Test Report

CONTENTS

TEST REPORT

TEST PLAN

INTRODUCTION

This is the test plan for ******. This will contain the testing methods and procedures followed for the testing of the entire game.

TEST COVERAGE

Testing has been done to check for errors within the core functionality of both the high level and low level game components. It has also been done to check for flaws in the HCI components of the game. It has been done for…

- UI of the game
- Networking
- Artificial Intelligence
- Engine and game logic
- Renderer
- Other utility classes

TEST METHODS

Testing took the form of three distinct types; JUnit testing, integration testing, and usability testing. The testing was done throughout the project. JUnit tests were created where required as the project went on. The JUnit tests were mainly used to test core methods within components. These tests were run to check for errors when adding to basic single components and to also check for errors when integrating components together. Usability testing was done towards the end of the project when we had some software with a functional UI which could have its usability tested.

PASS/FAIL CRITERIA

UNIT TESTS

For a unit test of a component to be given a pass all of its test cases had to run successfully. This was because a minor problem in a single class could have led to a catastrophic flaw in the overall functionality of a component.

INTEGRATION TESTING

For an integration test to be given a pass the Unit tests for each class involved in the integration had to be given a pass. This was because all of the functionality tested in the Unit Tests could become unpredictable if components integrated don't interact correctly or communicate correctly.

USABILITY TESTING

Usability Testing did not have a pass or fail criteria. Observations were made about how the user interacted with the game and users suggested changes they would like to see if any.

JUNIT TESTS

Summaries of our JUnit tests can be found in the appendix of this test report. The JUnit tests themselves can be found on the SVN within the test folder inside the junitTesting package.

INTEGRATION TESTING

We integrated individual components incrementally and used regression testing. We adopted a bottom-up integration technique, writing JUnit tests for the low level components to start off with. We then tested the higher level components that utilised these low level components. An example of this is that we wrote JUnit tests for the low level Unit subclasses. These unit classes and also the Renderer were tested before they were used during the implementation of Engine. Engine was then tested using the pre-tested low level components as a basis.

Engine was then merged with the networking components to take us up another level in our design. After this integration, engine tests and also network tests were run again to check for any errors. The UI was then integrated with this new joint networking and engine component. Whilst working on the UI, we tried to make it so it would join with both the engine and networking components separately as easily as possible. This made it easier for us when we joined the UI with the two integrated components.

The tests at each level of integration allowed us to check that individual components 'interact correctly', 'pass data correctly' and 'function cohesively' (E.Claridge, 2014).

Because we took an incremental, bottom-up approach, we always had a working system. This was beneficial as we were always had a base to add functionality to. It also made it easier to find errors as we were noticing them in stages at each level of implementation. This made them easier to find and fix than if we would have adopted an alternative approach such as big bang.

USABILITY TESTING

User testing of the UI occurred during the development of the in game and client UI. User testing was also done during an actual gameplay session towards the end of development. Testing was done across multiple demographics to ascertain how users with different gaming experience and age used, understood and engaged with the UI and game itself.

CLIENT UI USER TESTING

Test ID	Age of User	Gaming Experience	Positives	Negatives	Changes Made
1	19	Experienced	Clear and concise Easy to use	Do not allow the setting of IP address by default as some users do not understand what this is All panels need better graphics Buttons too small and need graphics	Added graphics to all panels including the front panel Created a function that collates all the IP addresses that are suitable to connect to for the game and displays these in the IP address box, so that users do not need to search for their own IP
2	22	Experienced	Easy to understand and use	Apply button in the settings panel should only be active when changes are made to the settings panel	
3	53	Little to none	The number of clicks needed to navigate through the menus is kept to a minimum Clean design in terms of layout and easy to read	Front Panel needs better graphics When setting volume and sound effects, you should be able to hear sample sound Tooltips are needed to let the user know what to do when setting up a game	Added graphics to all panels including the front panel Tooltips added to every component, so that users who may be confused can understand what is needed of them

				Do not allow the setting of port number by default	**Created a check box which can be ticked in order to change port number. By default this port number cannot be change**
4	**52**	**Little to none**	**All text is easily readable.** **The layout of buttons and text is good.**	**Needs to look more appealing** **Needs to have some way to display to the user what needs to be done at certain parts of the UI – for instance, what is an IP?**	**Added graphics to all panels including the front panel.** **Tooltips added to every component, so that users who may be confused can understand what is needed of them.**
5	**16**	**Average**	**Meets the necessary requirements** **Intuitive** **Backgrounds looks nice**	**Settings panel should be fully functional** **Should have a check to make sure that the length of the name input is not over a certain length** **The lobby panel should tell the user when** **AI should be given a name with the tag [AI] so that the user knows that they are AI**	**AI given a name with the tag [AI]**

