# Settlers of Catan

JH Project - Group C

18th April 2017

Supervisor: Alice Toniolo

University of St Andrews

George Alexander

Taylor Alexander

Iona Gilbraith

Anna Grosul

**Abstract**

The rise of computer gaming has seen many popular board games translated for the online community. Adjusting to this popular trend, the class was given the objective of taking the famous board game *Settlers of Catan* and adapting it to be played on a computer. Every team, group of four to five people, was to separately build a version of the game board or an AI agent or both. We, Group C, created an implementation of the *Settlers of Catan* game board which other teams' AI agents could play on. This report details our aims and objectives, in addition to our design and implementation.

## 1 Declaration

We declare that the material submitted for assessment is our own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated. The main text of this project report is 9,469 words long, including project specification and plan. In submitting this project report to the University of St Andrews, we give permission for it to be made available for use in accordance with the regulations of the University Library. We also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. We retain the copyright in this work, and ownership of any resulting intellectual property.

# Contents

## 2 Introduction

Over the past few decades, the ability to play popular games online has been a rising trend. Traditional games such as poker, chess, monopoly etc. have obtained a plethora of programs available on the net, that users can play with virtually. The project objective is to create a software representing the popular game *Settlers of Catan*, which was created in 1995 by Franckh-Kosmos Verlag. The purpose of this paper is to provide insight about the implementation methods used to convert the physical board game, *Settlers of Catan*, into a multi-player online game.

The computer science class was divided into small groups of either four or five students. Both the Single Honours and Joint Honours computer science students were to collaborate together, allowing for Single Honours teams' Artificial Intelligence (AI) to be connected to any game board.

Since this group is comprised of four Joint Honours students, the amount of work required was to be reduced. This Joint Honours team decided to focus on the game logic, or the board/server, of the game rather than the AI. As such, the elements we were required to make as part of our submission were a board containing all of the rules and setup required for the game, a User Interface allowing this board to be visually represented to players, allowing them to see where they have placed roads, settlements etc. and which resources are available from which hex, and networking capabilities, allowing multiple human or Single Honours groups' AI agents to play simultaneously on the board across different machines.

A brief outline of what this paper contains is as follows. Section 3 will outline the details of the submission, what files are available and how to run the files. Section 4 will discuss the different technologies used to complete the project as well as the design decisions and class structure. Section 5 will provide a detailed overview of the actual implementation of the software. Section 6 will briefly provide information about the user interface. Section 7 and 8 are more about the collaboration between the team members and type of organization structure that the team used. Finally Section 9 will provide a self evaluation of the team compared to other available online platforms for *Settlers of Catan* and section 10 will show testing results.

# 3 Project Details

## 3.1 Submission Overview

### 3.1.1 Included in Submission

In our submission we have a **“code”** directory, which contains all the source code that we have created throughout the project. This consists of the following packages:

- **“completedCatan”**, which contains a working version of our project. This allows you to play *Settlers of Catan* across a network using only our version of the game. **“internal-Catan.jar”** is the runnable jar file associated with this code.
- **“game”**, which contains a more developed version of the project, allowing some inter-group functionality, with **“externalCatan.jar”** as the runnable jar file.
- **“projectX”**, which contains the networking version of our project, but with a more extensive UI. **“GUICatan”** can be used to run this version of the project.
- **“protobuf”**, which contains the “.proto” files from inter-group protocol used for the project.
- **“intergroup”**. which contains the “.java” files compiled from the relevant “.proto” file.

Our code is dependent on the jar file **“protobuf-java-3.1.0.jar”**, which has also been included. The files **“ExampleTextLayout”** and **“TextLayout”** contain the ASCII board that we use for our UI, both in Java form, and printed form. **“Test game”** contains output from a test session of the game itself, and **“test.txt”** contains some test input. We have also submitted our group report, under the file name **“Report.pdf”**, as well as our own personal reports, given as **“PersonalReport.pdf”**.

