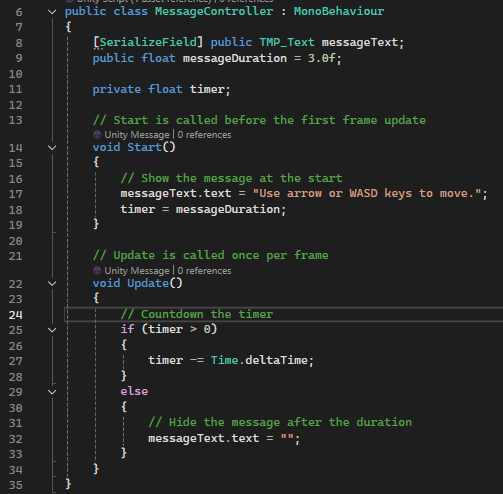
4.Removed flying object as it did not work as expected.

5. Jump working

6. Fixed code



7. Message script created



# Task 07

## 5.2

Compile the game into an executable file to run on any Unity compatible windows  
machine.

To compile a Unity game into an executable file for Windows, I used Unity’s built-in build system. To prepare my project, I went to File > Build Settings. Added all the scenes I want to include in my build. Before building, I tested my game thoroughly to identify and fix any bugs or issues. To build the project, in the build settings window, I clicked build button. A file browser window appeared. I chose a destination folder to build.

# Task 08

## 7.1

A) Based on your experience of creating the game

i. Explain the principles used in the game creation (for game mechanics, language  
choice, conventions...etc)

Game creation is a complex process that involves a variety of principles to ensure a successful and engaging experience.

For example, Game Mechanics:

The cycle of gameplay should be clear, rewarding, and easy to understand. To do that every game should have some challenge and reward. A well-balanced game offers challenges that are neither too easy nor too difficult, providing a sense of satisfaction upon completion. Secondly, progression a player should feel a sense of accomplishment as they progress through the game, unlocking new abilities, challenges, or content.

Language Choice:

C# is Unity’s primary language, streamlining the integration of code with Unity’s game objects, physics, and engine events. C’ supports object-oriented programming, which aligns with the structure such as classes for player, coins, and enemies. This has a vast developer community, and substantial amounts of resources and support are easily accessible via Unity Documentation, accelerating the development process.

Coding Conventions:

Game text should be clear, concise, and easy to understand, avoiding jargon or overly complex messages. Another important thing to consider is consistency. Consistent font, colour, and text throughout the game help to create a cohesive experience.

Naming conventions for descriptive variables and file names to convey their purpose. For example, CamelCase is used for variables and Pascal Case for classes, aligning with C# and Unity conventions.

Code Commenting is another important way of showing that the code is organized into functions and classes that handle specific tasks to improve readability and reusability.

Finally, User Experience (UX) and User Interface (UI):

The game should be intuitive and easy to navigate, with clear instructions and minimal frustration.

The UI should be visually appealing and functional, providing easy access to essential information and controls.

iii. Explain the potential advantages of using object orientation.

Object-Oriented Programming (OOP) is a programming paradigm that organizes software design around data rather than functions and logic. This approach offers several advantages, particularly in game development. This can significantly improve the quality, maintainability, and scalability of software applications or projects. There are 4 major paradigms which are used in object orientation.

Inheritance, Abstraction, Polymorphism, Encapsulation and Maintainability.

Inheritance:

Inheritance allows us to create new classes (child classes) that inherit properties and methods from existing classes (parent classes). This reduces code duplication and promotes code reuse.

Another advantage of this is that it creates a clear hierarchical relationship between classes, making code more organized and easier to understand.

Extensibility: New functionality can be added by creating child classes that extend the parent class.

Abstraction:

In object-oriented programming, Abstraction involves focusing on the essential features of an object while hiding the unnecessary details. This principle allows developers to manage complexity by breaking down large systems into smaller, more manageable components.

It hides implementation details. Abstraction focuses on what object does rather than how it does. This means that the internal workings of an object are hidden from the user, making it easier to interact with and understand.

Increased Reusability: Abstract classes and interfaces promote code reuse by providing a common framework for different implementations.

Enhanced Security: By hiding sensitive information, abstraction can help protect against unauthorized access and manipulation.

Overall, abstraction is a powerful tool that enables developers to create well-structured, maintainable, and reusable quality software.

Polymorphism:

Flexibility: This allows objects of different classes to be treated as if they were objects of a common superclass, making code more adaptable.

Dynamic Binding allows the specific method to be called at runtime, based on the actual type of the object, rather than its declared type. This enables more flexible and dynamic behaviors.

Encapsulation:

Data Protection: Encapsulation can help protect sensitive data by making it accessible only through well-defined interfaces. This reduces the risk of accidental or malicious modification of data.

Reusability: Encapsulated objects can be reused in various parts of the application or in other applications.

Maintainability:

Easier to Understand: OOP promotes a more structured and organized approach to software design, making code easier to understand, debug, and maintain.

Easier to Modify**:** Changes can often be made to a specific object or class without affecting other parts of the application, reducing the risk of introducing new bugs.

## 7.2

B) Describe how assembly language could be used, and how it differs from high level languages, especially for game development.

Assembly language is a low-level programming language that interacts directly with a computer’s hardware. It provides precise control over the machine’s operations, making it suitable for tasks that require maximum performance or specific hardware interactions.

How assembly language is used:

Assembly language is often used to write device drivers, as it allows for direct interaction with hardware components like graphics card, sound card, and network interfaces.

Systems that require precise timing (Real-time systems) such as those used in medical devices, industrial automation, and aerospace, often rely on assembly language. Even though some parts of operating systems, particularly the kernel, are written in assembly language to optimize performance and security.

It is not common for modern game development, but assembly language can be used to optimize specific parts of a game, such as time-critical routines or graphics rendering.

How is different from High-Level Languages:

High-Level Languages, like Python, C++ or java are designed to be more human-readable and easier to program in. They take away the complexity of hardware, allowing developers to focus on the logic of their programs.

There are some key differences between assembly language and high-level languages, especially in the context of game development.

1. Performance: Assembly language can offer significant performance advantages, as it allows real-time control over hardware resources. However, it is more time consuming to write and debug.

2. Portability: Assembly language is specific to a particular processor architecture, making it less portable than high level languages.

3. Complexity: Assembly language is significantly more complex to learn and use than high-level languages. It requires a deep understanding of computer architecture and assembly language syntax.

Why not assembly language is good for modern game development:

While assemble language offers performance benefits, it is not practical to use for modern game development due to the complexity, portability, and time-consuming. Writing large-scale games in assembly language is extremely time-consuming and error prone. High-level languages offer significantly faster development times, allowing teams to iterate more quickly. The other reason is that assembly code is often tied to specific processor architecture, making it difficult to port games to different platforms.

Overall, High-Level languages are the preferred choice for most game development projects due to their ease of use, portability, and ability to handle complex game logic.

# References:

Technologies, U. (n.d.). *Unity - Manual: Asset Workflow*. [online] docs.unity3d.com. Available at: <https://docs.unity3d.com/2020.3/Documentation/Manual/AssetWorkflow.html>.

‌

