Game Requirements of Survivor

Zhang QianQi 1210029539

ZhuLiAn 1210016696

ZhuangYing 1210002147

PangJiaHao 1210002221

Distribution of work:

Zhuang Ying is responsible for the front-end of the game, need to design the game UI and page layout and game interaction, and at the same time for the game scriptwriting, writing the game plot, accounting for 25%.

Zhang QianQI is mainly responsible for the back-end of the game, writing code for the game as well as building the basic framework, optimizing the game performance, accounting for 25% of the work.

Pang JiaHao is mainly responsible for database writing and game testing, and also respon- sible for the maintenance of the game, accounting for 25%.

Zhu LiAn is responsible for the front-end of the game, need to design the game's art part, adjust the game's data content, accounting for 25%. Your introduction briefly explains the problem you address, and what you've achieved towards solving the problem. It's an edited and updated version of your context and objectives from your topic outline document.

1 Requirement Document

1.1 Problem

1.1.1 Proposed Problem

With the popularity of Vampire Survivor, more and more game developers decided to develop such games, and referred to these games as survivor games. This has also led to more and more survivor games in the recent game market, and users can choose more and more games, but only a few games can be praised, users are very dissatisfied with most of the games, because those game developers did not carry out new ideas, but simply copied Vampire Survivor. Some developers have also made new ideas, but the creative content is very boring, trying to integrate the gameplay of multiple games, regardless of whether it is suitable for the game, resulting in the game content is very tedious and boring, the learning cost is too high, and the player feels negative feedback. The quality of the game is uneven, and players don't get the experience they want.

In order to solve this problem, our team conducted a careful study of several excellent survivor games and developed a survivor game that belongs to our team based on market feedback, because we believe that players who like survivor games will love our game. For example, we will follow the rules of survivor games, make the early difficult, the later harvest gameplay, and add our own innovative mechanism, can modify the character's own attributes.

(1) **Type of game:**

The type and nature of the game we need to develop, whether to borrow vampire survivors and other monster fighting upgrade games

(2) **Game theme and Story:**

How should we design the setting and plot of the game,
and the storyline that the player will experience?

(3) **Target audience:**

Who is the audience for the game? Their age, interests and gaming experience, where are we competitive

(4) **Market competition:**

Analyzing competition for similar games, what do we stand out for？

1.1.2 Possible Client Requests

Considering that players, also known as customers, will have certain requirements for the game, we need to consider the questions raised by players in advance, which can help us developers to get the problems that players will consider when playing the game.

We will make improvements and add some functions to meet the needs of players and make the development project more popular with players. We have considered the following five different aspects:

(From our team and a questionnaire)

(1) **Game progress:**

Players care about whether the game runs smoothly, how long it takes to load the game, how much memory the game takes up

(2) **Game Features:**

Players may request specific game features, such as multiplayer mode, virtual reality support, or social interaction

(3) **Game platforms:**

Determine which platforms the game will be released on, such as PC, mobile, console....

(4) **Game experience:**

Players may want a game to provide a specific game experience, such as exciting game action or deep role-playing elements as well as the game's art and action effects

(5) **Audio and visual effects:**

Players are likely to care about the audio and visual effects of a game, including music, sound effects, and graphics quality.

1.1.3 Our Solution

Our ending proposal includes, but is not limited to, a detailed description of the game's design philosophy, operations, and game mechanics. An introduction to the technical architecture of developing games, including the game engines and programming languages used.

We will set a development schedule Use agile development ideas to list the required. resources Work with team members to complete,and will also regularly discuss the game test plan,Including quality assurance and performance optimization, after the completion of development we will consider the maintenance of the game, and may describe the game launch plan, including launch platform and marketing strategy.

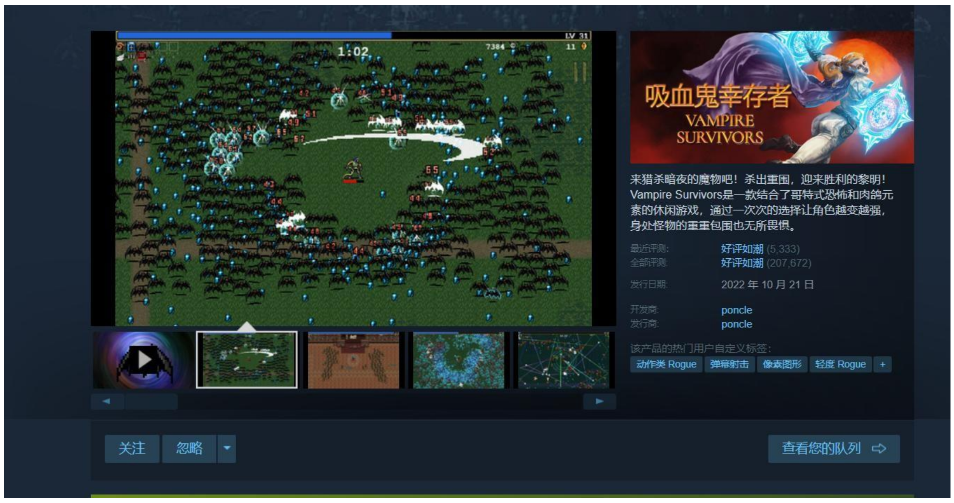
1.2 Background Information

At present, the popular survivor games on the market are "Vampire Survivor", "Potato Brothers" and "Origin plan", etc. These games are very classic, which also leads to more and more people playing this kind of game.

The popularity of "Vampire Survivor" lets game developers see another path, the operation of this kind of game is very simple, and limits the time of each game, which is very in line with the current fast-paced era, survivor games compared with other types of games, low production cost, developers can quickly complete the game content.

"Vampire Survivor" was released on December 17, 2021 rescue experience version, officially released on October 17, 2022, according to steam game catalog statistics, from the "Vampire Survivor" since the launch, the new release of survivor games more than 300, in two years, a year to release more than 150 games, The market space of surface survivor games is huge, but many games are just a surface imitation, there is no substantial new creativity, more than 300 games, only no more than 30 games can get special praise, indicating that less than one in ten good games.

It is essential to develop a good survivor-like game that is relevant to the general public, to break people's perception of the current market, and to increase people's choice.



来猎杀暗夜的魔物吧！杀出重围，迎来胜利的黎明！Vampire Survivors是一款结合了哥特式恐怖和肉鸽元素的休闲游戏，通过一次次的选择让角色越变越强，身处怪物的重重包围也无所畏惧。

最近评测:	好评如潮 (5,333)
全部评测:	好评如潮 (207,672)
发行日期:	2022 年 10 月 21 日
开发商:	poncle
发行商:	poncle

该产品的热门用户自定义标签:
动作类 Rogue　弹幕射击　像素图形　轻度 Rogue　+

1.3 Environments and system models

1.3.1 Environments

(1)Physical environment:

Players can play this game anywhere there is a computer

(2) **Hardware environment:**

The game needs to work well on a variety of computer hardware configurations, including different processors, graphics cards, and memory

