



School of Electronics, Electrical Engineering & Computer Science

CSC7083 Software Development

Save Our Planet

Technical report

Group Number: 01

Link to Video Demonstration: <https://youtu.be/v2-u25XeDGg>

Student Name	Student Number
Andrew McClelland	<u>40318021</u>
Damian McGill	<u>40078137</u>
Gary Donnelly	<u>40314509</u>
Noel Proctor	<u>40102820</u>

CSC7083 Peer Assessment: Save Our Planet

This Assessment Document is intended to provide you and your assessor with an overview of each group member's involvement delivery of the CSC7083 Project.

Each group should complete one Assessment Document and its content must be agreed by all group members. The completed form should be included at the start of your group's PDF report. ***Don't forget to fill in the Group Number.***

There are three main parts to the Assessment Document – the Evaluation, the Declaration and the Personal Statements (**spaces for each team member's personal statement are provided on the reverse of this sheet**). All parts must be completed – otherwise your group's report will not be marked. Arrange a group meeting to discuss the evaluation and personal statements, and see the note below!

Evaluation Group Number:01				
Name	Contribution to team-working and motivation ¹	Contribution to documented analysis, design and testing ^{1,2}	Contribution to working system code ^{1,2}	Peer Score (Range 85 – 115)
<i>Jane Doe</i> [Example only - you may overwrite this row!]	4	3	1	90
Andrew McClelland	4	5	4	112
Damian McGill	4	4	5	113
Gary Donnelly	4	5	5	114
Noel Proctor	5	4	4	113

Values 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 (<i>use the words shown in the example below!</i>)
<i>Jane Doe</i> [Example only - you may overwrite this row!]	<i>06/12/2019</i>	<i>I agree to the terms of the declaration</i>
Andrew McClelland	25/04/2022	"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."
Damian McGill	25/04/2022	"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."
Gary Donnelly	25/04/2022	"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."
Noel Proctor	25/04/2022	"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."

<i>Personal statement of (enter name):</i>	<i>Andrew</i>
The following were my most significant contributions to the project (100 words or less):	
<p>Produced the virtual board. I was involved in the development of the square class and made some changes to the main class. Helped with some content on the Use cases.</p> <p>I helped draw up the Acceptance Tests table and carried out some of the tests as well as producing a summary of the group's testing process. I attended all meetings and helped assigning tasks for weekly completions.</p>	

<i>Personal statement of (enter name):</i>	<i>Damian</i>
The following were my most significant contributions to the project (100 words or less):	
<p>I was involved in the initial design of the game board and provided a base model for the game board based on the project specification and contributed to use case scenarios. Throughout development I was responsible creation of the initial design of the board class to hold each of the game squares. I redesigned the board class to allow for easy future extension of the game board if the end user desired more game squares. Also authored some methods in the Main Game & completed Junit testing and user acceptance tests. I have also attended all meetings bar one.</p>	

<i>Personal statement of (enter name):</i>	<i>Gary</i>
The following were my most significant contributions to the project (100 words or less):	
<p>I was involved throughout the entire project lifecycle. I played a lead role in the Requirement Analysis process and authored the Use Case Diagram and Use Case Descriptions sections. I helped develop, and design, the UML Sequence Diagrams. I was also heavily involved in authoring the Main Game class by designing and coding many of its methods, whilst adhering to OOP best practice principles. I also carried out some acceptance testing elements. I attended all meetings and sporadically helped assign future key achievable targets for the next meeting.</p>	

<i>Personal statement of (enter name):</i>	<i>Noel</i>
The following were my most significant contributions to the project (100 words or less):	
<p>I was involved throughout the project development lifecycle. During initial development I contributed to designing use cases, and in the planning of what field areas would look like. During development, I was heavily involved in developing the object classes for each of the field areas. I developed the UML class diagrams, the Design & Future development sections of the report, and played a forward role in project organisation and management. I have also attended all meetings.</p>	

Table of Contents

Game Overview and Introduction	1
Requirement Definition and Specification.....	5
Requirement Analysis	13
Design (UML Class diagram and related information)	16
Future Development	18
Testing	20
Appendix	21

List of Figures and Tables

Fig 1: Game Board	1
Fig 2: Select Number of Players	2
Fig 3: Player Name Entry Screen	2
Fig 4: Players Options	2
Fig 5: Rolling the Dice and Moving Around the Board	2
Fig 6: Player Attempts to Roll Dice for a Second Time	2
Fig 7: Player Purchasing an Area	3
Fig 8: Player Declines to Purchase. Area Offered to Other Player	3
Fig 9: Player Pays Rent	3
Fig 10: Player Tries to Develop Area Without Owning All Area Fields	4
Table 1: Upfront Cost and Rent for Each Field	4
Fig 11: Use Case Diagram	13
Fig 12: Player Turn Sequence Diagram	13
Fig 13: Options to Purchase Sequence Diagram	14
Fig 14: Develop Square Sequence Diagram	15
Fig 15: Quit Game Sequence Diagram	15
Fig 16: Initial Class Diagram	16
Fig 17: Final Class Diagram	17
Fig 18: Future Development UML	18
Fig 19: Implementation of Building Objects	19

Game Introduction, Overview and Rules.

Introduction

Save our planet is a Java based board game for 1-4 players where each player is out to build a better, greener future for future generations. Players must use ECO Coins to purchase and develop areas around the game board. The game board is made up of 12 tiles which are divided into fields which are then further divided into areas. The fields and areas included in the game are as follows in the game board image:



Figure 1: Game Board

Rules

To control the flow of the game, rules have been agreed upon based on our use case scenarios and have been implemented in the code and will be displayed throughout the overview.

1. There can be between 1-4 players.
2. Each player will start the game with 1000 ECO Coins
3. Players must choose a name with a length of at least one character.
4. Players cannot use the same name.
5. Players must own all areas within a field before developments can be built.
 - a. Players must build 3 Minor Developments in an area before they can build a Major Development
6. If a player lands on a tile owned by another player, they must pay rent to the owner.
 - a. The owner of the tile has the option to decline the payment.
7. If a player lands on an unowned tile and does not wish to purchase it, they may offer it to other players.
8. When each player passes ECO Base Start, they will receive 200 ECO Coins.
9. When any players ECO Coin balance reaches zero, the game is over.

10. If any player choses to quit the game, the game ends for all players.

Overview

To begin the game, the users must first decide how many players there will be, between a minimum of 1 and a maximum of 4, if a number outside of this range is entered, or any other character is entered, the users will be prompted to enter a number between 1-4.

```
Hello, welcome to the 'Save Our Planet' Board Game.  
How many players are taking part? (please enter a number between 1 - 4)  
s  
Sorry, thats wrong. Please enter a 'number'..... between 1 - 4.  
7  
The number of players must be between 1 - 4. Please try again.  
4  
Please enter the name for player 1:
```

Figure 2: Select Number of Players

Players must then enter their chosen names. The name of the player must be at least one character long and may only contain letters a-z / A-Z, the name must be unique. If the user choses to enter a name of length 0 or use numbers, they will be prompted to try again with letters.

```
You must enter a name (a-z / A-Z)  
Damian  
Please enter the name for player 2: Damian  
That name's already taken. Please choose another.  
Please enter the name for player 2: Gary  
Please enter the name for player 3: Andrew  
Please enter the name for player 4: Noel  
Thank you.
```

Figure 4: Player Name Entry Screen

```
Damian - your current ECO Coin balance is 1000.  
Please select your action from the list below:  
1. Roll your dice  
2. Develop a minor dev on an area  
3. Complete build on an area to help complete an Energy Field  
4. Skip your turn entirely or move to next player after rolling  
5. View the current player(s) status  
6. View the current board status  
7. Quit game  
Please select your option by pressing the relevant number
```

Figure 3: Player Options

Upon entering the player names, the game starts and player one is presented with their ECO coin balance and an array of options.

To advance the game, players may roll two six-sided die once per turn to move around the game board and will be provided with the information of the square they have landed on. If the player attempts to role the die again, they will be presented with a message advising they cannot take this action again this turn.

```
Please select your option by pressing the relevant number  
1  
Damian, the square you are currently on is ECO Base within Start.  
Rolling dice.....  
Damian, you have rolled a 5 and a 3 totalling 8.  
You have landed on Nuclear in Renewable Energy.
```

Figure 5: Rolling Dice and Moving Around The Board

```
Please select your option by pressing the relevant number  
1  
Sorry, you can only roll the dice once per turn.  
Please press enter to continue.....
```

Figure 6: Player Attempts To Roll Dice For A Second Time

When players land on an unowned tile, they have the option to purchase that tile at a cost to their ECO Coins. Purchasing an area updates the owner variable attached to the square with the players name. This will allow the player to receive rent payments from other players who subsequently land on that tile. To maximise the rental income from each of their tiles, players should aim to build developments on each tile that they own.

```

You have landed on Nuclear in Renewable Energy.
This area available to purchase. It costs 400 ECO Coins. Would you like to purchase control it? Please enter Y or N
y
Checking ECO Coin balance.....
You have sufficient funds for purchase.
Beginning transaction.....
-----
Transaction completed.
-----
Your ECO Coin balance was: 1000.
You bought control of the Nuclear area for 400 ECO Coins.
Your new ECO Coin balance is: 600.
You now own the Nuclear area within Renewable Energy energy field.

```

Figure 7: Player Purchasing An Area

If the player chooses not to buy the area, they can offer the area to another player for the same cost. They simply must enter the desired players name to offer the area to and the desired player must say yes or no. If the player does not wish to purchase, the original player may offer to another player. Allowing another player to purchase the area utilises the same method as purchasing during the original players turn.

```

Gary, you have rolled a 3 and a 1 totalling 4.
You have landed on Carbon Capture in Technology.
This area available to purchase. It costs 300 ECO Coins. Would you like to purchase control it? Please enter Y or N
n
Would you like to offer this area to another player? Please enter Y or N
y
What other player would you like to offer the area to?
Damian
Andrew
Noel
Noel
You have offered Noel to control the area.
Noel, do you accept this offer and want to purchase control of said area? Y or N
n
Noel has declined your offer.
Would you like to offer this area to another player? Please enter Y or N

```

Figure 8: Player Declines to Purchase an Area. Area Offered To Another Player

If the player lands on an area owned by another player, they must make an offer to pay rent to the owner of the area. The owner may decline to accept the payment.

```

Noel, you have rolled a 3 and a 6 totalling 9.
You have landed on Hydroelectric in Renewable Energy.
This area is currently owned by Damian.
You have landed on an area that is controlled by another player. You must contribute 15 ECO Coins towards their ECO fund.
Damian, do you wish to accept this funding contribution? Please enter Y or N

```

Figure 9: Player Pays Rent

Developments improve the quality of the area and come in two distinct types: Minor and Major Developments. Each area can hold 3 Minor Developments and one Major Development.

Minor Developments provide small boosts to the rental income for each area, but that is incremented with each build. Major Developments provide a large boost to rental income. The ECO coin cost of each of the Major & Minor Developments is reflective of how eco-friendly the field in question is. For example, fossil fuels have the lowest upfront cost for an energy source but have a low return value when compared to Renewable energies which cost more up front but have the largest returns in the game.

A list of all Up-Front, Development & rental costs can be found below:

Field	Up Front Cost	Minor Dev	Major Dev	Base Rent	Rent (1 Dev)	Rent (2 Dev)	Rent (3 Dev)	Rent (Major Dev)
Start	0	0	0	0	0	0	0	0
Community	100	50	100	5	10	15	20	50
Technology	300	150	300	15	30	45	60	150
Fossil Fuels	200	100	200	10	20	30	40	100
Renewable Energy	400	200	400	40	70	100	125	200

Table 1: Details of upfront costs & rent for each field.

To unlock the ability to build Minor Developments within an area, the player must first own all areas within a given field. After building 3 Minor Developments on an area, the player then unlocks the ability to build one Major Development to finish off the area. If any player attempts to build a development before they have met the criteria, a message will be displayed to advise them.

Please select your option by pressing the relevant number
2
You do not own any areas that qualify for development.

Figure 10: Player Tries To Develop Area Without Owning All Area Fields

During each turn the player can choose which actions they would like to take. Each player can roll the dice to move around the board or choose to build the available developments for their areas. They can also review the status of all players and squares on the game board. While there is no limit to the number of actions a player can take on any given turn, they may only develop as far as their ECO coin balance will allow them.

If one player runs out of ECO Coins, or all areas on the game board have a Major Development, the game will end in failure or success respectively for all players. The game continually checks players ECO Coin balance and checks if all sites have a major development at the start of each round.

