

Faculty of Applied Sciences

Bachelor of Science in Computing

**COMP490 Final Year Project**

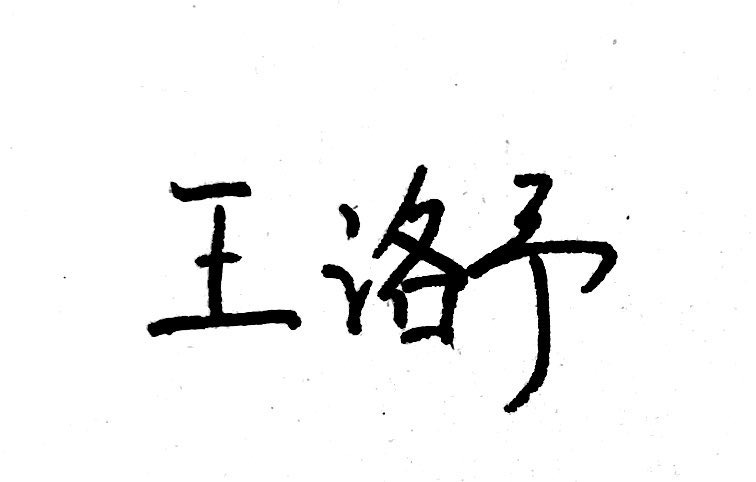
**Progress Report**

Academic Year 2022/23

|  |  |
| --- | --- |
| Indie Game Design and Development | |
|  |  |
| Project number: | Your project number |
| Student ID: | P1908404 |
| Student Name: | Vivia Wang |
|  |  |
| Supervisor: | Liam Lei |
| Assessor: | Your assessor |
|  |  |
| Submission Date: | Submission date |

Declaration of Originality

I, Wang Luoyu, declare that this report and the work reported herein was composed by and originated entirely from me. This report has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given in the bibliography.



2022/11/04

Abstract

Computer game is a excited and interesting topic for all of us since it was born. This report introduce the final year project and its progress. As a indie game, this report shows its background and analyzing its best making software. The functions’ building progress in this game by now is the main part of this report. After that, the report also shows the future work for this project in the next semester.

Table of Contents

[List of Tables 7](#_Toc31099)

[1 Introduction 8](#_Toc15120)

[1.1 Objectives 9](#_Toc18436)

[1.2 Risk Assessment 10](#_Toc4992)

[1.3 Summary 11](#_Toc25527)

[2 Background and Related Work 12](#_Toc32254)

[2.1 Background 12](#_Toc26004)

[2.2 Related Work 15](#_Toc17920)

[3 Completed Work 19](#_Toc28644)

[3.1 Project Outcome 20](#_Toc7849)

[4 On-going and Future Work 42](#_Toc7858)

[4.1 Partially Done Work 42](#_Toc2946)

[4.2 Functions to be Added 43](#_Toc30523)

[4.3 Work for the 2](#_Toc23987)[nd](#_Toc23987) [semester 43](#_Toc23987)

[5 Conclusion 45](#_Toc14612)

[References 46](#_Toc12167)

Table of Figures

[Figure 1 Steam——Developed by Valve company, is one of the most popular game platform in the world 9](#_Toc20757)

[Figure 2 Probability impact matrix before proposed solution 10](#_Toc30137)

[Figure 3 Indie developers can escape the adverse effects caused by the pandemic 13](#_Toc25680)

[Figure 4 A screenshot for Vampire Survivors 14](#_Toc1871)

[Figure 5 The development interface of Unity 16](#_Toc6161)

[Figure 6 The logic structure for this program 18](#_Toc3998)

[Figure 7 Part of the script for player movement 19](#_Toc27318)

[Figure 8 Different animation when the character idling and moving 19](#_Toc17182)

[Figure 9 Part of the script for formulate weapon data type 20](#_Toc4234)

[Figure 10 Part of the script for weapon manager 21](#_Toc13715)

[Figure 11 12 Part of the script for close-range weapon attack 22](#_Toc24985)

[Figure 13 The effect for close-range weapon 22](#_Toc22149)

[Figure 14 Part of the script for projectile weapon 23](#_Toc22979)

[Figure 15 the effect for projectile weapon 23](#_Toc4593)

[Figure 16 17 Part of the script for character’s HP system 24](#_Toc24336)

[Figure 18 The effect for HP bar 24](#_Toc26942)

[Figure 19 20 Part of the script for EXP system 25](#_Toc16963)

[Figure 21 22 The effect for EXP bar 25](#_Toc6258)

[Figure 23 24 Part of the script for formulate upgrade data type and upgrade option pool 26](#_Toc11698)

[Figure 25 Part of the script for upgrading system 27](#_Toc7078)

[Figure 26 27 Upgrading options in modulation interface 28](#_Toc30292)

[Figure 28 Part of the script for initialize armor’s basic functions 29](#_Toc10424)

[Figure 29 Part of the script for passive item class 30](#_Toc796)

[Figure 30 The player is attacking by an enemy 30](#_Toc28991)

[Figure 31 Part of the script for basic function for the enemy 31](#_Toc16005)

[Figure 32 Part of the script for popup message pool 32](#_Toc16580)

[Figure 33 Part of the script for set active or not for each message object 33](#_Toc18031)

[Figure 34 The effect for damage popup 33](#_Toc19247)

[Figure 35 Part of the script for chests’ behaviour 33](#_Toc20009)

[Figure 36 A chest in the game 34](#_Toc16255)

[Figure 37 Part of the script for spawning system 35](#_Toc26319)

[Figure 38 The actual range for spawning enemy 36](#_Toc29374)

[Figure 39 Part of the script for destroying system 37](#_Toc8662)

[Figure 40 The music editor in Unity 37](#_Toc14132)

[Figure 41 The game’s start menu 38](#_Toc16869)

[Figure 42 Script for pausing the game 38](#_Toc13276)

[Figure 43 The condition for losing the game 38](#_Toc28134)

[Figure 44 Game over panel in the game 39](#_Toc20500)

[Figure 45 A coin in the game 40](#_Toc6750)

[Figure 46 The gantt chart for the second semester 42](#_Toc1944)

# List of Tables

[Table 1 Table of prioritized risk 9](#_Toc14643)

[Table 2 Comparison between Game Engines 16](#_Toc6701)

# Introduction

Do you play video games? What is your favourite game? Video games have been very popular with teenagers since their occurred. Today, with video games’ development, they involve more and more concepts and aspects about our life, and their themes have gradually become more and more profound and different from the initial "just for fun". Like movies and plays, many people are now calling games "the ninth art." Depends on different types of developers, now there are two main types of games, which are indie games and mainstream game developed by large companies.

However, for adults, as the pace of society accelerates, they have to devote more time to their work or study, and the time for them to play video games is often fragmented. This makes it difficult for them to play games which require time to practice their gaming skills, or because the gaps are too long, they tend to forget the stories of games with large-scale continuous plots. For these reasons, when many adults want to play games in their free time, they have to face these frustrating facts, so it's hard for them to enjoy gaming in this situation.

Therefore, that is why it is necessary to design some casual mini-games for adults. They often do not have obscure plots, do not require too much learning time cost, and allow players to pass a level within a few minutes to ten minutes, so players can relax and enjoy a sense of accomplishment quickly. Also, since these types of games tend to be played in fragmented times, they are best played on mobile platforms as well, such as mobile phones or game consoles.

