CSC7083 – Software Engineering

Save Our Planet

Group 6

Matthew Walsh -15316068
Anthony McGarrigan –20109008
Glenn McGookin – 40242327
Anthony Considine – 12170992
Niall Murphy – 40245248

23rd April 2020

CSC7083 PEER ASSESSMENT: SAVE OUR PLANET

Evaluation	Group Number: 6			
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)
Anthony Considine	3	4	3	105
Anthony McGarrigan	5	5	5	115
Glenn McGookin	3	4	3	105
Niall Murphy	3	4	3	105
Matthew Walsh	5	5	5	115

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

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

Declaration

"I declare that I have read the Queen's University regulations on plagiarism, and that any contribution I have made to 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
Anthony Considine	*22/04/2020*	*I agree to the terms of the declaration*
Anthony McGarrigan	*22/04/2020*	*I agree to the terms of the declaration*
Glenn McGookin	*22/04/2020*	*I agree to the terms of the declaration*
Niall Murphy	*22/04/2020*	*I agree to the terms of the declaration*
Matthew Walsh	*22/04/2020*	*I agree to the terms of the declaration*

Contents

1 INTRODUCTION TO THE PROJECT

1.1 Introduction

The following report accompanies the submission of our 'Save Our Planet' virtual board game. It will set out to detail the key steps involved in our design process, implementation, testing and general project management.

1.2 Background to the Project

The team was tasked with the development of a virtual board game based on the principle of the popular Monopoly game, with a thematic focus on 'caring for the planet'. The implementation of this theme, and how it dictates the rules of the game, were left to the discretion of the development team.

1.3 Personnel & Responsibilities

The project has been delivered by a team of 5 developers, listed below:

- Anthony Considine;
- Anthony McGarrigan;
- Glenn McGookin;
- Niall Murphy; and
- Matthew Walsh.

1.4 Progress & Monitoring

Project management and monitoring are detailed in greater detail in Appendix 4. In summary, the project monitoring took place via the following primary methods:

- Weekly Team Meetings: In the initial project stages these principally took place face to face in the context of timetabled classes. As the COVID-19 situation progressed these were migrated to MS Teams;
- File Sharing: While the project specification suggested the use of Canvas for the purpose of any file sharing, MS Teams was preferred medium for the team owing to the familiarity of it being an established tool for the team to work via due to its use in previous projects for the group.
- Version Control: Distribution and sharing of files relating to the project were primarily done via the MS Teams platform. Program code version control was managed via the GitLab platform. Both of these aspects are demonstrated in Appendix 3.

1.5 Presentation

Presentation of our completed project can be viewed via the YouTube link below:

https://www.youtube.com/watch?v=bNWc4vJS-mo&feature=youtu.be

2 REQUIREMENTS ANALYSIS

2.1 Introduction

The gameplay, detailed below, is based on the following rationale consistent with the environmental theme of the assignment.

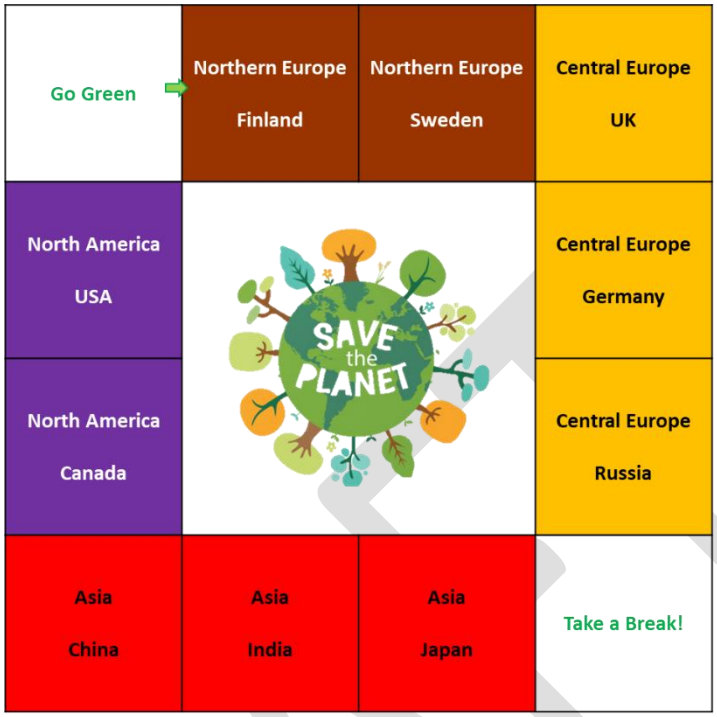
- Each player will start the game with a specified carbon footprint assigned to them;
- As they move around the board, if they land on an unoccupied area, they have the opportunity to establish themselves there. To do so, they assume a carbon footprint cost;
- The carbon cost associated with each area is based on a general ranking of their real-world carbon dioxide emissions[1]. Northern European countries, who have a relatively low emissions rate, incur a lower cost for establishing there. A player incurs a higher cost for establishing themselves in a larger polluter.
- If a player lands on a region someone else is established on, they must pay a 'Carbon Tax' whereby they assume a proportion of the carbon cost held by the other player.
- If a player owns an entire region they have the opportunity to install environmental developments (e.g. small development = wind farm, large development = hydroelectric plant) which will incur an initial carbon cost for their production & installation.
- Once installed, they will enable the occupier to charge a higher carbon tax on anyone who lands on their area.
- There are 3 game ending scenarios:
  - A player quits the game. That person defaults to last in the game, and the remaining players are ranked on the current state of play;
  - One player offsets their carbon footprint completely to 0. The game ends and the players are ranked;
  - One player breaches an upper carbon limit. The game ends, and the players are ranked.

2.1.1 The Game Board

While the game will be played via text only, with no actual representation of the board via a GUI – the figure below displays the board, the text game will seek to replicate to the user.

[1] *https://en.wikipedia.org/wiki/List_of_countries_by_carbon_dioxide_emissions*

Figure 2.1: Save Our Planet Game Board Representation



2.2 Game Rules

2.2.1 Setup

- The game will start with 2 – 4 players;
- Each player must enter a name which must be unique;
- Each player is allocated an initial 'carbon footprint' of 4,500 tonnes of CO2 at the start of the game;
- Each player starts at the 'Go Green' area;
- Each player will take their turn in the order they entered the game in; and
- A player will roll two dice at each turn.

2.2.2 Start & Move

- Each player takes a turn in rolling the dice
- After the last player rolls the dice, a new round starts
- The dice will be the indicator of how many spaces to move the player clockwise around the board
- The player will remain on the square occupied, and will proceed from this position on the players next turn
- Two players may occupy the same square at the same time

- Each time a player lands on or passes the 'Go Green' square, the player will reduce their carbon footprint by '600' amount
- If a player lands on the 'Take a Break' area, nothing happens, and next player will take their turn.

2.2.3 Controlling Areas & Fields

- If a player lands on a square they do not control, the player may choose to develop on the area, and incur a specified carbon cost which will be added to their total carbon footprint.
- If a player decides not to acquire the area, the next player takes their turn.
- When a player lands on an area already owned by another player, they will acquire a portion of the owners carbon footprint. This is added to the players total carbon footprint, and deducted from the owner's.
- In addition, the player who lands on the area, has the option to purchase the area from the player who owns it.
  - The player who owns it has the decision to accept or decline the request.
  - If the player who owns the area accepts the purchase request, the player who requested the purchase acquires the area from the player who owns it and will acquire its carbon cost while the player who owns the area, their carbon footprint will be reduced by the areas carbon cost.

2.2.4 Developing Areas

- A player must control all areas in a field before they create any developments on an area within that field.
- A player can create a minor or major development on any of the areas in the field that is owned.
- A major development can only be developed when a minor development has been created on each of the areas in the owned field.
- A player can only have three minor and one major development per area in each field in total.

2.2.5 Ending the Game

- A player can choose to quit the game at the start of their turn and not the end.
- If a player decides to quit the game, they will lose and the player with the least carbon footprint wins.
- If a player eliminates their carbon footprint, the game ends, and the player with no carbon footprint (or less than) wins. If a players carbon footprint exceeds the set threshold (9000 MAX CARBON LIMIT), the game will end, and the player with the lowest carbon footprint wins.

Table 2.1: Areas and their associated outlay costs & off setting potential

Field	Area	Cost to Purchase	Cost per Small Development	Cost of Large Development	Pay Pollute Cost	With 1 Minor Development	With 2 Minor Developments	With 3 Minor Developments	With a Major Development
Northern Europe	Finland	100	100	100	30	90	270	810	1,620
Northern Europe	Sweden	100	100	100	30	90	270	810	1,620
Central Europe	UK	200	200	200	60	180	540	1,620	3,240
Central Europe	Germany	200	200	200	60	180	540	1,620	3,240
Central Europe	Russia	200	200	200	60	180	540	1,620	3,240
Asia	Japan	200	200	200	60	180	540	1,620	3,240
Asia	India	200	200	200	60	180	540	1,620	3,240
Asia	China	200	200	200	60	180	540	1,620	3,240
North America	Canada	300	300	300	90	270	810	2,430	4,860
North America	USA	300	300	300	90	270	810	2,430	4,860

2.3 Use Cases

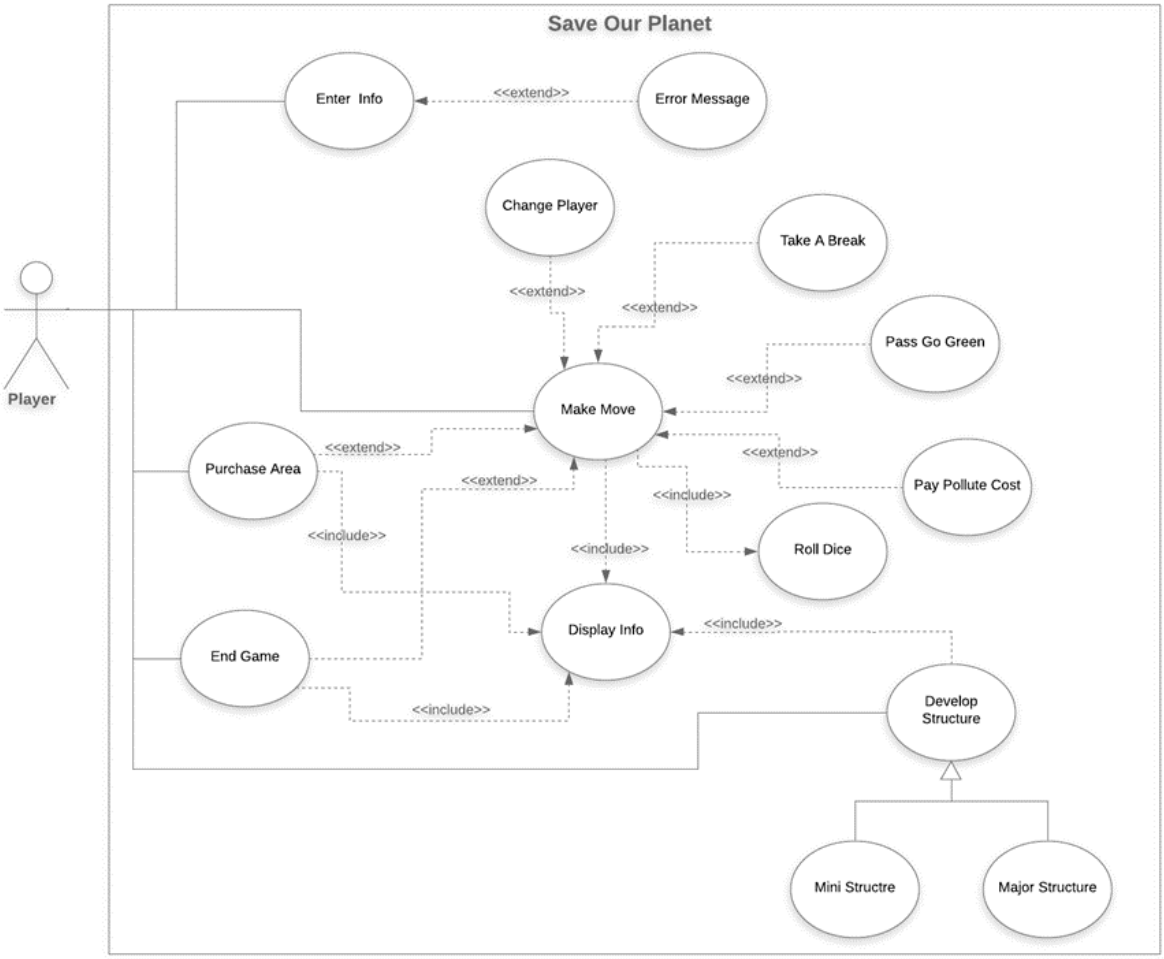
With the requirements and the rules of the game set, the team then sought to develop a comprehensive list of use cases. The table below displays the list of use cases derived for the game.

Table 2.2: Save Our Planet Use Cases

Use Case Reference	Use Case Name
UC1	Enter Info
UC2	Display Error Message
UC3	Make Move
UC4	Display Info
UC5	Change Player
UC6	Roll Dice
UC7	Pay Pollute Cost
UC8	Go Green
UC9	Purchase Area
UC10	Take a Break
UC11	Develop Structure
UC12	Develop Minor Structure
UC13	Develop Major Structure
UC14	End Game

The full details of the use cases, detailed above, can be found in Appendix VI. Figure 2.2. below presents the final Use Case Diagram adopted by the group.

Figure 2.2: Use Case Diagram (Final Iteration)

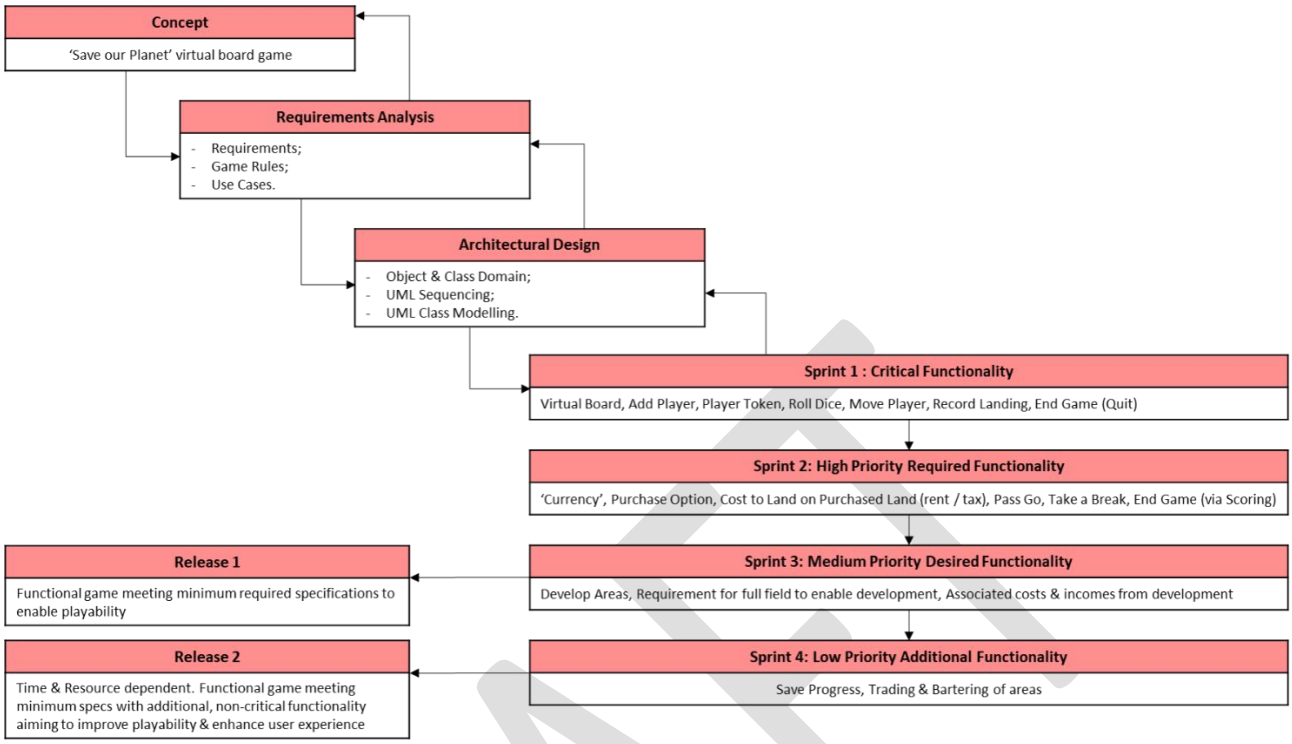


2.4 Development Methodology

For the development of this game, the group agreed upon the adoption of an 'iterative' development model. This methodology has been chosen as the requirements for the game are well established, while also allowing for the inclusion of 'Agile' practises in the development process. The well-established nature of the requirements, and the 'adherence to process' constraints within the project specification itself, were not seen as conducive to a fully agile development process. This iterative model adopted is seen as the best way to allow both these influences to work together.

Our application of this iterative model is detailed in Figure 2.3 below.

Figure 2.3: Iterative Design Methodology



Through our Iterative Design Methodology, it was envisaged that the full core requirements of the project could be realised within 3 development sprints. Sprint 1 would see the implementation of the foundational elements of the game's 'engine'. Sprint 2 would see the implementation of the minimum elements required for a game to be played. Sprint 3 would see the enhancement of the game to fully meet the requirements set. This would be our initial release.

The methodology made provision for an additional sprint for the development of non-essential functionality that would seek to enhance the user experience further. However we were advised that to do this post-release 1 would require a repeat of the entire methodological process up to that point. The group agreed there was insufficient time resource in order to carry out this part.

2.4.1 Methodology Timetable

The timetable and schedule of activities to accompany the methodology above are presented in detail in Appendix 4 of this report.

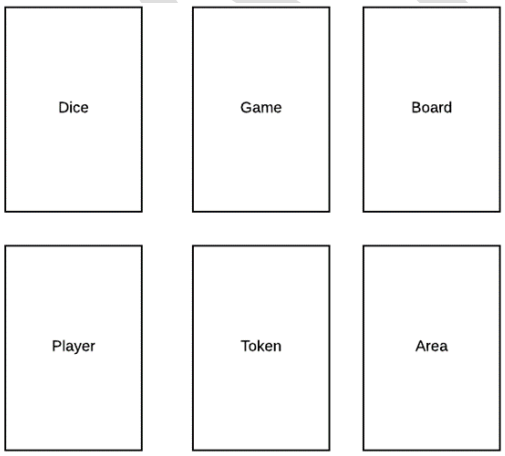
3 REALISATION

This section details the process by which the requirements and use cases were developed and then utilised to build a framework of game design and then implementation. Each section presents a detailed step by step guide on the process of candidate discovery, through to the development of domain models and the sequencing diagrams required for the UML class diagrams for each iteration. Diagrams to this effect are presented in this section as well as Appendix 4 of this report.

3.1 Candidate Classes & Object Discovery

To perform initial steps we performed our candidate object discovery by reading through each use case and noting down any nouns (objects), doing (operations), being (inheritance) and having (aggregation) verbs, adjectives (attributes) and constraints so that we could later gather these nouns and eliminate any vague, ambiguous or repeated noun and isolate the remainder to our candidate classes which would form the basis of the Save Our Planet Game. Through analysis of noun phrases present in the textual descriptions of the system scope (stated in use cases) identified Player, Token, Dice, Game, Board and Area as candidate conceptual classes to be included in the initial domain model.

Figure 3.1: Candidate Conceptual Classes



Depicted in table 3.1, the highlighted rows show the candidate classes we would use to form the basis of the Save Our Planet Game and aid in modelling our initial Domain Model. A detailed overview of our approach to our candidate object discovery to aid in our realisation can be found in Appendix VI, 8.3 Candidate Object Discovery.

Table 3.1: Candidate Classes

Candidate Objects	NOUN/OBJECT	Elimination Reason	Responsibilities	Collaborators
	Display info	Action		
	Console	Vague		
	User	Is Player		

1	Token		Marks the position of a player on the board Move to specific square Inform the player of their current position on the board	Interacts with square Interacts with board
2	Player		Move the token Take a turn Develop Pay Carbon Footprint Pay Pollution Pass Go Green Return dice total	Interacts with token Interacts with dice Interacts with carbon footprint Interacts with developments
3	Dice		Throw and return a whole number	
	System	Is Game		
4	Game		Keep a record of players Keep a record of game stats Know whose turn it is	Interacts with player Interacts with board
5	Board		Knows the order of squares on the board Knows what squares belong to a field The board cannot be changed	
	Destination square	Is Square		
	Square	Is the same as area		
6	Area		Know its own attributes (rate, name, pollution cost, owner)	Interacts with field Interacts with player Interacts with carbon footprint
	Carbon footprint	Attribute of player; cannot exist without a player being created		
	rate	Attribute of square; square is given a purchase rate/cost		
	Pollute cost	Attribute of square; cost given to player landing on square		
	Field	Attribute of area		
	Minor structure	Inherits from structure		
	Major structure	Inherits from structure		
	Scoreboard	Action / derived from carbon footprint		
	Go green square	Inherits from square		
	Take a break square	Inherits from square		
	Structure	Is an attribute of an area		

3.2 Domain Models

The initial steps of our process investigated the behaviour of the Make Move, Enter Info and Roll Dice use cases and are aimed at discovering the functionality needed to perform a basic scenario of the Save Our

Planet gameplay. This first iteration/simulation of gameplay involves 2-4 players playing a number of rounds until a player decides to quit. Playing a round entails each player taking a turn by rolling two dice and advancing their token clockwise around the game board from the currently occupied area to a destination area. The destination area is determined by summing the value of the dice rolls and adding this value to the index of the area the player's token is positioned on at the start of the turn.

The player's name, the sum of the dice rolls and information about the destination area are displayed in the console window. This iteration of gameplay is not concerned with allocating resources, purchasing an area, developing a field, and winning or losing.

What we know:

- A game is played by 2-4 players and is played on a single board;
- Each player has a unique name;
- Each player moves a token around the board, and the player can own 1 token from a selection of 8 tokens;
- Each token has a name (i.e., boat, car, plane, etc.);
- Each player's starting position is the go green area;
- The game is played by rolling two dice, the summed face value of the rolled dice is added to the index of the currently occupied area to determine the value of the destination area;
- An area has a name and index, the index determines the position of each area in the board layout;
- An area can be occupied by 0-4 tokens at any time during gameplay;
- The game board consists of 12 areas, a go green area, take a break area and 10 property areas.

