Software Engineering

CSC7053 Project: Technopoly

Group 24

James Clark, Allen Loyola, Jonathan Ashe, Shona Tolan
3/12/2019

CSC7053 Peer Assessment: Technopoly

Evaluation Group Number: 24				
Name	Contribution of time and effort[1]	Contribution to team-working and motivation[1]	Contribution to the deliverables [1,2]	Peer Score (Range 85 – 115)
Allen Loyola (40088753)	4	4	4	100
Shona Tolan (40024271)	4	4	4	100
James Clark (40083514)	4	4	4	100
Jonathan Ashe (40040887)	4	4	4	100

[1]Values: 1 = Less than average; 2 = Slightly less than average; 3 = Average; 4 = Slightly more than average; 5 = More than average

[2]This value should consider contributions in the round – direct contributions to required deliverables, and contributions that have made the deliverables possible.

Declaration

"I declare that I have read the Queen's University regulations on plagiarism, and that any contribution I have made to this assignment is my own original work, except for any elements that I have clearly attributed to third parties. I understand that this assignment will be subject to an electronic test for plagiarism and will also be subject to the University's regulations concerning late submission if it is received after the deadline."

Name	Date	Confirmation
Allen Loyola	14/03/2019	
Shona Tolan	14/03/2019	
James Clark	14/03/2019	
Jonathan Ashe	14/03/2019	

Contents

Requirement Analysis

Use Case Descriptions (ST, AL, JA, JC)

The following use cases were written at the beginning of the project to guide us into the coding process.

Flow of Events for **Enter Number of Players** Use-Case	
Objective	To enter the number of players that will take part in the game.
Actors	Player
Triggers	Game has been started.
Preconditions	No game play has begun.
Main Flow	1. System will request the number of players. 2. Player will enter the number of players that will be taking part. 3. System will ensure that a valid number between 2 and 4 (inclusive) has been entered.
Alternative Flows	At 2, less than 2 players or greater than 4 players may be entered. System will request number of players again and continue to step 3.
Post-Condition	Valid number of players has been set for the game. System proceeds with the Enter Player Names use case.
Inclusions	The Enter Player Names Use case is included at 3.

Flow of Events for **Enter Player Names** Use-Case	
Objective	To assign names to the players.
Actors	Players
Triggers	Use case Enter Number of Players has been completed.
Preconditions	Game play has not yet begun and a valid number of players are playing.
Main Flow	1. System will request name of 1st player. 2. Player will enter 1st player's name. 3. System will request name of 2nd player. 4. Player will enter 2nd player's name. 5. System will request name of 3rd player. 6. Player will enter 3rd player's name. 7. System will request name of 4th player. 8. Player will enter 4th player's name. 9. The system will determine the order of play randomly by shuffling the order of the players from the order they were entered.
Alternative Flows	At 4, a name already entered for another player may be entered. System will request the player's name again. At 5, only two players may be playing, use case terminates. At 6, a name already entered for another player may be entered. System will request the player's name again. At 7, only three players may be playing, use case terminates. At 8, a name already entered for another player may be entered. System will request the player's name again.
Post-Condition	Player's names have been stored and an order for play randomly determined.

Flow of Events for **Change Location** Use-Case	
Objective	To change location around the game board.
Actors	Player
Triggers	Player has chosen to roll the dice.
Precondition	It is the player's turn.
Main Flow	1. Player rolls the two die and the value of the rolls are recorded. 2. System informs the player of the total value of their rolls i.e. how many squares they will be moving. 3. System locates player's current position on the board. 4. System moves the player the appropriate amount of squares based on the value of their roll total. 5. System displays which square the player has landed on and any options available to them based on that square.
Alternative Flows	N/A
Post-condition	Player has landed on a square and play has continued with the appropriate use case.
Extension Points	At 4, player has passed the Go square, system proceeds to the Pass Go use case. At 5, square is the Go square, system proceeds to the Pass Go use case. At 5, square is the Stand By square. System proceeds to the next player's turn. At 5, square is a mining setup owned by another player, system proceeds to the Pay Player use case. At 5, square is a mining setup owned by the player, system informs the player and proceeds to the next player's turn. At 5, square is an un-owned mining setup, player may choose to purchase the setup, system proceeds to the Buy Mining Setup use case.

Flow of Events for **Pass Go** (Collect GameCoin) Use-Case	
Objective	To collect gameCoin by passing or landing on the Go square.
Actors	Player
Triggers	Player passes or lands on the Go square.
Precondition	Player has rolled dice and has changed location.
Main Flow	1. Once player has been informed of the square they have landed on, the system will inform the player how much gameCoin they are receiving. 2. System enters the appropriate amount of gameCoin into the player's total gameCoin balance. 3. System displays the player's new total gameCoin balance.
Alternative Flows	N/A
Post-condition	Player's total gameCoin balance has been increased by 200 gameCoin for passing or landing on Go. System proceeds with the remainder of the player's turn.

Flow of Events for the **Pay Player** Use-Case	
Objective	To pay another player rent.
Actors	Player
Triggers	Player lands on a mining setup owned by another player.
Precondition	Game play has started, it is the player's turn.
Main Flow	1. System will deduct the cost in gameCoin of the mining setup rent from the player's total gameCoin balance. 2. System will increase the owner of the mining setup's gameCoin balance with the gameCoin rent collected from the paying player. 3. System will display both players their new gameCoin balances.
Alternative Flows	N/A
Post-condition	Player's gameCoin balance has been reduced and other player's gameCoin balanced has been increased.
Extension Points	At 1, the deduction of gameCoin will leave the player with a balance of less than zero. System will proceed to step 2 and 3, after carrying out step 2 and 3 system will then proceed to the End Game use case.

Flow of Events for the **Buy a Mining Setup** Use-Case	
Objective	To buy a mining setup.
Actors	Player
Triggers	Player has chosen to buy a mining setup.
Precondition	Player has landed on a mining setup that is not already owned by another player, the player does not already own it and the player has sufficient gameCoin to afford to buy it.
Main Flow	1. System will deduct the cost in gameCoin of the mining setup from the player's total gameCoin balance. 2. Ownership of the mining setup will now be set to the player. 3. System will alert the player that they now own the mining setup and will display their new gameCoin balance.
Alternative Flows	N/A
Post-condition	Player now owns the mining setup. System proceeds with next player's turn.

Flow of Events for the **Manage Mining Setups** Use-Case	
Objective	To attempt to carry out a megawatt upgrade or establish maximum mining capacity on a mining setup.
Actors	Player
Triggers	Player has chosen the option to manage their mining setups.
Precondition	It is the player's turn.
Main Flow	1. System will display all of the Mining Setups owned by the player, along with their field type, the cost of an upgrade, the upgrade level the setups are currently on and their current rent cost. Their current balance will also be displayed. 2. Player will select the Mining Setup up they wish to upgrade. 3. System will deduct the cost in gameCoin of the upgrade from the player's total gameCoin balance. 4. System will assign the new gameCoin rent price for landing on this mining setup. 5. System will alert the player that the upgrade has been carried out , will display their new gameCoin balance and will display the new rent cost should another player land on this upgraded mining setup.
Alternative Flows	At 1, player will not own any Mining Setups, player will be informed of this and use case will end. System will return the player to their choice of turn options. At 2, the player will not own the entire field, player will be informed of this and use case will end. System will return the player to their choice of turn options. At 2, player may decide not to proceed with an upgrade and instead choose to cancel. System will return the player to their choice of turn options. At 2, 3 upgrades may have already been applied, use case will continue but instead of upgrading further maximum mining capacity will instead be established along with a new rent cost. At 2, maximum mining capacity has already been applied to the Mining Setup. No further upgrades can be made. System will return the player to their choice of turn options. At 3, the deduction of gameCoin would leave the player with a balance of less than zero. System will not carry out the upgrade and will return the player to their choice of turn options.
Post-condition	Mining setup has been successfully upgraded and a new rent cost has been applied to it. System will return the player to their choice of turn options.

