Testing methods

The original teams testing report can be found here: [1]

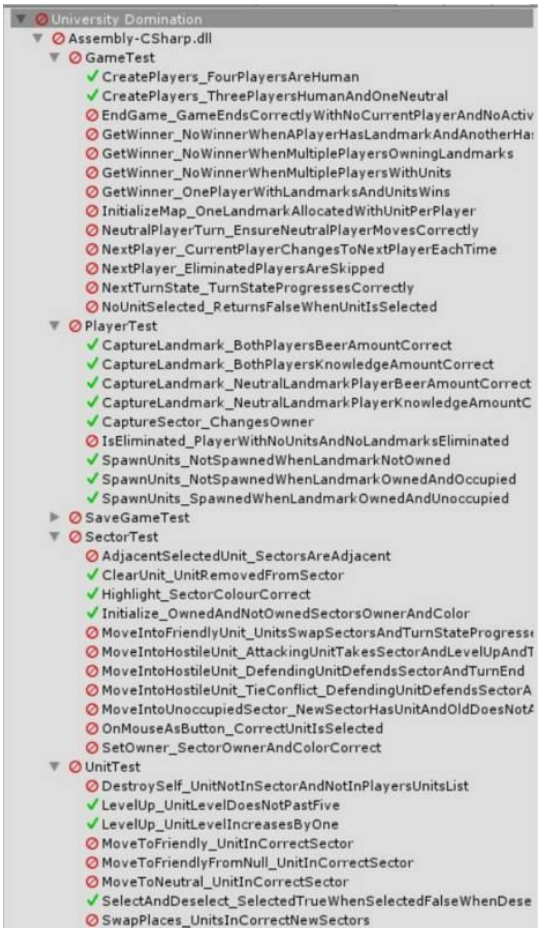
With the project we picked up being built using Unity, it only made sense to carry on making use of Unity's built in test tools for testing the project wherever possible. Tests have been written in separate testing classes, related to classes used in the game, e.g. Unit and UnitTest classes. Unity handles testing within the editor, making it very easy for us to quickly test whether or not a feature is operating correctly at the click of a button, along with useful error messages should it not. All of these Unity tests made use of assertion tests, where the actual outcome of the testing script was compared with the expected outcome. A test would pass if the two were the same. Should a test fail, we would go back and first inspect whether the test is operating correctly (the test was calling the correct methods at the correct time with the correct parameters). If the test was correct, we would then use any error messages to try resolve the issue and only pushing the new code to the master branch once we were satisfied everything was as it should be. Code that has passed testing is then refactored and tested again to make sure the new, cleaner version still works as it should. Tests were also refactored when necessary. All tests were written in a consistent style to keep the code easier to read and understand. We structured our tests names like [UnitOfWork_StateUnderTest_ExpectedBehavior] [2] to make it immediately obvious what the test was for and what was supposed to happen, streamlining our development. Unit tests were written to test the back end aspects of the game. We had a list of core tasks that the game must perform to meet Assessment 3 deliverables which we followed when writing and testing the game as development progressed. By following this easy to read and understand checklist, it was clear what was being coded and tested. Each method found in the game scripts on the whole had at least one test to ensure that part of the game functioned as expected, allowing us to gain a more complete understanding on how complete our project was. The actual tests can be found in the 'Assets, Unit Tests' within the Unity project or our github link [3] and must be run from within Unity's test runner. We also needed to test the UI element of our game; these tests were carried out differently to the unit tests. Unity's testing features would have been ideal testing the UI, however many parts could not be tested in this way, due to them needing a human's opinion, to make sure the game is displaying information correctly e.g. information is displayed where it should be on the display. These tests were undertaken by hand using black box methodology and a table for tests was completed to keep a record on how the UI was progressing during development. The actual updated tests can be found in the "Black Box Tests" table here: [4]

Testing report

Unit tests

After testing each method in each class for the game using Unity's built in testing tools, we found that only 16 unit tests now pass after out continued implementation of the project (Fig. 1). Having picked up the previous teams tests many of the tests have now not been able to pass due to some difficulty we had with using the previous teams architecture. We troubleshooted the error for a few weeks but could not find a suitable solution as to why this was happening. The failed tests however are not representative of how functional the game is as through play testing the game all functionality works as expected. The updated unit

tests table for Assessment 3 can be found at [5]. Figure 1: A screenshot of the test results of all 43 unit tests in Unity's test runner



Black Box Tests

The Black Box Tests were conducted manually to confirm information was being displayed correctly, and all black box tests passed excluding one. Since we have fully implemented the game there were a lot of black box tests to carry out and update from the previous team. We completely redesigned the GUI and added a start menu along with multiple different dialog boxes. While not as time-efficient as automated testing, manual testing like this was as reliable and more intuitive in the case of testing graphical systems like the GUI. The black box tests for Assessment 3 can be found at: [4]

Additional Play Testing

This is something the previous team never really did. We decided it was particularly important in this assessment as we are aiming to implement the final version of the game, hence everything needed to be working as perfectly as possible. Many previously unobserved bugs were identified this way such as:

- The neutral player being able to attack owned sectors
- The mini-game being triggered more that once when discovered on the same sector
- The neutral player was able to trigger the mini-game when moving across sectors

All the bugs were therefore resolved due to this play testing and would have most likely otherwise been missed.

References

[1] SEPR “Testing Report” Laser Dolphin Games [Online]. Available: https://sepr-team-margaret.github.io/content/Test2.pdf

[2] R. Osherove, "Naming standards for unit tests," Roy Osherove, 3 4 2005. [Online]. Available: http://osherove.com/blog/2005/4/3/naming-standards-for-unit-tests.html. [Accessed 9 12 2016].

[3]SEPR “Testing” Github [Online]. Available: https://github.com/RiskyDevelopments/UniversityDomination/tree/master/Assets/Unit%20Tests

[4] SEPR “Black Box Tests Updated Table” Risky Developments [Online]. Available: http://www.riskydevelopments.co.uk/documents/UpdatedBlackBoxTesting.pdf

[5] SEPR “Unit Testing Updated Table” Risky Developments [Online]. Available: http://www.riskydevelopments.co.uk/documents/UpdatedUnitTesting.pdf