Software Testing Report

Group 1
Assessment 2

Emma Hogg
Isaac Rhodes
Isioma Offokansi
Toby Meehan
Seungyoon Lee
Ali Yildirim

(a) When we first received the project it was important to test the project before continuing with the implementation and change report. This was vital to do in order to find sections of code which were not covered by tests as well as which parts of the code had failures. We did not want to change code that is not working as it could cause further problems down the road. In order to test the current code as efficiently as possible, each member of the team took a different clear section to create tests for. For example, one person tested the “Entity” folder of code and another the “Stations. We therefore could progress with the testing quickly so we could start to develop as soon as possible.

Automated testing has many advantages over manual testing as manual testing can take up a lot of time and power. We therefore understood the importance of ensuring we used automated testing as much as we possibly could. We used a lot of automated testing to ensure all Assessment 1 were met and there were no bugs. We then manually tested the remainder of the code required to prove that each of the requirements were met. We did this to ensure that all the previous requirements were met and working correctly for the customers needs otherwise the project would be unsuccessful with missing attributes.

Any tests that failed were noted down in a document so that the team were aware of these and they would be fixed. We made sure that the fixing of the failures was a priority as altering failing code would cause problems further down the road when trying to use the faulty code. We ensured that no one had an extreme volume of work and that it was distributed evenly to ensure efficient progress.

The tests that we coded were mostly unit tests which looked at each of the functions within the code to ensure that they did what they were expected to do. To do these tests we used JUnit which is a Java testing framework. This perfectly fits into our project due to our code also being written in Java. It clearly showed whether a test passed or failed and therefore it was easy to identify any problems. These tests would be run after every change to the repository to ensure that all the requirements for assessment were still met after the changes to the code. After we fixed all problems in the inherited code we needed to ensure that any new code was tested thoroughly. We had some members of the team who focussed on implementation while others tested. This ensured that all members of the team had an understanding of the code base in case any team members required help with any of the coding or testing. These tests were also run after every code change to ensure that nothing was causing any failures while developing the code. Without these there would be a risk that near the end of the project an issue could be found last minute and cause a lot of problems and there would not be enough time to fix it.

Throughout this part of the project, it was important for our team to remember it is not possible to exhaustively test the code. This is vital to remember as there is a risk that time could be wasted testing unimportant code rather than focusing on the true requirements of the project. As a team, we focused on the coverage of our tests. As described previously, we did not require 100% coverage as this may be too costly however a high percentage of coverage is needed to ensure that the main bulk of the code is covered. We regularly used statement coverage to ensure that the majority of functions that need to be tested are covered to prove that a requirement is met. In doing this however, it was important that we look at mutation coverage. This ensures that given a fault, does the code deal with it? This form of testing has ensured that there is nothing the user can do during the game that can break the code.

Finally, we also ensured that we created a traceability table in order to prove that all requirements were met and successful using testing. We corresponded each requirement with one or more tests to prove these were met. This can be found using the following link: Traceability Matrix

Testing was a major part of our project to ensure that our game was functional with no errors or bugs. Our automated tests covered 73.5% of our overall testing while the remaining 26.5% were manual tests. This proved the high volume of testing that we felt was required to ensure that everything was working as expected with no bugs. Since we covered 50% of the code using automated JUnit tests we ensured that there was as little manual testing as possible as this can be very costly and time consuming. We did not test all of the lines of code due to the fact this would be inefficient and not essential. We picked the sections of code that we felt were vital to test in order to ensure the code functioned as expected without any errors or bugs occurring.

Furthermore, when testing our code, we considered that we needed to ensure that all requirements that were necessary had been implemented in the code and were working as expected. It was crucial that all requirements were met to ensure that our game was as successful as possible. In order to check that this was the case we used a traceability matrix. The traceability consists of matching the requirements to the tests that cover it. It helped to prove the fact that everything was covered by tests but also gave the tests meaning. The tests that the requirements can relate to can be both the automated tests as well as the manual tests therefore we needed to ensure that the tests were easily identifiable similarly to the requirements. The traceability matrix that we used can be found using the following link: Traceability Matrix

In order to ensure that our code was extensible, we needed to ensure that the code was thoroughly tested. Hence we covered the following areas when testing the code in detail to ensure that the code could be developed in the future with no problems:

- **Asset Tests** - This was used to ensure that all assets in the game were as expected and that nothing was missing that would affect the functionality of the game.
    - 100% of the asset tests passed
    - 100% of these tests were automated
    - This was vitally important to be 100% successful as we needed to ensure that all the assets that we needed to produce our graphics and sounds were available.
- **Customer Tests** - The customers are a vital part of the game and therefore these must function as expected. This covered how they moved, looked and placed their order
    - 100% of the customer tests passed
    - 81.8% of these were automated, 18.2% were manual - we required some manual tests in order to look at the graphics in the game.
    - We spent a lot of time ensuring that the customers functioned as expected. This is a major functionality that is vital for game play for the user. If this fails in any way the game would not be successful.
- **Cook Tests** - In order to play the game, the player must be able to control cooks and use them to interact with different stations. These tests ensured that this functionality was possible with no bugs
    - 100% of the cook tests passed
    - 85.1% of these were automated, 14.9% were manual - we required some manual tests in order to look at the graphics in the game.
    - The user must be able to control chefs in order to play the game. With this part broken the game would be unplayable and therefore unsuccessful. We therefore required that these tests all passed otherwise the game would be completely unusable.
- **Ingredients and Recipe Tests** - The user must know what they must make in order to play the game. Therefore, the game must show the correct recipe and allow the user to make this using the correct ingredients. We therefore had to test that this was the case.

