Module	SEPR
Year	2019/20
Assessment	2
Team	Dalai Java
Members	Jack Kershaw, Max Lloyd, James Hau, Yuqing Gong, William Marr, Peter Clark.
Deliverable	Software Testing Report

Testing

The objective of testing our code is to ensure that our program is working correctly, both in ways which would result in major code defects and in ways which would result in minor 'bugs' which may not be immediately noticeable. We have held group meetings in which we have devised a strategy to test our program. We have assigned roles to each member of our team in respect to testing certain aspects of our code; Peter is in charge of writing the unit tests, Max is in charge of ensuring the tests provided the maximum possible code coverage and James is in charge of making sure any code refactoring did not affect the implemented unit tests. Jack and Yuqing have been assigned the responsibility of performing manual tests; that is, tests which did not require a unit test, whilst William is in charge of performing the system tests. We have assigned different tests to different people to prevent a single point of failure in case a member falls ill or leaves the course (see Risk R2.2).

We have made the decision to use Unit testing to test each of our classes in turn; more specifically, we made the decision to use JUnit. We made this decision as unit tests are in general very fast, and it is easy to control the passed parameters in order to produce accurate tests. Including Unit tests also makes regression testing in the future more manageable as we can simply run the tests that we have created again to check that code functionality remains the same. We wrote our JUnit tests through adapting the examples provided in *Pragmatic Unit Testing In Java 8 with JUnit* for our own code and for use with IntelliJ. [1] Some classes were dependent on objects which could not be simulated in JUnit; to test these, we mocked the relevant classes using Mockito [2] and PowerMock [3].

In order for our tests to be an accurate measurement of the performance of our game, they are required to cover each relevant point in our User Requirements, and to achieve the highest line coverage possible, as well as proving that the tests still function as expected on the boundaries of partitions [4]. For each Unit Test, we have identified the equivalence partitions which require testing and have created a specific test for each of these partitions, as well as creating boundary tests to ensure that boundary values function as expected.

We are aware that there are some risks associated with unit testing; for example, as unit testing only tests specific functions, there may be errors in the game related to how the functions are used as opposed to the correctness of the functions themselves (see Risks R1.7 and R1.8). To mitigate this, we will also carry out System Testing in order to test the appearance and functionality of the game as a whole, as well as carrying out manual tests for functionality in which a unit test is inappropriate. For our manual tests, we maintained a documentation-based approach, ensuring we recorded test cases for each manual test as well as each unit test [5].

Traceability Matrix

We have created a Traceability Matrix which cross-references our test cases and user requirements. This will help us to prove that our testing comprehensively covers our defined requirements, and will make the process of accommodating changing requirements easier. Note that requirements not relevant for Assessment 2 have been omitted from the matrix.

Requirement Identifiers	Reqs Tested	UR_UX	UR_DEVICE	UR_DEMO_MODE	UR_RESUME	UR_FORTRESSES	UR_FIRE-ENGINES	UR_SINGLE_PLAYER	UR_GAME_OVER	UR_COMPARE_SCORE	UR_WRITTEN_IN_JAVA	FR_CHANGE_GAME_MODES	FR_RESUME	FR_NUM_OF_ENGINES	FR_CONTROL_ENGINE	FR_ENGINE_SPEC_WATER	FR_ENGINE_SPEC_SPEED	FR_ENGINE_SPEC_RANGE	FR_ENGINE_SPEC_DELIVERY_RATE	FR_ENGINE_SPEC_DAMAGE	FR_ENGINE_MAINTENANCE	FR_CONTROL_ENGINE_MOVING	FR_ENGINE_DESTROYED	FR_ENGINE_IN_RANGE	FR_FORTRESS_SPEC_WATER	FR_FORTRESS_SPEC_RANGE	FR_FORTRESS_SPEC_HEALTH	FR_FORTRESS_SPEC_DAMAGE	FR_FORTRESS_DESTROYED	FR_FORTRESS_IN_RANGE	FR_GAME_WIN	FR_GAME_LOSE	FR_NOTICE_GAME_OVER	NFR_TIMING	NFR_OPERABILITY	NFR_GAME_TIME	NFR_DOCUMENTATION
Test Cases																																				
Tested Implicitly	16					X	X	X			X			X		X	X	X	X	X					X	X	X	X							X		X
Compare_Scores_Test	1									X																										
Demo_Mode_Test	2			X								X																								
Engine_Destroyed_Test	1																						X													
Engine_Repair_Test	1																				X															
Fortress_Destroyed_Test	1																													X						
Game_Length_Test	1																																		X	
Game_Over_Test	4								X																						X	X	X			
In_Range_Test	2																							X						X						
Move_Engine_Test	1														X							X														
Multiple_Devices_Test	1		X																																	
Pause_Screen_Test	2				X								X																							
Screen_Appearance_Test	1																																	X		
User_Experience_Test	1	X																																		

