ArtemisLite Project Report 2021

Introduction

The ability to work effectively as part of a team, putting into practice the most appropriate practices is crucial in the world of software engineering. The following report intends to display the development process in the creation of a virtual, Monopoly-like, board game based on the real life lunar landing mission, Artemis. A number of deliverables consisting of UML diagrams, design documents and testing plans are produced throughout the different stages of the development, reflecting what a customer may request in a real-world professional setting. These stages of development range from the early requirements analysis phase to the final testing process of the software.

Requirements Analysis

Gameboard

Figure1 below showcases the virtual game board, inspired by the actual working systems of the Artemis Mission. For the game resources required for buying, developing and paying rent within the game, the terms 'Workforce' & 'Equipment' are used. There are four systems, two consisting of three adjacent Tiles and two consisting of two adjacent Tiles. The system names are defined in the games welcoming messaging, for reference; System1: Exploration Ground Systems (EGS), System2: Orion Crew Module (OCM), System3: Space Launch Systems (SLS) and System4: Parking Orbit (PO). Systems 1&2 require the least number of resources to purchase and develop, imitating the initial phases within the real-life Artemis Mission. Meanwhile, system4 is the most expensive, replicating the final phases, including 'Deep Space Travel Checks' & 'Leave Earth Maneuverer'.

Figure 1: Gameboard Virtual Layout

Use Case Descriptions

Below is the text-based narrative of the game's functionality, step-by-step interaction between the actor and the system. It describes the outcomes of an action taken to accomplish a specific goal, along with any pre conditions, alternative flows and post conditions. The diagrams were created as a team during the initial phases of the project, special thanks to Mr. Moshen for his advisory role.

Flow of Events for *Select Players* use-case	
Objective	Selecting the number of players and set names
Preconditions	Have ArtemisLite game opened on Eclipse console.
Main flow	1) Console prompts the actor for a number of players. 2) Actor inputs the number of players (>1 and <5). 3) Actor presses Enter on the keyboard. 4) Console prompts user for name of player 5) Actor inputs the name of each player into the console. 6) Actor presses Enter on the keyboard after the name. 7) Repeat step 4 through 6 for every player 8) Game creates players 9) A welcome / instruction message is shown in the console.
Alternative flows	● At 2, the player enters less than two or more than 4 players. In this case a message appears informing the actor of error, and the user must enter the correct number of players (Alternative 1). ● At 5, the actor enters the same name for more than 1 player. In this case, an error message appears, and the user must enter a different name (Alternative 2).
Post condition	The number and names of each player have been decided and the game is ready to begin.

Flow of Events for *Take Turn* use-case	
Objective	Move selected player to a new position on the virtual board
Preconditions	Have completed the *Select Players* use case.
Main flow	1) Selected Player is prompted by the console if they want to continue playing. (Y/N) 2) Player inputs Y. 3) Two virtual dice are rolled. 4) The sum of the dice is displayed to the player. 5) The player is moved to the required game tile. 6) The description and status of the tile is presented to the player. 7) Extend buy/develop/pay rent 8) Player turns ends.
Alternative flows	● At 2, the player selects No. Turn ends (Alternative 1). ● At 7, if the tile is owned by a different player, the console prompts the owner to charge rent. *(Extension point: Pay rent)* (Alternative 2). ● At 7, if the current player does not wish to buy tile, the purchase option is passed on to the next player *(Extension point: buy tile)* (Alternative 3). ● At 7, the game checks system ownership. If a player owns a whole system, the game prompts the owner to develop tiles within the owned system (*Extension point: develop tile)* (Alternative 4). ● At point 7, the last development has been made. Game calls Win game use case. (Extension point: Win game) (Alternative 5). ● Player lands on free square (Alternative 6). *Notes:* ● At 7 if a player does not own a full system, no development is possible. ● At 5, player turn may result in player passing go which allows them to collect resources. In this case resources are collected. The next player then takes their turn.
Post condition	The player has moved to the appropriate game tile.

Flow of Events for *Develop Tile* use-case	
Objective	Player develops Tile
Preconditions	Player owns all system Tiles It is the player's turn at present
Main flow	1) If developments are available, game displays available Tiles for development along with Tile stats. 2) Game asks player if they want to develop 3) Player selects Y 4) Game asks player which Tile to develop 5) Player inputs number of Tile to be developed 6) Game develops Tile 7) Game deducts development fee from players' resources. 8) Game displays players' current resources. (*Include Display Balance)*
Alternative flows	● At 3, the player selects No. Game continues on to the next player.
Post condition	Player develops Tile

Flow of Events for *Pay Rent* use-case	
Objective	Player pays rent to another player
Preconditions	Player lands on other players' Tile
Main flow	1) Owner is prompted to collect rent. (Y/N) 2) Owner inputs Y. 3) Game deducts rent fee from current player's resource and adds them to tile owners resources 4) Game confirms rent paid and displays resources for all players. *(via Display Player Stats)*
Alternative flows	● At 2, owner inputs No. In this case, no fees are deducted. (Alternative 1). *Game continues.* ● At 3, the current player does not have enough resources to pay the fee, this triggers the game over. All player resources are displayed and game over message. (Extension point: game over) (Alternative 2).
Post condition	Rent resources transferred to owner's balance

Flow of Events for Win game use-case	
Objective	Players win the game
Preconditions	All Tiles in every system are fully developed
Main flow	1) The game checks if all systems are owned 2) The game checks if every Tile is fully developed (3minor & 1major) 3) Win game condition is set to true *4)* The game calls the display win game message method. *(Extension point: end game*
Alternative flows	● At 1) If more than one player owns Tiles in the same system, winning is not possible, game ends *(Extension point: end game*
Post condition	Players have won the game

Flow of Events for *Display Stats* use-case	
Objective	Display current game stats
Preconditions	Take turn ended
Main flow	1) The game looks up the current stats for all tiles and players 2) The game outputs to the console the current game stats and reason for any updates
Post condition	New game stats output to screen

Flow of Events for End game use-case	
Objective	Output end game Message and game stats
Preconditions	All systems fully developed
Main flow	1) Check win game condition 2) Win game condition is true 3) display win game message 4) Display the game state, and player stats
Alternative flows	● At 2) Win game condition is set to false (Alternative 1). ● At 3) display game lost message
Post condition	Game is finished

Using the text-based description of the game flow, a visual representation of the game's use cases could be expressed in the form of a UML Use Case Diagram.

Use Case Diagram

Figure2 below demonstrates the relationship between the use cases and the actors. The diagram models the ArtemisLite game flow and was created using LucidChart.com. The *includes* and *extends* relationships are visible, illustrated by dashed arrows.

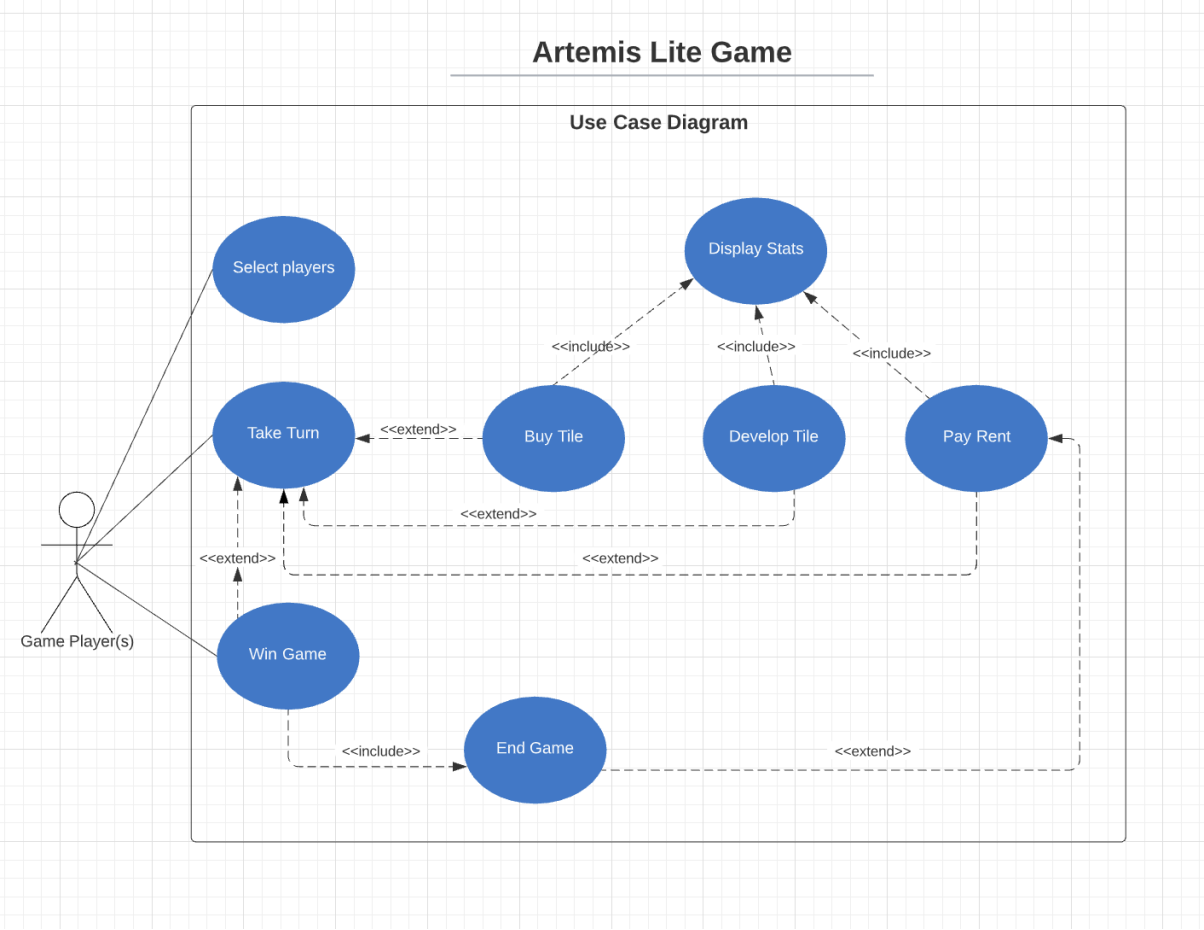

Figure 2 Use Case Diagram

Realisation

UML Sequence Diagrams

Figures 3 to 9 below consist of the game's UML Sequence diagrams, showcasing how the internal software components make method calls to each other, and their interaction with the actor. The ordering below follows the same assembly of the use cases description tables above, from 'Select Player' to 'Win Game'. Where a grey outlined box is present, labelled 'Alternative' along with a number, is a representation of the game's flow, alternative to the normal happy flow. Each alternative flow is displayed at the bottom of each individual sequence diagram, contained within a separate blue bordered box. The start point for each alternative flow will be identified within the normal flow.

