Part 4. Testing

Team 5 | Team Pending

Thom Robinson
Iris Scoffin
Ayman Elsayed
Annice Headman
Izaac Threlfall
Yihong Zhao

Summary

Testing is an essential part of any software engineering project in order to validate that the program meets the requirements, and to reveal failures in the code. When it came to testing our game, we used a combination of unit testing and manual testing.

For unit testing, we used JUnit 4.13 and aimed to test against functional requirements to ensure our game contained the features we set out to include. Unfortunately, parts of the code were coupled with libGDX functionality meaning they could not be tested. These parts had to be refactored to make them testable however, this still only made it possible to test logical code components. Manual tests were needed to ensure the game looked and played correctly.

Manual testing was used to ensure parts that could not be unit tested were working correctly. They also allowed us to confirm that what was unit tested correlates correctly to what happens in game. These manual tests involved playing the game with the aim of completing or observing the actions detailed by a requirement.

These tests are appropriate for this project as they provide evidence that our game meets the requirements as laid out in the requirements specification. They also give us good coverage of the game as a whole, with unit tests logically testing the key features. Then Manual tests confirm those unit tests and cover the rest of the requirements that cannot be unit tested.

Report

Automated Tests

The automated JUnit tests consist of five files, with each file containing the tests corresponding to a different aspect of the game. The written automated tests cover **65 percent** of the functional requirements as the rest are not testable using JUnit and will be covered by manual tests.

These five aspects are:

- Ships
- Colleges
- Obstacles and Weather
- Pickups
- Game Features

Ships

The ship tests contain all of the tests relating to both the player ship and enemy ships. This includes testing each direction of the players eight-way movement system, testing that bullets are created, deal damage and can destroy enemy ships, and testing that players health regenerates, players can die and they can gain experience over time and level up.

All of the tests written passed, however the ship tests are not quite complete due to being unable to test the movement of the enemy ships. This is because enemy ships move randomly until the player is near before moving toward them. This made refactoring the code to allow for testing difficult and will be tested manually.

Colleges

For testing colleges, automated tests were written for testing that colleges can take damage when hit, colleges can be destroyed, colleges will shoot at the player and that colleges give plunder and experience to the player. All of these tests pass and are functionally correct and complete.

Obstacles and Weather

When testing obstacles we tested that each type of obstacle could be created and each of their unique properties worked correctly. This meant rocks, icebergs and sea mines were all tested separately with each of these tests passing.

However, we could not test each weather effect as they are generated randomly. So to test weather, we tested if a weather effect is added to the game. In order to test each weather effect the code would need to be edited to allow a specific weather effect to be created. The test for weather passed making the weather and obstacle tests correct but not complete.

Pickups

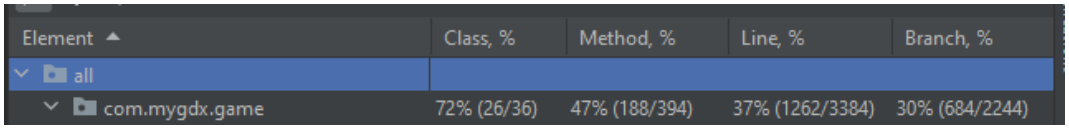
Pickup tests consist of testing that the pickup is removed from the world once the player collides with it, the powerup changes the player's stats according to which powerup it is and that the player cannot pick up a powerup if they have insufficient gold. These tests for pickups are both functionally complete as they cover all aspects of pickups. They are also functionally correct as they prove the code gives the correct result.

Game Features

Testing Game Features relates to testing the game's saving feature and testing that both of the win conditions are met and that the game ends. These tests all pass and are functionally complete and correct as they cover all related requirements and provide the correct results.

Code Coverage

These automated tests gave a line coverage percentage of **37 percent**, which is not as low as it seems due to our use of libGDX. The implementation of libGDX is not testable using automated tests, and libGDX components make up a significant portion of the overall code, which lowers the line coverage dramatically. The Class coverage percentage is **72 percent** showing all classes containing important logic are covered by our tests.



Element	Class, %	Method, %	Line, %	Branch, %
all				
com.mygdx.game	72% (26/36)	47% (188/394)	37% (1262/3384)	30% (684/2244)

Manual Tests

Manual tests were conducted by aiming to complete actions through playing the game to prove that a requirement has been met. This was done for both functional and non-functional requirements and allowed for testing of parts of the game that could not be automated. Manual tests also proved that the JUnit test correlated correctly to gameplay. Some of the non-functional requirements did not need to be tested as they were ingrained into the design of the whole game such as NF_Players, the requirement that the game be single player.

All of these manual tests provide the expected outcome to meet the requirements and cover **100 percent** of the functional requirements and the necessary non-functional requirements making the tests functionally complete. The conducted manual tests are also functionally correct as they provide the correct output for all test cases. There were also no errors reported when conducting manual tests.

Testing material can be found on the website:
Testing evidence : https://eng1-team5.github.io/static/A2/Test2_C.pdf
Tests : https://eng1-team5.github.io/static/A2/JUnitTests.zip