Phase Three Report

Most important test files created for each aspect of the game :

During the course of this project all of our group member had some ideas about what in particular they wanted to test when it came time for testing. Most of us mainly dealt with the classes that we wrote the code for. In the end we ended up with lots of test files which resulted in a very through testing of our game. Bellow are a list of some of the test files we wrote, what they were each for and why we decided to write that particular test file for our game.

1) Character class:

1) **characters-test**

The first set of test files written was for the character class. This consists of the test file characters-test in which we tested the functionality of our program when the players, both the main player and the enemies were blocked by various objects, like the escape helicopter. Static and dynamic obstacles and so on.

2) **enemy-generator-test**

The second test file written for the character class, was specifically for the enemy generator class, hear we tested the functionality of our program when it came to enemy generation. The number and position of the enemies made. We mainly focused on whether or not the correct number of enemies were created and whether or not they were placed in the correct part of the map(the roads).

3) **enemy-test**

The third test file for our character class was the enemy-test file, in this we tested for correct behaviour of the enemies after they were created. Examples of this include whether or not the enemies are able to follow after the main character in all directions that are not blocked by any dynamic or static obstacle.

4) **player-test**

The final test file for the characters class was to do with the player of our game, which is the main character controlled by the user. The test scenarios for this section of our code includes but is not limited to: testing if the player was created,

Phase Three Report

and whether or not it was positioned in a proper location of the game (the roads),we also tested for other important aspects like if the player could move in all open directions not blocked by any static or dynamic characters. Further tests included the behaviour of the main character after arriving at the escape helicopter with various amount of fuel and reward collected. Which is on of the necessary winning conditions of our game. Another important aspect we tested for was how the character would behave when close to any enemies and finally when colliding with enemies.

2) Maze-game class:

1) **game-timer-test**

one of the most important test files we wrote for the maze game class, was to do with the game timer. Here we tested using assertions for a variety of things from whether the timer was creates successfully or not to whether or not the score was calculated correctly. From the remaining time in our game. The most significant of these however was the behaviour of our game when the timer finished counting down and we were out of time. In this case we tested whether our game would terminate due to the time running out.

2) **level-generator-test**

The second most important thing we tested for in this class was, the level generator functions. Here we tested for the game level being created successfully. For example, the map should not be created with all the squares(game panel blocks) as barriers , or conversely using all roads and no barriers. This was one of the most important tests as other functions of the game depend heavily on this.

3) Objects class:

1) **grass-barriers-test**

The most note worthy test file for our objects class, would be the one we wrote for testing the grass barriers which is a form of static barrier in our game. We tested for whether or not the grass barriers gets generated properly and put in the correct locations in the map. This is very important as improper placement of the grass barriers will result in an unwinnable game.

Phase Three Report

4) Rewards and punishments class:

1) **punishment-generator-test**

The first test file for our rewards and punishments class is the punishment generator file, which testes for using assertions whether or not all the punishments for the characters are generated properly on the map. The punishments will make our main character loose points resulting in a lower over all score. Once again the main aspect tested was whether or not the correct number of punishments would be generated and if their locations would be properly placed(not all on top of each other).

2) **rewards-generator-test**

The second test file had to do with the rewards generator, were we tested for essentially the same thing as the punishment generator. We tested for the correct number of rewards and bonus rewards being created and checked for position overlaps, to confirm that no rewards were generated on the same location of the map, creating overlapping rewards.

5) Other features tested:

1) **winning-condition-test**

In this class we created another two test files that had to do with extra integration tests that needed to be included but did not explicitly fit in any of the other categories. The first one of the above test was checking the winning condition of our game after all the rewards have been collected and we have arrived at the escape helicopter ready to depart (this is how the game is won for our version of the maze game).

2) **keyboard-movement-test**

The second of these extra integration tests was to check whether or not the main character/ player would be able to be controlled by the keyboard. In this section we used assertions to make sure that there was player movement once a permissible keyboard key was pressed. That is When the character is not blocked by any static or dynamic obstacles.

Phase Three Report

Note:

We included many more unit tests for our maze game, however for briefness only the most important ones were mentioned above. Our general rule of thumb when deciding what to spent time writing the test files for was anything that was not trivially simple or non significant in terms of the overall positive or adverse effects it could have on our program during execution.

Important interactions between classes that we tested for:

In our game there are many important interactions between the classes as virtually all the classes are reliant on other classes . Or will need to work with other classes in some way. However, By far the most important interaction between two or more classes in our game, would be the one between the level generator and the enemy and main player generator. Both of which were tested indirectly by testing the enemy and player generator class which needs to interact with the level generator class. Meaning if the tests are successful that means that our interactions are working properly. This is also included in the section above where we discussed the important unit tests that we wrote for our game.

Measures taken to ensure the quality of our test cases:

As mentioned above all of our group members had a fairly good understanding of the things that they wanted to test for since the second phase of the project when we assigned classes to different group members. With that said, we employed certain measures while writing test files to ensure high quality test files. The first one of the said measures we employed to ensure high quality test files, was to make the testing with in each class as random as possible so that we did not miss anything because of designer bias. The second of these measures was to refractor test code as well as our source code whenever we needed to improve the compatibility of that section of code. Furthermore, we tested all aspects of certain classes not leaving anything untested. In this way it could be said that certain areas/classes of our code had 100% line coverage an well as branch coverage. However, overall we do not have a 100% coverage which will be discussed later in the report. Moreover for some test classes we decided to use MC/DC which gave us much more reassurance in term of our code being robust as MC/DC is more suitable for covering all possibilities of variables.

Phase Three Report

Features and code segments not covered by tests:

In our group we tested all the classes that we had written for phase two and took the time to ensure all the aspects of all the classes were accounted for in terms of testing. With that being said there were part of our code that we did not test explicitly as it would have been a waist of time and resources, as other parts of our code that would have been dependent of the untested segments worked perfectly fine. Furthermore, we did not test any external libraries and dependencies used . We only tested the code written by our group. Moreover, very simple methods that we did not deem fit to be tested were also left out. This would include almost all of our getters and setters, as they are simple enough where writing tests for them would not be a productive use of time. All this goes to say that only the key parts of our code was tested. For the areas that we did test however we made sure to cover all bases and write good comprehensive tests.

Branch and line coverage of our maze game:

Like mentioned above since some areas of are code was not tested, because we deemed it unlikely to fail or not worth spending time to write test for, we will not have perfect test coverage. Using intellij's built in tools we found that the line coverage for our code is around 90% and the branch coverage is around 83%, this agrees with the approximation that our group made by hand to calculate both the line coverage and the branch coverage.

Things learned from writing and running test files:

Although this phase of the project was somewhat less eventful then the last we still had ample opportunity to learn new things relating to testing, the main lesson here was compatibility and coupling of code , which resulted in us doing some minor refactoring. The minor problem we were having was that because some parts of out code were more dependent on other classes then they should have been, it made it difficult to write test cases for. Therefore we had to simplify things a little bit before writing some of our test cases. The main lesson being that we should pay more attention to coupling and compatibility from the start so that writing tests is a bit more straightforward when it come to it. To be more specific we made some small changes to the character class of our game for the reasons specified above.