CS 4ZP6A: Test Plan
Cat and Mouse Game

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

November 2, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
11/2/2016	1.0	Test Plan 0

1 General Information

1.1 Purpose

The purpose of the Test Plan is to detail the testing process of the project. What to test, how to test each item, and when to test is discussed in detail below.

1.2 Scope

The first test plan will be to test the basic components of the game, making sure that key elements such as multiplayer are functional, as well as basic design concepts such as map generation, terrain, and physics. The first test plan will not involve "game balancing" concepts or detailed graphical testing.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
API	API stands for Application Programming Interfaces.
PUN	PUN stands for Photon Unity Networking

Table 3: **Table of Definitions**

Term	**Definition**
Cat	One of the character the user can choose to play as in the game
Mouse	The other character the user can choose to play as in the game

Table 4: List of constants

Constant	Value	Description
v	1.0	Character base walk speed
β	Variable	Character movement speed bonuses, total movement speed = $v \times \beta$
g	9.8	Game gravity, accelerates the player downwards at 9.8 units per second if they are not on the ground
j	10.0	Character jump force
c	60.0	Camera sensitivity (degrees of rotation per inch of mouse movement)

1.4 List of Constants and Variables

2 Plan

2.1 Software Description

The game will be developed using the Unity game engine, using the Photon Unity Networking framework for enabling multiplayer design. C# will be used as our main coding language.

2.2 Test Team

The project designers will supervise the testing process, acting as the leaders for the test team. Individuals outside of the development team will also be asked to test the game and provide insight on the basic testing elements which we are targeting.

2.3 Automated Testing Approach

Some automated testing can also be done, through creation of certain functions in the code. For example, if we wished to test our map creation algorithm, we could write a function to automatically create 100 or more different maps and test to see if each one matched our requirements.

2.4 Testing Tools

We will use Unity as our primary testing tool, utilising both members of the development team as well as outside testers to review the code and game to make sure everything is functioning as required. We will also need at least two computers in order to test the networking and multiplayer capabilities, as well as a facility with Internet access (such as a classroom).

2.5 Testing Schedule

Table 5: **Testing Schedule**

Test Date	**Testers**	**Objective**
11/12/2016	Creators of the game	Completion of a demo of the game.
11/13/2016	Creators of the game	Testing of all unit test cases.
11/18/2016	Player Testers	Testing of our proof of concept demo game.
11/20/2016	Creators of the game and Player Testers	Load Test.
11/25/2016	Creators of the game	Demonstration of proof of concept demo.

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Movement Input Tests

Test 3.1.1.1:	**Walk forward, starts from stationary**
Description:	Tests to see if the player walks forward when the corresponding forward key is pressed, when the player is initially standing still
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0
Input:	Keyboard button (W key or up arrow key) being pressed and held down for 1 second, then released
Output:	Player velocity
Pass:	Player moves forward with a velocity of $v + \beta$

Test 3.1.1.2:	**Walk to the left, starts from stationary**
Description:	Tests to see if the player walks sideways to the left when the corresponding left key is pressed, when the player is initially standing still
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0
Input:	Keyboard button (A key or left arrow key) being pressed and held down for 1 second, then released
Output:	Player velocity
Pass:	Player moves to the left with a velocity of $v + \beta$

Test 3.1.1.3:	**Walk to the right, starts from stationary**
Description:	Tests to see if the player walks sideways to the right when the corresponding right key is pressed, when the player is initially standing still
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0
Input:	Keyboard button (D key or right arrow key) being pressed and held down for 1 second, then released
Output:	Player velocity
Pass:	Player moves to the right with a velocity of $v + \beta$

Test 3.1.1.4:	**Walk backwards, starts from stationary**
Description:	Tests to see if the player walks backwards when the corresponding backwards key is pressed, when the player is initially standing still
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0
Input:	Keyboard button (S key or down arrow key) being pressed and held down for 1 second, then released
Output:	Player velocity
Pass:	Player moves backwards with a velocity of $v + \beta$

Test 3.1.1.5:	**Continuous movement in one direction, when the same key is held down**
Description:	Tests to see if the player continues walking in a given direction when the corresponding key is pressed and held down, when the player is already walking in a given direction. For example, when the forwards key is held down when the player is already moving forwards, when the right key is held down when the player is already moving to the right, etc.
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of $v + \beta$
Input:	Keyboard button being pressed and held down for 3 seconds, then released
Output:	Player velocity
Pass:	Player continues moving in the given direction with a velocity of $v + \beta$