### 3.1.2 How to Run

To compile and run this project, open the files as a Java project in the Eclipse IDE and run the “Catan.java” file in either the completedCatan package, game package, or projectX package, depending on the version you would like to examine. Otherwise, use the “xx.jar” files associated with the version you would like to use, and type the following command into command line to run the game: “java -jar xx.jar”.

# 4 Architecture

Our architecture is a Client-Server implementation, which consists of a simplistic front-end, and the all of the computation on the server. We focused primarily on ensuring the game logic was complete and correct, before connecting it up to clients using our own internal networking, and then expanding to Protocol Buffers. Here, “game logic” refers to our implementation of the rules of *Settlers of Catan* itself[5], such as how to set up the board before play, how players are allowed to trade, how resources are distributed etc.

## 4.1 Technology

First, we had to decide on what technology would be used for our implementation. As we are a Joint Honours team, we had very little experience of programming languages apart from Java and C. This immediately ruled out implementing our game as a web application, since we were anxious about using a new language in such a short time period and small scope of credits awarded to us. We decided to use Java, as we felt that its flexibility would be of more use to us and we were more familiar with using it for this project.

### 4.1.1 Eclipse

We decided on using Eclipse as our IDE [3]. Again, this was used since all members of the team had previously used Eclipse and so we felt it would be much easier to continue using the same tools. Also, using an IDE itself proved to be very helpful, as it was more trivial to compile and run the program rather than having to build from the command line. The visual elements of Eclipse were also useful for us to test the program.

### 4.1.2 NetBeans

We also used the NetBeans IDE at the end of this project, since its “drag-and-drop” features were found to be very useful in implementing our Graphical User Interface [4].

### 4.1.3 Protocol Buffers

A “Protocol Buffer” is a way of converting data into a standard structure, regardless of original syntax, language etc.[2] Protocol Buffers were needed to ensure that, regardless of how each group decided to implement their own game logic, they were able to communicate and play on other groups’ boards, as all necessary data could be converted to a standard format.

As part of the collaboration between the groups, the decision was made to use Protocol Buffers to send information between the server and the clients. Since there are many different board states and pieces of information in *Settlers of Catan*, using JSON to send updates would require a large file design to be decided on, which would need to be sent every time an action is performed. This seemed wasteful, and hence the idea for Protocol Buffers was introduced. Protocol Buffers were much simpler to use than JSON, as we only needed to send the new information between the server and the clients, and so it was a lot more efficient. Since different teams were using a variety of coding languages, it also seemed Protocol Buffers were the logical choice as they were language-neutral.

## 4.2 Back-end architecture

The use of Java allowed us to take an object-oriented approach to the architecture of the project. This allowed us to split the tasks a lot better, as each member could focus on writing methods in specific classes. This meant that our work was mostly independent of each others', and so when committing changes to GitHub, there were very little merge conflicts.

We discussed very early on what classes we felt would be needed throughout our implementation, and began with these. This created a very good starting point for our code. As we progressed, we added to these classes, and also creating new ones during refactoring where needed. This ensured all methods were located in the correct places, as well as keeping classes short and readable. Our "Catan" class is where our "main" method is situated. It creates a "Game" and a "Board" object. The initial setup uses these objects, to ensure everything is instantiated before the main game has begun. A while loop was then run, in which each pass, a "Turn" object is created to let each player have their turn. This continues until a player has won the game.

For the purposes of our implementation, the following definitions apply. "Game" mainly refers to the state of the bank, keeping track of how many resource cards etc. are available at a particular moment during the game. "Board" refers to the state of the board used for gameplay, but not the elements relating to players, i.e. the board stores which resources are available at each hex, the robber's location, and which number needs to be rolled for players to gather a certain hex's resource. "Player" keeps track of the information relevant to each individual player, storing which resource and development cards they currently have in their hand, where they have settlements and roads built, and if they own any ports.

