Evaluation and Testing Report

Evaluation Approach

We firstly had to ensure the current version of the product met the criteria in the initial brief[1]. To do this we reviewed the original set of requirements for the game to make sure they were precise enough for the final development phase. We needed to ensure that at the very least all functional and non-functional requirements did not contradict the original assessment brief and that everything that was asked of us was covered within our requirements. Along with this, checking all other types of requirements were met was also essential as we wanted a polished marketable game at the very end of this development phase. Nothing was changed or added to the original set of requirements as with respect to the original brief as we thought they encapsulated everything in the brief adequately.

During Assessment 4 we received a new set of additional requirements [2] we had to implement into the game. After receiving the requirements changes we updated the requirements table to accommodate them. Four new requirements were added to the table to make sure we fully understood what needed to be implemented within the game along with the associated risks and rationale to go with them. This meant we could prepare for things that could go wrong with implementing the new features ahead of time. As a team we also read through the new requirements to ensure they encapsulated the new briefs changes to the game and brought any relevant questions to the end user about uncertainties of their needs. By addressing any ambiguities we had with the new brief this allowed us to be confident with the direction we were heading in for this development phase.

When we felt the game was close to being a finished product we did a requirements check and cross referenced the requirements with the functionality of the game. Any requirements that had not been fulfilled within the final build of the game were highlighted orange; as shown in our updated table [3]. This was the case with only one requirement and it did not affect the games functionality with what had been asked of us from the brief. The decision to drop this requirement was discussed within the group before we proceeded. This method of cross referencing our requirements to the game allowed us to ensure that our product had met the brief and that as a team we were satisfied with the end product. Finally, Once the game was finished we had to validate that the final executable was running as expected. This was ensured through numerous play tests which allowed us to check that the functionality and performance of the game was up to scratch.

Testing Approach

In order to carry out the testing of the final product we had to establish what we considered high quality software to be. Although a single definition of software quality doesn't exist[4], it seemed clear from our research[5] that quality is based on several factors such as: maintainability, portability, functionality, performance, compatibility, usability, reliability and security. However, depending on the nature and aims of each software, some of these factors will be more important than others.
Therefore we have identified four main criteria, which we believed were most relevant to our project to assess the quality of the product:

- Reliability - The presence of any bugs and whether the game is fully playable, from start to finish, without encountering any issues.
- Functionality - The software implements what it is intended to do and meets the requirements.
- Usability - The game is easy to use without needing instructions.
- Maintainability - The code is modular, readable and uniform in style so that the software can easily be maintained and expanded by a future team.

To guarantee that the game was reliable and bug free we designed extensive Black-box tests that would ensure that every function of the game worked under a variety of conditions. Along with the black box testing we played through the game to see how the software would perform under uncontrolled conditions, any bugs that were found during this phase were recorded on Jira for the developers to fix.

Functionality was assessed by cross referencing the project requirements with the features of the final build of the game, the methods used are described in detail in the next section of this report.

To assess usability we asked people that did not take part in the development to navigate through the menus and play the game, checking if they ever struggled or needed additional instructions to proceed. Whenever a user had an issue it was noted and added on Jira so that GUI designer could improve the interaction and make the user experience as intuitive as possible.

To assess maintainability we reviewed and overhauled the entire code base to ensure that the functionalities of the software were separated into independent modules. During the review we also noticed that, due to contributions from various teams, different styles of writing code were used in the project. This meant that there was not much consistency between classes with some using PascalCase and others using CamelCase. Variables names were also inconsistent, often not being very descriptive. This led to vigorous refactoring of the code to make sure that it would take minimal effort to read, understand and modify.

Testing Method

We carried out our tests using the Black-box method, so we set the game up with specific conditions to test if the game would behave as expected. Black-box testing was an appropriate method because it allowed us to test the software in the same environment that

it will be running in when the game is released. It also shows that all the modules of the program work together successfully rather than just testing if they work individually and gives the ability to test the error handling of the client.
However, doing only Black-box testing can lead to some conditions where the game isn't tested so we also had play testing with people who weren't associated with the product as we decided this would be a good way to discover more obscure bugs.
We considered designing unit tests, but after contacting CR Games to discuss their testing approach we were informed that they had not been able to implement automated unit testing as they encountered compatibility issues with the LibGDX library.
We therefore concluded that our Black-box and play testing would be sufficient to guarantee no bugs would be found.

Initially we re-ran all the tests from Assessment 2 as well as adding in new tests to check that all of CR Games' implementations worked well before adding more to the game. Through play testing we found that the tests we had already written were not sufficient to test the game in depth and therefore we added a new section named "Additional Game Testing" which we used to test features later in the game and report more bugs. Once all the tests we had written passed we continued with development continuously adding in more tests as new features were added.

When a bug was found it would be added to Jira with an explanation of the cause, circumstances and if appropriate, depending on the cause and result of the bug, a stack trace. This would allow the developers to solve the issue as quickly as possible. Using Jira allowed the developers to keep track of what bugs were found and who was working on fixing each one to ensure that two people didn't try to fix the same bug which could cause conflicts and waste time.