POTENTIAL IMPROVEMENTS FOR FUTURE DEVELOPMENT

- Settings panel should become fully functional in order to change the volume of sounds and music.
- Buttons in the UI should keep to the same style of the in-game UI – they are currently default swing buttons.
- Should check for the length of the name input when creating a game.
- The window of the UI should have the icon of the game instead of the default java icon; it could also have a title on the frame

USER TESTING OF GAME AND GAME UI

Test ID	Age of User	Gaming Experience	Positives	Negatives	Changes Made
1	53	Little to none	Good graphics in the game UI Good music – well suited to the game Good choice of colours for the map and map entities Popping sound of AI characters moving around is a nice way of knowing that they are moving without them being on your screen.	Would never have thought to go to the menu to quit a game Slow down the AI – they are too quick! Highlight on unit buttons would be nice	Added highlight on unit buttons when the mouse hovers over the button
2	26	Experienced	The functional mini map is excellent Colour of text changing depending on if you can afford a unit or not is informative. The scroll function works very well Intuitive game flow and actions Art style works really well for the genre of game	Fog of war should be added to the game The in game UI should display the units name	Added the unit name into the UI

			Highlighting movement ranges is intuitive and movement itself is fluid		
3	**20**	**Experienced**	**Zoom function works very well** **Interface is clear, intuitive and slick** **Playing a game is fun and challenging against AI**	**Interface could have larger text** **AI should have proper names**	**Larger text implemented into game** **Proper names given to AI**
4	**16**	**Average**	**Likes the colour of the hover over the tile** **Likes the way the mouse icon changes base on the way the user interacts with the mini map** **Likes the functionality of the mini map** **The game runs well and at a good pace – fun to play against friends and AI.** **Art style works well and the units are interesting and entertaining**	**Background of game should fit the game theme more** **Timer should not be displayed in seconds**	**Added a new background into the game**

POTENTIAL IMPROVEMENTS

- When it is the opponents turn it would be good to focus on the map on the area of where movement is occurring.
- Game timer should be displayed in terms of minutes and seconds, rather than just minutes.
- Fog of war could be added to the game – this was attempted, but proved too difficult for the time scale of the project.
- Colours of a player, their team and their enemy should be better indicated to players in the game.
- Mini map could be used to show where your own base, enemy bases and ally bases are being built.

APPENDIX A

Test ID	Test Method	Test Class	Class Tested	Test Description	Test Scenario	Expected results	Actual Results	Overall Test result
1	moveUnitInRange()	TestingEngineMethods	Engine	A unit (Marine) is added to the world on tile[0][0] and is selected using the leftClickMap in engine. It is then moved to in range tile[1][1] using the moveUnit method in engine.	The test runs the moveUnit method and asserts that it returns true.	moveUnit returns true to to indicate movement was successful.	TRUE	PASS

2	moveUnitOutOfRange()	TestingEngineMethods	Engine	A unit (Marine) is added to the world on tile[0][0] and is selected using the leftClickMap in engine. It is then moved to an out of range tile[59][59] using the moveUnit method in engine.	The test runs the moveUnit method and asserts that it returns false.	moveUnit returns false to to indicate movement was unsuccessful.	FALSE	PASS
3	moveUnitOccupiedTile()	TestingEngineMethods	Engine	A unit is added to the world on a tile[04][04] and and left to occupy it. Another unit (Marine) is created on tile[0][0] and is selected using the leftClickMap method in engine. It is then moved to the occupied tile[04][04] using the moveUnit method in engine.	The test runs the moveUnit method and asserts that it returns false.	moveUnit returns false to to indicate movement was unsuccessful.	FALSE	PASS