Test 3.1.1.6:	**Jump, while standing still**
Description:	Tests to see if the player performs a jumping action, changing their velocity in the Y axis
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0
Input:	Keyboard button for jumping (space) being pressed and released immediately
Output:	Player velocity
Pass:	Player's Y velocity increases by j, and is then accelerated by $-g$ every second until the player reaches the ground again, where they first started

Test 3.1.1.7:	**Jump, while already in motion in a given direction**
Description:	Tests to see if the player performs a jumping action, changing their velocity in the Y axis, while already moving in a given direction
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of $v + \beta$ in a given direction
Input:	Keyboard button for a directional movement is being pressed and held down, then the keyboard button for jumping (space) is pressed and released immediately
Output:	Player velocity
Pass:	Player's velocity in the given direction would not change, staying at $v + \beta$, but the player's Y velocity increases by j, and is then accelerated by $-g$ every second until the player reaches the ground again.

Test 3.1.1.8:	**A change in direction**
Description:	Tests to see if the player will change directions when a different key is also pressed and held down than what was held down initially
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of $v + \beta$ in a given direction
Input:	Keyboard button being pressed and held down for 3 seconds, then a different directional button being pressed and held down as well
Output:	Player velocity
Pass:	Player moves in the vector addition of the two directions corresponding to the keys pressed, with a velocity of $v + \beta$

Test 3.1.1.9:	**Returning to a stationary position, when originally in motion**
Description:	Tests to see if the player will stop moving when all keys are released
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of $v + \beta$ in any given direction
Input:	Any keyboard button being pressed and held down for 3 seconds, then all buttons are released
Output:	Player velocity
Pass:	Player velocity becomes 0 and stops moving

3.1.2 Camera Input Tests

Test 3.1.2.1:	**Camera rotation along the Y axis, counterclockwise**
Description:	Tests to see if the game camera will rotate in a counterclockwise direction when the mouse is moved to the left
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0, and camera positioned at a default of 0 rotation along both the X and Y axis
Input:	The mouse is moved to the left by 1 inch
Output:	Camera rotation
Pass:	Camera rotates 60 degrees counterclockwise along the Y axis

Test 3.1.2.2:	**Camera rotation along the Y axis, clockwise**
Description:	Tests to see if the game camera will rotate in a clockwise direction when the mouse is moved to the right
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0, and camera positioned at a default of 0 rotation along both the X and Y axis
Input:	The mouse is moved to the right by 1 inch
Output:	Camera rotation
Pass:	Camera rotates 60 degrees clockwise along the Y axis

Test 3.1.2.3:	**Camera rotation along the X axis, backwards**
Description:	Tests to see if the game camera will rotate in a backwards direction when the mouse is moved upwards
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0, and camera positioned at a default of 0 rotation along both the X and Y axis
Input:	The mouse is moved forward by 1 inch
Output:	Camera rotation
Pass:	Camera rotates 60 degrees backwards along the X axis

Test 3.1.2.4:	**Camera rotation along the X axis, forwards**
Description:	Tests to see if the game camera will rotate in a forward direction when the mouse is moved downwards
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0, and camera positioned at a default of 0 rotation along both the X and Y axis
Input:	The mouse is moved backward by 1 inch
Output:	Camera rotation
Pass:	Camera rotates 60 degrees forward along the X axis

Test 3.1.2.5:	**Camera rotation across multiple axis at once**
Description:	Tests to see if the game camera will rotate in the correct fashion when the mouse is moved in a non basic direction such as to the top left, top right, bottom left, and bottom right etc.
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0, and camera positioned at a default of 0 rotation along both the X and Y axis
Input:	The mouse is first moved 1/2 inch forwards and to the left, then 1/2 inch forwards and to the right, then 1/2 inch backwards and to the left, then 1/2 inch backwards and to the right
Output:	Camera rotation
Pass:	Camera rotates 30 degrees counterclockwise along the Y axis and 30 degrees backwards along the X axis, then camera rotates 30 degrees clockwise along the Y axis and 30 degrees backwards along the X axis, then camera rotates 30 degrees counterclockwise along the Y axis and 30 degrees forwards along the X axis, then camera rotates 30 degrees clockwise along the Y axis and 30 degrees forwards along the X axis.