We have built Save Our Planet primarily with the customers required specifications in mind, but we have also focussed on expandability. While the maximum number of players is defined above as 4, the process of changing the maximum number of players is as simple as changing the value of a Final Static Int in the MainGame file named maxPlayers. All data structures that manage the current player base are ArrayLists and as such, can be as big or as small as the customer would like. Likewise for the size of the game board. This is defined in the MainGame file as a final static int named maxBoardSize which is defaulted to 12. If the customer so wishes, they can update the source code to allow as many or as little squares to be created for each game. This is based on a local CSV file which contains Field Names and area names. If the max number of squares on the game board is set to a number larger than the number of defined Field & Area names in the CSV, they will be recycled, and each new iteration of a field and area will display a numerical order. While both features have not been requested in the Project Specification, they each display that we have created the project with future adaptations in mind.

Requirement Definition and Specification

2.1 Project Specifications, Interpretations and Assumptions

Deliver a well-designed and well-documented working system that satisfies the requirements scope for a virtual board game called 'Save Our Planet'.

Utilise OOP and UML methodologies to deliver emphasis on requirements analysis, system design, software implementation and system testing to deliver a natural language only solution that meets the project scope.

Whilst collaboratively and successfully working as a team of four, we aim to develop a working system through a 'use case driven' development process.

To satisfy customer requirements it will require us to identify how actors will interact with the game by identifying the main Use Cases of the program. Assumptions were made at this stage help draw up a diagram and Use Case Descriptions.

Assumptions

1. The game has between 1 and 4 players. Players are the 'Actors'
2. Player names should be entered at the beginning (we do not allow duplicate name entry).
3. The players take turns. They throw two virtual dice at a time during their turn.
4. Players are told where they have landed and their obligations or opportunities.
5. Update player resources / balance where appropriate.
6. 12 Squares in total - divided into following:
4 energy fields (designated: "Community", "Technology ", "Fossil Fuels ", "Renewables").
10x energy squares, 1x start square and 1x rest square.
(2x fields have 3 squares and two have 2 squares = totalling 10).
8. To develop an area you must own all areas within said energy field.
9. Are development is called minDev and majorDev and they have respective costs to develop.
10. If another player lands on 'your' square rent is payable
(the greater the development the higher the rent cost - there are 5 levels of rent charges:
noDevRent / oneDevRent / twoDevRent / threeDevRent / majorDevRent).
11. If a player runs out of resources the game ends for all.
12. If a player chooses to quit, game ends for all.
13. Once all fields are developed the game is successfully finished and the planet is saved.

2.2 Identifying Use Cases

1. Number of Game Players
2. Assigning Player Name
3. Player Turn
4. Develop an Area
5. Dice Roll
6. Assigning Square Ownership
7. Increase Resources
8. Offer Square Ownership
9. Decrease Player Resources
10. Display Player Resource Balance
11. Quit Game
12. Final Game Status
13. Failed Mission
14. Successful Mission

Identifying relevant actors was straightforward, we had some discussion around the fact that an actor's role may change if they owned squares for development. However, it was decided to create *Develop an Area* as a use case as this allowed greater flexibility, especially if the game was to be further developed by stakeholders i.e., expand the game to allow more areas to be developed. We extracted conclusions from the 'Core Customer Requirements' section of the project specification and based our Use Case Descriptions on these.

2.3 Use Case Descriptions

Use Case 1: Number of Game Players

Name	Ascertain number of players.
Objective	Enter total number of players taking part.
Actors	Player(s).
Precondition	None.
Main Flow	1. Game prompts for current user to enter number of players playing. 2. Current user enters the number and hits enter on keyboard.

Alternative Flows	At 2; - If user enters a letter or other character, response asks for a number. - If user enters a number outside range, response stipulates accepted range and asks user to try again. - If user doesn't enter an int value, response stipulates accepted range.
Post Condition	System records the number of players electing to play.

Use Case 2: Assigning Player Name

Name	Ascertain name of players.
Objective	Enter the names of players taking part.
Actors	Player(s).
Precondition	User has completed 'Use Case 1'
Main Flow	1. Game prompts current user to enter name of player 1. 2. Current user enters the name and hits enter on keyboard. 3. Repeat until number of players from 'Use Case 1' are assigned a name.
Alternative Flows	At 2; - If user enters a number, response asks for a name a-z/A-Z. - If user enters a blank name, response asks for a name a-z/A-Z. - If user enters a duplicate name, response asks for a different name.
Post Condition	System records player names and display them along with their initial ECO coin balance along with main playing menu list.

Use Case 3: Player Turn

Name	Ascertain which players turn it is.
Objective	Identify whose turn it is.
Actors	Player(s).
Precondition	User has completed 'Use Case 1' and '2'.
Main Flow	1. Game displays the name and balance of the user whose turn it is. 2. Game asks this user to select an action from list below. 3. User enters corresponding digit on keyboard and hits enter. 4. This calls on the 'Dice Roll' Use Case (no. 5).
Alternative Flows	At 3; - Player can decide to skip their turn. - Player can decide to develop an area if available to do so. This calls on the 'Develop an Area' Use Case (no. 4).

Post Condition	System records player decision (roll dice / skip / develop).
-----------------------	--

Use Case 4: Develop an Area

Name	Develop an area within an energy field.
Objective	Allow a player to develop an area they own within an energy field.
Actors	Player(s).
Precondition	User has completed all relevant Use Cases and it is their turn to 'play'. They must be the assigned owner of all areas within corresponding field.
Main Flow	<ol style="list-style-type: none"> 1. User enters the name of the area available to develop. 2. System checks current player owns all areas in field. 3. System checks area isn't already developed. 4. System checks current player resource balance for affordability. 5. Area is developed (min or major). 6. System displays updated player resource balance (Use Case 10).
Alternative Flows	<p>At 2;</p> <ul style="list-style-type: none"> - If user doesn't own all areas within field, response states this. <p>At 3;</p> <ul style="list-style-type: none"> - If user resources are insufficient, response states this. <p>At 6;</p> <ul style="list-style-type: none"> - If user has other areas which qualify for development, they can develop those (call this Use Case again). - User may decide for no further development and main playing menu list will display again (Use Case Dice Roll - No. 5)
Post Condition	A development has taken place, main playing menu list displays and player decides their next action.

Use Case 5: Dice Roll

Name	Roll dice and display opportunities / obligations.
Objective	Roll both dice and display resulting player opportunities or obligations.
Actors	Player(s).
Precondition	Relevant Use Cases completed and Game is in progress. It is the current users turn and they have selected option '1' (roll dice).
Main Flow	<ol style="list-style-type: none"> 1. System rolls both dice. 2. System combines both results (randomly generated) and records the total. 3. Player 'moves' along squares which equals the total from rolling. 4. The square (area) name is displayed along with opportunities/obligations. 5. Player chooses an option and enters corresponding number.

Alternative Flows	At 5; - If available, player can decide to purchase area (Use Case 6). - Pass start and increase resources (Use Case 7). - Offer area to a different player to purchase (Use Case 8). - Decrease player resources (Use Case 9).
Post Condition	System records the outcome of the players decision based on the opportunity / obligation ascertained.

Use Case 6: Assigning Square Ownership

Name	Purchase (control) an available area.
Objective	Player completes the purchase (owns) an available area.
Actors	Player(s).
Precondition	Relevant Use Cases completed, and Game is in progress. Current player rolls dice and lands on a square no other player controls.
Main Flow	1. Game displays square player occupies, displays the individual and combined dice total and the new square player has landed on. 2. If available, gives an option for current player to purchase control. 3. If player decides to buy, system checks player resources for affordability. 4. Current player is assigned control of respective area. 5. System displays updated player resource balance (Use Case 10).
Alternative Flows	At 2; - System also displays offer square to a different player (Use Case 8). At 3; - If player has insufficient funds for control, system subsequently calls Use Case 10 and then Use Case 3.
Post Condition	System records player now controls that area (square).

Use Case 7: Increase Resources

Name	Increase current players resources balance.
Objective	Increase current players resources balance by passing the Start square.
Actors	Player(s).
Precondition	Relevant Use Cases completed and Game is in progress. It is the current users turn and they land on/pass 'Start' (sq.1) after rolling.
Main Flow	1. The current player either lands on or passes the Start Square (Sq. 1). 2. Current players resources balance updates (increases +200). 3. Calls the update player resource balance use case (No.: 10).
Alternative Flows	None.
Post Condition	Current player resource balance updates (+200 ECO Coins added).

Use Case 8: Offer Square Ownership

Name	Offer control of area to a different player.
Objective	Current player has rejected offer of control and offers to another player.
Actors	Player(s).
Precondition	Relevant Use Cases completed, and Game is in progress. Current player rolls dice and lands on a square no other player controls.
Main Flow	1. Game asks current player if they want to offer area to another player. 2. Current player confirms yes. 3. The current player can decide which other player is offered area. 4. Use Case 6 is performed for the 'other' player if they choose to purchase control of this area. 5. Use Case 10 is subsequently called.
Alternative Flows	At 3; - The designated player also rejects offer to purchase control. - Current player is asked if they want to offer it to another different player (above step is repeated if necessary). - Use Case 3 is called.
Post Condition	System records which player now controls respective area.

Use Case 9: Decrease Player Resources

Name	Pay rent for landing on square a different player control.
Objective	Take resources from player and assign to another player who controls square.
Actors	Player(s).
Precondition	Game is in progress and current element is controlled by another player. Current player rolls dice and lands on this pre-controlled area (square).
Main Flow	1. Game displays message current player has landed on an area (square) controlled by a different player and a contribution is required. 2. Game asks other player if they wish to accept contribution. 3. Other player does accept the contribution. 4. Current player resources are checked for affordability. 5. Resource balance updates on both players. 6. Calls Use Case 10 for both players.
Alternative Flows	At 3: - Player rejects the rent (calls Use Case 10). At 4: - If player has insufficient funds Quit Game is called (Use Case 11).
Post Condition	Both players resources balances are updated.

Use Case 10: Display Player Resources Balance

Name	Display current player resource balance and its reason for change.
Objective	Display current player resource balance and what changed it.
Actors	Player(s).
Precondition	Relevant Use Cases completed, and Game is in progress.
Main Flow	<ol style="list-style-type: none">1. Player selects an action which results in a change to resource balance.2. System records the amount to be updated against resource balance.3. System records the reason for the resource balance update.4. System displays current resource balance before calculation.5. System performs calculation on current resource balance (+ / -).6. System displays new resource balance.7. System displays the action which caused the resource balance update.
Alternative Flows	At 5: <ul style="list-style-type: none">- If player has insufficient funds Quit Game is called (Use Case 11).- If all areas have major dev's on them Quit Game is called (Use Case 11).
Post Condition	Player, or players, resource balance(s) are updated.

Use Case 11: Quit Game

Name	Quit Game.
Objective	Ends the game for all players.
Actors	Player(s).
Precondition	Game is in progress.
Main Flow	<ol style="list-style-type: none">1. Player selects option 7 - which is quit game.2. System displays message to confirm selection (which will end game).3. Current player confirms decision to end game.4. Final Game Status is called (Use Case 12).
Alternative Flows	At 1: <ul style="list-style-type: none">- If quit game due to insufficient player resources is called Final Game Status (Use Case 12) is triggered.- If quit game due to all fields being fully developed then Final Game Status (Use Case 12) is triggered. At 3: <ul style="list-style-type: none">- If player reneges on original decision the game continues.
Post Condition	Game ends.

Use Case 12: Final Game Status

Name	Final game status when ended.
Objective	When game has ended, display the current status of the game to date.

Actors	Player(s).
Precondition	Game has been ended.
Main Flow	<ol style="list-style-type: none"> 1. System displays remaining resource balance by player name. 2. System displays current area ownership. 3. System displays current level of development (min & major). 4. System calls Failed Mission (Use Case 13).
Alternative Flows	At 2: - If all areas / fields are fully developed Successful Mission is called (Use Case 14).
Post Condition	Final Game Status is displayed.

Use Case 13: Failed Mission

Name	Mission was not completed.
Objective	Display the failed mission message.
Actors	Player(s).
Precondition	Quit Game is called.
Main Flow	1. Game displays the failed mission message to the players.
Alternative Flows	None.
Post Condition	Game ends.

Use Case 14: Successful Mission

Name	Mission was completed.
Objective	Display successful mission completion message.
Actors	Player(s).
Precondition	All areas, and therefore fields, are fully developed.
Main Flow	1. Game displays the successful mission completed message sequence.
Alternative Flows	None.
Post Condition	Game ends.

