Matthew McBride, Michael McFerran, Conor McShea, Eoghan O’Connor

CSC7053 Project:  
Save Our Planet

Group 21

# **CSC7053 Peer Assessment: Save Our Planet**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Evaluation** | Group Number: | | | |
| Name | Contribution to team-working and motivation1 | Contribution to documented analysis, design and testing 1,2 | Contribution to working  system code1,2 | **Peer Score**  (Range 85 – 115) |
| Matthew McBride |  | 4 | 4 | 100 |
| Michael McFerran | 4 | 5 | 4 | 100 |
| Conor McShea | 4 | 4 | 3 | 100 |
| Eoghan O’Connor | 4 | 4 | 5 | 100 |

1Values for contribution: 1 = Minimal Contribution; 2 = Reasonable Contribution; 3 = Good Contribution; 4 = Very Good Contribution; 5 = Excellent Contribution

2This 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 the attached submission is my own original work, except for any elements that I have clearly attributed to third parties. I understand that this submission 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 *(use the words shown in the example below!)* |
| Matthew McBride | 11/03/2020 | I agree to the terms of the declaration |
| Michael McFerran | 11/03/2020 | I agree to the terms of the declaration |
| Conor McShea | 11/03/2020 | I agree to the terms of the declaration |
| Eoghan O’Connor | 11/03/2020 | I agree to the terms of the declaration |

Contents

[**CSC7053 Peer Assessment: Save Our Planet** 1](#_Toc34863781)

[**Requirements Analysis** 3](#_Toc34863782)

[**Use Case Descriptions** 3](#_Toc34863783)

[**UML Use case Diagram** 7](#_Toc34863784)

[**Gameboard and fields Guide** 8](#_Toc34863785)

[**Realisation** 10](#_Toc34863786)

[**Sequence diagrams** 10](#_Toc34863787)

[**Design** 20](#_Toc34863788)

[**UML Diagram** 20](#_Toc34863789)

[**Process - Appendices** 22](#_Toc34863790)

[Appendix I 22](#_Toc34863791)

[Appendix II 3](#_Toc34863792)0

[Appendix III 47](#_Toc34863793)

[Appendix IV 52](#_Toc34863794)

# **Requirements Analysis**

# **Use Case Descriptions (EoC, CMcS)**

The use cases presented in this write up were created at the beginning of our group sessions to allow us to get an idea of how we would like to code our game. As we progressed, we altered our use cases to mirror how our game was shaping up.

|  |  |
| --- | --- |
| Flow of events for Register Player use case | |
| **Objective:** | Register player(s) |
| **Precondition:** | The game is loaded. |
| **Main flow:** | 1. The system will request number of players that will be playing. 2. Player 1 will enter name. 3. Player 2 will enter name. 4. Player 3 will enter name. 5. Player 4 will enter name. 6. The system stores the players’ names. |
| **Alternative flows:** | At 1, if an invalid number of players is attempted to be added (below 2 or above 4) an error message will appear, and a valid number will have to be entered to proceed.  At 3, 4 and 5, the system will not allow two or more players to have the same name.  At 4, if only two players are playing use case will continue to 6.  At 5, if only three players are playing use case will continue to 6. |
| **Post Condition:** | The system is ready to begin the display players and resources use case. |
| **Inclusions:** | The display players and resources use case is included. |

|  |  |
| --- | --- |
| Flow of events for Turn Options use case | |
| **Objective:** | Display options available to player |
| **Precondition:** | It’s a player turn |
| **Main flow:** | 1. The game will display all options available to a player. 2. The player will select an option |
| **Alternative flows:** | At 2, if a player selects ‘Take turn’ the game will proceed to the Roll Dice use case.  At 2, if player selects ‘Upgrade property’ the game will proceed to the upgrade property use case.  At 2, if player selects ‘End Game’ the game will proceed to the End Game use case. |
| **Post Condition:** | Player has selected an option. |

|  |  |
| --- | --- |
| Flow of events for Roll Dice use case | |
| **Objective:** | Player rolls dice |
| **Precondition:** | Player choses to take turn. |
| **Main flow:** | 1. The game will roll two dice. 2. The game will display the value of the two dice. |
| **Alternative flows:** | N/A |
| **Post Condition:** | Dice have been rolled and values displayed. |

|  |  |
| --- | --- |
| Flow of events for Move Player use case | |
| **Objective:** | Move player around game board |
| **Precondition:** | 1. Player rolls dice |
| **Main flow:** | 1. The player will move clockwise around the board based on the number they rolled. 2. Player will land at a new square. 3. The game will tell the player where they have landed and what their obligations or opportunities are. |
| **Alternative flows:** | N/A |
| **Post Condition:** | The player will have moved around game board. |
| **Extension Points:** | At 2, if the player lands on or passes the pass go square the game will proceed to the pass go use case.  At 3, if the square is already under ownership the system will proceed to the pay player use case.  At 3, if the area is available for purchase and the player chooses to purchase the area the game will proceed to the buy property use case. |

|  |  |
| --- | --- |
| Flow of events for Pass Go use case | |
| **Objective:** | Increase players resources if they pass go |
| **Precondition:** | 1. The player rolls past or lands on go |
| **Main flow:** | 1. The game will tell the player where they have landed. 2. The player will receive X amount of resources. 3. The game will update the player’s balance. 4. The game will display the player’s new balance and the reason for the change. |
| **Alternative flows:** | N/A |
| **Post Condition:** | The players resources will have increased by X amount. |

|  |  |
| --- | --- |
| Flow of events for Buy Property use case | |
| **Objective:** | Purchase a property. |
| **Precondition:** | 1. The player lands on a property. 2. The property is not already bought. 3. The player has enough resources to purchase property. |
| **Main flow:** | 1. The game tells the player if the property can be bought or if it is already under ownership 2. The player buys the property. 3. The game will deduct the cost of the property from the player’s balance. 4. The player will now have ownership over this property. 5. The game will display the player’s new balance and the reason for the change. |
| **Alternative flows:** | At 1, if the property is under ownership of another player, tax will have to be paid to said player. |
| **Post Condition:** | The player owns the property. |

|  |  |
| --- | --- |
| Flow of events for Upgrade Property use case | |
| **Objective:** | Upgrade a property |
| **Precondition:** | 1. The player must own the entire field 2. It is the players turn. |
| **Main flow:** | 1. The game will display all the properties owned by the player, level of development and their cost to develop. 2. The player selects which property in the field they wish to upgrade. 3. The system will deduct the cost of the development from the player’s resources. 4. The game will display the player’s new balance and the reason for the change. |
| **Alternative flows:** | At 1, if a player does not own any properties, the system will inform the player of this, the system will return the player to their choice of opportunities.  At 2, if the player does not own all the areas in a field, the system will inform the player of this, the system will return the player to their choice of opportunities.  At 2, if three upgrades are already developed on a property and the player wishes to upgrade this property a major upgrade will occur.  At 3, if this upgrade drains a player’s balance to zero, the game will inform the player of this and return the player to their choice of opportunities. |
| **Post Condition:** | The player’s property will be upgraded, and the property will be assigned a new tax cost. |

|  |  |
| --- | --- |
| Flow of events for Pay Player use case | |
| **Objective:** | Pay another player |
| **Precondition:** | 1. Player lands on opponent’s property. |
| **Main flow:** | 1. The game will deduct the cost of landing on an opponent’s area from the players resources. 2. The game will then update the opponent’s resources adding the cost of the player landing on the area. 3. The game will display both player’s new balance and the reason for the change. |
| **Alternative flows:** | N/A |
| **Post Condition:** | Player is deducted of resources; opponent player receives resources. |
| **Extension Points:** | At 1, if the deduction of resources leaves the player with zero balance the system will complete step 2, then the system will proceed to the end game use case |

|  |  |
| --- | --- |
| Flow of events for End Game use case | |
| **Objective:** | End the game |
| **Precondition:** | 1. Player has run out of resources 2. Player no longer wishes to play |
| **Main flow:** | 1. The game will display the winner and their balance. 2. The game will display player rankings |
| **Alternative flows:** | N/A |
| **Post Condition:** | Game over. |

# *A close up of a necklace Description automatically generated***UML Use case Diagram (MMcB, MMcF,CMcS, EOC)**

*Board proposal fig.1*

Above is the final use case diagram for the save our planet game, this diagram models the functionality of the game system moving forward.

The use case diagram shows all the options that are available to the player when interacting with the save our planet application. The first interaction a player will have with the game is to register, as outlined above in the descriptions this method involves selecting number of players and entering player names. Once players have been successfully registered the game is ready to begin.

The next interaction the player has with the game is to select a turn option, from here the player has three options, they can choose to take their turn which invoke the roll dice method, another option is to upgrade a property they own this can also be completed after a player moves and finally they can choose to end the game, this is highlighted above through the extends relationships between the use cases.

When a player rolls the dice they will move around the board, as there is an extends relationship, and from the playerMoves() method, from here there are a number of events which can occur, this is highlighted in the above diagram through the extends relationship, if a player passes go or lands on said square when moving around the board the game will activate the Pass Go method. Another event which could occur is that player wishes to buy the property they land on, triggering the Buy property method and similarly if the player moves around the board and lands on an opponent players property the Pay Player method will be invoked, if a players balance is drained to zero as result of paying another player then the End Game method is invoked as the two objects have an extends relationship. A player can end the game at any time simply by choosing to during their turn options.

# **Gameboard and fields Guide**

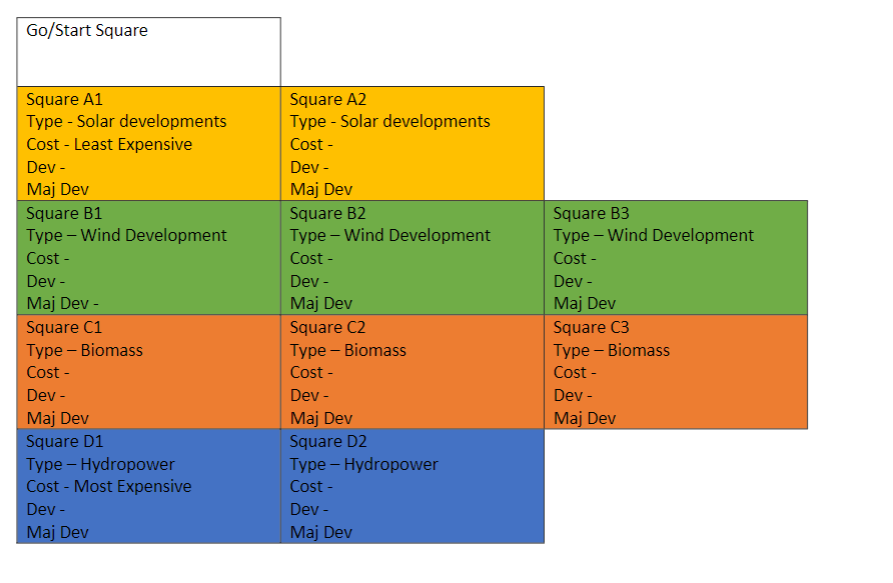
Save our planet is a programmed, simplified version of monopoly that, instead of creating an environment of capitalism and gain, focuses on renewable energy sources, and the management of resources to maintain them, in lieu of using non-renewable methods. The main goal of the game is to acquire as many game squares as possible, buy property and invest in them in the form of square upgrades. The players must also maintain the currency allocated to you in order to continue your chosen schemes, which can be achieved by earning rent from players landing on their purchased game squares. If a player is to run out of resources, the game will end, however should a player wish to end the game, the entire game will come to an end too. Our gameboard has been created as a visual aide for the players, who may wish to follow along while playing, to help simulate a traditional gameboard experience, a pricing grid has also been included to help the player make educated choices when playing.