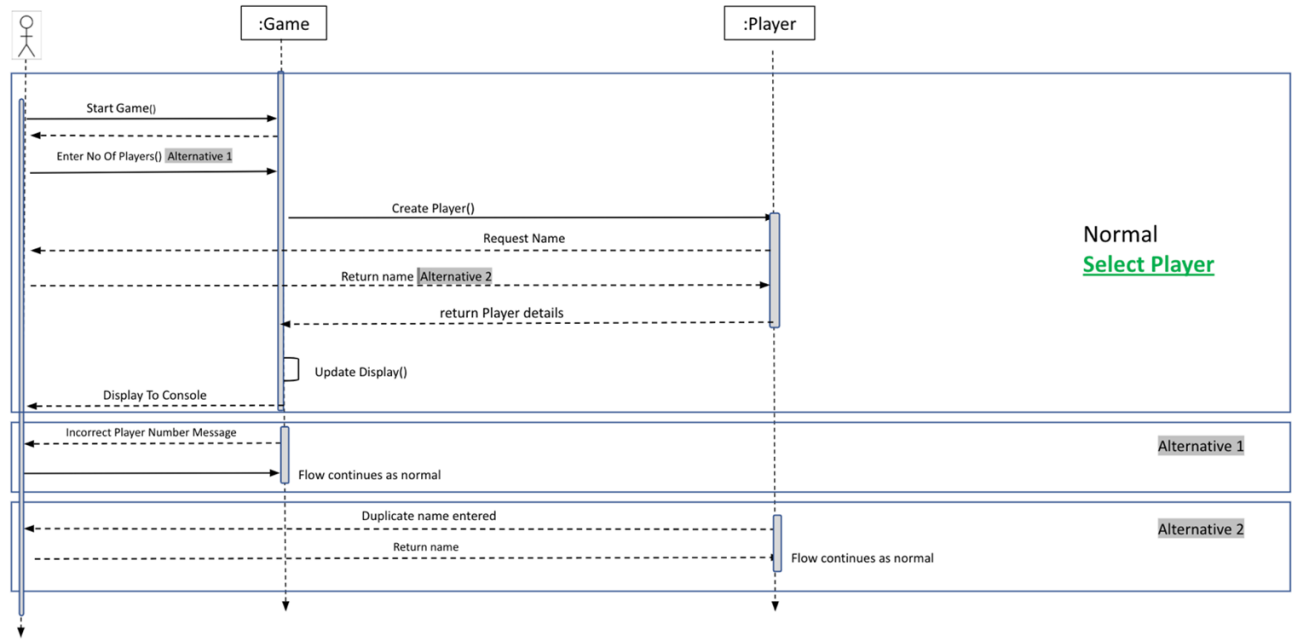

Figure 3: Select Player Sequence Diagram

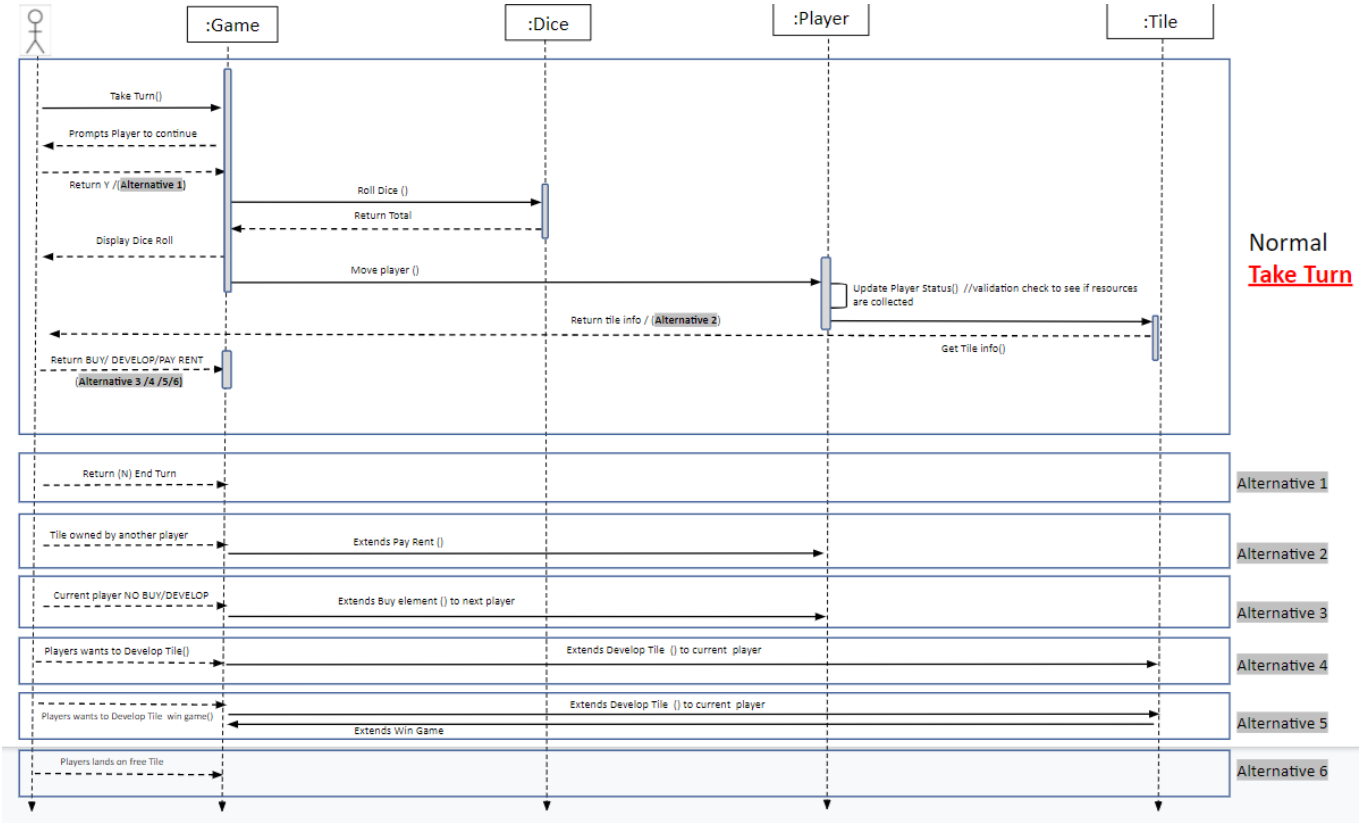

Figure 4: Take Turn Sequence Diagram

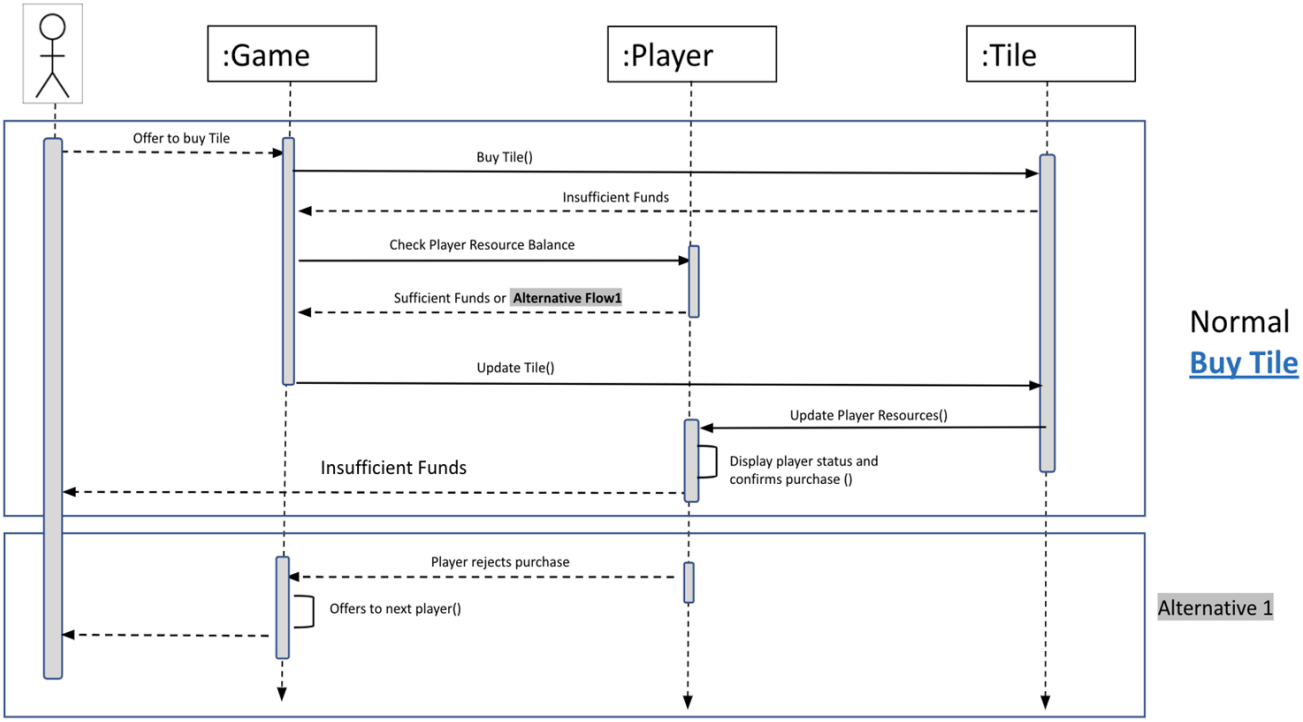

Figure 5: Buy Tile Sequence Diagram

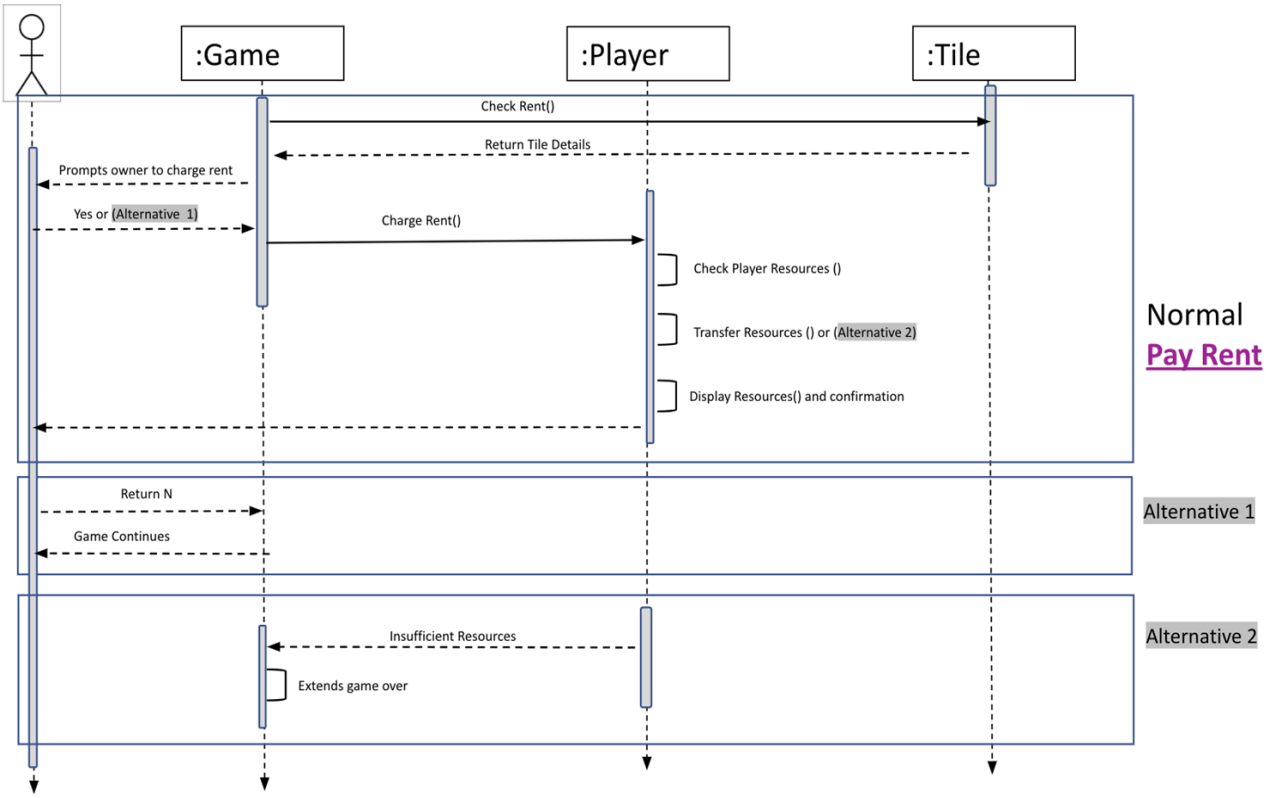

Figure 6: Pay Rent Sequence Diagram

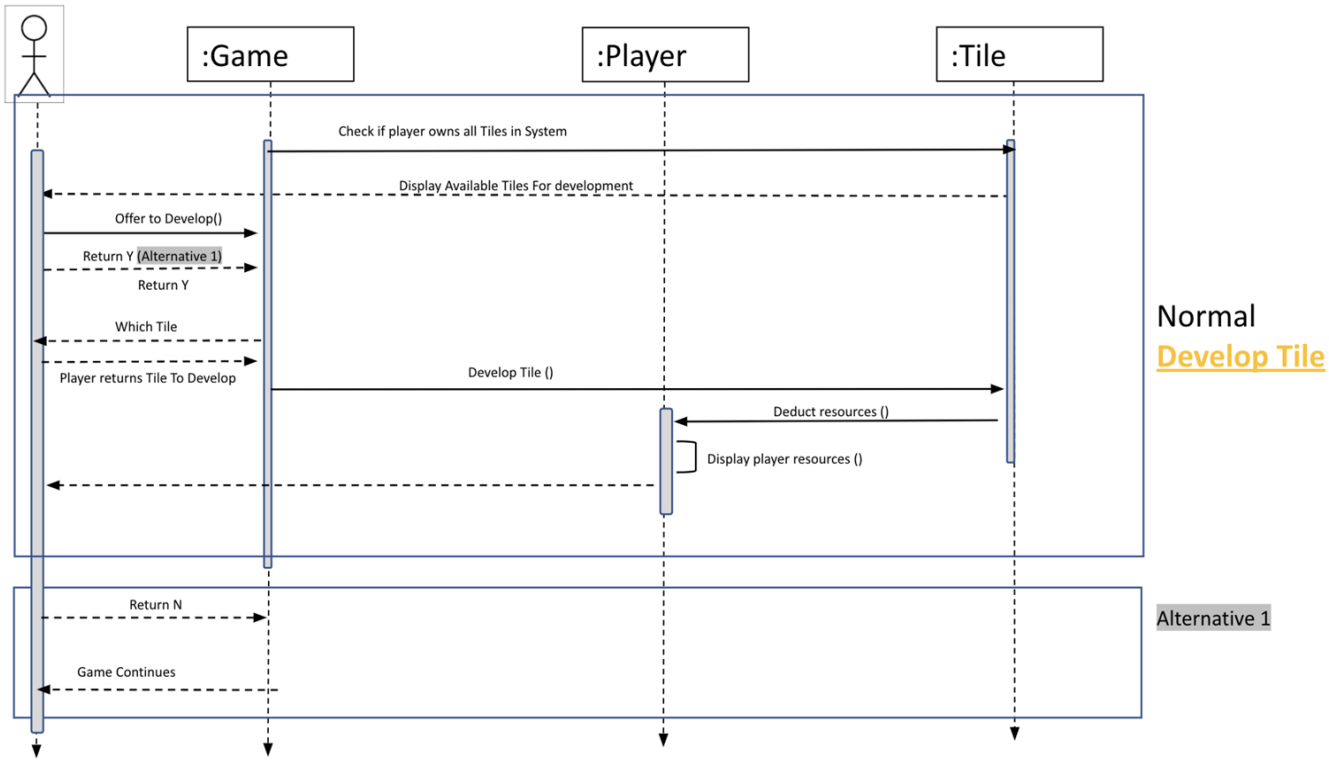

Figure 7: Develop Tile Sequence Diagram

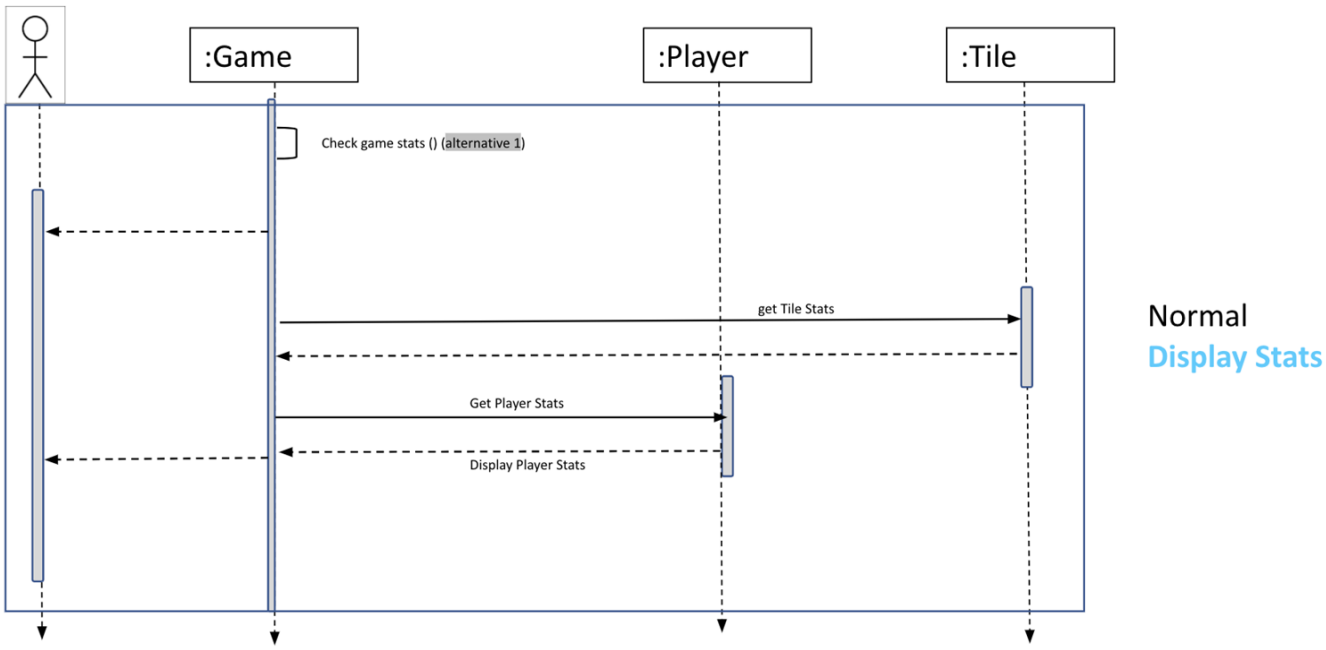

Figure 8: Display Stats Sequence Diagram

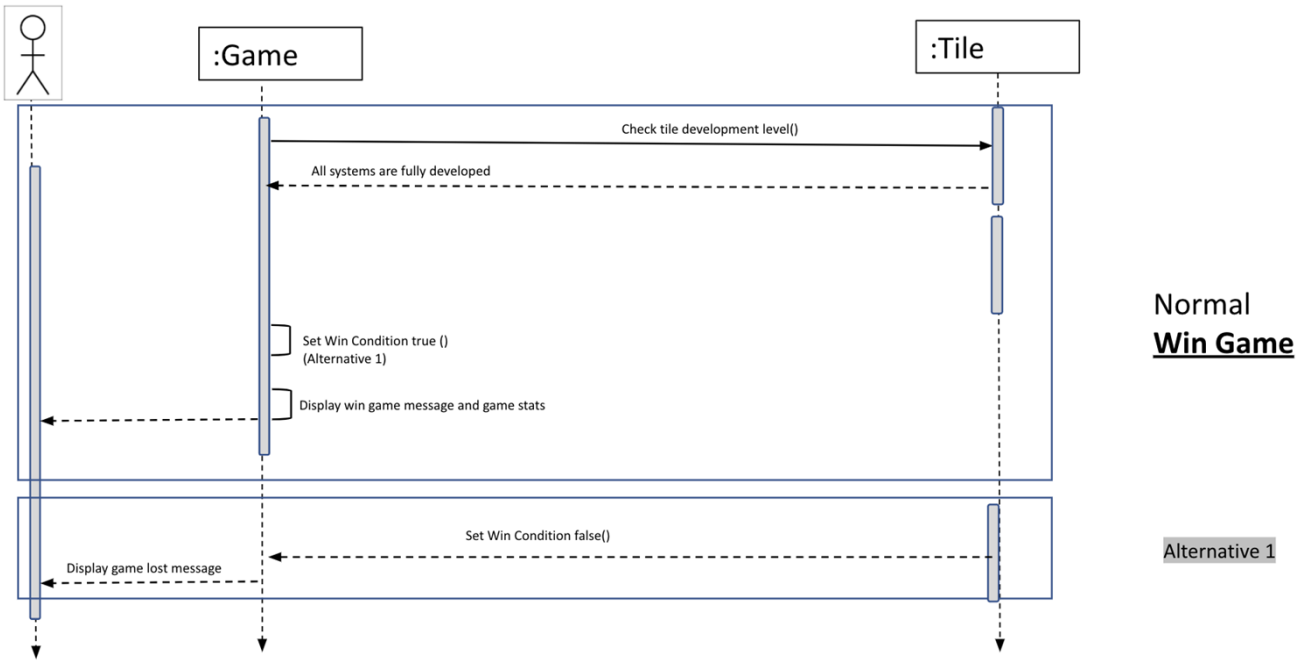

Figure 9: Win Game Sequence Diagram

Design

Class Diagrams

Figure10 below displays the initial draft of the UML Class Diagrams, whereas figure11 presents the final. Figure11 corresponds closely to the coded implementation of the game, exposing how classes and methods support the sequences of method calls described in the section above.

When coding, it was important to keep the maintainability and extensibility of the game in consideration. The game's extensibility could come in the fashion of extra Players, Tiles and Systems. Rather than hard-code these variable limits, they have been set using constants, which can be easily amended and found at the top of the classes. As the Separation of Concerns principle was adhered to, the majority of code is cohesive and with single responsibility, making further game developments less likely to invite bugs.

Within any software development project, it's likely that things are going to require adjustment during development and after production. Hence, it was important that the games' software was easy to read and maintain. Coding with the support of the use case, class and sequence diagrams ensured that code was recycled where possible. Clear and concise commenting, along with meaningful variable and method names, increases the codes readability.