Game rules allow a square to be offered to other players if the player, whose turn it is, declines to purchase. It was decided to create *Offer Square Ownership* use case to allow current player to offer area to another player, this made use of *Display Player Resource Balance* and improved the overall playability of the game.

2.4 Use Case Diagram

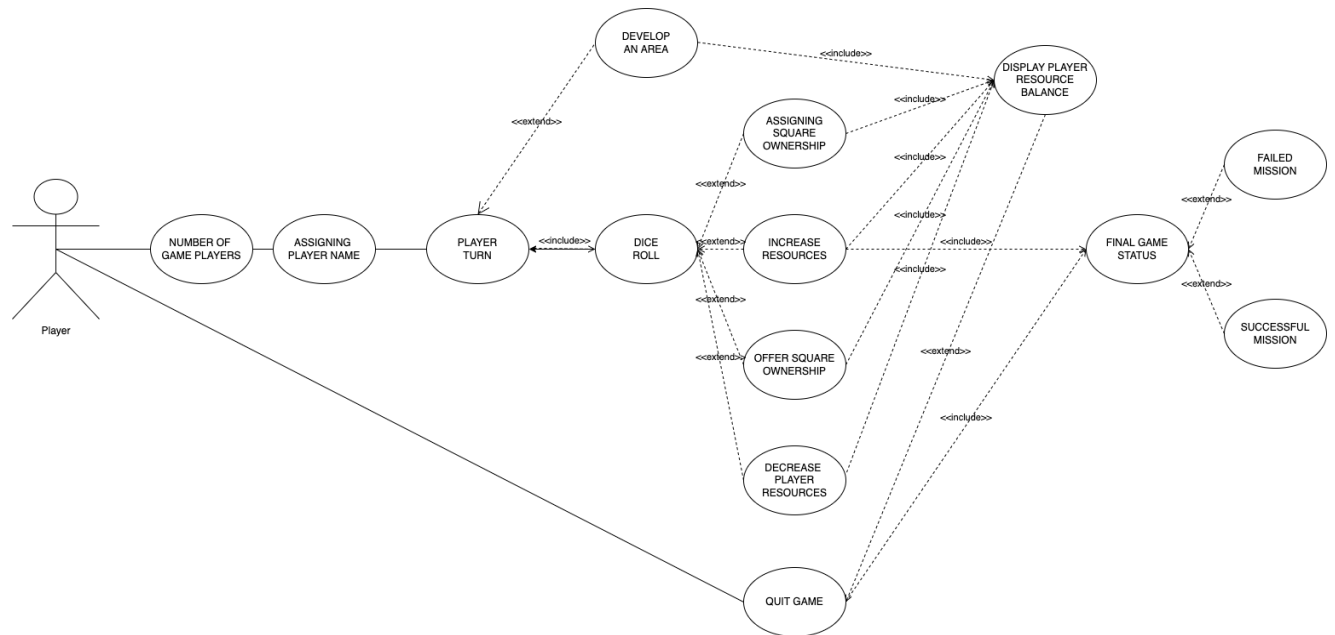


Figure 11: Use Case Diagram

Requirement Analysis (Realisation/ Sequence Diagrams)

Player Turn and Options Diagram:

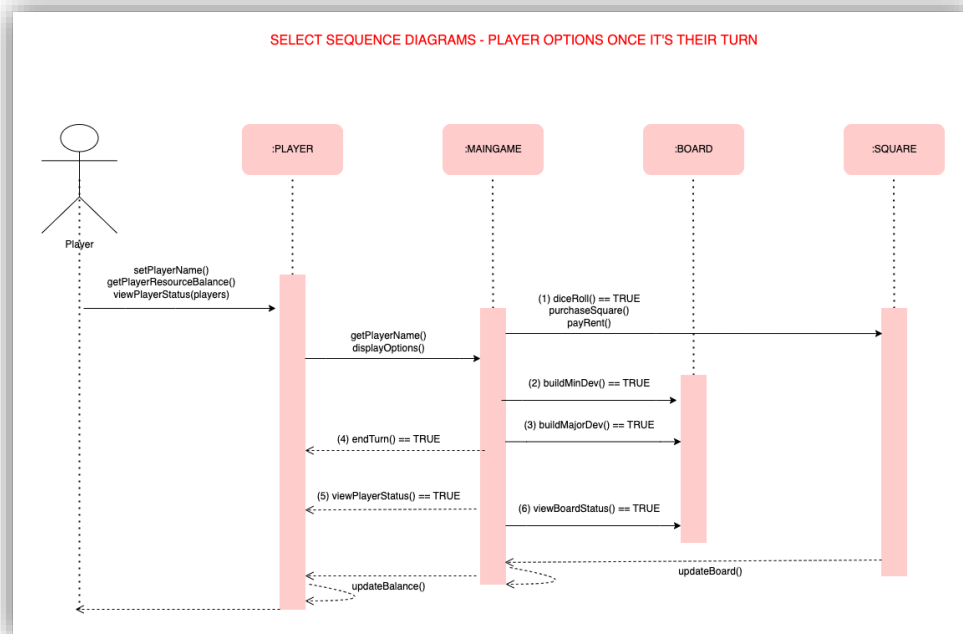


Figure 12: Player Turn Sequence Diagram

Landing on a Square, Options to Purchase / Offer to a different Player / Pay Rent Diagram:

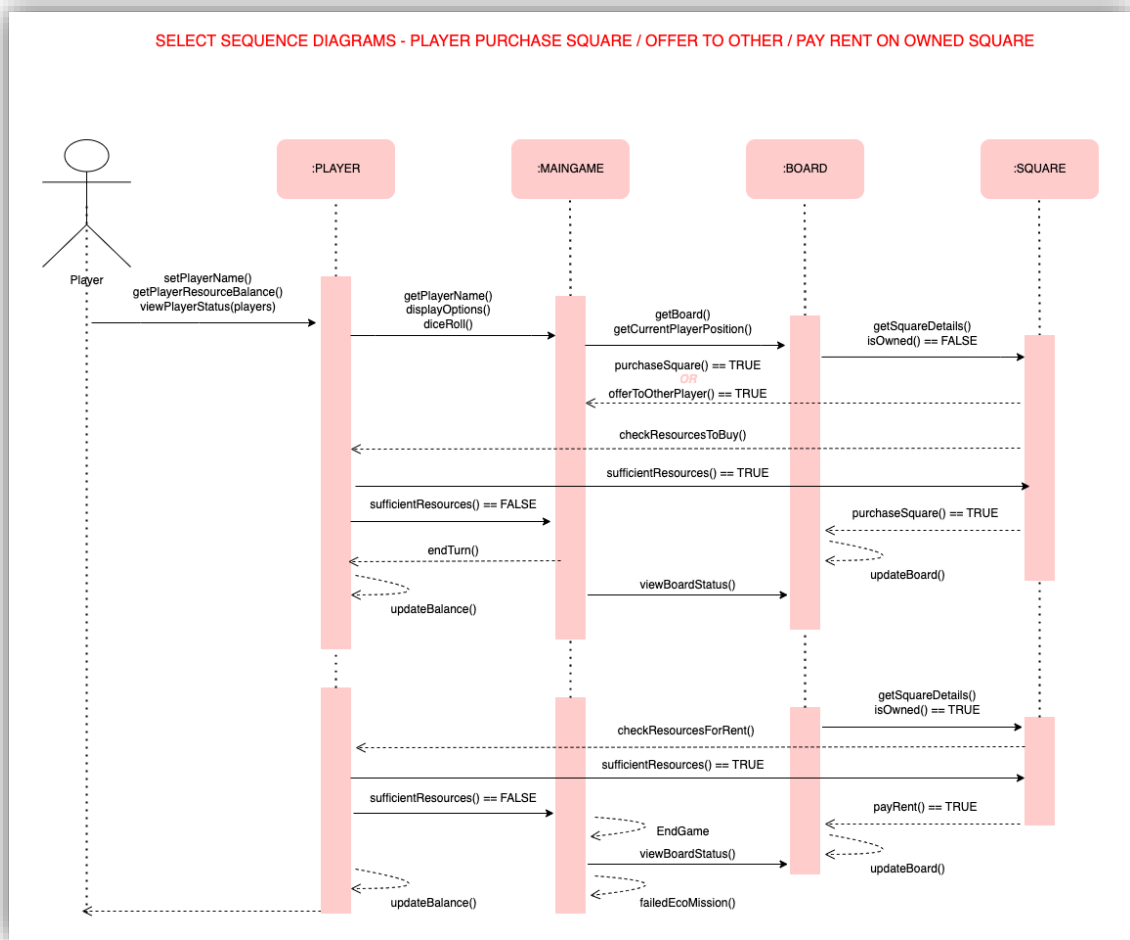


Figure 13: Options to Purchase Square Sequence Diagram

Player Developing a Square Diagram:

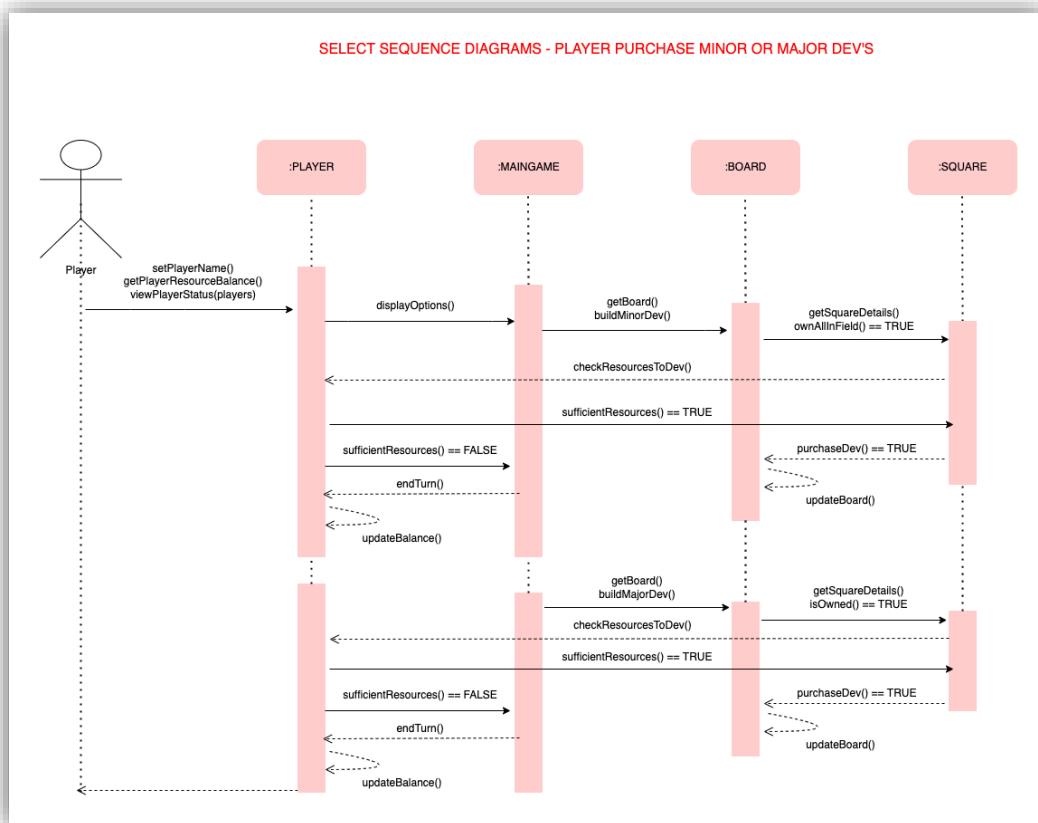


Figure 14: Develop Square Sequence Diagram

Quit Game and Successful / Failed Mission Diagram:

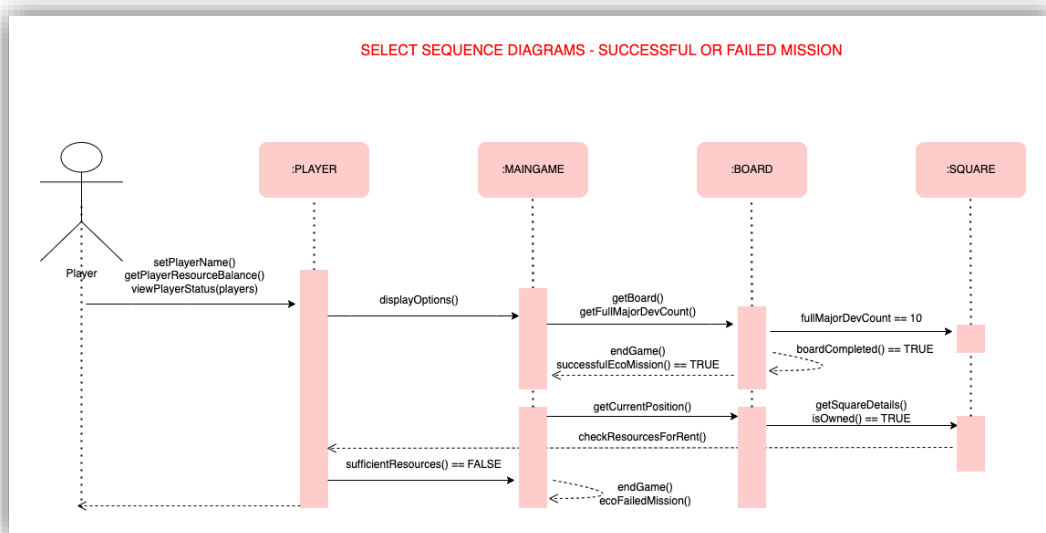


Figure 15: Quit Game Sequence Diagram

Design (UML Class diagram and related information)

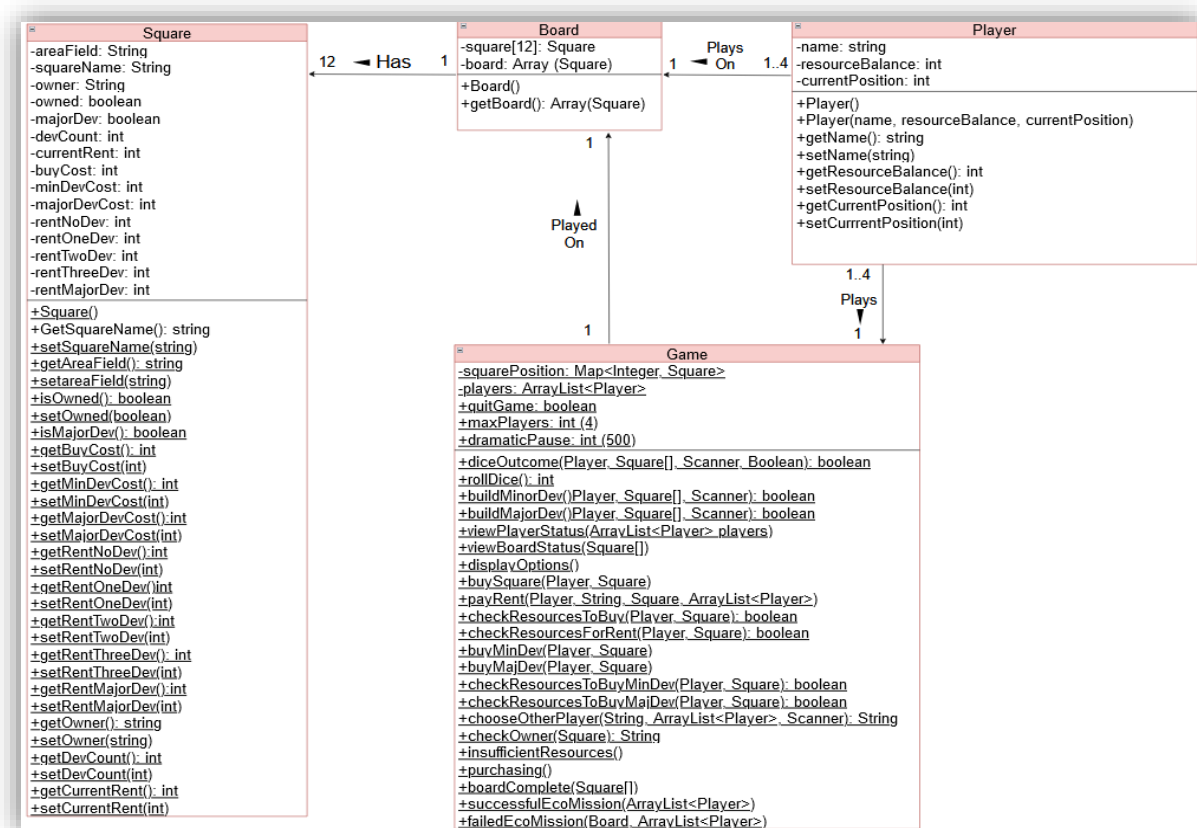


Figure 16: Initial Class Design

The UML Class diagram (Fig 16) above shows the initial class structure of the SaveOurPlanet game. In our early meetings we conducted a Miro board brainstorming session and decided that the minimal viable product would require 4 classes called Game (the driver), Player, Board, and Square. The possibility of creating object classes for each of the area fields was also discussed, however we decided that as we would be following an iterative development model, we would focus on developing the game based on these 4 classes, with the possibility of developing the class model further in future iterations.

During the second development iteration we implemented the class design for each of the area fields (Fig 17). While this does not initially provide any additional functionality, it provides the ability and flexibility to develop the game where the initial design did not. For example, each area field can now be programmed to have different functionality specific to that field. Please see “Future Development” section below future planned development.

‘Square’ contains all information related to an individual square on the board. This includes the squares field, name, owner, currentRent, and associated getter and setter methods for each private variable. Each ‘Square’ has one ‘Field’ of type ‘FossilFuel’, ‘RenewableEnergy’, ‘Community’, or ‘Technology’. ‘Square’ is consumed in ‘Board’ with 1 board having 12 squares contained within an ArrayList of type Square. ‘Board’ contains only two methods. A constructor which instantiates the board, and getBoard() which returns a Square array. The player class contains all information required about each individual player and is consumed in Game, with 1 game played by between 1

to 4 players. Each Player class has 3 private variables (name: string, resourceBalance: int, currentPosition: int) and associated getters and setters.

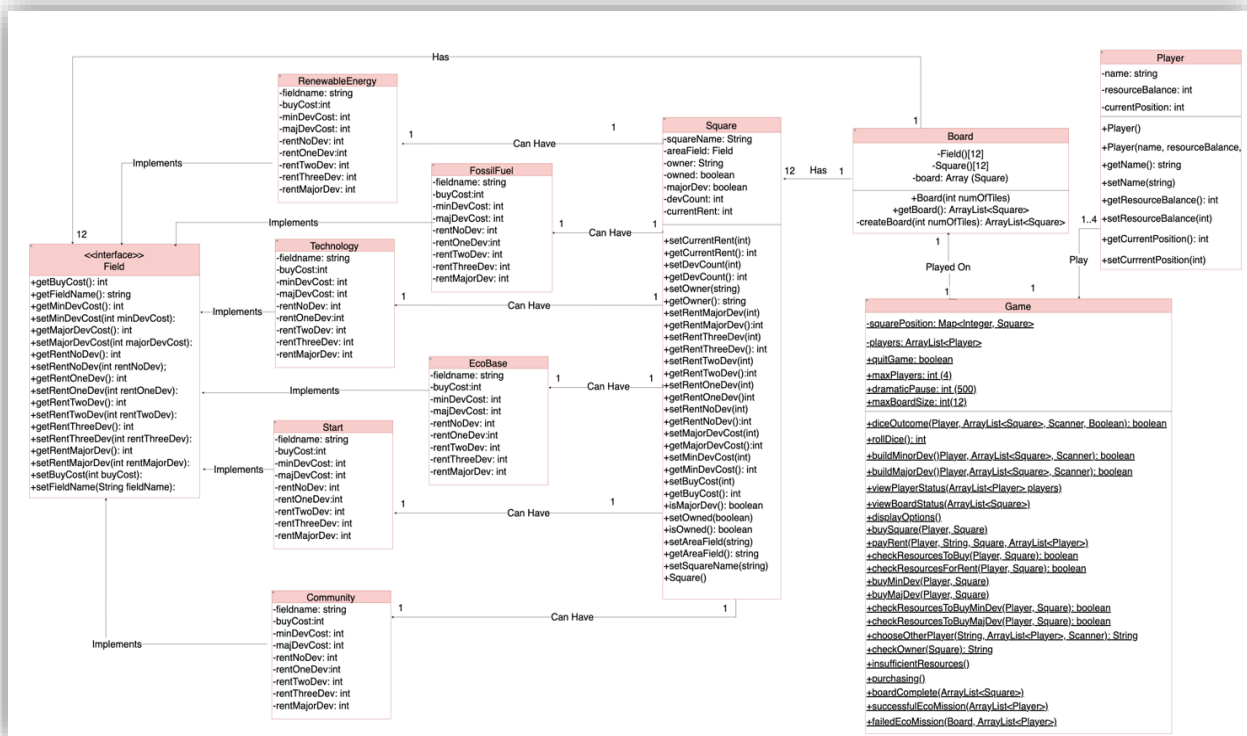


Figure 17: Final Class Design

‘Game’ is the driver class and contains most of the methods and game logic. ‘maxPlayers’ sets the maximum number of players the game can have, allowing the maximum number of players to be easily changed by developers at any time, while the HashMap ‘squarePosition’ maps each Square in the Board to a position.

During later development the board class was changed from implementing an Array of type square to implementing an ArrayList of type square. This change allows the size of the game board (number of squares) to be easily extended in the future. Such changes would be more difficult if a standard Array was used, as once the size of the array is set it cannot be changed. The use of an ArrayList allows flexibility in size of the game board. This change included the addition of a final static int named maxBoardSize which is declared in the Game class and is passed as a variable into the Board class constructor which in turn is passed into the createBoard() method which creates the board through reading in data from a local CSV file. Using a CSV file to hold the details for the squares in the board allows for easy updating and extension.

Please see “Appendix 1.1: Methods Table” for a description of each method in the Game class:

Future Development

There are numerous avenues for further development. One, as mentioned above, would be to make the game bigger by expanding the number of squares on the board or the number of players in the game. This can easily be done by adjusting the `maxPlayers` and `maxBoardSize` final static integers. However, as we have created classes for each of the area fields, we can easily develop the game further by adding specific functionality for each of the fields.

Currently all variables for the field are hardcoded in each field class. We can make this more flexible and add a layer of complexity to the game by changing from a hardcoded approach to a read in approach. This could be incorporated into the CSV file that is read in during the creation of the game board. This would require an additional loaded constructor for each field class that could be created during each iteration of the file reading loop and assigned as each game square is created and loaded onto the board `ArrayList`.

Below is an UML Class diagram of what future class development might look like:

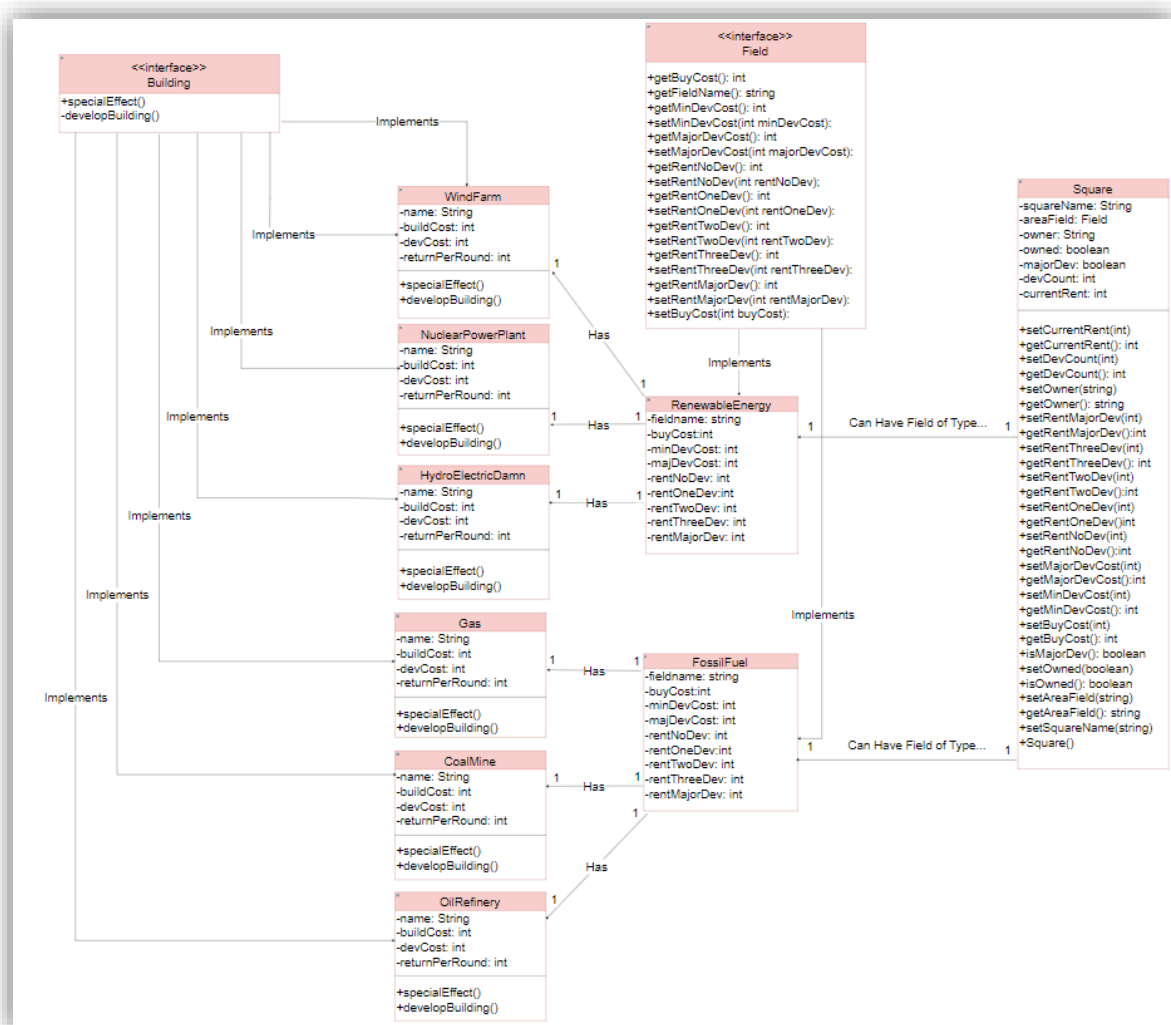


Figure 18: Future Development UML

As can be seen in the diagram above, a new Interface, 'Buildings' has been added, with several new classes which implement the Buildings interface. These new classes represent unique building objects, each of which can have its own functionality specific to that building, but sharing common

functionality inherited from the interface. Each of these new buildings belong to an area field. For example, in Fig 19 below, we can see that the RenewableEnergy Field has an array of Buildings of type NuclearPowerPlant(), WindFarm(), and HydroElectricDamn().

```
public abstract class RenewableEnergy implements Field {

    private String fieldName = "RenewablEnergy";
    private int buyCost = 400;

    Buildings NuclearPlant = new NuclearPowerPlant();
    Buildings WindFarm = new WindFarm();
    Buildings HydroDamn = new HydroelectricDamn();

    private int minDevCost = 150;
    private int majorDevCost = 300;
    private int rentNoDev = 15;
    private int rentOneDev = 30;
    private int rentTwoDev = 45;
    private int rentThreeDev = 60;
    private int rentMajorDev = 150;

    public Buildings[] getBuildings() {
        return buildings;
    }

    private Buildings[] buildings = {NuclearPlant, WindFarm, HydroDamn};

    public RenewableEnergy() {
        super();
    }
}
```

Figure19: Implementation of Building Objects

This design allows individual buildings to have unique functionality within a field. For example, a unique specialEffect() method. This means that the player has complex and rewarding decisions to make. They must consider the Squares initial buy cost and future development opportunities, alongside the potential return on investment on future developments and any special effects they may have. For example, purchasing a FossilFuel Square may look lucrative at first, with its high return per round on Coal Mine and Oil Refinery buildings, however this may not be so attractive if these buildings have a specialEffect of having a 50% chance of being subject to a 'Green Tax' per round.

Testing

Unit testing carried out by the team consisted of both static and dynamic analysis techniques. During team meetings any problems identified during development were discussed and group members were assigned to fix or check for errors. Issues were presented in an informal way (screen sharing) and errors identified were corrected later, by the assigned group member, this also included any code inspections.

Dynamic analysis played a large part of testing as soon as classes were completed, both black box and white box aided the identification of issues and aid further development and ensure the game played as intended. Unit testing was primarily carried out on the classes square, board and player, the main aim of these tests was to ensure that variables were initialised correctly, and the getters and setters were performing as expected. The board class required the most testing to ensure that values of each square was returning values within the values of each game area. This testing was done throughout development and provided useful feedback to the group allowing completion of a working game relatively quickly. JUnit testing was also useful when extra features were added during development, as testing verified a working solution preventing a break in existing code. The results of some of the Junit testing can be seen below and all results of testing are in the project folder.

Acceptance testing was carried out by the group upon completion of the game so that any issues or errors could be identified which may have been overlooked during the development stages.

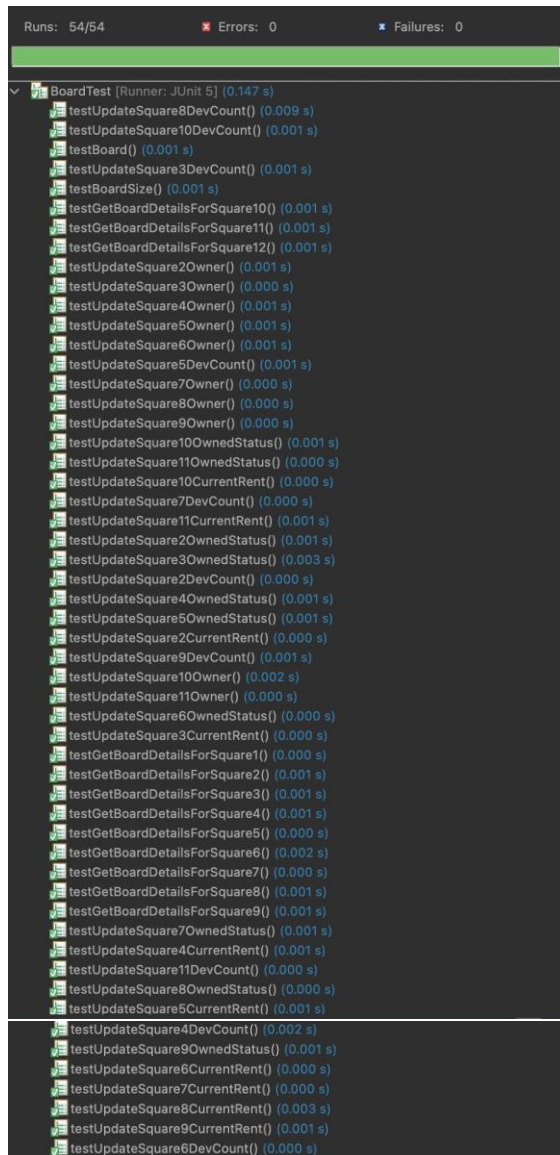
In order that an accurate acceptance plan was carried out, tests were devised around the 14 use cases. Tests were split amongst the group to be carried out during gameplay and results were recorded in the table contained in the appendix. The tests allowed the group to confirm if gameplay matched the original requirements devised as part of the use cases.

END OF MAIN REPORT

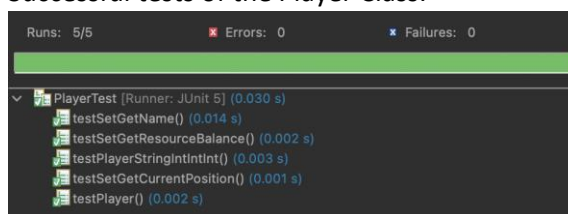
Appendix 1: Test Plan

1. Test Plan

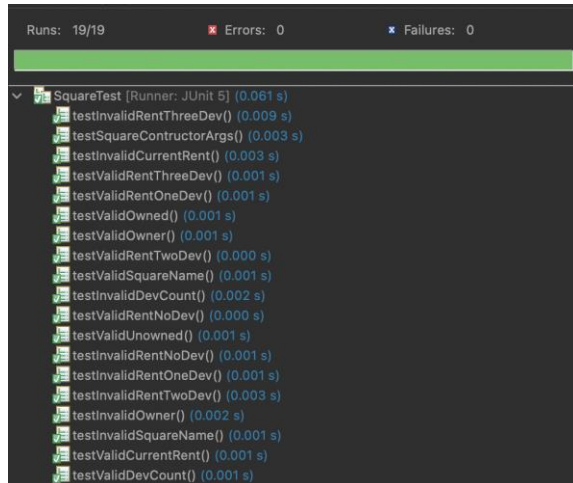
Successful tests of the Board class:



Successful tests of the Player Class:



Successful tests of the Square Class:



-User Acceptance testing plan

User Acceptance Tests

ID	Use Case Name	Description of Test	Test Initialisation	Test Inputs	Test Procedure	Expected Results	Passed or Fail	Tester
1	Number of Game Players	Test that the game accepts a valid number of players	Enter number of players	Number of players: "1"	Enter the number and press enter.	Game will for the name of the first player.	Pass	Gary/Damian
2	Number of Game Players	Test that the game rejects a number less than 1.	Enter number of players	Number of players: "0"	Enter the number and press enter.	Game will correct user and ask for between 1 and 4 users again.	Pass	Gary/Damian
3	Number of Game Players	Test that the game rejects a number more than 4.	Enter number of players	Number of players: "5"	Enter the number and press enter.	Game will correct user and ask for between 1 and 4 users again.	Pass	Gary/Damian
4	Number of Game Players	Test that the game rejects a letter or character.	Enter number of players	Number of players: "a"	Enter the character and press enter.	Game will request that the player(s) enter a number.	Pass	Gary/Damian
5	Number of Game Players	Test that the game rejects a white space for a name.	Enter number of players	Number of players: " "	Press spacebar and press enter.	Game will request that the player(s) enter a number.	Pass	Gary/Damian
6	Assigning Player Name	Test that the game accepts a valid name	Enter player name	Name: "Player1"	Enter the player name and press enter.	Game will request the next player name or start the game if only one player.	Pass	Noel/Andrew
7	Assigning Player Name	Test that the game rejects whitespace for a name	Enter player name	Name: " "	Press the spacebar once and press enter	Game will request that the player(s) enter name with character a to z/A - Z.	Pass	Noel/Andrew
8	Assigning Player Name	Test that the game rejects a number for a name	Enter player name	Name: "1"	Enter the number and press enter.	Game will request that the player(s) enter name with character a to z/A - Z.	Pass	Noel/Andrew

9	Assigning Player Name	Test that the game rejects duplicate names	Enter next player name	Name: "Player1"	Enter duplicate player name and press enter	Game will advise name already taken request player chooses different name.	Pass	Noel/Andrew
10	Player Turn	Test that the game displays whose turn it is	All players names have been correctly entered		Player takes turn	Player whose turn it is has balanced displayed and their name displayed correctly	Pass	Noel/Andrew
11	Player Turn	Test user is asked to select an action	Player's whose turn it is has name and balance displayed		Player takes turn	Player offered 7 options from list	Pass	Noel/Andrew
12	Player Turn	Test user can select option from list	Player has been offered 7 options		Player enters a number greater than 7	Player name and balance are displayed and asked to enter a relevant number	Pass	Noel/Andrew
13	Player Turn	Test user can select option from list	Player has been offered 7 options	Option: "1"	Player enters number 1	Game displays result of each dice, the total and the square the player has landed on	Pass	Noel/Andrew
14	Player Turn	Test user can select option from list	Player has been offered 7 options	Option: "2"	Player enters number 2	Game advises if player can develop an area	Pass	Noel/Andrew
15	Player Turn	Test user can select option from list	Player has been offered 7 options	Option: "3"	Player enters number 3	Game advises if player can complete energy field	Pass	Noel/Andrew
16	Player Turn	Test user can select option from list	Player has been offered 7 options	Option: "4"	Player enters number 4	Game moves to next player	Pass	Noel/Andrew

17	Player Turn	Test user can select option from list	Player has been offered 7 options	Option: "5"	Player enters number 5	Game displays current player balances	Pass	Noel/Andrew
17	Player Turn	Test user can select option from list	Player has been offered 7 options	Option: "6"	Player enters number 6	Game displays current board status	Pass	Noel/Andrew
18	Player Turn	Test user can select option from list	Player has been offered 7 options	Option: "7"	Player enters number 7	Game any player to quit the game	Pass	Noel/Andrew
19	Dice Roll	Test selecting 1 displays the dice roll result to screen	Player has been asked to select an option and they selected 1	Option: "1"	Player enters number 1	Game displays dice score and total score of roll	Pass	Gary/Damian
20	Dice Roll	Test system 'moves' player correct number of squares displayed by roll of dice.	Player has selected option 1		Check player's new position is displayed	New position displayed on screen		Gary/Damian
21	Dice Roll	Test system displays correct player opportunities and obligations	Player has opted to roll dice and result has been displayed by system		Check player's opportunities and obligations are displayed	Player obligations and opportunities are displayed correctly by system	Pass	Gary/Damian
22	Assigning Square Ownership	Test user can select an option to control an available area	System has displayed players opportunities	Option: Press "Y"	Player enters Y	Player's choice is confirmed player's resources are checked	Pass	Gary/Damian