As the game has been written in the Java programming language, the game has no visual aspect on screen, other than the text output that will be presented within the console. To help visualise the gameboard for the player, a graphic board has been designed by the team, as an aide for the player to reference when playing. This could be used as a game board in conjunction with the programme, or just used as a visual aide, it will be up to the player.

***Field Key***

|  |
| --- |
| Solar Energy |
| Wind Energy |
| Biofuel Energy |
| Hydropower Energy |
| Go/Pause |

***Gameboard square proposal***

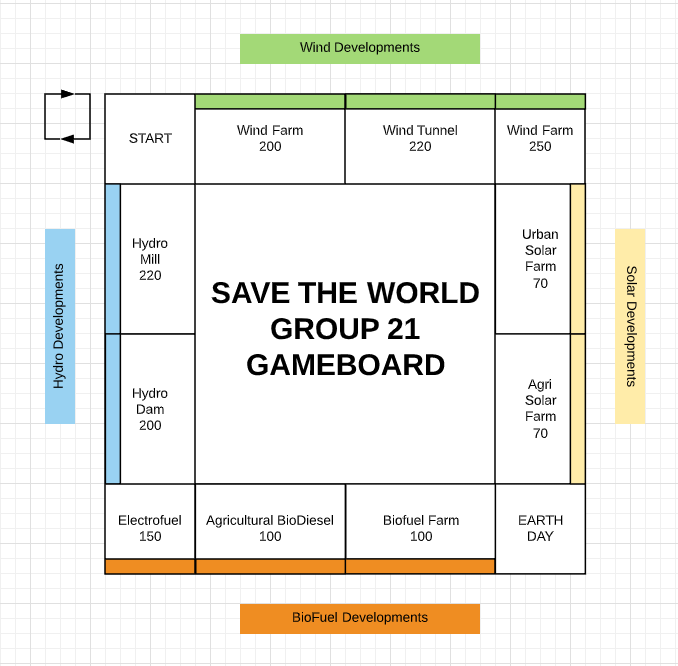
Our initial planning for our game squares started with this square proposal, which was adhered to throughout the development process

*Board proposal fig.2*

**Price Grid**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No.** | **Name** | **Initial  Cost** | **Upgrade  Cost** | **Rent** | **Upgrade 1** | **Upgrade**  **Max** |
| 0 | Go (collect ) | N/A | N/A | N/A | N/A | N/A |
| 1 | Urban Wind Farm | 200 | 125 | 75 | 250 | 500 |
| 2 | Wind Tunnel | 220 | 140 | 75 | 250 | 500 |
| 3 | Offshore Wind Farm | 250 | 150 | 100 | 250 | 500 |
| 4 | Urban Solar farm | 70 | 100 | 60 | 150 | 250 |
| 5 | Solar Energy | 70 | 100 | 60 | 150 | 250 |
| 6 | Pause | N/A | N/A | N/A | N/A | N/A |
| 7 | Biofuel Farm | 100 | 150 | 70 | 175 | 300 |
| 8 | Agricultural Biodiesel | 100 | 160 | 70 | 175 | 300 |
| 9 | Electrofuel | 150 | 175 | 90 | 200 | 300 |
| 10 | HydroDam | 200 | 200 | 100 | 250 | 500 |
| 11 | Hydromill | 220 | 220 | 100 | 250 | 500 |

***Gameboard***



*Gameboard fig.3*

# **Realisation**

# **Sequence diagrams (CMcS)**

**Register player Sequence Diagram**

**A screenshot of a cell phone

Description automatically generated**

Fig.4

The above sequence diagram details the sequence of events and methods that take place to register players into the game. Once the game is run it will prompt the user for the number of players, the first interaction the player has with the game is to enter the number of players playing, the GameManager will then check a valid number of players has been entered. The registerPlayers() method will then loop based on the number of players prompting each player for their name, checking that no two names are the same, once this is complete the system will store the name of each player by adding each player to the players array, finally the player class will return the players and the game is ready to commence.

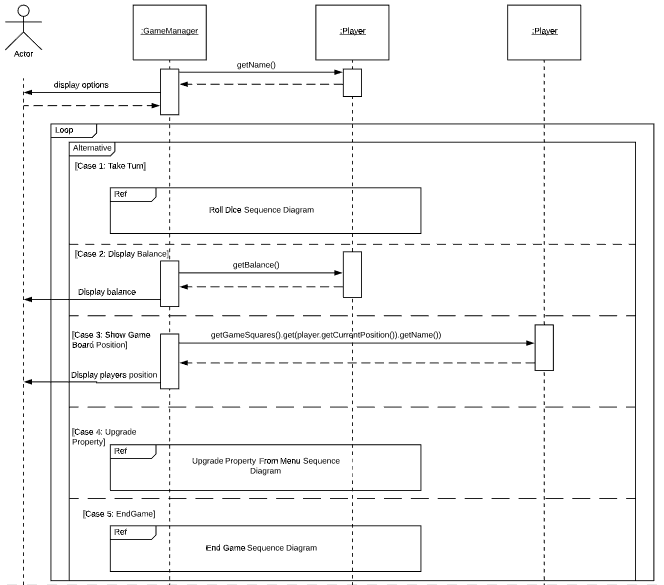
**Turn Options Sequence diagram**

Fig.5

Figure 5 details the sequence of events for turn options use case. The TurnOptions() method is called upon every time it’s a player’s turn, and it displays to the player the options they have. If a player selects to take their turn, then the RollDice() method will be invoked.

The second option is to display balance, the third option is to show the players current position on the gameboard, the fourth option is to upgrade a property and finally the last option available to a player is to end the game. this method will loop through the menu unless the player selects to ‘Take Turn’ or ‘End Game’ in either case the method will break from the loop and the respective methods will be called.

A screenshot of a social media post

Description automatically generated**Roll Dice Sequence Diagram**

Fig.6

The player rolls dice sequence diagram extends on from the turn options sequence diagram as when a player chooses to take their turn, the TurnOptions() method calls upon the PlayerRollsDice() method and from here the game will roll two dice and output the result to the user.

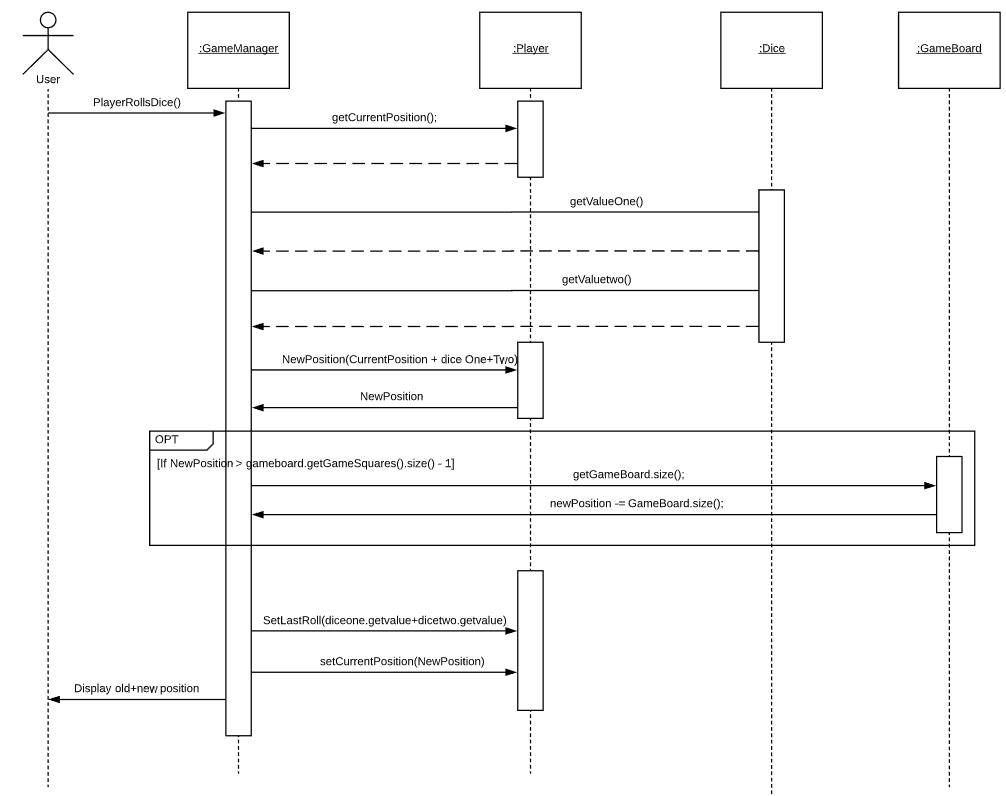
**Move Player Sequence Diagram**

Fig.7

Here we have the sequence of events that occurs when a player moves around the gameboard. The PlayerMoves() method is included from the PlayerRollsDice() method. The GameManager gets the current position of the player and moves the player around the board by adding the two dice values and adding the result to the current position, if a player rolls a 12 the new position of the player is simply reset. The game manager will then store the last roll of the player and will set the current position of the player to the new position they landed on. The GameManager will then display to the user what square they were on and where they have moved too.

**Pass Go Sequence Diagram**

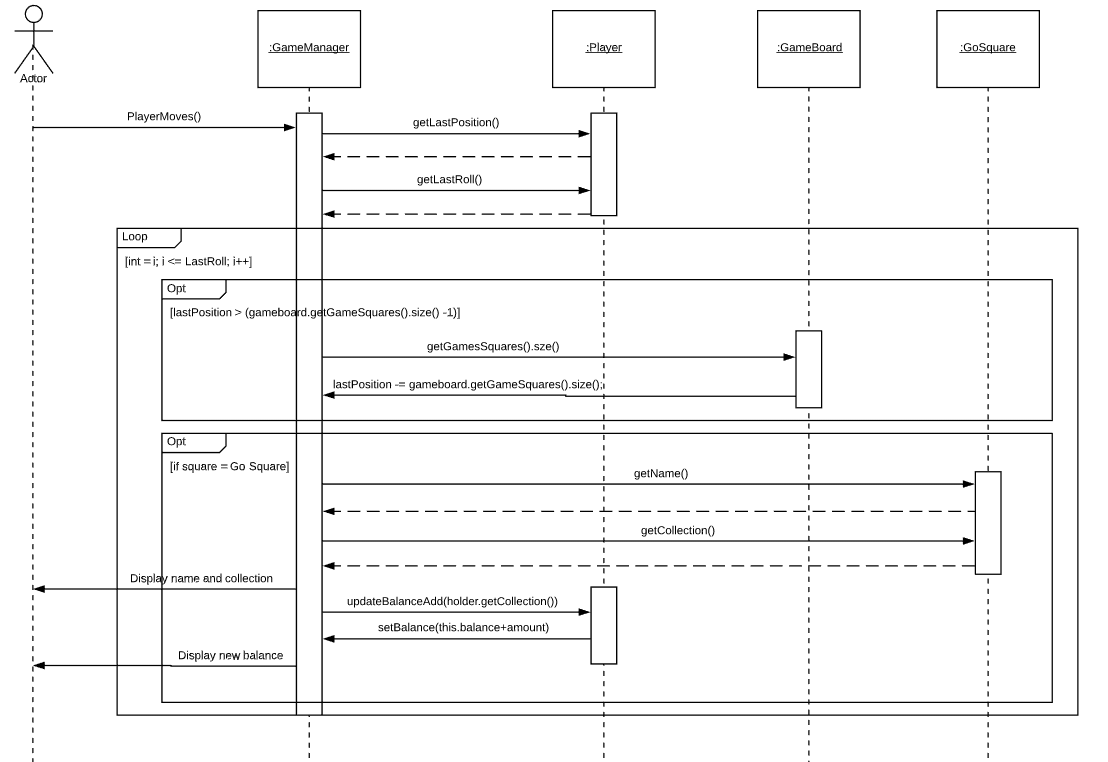


Fig.8

The above sequence diagram details the sequence of events that happens when a player passes go. This event extends from the PlayerMoves() method as a player may pass go on their turn, the GameManager will invoke a CheckIfPassed() method which gets the players last position and last roll, the system will then loop based on the number the player last rolled checking each square, if a player does a loop of the board the last position will reset. However, if a square passed is equal to a go square the GameManager will display to the user that they have passed go. The GameManager will update the balance of the player and will display the players new balance.