Figure 3.2: Initial Domain Model

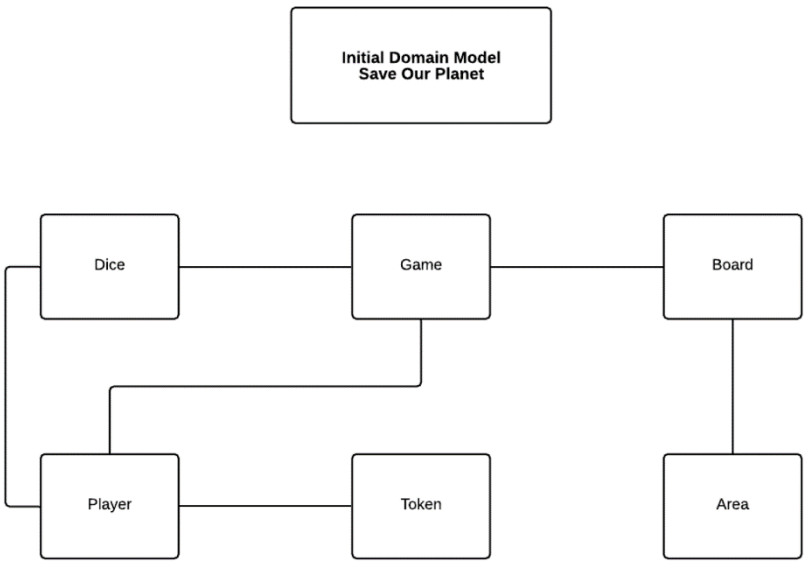
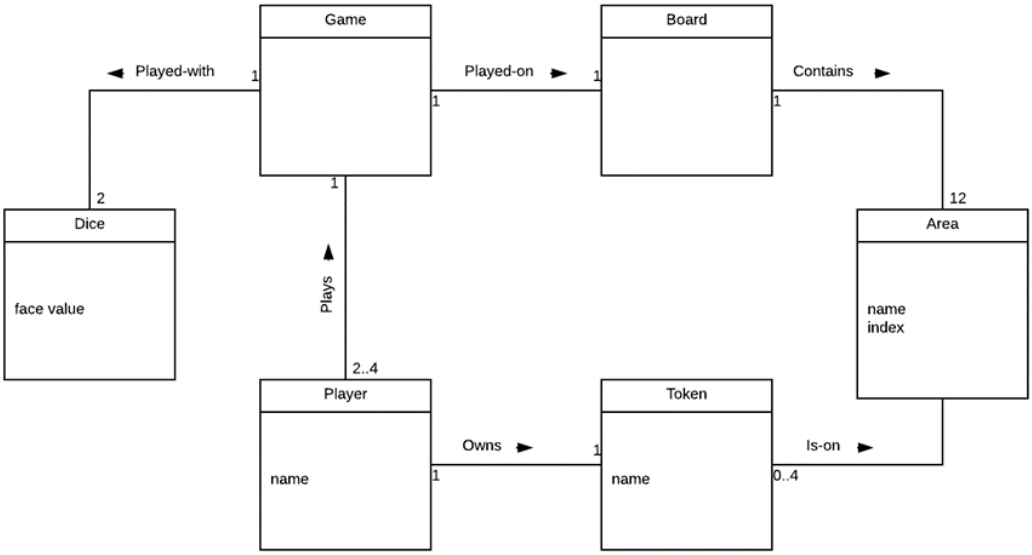


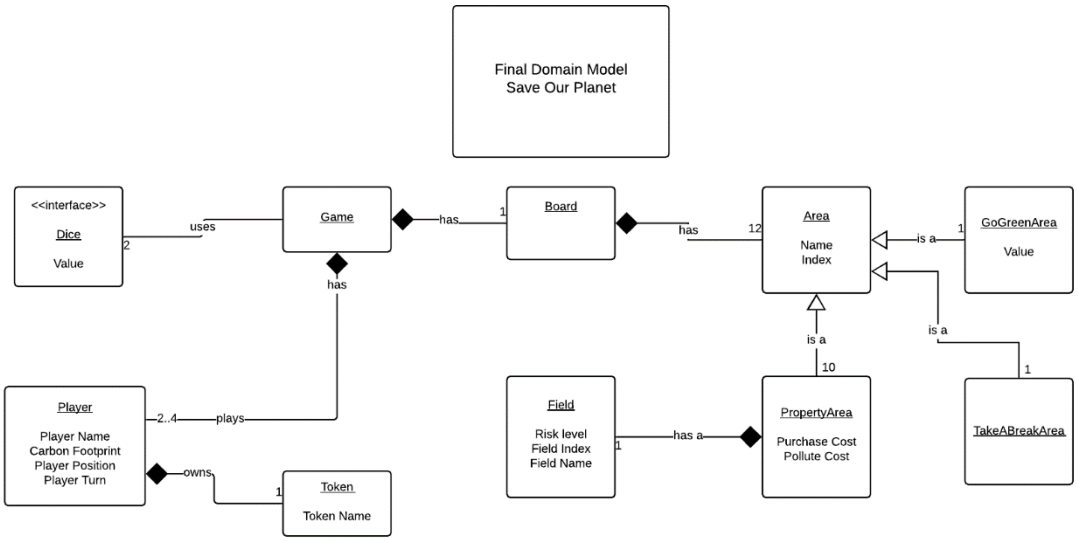
Figure 3.3: Associations, Attributes & Multiplicity



Our Final Domain Model, detailed in Figure 3.4, details the required attributes for each class, the cardinality is detailed to better describe the associations/relationships between each class, i.e. Token is a candidate for composition of Player, Player is a candidate for composition of Game and Area is a candidate for composition of Board.

We realised that the Area classes shared the same attributes (i.e. the Area Name and Area Index) therefore, we concluded that inheritance would be a better method to model this aspect of the game which would allow us to collect the different types of areas in one container (i.e. ArrayList <Area>), in turn allowing us to create the Board.

Figure 3.4: Final Domain Model



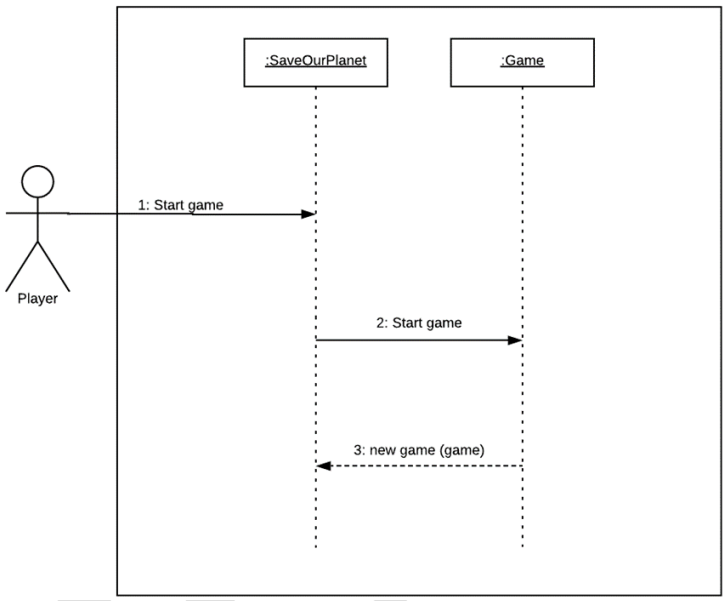
3.3 Sequencing and Implementation

The full cohort of sequence diagrams relating to the following section can be found in Section 8.7 of Appendix 4. Development of this aspect took place throughout 3 iterations in line with the development sprints.

3.3.1 Iteration 1

The diagram below describes how the Player actor initialises gameplay by interacting with the Eclipse system and running the main class SaveOurPlanet within the package. The main class communicates with the Game class by sending a start game message, and the Game class will reply with a message instructing the main class to construct a new game object.

Figure 3.5: Start Game Sequence Diagram



Changes

During implementation, we changed Token to an attribute of Player as the token only represents where the player is on the board. The Player knows their location and where they are on the board and therefore only uses this option. A player uses a token in order to be represented on the board.

Due to inexperienced programmers, during implementation, we underestimated the attributes needed for each Class to build a robust game. This included;

- Boundary values;
- Token names;
- How to store game values (i.e. tokens and players of the game);
- Inputs (player interaction and where these belong);

We realised that an instance of the Board class could not exist without a collection of Area objects being instantiated, likewise, an area cannot exist without a board; therefore Area is a composition/attribute of a Board type object.

With Token no longer storing player location, *getLocation()* + *setLocation()* are now redundant. We discovered player would have a playerPosition (they would know what area they are on) & playerTurn (would assign the order of play to the players in the ArrayList as an array list is dynamic, and positions aren't fixed).

We also realised the value *rollDice()* returns could be used in methods implemented in the Game class. The *rollDice()* method is used in every turn; this make sense to implement the Dice Class as an Interface.

Naming conventions amended to reflect use cases;

- Token needed a name attribute *(String name;)*;
- Player Class became GamePlayer *(due to a clash in naming conventions)* - when the Sound Effects Interface was introduced, the supporting jar library use a class named Player to play the sound effects included in the game);
- Token Class became PlayerToken (due to a clash in naming conventions).

Discoveries

When reviewing our use cases, during the implementation, it was clear that each of our use cases was a method.

- To make the game easier to develop, we broke each use case into their own method[2].
- With repeating functionality, we created separate methods for these[3];

To enhance gameplay, we incorporated print banners for startGame, endGame, gameStats & gameResults which could be called in other methods when needed.

Incorporated *rollDiceSound()*, *movePieceSound()* & *resultsSound()* when game ends to make game more engaging. Due to conflicting naming conventions with supporting libraries, Player class needed to change to GamePlayer & Token to PlayerToken for clarity. We noticed that some of the multiplicities between classes could be represented as final values in each class.

An instance of the Game class requires a service from an instantiation of the Board class, the game needs a board in order for players to move around, purchase areas, pay pollute costs, develop structures, etc. We can say that the game cannot function without the inclusion of a board, and likewise a board has no function without the existence of the game. Therefore, the association between the Game and Board classes can be described as a composition. The Game class owns the Board class which suggests that as the Game object is destroyed; the garbage collector will remove both the Game and Board objects from the heap memory.

[2] *To modularise these components*
[3] *Make it easier to test and debug*

The Player class requires a service that has been defined as aggregation. That is to say, an object of type Player can live independently from the associated object of type Token (i.e. a player can select a different token) and similarly, an object of type Token can exist independently from the associated object of type Player (i.e. a token can be assigned to a different player). As the Player object is destroyed, the garbage collector uses the object reference to locate and remove the Player object from the heap memory and hence the reference to the Token object contained as a piece of data within the Player object. However but another reference to the Token object will exist independently ensuring the Token object is not removed from memory.

3.3.2 Sprint 2

The following process considers the data and functionality modelled during the second iteration of gameplay and investigates the Pay Pollute Cost and Purchase Area use cases with the objective of discovering the data and behaviour needed to allow a player to purchase an area they have landed on or pay a pollution cost to another player who owns the area landed on.

- The simulation involves 2-4 players rolling the dice and advancing their position around the board the number of areas corresponding to the starting position and the value of the dice roll;
- Each player receives 4500 carbon footprint at the start of the game;
- Each player has 600 carbon footprint reduced on every occasion they pass or arrive on the Go Green area;
- If a player quits, the game ends;
- If a player's carbon footprint value is equal to or less than zero the game ends;
- If a player's carbon footprint value is equal to or more than the maximum limit the game ends;
- If the area is owned by the player that arrived on it, nothing happens;
- If the area is owned by another player other than the player that landed on it, the owner has the carbon footprint value reduced and the player the landed on the area has their carbon footprint value increased;
- If the area landed on is not owned by another player the player landing on the area has the option to purchase the area.

Changes

None

Discoveries

Inheritance is possible since a property area **is-a** type of area, go green **is-a** type of area and take-a-break area **is-a** type of area, and there is a duplication of field data (i.e. areaName and areaIndex) and methods (i.e. setName, getName, etc).

The common data and behaviour of these three area classes have been generalised into a single class (i.e. Area). PropertyArea, GoGreen and TakeABreak classes are to be specialised forms of the parent class Area

while satisfying the specifications of the Area class in all respects. There is no need to duplicate the common data and behaviour.

- Three different types of areas (need to collect all types of areas in one container: polymorphism allows a PropertyArea, TakeABreakArea and GoGreenArea to take the form of their parent class Area).

Discovering attributes & methods for PropertyArea Class:

- The attribute name will be inherited from the Area class;
- The methods setName(name) and getName() will be inherited from the Area class;
- The attribute index will be inherited from the Area class. An index attribute is needed when constructing a property area so an area's position in the board layout can be assigned (i.e. index for a property area can be 1, 2, 3, 4, 5, 7, 8, 9, 10 or 11). When the Board constructor is invoked, a loop will begin to iterate and create the game board. The index of each iteration can be passed as a parameter when creating an area, in order to control the position of the area in the board layout;
- The methods setIndex(index) and getIndex() will be inherited from the Area class;
- Discovered two types of functionality when a player lands on a property area:

Functionality	
Purchase an area	• The method setAreaCost(purchaseCost) will assign a cost for purchasing the area when invoked, the purchase cost can be passed as a parameter • The method getAreaCost() will return the cost of purchasing the area • The method setIsOwned(isOwned) will assign ownership of the area to the player, the number associated with the player's turn can be passed as a parameter • The method getIsOwned() will return the number of the player who owns the area when invoked
Pay a pollution cost	• The method setPolluteCost(polluteCost) will assign a pollution cost for the area when invoked, the pollute cost can be passed as a parameter • The method getPolluteCost() will return the pollute cost for the area

- Discovered needed to check players carbon Footprint against MIN & MAX carbon limits to determine whether or not the game needs to end.
- Discovered attributes for GoGreen Class:
  - index = 0: This attribute will be inherited from the Area class, and the value is used to assign the go green area's position in the board layout.
- Discovered attributes for TakeABreak Class:
  - index = 6: This attribute will be inherited from the Area class and the value is used to assign the take-a-break area's position in the board layout
- Discovered final attributes for Board Class:
  - LOOP_LIMIT = 12: This attribute will control the number of iterations needed to create the game board; the game board currently consists of 12 areas. This value can be easily changed for future growth.

- Discovered final attributes for PropertyArea Class:
  - Property area names: FINLAND = 'Finland, SWEDEN = 'Sweden', UK = 'UK', GERMANY = 'Germany', RUSSIA = 'Russia', JAPAN = 'Japan', INDIA = 'India', CHINA = 'China', CANADA = 'Canada', USA = 'USA'
  - DEFAULT_OWNER_VALUE = 0: This final attribute will be passed as a parameter in the setIsOwned to initialise the ownership of the property area to a default value. The value of isOwned is updated to the player's turn value of the area is purchased.
  - PURCHASE_COST = 100: This final attribute will be passed as a parameter in the setIsOwned method to initialise the cost of purchasing the area.
  - POLLUTE_COST= 30: This final attribute will be passed as a parameter in the setPolluteCost method to initialise the cost of landing on this and paying a pollution cost.
- Discovered final attributes for GoGreenArea and Game Classes:
  - MAX_CARBON_LIMIT = 9000
  - MIN_CARBON_LIMIT = 0
  - END_GAME = 2
  - GO_GREEN_VALUE = 0

As previously discovered with the association linking the Game and Board classes, an instance of the Game class requires a service from an instantiation of the GamePlayer class; the game needs at least two associated GamePlayer instances for the game to have a logical existence. We can say that the game cannot function without the inclusion of players, and likewise a player(s) has no logical function without the existence of the game. Therefore, the association between the Game and GamePlayer classes can be described as a composition. The Game class owns the GamePlayer class, which suggests that as the Game object is destroyed, the garbage collector will remove both the Game and GamePlayer objects from the heap memory. A Game object has-a GamePlayer object.

As the game must have 2-4 players, an array list can be utilised to store the collection of players. In terms of future growth for the system, if game requirements changed, it would be easy to manipulate the data model and code in order to increase or decrease the number of players the game involves.

We know that property area is-a type of area, go green is-a type of area and take-a-break area is-a type of area. Since we also noticed the duplication of field data (i.e. areaName and areaIndex) and methods (i.e. setName, getName, etc), the common data and behaviour of these three area classes have been nominated to be generalised into a single class (i.e. Area). PropertyArea, GoGreen and TakeABreak classes are defined as subclasses that can inherit the common information from the superclass Area. This inheritance/subtyping will allow PropertyArea, GoGreen and TakeABreak classes to be specialised forms of the parent class Area while satisfying the specifications of the Area class in all respects.

There is no need to duplicate the common data and behaviour when the three subclasses inherit from the superclass it can be assumed that the data and behaviour they inherit shall be the same in all classes.

The game board consists of 12 areas, suggesting that the Board class requires service from the Area class. Since an instantiation of the Board class without the inclusion of areas would serve no logical purpose, and instances of the PropertArea, GoGreen and TakeABreak classes must be collected and arranged in a particular order to represent the game board the association between the Board and Area classes is a composition. The Board class owns the Area class.

Polymorphism (as the three subclasses each pass at least two IS-A tests we can have polymorphism, e.g. a property area is-a property area and a property area is-a area) provides the ability for the subclasses to take the form of the superclass when needed. As mentioned above, polymorphism gives the capability to collect a go green object, a take-a-break object and the 10 property area objects that comprise the game board. These objects can take the form of their parent class Area when representing the game board layout, and each object can be accessed on an individual basis and type-casted to a property area, go green area or take-a-break area so that their specialised behaviour can be obtained.

For extensibility and maintainability reasons, the Board object can be defined as an array list that collects the board areas within a single container in a particular order; therefore a single variable can store multiple data types. In the event that game rules change to include additional areas or remove areas, the altercation to the data model and code would require minimal effort. Polymorphism will also allow the ability to add areas of different types (i.e. go to jail area as featured in the Monopoly game) in future versions of the Save Our Planet game.

This would demonstrate good design with regard to extensibility as new functionality can be added by inheriting new data types from the parent class. The methods and data stored in the parent class will not need to be altered to accommodate the new classes.

3.3.3 Sprint 3

The third iteration of gameplay considers the data and behaviour modelled during the previous gameplay iterations and aims to understand the data and operations needed to permit a player to own a field(s) and develop minor/major structures.

- The player must own all the areas in a field before they can develop structures on the areas within the field
- The value of the isOwned attribute can be used to check if the player owns an area
- The player can only develop three minor structures on an area
- The player can only develop one major structure on an area
- The area must contain three minor structures before a major structure can be created
- The game must check how many fields the player owns
- If the player owns one or more fields the game must prompt the player to develop the field(s) at the beginning of the player's turn
- If the player decides not to the develop any structures the player must continue their turn by rolling the dice

- Given, the player has fully developed the areas within a field, the user must not be prompted to develop any areas within that field
- One of the fields is the most costly to develop structures on its areas, another field is the least costly to develop structures on its areas, and the remaining two fields are the same cost to develop structures on their areas

Discoveries:

- Field is a composition of PropertyArea;
- Field could have been an aggregation of Area class, but given the progress of the game implementation this would have resulted in the unnecessary restructuring of the program, although this may have better reflected the problem space;
- When iterating through the areas container in Board to discover who owned an area, we discovered that unless the areas container was cast as the PropertyArea Class, we were not able to access the methods relating to a specific property area to retrieve the relevant information about each property area (i.e. getIsOwned, getAreaCost etc);
- Discovered major and minor development was an attribute development count in property Area;
- Discovered assigning playerTurn (GamePlayer Attribute) value to isOwned (PropertyArea attribute) value allowed us to determine ownership of a property area and fields;
- Discovered we needed to create a count of the areas owned by each player during their turn, and match them against the total areas in each field. If the count of owned areas of a particular field matched the field size, then the player owned that field and a menu would appear asking them if they wish to develop (this would only appear if they owned an entire field or multiple fields). We used the field index to add to an ArrayList (fieldList <<Integer>> in Game Class), so we could loop through the fields owned by that player. We opted to use an ArrayList container, as this is dynamic and can grow as a player acquires fields.

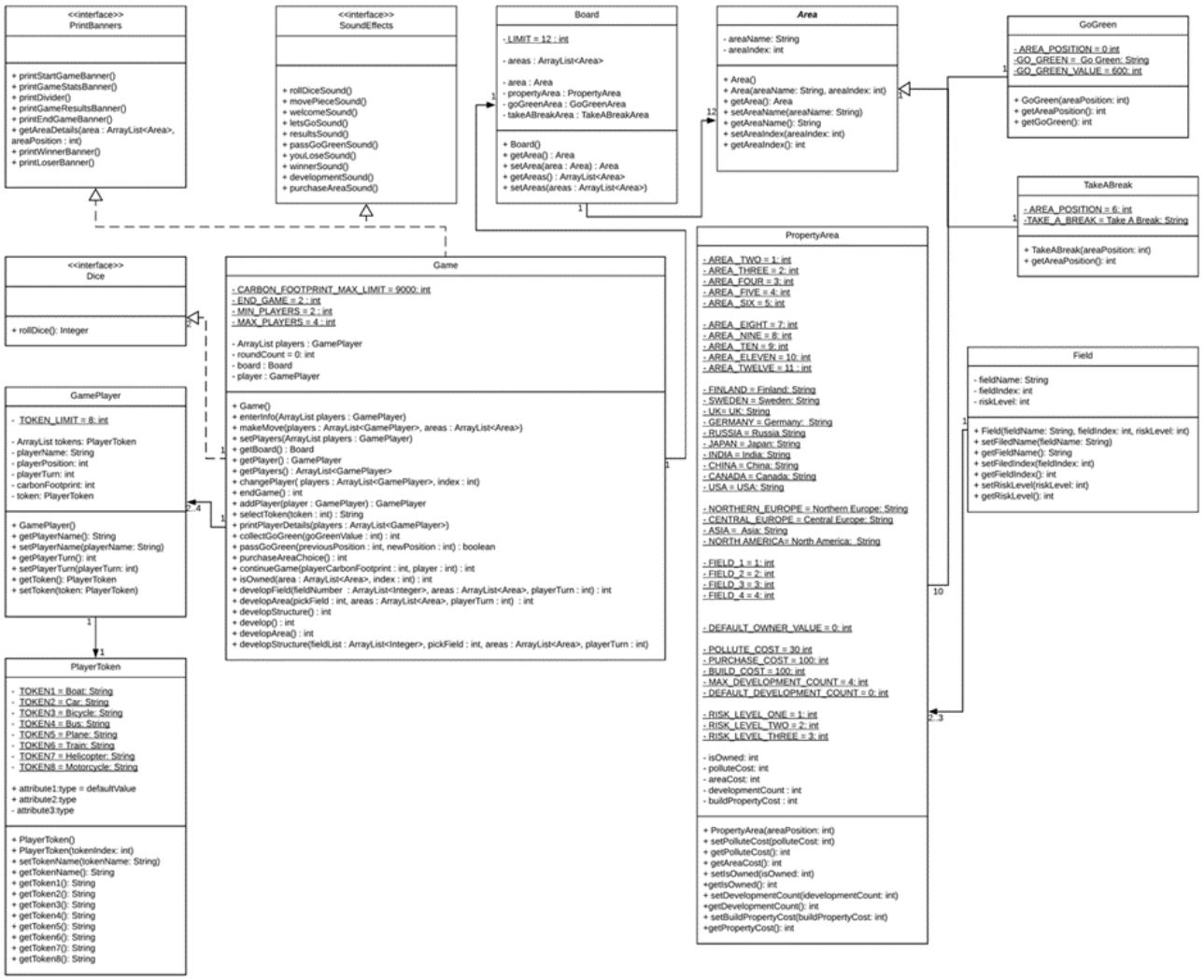
4 DESIGN

4.1 Introduction

This section displays the final implemented UML design for our 'Save our Planet' game. It is the culmination of each of the analysis and design steps covered in Sections 2 & 3, and the discoveries that emerged and changes required over the course of our 3 Sprint implementation.

4.2 UML Class Diagram

Figure 4.1: FINAL Save Our Planet UML Class Diagram



4.3 Maintainability & Extensibility

Considerations around maintainability and extensibility of the program were a core tenement to the game's design through careful adherence to the primary principles of Object Orientated Programming. Below we present aspects of the good design within the game in this respect.

Maintainability

We opted to use Collections/ArrayLists to store game data due to the supporting libraries and methods that are associated with these. We considered the impact this may have on the performance of the game (as these collections allocate more space in memory compared to other containers) but the pros outweighed the cons, and this cemented our decision.

Use of **Method Overloading** allowed for flexibility in calling similar methods which perform similar actions but with different data, allowing us to save memory space. This has also allowed for consistency and efficiency in the readability of the code, e.g. the *developArea()* method. The use of **Composition** has allowed us to define 'has-a' relationships between classes, allowing for the use of inheritance or composition for reusing code.

The use of **Polymorphism** has allowed us to gather different types of areas into one collection. From this it allowed us to separate objects when needed so to access their behaviours on an individual basis, e.g. Area Collection: (Areas ArrayList) of different types of areas (PropertyArea, GoGreenArea, & TakeABreakArea).

