SAVE OUR PLANET

CSC7053

Michael Carey-Small, David Grech, Eoin Henry, Padraig Kearney

Introduction and Game Narrative [DG]

Scientists predict in less than 20 years increased atmospheric carbon will cause irreversible and catastrophic global warming, triggering a global mass extinction event: with the potential loss of 80% of life on the planet. A new carbon capture technology was discovered that can reduce atmospheric carbon in excess of 70%. The world superpowers are too busy fighting over dwindling oil reserves to invest in this new technology. Something needs to be done.

Around the globe groups of non-profit organisations are joining together to raise the money to build the devices. In order to speed up the development process, a challenge has been set by a large group of investors to be the first non-profit to raise the resources to build the technology. Once a total of 20,000 greencoin is raised the devices can be built! Greencoin is a brand new innovate cryptocurrency, which derives its value from environmental projects.

The investors provide an initial 5,000 greencoin resources to each player. To help raise additional funding the investors have identified ten greenregion areas across the planet, all with potential developments which require funding. Players are encouraged to visit these areas in order to assess their development potential and invest their greencoin resources to expand these projects. Players must own all of the development sites in a given region, before they can develop one project in that region. Players are encouraged to develop each site as they wish but achieving a major stage development is desirable as it will provide the greatest benefit for the planet.

As soon as a player has raised 20,000 greencoin the game is ended, and a winner is declared. Afterwards the carbon capture devices can be built, and the planet is saved. If a player visits an area owned by another player, they contribute to that project in order to help their fellow eco warrior and in doing so helping to save our planet. Everyone must work together to save our planet! Every time a player passes the home square, they collect 500 greencoin additional resources from investors in the project.

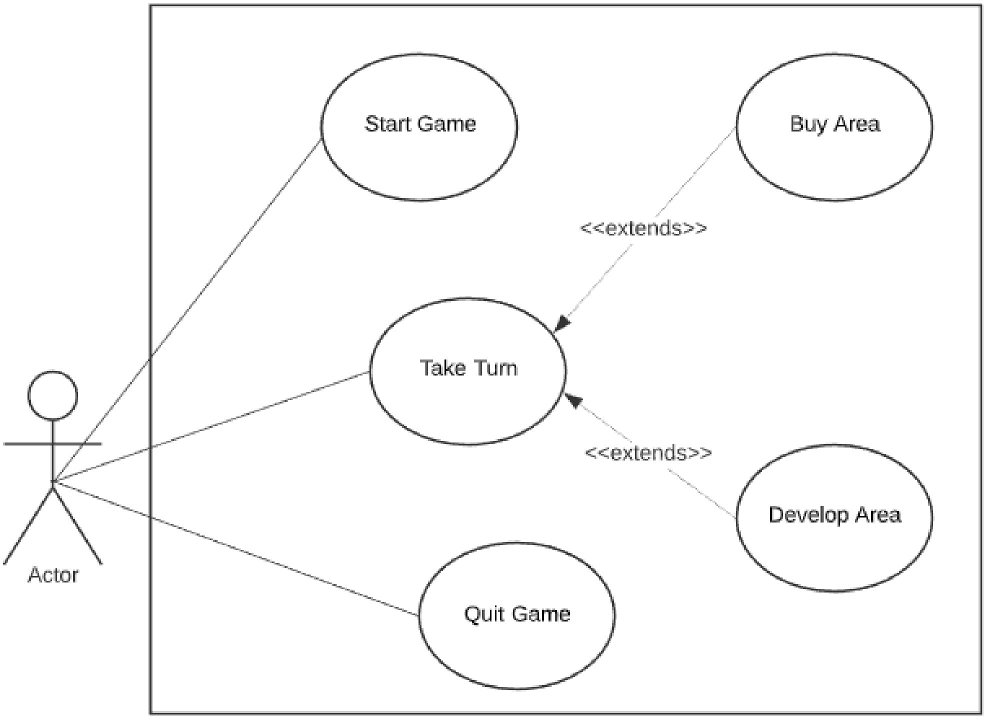
If one player decides to finish the challenge, then investors would unfavourably on the project and further funding stops. The challenge cannot continue with no further investment, in this case the only hope is for all players to combine all their green coin to raise the funds as a group. If the green coin total for all players is 20,000 or more our planet is saved and the group will go down in history for saving our planet! This generation and future generations will live in peace, working together for a sustainable future. If the total is below 20,000 greencoin time has run out and the inevitable end is coming. This was the final hope for the future of humanity and most of life on the planet. All that is left to do now is enjoy our final days on this once magnificent planet.

Our full detailed game guide including details of each square is available at the start of the appendix section.

Requirements Analysis [MCS]

According to Grady Booch, James Rumbaugh and Ivar Jacobson when writing the unified modelling language a use case is “A set of sequences of actions, including variants that a system performs to yield an observable result of value to an actor. Graphically a use case is rendered as an ellipse.” Use cases describe the system behaviour that is visible to a user or to another system, these use cases are initiated by users or systems which are called actors. These actors typically represent a role that a human, a hardware device or another system plays within a system. It is with keeping this in mind that we were able to develop the use cases for our Save Our Planet board game. When creating the use case diagrams we thought about the high level actions of the game and analysed what the system would actually do. The main components within a use case is firstly the main objective of the use case and what it is trying to achieve when carrying out the steps involved in the main flow. Also there is a section on Preconditions within the use case which need to be fulfilled prior to the undergoing of the particular use case. The Main Flow of the use case descriptions illustrates the sequence of steps that are carried out between the actor and the system for an ideal complete condition. Next we have the alternative flow section of the use case this section highlights what else can could happen when this use case is carried out. After this we have the post condition section this part of the use case highlights different factors that are initiated as a result of that particular use case being carried out. Finally we have the extension points, an extension point identifies the point in the base use case where the behaviour of an extension use case can be inserted. The extension point is specified for a base use case and is referenced by an extend relationship between the base use case and the extension use case. With this in mind we will begin to look at the use cases descriptions and diagrams we have created.

Use Case Diagram [MCS]



Firstly, we identified the player as the main actor of our system, the system at present would only have a single actor as the game is quite a simple program. The actor sits outside of the box and the lines from the actor shows how they would interact with our system. The UML case diagram itself is designed to show high level choice and further details of how these are realised are shown in later sections via our UML Sequence diagram. From our analysis we determined several choices the player would like to make. The choice to start the game, take their turn and quit game. With optional extension points from the take turn ellipse indicating that a player can purchase an area when they have visited. In addition to an option to develop an already owned area to receive better contributions when another player visits.

Further details about the ellipses may be found in our use case description section.

Use Case Descriptions [MCS]

Flow of Events for the Start Game Use Case	
Objective	For Players to start the game Players to start the game by registering their details. No duplicate player names are allowed.
Precondition	There must be a minimum of 2 and a maximum of 4 players in order to start the game successfully
Main Flow	1. Players start the game 2. Players enter their names as requested by the system
Alternative Flows	1a. Player quits game 2a. Player enters invalid name, duplicate details used by existing player. Requests new details
Post-condition	2-4 players have started the game
Extension Points	-

The player would be required to have the option to start the game should they wish. This occurs at the start of the game, in order to register the players to partake in the game. To do this the players must enter the total number of players which must be between two and four and entered in a numeric value, otherwise the system throws an error. The system asks the players to enter the number of players twice more before aborting the game. An extension point occurs where each of the players' names that are playing the game are checked ensuring that no two players names are the same. Providing the above conditions have been completed correctly and successfully the game will start. The post condition is that two to four players have now successfully registered and can now get ready to initialise the game by taking a turn.

Flow of Events for the Take Turn Use Case	
Objective	A player takes a turn
Precondition	Start Game Use Case
Main Flow	1. The player starts their turn 2. The player rolls the dice and is moved the rolled number of squares on the board 3. The player lands on a square
Alternative Flows	1a. Player quits game 2a. If the player completes a lap of board, gets an additional 500 greencoin 3a. Area is owned, contribution to another player 3b. Area is available for investment, see buy area use case 3c. Player unable to pay contribution, game ends
Post-condition	Player has completed their turn, player will start next turn from their current position
Extension Points	1. Buy Area Use Case 2. Develop Area Use Case

Next we have the take turn use case description, arguably the most important use case within the game, the main objective of this use case is enabling a player to take their turn. Initially the player is offered the choice to roll the virtual dice, if they choose no the system prompts them to ask if they would like to quit the game, if the player selects yes the game ends, however if the select no the game rolls the dice and moves the player.

Following this move the player has several options depending on the number they rolled and the status of the area on which they visit, if they pass the home area they are rewarded with a bonus amount of Greencoin from investors. If they visit on an unowned investment, they are given the option to buy that development providing they have the sufficient funds to do so. However, if that area is owned by any of the other players in the game, then the players will contribute to the development of that area should they visit. This is providing they have sufficient funds, however, should they not the game shall end. At the end of the turn the player may be given the option to develop any areas that they have. Provided that they own all areas within a region which will increase the contribution should another player visit that area. Following the completion of this use case the players turn ends.

Flow of Events for the Buy Area Use Case	
Objective	Player purchases area If a player lands on an unowned area, they may purchase the area for initial fee.
Precondition	Take turn use case Player has enough resources to fund purchase of the area The area that they landed on is currently not owned by another player
Main Flow	1. Player gives resources in exchange for the area 2. Player becomes the registered owner
Alternative Flows	1a. Player chooses not to buy area
Post-condition	Player owns area, can receive contributions when other players land on area
Extension Points	-

One of the extends options to the take turn use case is the optional activity for the player to buy an area. It extends from the take turn use case and allows, if the area is available for purchase, the option to buy an area in the given region. The preconditions for this use case are that the players must have first taken their turn and have enough Greencoin in order to purchase the area. If these conditions met the player exchanges Greencoin in exchange for the area they become the registered owner of this area and from that point they will receive any contributions where another player visits it for the duration of the game. An alternative flow occurs if the player chooses not to purchase the area, and simply continue with their turn.

Another extends use case is the develop area use case description, this use case describes what happens when a player chooses to develop one of their owned areas. This use case also extends from the take turn use case and occurs just before the turn ends. A player doesn’t need to land on that area in order to develop it however, they do need to own all areas within that region. In order to develop then the player must have sufficient greencoin in order to pay for the development. If these requirements are met the player is given the option to develop their area and the development cost is deducted from their greencoin. An alternative flow occurs when a player has been successful in upgrading the development three times the player will be given the option for a major development of the area. Post conditions are that after upgrading the area the player can now receive greater contributions once another player has visited their area.

Flow of Events for the Develop Area Use Case	
Objective	To develop an owned area A player may further develop an owned area on their turn, no need to be on that area, one development allowed per turn
Precondition	Take turn use case Player has enough greencoin to develop All areas in region owned by player
Main Flow	1. Player pays to develop area
Alternative Flows	1a. Major development option available (owned development is at stage 3)
Post-condition	Player has developed his owned area, can receive increased contributions
Extension points	-

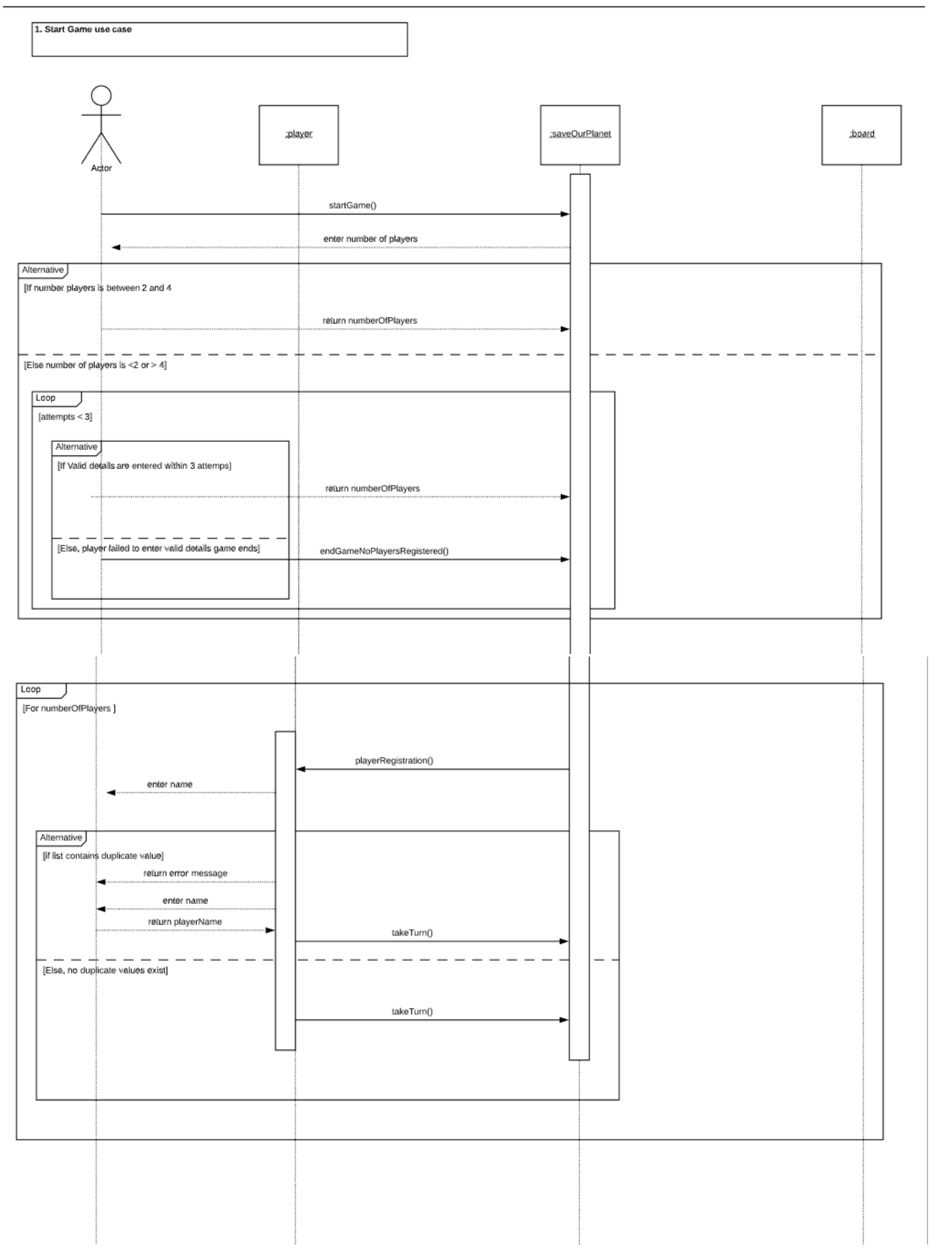
Flow of Events for the Quit Game Use Case	
Objective	A player wishes to quit game and therefore the game ends for all players
Precondition	It must be the start of the players turn Alternatively, this use case may be invoked once player hasn't got enough resources to fulfil obligations
Main Flow	1. Player chooses to quit game 2. The game ends, as a result
Alternative Flows	1a. Player decided to continue
Post-condition	Funds are tallied up, winner declared
Extensions points	-

Finally, we have the quit game use case description, the objective of which is for the player to quit the game and in doing so the player ends the game for all players. For the player to do this it must be the start of the players turn, a player cannot end the game unless it is their turn. Although, alternatively if any of the players run out of greencoin at any stage during the game the game will also quit. As part of the main flow the player will be asked are they sure, and if so the game will end. An alternative flow occurs where a player changes their mind and decides to continue. The game will end and all of the players funds are tallied up and a winner of the game will be announced providing that there is one.

Realisation [PK]

The use case descriptions are a high level detail of how a player interacts with the system. The UML sequence diagram shows method calls between the objects created in our game, and the sequence of events that leads to the successful implementation of these ellipses identified earlier.

> **<u>For clarrification purposes</u>**
>
> board is an object of class Square. Board stores information in an arraylist about each of our development areas identified by the investors
>
> player is an object of class Player.
>
> saveOurPlanet (Abbreviated SOP) is an object of class GameController and is initilised in the main method. It uses methods from within the GameController class in order to create the game envirnoment.
>
> The actor on the left of the sheet represents the interaction via the console window, where a user's input is required for decision making purposes etc.
>
> In the case of requiring input from the user the values yes or shorthand y are excepted in addition to no and n.



CASE 1;

After the user has confirmed that they wish to start the game, the user selects start game method from within the SOP object. The user is then required to provide the number of players taking part in the game. Values from two to four only will be accepted as per the client's specification. Values outside this range will trigger an exception message to the console and the system will request the user to enter the details again. This will loop for a maximum of three attempts, after which the user will be presented with an exit game message from an endGameNoPlayersRegistered method within the SOP object. Upon successful completion of this stage the number of players will be stored in a variable used to create the number of player objects held within the game.

