Software Testing Report

Testing Methods and Approaches

Considering that the software we are developing is not a critical system and that the size of our project is relatively small we have identified: Peer testing, White-box testing trough Unit tests and Black-box testing as the most appropriate methodologies to ensure correct functionality of the software.

Since for our implementation we followed the Test Driven Development Method, we initially planned and created several tests based on the requirements of our software, such tests were fundamental for ensuring that requirements would be met by our implementation. However, during the course of development new tests have been added to check the correctness of the additional features.

Peer testing was one of the styles of testing we used from the start of the project implementation and throughout. We decided to start using peer testing from the very beginning of the project implementation stage to ensure consistent and conflict free code was being produced. Through github we had to peer review each others code before a branch could be merged to master. The software that github provided to review code changes allowed us to be thorough when we reviewed each others code; it allowed us to meticulously go through line by line every change showing us which lines of code had been added and removed by a group member. Meeting in groups also allowed us to go through the latest version of our code to do live testing rectifying any errors found quickly. Peer testing is particularly useful at this stage of the project as it is very easily done.

We used Black-box testing to report whether certain things worked in our program. For Black-box testing the tester does not have knowledge of the programs internal workings, they simply carry out actions, e.g. pressing a button, and report the outcome that occurs. One of the advantages of Black-box testing is that we don’t need to know exactly how the code is working [1] therefore members of the team who are less familiar with programming could assist in testing just as effectively. However, this can lead to many test paths not being traced which could leave unseen errors [2]. These unseen errors can however be picked up through White-box testing. At the start of the project, and as we were developing we added tests to our testing design for Black-box tests, and then carried out each one recording success or failure.

One of the biggest challenges for agile teams is being able to maintain working code despite the frequent builds and changes[3]. White-box unit tests enabled us to solve the issue by allowing us to manually run tests defined in our test plan every time the code was modified. This was useful in our agile approach as it allowed us to repeatedly ensure the smallest parts of the program were working, and increased our scrutinising and the efficiency of the code.

We also found it important to complete requirement testing, where we continually tested our overall project with the current requirement document. This ensured that we were hitting targets and achieving the goals that we had laid out for ourselves, while also ensuring we stayed close to what the client asked for.

Testing Analysis

Black-box Testing

We set up multiple Black-box tests using different methods to derive them. The main method we used to derive Black-box tests was through functional testing by looking back at our requirements table to test that certain core aspects of the game were working. The rest of the tests were derived via play testing the game. The results of our tests can be viewed on our website. [4]

Testing Results

The vast majority of the Black-Box tests passed, in most cases the user was able to navigate through all menus and submenus, change settings, setup a game and play the game from beginning to end with very few issues. All tests were recorded, and for those that failed, attempts at a solution to the issue were made.

Tests That Failed

Of the 39 Black-Box tests conducted, only 1 failed. [4] (Additional Validation Testing - 1) This tests the validation of player names and should not allow the game to start if any player names contain special characters or are not unique. In this case a message should be displayed explaining to the user why the game has not started and give them a chance to remedy this before attempting to start the game again. However player names such as “Player1” and “Player 1” were allowed to exist within the same game, which we found to not be ideal.
The initial attempt at fixing this issue had caused other issues in the validation of player names, either incorrectly not starting the game, or starting the game when the player names should not have successfully validated. However, Due to time constraints, the code was reverted and the issues remains

White-box and Unit Testing

We performed our unit tests with White-box testing. As the development process and our project grew we added tests to the test plan, and performed these tests often to ensure every method worked regardless of the new features we were adding and improving. All our White-box unit tests can be viewed on our website. [5]
It’s important to note that we have experienced major issues trying to implement automated JUnit tests in our development environment. Since we weren’t able to find a sensible solution, we eventually decided to run the tests manually by looking at the code and checking that the desired output was produced correctly. This was definitely time consuming

and we are aware that teams that will pick up our project will have to find an alternative solution for it, to improve test repeatability.

Testing Results

A high percent of our unit tests passed, as we were meticulous with our peer testing and code reviews. However occasionally an error arose and was caught by its unit test. Out of the 47 unit tests we designed and ran, only 3 failed. We attempted to create a solution to every issue found, and in every case we either attempted to solve the issue or created an innovative solution to ensure the issue wouldn't affect the current release stability.

Tests That Failed

One of the tests that failed originally was [5](OptionsScreen class - 1a). In this test we were checking the output of the getPossibleResolutions method, which created an array of possible resolutions for the monitor viewing the game. However an issue was caused when a unnecessarily small resolution was chosen by the user, which caused scaling issues with the UI. Therefore to resolve the issue we decided to put logic into the method that wouldn't allow the user to select a resolution under 1000x1000 pixels.[5](OptionsScreen class - 1b)

Another test we had issues with was the startGame method in the GameSetupScreen class. [5](GameSetupScreen class - 10) This method is responsible for initiating the game, and while the game does start as expected, with all the users settings and configurations, the map is sometimes drawn off centre. We attempted to fix the error however decided it was low priority as the actual functionality was not affected, and therefore focused our time on new features. However this is definitely something we aim to fix before any more releases.

The other test we ran that failed was in the method sectorOwnerChangeDialog. [5](DialogFactory - 5) This method is responsible for creating a dialog box to display the fact a sector has a new owner, but no option to move troops is available. While trying to uncover the cause of the error we noticed that the call to the method is never reached due to logic in Map.attackSector, however unfortunately we were unable to get a solution before the first release.

Requirement Testing

While we performed unit tests on every method, we also found it important to test our system as a whole, and ensure we were sticking to the client's requests. Therefore we repeatedly ran requirement testing that can be viewed on our website. [6]

Testing Results

None of these tests failed, however there are multiple requirements that haven't been tested yet, as they require the whole system to be built. The requirements that have passed have been well met, and we are confident that the future progress of the program will meet every requirement that is set out.

References

[1] Software Testing Strategies and Best Practices | Atlassian [Online]. Available: https://www.atlassian.com/software-testing/?tab=manual-software-testing [Accessed 22/11/2017]

[2] Black Box Testing - Software Testing Fundamentals [Online]. Available: http://softwaretestingfundamentals.com/black-box-testing/ [Accessed 22/11/2017]

[3] Steve Miller “Top 5 Common Challenges for Agile Testing Teams” [Online]. Available: https://blog.smartbear.com/sqc/top-5-common-challenges-for-agile-testing-teams/ [Accessed 21/01/2018]

[4] SEPR “Black-box Tests” Risky Developments [Online]. Available: http://www.riskydevelopments.co.uk/documents/BlackBoxTests.pdf [Accessed: Jan. 22 2018]

[5] SEPR “Unit Tests” Risky Developments [Online]. Available: http://www.riskydevelopments.co.uk/documents/UnitTests.pdf [Accessed: Jan. 22 2018]

[6] SEPR “Requirement Traceability Matrix” Risky Developments [Online]. Available: http://www.riskydevelopments.co.uk/documents/RequirementTraceabilityMatrix.pdf [Accessed: Jan. 22 2018]