CS 4ZP6A: Test Report
Cat and Mouse Game

Team #8, ClawSome Games
Yuan Gao (1330064)
Su Gao (1330065)
James Lee (1318125)
James Zhu (1317457)

Sunday 26th March, 2017
revision 0

Revision History

Revision	Date	Author(s)	Description
0	22.01.04	YG, SG, JL, JZ	created the test report document

Contents

1 Introduction

1.1 Purpose of the Document

This document summarises the testing procedures, results, and analysis conducted on the **Cat-and-Mouse Game**. The approach and scope to the tests outlined in this report is as specified according to the accompanying **Test Plan**.

1.2 Scope of the Testing

The tests which are conducted aim to verify that the basic components and systems of the product function as expected, as well as ensure that the *interaction* between these various modules function as expected and defined within the **.**

Tests will consist of two types:

1. **Automated unit testing**, which consist of test functions performing some major task(s) within the Product, provided with a certain input, with the corresponding output being compared to an expected value. This will be implemented using a test framework (such as ***Unity Test Tools***). These tests will be carried out primarily through the performing a series of *pre-defined* actions given a set **pre-condition** and an expected **post-condition**. These system tests cover all the modules of the Product: *Character Behaviours*, *Skill Behaviours*, *Vitality Behaviours*, *Win Conditions*, *Physics*, *Enemy Behaviours*, *Menu/interface behaviour*

2. **Usability Tests**, which consist of a User performing a set sequence of Actions in-game, while certain metrics relating to the *usability* of the product is analysed using pre-defined *quantitative* measures.

1.3 Organisation

As described above, this test report will consist of:

1. **Section 1:** An Introduction to the overall aims, scope and methods utilised in the testing of this Product.

2. **Section 2:** A description of the states, behaviours, and results measured from the System Tests, when given, as described above, a set pre-condition, a set of actions or behaviours carried out manually to change the state of the Product, and the resulting state's conformity to one or more expected post-conditions. This includes a description of the states, functions and results of the Automatic Unit Test Cases carried out.

3. **Section 3:** A description of the states, behaviours, and results measured from the Usability Tests, described in more detail below.

1.4 Usability Testing

In addition, Usability Testing will be carried out in accordance with our **Test Plan**. These tests consist of subjective, non-functional and user-evaluated tests which measure the *user experience* of the Product as whole, against certain pre-defined values for various aspects of the Product. This will then provide a quantitative measurement of the *usability* of Product in relation to the scope defined above.

This will consist of the following tests:

1. **Appearance Test**: This will measure the *ease of use* for Users to navigate the Interface by *self-direction*, or without prior knowledge or experience with the Product.

2. **Spelling and Grammar Test**: This will measure the correctness of *spelling* and *grammar* with regards to the Textual Information which is presented to the User. This will include data such as Character Skill Data, on-screen prompts to the User regarding the availability of certain Actions when in a specified State, the information presented on the Multiplayer Lobby with regards to the current State of the Multiplayer Session, etc.

3. **Control Test**: This will measure the *time* and *effort* required by a User to orientate themselves and become comfortable with the in-game controls.

4. **Speed and Latency Test**: This will measure the Connection Quality when connected to the Multiplayer Server, in terms of average *ping* values and *bandwidth utilisation* over a set period of time. This will be measured as a User completes- a match within the game, as this will represent an overview of the sequence of Actions to be expected in production environments.

5. **Graphical Frame Rate and Latency Test**: This will measure the *frame-rate*, in Frames Per Second, and *graphical latency* in Mill-seconds between each rendered frame. This test will be performed under the similar conditions as specified in the **Speed and Latency Test**, described above.

6. **Fault Tolerance Test** This will measure the response provided to the User with regards to In-Game and Networking errors which may present itself during the normal operation of the product. This will include analysis of the *effectiveness* of the Error Messages and/or prompts which are provided in assisting the User to remedy the issue with minimal *time*, *effort* and *reference to external sources.*

7. **Operational System Support Test**: This will measure the range of support for various Hardware and Software Environments as defined in **The Design Document**. A fully-working copy of the Product will be installed on several test environments, and the ability of these environments

to carry out common sequences of Actions in-game (such as connecting to a Multiplayer Lobby, starting a new Match on the Multiplayer Server, and utilising Skills).

8. **Graphical API Support Test**: This will measure the support for industry-standard graphical Application Programming Interfaces (APIs), such as environments supporting *DirectX 9*, *DirectX 11* and *OpenGL*. This can be verified through the observation of any rendering failure and graphical artifacts which may present itself during the operation of the product.

2 Automatic Unit Testing

2.1 Overview

The **Unity Engine** and the associated **Unity Test Tools Asset Library** facilitates the design of a highly-automated test environment.