4	buildBuildingInRange()	TestingEngineMethods	Engine	A building (Command Centre) is added to the world on tile[40][40]. Another building (Barracks) is then built on an in range tile[35][40] using the createBuilding method in engine.	The test runs createBuilding and asserts that it returns true.	createBuilding returns true to to indicate building creation was successful.	TRUE	PASS
5	buildBuildingOutOfRange()	TestingEngineMethods	Engine	A building (Command Centre) is added to the world on tile[40][40]. Another building (Barracks) is then built on an out of range tile[0][0] using the createBuilding method in engine.	The test runs createBuilding and asserts that it returns false.	createBuilding returns false to to indicate building creation was unsuccessful.	FALSE	PASS
6	buildArmyCorrectType()	TestingEngineMethods	Engine	A building (Command Centre) is added to the world on tile[40][40]. An army of a valid type (scout) is then built using	The test runs createArmy and asserts that it returns true.	createArmy returns true to to indicate Army creation was successful.	TRUE	PASS

				the createArmy method in engine.				
7	buildArmyIncorrectType()	TestingEngineMethods	Engine	A building (Command Centre) is added to the world on tile[40][40]. An army of an invalid type (Rocket) is then built using the createArmy method in engine.	The test runs createArmy and asserts that it returns false.	createArmy returns false to to indicate Army creation was unsuccessful.	FALSE	PASS
8	buildArmyRallyPointCheck1()	TestingEngineMethods	Engine	A building (Command Centre) is added to the world on tile[40][40]. An army of a valid type (scout) is then built using the createArmy method in engine.	The test runs createArmy and then asserts that the first available rally point from the getAvailableRallyPoint method in Building is not null.	The first rally point is not null to indicate that the scout was created on the correct tile.	TRUE	PASS

9	buildArmyRallyPointCheck2()	TestingEngineMethods	Engine	A building (Command Centre) is added to the world on tile[40][40]. Scouts are created on the first and second available rally point. An army of a valid type (scout) is then built using the createArmy method in engine.	The test runs createArmy and then asserts that the final available rally point from the getAvailableRallyPoint method in Building is not null.	The final rally point is not null to indicate that the scout was created on the correct tile.	TRUE	PASS
10	buildArmyRallyPointCheck3()	TestingEngineMethods	Engine	A building (Command Centre) is added to the world on tile[40][40]. Three scouts are created created to occupy all of the available rally points. An army of a valid type (scout) is then built using the createArmy method in engine.	The test runs createArmy and then asserts that it returns false.	createArmy returns false to to indicate Army creation was unsuccessful.	FALSE	PASS

11	moveUnitInstruction()	TestingEngineMethods	Engine	A NullPlayer is added to the Player Manager. A unit is then added to the world on tile[0][0]. A move instruction with a body of 00000101 is then sent to the engine.	The test first checks that the unit exists before the instrcuction is sent. It then checks that it is moved correctly by checking that the original tile contains null units and the target tile[01][01] contains a unit.	The unit is moved to the target tile correctly.	TRUE,TRUE,TRUE	PASS
12	attackUnitInstruction()	TestingEngineMethods	Engine	A NullPlayer is added to the Player Manager. A unit to attack with is then added to the world on tile[0][0] and a tile to attack is added on tile[01][01]. An attack instruction with the body of 00000101 is is then sent to the engine. The instruction is then read using the readInstruction method in engine.	The test asserts true that the after health of the target unit is less than the before health.	The attack is dealt with correctly.	TRUE	PASS

13	createBuildingInstruction()	TestingEngineMethods	Engine	A NullPlayer is added to the Player Manager. A unit to build from is then added to the world on tile[40][40]. A NEW_UNIT instruction with the body of 11353531 is is then sent to the engine. The instruction is then read using the readInstruction method in engine.	The test asserts true that the target tile[35][35] contains a unit and that it is an instance of Building and a Barracks.	The creation is dealt with correctly.	TRUE,TRUE,TRUE	PASS
14	createArmyInstruction()	TestingEngineMethods	Engine	A NullPlayer is added to the Player Manager. A unit to build from is then added to the world on tile[40][40]. A NEW_UNIT instruction with the body of 01353531 is is then sent to the engine. The instruction is then read using the	The test asserts true that the target tile[35][35] contains a unit and that it is an instance of Army and a Scout.	The creation is dealt with correctly.	TRUE,TRUE,TRUE	PASS

				readInstruction method in engine.				
15	testAttack()	TestingUnitMethods	Army	A marine is added to the world and a tank is added to the world. The tank is then attacked by the marine using the attack method in Unit.	The test asserts true that the health of the tank before the attack is greater than the it's health after the attack.	The attack is performed correctly.	TRUE	PASS
16	testAttackBonus()	TestingUnitMethods	Army	A rocket unit is added to the world and a tank is added to the world. The tank is then attacked by the rocket using the attack method in Unit.	The test asserts true that the health of the tank after the attack is equal the health before the attack minus the damage of the rocket multiplied by the bonus the rocket has against attack.	The attack is performed and and the correct damage is dealt.	TRUE	PASS
17	testAttackFriendly()	TestingUnitMethods	Army	A scout unit on team 1 is added to the world and a tank also on team 1 is added to the world. The tank is then attacked by the	The test asserts true that the health of the tank before the attack is equal to the health after the attack.	The attack is performed and no damage is dealt.	TRUE	PASS

