# Software Testing Report

## Testing Methods

In respect to the use of Scrum development methodology, our team has decided to adopt an incremental approach in testing, where tests are carried out during the development process as soon as a feature is finished during sprints. In addition, we also implemented a release testing phase towards the end of the development period to ensure all the initial requirements are met as well as complete integration between different features. To accomplish this, our team has decided to make use of Grey-box testing as it is composed of both Black and White box testing, allowing for very diverse and flexible approaches.

### White Box Testing

White-box testing, in which internal knowledge of the code structure is often required, was carried out mostly during the development process by our developers (team members who wrote the code) each time a new feature is implemented into the game to ensure it works correctly, matches respective requirements and integrates well with existing components. It is also sometimes used after bugs were detected through Black-box testing to determine the exact issue. Initially, we planned to utilise unit testing for this purpose, and although JUnit tests were used to perform unit testing on some of the different methods in our code **[1]**, we found that JUnit was not compatible with libGDX (the Java game-development framework we used to create our game) by default, and had to carefully construct our testing environment to not include anything which utilised libGDX. For instance, we had to create a new Ship constructor which did not initialise any textures, as it was causing errors with our unit tests. This meant that we were unable to test too much of our game's code using JUnit tests, directing us to focus on designing tests that worked through running the game and manually adding debugging code to our code to assist the process wherever needed. These are presented as a table in the testing material, with a requirement reference, description, method, predicted and actual result and whether it passed for each of the test **[2]**. Due to the compatibility issues between libGDX and JUnit tests, these tests are appropriate for the project so far as they are the most reliable alternative to test our methods in libGDX, however they are time consuming to run due to simulating via playing, and sometimes code needs to be edited to speed up the process. For example, it would be too time consuming to level up the player enough to be able to capture a college, therefore boss difficulty had to be reduced for testing.

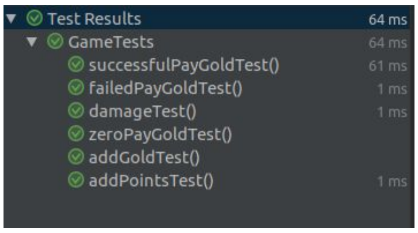
### Black Box Testing

Black-box testing, due to its nature of abstracting away from implementation details, was less rigid and therefore carried out at the end of the development process when all the currently required features had already been implemented. We had our tester (a member of our group not heavily involved in writing the code), played through the completed game and noted down any errors, bugs or unexpected encounters **[3]**. This approach is appropriate for the current stage of the project as it allows us to mimic tests from the user's point of view and provides subjective inputs from a different perspective in order to find errors that otherwise would have been overlooked by someone more involved in the coding process, particularly GUI-related errors. In addition to this, at the end of the development process, we also matched our game features with the initial requirements and fit criterias to check whether all of them had been met. While doing this we only cared about how these features worked and disregarded their implementations, following the principal of Black-box testing.

## Results

We conducted three main types of testing on our game: Unit testing, White-box walkthrough testing and Black-box walkthrough testing. Links to the testing materials can be found at the end of this document.

### Unit Testing

| Name | Description | P/F | Evidence |
|---|---|---|---|
| addGoldTest | Testing if addGold method in player class updates player's gold variable correctly. | P 🟩 |  |
| successfulPayGoldTest | Testing if method payGold(amt) returns true and changes players gold variables correctly, test ensures player has enough gold for upgrade. | P 🟩 | |
| failedPayGoldTest | Testing if method payGold(amt) returns false if player can't afford the upgrade. | P 🟩 | |
| zeroPayGoldTest | Testing if method payGold(amt) returns true if player's gold is equal to the cost. This tests an edge case. | P 🟩 | |
| addPointsTest | Testing if method addPoints in player class updates players points correctly. | P 🟩 | |
| damageTest | Generate a dummy ship, and simulate damage(amt) method. Test if method correctly changes Ship health. | P 🟩 | |

```
▼ ✓ Test Results                      64 ms
  ▼ ✓ GameTests                       64 ms
      ✓ successfulPayGoldTest()       61 ms
      ✓ failedPayGoldTest()            1 ms
      ✓ damageTest()                   1 ms
      ✓ zeroPayGoldTest()
      ✓ addGoldTest()
      ✓ addPointsTest()                1 ms
```

All of our unit tests passed. However due to the limitations of unit testing within libGDX, we can't claim that these tests are complete and correct as these tests are simulated without the use of a graphical end (we had to use a different Ship constructor which doesn't contain a texture file). However, we have good reason to assume these methods are correct as these successful unit tests are backed up by play through testing of the game itself.

### White-box Walkthrough Testing

We carried out 16 White-box walkthrough tests on our game to ensure that all requirements had been met and that there were no errors **[2]**. However due to the nature of the assessment, only requirements that are related to features we need to implement for Assessment 2 were considered. 16 out of 16 tests passed and gave the expected outcome, however we found three new bugs in the process in the form of GUI display errors.

Found bugs:

- Upgrade can be purchased at the Chemistry department and correctly applied to player's ship. However, the price display for the upgrade was incorrect.
- Player's ship can be repaired at departments and health correctly restored to full. However, the price display for healing was incorrect.
- Contact between the player's ship and the map's boundaries prints message on the screen, these messages should only be visible when colliding with islands.

These bugs have been documented and our developers have implemented fixes for them by changing the values of the numbers in the code.

We believe the tests are correct and complete as they follow the requirements for this Assessment so all the needed features have been considered, no more no less. In addition, in order to perform these tests we sometimes make changes to variable values within the code (points, gold,...) to test both regular and extreme cases so all cases have also been considered.

## Black-box Walkthrough Testing

We passed on the game to our tester, who was not heavily involved in the production of the code, for multiple exhaustive playtests. Most of the main components functions correctly however there were still four more errors  discovered in our game **[2]**:

- After a purchase has been made in the Department screen, the text does not update with a new price.
- In the College screen, even if the player's ship is already fully repaired and at full health, the repair button can still be click and a message generated.
- In both the Department and College screens, the user can continue repairing the ship and getting charged even if it is fully repaired by clicking the button.
- When switching from Main Menu to Sailing screen, for a brief amount of time clicking inside certain regions on the screen is still recorded as clicking the respective buttons on Main Menu.

These bugs have been documented and our developers have implemented fixes for them, with the exception of the bug with Main Menu as it is very hard to reproduce and the window for generating the error is within seconds so we haven't fully fixed it yet.

We believe the tests are correct and complete as they were found after a series of exhaustive playtests where every possible scenarios had been deliberately considered by our tester. In addition, the fact that the tester wasn't directly involved in the code writing process helped provide unbiased inputs from a subjective point of view.

## References

[1] SEPR “GameTests.java” Rear Admirals [Online] Available https://therandomnessguy.github.io/SEPR/Assessment/2/GameTests.java [Accessed: Jan. 20 2019]

[2] SEPR “Assessment 2 Testing Documentation” Rear Admirals [Online] Available https://therandomnessguy.github.io/SEPR/Assessment/2/TestingDocs.pdf [Accessed: Jan. 20 2019]