Existing codebase:

1. Design principles

SRP (Single responsibility principle)

Diagram

Description automatically generated

Above UML snippet shows that the code follows the Single responsibility principle. The YouWinBanner is a class which extends the Entity abstract class, which means the class only has the responsibility of modifying the YouWinBanner object. As for the benefit, it reduces the coupling between two classes. To be specific, in this way, the impact of code change will be reduced to other classes. Hence, the modification will be much easier.

Open-closed principle

Diagram

Description automatically generated

Above UML snippet shows that the code follows the Open-closed principle. All Entity classes are open for extension but closed for modification. The Platform, Slime and etc. are a kind of Entity object, but they have their own special functionalities. The code implements their features by the extension instead of modification. This will benefit its future development and maintenance.

Information expert

Diagram

Description automatically generated

Above UML snippet shows an information expert of GRASP. For example, LevelImp class involves all information which is required by heroIntersectsWithMushroom() method. Therefore, the heroIntersectsWithMushroom() method is stored in LevelImp class. Since the object can fulfill all method or task with its own information, it will reduce the connection between one class and another..

1. Design pattern

Factory design pattern

Diagram

Description automatically generated

This UML snippet shows how the code uses the factory design pattern to produce the FloorTile object. All objects of the FloorTile object are created by the floor tile factory. It reduces the level of coupling and improves the efficiency of producing objects that have complexed subclasses.

Builder design pattern

Diagram

Description automatically generated

Above diagram shows how the code uses the builder design pattern. It has two major benefits. First, this design pattern clearly separates the construction from the object. It makes the construction process controllable. In addition, it allows the programmer to make changes in the construction process of an object.

1. Documentation

Advantages:

This readme file is a good guideline for the player. It introduces the game story, controlling methods and features of the game so that the player can clearly understand how to play this game and what is the goal of this game.

Disadvantages:

There are two disadvantages of this readme file. Firstly, it doesn’t teach player how to start the game. As normal players, it is hard for them to run this game without any introduction. The second disadvantage is that the configuration of each level is involved in readme file. It violates the security of the coding.

1. Conclusion

In general, despite the above disadvantages, this implementation helps me to achieve my required functionalities rather easily. Firstly, it follows several design principles, such as single responsibility principle, open-closed principle and information expert. For the single responsibility principle, it makes each class has only one responsibility, which means I can locate the required functionality by searching in the responded class quickly. For open-closed principle, it makes the structure of class and its subclass very clear. Therefore, I can easily create my desired object. The information expert ensures that all the required information for a task will be included in the class. Therefore, I can simply use a class to do the required task without any extra work. Moreover, the code implements several useful design patterns. The factory design pattern is an efficient method to create object from complex subclasses. The builder design pattern provides a better control of construction process, which means I can access the construction process and even make changes in the construction.

Extension version:

1. Changes

Overall:

In the new version, I added three features for the game. There are levels, time and score, life, saving and loading.

Level:

For the level feature, the game has four levels now. Each level has a flag as the teleport point of the next level. The hero can move to the next level by touching the flag at the end of each level. In addition, touching the flag of the last level is the ultimate goal of the game.

Time and score:

Each level has a time limitation which is one hundred seconds. Peer second of remained time worth’s one point. For example, if the player completes the level with ninety seconds, he/she will get ten reward score. The game can record and display the player’s score at the left top corner now. There are two types of score. The first one is the score of current level. This score is a dynamic number, which depends on the remained time and the numbers of mushroom the hero collected. Another is the total score of the game, which is also displayed at the left top corner. When the hero touches the flag to the next level, the score of the current level will be added to the total score.

Life:

The hero has more than one life. The number of remained life will be displayed at the left top corner as well. When hero’s life equals 10, the game is over. Then, the picture of “game over” will be shown on the screen. After two seconds, the hero will respawn in the level1 with full health, and the score is refreshed to zero.

Saving and loading:

The game provides saving and loading functionalities for players. The player can save the game states by pressing the key “q” and loading the saved game states by pressing the key “s”.

1. Design principle

Open-closed principle

Diagram

Description automatically generated

In my extension working, I follow the open-closed principle. Above diagram is an example. It shows how I achieved the feature of showing the game over picture. In order to add this feature, I did not modify any existed class. Instead, I created a new class and extended the Entity abstract class. My objective is to reduce the impact of modification.

1. Design pattern

Observer design pattern

Diagram

Description automatically generated

I implemented the observer design pattern to display the score, life and time. There are three benefits of observer design pattern. The first benefit is low coupling. This design pattern is an efficient way to retrieve the information of subject with a low coupling. The second benefit is sending the information with less impact. The communication between the observer and the subject makes no impact on other classes. The last benefit is adding and removing the observer more flexibly. The observer can be easily added or removed by invoking the register() and unregister() method.

Memento design pattern

Diagram

Description automatically generated

I used memento pattern to implement saving and loading functionalities. As the memento pattern is an efficient way to store an object without compromising encapsulation, the game states can be saved and loaded easily.

1. Reflection

In this extension design, my code successfully implemented all required features such as the level, time, score, life, saving and loading. Thanks to the extensibility of the code, I was able to make further adjustments and improvements. As the given structure is very clear with OOP design principle, it is very easy for developers to extend the code with more features and functions. Accordingly, the player will have a more exciting experience of the game, as I added a timer.

However, I also encountered some outstanding issues during the process. The most outstanding issue is the coupling. When I first added some features, I found that the coupling became very high. In order to solve this issue, I implemented several design principles and design patterns to achieve a low coupling. For example, I used the observer pattern to replace the previous method. Although the coupling is lower than my first attempt, it is still higher than the original one. It does not achieve a perfect performance.