(3) **Software environment:**

The game needs to run well on the Windows operating system

(4) **Network environment:**

The game does not require an Internet connection to play

1.3.2 System Model

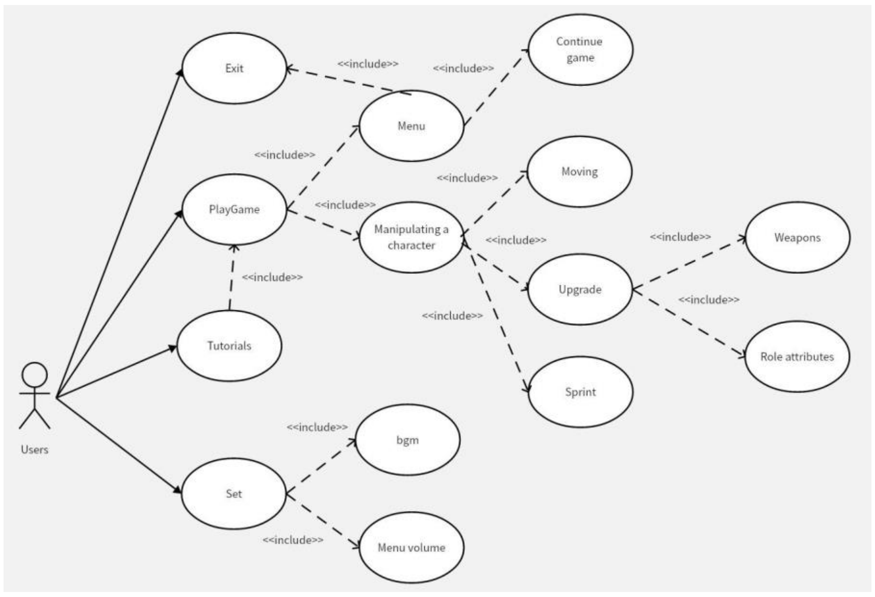
Functional requirements: The game needs to provide combat system, equipment system and other functions.

Performance requirements: The game needs to be able to respond quickly to player actions, for example, player movement and attack commands need to be executed within milliseconds.

Interface requirements: No interface requirements as a stand-alone game

1.3.3 Use case diagram

This use case diagram contains the main functions of the user



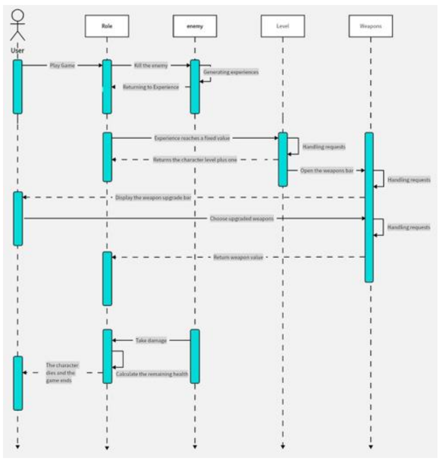
Users: Users can start the game, open the tutorial, open Settings and exit the game,

The start game contains the menu and the control character, the menu contains the continue game and the exit game, and the control character contains the movement. Upgrades and sprints, upgrades include weapons and stats

The tutorial includes starting the game

The Settings include bgm and menu volume

1.3.4 Sequence Diagram



The user's operation is as follows: Start the game, kill the enemy through the control of the character, the enemy will drop experience when the experience accumulated to a certain value, the character will upgrade when the upgrade pops up the weapon upgrade bar for the user to choose, and the weapon upgrade value will be returned to the character after the user has selected the weapon he wants to upgrade. When the enemy deals damage to the character, the character will deduct the damage value to calculate the remaining health. When the health reaches 0, the character dies and the game ends.

1.4 Function Requirements

1.4.1 Development Engine

Choose to develop in Unity engine and edit scripts in C# language

1.4.2 Operation Mode

W A S D control 2D vector movement, the Space key is the function key, ESC is the option/return key (then access the UI)

1.4.3 Character and weapon Settings

To achieve role-related functions, there must be a variety of different types of weapons, a variety of basic attributes, and access to previously designed control-related systems

1.4.4 Game Music

Search for sound sources, design and implement audio management system, system for different scenes for independent control

1.4.5 Surgical Style

The art style is unified to pixel style

1.4.6 UI

Design the whole UI system, such as the start and exit buttons of the menu interface, in-game pause and upgrade related interfaces, game end interface, etc

1.4.7 Enemy Settings

Realize a variety of enemies, both ranged and melee, with different state machines and different attributes

1.4.8 Item Settings

Design health items and experience gain items, will drop probability when the enemy dies, different enemies drop probability is different

1.4.9 Tutorial Settings

Design the novice tutorial interface, can enter from the main menu, not too complicated, directly tell the player each interface each button corresponding to what operation

1.4.10 Game packaging Settings

Package the project file, export it from unity, set up the relevant ICONS, etc

1.5 Non-functional Requirements

1.5.1 Performance Requirements

User requirements in software response speed, result accuracy, runtime resource consumption, etc

1.5.2 Reliability Requirements

There are no obvious bugs when users use, so that users who reach the operating environment of the device can use the software stably

1.5.3 Usability Requirements

With embedded software use tutorial, modular development

1.5.4 Operating Environment Constraints

User requirements for ease of use and aesthetics of the interface, as well as user-oriented documentation and training materials

1.5.5 External Ports

No external interface

1.5.6 Security Requirements

No user au is required and no user device data is required

2 Design Docament

2.1 Purpose

The purpose of development documentation is to design software requirements and standardize development patterns.

(1). Observe the market and determine the direction of the game.

(2). Make the development process transparent to facilitate flexible changes in requirements and development schedule control.

(3). Modify the game content and design the overall framework.

(4). Design use case diagram, class diagram and sequence diagram to make the development visual.

(5). Implement functional requirements.

(6). Implement non-functional requirements and improve system design.

We designed a survivor-like game called "Survivor," which requires the player to play as a young girl to survive a steady stream of robot attacks on a deserted land. The game is designed to be limited to 10 minutes, and the game automatically ends when 10 minutes are reached. Players need different weapon combinations to survive.

2.2 General priorities

Game type

We believe that defining the genre is the highest priority.

Playability

Interesting games can improve the game, even if a game art, music and other factors are good, there is no good playability will still be abandoned by users, playability determines whether the game will be recognized by the public.

Art

For a game, art is a plus, good art can improve the user experience when playing, let the user more immersed.

2.3 Design Outline

Mix bottom-up and top-down methods

We used a mix of top-down and bottom-up methods to make this game. In this way, we first designed the main framework of the entire project, and through discussion identified the details and functions that needed to be completed.

