Module	SEPR
Year	2019/20
Assessme nt	4
Team	Early Bird
Members	James Little, Ryan Vint, Adam Lynch, Georgina Martin, Kheng Yeoh, Tanay Malde, Cameron Devane-Waters
Deliverable	Evaluation and Testing Report

Section A

Testing and Evaluation Approach

Throughout the process of creating this product it has been both evaluated and tested many times using various methods. These have mostly consisted of unit testing to evaluate the low level logic of the code and high level end to end manual tests some designed to test the structure of the code some designed to evaluate some of the more complex functional requirements that automated tests were not able to be designed for.

For this assessment only a few small additions have been added to the overall game. This means that a few extra requirements had to be added based on these new product demands. To account for this smaller addition to requirements that faced in previous assessments we will be focusing a lot more on testing the quality of the code itself as much of the code has already been evaluated against the existing requirements. This will mean we will use a lot fewer low level unit tests and end to end tests based on functional requirements and more high level tests to test the quality of the program and make sure there are no unwanted bugs or crashes in the code.

For this purpose we will need to try and test as much of our code as possible so we will be using a combination of UI tests that tests all parts of the user interface of the game reactions and functions as they have been designed to, and exploratory testing performed by all our programming team. We will design UI tests to test all UI elements of our game, testing that things like buttons and key presses do what the user will expect them to do and cause no bugs or crashes.

Exploratory testing is done by all of the designers and programmers of the game, running the code and trying to perform actions that may cause bugs and errors, such as pressing the wrong keys at the wrong times, pressing buttons quickly over and over etc. These tests are judged to be complete and successful on code coverage, the percentage of the total lines of code that were run in each exploratory test. High code coverage in multiple exploratory tests with no errors reported means that we can safely assume that our code is of appropriate quality as we were unable to crash or cause it to error even with many attempts. Getting a high code coverage will also allow us to show that our code does not have wasted lines in it, sections of code that never run and are obsolete again showing that the code is of high quality.

We will be defining our code as high quality and well tested if all of our exploratory tests reach over 85% code coverage and have no errors, or errors that have been detected are fixed and then retested in subsequent runs and found to no long cause error. We will also be checking that of the 85% tested in each run there are no classes with functions that are not executed in any of the runs to check that all of the code is being used and we do not have any redundant code. We choose 85% as the minimum coverage as across 7 tests combined with checks from redundant code, we believe this will mean that all of our code has been tested sufficiently while not taking up a significant amount of everyones' time to reach code coverage of say 99%. We will be using the eclipse IDE's built in code coverage tool to calculate the code coverage of each attempt. Our code will also need to pass all UI tests to be considered high quality, and the unit tests will all be run before the final exploratory testing to make sure that all errors are found in our code.

However, even if our code is high quality it must be evaluated to make sure that it actually fits the brief that we have been given. To make sure we achieve this we will be using a traceability matrix of all of the updated requirements, containing the final requirements from assessment 3 and the additional requirements introduced in this assessment [1]. We will be running all appropriate tests from previous assessments as well as our new end to end tests written for our new requirements. We will not be writing new unit tests for this assessment as creating unit tests for the newly implemented features would be difficult to design when compared to manual tests. Testing things like saves still loading correctly when the game is closed and restarted is very difficult to do automatically as the tests would have to simulate program termination, where manually saving, closing and reloading the game is a lot easier and we can visually check that the save is correct easily. So we have decided that making sure the code itself is high quality is more important than spending time mocking all of the correct features to unit test these requirements.

Actual Testing

In the end to end tests designed for the new requirements our system passed 8/8 [[2]. These tests were designed around the new requirements and as can be seen from the traceability matrix [3] tested that all new requirements were either directly met, in the case of UR_SAVE, or met by completing all related functional requirements, like in the case of UR_POWERUP and UR_DIFFICULTY. In the first run of testing we realised that the GUI was not updating for fortress' varying attack values between difficulties but this issue was fixed to make the GUI display the correct values.

After combining the traceability matrices for the last 2 assessments to allow us to check that all requirements had been tested there were 3 requirements that had not been previously tested, FR_WATER, FR_ACCESS_MINIGAME and FR_STATION_DESTROY [1]. This meant that for a successful full evaluation of our code tests had to be designed for these requirements. We also found 2 user requirements UR_TRUCK_SPACE and UR_PATROLS that weren't directly linked to functional requirements but were still testable via end to end tests, unlike user requirement UR_ENJOYIBILITY which we can't end to end test or UR_FORTRESS which is linked to many functional requirements that we have successfully tested.

We ended up with 6 additional tests [4] and we passed 5/6. This failure was due to a change in the way that FR_WATER was implemented by the previous team. Instead of increasing the health of fortresses over time they increased the damage. This means that FR_WATER fails however the original requirement from the customer was to improve the fortresses overtime, the fortress still improves it is just in a different way so the customer's request has still been fulfilled the requirement FR_WATER just needs to be removed. All other tests have been passed meaning all functional requirements now have a test related to them as well as all the user requirements that can be tested via software testing.

