University of York

Department of Computer Science

SEPR - Assessment 3

Updated Testing Report

Team Craig

Thomas Burroughs

Huw Christianson

Joseph Frankish

Isaac Lowe

Beatrix Vincze

Suleman Zaki

Testing Report

Our team decided to do system testing using black-box tests, unit testing (white-box), and peer reviews (static tests). We felt this was sufficient since we were developing a small system which was not safety critical. Testing was still very important to us as it improved the quality of our code, increased the confidence of the rest of the team, and likely the stakeholders too.

We used test driven development when suitable which ensured our implementation satisfied the requirements. This was done by coming up with tests before implementing the software that was to be tested, giving faster feedback and helping developers understand what they need to code. Creating tests before beginning implementation also helps programmers to focus on key functionality rather than implementing, possibly game enhancing but unnecessary features. Most of these tests were black-box as they were easily written up compared to unit tests. After implementation other tests, both black-box and unit tests, were added to try for as much coverage as possible. Additional tests will be written, and code refactored when previously written tests fail. The newly created tests may specify more detail in the expected outcome to cover the requirements or potentially remove trivial details which caused the test to fail. For example, a black box test may state that a character collects a power-up by walking over it, however after implementing, the character must click on the power-up to collect it. This requirement is still met but the original test fails. We ran these tests as we went. Using agile methodology, we will perform black- tests and unit tests after each iteration. The following Sprint will then involve the refactoring of code if tests failed.

Our requirements testing document creates links between tests and requirements to allow us to easily show what requirements have and haven't been met with easy traceability. A lot of the tests cannot be linked directly to the requirements as a lot of the features we are testing are implicit. We also included a requirements traceability matrix to more easily visualise the relationship between our test cases and each of our requirements within the requirements specification. This helps the team to see whether each of the requirements have been tested and will help with Assessment 4 when requirements are changed. The traceability matrix can allow team member's to easily identify which tests need to be altered according to their associated changed requirement

The white-box testing was done using unit tests, more specifically JUnit 4. Programmers will be responsible for writing their Unit tests for the code the programmers implemented collectively. They will take no part in black box testing. We noticed quickly that our unit tests were useful for regression testing (checking the software still functions as expected after changes are made). Our implementation included game logic in some classes that implement Screen, notably our Level class. This made it very difficult to test with JUnit because it had to be rendered to function correctly. For this reason, mostly black-box tests cover these classes. We used IntelliJ which had built in the option to "run with coverage" which allowed us to measure the coverage of our tests. This gives both the team and stakeholders an idea of how thorough our testing is. However, this value cannot be taken at face value because our black-tests will not contribute to this metric. Therefore, the results from the coverage tests will allow programmers to identify which classes have not been covered by Unit tests and further black-box tests will be created in order to test these functionalities.

The black-box testing was done by running the game and making sure it acted as expected for that test. White-box tests are preferred over black-box since errors encountered during black-box testing can be difficult to reproduce and the amount of coverage cannot be easily measured. Luckily since LibGDX itself has been thoroughly tested [3] we were able to focus on testing what we added on top of what LibGDX offers to developers. As mentioned previously when expected outcomes differ to actual outcomes code will be refactored in the following sprint or tests will be altered to remove trivial details. A final detail in our approach to black-box testing will be that testers who undertake black-box testing will have no knowledge of the current implementation. This will help to avoid bias when playing the game and create a more accurate representation

of a user's interaction, hopefully helping to find more unanticipated problems.

We created a GitHub project board to assign and manage tasks, one column in the project board was "To Review". When a task that involved coding was completed it was added to this section and then another member of the team had to verify the code. If they agreed with the implementation, they moved it to the "Done" column, otherwise they moved it back to "In Progress" column and told the person who originally moved it why. This made sure all code was double checked and up to standard. All code added to the git repository was reviewed this way, greatly reducing the amount of poor code in the final software system. This is a form of static testing since the code does not have to be run. This allowed us to test code before it is fully implemented which was very useful for more difficult problems that couldn't be solved straight away.

