Phase 3 Report

Sahil Janjua, Antonio Benzan, Daniel Dai, Matthew J Maxwell

In phase three, we took our final steps in the development of this project with the introduction of testing, debugging and refactoring. We started off by choosing what methods would require testing, and which methods would not, this was done by following the material learned in class, and by singling out individual features that would need to be tested. To start, we eliminated all the methods that handled the graphical interface, private methods, and smaller methods that were a part of bigger methods. The methods that were tested in the character package were movement, position, and bounds checking in the entity class, different variations of the update method in the guard class, and specific update methods, depth method, and three situations where we use the score method in the thief class. In the map package, we had tests for the enter effect method in the coin class, the mugging class and the guard station class, tests for loading coins, guards, guard stations, walls and map dimensions for the level loader class, a correct wall placement test for the room class and tile generator class, enter effect, a size getter, position method and remaining coins method for the tile map class, and several loading methods for the random maze generator class. And finally, in the game package, we have the tests for open, close, add score, set thief, and game won methods that all reside in the game class. The remaining public methods that were not tested, are getters and setters that are used in the methods listed above, as those methods played a large part in the methods listed. Several constructors, our main position class, and the direction class were also not tested, as it would be redundant since they are implemented in almost every method in this program, and if they were not working, none of the other methods would work as well. Also the professor had indicated that she wanted us to not add unneeded tests and keep the testing as concise as possible.

When we started our testing, one of the questions we had was how to handle the testing around the graphical components of the program, we decided on examining the interactions between the game's logic and user interface, we tried testing these methods that held these interactions as a whole, however, anytime we tried testing the graphics, the system would crash making it impossible to test it as a whole. We tried adjusting the dependencies to be able to achieve our goal of testing the UI, but that also failed. We were advised by the professor to not test the graphics, and focus on the logic and key components from the methods that focused on the aforementioned interactions. One interaction we found was the update UI position methods, these methods were used to make sure that the coins, characters, guard stations and any other moving parts would change position depending on the circumstances in the game. To test this, we broke this down into several different components and tested the logic associated with this to see if it accomplished its purpose. All the update methods were also included in the standard play class we created, where update was called for the moving parts of the game, if these components weren't working, the game would not function as we wanted, however, after playing the game several times, we can confirm that the UI is working as planned. Another example was the draw initial method that exists in the map class, this method initializes local image variables, this was a method that we had a lot of trouble developing tests for, so we met with the instructor who told us to not develop any tests for this. We also took a look at the logic involved with deciding on which ending screen to show when the player rather loses, finishes the game with no coins or finishes the game with coins. We also tested the logic associated with this one by creating a won game test that created a game and checked if it was true that the thief won when he exited the maze with enough coins. We also played the game several times to make sure that all end game scenarios were correctly done and all the right ending screens showed up. To control the players movement in the game, we also included event handlers that would take inputs from the user to work within the game. These were simpler to test compared to all the user interface methods. We used the update after move tests in the thief test class, and the move tests in the entity class to make sure the event handlers worked as planned.

When we were writing our tests, everyone in our group wanted to ensure that we would write tests that would cover almost all possible outcomes so that we could find any bugs, and better optimize our code. We made a point not to write too many test cases as we wanted to write high quality test code, instead of high quantity. To ensure the quality was to our expectations, we would hold meetings on discord often so that we could discuss strategies we wanted to take in testing. We started off by dividing the responsibilities by who did what in the packages, since Sahil wrote all the methods in the character package, he wrote the tests for the methods in character, Matthew wrote the methods for the map package so he wrote the tests for a majority of the map methods, Antonio did the game package, so he wrote the game tests, and Daniel helped Matthew write tests for the map methods due to the size of the map package and his involvement in writing code for that package. While this was mainly how we worked, there were several methods in every package that had overlap with other methods from different packages, to make sure that these tests were implemented correctly, members of the group would privately message each other, and discord to do pair programming for those tests. Through the collaborations on discord, members would also critique other members test code to offer certain situations that may show bugs in the code, and offer better ways to write certain tests code so it can better cover different scenarios of the game.

Another way we tried to ensure quality, was by focusing on code coverage. Members of our group had access to the Intelli J IDE which provides its users with tools to measure line and branch coverage. However, when we started following the notes provided in class to calculate coverage, we realized that calculating coverage for our code was difficult due to the way it was written. Luckily using the tool, we were given a rough idea of how our coverage was. When reading our code coverage, keep in mind, that the tool also looks over GUI and private methods associated with the test we made, and we were informed that we were not supposed to test GUI and private methods, so the coverage will most likely be a few percent higher than these approximations. In the character class, we were able to achieve 100% branch coverage, and 42% line coverage for the entity class, 40% branch coverage, and 26% line coverage for guard, and 80% branch coverage and 50% line coverage for thief. Our game test that tested methods in the game package was able to achieve 100% branch and line coverage for direction and position, and 58% branch and 28% line coverage for game. In the map package, we were able to achieve 42% branch and 20% line coverage for the coin class, 60% branch and 47% line coverage for guard station, 100% line and branch coverage for level loader, tile generator and random maze generator classes, 40% branch and 29% line coverage for mugging, 37% branch and line coverage for room, and 66% branch and 62% line coverage for the tile map test. Looking at the percentages we were able to obtain for the code coverage, our tests were able to cover a good amount of the potential outcomes of the game.

Overall, we are very proud of the test code we have written for our project, and we believe that we have created tests that are concise and cover almost all the scenarios that could happen in the game. Before phase three, none of the group members had any experience in writing test code for programs this big, or even writing tests in a group setting, with this experience, we believe we have better understood how to separate responsibilities while working together to optimize our results. While we are grateful of the opportunity provided to write tests for a project like this, all group members agreed that writing test code for a game that has a graphical interface, and a game set up like this was not only difficult, but unnecessary, but these are only our personal opinions, we can still see the positives of doing this something like this in a classroom and workplace environment. With the testing, we were able to fix key bugs we found in the code. This included fixing portions of the map generator that didn't always have the enter and exit maze locations at the correct spot, fixed bugs that had guards and coins not spawning on the map when they should be present, and bugs where the guard didn't chase the thief, or sometimes moved directly onto the thief when it was quite a bit distance away from the player. Fixing all these bugs has helped us in providing our users with a much smoother and more seamless gameplay experience.