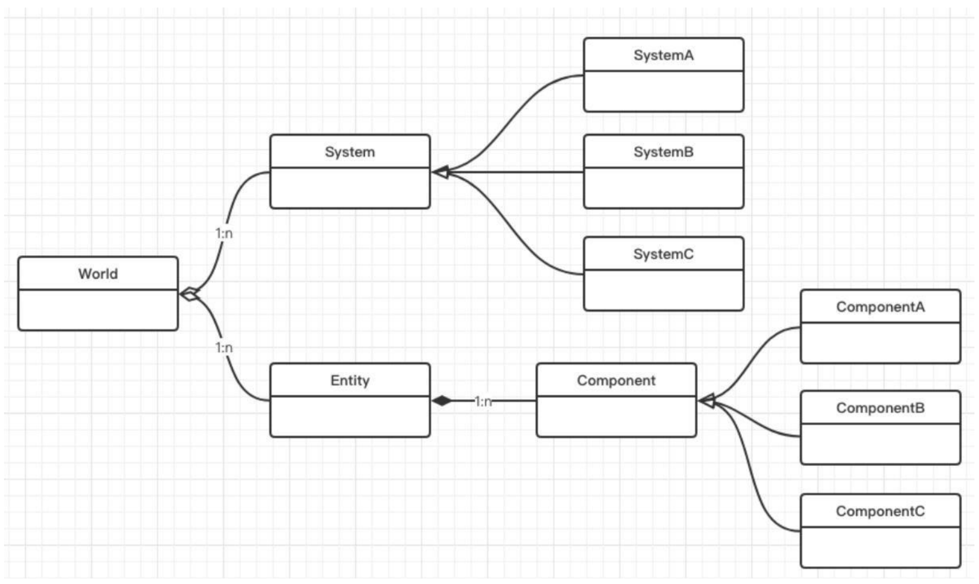
We follow the following principles

Divide and rule

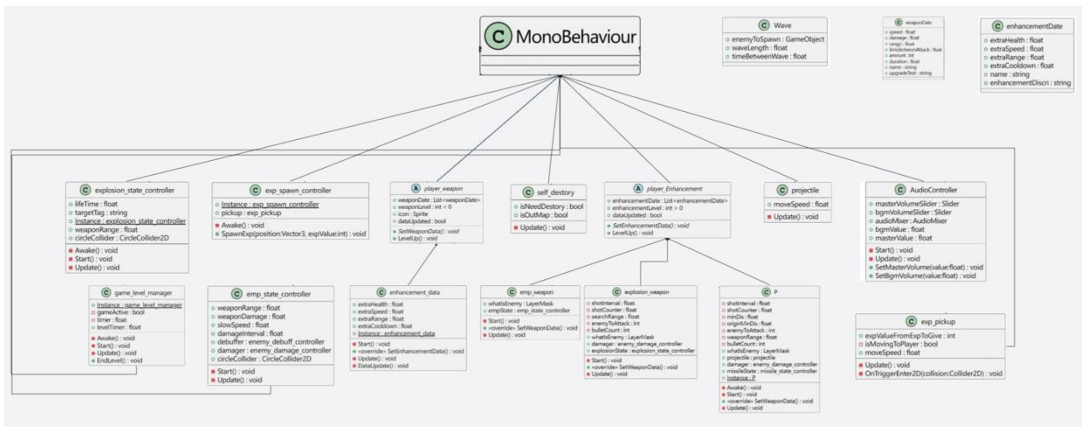
Each large part of the function is broken down into small parts. Assign tasks and responsibilities to specific members.

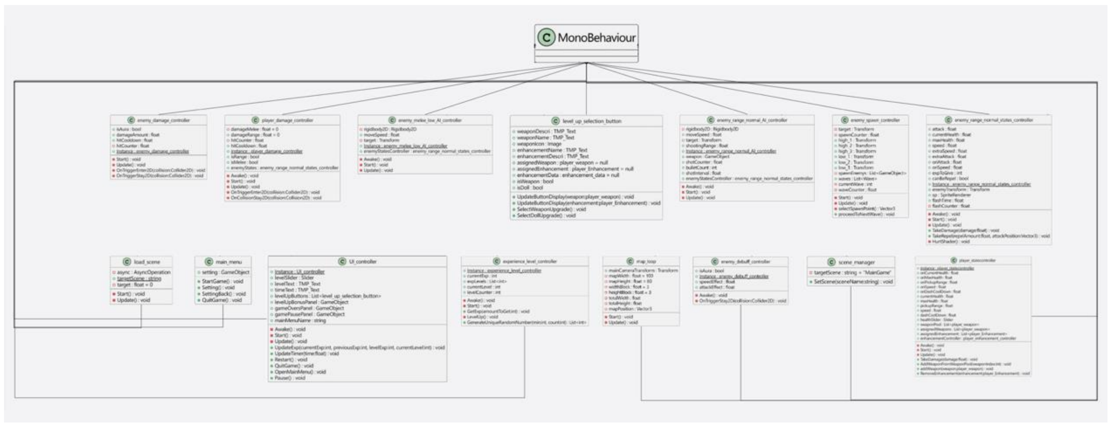
2.4 ECS architecture

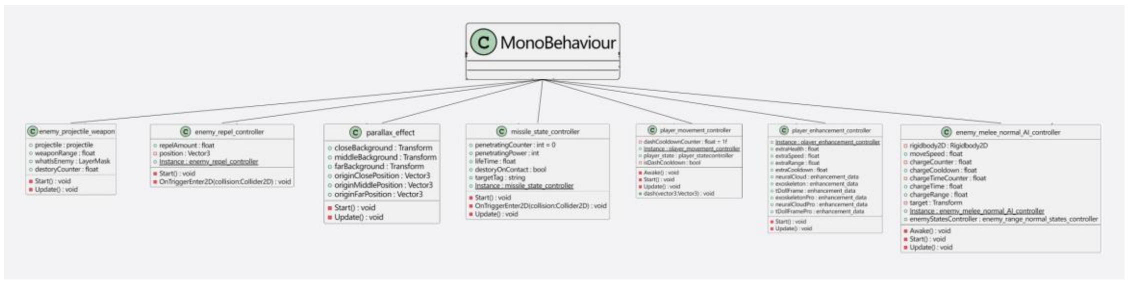
The game adopts the ECS architecture, in the ECS architecture, the components only save the state without behavior, the entity is mounted with components, and finally the system is responsible for obtaining the relevant components to complete the corresponding operations, the core idea is to separate the data and operations. Systems have only methods, and components have only member variables.



2.5 Class Diagram.







We use the Unity engine to develop the game, all classes inherit MonoBehaviour class, scripting language each class attached to the object to complete the action of each module

2.6 Main design problems

Game type, art style direction, and game type must be carefully considered for a game. We chose Survivor games with good recent market sales as the main game type, and considered horizontal version of Castlevania as the main game type, but rejected it after careful consideration. Castlevania-like games are difficult to produce, and it is not easy for novice game developers to complete this type. Therefore, it is converted to a survivor game, which is difficult and easy to use, and the members of the group have more playing experience for this type of game.

The art style direction is pixel style. Members of the group have not studied painting in detail, so they cannot draw other fine art styles in a short time. They need a style that can produce fine art effects quickly. Pixel style is easy to use, so the art style is pixel style.

2.7 Other design details

At the beginning of the design process, it was decided to add different characters, each role has a different main weapon, and designed ten available weapons, hoping that users can find the game experience they want through the combination of different weapons during play.

However, during the development process, it was found that the excessive number of weapons led to bugs in the compatibility of some weapons, and the mechanism and attributes of weapons were difficult to balance, and the learning cost was too high. Therefore, we reduced the number of weapons to four weapons, and the attributes of characters themselves could be upgraded when upgrading, so that users would no longer focus on upgrading weapons.

3 Testing

Game software testing is a complex and detailed process designed to ensure the quality of the game and the player experience. Here are some steps to follow in playtesting

3.1 Functional test

Test whether all features of the game work as designed, including game mechanics, user interface, story, etc.

Find and document any functional errors or problems.

We tested it according to the original design intention and design concept, and then the test results found that this is a decompressed, constantly upgrade the level to improve the player's sense of access to the game, in the beginning of the difficulty as the level continues to increase, the difficulty of the game also decreases, and after discussion with the developer, the game mechanics are not a problem. The user interface of the game tested is also very thematic and fits in well with the content of the game.

3.1.1 Test environment Settings

- Prepare the hardware and software environment required for testing.

- Make sure you have a test environment for different platforms (e.g. PC, console, mobile).

We developed for computer games, only run on the computer, do not support other platforms.

3.1.2 Performance/compatibility testing

- Ensure that the game works properly under different operating systems, hardware configurations, and network environments.

- Test game performance at different resolutions and screen sizes.

3.1.3 Operation test environment

Win11+corei7 (integrated display)

Windows 10+core i7 (only)

Mac+Sonoma14.0 (Integrated display)

Mac+ios13.7 (Integrated Display)

The game we generated can be opened in the windows system, but it cannot run our ddl program on mac. During the query, we learned that the program that wants to run in the ios system must be certified by apple, which in my opinion is a little weak, because the test cycle is too short, we did not apply for the certification successfully. So we can only run the script code to achieve, and then found in the experiment, each environment will produce different errors, may produce such problems under the same system, after the investigation found that the unity development platform will produce different compilation environments according to different systems, which is very headache, because the script code is not rigorous, Such problems arise.