Test 3.1.2.6:	**Maximum camera rotation along the X axis, upwards**
Description:	Tests to see if the game camera will stop rotating once maximum rotation is achieved at 90 degrees upwards
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0, and camera positioned at a default of 0 rotation along both the X and Y axis
Input:	The mouse is moved forward by 2 inches
Output:	Camera rotation
Pass:	Camera rotates 90 degrees backwards along the X axis, and then stops

Test 3.1.2.7:	**Maximum camera rotation along the X axis, downwards**
Description:	Tests to see if the game camera will stop rotating once maximum rotation is achieved at 90 degrees downwards
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0, and camera positioned at a default of 0 rotation along both the X and Y axis
Input:	The mouse is moved backwards by 2 inches
Output:	Camera rotation
Pass:	Camera rotates 90 degrees forwards along the X axis, and then stops

3.1.3 Map Tests

Test 3.1.3.1:	**Collision with the ground**
Description:	Tests to see if the player experiences collision with the ground when they perform movement actions in the maze
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0, and camera positioned at a default of 0 rotation along both the X and Y axis
Input:	Random direction keys are pressed and then released
Output:	Player velocity
Pass:	Player velocity changes in the appropriate directions, and the player never ends up with a Y position lower than the ground of the map

Test 3.1.3.2:	**Collision with a wall object**
Description:	Tests to see if the player experiences collision with wall objects that they encounter within the map environment
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0, and camera positioned at a default of 0 rotation along both the X and Y axis, with a wall directly in front of the player
Input:	The forward keyboard key is pressed and held down for 1 second
Output:	Player velocity
Pass:	Player moves forwards with a velocity of $v + \beta$, then upon reaching the wall the player's velocity becomes 0 and they stop moving

Test 3.1.3.3:	**Collision with a stationary object**
Description:	Tests to see if the player experiences collision with stationary, non-wall objects that they encounter within the map environment
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0, and camera positioned at a default of 0 rotation along both the X and Y axis, with an non-wall stationary object (such as a tree) directly in front of the player
Input:	The forward keyboard key is pressed and held down for 1 second
Output:	Player velocity
Pass:	Player moves forwards with a velocity of $v + \beta$, then upon reaching the object the player's velocity becomes 0 and the latter stops moving

Test 3.1.3.4:	**Collision with an in-motion player within the map environment**
Description:	Tests to see if the player experiences collision with an in-motion non-player character within the game environment
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the current player (Player 1) currently having a velocity of 0, and camera positioned at a default of 0 rotation along both the X and Y axis, with another player (Player 2) currently in the map environment on another computer having a velocity of 0. Player 2 is located behind Player 1
Input:	The forward keyboard key is pressed and held down for 1 second
Output:	Player velocity
Pass: Player 1 is stationary	Player 2 moves forwards with a velocity of $v + \beta$, then upon reaching Player 1 the Player 2's velocity becomes 0 and the latter stops moving

Test 3.1.3.5:	**Pick up and use a consumerable**
Description:	Tests to see if the player is able to pick up a consumerable (such as a potion) in the map and and utilise it immediately to boost their attributes
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0, and camera positioned at a default of 0 rotation along both the X and Y axis, with a consumerable object (such as a health potion) directly in front of the player
Input:	The forward keyboard key is pressed and held down for 1 second
Output:	Player velocity & Game state
Pass:	Player moves forwards with a velocity of $v + \beta$, then upon reaching the object the effects associated with that consumerable are applied to the attributes of the player

3.1.4 User Interface and Heads Up Display (HUD) Tests

Test 3.1.4.1:	**Minimap Accuracy Test**
Description:	Tests to see if the minimap at the side of the screen accurately depicts the area around the user
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0, and camera positioned at a default of 0 rotation along both the X and Y axis
Input:	The player presses keyboard direction keys to move around in the maze
Output:	Minimap display
Pass:	The player is able to use the minimap to navigate the maze correctly without getting stuck

Test 3.1.4.2:	**Ability Use Test**
Description:	Tests to see if the abilities the player has can be used appropriately
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0, and camera positioned at a default of 0 rotation along both the X and Y axis
Input:	The player presses a keyboard key corresponding to an ability
Output:	Game state
Pass:	The player uses the corresponding ability, and the ability is put on a cooldown