23	Assigning Square Ownership	Test user is assigned control of respective area	Player has opted "Y" to purchase square		Check system has assigned correct square	Player is assigned control of respective area	Pass	Gary/Damian
24	Assigning Square Ownership	Test player resource is correctly updated	Player has been assigned square by system		Check system updates player's resource balance	Player's resource balance reflects correct amount for respective square	Pass	Gary/Damian
25	Offer Square Ownership	Test system asks player if they want to offer area to another player	Player has been offered to purchase control of square		Check the player is asked if they want to offer area to another player	Player is asked if they wish to offer area to another player	Pass	Andrew/Noel
26	Offer Square Ownership	Test user can choose to offer an area to another player	Player has been asked if they want to offer area to another player	Option: Press "Y"	Player enters Y	Player's choice is confirmed	Pass	Andrew/Noel
27	Offer Square Ownership	Test user can decide which player they can offer	Player has chosen to offer area to another player	Option: Enter name of player from list	Player enters chosen 'other' player	Player is offered all other available players to select one	Pass	Andrew/Noel
28	Offer Square Ownership	Test 'other' player is offered area	Player has selected 'other' player the respective area	Option: Press "Y"	Player decides if they wish to purchase control of area offered	Player selects if they wish to purchase available area	Pass	Andrew/Noel

29	Offer Square Ownership	Test player has option to offer 'other' players if declined by first 'other' player	'Other' player has declined offer to purchase area	Option: Press "N"	'Other' player enters "N" to decline offer to purchase	Player is asked if they wish to offer are to other players	Pass	Andrew/Noel
30	Offer Square Ownership	Test 'other' player's balance is displayed	Player has been assigned square by system		Check system updates player's resource balance	Player's resource balance reflects correct amount for respective square	Pass	Andrew/Noel
31	Player Turn	Test player can only have one dice roll per turn	Player has already rolled their dice and completed their obligations	Option: Press 1	Player presses 1	Player is reminded they have already rolled a dice and completed their turn	Pass	Gary/Damian
32	Develop an Area	Test that user can select an area to develop	Player has selected option "2" to develop an area they already own	Option: Press 2	Player presses number 2	Player's balance is checked and offered relevant area	Pass	Gary/Damian
33	Develop an Area	Test the game rejects incorrect area input	Player owns an area on their turn	Option: "fuel"	Enter incorrect area name and press Enter	Player is offered to retype the name of the area correctly	Pass	Gary/Damian
34	Develop an Area	Test the gamer accepts the	Player owns an	Option: "Oil"	Enter the correct name of area	Player's input is accepted and ECO coins are deducted from their account	Pass	Gary/Damian

		correct area to develop	area on their turn.					
35	Develop an Area	Test the user cannot develop a area with insufficient resources	Player has a low balance and has been offered an area to develop	Option: "Oil"	Enter the correct name of area to purchase	Player's attempt to purchase area is declined due to insufficient ECO coins	Pass	Gary/Damian
36	Increase Resources	Test the user receives 200 ECO coins when they pass the Start square	Player has passed the Start square on their turn		Player passes the Start square on their turn	Player balance update +200 ECO coins	Pass	Andrew/Noel
37	Decrease Player Resources	Test game displays player has landed on another player's area	Player has rolled dice and landed on another player's area		Take turns until player lands on another player's area	Player is advised they have landed on an area controlled by another player	Pass	Noel/Damian
38	Decrease Player Resources	Test player can reject contribution	Player has been offered a contribution	Option: "N"	Enter "N" to reject contribution	Player can reject contribution and both players accounts remain unchanged	Pass	Noel/Damian
39	Decrease Player Resources	Test player can accept contribution	Player has been offered a contribution	Option: "Y"	Enter "Y" to accept contribution	Player has correct amount of ECO coins withdrawn from their balance	Pass	Noel/Damian
40	Decrease Player resource	Test correct input when offered a contribution	Player has been offered a contribution	Option: "1234"	Enter incorrect choice to console	Player asked to confirm Y or N as their choice	Pass	Noel/Damian

41	Decrease Player resource	Test player's affordability is checked	Player has accepted contribution		Current player's balance checked	Player's remaining balance correctly displayed	Pass	Noel/Damian
42	Decrease Player resource	Test balance of both players accounts are correct after the transaction	Player has accepted contribution			Players balances are correctly adjusted after the transaction	Pass	Noel/Damian
43	Decrease Player resource	Test game ends when player has insufficient funds	Player has accepted contribution and current player has insufficient funds			Player has correct amount of Eco coins withdrawn and the game ends	Pass	Noel/Damian
44	Display Player Resource Balance	Test a player's resource balance is adjusted correctly	Player accepts option to develop an area	Option:"2"	Current player selects to develop an area	Player has corrected balance shown before transaction	Pass	Noel/Damian
45	Display Player Resource Balance	Test a player's balance is displayed correctly after transaction	Player accepts option to develop an area	Option: "2"	Current player selects to develop an area.	Game displays correct balance after transaction is complete	Pass	Noel/Damian
46	Display Player Resource Balance	Test the game displays the reason for the change in resource balance	Player has purchased a field to develop		Current player selects to develop a field	Game displays reason for change in current player's balance.	Pass	Noel/Damian

47	Quit Game	Test the game can end when option 7 is selected	Player has opted to end the game	Option:"7"	Current player has entered "7" from the options list	Game ends	Pass	Andrew/Gary / Noel/Damian
48	Quit Game	Test the game ends when all fields are fully developed	All fields are fully developed		Play game until all fields are fully developed	Game ends	Pass	Andrew/Gary / Noel/Damian
49	Failed Mission	Test the game displays Failed mission message	Player has insufficient ECO funds to continue		Current player has insufficient funds to continue	Failed Mission message is displayed	Pass	Andrew/Gary / Noel/Damian
50	Successful Mission	Test the game displays successful mission message	All areas are developed fully		Develop all areas fully with players	Successful message is displayed	Pass	Andrew/Gary / Noel/Damian
51	Final Game Status	Test game displays remaining resource balance by player name	Game has been completed		Game has finished either successful or failure	Resource balance is displayed by player name	Pass	Andrew/Gary / Noel/Damian
52	Final Game Status	Test the game displays current area ownership	Game has been completed		Game has finished either successful or failure	Current area ownership is displayed	Pass	Andrew/Gary / Noel/Damian
53	Final Game Status	Test the game displays current level of minor and major development	Game has been completed		Game has finished either successfully or failure	Minor and Major development is displayed	Pass	Andrew/Gary / Noel/Damian
54	Final Game Status	Test the game displays final game status	Game has been completed		Game has ended	Game end status is displayed	Pass	Andrew/Gary / Noel/Damian

Appendix 1.1: Methods Table

Method Name	Description
rollDice()	A simple method returning a random number between 1-6.
diceOutcome(Player, ArrayList<Square>, Scanner, Boolean):	Moves the players position on the board according to the number returned by rollDice(). If player passes Go, increments players ECOpoints by 200. Checks if the square landed on is owned by another player and if rent is owed. If the square is not owned by another player, asks the player if they want to buy the square. Calls rollDice(), buySquare(), purchasing(), chooseOtherPlayer.
buildMinorDev(Player, ArrayList<Square>)	Checks if all squares in the field are owned by the current player. Allows the player to develop if they own all the areas in the field. Does not allow them to develop if they do not own all areas in the field. Calls checkResourcesToBuyMinDev(), insufficientResources(), sufficientResources(), buyMinDev(), purchasing().
buildMajorDev()Player, ArrayList<Square>, Scanner)	Checks if the square has 3 minor devs built. If true, asks the player what area they would like to develop. If the user has enough ECO points and they wish to proceed, calls buyMajDev(). If the user does not have enough ECO points, displays a warning message to warn the user they will go bankrupt if they proceed. Calls checkResourcesToBuyMajDev(), insufficientResources(), sufficientResources(), buyMajDev(), purchasing().
viewPlayerStatus(ArrayList<Player> players)	Displays each players name and ECO coin balance to screen.
viewBoardStatus(ArrayList<Square>)	:Displays the name, field, ownership, development status, and rent of each square on the board

displayOptions()	Displays a list of game options for the player to choose from.
buySquare(Player, Square)	Changes ownership status of square passed in. Decrements ECO points of the Player by the Squares buy cost.
payRent(Player, String, Square, ArrayList<Player>)	Subtracts rent owed from the current players ECO points balance. Increments the Square owners ECO coins by the Squares rent cost.
checkResourcesToBuy(Player, Square).	Checks that a player has enough resources to buy an area
checkResourcesForRent(Player, Square)	Checks that a player has enough resources to pay rent of the current square.
buyMinDev(Player, Square)	Decrements the players ECO points buy the cost to build a minor development on the square. Increments the squares devCount variable.
buyMajDev(Player, Square)	Decrements the players ECO points buy the cost to build a minor development on the square. Sets squares isMajorDev variable to true
checkResourcesToBuyMinDev(Player, Square)	Checks if the current player has enough resources to purchase a minor development on the current square.
checkResourcesToBuyMajDev(Player, Square)	Checks if the current player has enough resources to purchase a major development on the current square.
chooseOtherPlayer(String, ArrayList<Player>, Scanner)	Allows the player to choose another player to offer the current square to if it is not already owned by any other player.
checkOwner(Square)	Returns the owner of the square passed in.
quitGame()	Sets quitGame to true.
insufficientResources()	Called when player does not have enough funds to purchase a square or development. Displays message telling the player that they have insufficient funds.
sufficientResources()	Displays sufficient resources message.
purchasing()	Displays message for purchasing an area.
boardComplete(ArrayList<Square>)	Checks is Major devs have been built on all areas.

successfulEcoMission(ArrayList<Player>)	Displays success message to game winners.
failedEcoMission(Board, ArrayList<Player>)	Displays unsuccessful game completion message.

Meeting Minutes

CSC7083 Group Component Work (GCW)

Date and Location: 19/01/2022 – Teams call

Meeting Objectives: Introductions

Attendees: DMG, NP, AMC, GD

Apologies: N/A

[Agenda:](#)

Introductions.

Agree initial plan of action.

[Discussions \(challenges, and progress by each team member on previous tasks\):](#)

DMG - made group aware of pregnancy & plan to mitigate impact on group work.

[Action Plan \(with allocations to each team member\)](#)

All - Read assignment spec & come up with a use case example and requirements

Meet again Wednesday 26th Jan to provide updates on action plan

Meeting Minutes

CSC7083 Group Component Work (GCW)

Date and Location: 26/01/2022 – Teams call

Meeting Objectives: User Requirements & Basic game structure

Attendees: DMG, NP, AMC

Apologies: GD

Agenda:

Review proposed Requirements & Use Cases.

Discussions (challenges, and progress by each team member on previous tasks):

Reviewed use cases & requirements.

Discussed basic game logic & rules.

NP - proposed very well thought out rules & logic for determining the winner of the game & what each field would be categorised as.

DMG - proposed game board consisting of 12 squares broken into fields & areas as per the project spec.

Action Plan (with allocations to each team member):

All – get GIT set up on local machines in advance of building game code.

AMC - proposed after meeting that Shared drive would be more useful for documentation & planning than GIT would be at this early stage.

Set up 02/02/22 : <https://qubstudentcloud.sharepoint.com/sites/SoftwareEngineeringGroup1>

Meeting Minutes

CSC7083 Group Component Work (GCW)

Date and Location: 26/01/2022 – Teams call

Meeting Objectives: User Requirements & Basic game structure

Attendees: NP, AMC, GD

Apologies: DMG

Agenda:

Review proposed Requirements & Use Cases.

Discussions (challenges, and progress by each team member on previous tasks):

Sharepoint space set up:

<https://qubstudentcloud.sharepoint.com/sites/SoftwareEngineeringGroup1>

Team discussed pressures of project for another module and decided to put Software engineering on hold until Web Development project was complete (Week commencing 21st March)

Action Plan (with allocations to each team member):

Regroup Week commencing 21st March to further discuss project.

Meeting Minutes

CSC7083 Group Component Work (GCW)

Date and Location: 26/03/2022

Meeting Objectives: Assess where we are and where we need to go.