Flow of Events for the **End Game** Use-Case	
Objective	For the game to end and a winner to be declared.
Actors	Player
Triggers	Player has chosen the option to end game or a player's total gameCoin balance has gone below zero.
Precondition	Game play has begun and it is a player's turn.
Main Flow	1. System shows all players' final details. 2. System will declare the winning player based on the highest total gameCoin balance remaining.
Alternative Flows	N/A
Post-condition	Game has finished, winner has been declared.

UML Use Case Diagram (JC, ST, JA, AL)

Below we have the final Use Case Diagram designed to help understand how we would develop the code into a full Technopoly game.

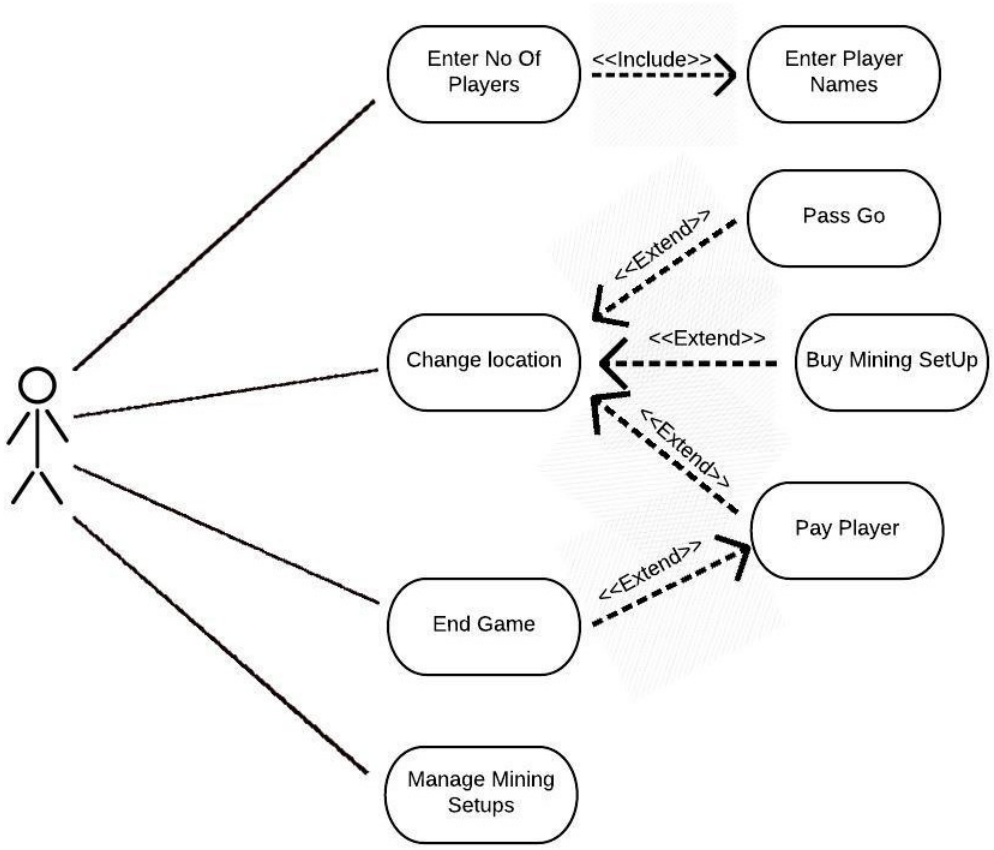


Figure 1: Use case diagram

The Use Case Diagram is designed with the intention of showing all the options that the player (actor) directly has access to. At the beginning of the game the player will enter the number of players which always includes in its normal flow entering the player names. When the player decides to change location, they may Pass Go and collect gameCoin, land on and potentially buy a mining Set-Up or land on another player's property and pay them rent. All these methods extend from the change location behaviour. At any point in the game the player can end the game by choosing to. The game may end when a player lands on another player's property. If they don't have enough game coin to pay the player, then the end game behaviour is invoked via the extension. The final behaviour that the player can operate is the manage mining setups.

Guide to Virtual Board (JC, JA, ST, AL)

The theme of our Technopoly virtual board is based on cryptocurrency in the real world. Each player is given a fictional amount of cryptocurrency called gameCoin. The board is set with 12 squares including 4 fields (made up of 10 purchasable mining setups), a Standby square and a Pass Go square. As the players make their way around the board they land on squares and can purchase mining setups with their Game Coin. Once a full field is owned the player may perform a megawatt upgrade on a mining setup within that field. These megawatt upgrades can be done 3 times before the equivalent of a monopoly hotel can be established, in this case known as establishing maximum mining capacity on a mining setup. Megawatt upgrades and maximum mining capacity all increase the rent value of the setup meaning if another player lands on it they have to pay more gameCoins to the setup's owner. Below is a pricing table for the game and a small diagram of how our board looks/functions.

Field Key

Local Machines
Dedicated Machines
Home Mining Rigs
Largescale Industrial Mining
Free Squares

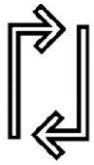
Pricing Table

No.	Name	Initial Cost	Upgrade Cost	Rent	Rent with 1 Upgrade	Rent with 2 Upgrades	Rent with 3 Upgrades	Rent with max capacity
0	*Go (Collect 200)*							
1	Standard PC	60	50	2	8	24	60	120
2	High End Gaming PC	60	50	2	8	24	60	120
3	Low End GPU	140	100	10	40	120	300	600
4	High End GPU	140	100	10	40	120	300	600
5	Dual GPU	160	100	12	48	144	360	720
6	*Stand By*							
7	DIY Rig	260	150	22	88	264	660	1320
8	Compact Rig	260	150	22	88	264	660	1320
9	Custom Rig	280	150	22	88	264	660	1320
10	Dedicated Mining Centre	350	200	35	140	420	1050	2100
11	Climate Controlled Mining Centre	400	200	50	140	600	1500	3000

Game Board

Local Machines
Dedicated Machines
Home Mining Rigs
Largescale Industrial Mining
Free Squares