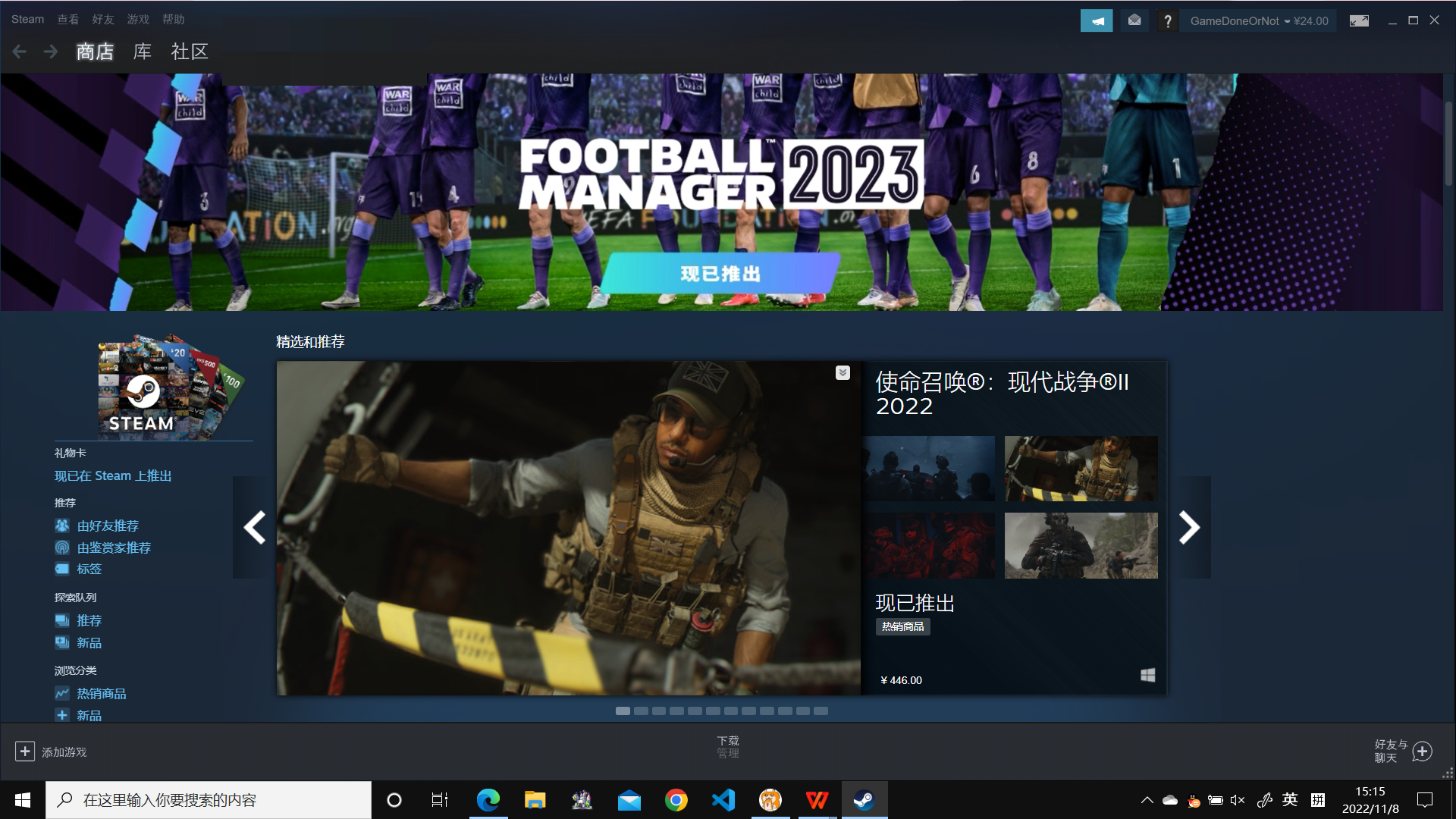


Figure 1 Steam——Developed by Valve company, is one of the most popular game platform in the world

## Objectives

The main objective of the project is to develop an indie lightweight casual indie 2D game with multiple levels, which allows players to enjoy the sense of achievement brought by completing levels while attack, dodge and solve enemies, discover different maps and pick up buff items.

This project aims to achieve:

* Design the whole structure of the game
* Create the video and sound effects that are used in the game
* Develop the game interface
* Develop the different game level
* Develop the game interaction system
* Write the scripts which implement specific game functionality

After achieving these, this project also needs to be added some advanced features. Such as running on multi-platforms, or different difficulty levels to make it more interesting and easy to play for everybody.

## Risk Assessment

Table 1 Table of prioritized risk

|  |  |
| --- | --- |
| Priority | Risk Identifier and Description |
| 1 | Risk 1: Games made with Unity have a lower threshold for cracking, and users are more likely to develop plug-ins to cheat. |
| 2 | Risk 2: If the developer changes the Unity versions during development, the project may not continue to be developed normally due to incompatibility between different versions. |
| 3 | Risk 3: If the player use an old device to play, the CPU/GPU performance may not allow the game to play normally. |
| 4 | Risk 4: When the game is played on devices in different sizes, the window of the may not be displayed normally due to the different aspect ratios of the screens, or the images may become blur. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Probability** | High | Risk 4 |  |  |
| Medium |  | Risk 3 |  |
| Low |  | Risk 1 | Risk 2 |
|  |  | Low | Medium | High |
|  |  | **Impact** | | |

Figure 2 Probability impact matrix before proposed solution

## Summary

This report is organized as follows: Chapter 2 introduces the background and related works of my work. Chapter 3 presents the design and game functions. Chapter 4 shows the future work for the project, and Chapter 5 concludes the content of this report.

# Background and Related Work

This chapter contains two sections, providing the description of the background and related work of our project. Indie game’s introduction, and the prototype for the project’s kind of game’s introduction are in the *background* section, and the using developing techniques are discussed in the *related work* section.

## Background

This project aims to be a lightweight indie 2D game. Generally, ‘Indie game’ symbolizes originality and forward-thinking, especially in music and design. For the developers, An indie game’s developer is any business, developer or designer that is not associated with a large corporation, especially a global one[1]. Due to this different quality from games made by large companies, indie games have always been loved and recommended by certain players.

In recent years, more and more people don't want to be constrained by all the restrictions that the big game companies place on creating titles for profit, and want to make their own games. Although they may not necessarily have the same beautiful modelling and scenery as the big game companies’ work, indie games still have a loyal following due to their creative freedom and the variety of types and gameplay. In addition, many indie game creators do not consider developing games as work, but as a hobby of sorts, in their spare time. This allows them to remain inspired and passionate about their work.

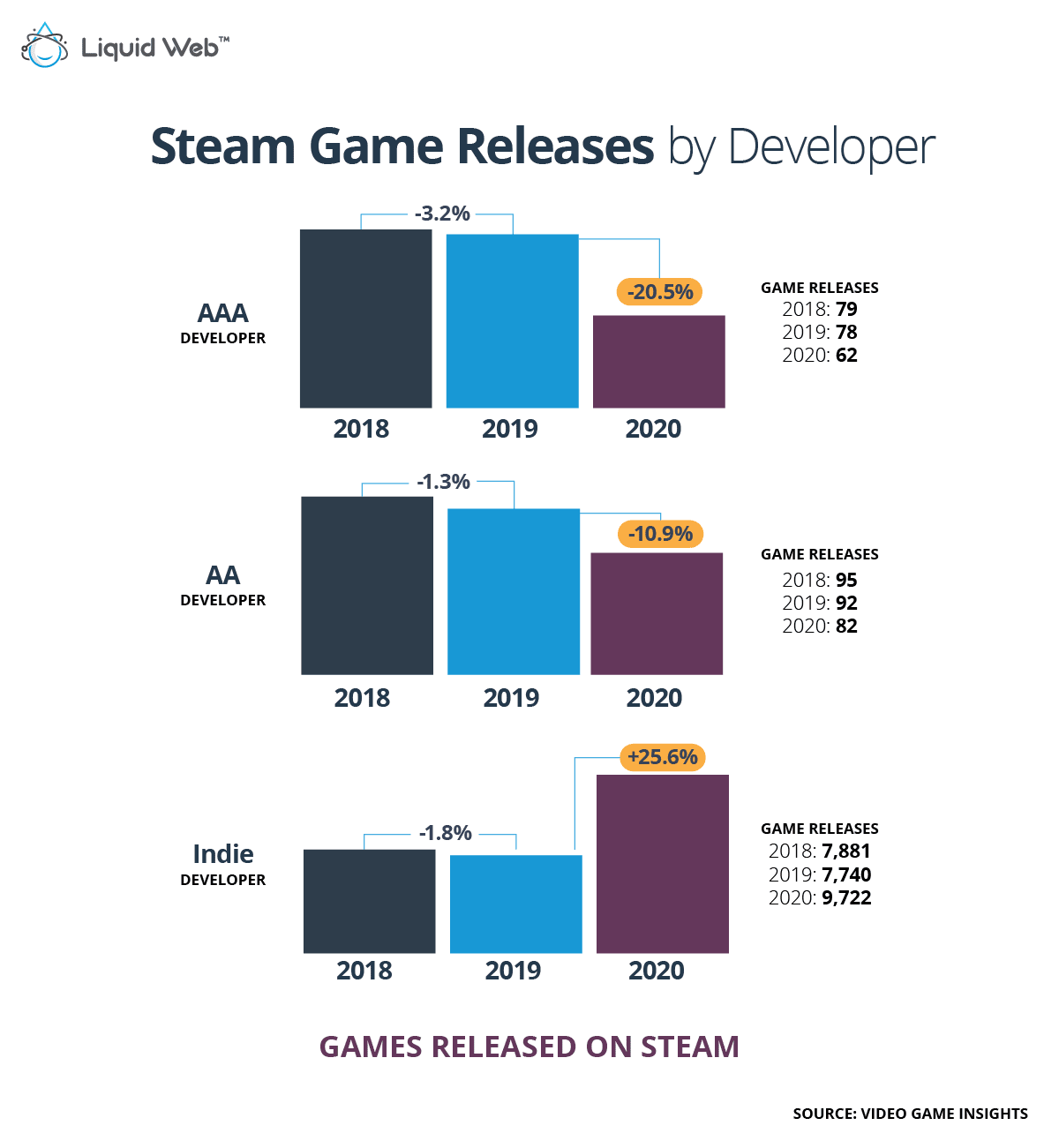


Figure 3 Indie developers can escape the adverse effects caused by the pandemic[2]

As a prototype for the project, *Vampire Survivors* is a time survival game with minimalistic gameplay and rogue-like elements[3]. In this type of game, almost all attacks are performed automatically by the system after a specific time interval. The player does not need to do anything to trigger an attack. In fact, there are only two actions that the player needs to perform in a game of limited duration: to navigate through the thousands of monsters moving towards the player to avoid being hit, and to decide which weapons to increase or upgrade when picking up the drops used to increase experience and upgrade to become more powerful in order to deal with more powerful monsters.



