Testing Methods

The system was tested with an incremental approach. The Scrum framework utilised by the team relies on dividing development up into short increments which lead to white box testing (detailed below) of each section of the code as soon as it's finished, which is useful for ensuring confidence in every aspect of the code as well as being in line with our methodology.
When the system neared completion black box testing (detailed below) was used, basing tests off our requirements and fit criteria ensures that the system confers with the requirements we set out.

For some of the tests we will use a test group. This group shall consist of 10 participants who will be told to complete several tasks within the program and fill out a questionnaire when they are done. This will be done as a means to test the more abstract requirements of the program such as aesthetics.This benefits our project by making sure that the system is functioning properly at each and every point of development - if any issues arise then these can be dealt with swiftly by a developer and tested again to ensure full functionality. Each test will be fully documented with images, code snippets, expected results and comments so that any other developers that want to check or re-run these these tests are able to, and can have a clear understanding of each section. Also included will be the fit criteria for that test, and how we achieved those criterias. This is to make sure for certain that each of our requirements is met. If part of the test fails or is not satisfactory, tests will be shown until that specific requirement is completed. Tests will differ between writing debug code snippets to test functionality and simple runthrough tests to make sure the system works as planned. Aesthetic functionality for the front-end User Interface will also be tested, to make sure all buttons are mapped and placed in the correct way.

Our test plan shown below makes sure that all tests are understandable, repeatable and coincide with our fit criteria.

White Box Testing

White Box testing is appropriate when initially writing the code, as the tests are drawn from the requirements, this ensures that the code written will perform the required function. At some points White Box testing was used after an error was found with Black Box testing, to determine exactly where the error in the code occured.

To implement white box testing we used the debug features Unity provides to assess what was happening within each function. Once it was proven that the functions had the exact effect needed we removed the debugging code so as not to completely flood the console with data.
Here is a link to a table that briefly describes how each function was debugged, what the purpose of that function is and what script it is in.
- Debug Test spreadsheet: [link]

Black Box Testing

For this part of the project, when a feature was implemented a series of Black Box tests were devised to ensure the feature worked correctly, the tests were given a 'Brief Description', 'Fit Criteria and User Requirement', 'Expected Result', 'Actual Result', and 'Evidence for the Actual Result'.
The Black Box tests we used involved: Input which should fail, which we labelled erroneous; input which should work, which we labelled normal; and input which should work, but is at the extremes, which we labelled extreme data. These tests were written as they encompass anything which the user would be able to do from within the interface.

When the Black Box tests were completed, the results were stored in a shared google document, highlighted in red, and the person who wrote the feature was notified to fix it. Once the feature was fixed, the tests were run again to ensure the feature was working correctly.
Black Box testing, is appropriate at this stage, because it emulates the game as the user is operating it, it also tests how the game works in the environment of the users computer. Black Box testing also ensures that the interface correctly uses the functions that have been developed. Should the function work perfectly, it would pass the white box testing, however if the interface doesn't correctly call the function, the White Box Testing wouldn't catch the bug.

Black Box testing is useful, as the person who wrote and completed the tests, was not the person who wrote the code, another set of eyes results in errors being caught, that would have not been caught by the programmer.

Testing report

General Test Plan

We ran tests to make sure that units are assigned correctly, something crucial that would skew the game balance if a player had incorrect units. For this we used multiple tests with different objectives which looked at how Units are distributed throughout the entire game. Territories that are attacking, being attacked and nearby will be checked between the expected and actual results to see whether units are correctly assigned and removed. These tests all came back without any errors - showing that the unit distribution system is correctly simulated every time the software is used.

We ran a test to make sure that territory is assigned correctly at the start of a game, something crucial for correctness, as if not that issue would skew the game balance if a player started with an incorrect amount of territories. For this we used a test which looks at how territories are distributed throughout the start of the game. These tests all came back without any errors - showing that the territory distribution system is correctly simulated every time the software is used.

Tests were also run to make sure that territory areas were correctly displayed on the in-game map. Without this it would leave the system with an incorrect map of the University, which would go against our requirements. Multiple tests were ran to make sure each of these locations correctly displayed on the map. Every single test came back without error, showing that our map is loaded correctly every time the game is played.

Tests were also run to check that conflict was resolved correctly each time a territory decided to enter conflict with an enemy territory. This was done using multiple tests, which check conflict at specific points in a game to make sure that results are correct compared to what's expected. Territories that are attacking, being attacked and nearby will be checked between the expected and actual results to see whether conflict results in the expected outcome, with randomness being accounted for by the test to have a complete test. These tests all came back without any errors - showing that the conflict system is correctly simulated every time the software is used.

Unit tests involved with conflict that were erroneous such as attempting to attack an adjacent territory with a minus or null amount -- such as -1 and 0 -- were also run. If any of these tests ran without error the system wouldn't be correct and thus flawed. All these tests came back with a correct error message - this showed us that our system correctly relays error messages when there are any issues within the unit input system.

Further Testing

Each of our team members put time into playing the game as expected, trying to find bugs outside of the written tests, because no set of tests can encompass absolutely everything. Also once we had fixed all outstanding bugs, we redid all black box testing to assure that fixing one feature hadn't affected another, but no evidence was provided if it worked, as we already had evidence from the previous black box test.

Test Results (including statistics)

We used Test Driven Development, so when it came to running final tests, very few errors presented. Of 58 initial Black Box tests, there was one failure, the errors were those that were difficult to catch with the initial White Box Testing.

The errors caught were:

(G6.5) No error message or popup was shown when a player entered a string ("a") into the text field when selecting the amount of units to move. Nothing happened which means there was no incorrect move but no information regarding the players mistake was given either.

Evidence review

For all viable sections of the black box testing, we provided evidence, in the form of screenshots. Once the initial tests were complete, a fresh set of eyes reviewed the evidence. The evidence review was useful given that a few of the Black Box tests were questionable, the initial expectation of what was defined as an acceptable result was incomplete.

For tests 13.3, 13.4, 17.3, and 17.4 passing the test was defined as “An error message being presented, and the action failing to take place”. This initially seemed a reasonable successful outcome, however the wrong error message was presented and due to carelessness and it still meeting the initial requirement the test passed. Upon review of the evidence, the reviewer found the bug and fixed it, the test was redefined to be more thorough and similar tests were given greater scrutiny.

GUI Testing

We wrote tests for features that had to be implemented during this section of the assessment, including parts of the user interface, all but 1 of the tests passed, but the error was rectified, so we are confident in our user interface so far.

Testing Material

A link to our testing material, the evidence and results gathered from following our testing design, can be found here: [link]

Traceability

A spreadsheet showing the mapping between requirements, testing, and the implementation can be found here: [link]