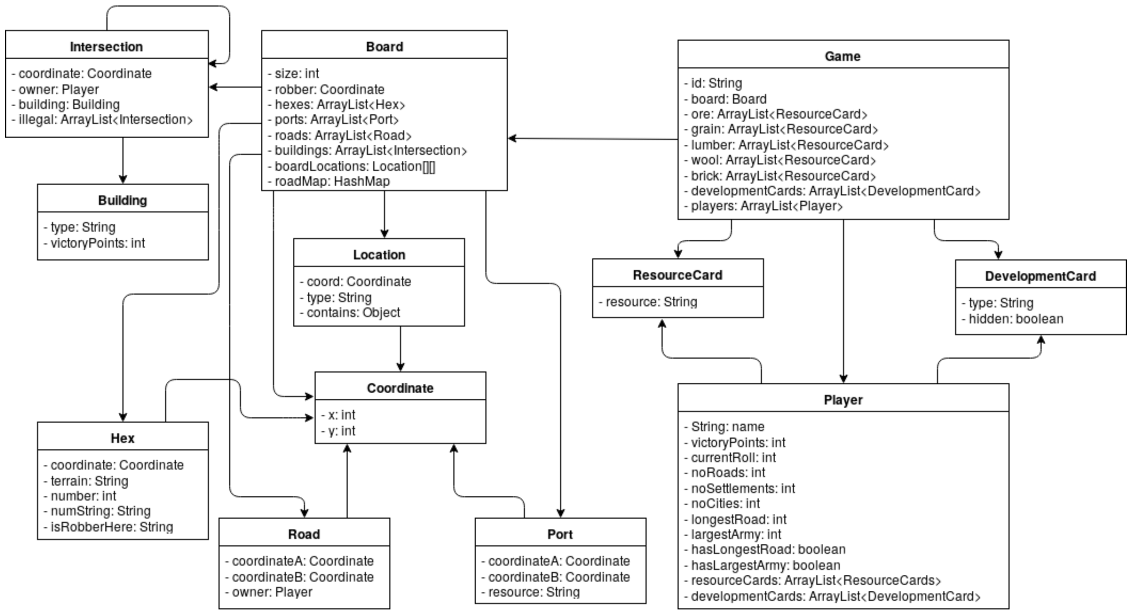


Figure 1: The original classes implemented

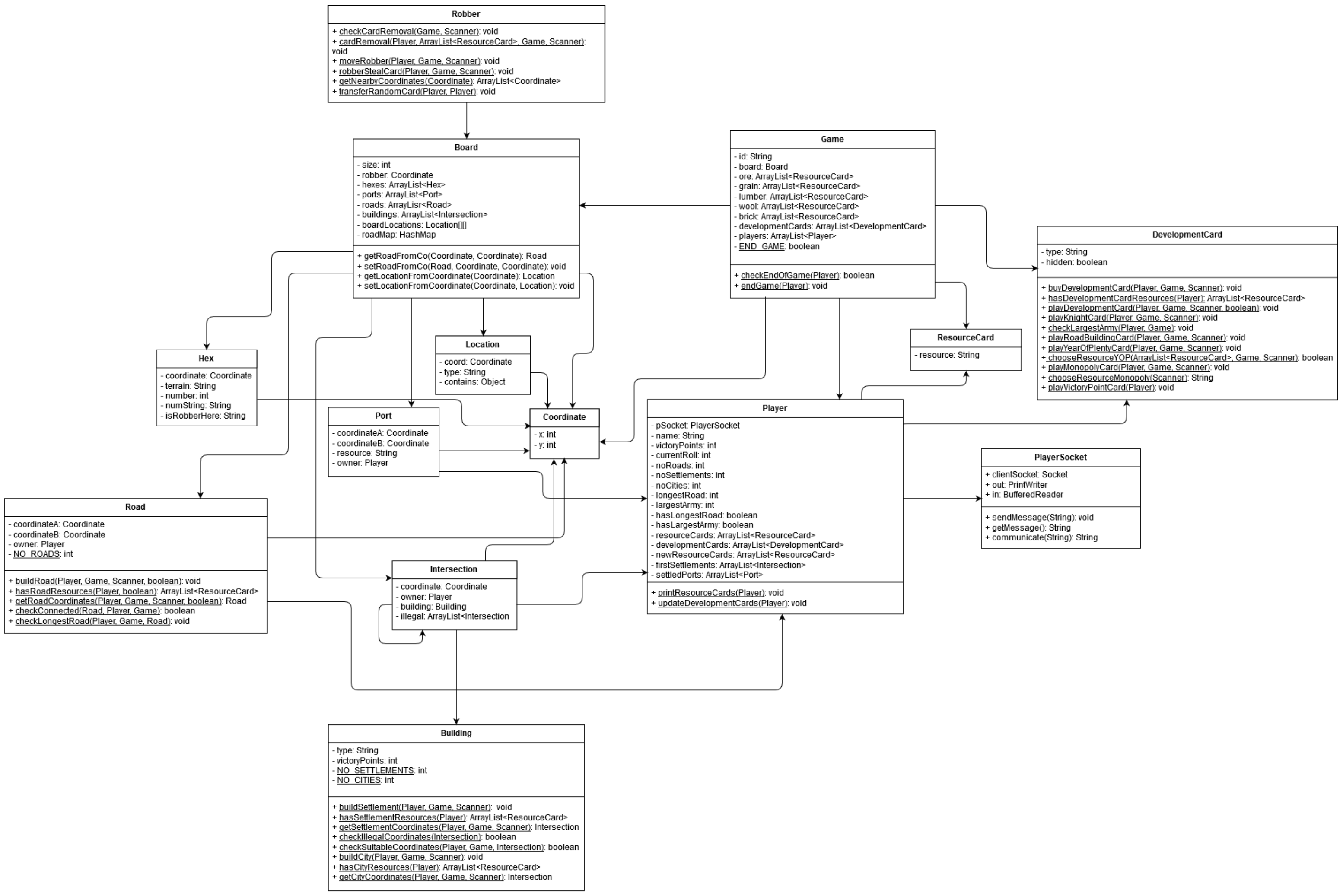

Robber
+ checkCardRemoval(Game, Scanner): void
+ cardRemoval(Player, ArrayList<ResourceCard>, Game, Scanner): void
+ moveRobber(Player, Game, Scanner): void
+ robberStealCard(Player, Game, Scanner): void
+ getNearbyCoordinates(Coordinate): ArrayList<Coordinate>
+ transferRandomCard(Player, Player): void
Board
- size: int
- robber: Coordinate
- hexes: ArrayList<Hex>
- ports: ArrayList<Port>
- roads: ArrayLisr<Road>
- buildings: ArrayList<Intersection>
- boardLocations: Location[][]
- roadMap: HashMap
+ getRoadFromCo(Coordinate, Coordinate): Road
+ setRoadFromCo(Road, Coordinate, Coordinate): void
+ getLocationFromCoordinate(Coordinate): Location
+ setLocationFromCoordinate(Coordinate, Location): void
Game
- id: String
- board: Board
- ore: ArrayList<ResourceCard>
- grain: ArrayList<ResourceCard>
- lumber: ArrayList<ResourceCard>
- wool: ArrayList<ResourceCard>
- brick: ArrayList<ResourceCard>
- developmentCards: ArrayList<DevelopmentCard>
- players: ArrayList<Player>
- END_GAME: boolean
+ checkEndOfGame(Player): boolean
+ endGame(Player): void
DevelopmentCard
- type: String
- hidden: boolean
+ buyDevelopmentCard(Player, Game, Scanner): void
+ hasDevelopmentCardResources(Player): ArrayList<ResourceCard>
+ playDevelopmentCard(Player, Game, Scanner, boolean): void
+ playKnightCard(Player, Game, Scanner): void
+ checkLargestArmy(Player, Game): void
+ playRoadBuildingCard(Player, Game, Scanner): void
+ playYearOfPlentyCard(Player, Game, Scanner): void
+ chooseResourceYOP(ArrayList<ResourceCard>, Game, Scanner): boolean
+ playMonopolyCard(Player, Game, Scanner): void
+ chooseResourceMonopoly(Scanner): String
+ playVictoryPointCard(Player): void
Hex
- coordinate: Coordinate
- terrain: String
- number: int
- numString: String
