Software Testing Report

Testing Methods and Approaches

In order to decide on an appropriate testing method and approach, we conducted research into software testing before beginning implementation, as our choice of testing method could influence our strategy for coding. Having followed IEEE standard for our Requirements specification, and conducting research into the IEEE standard for software testing[1], we decided to apply this method.

The main factor that influenced our decision, was the advertised adaptability of this method, as it "can be used by any organization when performing any form of software testing"[2]. This was important for us as ours is a small project and we have a very short time frame, so our testing needs to be thorough, precise and fast. The IEEE standard allows for this, as it describes a an outline for testing which can be used with any testing approaches, so we were able to research into the most appropriate ones for us. We did, however, decide that a lot of the aspects involved in this standard were unnecessary for the testing, therefore we have only chosen a select few concepts to put into action, such as the test traceability matrix, and a simplified Dynamic testing process. We will also be using a static dynamic testing process during code reviews, and the developers will be testing the game throughout the development process to catch obvious bugs, using features such as a test scene.

The first method we will be using is white-box testing, more specifically automated unit testing, which consists of writing a piece of code to test that a single unit of the source code performs in the desired way, giving the correct outputs for given inputs. This method is appropriate for our situation as our abstract architecture indicates that there will be a lot of classes, functions, and procedures in the code, which will need to be tested as soon as possible in order to detect bugs on in the testing process. The unit test can also be run once all the units have been integrated to ensure they still run as expected. Once the code passes these tests, it is refactored. We will also be performing code reviews, which involves systematically going through the code to find mistakes, improve the overall quality of software and ensure consistency between procedures.

Our second testing approach uses black box testing, where the internal structure of the software is unknown, and test cases are produced using the requirements specification and risk assessment. This testing approach suited our project, as it should be carried out by a tester who has no knowledge of the internal structures of a website, and due to the team roles we implemented, only a few team members are working on the internal structures of the game, so the remaining members are able to perform effective black-box testing. Another advantage of this testing approach is that we are able to assess what the user experiences, and ensure that the current program being tested has no visible bugs that would be apparent to the user. This is appropriate for us, as the main output of our game is shown on screen graphically, which is harder to test quantitatively. The main disadvantage of black-box testing is that it can be difficult to produce test cases if the requirements are not concise, and that it is difficult to identify all possible inputs in limited testing time, which is very relevant to our project due to our short timeline. To overcome these issues, we have updated the requirements specification to ensure precision and testability of all requirements, and focused on key test cases that wouldn't be considered by other testing approaches, methods for which have been discussed by P. J. Schroeder and B.Korel [3].

The black-box testing we will perform on our system comprises of Functional testing , which "bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output"[4] , Acceptance testing which is "conducted to determine if the requirements of a specification or contract are met"[5] and System testing[6], that is performed on a complete system to evaluate the system's compliance with the requirements. These will be linked to the test cases in the 'Test Type' Column in the testing table *(see 'Black-Box testing" document)*.

Acceptance testing for the requirements was carried out as part of the black-box testing and we produced a traceability matrix *(see 'Traceability Matrix' Document)* within our testing documentation to map tests to specific requirements. This allowed us to track and confirm the testing of the software against all of our requirements.

All these methods of testing, apart from system testing, will be carried out once the appropriate module of the game is complete. E.g. once the 'Continue' button has been created and coded, it is tested. This allows for fast paced development, and an almost concurrent relationship between testing and implementation.

The system testing will be done once all aspects of software have been linked together, to ensure the final software is acceptable.

Test Report and Statistics

We conducted two main types of testing on our game, which were manual black box testing and unit testing, links to the test plan and the evidence of tests run for both can be found at the end of this document. The test table explains the test that was run, expected and actual outcome, whether or not the test passed and the test author so we could implement ownership of tests. Within our blackbox testing table we also implemented a column stating if the tests were also acceptance or functional tests. Many of our tests as well as being black-box, also fitted into the category of being either functional or acceptance and in some cases, all three. We tested that all the requirements that were physically testable were fulfilled and we documented and explained this in the testing matrix which can also be found referenced at the end of this document.

