**Technical Design Document**

Name (student): Eric Li

Student number: 456873

Class: CMGT 2G

Teacher: Armando Gerard

School: Hanze University of Applied Sciences

City and study year: Groningen, 2023/2024

Submission Date: 22/3/2024

# **Table of Content**

1 Introduction

* 1.1 Case Choice
* 1.2 Self-Assessment
* 1.3 Personal Learning Goals

2 Research

* 2.1 Game Features Review
* 2.2 Requirement List

3 Technical Design

* 3.1 Description
* 3.2 UML charts
* 3.3 HUD Design & Editor Tool Design

4 User Guide

* 4.1 How to Use This?

5 Iterations

* 5.1 UML Charts
* 5.2 C# Functions

6 Reflection

* 6.1 My Learning Process

7 Reference

* 7.1 Used Source & Tutorials

# **Introduction**

Case Choice

Heavily inspired by Vampire Survivor, I chose the leveling system to develop in the Micro FPS project in Unity. I like Vampire Survivor’s approach to a rogue-like level-up system which extremely increases the replayability and progression of this simple gameplay. In my opinion, the leveling system would be a good attempt to create a complete system for me since I hadn’t done such a thing before and it could be also interesting and fun to develop the case with some creative mechanics in an FPS game.

Self-Assessment

As a fresh gameplay programmer, after one year of exploring, I’m now familiar with Unity’s components as well as basic C# in Unity, I’m good at making prototypes rapidly but find it hard to organize and develop complex systems because I don’t have much knowledge about coding in general and it’s always difficult for me to learn new concept without context and examples. Also, I might be too lazy to look for different (better) solutions once I have the basic functions working.

Personal Learning Goals

For this programming focus track, First I wish I could have a wider vision of how different classes and scripts could work with each other and clarify the relationships between them so that I would stop writing all the code in one or two scripts and hopefully have a cleaner and more efficient structure of my code in the game. What’s more, I would take the initiative to ask what I don’t understand about C# in Unity including the terms, functions, used conditions to learn more about programming logic, and also try to go through different solutions to iterate my code.

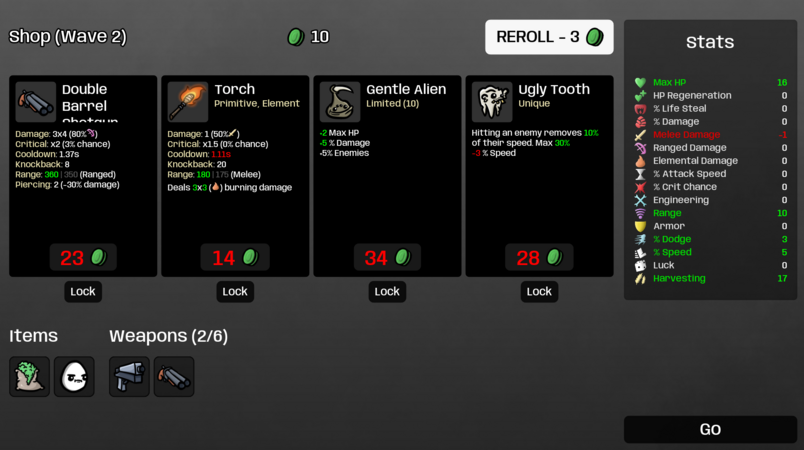
# **Research**

Game Features Review

Tho the combos of items from Vampire Survivor are more complicated than I wanted to implement, I got ideas as a start point to structure the code. The player can collect the experience points dropped by dead enemies, once the experience fill up the bar, the game offers 3 random options to upgrade.



Another game Brotato, a wave-based roguelike game where you play in a fixed-size map to survive, player can upgrade the stats and weapons in the shop in between rounds. Tho it doesn’t have a leveling up feature, this shop system serves as one by providing procedural generated weapons and items (random stats).



While the first two games doesn’t have a system for players to use active abilities, 20 minutes till dawn gave me some insights on how to include a simple ability system which offers more intuitive feedback on how they are preforming.



After a few research on the games that has similar features I want to implement, I had these conclusions to help me integrate them in the Micro FPS project..

The basic leveling system in Vampire Survivors works as:

1. Enemies would drop the gems on death and the player can absorb the gems to gain experience.
2. When the player collects enough experience of the current level, they will gain a level-up, each level-up requires more experience than the previous ones.
3. Upon level-up, the game is paused and the player is given 3 or 4 power-up options to choose from in a UI panel. If they already have the chosen item, the item will be upgraded to the next level.

The role of leveling system in these games:

1. Balancing Difficulty: The game becomes harder and harder as time passes so the leveling system gives players more power to fight against the challenging opponents.
2. Progression and Engagement: Except that it; 's the way the player gets stronger to beat enemies in the game, Either the passive power-ups or active abilities themselves let them experience the learning process of how to kill different types of monsters strategically.
3. Replayability and Player Choice: The options (items) in the level-up panel are totally random, the potential upgrades and combinations of the power-ups/ abilities allow the player the explore different ways to win.

How to interact with it:

1. Move the character close to the experience points on the ground to absorb them and gain experience, the experience bar shows the current experience and how much you still need to reach the maximum experience of this level.
2. When the level-up panel opens with 3 options, the player can click the option to get power-ups and then continue the game.
3. The functions of power-ups are applied to the character from then on and could be upgraded further in this run of the game.

Requirement List

1. An experience manager.
2. A level-up system.
3. Power-up and ability logic.
4. A random loot system allows the enemy and the level-up panel to create random targets based on drop chance.
5. Enemy and Airdrop spawners.
6. Airdrop behavior.

# **Technical Design**

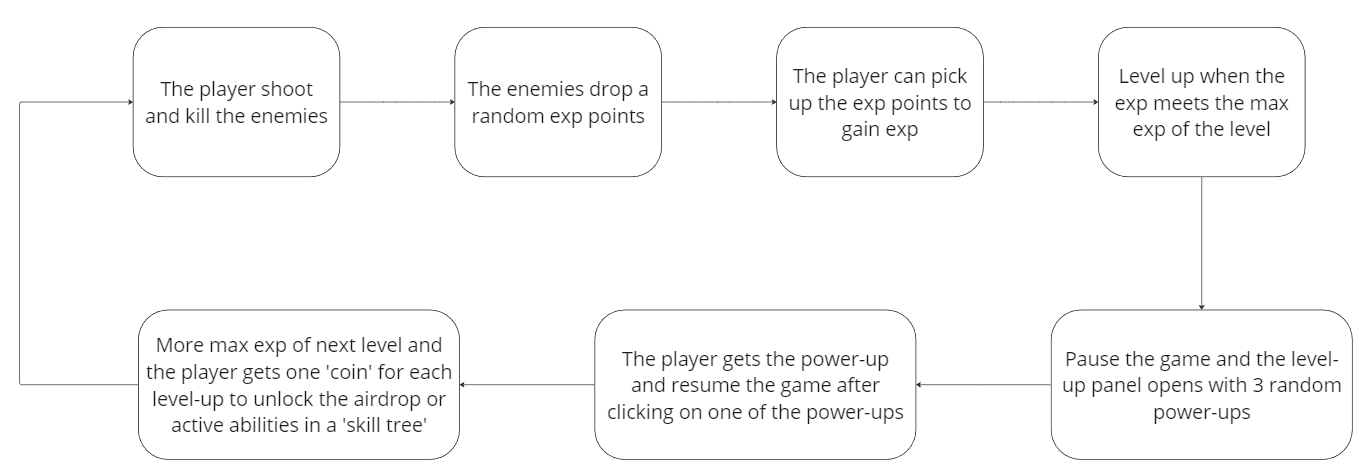
Description

The player is placed on a flat map at the start of the game, the mission is to survive as long as possible by killing all the threats nearby. Enemies would drop different types of experience points randomly on death and the player is able to pick up them on collision, then the player’s experience is increased based on their value.

