Module	SEPR
Year	2019/20
Assessment	4
Team	FarmJabStudio
Members	Jay Brooks, Amy McArragher, Rian McQuillan-Howard, Benjamin Kelly, Ahmed Tariq, Marcel Miro Puges, Faaiz Shanawas
Deliverable	Evaluation and Testing report

Evaluation and Testing Report

Evaluation Report

Part A

To determine if we had met the brief, we used the product brief's [1] together with testing. The reason behind this approach, is that we consider the prerequisites of the project, as well as our client's needs, to be a fundamental basis for feedback on our progress towards the project's completion.

In order to do this, we looked at 2 things. We first compared the features that we implemented with the requirements set out. Then we tested these features, matching tests with requirements to ensure a full spread of testing of the product. We did this with each assessment we were given, following what the groups from which we inherited the project did. For this final assessment we faced some challenges with testing (see more on this later in the testing report), however we still managed a full test suite of all the requirements set out.

At the start of assessment 4, we received 3 new requirements that we had to implement to our game, therefore we had to modify the requirements document [2] from the previous assessment to include these new changes. After that, we started by comparing the required features with the game itself. Our first approach to do this was to play the game and check if we could see the brief's features included in the game.

Apart from that, we also examined the requirements document and reviewed the completed and pending requirements to check if the brief's specifications were already included. We also had to make sure that no requirements came into conflict with one another or with the brief's features. Finally, we had to ensure that all requirements were clear and left no room for interpretation, including functional and non-functional requirements, given that this could have led to a misinterpretation of the game's actual progress.

As we came close to the assessment's deadline, and the implementation phase of the project was completed, we had to re-evaluate our game. Initially, we did that by playing and testing. Our game consists of 4 fire trucks, 6 ET fortresses and 5 power-ups, each of these being different and having their special ability, such as a greater movement speed or greater attack damage. The game is based on difficulty levels, meaning that the player can choose between easy, normal and hard, and these difficulties change the base damage, fire rate and health of the ET fortresses, as well as the game's score. As well as this, we also included the feature, where fortresses are harder to kill over time, and after a fixed time, the fire station gets destroyed so that fire trucks can't regenerate health or water anymore. Our approach on giving our game the ability to save its progress, was done by saving all of the game's current progress to a local file named '1.kroy', so that when we press the button 'Load Game' on the menu screen, we would only need to scan if some local file named '1.kroy' exists to load its progress.

In conclusion, due to missing details in our game, we can't say that we have met the brief to the full extent, but we do believe that we have created a well-rounded game that meets most of the client's needs, and in regards to gameplay we have implemented every aspect of the game in some form or another.

Testing Report

When we inherited the product for assessment 4 from NP Studios, we inherited a full test suite (*https://npstudios.github.io/testing/#assessment-3)*, however for assessment 4 testing was rather challenging. We inherited a good unit test suite from the previous group; however, we received no black box user tests. This meant that we knew that a lot of the code worked as intended in the artificial bounds that the unit tests created for it, however we did not know how different parts of the game would interact with each other. Because of this we decided to add some black box user tests to make sure that we tested as thoroughly as possible.

Before we implemented anything new to the project, we double checked all of the tests, making sure that they worked from the previous group. These had a 100% pass rate, showing that we inherited a strong code base to work from. We then proceeded to implement different aspects of the game.

Unfortunately, when implementing the powerups, the implementation team changed the way that rendering works within the game, this broke the headless backend that allows us to write tests with libgdx and junit. Because of this, the classes that require rendering (the classes that inherit from Entity()) were not able to be tested. However, we can (and did) test that the rendering works, as this is a visual test not possible with unit testing.

Because of this change in rendering, our functional unit tests went from 44 to 25. This is very unfortunate, however the methods tested in the tests that we lost were not changed by our group, this allows us to have a reasonable assumption that this code still works due to the artificial environment created by unit testing. What we didn't know was how these classes would interact with the new rendering system and new features. Luckily this is all visual, so we can be relatively confident running black box user tests in order to determine if the code works or not.

Due to this we deemed it not time effective to try and change the code base to get the unit tests to work. This is obviously not ideal; however we still feel that the game is tested thoroughly enough despite this.

By looking at our test design and documentation document (https://marcelmiro.github.io/SEPR-Assessment-4/files/Assessment4/testing/Current%20test%20design%20and%20documentation.pdf?fbclid=IwAR1YCZGkUMu0xynvqctGuqxsRwbT-N_PeUaQZJ13HOC90ZLuarYDS5C40D8*)* [3] you can see that we have a 100% pass rate, with both the old tests which were re-run after we implemented the new features, and the new tests, which were testing the new features.

