# Evaluation and Testing Report

SEPRet Studios
Anthony Nel, Ben Silverman, Jake Billington, Leah Moss and Shouyi Yuan

# Evaluation

**Our chosen evaluation method.**

As a team we decided we would follow a four-step method to verify the requirements met by our final product: Inspection, Demonstration, Testing and Analysis [1]. We thought this was an appropriate method as it supported verifying requirements in different ways; some requirements may be more easily verified through testing than inspection, for example.

*Inspection*

This step involves non-destructively examining the final game to determine certain characteristics about it. For instance, we could inspect our game visually to confirm that there are at least four fire trucks present at the start of the game (thereby fulfilling the UR_FIRE_TRUCKS requirement) or aurally listen for the game's sound effects (fulfilling FR_SOUND). The inspection step is good for quickly identifying "small" requirements that don't require a complicated scenario to be set up or can't have a unit test written (they might not be best visualised in code).

*Demonstration*

This step is tied quite closely to our manual test plan, as it involves operating the game as it would be played by a real user, to verify that we achieve the results we expect. An example of this in our game would be executing the MAN_DESTROY_TRUCK test to demonstrate fire trucks being destroyed (and show the game meeting FR_FIRE_TRUCKS).

*Test*

We performed the test step using JUnit tests in IntelliJ. Any requirements that could be tested with a set of predetermined inputs were examined here, as we could quickly and repeatedly run the tests when we made changes to the code, ensuring the requirements were still met.

*Analysis*

We interpreted this as quite similar to the testing step, and didn't find any requirements we thought would best be verified via this step. We agreed that more "overarching" requirements testing the stability of our game would be examined here.

**Evaluation results.**

*User Requirements:*

Of the 19 total User Requirements, the only requirement we did not meet was UR_SCALABILITY. This was a lower ("May") priority requirement expressing that "the game should be able to be played on other platforms". We also noted that UR_ENJOYABILITY was hard to quantify as having been met without meeting with a stakeholder to hear their opinions.

*Functional Requirements:*

We were able to successfully meet all 19 of the Functional Requirements outlined in our final requirements documents [4]. When implementing our solution for the FR_POWER_UPS requirement we chose a power-up enabling fire trucks to instantaneously be fully repaired and refilled, and noticed a possible conflict between this and FR_STATION_DESTROY. However, since FR_POWER_UPS was listed as a higher priority ("Shall") than FR_STATION_DESTROY ("Should") we agreed that this was acceptable.

*Non-Functional Requirements:*
We achieved the fit criteria for all three of our Non-Functional Requirements, and as such agreed as a team that these requirements were met.

**Improving our evaluation results.**
First and foremost, we could improve our evaluation results by ensuring *every* requirement we listed is met. This would increase the likelihood of our client being satisfied with the end result, and we feel it's a strong indicator we have developed the product they asked for.
Additionally a meeting with our client near the end of development to demonstrate the product would be extremely beneficial; it provides them with an opportunity to express which areas of the product they're happy with, and which can still be improved upon. We believe this is a good approach as the client is best-placed to determine how well we've met their brief, and it could save us time during evaluation by allowing them to quickly flag any remaining problems.

# Testing
We decided, ideally, that "appropriate code quality" should be state where a product passes all of the requirements set out in its brief, and has accompanying tests as proof of code working as intended. Alongside this the tests themselves would have a high method coverage. This provides a high degree of confidence when developing new features as to whether it'll end up breaking something else in the code (and allows developers to more quickly spot and fix the issue). If the above isn't possible within a given timeframe, we agreed that a product meeting all of the functional requirements could also qualify as "appropriate code". The sections below outline how we approached testing throughout Assessment 4, and how that approach ensured appropriate code quality.

**Changes in testing approach from Assessment 3.**
The core approach to our testing hasn't changed much from Assessment 3. Like before, we split our testing into two sections: manual testing and unit tests.
We continued to manually test our product in Assessment 4 because often there are areas that can be more quickly and easily tested with a manual approach [2], like ensuring fire trucks or fortresses are removed from the game when they reach zero health. We thought our method to track the manual tests we performed was well-suited to displaying both the test results and the process to reproduce said results in a clear and verbose way. If we'd had a much larger number of manual tests, we would likely change the testing layout to either a spreadsheet or perhaps some specialised software like Azure DevOps [3]. Overall we increased the number of manual tests from 22 to 31, allowing us to cover the new functional requirements introduced in Assessment 4 [4] as well as to test some of our existing requirements in more detail (like the MAN_MINI requirements).
We performed unit tests with JUnit in Assessment 4 to ensure high quality code and help us find bugs throughout the development process [5]. By writing tests as we developed new features, any new problems would be flagged early on giving us more time to fix them and also reducing the chances that an error would go unnoticed and cause problems in the future. Unit tests also help us to calculate a code coverage during the project, which can act as an indicator of how stable our product is.

**Building on the previous tests and accommodating Assessment 4 requirements.**

The existing tests in the previous team's codebase were adequate, but we needed to rewrite some and produce more in order to meet the new and changing requirements from Assessment 4, as well as writing new tests for requirements that they missed in Assessment 3.

*FireTruckTypeTest.java*

All code testing that the fire truck types all had different values for HP, speed, AP and range were modified to include the new fire truck types from the previous assessment.