2.2 Methodology

Pre-Conditions and **Post-Conditions** may be specified and compared using *assert* specifications and invariants to verify adherence to a (range of) expected output values and states. The Unity Test Tools utilises a *model-based* testing approach, which allows for *Testing Scripts* to be attached to **GameObjects** and access any associated *behavioural scripts*. Functions can then be defined within the Testing Scripts, as well as the Unity Inspector itself, which then conducts a comparison of a the value of a specified variable within the behaviour script (such as *currentHealth*, for instance), to a set of expected values as defined in a post-condition.

2.3 Scope

The Scope of these Automatic Unit Tests should cover major Functions and Methods defined within each Module of the product, as well as any Interfaces present to allow for defined interactions **between** these Modules.

2.4 Execution of Test Cases

The execution of the Test cases will be through the *Run Tests* buttons provided within the Unit Tester Interface, a part of the Unity Test Tools in the Inspector Window of the Unity Integrated Development Environment.

2.5 Retrieval of Test Results

The results of the Unit Tests will be displayed through an Indicator (displaying a result of either *Pass*, or *Fail*) within the Unit Tester Interface, as described in the **Execution of Test Cases** section above.

2.6 Analysis of Test Results

The results of the Unit Tests will be analysed primarily based on the return of a *Pass* or *Fail* property, as the objective nature of Unit Testing does not allow for any *variance* in the actual result from the expected results, when given a certain set of Inputs. Should a test return *failure*, then, assuming that the **pre-condition** and the *input to change state* holds, then it can be deduced that the Action or Behaviour under observation is *not* operating as expected.

2.7 Movement Input Tests

2.7.1 Test 3.1.1.1 to Test 3.1.1.4: Walk in a direction, starts from stationary Test

void FixedUpdate()

Checks user input for necessary keys for movement, changes movement speed, attack power, checks if user can move.

Pre-Conditions: User must be in a game, after joining the lobby and selecting a room and have the game start.

Test Inputs: A keyboard so that the user can press keys, which FixedUpdate() will check for. For these tests, the keys 'w','a','s','d' will be checked by the function FixedUpdate().

Expected Results: If the user pressed 'w' on their keyboards for 1 second, the user controlled character will move forward from a stationary position for a second. If 's', user controlled character will move backwards from a stationary position for a second. If 'a', user controlled character will move to the left from a stationary position for a second. If 'd', user controlled character will move to the right from a stationary position for a second.

Actual Results: Passed. The character the user controls will move to a direction given by the user when the character starts from a stationary position.

Analysis of Performance: The *pass* result from this Test Case indicates given input from the user does indeed cause the character to move to the intended direction from a stationary state. This test case is not expected to involve any considerable performance measurements.

2.7.2 Test 3.1.1.5: Continuous movement in one direction, when the same key is held down

void FixedUpdate()

Checks user input for necessary keys for movement, changes movement speed, attack power, checks if user can move.

Pre-Conditions: User must be in a game, after joining the lobby and selecting a room and have the game start.

Test Inputs: A keyboard so that the user can press and hold keys, which FixedUpdate() will check for. For this test, the keys 'w','a','s','d' will be checked by the function FixedUpdate().

Expected Results: If the user holds down either 'a', 'b', 'c', 'd' on their keyboards, the character will continue moving in the direction of the key held, and will stop if the key is released.

Actual Results: **Passed**. The character controlled does move to the given input's direction until the button is released.

Analysis of Performance: The *pass* result from this Test Case indicates that the user is able to move their characters in a given input's direction for however long the button was held for. This test case is not expected to involve any considerable performance measurements.

2.7.3 Test 3.1.1.6: Jump, while standing still

void FixedUpdate()

Checks user input for necessary keys for movement, changes movement speed, attack power, checks if user can move.

Pre-Conditions: User must be in a game, after joining the lobby and selecting a room and have the game start.

Test Inputs: A keyboard so that the user can press keys, which FixedUpdate() will check for. For this test, the space key (when user presses the space button) will be checked by the function FixedUpdate().

Expected Results: If the user pressed the spacebar, the character will perform a jump action (so their Y axis velocity is increased, and then they drop back down to the ground after a while)

Actual Results: **Passed**. The character performed a jump action after user hit the spacebar key.

Analysis of Performance: Although the character does change they Y axis position during the jump action for a specified amount of time, certain tests have shown us that the character jumps higher than expected. These instances are rare, and as such the result is still satisfactory. This test case is not expected to involve any considerable performance measurements.

2.7.4 Test 3.1.1.7: Jump, while already in motion in a given direction

void FixedUpdate()

Checks user input for necessary keys for movement, changes movement speed, attack power, checks if user can move.

Pre-Conditions: User must be in a game, after joining the lobby and selecting a room and have the game start.

Test Inputs: A keyboard so that the user can press keys, which FixedUpdate() will check for. For this test, the space key (when user presses the space button) and directional keys ('w','a','s','d') will be checked by the function FixedUpdate().

Expected Results: If the user is currently moving their character in a direction using the keys 'w','a','s','d' (with the keys being held), and then presses the space key, the character should be performing a jump action towards the direction it was going before the jump key was hit.