We have included an analysis of the new unit test suite here (https://marcelmiro.github.io/SEPR-Assessment-4/files/Assessment4/testing/Unit%20Testing%20Analysis.pdf) [4] this details the effect that losing many of the previous groups unit tests had on our number of tests, line/method coverage and time efficiency.

The Junit tests are included here (*https://github.com/AlmightyWishPig/FarmJabNP/tree/master/tests*) to run them, clone the repository and right click on the tests 'src' folder and select 'run tests'. We have 25 unit tests and a 100% pass rate. Unfortunately, due to the implementation team making classes that inherit from 'Entity()' untestable, these had to be removed.

We have included the black box user testing evidence here (https://marcelmiro.github.io/SEPR-Assessment-4/files/Assessment4/testing/Current%20User%20Testing%20Evidence.pdf) [5], this details how we went about conducting the different tests, including screenshots showing the code working. These black box tests cover all the visual aspects of the game. Making sure that everything interacts visually as well as in the code.

We have included a traceability matrix here (https://marcelmiro.github.io/SEPR-Assessment-4/files/Assessment4/testing/Traceability%20Matrix.xlsx) [6], this shows the tests coverage of our list of requirements. Allowing us to make sure that we have tested all the different requirements from the end user.

Our testing for this assessment mainly focused on testing the new features that were implemented, making sure that they worked and didn't break any other part of the game. Since we didn't alter the code that we obtained from the previous group we decided that re-creating the old tests would be over testing and maintaining a consistent level of testing throughout the game was more important.

As well as testing for assessment 4, we needed to look again at the tests done by the groups before us. For assessment 3, NP Studios inherited a significantly sparse test suite from DicyCat in assessment 2. Due to the challenges with testing with libgdx, DicyCat had no unit tests, and simply relied on black box testing. When NP Studios took on the project, they rectified this and added a significant number of unit tests to the project, however, took away all of the black box user tests.

For our testing we took both different testing methods and attempted to do both, despite our fewer number of unit tests, our code and product as a whole is sufficiently tested to allow us to say that the game is a final product

Part B

A full list of updated requirements can be found here:
https://marcelmiro.github.io/SEPR-Assessment-4/files/Assessment4/testing/Part%204%20Requirements.pdf

