**University of York**

**Department of Computer Science**

**SEPR - Assessment 4**

# *Evaluation and Testing*

**Team Craig**

Thomas Burroughs

Huw Christianson

Joseph Frankish

Isaac Lowe

Beatrix Vincze

Suleman Zaki

# Evaluation and Testing Report

## Evaluation

In order to determine that our final product meets the brief set out by the initial specification, as well as the additional requirements specified in Assessment 4, we have defined a set of vigorous, point-demarcated requirements. All requirements can be seen in our updated requirements document [1]. The additional requirements for Assessment 4 were elicited using the election process as described in our updated requirements document [1]. The team discussed each new requirement in detail to ensure that each clearly represented what was specified in the product brief. Any ambiguities were discussed in the group to remove any vagueness from the requirements. Furthermore, a suitable fit criterion was included to allow the requirement to be measured, which in turn allowed the team to test if the requirement was met. The team broke down all new additional requirements into smaller, more manageable requirements, each with clear definable metrics of completion. The splitting of requirements helped the team to implement our agile methodology, an adaption of the scrum method. As Scrum is an iterative approach, tests had to be completed after each weekly sprint. Having smaller, simpler requirements meant that tests could take place more regularly and errors flagged at after each sprint. This has enabled the team to ensure that each distinct part of the implementation has been tested, which as whole meets the additional requirements.

To ensure that our final product doesn't just meet the requirements defined in Assessment 4, but also meets the requirements defined in previous assessments, and hence the full product brief, the team performed regression testing. This was vitally important as changes to our code in Assessment 4 may have affected the functionality of the game, and thus prevented some requirements being met. However, before this, the team had to ensure that all previous requirements were free of ambiguity and accurately depicted the final product brief. Once suitable checks of the previous requirements were made, the team used inherited tests to test that each of these requirements had been satisfied.

Another key aspect of our testing approach was our inclusion of requirement testing and subsequent traceability matrix. Our requirements testing table [2] helps the team to easily identify which requirements have been tested by which tests and whether these tests suitably signify whether the requirement has been satisfied. This was determined by our 'fit criterion'. Additional comments in our requirements testing table clear up any ambiguities that could be derived from our testing, ensuring that our product meets the brief. For example, if a test only partially tests a requirement this was noted to ensure that further tests were made to fully cover the partially satisfied requirement. The traceability matrix [3] was used to simplify the traceability of our requirements. After each sprint, team members could make changes to the matrix by using a colour coded system to note any new tests and any newly satisfied requirements. Towards the end of the implementation phase, the team could see which requirements had not been coloured and hence required additional tests. The cross-referencing between tests and requirements ensures that each of our requirements have been tested and, since our requirements accurately capture the product brief, we are confident that our game meets the required specification.

### Testing Approach

Before any testing of our product took place, the team undertook research in order to determine how to measure the quality of our code. By reading through Somerville's Software Engineering [4] we came across the ISO standard [5] which describes a software quality model by categorising software quality into six metrics [6]. As a team, we discussed each of these metrics and decided which of them would prove useful for testing the quality of our code in our project. Most of our decisions to omit some metrics were purely based on the scale of our project compared to the industry norm. We did not think it was feasible to test our code against all of these metrics, e.g efficiency and re-usability, due to time constraints and limited resources.

The team specifically chose the following 4 metrics:

- **Functionality**: Used to test how well our software product provided the desired functionality and assessing the extent to which product satisfies our functional requirements [6].
- **Reliability:** Used to test how well our software product maintains a level of system performance under various conditions [6].

- **Usability:** Used to test how well our software product can be understood, learned, used and liked by both developers and users [6].
- **Maintainability:** Used to test how well our software product can be maintained and assessing the effort needed to modify our software product with changing requirements [6].

Our testing approach through all of the project has remained fairly consistent, with the testing team using black-box tests, white-box tests (in the form of Unit tests) and requirements testing. As part of our agile methodology each test needed to be completed after each sprint in order to test the functionality and reliability of our code. This took the form of regression testing. Although no new Unit tests were written for Assessment 4, previous Units tests, as seen in our Unit tests evidence document [7] proved vital for reliability testing. By running previously passed tests after each sprint, the team could ensure that each piece of functionality was functioning as expected even after changes to the code base had been made. This adds to the reliability of our code as we can ensure that each piece of functionality runs as expected even under different conditions, such as a modified code base. If any tests failed when performing regression testing the team refactored the code during the next sprint, and the previous tests were run again.