- isRobberHere: String
Location
- coord: Coordinate
- type: String
- contains: Object
ResourceCard
- resource: String
Port
- coordinateA: Coordinate
- coordinateB: Coordinate
- resource: String
- owner: Player
Coordinate
- x: int
- y: int
Player
- pSocket: PlayerSocket
- name: String
- victoryPoints: int
- currentRoll: int
- noRoads: int
- noSettlements: int
- noCities: int
- longestRoad: int
- largestArmy: int
- hasLongestRoad: boolean
- hasLargestArmy: boolean
- resourceCards: ArrayList<ResourceCard>
- developmentCards: ArrayList<DevelopmentCard>
- newResourceCards: ArrayList<ResourceCard>
- firstSettlements: ArrayList<Intersection>
- settledPorts: ArrayList<Port>
+ printResourceCards(Player): void
+ updateDevelopmentCards(Player): void
PlayerSocket
+ clientSocket: Socket
+ out: PrintWriter
+ in: BufferedReader
+ sendMessage(String): void
+ getMessage(): String
+ communicate(String): String
Road
- coordinateA: Coordinate
- coordinateB: Coordinate
- owner: Player
- NO_ROADS: int
+ buildRoad(Player, Game, Scanner, boolean): void
+ hasRoadResources(Player, boolean): ArrayList<ResourceCard>
+ getRoadCoordinates(Player, Game, Scanner, boolean): Road
+ checkConnected(Road, Player, Game): boolean
+ checkLongestRoad(Player, Game, Road): void
Intersection
- coordinate: Coordinate
- owner: Player
- building: Building
- illegal: ArrayList<Intersection
Building
- type: String
- victoryPoints: int
- NO_SETTLEMENTS: int
- NO_CITIES: int
+ buildSettlement(Player, Game, Scanner): void
+ hasSettlementResources(Player): ArrayList<ResourceCard>
+ getSettlementCoordinates(Player, Game, Scanner): Intersection
+ checkIllegalCoordinates(Intersection): boolean
+ checkSuitableCoordinates(Player, Game, Intersection): boolean
+ buildCity(Player, Game, Scanner): void
+ hasCityResources(Player): ArrayList<ResourceCard>
+ getCityCoordinates(Player, Game, Scanner): Intersection