When running all the old tests from previous assessments we did not include the failing tests from the previous assessment we did not include the tests that were already highlighted as failing in the previous project. These tests failed due to a lack of mocking not a code based error. The details of these tests can be found here https://ds1410.github.io/mozerella-bytes/assessment3/Change3.pdf, along with the details of the assessment 2 tests found here https://ds1410.github.io/mozerella-bytes/assessment2/index.html#testing. Combining these tests we had 73 additional manual and unit tests which we performed, all to passing outcomes.

UI Testing revealed no bugs but some inconsistencies that lead to confusion about game function. We had issues with the pause button not remaining a consistent colour when paused. The button was supposed to remain grey while the game was paused and green while the game was unpaused, however if while paused the player interacted with any of the other onscreen buttons, such as the controls, and then returned to the main game the button would turn green but the game would still be paused and the paused text would still display. As this was only a small issue we decided it would be a better use of our time to just remove the turning grey feature instead of attempting to solve all the cases when it turned the wrong colour and then use the time to improve the game and write up in greater ways. We also found inconsistencies with the use of the esc key, it wouldn't work in certain screens such as in the save screens, it would close the controls tutorial in some screens but completely close the game in the mini game controls screen. We also decided that within the game itself esc should take the player to the main menu as opposed to closing the game as closing the game without warning was frustrating for the user. In the end we passed 34/34 UI tests allowing us to conclude we have a quality UI. Full results for the UI tests can be found at [5]

Exploratory testing was carried out by all 7 members of our team on the game code in an attempt to find niche case errors in the code and try to cause the program to error so we can fix it before submission. We were also looking for redundant sections of code that could be deleted to clarify and improve the quality of our code. There was an error with pausing and movement highlighted however due to the size of the fix required and the limited time we had we were unable to sort this issue, it does not cause game crashes just small bugs with trucks, details can be found here [6].

Section B

Evaluation

We believe that our final code is a success as we have met all of our requirements [1] that our customer requested while also having high quality code. As can be seen from our traceability matrix [3] all of our functional requirements have been tested and have passed those tests, we have even used software testing to successfully show our implementation of unlinked user requirements like UR_PATROLS. There are some requirements that we could not test directly but were clearly achieved, these include UR_PLAYER, UR_PC, UR_CODE, UR_PLAYABLE and NF_PC which are all requirements to do with things like how the game must run, it must be coded in java, must be a single player game etc. These are all clearly true but software tests cannot be designed for them. The game also runs through the main success scenarios of both use cases passing them too.

The other 2 non-functional requirements have also been met, NF_RESPONSE states that the game should respond quickly to user actions, within the game all button presses and interactions with trucks and fortresses feel almost instant with no noticeable delay even when performing actions like loading and saving. When testing for NF_CONTROLS a family member was asked to play the game and given no further instruction that what was given by the game itself and got the hang of the controls and the aims of the game well within the allowed 2 minutes, with more time we would have liked to test this requirement further but due to current circumstances and time restrictions we were unable to test this on a larger group.

All of the information above shows that all requirements of all kinds have been met, however our game is not a total success due to the error found with pausing and movement of trucks in our exploratory testing. If given more time we could fix this error and our game would be a complete success and we could consider it a 100% finished project, however due to the time constraints and the lack of transparency from the previous team we were unable to identify and solve this issue in time. If the other team had informed us of the issue in their previous testing report then we would have been able to fix it in time, however due to the lack of knowledge of the issue we did not have time to solve it since it was discovered too late. The error itself is small and does not happen in a large percentage of game playthroughs, and also does not cause the game to crash, it is still playable meaning that our game can still be considered a success as most players will not discover the issue, however with more time we would have been able to solve the issue and submit 100% completed code.

In conclusion, due to the game meeting all of our requirements and use cases, as well as having extensively tested it and only having 1 small bug that was due to a previous teams coding and with more time or if the other team had highlighted this issue in their testing we could have fixed this issue we consider our project a success.

References

[1] Final requirements
https://jameslittlecs.github.io/Kroy_Final_Assessment/assessment4/Req4.pdf

[2] Our requirements end to end tests
https://jameslittlecs.github.io/Kroy_Final_Assessment/assessment4/EndToEndTestsOurRequirements.pdf

[3] Final traceability matrix
https://jameslittlecs.github.io/Kroy_Final_Assessment/assessment4/TraceabilityMatrix.pdf

[4] Old requirements end to end tests
https://jameslittlecs.github.io/Kroy_Final_Assessment/assessment4/EndToEndTestsOldRequirements.pdf

[5] Full UI testing
https://jameslittlecs.github.io/Kroy_Final_Assessment/assessment4/UITests.pdf

[6] Exploratory testing report
https://jameslittlecs.github.io/Kroy_Final_Assessment/assessment4/ExploratoryTesting.pdf