General Direction of Gameplay



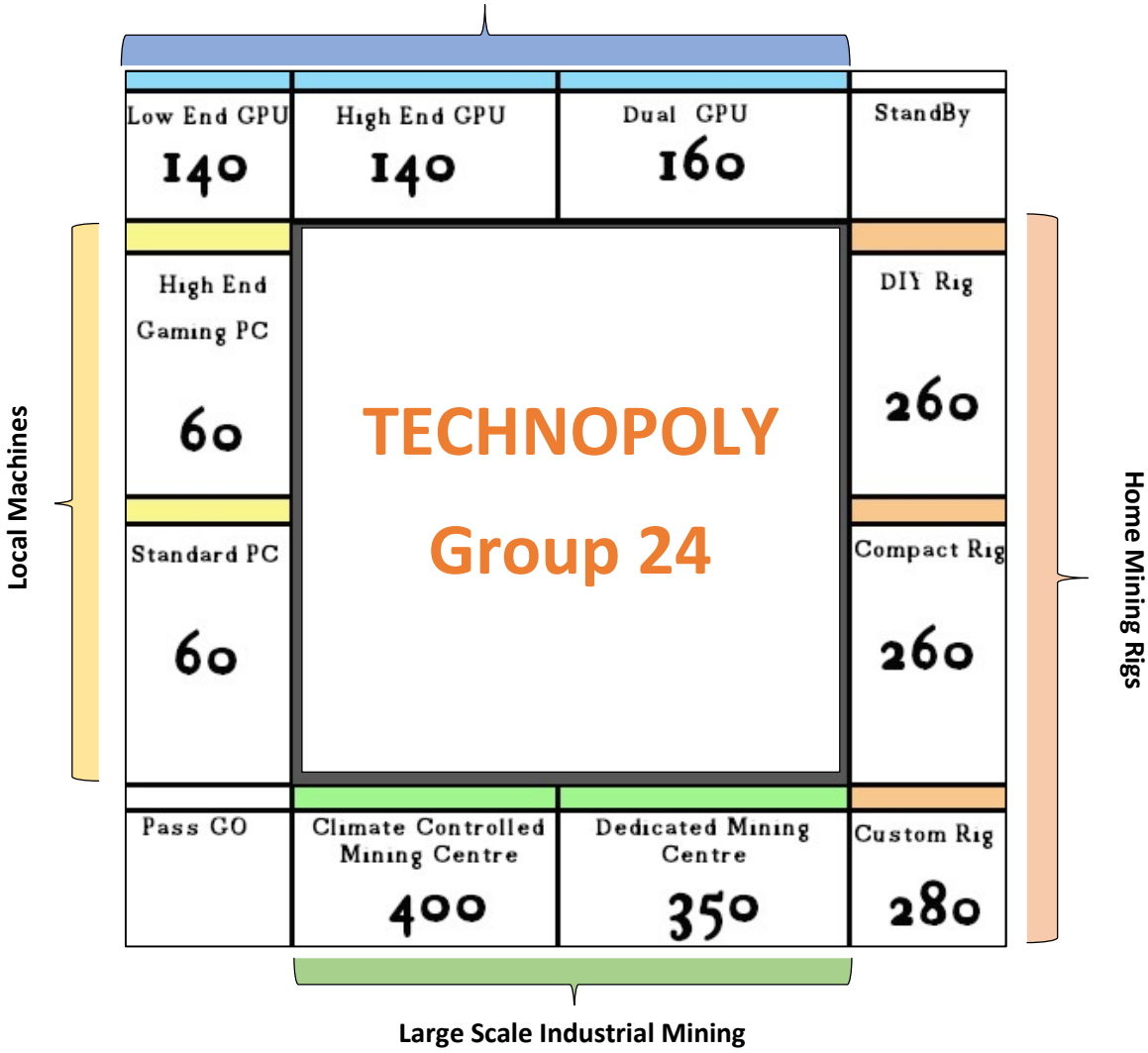


Figure 2: Game board design

We decided to bring our Technopoly cards to life to get an even better understanding of the game. Below you can see the cards with their prices, colour and name.

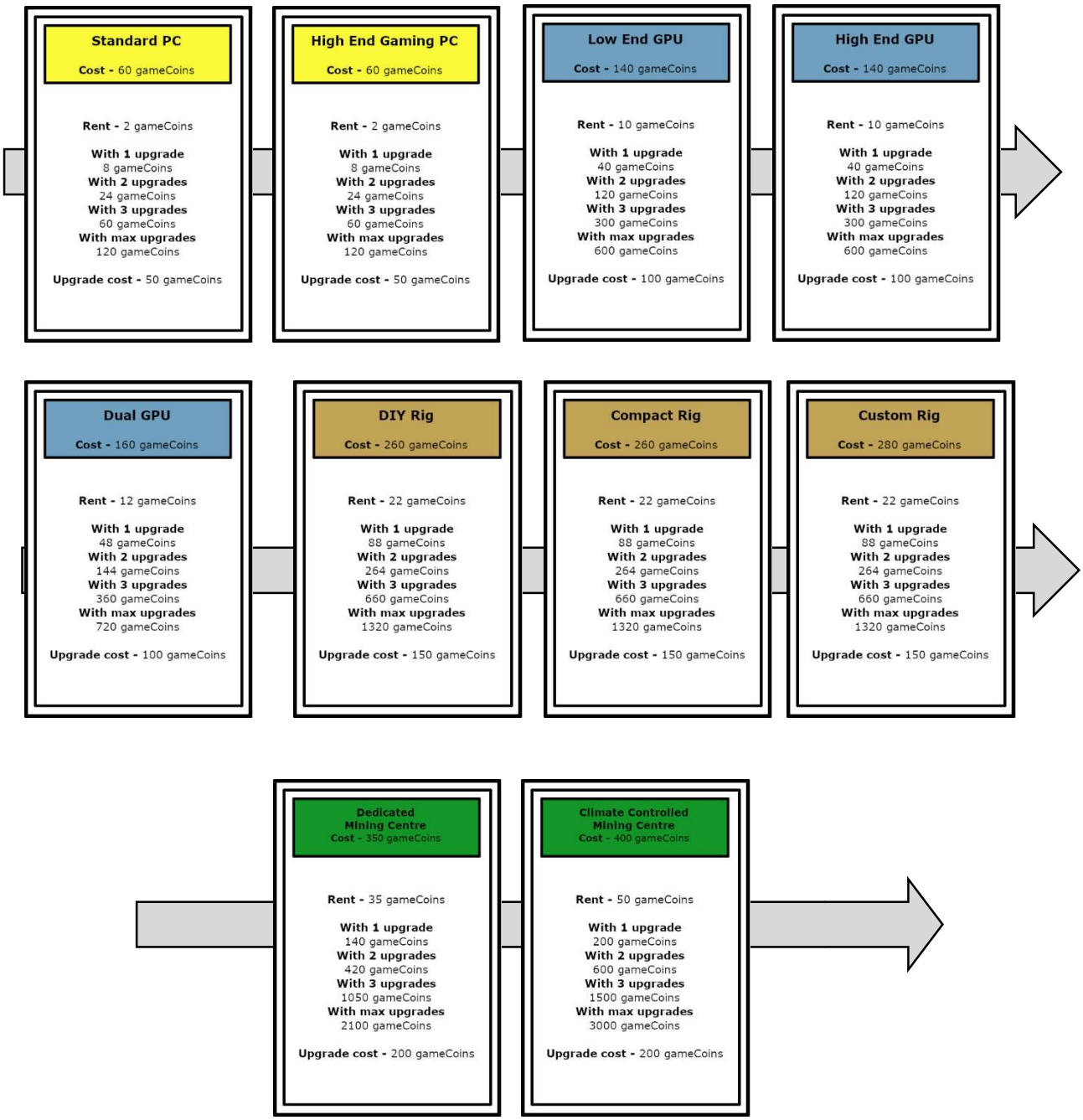


Figure 3: Title Deeds

Realisation

UML Sequence Diagrams (ST,JC)

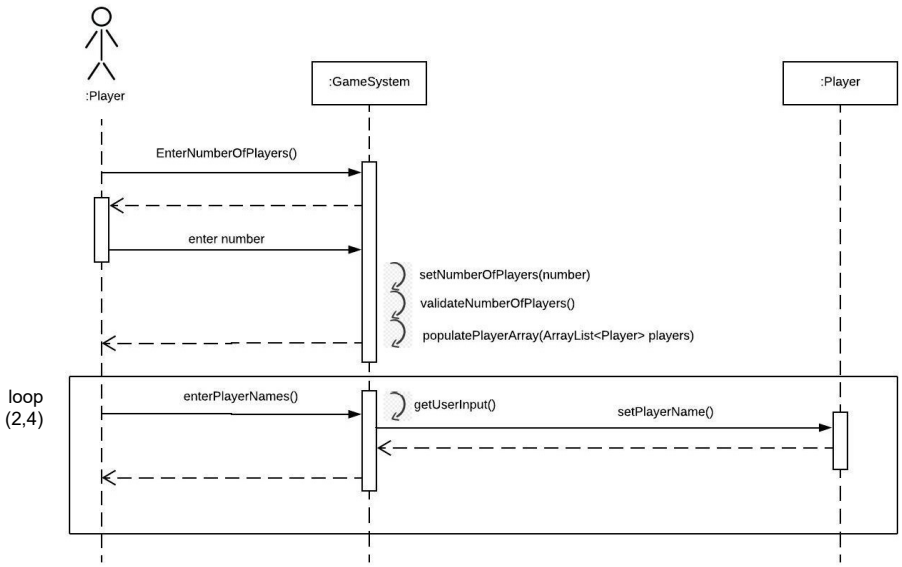


Figure 4: UML sequence diagram enter number of players and enter player names

The above sequence diagram shows the process for entering the number of players and player names into the system. The game begins and the enterNumberOfPlayers() method is immediately called from the game system. The player then enters a number, the number of players is validated and set (to ensure between 2 and 4) then an array list is populated. Then each player enters their name into the game and a method is called to check there are no duplicates. The system loops through this process until all players are entered into the system. The game can then begin.