Figure 2: The final version of our program, showing the associations between the important classes

## 4.3 Front-end architecture

We felt that our front-end should be kept as simple as possible. This meant that the entire state of the board could be kept on the server, and whenever clients wanted information they would request it from there. This ensured that there would be no inconsistencies between server and clients in terms of having to send updates between them, and hence the clients would never be out of sync with each other. Due to this, our front-end was relatively non-existent. We relied on the console as our “client”, except from in the package “projectX” which contained an extended version of our UI. This was then further extended for use with the inter-team protocol, where the clients needed some game state information, although we still tried to keep the client as simple as possible.

## 4.4 Client-Server Interaction

Throughout the progression of the project, the design of the client-server implementation was changed. The initial design had to be modified once the program was ready to connect with the inter-team AI (Artificial Intelligence) software. Once this step was approaching, client-server implementation had to be modified to fit the inter-team protocol. The two figures in this section outline the design of both the original client-server implementation and the modified design.

Originally, the logic of the game, along with Player 1, would act as the server for the game, where Player 1 would be the game "host". The rest of the players were the clients. Messages between the server and clients would be transferred using socket connections and sent via string inputs and outputs. As the game progressed, the implementation changed. Player 1 would still assume the role of "host", and set up the server, where they would then change into being a client. The game logic remained as the server and the other players remained as clients. Messages are now sent through Protocol Buffer socket connections. It is important to note, in the second client-server implementation all clients keep track of the board status and the personal resources that they have obtained. However, the resources of other players are unknown to each individual player.

Sockets were used in order to communicate between the clients and the main server. Since we were using Java to write our front and back end, we felt that it made sense to use the "Java.net" library for our networking. Each client had a socket associated with them, so this made it easy to send information to select people. When asking for a user to roll the dice, or asking what action they wanted to perform, we could send this directly to that player, without having to send the message to everyone else. This meant that when a player performed an action, we could send a separate message to the other players, informing them of the player's action. All our messages were initially sent as strings which were printed to the relevant client by a PrintWriter. Input from clients were also sent back as Strings and read in by a BufferedReader. This made the communication very simple to begin with.