    - 100% of the recipes and ingredients tests passed
    - 88.9% of these were automated, 11.1% were manual - we required some manual tests in order to look at the graphics in the game.
    - For the game to be successful the user must be able to see and make the recipes that the customer is demanding. If these recipes and ingredients do not appear or fail to work the game is impossible to play. We therefore needed to ensure that 100% of these tests passed.
- **Controller Tests** - The controller tests are responsible for ensuring that all the behind the scenes works as expected. Without these there is a risk that something could go wrong in the background therefore these tests are vital to prevent something occuring the future behind the scenes that is very difficult to find and solve
    - 100% of the recipes and ingredients tests passed
    - 100% of these tests were automated
    - It is vital that the game is functioning in the background. These controller classes are used to control different graphics, sounds and functionalities in the game. Without these the game could become limited or unplayable. We want the game to be inclusive as well and these controller classes make a vital contribution to this. We do not want any bugs or errors in these classes therefore and need them to function as expected.
- **Gamemode Tests** - The game must have two different game modes therefore it is important to check there are no bugs in either of these game modes. These tests cover this to ensure that this is the case.
    - 100% of the gamemode tests passed
    - 100% of these were automated
    - It was vital that 100% of these tests passed as we needed to ensure it was possible for the user to play both game modes. If either of these tests failed it would result in the removal of a major part of the game. These two modes needed to work to ensure success.
- **Power Up Testing** - A major requirement of the game is to allow users to be able to pick up and use power ups. These tests are used to ensure that the power ups appear and react as expected.
    - 100% of the power up tests passed
    - 100% of these were automated
    - It was vital that 100% of these tests passed as we needed to ensure this was a functionality that did not fail or cause any problems in the game. If this functionality was broken this would result in a lack of success for the game so it was vital that all of these tests passed.
- **Station Testing** - The user must be able to interact with the station using the chefs and the specific stations must respond in the correct way. These tests ensure that this is the case
    - 100% of the station tests passed
    - 100% of these were manual; we decided to do this in this way as the automated tests were becoming too complex and it was becoming difficult to test random variables.
    - We ensured that all of these tests passed as if there was a bug in this section the game would become unplayable. The user must be able to use the cooks and interact with the stations to produce the food for the customer. Without the stations, it would be impossible to fulfil any task therefore it is vital that all of these tests succeed.
- **Saving Testing** - The game must be able to be saved by our user at any point in time. We needed to test this as it was a major requirement that the customer requested therefore we spent time testing this section of the code.

    - 100% of the saving tests pass
    - 100% of these are automated
    - We knew the importance of ensuring that this functionality worked correctly without any bugs as even though it is not vital to the playability of the game it is an important requirement for the customer. Therefore, if this did not pass the game would not be successful.
- **Non-Functional Testing** - We need the quality of the game to be high in order for the game to be playable. Therefore these tests were made to ensure that this was the case
    - 100% of the non-functional requirements tests passed
    - 100% of these were manual
    - We manual tested each of the non-functional requirements in order to ensure that the graphics and the game were at a quality that the customer would expect. We did not want our game to be at poor quality and therefore be unplayable.

Our test results showed that 100% of our tests passed showing that our code was highly accurate with very no errors. This shows the reliability of the release of our code to any player who wants to use it. Due to this 0% of our tests failed. This is due to the high volume of time that we spent on the implementation to ensure that there were no bugs in the code. We also worked hard to eliminate any bugs or errors that were found when programming the code immediately so that our ultimate software would be error free. This helps with the extensibility of the code as in the future it will be easy to add to the code due to the fact it is already known that there are no bugs in the code from the beginning. All of our tests were complete and correct.

During the project we produced 30 manual tests all of which succeed on our current implantation. This is due to them all testing for major failures in the project. They all have a wide scope aimed at testing for key functional and non-functional requirements or user requirements, all of which are fundamental to the gameplay of the project. The main reason why these tests are manual is due to the fact that they have the advantage of being able to deal with random variables as well as using them to test to see if entities have rendered into the main screen and if they can be interacted with correctly. The manual tests that we carried out can be seen using the following link: Manual Tests. They have all been described in detail so that they can be repeated by any person wanting to extend the code and repeat the tests in the future.