Testing

James Smith
Amanda Ling
Fran Medland
Hannah Vas
James Kellett
Malik Tremain
Mischa Zaynchkovsky

4a - Testing methodology and rationale

We decided to use a mix of playthrough testing (PT) and Junit test, where each testing method was most appropriate. Junit testing was used for most of the functional aspects of the code (scoring, energy consumption etc.); PT testing was used for things like game state transitions, graphics etc..

We thought it would be most appropriate to prioritise PT as the nature of the product we were creating would be graphical and an experience for the user. PT requires no further set-up and would be an effective method for discovering any major bugs. Our chosen way of executing PT could be split into 2 parts.

For the first part, we, the implementation and testing team, would PT our newly added features or modified code. We conducted “isolated PT”, meaning we played through the game with the newly added feature as the focus of our investigation. This allowed us to catch any issues as they were created and to ensure the correctness of the code we added.

The second part was the extensive PT that we conducted at the end of the implementation stage. We did this to ensure that the product would work as expected and to catch any integration issues that may have been missed during the isolated PT.

As mentioned Junit testing focussed on the functionality of the code itself and the logic within. Junit was used to test the leaderboard system, scoring system, button creation, energy consumption and in-game time.
A large portion of this process was dedicated to refactoring the code to make it better suited to automated testing. Our newly implemented requirements, scoring and leaderboards were already suitable as they had been created in their own isolated class. The energy, buttons and time however were not. These aspects of the program were separated into their own Java classes, improving the ease of use and effectiveness of our Junit tests.

Finally, we took inspiration from Hoare triples and program verification approach to verifying correctness by including assertion in important core procedures. The assertions would validate the arguments to the procedures were in the correct format. Atop of this we also used assertions to ensure that the program was in the correct state before and after the execution of the procedure. This allowed us to be confident that the program would work as expected.

4b - Testing Results and Analysis

Test Statistics:

JUnit Tests: 37 passed, 0 failed, 37 total.
Play Tests: 39 passed, 6 failed, 45 total.

Of our 37 JUnit tests, each one passed. As mentioned above, we tested the leaderboard, scoring, energy, time, and button creation functionalities using automated testing. These tested the broken-down sub-functionalities of each aspect. For example, we tested the addition of a new score to a full leaderboard (test J_22) in the leaderboard tests. These tests were very thorough ensuring each functionality was tested to completion. Our 100% success rate in our JUnit tests shows that the correctness of our code was very high.

Above we mention that a lot of time spent on testing was dedicated to refactoring the code for automated test suitability. Ideally, we would have tested much more of the program using automated testing. Whilst certain aspects of the program were fairly easily refactorable, due to the way the previous group structured their game it was not feasible to automatically test a fair portion of their game. If we were to further automate our testing process we would have had to completely reprogram/refactor the code given to us, restructuring everything, which was not a realistic task. So, we decided to prioritise JUnit tests for the correctness of our program, ensuring that inputs within the program resulted in the correct outputs. As a result, it could be argued that our automated testing did not capture a complete overview of the game. To account for this, we focussed more of our efforts on playthrough testing. We ensured that the playtesting conducted filled in the gaps left by our automated tests.

Overall our playthrough testing was very successful with only 13% of tests failing. These are explored in more detail below:

PT_1 and PT_22:
PT_1 and PT_22 tested that upon start-up of the game, the player was presented with a screen outlining objectives, context and controls of the game. PT_1 was also testing that the ‘Start Game’ button was present and functional. The tests failed because, upon game start-up, a main menu is displayed with 4 buttons: start the game, controls, settings and exit. This is not the outcome expected as per the requirements linked to the test. The second part, testing that the start game button worked, was successful.
The failure of these tests is not an issue, as although it was not done in the way initially planned, the user can still find all the information that was being tested for initially. They simply have to navigate to the controls screen using the buttons on the main menu.

PT_12:
PT_12 was testing that a game over screen is displayed if a player performs poorly. This test was unsuccessful because although an end screen is displayed, there is no indication as to whether the player was successful or not. This was a deliberate decision. With the addition of the leaderboard system, we felt there was no need for a pass/fail in the end screen. The measure of the player’s success is whether they have been able to make their way into the top 10 scores. In addition, the addition of streaks made it difficult to keep the scoring below

100 (as was initially planned by group 25) so gauging whether a player passed/failed solely by score would be near impossible.

P_25:
P_25 tested that the player's score was calculated upon ending the game and that they would be awarded a ‘grade’. The player’s score is successfully calculated however no grade is awarded in the end screen. As discussed above, due to the additional requirements received in assessment 2, gauging a player’s success with their score is difficult, and so we chose not to implement this.

PT_32:
PT_32 was testing small animations when a player completed different activities throughout the game. The failure of this test was not an issue. Animations are included when sleeping, and they were not required for studying as the user played a mini-game instead. Implementation of these animations for the other activities was an added complexity that we did not feel was necessary for the game. The game still plays well without them, and their addition would simply be game polishing, which was beyond the scope of the assessment.

PT_34:
PT_34 was the final playthrough test that failed. For success, the test required a ‘constructive and positive message’ to be displayed at the end of the game regardless of the player's score. The test failed because we did not implement this message into the end screen. Once again, the addition of this message is simply game polishing that was unnecessary for the assessment.

Overall our playthrough tests were successful. We had a small percentage of test failures and those that did fail had no great impact on the game’s playability or the program’s stability.

Looking at testing holistically, we were very successful. Our testing plans ensured every requirement was tested, with many having multiple tests linked to them. With an overall testing accuracy of 92% it’s safe to say that our game was successfully developed.

Testing Material URLs

Automated testing

Test-Table: JUnit Tests:
TestingTables Junit

Automated testing reports:
Gradle test report
JaCoCo test report

Manual testing

Test-Table: Playthrough Tests:
TestingTables-PT-FR
TestingTables-PT-NFR
TestingTables-PT-UR
TracabilityMatrix-FunctionalSoftwareRequirements
TracabilityMatrix-Non-FuntionalRequirements
TracabilityMatrix-UserRequirements