Actual Results: **Passed**. The character jumps towards the direction they are going in.

Analysis of Performance: The *pass* result from this Test Case indicates that the user was able to have their character perform a jump action that is going in a certain direction. This test case is not expected to involve any considerable performance measurements.

2.7.5 Test 3.1.1.8: A change in direction

void FixedUpdate()

Checks user input for necessary keys for movement, changes movement speed, attack power, checks if user can move.

Pre-Conditions: User must be in a game, after joining the lobby and selecting a room and have the game start.

Test Inputs: A keyboard so that the user can press and hold keys, which FixedUpdate() will check for. For this test, the keys 'w','a','s','d' will be checked by the function FixedUpdate().

Expected Results: If the user is currently moving the character in one direction (button held), and then tries to move them in another, the the result would be that the character moves in the direction of the 2 buttons being held. For example if the forward button is held (key 'w'), and the left button is held (key 'a'), then the character will move diagonally to the left.

Actual Results: **Passed**. The intended behaviour takes place when 2 directional keys are held.

Analysis of Performance: The *pass* result from this Test Case indicates that the user is able to move their characters in a direction determined by the two buttons being held. This test case is not expected to involve any considerable performance measurements.

2.7.6 Test 3.1.1.9: Returning to a stationary position, when originally in motion

void FixedUpdate()

Checks user input for necessary keys for movement, changes movement speed, attack power, checks if user can move.

Pre-Conditions: User must be in a game, after joining the lobby and selecting a room and have the game start. Also, directional buttons must be held down as well (to indicate the character is in motion).

Test Inputs: A keyboard so that the user can press and hold keys, which FixedUpdate() will check for. For this test FixedUpdate will check for when no keys are being pressed.

Expected Results: If the user's character was currently in motion and then releases all/any key being held down, then the character will stop moving.

Actual Results: **Passed**. Characters will stop moving as soon as all/any key being held down are released.

Analysis of Performance: The *pass* result from this Test Case indicates that the user was able to stop their character from moving by releasing all/any keys being held down. When this was testing on an ice platform, the character will take a small amount of time to stop completely (as expected from moving on ice). This test case is not expected to involve any considerable performance measurements.

2.8 Camera Input Tests

2.8.1 Test 3.1.2.1 to Test 3.1.2.2: Camera rotation along the Y axis, counterclockwise/clockwise

void CamControls()

Checks for user mouse input and will keep track of X and Y axis changes when the mouse is being moved. Enables rotation of the camera depending on how the mouse is moved.

Pre-Conditions: User must be in a game, after joining the lobby and selecting a room and have the game start.

Test Inputs: A mouse that the user can move so that CamControls() can keep track of X and Y axis values.

Expected Results: If the user moves their mouse to the left for an inch, then the camera will look to the left (rotates 60 degrees to the left). If the user moves their mouse to the right for an inch, then the camera will look the the right (rotates 60 degrees to the right).

Actual Results: **Passed**. When the user moves their mouse to the left or right, the camera looks to the corresponding direction.

Analysis of Performance: The *pass* result from this Test Case indicates that the user was able to get the camera to look where they moved their mouse to. This test case is not expected to involve any considerable performance measurements.

2.8.2 Test 3.1.2.3 to Test 3.1.2.4: Camera rotation along the X axis, backwards/forwards

void CamControls()

Checks for user mouse input and will keep track of X and Y axis changes when the mouse is being moved. Enables rotation of the camera depending on how the mouse is moved.

Pre-Conditions: User must be in a game, after joining the lobby and selecting a room and have the game start.

Test Inputs: A mouse that the user can move so that CamControls() can keep track of X and Y axis values.

Expected Results: If the user moves their mouse down for an inch, then the camera will look down (rotates 60 degrees downwards). If the user moves their mouse up for an inch, then the camera will look up (rotates 60 degrees upwards).

Actual Results: **Passed**. When the user moves their mouse up or down, the camera looks to the corresponding direction.

Analysis of Performance: The *pass* result from this Test Case indicates that the user was able to get the camera to look where they moved their mouse to. This test case is not expected to involve any considerable performance measurements.

2.8.3 Test 3.1.2.5: Camera rotation across multiple axis at once

void CamControls()

Checks for user mouse input and will keep track of X and Y axis changes when the mouse is being moved. Enables rotation of the camera depending on how the mouse is moved.

Pre-Conditions: User must be in a game, after joining the lobby and selecting a room and have the game start.

Test Inputs: A mouse that the user can move so that CamControls() can keep track of X and Y axis values.

Expected Results: If the user moves their mouse 1/2 inch fowards and to the left, then 1/2 inch forwards and to the right, then 1/2 inch backwards and to the left, then finally 1/2 inch backwards to the right, then the camera should rotate 30 degrees counterclockwise along the Y axis, 30 degrees backwards along the X axis, then 30 degrees backwards along the X axis, then 30 degrees counterclockwise along the Y axis, then 30 degrees forwards along the X axis, then 30 degrees clockwise along the Y axis, and finally 30 degrees forwards along the X axis.