3.1.4 Resolution test

1280 x 800 (13.3, 14.1, 15.4 inch notebook) : **Qualified**

1440 x 900 (17.1 inches, 19 inches) : **Qualified**

Since the games we develop are computer games, we only test them on desktops and laptops

3.1.5 User interface test

- Test the usability and ease of use of the game interface.
- Check that fonts, ICONS, menus, instructions, etc. are clear and easy to understand.

We have tested the game interface buttons and the game interface guidelines:

The guidelines are clear and meet expectations.

However, in the course of the game, for the introduction of the game content and the introduction of the weapons, the testers felt that the introduction was a little bloated and long. After communicating with the developers, they decided to reduce the introduction content and increase the introduction of animation, so that the interface looked more concise and the introduction was more clear

3.2 Security test

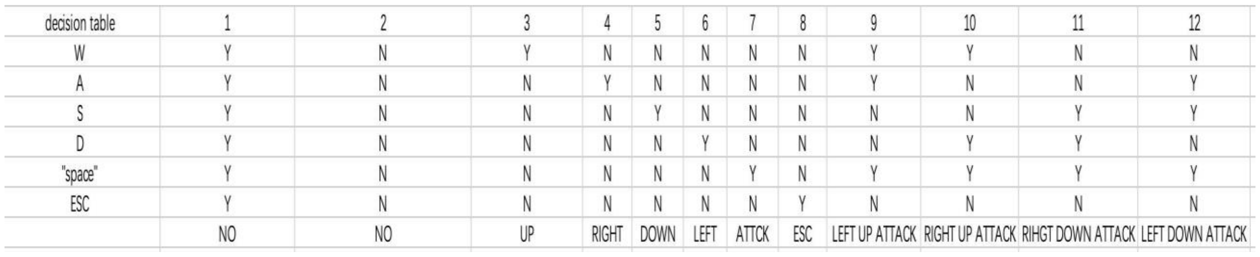
- Test for security vulnerabilities in the game, such as data protection and privacy issues.
- Ensure that transactions and communications in online games are secure.

Since our game is a single-player game, we only run on our own computer and do not need to worry about security issues

3.3 Black box testing

3.3.1 Decision table

We made a decision table with a description of the buttons in the game, and some possible scenarios:



decision table	1	2	3	4	5	6	7	8	9	10	11	12
W	Y	N	Y	N	N	N	N	N	Y	Y	N	N
A	Y	N	N	Y	N	N	N	N	Y	N	N	Y
S	Y	N	N	N	Y	N	N	N	N	N	Y	Y
D	Y	N	N	N	N	Y	N	N	N	Y	Y	N
"space"	Y	N	N	N	N	N	Y	N	Y	Y	Y	Y
ESC	Y	N	N	N	N	N	N	Y	N	N	N	N
	NO	NO	UP	RIGHT	DOWN	LEFT	ATTCK	ESC	LEFT UP ATTACK	RIGHT UP ATTACK	RIHGT DOWN ATTACK	LEFT DOWN ATTACK

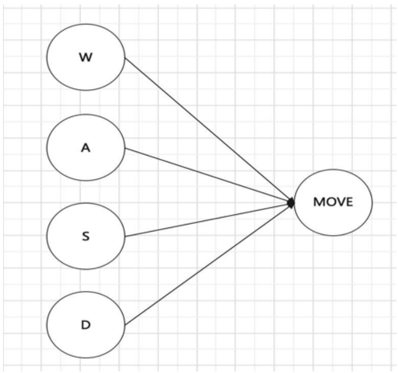
3.3.2 Partition equivalence class

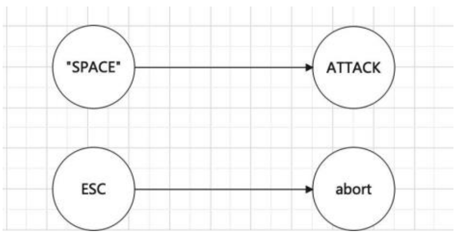
We describe all the buttons and interfaces in the game and all the apis, divide them into equivalence classes, and test all these interfaces:

class	valid equivalence class	invalid equivalence class
Esc	Esc	Other Computer Keys
W	W	Other Computer Keys
A	A	Other Computer Keys
S	S	Other Computer Keys
D	D	Other Computer Keys
" "	" "	Other Computer Keys
(START)Button	(START)Button	Click on the other buttons
(TUTORIAL)Button	(TUTORIAL)Button	Click on the other buttons
(SETTING)Button	(SETTING)Button	Click on the other buttons
(EXIT)Button	(EXIT)Button	Click on the other buttons
(Back Main Menu)Button	(Back Main Menu)Button	Click on the other buttons
(Continue)Button	(Continue)Button	Click on the other buttons
1Button	1Button	Click on the other buttons
2Button	2Button	Click on the other buttons
(StartGame)Button	(StartGame)Button	Click on the other buttons
(MainMenu)Button	(MainMenu)Button	Click on the other buttons
(MASTER VOLUME)Slideway	(MASTER VOLUME)Slideway	Click on the other buttons
(BGM VOLUME)Slideway	(BGM VOLUME)Slideway	Click on the other buttons

3.3.3 Cause and effect diagram

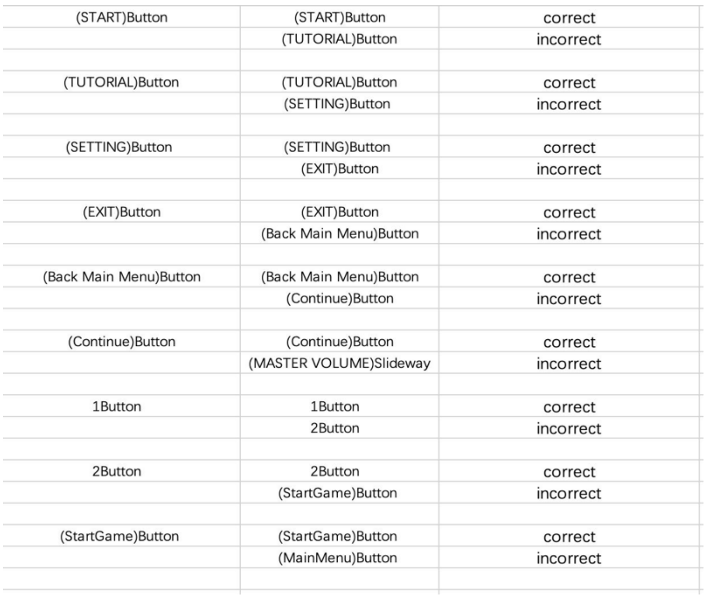
Our implementation of the function describes the cause-and-effect diagram, for example, that a move can only be made if any of the WASD keys are pressed, that an attack occurs if and only if the space bar is pressed, and that an exit occurs only if the ESC key is pressed:



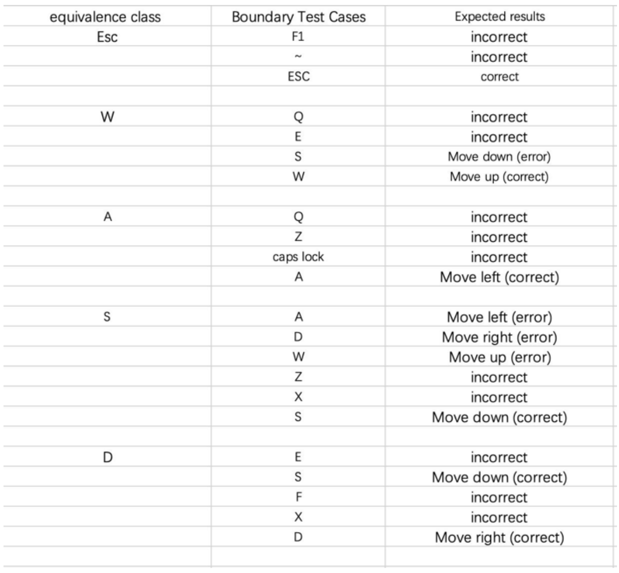


3.3.4 Boundary test

We first divide equivalence classes, and then complete the boundary test by testing the surrounding equivalence classes, such as W, that is, testing the invalid equivalence classes of Q, A, S, D, and E



(START)Button	(START)Button	correct
	(TUTORIAL)Button	incorrect
(TUTORIAL)Button	(TUTORIAL)Button	correct
	(SETTING)Button	incorrect
(SETTING)Button	(SETTING)Button	correct
	(EXIT)Button	incorrect
(EXIT)Button	(EXIT)Button	correct
	(Back Main Menu)Button	incorrect
(Back Main Menu)Button	(Back Main Menu)Button	correct
	(Continue)Button	incorrect
(Continue)Button	(Continue)Button	correct
	(MASTER VOLUME)Slideway	incorrect
1Button	1Button	correct
	2Button	incorrect
2Button	2Button	correct
	(StartGame)Button	incorrect
(StartGame)Button	(StartGame)Button	correct
	(MainMenu)Button	incorrect



equivalence class	Boundary Test Cases	Expected results
Esc	F1	incorrect
	~	incorrect
	ESC	correct
W	Q	incorrect
	E	incorrect
	S	Move down (error)
	W	Move up (correct)
A	Q	incorrect
	Z	incorrect
	caps lock	incorrect
	A	Move left (correct)
S	A	Move left (error)
	D	Move right (error)
	W	Move up (error)
	Z	incorrect
	X	incorrect
	S	Move down (correct)
D	E	incorrect
	S	Move down (correct)
	F	incorrect
	X	incorrect
	D	Move right (correct)

3.4 White box testing

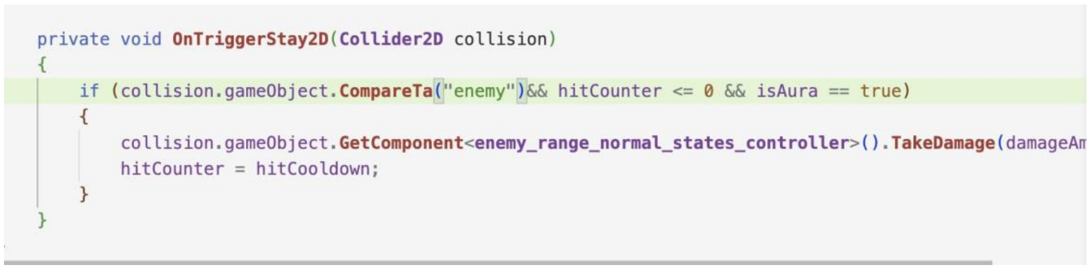
3.4.1 Static testing

-Formally review the basics

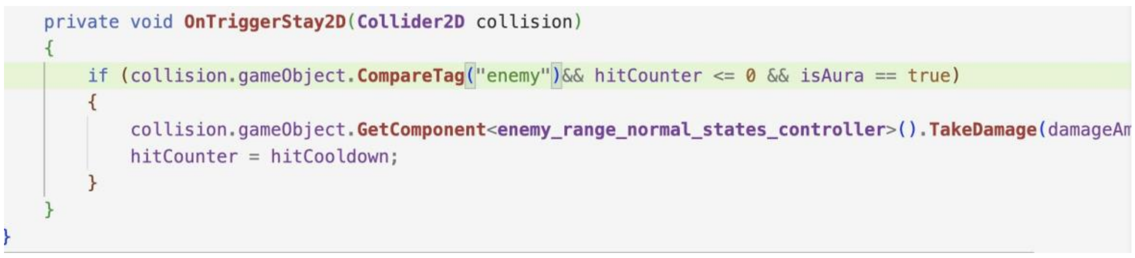
-Peer review

-Walk-through

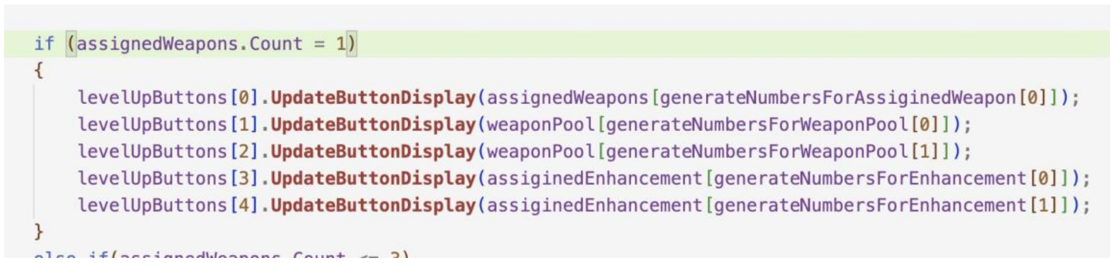
-Inspect



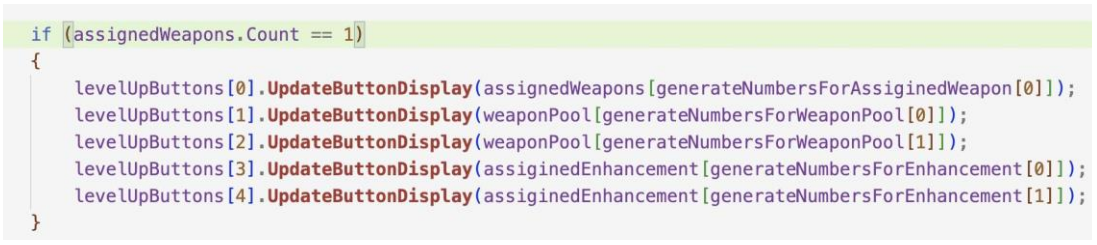
```
private void OnTriggerStay2D(Collider2D collision)
{
    if (collision.gameObject.CompareTa("enemy")&& hitCounter <= 0 && isAura == true)
    {
        collision.gameObject.GetComponent<enemy_range_normal_states_controller>().TakeDamage(damageAm
        hitCounter = hitCooldown;
    }
}
```



```
    private void OnTriggerStay2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("enemy")&& hitCounter <= 0 && isAura == true)
        {
            collision.gameObject.GetComponent<enemy_range_normal_states_controller>().TakeDamage(damageAm
            hitCounter = hitCooldown;
        }
    }
}
```