To test our functional requirements and also to test our code against our functionality metric, we implemented both black-box tests and Unit tests using IntelliJ's inbuilt testing feature. For Assessment 4, the team focused on using black-box tests rather than Unit tests due to the short time frame of the assessment and the ease in their creation and execution. More specifically, we believe that the additional requirements could easily be tested using black-box tests and Unit tests proved difficult due to the fact that most functions only worked in the context of events fired by LibGDX which couldn't be triggered on demand , so made Unit testing difficult. Test cases for each black-box tests were devised before any implementation began along with an expected outcome. When possible, these test cases were based on the fit criterion from our requirements document [1]. This allowed us to be confident that the passing of our tests indicated that our functional requirements had been satisfied. This ties in with our requirement testing as some of our tests are derived from our requirements and hence directly test whether the requirements were met, as seen in our requirement testing document [2]. This was an important factor when assessing the quality of our code as our team felt that in order for our code to be deemed of high quality it must be functional, reliable, usable and maintainable but also satisfy the product brief.

The manner in which our black box tests were executed helps us to asses the reliability and usability of our code and product. Three team members were responsible for performing black-box tests in each sprint, all of whom had no previous knowledge of our code base. This helps to remove any bias when playing the game and replicates a user experience as close as possible within our team. Each tester performed all the black-box tests and noted whether any tests failed in our black-box test evidence document [8]. Any problems with usability were noted in our additional comments column and discussed at our next weekly meeting. By having all three testers perform the same tests on their own systems we could ensure that each piece of functionality was functional and reliable ( as worked correctly for all three testers) for numerous users. A test was not stated as 'passed' unless the test passed for all three of our testers, and therefore a passed test indicates a working and reliable functionality.

In order to assess the maintainability and usability of our code, we focused on how well tested our code is and whether it is easily understandable and modifiable [9]. To ensure that our code is well tested we tried to maximise our testing coverage. We did this by increasing the number of black-box tests even if they do not explicitly test a requirement, but the correct functioning of our code. Furthermore, the understandability of code was improved by sticking to expected coding standards set by the group, including naming conventions, and ensuring all code was well commented. This has also helped us to measure the usability of our code, as the more the team commented and abided by the coding conventions, the greater the confidence that the code was easily usable by new teams who may take on the project. However, this was not of the utmost importance as we knew that our game would not be inherited after Assessment 4, but considerations were still made to improve usability and maintainability.

Since the project required a change in requirements, when researching which groups game to take on for Assessment 4, well-modulated and commented code was a high priority. It was very important for our team to maintain a cohesive modular code architecture to allow for changes in requirements, despite being the last assessment. Removing any duplicated code helped the team to improve the readability of our code. Any code which needed to be used several times was structured appropriately to allow for easy inheritance as can be seen by our updated UML diagram [10]. The maintainability of our code was also improved by increasing traceability through our traceability matrix. If another team took on our product, they would easily be able to identify which tests test which requirements and should any requirements change the associated test could easily be identified and altered . Due to all the above, we believe that the usability and maintainability of our code meets the expected standard laid out by the ISO [5].

If the team had a longer time to work on our software product we would have liked to have improved testing for non-functional and performance requirements, such as requirements N1 and P1 [1]. This would improve the assessment of the reliability and usability of the code. For both of the requirements above, we would have preferred to have tested them with a larger cohort of users, unfamiliar with our game, to ensure that all errors could be flagged and any issues with usability identified. With more time, further Unit tests could have been created to maximise our testing coverage and test all possible execution paths. Despite this, the team believes that our code meets the appropriate standard for a project of this nature and meets all the requirements laid out in the brief.

## Testing Results

There are separate documents showing evidence of black-box tests [8] and white-box tests [7] each noting the Tests ID, a description of the test, whether the test passed or failed and additional comments to note any ambiguities in the tests.  For Assessment 4, an additional column was added to each evidence table to state any associated requirements.

