Evaluation and Testing Report

Evaluation approach

To evaluate our final product, we first had to make sure it met all the criteria within the initial brief. This was achieved by going through all the original requirements, checking whether they align with and fully cover the assessment brief. No change was needed to this set of requirements as we believed they had properly included everything within the original brief.

For Assessment 4, the client asked for a change in the existing requirements to be implemented. On receiving the announcement, we got together as a team and went through it together, noting down any uncertainties for further clarification with the client. Addressing these ambiguities prior to the development phase consolidated our confidence with the direction we were heading in. After all internal and external discussions had finished, two new functional requirements **F19** and **F20** were added to the original set of requirements to reflect our understanding of what were asked for along with their associated risks.

For this assessment, we also carried out rigorous bug clearing and code refactoring on the inherited project chosen in the Selection phase as soon as possible after we got it. This was so that we could organise team meetings for all of us to have a look at the core functionalities and reached an agreement on their quality. This was done as before new requirements were added, we wanted to make sure that the existing product worked.

The newly revised list of requirements later on played a crucial role in our evaluation process. As the game got closer to being finished, we started checking if requirements had been fulfilled by cross checking them with the game's functionalities. All requirements had been achieved apart from one (**F16**), which we agreed as a group to not implement and was noted down to be referred to in the Requirement Report. This process of cross referencing our list of requirements to the final product ensure that we had fulfilled everything asked for in the brief, as well as satisfied with the final product as the requirement list had been put together as team after thorough discussion and input from every members.

Last but not least, once we finished the game the final executable was exhaustively checked to make sure that it was running as expected. This was achieved through multiple playthrough test runs which allowed us to validate various different aspects of the game.

Testing approach

We got started by carrying out researches into how a software is considered good quality. After careful consideration, we decided to adopt the ISO/IEC 25010:2011 standard **[1]** as it was the newest international standard for the evaluation of software quality issued by the International Organization for Standardization (ISO). The standard has 8 main characteristics: Functional suitability, Performance Efficiency, Compatibility, Usability, Reliability, Security, Maintainability and Portability. However, due to the nature of our project, we have selected four main attributes from this list, which in our opinion were most relevant to our product, to measure software quality:

- Functional suitability: The software implements features that satisfy all stated or implied requirements.
- Usability: The software does not require substantial effort to be used.
- Reliability: The software is expected to perform its intended features in a satisfactory manner.
- Maintainability: Minimal effort is needed to make specified modifications to the software.

Functional suitability was assessed by cross checking the features within the final version of the game against the revised list of project requirements. Both Black-box and White-box testing methods were used to carry this out.

Usability was assessed through Black-box testing with an external member with no knowledge of the game and a team member not heavily involved in code writing. Any difficulties these users encountered while playing the game were noted down and passed onto the development team so that the GUI and game balances could be improved in accordance.

Reliability was assessed through rigorous Grey-box testing. Any bugs found during this phase were noted down and passed to the development team to be fixed.

Maintainability was assessed through extensive review of the codebase to make sure that the code is modular, well formatted and commented. In addition, all new addition to the inherited codebase has been highlighted in the comment for increased visibility.

Testing Methods

An approach similar to that of Assessment 2 was adopted by our team for this phase of development: an incremental approach during the development process where tests are carried out as soon as a feature is finished during sprints, and a release testing phase towards the end to ensure integration. Similarly, our team has decided to take up Grey-box testing as the main testing method following Assessment 2. We also made an effort to incorporate Unit Testing in our game however incompatibilities between LibGDX and JUnit prevented us from doing so. Limewire had solved this problem in their implementation by adopting a framework called Mockito, sadly as some of our members' laptop had problem setting up this framework we decided to remove it to ensure synchronisation between teammates - something crucial within a Scrum team.

White Box Testing

This was carried out mostly during the development process by our developers everytime a new feature is implemented into the game or an old feature is adjusted to make sure it functions correctly, satisfies respective requirements and integrates well with existing components. These tests are most appropriate for the project so far, albeit not as subjective as Black-box testing, as we can undermine the biggest overhead of simulating via playing: time by tweaking the game or add cheats to it. For instance, a hotkey can be used to add gold - removing the need for lots of grinding of battles allowing the testing of features such as game completion and boss battles to be more effectively tested.

We carried out a myriad of White-box walkthrough tests on our game throughout the development process to ensure that all requirements had been met and that there were no errors. A lot of bugs had been found along the way with most fixed long before we reached release testing phase. During the release testing phase most tests we carried out passed and produced the expected outcome, however we still found three new bugs in the process:

- Detection for completion of the minigame is slightly buggy; it doesn't always detect that you have won the game and can sometimes cause the game to crash.
- Dragons can sometimes breach the edge of the map.
- Dragons have a line behind them while flying.

