Software Testing Report

Summary of Testing Methods

We have adopted a test driven development approach to programming for the back--end part of the game, which involves writing the tests which will test the functionality of our software, before writing any code. Test-driven development revolves around the cycle of:

1. Writing a test
2. Writing the minimum amount of code required to pass that test
3. Refactoring the code to keep code clean and high in quality

Beck describes this process quite elegantly with the phrase “Red, Green, Refactor” [1]. This approach to software engineering supports the iterative agile development methods we are using to manage and complete the project, while also providing us with a suite of tests that we can run to check that nothing has broken when a new feature is implemented. We use JUnit to create our automated testing programs, as it is very easy to use and provides precise details about failing test cases.

For the front-end GUI, it was hard to create unit tests as checking that widgets of the user interface display in an aesthetically pleasing way isn’t really measurable by a quantity as such. Instead of using unit testing, it was a case of trial and error to make sure that the UI displayed and behaved as required. We generated short descriptions of actions that a player should be able to perform, and ensured that the player could complete these actions without an bugs or crashes.

We also tested our implementation against our requirements, to ensure that while we were implementing the game we were sticking to the requirements.

Report of actual tests

Unit Tests

Our unit tests were conducted using JUnit (http://junit.org/junit4), they can be found in the test gradle package (https://github.com/TeamFractal/Roboticon-Quest/tree/v1.0.1/test). The full plan and results of these tests can be found at http://teamfractal.github.io/assessment2/Unittests.pdf

We tested all of the features in the backend classes, and almost all of these tests succeeded. This is because we were using Test Driven Development for this part of the implementation, and so we were constantly ensuring that our code passed the unit tests as we wrote it.

The one Unit Test which does fail is 22, which cannot currently pass as the nextPhase() method it tests relies on an instantiated gameScreen, which is not currently the case as the GUI is not fully loaded.

GUI Tests

To ensure that the GUI behaved correctly and as expected, we created a list of test scenarios that describe actions that the player should be able to complete, how the GUI should behave when they are completed, and how it should behave when they are unable to be completed. We have so far only written tests for those features that we have implemented at this stage, as the GUI behaviour for unimplemented features is not yet fully defined. The full plan and results of these tests can be found at http://teamfractal.github.io/assessment2/GUITesting.pdf

All of these tests pass, so we are confident that our GUI, as implemented so far, behaves as expected. We believe this tests are complete because they cover all aspects of the GUI. We believe these tests give us the correct results because we ran them several times as we were developing the code.

Requirements Acceptance Tests

We regularly tested our implementation against all of our requirements during development to continuously ensure that our it was meeting the requirements. The latest results of these tests can be found at http://teamfractal.github.io/assessment2/RequirementsTesting.pdf

Some of these tests fail at the current implementation, but this is due to features which are yet to be implemented (this is noted by the relevant tests). Specifically, the requirements which are not yet implemented at all are 1.2.1, all of 3.x.x, 4.1.1, 4.2.1, 6.1.2, 7.1.4, 8.1.2, all of 9.x.x and all of 10.x.x. There is one requirement which has so far only been partially implemented (and so the acceptance test also currently fails), which is 8.1.1. To enable these tests to pass, the remaining features need to be fully implemented.

All the features that have been implemented so far pass the relevant acceptance tests, so we are confident that all these features conform to the requirements.

Bibliography

[1] K. Beck, Test-driven development: By example, 1st ed. Boston, MA: Addison-Wesley Educational Publishers, 2002.