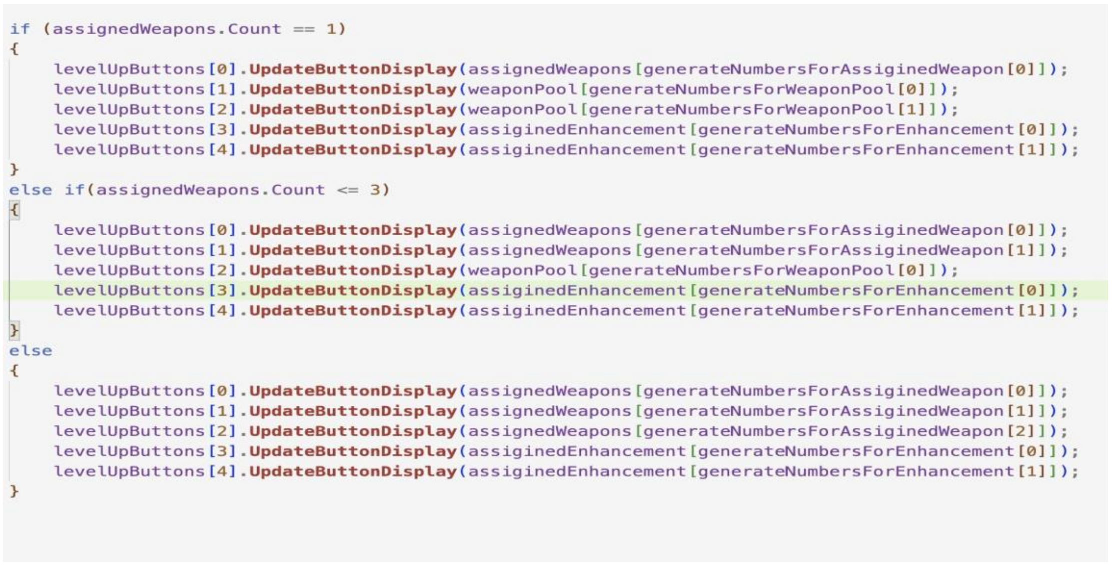
```
  if (assignedWeapons.Count = 1)
  {
      levelUpButtons[0].UpdateButtonDisplay(assignedWeapons[generateNumbersForAssiginedWeapon[0]]);
      levelUpButtons[1].UpdateButtonDisplay(weaponPool[generateNumbersForWeaponPool[0]]);
      levelUpButtons[2].UpdateButtonDisplay(weaponPool[generateNumbersForWeaponPool[1]]);
      levelUpButtons[3].UpdateButtonDisplay(assiginedEnhancement[generateNumbersForEnhancement[0]]);
      levelUpButtons[4].UpdateButtonDisplay(assiginedEnhancement[generateNumbersForEnhancement[1]]);
  }
  else if(assignedWeapons.Count <= 3)
```



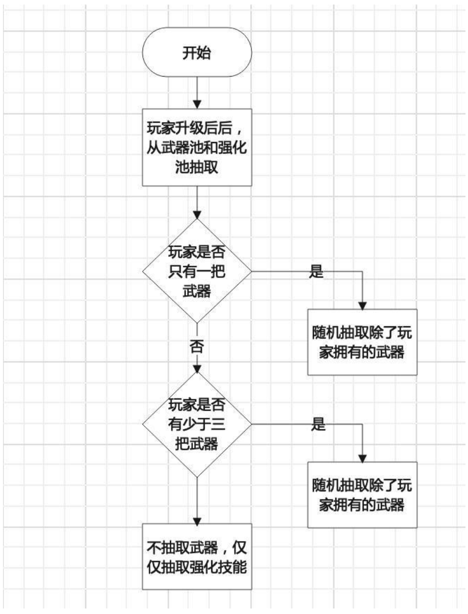
```
  if (assignedWeapons.Count == 1)
  {
      levelUpButtons[0].UpdateButtonDisplay(assignedWeapons[generateNumbersForAssiginedWeapon[0]]);
      levelUpButtons[1].UpdateButtonDisplay(weaponPool[generateNumbersForWeaponPool[0]]);
      levelUpButtons[2].UpdateButtonDisplay(weaponPool[generateNumbersForWeaponPool[1]]);
      levelUpButtons[3].UpdateButtonDisplay(assiginedEnhancement[generateNumbersForEnhancement[0]]);
      levelUpButtons[4].UpdateButtonDisplay(assiginedEnhancement[generateNumbersForEnhancement[1]]);
  }
```

3.4.2 Dynamic test

Update the upgrade interface option button, the player's upgrade options are randomly selected from the weapon pool and the enhancement pool, when the player has a weapon, one upgrade option is the existing weapon, and the other two are drawn from the new weapon



```
if (assignedWeapons.Count == 1)
{
    levelUpButtons[0].UpdateButtonDisplay(assignedWeapons[generateNumbersForAssiginedWeapon[0]]);
    levelUpButtons[1].UpdateButtonDisplay(weaponPool[generateNumbersForWeaponPool[0]]);
    levelUpButtons[2].UpdateButtonDisplay(weaponPool[generateNumbersForWeaponPool[1]]);
    levelUpButtons[3].UpdateButtonDisplay(assiginedEnhancement[generateNumbersForEnhancement[0]]);
    levelUpButtons[4].UpdateButtonDisplay(assiginedEnhancement[generateNumbersForEnhancement[1]]);
}
else if(assignedWeapons.Count <= 3)
{
    levelUpButtons[0].UpdateButtonDisplay(assignedWeapons[generateNumbersForAssiginedWeapon[0]]);
    levelUpButtons[1].UpdateButtonDisplay(assignedWeapons[generateNumbersForAssiginedWeapon[1]]);
    levelUpButtons[2].UpdateButtonDisplay(weaponPool[generateNumbersForWeaponPool[0]]);
    levelUpButtons[3].UpdateButtonDisplay(assiginedEnhancement[generateNumbersForEnhancement[0]]);
    levelUpButtons[4].UpdateButtonDisplay(assiginedEnhancement[generateNumbersForEnhancement[1]]);
}
else
{
    levelUpButtons[0].UpdateButtonDisplay(assignedWeapons[generateNumbersForAssiginedWeapon[0]]);
    levelUpButtons[1].UpdateButtonDisplay(assignedWeapons[generateNumbersForAssiginedWeapon[1]]);
    levelUpButtons[2].UpdateButtonDisplay(assignedWeapons[generateNumbersForAssiginedWeapon[2]]);
    levelUpButtons[3].UpdateButtonDisplay(assiginedEnhancement[generateNumbersForEnhancement[0]]);
    levelUpButtons[4].UpdateButtonDisplay(assiginedEnhancement[generateNumbersForEnhancement[1]]);
}
```


开始
玩家升级后后，从武器池和强化池抽取
玩家是否只有一把武器
是
随机抽取除了玩家拥有的武器
否
玩家是否有少于三把武器
是
随机抽取除了玩家拥有的武器
不抽取武器，仅仅抽取强化技能

(1)Esc	Bring up the menu
(2)W	moves upward
(3)A	moves to the left
(4)S	moves down
(5)D	moves to the right
(6) ""	attack

(7)(START)Button: Click the button to enter the game

(8)(TUTORIAL)Button: Click the button to enter the tutorial

(9)(SETTING)Button: Click the button to enter the setting

(10)(EXIT)Button: Click the button to launch the game

(11)(Back Main Menu)Button: Click the button to return to the main menu

(12)(Continue)Button: Click the button to continue

(13)1Button: Click the button to go to the first side of the tutorial

(14)2Button: Click the button to go to the second side of the tutorial

(15)(StartGame)Button :Click the button in the tutorial to go directly to the game

(16)(MainMenu)Button :Click the button in the tutorial to return to the main menu

(17)(MASTERVOLUME)Slideway:

System music adjustment slide for the video on the main screen

(18)(BGM VOLUME)Slideway:

7-2-3-4-5-6:The basic functions of the game can be achieved

7-1-11: Return to the main menu in the game interface

7-1-12: Continue the game after selecting ESC in the game

8-1: Reach the first page of the tutorial

8-2: Reach the second page of the tutorial

8-15: Go straight to the game in the tutorial screen

8-16: Go back to the main menu in the tutorial screen

9-17: Adjust the video music size on the home screen

9-18:Adjust the size of the in-game music

10: Click the button to exit the game directly

3.4.3 Code Coverage Test

Make sure the test covers all code paths. This means checking that every walk, attack, and damage feature in the game can be triggered in the test.