Attendees: Noel, Damian, Andrew, Gary

Apologies:

Agenda:

1. Discussion to find out where we are and where we need to go.
2. Agree a meeting and delivery schedule going forward.
3. Assign tasks to be completed for the next meeting.

Discussions (challenges, and progress by each team member on previous tasks):

1. Return to meetings following a 4 week break to focus on WebDev.
2. We discussed where we currently are as a group, picking up from our last meetings. We at this point we have defined requirements, some user stories, the game board, and rules for fields and areas.
3. Agreed that the next steps would be to finalise our user stories.
4. Gary talked us through a code review of what he had been working on based on the user stories and game rules previously defined.
5. Agreed that we would meet 2x per week going forward until the project is finished (Mondays and Fridays at 7pm)

Action Plan (with allocations to each team member):

- 1- Action for next meeting is for each team member to come up with 2 user stories for the next meeting.
- 2- Gary & Noel– create Board and Square Classes and tests based of game rules.
- 3- Damian & Andrew – create Player Class and tests based of game rules.

Meeting Minutes

CSC7083 Group Component Work (GCW)

Date and Location: 28/03/2022

Meeting Objectives: Review Use Cases and Assign Further Work

Attendees: Noel, Damian, Andrew, Gary

Apologies:

Agenda:

1. Define report Structure.
2. Assign further work items (UML diagrams, Game rules section, and Board creation).
3. Code and Use Case Review.

Discussions (challenges, and progress by each team member on previous tasks):

1. Discussed current work complete. Gary discussed the current use cases he has typed up for the report based on the use cases combined in the excel.
2. Damian you've proposed to add a penalty if the user decides to skip their turn.
3. Discussed what happens if two people have the same name and decided that two players should not be allowed to enter the same name.
4. Decided to go with the circular game board design (gameboard.jpg).

Action Plan (with allocations to each team member):

1. Noel – Create UML Class diagram.
2. Gary – Create UML Sequence diagram.
3. Damian- Type up game rules and overview/introduction.
4. Andrew – Going to complete the Game board and UML case diagram.

Meeting Minutes

CSC7083 Group Component Work (GCW)

Date and Location: 31/03/2022

Meeting Objectives: Review current work complete.

Attendees: Damian, Gary, Andrew, Noel

Apologies:

Agenda:

1. Review UML Class diagram.
2. Review UML sequence Diagram
3. Discuss tile special effects for future amendments.
4. Discussion of git process.
5. Agree on next steps.

Discussions (challenges, and progress by each team member on previous tasks):

1. Noel showed his UML Class diagram and asked the group for feedback. Questions around whether we need to have all the getters and setters in the class diagram. It was agreed that he would reach out to Janak to ask.
2. Gary showed progress on UML sequence diagram. It was agreed that he would send off to Janak for feedback.
3. Discussion around GitLab use and how to properly push to dev branches and create a merge request.
4. Talked about potential to add special effects to each tile as a future improvement.

Action Plan (with allocations to each team member):

1. Agreed to continue with current actions agreed in last meeting and to present work at next meeting on 4th April.

Meeting Minutes

CSC7083 Group Component Work (GCW)

Date and Location: 04/04/2022

Meeting Objectives: Review current work complete.

Attendees: Damian, Gary, Andrew, Noel

Apologies:

Agenda:

1. Reviewed code and work done so far.
2. Discuss Test Plan and adherence to process.
3. Discuss bugs found so far.

Discussions (challenges, and progress by each team member on previous tasks):

1. Damian discussed a change to the board class from implementing an Array of type square to implementing an ArrayList of type square. This change was required to allow the game board to be extended in the future if desired.
2. Discussed how we would proceed with Junit testing and assigned to code to be written and uploaded.
3. Assigned Game methods to be written to individuals.
4. Noel creating classes for area fields but going to work on side branch to see if its implantable.

Action Plan (with allocations to each team member):

1. Andrew assigned buildMinorDev, BuyMinorDev, displayOptions, checkResourcesToBuyMinorDev, rollDice, sufficientResources, insufficientResources methods.
2. Gary Assigned viewPlayerstatus, diceOutcome, boardComplete methods
3. Damian Assigned buildMajorDev, checkResourcesToBuyMajorDev, quitGame, successfulEcoMission, failedEcoMission methods
4. Noel Assigned chooseOtherPlayer, payrent, buysquare, checkResourcesForRent, checkOwner, viewBoardstatus, purchasing methods

Meeting Minutes

CSC7083 Group Component Work (GCW)

Date and Location: 07/04/2022

Meeting Objectives: Review New Classes and Report Progress

Attendees: Damian, Gary, Andrew, Noel

Apologies:

Agenda:

1. Review new classes developed and decide if they will be implemented into the code
2. Decide on date for finalising report.
3. Assign testing tasks

Discussions (challenges, and progress by each team member on previous tasks):

1. Discussed the new class structure. It was decided that it would be implemented into the code.
2. Provisional deadline of 15th April set for have first draft of a completed report.
3. Discussed testing process and who would lead.

Action Plan (with allocations to each team member):

1. Report first draft deadline 15/04/2022.
2. Gary and Andrew conducting user testing and junit testing current game code.
3. Noel developing potential building classes.
4. Damian making fixes to code identified in review.

Meeting Minutes

CSC7083 Group Component Work (GCW)

Date and Location: 22/04/2022

Meeting Objectives: Review New Classes and Report Progress

Attendees: Damian, Gary, Andrew, Noel

Apologies:

Agenda:

1. Review of current code.
2. Review current state of report
3. Review tests and user acceptance criteria.

Discussions (challenges, and progress by each team member on previous tasks):

1. It was discussed that the report is now in a final state and that work now needs to be prioritised on formatting, proof reading and building the appendix.
2. Future improvements section reviewed and signed off on.
3. Review of tests carried out during the week by Andrew.

Action Plan (with allocations to each team member):

1. Noel to format report over the weekend and build the appendix, adding in screenshots of project management.
2. Andrew, Damian and Gary to proofread report after formatting complete.

Meeting Minutes

CSC7083 Group Component Work (GCW)

Date and Location: 25/04/2022

Meeting Objectives: Review New Classes and Report Progress

Attendees: Damian, Gary, Andrew, Noel

Apologies:

Agenda:

1. All sign off on final report.
2. Agree on scores for all group members.
3. Submit report.

Discussions (challenges, and progress by each team member on previous tasks):

1. It was agreed that everyone was happy with the final report and individual contributions. All members have signed off each members contribution.
2. It was agreed by all group members that every member of the group made a very good contribution throughout the project, attending all meetings and submitting work on time. Gary was awarded a 5 for code contribution to recognise the vital contribution he played in design and helping others. Noel was awarded a 5 in project management to recognise the contribution made leading project delivery.
3. It was agreed that Noel will upload and submit the video report on 25th May 2022.

Action Plan (with allocations to each team member):

1. Final report signed off on. All group members have agreed to scores and report uploaded on 25th May 2022.

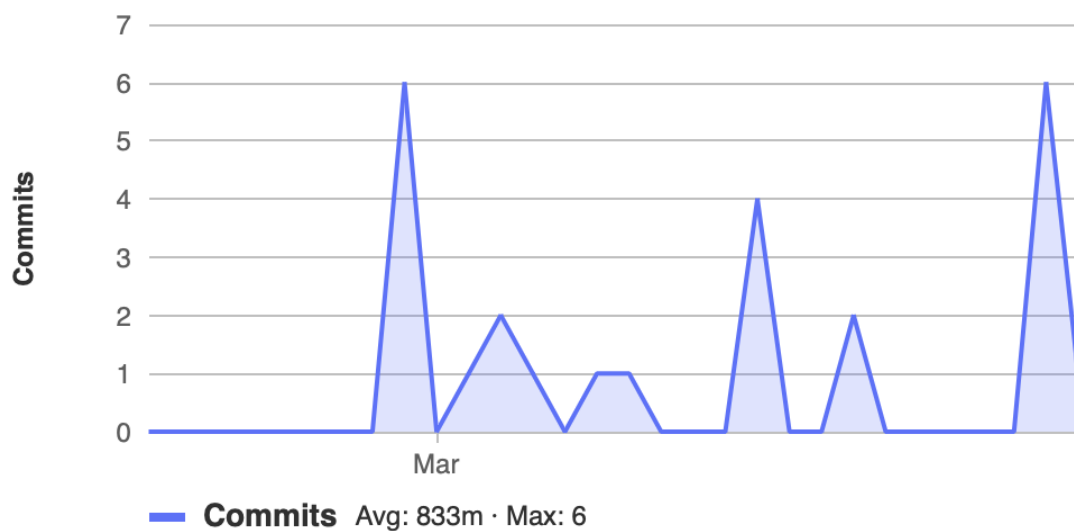
Appendix 3: Project Management

Gitlab contribution graphs

Andrew:

40318021

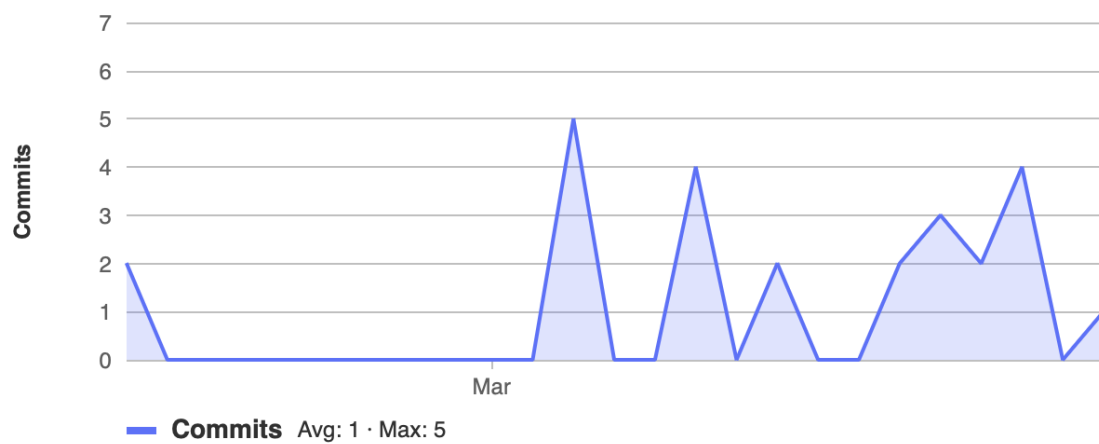
25 commits (amcclelland21@qub.ac.uk)



Damian:

40078137

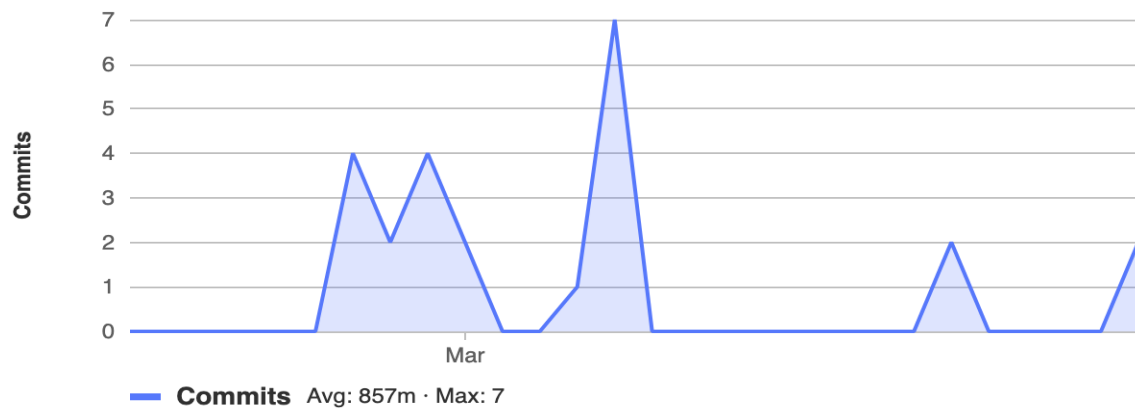
25 commits (dmcgill01@qub.ac.uk)



Gary:

40314509

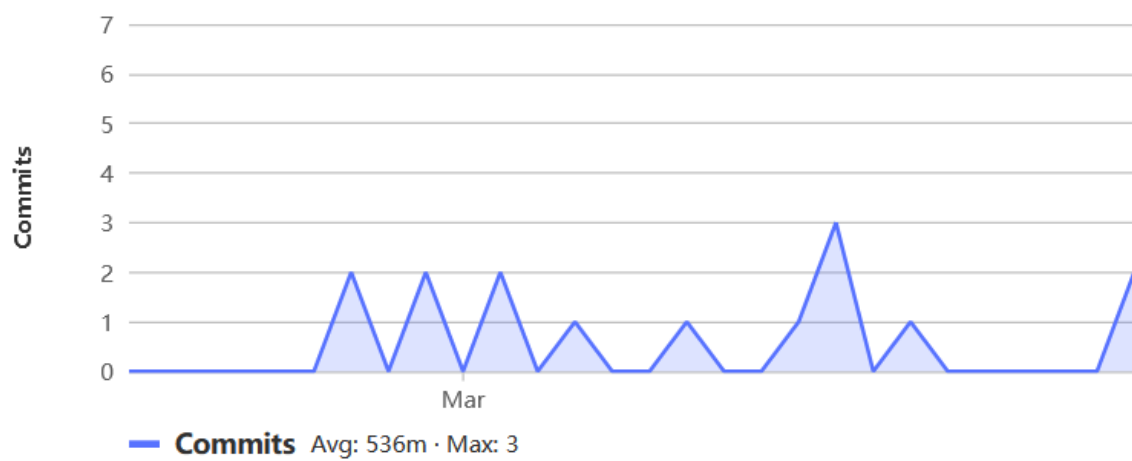
24 commits (gdonnelly90@qub.ac.uk)



Noel:


































Noel





































15 commits (40102820@qub.ac.uk)



Sample of GitLab Commit History

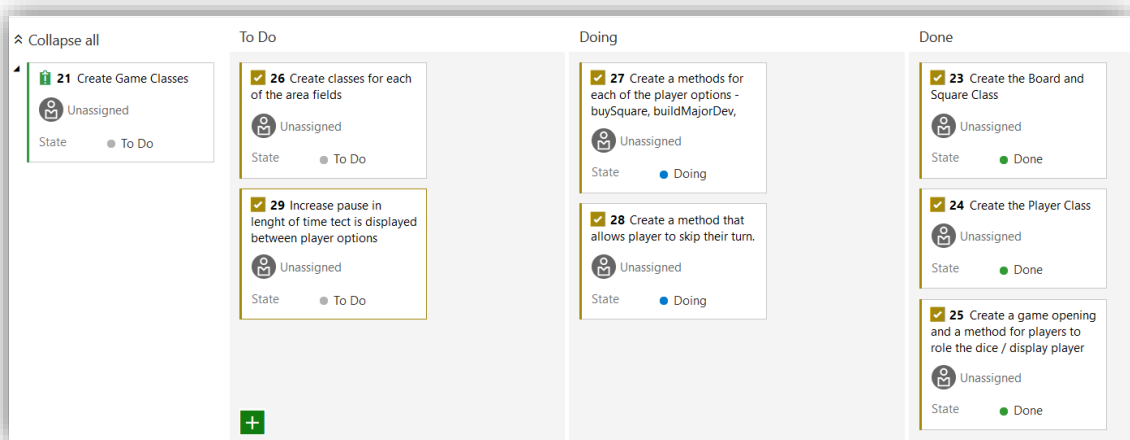
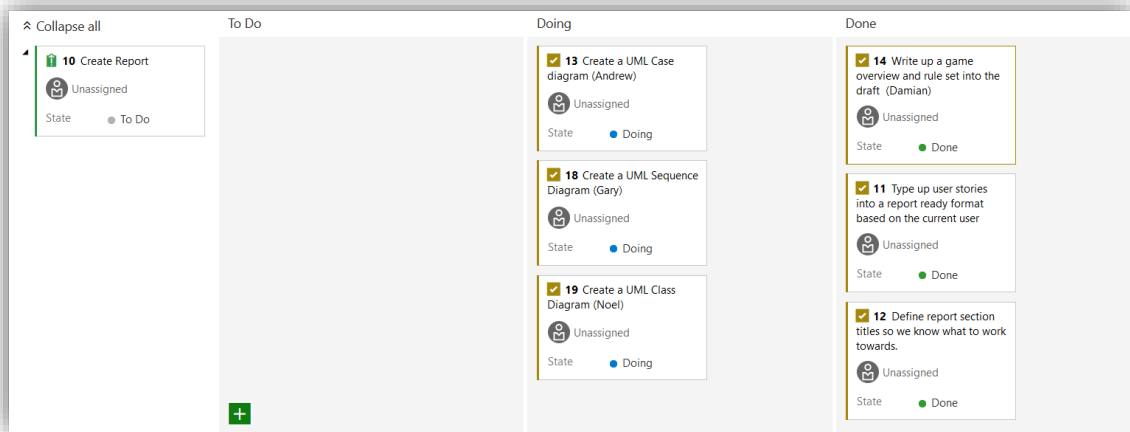
 Fixing some code in player class Noel authored 3 weeks ago	e8a90160		
 Square class 40318021 authored 3 weeks ago	46a0e87f		
 Delete Square.java 40318021 authored 3 weeks ago	b3992cf5		
 Square class, set lower and upper vars 40318021 authored 3 weeks ago	8ba864ef		
30 Mar, 2022 2 commits			
 Use Cases Diagram 40314509 authored 3 weeks ago	e106a7a2		
 Use Cases 40314509 authored 3 weeks ago	f2399664		
29 Mar, 2022 9 commits			
 Merge branch 'Dev' into 'main' ... 40102820 authored 3 weeks ago	4349eb54		
 Commit of Player Class to Dev branch Noel authored 3 weeks ago	0947f9ec		
 Merge branch 'revert-f11fab2a' into 'main' ... 40102820 authored 3 weeks ago	6884bea3		

 Merge branch 'DevAM' into 'main' ... 40318021 authored 3 weeks ago	17b53a45		
 Replaced Square.java / added constructor and public methods to Player class 40318021 authored 3 weeks ago	7bc216ef		
 Merge branch 'DevNP' into 'main' ... 40102820 authored 3 weeks ago	b00a834c		
 Miro board from first brainstorming design session Noel authored 3 weeks ago	be329e90		
 Merge branch 'DevNP' into 'main' ... 40102820 authored 3 weeks ago	7f6d5c7b		
 Update to UML Class Diagram and code update to player class. Noel authored 3 weeks ago	8d781914		
01 Apr, 2022 3 commits			
 Update MainGame.java 40314509 authored 3 weeks ago	3da67a35		
 Merge branch 'DevGD' into 'main' ... 40314509 authored 3 weeks ago	111d8b82		
 MainGame initial workings upload 40314509 authored 3 weeks ago	0c110b26		
31 Mar, 2022 19 commits			
 Merge branch 'DevGD' into 'main' ... 40314509 authored 3 weeks ago	52d6d53a		
 Merge branch 'DevAM' into 'main' ... 40318021 authored 3 weeks ago	a7e57dad		

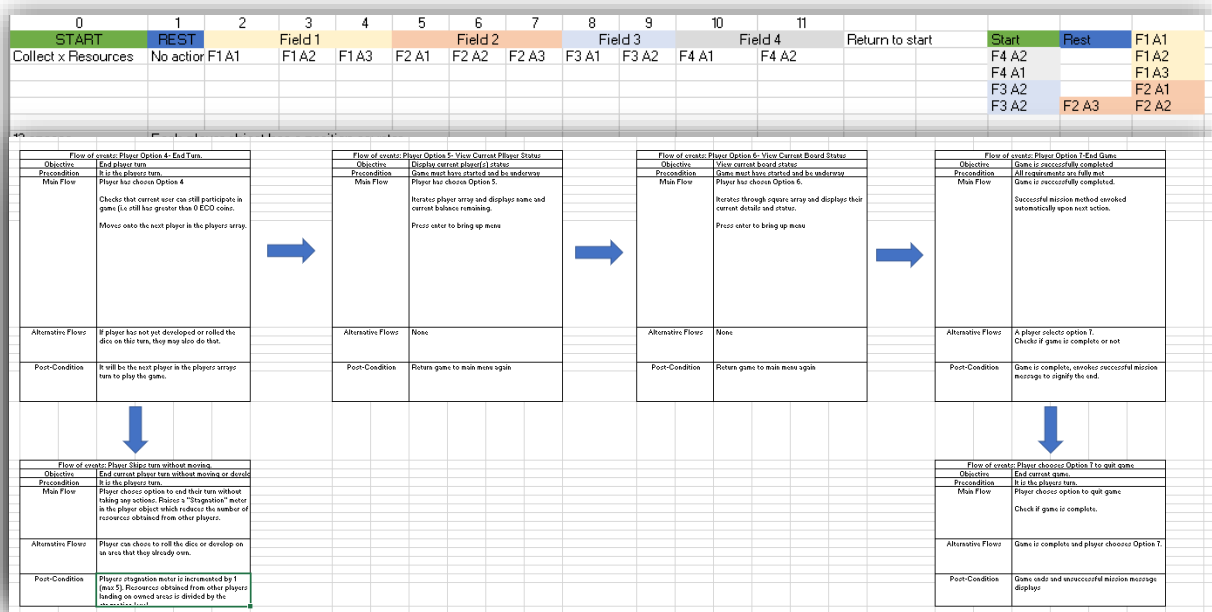
	adding Check resources methods to MainGame 40078137 authored Apr 06, 2022	de381f41		
	Merge branch 'DevDMG2' into 'main' *** 40078137 authored Apr 06, 2022	cb3172b4		
	moving MainGame.java to package folder 40078137 authored Apr 06, 2022	482dfe32		
05 Apr, 2022 7 commits				
	Update MainGame.java 40314509 authored Apr 05, 2022	bda6ee84		
	Update MainGame.java 40314509 authored Apr 05, 2022	37d35ecc		
	Update MainGame.java 40314509 authored Apr 05, 2022	868d62db		
	Update MainGame.java 40314509 authored Apr 05, 2022	a93aa857		
	Update MainGame.java 40314509 authored Apr 05, 2022	83285368		
	Update MainGame.java 40314509 authored Apr 05, 2022	bb82b984		
	Update MainGame.java 40314509 authored Apr 05, 2022	8b1e6b1e		
04 Apr, 2022 12 commits				
	Merge branch 'DevNP' into 'main' *** 40102820 authored Apr 04, 2022	685058b4		
	Class development for area fields. TBC wether this will be used in final submitted code. Noel authored 2 weeks ago	eb0739ca		

Examples of DevOps Task Tracking

Collapse all 16 Create Game Board Unassigned State ● To Do	To Do	Doing	Done
			17 Decide upon and finalise final game board Unassigned State ● Done
			32 Create a visual depiction of the Game Board Unassigned State ● Done
	8 Review User stories at next team meeting and make sure requirements are being Unassigned State ● To Do		7 Each Person develop user stories for Monday 28th Unassigned State ● Done
	31 Develop the game rules. Add area fields to excel doc and decide on class Unassigned State ● To Do		30 Each group member add 2 user stories each to the excel sharepoint doc for next Unassigned State ● Done



Early Excel Use Case Development and Field Planning



Early Field and Game Rule Development

Field NameField Area		Field NameField Area		Field NameField Area		Field NameField Area	
Renewable Energy Base Cost: £100	Nuclear High Cash reward. Zero Carbon cost. Very high cost to buy Effect: Every turn there is a 1% chance of an explosion, ending the game for that player. Buildings in this area: -Nuclear Power Plant	Fossil Fuels Base Cost: £150	Coal High Carbon Medium Cost High Cash Effect: 1% additional carbon tax per round 5% chance of a one of Green Levy of 10% of cash per round. Building: Coal Mine	Technology Base Cost: £200	Electric Cars Low Carbon High Cash High Cost Buildings: Tesla	Community: Base Cost £50	Charity Zero Cash Zero Carbon Very cheap to buy Effects: -100 carbon every round Building types -Oxfam -Etc etc
	Hydroelectric Medium cash Reward Very Low Carbon Cost High Cost to buy. Buildings in this area: -Hydroelectric Dam		Gas Medium Carbon Medium Cost Medium Cash Buildings -SSE Airtricity -Power NI		Negative Carbon production Zero Cash High Cost Effects: Reduces players carbon by 5% each turn but does not produce any cash. Buildings -Carbon Capture inc -The Renewable		Recycling Centre Low Cash Zero Carbon Very Cheap to buy Effects: 1% chance per round to have Carbon reset to zero
	Wind Energy Low Cash Reward Low Carbon Cost Low Cost to buy Effects: Government Subsidies. +2% cash per round. Buildings: -Large Wind farm (produces most cash) -Small Wind farm (produces least cash)		Oil Very High Cash High Carbon High Cost Effects: 2% chance of an oil spill per round costing the player 50% to their cash to clean up Buildings -Oil Refinery				