

Software Testing Report2

Cohort 1, Team 1:

Connor Blenkinsop

David Luncan

Emily Webb

Muhidin Muhidin(MO)

Sam Laljee

Sooyeon Lee

Testing Methods and Approaches

Our testing plan was split into two sections: white box testing that would be carried out through unit tests using JUnit, and black box testing executed manually.

White Box Testing:

White box testing is a method that assesses the internal structure and design of the code. We used JUnit to write unit tests that focus on small sections of code and test them without accounting for their interaction with the rest of the program. A high coverage of unit tests ensures that all individual aspects of the code are functioning as intended, before testing how everything works together to create the game for the user.

We aimed to produce a smaller quantity of tests for important sections of code rather than a higher coverage of less useful tests, to make the best use of time. The main focus of the unit tests was to verify logic and calculations, such as earning the correct amount of XP. We also used them to ensure everything initialised as it was supposed to. We did this as these are harder-to-prove facts of the game, to ensure the integrity of the logic behind it.

We decided to begin white box testing at the same time as implementation so that the original code can be tested while the new features are being added. This also allows time for refactoring the code to be compatible with jUnit.

Black Box Testing:

Black box testing is focused on testing the functionality of the code without needing any knowledge of the underlying program. Due to time constraints, we chose to carry this out manually, writing user tests that assessed how well the system performed as a whole from a user's perspective, rather than the strength of its constituent parts.

Our requirements were the basis of the testing scenarios we created, as they were derived from the requests of the customer and therefore a good criterion to measure the success of the game against.

We chose to do black box testing later on so we could ensure a good coverage of tests while avoiding repeats. This type of testing could be used to fill in the gaps of what couldn't be covered by white box testing, such as non-functional requirements.

Report and Summary































White Box Testing:

By the end of testing, all tests that were run passed. Testing granted us a far better understanding of the underlying code base, but more importantly validated assumptions and allowed us to rectify instances where the code missed those assumptions.

We chose to ignore getters and setters and trivial functions and to unit test only more complex functionality. Though this meant a *far* lower coverage for far more time invested, these are more meaningful, life applicable and useful tests, that allowed us to catch subtle bugs in the code. Setting up the testing environment required rewriting files such as the gradle files (to align and add dependencies). The code needed significant refactoring - allowing the world class to initialise required mocking through mockito, but also running LibGDX in headless mode, in turn requiring some alterations to the code (visible in the world class of the attached code). Other alterations included modifying the folder hierarchy to be consistent with that of the testing suite even before any code was write, and the creation of many accessory functions to allow very opaque and private classes like the World class to be tested.

The limiting factor of white box testing, aside from time, was the lack of new code development. With less than 3 days to go until the end of the new project, very little new code had been added to github, giving very little time to test even if uploaded closer to the deadline.

Some quick white box testing statistics:

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
 main.game.core		54%		62%	15	29	61	115	12	21
 main.game.world.playerObjectives		61%		50%	30	41	28	54	24	35
 main.game.world.ui		96%		50%	2	6	5	78	0	4
 main.game.world		49%		42%	56	80	105	230	3	12
 main.game.world.playerStats		71%		41%	17	36	18	66	11	30
 main.game.world.content		50%		40%	40	73	78	162	22	50
 main.game.world.player		44%		25%	42	58	68	126	15	31
 main.game.menu		0%		0%	38	38	224	224	29	29
 main.game		7%		0%	21	22	48	49	8	9
 main.game.enums		0%		n/a	1	1	2	2	1	1
Total	3,198 of 5,631	43%	207 of 318	34%	262	384	637	1,106	125	222

Important Remarks:

main.game.enums - this class contains only 2 lines of code that could only be accessed through the unimplemented pause class, therefore not testable

main.game.menu - by far the biggest lot of untested code, a mix of purely graphical and new and unfinished sub-classes. This used to be a relatively small and inseparable from UI class until new code was added very close to the deadline. The large amount of new code was in part purely graphical, and in part required the mocking of a listener to be made or code to be heavily refactored - both time consuming and laborious activities that there was not enough time for.

main.game.world.content - this is where most of the new development has occurred. This is also the class I would most have liked to test more if there would have been time to do so.

main.game - mainly purely graphical features and some features added very late on

Excluding the 3 worst covered classes (which are all almost entirely inseparable graphics), and the large amount of not-yet-functional new code, **coverage** would be closer to **60%**

The only major bag found and fixed was in the World class, regarding the newly added Obstacles, it was fixed simply by a breakpoint, but would cause two of the 3 obstacles to not always apply or apply fully in very specific situations.

Black Box Testing:

Black box testing allowed us to cover the remainder of the game that couldn't be tested by the more time-consuming unit tests. It confirmed that, at least from a user perspective, the game would be functional and do what was expected. We chose to screen-capture gameplay as proof of successful user tests.

We ran 22 manual tests in total, which assessed the features required by the customer, each linking to a requirement. To track which tests satisfied which requirements we created a traceability matrix. This helped to visualise progress as well as ensuring the tests were exhaustive but not repetitive of what had already been tested.

The majority of manual tests (~80%) were successful as most requirements were met, however, the following tests did not pass:

4. Game difficulty

6. Save state

22. Power-ups

The save state button, although present on the pause menu, wasn't yet functional and a previous saved game could not be resumed. The other two features were also not completely implemented and didn't appear in the game map.

Another issue that arose during user testing was the strange movement of the enemy ships. Though they successfully move towards the player, they change direction too frequently causing them to flip backwards and forwards while in pursuit, creating a glitching effect. They also move through other objects and each other, often layered whilst following the player. This doesn't affect gameplay but has an aesthetic impact. Given more time to complete implementation, these issues would have been resolved and all user tests successful.

Closing remarks:

While a decent combined test coverage was achieved, the limiting factor in the quality and thoroughness of testing was implementation, with new parts of the game being added and tested with only hours until the deadline, and much of the implementation still missing. Throughout the process of testing, the dev team were kept in a close loop for bugs to be reported and fixed adequately.

Proof of testing and report for testing:

- Proof of user tests:
https://drive.google.com/drive/folders/15_CE-mwbE3WIHQ Cem9HP8xRz6oXxe97b?usp=sharing
- User Test Table: https://eng1team1.github.io/testing/black_box_testing_table.pdf
- Traceability matrix https://eng1team1.github.io/testing/tracibility_matrix.pdf
- White Box Test Table https://eng1team1.github.io/testing/white_box_testing_table.pdf
- White Box Test Code https://eng1team1.github.io/testing/final_testing.zip
- White Box Test Report https://eng1team1.github.io/testing/jUnit_test_report.html

Resources Used:

- <https://junit.org/junit5/docs/current/user-guide/#overview-getting-started>
- <https://www.tutorialspoint.com/mockito/index.htm>
- <https://www.codejava.net/testing/junit-tutorial-for-beginner-with-eclipse>
- <https://stackoverflow.com/questions/31579398/libgdx-java-junit-testing>
- <https://www.baeldung.com/jacoco>