Software Testing Report - Team 14

Babar Khan
George Rogers
Jacob Turner
James Crump
Chloe Remmer
Shijie Lin

Software Testing Report

Our Plan:

What to test?

We are aiming to test every requirement individually as laid out in the requirements specification. We want to go through each requirement and test that what we have implemented works and that it fulfils the requirements. Whilst this may not ensure that our code is well optimised, it should ensure that the customer is happy with our finished project as their requests for the code are fulfilled.

How will it be tested?

In order to test the code we want to first run unit tests to ensure that the different parts of the code are working as intended. This also means we can check many of the units against the requirements and see that if a requirement is not met, what unit of code may be causing the problem. As we have a very complex set of code it is not possible to unit test for many requirements as they are often dependent on many different units of the code working in tandem. For these units we will perform manual system testing to see if the project works wholly and fulfils the remaining requirements. We are also finally going to check the line coverage of the code to see that what we have written is being used fully and so that we can see if there are sections of the code that are redundant.

Why are we testing?

We are testing to make sure that the code is working as we intend it to and is fulfilling the requirements. We also want to try to remove any code that is not being used.

Who will be testing?

Jacob Turner, Babar Khan, George Rogers

The Tests

Passed

87/88 test passed successfully giving us a 98.9% pass rate. This is a very high pass rate and shows that we successfully managed to fulfil almost all the requirements.

Testing Table

Please see website or go here

Failed Test

FR_OBSTACLE_SPAWN

This test failed because our objects spawn randomly. It is the case that we could have made the objects spawn in a set pattern to ensure that there is never a set of obstacles that are unavoidable. We did try to limit the obstacles on screen to greatly reduce the chance of the obstacles being unavoidable. We decided that we would rather the game have a high level of replayability than be scripted on its difficulty.

Line Coverage

Our overview of line coverage can be found on our website or here

Some of our code is not testable and we have reflected that in our line coverage by having our score based on testable code, this excludes any empty lines, comments, or systems which cannot work with unit tests. Overall the unit tests covered 54.2% of testable lines of code, and 11.69% of overall lines of code. Any parts of the code that solely rely on rendering such as the screens have been omitted from the calculation as these cannot run in a headless application.

Traceability Matrix

Our traceability matrix can be found on our website or here

It covers our black box testing and represents the relationship between requirements and test cases. It shows that all requirements have at least one test case for them. It also means that we know which test cases we will have to change when requirements change. We have no redundancy as no tests are overlapping.

Evaluation

For many of the requirements we tested the desired output was rather vague or very difficult to test. For example, to test UR_UX we had to test whether the game was enjoyable. Given this is an entirely subjective test it is hard to decide whether we passed this test or not. Some other tests required far more time to complete than we had available to us, and whilst we passed these tests, we did not test every possible scenario of the requirement. This does not mean that they were useless tests, we can see that the code has fulfilled requirements even if it is not tested to its absolute limits.