Phase 3 Report

GitLab IDs:

- Ryan - rat2
- Jason - xgill15x / jga132
- Luan - luan_nguyen_3@sfu.ca
- Zihao - zxa43

Unit and Integration Tests

Unit Tests:

- Key Input - We simulated key presses to see if they would match the direction that the player is facing and made sure that the player would not move if a key is not being held down.

- Player Images - After initializing the player, we tested to see if the images were imported properly. We used assertNotNull to verify that the images were being read.

- Player Score - Once the player's score falls below 0, the game is over. We used assertEquals to verify that the game was in the game over state.

- Player Update - When the player updates, it should move in the direction it is head when a key is being pressed and the sprite should alternate. We used the

- Player Reset - We added a feature where the user could restart the game once it ends. We used assertEquals on the player's x and y coordinates to see if the player was moved to the starting position correctly.

- Enemy Random Movement - We made it so that the enemies would move in a random direction when not in the range of the player. This feature has a 2-second cooldown for when the enemies are able to change directions. We used assertEquals to see if the cooldown was reset when the alien changed directions.

- Enemy Follows Player - We made it so that the enemies would follow the player once it got too close. We used assertTrue to test that this state was activated once the player was in range.

- Enemy Collides with Player - If an enemy touches a player, the game is automatically over. If collision is detected, the game should be put into the game over state and the player's score should fall below 0.

- Pathfinder AI - We created a class that the enemies used to find the shortest path to the player. We needed to test that each map tile was being converted into a node that the class could use to calculate distance.

Integration Tests

- GamePanel State Test - Test if it is game over or if the game is paused

- GamePanel Reward Test - Test if rewards are being placed correctly on the map

- Collision Tests - Tests for collision with tiles, between the player and an enemy, the player and rewards, and collision between the player's projectile and an enemy.

Test Quality and Coverage

Measures Taken for Test Quality

When testing, we made to cover all the methods in our code to make sure that they were executing the expected behaviour. To do this, we wrote tests for individual methods and compared the actual output with the expected ones. While running the game to ensure that there were no game-breaking bugs, we reran the tests if we added anything new to the code. This further ensures test quality as it verifies that our tests still pass after any code changes. We also followed the Testing Pyramid by testing if our units behaved properly before moving on to interactions between multiple objects.

Coverage

AI - 72% line coverage, 100% class coverage, 87% method coverage
Entity - 88% line coverage, 100% class coverage, 85% method coverage
Game - 78% line coverage, 62% class coverage, 66% method coverage

The methods that were not tested involved the HUD and sprites of the objects. The HUD did not need testing as it only displayed the parts collected and the player's score, which were already tested. The sprites did not need testing, as the game was running fine, but it would not run at all if there was anything wrong with importing them.

Findings

What we Learned

We learned that testing the methods individually did not completely ensure that there were no errors, as that did not guarantee that they would behave properly when used together. By using the Testing Pyramid in our testing process, we were able to create better quality tests, as it ensured that the main components of our game were functioning properly before moving on to the interactions between those main components. We also learned that testing and debugging code was a long and rigorous process, as we discovered that certain aspects of our game were not actually behaving well in all the edge cases.

Improvements and Changes to our Code

Due to having different operating systems, we discovered that our code did not work universally and that certain methods had to be rewritten to work on both MacOS and Windows. We also improved the efficiency of our code by finding redundant code or unused methods and removing them. Another issue was that the hitbox of our player was too large, and would make it difficult for the player to pass through paths that were only one tile wide. Additionally, we found that our inanimate objects were too reliant on the methods that we wrote for our animated entities, and the majority of the attributes and methods that were being extended were not being used. We instead wrote a separate class that had attributes and methods that were specifically designed for those inanimate objects. Lastly, we also added more levels to improve the replayability of our game.

Bugs That we Fixed

One bug that we discovered was that the enemies would get stuck if the path to the player was only one tile wide, which caused the enemies to stay in place when they would try to follow the player. Another bug that involved the enemies is that, if there was already an enemy in one of the tiles that they would spawn in and the game tried to spawn another one, the two enemies would get stuck on that spawn tile. To fix these two bugs, we removed collision between enemies. Furthermore, we also found a bug with reading the high score from the text file. If the text file did not already exist, the game would compile, but it would crash on the title screen because it was trying to read a null file. This was fixed by checking to see if the file existed and, if not, create the file with a default high score of 0. Creating test methods did not fully get rid of our bugs, as the first two bugs with the enemies were discovered by running the game, and not

through our test methods. Thus, as discussed in lectures, we learned that the individual testing of methods did not guarantee that there were no bugs present.