In addition to this, normalising out our methods allows them to be easily reused and quickly fixed. Setting boundary values as FINAL parameters allows for a single point of change that can then be reflected throughout the game. The use of the Comparator import allowed us to easily create a temporary leaderboard ArrayList, without impacting the order of play, whilst still reflecting the correct ranking of each player.

Extensibility

The Game class acts as an overall controller for communication between all classes involved in the game and by doing so it dictates the sequence of behaviours. As such the behaviour of the game can be easily changed through modification of the Game class, e.g. adding additional methods, editing *makeMove()* method etc.

Other examples of where the principles of OOP have allowed for greater extensibility include:

- Dynamic collection of board, token and players in an ArrayList allow for it to be extended easily;
- Utilising interfaces for print banners, sound effects and rollDice enable the developer to amend these aspects without affecting gameplay functionality;
- Changes that would be required to all Areas can be done in one class, abstract Area class through inheritance. This allowed for code reuse and for minimising redundancy.

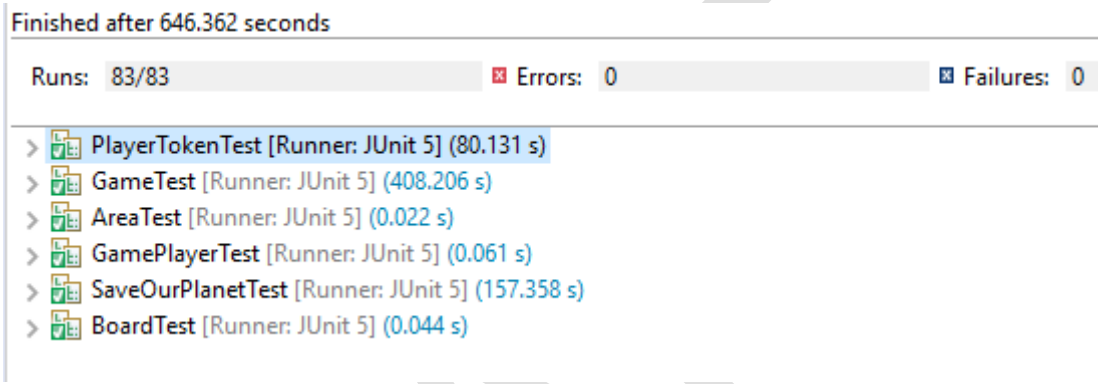
5 APPENDIX I – TESTING

5.1 Test Plan

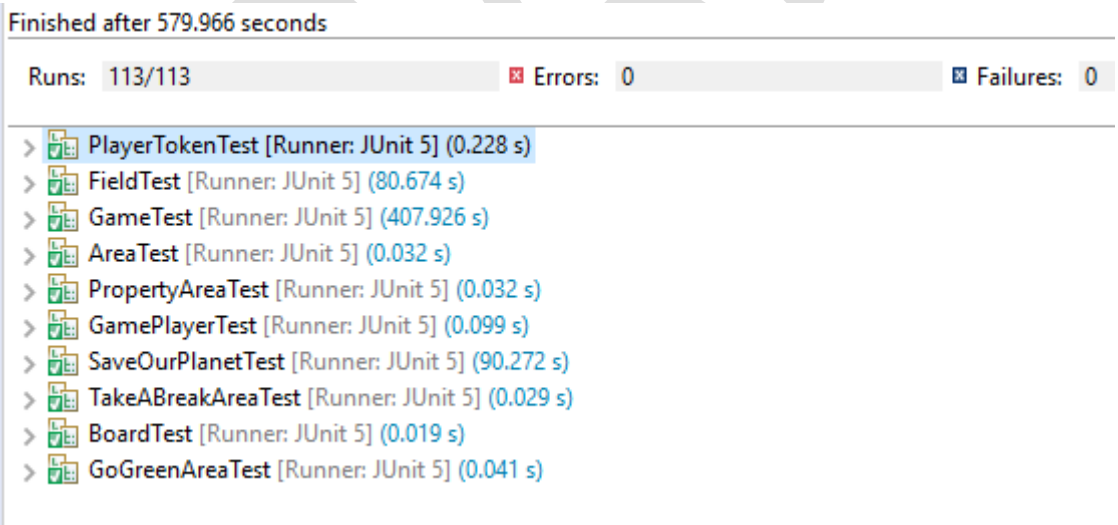
The full test plan for our project is contained as an external appendix file, located in the Appendix 1 folder accompanying this report.

5.2 Unit Testing

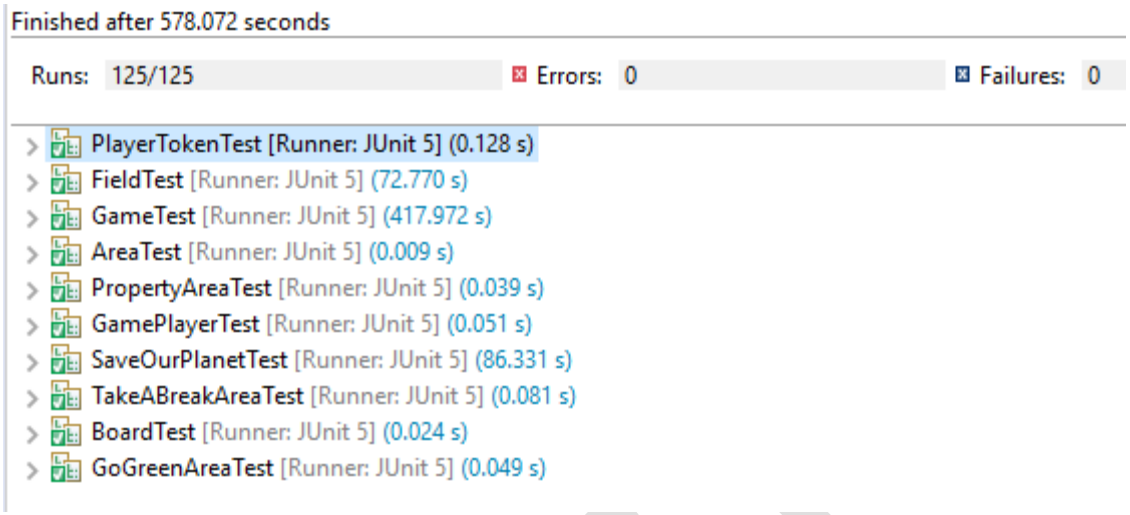
5.2.1 jUnit Tests Iteration 1



5.2.2 jUnit Tests Iteration 2



5.2.3 jUnit Tests Iteration 3



The full suite of jUnit tests, demonstrated as working in the preceding screeschots, can also be found in the Appendix 1 folder accompanying this report.

6 APPENDIX II – MINUTES

From the outset the team sought to maximise the use of 'in person' meetings within the EEECS campus building. From the beginning of March, pressures from other delivery deadlines and the onset of the COVID-19 restrictions on personal movement saw team meetings migrate permanently to the MS Teams platform.

Meeting 1 – Requirements Elicitation

Title : Requirements Elicitation
Date & Location: 15/01/2020Meeting Via Teams
Attendees: • Glenn McGookin; • Anthony McGarrigan; • Anthony Consadine; • Niall Murphy; • Matt Walsh;
Apologies: N/A
Absent: N/A
Agenda: • Review requirements of the game that the client has provided; • Agree on themes based on 'caring for the planet'; • Develop rules that meet the client requirements based around these themes.
Summary of Discussion: • Team agreed upon initial action points; • Set a timeframe for reviewing requirements; • Discussed themes, agreed more time required to develop themes of the game.
Action Points: • Set a date for follow up meeting to discuss and review action points; • Agreed each team member to develop a set of rules to be reviewed at the next meeting; • Each member to develop a 'theme' for the game and review, decide upon one to progress with; • Once rules and 'theme' has been agreed, we can consider developing system use cases.

Meeting 2 – Requirements Elicitation Review & Use Case Development

Title : Requirements Elicitation Review & Use Case Development
Date & Location: 22/01/2020 – EEECS Building
Attendees: • Glenn McGookin;

- Anthony McGarrigan;
- Anthony Consadine;
- Niall Murphy;
- Matt Walsh;

Apologies: N/A

Absent: N/A

Agenda:

- Review the game rules and themes developed;
- Decide on next action points,

Summary of Discussion:

- Team discussed game rules and functionality;
- Agreed core functionality has been met and rules satisfied;
- Team agreed upon the theme created.
- Agreed on an agile (iterative) approach to develop use case prototypes. Consisting of three iterations of the game;
  - Iteration 1: simple functionality (player moves & quits);
  - Iteration 2: player acquires an area and pays a resource for acquiring the area and pays a pollute cost if a player lands on an area that is already owned by another player, players carbon is reduced when they pass Go Green; Carbon Footprint will need introduced and elaborated upon. Players can win or lose;
  - Iteration 3: players can develop an area on a field that is owned.

Action Points:

- Development of use cases and use case diagrams for each iteration of game play.
- Realise & finalise use cases, core requirements, rules & objectives. Discuss and agree if these have been satisfied in next week's meeting.
- Candidate object discovery can proceed to determine our candidate classes to allow us to plan initial domain model once use cases have been finalised.

Meeting 3 – Use Case Modelling Review & Domain Modelling Development

Title : Use Case Modelling Review & Domain Modelling Development

Date & Location: 29/01/2020 EEECS Building

Attendees:

- Glenn McGookin;
- Anthony McGarrigan;
- Anthony Consadine;
- Niall Murphy;
- Matt Walsh;

Apologies: N/A

Absent: N/A
Agenda: • Review Use Cases and use case diagrams; • Determine if we can proceed to Domain Modelling Stage.
Summary of Discussion: • Discussed use cases and use case diagrams. Team happy with use cases and feel all core functionality from user requirements has been satisfied; • Team feel we can proceed to the next stage of development; • Discussed AOR's for each development stage.
Action Points: • Candidate Object Class Discovery to be carried out to realise initial Domain Model – (Matt W & Anthony McG to investigate) • Discuss and agree upon classes discovered for initial domain model in next week's meeting.

Meeting 4 – Domain Modelling Review & Sequence Diagram/Test Plan Development

Title : Domain Modelling Review & Sequence Diagram/Test Plan Development
Date & Location: 06/02/2020 EEECS Building
Attendees: • Glenn McGookin; • Anthony McGarrigan; • Anthony Consadine; • Niall Murphy; • Matt Walsh;
Apologies: N/A
Absent: N/A
Agenda: • Review & Agree Initial Domain Model; • Discuss next steps in development life cycle;
Summary of Discussion: • Team agreed with initial candidate classes discovered from the Candidate Object Class Discovery Exercise. Initial Classes of; - Game; - Board; - Player; - Area; - Token; and - Dice.

- Team discussed sequence diagrams related to the first iteration of game play which is aimed at the functionality needed for players to start a game, move round the board and quit.
- Team agreed upon a test plan that will reflect each step of the game development based upon the use cases and requirements for iteration 1.

Action Points:

- Team to develop initial sequence diagrams relating to the first iteration of the game. (UC1: Enter Info, UC2: Display Error Message, UC3: Make Move, UC4: Display Info, UC5: Change Player, UC6: Roll Dice & UC14: End Game). – Matt W & Anthony McG to develop.
- Test plan to be designed that covers functionality discovered in first iteration of game play. – Glenn McG to develop.
- Review developed Sequence Diagrams & Test Plans in next meeting.
- Discuss and finalise Project Gantt Chart – Niall to develop.
- Collate project tasks to start to develop backlog items.

Meeting 5 – Iteration 1: Sequence Diagram & Test Plan Review

Title : Iteration 1: Sequence Diagram & Test Plan Review

Date & Location: 13/02/2020 EEECS Building

Attendees:

- Glenn McGookin;
- Anthony McGarrigan;
- Anthony Consadine;
- Niall Murphy;
- Matt Walsh;

Apologies: N/A

Absent: N/A

Agenda:

- Review developed sequence diagrams for iteration 1;
- Review Test Plan;
- Discuss Gantt chart timeline;
- Review backlog items and assign responsibilities;

Summary of Discussion:

- Team discussed sequence diagrams and domain model and agreed that these reflected the initial Use Case Diagram and Use Cases and satisfy functionality to develop first iteration of gameplay (updated Domain Model);
- Team agreed upon developed test plan;
- Team agreed with the timeline defined in Gantt chart;
- Discussed backlog items and assigned these to team members.

Action Points:

- Development of first iteration of game play. – Antony McG & Matt W to develop (based off Anthony C's initial Player Class);
- Plan JUnit test that correspond with first iteration of gameplay. – Anthony C to develop;

- Plan acceptance tests to be conducted after implementation of first iteration of Game Play. – Glenn to conduct;
- Aim to meet and discuss progress of iteration 1 next week.

Meeting 6 – Iteration 1: Test Plan Implementation, Iteration 2: Sequence Diagram Development

Title : Iteration 1: Test Plan Implementation, Iteration 2: Sequence Diagram Development

Date & Location: 20/02/2020

Attendees:

- Glenn McGookin;
- Anthony McGarrigan;
- Anthony Consadine;
- Niall Murphy;
- Matt Walsh;

Apologies: N/A

Absent: N/A

Agenda:

- Review implementation of code for first iteration;
- Discuss Acceptance testing & Junit testing;
- Plan activities for iteration 2.

Summary of Discussion:

- Matt W & Anthony McG completed code for first iteration of game play which cover the first 7 use cases. The requirements for each use case were considered in the implementation of the code.
- A quick demo of the working code with alternative flows considered was performed, and team agreed that code for iteration one had been satisfied and was ready for testing.
- Team discussed formal testing;
    - Acceptance testing to adhere to the requirements of use cases included in iteration 1
    - JUnit test suites to reflect iteration 1 in terms of testing the reliability of each Class, with the use of BVA & Equivalence Partitioning (creation of objects and their data and behaviour).

Action Points:

- Acceptance testing to be performed on first implementation of code – Glenn McG to complete.
- JUnit test suites to be developed to test each Classes methods and attributes. – Anthony C to develop.
- Development of sequence diagrams for 2nd iteration (UC7: Pay Pollute Cost, UC8: Go Green, UC9: Purchase Area, UC10: Take A Break) to aid in development of Domain Model and UML Class Diagram to support the second implementation of code. – Anthony McG & Matt W to complete.
- Update the project report with current progress. – Niall to complete.
- Review Meeting on progress of this week's action points next week.

Meeting 7 – Iteration 1: Test Plan Review, Iteration 2: Sequence Diagram Review

Title : Iteration 1: Test Plan Review, Iteration 2: Sequence Diagram Review
Date & Location: 27/02/2020 MS Teams
Attendees: • Glenn McGookin; • Anthony McGarrigan; • Anthony Consadine; • Niall Murphy; • Matt Walsh;
Apologies: N/A
Absent: N/A
Agenda: • Cover Acceptance Tests; • Discuss JUnit test suites; • Discuss any defects and bugs discovered; • Discuss progress of sequence diagram activity for iteration two.
Summary of Discussion: • Anthony McG & Matt W explained the sequence diagrams for this iteration are taking longer than anticipated. Aiming to have these completed within another week with an updated Domain Model and UML Class Diagram to allow code development of iteration two. • Team agreed with the addition of a second project, less time would be dedicated to the Save Our Planet project to allow each team member to focus on the second project. Team aware this may put us behind schedule. • Glenn M proposed due to availability constraints the acceptance criteria for each product backlog item, should be reviewed and agreed verbally and acceptance recorded in the test plan document. • Team decided that a formal test period should be implemented once all iterations are complete to address any regression issues and confirm overall compliance • Anthony C confirmed that JUnit tests had been created and executed in line with the plan. These test validation around setting up of players, tokens and navigating around the board along with ending the game
Action Points: • Aim to complete sequence diagrams, Domain Model & updated UML Class Diagram for iteration 2. - Matt W & Anthony McG to continue. • Team to catch up next week to discuss current progress and next steps.

Meeting 8 – Iteration 2: Sequence Diagram Review & Code Implementation

Title : Iteration 2: Sequence Diagram Review & Code Implementation
Date & Location: 5/03/2020 MS Teams
Attendees: • Glenn McGookin; • Anthony McGarrigan;

- Anthony Consadine;
- Niall Murphy;
- Matt Walsh;

Apologies: N/A

Absent: N/A

Agenda:

- Review sequence diagrams;
- Discuss Code Implementation;
- Plan Acceptance Tests & JUnit test suites.

Summary of Discussion:

- Inheritance would allow us to create the Game *board (discovery of GoGreen, TakeABreak and PropertyArea sharing similar attributes (name & index) opened the possibility to incorporate inheritance into the data model)*. It was agreed Inheritance was a suitable way to move forward. This allows us to treat the different area types as the same object (Area) which allows us to collect all the areas in one container that represents the game board.
- Team agreed sequence diagram describing creation of game board was suitable way to move forward.
- Team agreed after reviewing sequence diagrams and design decisions, that second implementation of code could proceed.

Action Points:

- 2nd implementation of code to be completed, incorporating, carbon footprint, purchase area, pay pollute cost, win & lose. – to be completed by Anthony McG & Matt W.
- Plan JUnit test that correspond with 2nd iteration of gameplay. – Anthony C to develop.
- Plan acceptance tests to be conducted after implementation of 2nd iteration of Game Play. – Glenn to conduct.
- Aim to meet and discuss progress of iteration 2 in two weeks' time due to 2nd hand in expected on 16th March for web development.

Meeting 9 – Iteration 2: Test Plan Implementation, Iteration 3: Sequence Diagram Development

Title : Iteration 2: Test Plan Implementation, Iteration 3: Sequence Diagram Development

Date & Location: 18/03/2020 MS Teams

Attendees:

- Glenn McGookin;
- Anthony McGarrigan;
- Anthony Consadine;
- Niall Murphy;
- Matt Walsh;

Apologies: N/A

Absent: N/A

Agenda:

- Review code implementation iteration 2;
- Discuss Acceptance testing & Junit testing;
- Plan activities for iteration 3.

Summary of Discussion:

- Matt W & Anthony McG completed code for second iteration of game player which covered a further 4 use cases.
- A quick demo of the working code showing specifically the additions made (i.e. purchase an area and pay pollute cost) was performed. Team agreed that code for iteration 2 ad been satisfied and was ready for testing.
- A design decision was implemented in regard to the playability of the game, with introduction of sound effects, print banners, scoreboard and area cards (based on style of monopoly card). The team agreed that these features enhanced gameplay and made the game more engaging for end user.
- Team discussed formal testing;
  - Acceptance testing to adhere to the requirements of use cases included in iteration 2
  - JUnit test suites to reflect iteration 2 in terms of testing the reliability of each Class, with the use of BVA & Equivalence Partitioning (creation of objects and their data and behaviour).

Action Points:

- Acceptance testing to be performed on 2nd implementation of code – Glenn McG to complete.
- JUnit test suites to be developed to test each Classes methods and attributes. – Anthony C to develop.
- Development of sequence diagrams for 3rd iteration (UC11: Develop Structure, UC12: Develop Minor Structure, UC13: Develop Major Structure) to aid in development of Domain Model and UML Class Diagram to support the 3rd implementation of code. – Anthony McG & Matt W to complete.
- Update the project report with current progress – Niall to complete.
- Review meeting on progress of this week's action points next week.

Meeting 10 – Iteration 2: Test Plan Review, Iteration 3: Sequence Diagram Review

Title : Iteration 2: Test Plan Review, Iteration 3: Sequence Diagram Review

Date & Location: 25/03/2020 MS Teams

Attendees:

- Glenn McGookin;
- Anthony McGarrigan;
- Anthony Consadine;
- Niall Murphy;
- Matt Walsh;

Apologies: N/A

Absent: N/A

Agenda:

- Cover Acceptance Tests;
- Discuss JUnit test suites;
- Discuss any defects and bugs discovered;

- Discuss progress of sequence diagram activity for iteration three.

Summary of Discussion:

- Anthony McG & Matt W realised that UC11, UC12 & UC13 were actually one use case, and just alternate flows in the same use case (UC11: develop structure). With realisation of development count attribute in PropertyArea field. Discussed with team who agreed having this as an attribute satisfied the requirement for minor & major developments on an area.
- We realised a class Field which allowed us to group areas together for a player to develop areas on a field. We discussed with the team, potentially Area Class could be an aggregation with Field, since an area may or may not be part of a field but agreed to make Field a composition of a PropertyArea since this best suited the layout of the existing code. This decision was made as this resulted in less code restructuring.
- Team agreed that sequence diagrams and realisations met the requirements and could proceed with code implementation of iteration 3.
- Per discussion on 27/02 acceptance testing would be conducted and verbally agreed to close of each product backlog item. Glenn M confirmed that test plan has been authored and pending execution following the completion of iteration 3
- jUnit tests have been written to hit each method within each class In the code. In total these can be run in 10-20 minutes and should be run as a regression test following any updates to the code. For any new code new tests shall be created and some existing tests may need to be updated following any change to the code
- Acceptance criteria to be reviewed and agreed verbally once all iterations are complete to address any regression issues and confirm compliance.

Action Points:

- 3rd iteration of code to be completed, incorporating developing an area in a field, and deciphering how to determine if a field is owned. – to be completed by Matt W & Anthony McG
- Plan JUnit test that correspond with 3rd iteration of gameplay. – Anthony C to develop.
- Plan acceptance tests to be conducted after implementation of 3rd iteration of Game Play. – Glenn to conduct.
- Aim to meet and discuss progress of iteration 3 next week.

Meeting 11 – Iteration 3: Code Implementation Review

Title : Iteration 3: Code Implementation Review

Date & Location: 02/04/2020 MS Teams

Attendees:

- Glenn McGookin;
- Anthony McGarrigan;
- Anthony Consadine;
- Niall Murphy;
- Matt Walsh;

Apologies: N/A

Absent: N/A

Agenda:

- Discuss code implementation progress.

Summary of Discussion:

- It was realised that the action to develop a structure was the amalgamation of multiple actions which needed to be stepped through action by action;
  - Check if a user owned an area;
  - Check if these areas belonged to a field;
  - Check existing structures on each area (check structure count for minor/major/unable to develop);
  - Allow a player to pick another area to develop if area was already fully developed;
  - Check players carbon footprint to make sure development cost didn't cause their carbon footprint to surpass MAX CARBON LIMIT.
- With all these actions to incorporate, we tried to keep our design approach in line with the theme and approach that had been taken throughout each iteration (i.e. using index (int) values as player inputs/choices always starting from 1).
- Matt W & Anthony McG completed 3rd iteration of code. Provided a demo showing all the behaviours from each iteration, specifically developing a structure on an area. The team agreed the design and layout of the develop structure menus were user friendly (usability) and met the core requirements as stated in the requirements documentation.

Action Points:

- Acceptance testing to be performed on 2nd implementation of code – Glenn McG to complete.
- JUnit test suites to be developed to test each Classes methods and attributes. – Anthony C to develop.
- Update the project report with current progress – Niall to complete.
- Review meeting on progress of this week's action points next week and project progress.

Meeting 12 – Iteration 3: Test Plan Review

Title : Iteration 3: Test Plan Review

Date & Location: 08/04/2020 MS Teams

Attendees:

- Glenn McGookin;
- Anthony McGarrigan;
- Anthony Consadine;
- Niall Murphy;
- Matt Walsh;

Apologies: N/A

Absent: N/A

Agenda:

- Discuss Acceptance test results;
- Discuss JUnit test suites;
- Discuss any major bug or defects discovered;
- Discuss readiness to finalise project.

Summary of Discussion: • Glenn M covered the acceptance criteria with the meeting and updated the test plan document, with confirmation of acceptance criteria. • jUnit tests have been tidied up to remove duplication and also remove tests that were setup to test methods that were removed in the final iteration as they were unnecessary. A complete jUnit test suite can be run in 10-20 minutes that shall test each method within the code. Final number of tests is 125 and all are currently passing • Team agreed that with the successful completing of acceptance test and JUnit test suites that game satisfies the project requirements and is ready to release. Glenn M indicated that before the next meeting he would execute the formal test plan to identify any regression issues, conformance to requirements and end to end game play. If required, an adhoc meeting can be setup to discuss any defects or issues. But items identified during this period, will be addressed in future sprints. • Discussed final stages of the documentation and presentations and finalised date to complete these.
Action Points: • Arrange a time to meet to discuss & finalise report and record presentation video discussing our development process. Team agreed to meet via Teams 16/04/2020 to conduct this.