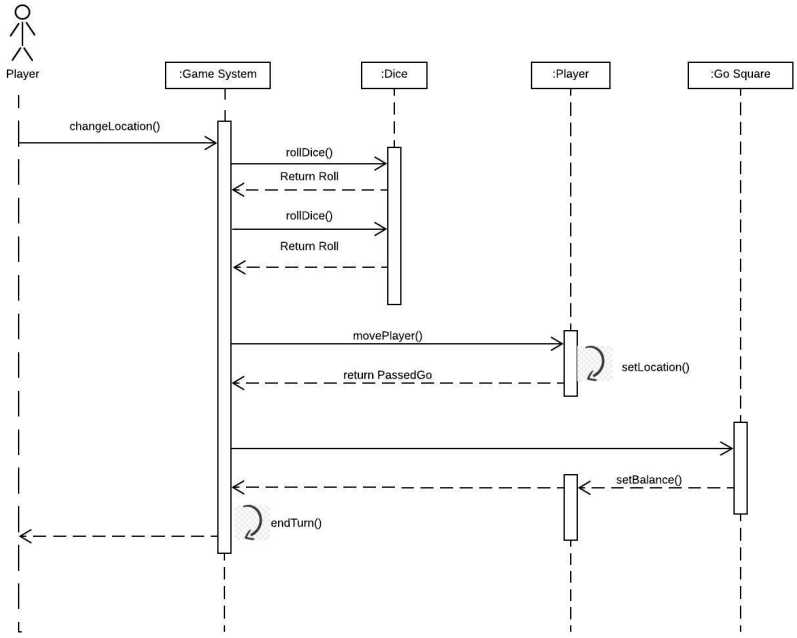


Figure 5: UML sequence diagram pass go

When the actor passes go in the game the following actions are taken out within the code. The player will already have invoked the changeLocation() method to decide to move. The system moves the player and returns if they have passed go. If the passed go return is true the Go square will set the player's new balance. This ends the player's turn moving on to the next player.

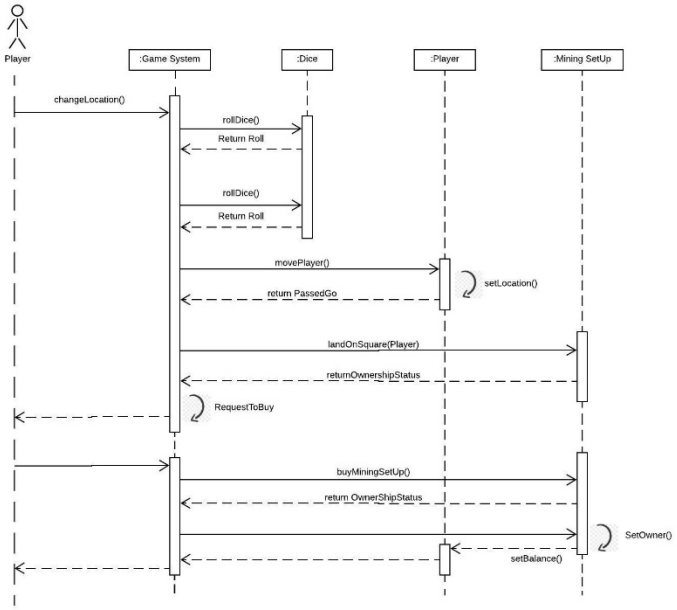


Figure 6: UML Sequence diagram buy mining setup

The above diagram is a realisation the player buying a mining setup within the game. The behaviour extends changing location so this is shown at the beginning. Once the player's new location is set and they have landed on a square the ownershipStatus of the square is determined in relation to the current player. If the setup is not owned the player is given the option to purchase the mining setup (buyMiningSetUp() method is invoked). The new owner and balance are set. The getBalance() is returned so the new owner can see their new balance and the sequence is over.

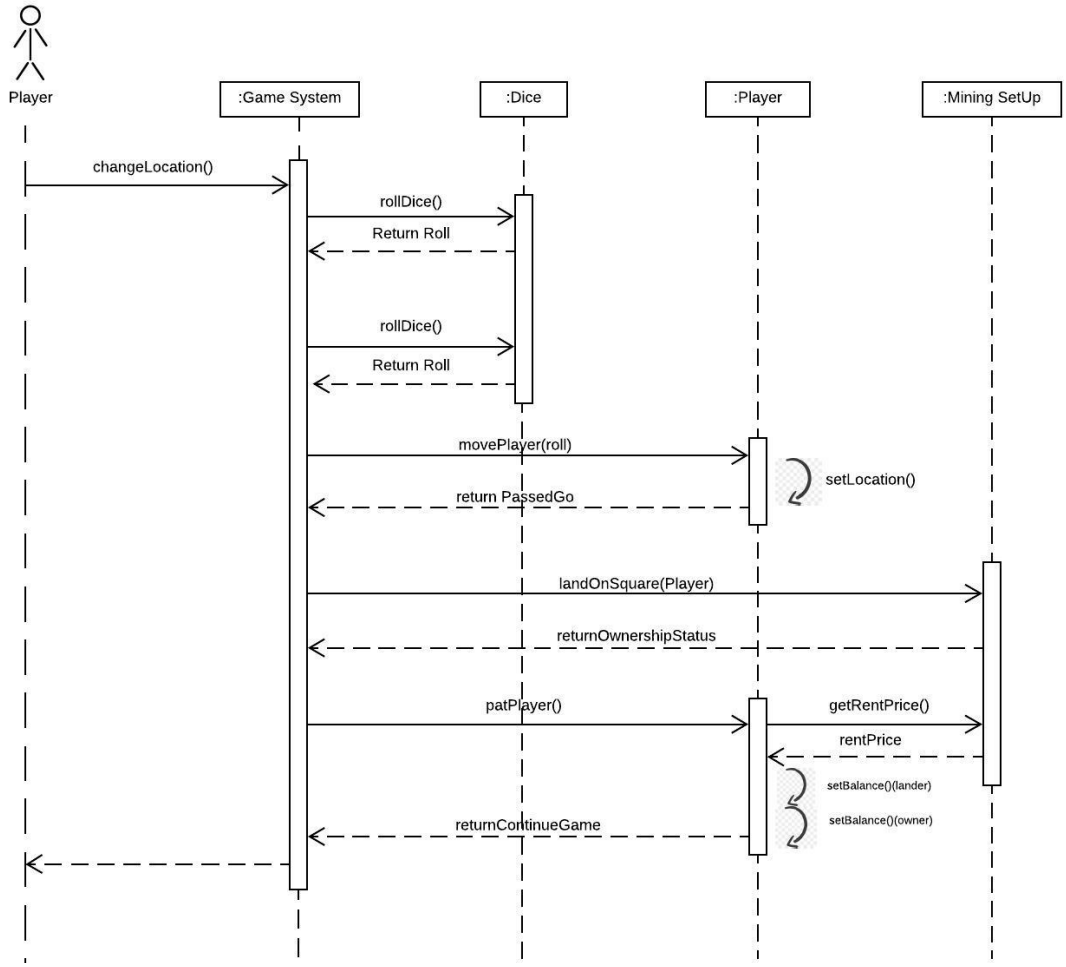


Figure 7: UML sequence diagram pay player

The above sequence diagram is for paying a player rent. This behaviour extends changing location, therefore this behaviour is shown first. Once the landOnSquare() method has been invoked the OwnershipStatus is returned, if another player owns the mining setup the current turn player must pay them rent in game coin. The payPlayer() method is called and then the miningSetup will getRentPrice and return that value. After this the players (lander and owner) balance will be set to a new amount of gameCoin (with rent subtracted and added appropriately). The game will then continue with the next player's turn.