The results of our testing are presented as a spreadsheet where each test has: an ID for reference; the circumstances of the test; instructions for carrying out the test; an expected result and a value to indicate whether the result had passed or not.
The results of each test were highlighted in various colours to indicate their status from "Not Yet Implemented" in yellow, "FAIL" in red and "PASS" in green, this helped quickly identify which tests needed repeating [6]. The tests highlighted in pink are the ones from Assessment 2 that we have re-run. All of the screenshots for the tests were kept in a separate document to keep the formatting neat and readable [7].

After multiple iterations of testing and bug fixing, 100% of our 77 tests passed. This shows that the code is reliable and potentially bug free. However, there is the possibility there could be an error where the testing was not thorough enough, but due to our many playthroughs and tests we are confident that no unfound bugs should have a significant impact on the game and its players.

Requirements Met By The Product

It is important within Assessment 4 that we meet all the updated requirements[3] as this will be the final finished product of our game. Below is a list of all the requirements and a small description as to how we achieved our goals in meeting them.

Constraint Requirements

Requirements C1 and C2 have been met and tested by getting users to play the game outside of our group and by running the game on various different machines including the university computers. The game meets C3 as it takes both mouse and keyboard input.

Functional Requirements

Requirement F1 has been met as average game time is within these constraints. Requirements F2, F3 and F4 were all met and implemented using our options/pause menu systems for toggling the turn timer on and off within the game. F5 and F6 requirements were met as the System for attacking is calculated based on the number of troops selected to attack also taking into account the difference in strength between the two parties. The Requirements F7, F8 and F9 have been met and implemented via a short card matching based minigame, which is triggered randomly when a player conquers a sector and allows players to obtain a bonus if they complete it. Requirements F10 and F11 have been met and the neutral player option can be accessed within the setup game menu as required by F15, functionally the neutral player works as described within these requirements. The requirement F12 for saving and loading is fully functional and can be accessed within the main/pause menu GUI. For requirement F13 our map is based on the university campus and colleges, with a few minor changes to distance between landmarks for balanced gameplay. For requirement F14 and F16 we included earning bonus troops based on the number of sectors conquered during a players turn, these troops can be allocated at the players choosing during their next allocation phase. Requirement F17 is fully functional and distributes sectors randomly in a balanced manor to each player before the start of a game. Dialog boxes and sliders have been used to implement requirement F18, easily allowing the player to allocate a certain number of troops during the allocation phase. Requirement F19 has functionally been implemented by visually showing the current number of undergraduate and postgraduate troops on a sector with two different numbered icons. The punishment cards in requirement F20 can be acquired through a bonus achieved within the minigame. There are 3 different types of punishment cards as required by F21.

Non-Functional Requirements

Non functional requirements NF1, NF2 and NF3 have all been met and were considered throughout development of the game through play testing from potential end users of our game. The soundtrack as required by NF4 is stylized towards the games artistic style and adds immersion to the game. The non-functional requirement NF5 was not met as including accessibility features within the game was too time consuming as other essential features needed to be finished and tweaked before the deadline. Requirement NF6 has been met to the best of our abilities.

Performance Requirements

The requirement P1 has been considered through the development of the game and tested on numerous machines to make sure the game runs smoothly and is playable.

References

[1] SEPR “Initial Scenario” Risky Developments [Online]. Available: http://www.riskydevelopments.co.uk/documents/SEPRScenario1-UniversityDomination.pdf” [Accessed: Apr. 29, 2018].

[2] SEPR “Assessment 4 Requirements changes” Risky Developments [Online]. Available: http://www.riskydevelopments.co.uk/documents/Ass4RequirementsChanges.pdf [Accessed: Apr. 29, 2018].

[3] SEPR “Updated Requirements” Risky Developments [Online]. Available: http://www.riskydevelopments.co.uk/documents/UpdatedRequirements.pdf [Accessed: Apr. 29, 2018].

[4] Search Software Quality “An expert suggests how to measure software quality” [Online]. Available: https://searchsoftwarequality.techtarget.com/opinion/An-expert-suggests-how-to-measure-software-quality [Accessed: Apr. 29, 2018].

[5] Testing Excellence “How do we Measure Software Quality in Agile Projects?” [Online]. Available: https://www.testingexcellence.com/how-do-we-measure-software-quality-in-agile-projects/ [Accessed: Apr. 29, 2018].

[6] SEPR “Black Box Tests” Risky Developments [Online]. Available: http://www.riskydevelopments.co.uk/documents/BlackBoxTests.xlsx [Accessed: May. 1, 2018].

[7] SEPR “Black-box Testing Screenshots” Risky Developments [Online]. Available: http://www.riskydevelopments.co.uk/documents/BlackBoxTestScreenshots.pdf [Accessed: May. 1, 2018].