				scout using the attack method in Unit.				
18	endOfTurnReset()	TestingUnitMethods	Army	A tank on team 1 is added to the world and a scout on team 2 is added to the world. The scout is then attacked by the tank using the attack method in Unit. The endOfTurnReset method is then called.	The test asserts true that the attackNumber and rangeMove before the attack are equal to the attackNumber and rangeMove after the endOfTurnReset.	The reset is performed and variables are set back correctly.	TRUE,TRUE	PASS
19	addUnit()	TestingWorldMethods	World	An Army is created ready to add to the world. The Army is then added using the addUnit method in World.	The test asserts false that the world does not already contain the unit before addUnit is called. It then asserts true that world contains the unit after addUnit is called.	The unit is added to the world correctly.	FALSE, TRUE	PASS

20	removeUnit()	TestingWorldMethods	World	An Army is created ready to add to the world. The Army is then added using the addUnit method in World. This unit is then removed from the world using the removeUnit method in World on each tile containing the added unit.	The test asserts false that the world contains the unit after the removeUnit method has been called.	The unit is removed correctly.	FALSE	PASS

21	noDebrisInBase()	TestingMapGeneration	MapGeneration	a Tile[][] is created using the GetTileMap() method inside of the MapGenearation class	The test will then itterate throgh all of the tiles that make up the areas of the base and check their type against the constant GRASS. If the tile is not Grass the Test will fail.	The Test should return without failing any of the base tests and pass.	the noDebrisInBase() method passed the test on map generation	PASS
22	variedMapSize()	TestingMapGeneration	MapGeneration	a MapGenration class is instantiated with the map size of 20 by 20 in order to check if the	the test checks that the final tile[][] that is produced is the same size as the	the Tile[][] should be the same size as the specified map size.	the variedMapSize() returned with no errors and	PASS

				generation still works for maps of different sizes. The map is then generated then the size is checked to see if it matches 20.	specified map size (20)		passed on the map.	
23	allTerrainTypesUsed()	TestingMapGeneration	MapGeneration	This test is to see that all of the terrain types in the map generator have been generated to ensure that it is a truly diverse map. By checking for all the types of terrain after the map is generated.	a map is created using the MapGenerations GetTileMap() method then the entire grid is checked against all of the different types of terrain and returns fail() if any type of terrain isnt generated.	the method should return with no fails as all of the types of terrain should be generated.	the method allTerrainTypesUsed() returned without failing.	PASS
24	randomness()	TestingMapGeneration	MapGeneration	this is a Test to see whether the maps are truly random by generating 2 different maps using the same instance of the MapGeneration class and checking that both of the generated maps are different.	one instance of the class Map Generation is created then the genmap() method inside the mapGeneration class is called to create 2 separate maps. These maps are then compared to each other to	the method should return without failing as both of the maps are generated randomly and should as a result be different.	the method returned without failing meaning that 2 different maps were generated.	PASS

					look for differences. If no differences are found then the test Fails.			
25	maxHeight()	TestingMapGeneration	MapGeneration	this is a Test to see if the genHeightMap() method ever creates a tile which is higher than the max height (255)	the test creates a MapGeneration instance then directly calls its genHeightMap() method which returns a int[][] . All of the values of the int[][] are then checked to see if they exceed the maximum height of (255)	the method should return without failing to show that none of the tiles are higher than the max height of the map.	the method maxHeightMap() returns without failing showing that the height is always less than the max height.	PASS

26	blockedPathTest()	AstarSearchTest	AStarSearch	This is a test to see whether the unit is correctly pathing round obsticals (other units) when it is calculating the optimal path to the goal.	a AstarSearch Scinario is set up one with a unit directly blocking the path to the goal and another with no unit blocking the path to the goal. The search with the block should then return a longer route to the goal if it is pathing correctly.	the method should return without failiing which will identify that the unit with the blocked path had a longer route.	the method blockedPathingTest() returned with no fail	PASS
27	terrainNavigated()	AstarSearchTest	AStarSearch	this is to test that the astar seach is properly pathing arround large obsticals and terrain obsticals.	two identical maps are created using the xml parser one with no terrain and one with terrain. The unit is then asked to travel to the same goal and the length of the path is compared.	the route for the map with obsticals should be longer as they have to path round the complex terrain.	the method terrainNavigated() should return with no errors.	PASS
28	invalidPath()	AstarSearchTest	AStarSearch	This test is to identify if the a aStarSearch is incorectly calculating a path to an invalid map point	the a star search is set up on a random map and then the unit is given a goal that is off the map (-1,10)	this should return a null route as it is impossible to navigate to the given location.	the method returned without failing showing that the route	PASS