Test 3.1.4.3:	**Ability on Cooldown Use Test**
Description:	Tests to see if the game allows use of abilities currently on cooldown
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	Custom in game state with the player currently having a velocity of 0, and camera positioned at a default of 0 rotation along both the X and Y axis, and all player abilities on cooldown
Input:	The player presses a keyboard key corresponding to an ability
Output:	Game state
Pass:	The ability is not cast, and a notification is displayed indicating that the ability is not yet ready to be used

Test 3.1.4.4:	**Menu Buttons Test**
Description:	Tests to see if the buttons in game menus perform their required function
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	The game is at the main menu
Input:	The player uses the mouse to select and press a button
Output:	Game menu state
Pass:	The corresponding game menu state is achieved: if new game was pressed, then the player is sent to the New Game Menu, if options was pressed, then the user is sent to an options screen, and if exit game was pressed, the game terminates

Test 3.1.4.5:	**Options Menu Test**
Description:	Tests to see if the sliders and functions in the Options menu perform their required purpose
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	The game is in the options menu
Input:	The player uses the mouse to select and press a button, or adjust a slider
Output:	Game option menu state
Pass:	The corresponding game menu state is achieved: if a different graphical setting was chosen, then that graphical setting is achieved (low, med, high), if the Audio slider was adjusted, then the audio levels adjust accordingly, if the user changed their keyboard hotkeys, then the changes are saved and used

Test 3.1.4.6:	**New Game Menu Test: Creating a game**
Description:	Tests to see if the user can create a new multiplayer session
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	The game is in the New Game Menu
Input:	The player presses the create new game button with the mouse
Output:	Game menu state
Pass:	The user is sent to the Multiplayer Multiplayer Hub, where they create and join a new Room, and are the only person inside that Room.

Test 3.1.4.7:	**New Game Menu Test: Joining a game**
Description:	Tests to see if the user can join an existing multiplayer session
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	The game is in the New Game Menu
Input:	The player presses the join game button with the mouse
Output:	Game menu state
Pass:	The user is sent to the Multiplayer Multiplayer Hub, where they join an already existing Room, and are not the only player in that Room. Each user can see the status of each other user in that Room.

Test 3.1.4.8:	**Entering the Ready State**
Description:	Tests to see if the users can enter a ready state while in an existing game Room
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	The player is in a Room with 3 other users, all users have not selected the Ready option yet
Input:	The player selects and presses the Ready button with the mouse
Output:	Game menu state
Pass:	The user enters the Ready state, and all other players in the same Room are able to see that the player is in the Ready state

Test 3.1.4.9:	**Starting a match when all players are Ready**
Description:	Tests to see if a new match can be started once all players are in the Ready state
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	The player is in a multiplayer session with 3 other users, all users have selected the Ready option and are in the Ready state
Input:	The player selects and presses the Begin Match button with the mouse
Output:	Game menu state
Pass:	The user as well as all other players in the same session enters the match, where a map is generated, each player is sent to their respective spawn location within the environment, and the match begins

Test 3.1.4.10:	**Player exiting a match**
Description:	Tests to see if a player can exit an active match and return to the Multiplayer Multiplayer Hub of the session
Type:	Dynamic Unit Test
Tester(s):	The creators of the game
Initial State:	The player is currently within the map environment of an active match on a multiplayer session
Input:	The player selects and presses the Exit Game button with the mouse
Output:	Game menu state
Pass:	The Player is taken out of the active match and is returned to the Multiplayer Hub for the session.

3.1.5 Networking Test

Test 3.1.5.1:	**Joining Multiplayer Hub Test**
Description:	Test to see if a user can join a Multiplayer Hub/server
Type:	Dynamic Unit Test
Tester(s):	The creators of the game and Game Testers
Initial State:	The user is in game menu but not in a Multiplayer Hub/server yet.
Input:	The player selects and presses the join server button with the mouse
Output:	Game menu state
Pass:	The user should now be in a menu asking them to choose between Cat or Mouse. They should also be able to see user info such as ping and IP address at the corner of their screen.