Once we were sure our networking performed as expected, we had to change the way our messages were sent and received. Since our board allows external AI to connect, we had to adhere to the inter-group protocol that was decided. This involved using "Protocol Buffers" which allowed clients to request an action and the server to grant or deny the action. At first, we believed that this would work well with our program, as our server does all the computation. During our conversion to Protocol Buffers, however, we noticed some messages that we send back and forth were not implemented in the protocol. This is due to the fact that the protocol relied on some data being preserved by the clients, so the clients did not always have to request information on the state of the board from the server. This made it a lot more difficult for us to use the protocol, since we than had to change a vast majority of our original code.

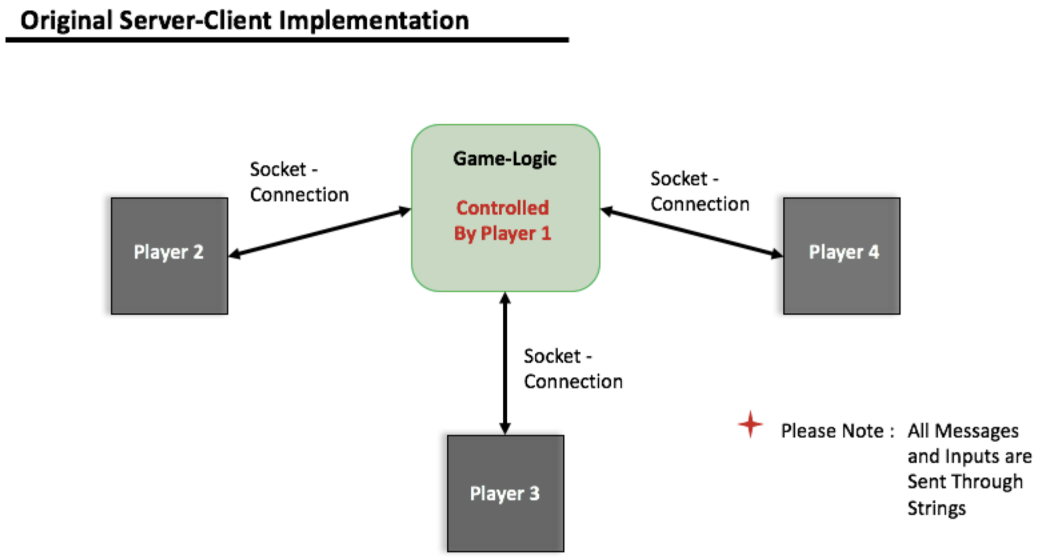


Figure 3: Initial Client Server Design

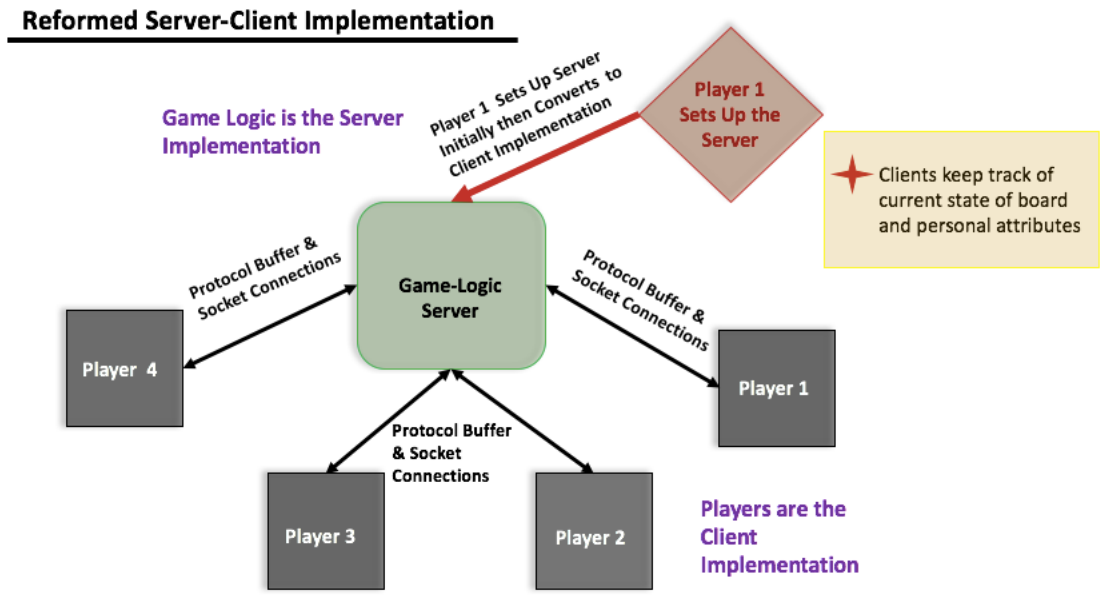

Reformed Server-Client Implementation
Game Logic is the Server Implementation
Player 1 Sets Up Server Initially then Converts to Client Implementation
Player 1 Sets Up the Server
Clients keep track of current state of board and personal attributes
Protocol Buffer & Socket Connections
Game-Logic Server
Protocol Buffer & Socket Connections
Player 4
Player 1
Protocol Buffer & Socket Connections
Player 3
Player 2
Players are the Client Implementation

Figure 4: Modified Client Server Design

## 5 Implementation Details

We began our implementation by writing pseudocode for most rules. This allowed us to easily figure out the expected difficulty for each part of the project, making it easier to split the work evenly. At this point we also decided on what data structures we would use, so that all sections of code would fit together. Once the tasks were split, we each worked on the implementation, with bigger tasks being assigned to pairs and smaller parts being individual projects.

A key part of the implementation was ensuring our program contained a correct set of rules. If this was not completed properly, our team's state of the board would be out of sync with other teams', which may cause large errors during gameplay. We achieved this by ensuring the structure of *Settlers of Catan* was preserved. This was done by splitting the implementation into two sections: initial game setup and the player turns itself. Having these two parts of the game completely separate really helped in making the program as true to the game as possible.