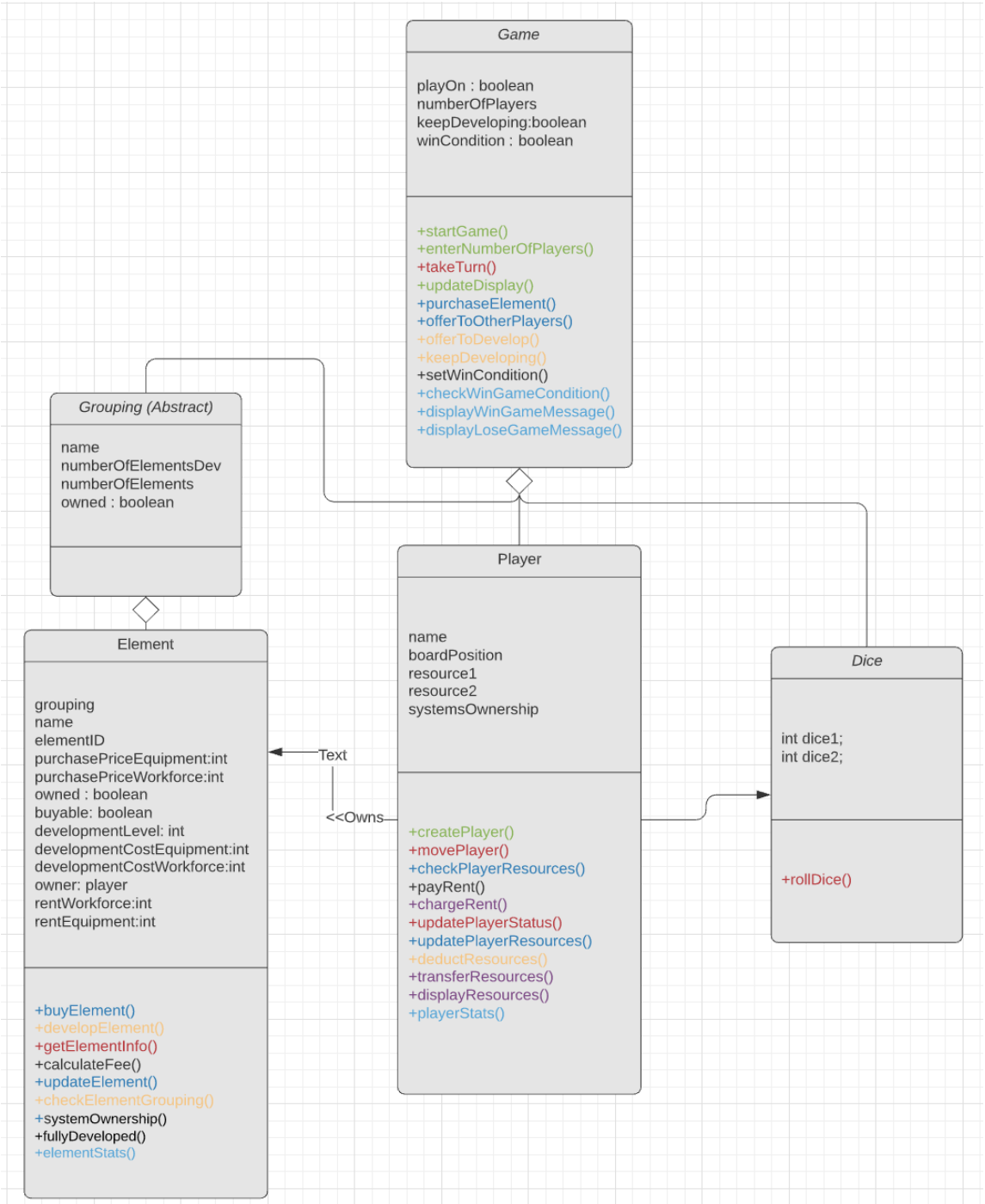

Figure 10 UML Class Diagram Draft1

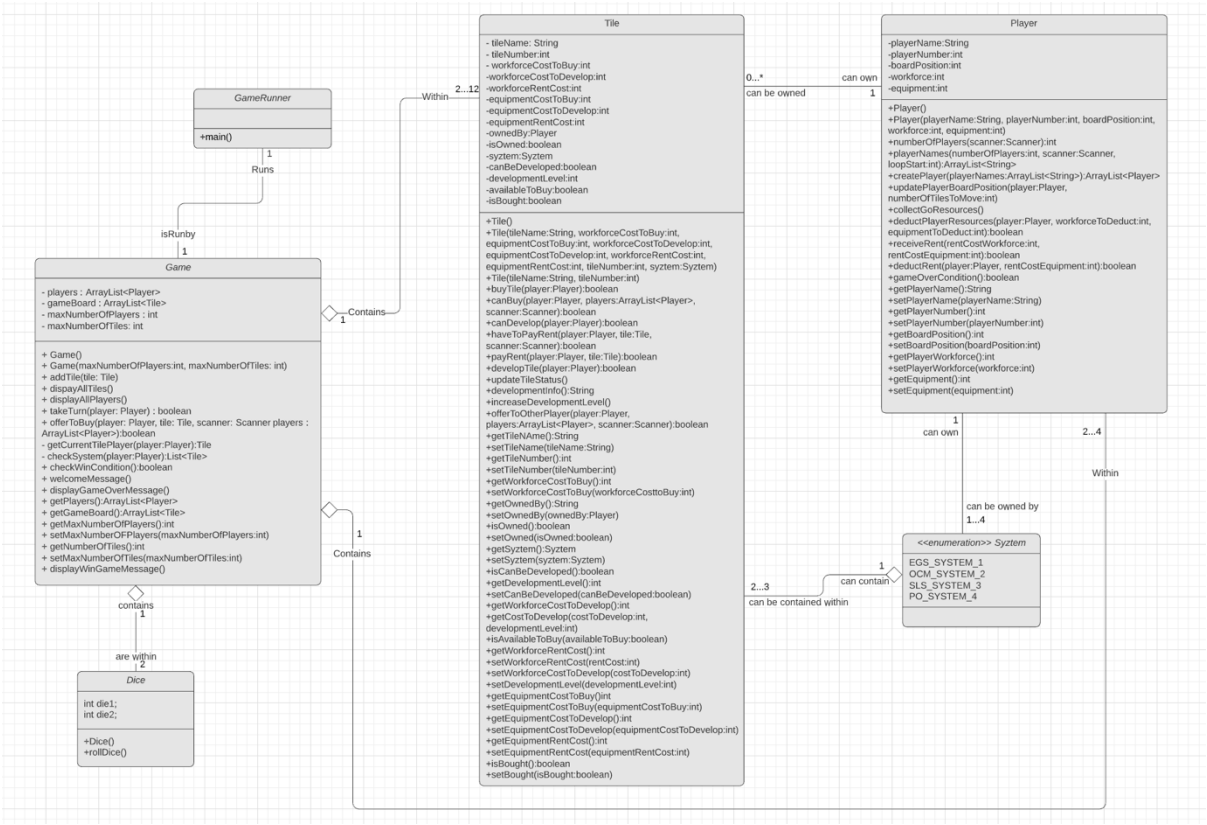

Figure 11 UML Class Diagram final

Testing

Test Plan

Provided below is an excerpt of the Test Plan for the completed system. The approach taken includes a variety of J-Unit code testing and user testing. The J-Unit tests for each of the tested methods includes testing for valid and invalid paths where appropriate, with an emphasis on prioritising the main functions required for normal flow game play. Boundary value analysis and equivalence partitioning testing has been implemented where appropriate. User testing involves selecting test cases and observing the outcomes of each while playing the game and determining if the actual outcomes correlate with the anticipated outcomes as presented to the game players. The test results demonstrate that the game performs as designed.

A full test plan is provided in the appendix.

ID	Use Case Reference	Description of Test	Test Initialisation	Test Inputs	Test Procedure	Expected Results	Passed Y/N
Dice Testing							
1	Roll Dice	Testing the dice roll	Instantiate new dice.	rollDice()>=2, rollDice()<=12	Instantiate a Dice object. Initialise as new Dice(). Pass rollDice method invokation >=2 and rollDice method invokation <=12 as argumnets in assertTrue method.	Dice roll will be within values 2-12	Yes
Game Testing							
8	Select Players	Test Add Player	Instantiate 2 Player objects, Game object, and valid max number of players int and valid max number of tiles int.	Player 1 and Player 2.	Instantiate 2 Player objects and Game object. Initialise both players as new, and Game with maxNumberOfPlayersValid=4 and maxNumberOfTilesValid=12 as game contructor values. Invoke addPlayer() on Game obj. ref. and add each player twice. Perform if statement, invoking getPlayers.size() ==4 on Game obj. ref. Invoke assertTrue with True expected.	AssertTrue returns True	Yes
22	Take Turn	Test Offer To Buy	Instantiate Game object and 2 Player objects, ArrayList of type Player, 1 Tile object, and 1 Scanner object.	Player 1 and Player 2.	Instantiate Game object. Output to console, 'Test offer to buy'. Instantiate and initialise 2 Player objects and add both to the Game by invoking addPlayer() on Game obj. ref. Initialise Player arraylist = game.getPlayers(). Invoke assertEquals with true as expected value, and invoke offerToBuy on Game obj. ref. as actual value with player1, tile1, scanner, players) as param args. Repeat assertEquals.	AssertEquals returns Equal	Yes
30	Take Turn	Test Take Turn	Instantiate Player object, and Game object, and 12 Tile objects.	Player name, board position 0, tiles 1-12, player.	Instantiate Player object, and Game object, and 12 Tile objects. Initialise player name by invoking setPlayerName("Helder") on the player obj. ref. Set player board position by invoking setBoardPosition(0); on the player obj. ref. Add each tile to the game by invoking addTile() on the game obj. ref. with each tile passed as param. arg. Prompt the tester to enter Yes: System.out.println("\n\n\n Enter yes"); Invoke assertTrue on game.takeTurn(player1). Prompt the tester to enter No: System.out.println("\n\n\n Enter yes"); Invoke assertFalse on game.takeTurn(player1).	AssertTrue returns True. AssertFalse returns False.	Yes
Player Testing							
37	Buy Tile/ Pay Rent/ Develop Tile	Test Deduct Player Resources - player has sufficient resources	Instantiate Player and valid player constructor values. Instantiate boolean for expected and unsuccessful resource deduction.	player1 = new Player(playerNameValid, playerNumber, boardPosition, workforce, equipment); boolean expected = true; boolean resourcesDeductedSuccessfully = player1.deductPlayerResources(player1, 100, 100);boolean expected = true; boolean resourcesDeductedSuccessfully = player1.deductPlayerResources(player1, 100, 100);	Instantiate Player and valid player constructor values. Instantiate boolean for expected and successful resource deduction. Initialise expected boolean as true. Initialise successful resource deduction boolean as player1.deductPlayerResources(player1, 100, 100). Invoke assertEquals with expected boolean obj. ref. as expected value and successful resource deduction obj. ref. as actual value. Initialise expected boolean as false. Invoke setEquipment(0) on player1. Initialise unsuccessful resource deduction boolean as player1.deductPlayerResources(player1, 100, 100); Invoke assertEquals with expected boolean obj. ref. as expected value and unsuccessful resource deduction obj. ref. as actual value. Invoke setPlayerWorkforce(0) and setEquipment(100) on player1. Initialise unsuccessful resource deduction boolean as player1.deductPlayerResources(player1, 100, 100); Invoke assertEquals with expected boolean obj. ref. as expected value and unsuccessful resource deduction obj. ref. as actual value. Invoke setEquipment(0) on player1. Initialise unsuccessful resource deduction boolean as player1.deductPlayerResources(player1, 100, 100); Invoke assertEquals with expected boolean obj. ref. as expected value and unsuccessful resource deduction obj. ref. as actual value.	AssertEquals returns Equal	Yes
41	Pay Rent	Test Deduct Rent	Instantiate Player and valid player constructor values, workforce and equipment rent deduction values and successful rent deduction boolean and expected boolean as true	int workforceRentDeduction =100; int equipmentRentDeduction =100; boolean sucessfulRendDeduction=player1.deductRent(player1, workforceRentDeduction, equipmentRentDeduction); boolean expected = true	Instantiate Player and valid player constructor values, workforce and equipment rent deduction values and successful rent deduction boolean and expected boolean as true. Invoke assertEquals with expected boolean as expected value and sucessfulRentDeduction as actual. Invoke assertEquals((workforce-workforceRentDeduction), player1.getPlayerWorkforce()); Invoke assertEquals((equipment-equipmentRentDeduction),player1.getEquipment());	AssertEquals returns Equal	Yes
Tile Testing							
51	Buy Tile	Test Buy Tile	Instantiate Player with valid constructor values, and Tile.	Player, Tile, boolean representing successful tile purchase, unsuccessful purchase, booleans for expected true and false.	Instantiate player and tile and booleans. Invoke tile.buyTile(player), set expectedTrue boolean to true. Invoke assertEquals with expectedtrue boolean as aexpected value and successful purchase boolean as actual values. Instantiate new layer and tile. Assign 200 workforce and equipment purchse price on new tile by invoking setWorkforceCostToBuy and setEquipmentCostToBuy. Set tile purchase unsuccessful boolean to false and expected false boolean to false. Invoke assertEquals with expectedFalse boolean as expected value and unsuccessful purchase boolean as actual value.	AssertEquals returns Equal	Yes
52	Take Turn	Test Can Develop	Instantiate 2 players and 2 tiles.	tile = new Tile(); player = new Player(playerNameValid, playerNumber, boardPosition, workforce, equipment); boolean canDevelopSuccessful = tile.canDevelop(player); canDevelopSuccessful = tile.canDevelop(player); player2 = new Player(playerNameValid, playerNumber, boardPosition, workforce, equipment); tile.setOwnedBy(player2); canDevelopSuccessful = tile.canDevelop(player);	Instantiate new player and tile and canDevelopSuccessful boolean initialised to tile.canDevelop(player); Invoke assertEquals with false as expected value and canDevelopSuccessful boolean as actual values. Set tile ownership to player and canDevelopSuccessful boolean initialised to tile.canDevelop(player); Invoke assertEquals with true as expected value and canDevelopSuccessful boolean as actual values. Instantiate new player with valid constructor values. Set tile ownership to new player and canDevelopSuccessful boolean initialised to tile.canDevelop(player); Invoke assertEquals with true as expected value and canDevelopSuccessful boolean as actual values.	AssertEquals returns Equal	Yes
53	Take Turn	Test Can Buy	Instantiate new player and tile.	tile = new Tile(tileName, workforceCostToBuy, equipmentCostToBuy, workforceCostToDevelop, equipmentCostToDevelop, workforceRentCost, equipmentRentCost, tileNumber, system); player = new Player(playerNameValid, playerNumber, boardPosition, workforce, equipment); boolean canBuy = tile.canBuy(player, playerList, scanner);	Instantiate new player and tile with valid constructor values, and boolean canBuy to tile with player, playerList and scanner as args. Invoke assertEquals with true as expected and canBuy as actual. Set player workforce to 0. Initialise canBuy to tile with player, playerList and scanner as args. Invoke assertEquals with false as expected and canBuy as actual. Set player workforce to 400. Set player equipment to 0. Initialise canBuy to tile with player, playerList and scanner as args. Invoke assertEquals with false as expected and canBuy as actual. Set player workforce to 0. Initialise canBuy to tile with player, playerList and scanner as args. Invoke assertEquals with false as expected and canBuy as actual. Set tile ownership to true. Set player workforce to 400. Set player equipment to 400. Initialise canBuy to tile with player, playerList and scanner as args. Invoke assertEquals with false as expected and canBuy as actual.	AssertEquals returns Equal	Yes