According to the test, the game walks smoothly, but in the first test, it was found that when the character is damaged, only special effects can be triggered, but the blood bar will not be reduced accordingly. This issue was then resolved in the second test, when a hit from either a long range or a short range enemy caused damage to the character

3.4.4 Logic test

Test game logic, such as attack logic, health reduction, etc. Make sure that the logic works correctly in different situations.

We have two sets of big logic, namely monster logic and player logic, but the player logic test is not very difficult, only need to be attacked by dropping blood, launch attacks, and pick up things dropped by monsters to upgrade, in the logic test, received attack drop blood test has been completed, attack logic in attacking monsters, attack power will gradually increase with the player's level. Players can choose to upgrade the value of a weapon or improve their own attributes when upgrading, which will gradually reduce the difficulty of the game to form a logical closed loop. The enemy logic should be further classified according to the type of monster, when entering the game at the beginning of the primary enemy, this enemy can only carry out short-range attacks, when the character and the enemy coincide, with a certain attack rate to make the character blood, and when the character's physical attack will reduce the corresponding amount of blood, when the blood is reduced to 0, the target will disappear. With the increase of time, the long-range enemy will participate in the battle, and the attack logic of this enemy is different from that of the progress enemy, and the attack distance of the enemy will increase significantly. This enemy will launch a straight line attack with the coordinate of the character as the target at that time. The logical thinking of this attack is to enable the player to avoid the possibility of being attacked, and the attack speed is not fast, so as to increase the player's survival ability.

The same logic as the first kind of enemy death, when attacked by the character, will reduce the corresponding health, when the health <=0, this disappears and drops experience points that the player can use to level up, and may also drop the corresponding health recovery equipment. The third type of enemy appears later in the game, when the character has a higher level, the enemy's health will be increased accordingly, and there will be two logical attacks at the same time.

when the distance is $1<X<5$, a remote attack will be used. When $0<=X<=1$, the character is given a close range attack, and the logic of damage and death is the same as the first two enemies. The logic of victory in the game is that the character wins when he kills the last third enemy and finally exits the game.

3.4.5 Loop testing

If the game contains loop logic (such as repeated attacks), make sure that these loops work correctly under both normal and abnormal conditions.

We set repeated attacks for each enemy, and set their attack time. We introduced the time () function to specify their attack rate. Moreover, our attack elements were single, and the probability of loop errors was extremely low.

3.4.6 Error Handling Testing

Make sure the game handles all kinds of expected and unexpected errors properly, such as when the player enters invalid commands.

We just set up A few regular game buttons, ESC, W, A, S, D, BUTTON (in-game button), if you do not enter these buttons, the game itself does not produce any reaction

3.5 User Experience Test

- Invite real players to participate in the test and collect their feedback.

- Observe player behavior to understand how playable and engaging the game is.

We tested people who have not played such games and found that they have a poor understanding and acceptance of such games, and the completion rate: 23.4%

We have a high level of understanding and acceptance of games tested by people who have played these games but have not played our games, passing rate: 52.1:%

We tested the developers and found that the completion rate did not meet expectations, only achieved, the completion rate: 75%

In testing, we found that the game was very difficult. When we talked to the developers, we found that the design was originally designed to be difficult in the early stage, and then when it reached a high level, the difficulty was reduced. In order to give players a high sense of accomplishment, we designed a very difficult early work. We talked to our test players and decided to slightly reduce the initial difficulty to enhance the experience. So we solved the difficulty problem by increasing the number and decreasing the number of enemies, and then found that after changing the number

People who have not played such games:

23.4% ----- 30.7%

People who have played such games but not ours: 52.1% ------ 66.7%

Developer :75% -------- 100%

Stress test

** **Performance Test** ** : Check the performance of the game under high load, such as whether there is a delay or crash during rapid succession of attacks.

We chose to create extreme situations, such as adjusting the number of enemies to a very high level, such as 1000, which would produce a lot of animation, and here is the situation we tested: (Environment: corei7 win10)

100	No deadlock
500	No deadlock
1000	There is no deadlock
2000	No deadlock
5000	There is a delay
3000	There are delays
2500	Slightly stuck

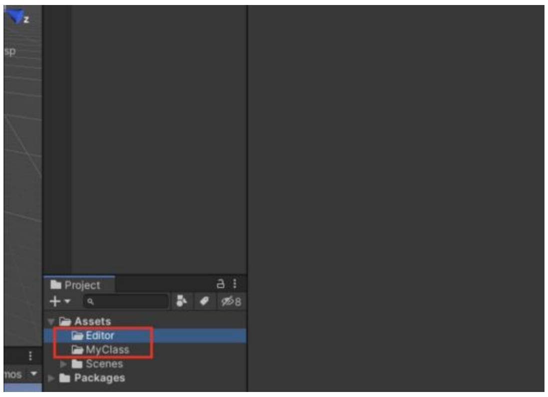
So we think the limit is around 2500, and because of the time we only test on the development platform computer, and because our game has a target of much less than 2500 enemies, we think it's extremely difficult to get stuck.

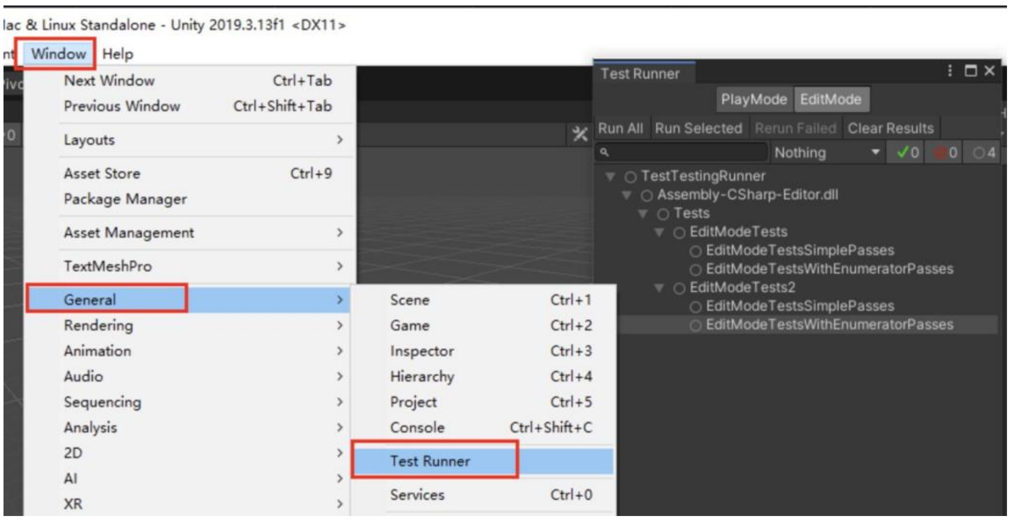
3.6 Automated testing

3.6.1 Unit test

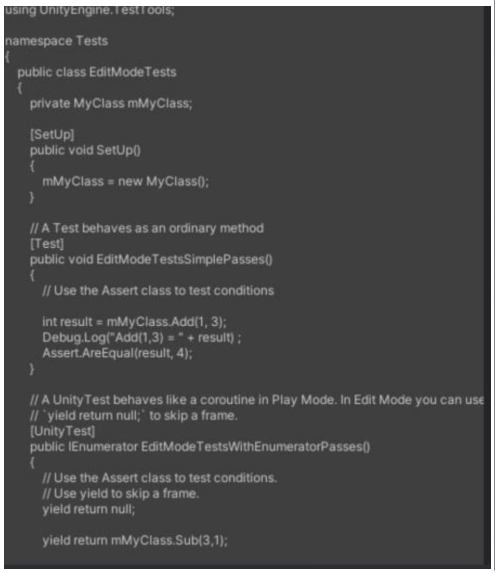
The Unity Test Framework (UTF) In "Play mode", you can see the test results directly in the unity scene. UTR is primarily used for unit testing