If the player’s experience reaches the maximum experience of the level, level up the player and show a level-up panel with 3 random power-up options there. After choosing one from the panel (mouse click), the player would get the passive power-up and continue the game with an upgrade (modification related to the character and the weapon). They also get a ‘coin’ when leveling up and can consume it by buying items (only a healing pick-up in the prototype) from airdrops or active abilities from a ‘skill tree’ (3 new abilities).

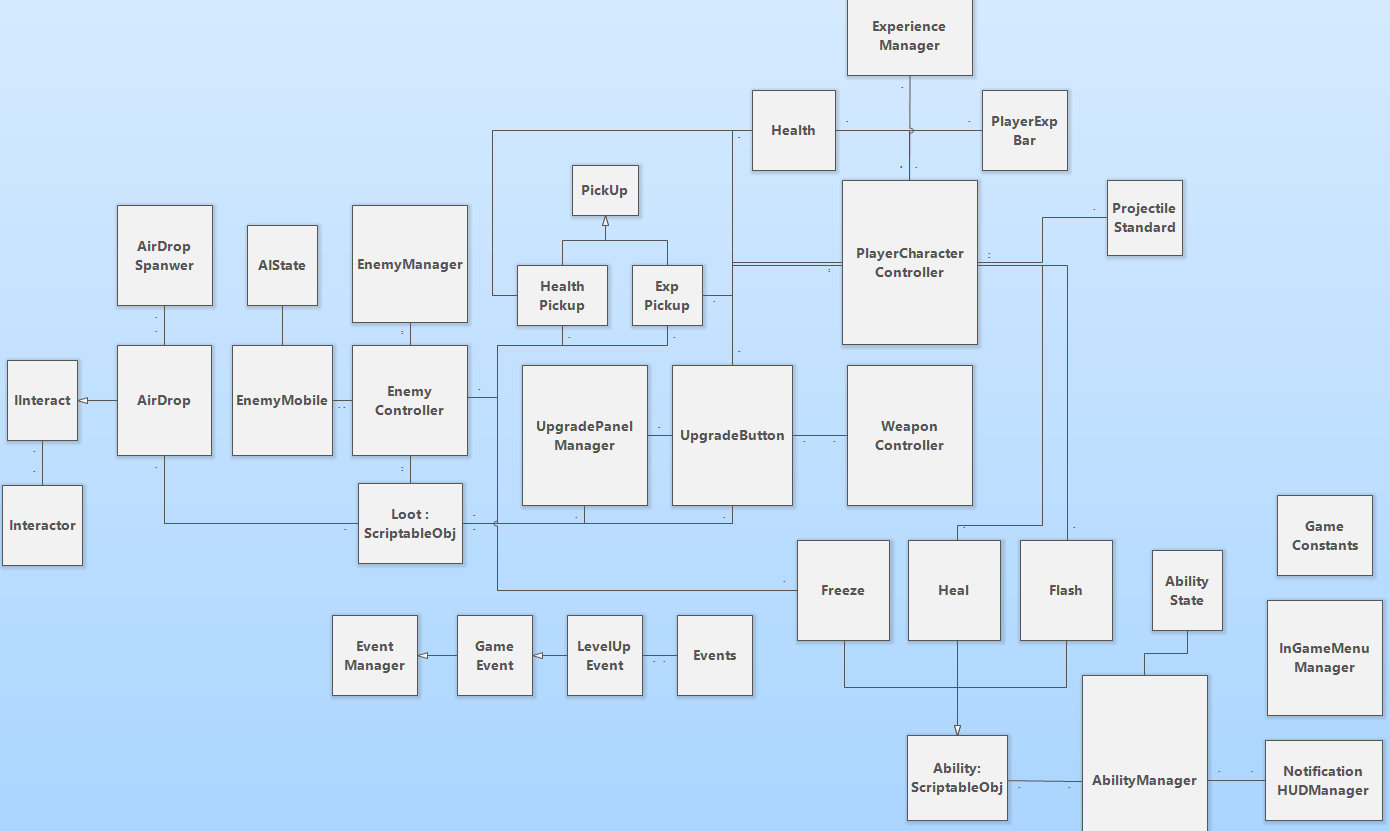
The enemy and airdrop are spawned in a random position over time on the map, while the enemy would be located according to the player’s position (always around the player but can’t be spawned within a radius to avoid a ‘jump scare’ to the player).

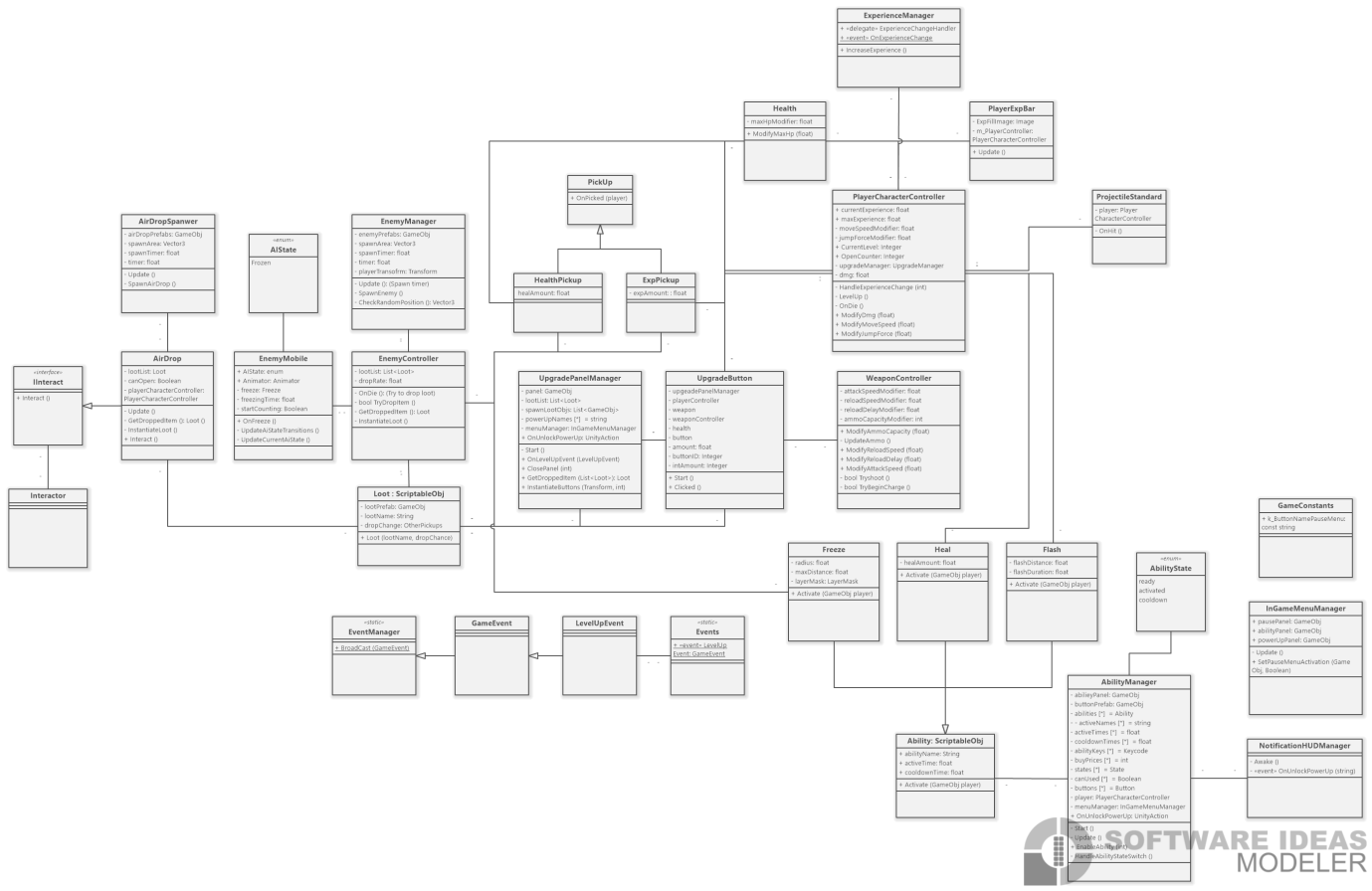
Flow Chats



UML Charts

Overview



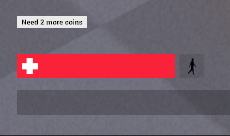
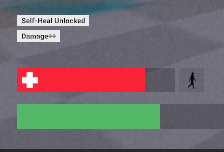


