Phase 3 Report

Group Number: 5

Group Members:
Joshua Kim
Jad Alriyabi
Marco Lai
John Sweeney

Date of Submission: 03/4/2022

<u>Unit and Integration Tests:</u>

Although our program did run successfully, many of the features we implemented were highly dependent on two Classes: Game and Gameobject. Gameobject encompassed abstract methods that became hard to initiate during our unit tests; the features we decided to test were the following:

- Player Movement
- Collision
- Regular and Bonus Rewards
- Punishment
- Enemy Movement
- Menu
- Barriers
- , etc

In our Collison Testing, we included multiple interlinked features such as player movement colliding with game objects causing specific outcomes that can arise in our game. Collision and Rendering our window are what we deemed to be the most critical components of our program since once the Police animation is near the player, that Ui will prompt a Menu asking the user if they want to play again.

For the GUI integration tests, we tested that the GUI would launch and would be able to cycle through the different states associated with game states. We focused on loading every sound available to the player for the Sound integration tests to ensure that each file loads appropriately. Moreover, for the Image integration tests, we tested that the image asset would properly load for the game user.

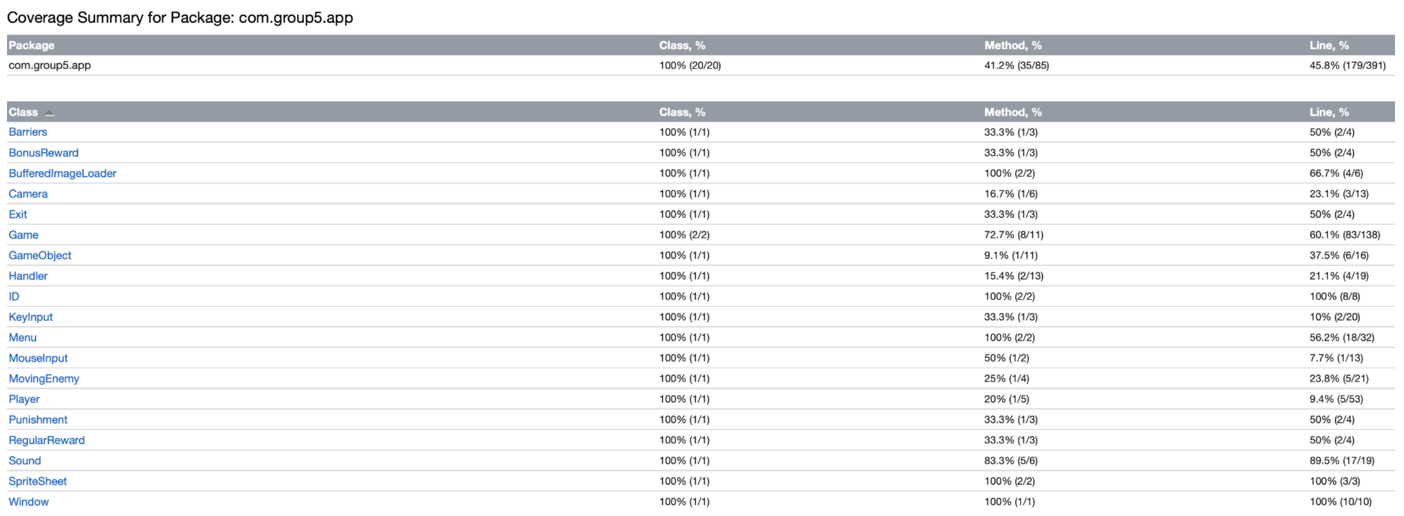
<u>Test Automation:</u>

As described in the requirements, these tests were created using the JUnit framework. Initially, getting familiar with the framework was our main concern as this was very new to us. We studied how to create unit and integration tests through lectures, and online videos, and helped each other understand through our regular meetings. We made sure the tests were following the Maven principles and allowing anyone to automatically compile and run the tests. The testing made us realize that our code wasn't very well designed and refactoring was an appealing option. We encountered difficulties as we were testing due to the high coupling present with many of the classes. This resulted in big and long modifications as one component would affect another and reveal bugs in the code.