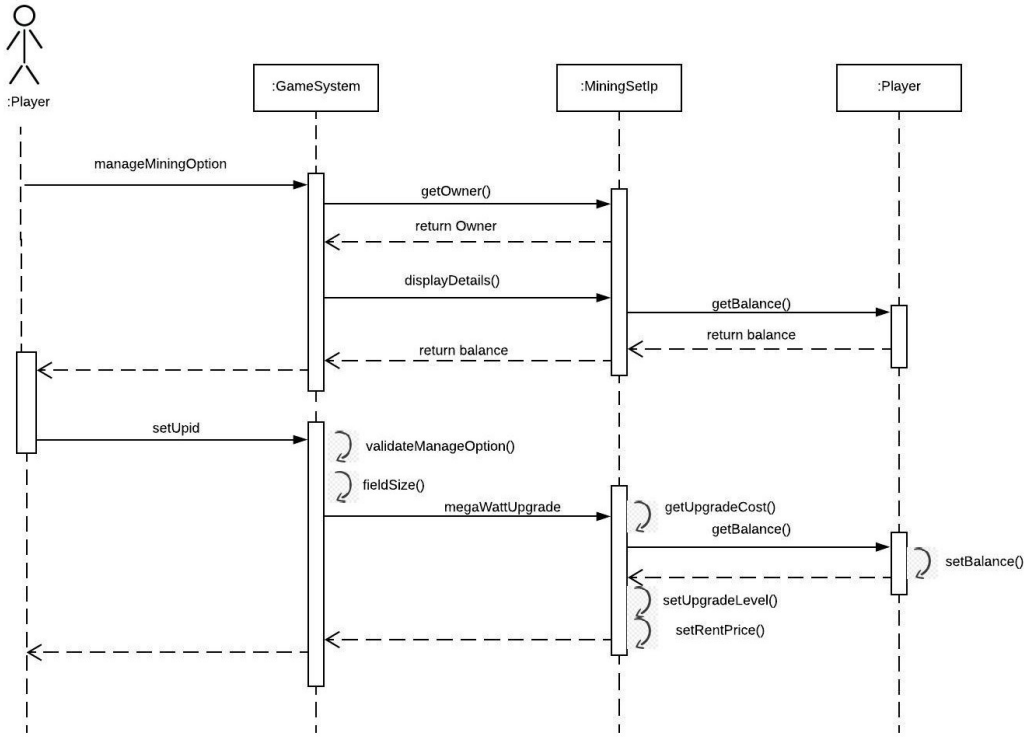


Figure 8: UML sequence diagram manage mining setup

When a layer chooses to manage their mining setups the manageOption() method is invoked. The system then gets the owner of all setups and returns the details of those owned by the current player. The player's balance is also returned to be shown. The player chooses a setup ID of the setup they would like to upgrade. Game system calls the megawatt upgrade method if appropriate (validateManageOption), cost of the upgrade is obtained from calling getUpgradeCost and player's new balance is set along with the new upgrade level and rent price.

Design

UML Class Diagram (ST, JC)

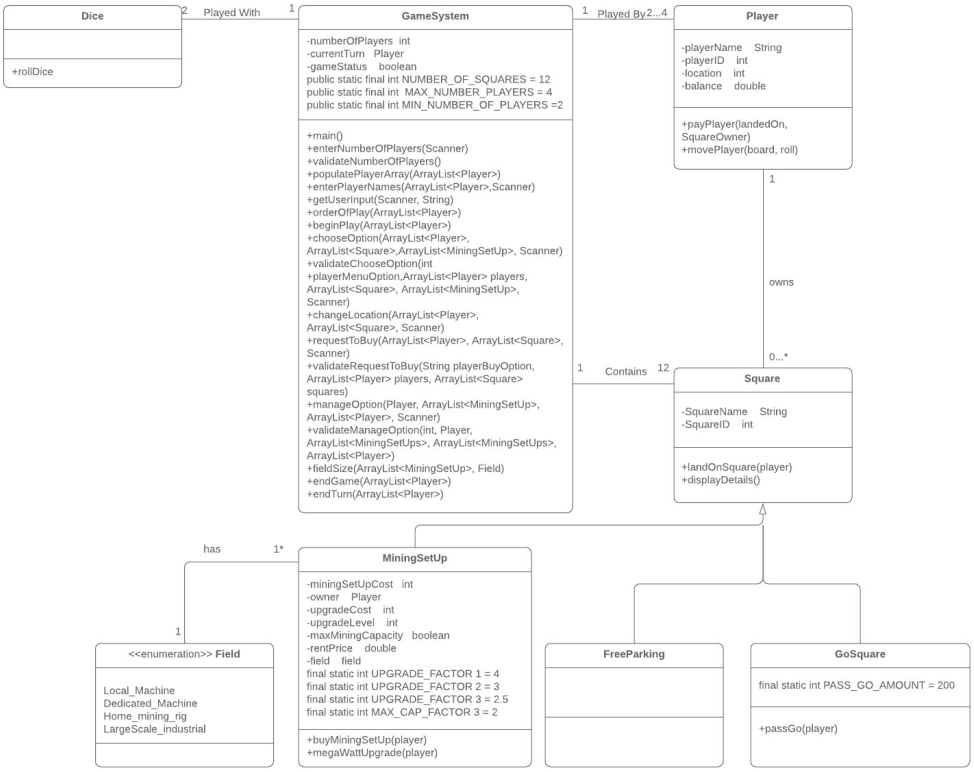


Figure 9: UML class diagram

As shown in the UML Class Diagram our system contains seven classes (GameSystem, Player, Dice, Square, MiningSetUp, GoSquare and StandBy) and one Enum (Field). While designing the system we have tried to take into account areas that in future the customer may wish to modify from the initial requirements.

While the requirements state a player is to throw two virtual dice, we wanted to ensure that should the need for only one dice or multiple die arise in future , this can be changed within the code as easily as possible. A single Dice class was created containing the method rollDice(). This method simulates rolling of only one dice so within the main GameSystem class two separate dice objects are instantiated and rolled. The sum of both rolls is used for changing the player location (changeLocation()).

Inheritance has been used where appropriate so that code can be re-used. The Square class has been created as a superclass to the various types of squares required, making MiningSetup, GoSquare and StandBy all the more specific subclasses of Square. Should a new type of square be required for the game, this can be added as another subclass of Square. Square has been created abstract to prevent a simple square being added to the game board. Squares are arranged in an ArrayList as a board for the game. An ArrayList was chosen as they support dynamic arrays that can grow as needed. The size can be changed should more squares be required for the game. The NUMBER_OF_SQUARES is stored within the GameSystem as a constant that can be accessed by various methods and classes . Again should the board ever need to change size this value only needs to be changed once.

Constants have also been used within the MiningSetUp and GoSquare classes for ease of change. The GoSquare contains the constant value PASS_GO_AMOUNT which is used to add gameCoin to a player's balance when they pass go. MiningSetup contains constants for calculating new rent prices after a mining setup has been upgraded: UPGRADE_FACTOR_1, UPGRADE_FACTOR_2, UPGRADE_FACTOR_3 and MAX_CAPACITY_FACTOR. The Player class contains the constants STARTING_LOCATION and STARTING_BALANCE. These constants have been used within the classes default constructor to ensure any Player created begins the game in the correct starting position and with the correct starting balance. Changing of either of these rules will only require change of the constant values.

MiningSetup contains an instance variable of type Field. Field has been created as an Enum rather than a String to ensure if any further squares are created they will have only one of the four currently approved Field types. If further Field types are required these can simply be added to the Enum field.

GameSystem contains many method that make use of the Scanner class. In order to perform adequate JUnit tests on the scanner inputs, methods containing scanners were broken down into two separate methods; one containing the scanner and one to validate the input received from the scanner. E.g. enterNumberOfPlayers() and validateNumberOfPlayers(). Additional validation is provided on enterPlayerNames() to ensure no duplicate names are entered.