HUD Design

Level-up Panel & Exp Bar:



Notifications:

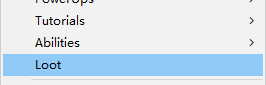
# **User Guide**

How to Use This?

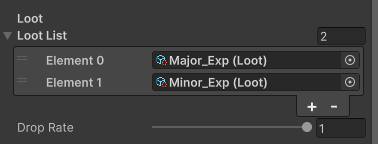
The enemy and airdrop spawner scripts are attached to ‘GameManager’ component in the project and the user can adjust the spawn intervals and also area range in the inspector.



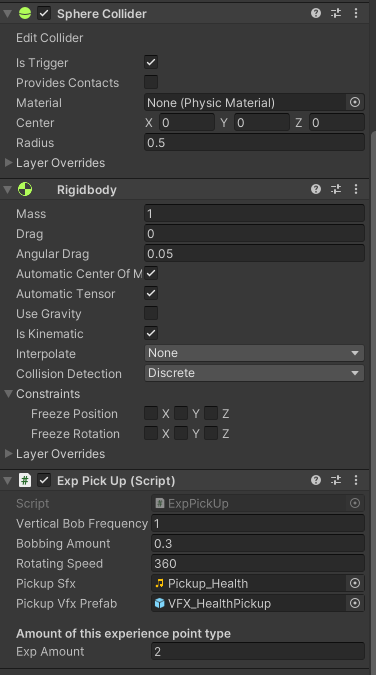
To make the enemy drops the experience points, the user needs to create a loot data from the asset menu (or changing the loot prefab in the existing loot data)



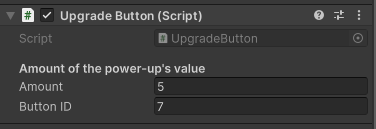
Then in the ‘EnemyController’ attached to enemy prefabs, the user can adjust the list of dropped items as well as drop rate.



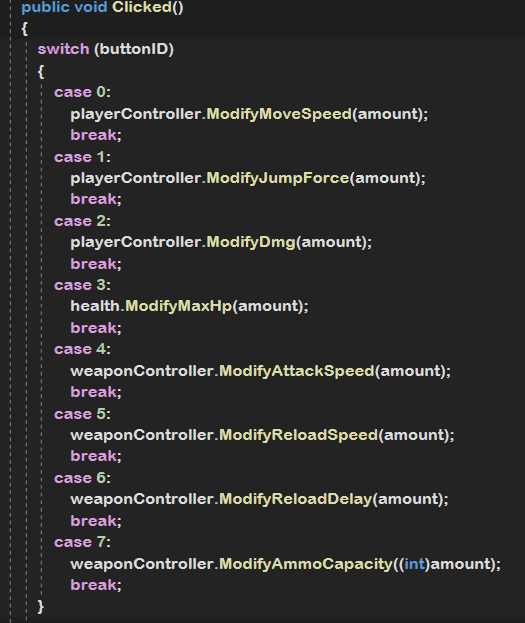
The experience points prefab is created with ‘Collider’ and ‘Rigidbody’ component, the user can adjust the amount of experience in the prefab .

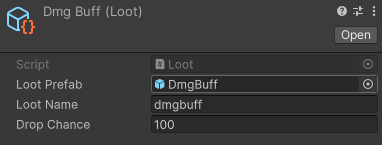


The passive power-up buttons need a UpgradeButton script attached with the amount of modifier and button ID to specify which buff it is.



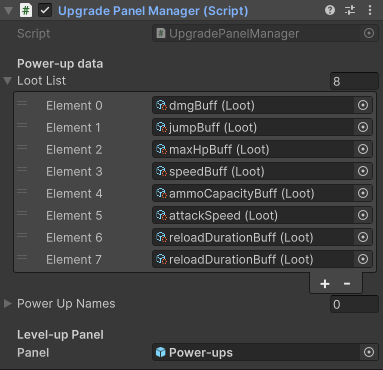
The power-up modifier is added to the target on click.



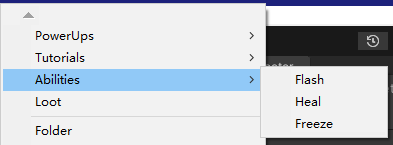


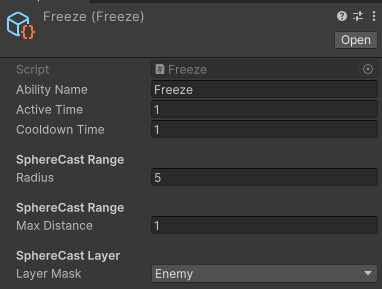
In the asset menu in the project folder (right click), the user can create loot data to contain the power-up button prefab and the drop chance.

The order of power-ups in the loot list of ‘UpgradePanelManager’ which is attached to ‘InGameMenu’ should be the same as button ID to match the correct functions.

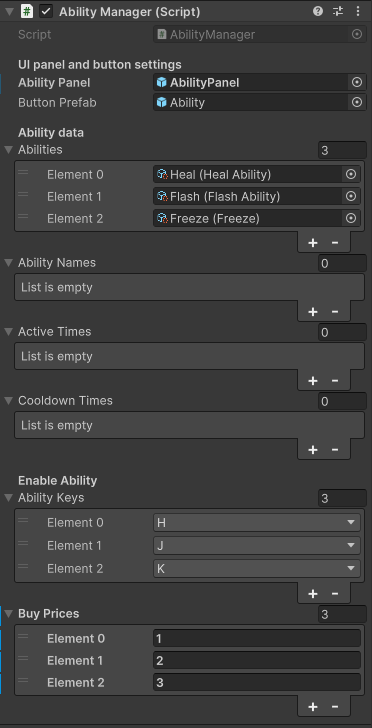


The active ability is created in the asset menu as ability data, the user can change the value of active time, cool down time and so in their needs.

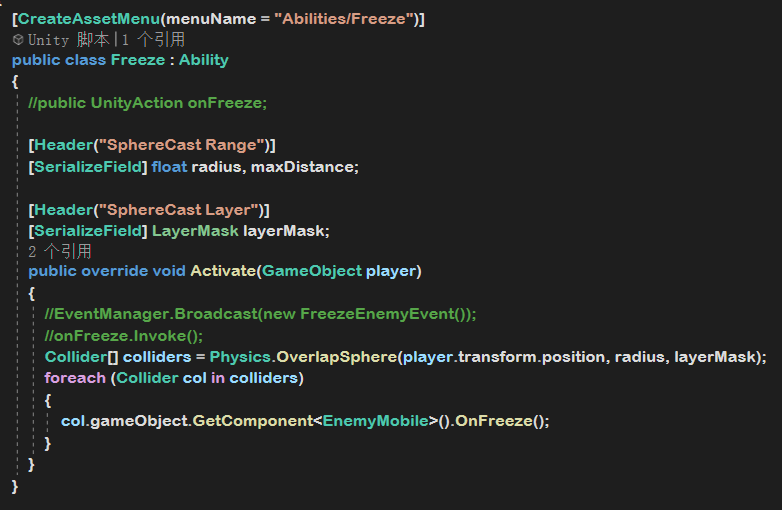




Then the ‘AbilityManager’ attached to the player handles the active keys and prices of each ability.