A close up of a map

Description automatically generated**Pay Player Sequence Diagram**

Fig.9

The above sequence diagram details the events that occur as a result of paying another player tax. This event also extends from the PlayerMoves() method. When a player moves the GameManager will invoke a CheckWhereLanded() method, this method as detailed above first checks if the square the player landed on is a property square, if the first condition is met a second check will occur to see if the square is owned and also not owned by the player that landed on it, if these conditions are met the GameManager will display the ownership of the square to the player and will invoke the payTax(Player) method. This method displays to the player, who is now the taxpayer, the cost of the tax and to who it is to be paid too. The GameManager will then update the taxpayers balance subtracting the cost of the tax, the players new balance will be displayed. The GameManager will then update the owners balance adding the tax to their current balance, the GameManager will then display the owners new balance.

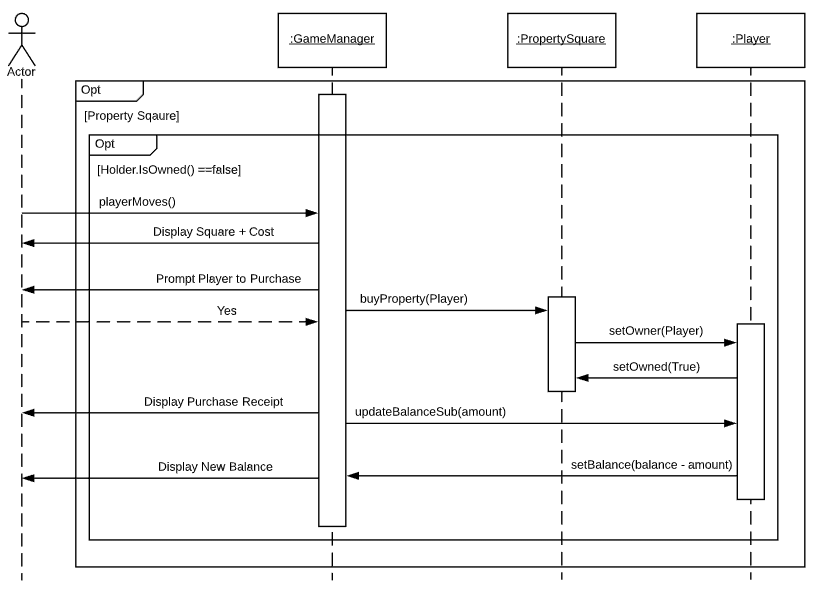
**Buy Property Sequence Diagram**

Fig.10

The above sequence diagram details the events for the buying a property. The player moves and the GameManager invokes the CheckWhereLanded() method. The first check here is that the player has landed on a property square. A second check will then occur, checking that the property square is not owned, when these conditions are met the GameManager will display the name and cost of the property prompting the user to purchase. The buyProperty(Player) method is then invoked which sets ownership of the square to the player. The game will display a receipt of purchase to the player and the updateBalanceSub() method is invoked, subtracting the cost of the property. The GameManager will then display the players new balance.

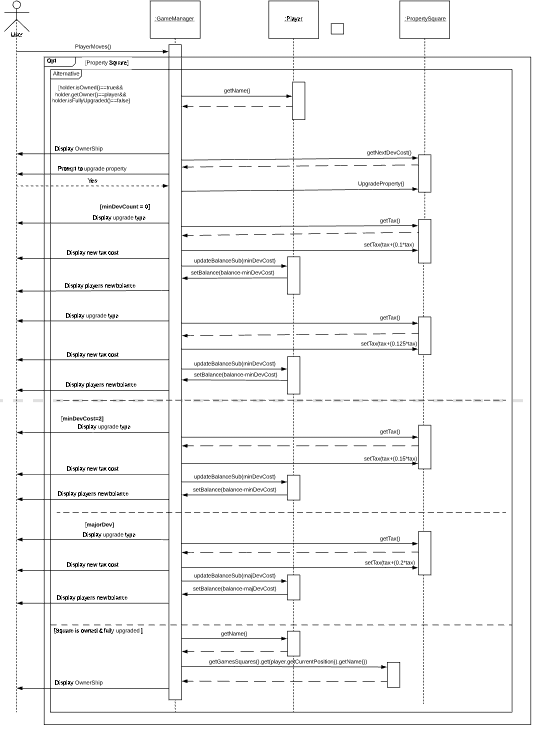
**U****pgrade Property (Player Moves) Sequence Diagram**

Fig.11

The above sequence diagram details all the events that occur when a player wishes to upgrade one of their properties. To upgrade a property the GameManager invokes the CheckWhereLanded() method, here there are several conditions with alternatives. The first condition to be met is that the property square is owned by the player and is not yet fully upgraded. The GameManager will get the square the player is on and the cost to develop and will prompt the player to upgrade the property. The game will then invoke the upgradeProperty() method, and again from here there are several conditions with alternatives. The first being that minDevCount = 0, this means that no upgrades have occurred on the property yet.

The GameManager will display the type of upgrade to the player, and will set a new tax to the property, this new tax cost will be displayed to the player and the game will invoke the updateBalanceSub() method subtracting the cost of the upgrade from the players balance, the GameManager will then display the players new balance.

Two other alternatives to upgrading a property as highlighted above in the sequence diagram is that a property could already be upgraded once or twice in this event the GameManager will go through the same sequence of events as if the property is being upgraded for the first time, however the GameManager will set the tax based on the level of upgrade.

Finally the last alternative is that if the minDevCount is equal to three, then the next upgrade will be a major development again the GameManager will go through the same sequence of events as a minor development, however as highlighted above the tax will increase to that of a major development and the players balance will updated based on the majDevCost for the property. If very first condition highlighted in the sequence diagram is not met this means the square if fully upgraded, in this case the GameManager will not allow the player to attempt any more upgrades, the GameManager will display ownership to the player.

**Upgrade Property From Menu Sequence Diagram**

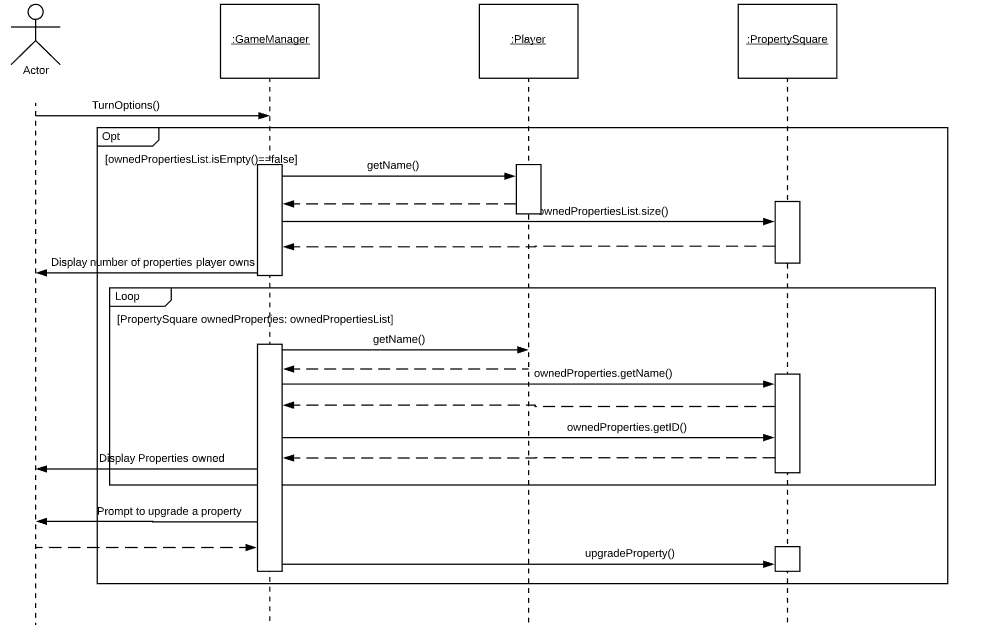


Fig.12

The above sequence diagram details how a player can upgrade a property from the turn options menu. The game first checks that the player owns properties and will then display the number of properties owned to the player. The game will then loop through all the properties a player owns displaying each of them. The game will then prompt the player to upgrade one of their properties. The GameManager will then invoke the upgradeProperty() method and the same sequence of events as highlighted in Fig.11 (Upgrade Property (Player Moves) Sequence Diagram) will take place.

**End Game Sequence Diagram**

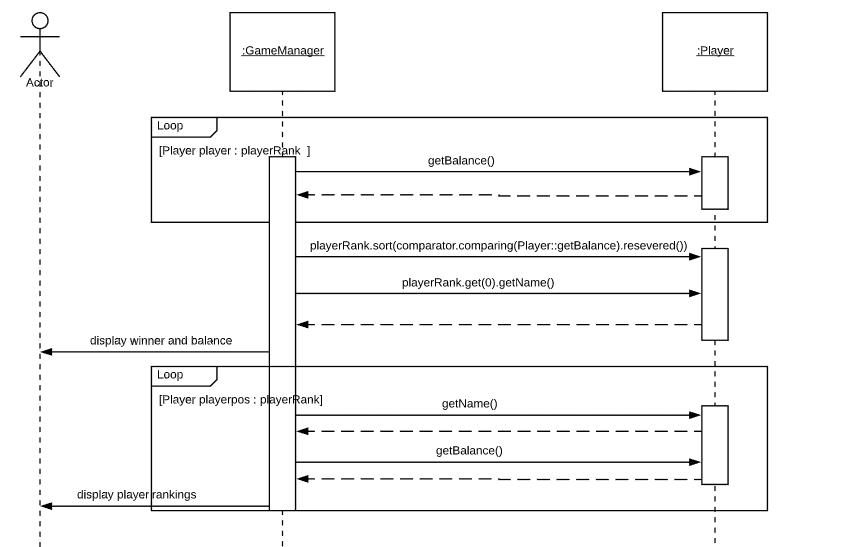
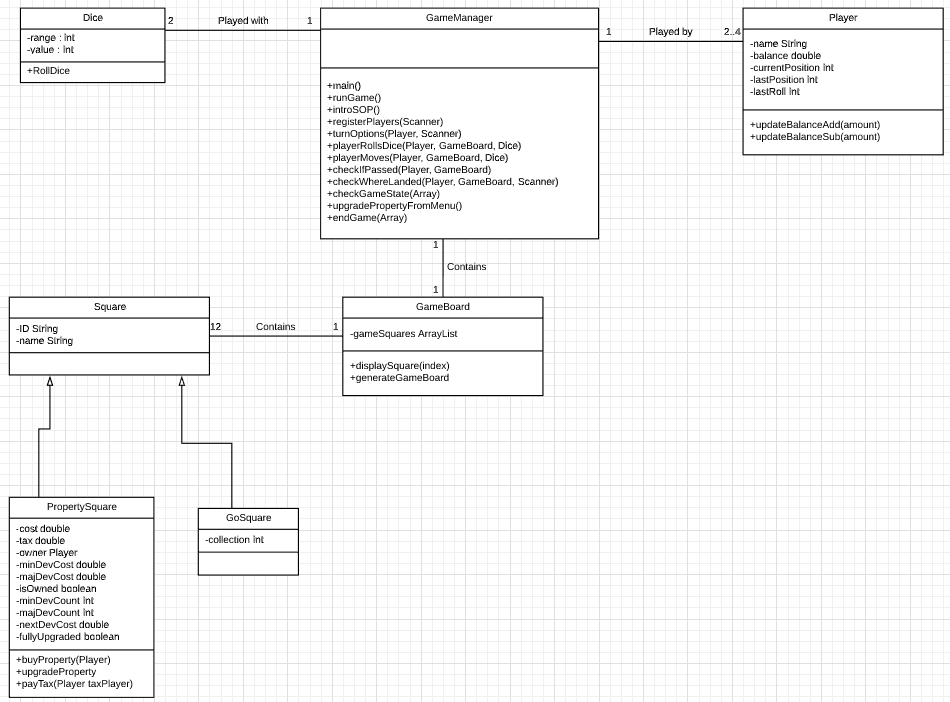