Meeting 13 – Project Finalisation and Scheduling of Release

Title : Project Finalisation and Scheduling of Release
Date & Location: 16/04/2020 MS Teams
Attendees: • Glenn McGookin; • Anthony McGarrigan; • Anthony Consadine; • Niall Murphy; • Matt Walsh;
Apologies: N/A
Absent: N/A
Agenda: • Discuss steps necessary for project finalisation and release; • Discuss any issues identified over the course of testing; • Review the video of gameplay produced for submission; • Discuss approach to video presentation to accompany project and report submission.
Summary of Discussion: • When reviewing the test plan and its results, no significant defects were identified in either the requirements, unit or acceptance tests. Some small issues did come up: - A number of bugs, relating mostly to Null Pointer Exceptions, were present in initial unit tests, but had cleared up in the later iterations of game development;

- Time delays caused by the sound effects were identified as a minor inconvenience during acceptance testing. However this was mostly due to the frequency and pace at which the testers were seeking to play the game at in order to complete testing in a timely manner;
- The most pertinent issue remaining at this point was the fact there was no limit placed on the number of characters for a player's name. Long names caused presentation issues with some of the game's text graphics. Anthony G committed to having this corrected.

- The group commended Glenn for a well presented a gameplay video. This aspect of the project submission was now considered completed.
- Attention turned to the second video to be submitted and how to approach this. After lengthy discussion it was agreed that the project implementation should be split into 5 sections, assigned to each team member with 1 minute to talk through their segment.

Action Points:

- The group agreed to develop a small script for their assigned segment, screen record this and send to Glenn to be patched together.
- The group agreed to reconvene on Sat 18th @ 1pm in order to finalise the submission and for project sign off.

Meeting 14 – Finalisation and Project Close

Title : Finalisation and Project Close

Date & Location: 18/04/2020 MS Teams

Attendees:

- Glenn McGookin;
- Anthony McGarrigan;
- Anthony Consadine;
- Niall Murphy;
- Matt Walsh;

Apologies: N/A

Absent: N/A

Agenda:

- Recording of team presentation video.
- Discussion of group peer review, and agreement on team member scoring.
- Final requirements for project submission / release.

Summary of Discussion:

- After a number of attempts, the group succeeded in recording a presentation of the project delivery process within the allotted time.
- The group discussed each other's contribution to project delivery. It was agreed that Matthew Walsh and Anthony McGarrigan's efforts should come in for particular commendation, and the subsequent peer review scores reflected this.
- The group discussed all final aspects that needed to be tidied before submission. The group agreed they were in a position to release the game and submit the project within the coming week.

Action Points:

- Final report draft to be circulated to group for review, amendment and approval for submission.
- Submission of the group project and completion of all group activity.

7 APPENDIX III – FILE SHARING & VERSION CONTROL

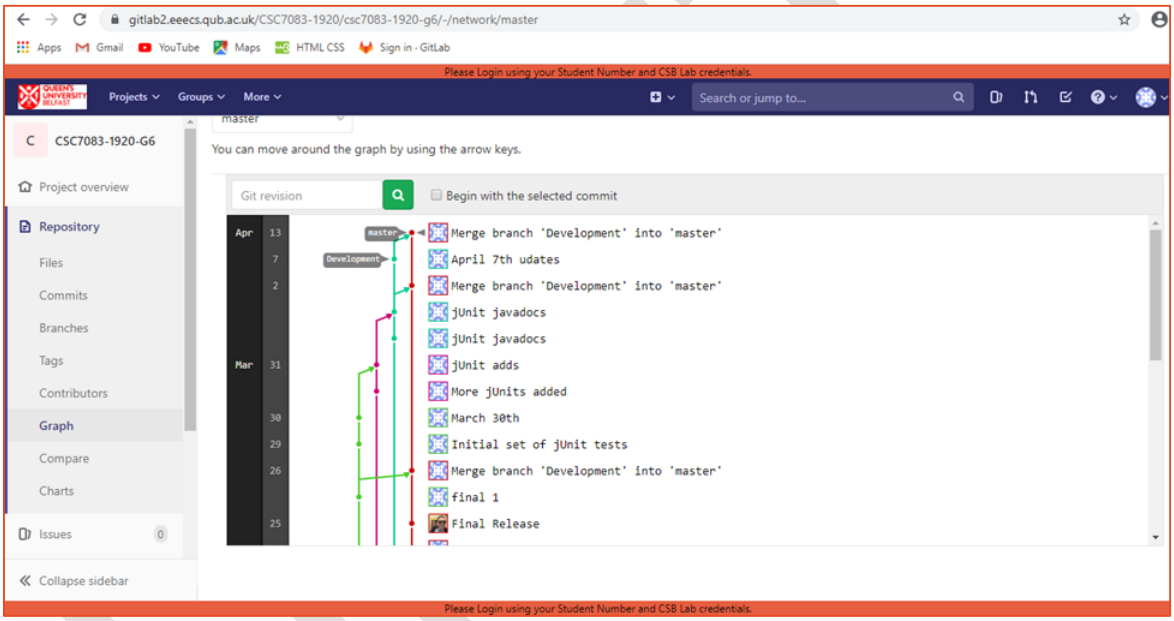
This section demonstrates the sharing and dissemination of files between the team via the MS Teams platform, and the use of GitLab for program version control.

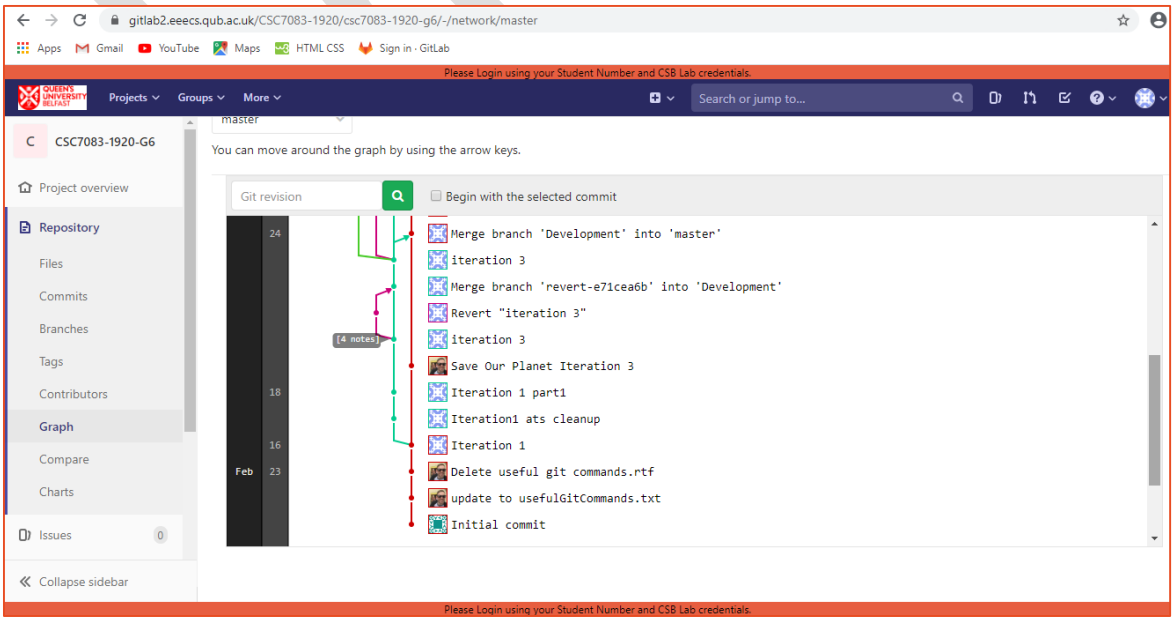
7.1 GitLab

Our Gitlab project can be viewed at the following location:

https://gitlab2.eeecs.qub.ac.uk/CSC7083-1920/csc7083-1920-g6

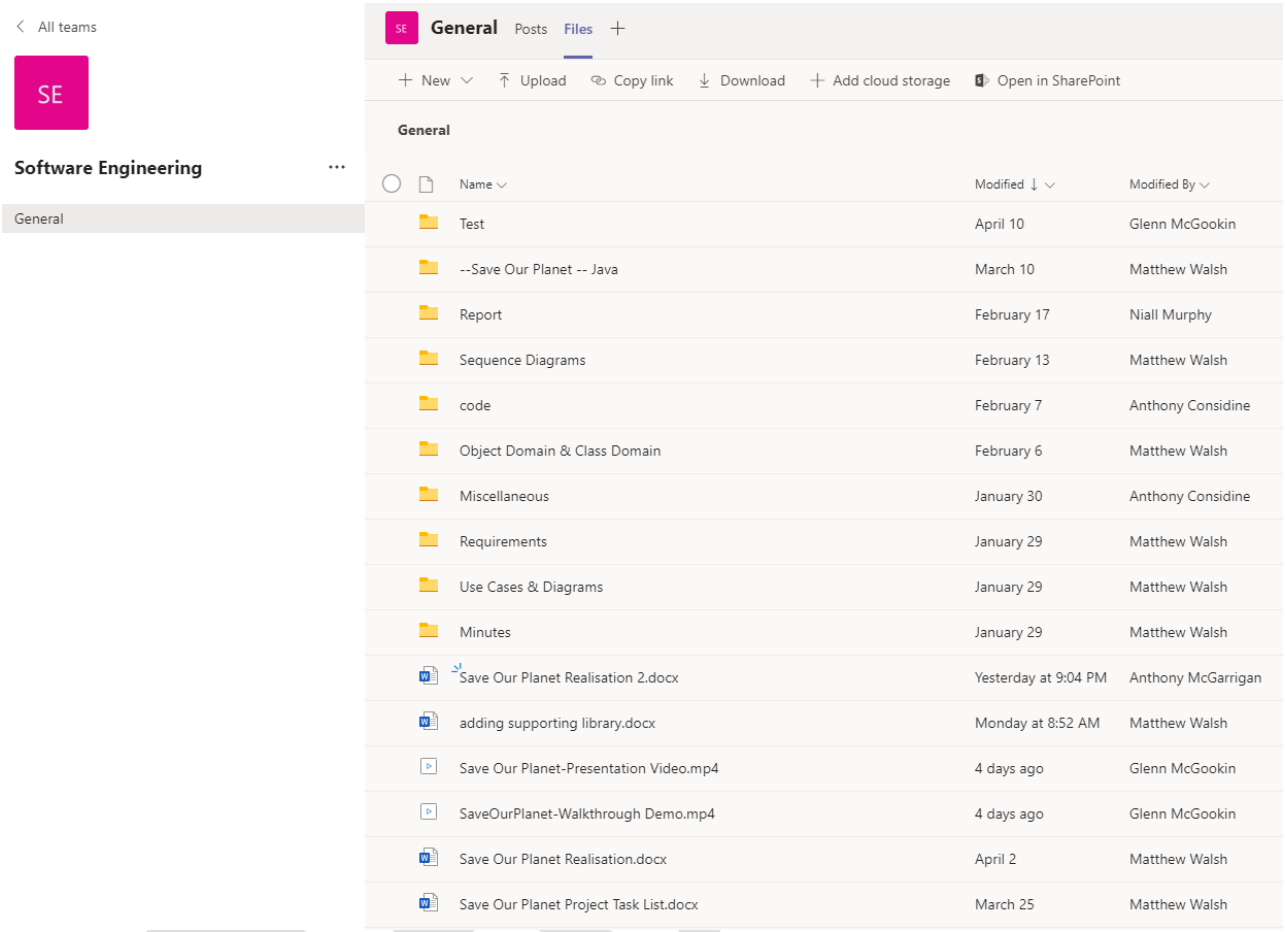
Developers worked from the Development branch which was merged with the Master branch after each phase was tested. Below are screenshots of the commit history for this project.





