Test Report

1. Test Plan

We employed two types of testing in our plan. The first was usability testing, where we wrote tests for all functionality in the game, including networking, changes between game states, and gameplay itself. We made changes to our original test plan from our specification as the game's design changed.

The second was JUnit testing, where we applied test cases to individual methods to see if they worked as expected. Due to the real-time nature of our game, some tests could afford to fail a certain amount of the time, as long as any exceptions thrown were dealt with, and the program could continue running as normal.

We developed our final product, then used both of these test methods to fix the remaining bugs, and make sure every aspect of the program was functional.

2. Usability Testing

Input	Expected output	Actual output
In state ENTER_IP:		
Enter 'foo' into txtIP and click "Join Server".	Nothing happens.	As expected.
Enter 'foo' into txtSendPort and click "Join Server".	Nothing happens.	As expected.
Click on "Join Server" with a valid IP address in txtIP and port in txtSendPort.	The game changes to the IN_LOBBY state.	As expected.
Click on "Join Server" with correct parameters. The server's game is already in progress.	Nothing happens.	Exception in PlayAreaGUI: paintComponent()
Click on "Join Server" with correct parameters. The server's lobby is full.	Nothing happens.	As expected.

Click on “Join Server” with a valid IP address in txtIP and port in txtSendPort. The server’s lobby is full, but contains AI players.	The game changes to the IN_LOBBY state. A bot is removed from the lobby.	As expected.
After entering an incorrect IP, click on “Join Server’ with correct parameters). The server’s lobby is not full.	The game changes to the IN_LOBBY state.	Nothing happens.
The client joins the lobby and the lobby contains no other players.	The client becomes the ‘party leader’, and btnAddAI, btnSubtractAI, and btnStartGame are visible on the screen.	As expected.
In state IN_LOBBY:		
A new player joins the lobby.	The new player’s address is added to txtPlayerList.	As expected. (Note: If the playerlist size reaches 8, the bottom address in the last does not display correctly.
Another player leaves the lobby.	The player’s address is removed from txtPlayerList.	As expected.
Another player leaves the lobby, and they are the party leader.	The player’s address is removed from txtPlayerList. If the client’s address is now at the top of the list, they become the party leader.	As expected.
The user clicks btnLeaveLobby.	The client changes to the ENTER_IP screen. The client’s address is removed from the player list.	As expected.
The party leader clicks btnAddAI, and the lobby is not full.	‘Bot n’ is added to the player list, where n is the bot’s ID number.	As expected.
The party leader clicks btnAddAI, but the lobby is full.	Nothing happens.	As expected.
The party leader clicks btnSubtractAI, and there are	The most recently added AI player is removed from the	As expected.

>1 AI players in the lobby.	player list.	
The party leader clicks btnSubtractAI, and there are no AI players in the lobby.	Nothing happens.	As expected.
The party leader clicks btnStartGame, and there is an even number of non-AI players in the lobby, and if only two players, at least one AI player.	The client changes to the IN_GAME state.	As expected.
The party leader clicks btnStartGame, and there is an odd number of non-AI players in the lobby.	Nothing happens.	As expected.
In state IN_GAME:		
The player spawns.	An updated HUD is displayed. The player has spawned in a random location, and is not colliding with any walls.	As expected.
The user presses ‘A’, and the bottom edge of their sprite’s bounding box is intersecting with a wall object.	The player accelerates to the left.	As expected.
The user presses ‘D’, and the bottom edge of their sprite’s bounding box is intersecting with a wall object.	The player accelerates to the right.	As expected.
The user presses ‘W’, and the bottom edge of the player’s bounding box is intersecting with a wall.	The player accelerates up for a set period of time, then falls downwards.	The player instantly appears at the maximum height of the jump instead of moving there over time.
The bottom edge of the player’s bounding box is not intersecting a wall.	The player vertical velocity will decrease over time.	As expected.