Test 3.1.5.2:	**Leaving Multiplayer Hub Test**
Description:	Test to see if user can successfully leave a server without errors
Type:	Dynamic Unit Test
Tester(s):	The creators of the game and Game Testers
Initial State:	The player is in a server
Input:	The player selects and presses the Leave Session button with the mouse.
Output:	Game menu state
Pass:	User should be back in the multiplayer server list menu. No errors should be displayed or occur.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel Test

Test 3.2.1.1:	**Appearance Test**
Description:	How easy it is for users to navigate the interface upon looking at it.
Type:	Static Unit Test
Tester(s):	Game Testers
Initial State:	User has not seen the interface for the game yet.
Input:	Mouse, Keyboard to navigate through the interface.
Output:	Game Menu State
Pass:	Game tester has no problem navigating the interface.
How test will be performed:	User will open the game, and will use their mouse and keyboard to navigate through the menu to access the game and Multiplayer Hub. The user must then look at the interface for both the Cat and the Mouse characters.

3.2.2 Usability Test

Test 3.2.2.1:	**Spelling and Grammar Test**
Description:	Check for correct spelling and grammar.
Type:	Static Unit Test
Tester(s):	Game Testers
Initial State:	User has not read any text in our game.
Input:	Mouse, Keyboard to navigate through the game.
Output:	Game Menu State
Pass:	Every piece of text must have perfect spelling and grammar.
How test will be performed:	The user will navigate every interface in the game and check for any spelling and grammar mistakes. If any mistakes are found, the user will write down where they found the mistake and if it is a spelling or grammar mistake.

Test 3.2.2.2:	**Control Test**
Description:	How easy it is for users to figure out controls.
Type:	Static Unit Test
Tester(s):	Game Testers
Initial State:	Users have entered a game of Cat and Mouse.
Input:	Mouse, and Keyboard to play the game.
Output:	Game Menu State
Pass:	Controls must be easy to understand and use. Users should be able to realize that the mouse controls camera movement, and that the keyboard will control movement.
How test will be performed:	Users will first need to join up a Multiplayer Hub and enter a game. After they enter a game, users should move the mouse to test the camera controls. Then, they need to use the keyboard controls (w,a,s,d,spacebar,etc.) to move the character around.

3.2.3 Performance Test

Test 3.2.3.1:	**Speed and Latency Test**
Description:	Test of connection quality when playing a match.
Type:	Static Unit Test
Tester(s):	Game Testers
Initial State:	Users playing a match of Cat and Mouse.
Input:	Mouse and Keyboard to be able to play the game, internet speed of at least 1Mbps.
Output:	Game Menu State
Pass:	Measured ping must not exceed 100ms with internet speed of at least 1Mbps.
How test will be performed:	Users will play a match of Cat and Mouse, in which ping will be displayed to them. As they are playing, developers will watch and record highest ping for each user.

Test 3.2.3.2:	**Graphics Frame Rate and Latency Test**
Description:	Test the frame rate and graphical latency of the game when a match is in progress.
Type:	Static Unit Test
Tester(s):	Game Testers
Initial State:	Players actively participating in a match of Cat and Mouse.
Input:	Mouse and Keyboard, other hardware of a suitable profile to run the game
Output:	Game Menu State
Pass:	Average frame-rate (monitored over a 2-minute period) should not dip below 30 frames per second. 99th percentile of rendered frames have a latency of 25 milliseconds or less.
How test will be performed:	Users will play a match of Cat and Mouse, in which ping will be displayed to them. As they are playing, developers will watch and record highest ping for each user.

Test 3.2.3.3:	**Fault Tolerance Test**
Description:	Test error messages regarding server connection.
Type:	Static Unit Test
Tester(s):	Game Testers
Initial State:	Users must be in a match and connected to each other.
Input:	Mouse, Keyboard to be able to interact with the game.
Output:	Game Menu State
Pass:	Error messages must be displayed when network connectivity has problems. To be more specific: An error message must appear if one user is unwillingly disconnected from a server (such as when their internet access is cut off) which explains why they were disconnected. Players who are not forcibly disconnected must still be able to play the game with no problems.
How test will be performed:	After two users join the same match, one will purposely shut off their internet access and get an error message. The other will be testing if they are still in the match and are able to perform actions such as moving their camera and character.