Actual Results: **Passed**. When the user moves their mouse to the given instruction, the camera looks to the corresponding direction.

Analysis of Performance: The *pass* result from this Test Case indicates that the user was able to get the camera to look where they moved their mouse to. This test case is not expected to involve any considerable performance measurements.

2.8.4 Test 3.1.2.6 to Test 3.1.2.7: Maximum camera rotation along the X axis, upwards/downwards

void CamControls()

Checks for user mouse input and will keep track of X and Y axis changes when the mouse is being moved. Enables rotation of the camera depending on how the mouse is moved.

Pre-Conditions: User must be in a game, after joining the lobby and selecting a room and have the game start.

Test Inputs: A mouse that the user can move so that CamControls() can keep track of X and Y axis values.

Expected Results: If the user moves their mouse upwards as much as they can, the camera will stop when the rotation degree is 90 upwards. If the user moves their mouse downwards as much as they can, then camera will stop rotation when the rotation is 90 degrees downwards

Actual Results: **Passed**. The camera will eventually stop following the user's mouse when moving up or down after max degree value is reached.

Analysis of Performance: The *pass* result from this Test Case indicates that the camera stopped when it had reached its bound when the user is looking up or down. This test case is not expected to involve any considerable performance measurements.

2.9 Map Tests

2.9.1 Test 3.1.3.1 to Test 3.1.3.4: Collision tests

void FixedUpdate()

Checks user input for necessary keys for movement, changes movement speed, attack power, checks if user can move.

Pre-Conditions: User must be in a game, after joining the lobby and selecting a room and have the game start.

Test Inputs: A keyboard so that the user can press and hold keys, which FixedUpdate() will check for. For these tests, user will have the move the character and run into different game objects.

Expected Results: For the collision test with the ground, when the user moves around, they should not fall below the ground. For the collision test with a wall, the character should not go through the wall and must be stopped by the wall. For the collision test with stationary objects, the character should stop moving once they hit said objects. For collision test with another player's character, the character should stop moving once they hit the second character.

Actual Results: **Passed**. The character successfully passed the first two test. The character did not stop once they hit a stationary object (such as a ball) or a player, but they did experience collision as they moved the object/player with them as they moved.

Analysis of Performance: The *pass* result from this Test Case indicates that the user's character can experience collision with different game objects in the game environment. This test case is not expected to involve any considerable performance measurements.

2.10 Map Tests

2.10.1 Test 3.1.3.5: Pick up and use a consumable

void FixedUpdate()

Checks to see if the user has collided with a consumable item.

Pre-Conditions: User must be in a game, after joining the lobby and selecting a room and have the game start.

Test Inputs: Mouse and keyboard input to move the character around.

Expected Results: Upon colliding with a consumable (powerup) the consumable item disappears and the corresponding effect occurs.

Actual Results: When the user collides with a consumable, the item disappears and the user immediately feels the corresponding effects of the item.

Analysis of Performance: The *pass* result from this Test Case indicates that the powerup system works as expected, and they will correctly give the player the correct bonuses upon use.

2.11 User Interface and Heads Up Display (HUD) Tests

2.11.1 Test 3.1.4.1: Minimap Accuracy Test

void Update()

Makes sure that the minimap at the side of the screen accuracy displays the area around the user.

Pre-Conditions: The user is currently playing a match.

Test Inputs: Mouse and keyboard input to move the character around.

Expected Results: The minimap follows the player's movements and correctly displays the area around the player.

Actual Results: The minimap follows the player's movements and correctly displays the area around the player.

Analysis of Performance: The *pass* result from this Test Case indicates that the minimap works as expected, and accurately represents the area around the user.

2.11.2 Test 3.1.4.2: Ability Use Test

void useSkill()

Upon a key press corresponding to an ability usage (Keyboard keys 1, 2, 3, 4), the corresponding ability is used if the ability is not on cooldown and has been learned.

Pre-Conditions: At least one ability has been learned and is currently off cooldown and ready to be used.

Test Inputs: Keyboard input.

Expected Results: Upon key press, the corresponding ability is used and then placed on cooldown. If the skill is a passive ability, then nothing happens.

Actual Results: When the user presses the skill key, the corresponding ability is used and then placed on cooldown. If the skill is a passive ability, nothing happens.

Analysis of Performance: The *pass* result from this Test Case indicates that the ability usage system works as expected.

2.11.3 Test 3.1.4.3: Ability on Cooldown Use Test

Checks user input, to see if the user pressed a button.

Pre-Conditions: The user is currently in a match, with at least one skill available, with that skill currently on cooldown.

Test Inputs: Keyboard input.

Expected Results: When the user presses the key corresponding to that skill, the skill will not be used and an indicator will show up indicating that the ability is still on cooldown and is not yet ready.

Actual Results: When the user presses the key corresponding to the skill, the skill is not used and an indicator shows up informing the player that the skill is still on cooldown.