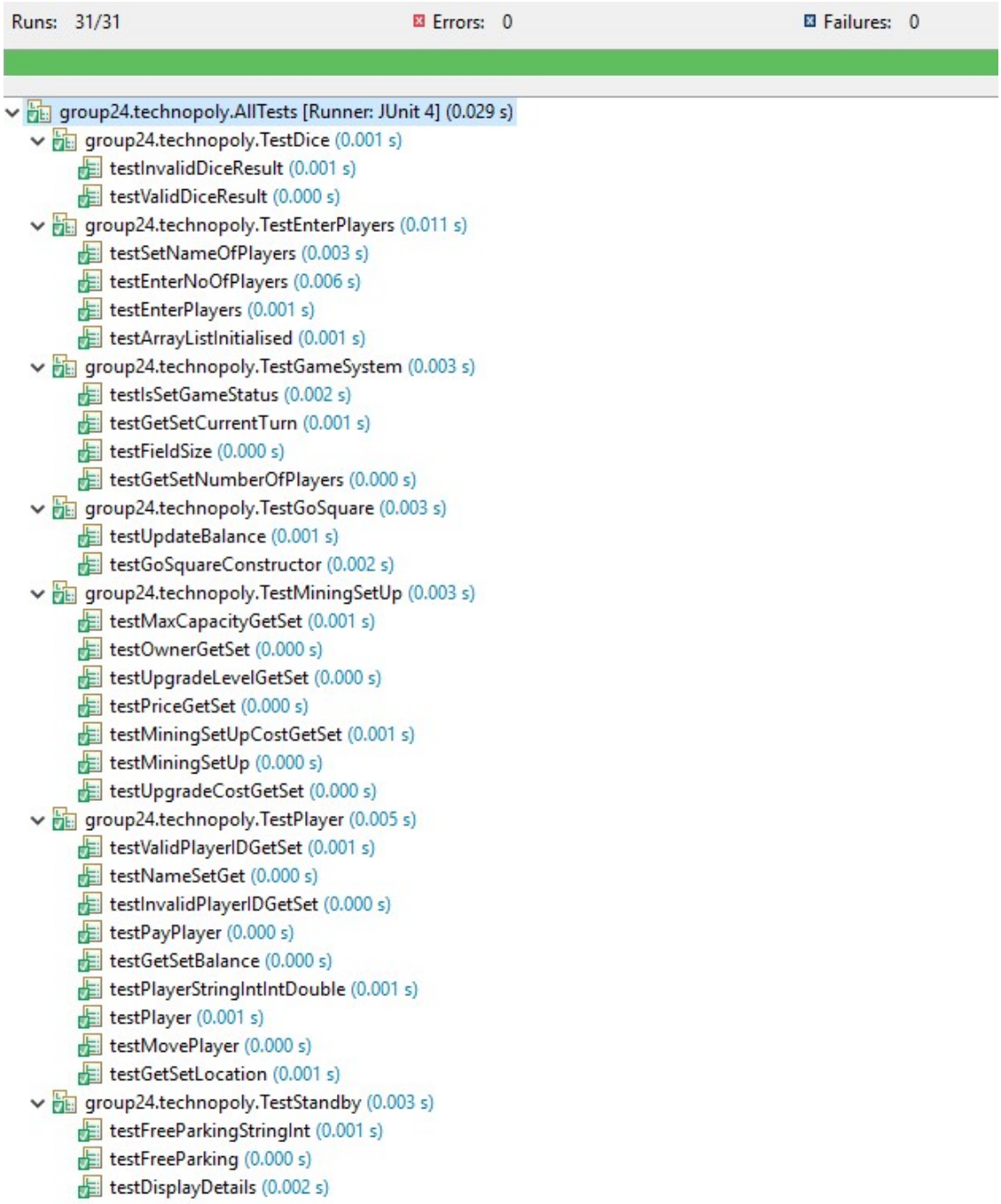
The requirements state that the game has up to four players, however play would not work with only one player. We therefore set a MIN_NUMBER_OF_PLAYERS at two and a MAX_NUMBER_OF_PLAYERS at four. If the customer wishes to change these values they only need to be changed in one place. An ArrayList of players has been used so changing the number of players will not affect the game functionality.

Appendices

Appendix 1

Test Plan (JA)

Details of the Junit testing carried out:



The testing was mainly focused around methods that included validation and where exceptions were used as exhaustive testing is not possible. Coverage was 26.3% in total, which includes the abstract classes which were not tested.

Planning for acceptance testing as follows was also carried out on the system:

ID	Use Case Ref	Description of Test	Test Initialisation	Test Inputs	Test Procedure	Expected Results	Pass / Fail
01	UC01	Testing enter number of players.	Run game.	A valid number of players (4).	Enter 4 as the number of players.	System accepts the value and proceeds to request name of first player.	Pass
02	UC01	Testing enter number of players.	Run game.	An invalid number of players (8).	Enter 8 as the number of players.	System displays that number of players is invalid and asks user to enter valid number of players again.	Pass
03	UC01	Testing enter number of players.	Run game.	A character other than a number.	Enter 'a' as the number of players.	System displays that number of players is not valid and asks user to enter valid number of players again.	Pass
04	UC02	Testing enter player names.	Run game, enter number of players as 4.	Enter player names: 1: p1 2: p2 3: p3 4: p4	Enter the four names when prompted.	System will prompt for one name at a time and after entering all four names will proceed with printing random order of play.	Pass
05	UC02	Testing enter player names.	Run game, enter number of players as 3.	Enter player names: 1: p1 2: p2 3: p3	Enter the three names when prompted.	System will prompt for one name at a time and after entering all three names will proceed with printing a random order of play.	Pass
06	UC02	Testing enter player names.	Run game, enter number of players as 2.	Enter player names: 1: p1 2: p2	Enter the two names when prompted.	System will prompt for one name at a time and after entering all two names will proceed with printing a random order of play.	Pass
07	UC02	Testing enter player names.	Run game, enter number of players as 4.	Enter player names: 1: p1 2: p2 4: p1	Enter the three names when prompted.	System will prompt for one name at a time and after entering the third name will request it again as it is a duplicate.	Pass
08	UC03	Testing changing location of player.	Run game with 2 players, enter different names for each player.	Enter player names: 1: p1 2: p2	On first player's turn choose option one to roll the dice.	System prints the values of each roll and will move the player (total number of their two rolls) squares around the board. System displays the square landed on and which options are available for it.	Pass
09	UC03	Testing changing location of player when landing on Go.	Run game with 2 players, enter different names for each player.	Enter player names: 1: p1 2: p2	Continue play with each player choosing option one, to roll dice on each turn until one player lands on Go.	System prints the values of each roll and will move the player (the total number of their two rolls) squares around the board. They system prints that the player has passed go and increases their gameCoin balance by 200 gameCoin.	Pass

ID	Use Case Ref	Description of Test	Test Initialisation	Test Inputs	Test Procedure	Expected Results	Pass / Fail
10	UC03	Testing changing location of player when landing on Standby.	Run game with 2 players, enter different names for each player.	Enter player names: 1: p1 2: p2	Continue play with each player choosing option one, to roll dice on each turn until one player lands on Standby.	System prints the values of each roll and will move the player (the total number of their two rolls) squares around the board. System displays that Standby has been landed on and cannot be purchased.	Pass
11	UC03	Testing changing location of player when landing on an un-owned mining setup.	Run game with 2 players, enter different names for each player.	Enter player names: 1: p1 2: p2	Continue play with each player choosing option one, to roll dice on each turn until one player lands on an un-owned mining setup.	System prints the values of each roll and will move the player (the total number of their two rolls) squares around the board. System displays the square landed on and offers them the choice to purchase the mining setup.	Pass
12	UC03	Testing changing location of player when landing on mining setup owned by them.	Run game with 2 players, enter different names for each player.	Enter player names: 1: p1 2: p2	Continue play, each player choosing option one, to roll dice on each turn and selecting 'y' to purchase any un-owned mining setups they land on until one player lands on mining setup owned by themselves.	System prints the values of each roll and will move the player (the total number of their two rolls) squares around the board. System displays the square landed on and informs them that the setup is owned by them.	Pass
13	UC03	Testing changing location of player when landing on a mining setup owned by another player.	Run game with 2 players, enter different names for each player.	Enter player names: 1: p1 2: p2	Continue play with each player choosing option one, to roll dice on each turn and selecting 'y' to purchase any un-owned mining setups they land on until one player lands on mining setup owned by the other player.	System prints the values of each roll and will move the player (the total number of their two rolls) squares around the board. System displays the square landed on and informs them that the setup is owned by another player. System then proceed to the pay player use case.	Pass
14	UC04	Testing a player passing Go.	Run game, enter number of players as 4.	Enter player names: 1: p1 2: p2 3: p3 4: p4	Continue play with each player choosing option one, to roll dice on each turn until one player passes go.	System informs player that they have passed go during changing location and that the receive 200 gameCoin. System will display player's new balance (with 200 additional gameCoin).	Pass
15	UC05	Testing a player paying another player for landing on their mining setup.	Run game, enter number of players as 4.	Enter player names: 1: p1 2: p2 3: p3 4: p4	Continue play with each player choosing option one, to roll dice on each turn and selecting 'y' to purchase any un-owned mining setups they land on until one player lands on mining setup owned by the other player.	System will subtract the rent cost of the setup landed on from the player who has landed on it. System will add the rent cost of the setup landed on to the owner's gameCoin balance. System will display both new balances.	Pass

