Software Testing

Cohort 1 Team 9

Dominic Hall
Firas Marzouk
Ben Morrison
Harry Whittaker
Amelia Wigglesworth
Zehang Li

4. (a)

We conducted two major rounds of testing across this project; one when we first acquired the original game files from the previous group (a series of black box tests to ensure requirements were met), and a second round of testing when we completed the implementation of our additions (a combination of unit testing and black box testing).

Black Box Testing

Black box testing allowed us to examine the functionality of the game we chose without having to delve deep into the understanding of the programming behind it. We first conducted black box testing before we made any changes to the code. This allowed us to test the standard of the game and how it met the list of user requirements from an outsider's perspective, with little to no knowledge of how the game itself was programmed. We checked whether the game met the requirements listed themselves, and the customer's requirements. Any missing, or miss-labelled, requirements would need altering in the corresponding documentation, and is referred to in the change report.

Thanks to the nature of black box testing relying on the person testing to have no knowledge of the project's code, it saved the team time as there was no requirement to read or familiarise ourselves with the code beforehand, and allowed work to commence on the second half of the assessment straight away.

Throughout our programming process we were constantly black box testing, as we would run the game to ensure added features worked as expected, and that any ambiguities or possible user softlocking (when a user is unsure how to progress) was eliminated. When the game reached a point where we met all the user requirements listed, we conducted one final series of black box testing to ensure a user with no knowledge of the game's internal code was able to play the game and all requirements were met.

White Box Testing

White box testing is conducting an assessment of the internal workings of the programming and ensuring there are no logical or security errors within the code. When we first acquired the new game to work on, we never carried out white box testing as we felt it to be a redundant task initially, as we would inevitably be altering a significant amount of the code ourselves. Instead, we decided to wait until our programming was complete before we conducted a series of unit tests on the whole programme itself (including what was left of the original code). This allowed us to cover all the attributes of the game, from checking the existence of required assets, to ensuring the functions within classes worked as expected.

In addition, we have also used the integration testing method to test whether our code is compatible with others in the process of integration. Each member responsible for linked modules needs to be in close contact with each other, testing each other's code in relation to their own modules. If there are problems related to variable passing between modules, then both parties will have to make certain compromises in their code. However, because this is the individual obligation of each team member, integration testing will not be reflected in the test records and test report.

4. (b)

During the black box testing of the game we acquired at the start of assessment 2, we explored the game, comparing the implemented features to those listed in the given list of requirements. We discovered that most of the requirements that were *not* met related to the attacking of hostile ships/buildings, and the ability for players to easily distinguish between friendly and enemy ships. We made it a goal of ours to best implement these requirements on top of those required for assessment 2.

We also found that the range around the bonus chest's players could discover around the map was far too large; a player could collect the chest while it was nowhere near the physical sprite on the screen. While we wanted to fix this bug on water, we deemed it a necessary feature to keep within the game as when chests spawned on land, it made it impossible for players to collect them, and thus unable to progress with the completion of their quests. A solution to this would be to implement two types of chests to collect; ones that solely spawn on land and have a larger collection radius, and those that spawn on water and have to physically be collided with. However, due to time constraints, we noted this bug as non-essential and have not implemented it in the final version of the game.

We conducted both white box and black box testing of our final version of the project once all the implementation was completed. Our unit testing had a coverage of: 74% classes, 65% methods and 57% lines of code. Because of the nature of LibGDX itself, the packages used such as `com.mygdx.game.UI`, were unable to be tested through unit tests, which brought down the coverage percentages significantly. Accounting for this, an estimated coverage of code tested would be higher, with the majority of essential classes and functionality being tested successfully.

To ensure compatibility between LibGDX and JUnit, we used an open source framework for unit testing on LibGDX framework based projects, developed by TomGrill on Github. We did not encounter any fatal errors during our unit testing of the final project, however there was one caused by the QueueFIFO.java within the utils directory. We expect this was a problem caused by mislabelling within the class that caused the queue to not be first-in first-out. However, as this class was originally implemented by the previous team and it was not a fatal error that caused issues with the running of the game, we deemed it not to be an essential issue to resolve.

Once all the unit testing was complete, we began our final round of black box testing to ensure the game was in fact playable and that all user requirements were met to a sufficient standard. We did not spend a significant amount of time retesting the features implemented in Assessment 1 as no changes were made to them, bar a few that did not meet their requirements in the original testing. These few requirements were retested, as well as all the new features our group implemented as part of Assessment 2.

Due to time constraints, we prioritised resolving some requirements over others first, and thus a few were left incomplete by the end of the project. One significant requirement we chose to not implement, and thus failing the black box test, was to feature friendly ships that weren't necessarily part of your team. We encountered several issues when we attempted to implement this requirement, as we were unable to have the AI ships distinguish between friendly and other hostile ships/bases. Instead we made it so all other ships were hostile toward you. This also meant several further functional requirements that relied on this feature being implemented also failed the black box testing, as the feature did not exist. For example, "if a hostile college's base is destroyed, all remaining enemy ships will change sides to help the player". As this does not affect the playability of the game, we chose to focus our efforts on resolving more significant issues and tests that failed.

The previous group had suggested including a final boss battle to mark the completion of the game. This was another requirement that was not met at either round of black box testing. However, as this was not a specific requirement set out in the product brief, we judged this to not be a significant failure to implement, and instead ensured the implementation of our other method of completion (a list of quests for the player to complete) was done successfully. Similarly to the previously unimplemented requirement, as it had no notable impact on the playability or progress to the game, despite it failing the black box testing, we chose to not resolve it in the final product.

In addition, although the black box test directly tests the encapsulated game based on the requirements, as some requirements raised are not directly reflected on the UI, so we have to use the white box test methods to test whether these requirements have been met. For example, the exp value of the player has no representation on our game interface, only the exp variable will be updated in the background and affect the subsequent process of the game, so we have to use the white box test as an alternative, although the test is to meet the requirements for the black box test.

More information about our tests can be found in our test evidence.

4. (c)

All testing content: https://assessment2.yorkpirates.uk/testing

JUnit Test Report: https://assessment2.yorkpirates.uk/junit-testing

Manual Testing:
Manual Testing Round 1 Summary - Inherited Game
Manual Testing Round 1 Results

Manual Testing Round 2 Summary - Final Game
Manual Testing Round 2 Results