1.0 Testing Methodology

The following report will explain the test activities and results that were performed as part of the testing of 'The Roboticon Colony of York'. The test cases for these tests along with screenshots can be found on the website here [1] and each test can be identified by their unique test ID.

Research was conducted into various means of conducting thorough and efficient testing during early phase 2. From this we were able to devise a testing method based on information amassed from SWEBOK V3 [2] and various online sources/blogs. This methodology consists of both black-box (acceptance) and white-box (unit and integration) testing methods to achieve a balanced approach that ensures reliable results.

In this methodology unit testing is completed during implementation, this precedes integration testing to ensure that the target functionality is performing as expected and without errors. Thanks to our use of travis these tests are automatically run after each push to the repository ensuring that the state of the test (i.e. passing or failing) will always be reflected in our testing log. Following this we proceed to integration testing, this comprises of reevaluating our unit tests and ensuring that class interactions proceed as expected. We adopted two approaches towards integration testing, big bang (integration testing of all classes) [3] and what we call progressive testing (integration testing on classes as they're developed). Progressive testing was completed during development, this process went undocumented as we believe it would have slowed development. For this reason we adopted our second approach of big bang testing, allowing us to produce adequate documentation. This also allow us to spend all of our time focusing on producing test documentation and addressing any issues that may arise. Lastly, following the completion of the majority of game features we proceeded to conduct acceptance testing. This comprised of ensuring that requirements were met by the implemented solution. This was accomplished through assigning strict acceptance criteria to each requirement and play testing to see if our solution reaches the specified criteria. Due to the nature of acceptance testing this was conducted once development was completed.

During implementation and initial testing our development team discovered an issue related to conducting unit tests on a game that involves networking. Due to the implementation there's a certain portion of the games functionality that cannot be unit tested. While this has proven to slow our testing progress, all of these special cases could be simply tested through debugging e.g. printing messages, sending text to the server etc. This may be also be handled through integration testing.

We have been storing the results of integration and acceptance tests within separate Excel documents in a clear and concise format, these can be found by following the link above.

1.1 Testing Justification

Unit testing allows for the testing of methods of well defined behaviours. This provides a means of testing the lowest components of our program ensuring that we produce a fully functional base on which to expand and develop more complex functionality. This also provides a means for developers to monitor their code more efficiently and ensure that any changes or new additions does not affect previous functionality.

Integration testing is used to test the interactions between multiple objects. Our testing methodology requires this manner of testing as there are few other means of testing networked games. Without such a testing method we would not be able to comment on the correctness and completeness of our solution.

Acceptance testing is required to ensure that our implemented project meets the requirements as set by the customer. This provides a means of conducting a traceability analysis and helps keep the developers on task ensuring that no redundant functionality is implemented.

We believe the combination of these three testing methods will allow us test the whole project at all levels and ensure that our code is as error/bug free as possible.

2.0 Test Summary

Our tests aim find as many failures as possible so that we can find solutions to make a fully functional and playable game.

We conducted unit and integration testing throughout implementation to check that target functionality performed as expected and without errors. Unit testing involves testing elements of the game that are separately testable in isolation [2]. Following successful implementation of game features, integration testing was conducted to verify the interaction among software components. Any elements that did not work as expected were immediately addressed. The ongoing testing of the game meant that it was easier to find the problem that could be fixed as soon as possible.

Acceptance testing was conducted when the majority of the game components had been implemented and passed the previous unit and integration tests. This checked that the requirements had been met by the implemented solution.

3.0 Test Results and Analysis

3.1 Unit Testing

32 unit tests were conducted with 100% passing.

3.2 Integration Testing

5 integration tests were conducted with 80% passing meaning that a total of 1 test failed, which was later addressed.

Test I4 failed as we were still able to place a roboticon after the round timer expired. This issue was resolved through resetting a variable within Player class at the start of each phase thus ensuring that roboticon placement could not be achieved at the end of the phase.

3.3 Acceptance Testing

19 acceptance tests were conducted with 79% passing meaning that a total of 4 tests failed. Tests that passed without any issues meant that no action needed to be taken and the requirement had been satisfied. To satisfy the customer's needs any tests that failed have been addressed to the best of our ability, as discussed below.

Test A7 failed when the user was unable to open the market from the button on the UI overlay. To enable this test to pass we added a line of code to make sure this worked as it is an integral feature of the game for the user to access the market. With test A7 now fixed this has increased our passing rate to 85%.

With the current implementation tests A13 and A15 also failed their acceptance tests. There is no end game condition or win condition for our game because this was not needed by the customer. Hence no action needs to be taken by the current team. To implement this feature all that is required is an extra state within the GameController that is accessed once all tiles have been acquired (which would also need to be implemented). Following this they will also have to implement the logic that calculates the winner based on their end game resources which is a relatively easy task.

Test A18 failed because there was no notification when there is no available game for the user to play. This would require implementation of some form of lobby system that would be continually updated. This task may be difficult as it would require the development of networking functionality which is not currently supported. Hence no action was undertaken because it is not a key feature of the game and it does not impact gameplay.

4.0 Conclusions

In this section we proceed to discuss the completeness and correctness of our testing methodology and the deficiencies of our system.

4.1 Unit Testing

As discussed in section 1.0 it is impossible to implement exhaustive unit testing in Unity due to the nature of GameObjects and their attached scripts. As such only a subset of all possible unit tests can be achieved, however we believe that any major issues regarding other classes have been addressed during progressive integration testing. We therefore believe that our unit testing is reasonably complete.

In regards to the correctness of these unit tests they are functionally correct, meaning the unit test successfully assert whether the functions produce expected output results from erroneous and valid input data.

4.2 Integration Testing

While the completeness of our integration tests may appear to be lacking, we believe that the majority of class interactions have been tested within our acceptance tests, leaving only networked interactions. Due to the simple nature of the game not much networking is required in terms of data transfers and client calls which are usually executed in tandem, meaning through testing a singular feature we are effectively testing multiple interactions. In addition as there is numerous factors (such as connectivity and bandwidth) that may affect the game it is impractical to test for all of these eventualities.

4.3 Acceptance Testing

The acceptance testing is complete as all testable requirements are addressed, therefore if there are any issues with the completeness of this testing it is solely attributed to the completeness of our requirements.

Bibliography

[1] SEPR, "Acceptance Tests," [Online]. Available: https://seprated.github.io/Assessment2/testing_evidence.zip. [Accessed 22 January 2017].

[2] P. Bourque and R. Fairley, “Guide to the Software Engineering Body of Knowledge,” IEEE, 2014, pp. 84-97.

[3] TutorialsPoint, “Big-Bang Testing,” [Online]. Available: https://www.tutorialspoint.com/software_testing_dictionary/big_bang_testing.htm. [Accessed 22 January 2017]

[4] SEPR, “Integration Tests,” [Online]. Available: https://seprated.github.io/assessment2.html. [Accessed 22 January 2017].