Software Testing Report

Summary of Testing

We decided that we would follow a test driven development approach, this allowed us to refactor and add code confident we did not change or break functionality. This approach fit well into our overall scrum development approach, it meant that team members could work more autonomously as they could test if their code integrated well without the need of other members. Following Sommervile [1], our testing process had two main goals, to demonstrate requirements have been met and to discover defects in the software. When we started initial planning and designing our tests, we decided it would be important to establish how we would establish a ranking of their importance. This ranking would allow us to decide how to prioritize as we have very limited time and it would take too long to test everything. The most important features contributed to core functionality and main requirements, eg. GameFlow features and character movement.

We decided on using a variety of different testing methods, including black box, white box, dynamic and static testing. Our white box testing was comprised full of Junit test cases, these were used for testing many features and classes however we felt that some features could be tested without. For each of our features and requirements we evaluated how appropriate each of the testing methods would be. LIBGDX gave us some issues regarding this as there were some methods and classes we wanted to unit test but could not as many sections depended on textures that wouldn't load in the unit tests. This resulted in a lot more black box testing that we would have liked as black box testing can be less reliable and takes much longer than unit testing. However we made sure they would be as reliable as possible by testing many features many times in edge case scenarios.

Unit testing is very appropriate for our stage of implementation. We are quickly iterating so this allows tests to be performed quickly after project members make changes. This makes sure no dependencies have been broken when code has been changed. As a result, problems that break previous code can quickly be removed. Our black Box testing method is also appropriate as it allows us to visualize the problem using our graphical version of the game. However, this method takes much longer than unit testing so isn't appropriate to use as regularly.

Brief Testing Report

All tests referenced can be found in our Testing material document [2]. The tests that have failed have mostly done so because we have not implemented certain features or classes. Test T4.1 failed as we have not yet implemented a way to guarantee player safety after respawn. To pass this test we need to implement some kind of timed invincibility, similar to the shield item, on spawning. Test T9.0 and T9.1 failed as we have only implemented 2 out of the required 3 player classes. To pass 9.0 we need to implement a third class, and to pass 9.1 this class must be a correctly functioning computer science student. T10.0 fails as we have not added a mini game. This can be fixed by adding a mini game, we didn't add one this assessment due to lack of time. T11 failed as we only have 3 locations currently. This is because we did not have time to add in more that 3 locations. We can fix this by adding

more buildings that users can enter. T12 failed as we have not yet implemented bosses,T13 fails for the same reason as we only have one enemy type. This can be fixed by implementing 2 boss classes. T1.0 passed and is complete as it tests all possible interactions a user could have with each of the power ups. T2.1 passed, however it is possible that there are edge cases that could have been missed however this is very unlikely as we have done many tests, each testing it has passed. T3.0,T3.1 and 4.0 passed, similar to the previous test there could be edge cases where these fails, however it is very unlikely as we have tested this many time over many scenarios.
None of our unit test failed. HealthItemTest tested for when a player had been damaged and that it resorted the correct amount of health, it also tested to make sure that it did not increase the health above the maximum health. SpeedItemTest made sure that the players speed was increased by the correct value. Which is complete as if this passes it achieves the result we want from a speedItem.

[1] I. Sommerville , Software Engineering, edition: 9, pp. 206, available: https://edisciplinas.usp.br/pluginfile.php/2150022/mod_resource/content/1/1429431793.203Software%20Engineering%20by%20Somerville.pdf
[2] URL for Testing Material: https://lloydbanner.github.io/SEPR-Team-7/Testing2.pdf