Testing and Evaluation method / approach

To test our product and ensure that its code was of an appropriate quality, we used two testing methodologies:

- The first method we used was unit testing, which allowed the team to write small tests to ensure that methods/units, within the code, function correctly. Our unit tests do not just cover 'the happy path' where nothing goes wrong, we also test that exceptions are thrown when they should be. When done correctly, this allows the team to evaluate sections of code to ensure they both work correctly and meet the requirements. This helps to increase the maintainability of our code, and to increase our confidence our code performs as expected. JUnit[1] was used to facilitate this, along with a continuous integration solution called Travis[2]. Travis would run Junit's tests each and every time a change was made to the repository, essentially providing live updates if tests failed when something was changed. This is both helpful for the developer, alerting them to the fact that something they changed had broken one of the requirements, and ensuring that the code is working as intended.

- Unit testing, even with complete coverage of your code, does not mean your code is perfect. There still could be cases where a user could provide bad input and cause your program to crash. To help alleviate this issue, we also used black box testing, which involves only giving the solution an input and looking at the output. Unlike unit tests, black box tests are performed by a human. This means it is easier to test things like graphical user interfaces or requirements which span more than one unit of the game. However, as black box tests are performed by a human you risk introducing inconsistencies when testers have varying opinions of whether the test is satisfied. Therefore we wrote our black box tests as clear, unambiguous scripts which the tester can perform. This allowed us to re-evaluate our product with respect to the requirements every sprint (and more often if needed) in a consistent manner.

While not explicitly stated by the requirements, the code needed to be of a certain quality. This quality involved:

- Ensuring the code is complete and works correctly
- Ensuring the code conforms to standard java conventions
- Sensible naming schemes
- SOLID[3] code

This quality is important to ensure that our team and future teams that may work on this project, work with a well maintainable, efficient and working solution. These testing methodologies, along with periodic code reviews, help us to achieve these goals by alerting us to errors and other issues.

To evaluate how well our solution met the brief, and thus the requirements, we used an objective-based evaluation approach. This approach looks at each requirement and produces a set of tests to analyse to what extent the requirement was met. This methodology is extremely useful to us as:

- It is centred around evaluating the extent to which each requirement was met by the end product, which was a core issue to both our team and the stakeholders.

- Objective-based evaluation works best in “tightly focused projects that have clear, supportable objectives.”[4], which, due to our detailed list of requirements, is very suitable for our project.
- Our testing methods, which were implemented to ensure quality of code, are already partially implemented as our requirement testing.

When the tests are run from the two testing methodologies their results are analysed to evaluate the extent to which a requirement was met. A requirement was; fully met if all related black box and unit tests pass, partially met if any but not all relevant tests passed and not met if no relevant tests passed. With this method of measurement we can evaluate how effective the solution is.

When we inherited the project from our predecessors, we found that our approaches to testing methodologies were almost identical. We both used Unit testing and a form of manual testing to ensure that the system works as intended and meets the requirements. However, we, in our previous project, used Unity Test Tools where as our predecessors used Junit. Both systems are very similar and do essentially the same thing, however NUnit is for c# and JUnit for java. We extend the existing JUnit test suite before making any additions or changes to the existing code. We did this so that we could have greater confidence that our changes were not breaking existing functionality. We increased code coverage within the behavioral layers from 32% lines covered to 85% lines covered. Then the unit tests were built upon when required, to test new or edited code. New additions to unit testing were labeled with “Added by JBT” in the comment string of each new test, and “Changed by JBT” was added in the comment string of each test that we changed. These changes can be found in our testing classes here. We also found that their existing unit and black box testing was similar to our own, and with some small tweaks in some places we adopted their existing tests. Here is a list of changed or added black box tests:

ID	Screen	Old Test	New Test
5	Buying a Plot	If the player clicks on the buy plot button and has sufficient gold, the plot will gain a coloured border (blue for player 1, red for player2)	If the player clicks on the buy plot button and has sufficient gold, the plot will gain a coloured border (blue for player 1, red for player2, pink for player3 and orange for player 4)
36	Start Screen		“-” button for AI players is disabled when AI player count = 0 and becomes enabled when AI player count > 0
37	Start Screen		“-” buttons for human players is disabled when human player count = 1 and becomes enabled when human player count > 1
38	Start Screen		There can only be a max of 4 human or AI players, and the “+” button becomes

			disabled when the max is reached.
39	Start Screen		You cannot start the game with only 1 player
40	Start Screen		You cannot have less than 1 human player.
41	End		End of game Score screen correctly scales with number of players
42	Multiple		Turns cycle correctly and chronologically through the number of players.
43	Effects being imposed		When a player gets the chancellor random event, they wait 15 seconds instead of trying to click the chancellor. The event should end automatically.
44	Effects being imposed		When a player gets the chancellor random event, they complete it by clicking on the chancellor. This should give them an extra 20 points for each chancellor caught on the end of game screen.

These changes were made to accommodate the change in requirements introduced in assessment 4. These changes to both the unit testing and black box testing mainly consisted of changes to the main menu UI, player and AI classes, as these were the main areas in which the program was obviously affected.

How the final product does, or does not, meet the requirements.

We used our objective-based evaluation method to measure the extent to which each of the product requirements were met. A requirement was; fully met if all related black box and unit tests pass, partially met if any but not all relevant tests passed and not met if no relevant tests passed.

As you can see here all of our unit tests. Also, you can see here that all of our black box tests pass. All of our requirements but one are covered by unit test or black box tests. One requirement (#36 Auction) did not have tests written for it, as it was not implemented as per the agreement with the stakeholder, as it was not in the scope for this assessment period.

See our full list of requirements here.

Bibliography

[1] JUnit, [Online]. Available: http://junit.org/junit4/ . [Accessed 15 April 2017].
[2] Travis CI, [Online]. Available: https://travis-ci.org/ . [Accessed 15 April 2017].
[3] Robert C. Martin, “The Principles Of OOD”, http://butunclebob.com/. [Online], Available http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod [Accessed: May. 1, 2017]
[4] Daniel L. Stufflebeam, "Evaluation Models," *New Directions For Evaluation,* Issue 89, pp. 17-18, Spring 2001. [Online]. Available: http://www.wmich.edu/sites/default/files/attachments/u58/2015/Evaluation_Models.pdf [Accessed: April 17 2017]