The user is then required to register their username details for the game. This is done via the PlayerRegistration method from the player class. The username must be between a minimum of one character and twenty characters. All values including special characters are accepted as a player's name, but a string outside of the allowable ranges will cause the system to output an exception message and require details to be entered again. Also, duplicate values must not be entered eg same username as another player, an occurrence of this will send a message to the console window requiring the user enters different details must be entered. Once all players' details are provided, this stage is deemed complete and the game prepares to move forward to the next case. As the game at present is relatively simple there is not great requirements for storing of player information. At the registration stage the players are also provided with a starting resource balance and a start position at the default index of the "home" area.

CASE 2;

The game loops for the specified number of times e.g. twenty laps = twenty years. Each lap of the board in our game represents a year of the players traveling the globe to raise greencoin in order to save our planet. The game also loops over the player's arraylist for the total number of registered players. The players select take turn, firstly they confirm that the wish to proceed. At the start of each players turn their details e.g. greencoin balance, position and name are displayed to the screen. If they select any option apart from y, they are deemed to wish to quit the game method is therefore the endGame method is called. Upon consenting to take their turn the dice method is called from the GameController and the number "rolled" on the pair of dices are outputted to screen. The player is moved to their new position on the board object by the movePlayer method which takes in a value from the rollDice method which simulates a pair of dices. The players new position is then set, within the moveDice method this is done by using the return value from the dice roll and using the setPosition method from within the player object. All players start at the "home area" by default. Once the position is set the players are moved on the board and the board object uses the players position, via the getPosition method, to correctly identify the players position on the board object and display relevant options based on the players position. The board objects are stored in an indexed array list, so the position can easily be identified, in addition to attributes based on the position inputs from the player.

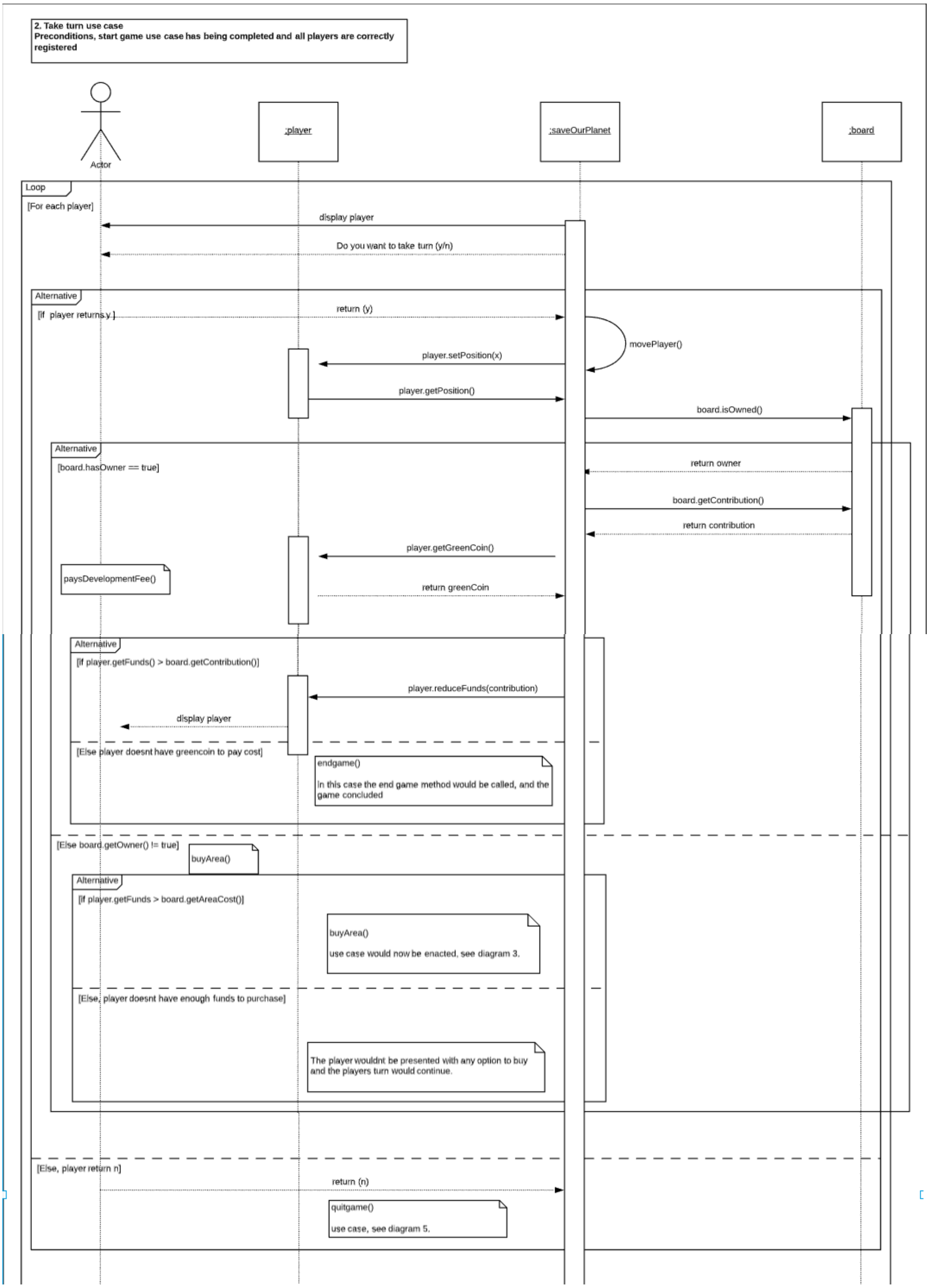
The first check is if the board object is owned (not applicable for the "home" or "Travel cancelled" squares, these two unownable board objects have limited attributes and are created using a different constructor than purchasable board objects. Should the board object be owned provide a true value the player would be obliged to contribute to the development of that players asset, all in the spirit of helping develop sustainable environmental solutions. Also at this stage within the system, there is a check to ensure the owner of the area is not the active player themselves, not shown on the diagram. If they are a brief message is displayed on the console. The UML sequence diagram assumes the player is not the registered area owner. A check is done to ensure the player has enough resources to complete the transaction, this is achieved by comparing the player object getGreenCoin balance to the relevant board object getContribution method. Should they have insufficient resources the endgame method will be called from within the SOP object and GameController class which will be discussed at the end of this diagram. The subsequent amount is retrieved from the board object and the players green coin account is deducted via a method from with the player object called reduceBalance. A method (not displayed on the UML sequence diagram called payAnotherPlayer ensures the correct party receives the contribution. Also, a check is done beforehand to ensure that the registered owner of the board object is not the player themselves. Amended balances for all affected parties are shown to the console window using a toString method.

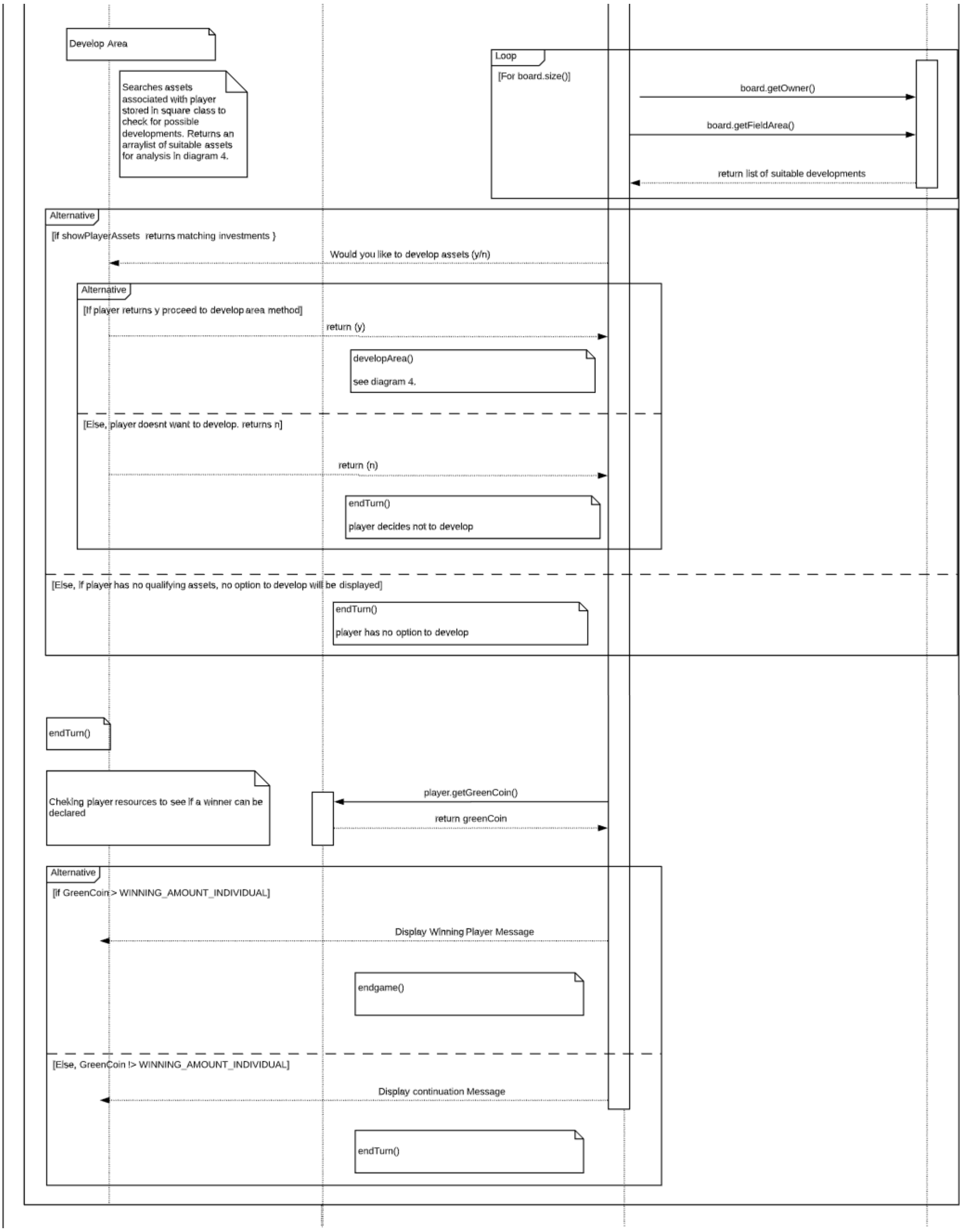
In the case of the owned check returning a false value the player now may be provided with the option to buy subject to having available greencoin resources, if not they will be deemed unable to purchase and their turn will continue onto the following stage. If they pass the resources check they will be presented with the option to buyArea. If they selected (y), then the buyArea use case will be trigged (more details of that process in diagram 3 and description). If they select (n) or any other input, then their turn will continue onto the next stage.

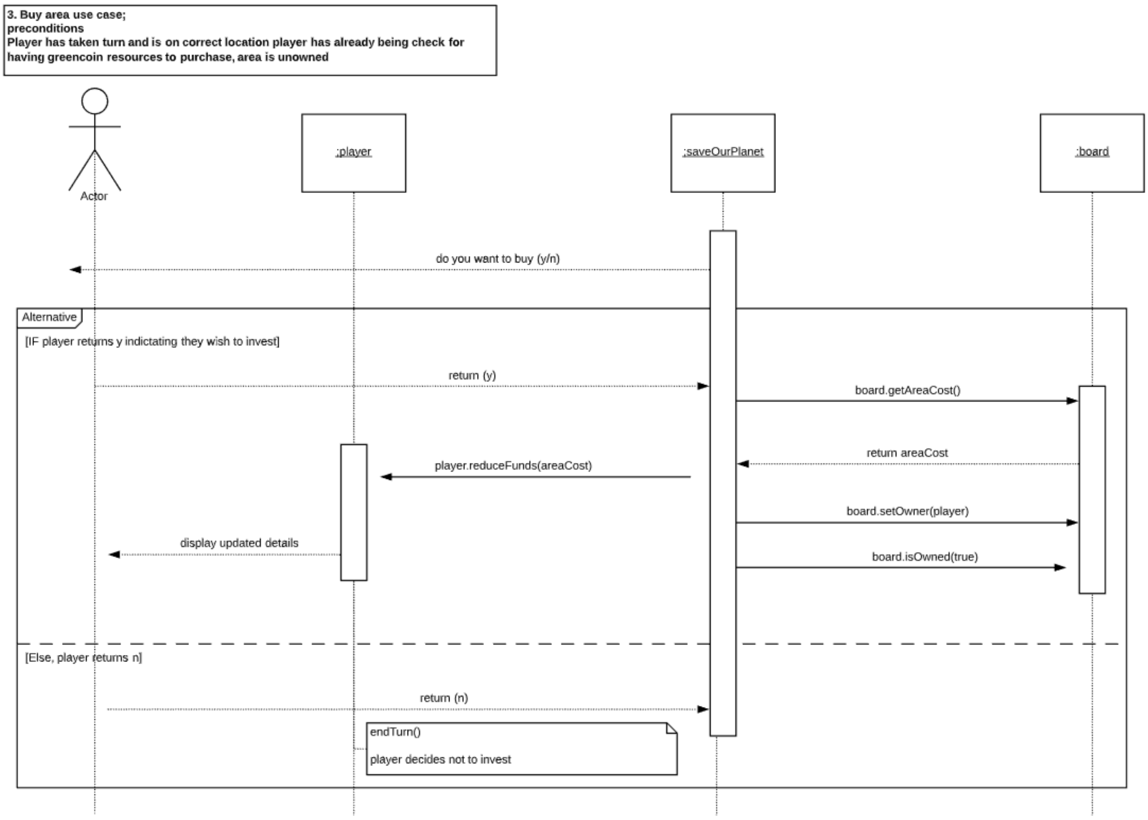
During the game the players are given the opportunity to invest in different global projects. Should they collect several in a similar region they may be presented with the option to expand their reach in that area by upgrading their areas to a higher development stage. Firstly, a method called showPlayerAssets quickly matches the player to registered owners in the board object using the getOwner method and comparing to the player object stored. Next the getRegionNames are checked to see if the correct number of areas in a given region are controlled by the player. This is done by the checkAreaForMatchingInvestments method. Shown on the right-hand side of the diagram for illustration purposes. If successful, the player is given the option to develop via the console window. If they return y then the develop area use case is started, see diagram 4 for more details. However, if they select no, then the game is deemed to reach the end of their turn and the endTurn method is called to be described below.

At the very end of the players turn the game determines if they have accumulated enough resources at this point to be declared the “winner”. The player object getGreenCoin method is called and value return for comparison. Should they exceed the threshold the game will finish with a message to the console window congratulating the player on completing the game and leading the battle against climate change, once the system knows the player has exceeded the threshold this is done by using the endgame method. If not a message of encourage is shown to the player, in addition to their current status. At the end of the for loop for a full complement of players, an up to date status of each of the players current standings is displayed in the console window in addition to owned board objects and status related to each, showAllCurrentPlayerAssets. The cumulative number of laps is also counted and displayed to the user, stating how many “years” they have left to save our planet. If during this turn the winning amount is not exceeded by any of the players the game continues to loop through the player list until the game is either won by a player or any player activates the quitGame use case at the beginning or during their turn, which will be further discussed in the final paragraph.

Should the total number of years be exceeded and there isn’t a clear winner the endgame method calculates a cumulative value of the total player resources and output each players end status to the console window, as we must all work together in order to combat challenges facing our planet. In the case where the group has exceeded threshold for the carbon capture device has being reached the congratulatory group message is displayed in addition to each player personal contribution. If not successful, another message is displayed commiserating with the group.


2. Take turn use case
Preconditions, start game use case has being completed and all players are correctly registered
Actor
:player
:saveOurPlanet
:board
Loop
[For each player]
display player
Do you want to take turn (y/n)
Alternative
[If player returns y]
return (y)
movePlayer()
player.setPosition(x)
player.getPosition()
board.isOwned()
Alternative
[board.hasOwner == true]
return owner
board.getContribution()
return contribution
player.getGreenCoin()
paysDevelopmentFee()
return greenCoin
Alternative
[If player.getFunds() > board.getContribution()]
player.reduceFunds(contribution)
display player
[Else player doesnt have greencoin to pay cost]
endgame()
In this case the end game method would be called, and the game concluded
[Else board.getOwner() != true]
buyArea()
Alternative
[If player.getFunds > board.getAreaCost()]
buyArea()
use case would now be enacted, see diagram 3.
[Else, player doesnt have enough funds to purchase]
The player wouldnt be presented with any option to buy and the players turn would continue.
[Else, player return n]
return (n)
quitgame()
use case, see diagram 5.