Appendices

Appendix 1 - Test Plan

ID	Use Case Reference	Description of Test	Test Initialisation	Test Inputs	Test Procedure	Expected Results	Passed Y/N
Dice Testing							
1	Roll Dice	Testing the dice roll	Instantiate new dice.	rollDice()>=2, rollDice()<=12	Instantiate a Dice object. Initialise as new Dice(). Pass rollDice method invokation >=2 and rollDice method invokation <=12 as argumnets in assertTrue method.	Dice roll will be within values 2-12	Yes
2	Roll Dice	Testing the Dice default constructor	AssertNotNull performed on Dice object	Dice object	Instantiate a Dice object. Initialise as new Dice(). Pass object reference as argument into assertNotNull().	Dice object will not be null	Yes
Game Testing							
3		Test valid Game Constructor	Instantiate Game object, valid max number of players variable, and valid max number of tiles variable.	maxNumberOfPlayersValid =4, maxNumberOfTilesValid = 12	Instantiate new game object. Pass in maxNumberOfPlayers and maxNumberOfTiles variables as constructor arguments. Invoke assertEquals passing in maxNumberOfPlayersValid ref as expected value, and invoke getMaxNumberOfPlayers() as actual value	Game object will be created sucessfully with valid number of players and valid number of tiles.	Yes
4		Test invalid Game constructor	Instantiate Game object, valid and invalid max number of players variable, and valid and invalid max number of tiles variable.	maxNumberOfPlayersValid =4, maxNumberOfTilesValid = 12, maxNumberOfPlayersInvalid =5, maxNumberOfTilesInvalid = 13. "Max number of players must be between 2-4" error message.	Instantiate IllegalArgumentException. Initialise as assertThrows(IllegalArgumentException.class, ()->. Instantiate new Game object with maxNumberOfPlayersInvalid as expected value, and maxNumberOfTilesValid as actual value. Invoke assertEquals with error message as expected value and invoke .getMessage on IllegalArgumentException reference.	Game object will not be created. Error message will be thrown.	Yes
5		Test invalid Game constructor	Instantiate Game object, valid and invalid max number of players variable, and valid and invalid max number of tiles variable.	maxNumberOfPlayersValid =4, maxNumberOfTilesValid = 12, maxNumberOfPlayersInvalid =5, maxNumberOfTilesInvalid = 13. "Max number of tiles must be between 2-12" error message.	Instantiate IllegalArgumentException. Initialise as assertThrows(IllegalArgumentException.class, ()->. Instantiate new Game object with maxNumberOfPlayersValid as expected value, and maxNumberOfTilesInvalid as actual value. Invoke assertEquals with error message as expected value and invoke .getMessage on IllegalArgumentException reference.	Game object will not be created. Error message will be thrown.	Yes
6		Test Add Tile	Instantiate 12 tiles, Game object, and add tiles to game.	Tiles 1-12.	Instantiate Game object and 12 Tile objects. Add each tile to the game by invoking addTile() on the game object reference. Perform if statement by invoking .getGameBoard() and size()==12 on game obj. ref. Invoke AssertTrue with True expected.	AssertTrue will returns True	Yes
7		Test Add Tile exceeding limit	Instantiate 12 tiles, Game object, and add tiles to game.	Tiles 1-12.	Invoke assertThrows with IllegalArgumentException as parm. arg., and add tile 12 to the game by invoking addTile() on the game obj. ref. twice.	AssertThrows will throw IllegalArgumentException	Yes
8	Select Players	Test Add Player	Instantiate 2 Player objects, Game object, and valid max number of players int and valid max number of tiles int.	Player 1 and Player 2.	Instantiate 2 Player objects and Game object. Initialise both players as new, and Game with maxNumberOfPlayersValid=4 and maxNumberOfTilesValid=12 as game contructor values. Invoke addPlayer() on Game obj. ref. and add each player twice. Perform if statement, invoking getPlayers.size() ==4 on Game obj. ref. Invoke assertTrue with True expected.	AssertTrue returns True	Yes
9	Select Players	Test Add Player	Instantiate 2 Player objects, Game object, and maxNumberOfPlayersValid and maxNumberOfTilesValid.	Player 1 and Player 2.	Instantiate 2 Player objects and Game object. Initialise both players, and Game with maxNumberOfPlayersValid and maxNumberOfTilesValid as game contructor values. Invoke assertThrows(IllegalArgumentException.class) and Invoke addPlayer() on Game obj. ref. and add both players.	AssertThrows throws IllegalArgumentException	Yes
10		Test Display All Tiles	Instantiate Game object and 3 Tile objects.	Tiles 1-3	Instantiate Game object and 3 Tile objetcs. Invoke addTile() on Game obj. ref. and add each tile to the Game. Invoke displayAllTiles() on Game obj. ref. and invoke assertTrue with expected value True.	AssertTrue returns True	Yes
11	Select Players	Test Display All Players	Instantiate Game object and 2 Player objects.	Players 1 and 2.	Instantiate Game object and 2 Player objects. Invoke addPlayer() on Game obj. ref. and add each player to the Game. Invoke displayAllPlayers() on Game obj. ref. and invoke assertTrue with expected value True.	AssertTrue returns True	Yes
12							
13	Select Players	testValidPlayerConstructor	JUnit test - instantiate player	Valid contructor param args	Instantiate player with valid param args. Perform assertEquals on each arg - name, number, boardposition, workforce, and equipment.	Returns pass.	Yes
14	Select Players	testInvalidPlayerConstructor	JUnit test - instantiate player	Invalid player name, IllegalArgException	Instantiate illegalArgumentException. Initialise as assertThrows with		Yes
15	Select Players	testCreatePlayer	Actor responds to game prompt to choose num of players	ArrayList of player names, 2 players	Instantiate player list, initialised with player with playerName list containing 2 players passed as param arg. Assert true to check if list contains player 1 and 2.	Returns pass.	Yes
16	Select Players	Test number of players - valid	Actor responds to game prompt to choose num of players	num of players 2,3,4	Actor enters 2 when prompted for number of pplayers by the game. Repeat test with both 3 and 4 players.	Game proceeds to prompt for player names.Test pass determined by successful creation of corresponding num of players.	Yes

17	Select Players	Test number of players - invalid	Actor responds to game prompt to choose num of players	num of players 1 and 5	Actor enters 1 when prompted for number of players by the game. Repeat test with 5 players.	Game outputs error message to console informing of incorrect number of players and prompts actor to try again.	Yes
18	Select Players	Test player name creation	Game prompts actor to enter player names	PlayerName1, PlayerName2	When prompted by the game, actor enters PlayerNames 1 and 2.	Game outputs to console palyer information, including accurate player names.	Yes
19	Select Players	Test display of all tiles	Actor runs game	None	Actors initiates run game	All tiles are displayed to the console, including full contructor details	Yes
20	Select Players	Test display of all players	Actor inputs required number of players and names	Quantity of players - 2, PlayerNames : PlayerName1, PlayerName2	Actor inputs 2 when prompted for number of players. Actor inputs 'playername1', and 'playername2' when prompted for playernames.	Console displays player information including player number, playername, board position, and resources	Yes
21	Select Players	Game displays welcome message	Actor has entered playernames	Playernames - enter	Actor enters playernames and presses enter key	Welcome message displays on console	Yes
22	Take Turn	Test Offer To Buy	Instantiate Game object and 2 Player objects, ArrayList of type Player, 1 Tile object, and 1 Scanner object.	Player 1 and Player 2.	Instantiate Game object. Output to console, 'Test offer to buy'. Instantiate and initialise 2 Player objects and add both to the Game by invoking addPlayer() on Game obj. ref. Initialise Player arraylist = game.getPlayers(). Invoke assertEquals with true as expected value, and invoke offerToBuy on Game obj. ref. as actual value with player1, tile1, scanner, players) as param args. Repeat assertEquals.	AssertEquals returns Equal	Yes
23	Take Turn	Test Get Current Player Tile	Instantiate Game object, 1 Player object, and 1 Tile object.	Tile object, game object, player object.	Instantiate game object, player object and tile object. Invoke setBoardPosition(0) on Player obj. ref. Invoke addTile(tile obj. ref.) on Game obj. ref. Invoke assertEquals with tile obj. ref. as expected value and actual value as game.getCurrentPlayerTile() with Player obj. ref. as param. arg.	AssertEquals returns Equal	Yes
24	Take Turn	Test Check System	Instantiate Game object, 1 Player object, and 12 Tile objects.	Player obj, tiles 1-12	Instantiate Game object and 12 Tile objects. Set the player name by nvoking .setPlayerName on Player obj. ref. Add Tile1 to the Game by invoking the addTile () on the Game obj. ref. Invoke setOwned on 1 Tile obj. Invoke setOwnedBy on same Tile obj. and pass player1 as param. arg. Invoke setCanBeDeveloped on Tile obj. with param. arg. set to True, and invoke setDevelopmentLevel on Tile obj. with level set as 1. Repeat for all tiles. Instantiate List of type Tile and invoke checkSystem() on Game obj. rfe, passing player1 as param. arg. Invoke assertTrue on List with argument == 10. Invoke assertEquals() with tile as expected value and List.get(0) as actual.	AssertTrue returns True. AssertEquals returns Equal	Yes.
25	Take Turn	Test Check Win Condition	Instantiate Game object and 2 Tile objects.	Tile1 and Tile 2	Instantiate Game and Tile objects. Invoke setDevelopmentLevel(4) on both Tile objects. Invoke addTile on Game obj. and add both Tiles. Invoke assertTrue() on game.checkWinCondition(). Invoke setDevelopmentLevel(1) on Tile1. Invoke assertFalse() on game.checkWinCondition().	AssertTrue returns True. AssertFalse returns False.	Yes
26	Take Turn	Test Get Players	Instantiate ArrayList of type Player and Game obj.	Player Arraylist and Game obj.	Instantiate ArrayList of type Player and Game obj. Invoke assertEquals with the ArrayList as expected and game.getPlayers() as actual.	AssertEquals returns Equal	Yes
27	Take Turn	Test Set Get Max Number Of Players Valid	Instantiate Game object and max number of players valid var	max Number Of Players Valid var	Invoke setMaxNumberOfPlayers(maxNumberOfPlayersValid) on Game obj. ref. Invoke assertEquals with maxNumberOfPlayersValid as expected value and testSetGetMaxNumberOfPlayersInvalid as actual value.	AssertEquals returns Equal	Yes
28	Take Turn	Test Set Get Max Number Of Players Invalid	Instantiate Game object,max number of players invalid var, and IllegalArgumentException	IllegalArgumentException. Max number of players invalid var. Error message - "Max number of players must be between 2-4"	Instantiate Game object. Instantiate IllegalArgumentException and initialise as IllegalArgumentException.class, () ->{ game.setMaxNumberOfPlayers(maxNumberOfPlayersInvalid); Invoke assertEquals with error message as expected value and invoke getMessage() on Ill as actual value.egalArgumentException	AssertEquals returns Equal	Yes
29	Take Turn	Test Set Get Max Number Of Tiles Valid	Instantiate Game object,max number of tiles valid var	maxNumberOfTilesValid	Instantiate Game obj. Invoke setMaxNumberOfTiles with maxNumberOfTilesValid as param. arg. Invoke assertEquals with maxNumberOfTilesValid as expected value and game.getMaxNumberOfTiles() as actual value.	AssertEquals returns Equal	Yes
30	Take Turn	Test Take Turn	Instantiate Player object, and Game object, and 12 Tile objects.	Player name, board position 0, tiles 1-12, player.	Instantiate Player object, and Game object, and 12 Tile objects. Initialise player name by invoking setPlayerName("Helder") on the player obj. ref. Set player board position by invoking setBoardPosition(0); on the player obj. ref. Add each tile to the game by invoking addTile() on the game obj. ref. with each tile passed as param. arg. Prompt the tester to enter Yes: System.out.println("\n\n\n Enter yes"); Invoke assertTrue on game.takeTurn(player1). Prompt the tester to enter No: System.out.println("\n\n\n Enter yes"); Invoke assertFalse on game.takeTurn(player1).	AssertTrue returns True. AssertFalse returns False.	Yes
31	Take Turn	Take Turn	Game prompts player to take turn - Yes	Player names have been entered	Player selects enter after inputting playernames	Game prompts player to continue playing. Player selects yes.	Game displays players current position, dice roll, and players new position.
32	Take Turn	Take Turn	Game prompts player to take turn - No	Player names have been entered	Player selects enter after inputting playernames	Game prompts player to continue playing. Player selects No.	Game displays message informing of players choice to no longer continue. Game ends for all players.
Player Testing							

