Testing Report

Cohort 1 Group 1:
Ileri Adegeye
Maida Ahmed
Holly Ainsworth
Anthony Colin Jones
Luca Gilligan
Alex Joyner
Minhaj Zaidi

Testing Process:

We used a combination of automated unit and integration tests and manual end-to-end tests to test our program. This allowed us to verify that the game met the requirements and contained no major faults, whilst still enabling the vast majority of tests to be run automatically. The tests were created once development had mostly finished since we decided to use white-box testing, so had to have the code available before creating tests for it.

Unit and Integration Tests:

Due to our project using the Entity-Component-System (ECS) model, we were able to test a large part of our program using automated testing, as we could easily see the output of a given function by the messages sent between the layers of the model. As well as this, the code is very modular, allowing tests to be performed on specific parts of the system easily.

We used LibGDX's built-in headless backend to perform tests and to provide a fake of the game without having to load graphics as well. We also used the JaCoCo plugin to measure test coverage, using line coverage and branch coverage as our main metrics.

End-to-End Tests:

Since our project is a game, a number of tests had to be performed to ensure the GUI and sounds were working correctly. We were unable to test this automatically because the fake we used for automated testing is not designed to run graphics. As well as this, we wanted to ensure that the game met the User Requirements, and due to the high-level nature of these requirements the easiest way to test them was with End-to-End tests.

The manual tests were designed beforehand, with a list of instructions for the tester to follow. Tests were marked as either fully passed, partially passed or failed. The tester was told to take screenshots of any reasons why the test failed to show where and what happened.

The test-cases for the Manual tests can be found on the team's website:
https://eng1-c1g1.github.io/assets/Manual%20Testing%20Process%20and%20Results.pdf

Test Report

The full automated test statistics can be found on the team's website.
Test Results: https://eng1-c1g1.github.io/test_report/index.html
Test Coverage Report: https://eng1-c1g1.github.io/test_coverage_report/index.html

The results of the manual test can be found on the team's website:
https://eng1-c1g1.github.io/assets/Manual%20Testing%20Process%20and%20Results.pdf

Automated Testing

60 automated tests were run, some of which tested multiple input conditions. All 60 of the tests passed. This resulted in no failures of the requirements.

These tests covered 50% of the instructions and 42% of the branches. The vast majority of the uncovered instructions and branches were related to audio or graphical output, so could not be effectively tested using the testing framework. Of the ~4000 instructions not covered, less than 100 of them were not related to audio or graphical output, which is ~1% of the whole codebase.

Manual Testing

Of the 7 manual test-cases that were run, 5 passed fully, and 2 passed partially. The 2 that only partially passed were GUI_TEST and HITBOX_ALIGNMENT_TEST. However, these only resulted in a very minor failure of one requirement.

GUI_TEST only partially passed because the Title Screen was off-centre, due to the Leaderboard and Achievement panels not having a fixed-width, allowing them to change size based-on the length of text in them. This didn't result in any explicit failures of requirements.

To get GUI_TEST to pass fully would just involve changing the Leaderboard and Achievement panels to have a fixed width.

HITBOX_ALIGNMENT_TEST only partially passed because there were a few instances of the wall hitboxes not being consistently aligned. This meant that when moving along a wall, in some places you would stop moving as you encountered a protruding hitbox from another section of wall. This resulted in a minor failure of the UR_BOUNDARIES requirement.

To get HITBOX_ALIGNMENT_TEST to pass fully would just involve tweaking the collision layer in the Tile Map to ensure that these hitboxes lined up.

Completeness and Correctness

As can be seen in the manual test report, the manual test-cases cover a large number of the requirements listed in the Requirements document. The automated tests cover the

remaining, more functionality-based requirements like: FR_POSITIVE_EVENTS, UR_LEADERBOARD, UR_ACHIEVEMENTS, etc.

The only requirements not covered by the tests are UR_THEME and FR_THEME, which are too subjective to be covered by tests, however users were asked about this during the User Evaluation.

The tests that we designed and used have a high level of functional completeness, since they cover almost all of the requirements set out in the Requirements document, and the requirements they don't cover have been "tested" during the User Evaluation.

Guaranteed correctness cannot be feasibly achieved due to the complexity of the game. To combat this, tests were designed to test potential failure points, such as boundary conditions. Where appropriate, the automated tests use valid, boundary and invalid data to test functions and systems to maximise the chance of finding any errors. We used white box testing to ensure that all boundary conditions were covered.