Figure 4 A screenshot for Vampire Survivors

The gameplay of this type of game can be summarised as follows:

* Player can move freely in a very large game map.
* There will be a constant flow of enemies moving in the direction of the player. Once touching an enemy, the player's character will lose HP.
* Players start the game with at least one weapon, which has a fixed attack frequency, and they need to control the weapon by moving their character so that it can hit the enemy.
* After kill an enemy, the enemy drops pick-up items. Players can pick them up by approaching them. Picking up such drops will increase the player's experience value and when it reaches a certain value, the player will automatically upgrade, which can make the player stronger.
* As players move around the map, they will find treasure chests. A attack of any type will open the treasure chest. The treasure chests contain items that help the player, such as restoring HP or gaining shields.
* A timer will be displayed in the game screen to allow the player to see how long a game has been played. When the timer reaches the required length of time and the player is still alive, the game is considered passed.

## Related Work

On the technical aspect, this project will be built with the Unity game engine. Unity is a cross-platform game engine initially released by Unity Technologies, in 2005, using C# to do the script. The focus of Unity lies in the development of both 2D and 3D games and interactive content[4]. In addition, Unity has very convenient packaging built-in exporting tools, which can easily deploy the completed game project on a variety of operating systems. The games made with the Unity engine make up for 50% of all mobile games. Every day, there are 15,000 new projects made within the platform. Unity engine has become a development environment for such extremely popular games as *Pokemon Go*, *Rust*, *Pillars Of Eternity*, and *Escape from Tarkov*[5].

Unity uses C# for users to write scripts. As a statically typed language, C# is easy to read and understand, making it easier to find errors in code and also understand the code. Another advantage for C# is that it is an Object-Oriented Programming language. This makes it highly efficient, flexible, scalable, and easy to maintain[6].

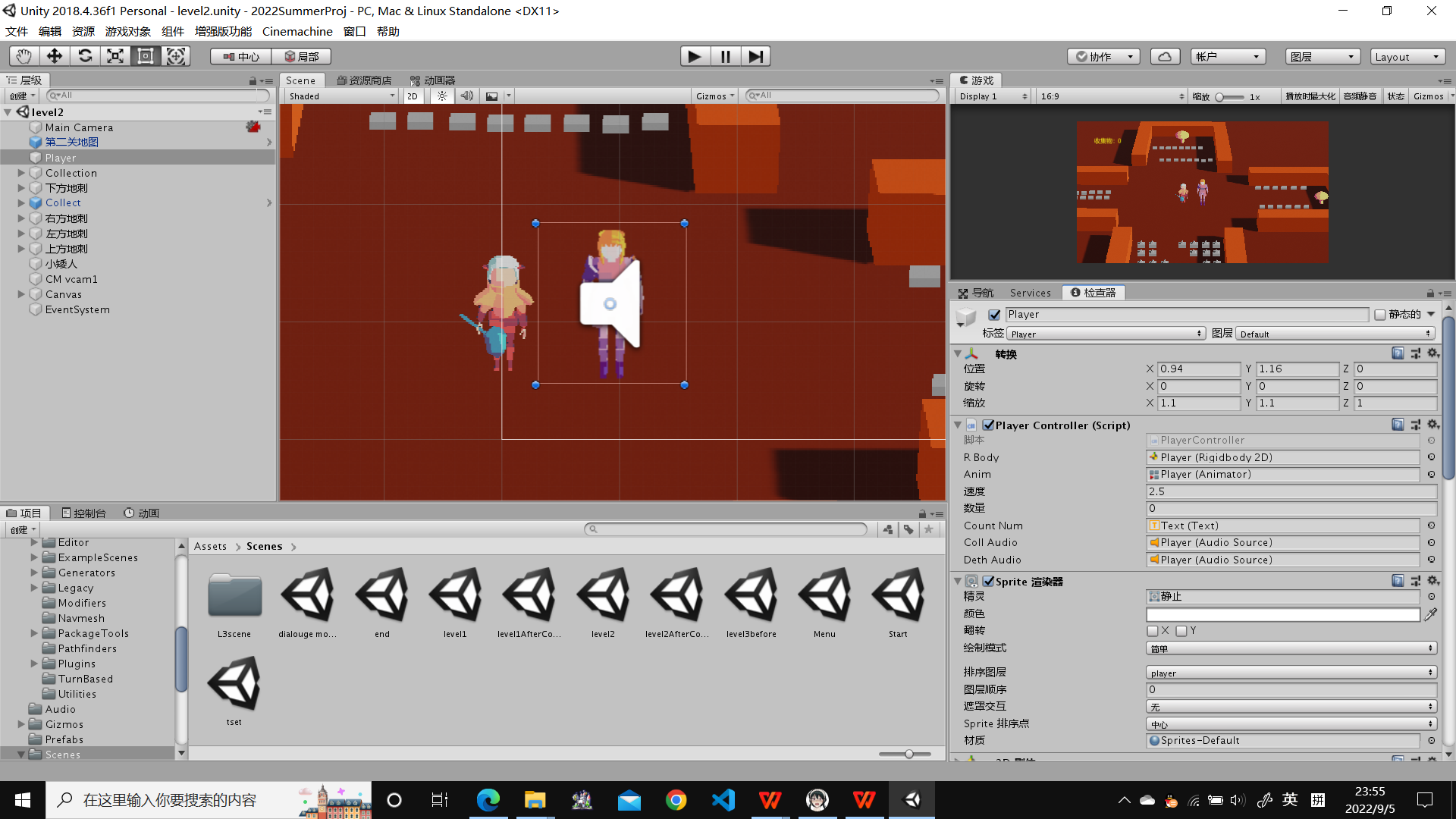


Figure 5 The development interface of Unity

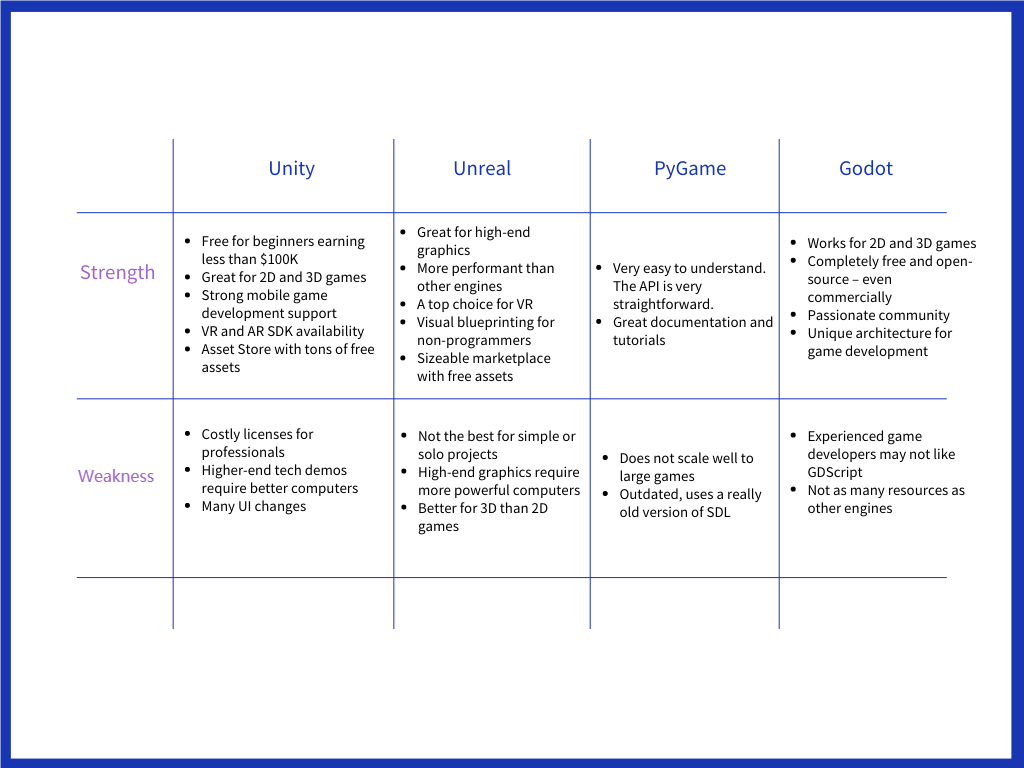
Also, there are other different engines for game developing. We will do the comparison in the next paragraphs.

