Planned Testing Methodology:

We decided our project would have a testing method matching our constraints, time, budget, requirements, risks, etc. Budget is no issue but our time is limited, as our project has 29 original main requirements with some being more important than others, and considering some will split into smaller requirements of their own, we won't be able to test everything, so we can prioritise our tests on areas and requirements that would be critical to the system, e.g. making sure the screens transition properly depending on the game state, requirement 1.8 in the updated requirements, or making sure that the user is able to progress properly without interaction issues, requirement 1.4. All come under large risk areas, but there are testing methods in place to make sure they can be tested appropriately.

Though we have based testing around requirements, should we still have a problem indirectly linked to the requirement, we refine the requirement instead. The order we follow, Unit Testing > Integration > System > Acceptance, Integration and system can come under white box, however in this scenario I refer to them as black box, which covers acceptance testing as well as those two, not only was it recommended by a member of staff but also by peers we spoke to. Our tests have to be refined due to the hierarchy of the game, and it is added to and refined, rather than all coded from the start.

1. We will begin our testing with a short static testing, going through existing documents such as requirements and early code, without executing anything, noticing structural faults with our foundation, earlier it’s found the better.

2. We use unit testing to test that each function works, this generally comes under white box testing. This is simpler in marking down our progress as there is a finite and recorded number of functions to test. When this is complete for a section we can move on to black box testing.

3. A large part of our method of testing will be black box testing as it is better suited for game development, testing more how it functions without knowing it’s internals, in game development the user relies not on the knowledge of internals, but more seeing how it would function when actually used.
    a. It begins with integration testing, grouping individual unit tests that rely on each other to see if their interactions themselves have faults, a hybrid approach seemed the most appropriate method as we were not sure how the parts would interact, combining a top-down and bottom-up approach. These however mean it is important to have documents detailing module interaction.
    b. The second is system testing, taking place after integration, in some examples it is used as a final test however since we’re doing game development programming is our last, testing that everything works together, based on our requirements, this will be important as we may have unforeseen errors up to the integration testing stage.
    c. Acceptance (Alpha) testing is our final set of tests, it’s ad-hoc so it doesn’t follow a specific procedure, not done by us but by users, people we find who can test the system.

Test Results:

We ran twenty-two black box tests on functionality of the game and fifteen Unit tests, with both we covered most areas of the game, with unit testing focusing more on the specifics of calculations and results.

Black box testing had the largest variety of testing, we had many failures that were soon corrected, many of which were just minor tweaks below I've listed the results from the before testing runs and the after testing runs, nine Black box failures were fixed and one Unit Test.

Black Box:
Initial Test run: 10/22 Fails
Consequent Test run: 1/22 Fail
Website Link:
https://davidjnorman.github.io/SEPR/assessment_2/BlackBoxTestingEvidence.pdf

Unit Test:
Initial Test run: 2/15 Fails
Consequent Test run: 0/15 Fails
Website Link: https://davidjnorman.github.io/SEPR/assessment_2/UnitTestingEvidence.pdf

The only test that ended up failing was related to an unimplemented piece of code so this had no effect on the product as a whole, all other test failures were relatively simple to correct. To enable this test to be passed the Hud needs to be moved from the main screen to the main game class and some tweaks to how it's called, this would allow the hud to be constantly updated through battle as well as sailing which allows the values to be evidently changed.

Our unit test coverage is lower than hoped, covering only four functional sections, player, building, battle and quests, this was due to LibGDX; LibGDX and Unit testing has severe problems which caused many of my tests to fail which stops them from being passed, I avoided this issue when I ran my tests by commenting out the LibGDX portions, and in some cases had to make a function to run only the related code. In the end it's identical functionality with the libGDX features removed, the problem with this is it means that Unit Testing isn't easy. Links to Unit test files can be found in the Zip file below:

https://davidjnorman.github.io/SEPR/assessment_2/TestScripts.zip

Many of the tests, more specifically black box tests were based on requirements one way or another, for example screen transition tests fell under requirement 1.08, the comparison of tests ran to requirements can be found within my traceability matrix with the link below:

https://davidjnorman.github.io/SEPR/assessment_2/TracabilityMatrix.pdf

This matrix shows all information related to each requirement and compares the test IDs to them, it does this by highlighting corresponding areas in blue where the test fulfills the requirement in some way, only two requirements of the sixteen could not be implemented and this was due to time constraints for the leaderboard, and that the implementation of campus east was left for the second assessment which we left as not to exceed the brief for the second.

The matrix in terms of general coverage is almost complete, with an accurate range of correctness throughout the tests, there were some more tests we could have performed by testing the particulars of the code with player stress testing it and finding any errors they could but none appeared during our playtests.

To provide a more complete range of testing and coverage, as well as a plan for later assessments is to learn how to unit test with junit, I believe it is possible but will require research to test, once that is done we will be able to more finely test the other aspects of the game as well as areas that might not be clear, i.e. through visuals, due to it functioning relatively as expected we may miss any miscalculations.

The failed black box tests include;

1. Attributes loading improperly, which was fixed by comparing how it was loaded.
2. HUD button problems which was corrected by pushing group changes.
3. Health reduction failing to happen during battle which was fixed by using a simpler code implementation.
4. Unique enemies, for some reason during battle we ended up with an issue where everyone lost health, originally our worry was that it wasn't getting a unique enemy each time however this was actually an issue of all enemies being attacked at once.
5. The same problem applied for buildings however it came with the addition of every building being one hit killed, this was fixed by reverting to a previous state and recoding the same implementation to find the issue.
6. We had an issue where it wasn't clear where to go to fight a college, by creating a small port asset it became much more evident.
7. The game not being evident as the university of york was fixed with some asset changes.
8. The win screen hadn't displayed properly due to an issue with how we called a specific main game attribute.
9. Finally we had an issue with ship rotation, the ship could rotate but the calculations were off, another member of the group had been working on the calculations and hastily resolved the issue.

All of which were resolved, aside from the quest loading test which was deemed unimportant as it was based on implementation that we intentionally left out.

Appendix:

Main: https://davidjnorman.github.io/SEPR/assessment_2/BlackBoxTestingEvidence.pdf
Backup:https://drive.google.com/a/york.ac.uk/file/d/1EBxLx1MtSfbSeLhzbl3f6kjRVeDkiUBJ/view?usp=sharing

Main: https://davidjnorman.github.io/SEPR/assessment_2/UnitTestingEvidence.pdf
Backup:https://docs.google.com/document/d/1rVMZaJyyN_zpJA2wxnuqFS7KitbLA100GfAe5Fni6L4/edit?usp=sharing

Main: https://davidjnorman.github.io/SEPR/assessment_2/TestScripts.zip
Backup:https://drive.google.com/a/york.ac.uk/file/d/10GYtZ-ptGbuSD4Sq36NpWJAyiPMjN_Cu/view?usp=sharing

Main: https://davidjnorman.github.io/SEPR/assessment_2/TracabilityMatrix.pdf
Backup:https://drive.google.com/a/york.ac.uk/file/d/1bAGFbl0g6IIijJ-dqmeeIZAxQtZUaS_6/view?usp=sharing