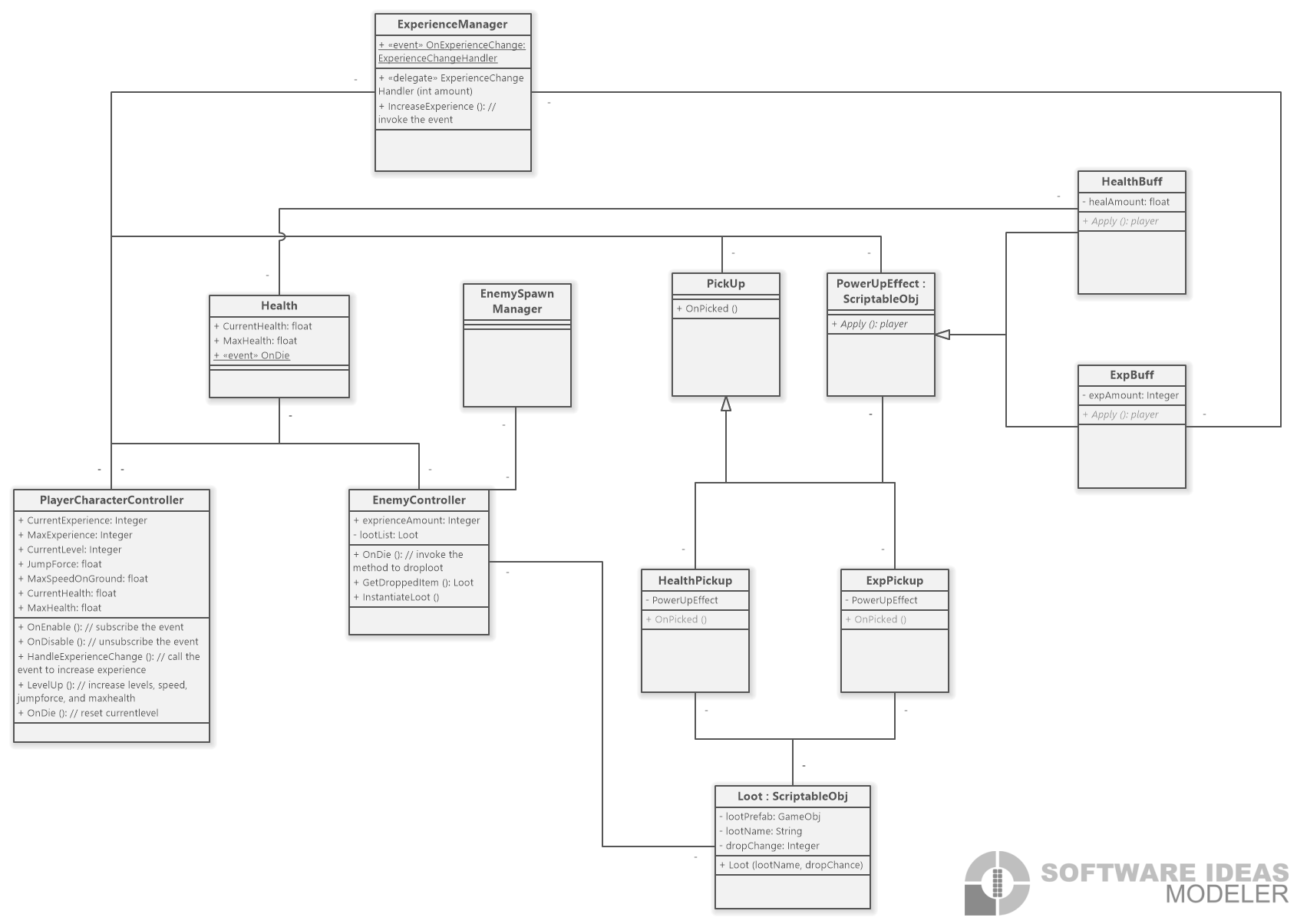
If programmers want to add more ability functions, create a script inheriting Ability for it and add it in the Abilities array of ‘AbilityManager’, write the function in overrride activate method.



