Testing

Module	SEPR
Year	2019/20
Assessment	3
Team	York Fire Marshalls
Members	Samuel Whitehead, Andrew Connor, Alex Dawson, Finn Jackson, Fred Dodd, Phuong Kha
Deliverable	Testing

Testing Method and Approach

To identify which form of testing best suits our group, we did research into testing methods and plans. Taking into account the small size of our group we decided on the IEEE 829-2008 Standard [5] for the test plan and our testing method was Black-Box testing

Using the IEEE 829-2008 Standard we created a test plan that helped identify the key components of our testing. These key components include: Test Items, Features to be tested, Features not being tested, Approach to Testing, Pass/Fail Criteria and Test Deliverables. This allowed us to clearly and easily set up our tests in order to ensure we were on track, meeting all requirements and that it was funcional.

The method of testing that we chose after our research is the Black-Box testing method. Essentially this method works by examining the functionality of the product without delving into the code and inner-workings. This allows us to test the product the way that the user will interact with it, ensuring that the experience is exactly what we had planned.

The way we'll use this method is by having a list of tests, with their expected results, in a document. The tester will then go down the list completing all the tests manually, if the tests outcome matches the expected result the test is passed, if not the actual outcome is recorded and reported to the rest of the group. These tests were performed after any big change in the code, as well as once every week. This was done to ensure that we were making the correct amount of progress and that to test that we hadn't broken any previously working functions.

While the chosen tests are meant to test for every and all actions the user would want or need to do it is impossible for us to account and test for every action a user would take and so it is important to note that this method, as well as any method of testing is not 100% successful at detecting bugs and errors.

A requirement traceability matrix will also be used in conjunction with our test units to ensure that what we are testing for also matches the requirements for our product. This is essential as without it, it is highly likely and easy to add features that are either not necessary or miss features that are. Because of this, during our weekly test, we would also update the requirement traceability matrix to check that we were fulfilling the requirements needed for our product.
We have since updated the traceability matrix as we had found that it was not easy to trace which tests had fulfilled each requirement. Previously many tests had not matched any requirements, so these have also been removed as we had found them unnecessary.

White-Box testing was trialed by our group, but due to the changing nature of our code it ended up being more effort to create and adjust the tests every time, than if we stuck with Black-Box testing. Creating unit tests on libgdx required a headless instance[3] of the game. Junit was integrated into the project and Azure built every project on push but we could not figure out how to implement unit tests for the game.

Testing report

On the website it is possible to locate our final testing log. This is an example of how we tested our game throughout the implementation process.

Test results [2]

Number of tests passed: 27
Number of partial test passes: 1
Number of tests failed: 1

Earlier logs had more fails but since this is the end of the current round of implementation, most of our tests were completed.

TC23 failed because we had not yet implemented the Fire Station destruction and the difficulty did not increase with time. This will come in later iterations of the project, but in its current state it is not in, hence the fail.

TC25 was only a partial pass because of a lack of resources. Without a Mac computer, we were unable to test our game on all three OS's that we wanted to test them. While it did pass the Windows and Linux test we had no way of knowing if it would pass the Mac test. Because of this, we could not definitively state that it had failed, nor could we claim that it had past, so we decided to record it as a partial pass for these reasons.

Requirement Traceability Matrix [1]:

A backwards traceability test matrix was employed, mapping all of the test cases to the functional requirements. As some of the functional requirements have not been implemented, there were some cells within the matrix that were left empty. From the completed traceability matrix, we can draw the conclusion that the product has not strayed from the updated requirements. Out of the 17 functional requirements in the matrix, 11 passed. The other 6 functional requirements had either been partially and not implemented, meaning the tests weren't fully passed. The lack of implementation for these requirements has been justified in IMPL2 [3].

Completeness and Correctness of Our tests

Even though the large majority of our tests passed, it is still possible for bugs and errors to occur. The tests did not account for all bugs in the game, this is due to the nature of testing. An example of a bug known in the game but not recorded on the testing sheet is:

> If the pause menu is brought up when a user is holding down the mouse button over a menu button then it will break that button, making it impossible to interact with it.

While highly unlikely that a user would do this combination of keys it is still possible.

This is proof that while our testing was extensive it still cannot account for every bug and error in the code. That being with our testing it becomes increasingly less common for these bugs and errors to affect the user.

The use of mainly manual testing was not the ideal way to test the code, but under our circumstances stood as the only practical way of testing our code, without substantial more effort on our part. We had a continuous integration pipeline that built our code on push but that only told us if the code could compile and nothing else.But with the relative small size of our game it was acceptable and achievable.

The pesticide paradox restricts also restricts us from finding every bug. The pesticide paradox states that every method used to find a bug, leaves a small residue bug for which that method is useless. Meaning that we can never definitely say our code is bug free.

Overall, we believe that our testing has been suitable and that it covers enough that the game should be functional. Since it tests all the major interactions a user will take, there shouldn’t be any problems when a real user plays the game.

Links

[1] Traceability Matrix [2s.1]
https://docs.google.com/spreadsheets/d/1x0CG9MiFtrh2Vxmv7KBtYZwpXpDp49aunIc_EKc0hq0/edit#gid=0

[2] Testing Design/Evidence [2s.2]
https://docs.google.com/document/d/1r1rafJILcJe_3tJiGrFYW8jeuJDwc1FPEGblsqmtd3M/edit

[3] Implementation Document
https://docs.google.com/document/d/1Q3L1_HbhJCZ-UGAmRamI0bkDNyfXKLhxXQoO6BJtKIw/edit

[4] Headless testing
https://github.com/TomGrill/gdx-testing

[5] IEEE 829-2008
https://standards.ieee.org/standard/829-2008.html