Black Box Testing

In total we carried out 46 initial manual black box tests on our game to check that requirements were met and also that the game functioned correctly. Out of these 46 initial tests, 9 tests failed, giving an outcome which was not expected from the test, and 37 passed, providing the correct outcome. We wrote our blackbox tests in the mindset that they should be mutually exclusive so that the testing is efficient and also collectively exhaustive to ensure the testing is effective.

Failed Tests

9 of our tests failed and they failed for different reasons. Out of the 9 tests, 6 failed because they physically could not be tested at this moment in the development stage. We created many of the tests before we started the development stage. Due to time constraints and the fact some features were not required for assessment 2 we decided not to implement all the features for the game that have tests included in the test table. As they were not implemented, it was impossible to test them, so this was noted down in the Actions Taken / Comments section of our table. The remaining 3 tests which failed were errors in our game that were yet to be fixed when their testing was initially carried out. The game would have still met the all the requirements with these tests failed, however it would have severely reduced the overall quality and enjoyment of the game if they were not passed. We documented the failed tests and the development team put fixes in place in the code to correct these errors. We ran the test again on the refactored code for the failed test and and all of them passed second time round.

Below are the tests which initially failed and then were rectified:

- User should not be able to move character whilst character select menu is open
- Player is always seen on screen throughout level
- Character walks to edge of the scene

Below are the tests which failed due to the fact the features were not implemented:

- Engineering level completion
- Player interacts with coins
- ‘Load’ button is clicked
- ‘Save’ Button is clicked
- Character interacts with vending machine in Ron Cooke Hub
- Character interacts with arcade game in Ron Cooke Hub

Unit Testing

We carried out a total of 26 unit tests on our game to test the core functionality and divided them into 5 different categories, depending on what overall section of the game the internal method contributes to. The tests can be found in the following directory of our games source code ../Assets/Testing. The ZIP file with all of our unit tests is also linked at the bottom of this document for reference. Unity has an in built tool for unit testing, so we used this to assist us with this type of testing and ran all the unit tests using this tool. The results of the unit test revealed that 25 out of our 26 unit tests passed on the first round of testing, which showed that the development teams code was correct, and all of these 25 internal methods functioned as they should. The singular test that did fail, was due to method calls on null values for canvas / player. This bug in the code was then rectified and the identical test was run again, which passed to prove that this error had been corrected.

To ensure exhaustive testing of the core features of the game, we unit tested all of the game management classes. We split the sections of the unit testing up into the following areas:

- CharSelectScreenTesting
- GamePropertiesTesting
- LevelManagementTesting
- PauseScreenTesting
- WorldMapTesting

Unity has a few testing limitations which made it difficult to check if conditions were definitely met. One of these limitations was that we were not able to test for listeners being assigned to buttons or nodes. This was documented in our unit testing table and the overlap with black box testing confirmed that these features were functioning correctly.

URLs to Testing Material

Unit Testing Evidence

https://sixtysixgeesegames.gitlab.io/website/assets/assessments/2/TestEvidence_UnitTests.pdf

Unit Tests (ZIP of actual tests)

https://sixtysixgeesegames.gitlab.io/website/assets/assessments/2/UnitTests.zip

Black-box Testing Evidence

https://sixtysixgeesegames.gitlab.io/website/assets/assessments/2/TestEvidence_Black-box.pdf

Test / Requirement Matrix

https://sixtysixgeesegames.gitlab.io/website/assets/assessments/2/TracabilityMatrix.pdf

Bibliography

[1] The International standard for software testing, ISO/IEC/IEEE 29119, 2013
Software Testing
[2] IEEE (2013) *International Standard - Software and systems engineering —Software testing —Part 1:Concepts and definitions* [Online] Available: http://standards.ieee.org/findstds/standard/29119-1-2013.html
[3]P. J. Schroeder and B. Korel, *Black-box test reduction using input-output analysis* [Online] 5, Sept. 2000 Available: https://dl.acm.org/citation.cfm?id=349042
[4] Wikipedia (2017, Dec, 27) *Functional Testing* [Online] Available: https://en.wikipedia.org/wiki/Functional_testing
[5] Wikipedia (2017, Dec, 13) *Acceptance Testing* [Online] Available: https://en.wikipedia.org/wiki/Acceptance_testing
[6] Wikipedia (2017, Dec, 20) *System Testing* [Online] Available: https://en.wikipedia.org/wiki/System_testing