Analysis of Performance: The *pass* result from this Test Case indicates that the ability cooldown code is working as expected.

2.11.4 Test 3.1.4.4: Menu Buttons Test

Checks user input, to see if the user pressed a button.

Pre-Conditions: The game is in a state where a button can be pressed.

Test Inputs: Mouse input.

Expected Results: Upon a button press, the corresponding correct action is performed.

Actual Results: **Passed**, When the user presses a button, the correct action is performed.

Analysis of Performance: The *pass* result from this Test Case indicates that the function indicated above works as expected, as well as the menu buttons working as expected.

2.11.5 Test 3.1.4.5: Options Menu Test - Changing the Screen Resolution

void drpScreenResolutionOnValueChanged (int index)

Updates the Screen Resolution option, on change of selection in the drop-down menu.

Pre-Conditions: setResolution is set to one of the supported supportedScreenResolutions indices.

Test Inputs: drpScreenResolutionOnValueChanged is passed in with index values from 0 to the highest index within the supportedScreenResolutions array.

Expected Results: setResolution is set to the Resolution value stored within the index value being tested within the supportedScreenResolutions array.

Actual Results: **Passed**, for each index value passed into the function.

Analysis of Performance: The *pass* result from this Test Case indicates that the function indicated above works as expected. This test case is not expected to involve any considerable performance measurements.

2.11.6 Test 3.1.4.5: Options Menu Test - Changing the Full-Screen Option

void tglFullscreenOnValueChanged (bool status)

Updates the selected Full Screen Option, on toggle of the specified Toggle Box object.

Pre-Conditions: setFullScreen is set to the current isOn status of the Toggle Box for Full Screen object, which specifies whether the game is operating in Full Screen Mode.

Test Inputs: tglFullScreenOnValueChanged is passed in with boolean values *true* and *false*, corresponding to the isOn status of the Toggle Box for Full Screen, the latter which is triggered manually between the On and Off positions.

Expected Results: setFullScreen is set to the status of the isOn attribute of the Toggle Box for Full Screen object

Actual Results: **Passed**. When the Toggle Box is unticked, the setFullScreen is set to either *true* or *false*, corresponding to the respective isOn status of the Toggle Box.

Analysis of Performance: The *pass* result from this Test Case indicates that the function indicated above works as expected. This test case is not expected to involve any considerable performance measurements.

2.11.7 Test 3.1.4.5: Options Menu Test - Changing the Graphics Quality Option

void drpGraphicsQualityOnValueChanged (int index)

Updates the selected Graphics Quality option, on change of selection in the drop-down menu.

Pre-Conditions: setGraphicsQuality is set to one of the supported supportedGraphicsQuality indices.

Test Inputs: drpGraphicsQualityOnValueChanged is passed in with index values from 0 to the highest index within the supportedGraphicsQuality array, corresponding to the drop-down selection selected within the Graphics Quality Drop Down Menu.

Expected Results: setGraphicsQuality is set to the Graphics Quality value stored within the index value being tested within the supportedGraphicsQuality array.

Actual Results: **Passed**, for each index value passed into the function.

Analysis of Performance: The *pass* result from this Test Case indicates that the function indicated above works as expected. This test case is not expected to involve any considerable performance measurements.

2.11.8 Test 3.1.4.5: Options Menu Test - Changing the Volume Slider Option

void slidVolumeLevelOnValueChanged (float silderValue)

Updates the selected Volume Level, on change of the value of the Volume Slider.

Pre-Conditions: setVolumeLevel is set to a value between 0 and 1.0, corresponding to the current Volume Level of the product.

Test Inputs: slidVolumeLevelOnValueChanged is passed in with floating-point values between 0 and 1.00, at increments of 0.05, to set the VolumeLevel. This corresponds to the value currently specified by the Volume Slider.

Expected Results: setVolumeLevel is set to the specified floating point value corresponding to the current value of the Volume Slider.

Actual Results: **Passed**, for each floating point value passed into the function.

Analysis of Performance: The *pass* result from this Test Case indicates that the function indicated above works as expected. This test case is not expected to involve any considerable performance measurements.

2.11.9 Test 3.1.4.5: Options Menu Test - Clicking the Apply Button

void btnPnlGameOptionsApplyClick

Applies any changes made to game options and opens the Main Menu.

Pre-Conditions: Each of the setResolution, setFullScreen, setGraphicsQuality, setVolumeLevel, setVolumeLevelMute is set to a value in its respective acceptable range, and the MainMenu Scene exists.

Test Inputs: Each of the setResolution, setFullScreen, setGraphicsQuality, setVolumeLevel, setVolumeLevelMute is set to a pre-defined value of: *1, false, 1, 0.50, false* respectively.

Expected Results: The game is set to: *the second-lowest available resolution, displayed in windowed mode, using the Faster graphic quality setting, with a Volume Level set to 0.5 on the Volume Slider, and the Volume not muted.*

Actual Results: **Passed**. The above values are present and states updated in the game correspondingly when the Apply Button is clicked.