33	Select Players	Test Valid Player Constructor	Instantiate Player object, valid player name, number, board position, workforce, and equipment.	playerNameValid, playerNumber, boardPosition, workforce, equipment	Instantiate Player obj, valid player name, player number, board position, workforce, and equipment. Invoke assertEquals(playerNameValid, player1.getPlayerName()); assertEquals(playerNumber, player1.getPlayerNumber()); assertEquals(boardPosition, player1.getBoardPosition()); assertEquals(workforce, player1.getPlayerWorkforce()); assertEquals(equipment, player1.getEquipment());	AssertEquals returns Equal	Yes
34	Select Players	Test Invalid Player Constructor	Instantiate IllegalArgumentException, Player object, invalid player name, number, board position, workforce, and equipment.	playerNameInvalid, playerNumber, boardPosition, workforce, equipment. Error message "Player name cannot be blank"	Instantiate IllegalArgumentException. Initialise as assertThrows with IllegalArgumentException.class, and player obj. ref. initialised as new player. Invoke assertEquals with error message as expected value and getMessage() invokation on IllegalArgumentException obj. ref. as actual value.	AssertEquals returns Equal	Yes
35	Select Players	Test Create Player	Instantiate ArrayList of type Player, 2 Players.	ArrayList of Players obj. ref, player1 and player 2 obj. ref.	Instantiate ArrayList of type Player, and 2 players. Add players to ArrayList. Check size of player ArrayList and that list contains both players by invoking .size()==2 and .contains(player1) and .contains(player2) on list. Invoke assertEquals(true)	AssertEquals returns Equal	Yes
36	Select Players	Test Num Of Players	Instantiate int to hold expected number of players. Instantiate Scanner obj.	Scanner obj ref, numbmer of players 1, 2, 5.	Instantiate int to hold expected number of players. and actual number of players. Instantiate Scanner obj. Output to console, message to prompt tester to select corresponding number of players, 1, 5 , 2. Invoke numOfPlayers(scanner obj. ref.) on Player and initialise to actua number of players int ref. Invoke assertEquals with expectedNumberOfPlayers as expected value and numberOfPlayers as actual value.	AssertEquals returns Equal	Yes
37	Buy Tile/ Pay Rent/ Develop Tile	Test Deduct Player Resources - player has sufficient resources	Instantiate Player and valid player constructor values. Instantiate boolean for expected and unsuccessful resource deduction.	player1 = new Player(playerNameValid, playerNumber, boardPosition, workforce, equipment); boolean expected = true; boolean resourcesDeductedSuccessfully = player1.deductPlayerResources(player1, 100, 100);boolean expected = true; boolean resourcesDeductedSuccessfully = player1.deductPlayerResources(player1, 100, 100);	Instantiate Player and valid player constructor values. Instantiate boolean for expected and successful resource deduction. Initialise expected boolean as true. Initialise successful resource deduction boolean as player1.deductPlayerResources(player1, 100, 100). Invoke assertEquals with expected boolean obj. ref. as expected value and successful resource deduction obj. ref. as actual value. Initialise expected boolean as false. Invoke setEquipment(0) on player1. Initialise unsuccessful resource deduction boolean as player1.deductPlayerResources(player1, 100, 100); Invoke assertEquals with expected boolean obj. ref. as expected value and unsuccessful resource deduction obj. ref. as actual value. Invoke setPlayerWorkforce(0) and setEquipment(100) on player1. Initialise unsuccessful resource deduction boolean as player1.deductPlayerResources(player1, 100, 100); Invoke assertEquals with expected boolean obj. ref. as expected value and unsuccessful resource deduction obj. ref. as actual value. Invoke setEquipment(0) on player1. Initialise unsuccessful resource deduction boolean as player1.deductPlayerResources(player1, 100, 100); Invoke assertEquals with expected boolean obj. ref. as expected value and unsuccessful resource deduction obj. ref. as actual value.	AssertEquals returns Equal	Yes

38	Buy Tile/ Pay Rent/ Develop Tile	Test Deduct Player Resources - player has insufficient equipment	Instantiate Player and valid player constructor values. Instantiate boolean for expected and unsuccessful resource deduction.	expected = false;
				player1.setEquipment(0);
				boolean resourcesDeductedUnsuccessfully = player1.deductPlayerResources(player1, 100, 100);
					Instantiate Player and valid player constructor values. Instantiate boolean for expected and unsuccessful resource deduction. Set player equipment value as 0 by invoking setEquipment(0) on player obj. ref. Initialise expected boolean as false. Initialise unsuccessful resource deduction as player1.deductPlayerResources(player1, 100, 100). Invoke assertEquals with expected boolean obj. ref. as expected value and unsuccessful resource deduction obj. ref. as actual value.	AssertEquals returns Equal	Yes

39	Buy Tile/ Pay Rent/ Develop Tile	Test Deduct Player Resources - player has insufficient equipment and workforce	Instantiate Player and valid player constructor values. Instantiate boolean for expected and unsuccessful resource deduction.	player1.setPlayerWorkforce(0);
				player1.setEquipment(100);
				resourcesDeductedUnsuccessfully = player1.deductPlayerResources(player1, 100, 100);
					Instantiate Player and valid player constructor values. Instantiate boolean for expected and unsuccessful resource deduction. Set player equipment value as 0 by invoking setEquipment(0) on player obj. ref. Initialise expected boolean as false. Initialise unsuccessful resource deduction as player1.deductPlayerResources(player1, 100, 100). Invoke assertEquals with expected boolean obj. ref. as expected value and unsuccessful resource deduction obj. ref. as actual value.	AssertEquals returns Equal	Yes

40	Buy Tile/ Pay Rent/ Develop Tile	Test Deduct Player Resources - player has insufficient workforce	Instantiate Player and valid player constructor values. Instantiate boolean for expected and unsuccessful resource deduction.	player1.setEquipment(0); resourcesDeductedUnsuccessfully = player1.deductPlayerResources(player1, 100, 100);	Instantiate Player and valid player constructor values. Instantiate boolean for expected and unsuccessful resource deduction. Set player equipment value as 0 by invoking setEquipment(0) on player obj. ref. Initialise expected boolean as false. Initialise unsuccessful resource deduction as player1.deductPlayerResources(player1, 100, 100). Invoke assertEquals with expected boolean obj. ref. as expected value and unsuccessful resource deduction obj. ref. as actual value.	AssertEquals returns Equal	Yes
41	Pay Rent	Test Deduct Rent	Instantiate Player and valid player constructor values, workforce and equipment rent deduction values and successful rent deduction boolean and expected boolean as true	int workforceRentDeduction =100; int equipmentRentDeduction =100; boolean sucessfulRendDeduction=player1.deductRent(player1, workforceRentDeduction, equipmentRentDeduction); boolean expected = true	Instantiate Player and valid player constructor values, workforce and equipment rent deduction values and successful rent deduction boolean and expected boolean as true. Invoke assertEquals with expected boolean as expected value and sucessfulRentDeduction as actual. Invoke assertEquals((workforce-workforceRentDeduction), player1.getPlayerWorkforce()); Invoke assertEquals((equipment-equipmentRentDeduction),player1.getEquipment());	AssertEquals returns Equal	Yes