Fig.13

Finally, the last sequence diagram details the endGame() method, this event can extend from the either the turn options (player choses to end game) or payPlayer() methods. Here an array is created called playerRank and the GameManager will loop and get each players balance and add it to the playerRank array. The array will be sorted and the GameManager will get the player with the highest balance and will display this player and their balance as the winner. The GameManager will loop again displaying the player rankings. The game will then have ended.

# **Design**

# **UML Diagram (MMcB)**



UML Class connection fig.14

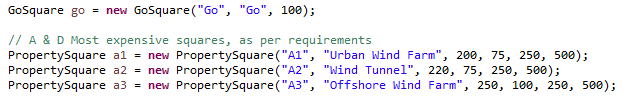
As the UML diagram shows, the game we have created contains seven classes. These classes are:

* GameManager
* Dice
* Player
* Square
* GoSquare
* PropertySquare
* Gameboard

Initially we had tried to utilise interfaces within our game build, but they proved to be too troublesome and not essential for what we were trying to achieve. Another aspect we kept in mind during the creation phase, was the potential requirement for editing of the game’s characteristics, should a customer wish for changes to occur in the future.

One such example for accessibility with the code, is the way in which we handled the ‘GameBoard’ class. When conceptualizing the method in which we would store our gameboard, we opted to use an ArrayList to store them easily. While following the requirements set out for us at the start, we only added twelve squares as per requested. However, by using an ArrayList, the ability to add more squares to the gameboard is possible and easy to manage, due to their dynamic nature and ability to grow without any damaging features to the relating code. This eliminated the necessity to create more classes, should the customer want a bigger game board in the future.

With inheritance in mind for any future changes that may be made to the game if needed, as a group we decided to use an OOP solution when creation the gameboard, starting with the Square class. With this class, we were able to create basic methods and starter variables, that could be inherited to the subclasses that would utilise them. The two subclasses in relation to the Square class, are PropertySquare and GoSquare. These classes each have their own methods, with business rules built into their setters, that draw from the set structure of the Square class, namely the ‘ID’ and ‘Name’ String variables. For example, should the customer wish to alter the amount of resources granted to the players during the game, a simple change of the variable is possible in the GameBoard class. A conscious decision to not cement the variables of the gameboard was made with the thought of future implementation and alteration in mind, and the ability to do so with ease. If in the future a customer did wish to add further square types to the game, the constructor can be easily utilised.

 Fig.13 Example of our GameBoard creating using constructor with args

Our Dice class has adhered to the requirement of two dice being used in play, to a maximum of twelve possible in a max roll. However, should this need changed in the future, the range can be easily altered within the Dice class, as it has been set by using a final Int. Although it is currently set at six, a change would have instant impact with no detriment to the involved code that makes use of this variable.

Our GameManager class hosts many of the methods required for the game to function, acting as an engine of sorts for the game. These methods inter link in a manner that allows for the game to run efficiently. In order to make sure that the game can store the information needed for the game to play, an Array has been utilised to store player names, starting resource amount and starting position, position being important as this ensures that all players start the game on the ‘Go’ square.  
Validation has been adhered to within this class, as we made sure to include checks that do not allow for the duplication of player names, and count checks that only allow for the player to complete three minor upgrades and one major upgrade. A player count of two to four players has also been included, with print out messages informing players of invalid inputs. This has been achieved by using a final Int within the Player class titled playerNumberMax, set at four, and a switch case within our registerPlayer method in the GameManager class beginning at two, stopping the game being played solo. Although this has been put in place to ensure that requirements have been met, the code could be easily altered in the future should they change.

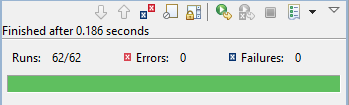
# **Process - Appendices**

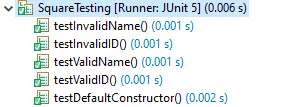
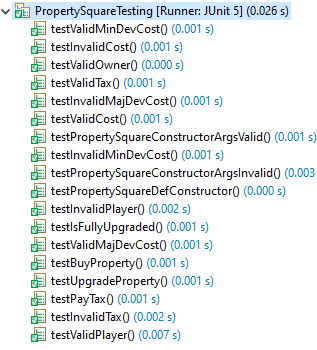
# Appendix I

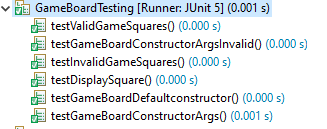
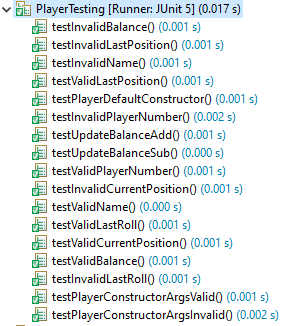
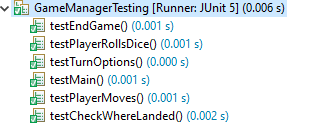
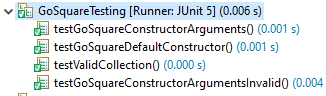
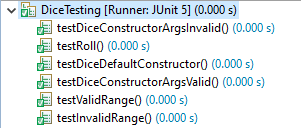
**Junit (MMCF)**

Junit testing was performed on our code to ensure that all possible outcomes we have predicted have been covered.

Successful Tests:



Tests Performed:



**Acceptance testing (EoC)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Use Case** | **Test Description** | **Test Initialization** | **Test inputs** | **Test Procedure** | **Expected Result** | **Pass/Fail** |
| **1** | Register Players | Testing player registration | Run game | A valid number of players (2 – 4) | Entering ‘4’ as the number of players | Value accepted; continue with the registration process entering p1’s name | Pass |
| 1b | Register Players | Testing player registration | Run game | An invalid number of players | Entering ‘6’ as the number of players | Value failed, user is prompted to re-enter a valid number of players | Pass |
| 1c | Register Players | Testing player registration | Run game | Entering a string for number of players | Entering ‘five’ as the number of players | Value failed, user is prompted to re-enter a valid number of players | Pass |
| **2** | Register Players | Testing player registration, entering player names | Run game, number of players set to ‘4’ | Entering player names  1: player1  2: player2  3: player3  4: player4 | Enter the player names when prompted | Values accepted; game will formally begin a with player1’s turn | Pass |
| 2b | Register Players | Testing player registration, entering player names | Run game, number of players set to ‘4’ | Entering player names  1: player1  2: player1  3: player3  4: player4 | Enter the player names when prompted | Value for player 2’s name will fail as it is a duplicate, user will be notified and prompted to re-enter a name for player 2 | Pass |
| 2c | Register Players | Testing player registration, entering player names | Run game, number of players set to ‘4’ | Entering player names  1: player1  2: player2  3: player2  4: player4 | Enter the player names when prompted | Value for player 3’s name will fail as it is a duplicate, user will be notified and prompted to re-enter a name for player 3 | Pass |
| 2d | Register Players | Testing player registration, entering player names | Run game, number of players set to ‘4’ | Entering player names  1: player1  2: player2  3: player3  4: player3 | Enter the player names when prompted | Value for player 4’s name will fail as it is a duplicate, user will be notified and prompted to re-enter a name for player 4 | Pass |
| **3** | Register Players | Testing player registration, entering player names | Run game, number of players set to ‘2’ | Entering player names  1: player1  2: player2 | Enter the player names when prompted | Values accepted; game will formally begin a with player1’s turn | Pass |
| 3b | Register Players | Testing player registration, entering player names | Run game, number of players set to ‘2’ | Entering player names  1: player1  2: player1 | Enter the player names when prompted | Value for player 2’s name will fail as it is a duplicate, user will be notified and prompted to re-enter a name for player 2 | Pass |
| **4** | Register Players | Testing player registration, entering player names | Run game, number of players set to ‘3’ | Entering player names  1: player1  2: player2  3: player3 | Enter the player names when prompted | Values accepted; game will formally begin a with player1’s turn | Pass |
| 4b | Register Players | Testing player registration, entering player names | Run game, number of players set to ‘3’ | Entering player names  1: player1  2: player1  3: player3 | Enter the player names when prompted | Value for player 2’s name will fail as it is a duplicate, user will be notified and prompted to re-enter a name for player 2 | Pass |
| 4c | Register Players | Testing player registration, entering player names | Run game, number of players set to ‘3’ | Entering player names  1: player1  2: player2  3: player2 | Enter the player names when prompted | Value for player 3’s name will fail as it is a duplicate, user will be notified and prompted to re-enter a name for player 3 | Pass |
| **5** | Turn Option | Testing UI and functionality of options for the players turn | Players have been registered | Entering player names  1: player1  2: player2  Selecting option 1 “Take Turn” | Enter the player names when prompted, selecting option 1 “Take Turn” | The system will display the options of the current players turn, option 1 will be selected and the system will continue to roll dice and move the player. | Pass |
| **5b** | Turn Option | Testing UI and functionality of options for the players turn | Players have been registered | Entering player names  1: player1  2: player2  Selecting option 2 “Display Balance” | Enter the player names when prompted, selecting option 2 “Display Balance” | The system will display the options of the current players turn, option 2 will be selected and the system will display the current players balance, returning to the turn options menu. | Pass |
| **5c** | Turn Option | Testing UI and functionality of options for the players turn | Players have been registered | Entering player names  1: player1  2: player2  Selecting option 3 “Show Position” | Enter the player names when prompted, selecting option 3 “Show Position” | The system will display the options of the current players turn, option 3 will be selected and the system will display the current player position, returning to the turn options menu. | Pass |
| **5d** | Turn Option | Testing UI and functionality of options for the players turn | Players have been registered | Entering player names  1: player1  2: player2  Selecting option 4 “Upgrade Property” | Enter the player names when prompted, selecting option 4 “Upgrade Property” | The system will display the options of the current players turn, option 4 will be selected and the system will prompt the user to which property they wish to upgrade and apply the upgrade, returning to the turn options menu. | Pass |
| **5e** | Turn Option | Testing invalid input for player options | Players have been registered | Entering player names  1: player1  2: player2  Selecting option 5 “Quit Game” | Enter the player names when prompted, selecting option 5 “Quit Game” | The system will display the options of the current players turn, option 5 will be selected and the system will end the game, printing the player rankings. | Pass |
| **6** | Roll Dice | Testing player rolls dice | Run game with 2 players | Entering player names  1: player1  2: player2 | On player1’s turn, option take turn is selected | The system will prompt the user of what numbers they have rolled. | Pass |
| **7** | Move Player | Testing move player | Run game with 2 players | Entering player names  1: player1  2: player2 | On player1’s turn, option take turn is selected and dices have been rolled | The system will prompt the user of where they have moved too. | Pass |
| **7b** | Move Player | Testing check player options | Run game with 2 players | Entering player names  1: player1  2: player2 | On player1’s turn, option take turn is selected, dices have been rolled and they have moved position | The system will prompt the user as to what their options are in their new position I.e. buy property, upgrade property, pay another player | Pass |
| **8** | Buy Property | Testing check player options - Landing on an unowned property | Run game with 2 players | Entering player names  1: player1  2: player2 | On player1’s turn, option take turn is selected, dices have been rolled and they have moved position to a free property | The system will prompt the user that they have landed on an unowned square and ask if they would wish to purchase it. If Yes, the user will purchase the property and their balance will update, if No, their turn will end. | Pass |
| **9** | Pay Player | Testing check player options - Landing on another players owned property | Run game with 2 players, preassign property's to player2 | Entering player names  1: player1  2: player2 | On player1’s turn, option take turn is selected, dices have been rolled and they have moved position to an owned property. | The system will prompt the user as to who’s property they have landed on and what they owe that player, proceeding with a payment being taken. | Pass |
| **10** | Upgrade Property | Testing check player options - Landing on a player owned property | Run game with 2 players, preassign property's to player1 | Entering player names  1: player1  2: player2 | On player1’s turn, option take turn is selected, dices have been rolled and they have moved position to a property they own. | The system will prompt the user as to where they have landed, that they own that property and ask if they wish to upgrade the property. If Yes, the user will purchase the upgrade and their balance will update. If No, their turn will end. | Pass |
| **10b** | Upgrade Property | Test upgrading a player's property from the turn options menu | Run game with 2 players, preassign property to player1 | Entering player names  1: player1  2: player2 | On player1’s turn option upgrade property is taken. | The system will prompt the user as to which property they would like to upgrade, once selected the property will be upgraded and the users balance will be printed. | Pass |
| **10c** | Upgrade Property | Testing check player options - Landing on a player owned property | Run game with 2 players, preassign property's to player1 with 3 minor developments | Entering player names  1: player1  2: player2 | On player1’s turn, option take turn is selected, dices have been rolled and they have moved position to a property they own | The system will prompt the user as to where they have landed, that they own that property and ask if they want to apply a final upgrade to the property. If Yes, the user will purchase the upgrade and their balance will update. If No, their turn will end. | Pass |
| **11** | Pass Go | Testing check if passed | Run game with 2 players | Entering player names  1: player1  2: player2 | On player1’s turn, option take turn is selected, dices have been rolled and they have moved passed the collection point | The system will prompt the user that they have passed the collection point and the amount that they are due, the users balance will then update. | Pass |
| **12** | End Game | Testing the implantation of ending the game | Run game with 2 players | Entering player names  1: player1  2: player2 | One of the player’s balances is zero or below | The system will prompt that the game has ended, displaying a score list showing which player have ranked 1st, 2nd 3rd etc. | Pass |
| **12b** | End Game | Testing the implantation of ending the game | Run game with 2 players | Entering player names  1: player1  2: player2 | One of the player’s has selected the quit game option form the menu. | The system will ask the user if they are sure they wish to quit the game, if yes the system will prompt that the game has ended, displaying a score list showing which player have ranked 1st, 2nd 3rd etc. | Pass |