### Black-Box Statistics and Results

3 testers conducted a total of 39 black-box tests of which 1 failed on the first run. The team agreed that a failed test is one in which the outcome did not match the expected outcome defined before the test was conducted. The remaining  38 tests passed on the first time of asking and hence no further changes of code needed to be made by the team.

Test 9.14 failed due to the over-specification of the test case. To explain further, the original test case explained that the 'cure' item must turn zombies into non-zombies within an exact radius. However, since this could not be quantitively verified using black-box tests the test had to be generalised.  We chose not to refactor the code as we believe the functionality does satisfy requirement F12, as can be seen in our requirements testing document [2]. The test case was modified to more accurately test the functionality and verify that the requirement had been met without any unnecessary test case details that may have given a false-negative result.

On the final running of all 39 black-tests, all 39 passed, giving a 100% pass rate.

### Unit Tests Statistics and Results

No additional Unit tests were written for Assessment 4 as additional functionality was tested using black-box tests alone. However, all 36 previous Units tests were used for regression testing. Although some Unit tests did fail between sprints when additional functionality interfered with previous code, at the end of the implementation phase all 36 Unit tests passed, giving a 100% pass rate.

Our 100% pass rate for both our black-box tests and unit tests gives us great confidence that the core functionality of our game operates as desired and tests cover a large proportion of all code.

## Meeting the Requirements

At the end of Assessment 4, we believe that our product meets all the requirements as specified in our requirements table [1]. As a team, we have determined this through the successful execution of our requirements testing [2] and supporting traceability matrix[3].

The team was satisfied that the game we inherited from Team Yeezy Games successfully satisfied all of the requirements up to Assessment 4. This was determined through regression testing at the start and after each sprint in Assessment 4. This assured the team that any modifications of the code did not break any inherited functionality and subsequently assured the team all previous requirements had been met.

As part of Assessment 4, the team added three additional requirements (F11, F12, F13) due to a change in the product brief. As a team, we are satisfied that all of the functionality that needed to be implemented to meet the requirements has been implemented successfully and is free of errors.

All of our requirements have been tested directly using either black-box tests or unit tests apart from REQ N6. Requirement N6 stated that the game must have an 8 bit aesthetic, however, this was not tested through either black-box tests or unit tests as we believe that the images provided as additional evidence for other black-box tests, and images as part of our User Manual, clearly indicates that levels, weapons, and characters in the game all have 8-bit aesthetics.

Although the team has concluded that all requirements have been met, we all agree that more rigorous testing should be undertaken to test non-functional and performance requirements. A lack of time and resources has meant that the extent to which we would have liked to have tested these requirements could not be conducted  (would have liked to use a larger cohort of use cases). Despite this, we do believe that our performance and non-functional requirements have been tested to the expected standard for this project and are confident these requirements have been met.

## References

[1]- Team Craig. *Updated Requirements- Assessment 4*. Available: https://teamcraigzombie.github.io/assets/downloads/UpdatedRequirements4.pdf

[2]- Team Craig. *Updated Requirements Testing - Assessment 4*. Available: https://teamcraigzombie.github.io/assets/downloads/UpdatedRequirmentsTesting4.pdf

[3]- Team Craig. *Traceability Matrix- Assessment 4*. Available: https://teamcraigzombie.github.io/assets/downloads/TraceabilityMatrix4.pdf

[4]- I. Sommerville. Software Engineering. Pearson, tenth ed, 2016 pp. 660-661

[5]- ISO. *ISO 9126 Software engineering -- Product quality*. June 2006.

[6]- ARiSA AB. *Software Quality ISO Standards.* Available: http://www.arisa.se/compendium/node6.html

[7]- Team Craig. *Updated White-Box Testing Evidence- Assessment 4*. Available: https://teamcraigzombie.github.io/assets/downloads/WhiteBoxTesting4.pdf

[8]- Team Craig. *Updated Black-Box Testing Evidence- Assessment 4.* Available: https://teamcraigzombie.github.io/assets/downloads/BlackBoxTests4.pdf

[9]-N. Kukreja. *Measuring Software Maintainability.* Feb 2015. Available: https://quandarypeak.com/2015/02/measuring-software-maintainability/

[10]- Team Craig. *UML Diagram- Assessment 4.* Available: https://teamcraigzombie.github.io/assets/downloads/UpdatedUML4.pdf