| | |
|---|---|
| **Module** | SEPR |
| **Year** | 2019/20 |
| **Assessment** | 4 |
| **Team** | Krojan Horse Expanding on NP Studios' Assessment 3 |
| **Members** | Abel Kent, Toprak Onat Sen, Vinu Jey, Tom Emmerson, Diana Udrescu, Toby Popov, Jamie Todd |
| **Deliverable** | Evaluation and Testing Report |

# Evaluation and Testing Report

## Evaluation method

For the overall evaluation of our project, we checked how our final work fulfilled the requirements set out by NP Studios (which included updates on the previous stage of the assessment by DicyCat). We also updated this document to include user requirements for each of the new features needed to be implemented in this stage of the assessment.

Using these requirements we decided upon the following metric for evaluating our work:

Looking at our User and System (functional and non-functional) requirements, we calculated a percentage of how many our implementation met. We gave the highest weighting of importance to User requirements then System as those are the ones that would most greatly impact a player's experience. We will evaluate our code based on the percentage each of them is met.

We also broke down the coverage of the User requirements into priority listing - SHALL, SHOULD and MAY - going from most to least important respectively. This allows us to evaluate how successful we were at implementing our most important tasks within our overall list.

We decided that this was the best way to evaluate how well our implementation met our requirements due to it being the easiest way for us to compare and contrast which areas may be lacking.

## Testing

The most reliable metric that we have available to judge the quality of our code is that of code coverage percentage. When we adopted this stage of the assessment from the previous group we noted that they only had 1% code coverage for their implementation.

Due to time constraints, and the large amount of coupling between components of the code, we decided against working through their code to implement new unit tests that would cover all of it, due to the fact that it was a largely functional implementation of the brief and would require a drastic redesign of the entire code to make pure functions which are themselves testable. Doing so could create more errors than fixing them, so instead we decided to go by a method of black box testing, noting down any errors that we encountered in their work to judge the existing code quality.

When we encountered a new error we noted it and had a discussion as a team to judge the severity of the error and its impact on the user.

*Errors we encountered:*

| Error | Severity |
|---|---|
| Black lines appearing between map tiles during play | Moderate, while it does not impact the player's ability to play the game it does significantly lower the visual quality of the experience. |
| Some of the trees on the game map do not have collision and can therefore be passed through | Minimal, these odd trees are few and far inbetween so the odds of encountering one in play is quite low, as well as the fact that due to the fact that most of the trees do in fact have collision, we decided that players would not go out of their way to drive into the trees. In addition to this, it has no significant effect on gameplay. |
| Cursor can sometimes be hidden on the menu - occurs infrequently | Minimal, the menu is fairly simple and the user is unlikely to get lost whilst navigating it, so in the unlikely event of this occurring the impact on the user should be minimal. |
| The Load/Save function does not carry fortresses' texture information. | Low, as it does not affect the gameplay in a meaningful way, and a player is fairly unlikely to encounter it during routine play. |
| Spawn position of the fire station moved each time the game was reloaded after a save. | N/A as this has been patched. Spawn position of the fire station is now set to be the same position rather than a relative position which caused it to move after each reload of the game. This was a small bug we noticed whilst implementing the save game feature. |

Code coverage

Much of the newer features we implemented for this phase of the assessment were tied to core game assets that we could not implement unit tests for, thus we could only rely on code coverage for the entire application.

*Code coverage at different stages of the assessment*

| Assessment | Coverage |
|---|---|
| End of stage 3 | 1% |
| End of stage 4 | 4% |

While this does show a relative increase in code coverage, it still leaves the overwhelming amount of our code uncovered and represents a lack of sufficient testing and leaves us unable to deem our code of an overall high quality by means of this metric.

Performance testing

We decided to also perform a performance test on our code in order to determine how the game manages physical assets whilst running.  Analysis of the task in windows task manager revealed that the process appears to continue to grow in memory, taking up an increasing amount as time goes on. This poses a concern to all users as we have yet to identify an upper bound for this growth, indicating a memory leak.

Short bursts of playtime tend to keep this memory consumption moderate, often in the range of 100MB to 350MB however as it continues to rise at a semi-constant rate over time (approx. 1 - 5 MB per second) means that a lower spec system may struggle very quickly, and even a more powerful one eventually being unable to cope after an extended period of time.

Although we did not notice this error near to the end of our development cycle, it was our consensus that this error was carried over from the previous assessment stage. In future work it may be a sensible decision to perform a performance test upon receiving work from another developer in order to identify a problem such as this sooner.

*An image of the resources consumed by the game moments after launch (Image A) and after a period of continuous usage (Image B).*

*Image A*

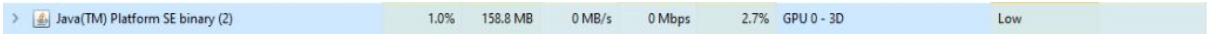
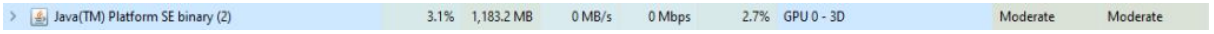



*Image B*




---

Looking at these pieces of data it is clear that our testing was lacking, due to both a substandard code coverage percentage and a lack of unit testing for the newer features implemented alongside the evident memory leak.. Given more time and resources - and a different work environment over the past couple of months - this may have been resolved at an earlier stage.

**Requirements breakdown and evaluation**

A document with a complete breakdown is available on our website.

| **User requirements** | SHALL | SHOULD | MAY | All |
|---|---|---|---|---|
| Total | 17 | 2 | 0 | 19 |
| Met | 17 | 2 | N/A | 19 |
| Percentage covered | 100% | 100% | N/A | 100% |

All user requirements were met by our implementation. During the consideration of the requirements we removed those that were rendered superfluous or redundant as it pertained to our views of the project and the more recently implemented features. This included all of the MAY requirements, hence why there are none here.

| **System requirements** | Functional | Non-functional | Total |
|---|---|---|---|
| Total | 20 | 4 | 24 |
| Met | 19 | 4 | 3 |
| Percentage covered | 95% | 100% | 95.83% |

The system requirements that we deemed was not met by our work was SFR_ET_IMPROVE_CONSTANT, which we did not implement due to time constraints and the fact it was a requirement carried over from the previous developer's outlook, and we did not feel it would improve the game's quality in a meaningful as it pertained to our user requirements.

Overall as our User requirements were completely met and of our System requirements, only one was not met, we judged our project to have met it's requirements to a successful degree - despite our subpar testing.

Final requirements list:
https://tomemmerson.github.io/krojanhorse.github.io/a4/final.pdf