# Appendix II

**Weekly Minutes - (MMcF)**

|  |  |
| --- | --- |
| Name (printed/typed) | Signature |
| MICHAEL MCFERRAN | MICHAEL MCFERRAN |
| EOIN O’CONNOR | EOIN O’CONNOR |
| MATT MCBRIDE | MATT MCBRIDE |
| CONNOR O’SHEA | CONNOR O’SHEA |

**Minutes for Group 21 Week commencing 19/01/20 Date of this minute 21/01/20  
The following team members were present:**

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

* Not Applicable, as this is our first meeting

Discussion

* Went over game rules and initial game design.
* User Stories, game board Layout based on renewable sources (A,B, C, D sections).
* Potential currency being energy.
* Referencing project specification to make decisions on player count(4), game board most expensive squares(for development) on sections with 2 squares, game ends with last player left with resources(energy).
* Future feature of project to be added; interactivity/educational aspect to give facts about renewable energy when a square is landed upon.

GameBoard Layout

* 10 squares and 1 go square.
* 4 sections/fields ( need all squares in a field to develop).
* Square naming structure(consistency for programming transparency); A1, A2, B1, B2, B3, C1, C2, C3, D1, D2. Sections A and B being the most expensive.

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

Name & Role (1): Michael

* Research on solar section .
* Potential facts interesting to users.
* Naming of this square section.

Name & Role (2): Matt

* Research on Hydro Section.

Name & Role (3): Eoin

* Research on wind research.
* Cost of development – percentage based on players resources.

Name & Role (4): Connor

* Research on biomass section.

General group consensus is to individually get more familiar with project specification before next week, and to get to grips with the key focus areas of the groupwork. A key to note is being able to develop in an agile methodology for efficiency. We are all going to individually begin work on use case modelling before collaborating on this next week.

**Minutes for Group 21 Week commencing 26/01/20 Date of this minute 28/01/20  
The following team members were present:**

|  |  |
| --- | --- |
| Name (printed/typed) | Signature |
| MICHAEL MCFERRAN | MICHAEL MCFERRAN |
| EOIN O’CONNOR | EOIN O’CONNOR |
| MATT MCBRIDE | MATT MCBRIDE |
| CONNOR O’SHEA | CONNOR O’SHEA |

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

Name & Role (1): Michael

* Worked on two solar squares, developed story for each;

Solar Home Square – incentivised install of solar panels on home (gains user money each roll?)

Solar Thermal Plant square- Ivanpah Solar Electric Generating System power plant (Mojave desert) with facts of operational capacity for future educational development

* Developed User cases

Name & Role (2): Matt

* Worked on User Case
* hydro energy section

Name & Role (3): Eoin

* User Cases
* Wind section research

Name & Role (4): Connor

* Biomass section research
* Use Cases

We collaborated on use cases together during our meeting and after. Without this, we can not proceed with developing a code solution to the project effectively. On the 30/01/20 Michael and Eoin attended a laboratory session to focus our efforts and see how we should successfully approach the projects next stages. Some notes from this are as follows;

* Focus on user actions for use case diagrams
* Alternate flows on sequence diagrams for paying user credit
* All user-based actions are stemmed from user rolling dice (turn based)
* Functions/methods are automatic, system takes care of them and user is blissfully unaware of how things operate in the background
* Basic flow of game : register players, Turn based on a loop of players, they have options to display balance, show gameboard position, end game, or take their turn, then they roll their dice and can end their turn by choosing to buy a property square if they land on one. Players start out with a set balance; they can upgrade their properties and pay a tax when land on another players property square. IF no one chooses to end the game prematurely, the game is won when only one person has money left.

Note- \*As discussed in sprint retrospective, story cards could have greatly improved our efficiency for requirements and hence use cases\*

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

Name & Role (1): Michael

* Coding – work on developing dice class and method to create random values for a player turn.
* Coding – work on player class
* Also: Getting familiar with gitlab to teach group how to use it, promotes collaborative programming

Name & Role (2): Eoin

* Game Board pricing
* Game board layout and object breakdown
* Code- begin on creating a structure for squares, property square and go square

Name & Role (3): Matt

* Working on game rules, based on project spec and group evaluation
* Working on use cases

Name & Role (4): Connor

* Story development – currency, developments, monopoly game structure
* Working on use cases

**Minutes for Group 21 Week commencing 02/02/20 Date of this minute 06/02/20  
The following team members were present:**

|  |  |
| --- | --- |
| Name (printed/typed) | Signature |
| MICHAEL MCFERRAN | MICHAEL MCFERRAN |
| EOIN O’CONNOR | EOIN O’CONNOR |
| MATT MCBRIDE | MATT MCBRIDE |
| CONNOR O’SHEA | CONNOR O’SHEA |

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

Name & Role (1): Michael

* GitLab, figuring out commits and how to implement collaborative work using the GIT GUI/ GIT BASH software.
* Code - Wrote a class for Dice, and how two random dice can generate a number to determine how far a player rolls on the board (typically 6 sided).
* Worked on player class and player properties.

Name & Role (2): Eoin

* Game Board Pricing .
* Developed a chart for cost on each square to develop and further develop.
* Created a super class of square, where property square and go square are sub classes of.

Name & Role (3): Matt

* Game Rules page written up.
* Figured out how to stop two players having the same name in registering player method.

Name & Role (4): Connor

* Made a lot of developments on creating use cases .
* Worked on story development.

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

Name & Role (1): Michael

* Do some light JUNIT testing of classes. So far this can only really be tests on class attributes and any created methods.
* Code- figure out how to move player around a gameboard using the dice class.

Name & Role (2): Eoin

* Continue coding- developing main method for player options, taking in from scanner ( switch cases and Boolean gates)
* Programming – index for game position, array for player position. Possible modulus operator to return index to zero for each iteration.

Name & Role (3): Matt

* Developing requirements analysis of report with gameboard layout, rules and such
* Final use cases.
* Move on to other sections of report.

Name & Role (4): Connor

* Finishing up use case diagrams.
* Starting UML diagrams.

**Minutes for Group 21 Week commencing 09/02/20 Date of this minute 13/02/20**  
**The following team members were present:**

|  |  |
| --- | --- |
| Name (printed/typed) | Signature |
| MICHAEL MCFERRAN | MICHAEL MCFERRAN |
| EOIN O’CONNOR | EOIN O’CONNOR |
| MATT MCBRIDE | MATT MCBRIDE |
| CONNOR O’SHEA | CONNOR O’SHEA |

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

Name & Role (1): Michael

* Tested gitlab directory.
* Taught group how to use gitlab to push and pull from master branch to local machine.

**EXAM WEEK**

The entire group have continued working on the previous weeks stuff, but we have decided to postpone progress and our focus is on studying for exams this week .

**Minutes for Group 21 Week commencing 16/02/20 Date of this minute 20/02/20**

**Sprint 1:**  
**The following team members were present:**

|  |  |
| --- | --- |
| Name (printed/typed) | Signature |
| MICHAEL MCFERRAN | MICHAEL MCFERRAN |
| EOIN O’CONNOR | EOIN O’CONNOR |
| MATT MCBRIDE | MATT MCBRIDE |
| CONNOR O’SHEA | CONNOR O’SHEA |

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

Name & Role (1): Michael

* Requirements analysis- Updating game rules based on game requirements .
* Code – worked on main class with Eoin to take in user options from console .
* Code – figured out how to move player around a gameboard using the dice class.
* Code- research/how to implement requirements we don’t have yet (upgrade any player owned properties).

Name & Role (2): Matthew

* Code – Confirming basic methods meet game requirements .
* Requirements analysis – Working out board pricing for starting balance, purchasing squares, upgrades and so on .
* Requirements analysis – Finalising gameboard layout, titles, development fields and such.