These problems have been noted and our developers have implemented fixes for the first two. For the last bug, we found out that it had to do with the sprite sheet we obtained online which was a very good one and did not require copyright. Taking this into consideration along with the fact that the displayed line was small and blended in pretty well with the surroundings we decided to not fix it.

Black Box Testing

Black-box testing of the final product was carried out at the end of the development process by our tester (a member of our group not heavily involved in writing the code) and one external member (with no knowledge of the game) **[2]**. We felt that Black-box testing was the most effective way to test the game at this point as the game had almost been finished and implementation details were disregarded, allowing for very quick isolation of requirements and fit criterias that were not met or functionality in the game which was not working correctly. It also tests the game's value as a final product as a real playthrough of the game was mimicked from the user's perspective and subjective inputs were provided from a different point of view in order to find errors that otherwise would have been overlooked by someone more involved in the coding process.

We carried out 31 Black-box walkthrough tests on our game throughout multiple exhaustive playtests. Among them, 30 passed and produced the expected outcome, 1 failed. This showed that our games was very robust and produced to a good standard, as well as proving the efficiency of the rigorous White-box tests we had carried out during the development process which had identified and fixed or accepted all problems.

Failed Test:

- **1.17**: This test checks whether there are different weather conditions implemented within the game, adhering to requirement **F16**. The reason the test failed was due to our team deciding not to implement this feature in our final product as we already have another feature acting as a randomly generated event.

Requirement Evaluation

With the conclusion of the project's production cycle in Assessment 4, it was essential that we completed all feasible requirements given to us, with an additional emphasis on the updated requirements given to us at the beginning of this quarter - alongside ensuring that the brief was met. Below is a list of all requirements **[3]** along with quick descriptions of how we met them or why we feel they were unnecessary.

Functional Requirements

Requirement **F1** was carried out by using the islands on the map to represent different colleges and departments of the University. **F2**, **F3** and **F4** were all implemented throughout the process, while **F5** was fulfilled by have multiple layers of difficulty; sea monsters, ships, and bosses, with the bosses being difficult to beat without having previously battled and upgraded. Requirement **F6** was completed in production, as was **F7**, which was developed by varying the amount of points received depending on whether it was a ship or boss that had been defeated. Similarly, **F8** was fulfilled, and **F9** was carried out by having colleges be captured once their respective boss had been defeated. The team implemented **F10** by having departments used purely for buying upgrades and health. This was done with requirement **F11** in mind, as these are bought with the gold acquired from enemies. We fulfilled **F12** by creating an optional maze minigame that rewards the player with points if completed successfully. Requirement **F13** was completed when the team worked on Assessment 3, whilst **F14** has been fulfilled as the player ship is displayed in the sailing screen whilst enemy ships are only shown on the combat screen. We have implemented the ability for players to save their game and come back to that save in a different session, satisfying requirement **F15**. As a group, we decided to miss out **F16**, as we felt the game was complex enough with sea monsters, ships of varying difficulty and a minigame, and a weather system would only confuse this balance. Multiple points of damage (hull and sails) were implemented into the combat system, meaning different outcomes depending on the area of attack, and the completion of **F17**. Further, requirement **F18** was completed as once colleges are defeated, they can be used in a similar way to departments or the player's home college. With Assessment 4 came two new requirements, **F19** and **F20**, which we fulfilled by implementing the ability to obtain crew members with differing perquisites for the player and the possibility of challenging sea monsters whilst sailing.

Non-Functional Requirements

We believe that we have sufficiently fulfilled requirement **NF1** by providing relatively intuitive controls and gradual increase in difficulty as the player discovers more of the game. **NF2** has also been tested, whilst the game was created with **NF3** constantly in mind. Further, **NF4** was strictly adhered to in the conception of the project.

Constraint Requirements

Requirement **C1** has been completed by submitting Assessment 4 before our given deadline, whilst throughout we have adhered to **C2**. Further, we have attempted to keep as close to **C3** as possible by constantly testing and adjusting the game where suitable. Finally, we believe we have also fulfilled **C4** by relating the game to the University of York.

References

[1] ISO/IEC 25010:2011. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, 2011. [Online]. Available: https://www.iso.org/standard/35733.html
[Accessed 30 - April - 2019]

[2] Rear Admirals Assessment 4 Black Box Testing, 2019. [Online]. Available: https://therandomnessguy.github.io/SEPR/Assessment/4/Black_Box.pdf
[Accessed 30 - April - 2019]

[3] Rear Admirals Assessment 4 Requirements, 2019. [Online]. Available: https://therandomnessguy.github.io/SEPR/Assessment/4/Updates/Req4.pdf
[Accessed 30 - April - 2019]