Test Quality and Coverage:

During the beginning stages of the phase, quite a bit of manual testing was done to find bugs and check the implementation of certain scenarios. This was a good initial step in checking for early bugs and possible fixes. For example, we tested whether the buttons would work when the buttons were not present. When we manually tested by clicking the invisible button, the button functionality was still working and hence found a bug.

The tests included in our project are those that are critical to the basic functions of the program. For example, functions that ensure that sprites in the game are onscreen, the game terminates when required, and that the game ends when the player has exhausted their health. The only notable tests that we did not complete are ones that are immediately evident or have no actual basis in verification. These include the exit function, since it is obvious that this has occurred and there is no real way to perform a test once the program has terminated. We also don't test Punishments, since the GUI clearly displays to the user when points have been deducted.



Coverage Summary for Package: com.group5.app

Package	Class, %	Method, %	Line, %
com.group5.app	100% (20/20)	41.2% (35/85)	45.8% (179/391)

Class	Class, %	Method, %	Line, %
Barriers	100% (1/1)	33.3% (1/3)	50% (2/4)
BonusReward	100% (1/1)	33.3% (1/3)	50% (2/4)
BufferedImageLoader	100% (1/1)	100% (2/2)	66.7% (4/6)
Camera	100% (1/1)	16.7% (1/6)	23.1% (3/13)
Exit	100% (1/1)	33.3% (1/3)	50% (2/4)
Game	100% (2/2)	72.7% (8/11)	60.1% (83/138)
GameObject	100% (1/1)	9.1% (1/11)	37.5% (6/16)
Handler	100% (1/1)	15.4% (2/13)	21.1% (4/19)
ID	100% (1/1)	100% (2/2)	100% (8/8)
KeyInput	100% (1/1)	33.3% (1/3)	10% (2/20)
Menu	100% (1/1)	100% (2/2)	56.2% (18/32)
MouseInput	100% (1/1)	50% (1/2)	7.7% (1/13)
MovingEnemy	100% (1/1)	25% (1/4)	23.8% (5/21)
Player	100% (1/1)	20% (1/5)	9.4% (5/53)
Punishment	100% (1/1)	33.3% (1/3)	50% (2/4)
RegularReward	100% (1/1)	33.3% (1/3)	50% (2/4)
Sound	100% (1/1)	83.3% (5/6)	89.5% (17/19)
SpriteSheet	100% (1/1)	100% (2/2)	100% (3/3)
Window	100% (1/1)	100% (1/1)	100% (10/10)

Findings:

When we ran the test cases, we realized that the game state variables such as coins collected count and health are directly related to the player class; we intended to move such variables to the player class instead of storing them in the main class. However, after consideration, we excluded the idea because such variables are also called from the main and menu class. Calling variables from the game object class might cause coupling, so we intended to keep our main class looking clean as it already involves most of the code. Since we are getting close to the deadline, we decided to avoid refactoring to minimize the probability of an error happening.

We did not have any structural changes to our main code during the testing phase, and we also implemented some add-ons functions. We added functions that play background music while the game is running and sound effects when the player collects regular and bonus rewards. We also modified and upgraded our UI; we have changed the font and font size and the coordinates, which has a better UI appearance. We added the health bar to the player, which goes down when the player hits the punishment game objects. A timer was also implemented to be precise as the nearest hundredth. Also, a pause function to the game can now be paused by pressing ESC.

The biggest challenge throughout the project is the implementation of the enemy class. We applied the same method to handle collision as implementing the player class into the enemy class. However, we keep getting unexpected movement results. The enemy would bounce out of bounds sometimes when it hits the boundary barriers but not barriers in the middle of the game board even though they share the same properties. During the testing phase, we tried to fix the error by testing the collision method under different conditions, substituting different velocity values, and relocating code fragments separately; unfortunately, we could not determine the proper solution.

While making the tests, we encountered some bugs. In particular, some bugs may have been created by multiple people working on a class simultaneously. For example, the player implements health and score from the game class, which resulted in errors when creating game objects during our testing phase.