ID	Use Case Ref	Description of Test	Test Initialisation	Test Inputs	Test Procedure	Expected Results	Pass / Fail
16	UC06	Testing a player buying a mining setup.	Run game with 2 players, enter different names for each player.	Enter player names: 1: p1 2: p2	Continue play with each player choosing option one, to roll dice on each turn until one player lands on an un-owned mining setup. Choose 'y' to purchase the setup.	System will deduct the cost in gameCoin of the mining setup from player's total balance. System will alert player that they now own the mining setup and will display their new gameCoin balance.	Pass
17	UC07	Testing a player attempting to manage mining setups when they do not own any setups.	Run game with 2 players, enter different names for each player.	Enter player names: 1: p1 2: p2	Continue play with p1 choosing option two to manage mining setups on their first turn.	System will inform the player that they do not own any mining setups. System will return the player to their choice of turn options.	Pass
18	UC07	Testing a player managing mining setups they own.	Run game with 2 players, enter different names for each player.	Enter player names: 1: p1 2: p2	Continue play with each player choosing to roll dice on each turn and selecting 'y' to purchase any un-owned mining setups. When a player has purchased a mining setup on their next turn select option two to manage mining setups.	System will display all of the Mining Setups owned by the player, along with their field type, the cost of an upgrade, the upgrade level the setups are currently on and their current rent cost. Their current balance will also be displayed.	Pass
19	UC07	Testing a player attempting to manage mining setups when they do not own a whole field.	Run game, enter number of players as 4.	Enter player names: 1: p1 2: p2 3: p3 4: p4	Continue play with each player choosing to roll dice on each turn and selecting 'y' to purchase any un-owned mining setups. When a player has purchased a mining setup on their next turn select manage mining setups and select the mining setup that has just been purchased.	System will inform player that they do not own the entire field of setups. System will return the player to their choice of turn options.	Pass
20	UC07	Testing a player changing their mind about upgrading their mining setups.	Run game with 2 players, enter different names for each player.	Enter player names: 1: p1 2: p2	Continue play with each player choosing to roll dice on each turn and selecting 'y' to purchase any un-owned mining setups. When a player has purchased a mining setup on their next turn select option two to manage mining setups. Then select the option to cancel.	System will return the player to their choice of turn options.	Pass

ID	Use Case Ref	Description of Test	Test Initialisation	Test Inputs	Test Procedure	Expected Results	Pass / Fail
21	UC07	Testing a player managing their mining setups.	Run game with 2 players, enter different names for each player.	Enter player names: 1: p1 2: p2	Continue play with each player choosing to roll dice. On p1's turn choose 'y' to purchase any un-owned mining setups landed on and on p2's turn choose 'n' to decline the purchase of any un-owned mining setups until p1 owns an entire field. On p1's next turn choose option two to manage mining setups and select a mining setup from the complete field.	System will deduct the cost in gameCoin of the upgrade from the player's total gameCoin balance. System will alert the player that the upgrade has been carried out , will display their new gameCoin balance and will display the new rent cost should another player land on this upgraded mining setup.	Pass
22	UC07	Testing a player managing a mining setup that has already been upgraded once.	Run game with 2 players, enter different names for each player.	Enter player names: 1: p1 2: p2	Continue play with each player choosing to roll dice. On p1's turn choose 'y' to purchase any un-owned mining setups landed on and on p2's turn choose 'n' to decline the purchase of any un-owned mining setups until p1 owns an entire field. On p1's next turn choose option two to manage mining setups and select a mining setup from the complete field. Choose option 2 again and select the same field again.	System will deduct the cost in gameCoin of the upgrade from the player's total gameCoin balance. System will alert the player that the upgrade has been carried out , will display their new gameCoin balance and will display the new rent cost should another player land on this upgraded mining setup.	Pass
23	UC07	Testing a player managing a mining setup that has already been upgraded twice.	Run game with 2 players, enter different names for each player.	Enter player names: 1: p1 2: p2	Continue play with each player choosing to roll dice. On p1's turn choose 'y' to purchase any un-owned mining setups landed on and on p2's turn choose 'n' to decline the purchase of any un-owned mining setups until p1 owns an entire field. On p1's next turn choose option two to manage mining setups and select a mining setup from the complete field. Choose option 2 again and select the same field again, then repeat a third time.	System will deduct the cost in gameCoin of the upgrade from the player's total gameCoin balance. System will alert the player that the upgrade has been carried out , will display their new gameCoin balance and will display the new rent cost should another player land on this upgraded mining setup.	Pass

ID	Use Case Ref	Description of Test	Test Initialisation	Test Inputs	Test Procedure	Expected Results	Pass / Fail
24	UC07	Testing a player managing a mining setup that has already been upgraded three times.	Run game with 2 players, enter different names for each player.	Enter player names: 1: p1 2: p2	Continue play with each player choosing to roll dice. On p1's turn choose 'y' to purchase any un-owned mining setups landed on and on p2's turn choose 'n' to decline the purchase of any until p1 owns an entire field. On p1's next turn choose option two to manage mining setups and select a mining setup from the complete field. Choose option 2 again and select the same field again, then repeat a third and fourth time.	System will inform the player that all 3 upgrades have been carried out but that maximum mining capacity will instead be established. System will deduct the cost in gameCoin of the upgrade from the player's total gameCoin balance. System will alert the player that the upgrade has been carried out , will display their new gameCoin balance and will display the new rent cost should another player land on this maximum mining capacity mining setup.	Pass
25	UC07	Testing a player managing a mining setup that has already been upgraded three times and maximum mining capacity has been established.	Run game with 2 players, enter different names for each player.	Enter player names: 1: p1 2: p2	Continue play with each player choosing roll dice. On p1's turn choose 'y' to purchase any un-owned mining setups and on p2's turn choose 'n' to decline the purchase of any un-owned mining setups until p1 owns an entire field. On p1's next turn choose option two to manage mining setups and select a mining setup from the complete field. Choose option 2 again and select the same field again, then repeat a third, fourth and fifth time.	System will inform player that no further upgrades can be applied. System will return the player to their choice of turn options.	Pass
26	UC08	Testing the game ending when a player runs out of gameCoin.	Run game with 2 players, enter different names for each player.	Enter player names: 1: p1 2: p2	Continue play with each player choosing to roll dice. On p1's turns choose 'y' to purchase any un-owned mining setups landed on and on p2's turn choose 'n' to decline any purchase, managing any mining setups once a whole field is owned by p1. Play until p2 runs out of gameCoin.	System shows all players' final gameCoin balances. System will declare the winning player based on the highest total gameCoin balance remaining as p1.	Pass

ID	Use Case Ref	Description of Test	Test Initialisation	Test Inputs	Test Procedure	Expected Results	Pass / Fail
27	UC08	Testing game ending when a player decides to end game before any players have gained or lost any gameCoin.	Run game, enter number of players as 3.	Enter player names: 1: p1 2: p2 3: p3	On p1's first turn choose option three to end game.	System shows all players' final gameCoin balances. System will declare the winning player based on the highest total gameCoin balance remaining as all players.	Pass
28	UC08	Testing game ending when a player ends the game and a player's balance differs from their beginning balance.	Run game, enter number of players as 4.	Enter player names: 1: p1 2: p2 3: p3 4: p4	Continue play with each player choosing to roll dice on each turn and selecting 'y' to purchase any un-owned mining setups they land on until p2's third turn; choose option three to end game.	System shows all players' final gameCoin balances. System will declare the winning player based on the highest total gameCoin balance remaining.	Pass

Appendix 2
Minutes Section (JC, AL)

Minutes for Group 24 Week commencing 3 Date of this minute 31/01/19

The following team members were present

Name	Signature
Allen Loyola	Present
James Clark	Present
Shona Tolan	Present
Jonathan Ashe	Present

Agenda

- Project kick off
- Discuss possible theme – Cryptocurrency
- Project overview - confirm main tasks

Task Reporting (Briefly list the progress for each team member in the last week.*)

Allen Loyola	Read project requirements
James Clark	Read project requirements
Shona Tolan	Read project requirements
Jonathan Ashe	Read project requirements

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

Allen Loyola	List of use cases for next meeting
James Clark	List of use cases for next meeting
Shona Tolan	List of use cases for next meeting
Jonathan Ashe	List of use cases for next meeting

Each team member to read document and take note of any use cases they find – To be combined at the next meeting.