*FortressTypeTest.java*

All code testing that fortress types had unique range, damage, fire rate and health were rewritten to include the extra fortress types from Assessment 3. The unit tests performing boundary tests on each fortress's range were also added to to include the extra fortress types from the previous assessment, with unit tests being written to validate the base damage values for all of the new fortresses. We also added 6 new tests covering the different damage amounts each fortress would inflict due to the new FR_DIFFICULTY requirement.

*FireStationTest.java*

Early on in Assessment 4 we made some modifications to the FireTruck class, changing the parameters that were required to instantiate instances of it. One of these changes was to include a copy of the maps collision layer as an attribute in the class. We modified all the requirements in *FireStationTest* to include this collisionLayer, implementing Mockito's mocking framework to keep the tests lightweight and not requiring an actual copy of the collision layer.

*PowerUpTest.java*

This was a new test class we added to verify the newly set FR_POWER_UPS requirement. We wrote 7 tests, checking that each powerup correctly applied its intended effect to the fire truck when picked up, and correctly removed its effect when no longer active.

**Changes to our test coverage from Assessment 3.**

Our code coverage in Assessment 4 was similar to the coverage we achieved in Assessment 3. We improved the total line percentage for the entities in our game from 45% to 63%, and brought the *Class %* up slightly from 83% to 85%. These are both steps in the right direction, but an ideal coverage would see a much higher *Method %* and *Line %* for the *Entities* package; with the same being said for the *Minigame* package.

**Changes to our traceability matrix from Assessment 3.**

We kept the same spreadsheet format for the Assessment 4 traceability matrix as we did with the Assessment 3 version, as it provides a very clear structure to track every test that was performed and the requirement/s it fulfilled. The biggest difference is an increase in the total number of tests being tracked; we added 31 new tests to the project since the start of Assessment 4, and feel we have more systematically tested each class (either manually or via unit tests) to ensure it functions as expected.

**Links to testing material:**

- Manual test plan:

- Unit test results:
- Test coverage:
- Traceability matrix:

# Met and unmet requirements

Our team believes we have successfully developed for and met the large majority of the requirements set out in Assessment 4. The tracking of requirements was done in a traceability matrix [6], which linked requirements to the tests verifying them, providing proof that they had been appropriately tested.

However, while it is every team's intention to adequately meet every requirement in a given product brief, unfortunately we encountered one or two User and Functional requirements which we were unable to confidently mark as having been satisfied. This is due to a few different factors: time constraints, the focus on higher priority requirements, and a lack of information during the evaluation phase.

**Requirements not fully met**

UR_SCALABILITY

This requirement stated that "the game should be able to be played on other platforms". Currently, our product has not been designed to work on mobile devices or tablets, so we could not declare this requirement as fulfilled. However, the nature of our project being developed with LibGDX means this is certainly a possibility in the future. A lot of LibGDX native functions for keyboard or mouse input translate directly across to working on touchscreens, for instance.

UR_ENJOYABILITY

This requirement is mainly on the list as being unmet because we haven't had the opportunity to verify with a stakeholder that the game is, indeed, enjoyable. The team have all had fun playing it during development, and we're very happy with the final product but believed it was still worth mentioning here.

FR_STATION_DESTROY

This requirement stated that "fire trucks cannot be repaired or refilled after the fire station has been destroyed". We have listed this requirement here as not fully met since, while we have developed it such that fire engines cannot be repaired *at the fire station* after it has been destroyed, they can still be repaired and refuelled by power-ups, which may conflict with the original intention of this requirement. This is a lack of clarification on our part, and something we should have contacted the client about.

**Link to final requirements:**

- Requirements:

**References**

[1] C. Adams, "What are the four fundamental methods of requirement verification?", *Modernanalyst.com*. [Online]. Available: https://www.modernanalyst.com/Careers/InterviewQuestions/tabid/128/ID/1168/What-are-the-four-fundamental-methods-of-requirement-verification.aspx [Accessed: 28- Apr- 2020].

[2] T. Wyher, "5 Reasons Why Manual Testing Is Still Very Important", *dzone.com*, 2016. [Online]. Available: https://dzone.com/articles/5-reasons-why-manual-testing-is-still-very-importa [Accessed: 28- Apr- 2020].

[3] "Run manual tests - Azure Test Plans", *Docs.microsoft.com*, 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/devops/test/run-manual-tests?view=azure-devops [Accessed: 28- Apr- 2020].

[4] L. Moss, B. Silverman, J. Billington, A. Nel and S. Yuan, "SEPRet Studios Assessment 4 Requirements", *Drive.google.com*, 2020. [Online]. Available: https://drive.google.com/open?id=1zCsmrD4WRoi2gvQsOvYVjYDegCXengTGqPZeuTeN7T4 [Accessed: 28- Apr- 2020].

[5] E. Novoseltseva, "8 Benefits of Unit Testing", *dzone.com*, 2019. [Online]. Available: https://dzone.com/articles/top-8-benefits-of-unit-testing [Accessed: 28- Apr- 2020].

[6] L. Moss, B. Silverman, J. Billington, A. Nel and S. Yuan, "SEPRet Studios Assessment 4 Traceability Matrix", *Drive.google.com*, 2020. [Online]. Available: https://drive.google.com/open?id=16lRXJv-l6QRJaN1-6pKXGyINB8ac9NiHnKJlsm-t5zA [Accessed: 28- Apr- 2020].