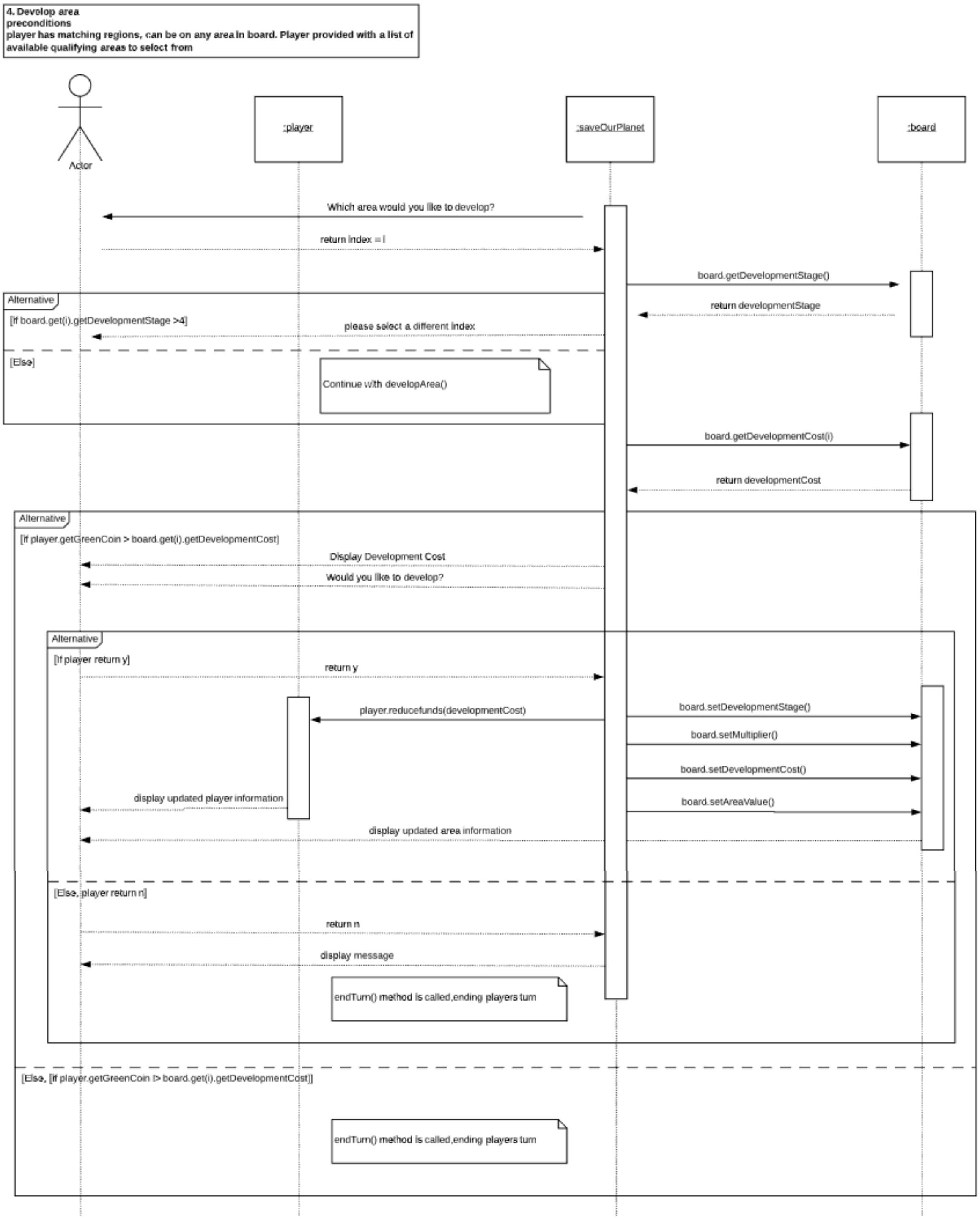

Develop Area
Searches assets associated with player stored in square class to check for possible developments. Returns an arraylist of suitable assets for analysis in diagram 4.
Loop
[For board.size()]
board.getOwner()
board.getFieldArea()
return list of suitable developments
Alternative
[If showPlayerAssets returns matching investments }
Would you like to develop assets (y/n)
Alternative
[If player returns y proceed to develop area method]
return (y)
developArea()
see diagram 4.
[Else, player doesnt want to develop. returns n]
return (n)
endTurn()
player decides not to develop
[Else, If player has no qualifying assets, no option to develop will be displayed]
endTurn()
player has no option to develop
endTurn()
Cheking player resources to see if a winner can be declared
player.getGreenCoin()
return greenCoin
Alternative
[If GreenCoin > WINNING_AMOUNT_INDIVIDUAL]
Display Winning Player Message
endgame()
[Else, GreenCoin !> WINNING_AMOUNT_INDIVIDUAL]
Display continuation Message
endTurn()



CASE 3;

For the purpose of this diagram it is assumed that the player has sufficient funds available in order to buy the area. It's also assumed that the check ownership criteria has been met.

The player is presented the area details and with the option to buy via the console window. Should the player return yes(y) the system runs several processes. Firstly, the board areaCost is received from the board object, and returned. The method reducePlayersFunds is called from the player object and reduces the resources available to the player by the amount returned from the board object. The board object next sets the owner as the player object by calling the setPlayer object and passing the player details. Next to be updated is the isOwned attribute from the default position of false to true. This will be used in future checks to see if an area is owned or not. An updated status of the board object, now showing the player as the registered owner is displayed to the console window in addition to providing an updated player object including resources balance will be displayed. Once in possession of an area the player may now be able to receive resources from players who land on that area based on the areaRent attribute and have the option to further develop the area in order to receive higher contributions once landed on. In case the player does not want to buy the area, the area remains unowned and the player continues to "take their turn".


4. Develop area
preconditions
player has matching regions, can be on any area in board. Player provided with a list of available qualifying areas to select from
Actor
:player
:saveOurPlanet
:board
Which area would you like to develop?
return index = i
board.getDevelopmentStage()
Alternative
return developmentStage
[If board.get(i).getDevelopmentStage >4]
please select a different index
[Else]
Continue with developArea()
board.getDevelopmentCost(i)
return developmentCost
Alternative
[If player.getGreenCoin > board.get(i).getDevelopmentCost]
Display Development Cost
Would you like to develop?
Alternative
[If player return y]
return y
player.reducefunds(developmentCost)
board.setDevelopmentStage()
board.setMultiplier()
board.setDevelopmentCost()
display updated player information
board.setAreaValue()
display updated area information
[Else, player return n]
return n
display message
endTurn() method is called,ending players turn
[Else, [If player.getGreenCoin !> board.get(i).getDevelopmentCost]]
endTurn() method is called,ending players turn

CASE 4;

Also, assuming that the preceding condition described in the takeTurn diagram has been met. The player is not required to be on their owned asset to upgrade but limited to one upgrade development in a given turn.

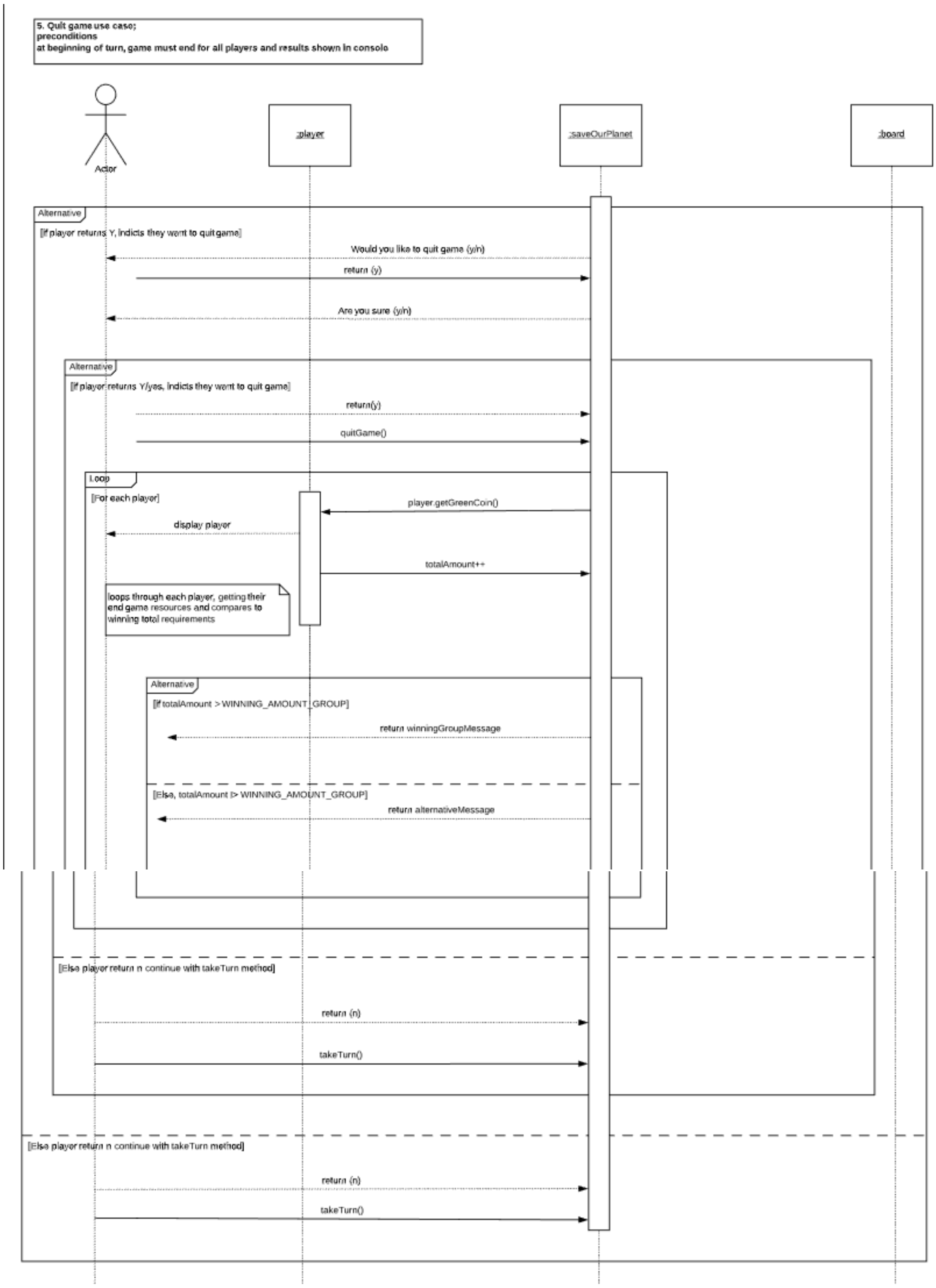
After determining that the player owns all areas in a given region. They are now presented with the opportunity of upgrading their areas. They can do so from any position on the board object. First the player is asked which area they wish to develop, and they return the index value associated with that area. On the UML sequence diagram this choice is represented by the letter i. The system checks if the area is already at the major development stage, and if so prompts the player for a different value. Otherwise the developArea method continues. A quick check is performed to ensure the player has enough greencoin available to fund the requested development, if not they are not presented with any further option and the endTurn method is called. The cost of such development is extracted using getDevelopmentCost and returned. The player is now asked if they wish to proceed with the development. If they respond n, the endTurn method is called, but if they respond y, the reducePlayerFunds method is called from the player object with the relevant development cost passed as a value. Once a player's greencoin has changed, details are output to the console window. Next several attributes are updated within the board object, with the setDevelopmentStage incremented to the next stage up to a maximum of three minor development stages. Once stage three has being achieved an additional stage described as a major development can now be enacted, which would yield a better contribution from players arriving. No development beyond this stage is permitted, as it is determined that any research beyond this point would not yield successful results. Board setMultiper is also updated, once the stage has been increased the level of contribution receivable also increases with each player visit. Again, with a greater increase in the case of a major development. Unfortunately, with rising development stages it increases the development cost to the player, this is increased within the developArea method after completing a stage and set via the setDevelopmentCost method. The resources received from other players in addition to investor funding are intended to encourage investment in the development and contribute towards these costs. Finally, with greater investment the board object values are increased and this is updated via the setAreaValue method. Once updated all information attaining to the developments are displayed to the player via the console window.

CASE 5;

Must be on that players turn at beginning. Results in the game ending for all involved.

At the beginning of each turn the player is asked a simple question whether they wish to continue the game or to exit. A follow up question "are you sure?" is presented in event of an error to try and catch players who may have inadvertently misread or entered the incorrect details. Should this happen in error the player will be returned to the beginning of the takeTurn method and continue as if nothing happened. However, if a player does want to quit the game, the takeTurn mechanic comes to a halt and the players' current resources are calculated. This calculation is done by looping through all the players in the arraylist and using the getGreenCoin amount from each player and storing the accumulated amount in a variable. Each individual player details are printed to the console window. This is then compared to the winning amount for a group variable.

Although, this is similar to the endgame scenario the player's resources are calculated to see if an individual or group has amassed the required resources threshold. It's largely assumed as the game hasn't come to a natural conclusion that there is no clear winner, so a group winner is more likely to be called at this stage. Again, like the endgame method mentioned earlier there are two messages based on either a winning team or unsuccessful attempt. Although seemingly similar to endgame, quitGame method differs in that the player themselves must enact the use of this method as opposed to the endgame method which is called from within the taketurn method. Where a player has either no resources left, a winner has been declared or the number of laps has exceeded the allowable value. The quitGame method also is available to players at several stages during the game, such as playerRegistration and is the only way a player can forcefully exit the game by their own choice.


5. Quit game use case;
preconditions
at beginning of turn, game must end for all players and results shown in console
Actor
:player
:saveOurPlanet
:board
Alternative
[if player returns Y, indicts they want to quit game]
Would you like to quit game (y/n)
return (y)
Are you sure (y/n)
Alternative
[if player returns Y/yes, indicts they want to quit game]
return(y)
quitGame()
Loop
[For each player]
player.getGreenCoin()
display player
totalAmount++
loops through each player, getting their end game resources and compares to winning total requirements
Alternative
[if totalAmount > WINNING_AMOUNT_GROUP]
return winningGroupMessage
[Else, totalAmount !> WINNING_AMOUNT_GROUP]
return alternativeMessage
[Else player return n continue with takeTurn method]
return (n)
takeTurn()
[Else player return n continue with takeTurn method]
return (n)
takeTurn()

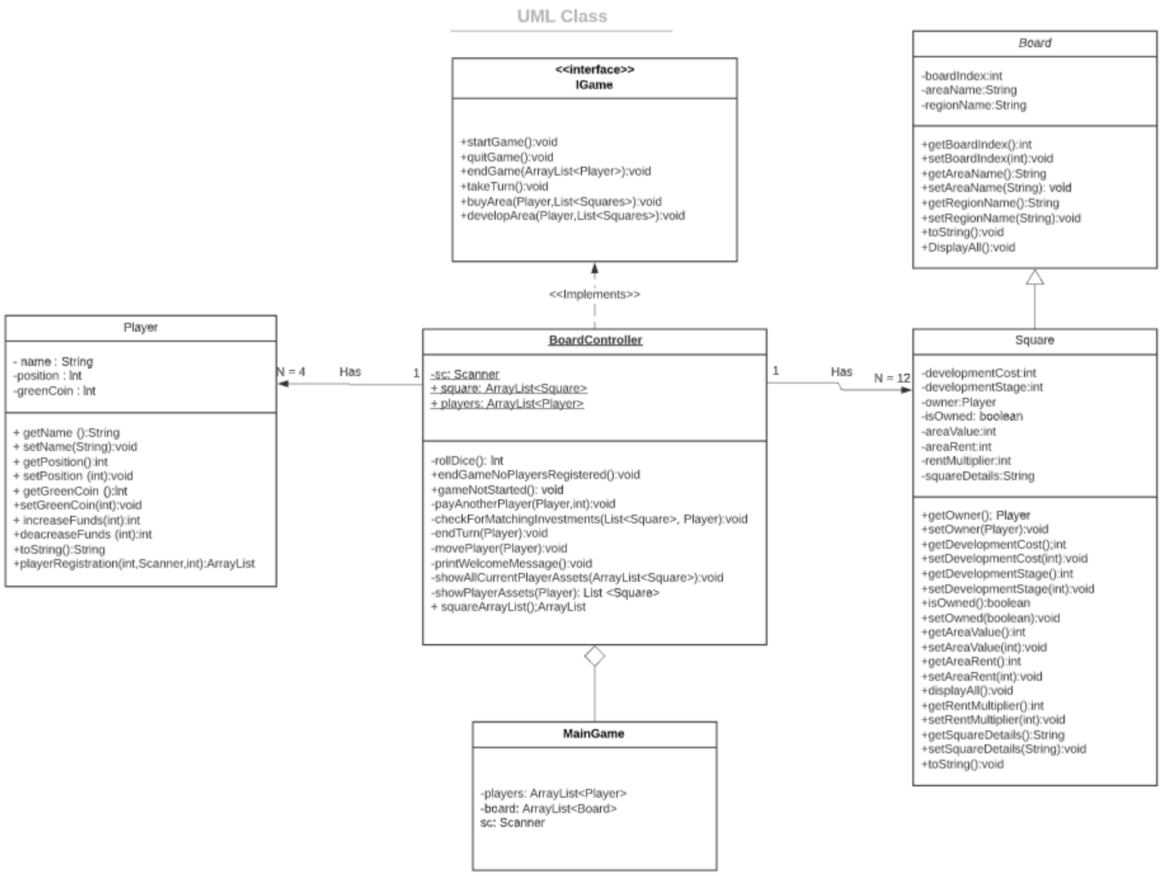
Design [EH]

In this group project we constructed a UML class diagram to illustrate the design of the game. A UML class diagram describes the structure of a system by showing the system's classes, their attributes, methods, and the relationships among other class objects. The purpose of class diagrams shows static structure of classifiers in a system, a diagram provides a basic notation for other structure diagrams prescribed by UML and they are helpful for developers and other team members. Business analysts can use class diagrams to model systems from a business perspective and to matching customers' requirements. A UML class diagram is made up of a set of classes and description of relationships between classes. When creating a UML class diagram the design needs to be clear, concise and informative. It also needs to be easy to read and understand so that the code for the project can be implemented and constructed.