Minutes for Group 24 Week commencing 4 Date of this minute 07/02/19

The following team members were present

Name	Signature
Allen Loyola	Present
James Clark	Present
Shona Tolan	Present
Jonathan Ashe	Present

Agenda:

- Combine Use Cases
- Divide Use Case descriptions workload between the group

Task Reporting (Briefly list the progress for each team member in the last week.*)

Allen Loyola	Use Cases submitted
James Clark	Use Cases submitted
Shona Tolan	Use Cases submitted
Jonathan Ashe	Use Cases submitted

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

Allen Loyola	• Use Case description
	  ○ Pay Player
	  ○ Develop Area
	  ○ Buy Area
	• Minutes

James Clark	• Use Case description
	  ○ Collecting Resources
	  ○ Managing Property
	  ○ Moving Location
	• Creating draft of Use Case diagram.

Shona Tolan	• Use Case description
	  ○ Enter Players
	  ○ Establish Hotel
	  ○ End Turn
	  ○ End Game
	• Combine use case descriptions / edit in accordance to game theme.

Jonathan Ashe	• Use Case description ○ Roll Dice

	○ Order of Play ○ Choose Option • Create document outlining game theme.

Minutes for Group 24 Week commencing 5 Date of this minute 14/02/19

The following team members were present

Name	Signature
Allen Loyola	Present
James Clark	Present
Shona Tolan	Present
Jonathan Ashe	Present

Agenda:

- Check work from previous meeting and discuss:
  - Final Use Case Descriptions
  - Use Case Diagram Draft
  - Game Theme
  - Research Coding Examples
- Discuss the Use Case sequence Diagrams along with appropriate code that can be written at this time.

Task Reporting (Briefly list the progress for each team member in the last week.*)

Allen Loyola	Minutes submitted
James Clark	First Draft Use Case Diagram submitted
Shona Tolan	Final Use Case Descriptions submitted
Jonathan Ashe	Game Theme document submitted

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

Allen Loyola	• Drafting Code
	  ○ Enter Players
	  ○ Board

James Clark	• Finalise Use Case Diagram • Minutes
Shona Tolan	• Draft Class Diagram • Drafting Use Case Sequence Diagram

Jonathan Ashe	• Drafting code
	  ○ Roll Dice
	  ○ Pay Player

Minutes for Group 24 Week commencing 6 Date of this minute 21/02/19

The following team members were present

Name	Signature
Allen Loyola	Present
James Clark	Present
Shona Tolan	Present
Jonathan Ashe	Present

Agenda:

- Check work from previous meeting and discuss:
  - Final Diagrams
  - Code Classes and Methods
  - JUnit testing begun
  - Minutes
- Divide work equally for next week's meeting

Task Reporting (Briefly list the progress for each team member in the last week.*)

Allen Loyola	-Enter Players class drafted -Board code researched
James Clark	-Minutes Submitted -Use Case Diagram edited and finalised
Shona Tolan	-All Skeleton Code Classes created - Basis of class diagram -Use Case Sequence Diagram drafted
Jonathan Ashe	-Roll Dice method drafted -Pay Player method drafted

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

Allen Loyola	• Move Player Method Drafted
James Clark	• Set-up GitLab • Minutes • End Game Method drafted
Shona Tolan	• Methods PassGO, buyMiningSetup, megaWattUpgrade and establishMaxMiningCapacity drafted
Jonathan Ashe	• Junit Testing on the Order of Play

Minutes for Group 24 Week commencing 7 Date of this minute 28/02/19

The following team members were present

Name (printed/typed)	Signature
Allen Loyola	Present
James Clark	Present
Shona Tolan	Present
Jonathan Ashe	Present

Agenda:

- Check work from previous meeting and discuss:
  - Code Classes and Methods
  - Gitlab set-up on all members computers
  - Testing increased
  - Minutes
- Divide work equally for next week's meeting (The Sprint starts) - regular contact arranged.

Task Reporting (Briefly list the progress for each team member in the last week.*)

Allen Loyola	-Enter Players Class completed -Players Class completed -Move Player method completed
James Clark	-Minutes Submitted -GitLab set-up as a group -End Game Method completed
Shona Tolan	-Remaining Code Classes built/ linked for game play.
Jonathan Ashe	-Junit Tested Order of Play -Creating Players method completed

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

Allen Loyola	• Paired Programming on overall game code
James Clark	• Report compilation(Requirement Analysis) • Redrafting Sequence Diagrams • Minutes
Shona Tolan	• Paired Programming on overall game code • Redrafting UML Class Diagram • UML Sequnce Diagrams for game interactions altered
Jonathan Ashe	• Junit Testing on code

Minutes for Group 24 Week commencing 8 Date of this minute 07/03/19

The following team members were present

Name (printed/typed)	Signature
Allen Loyola	Present
James Clark	Present
Shona Tolan	Present
Jonathan Ashe	Present

Agenda:

- Check work from previous meeting and discuss:
  - UML Class and sequence diagrams
  - Testing and test coverage on code
  - Sign off on coded game
  - Minutes
- Divide work equally for next week's meeting

Task Reporting (Briefly list the progress for each team member in the last week.*)

Allen Loyola	-Enter Players Class improved to avoid players entering the same name etc.
James Clark	-Skeleton of report completed -UML Class diagram finalised -Minutes Submitted
Shona Tolan	-UML Class diagram description drafted
Jonathan Ashe	-Junit Tested more code methods

*Printouts giving an overview of interim deliverables may be added as a supplement to these minutes.

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

Allen Loyola	• Testing code • Photoshopped Technopoly cards (report)
James Clark	• Finalising all diagrams on lucid chart(report) • Written commentary on sequence diagrams • Video of game playthrough • Guide to virtual Board (report) • Minutes
Shona Tolan	• Updated Use Case Descriptions

	• Final checks to all diagrams(report) • Written commentary on Class diagram(report) finalised.
Jonathan Ashe	• Testing of all code • Test Plan finalised(report)

Minutes for Group 24 Week commencing 9 Date of this minute 012/03/19

The following team members were present

Name (printed/typed)	Signature
Allen Loyola	Present
James Clark	Present
Shona Tolan	Present
Jonathan Ashe	Present

Agenda:

- Check work from previous meeting and discuss final hand in:
  - UML Class and sequence diagrams
  - Testing and test coverage on code
  - Sign off on coded game
  - Report finalising and sign off
  - Minutes
- Divide work equally for next week's meeting

Task Reporting (Briefly list the progress for each team member in the last week.*)

Allen Loyola	-Cards photoshopped -Code Tested
James Clark	-Diagrams added to report -Written commentary complete -Video of game playthrough edited/ complete -Guide to virtual board added to report -Minutes finished and added to report
Shona Tolan	-Use Case descriptions updated -Diagrams checked and finalised -Written commentary on class diagrams added to report
Jonathan Ashe	-Testing of code completed on all classes -Finalising Test Plan

*Printouts giving an overview of interim deliverables may be added as a supplement to these minutes.

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

Allen Loyola	• Proof reading of report/ formatting • Group upload of report and code
James Clark	• Final minutes added to report

	• Group upload of report and code
Shona Tolan	• Proof reading of report/ formatting • Group upload of report and code
Jonathan Ashe	• Finalise Test Plan (report) • Group upload of report and code