Test Statistics

There are separate files for testing evidence for white-box and black-box tests, but both use a tabular format. Each test has an ID to make it easily traceable to requirements, a description to give the reader a better understanding of the test, a section to say whether it passed or failed, and another column for any additional comments. An additional column is also present displaying the associated requirement which is being tested. Further traceability can be seen in both the requirement testing document and traceability matrix. ~~With the exception of tests 7.2 and 7.3, tests that cover unimplemented requirements have not been included.~~ See the references section of this document for links to all the testing material.

Black-box Testing Statistics and Results

40 out of 41 of our black-box tests passed by the end of the implementation for Assessment 3. The one test that failed is noted below with reason why and further details explained.

- 10.1- The player always faces in the direction of the mouse pointer.
    - This test failed as when the player is attacking it will not change direction. We didn't attempt to pass this test because we liked that it stopped the player from holding down the mouse button to attack as fast as possible. As a result, another test (10.1.2) was created with a different test case, in order to test this functionality as the previous team preferred (Payer cannot move when left button held). This test passed on it first attempt after its creation. As this test passed no refactoring of code was needed. Because the nature of the functionality of this feature was changed by the previous group no rectifications of code were made in order for the previous test (10.1) to pass

We think that our black-box tests are suitable as they cover all parts of the game that we couldn't test in a more controlled environment. Black-box tests by nature can be difficult to reproduce but we think that due to the detail in our Black-Box Testing Evidence document other people will be able to reproduce our tests and get the same results. A suitable link to black-box testing evidence is given at the bottom of this document.

Unit Testing Statistics and Results

34 of 34 white box tests passed by the end of the final sprint. We also ran the tests using IntelliJ's built is

code coverage runner. We were happy with our unit testing coverage because what was not tested in

this way was covered by black-box tests which was able to system and integration tests also. All Unit tests from previous game version ran as expected before implementation began. Some of the unit tests failed during regression testing after specific changes were implemented. For example, test…… failed after new zombie types were implemented, however programmers refactored the code during the next sprint resulting in the unit test passing on the next iteration.

We struggled to test classes that used LibGDX's Screen class with unit testing, these included Level, TownLevel, Halifax Level, CourtyardLevel, CSBuilding Level, GregsPlaceLevel, LibraryLevel, Loading Screen, Menu Screen, SelectLevelScreen, and TextScreen. We aimed to test these classes more thoroughly with black-box tests as an alternative.

We felt ZeprInputProcessor didn't need testing because it was, for the most part, just a standard LibGDX InputProcessor which has been tested by the LibGDX team. However, one-unit test WB1 has been used to test player movement which the ZeprInputProcessor is responsible for. What we did add was a mouse pointer position that is used for the player direction and player attacks, which was tested (tests 10.1, 10.1.2 and 10.2) and passed. We have attempted to get the best test coverage possible for all other classes.

URL Links to Testing Materials

Below are links to our evidence for both Black-box testing, Unit Testing and Requirements testing. Each test has been assigned a suitable Test ID to allow traceability throughout all our deliverables.

Updated Black-box testing evidence:
https://teamcraigzombie.github.io/assets/downloads/UpdatedBlackBoxTests3.pdf

Updated Unit Testing evidence: https://teamcraigzombie.github.io/assets/downloads/UpdateWhiteBoxTesting3.pdf

Updated Requirement testing evidence:
https://teamcraigzombie.github.io/assets/downloads/UpdatedRequirementTesting3.pdf

We have also included a link to our traceability matrix to show the relationship between our test cases and each of our requirements within the requirements specification.

Updated Traceability Matrix: https://teamcraigzombie.github.io/assets/downloads/UpdatedTraceabilityMatrix3.pdf

Updated Requirements: https://teamcraigzombie.github.io/assets/downloads/UpdatedRequirements3.pdf