We chose this UML structure as it was simple to maintain and understand. With the aim that it would be easy to store data for fast retrieval. The Player class stores items relating to the creation of Player objects, the information is kept hidden from other classes by using one of the four pillars of OOP, encapsulation. Where the data structure remains hidden from other classes but may be accessed by getter & setters.

The Board class is set to abstract because it is too abstract to be instantiated in the game. Smaller classes inherit the blueprint for their objects from the Board class in addition to adding their own unique identifiers. Again Board/Square classes are easily maintainable and new data can easily be added or amended. Both of these classes have business rules applied in regards of creating new types of objects such as allowable size/variables. Which allows for greater control of data being stored.

Below is the UML class diagram for our game.



Square Class and Board abstract [MCS]

The Square class required continual changes throughout the development of the game. In the end we decided upon a format that gave us two types of square, one which contained a purchasable area and another which contained the non-purchasable areas within the game i.e. the Home Square and the Travel Delayed Square. Array lists allowed us the greatest efficiency when cycling through the board, we matched the player's position to the number on the board in order to extract the squares details. The Square class inherits three parameters from the board class, these three parameters are the boardIndex, areaName and regionName. We chose these three parameters as these were the only three which we found that were essential in creating both types of areas.

Within the board class we had three set methods for storing valid values. The setBoardIndex method has a business rule to ensure that boardindex can only be set within a minimum and maximum range. The setAreaName and setRegionName methods must be set between 1 and 20 characters long. If the values are invalid an Illegal Argument Exception is thrown

The square class inherits three parameters from the Board class and has eight parameters of its own, each of which with its own unique function in the board that enable the game to function. All of the parameters have their own business rules when they are being created such as min and max values, and lengths. We also created separate constructors for the two types of board objects in our game as this allowed for faster creation of different types of objects. With the current setup of the square class it would allow us to expand upon the board if we chose to do so by adding new board object, although we would have to modify the business rule in setBoardIndex. This supplies squares with excellent functionality. The Square class has a many to one association with the BoardController class, with many square objects for one BoardController object. For example in our save our world game we have twelve squares in the game.

Player Class [DG]

The Player class holds information for each player. The instance variables are name, greenCoin and position. All variables have a get and set method so that player information can be stored and retrieved. A Player object is created using the constructor. Player information is passed into the constructor and must go through the setter methods to ensure there are no invalid values being set. The increase funds method takes in a parameter and increases player greencoin by that amount. The reduce funds method takes in a parameter and reduces player green coin by that amount.

The toString method takes all the players information and prints it to the console. The playerRegistration method creates two to four Player objects, which populate an array list. It is possible however to create as many Player objects as required. Users are asked for a valid name for each Player object. Should a duplicate be entered a further request for details will be issued. Player position default is set to "Home area" and all players are set with the same starting balance. Players stored in the array list can be easily accessed at any time and their variables changed using get and set methods for each variable. The Player class is connected to the board class with a many to one association. With the BoardController having many Player objects. With our game having a minimum of two and a maximum of four Player objects initially.

BoardController Class and IGame interface [EH][PK]

Within the BoardController class there are several methods related to the functionality of the game, several of these methods, essential in the running of the game are implemented from the IGame interface including the following.

startGame, an implemented method from the IGame interface. Uses the printWelcomeMessage method from within the class to display a greeting message in the console window. This method also uses the playerRegister method to create an arraylist of player objects in the BoardController class.

takeTurn is also implemented from the IGame interface and repeatedly loops over the player arraylist for a certain number of turns to create the save our world game object. Within this crucial method there are several calls to other methods including getting and setting variables from the Player and Square classes. At the end of the takeTurn method the game determines if there is a winner and if not using the endgame method, the loop continues until the game has either been won or the number of allowed turns is exceeded.

The quitGame method allows a player to quit the game. This method displays the end game details such as the selected outcome. This method also allows for all the players resources to be displayed at the end of the turn. It's the only time where a player can interact in order to finish the game.

The endGame method is used when one of the players playing the game has sufficient greencoin. The game is over and all the players final details are displayed, for example their resources, such as what areas they own and how developed each of the areas are. As well as the total greencoin amount of each player. The method finally declares if team has won and that the planet is saved.

The buy area method can be called in this class. The buy area method firstly gives the player the option to buy or not to buy the square they are currently on if it's not owned by another player. If the player buys the area they become the owner of that area. The cost of the area is reduced from the player's funds and they end their turn, if they have no eligible areas to develop.

The developArea method is called at the end of the take turn method. This method is only used if a player has all the areas in a single region and if they have enough greencoin to invest in a development. A player will be given a choice as to what areas they can develop. There are three development stages a player must have before they can invest in a major development. If a player chooses to develop the greencoin will be reduced from their funds before they end turn. If a player doesn't want to invest in a development they end their turn.

In addition to these implemented methods, there are several smaller methods in the BoardController class to help with the functioning of the game. SquareArrayList returns an Arraylist of square items used to populate the Square arraylist and create the board object for our game.

The printWelcomeMessage method displays a welcome message in the console window and sets the scene for the game using the game narrative. In controlling the player's position around our board object the method movePlayer gets the players previous position, takes a value from the rollDice method and sets the playersPosition at the new value relating to the number on the board object. The RollDice method simulates two dice being rolled and returns an int "rolled" value to be used in movePlayer and setting the player position.

Where a player lands on an area "owned" by another player the payAnotherPlayer method is used in the case where another player owns an area and are obliged to receive a contribution to development costs from the visiting player. The showPlayerAssets method loops over the board object arraylist and searches for the player values and returns a List of matching areas for further analysis. A method for checkForMatchingInvests uses the List mentioned previously to determine if the player has all the areas within a region and if so, allows the developArea use case to start.

The showAllCurrentPlayerAssets is a method optionally ran at the end of a players turn showing an updated state of play, with ownership of board objects display and available areas shown. endTurn another method called at the end of the takeTurn method and determines if the players individual greencoin balance has exceeded the required amount to be declared the "winner". If not an encouraging message is displayed to the console.

There are two method that are used in case of a player not being able to register successfully. TheendGameNoPlayersRegistered is a method designed to ensure the game doesn't crash in the event of no players registering and gameNotStarted is a method to catch the game in event of some general error. It allows the user to try restart the game before closing the system.

The method implemented from the interface serve as a template for creating the save our world game object. BoardController acts as central part of the system where most of the classes may interact. The board object itself is created within the mainGame method but instead of cluttering the method with many method calls and creation of objects, we created this class to essentially run the game in the background. Two arraylists are created, one storing Player class objects and one storing Square class objects for easy access by the BoardController class.

Many of the other methods are smaller in size and have basic functions such as rollDice which similates a diceRoll or movePlayer which just changes the player’s location. Many of these methods are private and not accessible to the classes outside this package which allows for a high level of control.

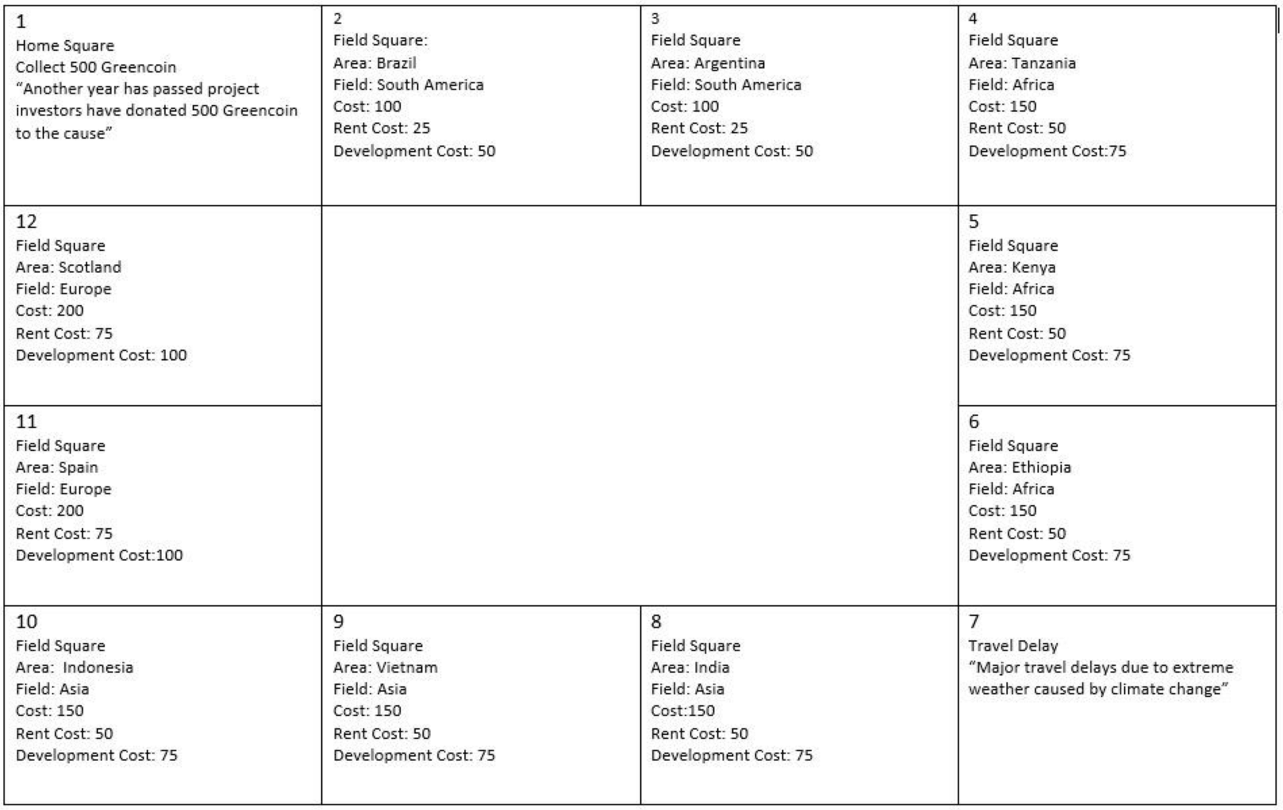
Larger methods such as take turn, buy area use the smaller method to reduce the amount of details they are required to store and process and it helps greatly with the maintainability of the system as faults can be quickly noted in smaller modules. Also, the smaller modules may be used when required in the case or repetitive coding such as the transferring of funds between players. Also, the endgame, endTurn and quitGame methods allowed for various different scenarios to be offered to the player, we hoping in doing so offering a varied experience to the players of the game.

The class itself is flexible, if a larger board was required it can easily be added to the arraylist. If a new type of board game similar in structure to save the world was required, all that would be needed to make that possible is the creation of a main method to create a board object. Originally we tested a larger board and different numbers of players, with multiple different types of squares with little issues. Players too can easily be increased or decreased, the class has several constant variables stored which allows the players to increase many of the games controls, such as number of turns, winning amounts.

Appendix: Game Guide [DG]

- There are 2-4 players
- Each player must enter their name to log in to the game
- Players take turns in the order they enter their names
- 2 dice are rolled each turn and a total given
- The player moves respectively based on the order they joined the game
- If a player visits an area they may buy it, if its unowned
- If an area is owned the player must invest in that project to help the global effort
- Once an area is owned, it can be developed. This increases the amount of contribution a player receives
- A player must own all areas in a region, before an area can be developed
- Three developments can be made on an area and one major development
- If a player passes the home area they collect 500 green coin from project investors
- If a player lands on the travel delayed area they don't have the privilege of buying a new area because of travel delays caused by extreme weather.
- If one player reaches 20,000 greencoin they have saved the planet with help from their fellow eco warriors.
- If a player runs out of resources all investor funding stops and the players funds are tallied up. If the total is above 20,000 the group effort has saved our planet! If it is below 20,000 then all hope is lost, and everyone disbands to enjoy their final days.
- If a player decides to quit the challenge, then all investors loss confidence and funding stops. The same happens as if a player has no resources left.

Game Board



1 Home Square Collect 500 Greencoin "Another year has passed project investors have donated 500 Greencoin to the cause"	2 Field Square: Area: Brazil Field: South America Cost: 100 Rent Cost: 25 Development Cost: 50	3 Field Square Area: Argentina Field: South America Cost: 100 Rent Cost: 25 Development Cost: 50	4 Field Square Area: Tanzania Field: Africa Cost: 150 Rent Cost: 50 Development Cost:75
12 Field Square Area: Scotland Field: Europe Cost: 200 Rent Cost: 75 Development Cost: 100			5 Field Square Area: Kenya Field: Africa Cost: 150 Rent Cost: 50 Development Cost: 75
11 Field Square Area: Spain Field: Europe Cost: 200 Rent Cost: 75 Development Cost:100			6 Field Square Area: Ethiopia Field: Africa Cost: 150 Rent Cost: 50 Development Cost: 75
10 Field Square Area: Indonesia Field: Asia Cost: 150 Rent Cost: 50 Development Cost: 75	9 Field Square Area: Vietnam Field: Asia Cost: 150 Rent Cost: 50 Development Cost: 75	8 Field Square Area: India Field: Asia Cost:150 Rent Cost: 50 Development Cost: 75	7 Travel Delay "Major travel delays due to extreme weather caused by climate change"

Detailed Description of Developments [DG]

Region: South America

Area: Brazil

There is a growing ecological movement in South America, fuelled by social unrest caused by climate change. The government are allocating large areas of once prime rainforest, lost to farming and logging, for reforestation. As more areas of the rainforest are purchased the new saplings absorb carbon dioxide from the atmosphere and slow down the greenhouse effect, buying more time to save our planet.

Minor Development: +1 Thousand Acre Reforestation

Major Development: +5 Thousand Acre Reforestation

Area: Argentina

The Argentinian government are selling large areas of the great wilderness Patagonia for nature reserves. Allocating this for nature reserves reduces widespread sheep and cattle farming. As a result, the more potent greenhouse gas methane is reduced in the atmosphere, slowing down climate change. Nature reserve production increases biodiversity and enables locals to leave their traditional farming lifestyle and make a living from eco-tourism and sustainable farming.

Minor Development: +1 Million Acre Nature Reserve

Major Development: +5 Million Acre Nature Reserve

Region: Africa

Area: Tanzania

Methane biodigesters are a cheap revolutionary technology that collect human waste in a sealed and sanitary container, often a large plastic drum. Bacteria found naturally in human waste digest the waste and produce methane. A low-tech gas capture and storage device is created using plumbing materials. The Biogas is perfect for cooking food for local people. Almost all rural people in Tanzania are cooking on charcoal or wood. Converting from charcoal to biogas reduces the release of greenhouse gases into the atmosphere.

Minor Development: +1 Thousand Methane Biodigesters

Major Development: +5 Thousand Methane Biodigesters

Area: Kenya

The Ocean conveyor belt brings powerful currents to the shores of Kenya. The Kenyan government are looking for outside investors to expand their production of tidal energy harvesting machines. The current system produces over 700 MW of energy! Future developments will power all nearby cities reducing the use of fossil fuel consumption and greenhouse gas production.

Minor Development: +1 Thousand MW Tidal Energy Machine

Major Development: +5 Thousand MW Tidal Energy Machine

Area: Ethiopia

Microbial fuel cells produce electricity from organic matter. They contain bacteria that digest organic matter and donate electrons as waste products. The electrons are harvested to produce electricity. This new green technology is cheap and easy to implement. Known locally as pee power the electricity generators are replacing dangerous kerosene lamps found in many people’s homes. The reduction in burning kerosene helps to slow down greenhouse gases and global climate change.