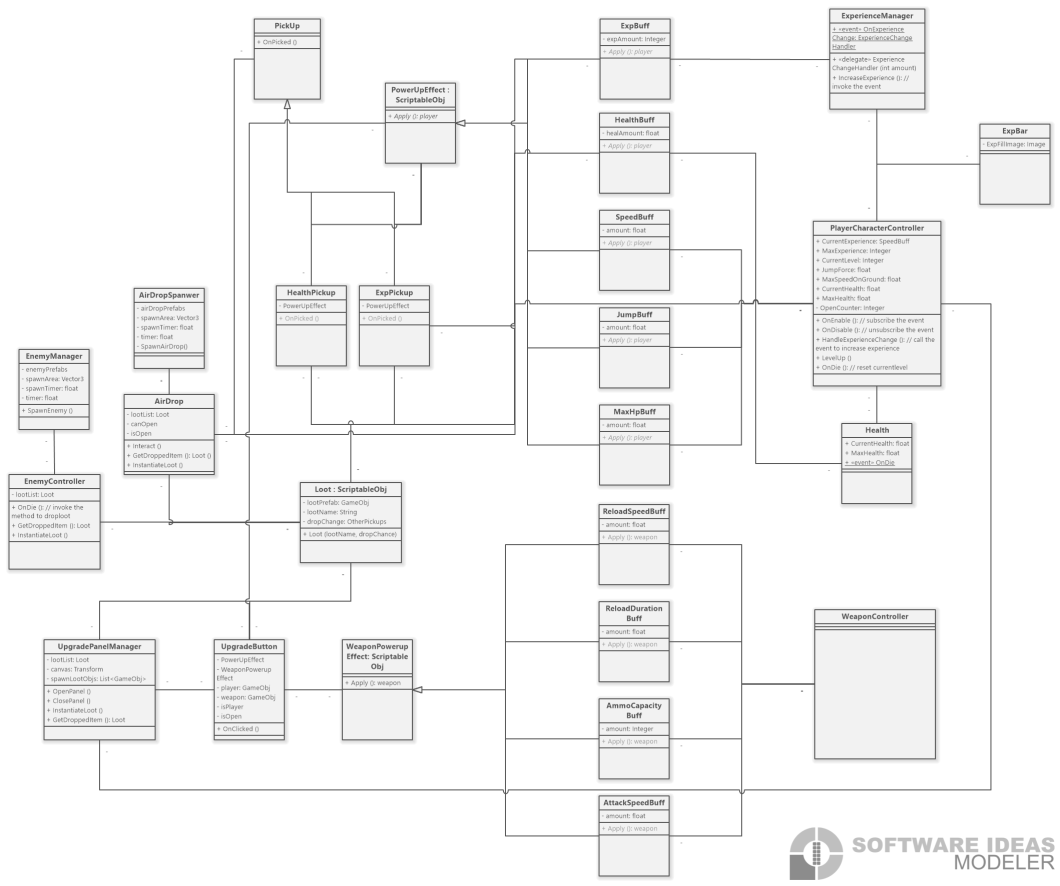
# **Iterations**

UML Charts

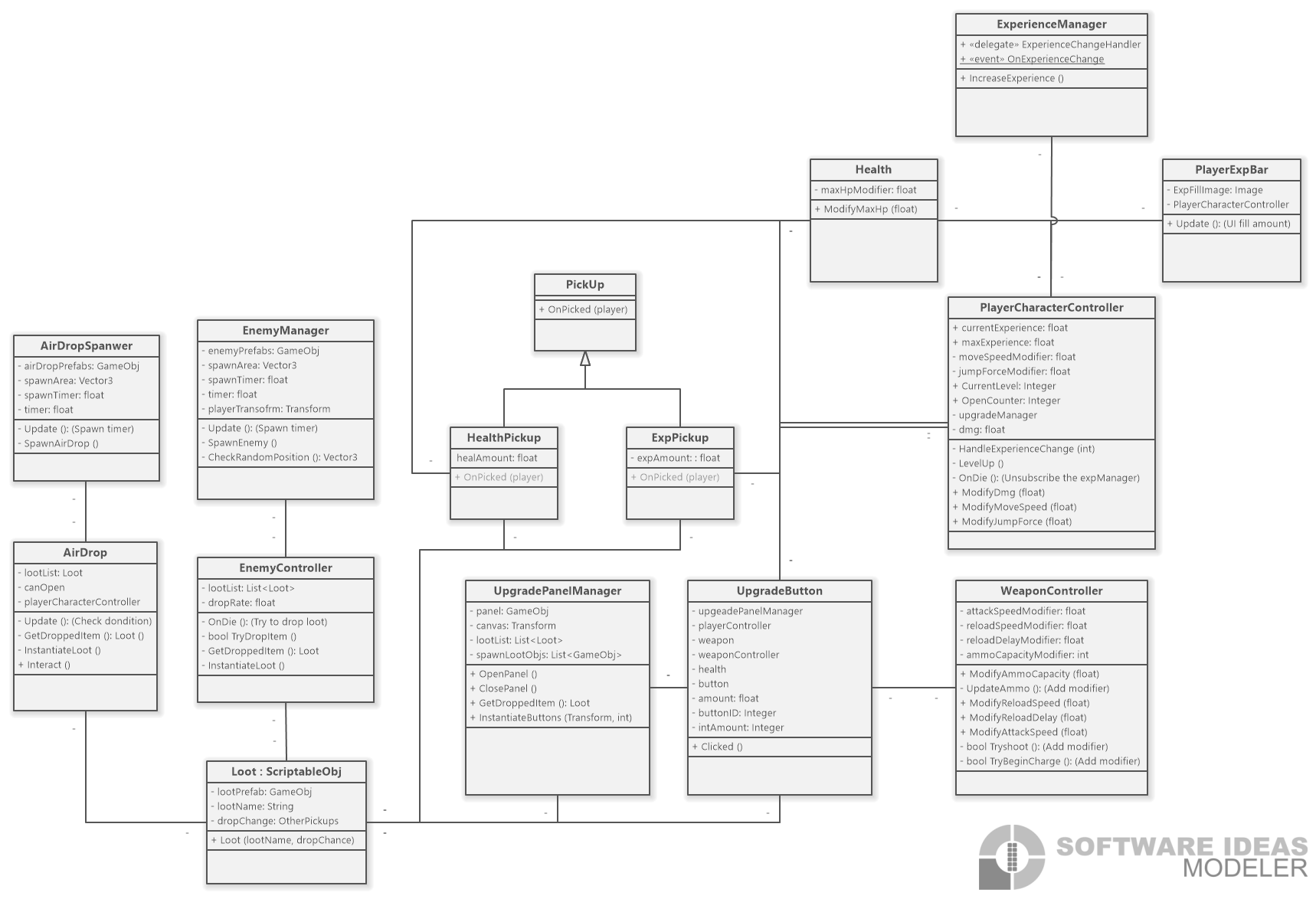
The version with simple experience, level-up system and enemy spawner.



The version I added different power-ups but the structure of them was too complex.

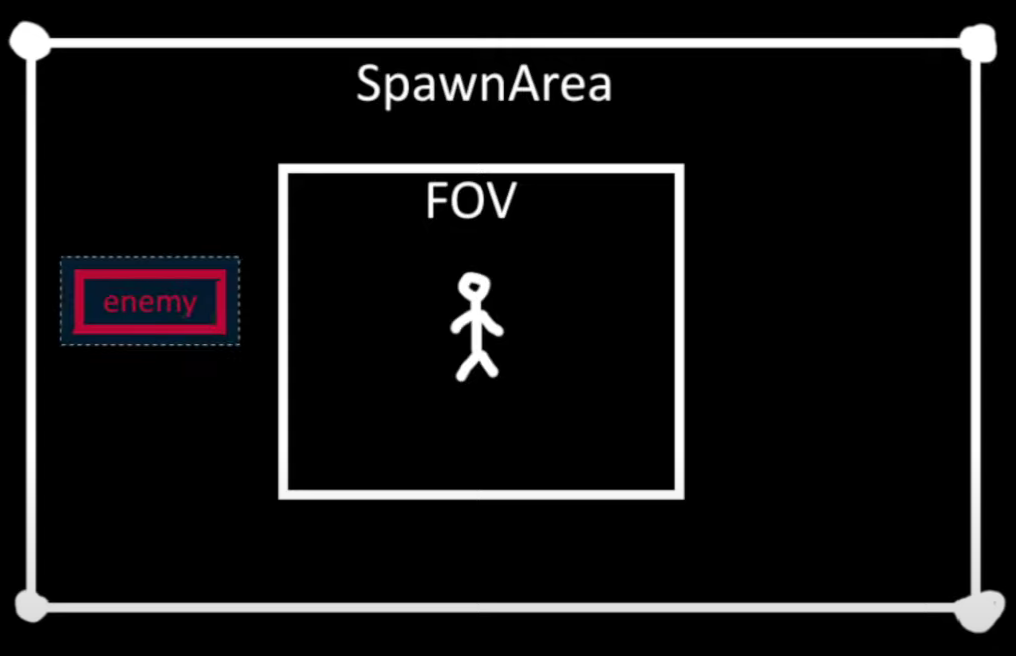


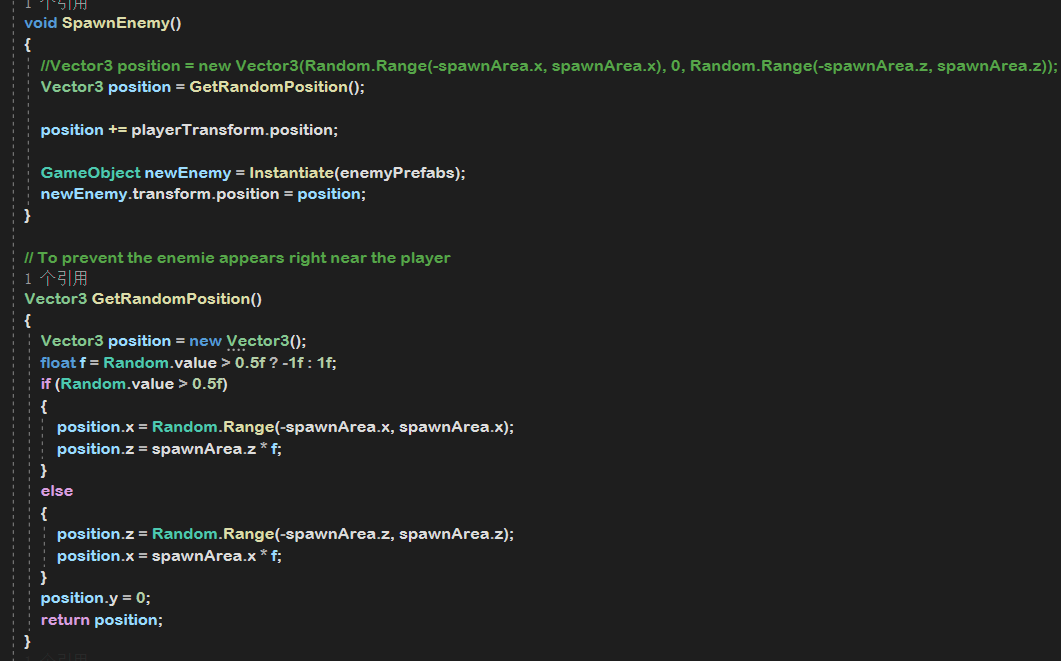
I thought this was my final version till I found the event system in the project itself and also other original script I could modify to suit my needs instead of creating a lot of new scripts myself.



