Software Testing Report

Methodology

During the development of the system, we employed various different testing methods in order to ensure that the system would meet its requirements and so there would be as few bugs as possible present. The testing approaches used were Unit Testing, White-box Testing and Acceptance Testing. Owing to the Agile approach taken during development, all testing was done incrementally.

The unit tests were implemented using JUnit. Using Java as the main body of our code, JUnit was a clear choice for performing tests to ensure the program was behaving how the programmers were expecting it to. They also allow quick tests to be run, without requiring the front-end of the product to be complete, allowing the back-end to be developed correctly. JUnit can also produce formal testing results, allowing the team to provide proof of their product's capabilities.

The white-box testing also came in the form of the programmers running the system periodically after making small changes in order to ensure both that the changes had not introduced any bugs and that the changes functioned as expected. This testing method was appropriate as it caused the programmers to face any programming problems as soon as they occurred, preventing any bugs to go unfound until later in the project development. This testing approach would not be sufficient to inspect the program's behaviour alone, but combined with the JUnit tests, the program can be tested at both the front-end and the back-end.

The acceptance testing came in the form of ensuring that the system fulfilled the fit criteria associated with the core requirements of the system. The system was repeatedly compared to these criteria in order to continually direct the development, and ensure the project's requirements were fulfilled.

Test Report

Unit Testing:

All of the unit testing results can be found in the url found at the bottom of the report. As you will see, it consists of multiple tables and these tables follow a certain format. Each table corresponds to a different class being unit tested, and each row corresponds to a different testing function. The columns of each table are "Test ID", "Name of Test Function", "Function(s) tested", "Function use", "Result" and "Details". The reason we chose to use this selection is we felt that the tests should be as replicable as possible given the same source code. Each test was given an ID to allow for referencing easily in literature, the test function is named to identify what we used to perform the test, the functions being tested are named in order to remove any ambiguity about what is being tested, the function use is stated to inform anyone who is not familiar with the code what the function being tested should do, the

result is of course reported to show whether the test failed or passed and the details allows us to go into more detail about how exactly the test function performs it's testing.

In its current incarnation, the system passes all of the unit tests. The unit tests were implemented using JUnit and were created and run in a white-box fashion, being created to ensure that the code was producing the expected results. This initially proved to be difficult as it was hard to run the libGDX code without needing the whole system to be running, but this was eventually circumvented using Mockito to simulate the graphics functions.

The classes that were tested in this way were ImageActor, AnimatedActor, MovingActor, Entity and NPC. With the exception of the functions in MovingActor, all of the functions that are tested are done so individually. The reason for MovingActor's tests being distinct in this regard is that the only change that MovingActor makes to AnimatedActor is effectively implementing acceleration, velocity and angular velocity and as these must work together to update the actor's positioning, it made sense they should be tested together.

White-box testing:
As stated earlier, this was done by the programmers after making small changes in order to ensure that the code worked as intended without bugs. Whenever one of these tests failed, a debugger would be run and then, once found, the erroneous code was modified and the test rerun. Due to the rather informal nature of this testing, it is hard to provide any statistics.

This style of testing was of particular importance in the development in our system as our system was developed using an Agile framework, and so favours the approach of making small changes to then run small tests.

Black-box testing:

All of the black-box testing results can be found in the url at the bottom of the report. The test data is presented in a single table, in which each row corresponds to a different test. The columns of the table are "Test ID", "Test", "Relevant Requirements", "Expected Result", "Actual Result", "Overall Result" and "Evidence". The reason we chose to use this selection is we felt that the tests should be as replicable as possible given the same source code. Each test was given an ID to allow for referencing easily in literature, the test is given a sentence long description that sums up what it involves, the test is then related to requirements from the requirements document in order to state the necessity and the relevance of the test, the expected result and actual result are then presented for comparison, the overall result is then marked as a pass or a fail and finally we present our evidence that the test was passed or an explanation for why it was not passed.

In total, there were 23 different black-box tests. Of these 23 tests, 19 passed and 4 failed. The 4 that failed were BB_SafeGain, BB_MiniGain, BB_AvoidGain and BB_Mini. The reason that BB_Mini and BB_MiniGain failed was due to the fact that the minigame has not yet been added to the game as this is due to be done at a later stage. BB_AvoidGain has also failed because of the thought that it should be fulfilled further down the line, but it is worth noting that BB_SafeGain was expected to pass. It appears that the ability to gain points when

accessing a new safe area has encountered some sort of bug, which has lead to it not working.

Acceptance Testing:

All of the black-box testing results can be found in the url at the bottom of the report. The test data is presented in multiple tables, in which each table corresponds to one of the classifications of requirements (Functional, Non-Functional or Constraint) and each row corresponds to a different test. The columns of the table are “Test ID”, “Requirement ID”, “Fit criterion”, “Result” and “Evidence”. We chose this format in order to make the tests as unambiguous as we can make them. Each test was given an ID to allow for referencing easily in literature, it is then given the ID for the requirement it is directly testing, the fit criterion is then stated in order to save the reader having to look in the requirements document to read the fit criterion and also to clearly state what the test was looking at, the result is then given as either a pass, partial pass or fail and finally a statement of evidence is given in the case of a pass or an explanation is given in the case of a fail or partial pass. Any test given a partial pass is deemed to be a test in which the system has what the fit criterion demands but simply not in the quantity that is demanded.

In its current incarnation, the system does not pass all of the 18 acceptance tests and in fact only accepts 9. The full list of tests that are not passed are : Acc_Points, Acc_Char, Acc_Area, Acc_Mini, Acc_Boss, Acc_Powers, Acc_Enjoy, Acc_Play and Acc_FPS. Of these, Acc_Char, Acc_Area and Acc_Powers are considered to be ‘Partial Passes’, as these failed purely due to not quite reaching the numbers of characters, areas and powerups that they respectively require. In addition to this, the only reason for Acc_Enjoy, Acc_Play and Acc_FPS failing is due to the tests that they require not having been run yet.

The reason for the majority of these failures is due to the system still being under development. While the system meets the requirements given at Assessment 2, it does not meet the full range of requirements at this time.

Testing Material

The results of the testing can be found here:
https://github.com/mh1753/AbstractDelete/raw/master/Documentation/Assessment%202/TestingResults.pdf