Such as Unreal, which is a famous and handy engine in development, too. But eventually, the Unity engine is chosen. Why? Firstly, Unity was built with mobile apps in mind, so development for these devices is very streamlined. On the other hand, Unreal is developed for AAA titles and those geared toward high-end devices. As an indie game with limited processing requirements and developer, it is clear that using Unreal is too cumbersome. What’s more, for individual developers and small enterprises, Unity has a special charge plan for them. They are free to use Unity to develop their projects, then publish them on open platforms with no extra fees. However, although the Unreal engine can be used freely, once the developer decides to publish the game as a premium one, then the 5% of earnings for Unreal’s company is necessary. In the long run, the Unity engine with streamlined functions and lower individual cost seems to be a better choice[7].

In addition, PyGame is also a relatively common game engine. PyGame is developed using the python language, which is an encapsulation of SDL, a popular game development library many years ago. However, the embarrassing point is compared with Unity, Python does not have an graphics lib or physics engine that can be used for game development. This leads to a problem, using PyGame will consume more time, energy and technical learning costs in the case of making the same style of game which made with Unity. Considering the time and cost of developing the project, Unity is more likely to be chosen.

What about Godot? Godot is an open-source game engine especially famous among beginners. It was released in 2014, and it is now the 4th most popular game engine[5]. Although Godot now has a growing number of users and good development prospects, as a game engine released in 2014, the content of Godot's access store is not very complete. For the same reason, its community is still under construction, which means that during the development of the project, developers may face some questions that have never been solved in the community. These factors will drag the progress of development. The developers will not meet these problems while using Unity, as its access store and community are under running for ten years longer by now.

Table 2 Comparison between Game Engines[8][9]



Since this project is a lightweight 2D game, it does not need those much complex features for 3D even VR games. Also, the project does not expect very high performance for the computer, so the weakness of Unity can be ignored. The most important is, as a learner for developing game, various assets and tutorials will be really helpful for the project. So in summary, using Unity seems the best choice for developing this project.

# Completed Work

This section focuses on the parts of the project that have been completed. As the type of project is a game, this section will present game functions as units. The various functions are structured as follows:

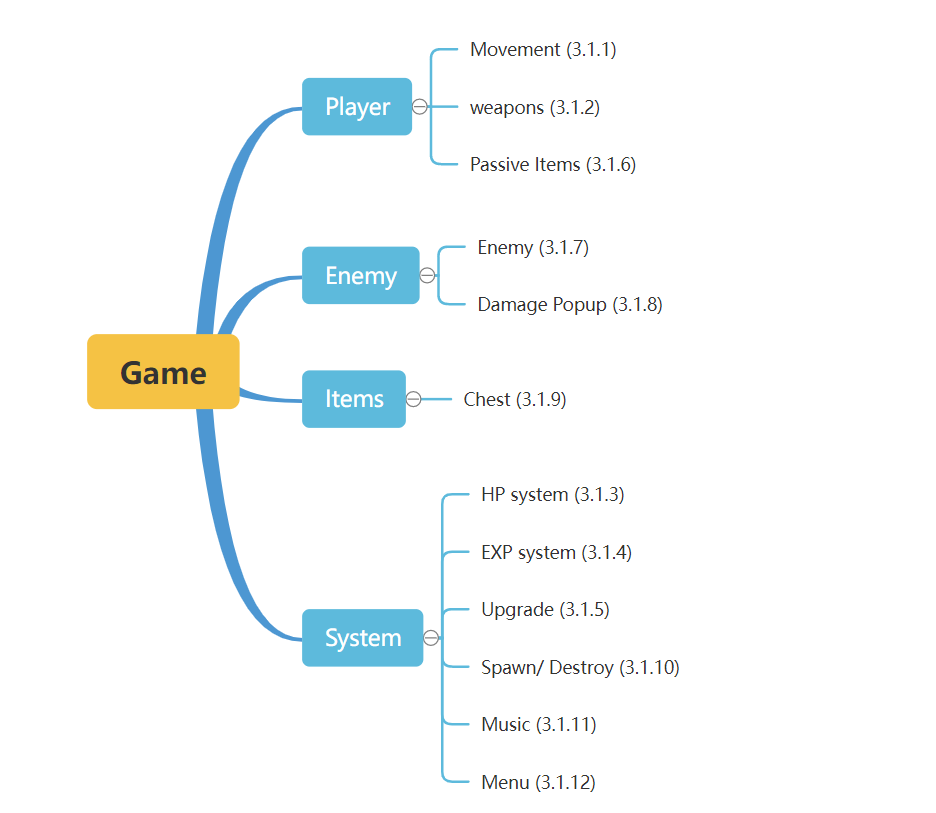


Figure 6 The logic structure for this program

Next parts will introduce these functions one by one.

## Project Outcome

### **Character Movement**

In the game interface, when the player presses the arrow keys, or "WASD", the character will move in the direction corresponding to the player's keys.

In the coding implementation, the method responsible for controlling the character's movement is set to get the value (-1, 0 or 1) entered by the user in the horizontal or vertical direction by pressing the arrow keys. This is then used to make the character move in the corresponding direction. This method runs continuously within the game as the game refreshes (in this case once every 0.04 seconds), thus ensuring that the character can constantly moving. At the same time, the character has different animations when it is walking and when it is idling.



Figure 7 Part of the script for player movement



Figure 8 Different animation when the character idling and moving

### **Weapons**

Weapons are used to fight and kill the enemy. In this project, a weapon can be seen as a game object with collision boxes and animation effects. When a weapon's collision box comes into contact with an enemy’s, it is treated as if it hits the enemy.

* Weapon Manager

To facilitate the management of the various weapon types and better reuse the code and to set different values for the weapons, it is necessary to create a weapon manager. First, we set a uniform standard for the basic data of all weapons. Set the type of parameters in this script file.

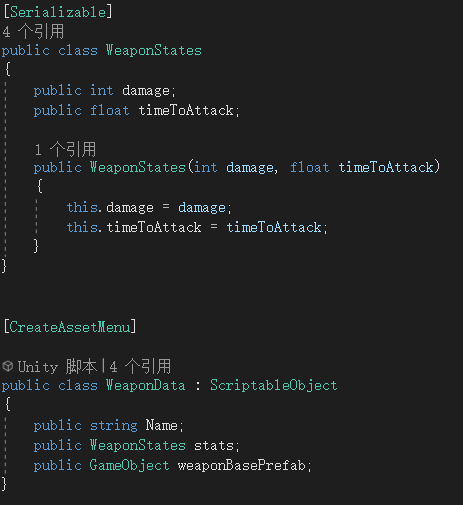


Figure 9 Part of the script for formulate weapon data type

Then, in the Weapon Manager script file, set the starting weapon for the character, which will be set as the first weapon for the character as the game start. The “AddWeapon” function is used to instantiate a weapon when it is needed. Then the new weapon will be added to the character. The specific data of each type is set in the Unity front-end interface, which is convenient for modulation.

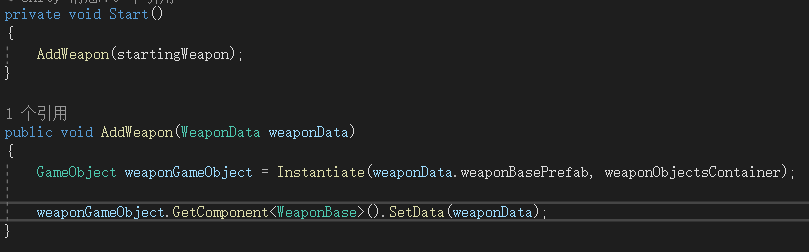
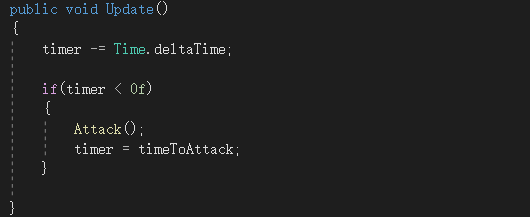


Figure 10 Part of the script for weapon manager

* Close-range Weapons

For close-range weapons, firstly, the weapon object needs to be bound with the character in the front-end of the Unity, which is then controlled by a timer to appear or disappear in the back-end (if the object appears at the moment its collider overlaps with the enemy’s, it is considered to have attacked the enemy).