Functions

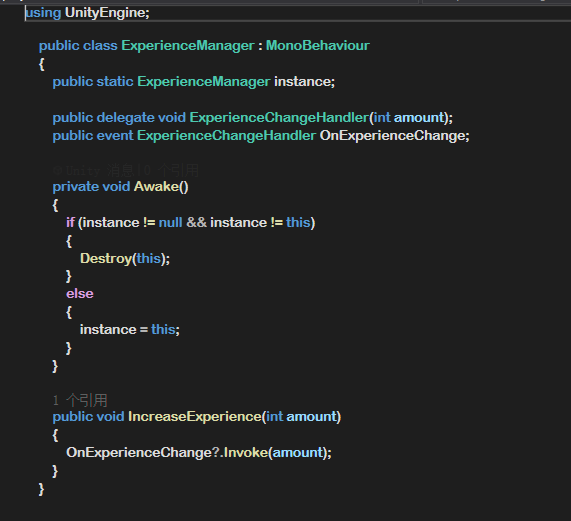
To prevent the situation that enemy is spawned right in front of the player, the position of enemies are iterated to only spawned in this area

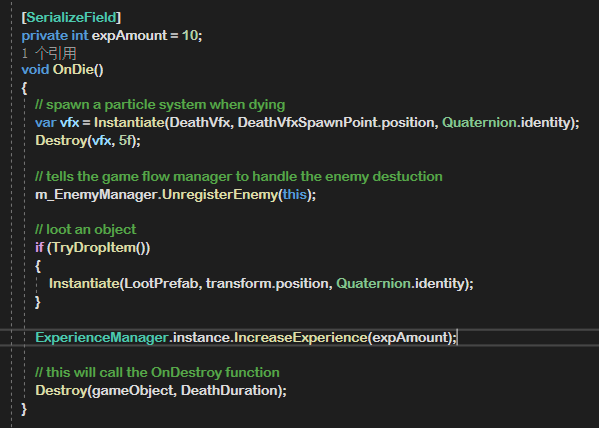




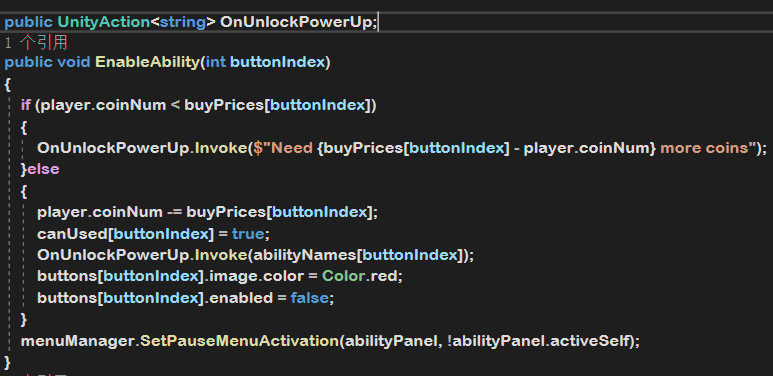
I found 3 different ways to do event functions in the project through the process.

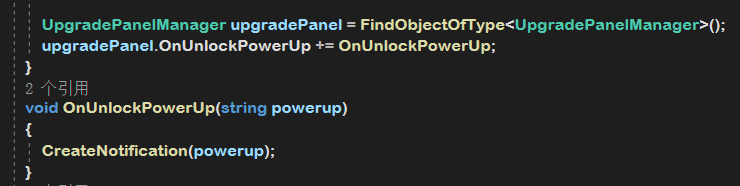
The first one I used:



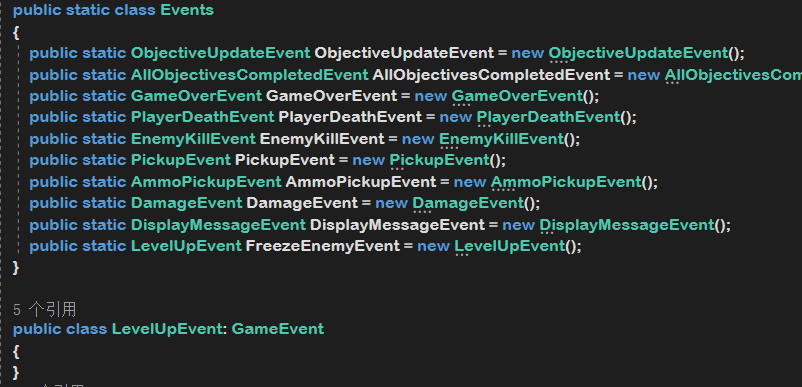


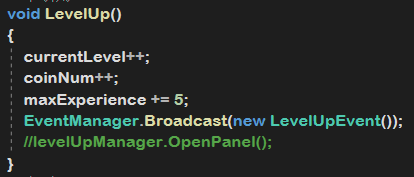
UnityAction:

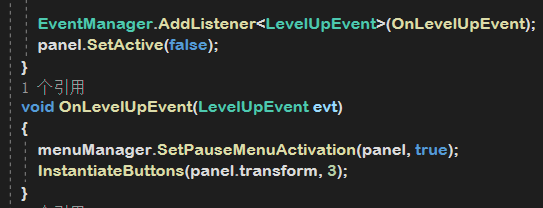




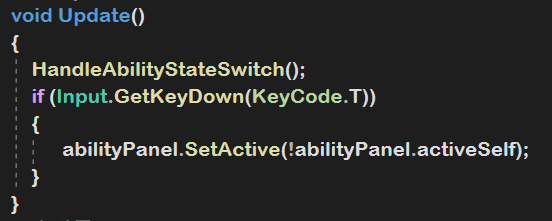
Event system from the project itself:

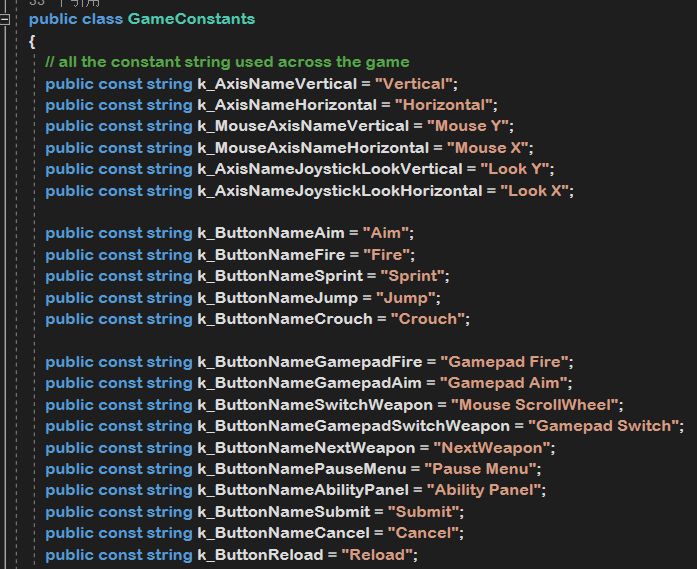


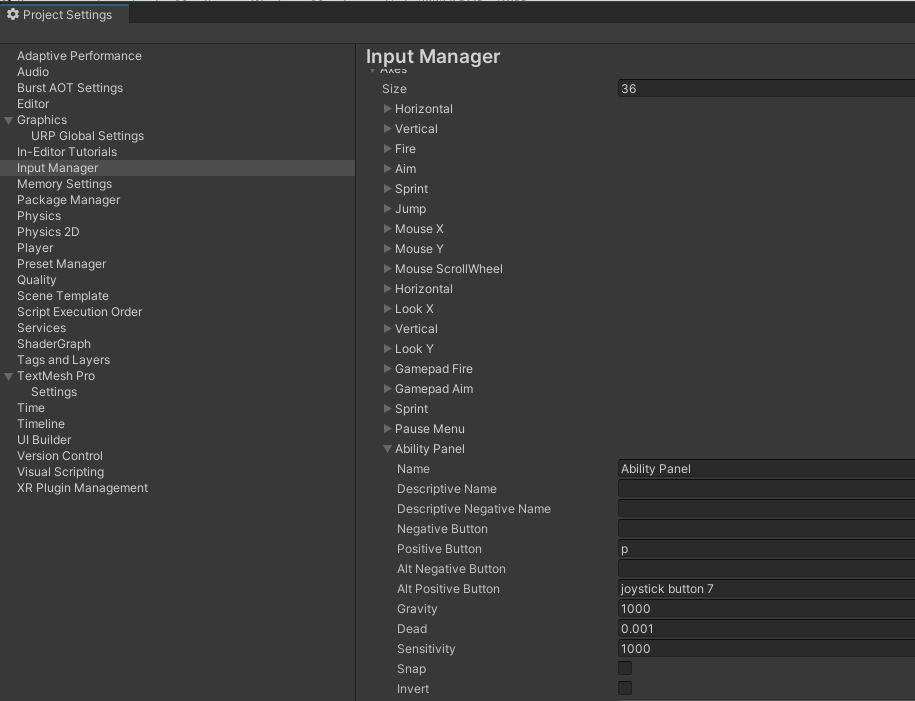


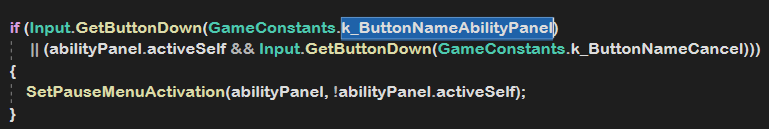


I was using ‘Input.GetKeyDown’ to toggle the ability panel before found out the project is using GameConstants and ‘InputManager’ to open the menu panels.

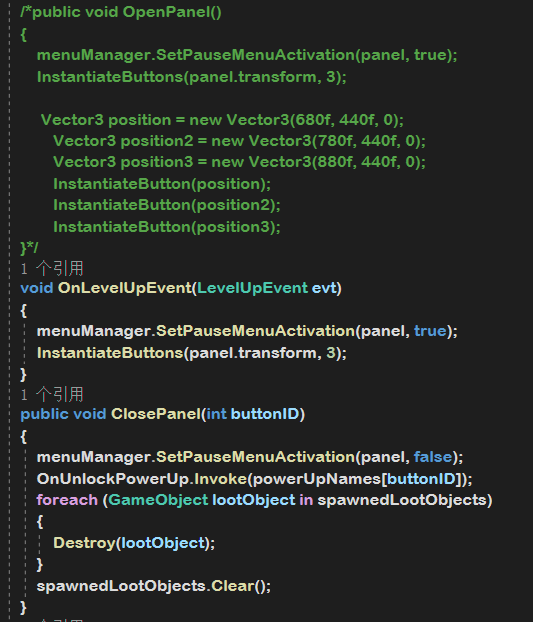




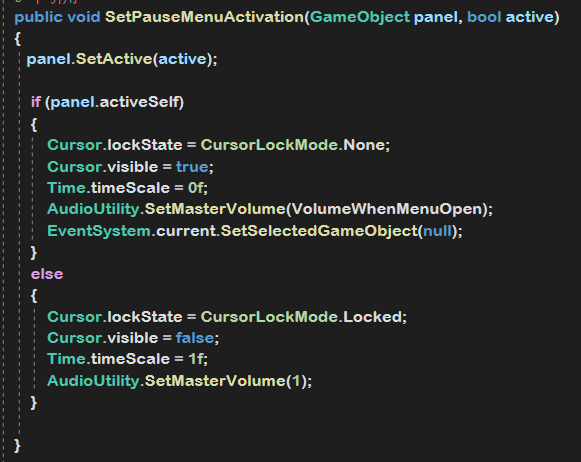




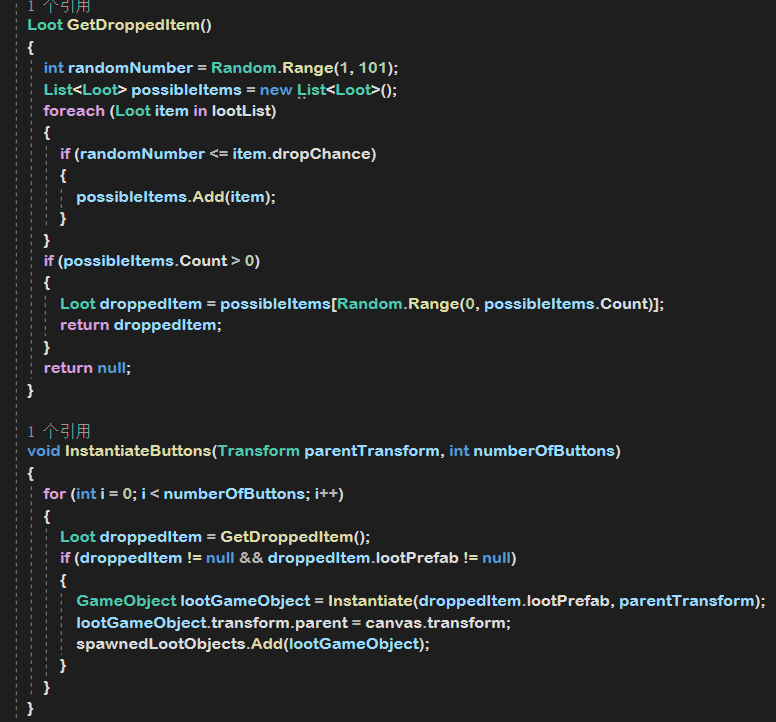
I set the button positions in the scripts at first then figureed to instantiate them in horizontal layout from a pareted transform.