42	Pay Rent	Test Receive Rent	Instantiate ints for workforce and equipment rent values, player, booleans for expected value and successful rent deduction.	int workforceRentGain =100; int equipmentRentGain =100; boolean expected = true; boolean deductRentSuccessfully = player1.recieveRent(workforceRentGain, equipmentRentGain);	Instantiate workforce and equipment rent gain variables, and booleans. Invoke assertEquals with expected int as expected value, and deductRentSuccessfully int as actual value. Invoke AssertEquals with workforce + workforceRentGain as expected value, and player1.getPlayerWorkforce() as actual value. Invoke AssertEquals with equipment+ equipmentRentGain as expected value, and player1.getEquipment() as actual value.	AssertEquals returns Equal	Yes
43	Select Players	Test Default Constructor	Instantiate Player.	Player player = new Player();	Instantiate Player and pass Player obj. ref. as value in assertNotNull invokation.	AssertNotNull returns not null	Yes
44	Select Players	Test Player Names	Instantiate ArrayList of type String, Player, number of players, Scanner.	numberOfPlayers, scanner, 1	Instantiate ArrayList of typeString and Player and initialise arrayList as Player.playerNames(numberOfPlayers, scanner, 1); Invoke assertEquals with numberOfPlayers as exected value and testPlayerNames.size() as actual value.	AssertEquals returns Equal	Yes
45	Take Turn	Test Update Player Board Position	Instantiate board position, player	boardPosition, player1, new board position 12	Invoke assertEquals with boardPosition as expected value and player1.getBoardPosition() as actual value. Invoke updatePlayerBoardPosition(player1, 10) on player and invoke assertEquals again on updated player. assertEquals(10, player1.getBoardPosition()); Reset player board position. Update board position to outsde of permitted range, 12. Invoke assertEquals with board position as expected value and player1.getBoardPosition() as actual value.	AssertEquals returns Equal	Yes
46	Take Turn	Test Game Over Condition	Instantiate boolean for expected, player	boolean expected = false; expected = true; player1	Instantiate boolean for expected as false. Invoke assertEquals with expect aboolean as expected value and expected, player1.gameOverCondition() as actual value. Set expected to true. Set player equipment resource to negative value to trigger game over condition, i.e., player1.setEquipment(-10); Invoke assertEquals with expected boolean as expected value and expected, player1.gameOverCondition() as actual value. Set player equipment resource to positive value, i.e. 100, and set player workforce to negative value to trigger game over condition, i.e., player1.setPlayerWorkforce(-10); Invoke assertEquals with expected boolean as expected value and expected, player1.gameOverCondition() as actual value. Set player equipment resource to negative value to trigger game over condition, i.e., player1.setEquipment(-10); Invoke assertEquals with expected boolean as expected value and expected, player1.gameOverCondition() as actual value.	AssertEquals returns Equal	Yes
47	Take Turn	Test Set Player Number	Instantiate player, playerNumber	playerNumber, 4, 5	Invoke assertEquals with playerNumber as expected value, and player1.getPlayerNumber() as actual value. Set playerNumber to 4 by invoking .setPlayerNumber(4);Invoke assertEquals with 4 as expected value and player1.getPlayerNumber() as actual value. Invoke assertThrows with IllegalArgument.class as expected value and player1.setPlayerNumber(5) as actual value. Repeat assertThrows with player1.setPlayerNumber(0) as actual value.	AssertEquals returns Equal. AssertThrows throws IllegalArgument Exception.	Yes
Tile Testing							
48		Test Tile Constructor All Arguments	Instantiate Tile with valid constructor values	tileName, workforceCostToBuy, equipmentCostToBuy, workforceCostToDevelop,	Instantiate Tile with valid constructor arguments. Invoke assertEquals for each of the following: assertEquals(tileName, tile.getTileName()); assertEquals(workforceCostToBuy, tile.getWorkforceCostToBuy()); assertEquals("Nobody", tile.getOwnedBy()); assertEquals(false, tile.isOwned()); assertEquals(tileNumber, tile.getTileNumber()); assertEquals(system, tile.getSyztem()); assertEquals(false, tile.isCanBeDeveloped()); assertEquals(0, tile.getDevelopmentLevel()); assertEquals(workforceCostToDevelop, tile.getWorkforceCostToDevelop()); assertEquals(true, tile.isAvailableToBuy()); assertEquals(workforceRentCost, tile.getWorkforceRentCost()); assertEquals(equipmentCostToBuy, tile.getEquipmentCostToBuy()); assertEquals(equipmentCostToDevelop, tile.getEquipmentCostToDevelop()); assertEquals(equipmentRentCost, tile.getEquipmentRentCost()); assertEquals(false, tile.isBought());	AssertEquals returns Equal	Yes
49		Test Tile Constructor Less Arguments	Instantiate Tile with valid constructor values	tileName, tileNumber	Instantiate Tile with valid constructor arguments. Invoke assertEquals for each of the following: assertEquals(tile	AssertEquals returns Equal	Yes
50		Test Default Constructor	Instantiate Tile with valid constructor values	Tile	assertNotNull with Tile as param. value.	AssertNotNull returns not null	Yes
51	Buy Tile	Test Buy Tile	Instantiate Player with valid constructor values, and Tile.	Player, Tile, boolean representing successful tile purchase, unsuccessful purchase, booleans for expected true and false.	Instantiate player and tile and booleans. Invoke tile.buyTile(player), set expectedTrue boolean to true. Invoke assertEquals with expectedtrue boolean as aexpected value and successful purchase boolean as actual values. Instantiate new layer and tile. Assign 200 workforce and equipment purchse price on new tile by invoking setWorkforceCostToBuy and setEquipmentCostToBuy. Set tile purchase unsuccessful boolean to false and expected false boolean to false. Invoke assertEquals with expectedFalse boolean as expected value and unsuccessful purchase boolean as actual value.	AssertEquals returns Equal	Yes
52	Take Turn	Test Can Develop	Instantiate 2 players and 2 tiles.	tile = new Tile(); player = new Player(playerNameValid, playerNumber, boardPosition, workforce, equipment); boolean canDevelopSuccessful = tile.canDevelop(player); canDevelopSuccessful = tile.canDevelop(player); player2 = new Player(playerNameValid, playerNumber, boardPosition, workforce, equipment); tile.setOwnedBy(player2); canDevelopSuccessful = tile.canDevelop(player);	Instantiate new player and tile and canDevelopSuccessful boolean initialised to tile.canDevelop(player); Invoke assertEquals with false as expected value and canDevelopSuccessful boolean as actual values. Set tile ownership to player and canDevelopSuccessful boolean initialised to tile.canDevelop(player); Invoke assertEquals with true as expected value and canDevelopSuccessful boolean as actual values. Instantiate new player with valid constructor values. Set tile ownership to new player and canDevelopSuccessful boolean initialised to tile.canDevelop(player); Invoke assertEquals with true as expected value and canDevelopSuccessful boolean as actual values.	AssertEquals returns Equal	Yes

53	Take Turn	Test Can Buy	Instantiate new player and tile.	tile = new Tile(tileName, workforceCostToBuy, equipmentCostToBuy, workforceCostToDevelop, equipmentCostToDevelop, workforceRentCost, equipmentRentCost, tileNumber, system);
				player = new Player(playerNameValid, playerNumber, boardPosition, workforce, equipment);
				boolean canBuy
				= tile.canBuy(player, playerList, scanner);
					Instantiate new player and tile with valid constructor values, and boolean canBuy to tile with player, playerList and scanner as args. Invoke assertEquals with true as expected and canBuy as actual. Set player workforce to 0. Initialise canBuy to tile with player, playerList and scanner as args. Invoke assertEquals with false as expected and canBuy as actual. Set player workforce to 400. Set player equipment to 0. Initialise canBuy to tile with player, playerList and scanner as args. Invoke assertEquals with false as expected and canBuy as actual. Set player workforce to 0. Initialise canBuy to tile with player, playerList and scanner as args. Invoke assertEquals with false as expected and canBuy as actual. Set tile ownership to true. Set player workforce to 400. Set player equipment to 400. Initialise canBuy to tile with player, playerList and scanner as args. Invoke assertEquals with false as expected and canBuy as actual.	AssertEquals returns Equal	Yes

54	Pay Rent	Test Pay Rent	Instantiate 2 players, tile, and boolean.	player = new Player(playerNameValid, playerNumber, boardPosition, workforce, equipment);
				player2 = new Player(playerNameValid, playerNumber, boardPosition, workforce, equipment);
				, tile = new Tile();boolean payRentSuccessful = tile.payRent(player, tile);
					Instantiate 2 players with valid constructor values and new tile. Set tile workforcerentcost and tile equipmentRentCost to 100/ Set tile owenedBy player2. Set payRentSuccessful = tile.payRent(player, tile); Invoke assertEquals with workforce+100 as expected value and player2.getPlayerWorkforce() as actual value. Invoke assertEquals with workforce+100 as expected value and player2.getPlayerEquipment() as actual value. Invoke assertEquals with workforce-100 as expected value and player2.getPlayerWorkforce() as actual value. Invoke assertEquals with workforce-100 as expected value and player2.getPlayerEquipment() as actual value. Invoke assertEquals with true as expected value and payRentSuccessful as actual value.	AssertEquals returns Equal	Yes

55	Develop Tile	Test Develop Tile	Instantiate new player and tile and 2 booleans.	player = new Player(playerNameValid, playerNumber, boardPosition, workforce, equipment); tile = new Tile(); boolean developTileSuccessful = tile.developTile(player); boolean developTileUnsuccessful = tile.developTile(player);	Instantiate new player with valid contructor values and ew tile. Set tile workforceCostToDevelop to 100 and equipmentCostToDevelop to 100. Initialise developTileSuccessful boolean = tile.developTile(player); Invoke assertEquals with true as expecetd and developTileSuccessful as actual values. Set tile workforceCostToDevelop to 1000 and equipmentCostToDevelop to 1000. Initialise developTileUnsuccessful boolean = tile.developTile(player); Invoke assertEquals with false as expected and developTileUnsuccessfulas actual values.	AssertEquals returns Equal	Yes

56	Take Turn	Test Have To PayRent	Instantiate 2 players, tile, and boolean.	player = new Player(playerNameValid, playerNumber, boardPosition, workforce, equipment);
				player2 =
				new Player("joe", playerNumber,boardPosition,workforce,equipment); haveToPayRent =tile.haveToPayRent(player, tile, scanner);
					Instantiate 2 players with valid constructor values and new tile. Set tile owned by player 2 and setOwned as true. Print to console "Enter 'yes' into the console". Set haveToPayRent =tile.haveToPayRent(player, tile, scanner); Invoke assertEquals with true as expected value and haveToPayRent as actual value. Print to console "Enter 'no' into the console". Set haveToPayRent =tile.haveToPayRent(player, tile, scanner); Invoke assertEquals with false as expected value and haveToPayRent as actual value. Set tile owned by player 1. Set haveToPayRent =tile.haveToPayRent(player, tile, scanner); Invoke assertEquals with false as expected value and haveToPayRent as actual value. Set tile owned as false. Set haveToPayRent =tile.haveToPayRent(player, tile, scanner); Invoke assertEquals with false as expected value and haveToPayRent as actual value.	AssertEquals returns Equal	Yes

57	Take Turn	Test Can Be Developed And Set Can Be Developed	Instantiate new tile	tile=new Tile();	Instantiate new tile. Invoke assertEquals with false as expected value and tile.isCanBeDeveloped() as actual value. Invoke updateTileDevelopStatus on tile with empty arg. Invoke assertEquals with true as expected value and tile.isCanBeDeveloped() as actual value. Invoke setDevelopmentLevel on tile with arg value 5. Invoke assertEquals with false as expected value and tile.isCanBeDeveloped() as actual value. Invoke setCanBeDevelopedon tile with arg value false. Invoke assertEquals with false as expected value and tile.isCanBeDeveloped() as actual value.	AssertEquals returns Equal	Yes
58	Buy Tile	Test Set Get Bought	Instantiate new tile	tile = new Tile();	Instantiate new tile. Invoke assertEquals with false as expecetd value and tile.isBought() as actual value. Update tile status by invoking tile.setBought(true); Invoke assertEquals with ture as expected value and tile.isBought() as actual value.	AssertEquals returns Equal	Yes
59	Pay Rent	Test Set Get Equipment Rent Cost	Instantiate new tile	tile = new Tile(tileName, workforceCostToBuy, equipmentCostToBuy, workforceCostToDevelop, equipmentCostToDevelop, workforceRentCost, equipmentRentCost, tileNumber, system);	Instantiate new tile with valid constructor values. Invoke assertEquals with equipmentRentCost as expected value and tile.getEquipmentRentCost() as actual value. Set tile equipment rent cost as 500. Invoke assertEquals with 500 as exoected value and tile.getEquipmentRentCost() as actual value.	AssertEquals returns Equal	Yes

60	Pay Rent	Test Set Get Workforce Rent Cost	Instantiate new tile	tile = new Tile(tileName, workforceCostToBuy, equipmentCostToBuy, workforceCostToDevelop, equipmentCostToDevelop, workforceRentCost, equipmentRentCost, tileNumber, system);	Instantiate new tile with valid constructor values. Invoke assertEquals with workforceRentCost as expected value and tile.getWorkforceRentCost() as actual value. Set tile workforce rent cost as 500. Invoke assertEquals with 500 as exoected value and tile.getWorkforceRentCost() as actual value.	AssertEquals returns Equal	Yes
61	Take Turn	Test Set Get Available To Buy	Instantiate new tile	tile = new Tile();	Instantiate new tile. Invoke assertEquals with false as expected value, and tile.isAvailableToBuy() as actual value. Set tile available to buy as true and invoke assertEquals with trueas expected value, and tile.isAvailableToBuy() as actual value.	AssertEquals returns Equal	Yes
62	Take Turn	Test Set Get Workforce Cost To Buy	Instantiate new tile	tile = new Tile();	Instantiate new tile. Set tile cost to buy. Invoke assertEquals with workforce cost to buy as expected value and tile.getWorkforceCostToBuy() as actual value.	AssertEquals returns Equal	Yes
63	Take Turn	Test Set Get Tile Number	Instantiate new tile	tile = new Tile();	Instantiate new tile. Set tile number. Invoke assertEquals with tileNumber as expected value and tile.getTileNumber() as expected value.	AssertEquals returns Equal	Yes
64	Take Turn	Test Set Get Tile Name	Instantiate new tile	tile = new Tile();	Instantiate new tile. Set tile name. Invoke assertEquals with tile name as expected value and tile.getTileName() as actual value.	AssertEquals returns Equal	Yes
65	Take Turn	Test Set Cost To Develop	Instantiate new tile	tile = new Tile();	Instantiate new tile. Set development level to 1. Set equipment cost to develop and workforce cost to develop to 0. Set cost to develop as tile.getWorkforceCostToDevelop(), tile.getEquipmentCostToDevelop(), tile.getDevelopmentLevel(). Invoke assertEquals with 100 as expected value and tile.getEquipmentCostToDevelop() as actual value. Invoke assertEquals with 100 as expected value and tile.getWorkforceCostToDevelop() as actual value. Set development level to 4. Set equipment cost to develop and workforce cost to develop to 0. Set cost to develop as tile.getWorkforceCostToDevelop(), tile.getEquipmentCostToDevelop(), tile.getDevelopmentLevel(). Invoke assertEquals with 800as expected value and tile.getEquipmentCostToDevelop() as actual value. Invoke assertEquals with 800 as expected value and tile.getWorkforceCostToDevelop() as actual value.	AssertEquals returns Equal	Yes