Minor Development: +10 Thousand Bio-Fuel Cell Generators

Major Development: +20 Thousand Bio-Fuel Cell Generators

Region: Asia

Area: India

India is one of the fastest growing economies in the world and a large producer of agricultural waste. A biogas plant uses agricultural waste to produce biogas. Agricultural waste is digested by bacteria in a large vat and the methane gas is collected and used for cooking, heating and conversion into electricity. Each development produces a much higher level of gas. Using biogas recycles freely available waste products to produce valuable energy that replaces fossil fuel energy sources.

Minor Development: +5 Thousand Bio-Gas Generators

Major Development: +20 Thousand Bio-Gas Generators

Area: Vietnam

Coral reef restoration in Vietnam boosts the local eco-tourism trade and helps the climate. Coral capture dissolved carbon dioxide in the sea and produce oxygen during photosynthesis. Coral reef restoration increases biodiversity in the sea and gives endangered species that rely on the coral reef an increased change of survival.

Minor Development: +1 Thousand Acre Coral Reef Restoration

Major Development: +5 Thousand Acre Coral Reef Restoration

Area: Indonesia

Indonesia has a special place on the most geologically active region in the world: the ring of fire. With 127 active volcanoes it is a prime location for geothermal energy production. Mount Merapi is the most active volcano in Indonesia and home to a revolutionary new geothermal energy plant that captures steam from the volcano and converts it into electricity. Each development produces 5 new energy plants, each at a different volcano.

Minor Development: +5 Geothermal Power Plants

Major Development: +10 Geothermal Power Plants

Region: Europe

Area: Spain

The scorching sun in Spain is a prime location for a large photovoltaic farm. The main primary producer of energy for the whole biosphere is the sun. So why not capture vast quantities of energy directly from it!

Minor Development: +1 Thousand MW Solar Power Farms

Major Development: +5 Thousand MW Solar Power Farms

Area: Scotland

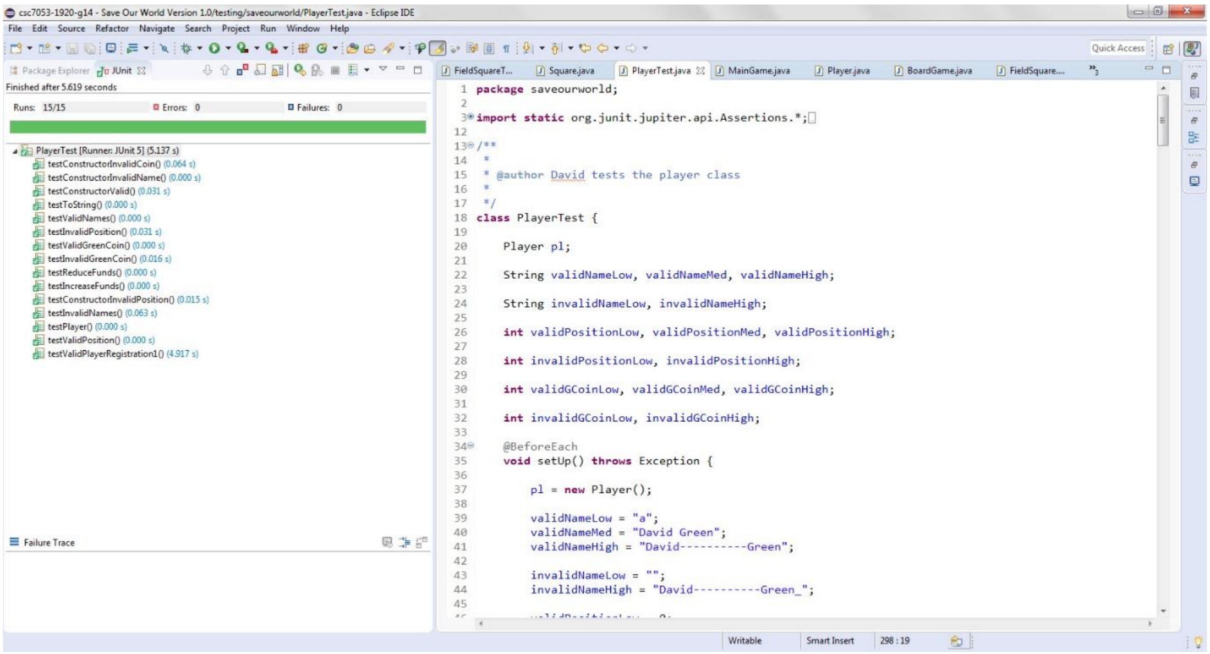
The brisk highlands of Scotland: remote, powerful winds and a vast wilderness. It’s the perfect place to capture energy from the constant power of the wind. Each development produces 1000 MW of electricity. A major development produces 5000 MW.

Minor Development: +1 Thousand MW Wind Farm

Major Development: +5 Thousand MW Wind Farm

Appendix 1 [DG]

Player Junit Test Confirmation

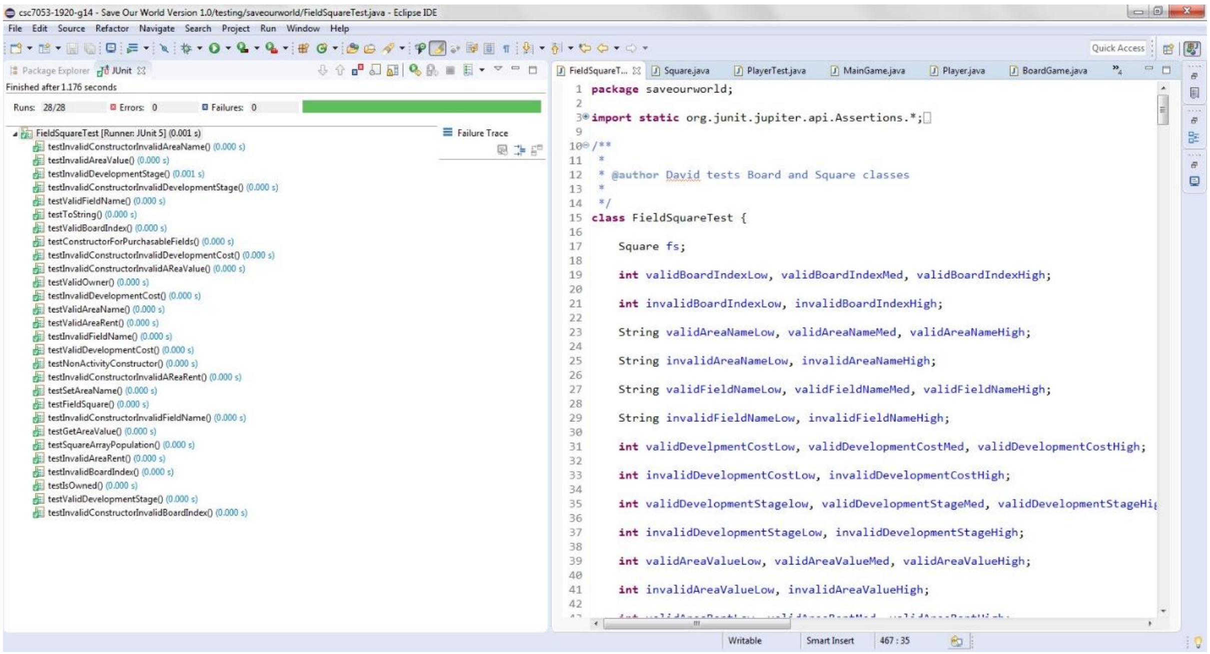


csc7053-1920-g14 - Save Our World Version 1.0/testing/saveourworld/PlayerTest.java - Eclipse IDE

```
Finished after 5.619 seconds

Runs: 15/15          Errors: 0          Failures: 0

PlayerTest [Runner: JUnit 5] (5.137 s)
    testConstructorInvalidCoin() (0.064 s)
    testConstructorInvalidName() (0.000 s)
    testConstructorValid() (0.031 s)
    testToString() (0.000 s)
    testValidNames() (0.000 s)
    testInvalidPosition() (0.031 s)
    testValidGreenCoin() (0.000 s)
    testInvalidGreenCoin() (0.016 s)
    testReduceFunds() (0.000 s)
    testIncreaseFunds() (0.000 s)
    testConstructorInvalidPosition() (0.015 s)
    testInvalidNames() (0.063 s)
    testPlayer() (0.000 s)
    testValidPosition() (0.000 s)
    testValidPlayerRegistration1() (4.917 s)
```

```java
package saveourworld;

import static org.junit.jupiter.api.Assertions.*;

/**
 *
 * @author David tests the player class
 *
 */
class PlayerTest {

    Player pl;

    String validNameLow, validNameMed, validNameHigh;

    String invalidNameLow, invalidNameHigh;

    int validPositionLow, validPositionMed, validPositionHigh;

    int invalidPositionLow, invalidPositionHigh;

    int validGCoinLow, validGCoinMed, validGCoinHigh;

    int invalidGCoinLow, invalidGCoinHigh;

    @BeforeEach
    void setUp() throws Exception {

        pl = new Player();

        validNameLow = "a";
        validNameMed = "David Green";
        validNameHigh = "David---------Green";

        invalidNameLow = "";
        invalidNameHigh = "David----------Green_";

```

Square Junit Test Confirmation (Refactored)



csc7053-1920-g14 - Save Our World Version 1.0/testing/saveourworld/FieldSquareTest.java - Eclipse IDE

```
Finished after 1.176 seconds

Runs: 28/28          Errors: 0          Failures: 0

FieldSquareTest [Runner: JUnit 5] (0.001 s)
    testInvalidConstructorInvalidAreaName() (0.000 s)
    testInvalidAreaValue() (0.000 s)
    testInvalidDevelopmentStage() (0.001 s)
    testInvalidConstructorInvalidDevelopmentStage() (0.000 s)
    testValidFieldName() (0.000 s)
    testToString() (0.000 s)
    testValidBoardIndex() (0.000 s)
    testConstructorForPurchasableFields() (0.000 s)
    testInvalidConstructorInvalidDevelopmentCost() (0.000 s)
    testInvalidConstructorInvalidAReaValue() (0.000 s)
    testValidOwner() (0.000 s)
    testInvalidDevelopmentCost() (0.000 s)
    testValidAreaName() (0.000 s)
    testValidAreaRent() (0.000 s)
    testInvalidFieldName() (0.000 s)
    testValidDevelopmentCost() (0.000 s)
    testNonActivityConstructor() (0.000 s)
    testInvalidConstructorInvalidAReaRent() (0.000 s)
    testSetAreaName() (0.000 s)
    testFieldSquare() (0.000 s)
    testInvalidConstructorInvalidFieldName() (0.000 s)
    testGetAreaValue() (0.000 s)
    testSquareArrayPopulation() (0.000 s)
    testInvalidAreaRent() (0.000 s)
    testInvalidBoardIndex() (0.000 s)
    testIsOwned() (0.000 s)
    testValidDevelopmentStage() (0.000 s)
    testInvalidConstructorInvalidBoardIndex() (0.000 s)
```

```java
package saveourworld;

import static org.junit.jupiter.api.Assertions.*;

/**
 *
 * @author David tests Board and Square classes
 *
 */
class FieldSquareTest {

    Square fs;

    int validBoardIndexLow, validBoardIndexMed, validBoardIndexHigh;

    int invalidBoardIndexLow, invalidBoardIndexHigh;

    String validAreaNameLow, validAreaNameMed, validAreaNameHigh;

    String invalidAreaNameLow, invalidAreaNameHigh;

    String validFieldNameLow, validFieldNameMed, validFieldNameHigh;

    String invalidFieldNameLow, invalidFieldNameHigh;

    int validDevelpmentCostLow, validDevelopmentCostMed, validDevelopmentCostHigh;

    int invalidDevelopmentCostLow, invalidDevelopmentCostHigh;

    int validDevelopmentStagelow, validDevelopmentStageMed, validDevelopmentStageHi

    int invalidDevelopmentStageLow, invalidDevelopmentStageHigh;

    int validAreaValueLow, validAreaValueMed, validAreaValueHigh;

    int invalidAreaValueLow, invalidAreaValueHigh;

```

Individual units of code are tested to assert they are working properly. An instance variable can be tested to make sure it is set properly and does not through an Illegal argument exception. In Junit the values at the boundaries, on either side, are tested. This ensures that not anomalous values can be set in the system. Running Junit after any changes to the source code helps to locate any potential flaws in the program.

In Field Square the board index is tested to make sure it is set within a minimum and maximum range. The same is done for area name, field name, development cost, development stage, area value, area rent and the rent multiplier. The property owned boolean is tested to make sure it is set to only true or false, if it is set to anything else an Illegal argument exception is thrown. The default constructor is tested to assert it is not null. The non-activity and purchasable field's constructors are tested to make sure the instance variables are being set and retrieved properly in the constructor. Invalid values are tested for all setters and constructors to make sure invalid values can't be set.

In the Player Junit test the name, position, and green coin setters are tested to make sure they are set within a minimum and maximum range and invalid ranges are tested. Constructors are tested with valid and invalid ranges. The increase funds method is tested to make sure it adds funds to a player. The reduce funds method is tested to make sure it reduces funds. The player registration method is tested to see if two player objects can be added to an array list of players. And does the array list match another array list created manually. The two array lists are compared using an assert equals method to make sure they match.

In integration testing Individual units are combined into one test. In the Field Square test, the final test is an integration test. An array list is populated with Field Square objects and then tested to see if it has the correct output. One Field Square object consists of many individual setter methods and the array list consists of many Field Square objects. This is an example of integration testing: combining many individual units and testing them together. The player registration test in player is another example of integration testing. An array list of players is populated and tested against an array list made manually.

Appendix 2 [EH]

Minutes for Group__14___ Week commencing _13/01/2020_________ Date of this minute ____13/01/2020_______

The following team members were present

Name (printed/typed)	Signature
Michael Carey-Small	
Eoin Henry	
David Grech	
Padraig Kearney	

Task Reporting

Name & Role (1): All member of the group

- An early design of the game board has been created
- All specifications of the game have been outlined

Actions Planned

Name & Role (1):Michael Carey-Small

- Decide upon three developments for each area on the game

Name & Role (2):Eoin Henry

- Decide upon an appropriate resource type, if money how much will it cost for each area and development.

Name & Role (3):David Grech

- Attempt to create a UML

Name & Role (4):Padraig Kearney

- Create a simple digital board in order to show the trip around the world

Minutes for Group___14__ Week commencing ____20/01/2020______ Date of this minute ___20/01/20________

The following team members were present

Name (printed/typed)	Signature
Michael Carey-Small	
Eoin Henry	
David Grech	
Padraig Kearney	

Task Reporting

Name & Role (1):Eoin Henry

- Pricing and the recourses that will be used was decided on.

Name & Role (2):Michael Carey-Small

- The potential locations and developments of the game have been created.

Name & Role (3):David Grech

- The UML use case diagrams' first version has been constructed.

Name & Role (4): Padraig Kearney

- Created a simple digital board in order to show the trip around the world

Actions Planned

Name & Role (1):Michael

- Step by step instructions of the game

Name & Role (2): Eoin

- Starting the player class

Name & Role (3):David

- To continue with the full creation of the UML diagram

Name & Role (4): Padraig

- Create a UML sequence

Minutes for Group 14 **Week commencing** 27/1/2020 **Date of this minute** 29/1/2020

The following team members were present

Name (printed/typed)	Signature
Michael Carey-Small	
Eoin Henry	
David Grech	
Padraig Kearney	

Task Reporting

Name & Role (1): Michael Carey-Small

- Completed step by step instructions for Use case diagram

Name & Role (2): Eoin Henry

- Started work on a Player class

Name & Role (3): David Grech

- Completed a Use Case diagram

Name & Role (4): Padraig Kearney

- Created first UML sequence diagram and plan for others
- Started working on a prototype version of the game

<u>Actions Planned</u>

Name & Role (1): Michael Carey-Small

- Start work on UML class diagram

Name & Role (2): Eoin Henry

- Create two Use Case Sequence Diagrams

Name & Role (3): David Grech

- Create two Use Case Sequence Diagrams

Name & Role (4): Padraig Kearney

- Create two Use Case Sequence Diagrams

Minutes for Group___14__ Week commencing ___03/02/2020_______ Date of this minute ___05/02/2020________

The following team members were present

Name (printed/typed)	Signature
Eoin Henry	
Michael Carey-Small	
David Grech	
Padraig Kearney	

<u>Task Reporting</u>

Name & Role (1): Michael Carey-Small

- Started to wotk on the UML class disgram

Name & Role (2):Eoin Henry

- Created two Use Case Sequence Diagrams

Name & Role (3):David Grech

- Created two Use Case Sequence Diagrams

Name & Role (4):Padraig Kearney

- Created two Use Case Sequence Diagrams

Actions Planned

Name & Role (1):Michael Carey-Small

- Use class diagrams

Name & Role (2):Eoin Henry

- Finish UML Sequence Diagrams

Name & Role (3):David Grech

- Finish UML Sequence Diagrams

Name & Role (4):Padraig Kearney

- Use case descriptions

Minutes for Group__14___ Week commencing ____10/02/20______ Date of this minute ____13/02/20_______

The following team members were present

Name (printed/typed)	Signature
Eoin Henry	
Michael Carey-Small	
David Grech	
Padraig Kearney	

Task Reporting

Name & Role (1):Michael Carey-Small

- First version of use class diagrams

Name & Role (2):Eoin Henry

- Finished UML Sequence Diagrams

Name & Role (3): David Grech

- Finished UML Sequence Diagrams

Name & Role (4):Padraig Kearney

- Use case descriptions started

Actions Planned

Name & Role (1):Michael Carey-Small

- Develop the class diagrams more

Name & Role (2):Eoin Henry

- Complete group minitues sheets

Name & Role (3):David Grech

- Finished UML case diagram

Name & Role (4):Padraig Kearney

- Develop class diagrams and finish use case descriptions

Minutes for Group___14__ Week commencing _____17/02/20_____ Date of this minute _____17/02/20______

The following team members were present

Name (printed/typed)	Signature
Eoin Henry	
Michael Carey-Small	
David Grech	
Padraig Kearney	

Task Reporting

Name & Role (1): Michael Carey-Small

- Completed the Class diagrams

Name & Role (2): Eoin Henry

- Week minutes updated

Name & Role (3):David Grech

- UML case diagrams completed

Name & Role (4): Padraig Kearney

- Use case descriptions have been updated and finished
-

Actions Planned

Name & Role (1): Michael Carey-Small

- Start on the board class code

Name & Role (2): Eoin Henry

- Tidy up use case descriptions by adding alternative flows to each.