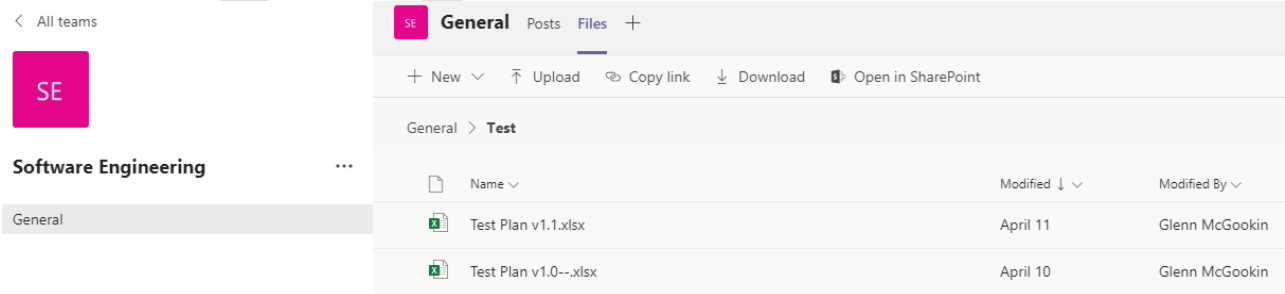
7.2 MS Teams

Entire Repository

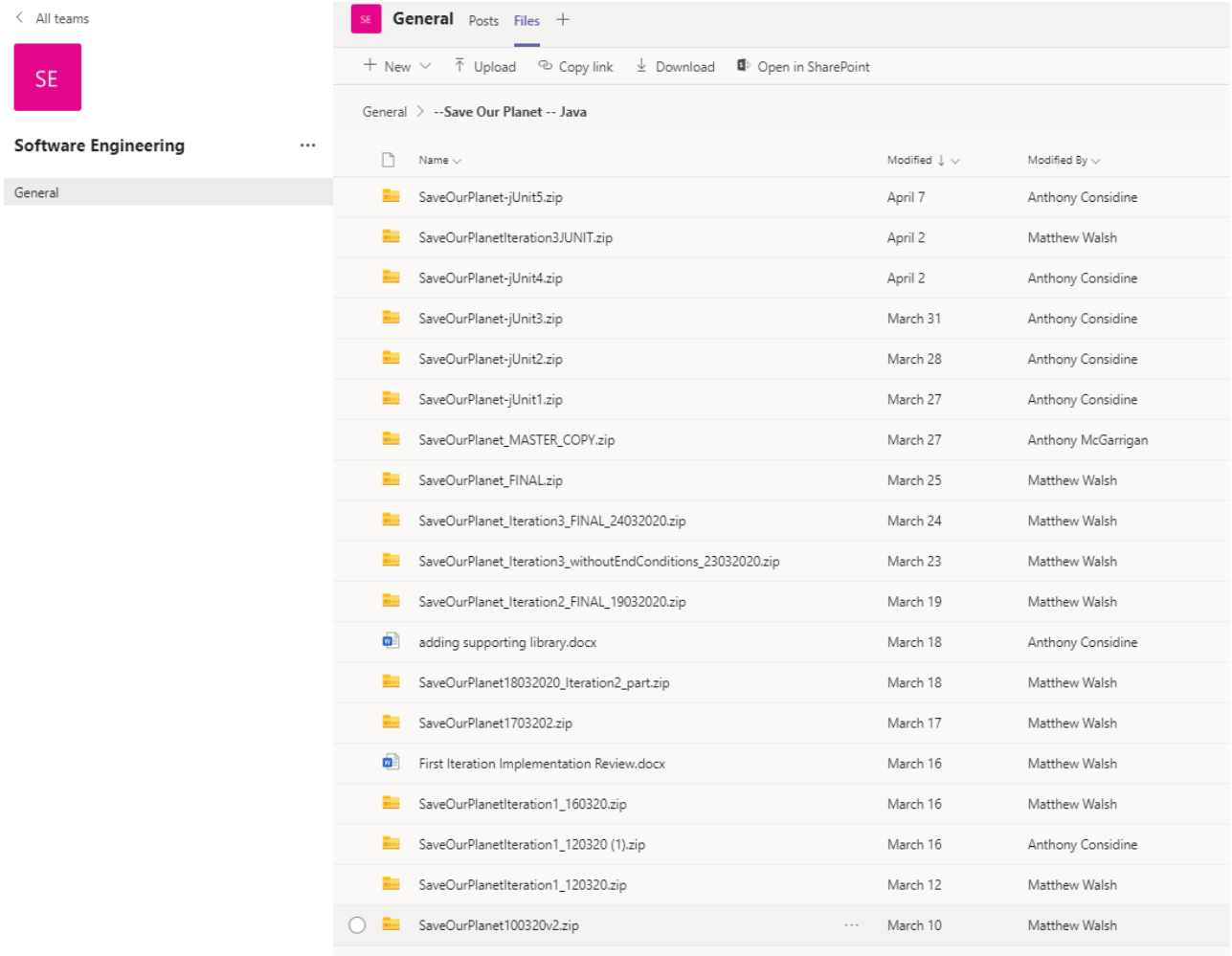


Sub-Folders

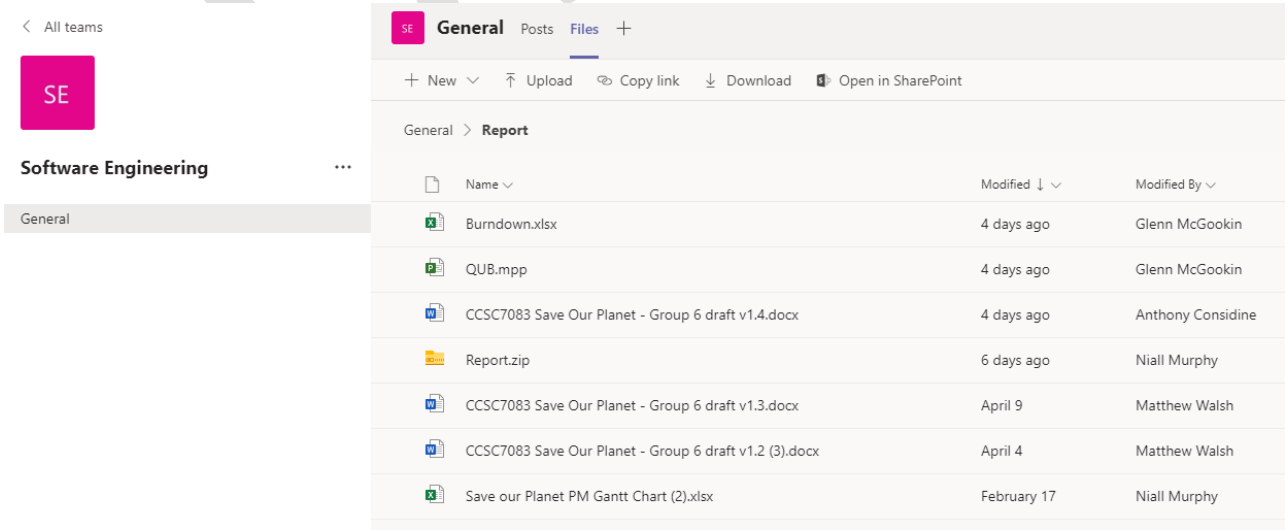
Test



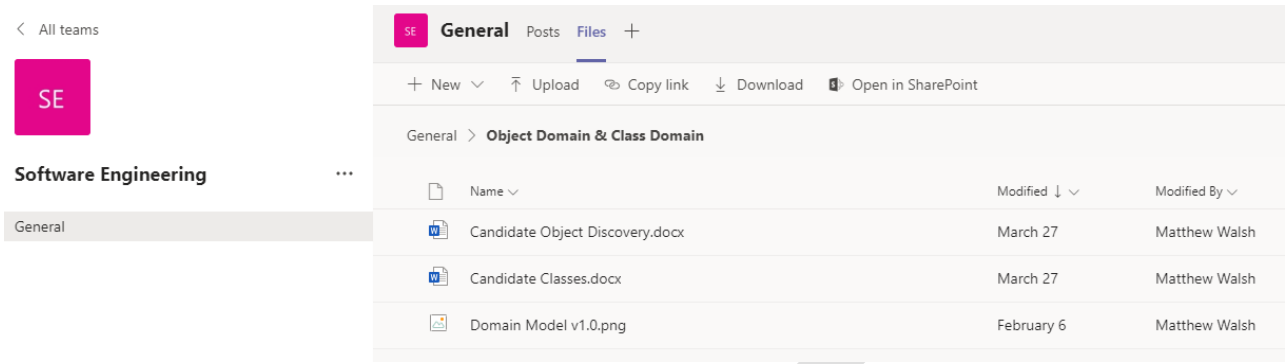
Save Our Planet – Java



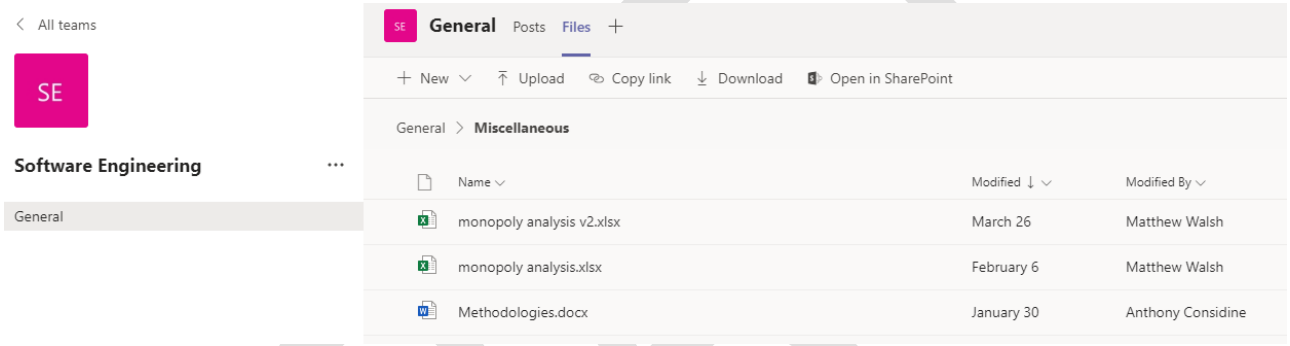
Report



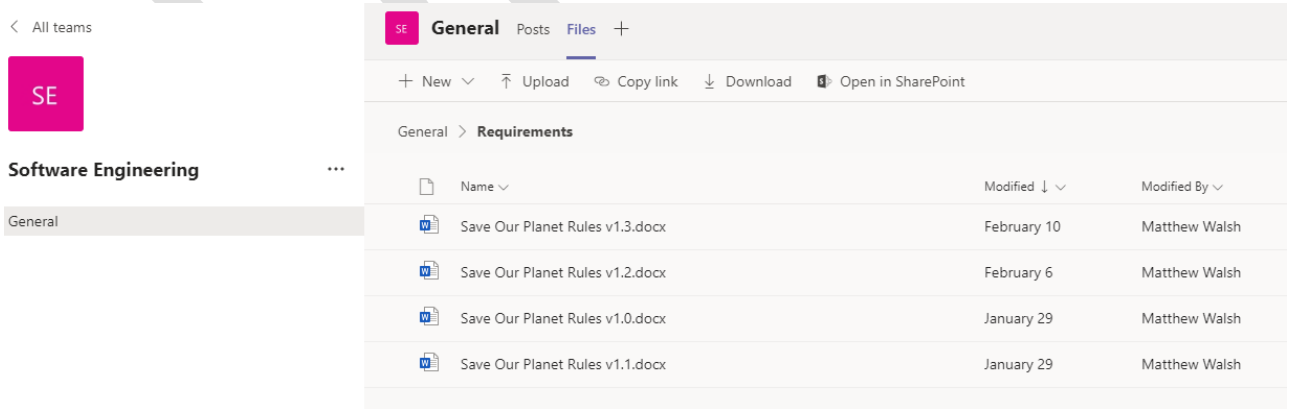
Object Domain & Class Domain



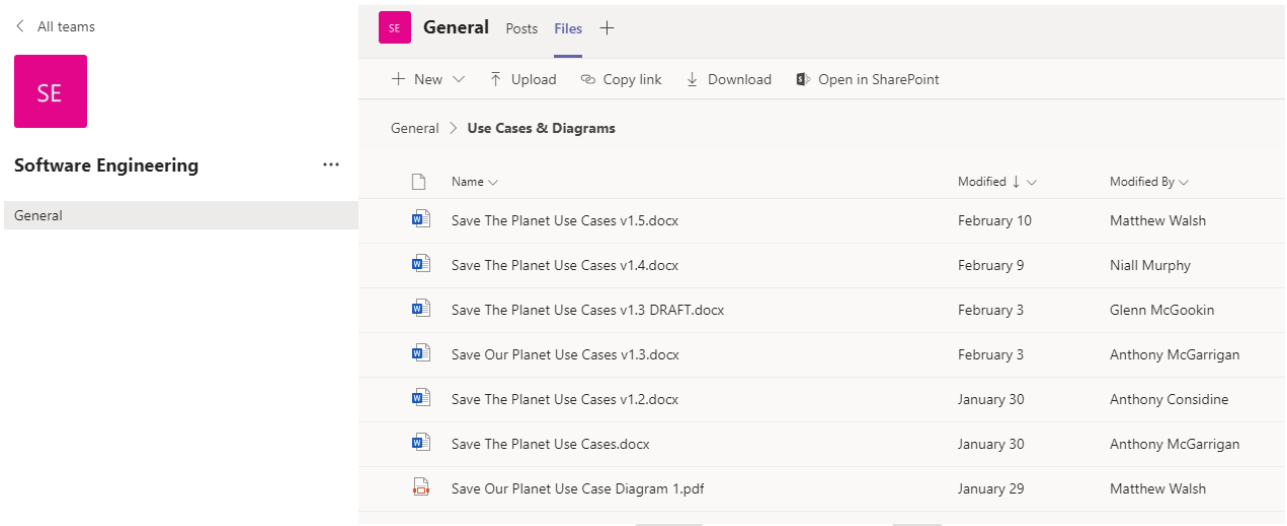
Miscellaneous



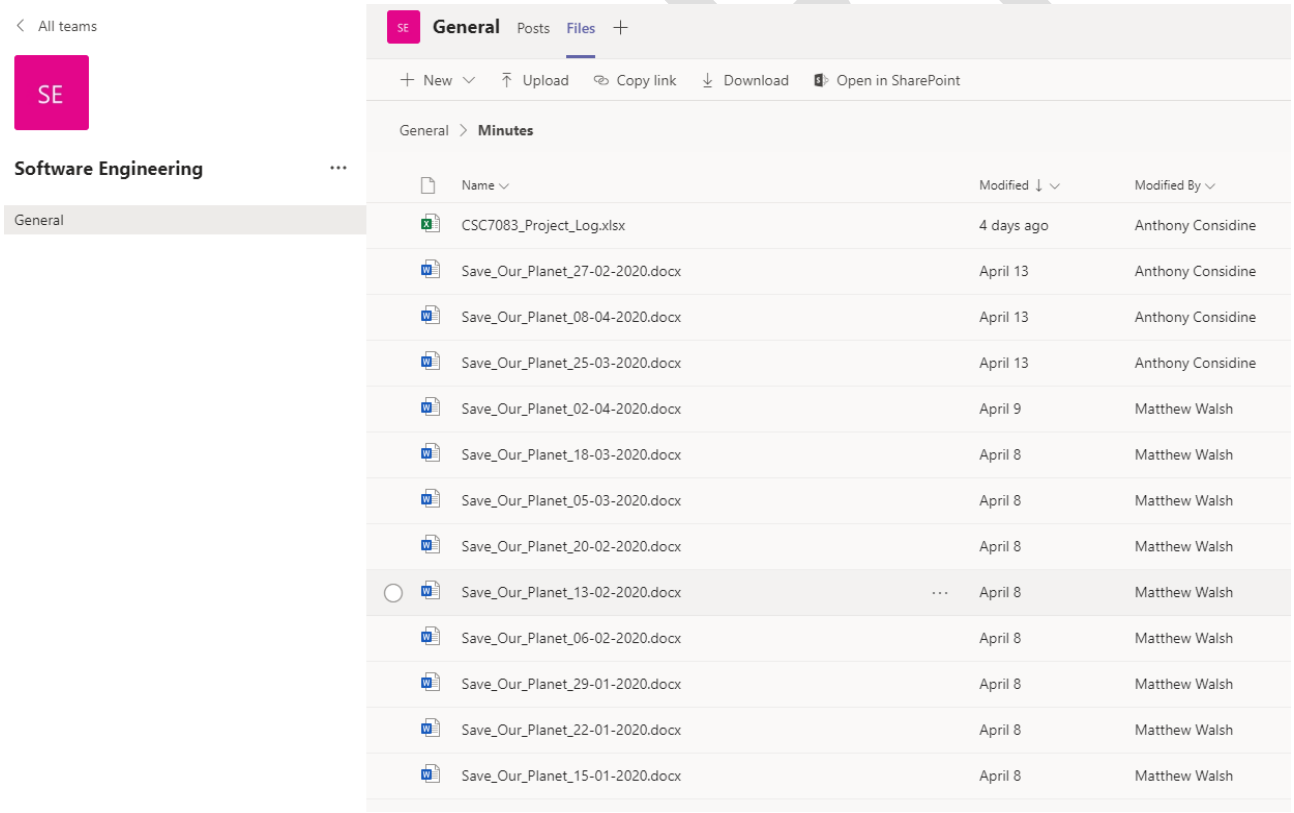
Requirements



Use Cases & Diagrams



Minutes



8 APPENDIX VI - PROJECT MANAGEMENT

This section comprehensively details the overall step by step software development process and the project management tools utilised. It details the development from identification of requirements, through to development, testing and delivery.

8.1 Timetable

The figure below details the summary timetable for project delivery.

Stage	w/c	Jan		Feb				Mar					Apr			
Stage 1 - Requirements Analysis		20	27	3	10	17	24	2	9	16	23	30	6	13	20	27
Define Objectives & Use Cases																
Use Case Model																
Testing																
Stage 2 - Architectural Design		20	27	3	10	17	24	2	9	16	23	30	6	13	20	27
Domain Model & Object Design																
Sequencing & UML Design																
Testing																
Stage 3 - Sprint 1		20	27	3	10	17	24	2	9	16	23	30	6	13	20	27
Development																
Testing																
Stage 4 - Sprint 2		20	27	3	10	17	24	2	9	16	23	30	6	13	20	27
Development																
Testing																
Stage 5 - Sprint 3		20	27	3	10	17	24	2	9	16	23	30	6	13	20	27
Development																
Testing																
Stage 6 - Finalisation		20	27	3	10	17	24	2	9	16	23	30	6	13	20	27
Acceptance Testing																
Report Development																
Release / Submission																

A full GANNT chart for delivery can be found in the external Appendix 4 folder accompanying this report.

8.2 Activity Schedule & Burndown

The full activity plan and burndown are available in the external appendix files accompanying this report.[4]

8.3 Requirements

Based on the rules of the game, outlined in Section 2 the following requirements were derived.

8.3.1 Core Requirements

Ref	Description
R001	The system shall accommodate up to 4 players
R003	The system shall ask Players to should be enter their names
R004	Players shall take turns during game play

[4] QUB.mpp will require MS Project or MS ProjectViewer app in order to view

R005	Players shall throw 2 virtual dice
R006	The system shall inform players where they have landed
R007	The system shall inform players of their obligations / opportunities
R008	User shall indicate their choice of action
R009	The system shall indicate the reason for a player's resources being changed
R010	The system shall indicate the player's new 'balance' (e.g. the 'funds' or 'credits' that are still available).
R011	The system shall start all players from a start square
R012	Players shall pick up their 'resources' from the start square
R013	The system shall include a square where nothing happens
R014	The system shall include four 'fields'
R015	The system shall include 'areas' within the 'fields'
R017	The system shall have 2 'fields' which consist of 3 'areas'
R018	The system shall have 2 'fields' which consist of 2 'areas'
R019	The system shall have 1 of the 2 'areas' with a 'field' which is the costliest field on the board to acquire and resource
R020	The system shall have 1 of the 2 'areas' with a 'field' which is the least costly field on the board to acquire and resource
R021	Players shall acquire all 'areas' in a field to build a development
R022	Players shall build a 'development' in any acquired area at the start of each turn
R023	The system shall have designated names for each 'development'
R024	The system shall have designated costs associated to each 'development', within an area
R025	Players shall pay or invest in another players acquired area when they land on it
R026	The system shall permit a maximum of 3 'development's in each area
R027	The system shall permit a maximum of 1 'main development' in each area, after a player has reached the maximum 'developments' in an area
R028	Players shall be required to give up resources if they land on an area that is acquired by another user, i.e. the more developed the area, the greater the resource consumed.
R029	The system shall end the game if 1 player decides they no longer wishes to play
R030	The system shall end the game if 1 player runs out of resources
R031	The system shall display the amount of resource each player holds, when the game ends,
R032	T system shall not convert 'developments', etc., to an equivalent value of 'resource units', when the game ends
R033	The system shall express the outcome of the game in a manner appropriate to the overall style of your version of Save Our Planet.

8.3.2 Objectives

Ref	Description
O-001	Start the program
O-002	Assign names to player users
O-003	Start a new game
O-004	Player users should be notified about errors
O-005	Player users will move in turn
O-006	Player users will roll a dice to determine how many spaces they can move
O-007	Player users will be notified of the status of the game
O-008	Player users will be notified of the status of their turn
O-009	Player users will be notified of available actions this turn
O-010	The system will manage user turns
O-011	The system will track a player user position within the game
O-012	The system will reduce the carbon footprint of the player user who purchased the area
O-013	The system will ensure that any reduction of carbon should be added to the visiting player user as a tax
O-014	The system will reduce the carbon footprint for the player user by landing on or passing the “Go Green” square
O-015	The player user can purchase an area to own
O-016	The “Take A Break” square will forfeit a player turn to benefit from gameplay
O-017	Player Users can develop structures to reduce their carbon footprint
O-018	Player Users can develop minor structures to increase the reduction value of their carbon footprint
O-019	Player Users can develop major structures to increase the reduction value of their carbon footprint
O-020	The system will end the game if 1 player decides they no longer wishes to play
O-021	The system will end the game if 1 player runs out of resources
O-022	The system will display a summary of carbon remaining for each player

8.3.3 Rules

Ref	Description
SR-001	A game requires a minimum of 2 player users to commence
SR-002	A maximum of 4 player users can take part in a game
SR-003	Player users must input their names as a string with at least 1 character
SR-004	Player users names must be distinct
SR-005	Player users should be suitably notified about erroneous inputs
SR-006	Player users should be instructed how to avoid erroneous inputs
SR-007	Player users must land on OR pass Go Green at least once to activate “Go Green”

SR-008	Players users can only reduce their carbon footprint once Go Green is activated
SR-009	Player users can choose to purchase an area by accepting and accumulating the carbon associated to this area.
SR-010	Player users who land on an area that has been purchased by another player user will incur a pay pollute fine
SR-011	Player users who land on Take a Break will forfeit this turn
SR-012	Player users should be notified of their status as each turn begins
SR-013	Player users should be notified of their status after each turn ends
SR-014	A minimum value of 2 can be rolled by a player user
SR-015	A maximum value of 12 can be rolled by a player user
SR-016	The system will add a pre-determined carbon fine to visiting player users carbon totals who land on a purchased area. The fine will increase depending upon the location and if structures have been built
SR-017	The system will subtract a pre-determined carbon value from the player users carbon total where they own the area. The value of the reduction will be equal to the value incurred by the visiting player user
SR-018	All users will benefit from passing Go Green. The system will deduct 200 carbon from their total carbon
SR-019	Player users who move and land on “Take A Break” will forfeit their turn
SR-020	A player user cannot develop a structure until they have purchased all areas in a field
SR-021	A player user can choose to add a structure any acquired area regardless of where they are on the board
SR-022	A player user can only 1 structure in 1 area during a given turn
SR-023	A player user can add a structure to an area can do so by accepting and accumulating the carbon associated to this area.
SR-024	A player user must build a maximum of 3 minor structures in an area, to build a ‘Major Structure’
SR-025	A player user cannot build more than 1 major structure in an area

8.4 Use Cases

8.4.1 *Enter Info* Use Case (UC1)

UC1:	**Flow of Events for the *Enter Info* Use Case**	
Actor:	Player user	
Objective:	O-001	Start the program
	O-002	Assign names to player users
	O-003	Start a new game
Preconditions:	User initialises the system.	

Main Flow:	1. The ***Error! Reference source not found.*** opens, and the console shall prompt the user to enter an integer to declare how many players are involved. 2. The user shall enter a number in the range 2 – 4 inclusive. 3. The user enters the name for each player. 4. The console prompts the user to select a token from a selection of car, boat, plane or helicopter that will be allocated to each player so that they shall make moves around the board. 5. The first player to have their name entered shall be the first player to make a move. 6. The console outputs the player's name and the value of their carbon footprint. 7. The console prompts the user to make a move by rolling the dice.
Alternative Flows:	1. At 2; If the user does not input an integer the **Error! Reference source not found.** is activated, and the console displays a message that shall inform the user that an appropriate type of value was not entered and prompt the user to re-enter a correct value. 2. At 2; Given the user enters a value greater than 4 or less than 2, then the **Error! Reference source not found.** shall activate and output a message via the console that informs the user about the type of error that has occurred, while prompting the user to re-enter a suitable number of players. 3. At 3; In the event that the user has entered the same game name for more than one player, then the *Display Error Message* (UC2) shall activate and output a message via the console that will prompt the user to re-enter unique game names for each player. 4. At 4; In the event the user selects the same board token for more than one player the **Error! Reference source not found.** shall be activated and the console will output a message that informs the user that the token selection was invalid, and prompt the user to re-select suitable tokens for each player.
Post Conditions:	The number of players, game names and tokens have been assigned to each player and accepted by the system, and the first player is ready to roll the dice.
Rules	SR-001: A game requires a minimum of 2 player users to commence SR-002: A maximum of 4 player users can take part in a game SR-003: Player users must input their names as a string with at least 1 character SR-004: Player user names must be distinct

8.4.2 *Display Error Message* Use Case (UC2)

UC2:	**Flow of Events for the *Display Error Message* Use Case**	
Actor:	Player user	
Objective:	O-004	Player users should be notified about errors
Pre-Conditions:	The player user has triggered the alternative flows in **Error! Reference source not found.**	
Main Flow:	1. The system shall display the error message dialog, providing the context that triggered the error message	
Alternative Flows:	There are no alternative flow	
Post-Conditions:	The system shall clear the erroneous input data and reset the console view providing context of what caused the error	
Rules:	SR-005	Player users should be suitably notified about erroneous inputs

	SR-006	Player users should be instructed how to avoid erroneous inputs

8.4.3 *Make Move* Use Case (UC3)

UC3:	Flow of Events for the *Make Move* Use Case	
Actor:	Player user	
Objective:	O-005	Player users will move in turn
	O-006	Player users will roll a dice to determine how many spaces they can move
Preconditions:	The number of players and player names have been determined and accepted.	
Main Flow:	1. The **Error! Reference source not found.** opens, and the console shall inform the player about their game information. 2. The console prompts the player to select 'Yes' or 'No' to determine whether they wish to continue playing the game. 3. The console shall prompt the player to roll the dice and make a move. 4. The **Error! Reference source not found.** shall open. 5. After the player's token lands on the destination square, **Error! Reference source not found.** opens and outputs a message via the console that informs the player about the attributes of the square they have landed on. 6. Upon a player completing a move the console shall output a message displaying the player's game statistics and the **Error! Reference source not found.** will open.	
Alternative Flows:	1. At 1; Given, the player controls enough areas to own a field that houses no eco-friendly structures or minor eco-friendly structures only, the console will prompt the player to determine if they wish to develop a minor eco-friendly structure on a single area within the particular field. Given, the player decides to develop a minor structure the **Error! Reference source not found.** shall open, only one minor structure can be developed in each turn and the player's carbon footprint rate is increased. 2. At 1; Given, the player controls enough areas to own a field that houses minor eco-friendly structures on each area within the field, the console will prompt the player to determine if they wish to develop a major eco-friendly structure on a single area within the particular field. Given, the player decides to develop a major structure the **Error! Reference source not found.** shall open, only one major structure can be developed in each turn and the player's carbon footprint value is increased at a larger rate than developing a minor structure. 3. At 2; Given, the user selects 'Yes' then the player loses the game and the **Error! Reference source not found.** is activated. 4. At 5; If the player's token has landed on an area owned by another player the *Pay Pollute Cost* use case (UC7) activates, the player currently taking their turn shall obtain a carbon footprint from the player who owns the area landed on. 5. At 5; Given, the player's token lands on the 'Take A Break' square, the *Take A Break* use case (UC10) is activated. 6. At 5; Given, the player's token lands on or passes the 'Go Green' square the **Error! Reference source not found.** activates and the player shall have 200 carbon subtracted from their carbon footprint value. 7. At 5; Given, the player's token lands on an area not owned by another player the **Error! Reference source not found.** shall open and the console will output information about the availability and	

	cost of developing structures on the area. If the player selects 'Yes' and the cost of purchasing the area results in the player's carbon value meeting the acceptable threshold, then the player will be prevented from purchasing the area. Conversely, if the player selects 'Yes' and the cost of obtaining the area does not result in the player's carbon value meeting the acceptable threshold then the **Error! Reference source not found.** shall be activated.	
Post Conditions:	The player's board token shall be moved clockwise to the destination square.	
Rules:	SR-007	Player users must land on OR pass Go Green at least once to activate "Go Green"
	SR-008	Players users can only reduce their carbon footprint once Go Green is activated
	SR-009	Player users can choose to purchase an area by accepting and accumulating the carbon associated to this area.
	SR-010	Player users who land on an area that has been purchased by another player user will incur a pay pollute fine
	SR-011	Player users who land on Take a Break will forfeit this turn

8.4.4 *Display Info* Use Case (UC4)

UC4:	**Flow of Events for the *Display Info* Use Case**	
Actor:	Player user	
Objective:	O-007	Player users will be notified of the status of the game
	O-008	Player users will be notified of the status of their turn
	O-009	Player users will be notified of available actions this turn
Pre-Conditions:	The game must be initialised	
Main Flow:	1. The system shall display a player users' details in context of the action taken 2. The system shall display area information in context of the user who has landed upon this space	
Alternative Flows:	There is no alternative flow	
Post-Conditions:	The user will be informed of their game status as per documented Use Cases	
Rules:	SR-012	Player users should be notified of their status as each turn begins
	SR-013	Player users should be notified of their status after each turn ends

8.4.5 *Change Player* Use Case (UC5)

UC5:	**Flow of Events for the *Change Player* Use Case**	
Actor:	System	
Objective:	O-005	Player users will move in turn
	O-010	The system will manage user turns
Preconditions:	It is the player's turn. The player has rolled the dice and moved their token to a new area on the board.	

Main Flow:	1. The player's turn ends after they have rolled the dice.	
Alternative Flows:	None	
Post Conditions:	Player will proceed with the next action of their turn.	
Rules:		There are no rules

8.4.6 *Roll Dice* Use Case (UC6)

UC6:	**Flow of Events for the *Roll Dice* Use Case**	
Actor:	System	
Objective:	O-006	Player users will roll two dice to determine how many spaces they can move
	O-011	The system will track a player users position within the game
Preconditions:	It is the player's turn.	
Main Flow:	1. The **Error! Reference source not found.** opens, and the console prompts the player to press 'Enter' in order to roll the dice. 2. The value of the dice is a random integer between the range 2 – 12 (inclusive).	
Alternative Flows:	There is no alternative flow	
Post Conditions:	The player's token is moved clockwise to a destination square that corresponds with the value of the dice roll.	
Rules	SR-014	A minimum value of 2 can be rolled by a player user
	SR-015	A maximum value of 12 can be rolled by a player user

8.4.7 *Pay Pollute Cost* Use Case (UC7)

UC7:	**Flow of Events for the *Pay Pollute Cost* Use Case**	
Actor:	Player user	
Objective:	O-012	Reduce the carbon footprint of the player user who purchased the area
	O-013	Any reduction of carbon should be added to the visiting player user as a tax
Preconditions:	It is the player's turn. The player has rolled the dice. The player's token lands on an area that is owned by another player.	
Main Flow:	1. The player accumulates a carbon footprint from the player who owns the area. The rate of the carbon transferred depends on the type of area the player has landed on. 2. The rate of carbon is deducted from the player who owns the area and added to the carbon of the player that is to pay the pollute cost. 3. If all areas in a specified field are owned, the pollute cost is greater. 4. If a minor structure is present in an area, the pollute cost is greater again. 5. If a major structure is present in an area, the pollute cost is greatest.	

	6. If the amount of carbon the player accumulates is greater than the carbon footprint limit, the player loses, and the **Error! Reference source not found.** is opened and the score board is displayed via the console.	
Alternative Flows:	There is no alternative flow	
Post Conditions:	The player accumulates the carbon footprint associated with their token landing on the area or the game terminates.	
Rules:	SR-016	The system will add a pre-determined carbon fine to visiting player users carbon totals who land on a purchased area. The fine will increase depending upon the location and if structures have been built
	SR-017	The system will subtract a pre-determined carbon value from the player users carbon total where they own the area. The value of the reduction will be equal to the value incurred by the visiting player user

8.4.8 *Go Green* Use Case (UC8)

UC8:	**Flow of Events for the *Go Green* Use Case**	
Actor:	Player user	
Objective:	O-014	The system will reduce the carbon footprint for the player user by landing on or passing the "Go Green" square
Preconditions:	It is the player's turn. The player has rolled the dice. The player's token lands on or passes the 'Go Green' square.	
Main Flow:	1. Given, the player's token lands on or passes the 'Go Green' square, a value of 200 shall be subtracted from the player's carbon footprint total. 2. Given, the player's carbon footprint value is reduced to a value equal to or less than zero then the player wins, and the **Error! Reference source not found.** is activated.	
Alternative Flows:	There is no alternative flow	
Post Conditions:	The player's token arrives on the destination square and 200 carbon shall be subtracted from the player's carbon footprint total.	
Rules:	SR-007	Player users must land on OR pass Go Green at least once to activate "Go Green"
	SR-018	All users will benefit from passing Go Green. The system will deduct 200 carbon from their total carbon

8.4.9 *Purchase Area* Use Case (UC9)

UC9:	**Flow of Events for the *Purchase Area* Use Case**	
Actor:	Player user	
Objective:	O-015	The player user can purchase an area to own
Preconditions:	It is the player's turn. The player has rolled the dice.	

	The player's token has landed on an area available to purchase.	
Main Flow:	1. The **Error! Reference source not found.** opens, and the console shall inform the player about the attributes of the area they have arrived on. 2. The console shall prompt the player to select 'Yes' or 'No' regarding the purchase of the area. 3. Given, the player selects 'Yes', ownership of the area is assigned to the player currently in-play and a carbon footprint value is added to the player's carbon footprint total. 4. Given, the player selects 'No' the player's turn is finished, the area remains in available status. 5. Given, the player selects 'Yes' and obtaining the area shall gain the player enough carbon that will increase the player's carbon value to meet the acceptable threshold, the **Error! Reference source not found.** shall open and the console will display a message informing the player that obtaining the area is not possible as their carbon threshold will be exceeded.	
Alternative Flows:	There is no alternative flow	
Post Conditions:	The player's turn is finished.	
Rules	SR-007	Player users must land on OR pass Go Green at least once to activate "Go Green"
	SR-009	Player users can choose to purchase an area by accepting and accumulating the carbon associated to this area.