Name & Role (3): Eoin

* Code – General creation of skeleton classes for code, testing of possible implementation of gameboard objects, player creation.
* Code – Created a registering player method in main game running class.

Name & Role (4): Connor

* Realisation – working on report.
* Reviewing our approach to educational benefits to the game.

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

Name & Role (1): Michael

* Realisation – Working on sequence Diagram review.
* Code – continuing researching how to implement requirements we don’t have yet.

Name & Role (2): Matt

* Requirements analysis – create structure and gather work on this section of report.
* Realisation and design – Working on finishing UML diagrams for report.

Name & Role (3): Eoin

* Code – create a player turn method to allow player to choose what to do on their turn.
* Code – create a player choice method that takes decisions from player turn method and does these tasks.
* Requirements Analysis – work on use cases.

Name & Role (4): Connor

* Requirements Analysis – work on use cases.
* Realisation – work on sequence diagrams.

**Minutes for Group 21 Week commencing 23/02/20 Date of this minute 27/02/20**

**Sprint 1**  
**The following team members were present:**

|  |  |
| --- | --- |
| Name (printed/typed) | Signature |
| MICHAEL MCFERRAN | MICHAEL MCFERRAN |
| EOIN O’CONNOR | EOIN O’CONNOR |
| MATT MCBRIDE | MATT MCBRIDE |
| CONNOR O’SHEA | CONNOR O’SHEA |

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

Name & Role (1): Michael

* Realisation - First pass review of sequence diagrams (must reflect changes in code ).
* Code - Figured out how to implement a player upgrade owned property(s) method during a player roll.
* Code -Fixed some issues in code that prevented it from correctly working.

Name & Role (2): Matt

* Code - Side branch of code to rename/ camelCase, comments, organise flow before pushing to master branch.
* Requirements analysis- gathered work for this section of report .

Name & Role (3): Eoin

* Code – created the player turn method .
* Code- created player choice method by passing in switch cases from player turn.
* Requirements Analysis – worked on use cases.

Name & Role (4): Connor

* Requirements analysis section of report updated with.
* Use Case Descriptions.
* Realisation- Working on updating sequence diagrams / new ones based on new code.

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

Name & Role (1): Michael

* Finish Writing Player upgrade owned property method Requirement by creating players owned properties array in player class.
* Realisation - Working on Sequence Diagram review.

Name & Role (2): Matthew

* Design - Work on UML diagrams.
* Code - Gameboard class/ object creation (ArrayList of squares/property squares/go square).

Name & Role (3): Eoin

* Code – flesh out player turn and player choice methods to display balance, take turn etc based on players decision.

Name & Role (4): Connor

* Requirements Analysis - Finish Use cases.
* Realisation – finish newly updated sequence diagrams.

**Sprint 1 brief review**

At the end of first sprint the code runs but doesn’t implement all requirements and has errors, this will be corrected in final sprint. The report is taking shape, but the key is collaboration is crucial to the agile workflow, and it has greatly improved our efficiency. Moving into final sprint (actual sprint) goal is to finalise game and have it working 100% with report as well completed to our best abilities. Sprint 1 was more of a test to ease use into an agile work process and we will be incorporating daily scrums to keep us on track in the final sprint.

**Minutes for Group 21 Week commencing 01/03/2020 Date of this minute 05/03/2020**

**Sprint 2**  
**The following team members were present:**

|  |  |
| --- | --- |
| Name (printed/typed) | Signature |
| MICHAEL MCFERRAN | MICHAEL MCFERRAN |
| EOIN O’CONNOR | EOIN O’CONNOR |
| MATT MCBRIDE | MATT MCBRIDE |
| CONNOR O’SHEA | CONNOR O’SHEA |

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

Name & Role (1): Michael

* Code - Added player upgrade property method.
* Code – reviewed code and tidy up loose ends.
* Code – tested some aspects of code operation ( as user and how they expect it to run).
* Finished reviewing sequence diagrams.

Name & Role (2): Matt

* Design – finished UML diagrams.
* Code – created gameboard class to store square/ property square(extends square), go square.

Name & Role (3): Eoin

* Code - Fully functional player turn menu with correct implementation of player choice .

Display balance, take turn, end game, check game position .

Name & Role (4): Connor

* Requirements Analysis - finished Use cases.
* Realisation – finished newly updated sequence diagrams.

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

Name & Role (1): Michael

* Finish JUNIT testing for all classes.
* Write method for game intro.
* Fix upgrade property method from menu to allow it to loop back to menu if player has no property or if player upgrades.
* Code - Fix issue where end game isn’t triggered when player balance drops below zero as result of paying tax to another player .
* Code – fix pause square (currently works as a go square with collection of zero ?)
* Appendix - Type up these weekly minutes for report

Name & Role (2): Eoin

* Process – test plan creation and acceptance testing
* Realisation – final review and edit of sequence diagrams

Name & Role (3): Matt

* Gathering report .
* Fixing issues with balancing (starting balance and such) of the game.
* Getting information about environmental causes for the game intro.
* Code – create end game method when triggered by a player reaching zero balance or pressing end game during their turn in the options menu.
* Requirements Analysis – update pricing structure of game for balance and flow of game.
* Update gameboard based on environmental theme.
* Edit write up based on game changes.

Name & Role (4): Connor

* Finish Sequence diagrams.
* Put appendix write up together, gather screenshots of code and such.

**Minutes for Group 21 Week commencing 08/03/2020 Date of this minute 10/03/2020**

**Sprint 2**  
**The following team members were present:**

|  |  |
| --- | --- |
| Name (printed/typed) | Signature |
| MICHAEL MCFERRAN | MICHAEL MCFERRAN |
| EOIN O’CONNOR | EOIN O’CONNOR |
| MATT MCBRIDE | MATT MCBRIDE |
| CONNOR O’SHEA | CONNOR O’SHEA |

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

Name & Role (1): Connor

* Edited and finished sequence diagrams ; Sequence diagrams to include new method for player upgrading owned properties during their turn
* Finished appendix type up and layout

Name & Role (2): Matt

* Gathering report, fixed acceptance testing- balancing starting values for player balance, upgrade prices.

Of squares etc. Got next dev cost to display correctly when asking user if they want to upgrade an owned property.

* Code – end game method is fully functional.
* Requirements analysis - fixed game balance.
* Edited write up and updated gameboard based on environmental theme.
* Passed into Michaels intro method, a series of facts and dialogue on the environmental cause of the game.

Giving the user some context on why and what they are doing. Improving the educational possibilities of the game.

Name & Role (3): Eoin

* Finished Test plan + acceptance testing.
* Finished review of sequence diagrams.

Name & Role (4): Michael

* Fixed JUNIT class testing to ensure they all ran successful.
* \*added functional requirement to code\*- that allows player to upgrade any owned property square during their turn, irrespective of if they are on that square or not.
* Fixed all issues of code functionality when running to ensure minimal errors user will experience and ensure user acceptance testing result will be acceptable. For example, when a player tries to upgrade a square when they own none, a printout will let them know this and loop back to the menu. Or if a player upgrades a property, it will then loop back to the menu and allow the player to take their turn, or display balance and such before taking their turn.
* Wrote an intro method for the game with a printed logo, layout and such.
* Fixed issue where end game wasn’t triggered after a player paid tax and balance dropped below zero ( final int balance minimum needed to be well below zero so check could be made and trigger end game).
* Code – fixed issue with the Pause square, and implemented it in gameboard class as an object solely for
* Appendix – typed up these minutes for report.

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

Name & Role (1): Michael

* Type up sprint log/ scrum meeting reports.
* Hartmann Orona Spreadsheet(with Matt).
* Sprint retrospective meeting with objectives, and outcome of sprints / group project.

Name & Role (2): Matt

* Finish video of game.
* Finalise Report/ check over it all.
* Upload code.

Nothing else left to do apart from Sprint retrospective meeting.

**Sprint 1 + 2 Retrospective Meeting Date 11/03/20**

|  |  |
| --- | --- |
| Name (printed/typed) | Signature |
| MICHAEL MCFERRAN | MICHAEL MCFERRAN |
| EOIN O’CONNOR | EOIN O’CONNOR |
| MATT MCBRIDE | MATT MCBRIDE |
| CONNOR O’SHEA | CONNOR O’SHEA |

**The following team members were present:**

Time spent on Meeting - 30 minutes

**Objectives**

1. What went well during the sprint?
2. What issues did we have?
3. What could be improved in the next sprint?

**Discussion**

1. The purpose of agile methodologies is to promote collaborative working, and a sprint approach was a great learning curve that has prepared us for working life. By design, sprints have daily scrum meetings to itemise and structure workflow. We can attest to this as our productivity improved greatly. We all agree that our efficiency in the second sprint was exponentially better than the hap hazard adhoc approach we took earlier in the project. By having documented product backlog and saying in person what we are working on and have worked on daily, gives us accountability and no excuse to have been slacking.

We were able to come together and deliver a working console-based game that is both educational and highly functional, we learned a lot about software engineering as a result and improved our problem-solving skills. Our confidence in ourselves has increased and gives us all a sense of satisfaction in our combined efforts.

1. The only major issue we really had was with Gitlab, and it is something we all agree on. Being that we have never used it before, it initially slowed us down while getting used to GIT GUI/ GIT BASH software. The amount of commits we had as a result were lower than expected, especially from Michaels point of view. “Differences in remote version and local version let to merge issues when pushing changes to the server-side code. Leading me to be afraid to push changes to gitlab incrementally and made me only do it once I implemented larger changes. This slowed down collaborative workflow and was a result of one fatal merge that lead to 2 hours of a fixing and creating a new branch.” In a way Gitlab is seen as a necessary evil, but the skills we learned from making mistakes using it will pay off in future sprints.
2. For The next sprint, we all acknowledge that a scrum methodology from the get-go will increase the success and implementation of our future projects. Accountability is a huge motivator and daily scrums focus workflow. Something like story cards would be hugely beneficial in the beginning stages of a project. Being able to brainstorm user requirements from a users’ perspective before beginning code would be a game changer. Looking back, we did struggle more so on focusing down on specific areas of what the requirements really were and what a successful implementation of these would be. Being specific, story cards allow for iterative steps to be taken to build upon the functionality of the product and removes the worry of a frantic approach to development. Which, without an agile approach can lead to over complicated and/or unnecessary time being wasted developing code that doesn’t meet the requirements. In addition, it stops wasted resources on developing something that the requirements/customer did not ask for.

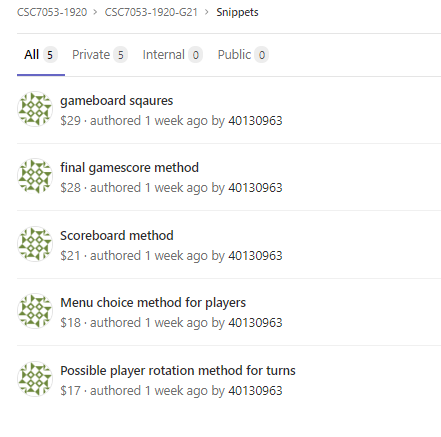
**Final Group words**

As a collective, we benefitted greatly from the project. It allowed us to think independently and collaborate to improve ourselves and push each other as a team. We understand the need to help each other as we all succeed together. It was a rewarding experience that has prepared us for our future careers in Software Engineering and beyond.