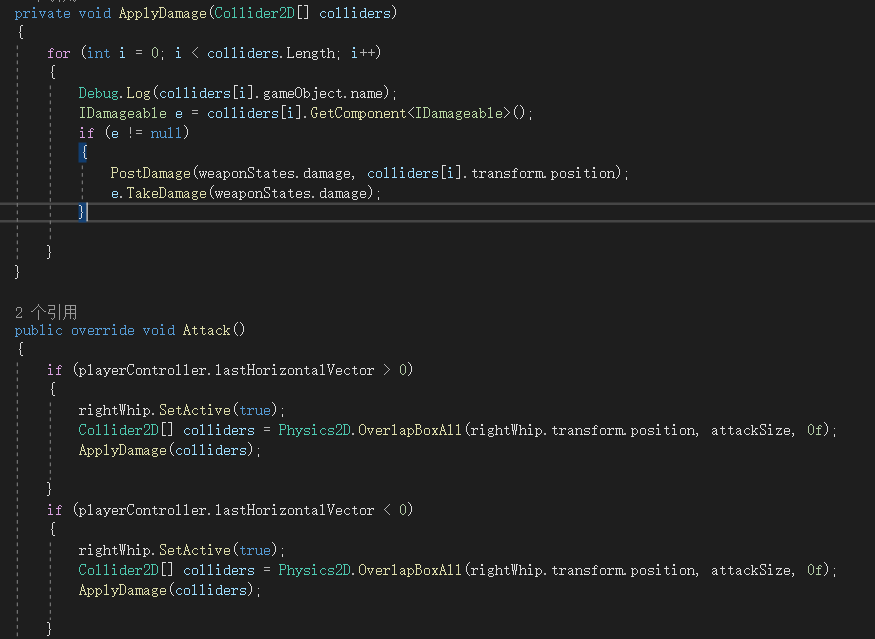


Figure 11 12 Part of the script for close-range weapon attack

The first part of the code set a timer to ensure the system will activate the weapon object after a certain time. Meanwhile the code in the second image controls the weapon object to make damage to enemies if they collide together, as well as changing the scale of the activated weapon object depending on the character's face scale.

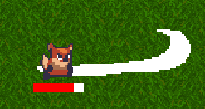
 

Figure 13 The effect for close-range weapon

* Projectile Weapons

Unlike close-range weapons, although projectile weapons also need to be generated in a direction and position determined by the player's direction and position, they have their own trajectory after generation and therefore require different processing logic.

As the code below, a projectile weapon is generated at fixed intervals. After it has been generated, it will move in a straight line until it hits an enemy, otherwise it will move until it has reached the maximum distance and then disappear (will be introduced in 3.1.10). When it hits an enemy, the projectile weapon makes damage to the enemy and destroys itself.

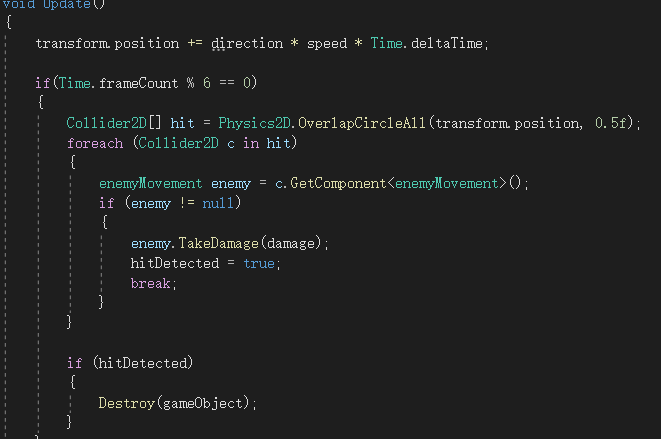


Figure 14 Part of the script for projectile weapon

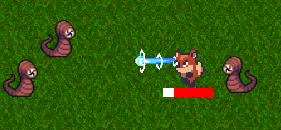
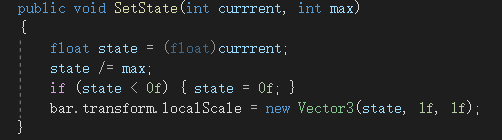


Figure 15 the effect for projectile weapon

### **Health Point (HP) System**

HP is a value used to keep track on the player's life value.

In this project, the HP bar is a red bar that follows the character's feet. When the character is attacked, the HP is gradually reduced. This behaviour is represented on screen as the red part of the bar gradually being replaced by white. When the player picks up an item that can heal, the HP is restored a certain value.



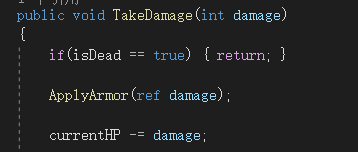


Figure 16 17 Part of the script for character’s HP system

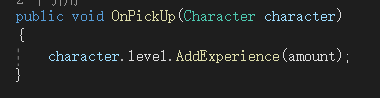


Figure 18 The effect for HP bar

### **Experience and Level Up**

When players defeat enemies, and when they pick up gems (drops from monsters after they die), they increase their experience value. When the experience value reaches a certain value, the player will automatically upgrade.

The visual progress of experience value is displayed at the top of the entire game screen, similar to the principle of the HP bar. The experience value for each specific action increase is set in Unity's front-end interface. The current formula for the amount of experience needed to upgrade is set the current level \* 1000.



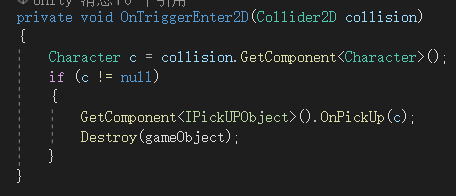


Figure 19 20 Part of the script for EXP system

JCC]U(M$@CJG$TCH9O1%5YB

OB[X%I@THMWRQ2~4]J%~A1M

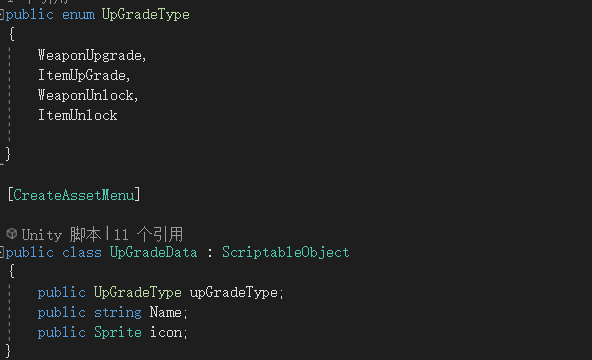
Figure 21 22 The effect for EXP bar

### **Upgrading**

To allow players to feel the joy of getting stronger over the game, in this project, a list will pop up for players to choose something that will enhance the character when the character upgrades. (This feature is still under development, this section focuses on the completed part, the unfinished parts will be covered in chapter 4).

All the options in the upgrade list are created in advance, and placed in a pool of options. At each upgrade, a certain number of objects (currently three) are randomly selected from the pool for the player to choose from. When the number of remaining game objects < the number of objects displayed in the list, the number of objects displayed = the number of objects remaining. After the user has made a selection, the selected object is removed from the pool to avoid duplication.

To do these, first we need to specify the data format for every option and then create a pool of objects.



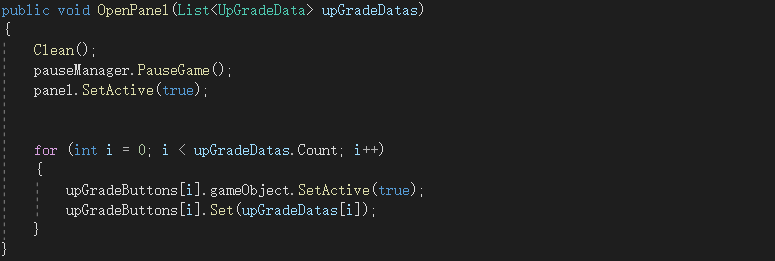


Figure 23 24 Part of the script for formulate upgrade data type and upgrade option pool

After the user has made a selection, the selected game object is bound to the character, then the object is cleared from the object pool, and the upgrade panel is put away while play continues.