66	Buy Tile	Test Offer To Other Player	Instantiate new tile, 2 new players, player list, 2 boolean	tile = new Tile();
				player2 = new Player(); player = new Player(); playerList.add(player2);
				playerList.add(player); boolean yesInput = tile.offerToOtherPlayer(player, playerList, scanner);
				boolean noInput = tile.offerToOtherPlayer(player,
				playerList, scanner);
					Instantiate new tile, 2 players, and add to playerlist. Output to console 'Enter yes'.
					Set yesIput boolean to tile.offerToOtherPlayer(player, playerList, scanner);
					Invoke assertEquals with true as expected value, and yesInput as actual value.
					Output to console 'Enter yes'.
					Set yesIput boolean to tile.offerToOtherPlayer(player2, playerList, scanner);
					Invoke assertEquals with true as expected value, and yesInput as actual value.
					Output to console 'Enter No'.
					Set noInput boolean to tile.offerToOtherPlayer(player, playerList, scanner);
					Invoke assertEquals with false as expected value, and noInput as actual value.
						AssertEquals returns Equal	Yes

67		Test Development Info	Instantiate new tile, string	tile = new Tile(); String testString = String.format("%-20s %-20d %-20d %-20d %-20d\n", tileName, tileNumber,	Instantiate new tile. Set tilename, tilenumber, development level(1), workforcecostto develop and equipmentcosttodevelop. Instantiate String initialised as player. Invoke assertEquals with String value as expected and tile.developmentInfo() as actual value.	AssertEquals returns Equal	Yes
68	Take Turn	Test Increase Dev Level Invalid	Instantiate new tile	tile = new Tile();	Instantiate new tile. Set development level as 4. Invoke increase development level () on tile. Invoke assertEquals with 4 as expected value and actual as tile.getDevelopmentLevel().	AssertEquals returns Equal	Yes
69		Test Set Get Syztem	Instantiate new tile, system	tile = new Tile();, system	Instantiate new tile. Set Syztem with tile as value. Invoke assertEquals with system as expected value and tile.getSyztem() as actual value.	AssertEquals returns Equal	Yes

Appendix 2 - Weekly Minutes

Minutes for Group **30**

Week commencing **2**

Date of this minute **27/01/21**

The following team members were present

Name (printed/typed)	Signature
Gareth	Present
Helder	Present
Joe	Present
Daniel	Present

Discussion Points:

- Coding structure of game (OOP, if statements, loops)
- The software model approach, iterative, prototyping
- What a first prototype might look like
- Best day/time for weekly meetings
- Winner / for the greater good

Actions Planned (Briefly list the actions required of each team member for the next week.)

Action for all members:

- Re read the game requirements
- Research UML diagrams and share with group
- Consider possible USE CASES in advance of next meeting

    - Register players
    - Roll dice
    - Take turn
    - Buy component
        - Offer to others
    - Develop component
        - Basic
        - Major
    - Pay fees
    - Collect resources
    - Current state of play?
    - Launch

Minutes for Group **30**
Week commencing **3**
Date of this minute **03/02/21**

The following team members were present

Name (printed/typed)	Signature
Gareth	Present
Helder	Present
Joe	Present
Daniel	Present

Discussion Points:

- UML sequence diagram on Lucid Chart
- UML use cases descriptions
- Game rules

Actions Planned (Briefly list the actions required of each team member for the next week.)

Action for all members:

- Participate in the development of use case descriptions
- Meet on Friday the 5th Feb
- Meet advisor on Tuesday Week4 to discuss use case drafts

Minutes for Group **30**
Week commencing **3**
Date of this minute **05/02/21**

The following team members were present

Name (printed/typed)	Signature
Gareth	Present
Helder	Present
Joe	Present
Daniel	Present

Discussion Points:

- Finalising use case descriptions for advisory meeting 9th Feb Tuesday
- Class and sequence diagram differences investigated
- Game board style/layout investigated and amended

Actions Planned (Briefly list the actions required of each team member for the next week.)

Action for all members:

- Meet advisors 9th Feb
- Read Mark Priestly article on sequence and class diagrams
- Start thinking about how game will translate to code

Minutes for Group **30**
Week commencing **4**
Date of this minute **09/02/21**

The following team members were present

Name (printed/typed)	Signature
Gareth	Present
Helder	Present
Joe	Present
Daniel	Present

Discussion Points:

- Team met with an advisor on canvas, discussed our use case descriptions
- Discussed next steps, i.e. how to represent in sequence diagrams

Actions Planned (Briefly list the actions required of each team member for the next week.)

Action for all members:

- Meet on the 11th Feb to begin sequence diagram drawing
- Review Mark Priestly article on sequence and class diagrams

Minutes for Group 30

Week commencing 4

Date of this minute 11/02/21

The following team members were present

Name (printed/typed)	Signature
Gareth	Present
Helder	Present
Joe	Present
Daniel	Present

Discussion Points:

- Team met and began to develop sequence diagrams (Select Players & Take Turn)
- Reviewed use case descriptions and amended some inconsistent language

Actions Planned (Briefly list the actions required of each team member for the next week.)

Action for all members:

- Meet Friday 12th to continue

Minutes for Group **30**
Week commencing **5**
Date of this minute **16/02/21**

The following team members were present

Name (printed/typed)	Signature
Gareth	Present
Helder	Present
Joe	Present
Daniel	Present

Discussion Points:

- Team met with advisor Moshen to continue discussions regarding finalised sequence diagrams. Format and content feedback positive from Moshen.
- Developed use case class diagrams
- Implementation discussion

Actions Planned (Briefly list the actions required of each team member for the next week.)

Action for all members:

- Meet again over the weekend to continue working

Minutes for Group **30**
Week commencing ***5***
Date of this minute **18/02/21**

The following team members were present

Name (printed/typed)	Signature
Gareth	Present
Helder	Present
Joe	Present
Daniel	Present

Discussion Points:

- Team met and continued working on sequence diagrams, use case descriptions and class diagrams.
- Develop element Use Case

Actions Planned (Briefly list the actions required of each team member for the next week.)

Action for all members:

- Meet again over the weekend to continue working

Minutes for Group **30**
Week commencing **6**
Date of this minute **24/02/21**

The following team members were present

Name (printed/typed)	Signature
Gareth	Present
Helder	Present
Joe	Present
Daniel	Present

Discussion Points:

- Sequence diagrams finalised
- Use case descriptions reviewed against sequence diagrams and amended
- Class diagrams reviewed and code walk through discussion had
- Variables and methods checked within class diagrams and colour coordinated to sequence diagrams

Actions Planned (Briefly list the actions required of each team member for the next week.)

Action for all members:

- Discuss coding options within next meeting

Minutes for Group **30**
Week commencing **8**
Date of this minute **11/03/21**

The following team members were present

Name (printed/typed)	Signature
Gareth	Present
Helder	Present
Joe	Present
Daniel	Present

Discussion Points:

- Added functionality to program code
    - Buy element method
    - Checks for system & tile ownership

Actions Planned:

- Develop tile method
- Pay rent method
- Win Game method/Condition

Action for all members:

- Meet again over the weekend to continue working

Minutes for Group **30**
Week commencing ***9***
Date of this minute **29/03/21**

The following team members were present

Name (printed/typed)	Signature
Gareth	Present
Helder	Present
Joe	Present
Daniel	Present

Discussion Points:

- Added functionality to program code
    - Develop element method
    - Tidied code

Actions Planned:

- Pay rent method
- Win Game method/Condition

Action for all members:

- Meet again over the weekend to continue working

Minutes for Group **30**
Week commencing **9**
Date of this minute **31/03/21**

The following team members were present

Name (printed/typed)	Signature
Gareth	Present
Helder	Present
Joe	Present
Daniel	Present

Discussion Points:

- Added functionality to program code
- Removed ability to buy pass go square and ‘free parking’ square
- Updated display all method to display only valid data for cost to buy/cost to develop/development level/owned by, i.e. removed values for pass go and free square tiles

Actions Planned:

- Pay rent method
- Win Game method/Condition

Action for all members:

- Meet again over the week to continue working

Minutes for Group **30**
Week commencing **10**
Date of this minute **03/04/21**

The following team members were present

Name (printed/typed)	Signature
Gareth	Present
Helder	Present
Joe	Present
Daniel	Present

Discussion Points:

- Added functionality to program code
- Develop element method completed
- Update resources for passing go method completed
- Offer buy element to next player method

Actions Planned:

- Pay rent method
- Win Game method/Condition

Action for all members:

- Meet again over the week to continue working

Minutes for Group **30**
Week commencing **10**
Date of this minute **07/04/21**

The following team members were present

Name (printed/typed)	Signature
Gareth	Present
Helder	Present
Joe	Present
Daniel	Present

Discussion Points:

- Added functionality to program code
- Added create player method
- Created set up player name method

Actions Planned:

- Pay rent method
- Win Game method/Condition

Action for all members:

- Meet again over the week to continue working

Minutes for Group **30**
Week commencing **10**
Date of this minute **10/04/21**

The following team members were present

Name (printed/typed)	Signature
Gareth	Present
Helder	Present
Joe	Present
Daniel	Present

Discussion Points:

- Added functionality to program code
- Pay rent method/use case tackled
- Tidied up code
- Implemented minor features,
    - e.g. development level rent values, updated starting resources, display player resources after passing go

Actions Planned:

- Finalise game functionality
- Make a start on report

Action for all members:

- Meet again over the week to continue working

Minutes for Group **30**
Week commencing **11**
Date of this minute **14/04/21**

The following team members were present

Name (printed/typed)	Signature
Gareth	Present
Helder	Present
Joe	Present
Daniel	Present

Discussion Points:

- Added functionality to program code
- Updated code to account for separate resources, i.e., equipment and workforce
- Re run game to check for bugs

Actions Planned:

- Finalise game functionality
- Make a start on report

Action for all members:

- Meet again over the week to continue working

Minutes for Group **30**
Week commencing **11**
Date of this minute **16/04/21**

The following team members were present

Name (printed/typed)	Signature
Gareth	Present
Helder	Present
Joe	Present
Daniel	Present

Discussion Points:

- Testing
- Final game run through
- Add additional code commenting

Actions Planned:

- Continue report write up
- Code divided and tested among the team according to test plan

Action for all members:

- Meet again over the week to continue working

Minutes for Group **30**

Week commencing **12**
Date of this minute **18/04/21**

The following team members were present

Name (printed/typed)	Signature
Gareth	Present
Helder	Present
Joe	Present
Daniel	Present

Discussion Points:

- Continue final Testing
- Report completion
- Peer assessment forms completed
- Video discussion for game functionality

Actions Planned:

- Prepare for submission
- Prepare video demonstration of game

Action for all members:

- Meet again over the week to continue working

Appendix 3 - Day-to-day Project Management

GitLab v Google Drive

The decision was made to utilize Google Drive as a shared workspace, as opposed to Git Lab. By doing so, the team could develop the project simultaneously, tracking fellow team members alternations live, assisting with team communication. Throughout the project, requirements have been achieved through a collective engaged effort, during periods where all members were amending and accessing the project files at the same time. This allowed for peer-to-peer problem solving and an overall, comprehensive understanding of the system by all team members.

To handle day-to-day project management, the Kanban-style, list-making application, Trello was utilized. Trello allowed the team to lay out the tasks that needed to be completed in a visual board format. Below are some examples of the Trello board at different points throughout the development.

29/03/2021