8.4.10 *Take a Break* Use Case (UC10)

UC10:	**Flow of Events for the *Take A Break* Use Case**	
Actor:	Player user	
Objective:	O-016	The "Take A Break" square will forfeit a player user turn to benefit from gameplay
Preconditions:	It is the player users turn. The player users has rolled the device. The player users token lands on the 'Take a Break' square.	
Main Flow:	1. If the player users lands on the 'Take A Break' square 2. The system shall activate the **Error! Reference source not found.** advising them that that play will move to the next player 3. The system shall activate the **Error! Reference source not found.**	
Alternative Flows:	There is no alternative flow	
Post Conditions:	The player users cannot perform any function until their next turn. The system will ask the next player users to take their turn	
Rules:	SR-019	Player users who move and land on "Take A Break" will forfeit their turn

8.4.11 *Develop Structure* Use Case (UC11)

UC11:	**Flow of Events for the *Develop Structure* Use Case**
Actor:	Player user

Objective:	O-017	Player Users can develop structures to reduce their carbon footprint
Preconditions:	It is the player's turn. The player owns all the areas in a specified field.	
Main Flow:	1. At the start of the players turn the console will open to prompt the user if they want to develop an area in a field that they own. 2. If a player opts to develop an area, the (UC4) *Display Info* opens and prompts the player to choose an area in a field to develop.	
Alternative Flows:	1. At 1; The player user activates **Error! Reference source not found.** 2. At 2; The player can activate the **Error! Reference source not found.** or **Error! Reference source not found.**	
Post Conditions:	The player may develop an area. The system shall allocate a carbon cost to the user balance for this structure The system will activate the **Error! Reference source not found.**	
Rules:	SR-021	A player user can choose to add a structure on any acquired area regardless of where they are on the board
	SR-022	A player user can only develop 1 structure in 1 area during a given turn
	SR-023	A player user can add a structure to an area by accepting and accumulating the carbon associated to this area.
	SR-024	A player user must build a maximum of 3 minor structures in an area, to build a 'Major Structure'
	SR-025	A player user cannot build more than 1 major structure in an area

8.4.12 *Develop Minor Structure* Use Case (UC12)

UC12:	**Flow of Events for the *Develop Minor Structure* Use Case**	
Actor:	System	
Objective:	O-018	Player Users can develop minor structures to increase the reduction value of their carbon footprint
Preconditions:	Main flow has been followed in **Error! Reference source not found.**	
Main Flow:	1. Given, a player opts to add a minor structure to an area, the **Error! Reference source not found.** activates. The system will display the updated details associated with that player user: - A player's carbon footprint is increased by 600 after developing a minor eco-friendly structure within the North America field. - A player's carbon footprint is increased by 450 after developing a minor eco-friendly structure within the Asia field. - A player's carbon footprint is increased by 300 after developing a minor eco-friendly structure within the Central Europe field. - A player's carbon footprint is increased by 90 after developing a minor eco-friendly structure within the Northern Europe field.	

Alternative Flows:	There is no alternative flow	
Post Conditions:	The system updates the player user's carbon footprint by the associated cost of the developing the minor structure within the area. The player user can continue with their dice roll.	
Rules:		There are no rules

8.4.13 *Develop Major Structure* Use Case (UC13)

UC13:	**Flow of Events for the *Develop Major Structure* Use Case**	
Actor:		
Objective:	O-019	Player Users can develop major structures to increase the reduction value of their carbon footprint
Preconditions:	Main flow has been activated in **Error! Reference source not found.**	
Main Flow:	1. Given, a player opts to add a major structure to an area, the (UC4) *Display Info* activates and will display the updated details associated with that player. - A player's carbon footprint is increased by 600 after developing a minor eco-friendly structure within the North America field. - A player's carbon footprint is increased by 450 after developing a minor eco-friendly structure within the Asia field. - A player's carbon footprint is increased by 300 after developing a minor eco-friendly structure within the Central Europe field. - A player's carbon footprint is increased by 90 after developing a minor eco-friendly structure within the Northern Europe field.	
Alternative Flows:	There is no alternative flow	
Post Conditions:	The system updates the player user's carbon footprint by the associated cost of the developing the major structure within the area. The player user can continue with their dice roll.	
Rules:		There are no rules

8.4.14 *End Game* Use Case (UC14)

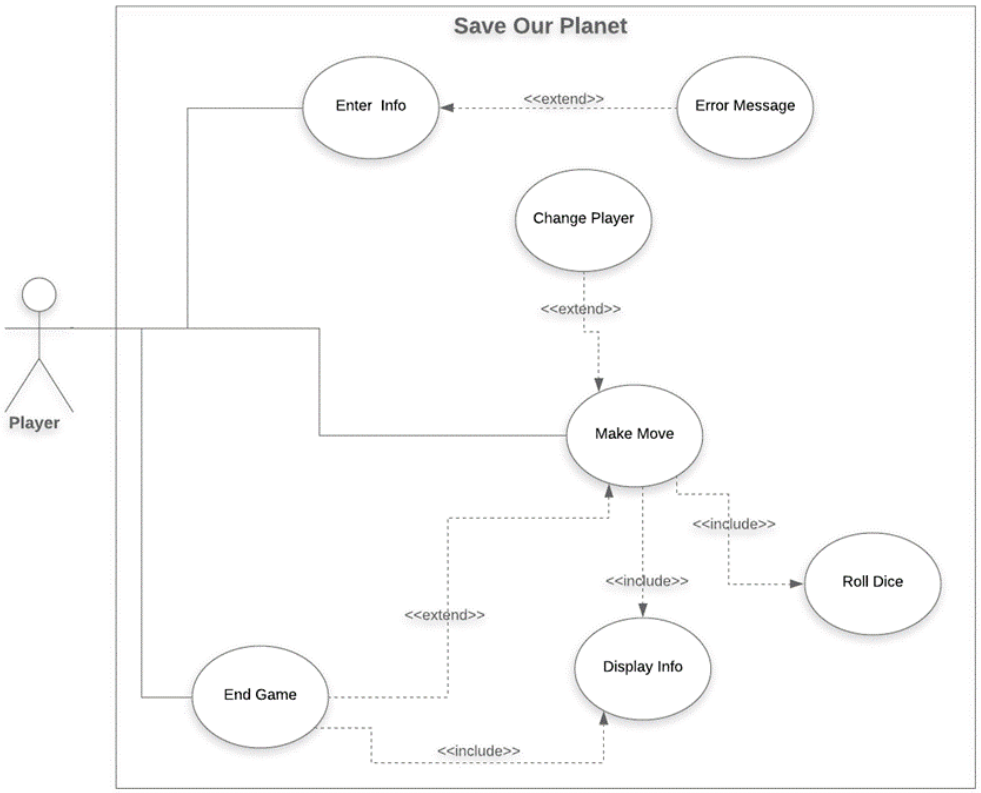
UC14:	**Flow of Events for the *End Game* Use Case**	
Actor:	Player user	
Objective:	O-020	The system will end the game if 1 player decides they no longer wishes to play
	O-021	The system will end the game if 1 player runs out of resources
	O-022	The system will display a summary of carbon remaining for each player
Preconditions:	The player has opted to end the game.	

Main Flow:	1. The **Error! Reference source not found.** opens and the players scores are output via the console in descending order (lowest to highest). The player with the least amount of carbon footprint wins the game. 2. Given, more than player has the same least amount of a carbon footprint then the game is declared a draw.	
Alternative Flows:	None	
Post Conditions:	Game play is terminated.	
Rules:		There are no rules

8.5 Use Case Diagram

The figure below represents the first iteration of gameplay, implemented to discover the core functionality of the system.

Figure 8.1.: Use Case Diagram – First Iteration



8.6 Candidate Object Discovery

The tables below represent the process of candidate object discovery which assisted us in our realisation of our candidate classes.

8.6.1 *Enter Info* Use Case (UC1)

UC1	NOUN/OBJECT	DOING/OPERATION	BEING/INHERITANCE	HAVING/AGGREGATION	MODAL/CONSTRAINTS	ADJECTIVE/ATTRIBUTE
1.	Display Info Console	Display info opens Prompt user to enter number of players		Integer to declare number of players		Number of players
2.	user	enters number of players			in range of 2-4 inclusive	
3.	user	enters name of player		name of player		Player name
4.	Console Token	prompts user to select token making a move around the board	Car, boat, plane, helicopter	Allocated to each player	From a selection of	Token type
5.	The player	Make a move			First player	Player name
6.	Console The player	Outputs player name		Value of carbon footprint		Carbon footprint Player name
7.	Console The player The dice	Prompts user to make move Make move by roll dice				

8.6.2 *Display Error Message* Use Case (UC2)

UC2	NOUN/OBJECT	DOING/OPERATION	BEING/INHERITANCE	HAVING/AGGREGATION	MODAL/CONSTRAINTS	ADJECTIVE/ATTRIBUTE
1.	The system	Display error message			Context of error message	

8.6.3 *Make Move* Use Case (UC3)

UC3	NOUN/OBJECT	DOING/OPERATION	BEING/INHERITANCE	HAVING/AGGREGATION	MODAL/CONSTRAINTS	ADJECTIVE/ATTRIBUTE
1.	Display info Console Player Game	Display info opens Inform player of game information				Game information
2.	Console Player Game	Player select yes or no			Select yes or no	
3.	Console Player dice	Roll dice Make a move Prompt player				
4.	dice	Roll dice				

5.	Player Token Destination square console	Display info opens Displays square information Token lands on square				Attributes of the square landed on
6.	Player console	Completing a move Display players game statistics Change player				Game statistics

8.6.4 *Display Info* Use Case (UC4)

UC4	NOUN/OBJECT	DOING/OPERATION	BEING/INHERITANCE	HAVING/AGGREGATION	MODAL/CONSTRAINTS	ADJECTIVE/ATTRIBUTE
1.	Player system	Display players details Action taken				Players details
2.	System Area Square Player	Landed on square				Area information

8.6.5 *Change Player* Use Case (UC5)

UC5	NOUN/OBJECT	DOING/OPERATION	BEING/INHERITANCE	HAVING/AGGREGATION	MODAL/CONSTRAINTS	ADJECTIVE/ATTRIBUTE
1.	Player	Roll dice			Their turn	

8.6.6 *Roll Dice* Use Case (UC6)

UC6	NOUN/OBJECT	DOING/OPERATION	BEING/INHERITANCE	HAVING/AGGREGATION	MODAL/CONSTRAINTS	ADJECTIVE/ATTRIBUTE
1.	Display info Console player	Console prompts player Player presses enter to roll dice				
2.	dice			Random integer	Range 2 – 12 inclusive	Dice face value

8.6.7 *Pay Pollute Cost* Use Case (UC7)

UC7	NOUN/OBJECT	DOING/OPERATION	BEING/INHERITANCE	HAVING/AGGREGATION	MODAL/CONSTRAINTS	ADJECTIVE/ATTRIBUTE
1.	Player Carbon footprint Area Rate	Player lands on area Player transfers carbon footprint			Player owns the area	Area type Area rate Rate of carbon
2.	Carbon Rate Area Pollute cost	Carbon deducted from player Player pays pollute cost			Player owns area	Rate of carbon

	player					
3.	Area Field Pollute cost			Pollute cost is greater	All areas in field owned	
4.	Area Field Pollute cost Minor structure			Pollute cost is greater	All areas in field owned	
5.	Area Field Pollute cost Major structure			Pollute cost is greater	All areas in field owned	
6.	Carbon footprint Player Game Console Scoreboard	Player loses Game ends Scoreboard displayed			Carbon footprint limit	Carbon footprint limit

8.6.8 *Go Green* Use Case (UC8)

UC8	NOUN/OBJECT	DOING/OPERATION	BEING/INHERITANCE	HAVING/AGGREGATION	MODAL/CONSTRAINTS	ADJECTIVE/ATTRIBUTE
1.	Player Token Square "go green' square	Lands on or passes go green square Carbon footprint subtracted			Carbon footprint value = 200	Carbon footprint total
2.	Player Game Carbon footprint	Game ends carbon footprint reduced Player wins			Players Carbon footprint <= 0	Carbon footprint value

8.6.9 *Purchase Area* Use Case (UC9)

UC9	NOUN/OBJECT	DOING/OPERATION	BEING/INHERITANCE	HAVING/AGGREGATION	MODAL/CONSTRAINTS	ADJECTIVE/ATTRIBUTE
1.	Display info Console Player area	Arrive on area Display info of area				Area attributes
2.	Console User area	Player can choose to purchase area			Purchase area (yes or no)	
3.	Player Area Carbon footprint	Players purchase area Value added to carbon footprint			The player in play	Carbon footprint total
4.	Player Area	Players turn Players turn ends			Player turn	Area status

		Player does not purchase area				
5.	Player Carbon Display info Console area	Increase players carbon footprint Exceed carbon limit Display cannot purchase message			Players carbon value lower than allowed limit	Carbon value Carbon limit/threshold

8.6.10 *Take a Break* Use Case (UC10)

UC10	NOUN/OBJECT	DOING/OPERATION	BEING/INHERITANCE	HAVING/AGGREGATION	MODAL/CONSTRAINTS	ADJECTIVE/ATTRIBUTE
1.	Player Take a break square Display info	Land on take a break square Display move message Change to next player			Player has no actions	
2.	Player	Change player				

8.6.11 *Develop Structure* Use Case (UC11)

UC11	NOUN/OBJECT	DOING/OPERATION	BEING/INHERITANCE	HAVING/AGGREGATION	MODAL/CONSTRAINTS	ADJECTIVE/ATTRIBUTE
1.	Player Console Area field	Display develop option			Player must own areas in a field Start of players turn	
2.	Player Display info Area Field	Player chooses to develop structure Player selects area to develop				

8.6.12 *Develop Minor Structure* Use Case (UC12)

UC12	NOUN/OBJECT	DOING/OPERATION	BEING/INHERITANCE	HAVING/AGGREGATION	MODAL/CONSTRAINTS	ADJECTIVE/ATTRIBUTE
1.	Player Minor structure Display info System Carbon footprint	Player develops minor structure Players carbon footprint increased		Structure has minor type structure		Development cost

8.6.13 *Develop Major Structure* Use Case (UC13)

UC13	NOUN/OBJECT	DOING/OPERATION	BEING/INHERITANCE	HAVING/AGGREGATION	MODAL/CONSTRAINTS	ADJECTIVE/ATTRIBUTE
1.	Player Major structure Display info	Player develops minor structure Players carbon footprint increased		Structure has Major type structure		Development cost

	System Carbon footprint					

8.6.14 *End Game* Use Case (UC14)

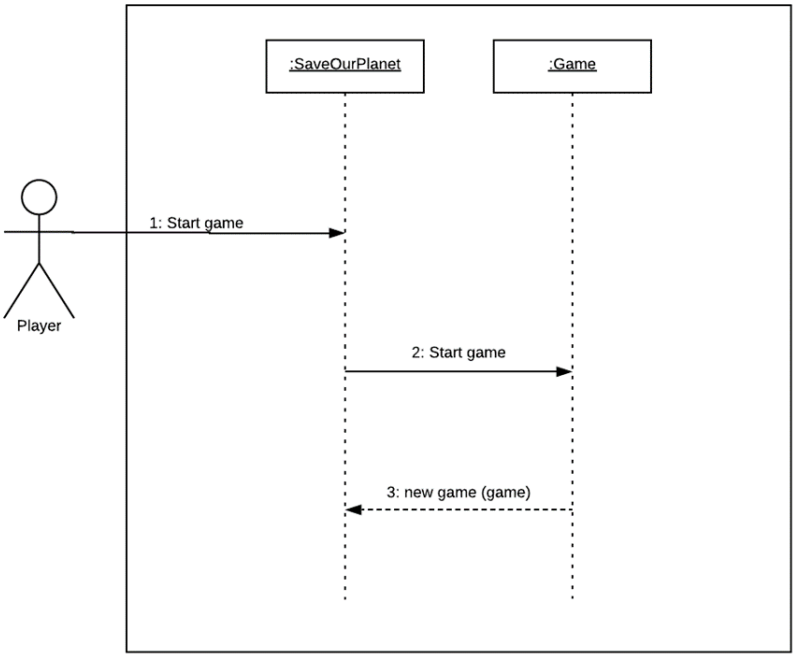
UC14	NOUN/OBJECT	DOING/OPERATION	BEING/INHERITANCE	HAVING/AGGREGATION	MODAL/CONSTRAINTS	ADJECTIVE/ATTRIBUTE
1.	System Display info Console Player Game Carbon footprint	Players scores are compared Players scores are displayed Player wins game			Display from lowest to highest carbon footprint	Players score
2.	Player Carbon footprint Game	Players scores are compared Players scores are displayed Player wins game			Two players have same score	

8.7 Sequence Diagrams, Domain Models & UML Diagrams

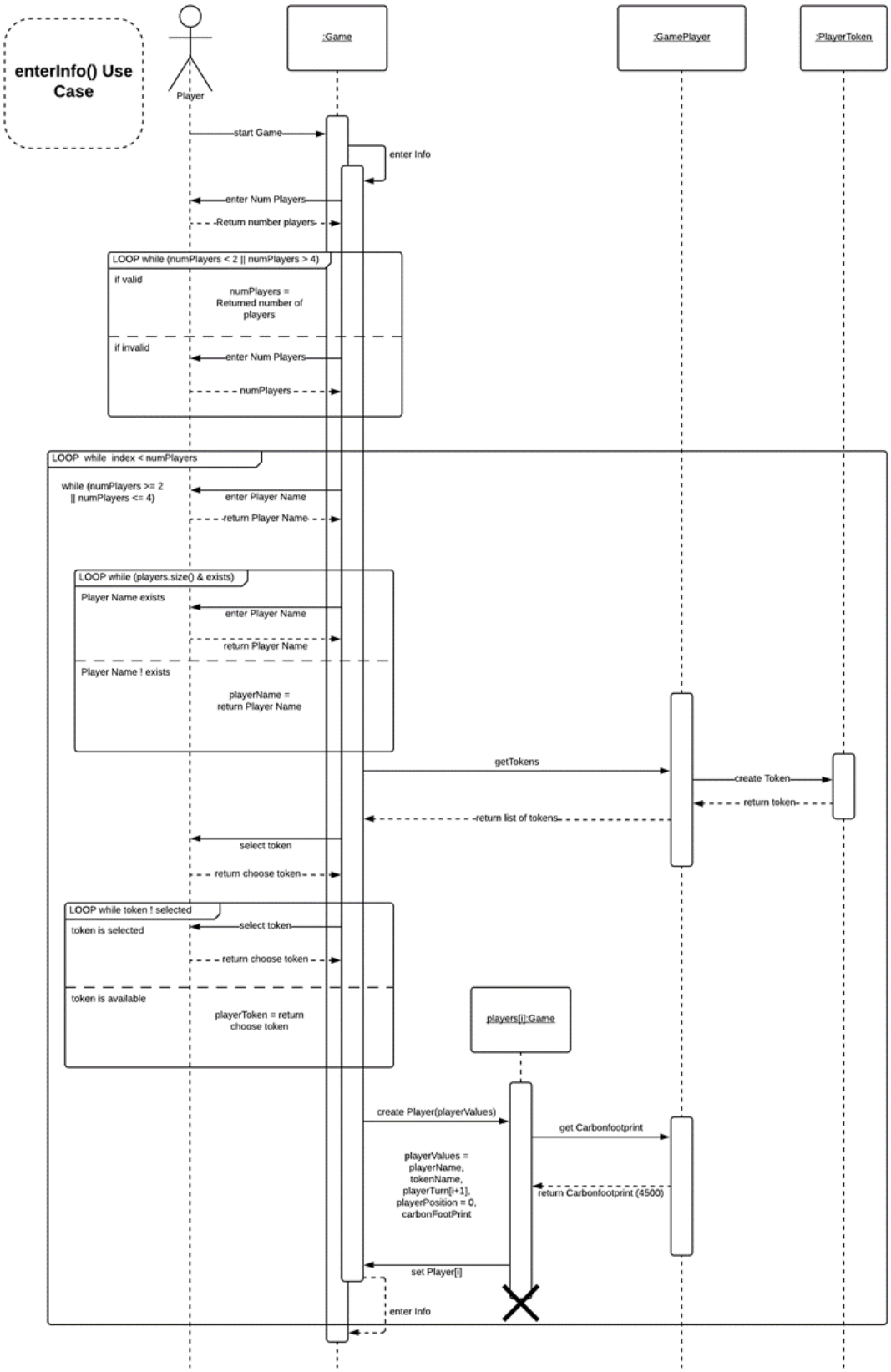
The following section outlines the sequence diagrams developed for each element of gameplay, along with the accompanying domain models and resulting UML diagrams. Development of this aspect took place over the course of 3 iterations in line with the development sprints.

8.7.1 *Iteration 1*

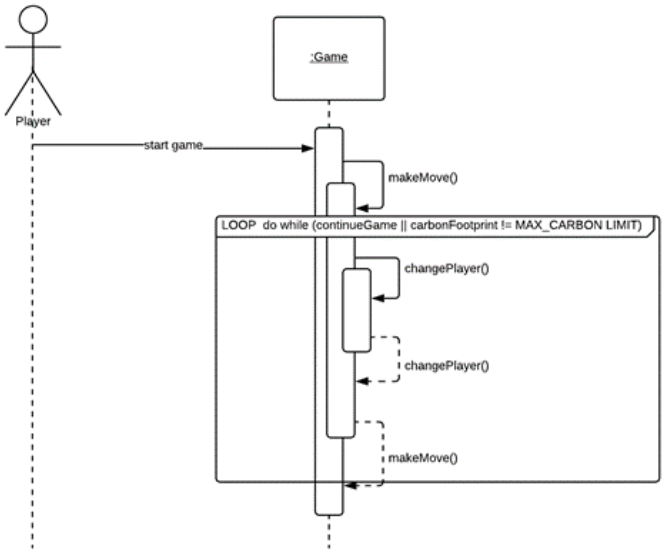
Player Starts Game



Enter Info Use Case (UC1)



Make Move Use Case (UC3)



player plays the game

- what does playing a game entail?

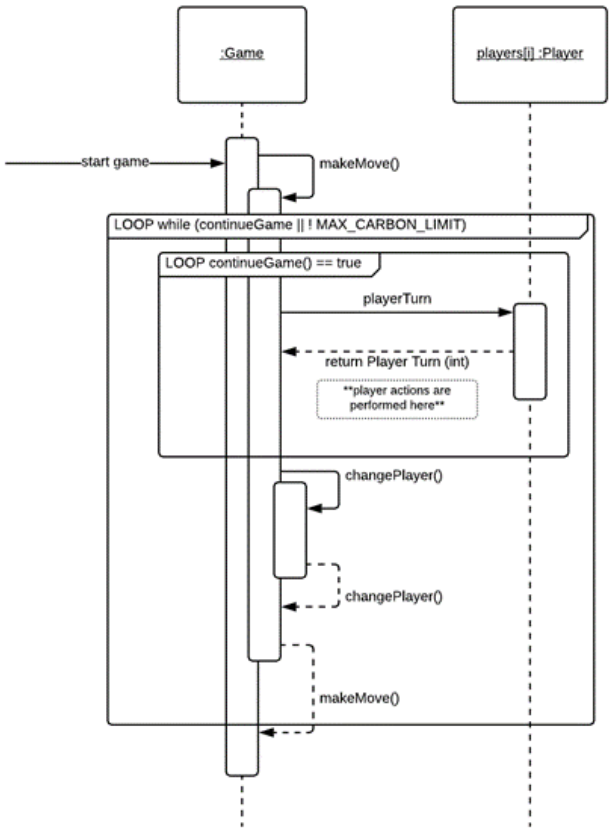
- player makes a move **makeMove()**
- each player takes turn, once each player has taken their turn another round starts **changePlayer()**

the game continues until one of two exit conditions are met (0 carbonFooprint || carbonFootPrintLimit) & **continueGame()** - player decides do they wish to continue

Game attributes
carbonFootPrint Limit (belongs to game)
continueGame(true || false / int)

player attribute
carbonFootprint (this can be compared to Limit int)

8.7.2 *Player Turn*



the players takes their turn

how do we know the players turn?
player knows their turn (**playerTurn(int)**)
playerTurn assigns the order of play to each player

what does a round entail?

