SEPR 2019/20 Assessment 4

Team CheatCodez

Jonathan Grout, George Lesbirel, Cheuk Wang Wu, Lauren Quarshie, Muaz Atif, Lillian Coultas

Evaluation and Testing Report

Our evaluation approach

*Requirement IDs are shown in **BOLD***

The approach we adopted for evaluating our product was simply to play through the game, known as play-testing, and consequently make team judgements as to whether or not our product met specific aspects of the brief. A clear disadvantage of this however would be our intuition as developers of the game, which may result in biased judgements i.e. not acknowledging a potential lack of certain features. On the other hand, the majority of the criteria were relatively uncomplicated to test. An example of this was the requirement for 6 fortresses and 4 fire trucks, which can be verified by playing the game and ensuring these features are accessible to the player.

Our approach to evaluating more complicated criteria was to collect feedback from users after they played through our game. An example of this was the evaluation of the requirement **GAME_DOCUMENTATION**, in which we let players read through the tutorial shown at the beginning of the game and asked them if they clearly understood the conditions for winning/losing the game. Various value judgements also had to be made on subjective requirements, such as what we as the developers deemed as violent (**NO_VIOLENCE**). In situations such as these we used our own opinions and experiences, and also referred to official guidelines where possible (such as the PEGI rating system for audience suitability in the context of the **NO_VIOLENCE** example above).

The product brief will be considered as “met” if and only if the specified requirements are implemented, and the associated manual/unit test has passed successfully. If we were not able to implement said requirements, a justification was made accordingly.

Our approach to testing

In order to evaluate our software quality, we focussed on measuring our code against the metrics of readability, reliability, usability, complexity and maintainability of the code base as we believe software that meets these standards can be considered of quality.

Readability

Ensuring the readability of our software was a main concern for us, as it can directly affect the maintainability of the code. We made sure our code was indented and well-formatted, which is assisted by Intellij IDEA (Our chosen IDE). We also ensured functions and variables were thoroughly commented and explained, so that it would be easier for anyone who decided to look at our code to understand.

Maintainability

As a guideline, we used the IEEE standard for software maintainability : *“The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment.”* [1]. To evaluate maintainability we mainly carried out static testing, which involved ensuring variables were not hard coded as well as checking that the comment ratio was appropriate (around 10-15%

for our code). We also aimed to achieve a specified requirement using as few lines of code as possible.

Usability
Usability is a measure of how easy and straightforward the game is in terms of user experience (not to be confused with level difficulties). We decided to invite 10 players to complete a set of objectives, consequently providing feedback of a score out of 5, on how user-friendly the game is. If 70% of users have a score over 3 we deemed the game as fairly usable, as the score reflects that most players feel that it is straightforward to play the game.

Reliability
Reliability can generally be tested by running the software for a set amount of time to calculate a mean time between failure, if any. We decided to each play the game for 30 minutes, and if a mean time between failure is more than 15 minutes, we deemed it as appropriate, as typically it will only take around 5-10 minutes for players to finish the game (in accordance with **TIME_ACCESSIBILITY**).

Complexity
Complexity can be a difficult item to evaluate, as there are multiple ways to complete a defined requirement. Our goal on evaluating complexity was as long as the code is easily readable and maintainable, and is not causing performance impact to our game such as lags or stuttering, then it is considered as acceptable.

In order to ensure the quality and functionality of our final product met our product brief, we have implemented and recorded a series of manual and unit tests. Our tests are written taking the product brief into account, so that we are not only assessing the quality of code, but also the functionality requested by our stakeholders.

For assessment 4, we took over from Mozzarella Bytes who were working on the same project as us for assessment 3. This meant that when we started working on features for assessment 4, we were quickly able to understand the code of the project. Mozzarella Bytes had fully tested everything they had made previously when working on the project meaning that it was only necessary for us to test the new features.

Similar to the previous assessment, we used a traceability matrix **[https://teamcheatcodez.github.io/project-kroy/supplements-final/tmatrix-a4.pdf]** to identify which requirements were checked by which tests. All requirements have at least one test checking them. In total there are 145 tests per requirement, which is a high number and would suggest that the testing is complete and provides coverage over the whole project. As we and previous groups have been testing much of the game manually, it is tough to provide an exact percentage of how much coverage our testing provides, but as every requirement is being tested it should be a relatively high amount.

The majority of the tests conducted were manual tests for two main reasons. Firstly, manual tests are the easiest to conduct as they do not require any extra code to be written in order to be run; meaning we could have a greater number of manual tests. Secondly, manual tests

test what the user would see when playing the game so it was most obvious here whether features are implemented correctly or not making them easy to test. Not all components of the code could be tested manually and in those situations we tried to provide unit tests to check that they are complete.

Meeting the requirements

*Requirement IDs are shown in **BOLD***

In order to accurately evaluate the extent to which our game meets the requirements, we have decided as a team to use the IEEE-STD-610 definition of software validation, which is as follows : *"The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements"* [1].

After inspecting the code and playing the game to completion multiple times, we found that the majority of the user requirements had already been implemented from the previous assessments. This allowed us to focus on the implementation of the new requirements relevant to assessment 4.

The only requirements that **have not been met** are those relating to a shop functionality within the game i.e. **OPEN_SHOP, BUY_ITEM, OPEN_SHOP_FUNC, BUY_ITEM_FUNC, GAIN_INCOME_ FUNC**. Due to time constraints, we decided not to implement any of the in-game shop requirements, as these were not necessary to meet the project criteria; they were only 'May' priority, not Shall, and therefore not required. Deciding not to implement them gave us time to focus on making the best possible game, and do our best to make sure no other aspect of the game had a rushed, buggy implementation.

Both the functional and non-functional requirements have been demonstrated and can be confirmed through a simple playthrough of the game until completion (win or lose). The performance requirements **GAME_DOCUMENTATION, GAME_ACCESSIBILITY** and **OPERABILITY** have been met by the game being playable on all standard computers in the computer science labs.

In terms of new requirements added for assessment 4, our team has implemented a difficulty setting accessible from the main menu in which the player can choose from an easy, normal or hard game. Also now accessible from the main menu is a game saving/loading functionality which will allow a player to save their game state and consequently load said state when wishing to continue gameplay. A new, optional feature now implemented is a rewards system in which a player is rewarded for completing tasks in a specified time period.

A final updated version of the requirements specification can be located here : https://teamcheatcodez.github.io/project-kroy/supplements-final/recspec-a4.pdf

References

[1]"Software Maintainability - What It Means and How to Achieve It - IEEE Journals & Magazine", *Ieeexplore.ieee.org*, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/5221065. [Accessed: 17- Apr- 2020]