Name & Role (3): David Grech

- Start on the player class code

Name & Role (4):Padraig Kearney

- Start on the start game method and main method code

Minutes for Group___14__ Week commencing _____24/02/20_____ Date of this minute _____25/02/20______

The following team members were present

Name (printed/typed)	Signature
Eoin Henry	
Michael Carey-Small	
David Grech	
Padraig Kearney	

Task Reporting

Name & Role (1): Michael Carey-Small

- Completed board class

Name & Role (2): Eoin Henry

- Weekly minutes were updated
- Use case description updated

Name & Role (3):David Grech

- Has completed the player class

Name & Role (4): Padraig Kearney

- Completed start game method
- Worked on the main game class

Actions Planned

Name & Role (1): Michael Carey-Small

- To complete the different square classes
  E.g fieldSquare, blankSqaure and goSquare.

- Start the buy area method

Name & Role (2): Eoin Henry

- Start the develop area method

Name & Role (3): David Grech

- Complete the main class code

Name & Role (4):Padraig Kearney

- Create an Interface class
- Continue with the main class code now called BoardGame

Name (printed/typed) Signature

Eoin Henry	
Michael Carey-Small	
David Grech	
Padraig Kearney	

Minutes for Group___14__ Week commencing _____02/03/20_____ Date of this minute ___02/20/20________

The following team members were present

Task Reporting

Name & Role (1): Michael Carey-Small

- Completed all the different square classes
- Finished buy area method

Name & Role (2): Eoin Henry

- Develop area method completed

Name & Role (3):David Grech

- Continued and completed a lot of the main game code

Name & Role (4): Padraig Kearney

- Created an interface class called IGame
- Continued and completed a lot of the main game code

Actions Planned

Name & Role (1): Michael Carey-Small

- Requirements and analysis section in report
- Finish up code

Name & Role (2): Eoin Henry

- Design section in report
- Finish up code

Name & Role (3): David Grech

- Test plan section of the report
- Finish up code

Name & Role (4):Padraig Kearney

- Realisation section in the report
- Finish up code

Minutes for Group___14__ Week commencing _____09/03/20_____ Date of this minute ______11/02/20_____

The following team members were present

Name (printed/typed)	Signature
Eoin Henry	
Michael Carey-Small	
David Grech	
Padraig Kearney	

Task Reporting

Name & Role (1): Michael Carey-Small

- Completed the requirement and analysis section of the report

Name & Role (2): Eoin Henry

- Completed the design section of the report

Name & Role (3):David Grech

- Completed the test plan section of the report

Name & Role (4): Padraig Kearney

- Completed the realisation section of the report

Actions Planned

Name & Role (1): Michael Carey-Small

- Tidy up code if needed
- Tidy up report if needed

Name & Role (2): Eoin Henry

- Tidy up code if needed
- Tidy up report if needed

Name & Role (3): David Grech

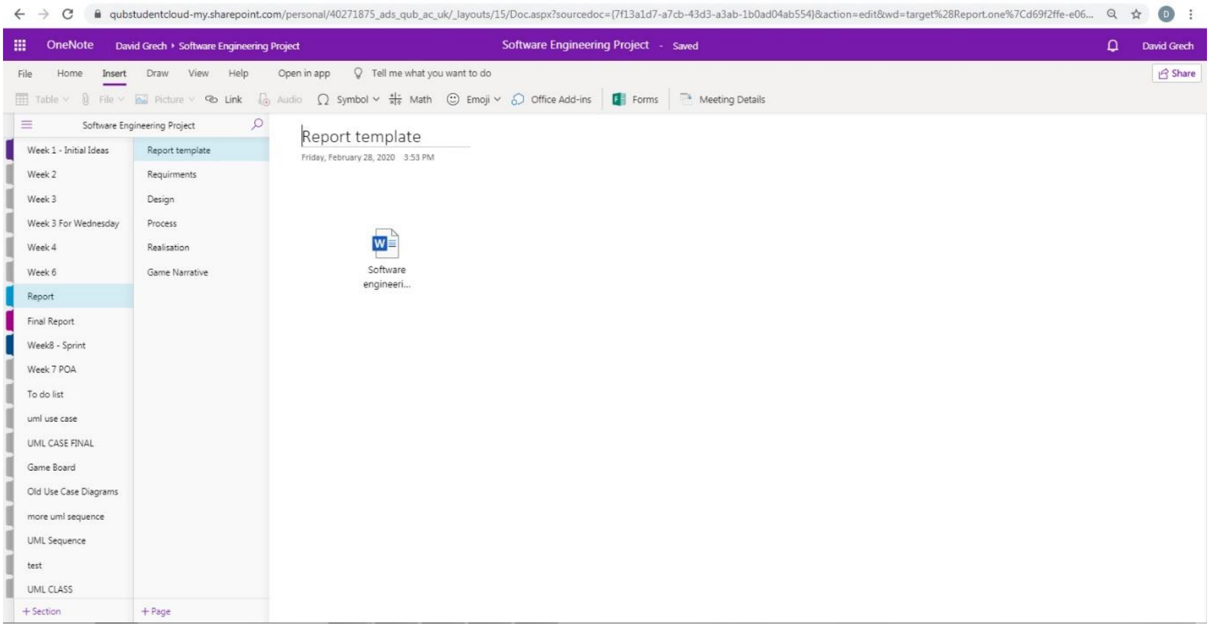
- Tidy up code if needed
- Tidy up report if needed

Name & Role (4):Padraig Kearney

- Tidy up code if needed
- Tidy up report if needed

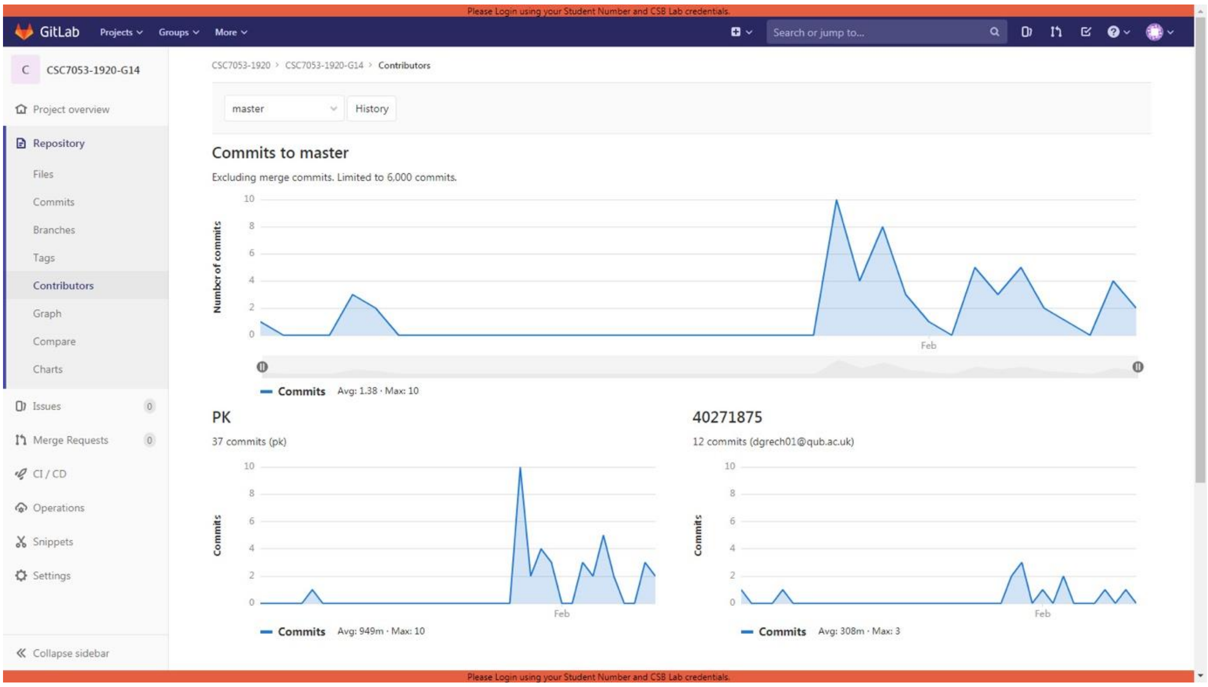
Appendix 3 [ALL]

One Note Team Management



We had two meetings each week and went to regular advisory meetings. We logged tasks for each of us to complete into OneNote and recorded when tasks were complete, see team minutes for more information.

Git Contributors Graph



Please Login using your Student Number and CSB Lab credentials.

Commits to master

Excluding merge commits. Limited to 6,000 commits.

Commits Avg: 1.38 · Max: 10

PK

37 commits (pk)

Commits Avg: 949m · Max: 10

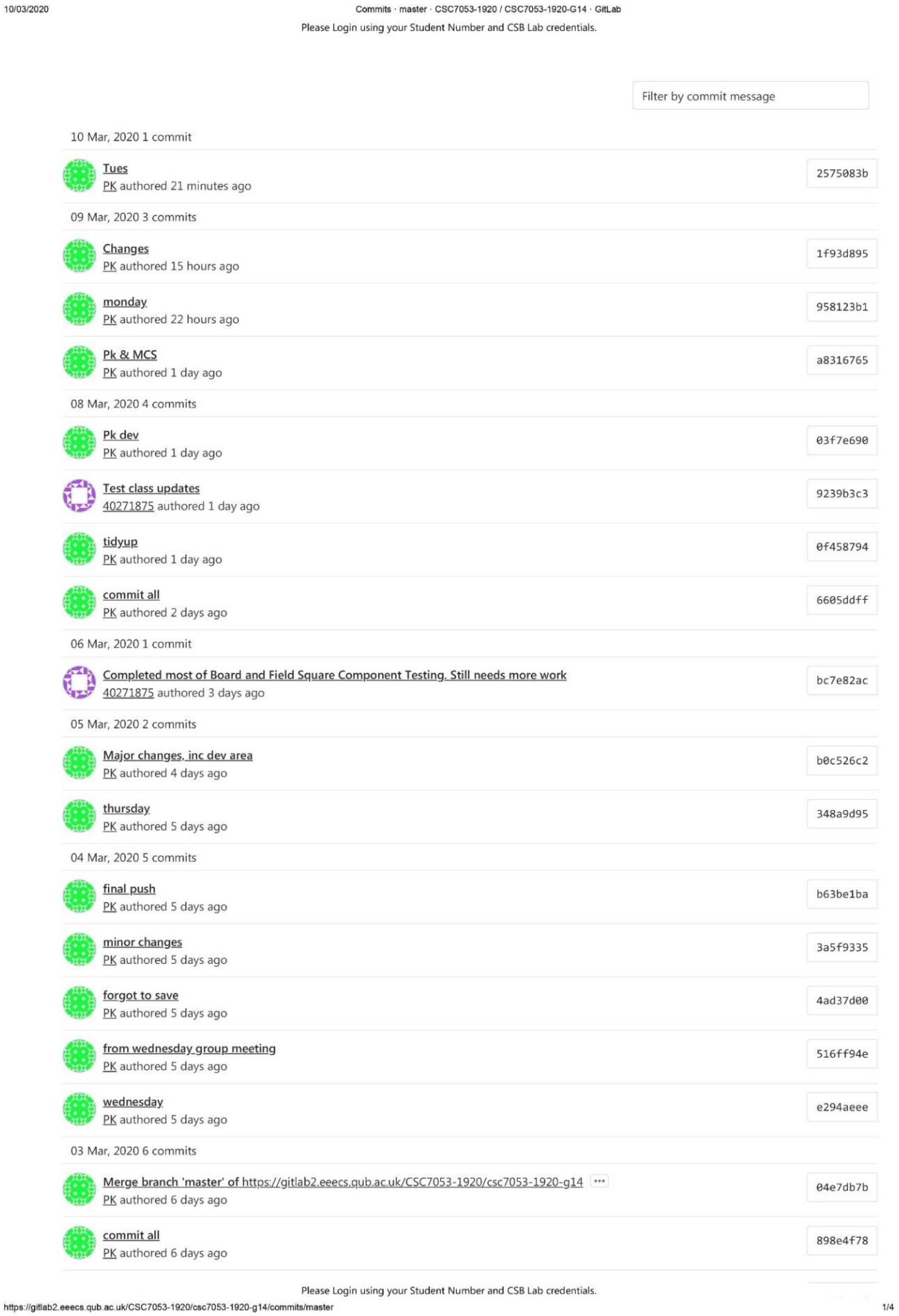
40271875

12 commits (dgrech01@qub.ac.uk)

Commits Avg: 308m · Max: 3

Please Login using your Student Number and CSB Lab credentials.

Git Commit History



10/03/2020 Commits · master · CSC7053-1920 / CSC7053-1920-G14 · GitLab

Please Login using your Student Number and CSB Lab credentials.