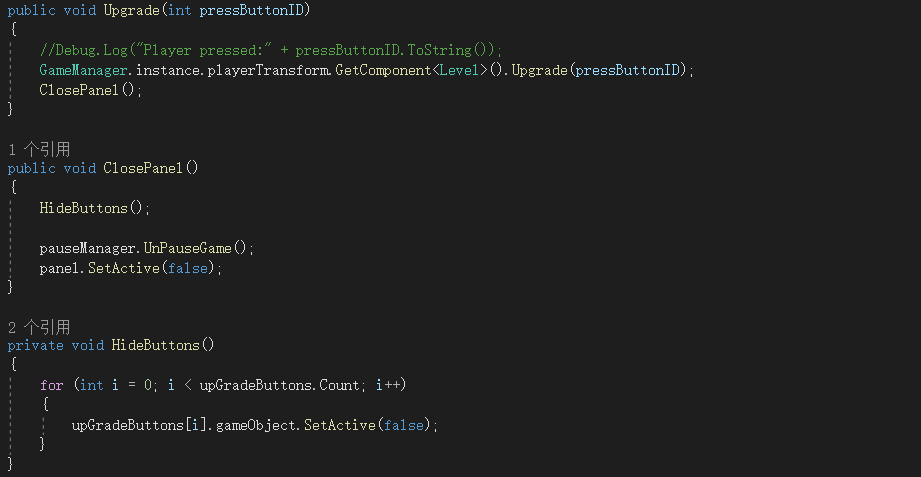
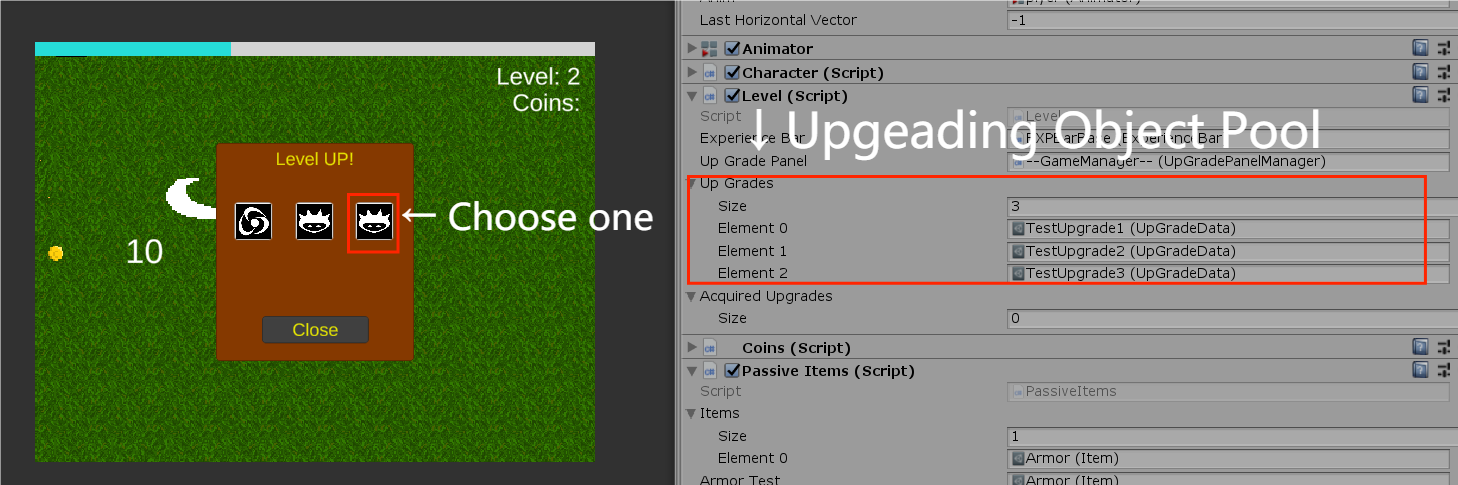


Figure 25 Part of the script for upgrading system

In the interface of the modulating, it can be seen that after the player has made a choice, the selected object that was in the pool of objects moves to the character’s.



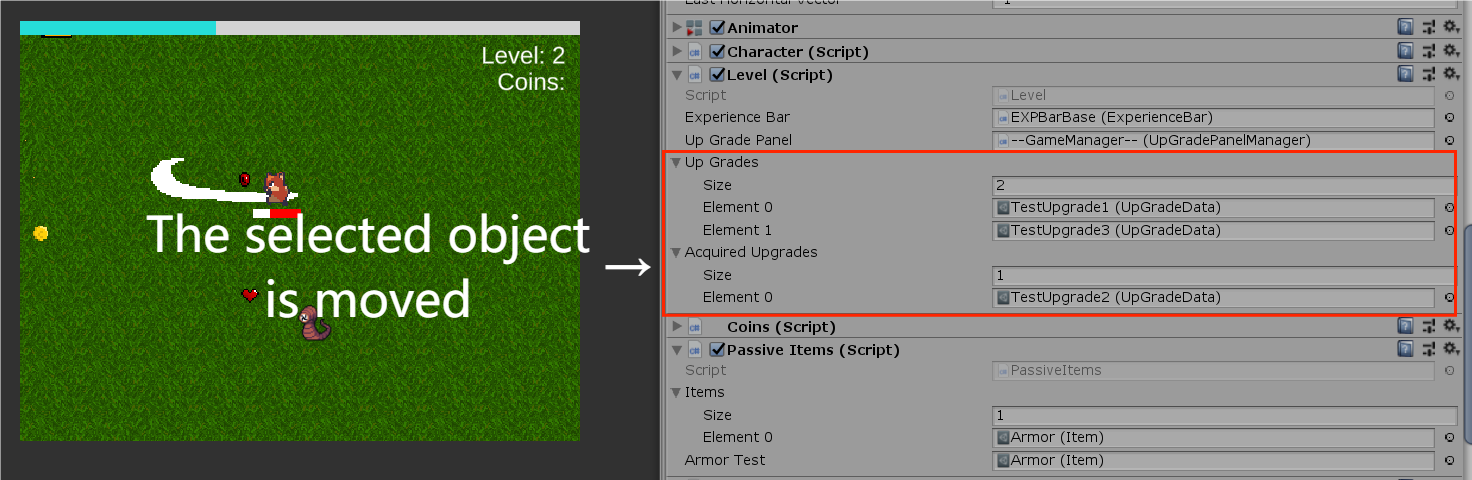


Figure 26 27 Upgrading options in modulation interface

### **Passive Items**

Passive items are items that do not require any action by the player and are constantly in effect once they have been added to a game (not exactly the same as weapons, passive items also do not require the player to control the position and direction of the action to ensure that they are effective or not). Passive items do not necessarily have a visual effect, and the game screen is not necessarily altered by the addition of a passive item outside of the fact that it works under certain conditions.

In this project, only armor is currently added as a passive item and it will be used as an example.

Firstly, create a scriptable object (a type of data container in unity, used for storing data which will be used quite often while reducing memory usage) for the item. Preliminary writing of the actions performed when the item is equipped and unequipped.

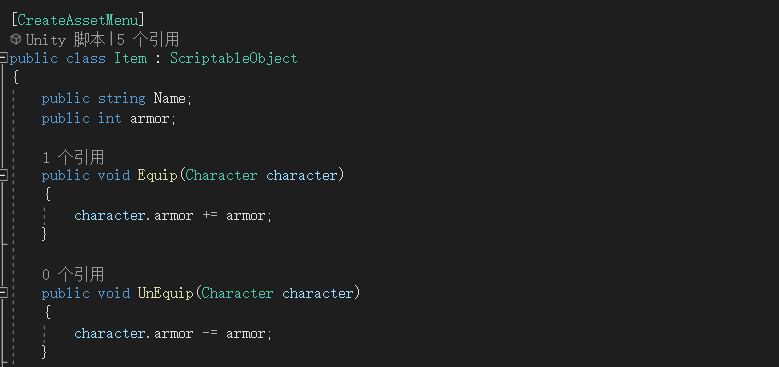


Figure 28 Part of the script for initialize armor’s basic functions

Then write the passive items class, a script that will be tied directly on the game object that needs to use the passive items. It is responsible for getting the item object that needs to be loaded and then applying it to the tied game object.

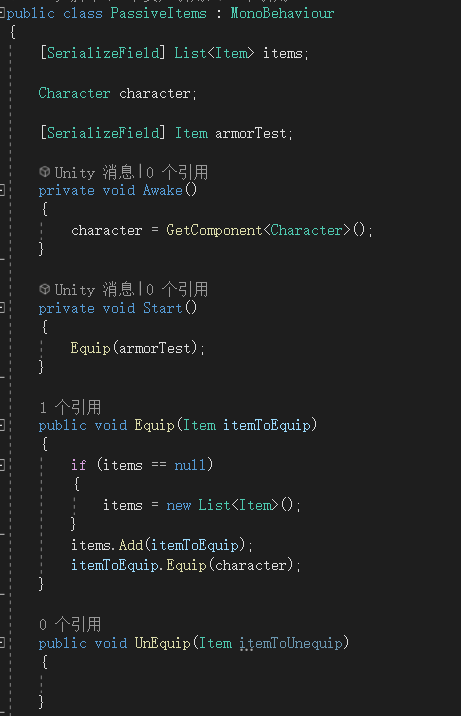


Figure 29 Part of the script for passive item class

For example, setting the value of armor to 1. It will reduce the damage by enemies to characters by one point each time when the game is update while an enemy comes into contact with the character (but the player will not see the "armor" directly on the character).



Figure 30 The player is attacking by an enemy

### **Enemy**

Enemies are an important part of the game. For them, two main things need to be done: setting up the common logic of the enemies (moving, claiming enemies, etc.) and controlling their constant generation and disappearance. This section focuses on the implementation of the first part, the second part of the functional implementation will be covered in 3.1.10.

