Testing

Group 12

Keira Lauder
Amy Mason
Dan Ho
Sina Khosravi
Yibing Wang

a).

We used dynamic testing which directly executed a segment of our code to verify if the system met all of the requirements and specifications. The aim was to not have any flaws or design errors in the code, but if failures were identified, we needed to amend them. We tested the code in a controlled environment.

We prioritised the requirements that needed to be tested because we understand that exhaustive testing is not reasonable nor is it possible to do so. When we identified the priority test cases, we started to design them; the test cases would be appropriate for this project because they would be able to identify how ready the code is to be completed and therefore sent to our customer.

After our prioritisation, we planned how we were going to test these requirements:

- We selected our units of code that aligned with the (prioritised) requirement that we are testing.
- We discussed which of the testings would be beneficial to use for this requirement.
- We discussed where we were going to test the requirement (what type of environment)
- Elicit why we were testing it and what purpose it served for our project.
- We assigned a member of our team who will take responsibility for the writing cases.
- We tested the requirement module to see if it passed or failed our test.

One of the testings that we used was Unit Testing. We used this because it was easy to write and control and therefore we were able to test a unit of work in the system to make sure that it is working efficiently and correctly to achieve a purpose or goal. Functional or black-box testing was used so we could study the inputs and related outputs of the functionality [1]. However, Unit Testing was not an efficient piece of testing for all of our requirements because some bugs and errors cannot be detected and unit testing mainly is effective in procedural code.

Integration Testing was very beneficial in our testing because it tested modules of our code and helped us to understand how each module performed on its own as well as how it interacts with other modules.This is appropriate for our project because each test can identify a weakness within our project which can help us in our waterfall software development process.

In addition to the above testing, we have tested the system as a whole (Black Box testing) so we could have a user's perspective when testing. This was essential because due to our project having a GUI, we needed to be able to test how the game will run when the user will be playing it. Within our black box testing, we tested the validation of the functional requirements and the non-functional requirements to fully identify any weaknesses within the system. Also, white box testing would be beneficial to verify the flow of input-output and to improve the usability of the game. This was mainly done manually.

b).

<u>Purpose</u>
The purpose of this document is to provide various activities performed as part of the testing for the ‘York Pirates!” game we have developed. In this report, we have mentioned the testing scope, whether it be in scope, out of scope or simply that the items have not been tested.

<u>Application Overview</u>
‘York Pirates!’ is a pirate themed game where the user is to take over colleges of the University of York by sailing around the lake and shooting projectiles from their boat. The aim of the game is to be the last college remaining after taking over the rest whilst completing tasks along the way. The game as a whole has been tested in the following way.

<u>Testing</u>
This section is used to explain about the functions and modules that we have used for testing and mention whether they were in or out of scope. If items have not been tested, we will note any constraints, dependencies or restrictions why these have not been tested.

- In Scope
    - We have completed all the asset unit testing within our project.We wanted to make sure that all the assets that we have used in the game would be readily available to the user when they play the game.This mitigated part of risk 012 as we have reduced the risk of the loss of images and data, therefore saving the game from crashing.
    - We have done some functional testing in our classes that we have created for assessment 2.
    - We tested to see if when a projectile hits both the player’s or college that its object will lose health. This was important to test for the requirements UR.CPTR_CLG, FR.CLG_ATTACK and FR.CLG_HEALTH.
    - We tested the movement of the player to make sure that when the player moves on the screen, that they are moving in the correct direction.
    - Part of assessment 2 was to have different power ups available on the screen for the player to collect throughout the game. Requirements UR.POWER_UP and FR.DISPLAY_POWER_UP needed us to test these. In order to test these, we tested to see if the powerup capability had been called without the player’s hibox hitting the power up, which of course did not work, and when the user had actually collected the power up, then the power up method had been called. This was successful.
    - Checking whether the powerups would expire correctly was also done through Thread.sleep(), to check that after a set amount of time, the power ups would finish correctly.
    - Following on from the previous test, we wanted to show the user their health in a bar above their sprite. This would make it easier to see what type of

difficulty they are playing under or if they have had a power up to restore their health. We have tested to see that once damage has been taken, that the health bar will decrease in size. This test was successful.