Create an Editor folder. Editor is used to write unit test code

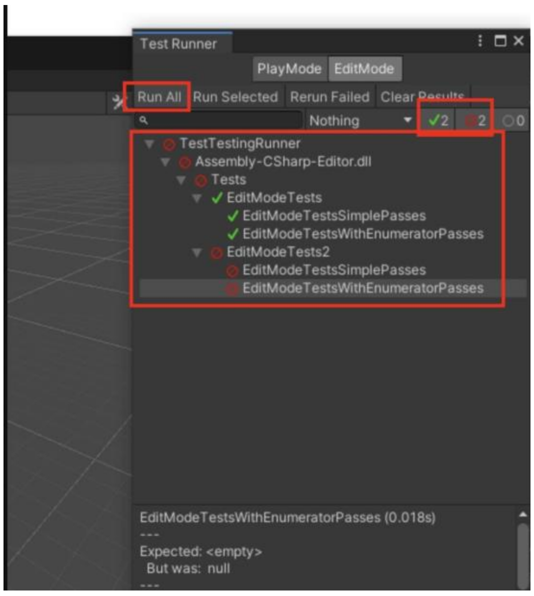




2.Write code, test the MyClass function, and intentionally Assert errors



```
using UnityEngine.TestTools;

namespace Tests
{
   public class EditModeTests
   {
      private MyClass mMyClass;

      [SetUp]
      public void SetUp()
      {
         mMyClass = new MyClass();
      }

      // A Test behaves as an ordinary method
      [Test]
      public void EditModeTestsSimplePasses()
      {
         // Use the Assert class to test conditions

         int result = mMyClass.Add(1, 3);
         Debug.Log("Add(1,3) = " + result) ;
         Assert.AreEqual(result, 4);
      }

      // A UnityTest behaves like a coroutine in Play Mode. In Edit Mode you can use
      // `yield return null;` to skip a frame.
      [UnityTest]
      public IEnumerator EditModeTestsWithEnumeratorPasses()
      {
         // Use the Assert class to test conditions.
         // Use yield to skip a frame.
         yield return null;

         yield return mMyClass.Sub(3,1);
```


Test Runner
PlayMode
EditMode
Run All
Run Selected
Rerun Failed
Clear Results
Nothing
✓2
2
0
TestTestingRunner
Assembly-CSharp-Editor.dll
Tests
EditModeTests
EditModeTestsSimplePasses
EditModeTestsWithEnumeratorPasses
EditModeTests2
EditModeTestsSimplePasses
EditModeTestsWithEnumeratorPasses

```
EditModeTestsWithEnumeratorPasses (0.018s)
---
Expected: <empty>
 But was:  null
---
```

3.6.2 Integration test

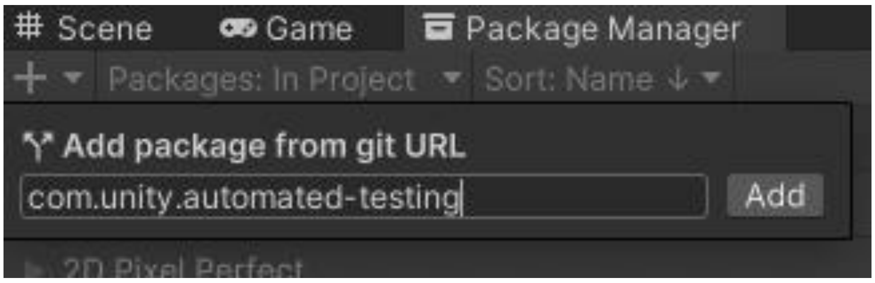
AltUnityTester,AltUnity tests have the ability to identify and interact with objects in a Unity scene

1. Open it in the Unity editor

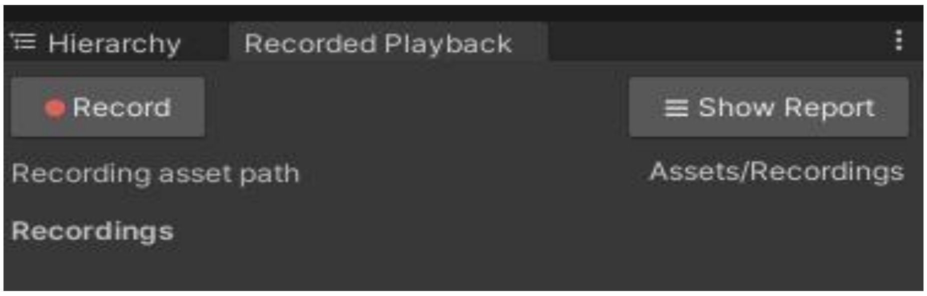
2. Package Manager (Window &gt; Package Manager)

Click (+) on the top left and select "Add package from git URL..."

3. Enter com.unity.automated-testing and click "Add" to add the package



4. After adding, open Recorded Playback (Automated QA &gt; Recorded Playback)



5. Open the scene that needs to be tested, click Record here, and then go to the user interface to operate according to the process that needs to be tested. It is recommended that the code be written in a standard point, and the place where the debug error is error.

6. After recording, just stop. The following recording result will be

generated, which can be named by itself. In essence, it is a json file

4 Maintenance Document

4.1 Perfect maintenance

Since the game we developed is a single-player game, it only runs on a specific computer, and the game data and process will only be saved on their own specific host, and will not be recorded and backed up through the network or other means, so we will simply maintain it.Make subtle improvements and optimizations to the game while keeping the core mechanics unchanged.

Implementation measures:

Regularly review and update the game's user interface to make sure it's friendly and intuitive. Continuously collect player feedback and make small adjustments to game mechanics based on feedback, such as balance adjustment, character ability optimization, etc. Optimize your game's load time and runtime efficiency to ensure it runs smoothly across a variety of hardware configurations.

4.2 Adaptive maintenance

To ensure the compatibility and performance of the game on different operating systems and hardware configurations, at present, we only support running the game on Windows operating system, among which Unity engine comes with translation files for us to release multi-platform games, so we will consider transferring our game to MacOS or Linux system in the later stage

The game is regularly updated to support new versions of the operating system, such as Windows 11, MacOS, etc.Optimize our game code to ensure optimal performance on different CPU and GPU configurations.

4.3 Preventive Software Maintenance

Prevent future performance and compatibility issues.

Regularly refactor the game code to improve its maintainability and readability. Implement rigorous quality control and code review processes to reduce potential defects and issues in the future. Technical documentation is updated to ensure that it reflects the current state of the game and the latest development standards.

4.4 Corrective Software Maintenance

Repair the defects and errors found in the development and use of the game, find the problems of the game itself through the feedback of the players, and actively modify the later stage.

Conduct regular game testing, including performance testing and compatibility testing, to ensure that all fixes work as intended.

Update the game to fix known security vulnerabilities and performance issues.

Monitoring and evaluation: Continuously monitor the game's performance metrics and user feedback to identify maintenance needs.

Planning and implementation: Based on the evaluation results, develop and implement maintenance plans to ensure the stable operation of the game and user satisfaction.

Testing and validation: Thorough testing of the game under maintenance, including functional testing and user experience testing, to ensure that all changes work as intended.

Documentation update: Timely update of relevant documentation, including technical documentation and user manuals, to reflect maintenance activities and software changes.