- each player will take a turn

- actions relating to that player will occur here (i.e. roll dice, move token etc)

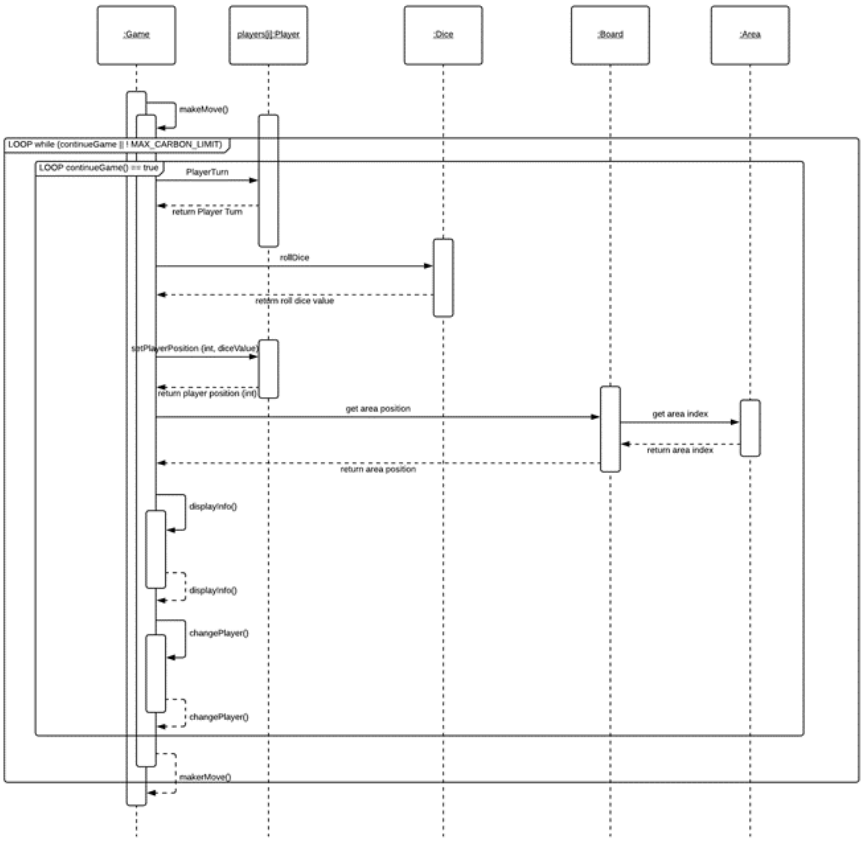
- at the end of each player turn, changePlayer changes to next Players turn

player attributes
playerTurn(int)

game methods
makeMove()
changePlayer()

game attributes
newRound / roundCount (int)

Player Moves Around the Board



a player takes a turn

what is involved with a player taking a turn?

- first iteration basic game play (without inclusion of payer winning a game/purchasing property/paying rent & developing an area in a field)

- a player will *ALWAYS* roll two dice (if a player will always use the dice, could this be an interface??)

- the value returned of the dice will be added to players current location and move player to new location
playerPosition(int)

from the returned dice value, how do we update the token/player location?

- how do we know a players location?

- a player or token could know the location of the player a token merely represents where the player is on the board. The player will more than likely know where they are on the board, token could be a name stored as an attribute in the Player class, in otherwords token is candidate to be a composition with the Player class

- The Game class needs to know the area the player is arriving on, and requests this information from the Board class, likewise the Board class communicates with the Area class to retrieve the area index. This suggests that Area is a candidate for composition with the Board class, and we also know that Board represents an ordered collection of areas that the game is played on.

Area attribute
index
name

dice methods
rollDice(Int/Integer)

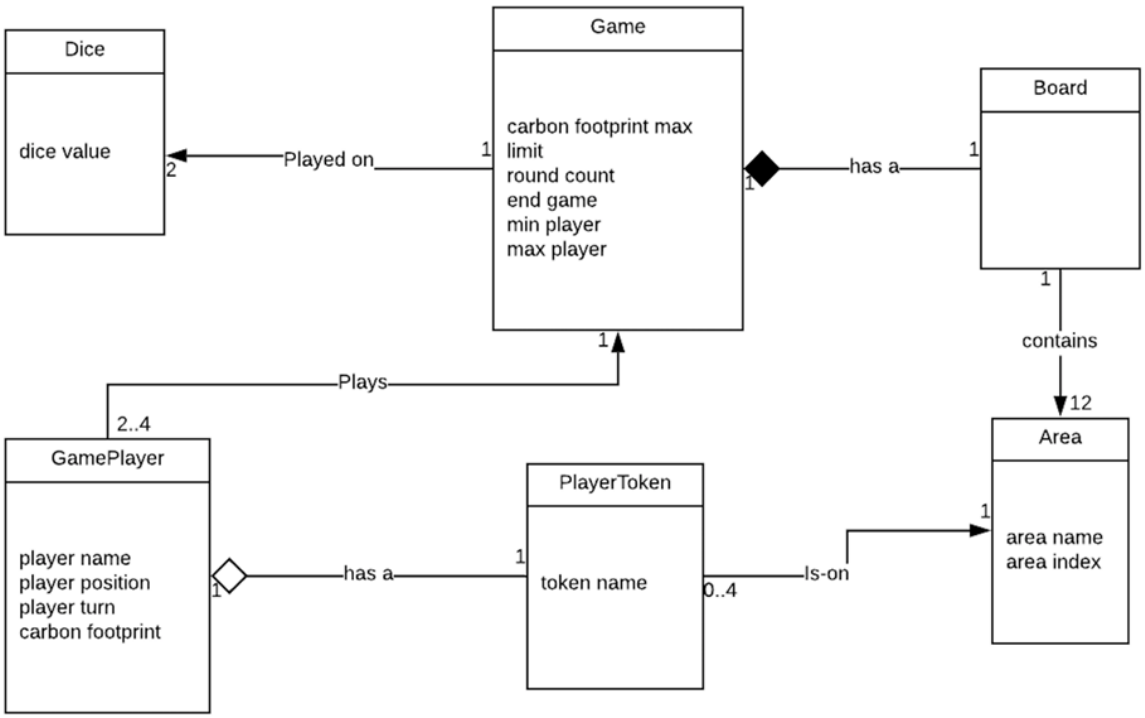
player attributes
playerPosition
playerToken

player methods
getPosition()
setPosition(newLocation)

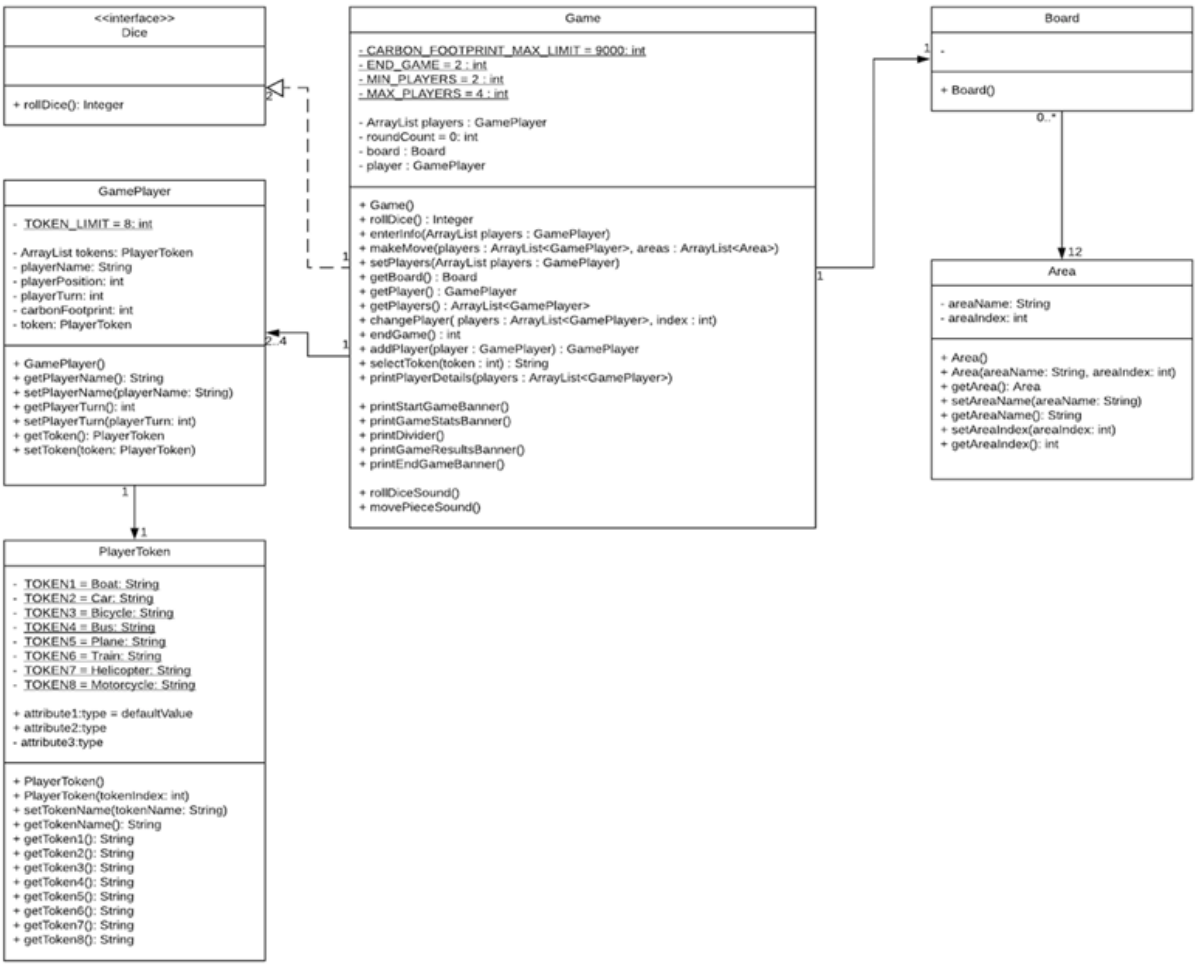
board methods
getPosition()

assumption here is the board doesn't move the token, the player moves the token (get and set methods interact with player and token object)

Domain Model Following Iteration 1

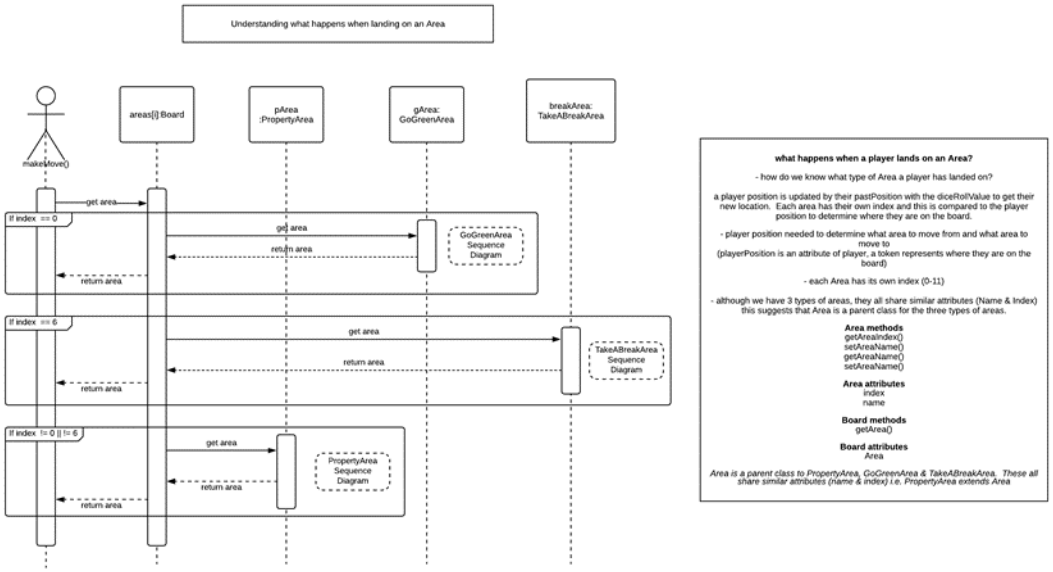


Iteration 1 UML Class Diagram

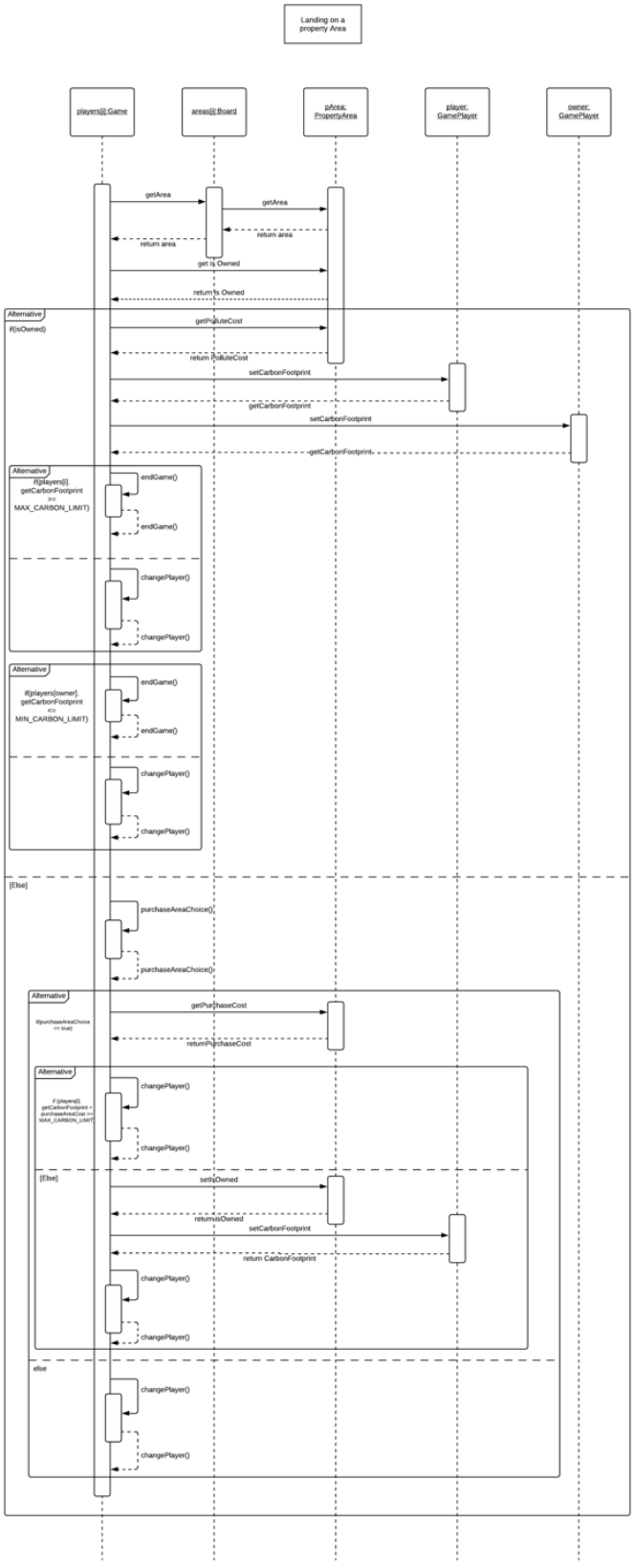


8.7.3 *Iteration 2*

Landing on an Area



Purchase Area & Pay Pollute Cost



what happens when a player lands on a property area?

- a property area can be owned
- a property area may not be owned
- depending on the ownership status of a property area will determine the actions that may occur if a player lands on a property area

if a player lands on a property area that is NOT owned
a player has the option to purchase that area

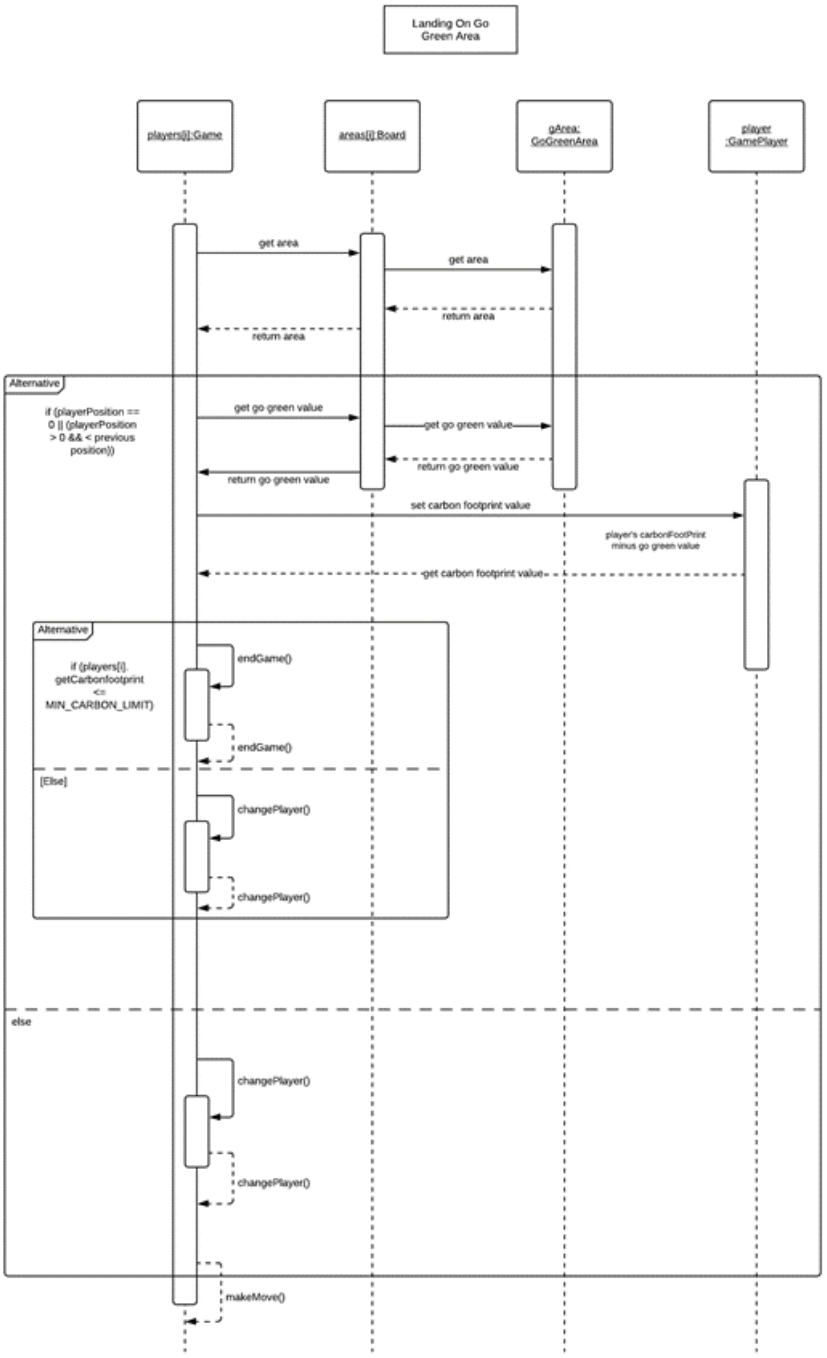
if a player lands on a property area that belongs to another player the player that lands on the area pays a pollute cost to the player that owns the area

Area methods (property Area/ not inherited specific to property area type)
purchaseArea()
payPolluteCost()
setIsOwned()
getIsOwned()
setPolluteCost()
getPolluteCost()

PropertyArea attributes (not inherited)
areaCost
polluteCost
isOwned

- it's clear that a propertyArea will have particular associated attributes as a propertyArea will have an attribute related to purchase and cost.
- an area could have an attribute owned or not? true or false? could this assign the player name/playerTurn to this attribute?

Go Green



what happens if a player lands on or passes the Go Green Area?

- the players carbonFootprint is reduced.