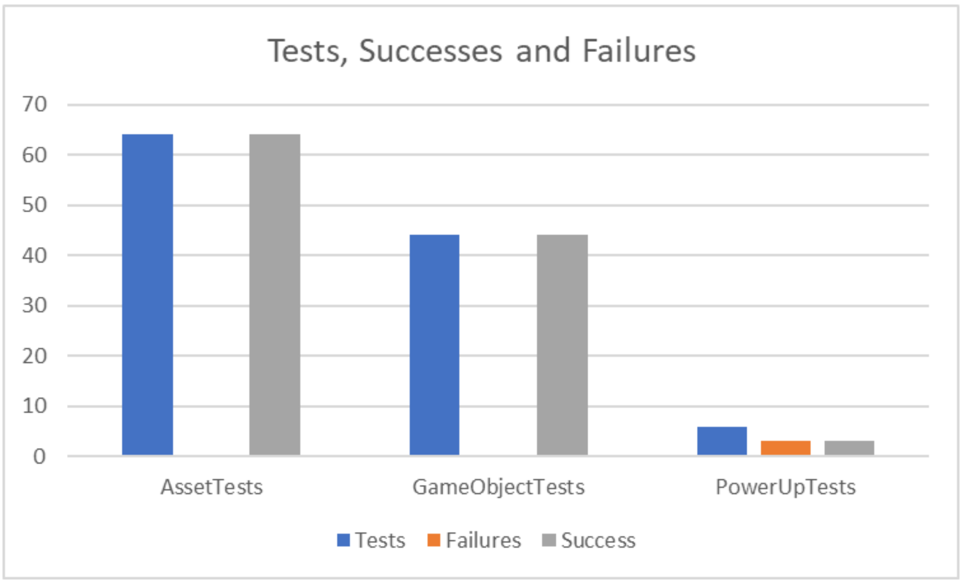
- To make sure that collision with objects was successful, we tested two new objects to see if our overlap() method would return true. This was successful. This test was important because it was key for projectiles to cause damage to other objects and for the player to pick up power ups, for example.

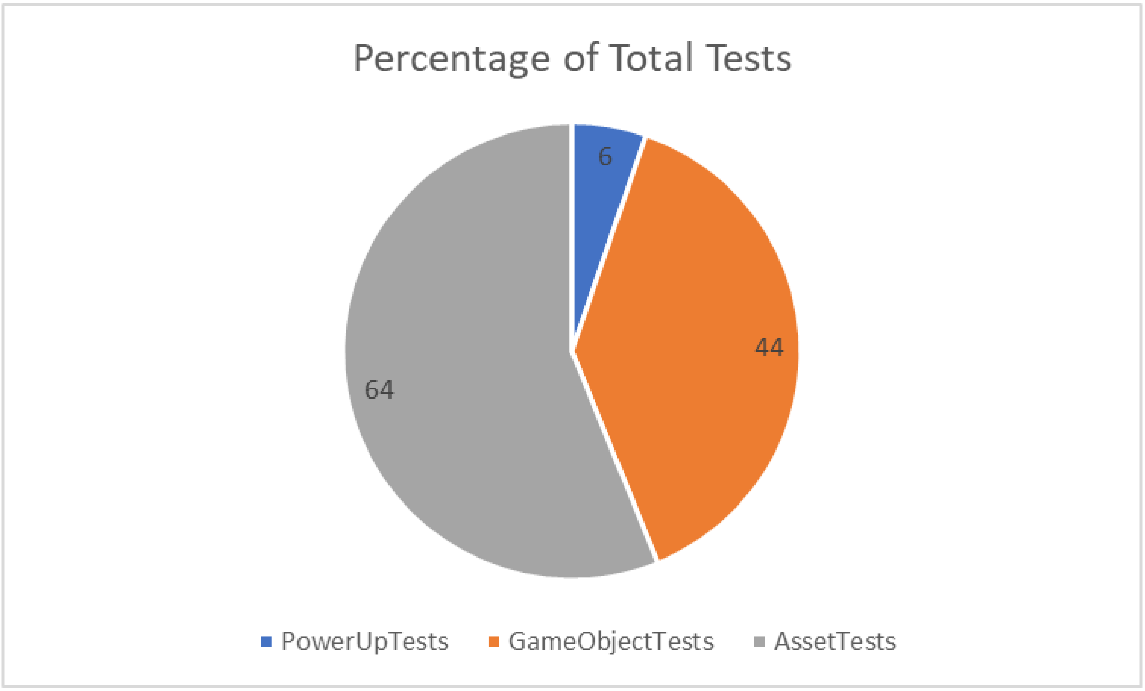
- Out of Scope
    - Testing for performance was done by us for the game. We ran the game ourselves to see if the game behaved to what we wanted it to. This made sure that the code was executing logically and correctly.
    - The UI was tested manually to make sure that everything was placed where we wanted it to be. Do to this, we also have a variable in our **YorkPirates** class called *DEBUG_ON* which is set to 'true' when we are testing, otherwise it is 'false'.
    - Graphics related to the game were also tested manually, to check whether the game looked the way we wanted it to.
- Items not Tested

Metrics

Here, we will display the test execution results along with graphs and charts for better visual representation. You will be able to see the testing summary on our website [2].

Class	Tests	Failures	Ignored	Duration	Success
AssetTests	64	0	0	20ms	100%
GameObjectTests	44	0	0	22ms	100%
PowerUpTests	6	3	0	55s 66ms	100%





Test Environment & Tools
We used Intellij as we found that it was able to run tests for us and create testing reports easier than other IDEs.

c) Test Results
Test Coverage Report

[1] I. Sommerville, *Software Engineering, Sixth Edition*. Harlow, England: Pearson Studium, 2001.

[2] “York Pirates! Test Reports” *York Pirates!* *https://keiral11.github.io/Team12Website/testreports/tests/test/index.html*