3.2.4 Operational and Environmental Test

Test 3.2.4.1:	**Operational System Test**
Description:	Test if the game works for a selection of Operating Systems.
Type:	Static Unit Test
Tester(s):	The creators of the game
Initial State:	Executable for the game has not yet been executed on every Operating System we want to test it on.
Input:	Computer running on Windows 7, computer running on Windows 8, Computer running on Windows 10. Mouse to click the executable file.
Output:	Game Menu State
Pass:	The game works on Windows 7, 8 and 10, and has no errors preventing it from being opened or played.
How test will be performed:	Developer will run the executable on machines running Windows 7, 8, and 10. File will be transferred to every machine, and the executable file will be clicked on to run the game. When the game is open, the developer will check for any errors when navigating menus and joining a game.

Test 3.2.4.2:	**Graphics API Test**
Description:	Test if game works with certain graphics APIs.
Type:	Static Unit Test
Tester(s):	The creators of the game
Initial State:	Game has not been played and tested with DirectX9 and DirectX11 graphics APIs.
Input:	Computers that can use DirectX9 and DirectX11 graphics APIs. Mouse to click on the executable.
Output:	Game Menu State
Pass:	Game can be ran and played in DirectX9 and DirectX11.
How test will be performed:	Developer will run the game on computers that have DirectX9 and DirectX11. The developer must check for if there are any problems with game objects after entering a game using both APIs.

4 Tests for Proof of Concept

We will have a proof of concept to demonstrate that our project ideas are possible.

4.1 Risks

Test 4.1.1:	**Photon Unity Network Test**
Description:	Tests whether Photon Unity Network framework can be compiled on Windows 7 to Windows 10. This is a big risk because if Photon Unity Network framework does not compile on the desires OS we are using, then the game's multiplayer will not work. state
Type:	Unit Test
Tester(s):	The creators of the game
Initial State:	Game is not yet launched.
Input:	Using mouse to double left click on the game's executable.
Output:	Game state
Pass:	The game will compile and launch successfully on Windows 7, 8,and 10.

Test 4.1.2:	**Player Load Test**
Description:	Test if game can handle amount of users that PUN states it can support. This is a risk because for the game to be successful, we want to ensure that the game can handle max amount of players. state
Type:	Load Test
Tester(s):	Game Testers
Initial State:	Clients in multiplayer menu, not yet in server.
Input:	Using mouse, click join server button to join a specific server.
Output:	Game state
Pass:	User's client will be in a multiplayer game. Game server should not crash when there is the max amount of players on simultaneously.

Test 4.1.3:	**Player Interaction Test**
Description:	Test if player interactions are concurrent. Successful multiplayer interactions is necessary for the product to be successful. state
Type:	Dynamic Unit Test
Tester(s):	Game creators and Game Testers
Initial State:	Two clients in same server in a state where they are facing each other and no actions are performed.
Input:	Using mouse and keyboard, press 'w' key to move towards each other.
Output:	Player Velocity
Pass:	The two players should move towards and eventually bump into each other which will stop forward movement.

4.2 Proof of Concept Testing

The Proof of Concept demo will be a game that supports multiplayer. Multiplayer will be handled using the Photon Unity Network (PUN) framework for the Unity Game Engine. The game's menu will let players choose a server hosted on a network to connect to and, on success, will be able to see other players currently within the game session.

Players are able to use the mouse and keyboard to interact with other players and the map environment within a game match, through the server.

Test 4.2.1:	**Proof of Concept Testing**
Description:	Test the demo of the game to show that all the risks can be overcome. state
Type:	Proof of Concept
Tester(s):	The creators of the game
Initial State:	Game not yet open.
Input:	Using mouse and keyboard to control character movement, camera movement, and menu navigation
Output:	Game State
Pass:	A game that uses PUN for multiplayer implementation. Players should be able to join a server and successfully interact with each other. Players can select between a cat and mouse character and successfully move the character and attached camera. The game should run on Windows 7, 8, and 10.

5 Appendix

5.1 Usability Survey Questions

Testers should fill this survey after playing the game for 10-20 minutes Select a number based on how much you agree with each of the question, 1 is highly disagree and 10 is highly agree. Please also write down why you agree or disagree with each question.

The time to connect to a game was reasonable.

1 2 3 4 5 6 7 8 9 10

The controls are responsive and easy to learn.

1 2 3 4 5 6 7 8 9 10

The menu is easy to navigate through.

1 2 3 4 5 6 7 8 9 10

Game servers were stable and latency was not an issue.

1 2 3 4 5 6 7 8 9 10