							returned was indeed null.	
29	blockedNode()	AstarSearchTest	AStarSearch	This is to test that astar will not return a path to a vailid map position that it is impossible to get to.	a unit is given a goal to get to that is surrounded on all avalable sides by blockages that prevent the search from pathing to the.	the method should return without failing as the route is null. This will show that it has failied to path to a blocked off location.	the method returned without failing.	PASS

30	testPointMaths()	TestingRenderer	Renderer	The test calls the renderer and checks the gridFromPixels() method by asking for a coordinate based on the screen size.	A game is setup correctly for the Renderer to run as expected. There is only one player in the player manager.	The test should return the correct Point(0,0) coordinates for the grid.	The method returned the Point(0,0) as expected.	PASS

31	testPointMaths()	TestingRenderer	Renderer	The test calls the renderer and checks the pixelsFromGrid() method by asking for a coordinate Point(0,0).	A game is setup correctly for the Renderer to run as expected. There is only one player in the player manager.	The test should return the correct coordinates for the grid based on the screen.	The method returned the correct point based on my screen size.	PASS
32	testWaterParser()	TestingMapGeneration	XmlParser	The test checks if the correct water edge is being generated by the parser for a custom map XML input containing a lake in it.	An map XML is input to the XmlParser. This map contains a lake for the Parser to generate water edges on. The generated map is then compared to another map that has the correct terrain.	The test should assert that both the map being checked and the correct map match.	The maps match as expected.	PASS
33	testLeftClick()	TestingEngineMethods	Engine	The test checks if the left click correctly selects a unit in the map.	The Engine is started with a single player. The action to leftClickOnMap() is called on Point(7,7).	The method should correctly select the auto-generated command centre for the player.	As expected, the correct tile was selected, with the correct unit on it.	PASS

34	testRightClick()	TestingEngineMethods	Engine	The test checks if the right click works correctly by trying to move a Scout.	The Engine is started with a single player. The left click action is given to select the command centre. A create action is given for it to create Scout unit. The Scout is then selected via left click, and then moved via right click.	The Engine should be able to correctly select the Command Centre and create a scout, then move it to the desired position.	The function works as expected.	PASS
	openTest()	TestNetworkingMethods	Server	Tests that the server is opened.	A new server is created on port '53551'.	The server is opened and the isOpen flag is set to true.	The server is opened and the isOpen flag is set to true.	PASS
	endGameTest()	TestNetworkingMethods	Server, Player	Test that the server closes on receiving a EndGame instruction from the host.	A new server is created on port '53552' . A new ClientPlayer is created and an end game instruction is sent by that player.	The server is opened and the isOpen flag is set to true. Subsequently the server is closed and isOpen flag is set to false.	The server is opened and the isOpen flag is set to true. Subsequentl y the server is closed and	PASS

							isOpen flag is set to false.	
	allocatedTest()	TestNetworkingMethods	Server, Player	Test that players are allocated and are receiving allocation information correctly.	A new server is created on port '53553' . A new ClientPlayer is created and it examines the instructions it is receiving.	The server is opened. The ClientPlayer examines its instruction and reads the instruction '2S240'.	The server is opened. The ClientPlayer examines its instruction and reads the instruction '2S240'.	PASS
	multiAllocationTest()	TestNetworkingMethods	Server, Player	Test that multiple players are allocated and are receiving allocation information correctly.	A new server is created on port '53554' . Two new ClientPlayers are created and they examine the instructions they are receiving.	The server is opened. The first ClientPlayer examines its instruction and reads the instruction '2S240'. The second ClientPlayer examines its instruction and reads the	The server is opened. The first ClientPlayer examines its instruction and reads the instruction '2S240'. The second ClientPlayer examines its instruction	