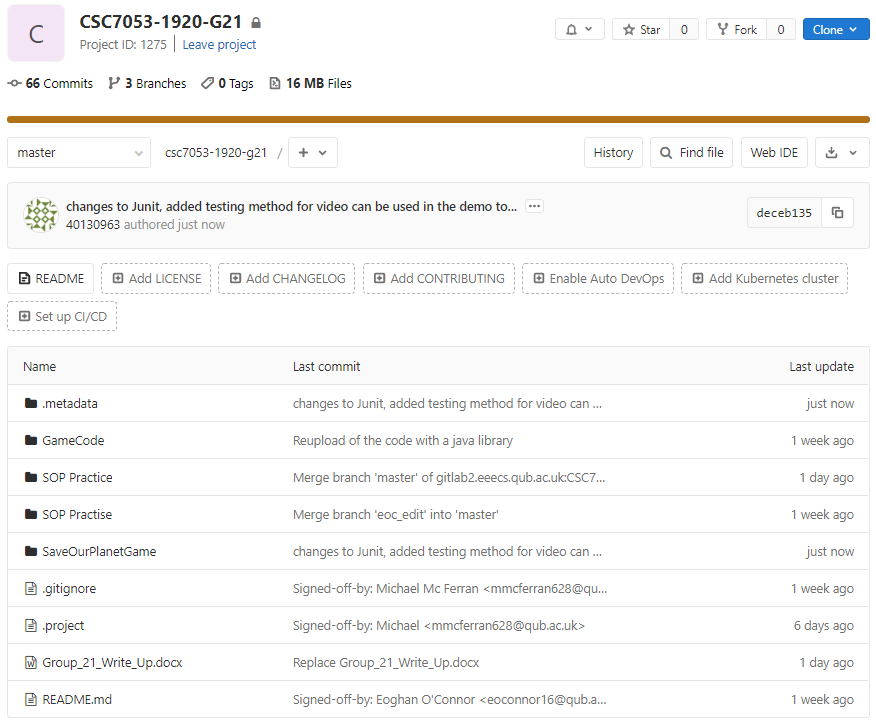
# Appendix III

**Gitlab Usage (MMcB)**

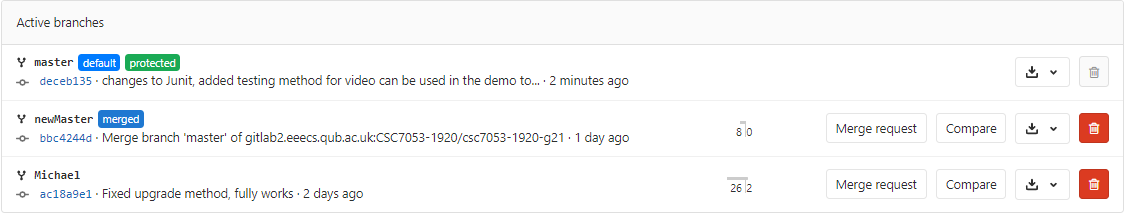
Snippet feature Usage

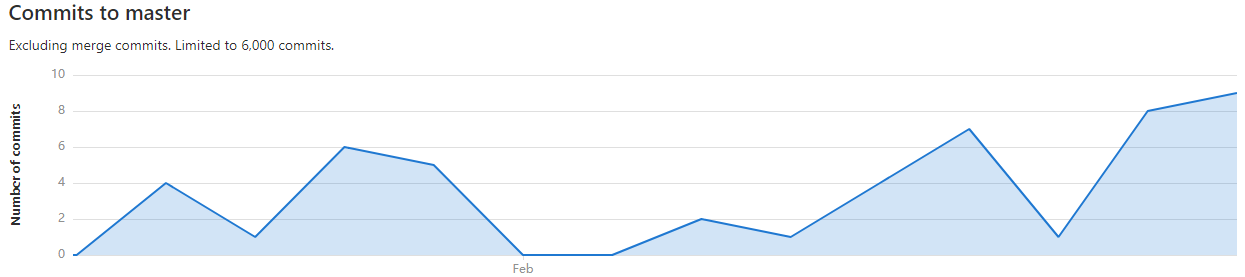


Snippets were a useful feature at the beginning of the project, as it allowed us to drop sections of code, such as methods or class structure, on to the Gitlab website. This would then allow us to grab the code with ease and work on it on our own machines. Using this feature was extremely beneficial and allowed us to work in a more collaborative manner outside of our group meetings.

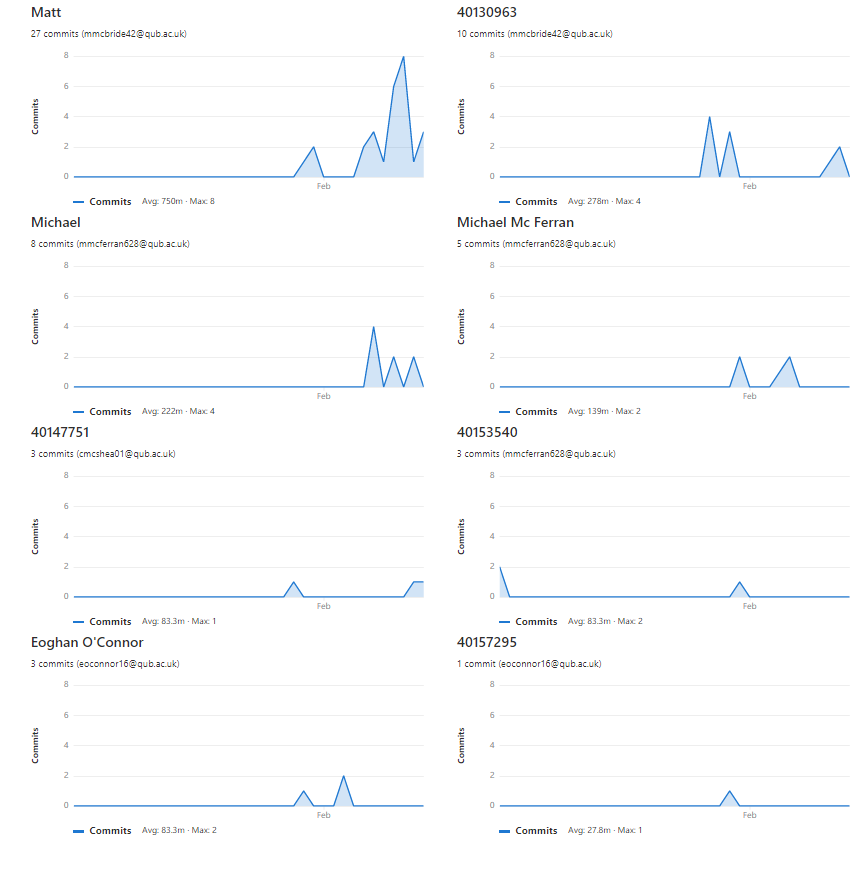
Gitlab Master Folder

Branches



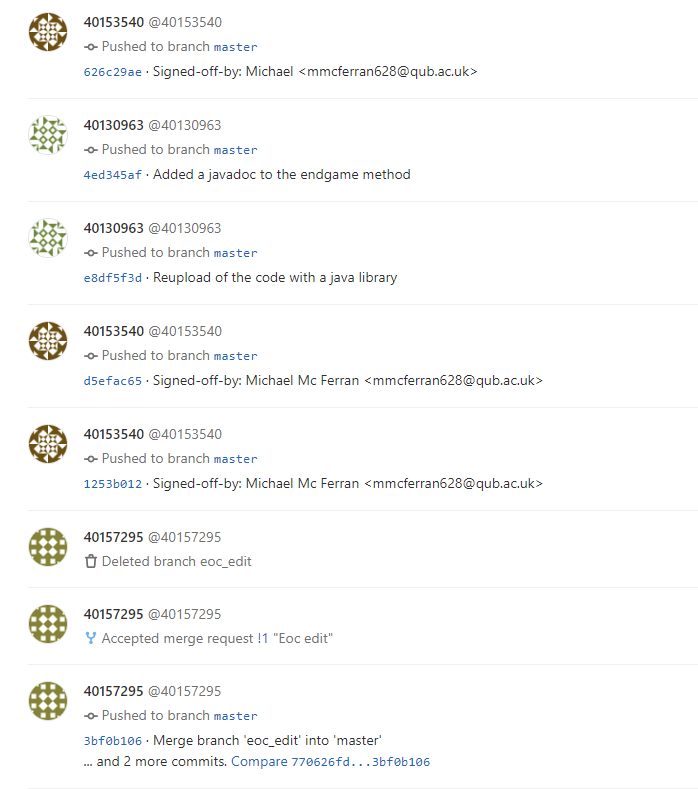
Commits Chart – Gitlab

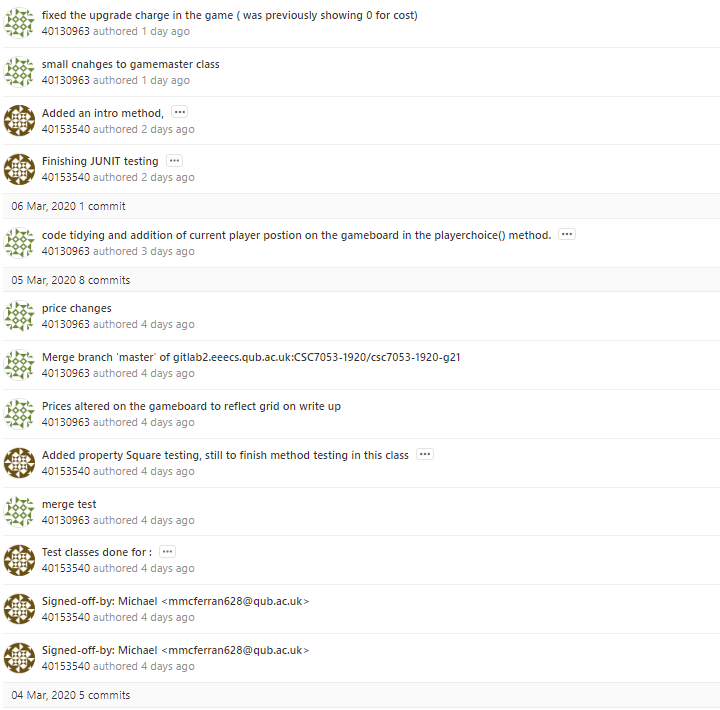
Commits per person

During our time using Gitlab, we faced some initial issues when committing changes to the game code. Due to issues surrounding this, involving loss of progress and difficulty getting new changes pushed onto the Gitlab application, some members of the group volunteered to upload the data, to ensure that it was stored correctly. Instances of paired programming also effected the amount of commits from team members.