Analysis of Performance: The *pass* result from this Test Case indicates that the function indicated above works as expected. This test case is not expected to involve any considerable performance measurements.

2.11.10 Test 3.1.4.6: New Game Menu Test - Creating a Game

void btnNewGameClick

New Game Button - Creates a new Game.

void switch CreateBtn

Creates a new Room and puts Player inside this Room.

Pre-Conditions: The user is currently in the MainMenu Scene.

Test Inputs: The btnNewGameClick function is executed. The user is then within the Multiplayer Lobby. The CreateBtn is clicked.

Expected Results: The lobby scene is loaded, and the MainMenu Scene is replaced by the lobby scene. The lobby scene corresponds to the Multiplayer Hub. The Player is then able to create a new Room within the Lobby, in which they are the solo player. The created Room should be visible to other Players connected to the Multiplayer Hub, upon selection of the RefreshBtn after the creation of the Room, and they should be able to join this Room.

Actual Results: **Passed**. When the btnNewGameClick function is executed, the Main Menu Scene is replaced with the lobby scene. Then, if the CreateBtn is clicked, a new Room is created and the Player is put inside it. Additionally, other Players are able to see and join the newly created Room from the Multiplayer Lobby, upon selection of the RefreshBtn after the creation of the Room.

Analysis of Performance: The *pass* result from this Test Case indicates that the function indicated above works as expected. This test case is not expected to involve any considerable performance measurements.

2.11.11 Test 3.1.4.7: Multiplayer Hub - Joining an Existing Room

void refresh

Displays all Rooms that have been created on the Multiplayer Server.

void JoinRoom(string RoomName)

Joins the selected Room on the Multiplayer Server.

Displays all Rooms that have been created on the Multiplayer Server.

void OnJoinedRoom

Lists the current client as a part of the specified Room.

Pre-Conditions: The player has an active Internet Connection to the Multiplayer Server (on Photon Cloud). There is currently one (or more) open Rooms on the Multiplayer Server.

Test Inputs: The roomName of the selected Room to join is retrieved from the GetRoomList function. Then the JoinRoom function is executed, passing in the roomName of the selected Room indicated.

Expected Results: The current Player is added to the list of Players within the selected Room, and the Room Information Screen is displayed, communicating to the Player the current Status (eg. other Players) of the Room and allowing them to select a corresponding (non-full) Team.

Actual Results: **Passed**.The correct roomName is passed as an argument into the JoinRoom function, and the Room Information Screen is displayed, communicating to the Player the current status of the Room and allowing them to select a Team.

Analysis of Performance: A specific measure of the performance of this Action is that the time to connect to a server should not exceed 20 seconds, and also that the connection to the Multiplayer Server should not be dropped at any time. Out of 10 iterations run, 8 iterations were able to connect to and join the specified Room successfully. The 2 failure cases may occur due to connectivity and bandwidth issues on either the side of the Client or the Server, in addition to the instability of the UDP protocol which is used. Thus, a pass rate of 80% , satisfactorily verifies the correctness of the functions which facilitates the joining of an Existing Room.

2.11.12 Test 3.1.4.8: Multiplayer Hub - Entering the Ready State

Verifies that the Player is able to enter a Ready State while within an existing Room.

Pre-Conditions: The player is in a Room with 3 other users, all users have not selected the Ready option yet.

Test Inputs: The player selects an Open Slot, through the pressing of a Button, underneath the respective Team they wish to join.

Expected Results: The Player is now in the Ready State for the selected Team. The Room Information Screen is updated to show the occupation of the Player within the selected Slot.

Actual Results: **Passed**. Upon selecting an Open Slot underneath the respective Team they wish to join, the Player is added to the list of Players assigned to the Team, and the Room Information Screen is updated to reflect the Player's Team selection.

Analysis of Performance: A specific measure of the performance of this Action is that the time from request to the update of the Room Information Screen to reflect the Player's Team Selection should not exceed 10 seconds, and also that the connection to the Multiplayer Server should not be dropped at any time. Out of 10 iterations run, 10 iterations were able to indicate and update the Room Information Screen to indicate the Player's Team selection successfully within the allocated time. Thus, a pass rate of 100% , satisfactorily verifies the correctness of the functions which facilitates the Player entering the Ready State within an existing Room.

2.11.13 Test 3.1.4.9: Multiplayer Hub - Starting a New Match when all Players in Room have entered the Ready State

Verifies that all Players connected to an existing Room, wherein everyone has entered the Ready State, may then enter a Match.

Pre-Conditions: The player is in a Room with 3 other users, all users are in the Ready State.

Test Inputs: A Player selects the Start Game button.

Expected Results: All the Players within the Room should be connected to the identical Game Match, and be spawned at on the Map at the designated location according to their respective Team.

Actual Results: **Passed**. Upon selection of the Start Game button, all Players are connected to the identical Game Match and spawned onto the Map.