Filter by commit message

10 Mar, 2020 1 commit

Tues PK authored 21 minutes ago	2575083b

09 Mar, 2020 3 commits

Changes PK authored 15 hours ago	1f93d895
monday PK authored 22 hours ago	958123b1
Pk & MCS PK authored 1 day ago	a8316765

08 Mar, 2020 4 commits

Pk dev PK authored 1 day ago	03f7e690
Test class updates 40271875 authored 1 day ago	9239b3c3
tidyup PK authored 1 day ago	0f458794
commit all PK authored 2 days ago	6605ddff

06 Mar, 2020 1 commit

Completed most of Board and Field Square Component Testing. Still needs more work 40271875 authored 3 days ago	bc7e82ac

05 Mar, 2020 2 commits

Major changes, inc dev area PK authored 4 days ago	b0c526c2
thursday PK authored 5 days ago	348a9d95

04 Mar, 2020 5 commits

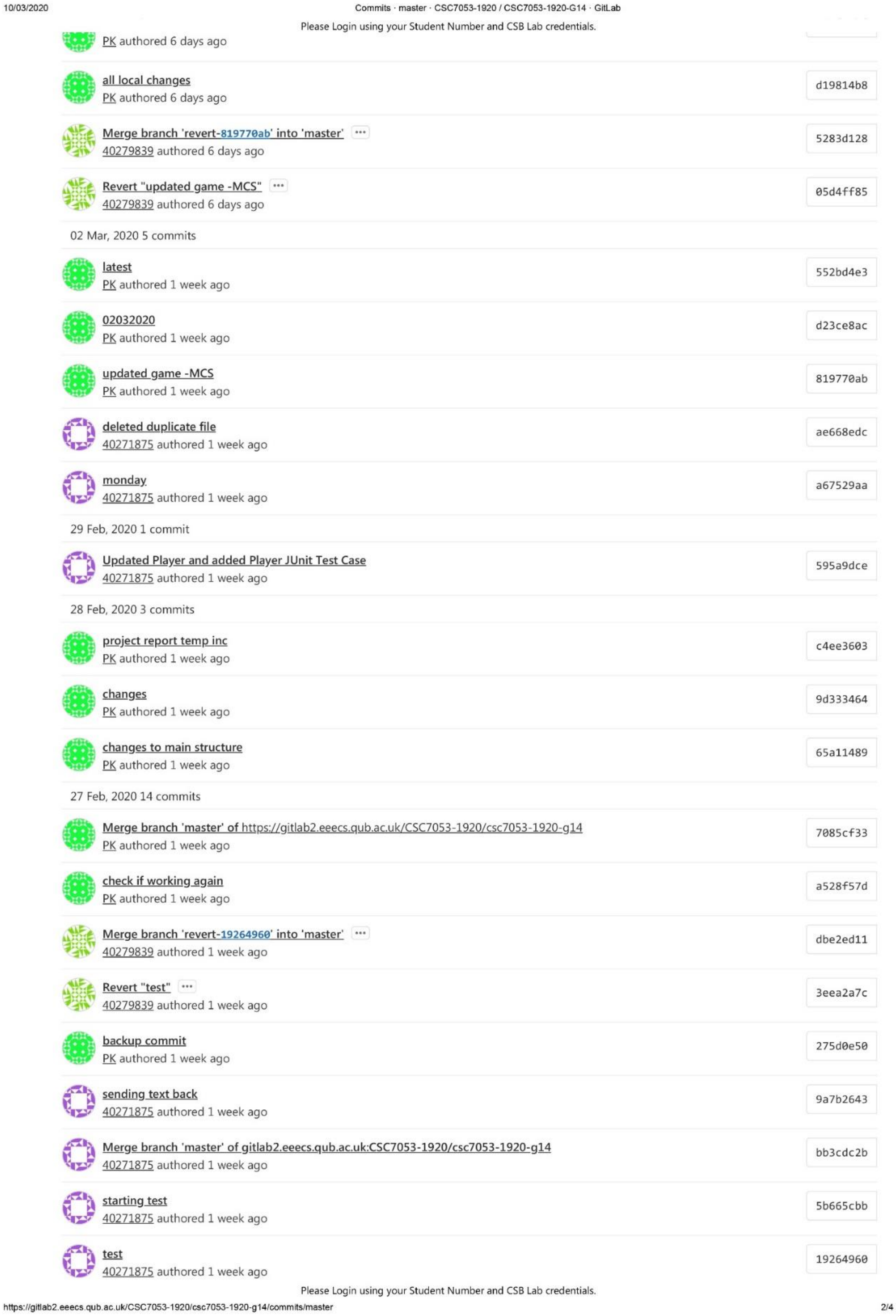
final push PK authored 5 days ago	b63be1ba
minor changes PK authored 5 days ago	3a5f9335
forgot to save PK authored 5 days ago	4ad37d00
from wednesday group meeting PK authored 5 days ago	516ff94e
wednesday PK authored 5 days ago	e294aeee

03 Mar, 2020 6 commits

Merge branch 'master' of https://gitlab2.eeecs.qub.ac.uk/CSC7053-1920/csc7053-1920-g14 ··· PK authored 6 days ago	04e7db7b
commit all PK authored 6 days ago	898e4f78

Please Login using your Student Number and CSB Lab credentials.

https://gitlab2.eeecs.qub.ac.uk/CSC7053-1920/csc7053-1920-g14/commits/master 1/4



10/03/2020 Commits · master · CSC7053-1920 / CSC7053-1920-G14 · GitLab

Please Login using your Student Number and CSB Lab credentials.

PK authored 6 days ago

all local changes
PK authored 6 days ago
d19814b8

Merge branch 'revert-819770ab' into 'master' ···
40279839 authored 6 days ago
5283d128

Revert "updated game -MCS" ···
40279839 authored 6 days ago
05d4ff85

02 Mar, 2020 5 commits

latest
PK authored 1 week ago
552bd4e3

02032020
PK authored 1 week ago
d23ce8ac

updated game -MCS
PK authored 1 week ago
819770ab

deleted duplicate file
40271875 authored 1 week ago
ae668edc

monday
40271875 authored 1 week ago
a67529aa

29 Feb, 2020 1 commit

Updated Player and added Player JUnit Test Case
40271875 authored 1 week ago
595a9dce

28 Feb, 2020 3 commits

project report temp inc
PK authored 1 week ago
c4ee3603

changes
PK authored 1 week ago
9d333464

changes to main structure
PK authored 1 week ago
65a11489

27 Feb, 2020 14 commits

Merge branch 'master' of https://gitlab2.eeecs.qub.ac.uk/CSC7053-1920/csc7053-1920-g14
PK authored 1 week ago
7085cf33

check if working again
PK authored 1 week ago
a528f57d

Merge branch 'revert-19264960' into 'master' ···
40279839 authored 1 week ago
dbe2ed11

Revert "test" ···
40279839 authored 1 week ago
3eea2a7c

backup commit
PK authored 1 week ago
275d0e50

sending text back
40271875 authored 1 week ago
9a7b2643

Merge branch 'master' of gitlab2.eeecs.qub.ac.uk:CSC7053-1920/csc7053-1920-g14
40271875 authored 1 week ago
bb3cdc2b

starting test
40271875 authored 1 week ago
5b665cbb

test
40271875 authored 1 week ago
19264960

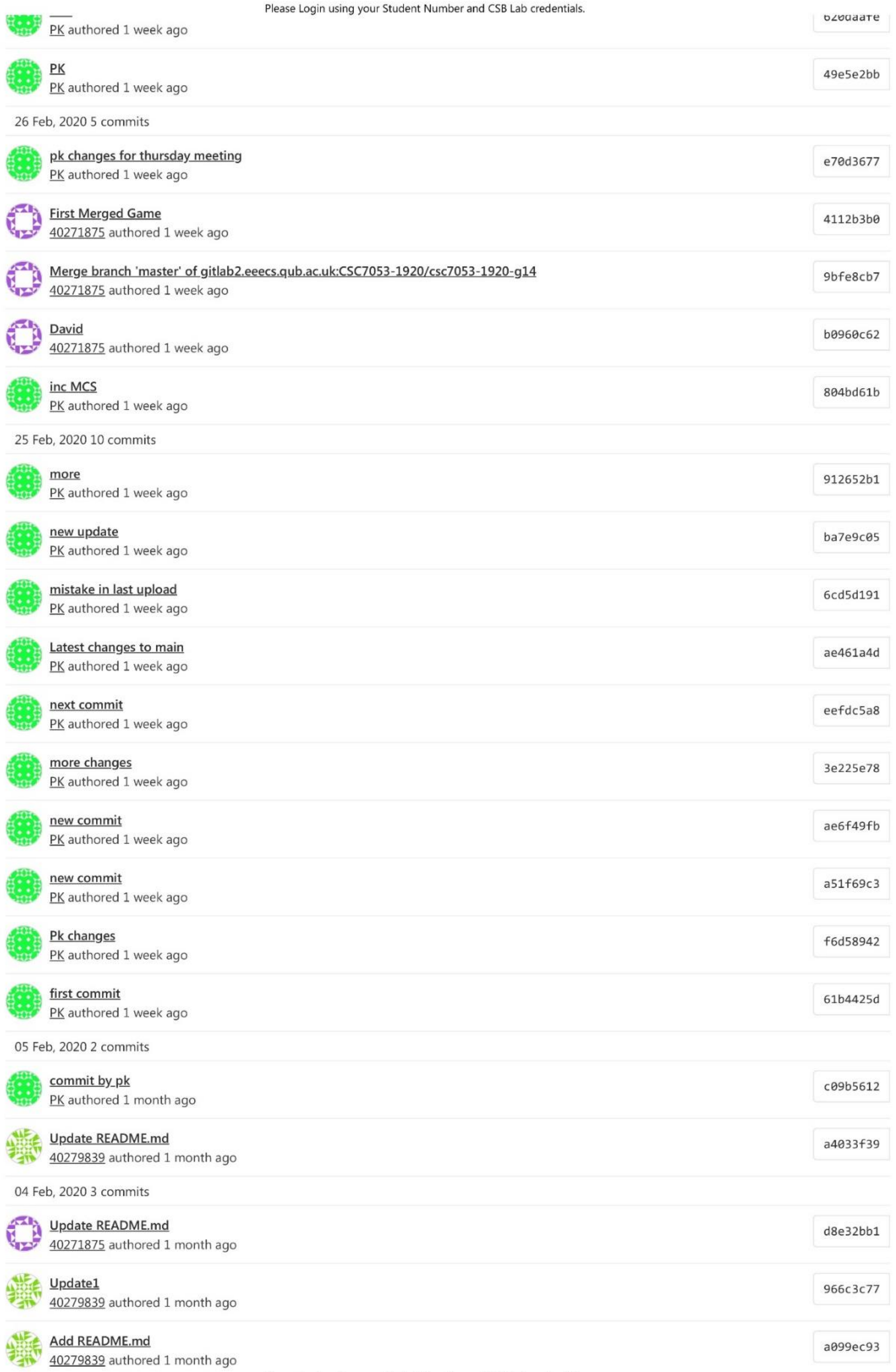
Please Login using your Student Number and CSB Lab credentials.

https://gitlab2.eeecs.qub.ac.uk/CSC7053-1920/csc7053-1920-g14/commits/master 2/4

Please Login using your Student Number and CSB Lab credentials.

10/03/2020 Commits · master · CSC7053-1920 / CSC7053-1920-G14 · GitLab

Please Login using your Student Number and CSB Lab credentials.



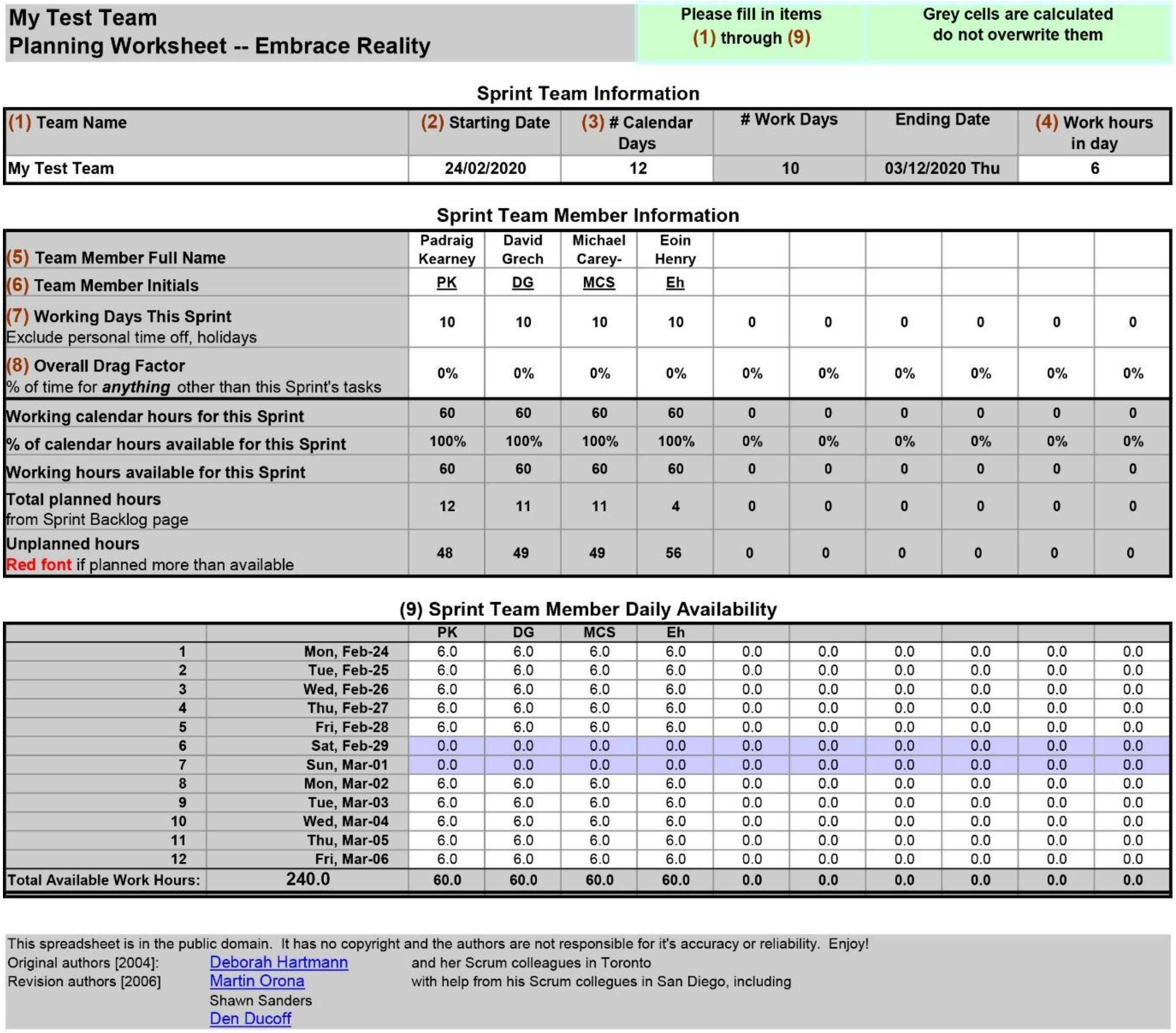
PK authored 1 week ago	620daafe
PK PK authored 1 week ago	49e5e2bb
26 Feb, 2020 5 commits	
pk changes for thursday meeting PK authored 1 week ago	e70d3677
First Merged Game 40271875 authored 1 week ago	4112b3b0
Merge branch 'master' of gitlab2.eeecs.qub.ac.uk:CSC7053-1920/csc7053-1920-g14 40271875 authored 1 week ago	9bfe8cb7
David 40271875 authored 1 week ago	b0960c62
inc MCS PK authored 1 week ago	804bd61b
25 Feb, 2020 10 commits	
more PK authored 1 week ago	912652b1
new update PK authored 1 week ago	ba7e9c05
mistake in last upload PK authored 1 week ago	6cd5d191
Latest changes to main PK authored 1 week ago	ae461a4d
next commit PK authored 1 week ago	eefdc5a8
more changes PK authored 1 week ago	3e225e78
new commit PK authored 1 week ago	ae6f49fb
new commit PK authored 1 week ago	a51f69c3
Pk changes PK authored 1 week ago	f6d58942
first commit PK authored 1 week ago	61b4425d
05 Feb, 2020 2 commits	
commit by pk PK authored 1 month ago	c09b5612
Update README.md 40279839 authored 1 month ago	a4033f39
04 Feb, 2020 3 commits	
Update README.md 40271875 authored 1 month ago	d8e32bb1
Update1 40279839 authored 1 month ago	966c3c77
Add README.md 40279839 authored 1 month ago	a099ec93

Please Login using your Student Number and CSB Lab credentials.

