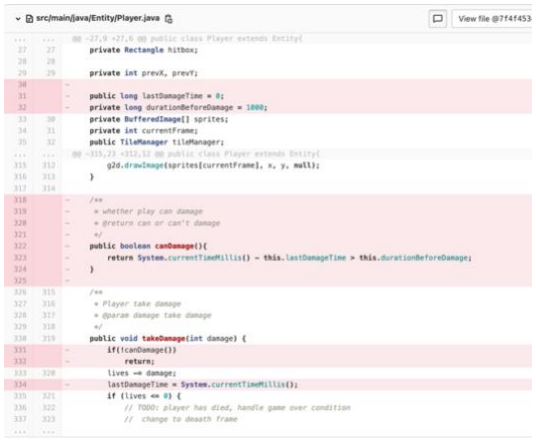
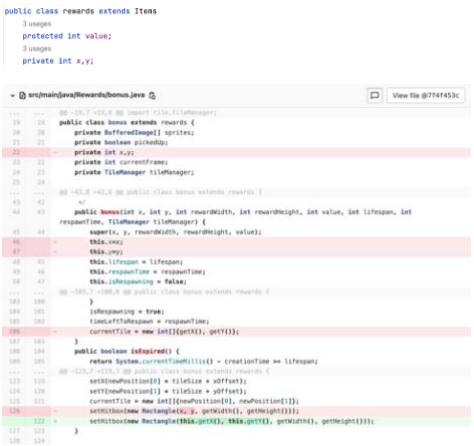
CMPT 276 Assignment 3
Group 14 Refactoring

Hanxi Chen (301417307)
Boyu Zhang (301414438)

1.Unused or Useless Variables.

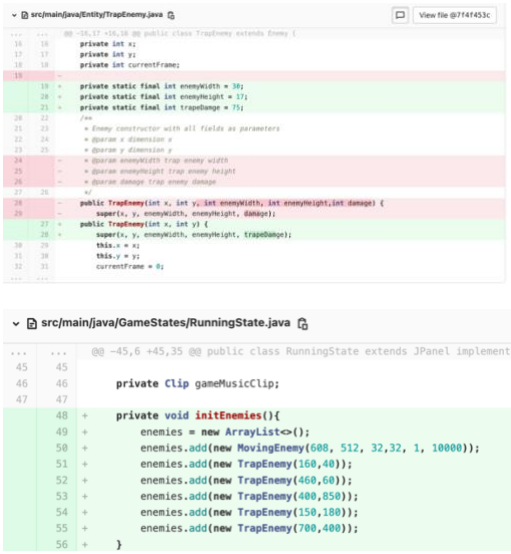


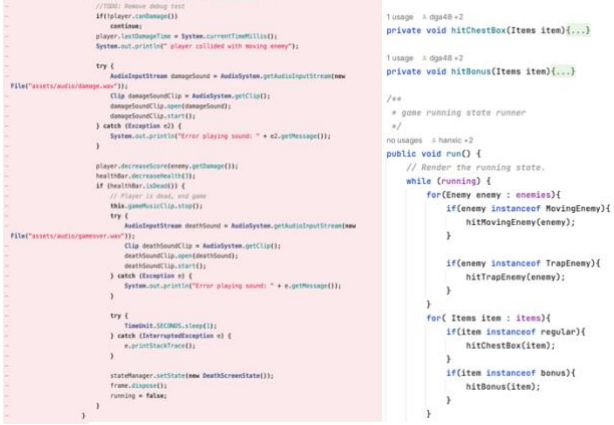
2.Code Duplication



3.Poorly Structure Code.

4.Long List of method parameters





Left: Previous

Right: After

5.Poorly Structure Code

6.Bad/Confusing Variable Name

```
return isTileSolid(playerLeft, playerTop)
        || isTileSolid(playerRight, playerTop)
        || isTileSolid(playerLeft, playerBottom)
        || isTileSolid(playerRight, playerBottom);
return isTileWall(playerLeft, playerTop)
        || isTileWall(playerRight, playerTop)
        || isTileWall(playerLeft, playerBottom)
        || isTileWall(playerRight, playerBottom);
```

Comment: The screenshots are from our GitLab web page of the project.

The red highlight represents code deletion, and green represents new (refactor) code.

More explanation on the next page

1. Unused or Useless Variables

   We set up a Player Class for the movable character (Player controlled) in the game. We originally planned to set fatal moving enemies and ordinary moving enemies, but we did not implement a variety of moving enemies during development. The deleted useless variables are designed to record whether the player can continue to receive damage after receiving damage. Currently in the game moving enemies that give damage as critical hits so these variables are no longer needed. So I delete the useless variables during refactoring.
2. Code Duplication

   The reward system of the game has an abstract “reward” class, and “bonus” rewards should inherit the characteristics of the “reward” class. But we additionally overwrite the x and y coordinates of “bonus” reward class that the “reward” class should have. (The coordinates are used to set the position of the item on the map). During refactoring I delete the variables we overwrite.
3. Poorly Structure Code

   When the game is running to detect the interaction between the player and the map elements, we originally stacked all the codes and all the conditions (cases) together in the running state class. After refactoring, the run method became concise and easy to understand, and all collisions Conditions are written as additional methods that will be called when necessary.
4. Long List of method parameters

   Our original trap enemy had too many parameters, which resulted in the need to enter many lines of repetitive code when creating it. Since we only have one kind of fixed trap in the game, I set the length, width and height variables for it before the trap's constructor. In this way, when the game is running, we only need to input the x and y axis coordinates as parameters when we want to generate the trap.
5. Poorly Structure Code

   In the running state class of our generated game, the original version has no structure and piles up all the code used to generate map elements. After the refactor, I created a separate method for the generator of each element. When it is necessary to change the generated element, I only need to go to the generator to change without adjusting the running class. At the same time the code becomes more understandable and manageable.
6. Bad/Confusing Variable Name

   In our initial map design, the game had a variety of terrain and barriers, but during the development process only walls were kept as obstacles. The map is composed of a circle of walls and has a door as an exit. The method originally designed to detect whether the tile where the obstacle is located can be collided has not been changed in time, resulting in the code review process to think that "IsTileSolid" includes both collision monster and trap conditions. "isTileWall" becomes more understandable after refactoring.