Software Testing Report

Summary of Testing Methods and Approaches

Unit Testing

We carried out unit testing on individual functions and classes across our program using a wide variety of data for every test - including boundary data where necessary. This method of testing allows us to ensure that each individual function is working correctly and producing the correct output for its given input.

We believe that unit testing is a very appropriate method of testing for our project as it allows us to ensure that all individual functions in our program are correctly functioning, therefore allowing us to say with some confidence that our program is fully functional as a whole. By building unit testing into our code base through the use of JUnits, we have been able to automate the testing of every unit across the program, with automated test reports being generated. This integration of automation eases our use of unit tests and makes it both simpler and quicker to ensure that the code is fully functional - and by extension making it easier to see which units are functioning incorrectly.

Black Box Tests And Acceptance Testing

We have also made use of numerous black box tests to ensure that our code functions correctly from the users perspective. By using this testing method, we were able to prove the reliability of the front end graphical interface by executing every interaction that we would expect the user to make, and ensuring that the desired and expected output is achieved for each interaction. In addition to standard Black Box testing, we have also integrated a series of acceptance tests. These are a series of tests with a binary (yes/no) output that allow us to check that our system meets its requirements and therefore confirm that it is acceptable for delivery to the customer.

Black box testing is an appropriate approach for our project as although it is less in depth, it is faster to complete and ensures a that a broad level of external functionality has been met. By carrying out a wide range of acceptance tests, we are able to easily identify any requirements that have been missed, and therefore confidently state that our program works as required and is therefore ready for delivery to the customer provided our acceptance tests have passed.

Tests with the Customer

Unfortunately due to time constraints we were unable to carry out any tests with our customer. However ideally we would have carried out additional black box testing with the customer present, allowing them to naturally play through the game and identify any errors, issues or even improvements that they may have.

Report of Tests and Results

Report of Unit Tests

We carried out 24 unit tests on functions inside our game that provided getter or setter functionality to internal class variables. Where necessary we carried out boundary testing on this data to ensure absolute functionality in all cases. Out of these 24 tests, 100% passed, allowing us to say with some confidence that all functions tested work accurately. *(For evidence of these tests please follow the link to the website found below).*

When unit testing, we focused on getter and setter functions, as well as a few additional functions that provide added functionality. To ensure accuracy of these tests, we ensured that boundary tests were carried out in every eventuality where appropriate, and that all possible input and output cases were tested. We are therefore confident of the correctness of our tests and therefore the correctness of our code.

However it must be acknowledged that by focussing on only a subset of functions, some functional units in our program were left untested. This approach allowed us to focus our time on fully testing areas which we believed to be most important - hence ensuring that our internal classes worked as desired. However, this methodology has led us to skip the testing of classes and functions whose effects are directly visible by the external user - such as those which implement and adjust the visual display. To counteract the lack of truly complete unit testing, we substituted unit testing in these areas for Black Box testing - a methodology that we believed allowed us to be more certain of the true visual output of the functions from the user's perspective, therefore making it easier and quicker for us to judge whether these outputs are correct. Whilst we are aware that our streamlined approach to unit testing may have led us to miss certain fail cases, we are confident that our thorough black box testing of the product will have identified any fail cases that might have arose; therefore leading us to conclude that this method of testing was correct.

Report of Black Box Tests

We carried out 40 tests on our game to test its functionality from an external perspective. 40 tests passed fully, meaning that we achieved a pass rate of 40% on our Black Box Testing.

When writing our black box tests we aimed to fully test every area of our programs functionality that is exposed to the end user. To attempt to achieve this goal, we manually traced every path available through our program from the end user's perspective, and then wrote black box tests to confirm the functionality of every interaction - i.e. button press, key press etc. that it was possible for the user to make. Through our use of this methodology, we believe that we have tested every interaction and output available to the user, and can therefore be confident of the correctness of our tests and therefore the functionality of our program from the external users perspective.

Report of Acceptance Tests

We carried out 34 on our game to test its ability to meet the initial requirements of our project. Out of 34, 29 passed, a % pass rate of 85% - meaning that we achieved 85% of our original requirements.

When writing our acceptance tests, we planned to test everything against our list of requirements from the client. These tests were manually checked by team members running through both the executable of the code and the requirements documentation. Overall, we failed 6 tests. These included the player choosing different detective costumes at the start of the game which failed due to a lack of time to implement the different designs combined with the requirement having a low priority since it can be implemented rather easily at the next stage of development. Each suspect having an individual personality and irrelevant clues being provided also failed due to a lack of time to create these personalities, however this requirement is just an extension of the character information in the database and should require a small amount of content generation. The lose condition to falsely accuse too many suspects was also not implemented due to a lack of time although most variables needed to create this feature are already implemented and implementation should not be difficult in the next stage of project development.

Testing Design and Evidence

All testing design and evidence of tests can be found on our website at https://teamfarce.github.io/MIRCH/testing/ .