Game element	**Associated requirement fulfillment and discussion**
Fire truck	Fire truck movement is completely controlled by user (**UR_INSTRUCT_ENGINES, FR_PRECISION**), Game ends once all Fire trucks destroyed (**UR_END_GAME, FR_END_GAME**), Each truck's health and fuel is indicated by two bars above it's sprite (**UR_REFILL_WARNING, UR_SEE_HUD**) to show the user when they need to refill although there is no explicit notification. Fire trucks are not able to pass through scenery, ET fortress or ET patrols (**UR_COLLISONS, FR_DENY_COLLISIONS**). The Fire truck is able to collect five different power ups by passing through them (**UR_FIRE_TRICK_POWERUP, FR_FIRE_TRUCK_POWERUP**). All Fire trucks have different stats (**FR_UNIQUE_ENGINES**). There is no choice between fire trucks for simplicity so no need for strategy (**UR_STRATEGY**) . There's only one level however, the Fire trucks start with full health at the beginning of it (**UR_FRESH_HEALTH, FR_AUTO_REPAIR**). When a Fire truck is destroyed there is no notification it has been destroyed as user is switched to a new fire truck when the previous one is destroyed **(FR_ENGINE_DESTROYED**).
Fire station	The Fire station is destroyed after the displayed 900 second countdown timer (**FR_DISPLAY_TIMER**). The fire trucks can refuel and repair at the firestation (product brief.) There is no notification when the fire station is destroyed as as we felt it was unnecessary due to the clearly labelled in game timer which indicates how long until the fire station is destroyed (**UR_ATTACK, NFR_INGAME_WARNING**)
ET fortresses	The game ends once all ET fortresses are destroyed (**UR_END_GAME, FR_END_GAME**), they are destroyed when their HP reaches 0 (**FR_ENEMIES_DIE, UR_INSTRUCT_ENGINES**). All ET Fortresses have unique stats and attack patterns (**FR_UNIQUE_ENEMY**). The damage and speed of fortress is determined by the difficulty level (**UR_GAME_DIFFICULTY, FR_GAME_DIFFICULTY**)
ET patrols	All ET patrols can be destroyed once their health reaches 0 (**FR_ENEMIES_DIE, UR_INSTRUCT ENGINES**). The damage and speed of ET patrols is determined by the difficulty level (**UR_GAME_DIFFICULTY, FR_GAME_DIFFICULTY**) All ET patrols have the same stats and attack patterns due to time constraints and unique ET fortress stats was enough for requirements (**FR_UNIQUE_ENEMY**)
Game conclusion	Game is won when all ET fortresses are destroyed, and lost when all firetrucks are destroyed and user is given the option to start new game or exit **(FR_NEW_LEVEL)** (**UR_END_GAME, FR_END_GAME**, product brief). However the game does not automatically end after a time period (**NFR_TIMER**).
Minigame	Completing the minigame restores the fire trucks health and fuel and gets progressively complicated with each attempt (**UR_MINIGAME**). The minigame is simple and easy to complete **(FR_MAIN_FOCUS**) The minigame is activated by pressing "E" while near the firestation, instead of a "refill tile" (**FR_OPEN_MINIGAME**) as it was a nice way of making the minigame fit in the context of the main game
Misc user experience	The game contains a "new game", "load game", "option" and "exit" button, it also has a "minigame" and "controls" button. (**UR_START_SCREEN, NFR_USER_INSTRUCTIONS, UR_EASE, NFR_MENU_UNDERSTANDABLE, NFR_BUTTONS**) It does not have a credits button as we felt this would not contribute to user enjoyment. It is possible to pause the game (**UR_PAUSE, FR_PAUSE**). The game contains a map, and above each fire truck is a progress bar indicating health and fuel (**UR_SEE_HUD, UR_REFILL_WARNING**). The game has been playtested to ensure appropriate pacing and difficulty (**UR_INTEREST**), The user may select a difficulty level upon starting a new game (**UR_GAME_DIFFICULTY, FR_GAME_DIFFICULTY**), user may save the current games progress and load it at a later date, however the game does not automatically save, we felt this was necessary due to the short length of the game (**UR_GAME_SAVE, FR_AUTO_SAVE, FR_SECURITY, NFR_SAVE, NFR_AUDIT, NFR_RESUME_TIME, NFR_SAVED_CONTENT**). The game contains music (**FR_PLAY_MUSIC,UR_MUSIC**). The game only contains 1 level, this was due to time and manpower constraints on the project (**FR_6_LEVELS, FR_NEW_LEVEL). **Game screen and user interactions controlled by the game stack with the current game state at the top of the stack **(FR_GAME_STATES).** The colour scheme and artwork is bold and all important text is laid out in clear large easy to read language and font in order for the user to read clearly **(NFR_READABILITY, NFR_ARTWORK, NFR_OPERATORS).** All

	user interactions with the game are instant with less than 1 second delay in order to make the game as easy as possible to play by giving the user full control. **(NFR_INTERACTIONS, NFR_OPERATORS).** Through rigorous testing and debugging the system is free from glitches. **(NFR_ERROR_PRONE)**

References

[1] - SEPR “Product Brief: Kroy” [Online]. Available at: https://vle.york.ac.uk/bbcswebdav/pid-3396020-dt-content-rid-8681478_2/courses/Y2019-006404/product-brief%281%29.pdf

[2] - FarmJabStudio “LIST OF REQUIREMENTS FOR KROY” [Online]. Available at: https://marcelmiro.github.io/SEPR-Assessment-4/files/Assessment4/testing/Part%204%20Requirements.pdf

[3] – Test Design and Documentation. Available at: https://marcelmiro.github.io/SEPR-Assessment-4/files/Assessment4/testing/Current%20test%20design%20and%20documentation.pdf?fbclid=IwAR1YCZGkUMu0xynvqctGuqxsRwbT-N_PeUaQZJ13HOC90ZLuarYDS5C40D8

[4] - Unit Analysis. Available at: https://marcelmiro.github.io/SEPR-Assessment-4/files/Assessment4/testing/Unit%20Testing%20Analysis.pdf

[5] - Current User Testing Evidence. Available at: https://marcelmiro.github.io/SEPR-Assessment-4/files/Assessment4/testing/Current%20User%20Testing%20Evidence.pdf

[6] - Traceability Matrix. Available at: https://marcelmiro.github.io/SEPR-Assessment-4/files/Assessment4/testing/Traceability%20Matrix.xlsx