Analysis of Performance: A specific measure of the performance of this Action is that the time from request of the Start of a new Match to the connection and spawning of each Player onto he Map should not exceed 20 seconds. Furthermore, the connection to the Multiplayer Server should not be dropped at any time. Out of 4 Players in the Ready State and connecting to a Game Match, 2 Players were able to connect successfully to the Match on the first attempt. These players which were not able to initially connect to the Match were able to connect on subsequent attempts (one on their 2nd attempt, and one on their 3rd attempt) Thus, a pass rate of 50% , is observed. Ideally, this should be raised to become a 100 % pass rate.

2.11.14 Test 3.1.4.10: Multiplayer Hub - Player Exiting a Match

Verifies that all Players connected to an active Game Match may at any time, leave this Match and return to the Multiplayer Hub of the current session.

Pre-Conditions: The Player is currently located within the Map Environment of an active Game Match on a Multiplayer Session.

Test Inputs: The Player selects the Exit Match button in the HUD with their mouse.

Expected Results: The Player is removed from the current Active Match and returned to the Multiplayer Hub, whilst still connected to current session on the Multiplayer Server.

Actual Results: **Passed**. Upon selected the Exit Match button, the Player is removed from the list of active players currently within the Match, de-registered from the corresponding Room, and taken back to the Multiplayer Hub.

Analysis of Performance: A specific measure of the performance of this Action is that the time from request of the Ending of a Match to the return to the Multiplayer Hub should not exceed 20 seconds. Furthermore, the connection to the Multiplayer Server should not be dropped at any time. Out of 4 Players within the Active Match and upon each selecting the Exit Match button, all 4 were able to leave the Match and return to the Multiplayer Hub within the allotted period of time. These Players were additionally able to remain connected to the Multiplayer Hub upon their successful exit from the Match. Thus, a pass rate of 100% , is observed, which satisfactorily verifies the correctness of the functionality provided to allow Players within a Active Match to leave from that Match and re-enter the Multiplayer Hub.

2.12 Networking Tests

2.12.1 Test 3.1.5.1: Networking - Joining a Room from the Multiplayer Hub

Verifies that all Players connected to the Multiplayer Hub is able to see a list of the currently-existing Rooms, and upon selection of one of these Rooms, is able to connect to and be seen by other Players within the Room on the Room Information Screen.

Pre-Conditions: The Player is currently connected to the Multiplayer Hub but has not yet created a Room nor selected an existing Room to join.

Test Inputs: The Player selects an available Room within the Room Browser Interface of the Multiplayer Hub.

Expected Results: The Player is added to the list of Players which are currently a part of the selected Room. The Player is then able to see the Room Information Screen for the selected Room, in addition to other Player already in the Room able to see the newly-added Player.

Actual Results: **Passed**. Upon the selection of an available Room within the Room Browser Interface, the Player is able to connect to the specified Room, see the statuses of the other Players currently connected to that Room, as well as update their own status within the Room (eg. by joining a Team).

Analysis of Performance: A specific measure of the performance of this Action is that the time from request of the joining of a Room to the display of the Room Information Screen should not exceed 10 seconds. Furthermore, the connection to the Multiplayer Server should not be dropped at any time. Out of 4 Players all selecting the same room in the Room Browser Interface, all 4 were able to connect to and see the Room Information Screen for the specified Room within the allotted period of time. Thus, a pass rate of 100% , is observed, which satisfactorily verifies the correctness of the functionality provided to allow Players connected to the Multiplayer Hub on the Multiplayer Server to join an existing Room as listed in the Room Browser.

2.12.2 Test 3.1.5.2: Networking - Leaving a Room and returning to the Multiplayer Hub

Verifies that a Player connected to an active Room are able to disconnect from that Room and return to the Multiplayer Hub.

Pre-Conditions: The Player is currently connected to a Room on the Multiplayer Server.

Test Inputs: The Player selects the Leave Session button within that Room.

Expected Results: The Player is de-registered from the list of Players currently connected to the specified Room, and is returned to the Multiplayer Hub, where they can choose to join another Room.

Actual Results: **Passed**. Upon the selection of the Leave Session button, the Player is removed from the list of Players currently connected to the Room, is disconnected from the Room, and then is returned to the Multiplayer Hub. This Player can then join another Room.

Analysis of Performance: A specific measure of the performance of this Action is that the time from request of the leaving of a Room to the display of the Multiplayer Hub should not exceed 10 seconds. Furthermore, the connection to the Multiplayer Server should not be dropped at any time. Out of 4 Players selecting the Leave Session button whilst within the same Room all 4 were able to return to the Multiplayer Hub within the allotted period of time. Thus, a pass rate of 100% , is observed, which satisfactorily verifies the correctness of the functionality provided to allow Players connected to a Room to disconnect from the Room and return to the Mutiplayer Hub.