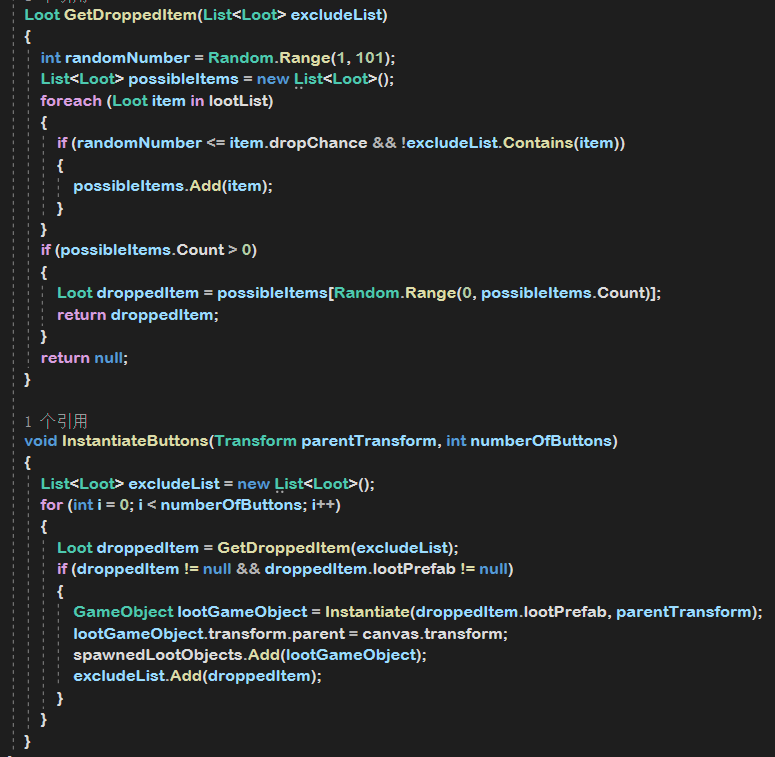


Also I paused the game myself by only set timescale before when leveling up the player but then modify the method from ‘InGameMenuManager’ so the other panels can also used this pause function.



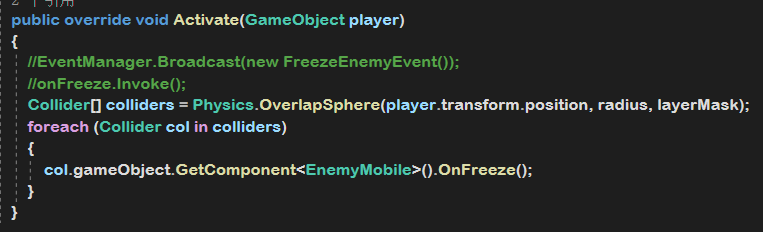
The power-ups could be spawned multiple times in the panel before then I added a exclude list to prevent that happening.

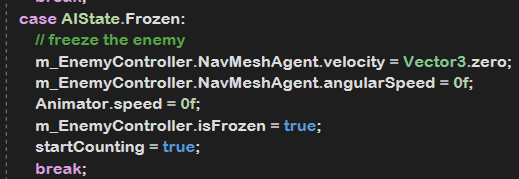




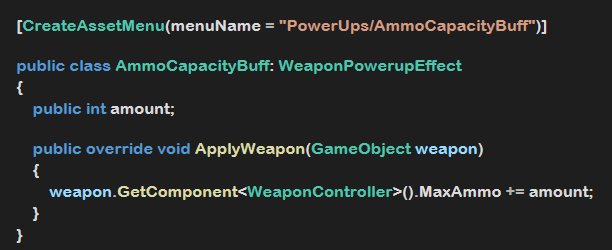
For the Freeze ability I was trying to add a modifier to all the speed (movement, rotation, animation) of the enemy but failed then modified the ‘EnemyMobile’ script to add a new ‘AiState’ to freeze the enemy’s behaviour based on NaveMeshAI.

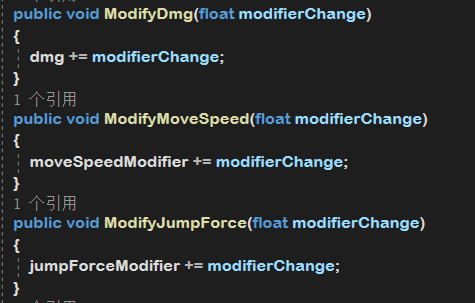
What’s more, I used event to fire the onFreeze method when activate the Freeze ability but it freezes all the enemies on the map while I only wanted a range of them frozen, then I made it by ‘overlapSphere’ detection.

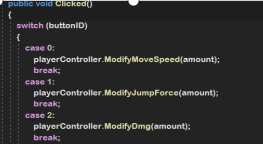




I used scriptable object for each of my passive power-ups at first but it didn’t turn out as a smart solution to such a simple function which basically only changes the value in other script, so instead I created the ‘Modify\_’ in the script I need to change values and call the target method when the button is clicked.







# **Reflection**

My Learning Process

I had no idea it could be this hard at the beginning and this focus track entirely flipped the opinion I held from the time I started coding which was last year, “If it works it works”. So I always knew that I was not good at coding but I was sort of proud of my prototyping skill with C# in Unity till I figured out that’s what they said ‘spaghetti script’. However, I’m really glad my teacher Armando not only talked about game programming but also coding logic in general to expand my understanding of why I should use and think of various solutions to a problem that’s already ‘fixed’ superficially.

I was trying hard to get out of my coding comfort zone by writing the special functions I’d never used or heard before such as overriding, virtual, interface, etc. The UML diagram helped me a lot at this point to let me have an overview of my script structure so that when I was stuck in coding or figuring out the solution I could always go back to the UML to check for alternative ways to reorganize the script relationships and optimize my solutions. In this process, I learned (game) programming from my own lesson, which was both challenging and rewarding at the same time because it underscored that there are often better solutions than what I currently have within a given context, which pushed me to always think and learn more from the magic coding world.

Modifying the micro FPS game provided a fascinating glimpse into how industry-standard projects manage complicated systems with such good organization. This experience granted me valuable insights into the essential relationships between scripts and mechanic functions within the game. On the other hand, it presented certain constraints as I aimed to accomplish my tasks within this framework. I found myself grappling with the balance between leveraging existing scripts for modifications and the potential necessity of creating entirely new scripts for my specific needs. This immersive journey has deepened my understanding of industry-standard practices and also offered a compelling experience as I imagined playing the role of an intern within the company while undertaking my coding tasks.

I want to become a decent programmer and from this focus track, I've gained a profound understanding that it’s not only about how much programming knowledge you have but more about solving problems and staying open to learning new things in this always-evolving field. I've got the hang of it, and moving forward, I'm eager to keep improving my understanding and stay curious about pragramming.

# **Reference**

Used Source & Tutorials

1. Wiki, C. T. V. S. (n.d.-b). Mechanics. Vampire Survivors Wiki. <https://vampire-survivors.fandom.com/wiki/Category:Mechanics>
2. Vampire Survivors review. (2022, October 27). Pcgamer. <https://www.pcgamer.com/vampire-survivors-review/>
3. Wright, S. T., & Janca, B. (2023, March 2). How Vampire Survivors went from hobby project to Game of the Year. GameSpot. <https://www.gamespot.com/articles/how-vampire-survivors-went-from-hobby-project-to-game-of-the-year/1100-6511980/>
4. Vampire Survivors Guide - IGN. (2022, June 7). IGN. <https://www.ign.com/wikis/vampire-survivors>
5. BMo. (2022, March 13). See your project to a cleaner setup with the OBSERVER PATTERN [Video]. YouTube. <https://www.youtube.com/watch?v=3FAHH1TYkjU>
6. Upgrades - Brotato Wiki. (n.d.). <https://brotato.wiki.spellsandguns.com/Upgrades>
7. Wiki, C. T. 2. M. T. D. (n.d.). Upgrades. 20 Minutes Till Dawn Wiki. <https://minutes-till-dawn.fandom.com/wiki/Upgrades>
8. TIMBER. (2021b, May 2). Creating an easy Ability System in Unity [Video]. YouTube. <https://www.youtube.com/watch?v=ry4I6QyPw4E>
9. James Makes Games. (2021, May 14). Pause in unity WITHOUT timescale [Video].YouTube. <https://www.youtube.com/watch?v=KPaEnLpu57s>