Software Testing Report

Methods and Approaches

We used an agile approach to our development process, and have generally conformed to the methodology set out in assessment 1 [1]. We have had regular scrums and discussions as a team, and have also split into 2 smaller groups for the development process - a software development team and a documentation team. This has made us able to have more frequent meet-ups as less people in a team equates to a smaller chance of a commitment getting in the way of a meeting. As an agile development team we also solved smaller problems and units in a frequent manner, utilising Git [2] to create new branches for smaller features before creating a pull request to merge the changes into our main Development branch. This method of version control ensures that there is little chance of data loss, and we used Travis-CI [3] to automatically build tests to ensure the safety of parent branches. Unit tests were created and ran before every major commit to test the old features to ensure that they still worked correctly.

Black Box Testing

Black Box Testing [4] is being used as part of the testing team is separate from the implementation team making it more efficient for us to assume the tester does not have knowledge of the internal structure of the items being tested.

White Box Testing

White Box Testing [5] is also being used to ensure that each individual method is functioning as it should independent of the external running of the game.
We can access testing tools through the unity asset store which will give us access to unit,integration and assertion tests. A further description can be found below.

Acceptance Testing

Acceptance testing for our project will ensure that all our requirements are met and the game is functional as a whole. This is reflected by the associated requirements in the Manual Test table [6].

Unit Testing

We have chosen to Unit test our scripts for our game and the Unit Test table [6] shows the testing scripts that are run and the individual tests within those scripts. These tests can be run everytime the code is updated ensuring that it has not been broken by the new additions. This is especially valuable for future-proofing the project. This is due to the tests being able to be re-run at any time through the Unity Editor after changing any code.

The files for the unit tests can be found here:
https://github.com/HenryCadogan/Clued-Up/tree/master/Game/Clued-Up/Assets/Editor

Report on Tests

All tests were successful.

Acceptance Testing

This shows the correctness of the program on a whole because of the acceptance tests performed proving the correctness of the game features. This means we have made the game to our requirements as shown in the results table. These tests also future-proof the sections they have tested in that they can be replicated when new features are added in the way described. They do not however future-proof the rest of the requirements which were not required for this assessment, additional tests will have to be written for this purpose.

Unit Testing

Not all of the methods have been tested in the unit testing as some are more well suited to being tested via manual testing but playing the game and observing the results. This is due to the whole game needing to be run for elements to be instantiated which is not feasible to do with this type of testing.

In our code we have individual classes for the different furniture in the rooms. The game was made like this and we have subsequently realised that they should be refactored into an abstract furniture class in the future given more time for the project. Due to this we have opted to not test these classes further than in the manual testing.

For a more complete method of testing we could have opted to test the aforementioned features anyway. We could have also done more in depth testing for the methods that we have tested.

References

[1] H. Cadogan, S. Davison, T. Fox, W. Hodkinson, C. Hughes and A. Pearcy, "Method Selection and Planning", *Wedunnit!*, 2016. [Online]. Available: http://wedunnit.me/webfiles/ass1/Plan1.pdf [Accessed: 10- Jan- 2017].

[2] “Git”, *https://git-scm.com* 2017. [Online]. Available:*https://git-scm.com* . [Accessed: 24- Jan- 2017].

[3]"Travis CI - Test and Deploy Your Code with Confidence", *Travis-ci.org*, 2017. [Online]. Available: https://travis-ci.org/ [Accessed: 24- Jan- 2017].

[4] "Black Box Testing – Software Testing Fundamentals", *Softwaretestingfundamentals.com*, 2017. [Online]. Available: http://softwaretestingfundamentals.com/black-box-testing/ . [Accessed: 24 Jan- 2017].

[5] "White Box Testing – Software Testing Fundamentals", *Softwaretestingfundamentals.com*, 2017. [Online]. Available: http://softwaretestingfundamentals.com/white-box-testing/ [Accessed: 17- Jan- 2017].

[6] H. Cadogan, S. Davison, T. Fox, W. Hodkinson, C. Hughes and A. Pearcy, "Tests.pdf", *Wedunnit!*, 2016. [Online]. Available: http://wedunnit.me/webfiles/ass2/Test-table.pdf . [Accessed: 24- Jan- 2017].