Control the common logic of the enemies with a single script. This script will constantly acquire the character's position and then make the enemy move towards the character's position and update it with the character's movement. Each enemy object has its own collider, and when it comes into contact with another collider, if that collider has the "character" property (it means that it is the player), the enemy will deal damage to it. Each time the player's weapon deals damage to an enemy, the enemy determines if it has more than zero HP left. If not, it destroys itself and has a chance of dropping a pick-up gem item.



Figure 31 Part of the script for basic function for the enemy

### **Damage Popups**

In order to give the player a visual sense of the strength of the weapon and the HP count of the enemy, it is necessary to display the value of the damage after the character successfully attacks the enemy.

First create a message management script to hold each pop-up message object as a object pool and have it displayed at the location where the script was triggered (the location where the enemy was attacked).

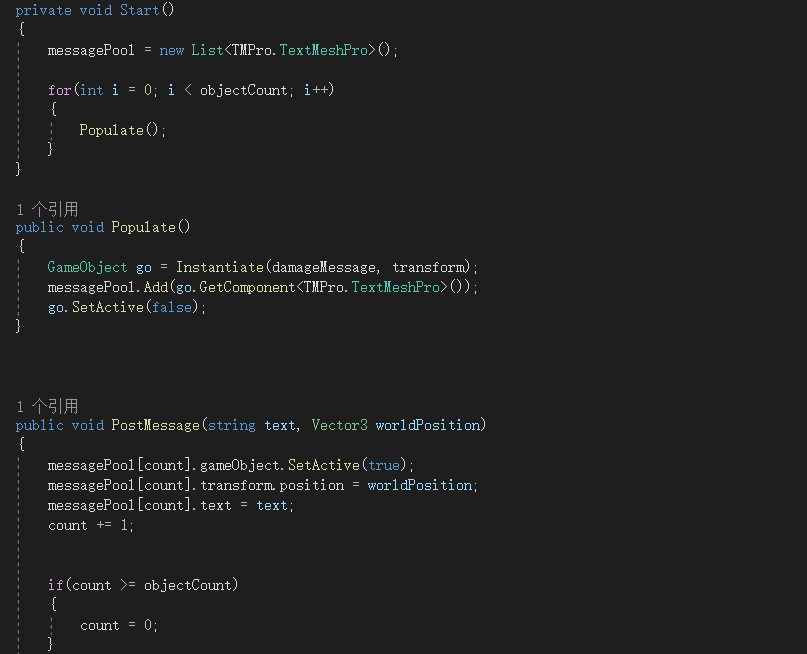


Figure 32 Part of the script for popup message pool

Then set the specific time that each damage message will last. After this time, the message object will be set non-active.

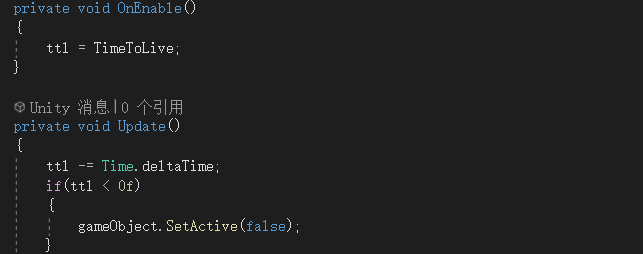


Figure 33 Part of the script for set active or not for each message object



Figure 34 The effect for damage popup

### **Chest**

Opening chests is a great way to get positive effect items in the game. A single attack of any type will open the chest.

When this script tied on a chest object, it is allowed to specify the type of item that will be generated when that object is destroyed.

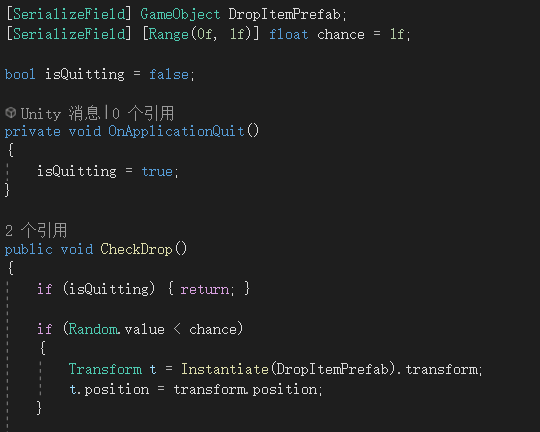


Figure 35 Part of the script for chests’ behaviour

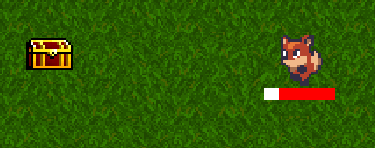


Figure 36 A chest in the game

### **Spawn/Destroy System**

In each game, the player has to run, dodge, kill enemies and pick up items in a large limited map. Therefore, we need to write scripts that constantly generate game objects to keep the game going.

Take the script for generating enemies as an example. Every time when its update, the system will do the generate command to generate a new enemy. The area for generating enemies is centred on the character and has a radius of a certain length (if the enemies are randomly generated on the map, the player will probably not find any enemies for a long time at the start of the game due to the large size of the map). But at the same time, in order not to have enemies spawning directly on the screen that the player sees - which would be too abrupt - there is another circular area with a smaller radius that should be used to set a disallow command on enemy spawning. So, the final area used for enemy generation should be a circular area with the player at the centre.

The logic for generating other pickups and chests is the same, except that they do not have the "speed" property and therefore they do not move.

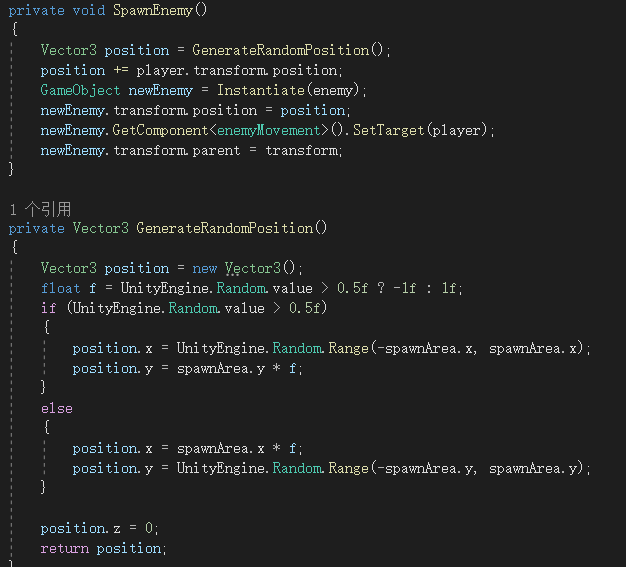


Figure 37 Part of the script for spawning system

After hanging for a period of time with the speed of the enemies adjusted to 0, you can see that the monster will only spawn some distance beyond the game screen (the area in the white box).



Figure 38 The actual range for spawning enemy

However, in order to prevent the quality of the game from being affected by the number of game objects that are constantly generated, we also need to design a game logic that allows game objects that are more than a certain value away from the player's character to be automatically destroyed. So, when the player runs away from the map and comes back, he/she does not have to face both the original enemy and the new enemy behind him.

This script is used to constantly obtain the player's position, determine the distance of the current game object from the player, and destroy itself when conditions are met.

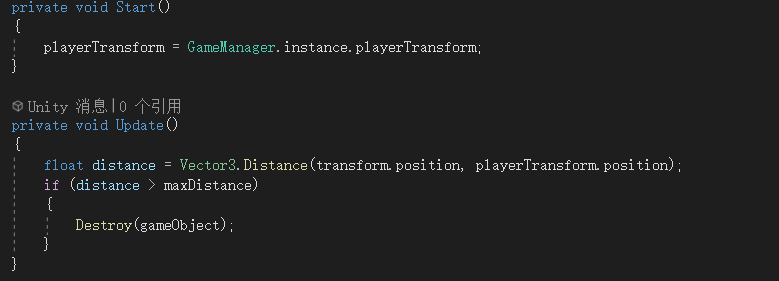


Figure 39 Part of the script for destroying system

### **Music**

Any good game needs to have music to go with it. In unity, the background music management component is often bound to the main camera object, as this is an object that is constantly activated at almost any time and therefore ensures that the background music is also uninterrupted.

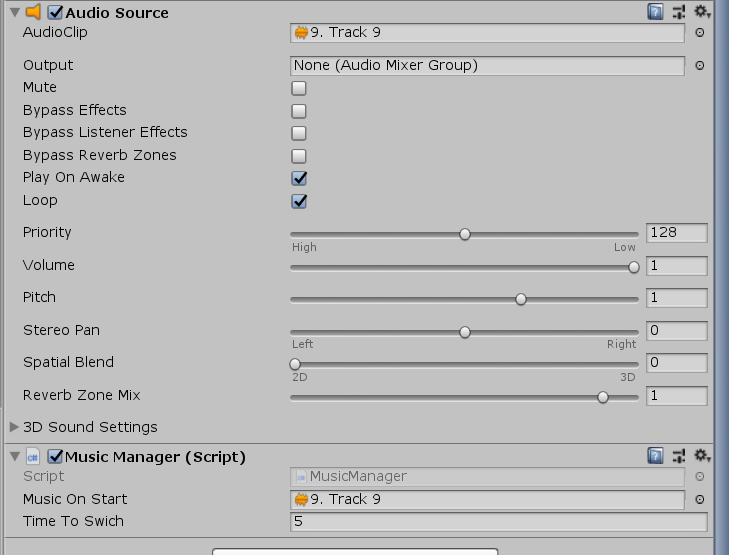


Figure 40 The music editor in Unity

### **Main Menu, Pause and Losing Game**

This section is used to introduce some of the basic control operations of the game.