The player's velocity reaches terminal in any direction.	The player's velocity in that direction is limited to the terminal velocity.	As expected.
The player is moving in a direction, and they collide with a wall.	The player's velocity in the direction that of the collision is set to zero.	As expected.
The player's bounding box collides with a bullet.	The player's health is decremented by one.	As expected.
The player's bounding box collides with an AI player.	The player's health is set to zero.	As expected.
The player's health reaches zero.	The clients of both the player and their teammate display 'Wasted!'. If there is only one other team left alive in the game, the round ends.	As expected.
Another team is eliminated, and the client's team is the only team remaining.	The clients of both the player and their teammate display 'Round Won!'.	
The team's Shooter player clicks the mouse.	A new bullet object is created at the position of the player, and a trajectory in the direction of the mouse cursor, travelling at a constant velocity. A new light source is created and moves with the bullet's position.	As expected.
A bullet object collides with a wall.	The bullet disappears.	As expected.
A bullet object collides with an AI player.	The AI player's health is decremented by one.	As expected.
The team's LightGuy player moves the mouse cursor.	The player's torch-beam updates and points towards the new mouse position.	As expected.
The last round ends.	The client changes to the SCOREBOARD state, then back to the IN_LOBBY state.	As expected.

3. JUnit Testing

Below is a list of a few examples of the JUnit testing we did throughout the project. During the project some members wrote their own testing code as they developed their systems. In some areas of the project, different members of the team would write test cases for each other. This helped us understand each others code better, and made the code higher quality, as we weren't just writing code that could pass our own tests. However, this could have caused some issues that our code was wrote to pass tests, not necessarily work as intended.
For all examples of our testing, please refer to the code stored on the SVN repository.

Method	Class	Class Tested	Description	Scenario	Expected Result	Action Result	Overall Result
position ToTilePos()	AiTest	BotMove	Makes sure that a player position is correctly converted to a TileMap position	Translating player positions to TilePositions, to be used in Ai	int gets converted into the tile position. In the given test, 2	2	Pass
estCostTo Position()	AiTest	BotMove	Gives the distance between the Ai player and it's target	Proximity detection	Boolean, whether or not the player is within the proximity, true & false (tests twice)	true & false	Pass
create Room()	MapTests	Room	Creates a room to be used in map generation	Used when importing rooms from XML	Creates a room, tests if a tile is how it was expected, Tile.WALL	Tile.WALL	pass
import Testing WALLS()	MapTests	CreateMap Objects	Makes sure a room is imported correctly from XML	Used when importing rooms from XML	Imports a room from xml, tests if a tile is how it's expected, Tile.WALL	Tile.WALL	
addRooms()	MapTests	TileMap	Makes sure rooms can be added to maps correctly	Used when placing imported rooms onto the TileMap	Creates a set room, places it on a map. Tests an expected tile. Tile.WALL	Tile.WALL	Pass
test Direction()	Bullet Tests	Bullet	Tests that directions are correct	Called when a bullet is moving	Creates a bullet, sets its direction to '2'	2	Pass
testIs	Collision	Collision	Confirms	Whenever a	Creates a	True/Fals	Pass

Blocked()	Detector 2 Tests	Detector2	that collision detector can see if a player is blocked	player moves	position to work from & places some blockage around the position. Test all four directions. True/False/True/ False	e/True/ False	
testGet MousePos()	Player Tests	Player	Confirms that a mouse position is set correctly	Sent to the server at every tick, used in many calculations	Sets a mouse position as (10,20)	(10,20)	Pass
testGet Blocked()	PosAnd VelTest	Player, CollisionDe tector2	Makes sure classes are working together correcty	Used in player movement	The player should not be blocked, false	false	Pass
testSet GameState()	ServerTo Client Tests	ServerToCli ent	Confirms that client and server can communica te game states	Throughout the game, especially in lobbies & connecting to the server	ServerState.Loa dingServer	ServerSta te.Loadin gServer	Pass
testIsRound Over()	STC_ Game Tests	STC_Game , Player	Makes sure the round doesn't end when the game still has alive players	Every tick of the game!	false	false	Pass
testGetFinal Scores()	Junit Tests	Overall Sever & Game code, including networking	Returns the final score of the game	When the game finishes	0 & 1	0 & 1	Pass
testUpdate Game1	Junit Tests2	Overall Sever & Game code, including networking	Sets positions of players in the server, and their velocities. Updates the game a few times, and tests the movement of the players	Constantly, as the game is being played	Correct positions	Correct Positions	Pass