Gitlab Master upload history examples

Example 1: General activity

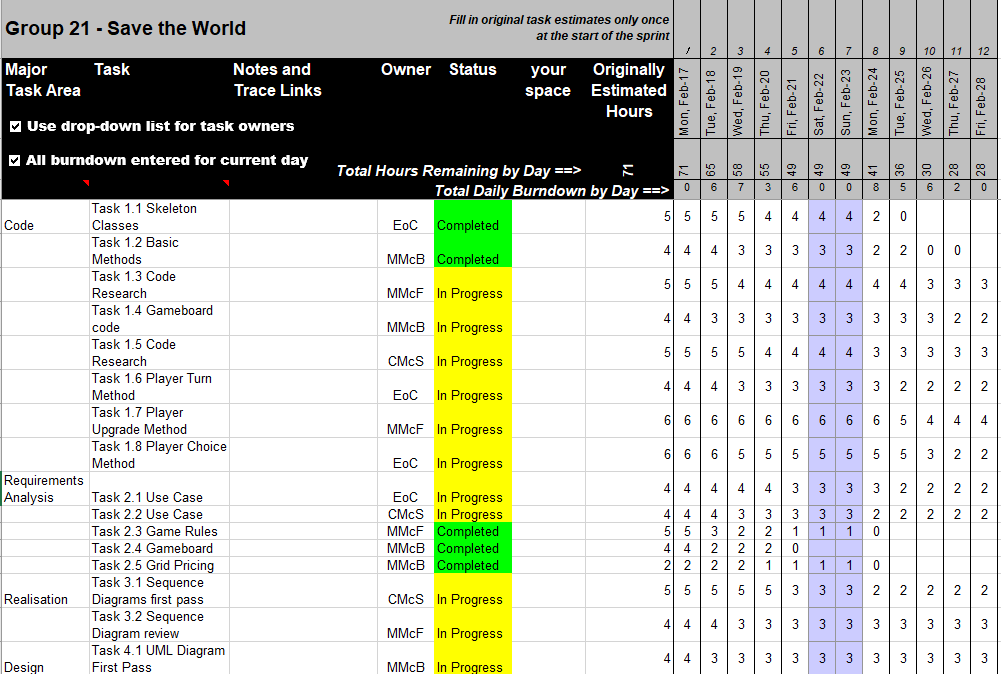


Example 2: Team Sprint activity

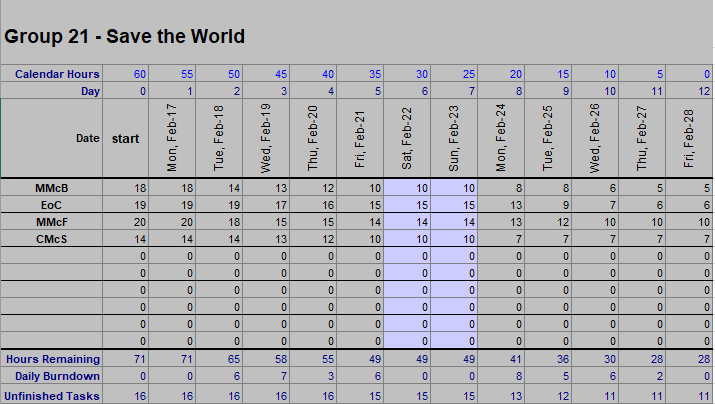
# Appendix IV

Sprint Backlog and Team/Team member burndown charts

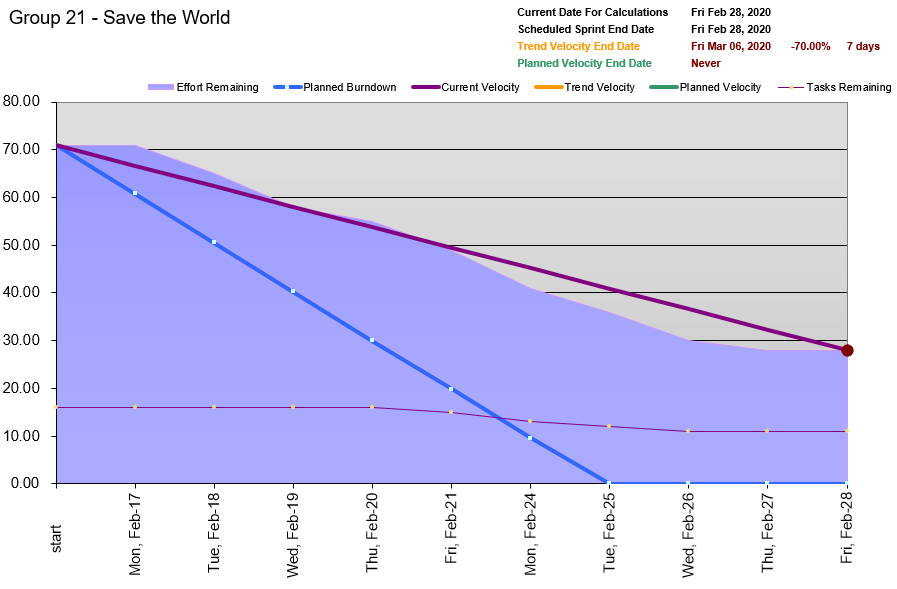
**Sprint One 17th Feb 2020 – 28th Feb 2020**

Sprint Backlog

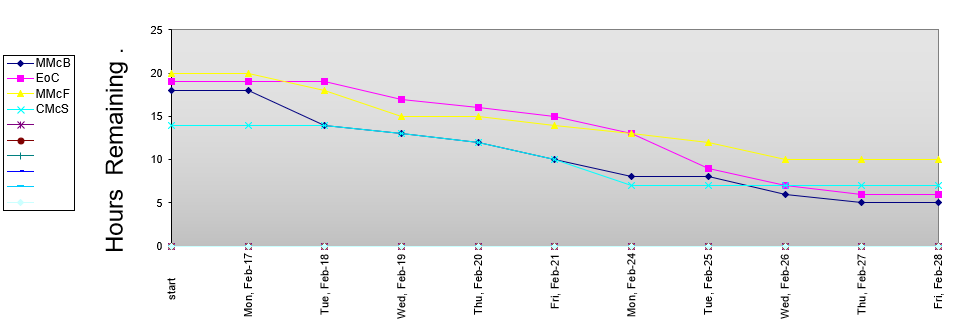
Team Member Burndown



Team Burndown Chart



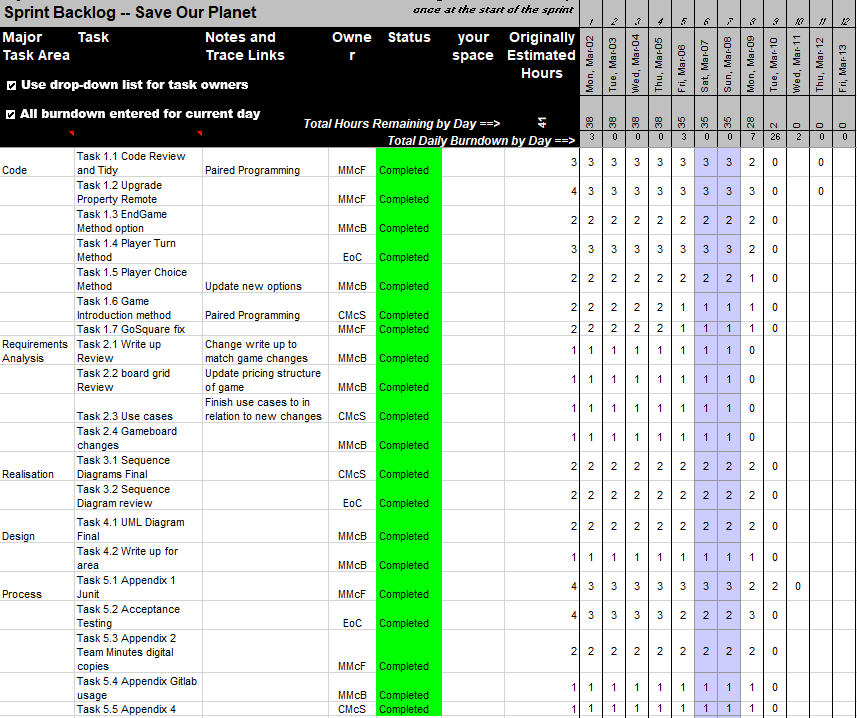
Team Member Burndown Chart

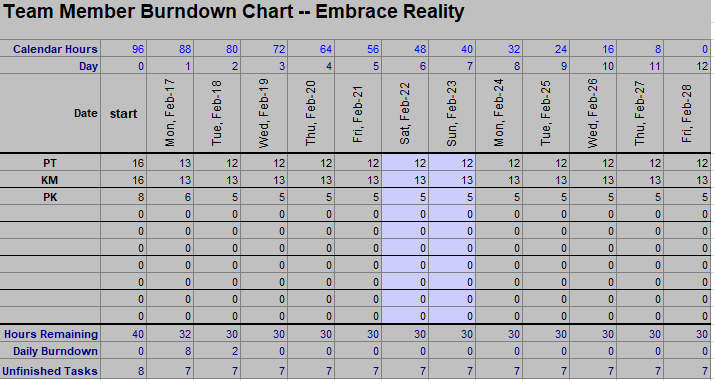


Progress was largely made in this Sprint to build the structure of the game and implement the basic methods, such as registerPlayer() and checkIfPassed(), that allowed the player to player count, names and to move around the board.

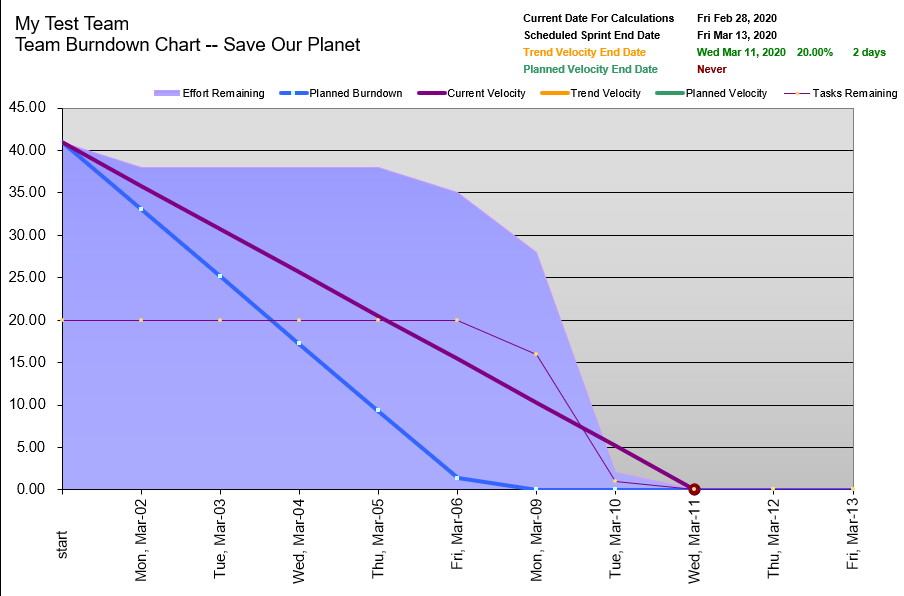
**Sprint Two 2nd March 2020 – 11th March 2020**

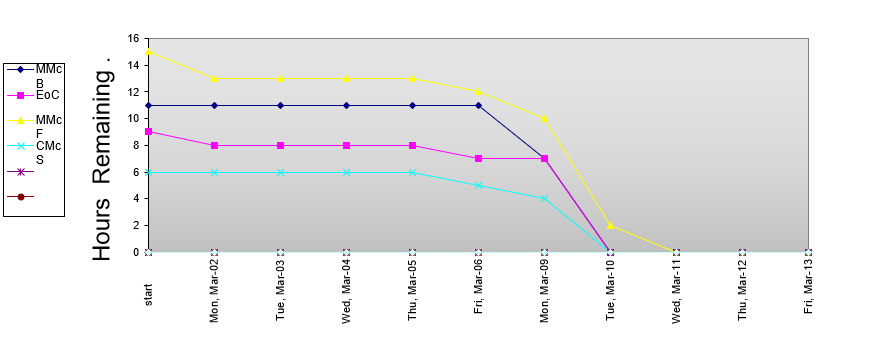
Sprint Backlog





Team Burndown Chart



Team Member Burndown chart

Desired deliverables were met in this final sprint period, to ensure all aspects of the project were wrapped up and ready for review, allowing for us to make any changes if necessary.