3 Usability Tests

3.1 Appearance Test

3.1.1 Test 3.2.1.1: Appearance Test

Pre-Conditions: The user currently has the game running in the main menu screen, and has not been exposed to the game before.

Test Inputs: Mouse, Keyboard to navigate through the interface.

Expected Results: The user has no problem navigating through the menus to a started match.

Actual Results: **Passed**. With some reading and a small amount of time, the user is able to successfully navigate from the start menu screen to actually playing a game.

Analysis of Performance: The *pass* result from this Test Case indicates that the user is able to successfully navigate through the game menus upon being freshly exposed to the game.

3.2 Spelling and Grammar Test

3.2.1 Test 3.2.2.1: Spelling and Grammar Test

Pre-Conditions: The user has not read any of the text in the game before.

Test Inputs: Mouse, Keyboard to navigate through the game.

Expected Results: The user encounters no spelling or grammar mistakes, and understands the meaning intended from every sentence.

Actual Results: **Passed**. With some reading and a small amount of time, the user understands what each button/skill is meant to do and also encounters no spelling errors.

Analysis of Performance: The *pass* result from this Test Case indicates that the user is able to understand the intended meaning from all text in the game, while also not encountering any spelling or grammar mistakes.

3.3 Control Test

3.3.1 Test 3.2.2.2: Control Test

Pre-Conditions: The user currently has the game running in a match.

Test Inputs: Mouse, Keyboard to play the game.

Expected Results: The users understand how to control their characters as well as how to win the game.

Actual Results: **Passed**. If the user read the instruction manual, then they are able to play the game and understand the win conditions and the controls very quickly. If the user skipped reading the instructions, then they were still able to play and understand the game after a few minutes.

Analysis of Performance: The *pass* result from this Test Case indicates that we should try to enforce reading the instructions before the user can play a game (through a tutorial or help tips in loading screens). However, even if this is not done the users will still be able to learn how the game works by themselves.

3.4 Speed and Latency Test

3.4.1 Test 3.2.3.1: Speed and Latency Test

Pre-Conditions: The user currently has the game running in a match.

Test Inputs: Mouse, Keyboard to play the game, internet speed of at least 1 Mbps upload and download.

Expected Results: The user has a measured ping of no more than 100ms.

Actual Results: **Passed**. With the necessary internet connections, users have an average low ping rate of 30 ms.

Analysis of Performance: The *pass* result from this Test Case indicates that the user is able to play the game with their controls feeling very responsive and with no visible "lag."

3.5 Graphical Frame Rate and Latency Test

3.5.1 Test 3.2.3.2: Graphical Frame Rate and Latency Test

Pre-Conditions: The user currently has the game running in a match.

Test Inputs: Mouse, Keyboard to play the game, computer with hardware of a suitable profile to run the game.

Expected Results: The user runs the game with an average frame-rate that never drops below 30 frames per second, and 99th percentile of rendered frames have a latency of 25ms or less.

Actual Results: **Passed**. With the necessary hardware, users experience an average frame-rate that never drops below 30 frames per second, and all frames are rendered with a latency of 25ms or less.

Analysis of Performance: The *pass* result from this Test Case indicates that the user is able to play the game with very smooth visuals.

3.6 Fault Tolerance Test

3.6.1 Test 3.2.3.3: Fault Tolerance Test

Pre-Conditions: The user currently has the game running in a match with other users.

Test Inputs: Mouse, Keyboard to play the game, internet speed of at least 1 Mbps upload and download.

Expected Results: If a user has internet connectivity problems, an error message appears which explains why they were disconnected. All other players with no internet connectivity issues stay in the game with no connectivity problems.

Actual Results: **Passed**. When a user loses their internet connection, they are notified, and all other players can continue playing the game with no issues.

Analysis of Performance: The *pass* result from this Test Case indicates that one user disconnecting will not cause the entire game to crash, and that other players can still continue playing the game.

3.7 Operational System Support Test

3.7.1 Test 3.2.4.1: Operational System Test

Pre-Conditions: The executable file has not yet been run on every operating system that we want to run it on.

Test Inputs: Computer running on Windows 7, 8 and 10. Mouse and Keyboard to open and play the game.

Expected Results: The game should run with no issues on all the above operating systems, provided that the necessary hardware and software is included.

Actual Results: **Passed**. The game runs without problems on all of the listed operating systems.

Analysis of Performance: The *pass* result from this Test Case indicates that the game can be played on all of the above operating systems.

3.8 Graphical API Support Test

3.8.1 Test 3.2.4.2: Graphics API Test

Pre-Conditions: The game has not yet been tested with DirectX9 and DirectX11 graphics APIs.

Test Inputs: Computers that have DirectX9 and DirectX11 graphics APIs. Mouse and keyboard to open and play the game.

Expected Results: The game can be run and played with no issues.

Actual Results: **Passed**. The game runs and can be played with no issues.

Analysis of Performance: The *pass* result from this Test Case indicates that the game can be run with no issues using the DirectX9 and DirectX11 graphics APIs.