Test Cases

ID	Description	Related Requirement	Category	Author	Status
Compare_Scores_Test	Tests the functionality to compare a user's score with others	UR_COMPARE_SCORE	Manual	Jack	Fail
Demo_Mode_Test	Tests whether the demo mode functionality is present	UR_DEMO_MODE	Manual	Yuqing	Fail
Engine_Destroyed_Test	Tests that the engine is correctly destroyed when said function is called	FR_ENGINE_DESTROYED	Unit, Functional	Peter	Pass
Engine_Repair Test	Tests whether the Fire Engines are repaired appropriately by the Fire Station	FR_ENGINE_MAINTENANCE	Unit, Functional	Peter	Pass
Fortress_Destroyed_Test	Tests whether the Fortress is destroyed when health reaches 0	FR_FORTRESS_DESTROYED	Unit, Functional	Peter	Pass
Game_Length_Test	Tests whether the game does not take an unreasonable amount of time to complete	NFR_GAME_TIME	Manual	Yuqing	Pass
Game_Over_Test	Tests whether an appropriate 'Game Over' screen appears when the game has	FR_GAME_WIN, FR_GAME_LOSE,	Manual	Jack	Pass

	been won or lost	FR_NOTICE_GAME_OVER			
In_Range_Test	Tests that the 'InRange()' function works for all entities	FR_ENGINE_IN_RANGE, FR_FORTRESS_IN_RANGE	Unit, Functional	Peter	Pass
Move_Engine_Test	Test that the fire engine moves the correct distance when prompted	FR_CONTROL_ENGINE_MOVING	Unit, Functional	Peter	Pass
Pause_Screen_Test	Tests that the pause screen functions as it should	FR_RESUME	Unit,Functional	Peter	Pass
Runs_on_PC_Test	Tests that the game runs correctly on a PC	UR_DEVICE	Manual	Jack	Pass
Screen_Appearance_Test	Tests that the Game Over screen appears very quickly once the game has been completed	NFR_GAME_TIME	Manual	Yuqing	Pass
User_Experience Test	Tests that the overall system offers a pleasant user experience	UR_UX	System	Will	Fail

Statement of Failed Tests

Most of our unit, system and manual tests passed, however there were a small number of failed tests. In particular, the manual Compare_Score_Test and Demo Mode Test failed as neither of these features have been implemented. The System test carried out to determine whether the overall system offers a pleasant user experience also failed due to the features mentioned in the Implementation Report.

Website

Our testing design and Unit tests can be found here:
Unit Tests: https://baffledwhiskey.github.io/A2%20Documents/Unit%20Tests.zip
Test Results: https://baffledwhiskey.github.io/A2%20Documents/Test%20Results.pdf

Bibliography

[1] J. Langr, A. Hunt and D. Thomas, *Pragmatic unit testing in Java 8 with JUnit*. 2015.
[2] "Mockito (Mockito 3.2.4 API)", *Javadoc.io*. [Online]. Available: https://javadoc.io/static/org.mockito/mockito-core/3.2.4/org/mockito/Mockito.html. [Accessed: 11- Jan- 2020].

[3] "Using PowerMock with Mockito", *GitHub*, 2017. [Online]. Available: https://github.com/powermock/powermock/wiki/mockito. [Accessed: 13- Jan- 2020].
[4] M. Pezzè and M. Young, *Software Testing and Analysis: Process, Principles and Techniques*, 1st ed. 2007, Chapter 12.
[5] J. Itkonen, M. Mantyla and C. Lassenius, "How do testers do it? An exploratory study on manual testing practices", *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009. Available: 10.1109/esem.2009.5314240 [Accessed 17 January 2020].