Appendix 4 [DG][PK]

Hartmann Orona Spreadsheet

Page 1:



My Test Team
Planning Worksheet -- Embrace Reality

Please fill in items (1) through (9)

Grey cells are calculated do not overwrite them

Sprint Team Information

(1) Team Name	(2) Starting Date	(3) # Calendar Days	# Work Days	Ending Date	(4) Work hours in day
My Test Team	24/02/2020	12	10	03/12/2020 Thu	6

Sprint Team Member Information

(5) Team Member Full Name	Padraig Kearney	David Grech	Michael Carey-	Eoin Henry						
(6) Team Member Initials	PK	DG	MCS	Eh						
(7) Working Days This Sprint Exclude personal time off, holidays	10	10	10	10	0	0	0	0	0	0
(8) Overall Drag Factor % of time for *anything* other than this Sprint's tasks	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Working calendar hours for this Sprint	60	60	60	60	0	0	0	0	0	0
% of calendar hours available for this Sprint	100%	100%	100%	100%	0%	0%	0%	0%	0%	0%
Working hours available for this Sprint	60	60	60	60	0	0	0	0	0	0
Total planned hours from Sprint Backlog page	12	11	11	4	0	0	0	0	0	0
Unplanned hours Red font if planned more than available	48	49	49	56	0	0	0	0	0	0

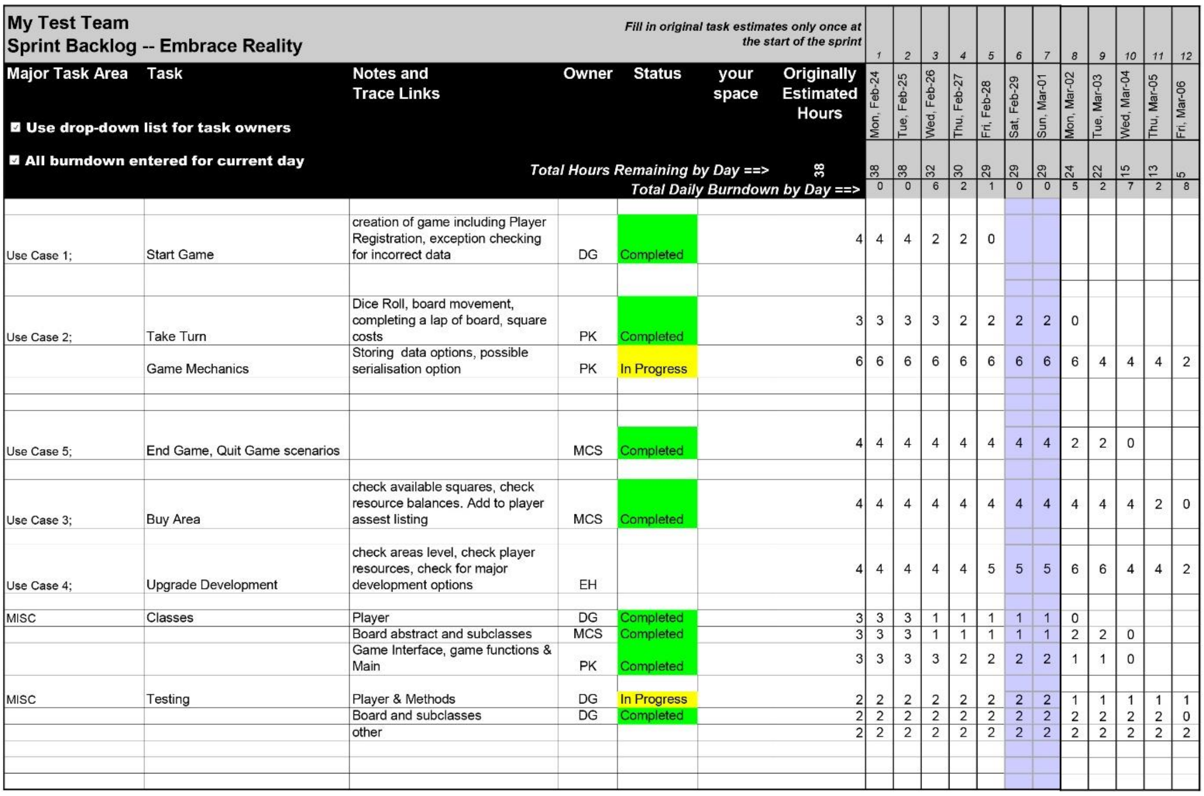
(9) Sprint Team Member Daily Availability

		PK	DG	MCS	Eh						
1	Mon, Feb-24	6.0	6.0	6.0	6.0	0.0	0.0	0.0	0.0	0.0	0.0
2	Tue, Feb-25	6.0	6.0	6.0	6.0	0.0	0.0	0.0	0.0	0.0	0.0
3	Wed, Feb-26	6.0	6.0	6.0	6.0	0.0	0.0	0.0	0.0	0.0	0.0
4	Thu, Feb-27	6.0	6.0	6.0	6.0	0.0	0.0	0.0	0.0	0.0	0.0
5	Fri, Feb-28	6.0	6.0	6.0	6.0	0.0	0.0	0.0	0.0	0.0	0.0
6	Sat, Feb-29	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	Sun, Mar-01	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	Mon, Mar-02	6.0	6.0	6.0	6.0	0.0	0.0	0.0	0.0	0.0	0.0
9	Tue, Mar-03	6.0	6.0	6.0	6.0	0.0	0.0	0.0	0.0	0.0	0.0
10	Wed, Mar-04	6.0	6.0	6.0	6.0	0.0	0.0	0.0	0.0	0.0	0.0
11	Thu, Mar-05	6.0	6.0	6.0	6.0	0.0	0.0	0.0	0.0	0.0	0.0
12	Fri, Mar-06	6.0	6.0	6.0	6.0	0.0	0.0	0.0	0.0	0.0	0.0
Total Available Work Hours:	240.0	60.0	60.0	60.0	60.0	0.0	0.0	0.0	0.0	0.0	0.0

This spreadsheet is in the public domain. It has no copyright and the authors are not responsible for it's accuracy or reliability. Enjoy!

Original authors [2004]:	Deborah Hartmann	and her Scrum colleagues in Toronto
Revision authors [2006]	Martin Orona	with help from his Scrum collegues in San Diego, including
	Shawn Sanders	
	Den Ducoff	

Page 2:



My Test Team
Sprint Backlog -- Embrace Reality

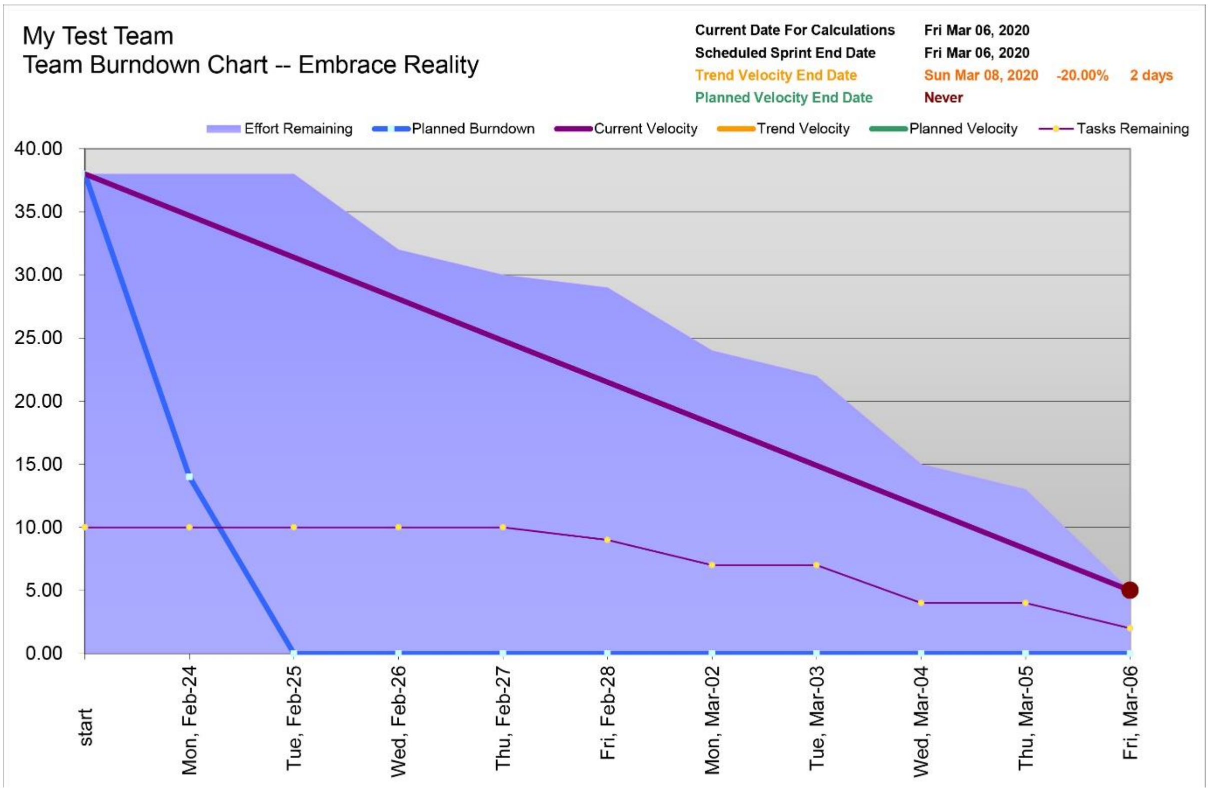
Fill in original task estimates only once at the start of the sprint

☑ **Use drop-down list for task owners**

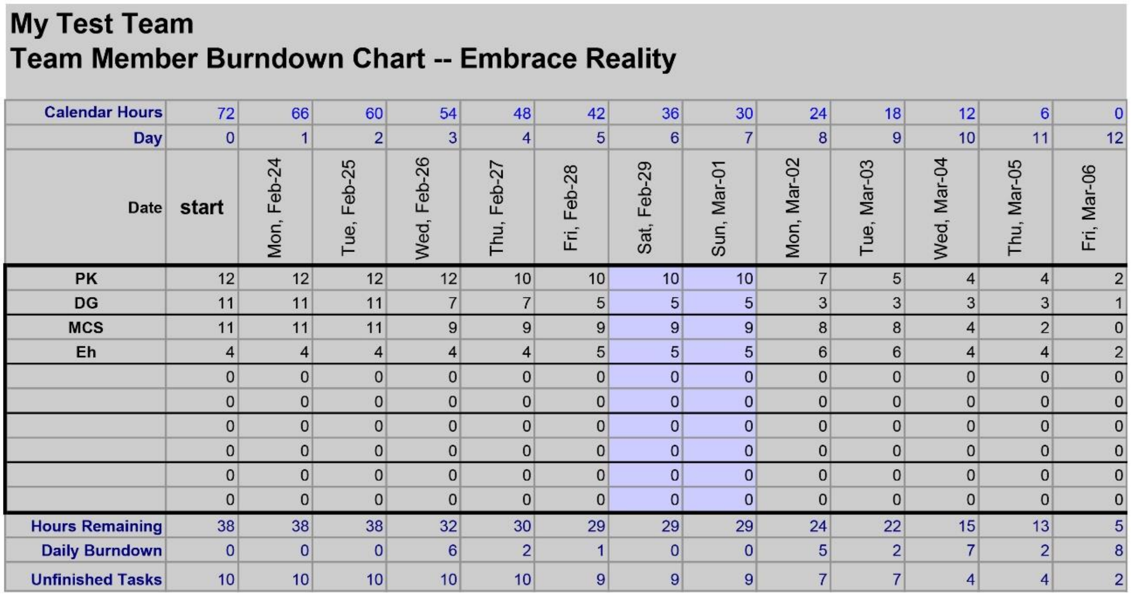
☑ **All burndown entered for current day**

Major Task Area	Task	Notes and Trace Links	Owner	Status	your space	Originally Estimated Hours	1 Mon, Feb-24	2 Tue, Feb-25	3 Wed, Feb-26	4 Thu, Feb-27	5 Fri, Feb-28	6 Sat, Feb-29	7 Sun, Mar-01	8 Mon, Mar-02	9 Tue, Mar-03	10 Wed, Mar-04	11 Thu, Mar-05	12 Fri, Mar-06
					Total Hours Remaining by Day ==>	38	38	38	32	30	29	29	29	24	22	15	13	5
					Total Daily Burndown by Day ==>		0	0	6	2	1	0	0	5	2	7	2	8
Use Case 1;	Start Game	creation of game including Player Registration, exception checking for incorrect data	DG	Completed		4	4	4	2	2	0							
Use Case 2;	Take Turn	Dice Roll, board movement, completing a lap of board, square costs	PK	Completed		3	3	3	3	2	2	2	2	0				
	Game Mechanics	Storing data options, possible serialisation option	PK	In Progress		6	6	6	6	6	6	6	6	6	4	4	4	2
Use Case 5;	End Game, Quit Game scenarios		MCS	Completed		4	4	4	4	4	4	4	4	2	2	0		
Use Case 3;	Buy Area	check available squares, check resource balances. Add to player assest listing	MCS	Completed		4	4	4	4	4	4	4	4	4	4	4	2	0
Use Case 4;	Upgrade Development	check areas level, check player resources, check for major development options	EH			4	4	4	4	4	5	5	5	6	6	4	4	2
MISC	Classes	Player	DG	Completed		3	3	3	1	1	1	1	1	0				
		Board abstract and subclasses	MCS	Completed		3	3	3	1	1	1	1	1	2	2	0		
		Game Interface, game functions & Main	PK	Completed		3	3	3	3	2	2	2	2	1	1	0		
MISC	Testing	Player & Methods	DG	In Progress		2	2	2	2	2	2	2	2	1	1	1	1	1
		Board and subclasses	DG	Completed		2	2	2	2	2	2	2	2	2	2	2	2	0
		other				2	2	2	2	2	2	2	2	2	2	2	2	2

Page 3:

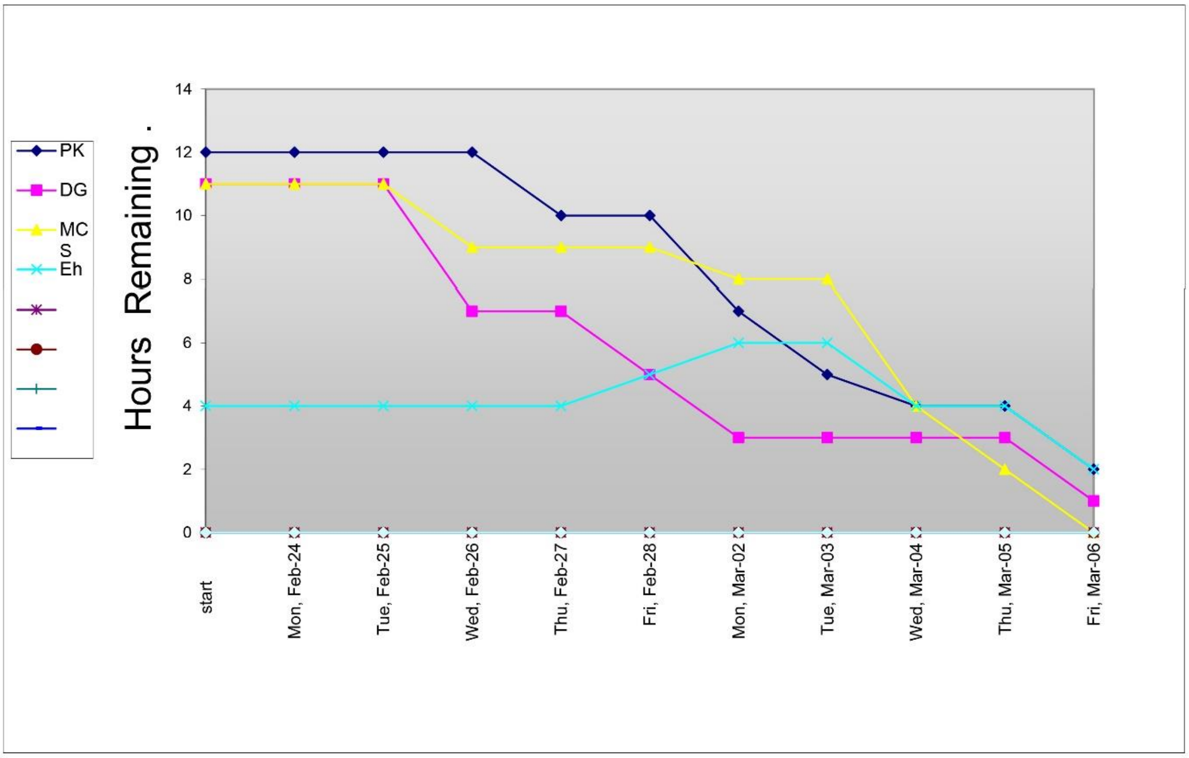


Page 4:



My Test Team
Team Member Burndown Chart -- Embrace Reality

Calendar Hours	72	66	60	54	48	42	36	30	24	18	12	6	0
Day	0	1	2	3	4	5	6	7	8	9	10	11	12
Date	start	Mon, Feb-24	Tue, Feb-25	Wed, Feb-26	Thu, Feb-27	Fri, Feb-28	Sat, Feb-29	Sun, Mar-01	Mon, Mar-02	Tue, Mar-03	Wed, Mar-04	Thu, Mar-05	Fri, Mar-06
PK	12	12	12	12	10	10	10	10	7	5	4	4	2
DG	11	11	11	7	7	5	5	5	3	3	3	3	1
MCS	11	11	11	9	9	9	9	9	8	8	4	2	0
Eh	4	4	4	4	4	5	5	5	6	6	4	4	2
	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0
Hours Remaining	38	38	38	32	30	29	29	29	24	22	15	13	5
Daily Burndown	0	0	0	6	2	1	0	0	5	2	7	2	8
Unfinished Tasks	10	10	10	10	10	9	9	9	7	7	4	4	2



During the two weeks we held several meetings, mainly on Mondays and Wednesday. In addition to several informal where we discussed progress on everyone assigned parts of the project. Before “Kicking off” the sprint, in the week previous we tried to finalise our designs of our UML case diagrams, sequence and Class in order for minimal changes to be made during the period except for in cases where changes was required due to coding complexity or impracticality. We tried to identify innovate solutions to many of the challenges in getting a minimum viable system in operation.

As per our Hartmann Orona diagram, we identified key activities from our UML requirement analysis and created tasks build on achieving these objectives in order to fulfil the customers’ requirements. Each scrum member then took on the role of achieving these goals. Initially we tried to estimate how much time would be required to complete each task and after making slow progress during the first week of the sprint, where some tasks started to prove more difficult than others. Possibly some task where underestimated in level of difficulty. We started to make rapid progress as a group and individuals at the beginning of week two and by Friday had a working model. Albeit with a couple of remaining less important items outstanding but they were easily sorted once most of the system was ready and junit testing activities where performed to identify potential areas of concern.