Go Green Attributes (inherited attributes)
areaName
areaIndex

Go Green Specific Attributes
GO_GREEN_VALUE = 600
Go Green value never changes, hence the Go Green value can be declared as a final

Go Green Method
getGoGreenValue()

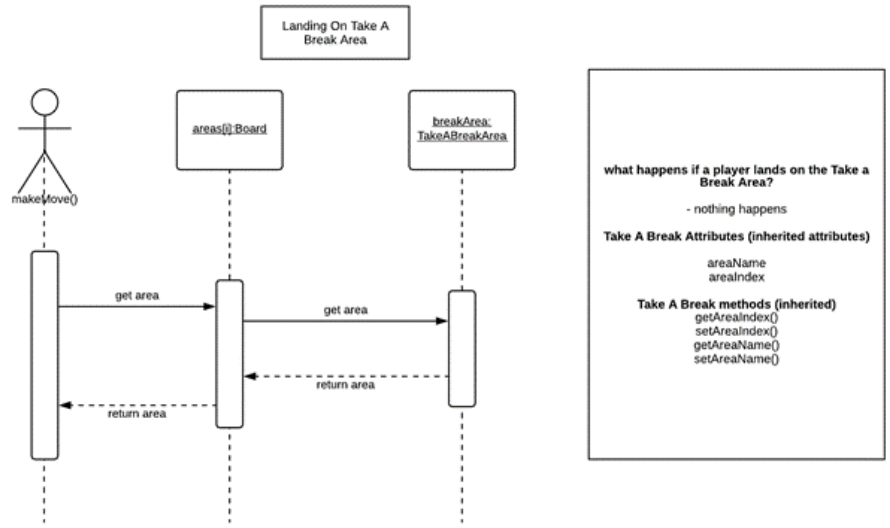
Go Green methods (inherited)
setName(), setIndex(), getName(), getIndex()

Player methods
setCarbonFootprint()

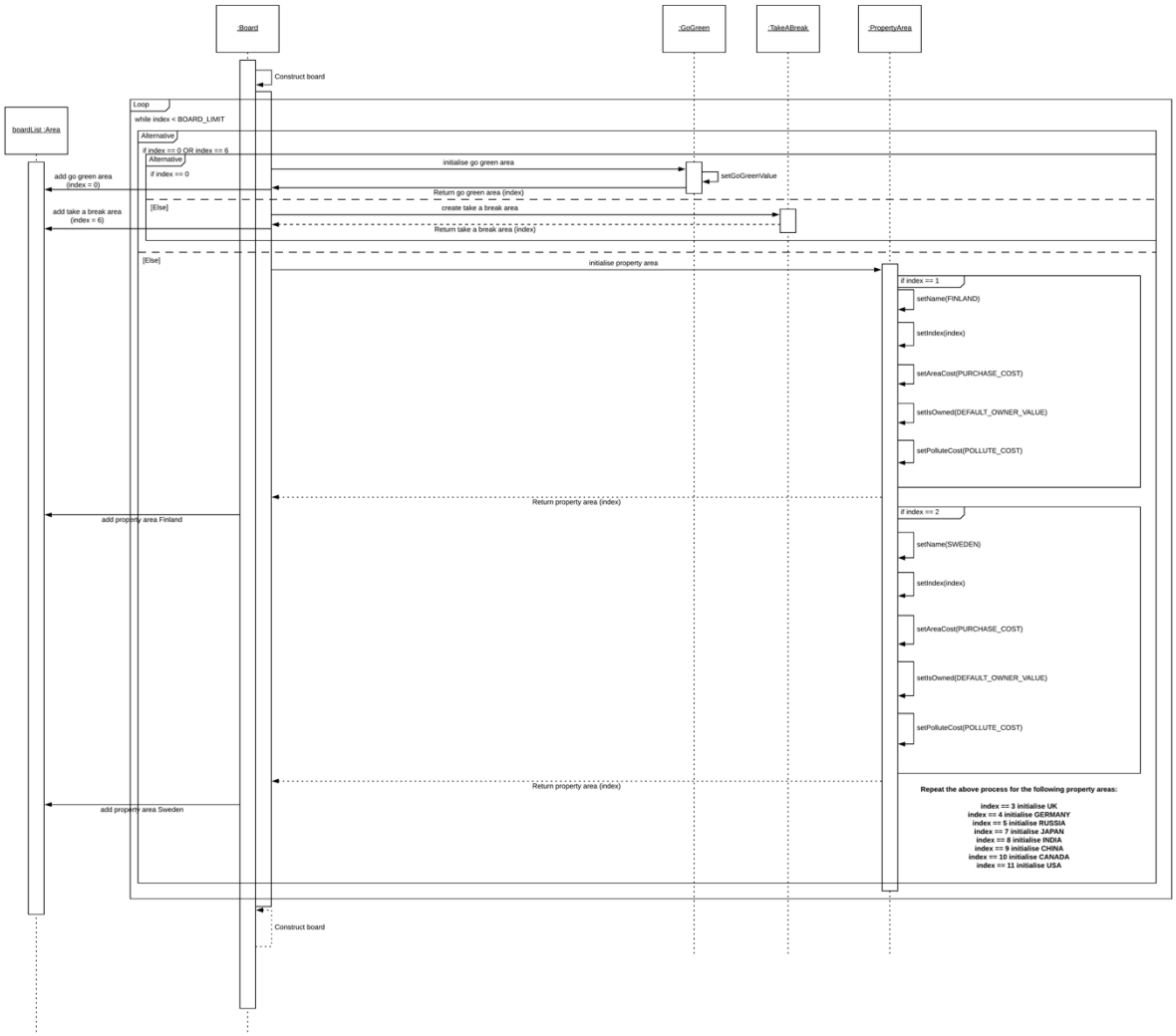
Game methods
endGame()
changePlayer()

Game attributes
MIN_CARBON_LIMIT = 0
this will can also be declared as a final as this value never changes
previousPosition
(this is a temporary value so may only be needed in the makeMove() method rather than being an attribute of GamePlayer or Game)

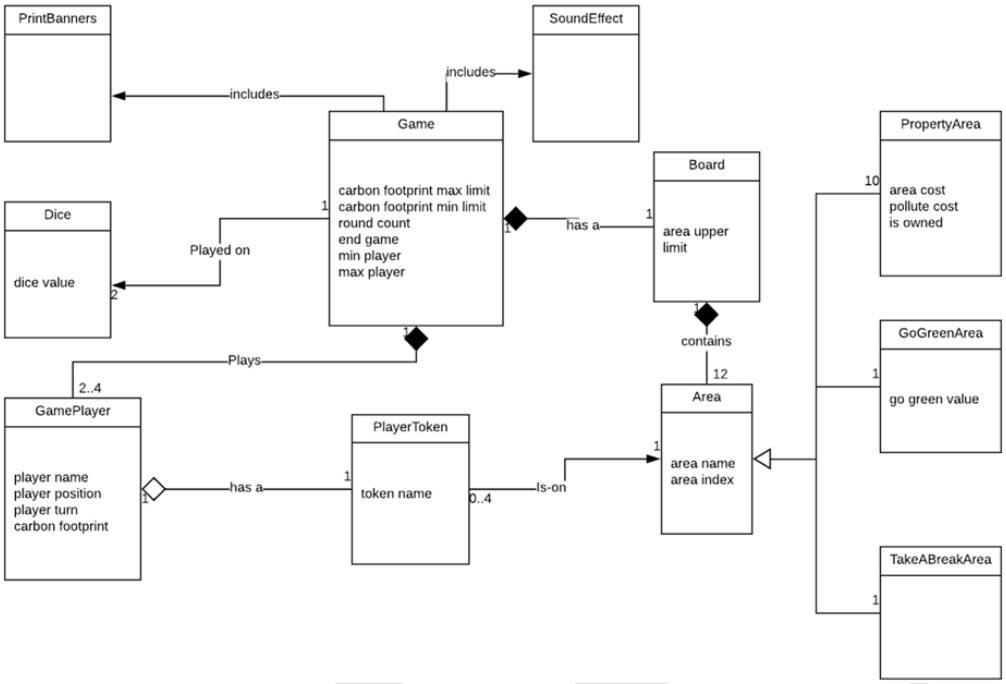
Take a Break Use Case (UC10)



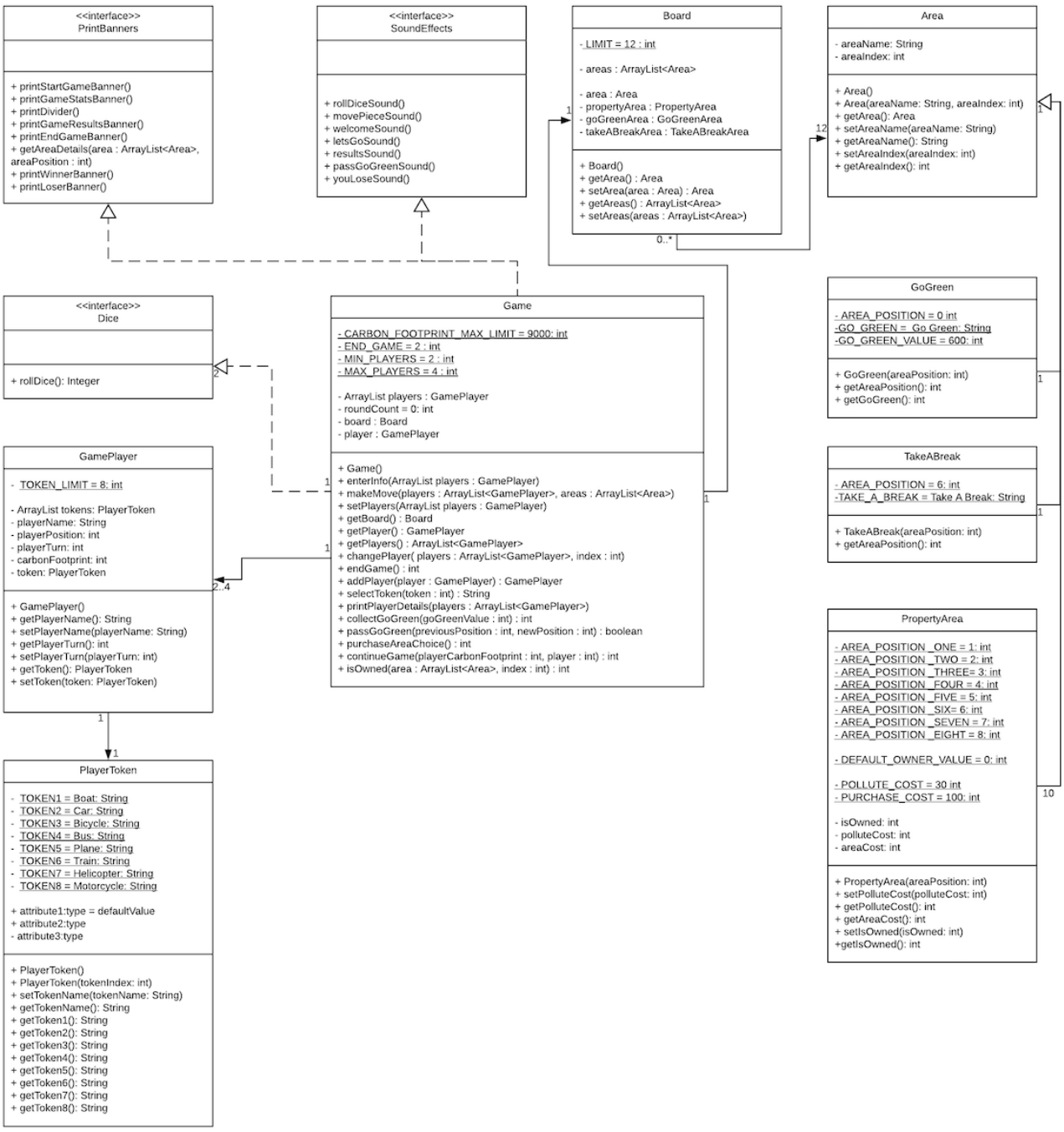
Board Creation – Board Class, GoGreen class, TakeABreak class & PropertyArea class



Domain Model Following Iteration 2

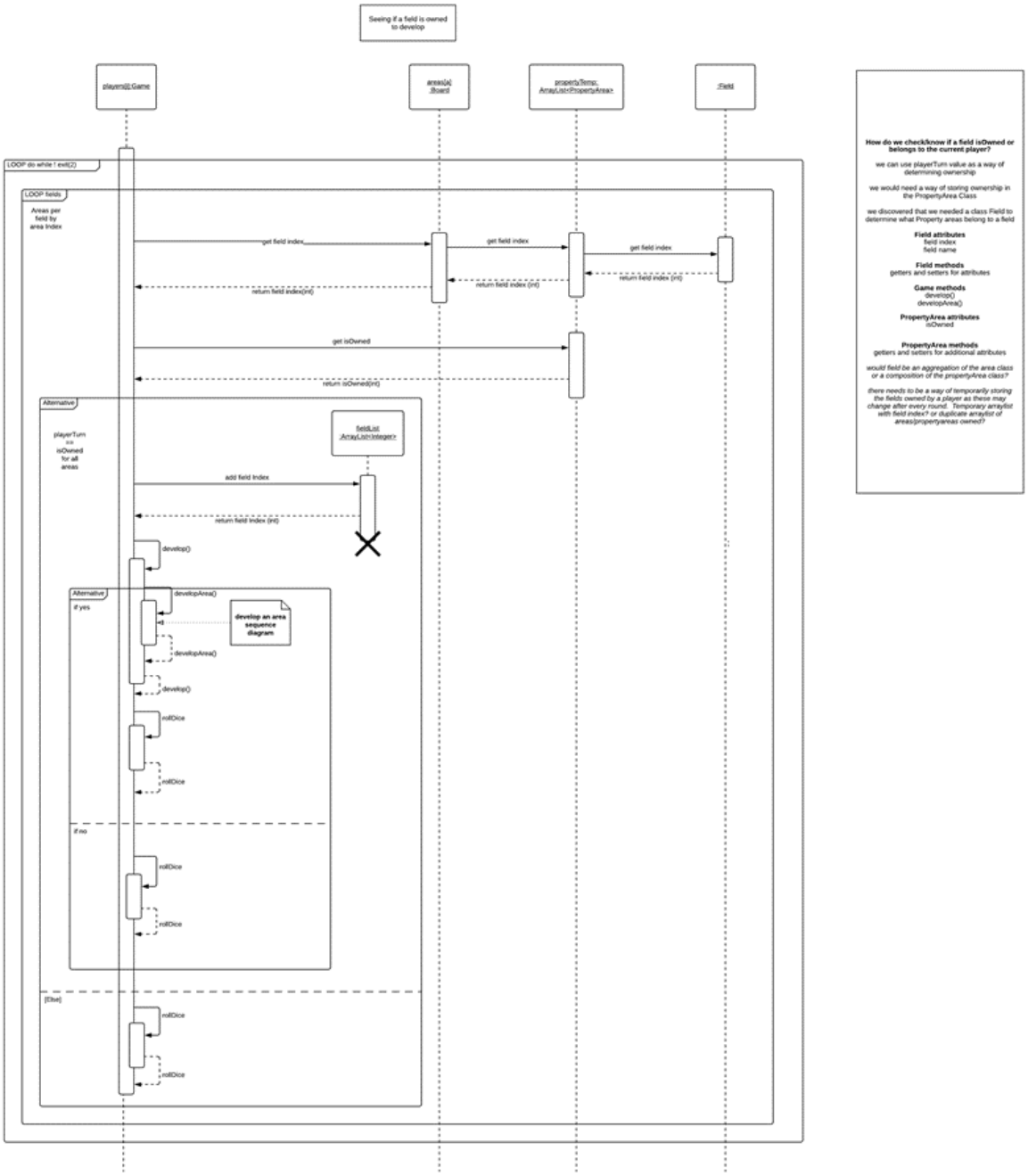


Iteration 2 UML Class Diagram

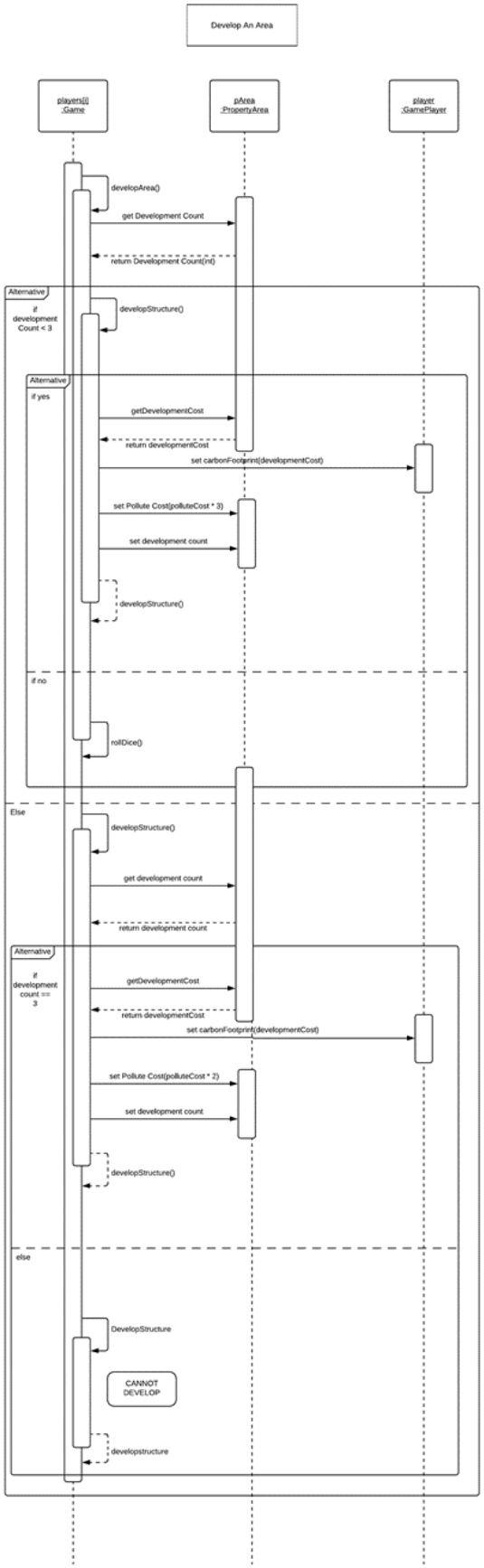


8.7.4 Iteration 3

Check if Field is owned



Develop Area in a Field



what does creating a development on an area look like?

- the player developing the square, their carbon footprint will be increased by the cost of the development (i.e. minor/major).

- the rent/(pollut cost) will need to be increased for the area to reflect the development. this will involve player owning the square and number of minors/majors on the square.

PropertyArea attributes
developmentCost
developmentCount

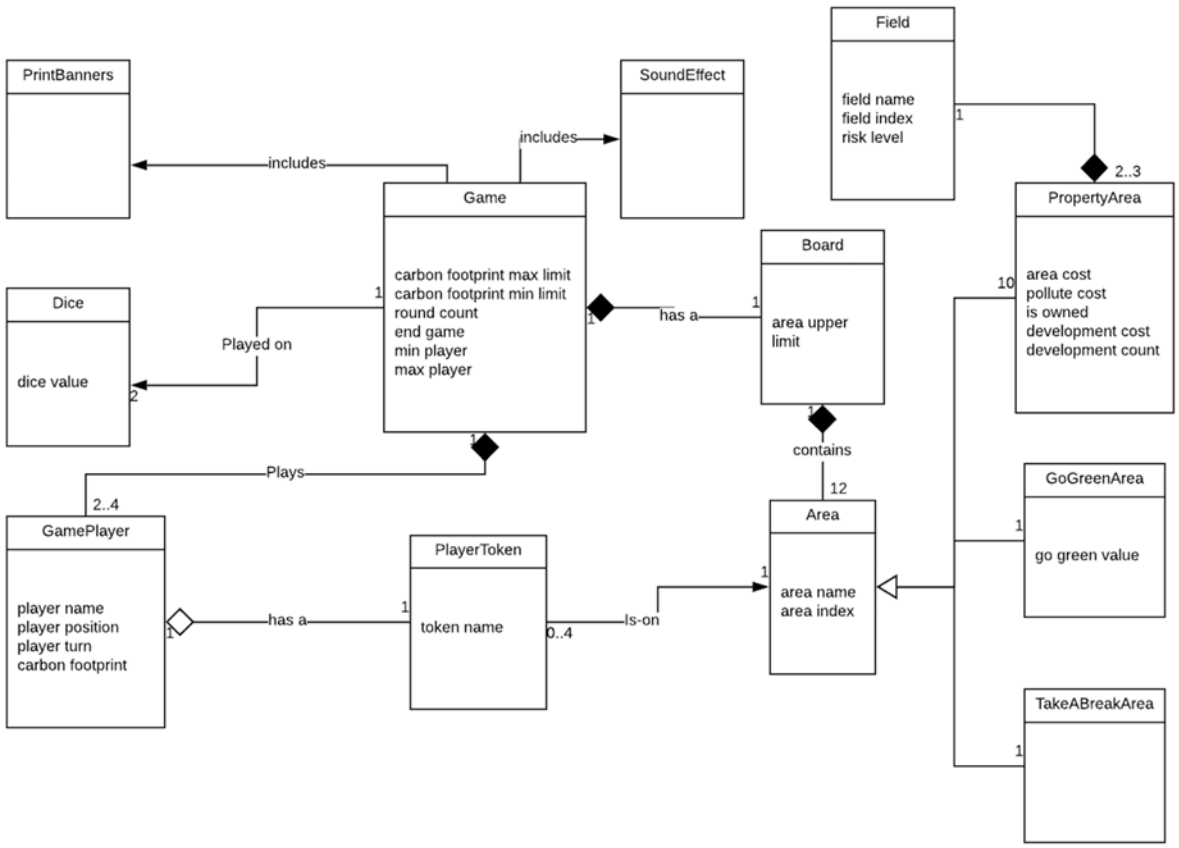
PropertyArearea methods
checkDevelopmentCount()
setPolluteCost()
createDevelopment()

Game methods
develop
developStructure

- the pollute cost to be worked out from initial rent cost, minimize need for static_final variables???
- we could have a static final attribute representing the rent cost and a count attribute holding how many developments the player has withnin the property square. The combination of these attribute values can be used to calculate the cost of rent. For example, a count of 1 indicates the player has developed 1 minor development and the static final value can be multiplied by 3, a count of 2 indicates the player own 2 minor developments, therefore the static final value can be multiplied by 3x3, and a count value of 3 suggests the static final value can be multiplied by 3x3x3. A count of 4 indicates the player owns a majot development, the full capacity, therefore the static final value is multiplied by (3x3x3) x 2. if count == 4 its no longer possible for user to develop on the area.

- when we look at this, it appears there isnt a need for the development class/object as an area will know all details about itself

Domain Model Following Iteration 3



The following describes attributes needed to create an instance of the Field class:

- fieldname: Stores the name for one of four possible fields: can be Northern Europe, Central Europe, Asia or North America;
- fieldIndex: Stores the value representing the position of the field: 1, 2, 3 or 4
- riskLevel: Holds the value needed to assign the different costs associated with developing structures and paying a pollution cost for property areas:
    - one field has the highest cost for developing structures on areas within the field and the highest cost associated with paying a pollution cost;
    - another field has the least cost for developing structures on areas within the field and the least cost associated with paying a pollution cost;
    - the remaining two fields have the same associated cost.

Can be used as a multiplying factor when calculating the cost of developing structures and paying a pollution cost for particular property areas.

The following describes methods needed to instantiate a Field object:

- setFieldName(fieldName): Assign the name of the field by passing Northern Europe, Central Europe, Asia or North America as a parameter;

- getFieldName(): Return the name of the field;
- setFieldIndex(index): Assign the index associated with the field: can be 1, 2, 3 or 4;
- getFieldIndex(): Return the field index;
- setRiskLevel(riskLevel): Assign the risk associated with the field, riskLevel is passed as a parameter;
- getRiskLevel(): Return the risk level associated with the field.

The following describes attributes needed to create an instance of the PropertyArea class:

- RISK_LEVEL_1 = 1, RISK_LEVEL_2 = 2, RISK_LEVEL_3 = 3: Final attributes representing the 3 levels of cost associated with developing structures on property areas within particular fields. i.e. one field is the most costly to develop, one field is the least costly to develop and the remaining two fields have the same cost to develop.
- BUILD_COST = 100: Static final that stores the default cost of developing a structure.
- areaCost = PURCHASE_COST * RISK_LEVEL: The areaCost attribute is required to hold the cost of purchasing the area, the default cost of purchasing an area is multiplied by the value of the risk level associated with the field the area is related.
- developmentCount: Stores the values 0, 1, 2, 3, or 4
  - 0 indicates that there is currently no minor/major structures developed on the area;
  - 1 indicates there is 1 minor structure developed on the area;
  - 2 indicates there is 2 minor structures developed on the area;
  - 3 indicates there is 3 minor structures developed on the area;
  - 4 indicates there is 1 major and no minor structures developed on the area.
- isOwned: The value for playerTurn (holds a number representing the order of play) is assigned to this attribute when the area is purchased, the value of this attribute is initialised with the default value of 0.
- buildPropertyCost: Stores the cost of developing a structure on a property area, can be initialised with a default build value.
- Field: References an instance of the Field class that is needed to create an instance of the PropertryArea class

The following describes methods needed to instantiate a property area object:

- Field(fieldName, fieldIndex, riskLevel): Constructs a new field object and accepts the field name, index and risk level parameters as arguments, communicates with the Field class when invoked.
- getBuildPropertyCost(): Return the cost of developing a structure on the property area
- setBuildPropertyCost(BUILD_COST * RISK_LEVEL): Assign the cost of developing a structure on a property area, the value obtained from multiplying the default build cost by the risk level of the associated field object is passed as a parameter.

- setDevelopmentCount(developmentCount): Assign the number of structures currently developed on the area.
- getDevelopmentCount(): Return the number of developments on the area:
  - Returning 0 implies there is currently no structures developed on the area;
  - Returning 1 implies there is currently 1 minor structure on the area;
  - Returning 2 implies there is currently 2 minor structures on the area;
  - Returning 3 implies there is current 3 minor structures on the area;
  - Returning 4 implies there us currently 1 major structure on the area.
- getAreaCost(): Return the cost associated with a property area.
- setPolluteCost(POLLUTE_COST * RISK_LEVEL): Assign the pollution cost for a property area, the value obtained from multiplying the default pollute cost by the risk level of the associated field object is passed as a parameter.
- getPolluteCost(): Return the cost associated with landing on a property area owned by another player and having your carbon footprint value increased.
- setIsOwned(isOwned): Assign the number associated with the player who has purchased the property area.
- getIsOwned(): Return the number of the player owning the current property area.
- getFieldName() (communicates with the Field class): Return the name of the field.
- setFieldIndex(index) (communicates with the Field class): Assign the index associated with the field.
- getFieldIndex() (communicates with the Field class): Return the field index.
- setRiskLevel(riskLevel) (communicates with the Field class): Assign the risk associated with the field.

getRiskLevel() (communicates with the Field class): Return the risk level associated with the field.

Iteration 3 UML Class Diagram

Presented in Figure 4.1 in Section 4 of the main report.