						instruction '2S241'.	and reads the instruction '2S241'.	
	singleDisconnect()	TestNetworkingMethods	Server	Test that if the only player connected to the server disconnects the server closes.	A new server is created on port '53555'. A new ClientPlayer is created and sends a 'Disconnect' instruction to the server.	The server is opened. The isOpen flag is set to true. The player sends a disconnect instruction and the server closes. The isOpen flag is set to false.	The server is opened. The ClientPlayer. The isOpen flag is set to true. The player sends a disconnect instruction and the server closes. The isOpen flag is set to false.	PASS
	multiDisconnect()	TestNetworkingMethods	Server	Test that if all players connected to the server disconnects the server closes.	A new server is created on port '53556'. Two new ClientPlayer are created and send a 'Disconnect' instruction to the server.	The server is opened. The isOpen flag is set to true. Both player send a disconnect instruction and the server closes. The	The server is opened. The isOpen flag is set to true. Both player send a disconnect instruction and the	PASS

						isOpen flag is set to false.	server closes. The isOpen flag is set to false.	
	playerAdded()	TestPlayerManager	Player Manager	Test that players that are added into the PlayerManager increase the size of the PlayerManager.	A new PlayerManager is created. Two new NullPlayer is created and added to the PlayerManager. The size of the PlayerManager before and after adding a player is recorded in both cases.	The PlayerManager is created and its size is initially set to zero. After adding a player the size increases to one. Adding another player increases the size to two.	The PlayerManager is created and its size is initially set to zero. After adding a player the size increases to one. Adding another player increases the size to two.	PASS
	hostCheck()	TestPlayerManager	Player Manager	Test that the first player added into the PlayerManager is also assigned as the host.	A new PlayerManager is created and the NullPlayer is added. The first NullPlayer to be added is compared to the player stored as	The first NullPlayer and the host stored in the PlayerManager are equal to each other.	The first NullPlayer and the host stored in the PlayerManager are equal to each other.	PASS

					the host in PlayerManager.			
	iteratorCheck()	TestPlayerManager	Player Manager	Test that the iterator, iterates over all players in the PlayerManager.	A new PlayerManager is created. Three new NullPlayers are added to the PlayerManager. A new player array is created and the players are added in the same order as they were added to the PlayerManager. A new player array is created. The PlayerManager is iterated over using the built in iterator. At each step of the iteration the player referenced by the iterator is added to the array. The two arrays are	The first array is created and filled with NullPlayers manually. The second array is created and filled with NullPlayers using the iterator. The arrays are both of the same size. The arrays are compared and are equal to each other.	The first array is created and filled with NullPlayers manually. The second array is created and filled with NullPlayers using the iterator. The arrays are both of the same size. The arrays are compared and are equal to each other.	PASS

					compared for equality.			
	getAtTest()	TestPlayerManager	PlayerManager	Test that players can be retrieved properly using their ID.	A new PlayerManager is created. Two new NullPlayers are created and added to the PlayerManager.	The first player is compared to the player returned by getPlayerByID(0) and is equal. The second player is compared to the player returned by getPlayerById(1) and is equal.	The first player is compared to the player returned by getPlayerByID(0) and is equal. The second player is compared to the player returned by getPlayerById(1) and is equal.	PASS

	peekTest()	TestPlayerManager	Player Manager	Tests that the next player can be retrieved from the PlayerManager without rotating it.	A new PlayerManager is created. Two new NullPlayers are created and added to the PlayerManager.	The first player is compared to the player returned by getCurrentPlayer() and is equal. The second player is compared to the player returned by peek() and is equal. The first player is compared to the player returned by getCurrentPlayer() and is equal.	The first player is compared to the player returned by getCurrentPlayer() and is equal. The second player is compared to the player returned by peek() and is equal. The first player is compared to the player returned by getCurrentPlayer() and is equal.	PASS
	disconnectTest()	TestPlayerManager	Player Manager, Player	Tests that a player can be disconnected from the PlayerManager i.e converting any player type to a NullPlayer	A new Server is created on '53556'. A new ClientPlayer is created. A new PlayerManager is created and the ClientPlayer is added. The players disconnect method is called.	The player in the first slot of the PlayerManager is checked to see if it is an instance of ClientPlayer and it is. The client players	The player in the first slot of the PlayerManager is checked to see if it is an instance of ClientPlayer and it is. The	PASS

						disconnect method is called. The Player in the first slot of the PlayerManager is checked to see if it is an instance of NullPlayer and it is.	client players disconnect method is called. The Player in the first slot of the PlayerManager is checked to see if it is an instance of NullPlayer and it is.	

REFERENCES

Lecture slides - Integration and system testing – Ela Claridge