Testing Methods and Approaches

We decided to organise our software testing approach into two main categories: White-box testing and Black-box testing. White-box testing examines the inner workings and structure of the code which will be useful for thoroughly assessing the ongoing implementation. Black-box testing evaluates the product from an external perspective and will be useful for ensuring the requirements are fulfilled appropriately by assessing the functionality from a user's perspective. We also carried out Requirements Testing as a separate form of testing for non-functional requirements that are not included in the Black-box tests. Despite not doing this for the previous assessment, we decided to adopt the Pi-rates use of Requirements Testing as it offered a new layer of testing that we had previously left out.

After researching various White-box testing methodologies we came to the conclusion that the Test Driven Development approach (as outlined in the paper by Janzen and Saiedian) would be most suitable for our project [1]. Janzen and Saiedian explain that Test Driven Development is an agile testing strategy that "requires writing automated tests prior to developing functional code in small, rapid iterations" [1,p.43]. This approach to testing is perfectly suited to our project due to its compatibility with the agile approach we are already following. TDD is an iteration-focused development strategy which encourages refactoring code to improve quality and permits the necessary flexibility to adapt to changes in requirements. We selected the JUnit tool to create automated unit tests. The advantage of automated testing is that tests are reusable and are therefore more reliable and consistent than manual approaches to testing. [1]

One benefit of the Test Driven Development approach is the emphasis on refactoring code to incrementally improve quality. ~~We devised a Peer testing strategy to enhance the benefits of TDD even further. Peer testing involves reviewing the code of other team members to identify areas for improvement.~~ We did not find a need to engage in peer testing as most code was done as a group and programmers had been reviewing one anothers code from the beginning in order to keep coding strategies coherent.Clark found evidence that Peer testing improves the quality of a software product and also encourages steady progress "since the students need to show their work to peers before it is actually due for submission, they are required to begin working on it much earlier" [2,p.47]. [1]

We also planned a Black-box testing strategy to assess the system's functionality and usability from an external perspective. The major advantage of Black-box testing over White-box testing is that testing is conducted from the user's viewpoint and does not require any specific knowledge of the implementation [3]. This is useful because it allows team members with less understanding of the internal implementation to contribute to testing. It also helps ensure that the game follows a user-friendly design and provides a desirable customer experience, essential qualities that cannot be assessed via White-box testing alone. [3]

We decided to plan our White-box tests by selecting and listing the critical methods of each class along with a detailed description of the expected outcome. The unit tests could then be written to ensure that the actual outcome of each method matches the expected outcome. We planned our Black-box tests in a similar fashion except that rather than listing methods for each class, we listed user inputs in various

scenarios along with the expected outcomes. This approach to planning was useful because it required us to describe the precise behaviour of every method and UI element in advance. [4] [5]

Test Report

White Box Testing

Overview

Unit tests were carried out on a total of 31 methods across 11 classes selected by criticality. These tests were written to test the expected outcomes of each method as outlined in the White-box testing spreadsheet [4]. The unit tests were then written alongside the code and a Test Driven Development approach was used to improve the code via iterative improvement and peer review. We capitalised on the benefits of automated testing, writing them as early as possible to enable maximum reusability throughout the implementation. We prioritised the methods to be tested based on their complexity and criticality to the product and went into detail outlining the expected behaviour of these methods. One area of improvement would possibly be to expand the completeness of our tests to cover less critical elements. [4] [6]

Results

All 31 methods now pass the tests however a couple of the methods did not pass first time. The addUpgrade() method (Room-3A) failed at first due to a small error in the code that caused an upgrade to be added to all the available upgrade slots rather than just once. This was quickly fixed by using a Boolean flag that is set to True when the upgrade has been added to an upgrade slot so a condition can be added to ensure that the upgrade is not added to any other empty slots once it has already been allocated. The method now only adds the upgrade once and passes all tests. There was also an issue with the deductGold() method (GameManager-2A) which failed at first because there was no check to ensure that the gold could not become a negative value. This was also easily rectified by adding a check to ensure that the new gold value is not a negative value and this method now passes all tests. All of the other tested methods pass their tests.
[4] [6]

Black Box Testing

Overview

At this stage of the implementation, Black Box tests were carried out for the Department screen and Combat screen. All of the Black-Box tests were planned from the user's perspective and focus on the front-end display as opposed to the back-end mechanics tested by White-Box unit tests. The test plan consists of a list of possible inputs from the user for a section of the application and an expected outcome. All critical features of the front-end are tested and recorded in the planning document and screenshot evidence is provided in a separate document. [5] [7]

Results

In total there were 11 Black-box tests (6 for Department screen and 5 for Combat screen) for the Assessment 2 implementation all of which passed, however 1 did not pass first time (Combat Screen 1A).

Requirements Testing

All the requirements applicable to Assessment 2 are fulfilled by the current implementation. This can be seen in the Requirements Testing document [8]. Requirements C1, F1, F3-F5 and F12-F17 will be fulfilled by the final product but are not relevant to the required functionality of the current implementation for Assessment 2 so have not been tested.

References

[1] D. Janzen, H. Saiedian, “Test-driven development concepts, taxonomy, and future direction”, *Computer (Volume: 38 , Issue: 9 , Sept. 2005)*, 2005, [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1510569. [Accessed: 3 December 2018]

[2] N. Clark, “Peer testing in Software Engineering Projects”, *ACE '04 Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30 - Pages 41-48*, 2004, [Online]. Available: https://dl.acm.org/citation.cfm?id=979974. [Accessed 3 December 2018].

[3] A. Rongala, “What is Black Box Testing: Advantages and Disadvantages”, [Online]. Available: https://www.invensis.net/blog/it/black-box-testing-advantages-disadvantages/. [Accessed: 3 December 2018]

[4] H. Clements, “White Box Testing”, *Pi-Rates Team Drive,* 2018, [Online]. Available: https://docs.google.com/spreadsheets/d/1mrGZAOprI3Qf4YFM-uYihb5jiJFNPFxlvw8ayVEEFSY/edit#gid=1608042115

[5] H. Clements, “Black Box Testing”, *Pi-Rates Team Drive,* 2018, [Online]. Available: https://docs.google.com/spreadsheets/d/1umgHhMdR-sNf1jhcDZsZcya9lLKQJGs4IxODU1JT7IQ/edit#gid=1791387930

[6] H. Clements, “White Box Testing Evidence”, *Pi-Rates Team Drive,* 2018, [Online]. Available: https://docs.google.com/document/d/1ZFWacKcUBPZSq_UI4GVDu4KpL70n2IxhtXpau5onCto/edit

[7] H. Clements, “Black Box Testing Evidence”, *Pi-Rates Team Drive,* 2018, [Online]. Available: https://docs.google.com/document/d/1Btf2EtIhmDVzjA_ipNMvvS8u64kqV76Qq1Xbvj1bRgY/edit

[8] H. Clements, “Requirements Testing”, *Pi-Rates Team Drive,* 2018, [Online]. Available: https://docs.google.com/spreadsheets/d/1xn_xeSguIiF8qYty3JPhlHO3a_0B0u6d94DRn4p1dr8/edit#gid=0