Phase 3 Report

LeTian Wang

Brendan Murray

Allen Chen

Scott Park

2.1 - Unit and Integration Tests

- **Features that need to be unit tested:**

  Key handling

  - Tested how the system would respond to keys being pressed and released in each game state. This included all of the movement keys and control keys in the play state, W,A,S,D and Pause functionality. Then in the title state the test made sure that we could navigate the menus and start or quit the game. In the finish state, tested the ability to successfully return to the main menu after completing the game by pressing enter.

  Sound

  - Check if sound is imported correctly and working properly when asked to play. Other tests covered looping the audio or stopping the audio clips successfully.

  Load Image from Resource

  - Load pixel png from resources and correctly link and scale each image
    - **E.x.** Player up1 image is linked to resources/player/main_character_up_1.png
    - **E.x.** make sure that when the file name is not correct, correct exception is being displayed

- Many different objects required a pixel png to be loaded such as the tiles, player and zombies. So testing this was very important

<u>LoadMap from Resource</u>

- Load and read a txt file that contains information about the structure of the maze and the tile type of each cell in the maze. This feature could potentially be expanded to support more maps so we tried to make the test general for potential future expansion.

<u>Constructing classes</u>

- We made sure that the constructors set each name properly and load the correct scaled image from the right resource directory. For some classes, more work had to be done to construct them, and we ensured that the fields were populated properly.

<u>Utility tool</u>

- Using utility function that get a bufferedimage and scale it to certain size and return the scaled version of bufferedimage

- **Important interactions between different components (Integration Testing):**

<u>Player Interaction</u>

- By using junit test, we made the player class to interact with gamepanel, zombie, and multiple other objects. And we made sure that the player is going the correct direction with corresponding key presses and the player is collecting rewards once collide with rewards, stop moving forward once collide with walls, take damage once touching zombie and trap, and stop taking damage once the player life reaches 0.

Enemy Chase

- We simulated cases where zombie would and would not detect the player to see if the pursuit behavior worked properly. To do this we checked if the zombie entity would start moving faster in the player's direction when the player entered the active chase range.

Simple Gameplay Test

- Used the Java Robot class to simulate a player playing the game. This test started with navigating the main menu to start the game. Then once in the game, the player would pause and then unpause the game. Finally, we set the game state to defeat and use the bot to navigate through the menu and close the game successfully.

2.1.1 - Test Automation

We use the maven project structure to test everything, and included junit api and engine in our pom xml file. In our readme.txt we included how someone can build compile and run our program with commands such as:

```
mvn compile
mvn clean install
Java -jar CMPT276_Group1-1.0-SNAPSHOT-jar-with-dependencies.jar
```

2.1.2 - Test Quality and Change

90%line coverage

41%branch coverage

- We did not cover the draw methods in unit testing due to the use of java.awt Graphics2D. We found that it was more than sufficient to test the draw methods indirectly via the other integration tests such as the gameplay test. Since these draw methods are called and updated frequently. Since we also tested loading the textures ahead of time we were sure that the draw method would have valid parameters.
- We tested certain methods to ensure that they did not throw exceptions under regular conditions. Other methods were tested to ensure they properly dealt with exceptions and did not throw them to the calling method.

2.1.3 - Findings

We learned that unit tests not only help us catch bugs, but also help us write code. In other words, it can help us design our code and ensure that it is not tightly coupled. For example, while testing the sound class, we realized that it would be much easier to work with and test if there were getter methods for some of the useful private members, such as the Clip class. Once these methods were added, it allowed us to test the code more easily and resulted in a less tightly coupled Sound class. After writing a few unit tests, we found some bugs in our program that would have been hard to spot otherwise. Our player class for example had a bug where after the player died to a zombie they would continue taking damage as if the game was still continuing, sending their health far into the negative. Another bug was with our pathfinding algorithm for the zombie, which was flawed in the steps it took to reach the player.