Create another game scene for the main menu. Now two different game scenes are loaded as two game levels. Click any one of the button to enter the game.



Figure 41 The game’s start menu

For pausing game, set the esc button to pause, then set the time scale become 0.



Figure 42 Script for pausing the game

In this project, losing game’s condition is the player character’s HP becomes 0. Then the panel will popup with a button to back to menu.

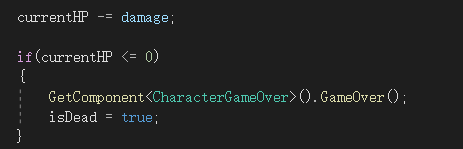


Figure 43 The condition for losing the game



Figure 44 Game over panel in the game

# On-going and Future Work

This section is used to introduce the work that will be carried out after this report. The first section is used to introduce some of the features that have not yet been completed, the second section introduces the new features that will be added and the third section provides a timeline for the project in the second semester.

## Partially Done Work

### **Shop and Collecting Coins**

Currently, collectable coin objects have been added to the game.

In the future, the project plans to allow the number of coins collected by the player to be inherited over multiple games, and to add a shop screen to the main menu so that players can use the coins collected to buy something to make them stronger in the game before starting the game.

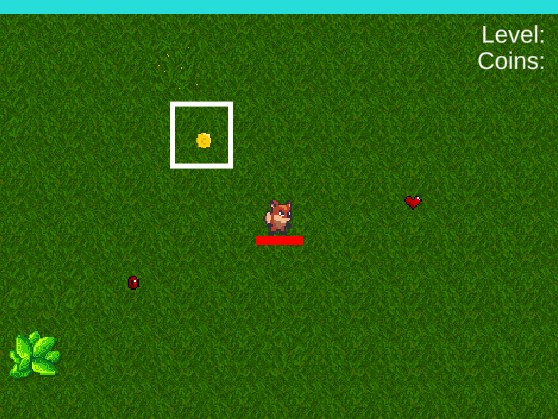


Figure 45 A coin in the game

### **Upgrade System**

A pool of upgrade options has now been added to the game for players to select and then add the chosen object to their character. The next step in the project is to create more upgrade options, and add actual features to each option so that characters can use them once the chosen object has been added (e.g. greater attack range, new types of weapons, etc.).

## Functions to be Added

This list shows the features planned to be added to the project in the next semester:

* Multiple attacks
* More enemies
* Shop System
* ……

## Work for the 2nd semester

This gantt chart shows the next steps of this program. The whole project should be done before 04/16/2023.

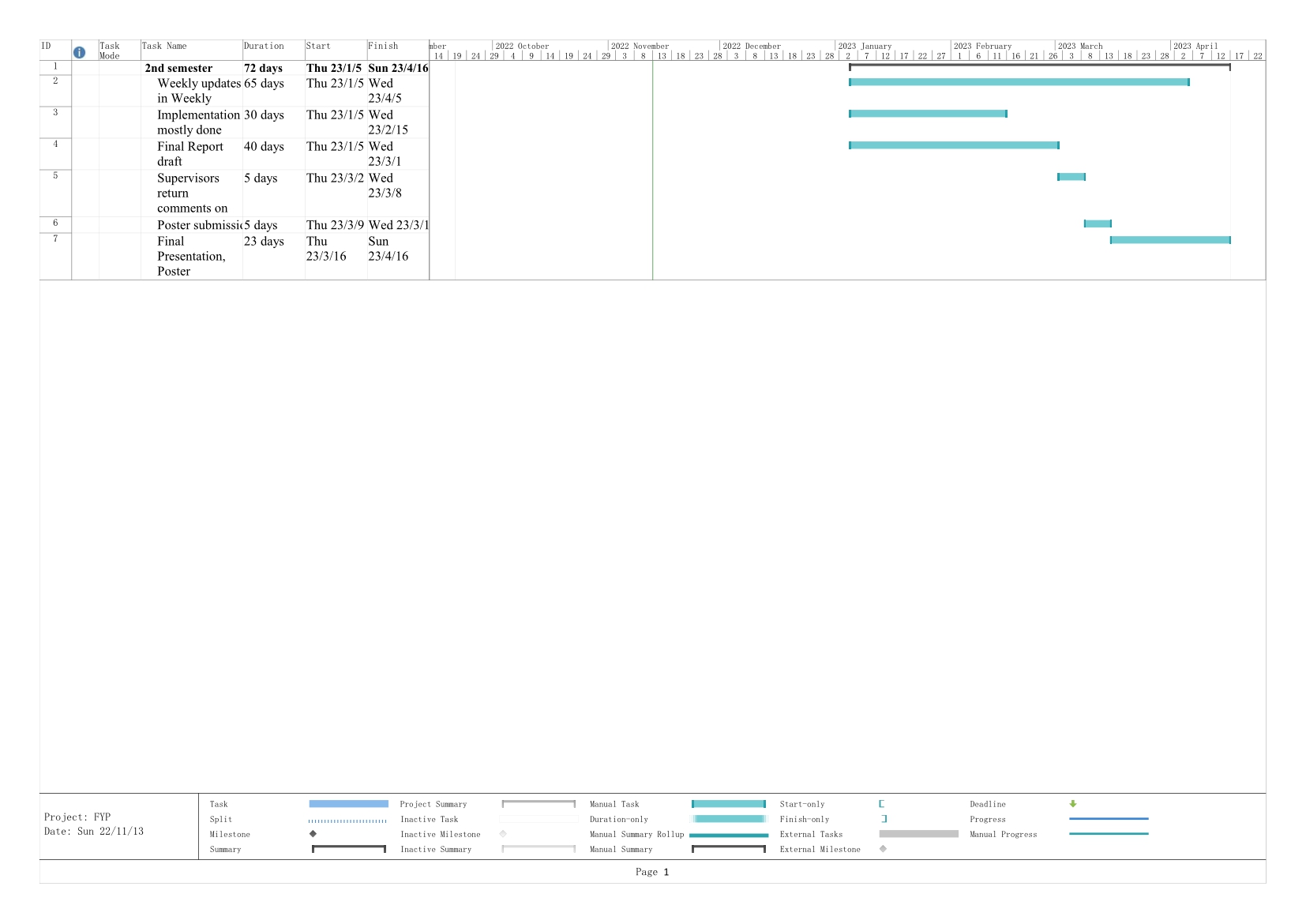


Figure 46 The gantt chart for the second semester

# Conclusion

This report begins by describing the difficulties that many working and university students have in playing the game and explains why this project is a lightweight 2D game. Then it is followed by a description of what makes indie games special and an analysis of the best tools to develop this project. In the third chapter, this report focuses on the various features that have been developed in this project so far with core code and actual results. Finally, the fourth chapter of the report maps out the second half of the project and the development timeline.

References

1. Kovanto, A. (2013). The Improvements for Indie Game Development.
2. “A world at play: The current State of the video game industry,” Liquid Web. [Online]. Available: https://www.liquidweb.com/insights/video-game-statistics/
3. “Vampire Survivors on steam,” Steampowered.com. [Online]. Available: https://store.steampowered.com/app/1794680/Vampire\_Survivors/
4. Unity - Introduction. (n.d.). Tutorialspoint.com. Retrieved September 3, 2022, from https://www.tutorialspoint.com/unity/unity\_introduction.html
5. Godot vs Unity: Detailed comparison in 2022. (2022, July 15). Whimsy Games. https://whimsygames.co/blog/godot-vs-unity-all-you-need-to-know/
6. Payne, J. (2021, May 5). Benefits of C#. CodeGuru. https://www.codeguru.com/csharp/benefits-of-c/
7. Omelchenko, S. (2021, April 13). Unity vs. Unreal: Choosing the best engine for 2022. Program-Ace. https://program-ace.com/blog/unity-vs-unreal/
8. L. Schardon, “Best game engines for 2023 – which should you use?,” GameDev Academy, 03-Oct-2022. [Online]. Available: https://gamedevacademy.org/best-game-engines/
9. “8 Best game engines for Python as of 2022,” Slant.co. [Online]. Available: https://www.slant.co/topics/10908/~game-engines-for-python