A challenging part of the project was designing an algorithm that would help us to calculate a player's longest road. We describe our implementation of this below.

## 5.1 Data Structures

Initially, the requirements upon the data structures, in terms of the way that we wanted to use them in different parts of the logic, were quite varied. Our requirements included the ability to set up the board in an efficient way, generate hex arrangements, and iterate through hexes, roads, and intersections individually in such a way that they could be in the correct order to be printed out in a simple Java stringBuilder. In addition to this, for the sake of logical calculation, it was required that all hexes, roads etc. be efficiently accessible through a coordinate system that was as intuitive as possible.

The coordinate system that we used is the one agreed upon in the inter-group protocol, where there are two axes: a y-axis vertically, with zero in the centre of the board, and an x-axis at a slant, running through the top right and bottom left vertices of the central hex. These competing requirements were achieved by initially creating ArrayLists of hexes, ports, roads and intersections. We then carefully added elements to these arrays in such a way so that their ordering was not reliant on x and y coordinates in our tilted coordinate system but by how they would be ordered for a massive view of a Catan board, printing out all elements in rows, starting at the top left and finishing bottom right. Having thus satisfied the requirement for easy printability, we then created two additional data structures referencing these exact same elements to facilitate access to our hexes and roads and so on, through the tilted (x, y) coordinate system.

Firstly a HashMap was created, and into this was put a road, mapped by a string consisting of four integers representing the coordinates of the road's endpoints. To get a road from two coordinates, the method converted the two coordinates into this string, and also ensured that the order of the coordinates would be consistent for all calls to the HashMap. For the hexes and intersections, a 2 dimensional array containing an object called "boardLocation", that could have as a variable either a hex or an intersection, was created. By iterating through the previously created ArrayLists, all the received hexes and intersections containing "boardLocations" were placed into the 2 dimensional array at a position corresponding to their coordinates. Access to this array was controlled by get and set methods, which automatically adjusted the coordinates by a simple offset to avoid negative indices. Overall, we felt these data structures proved to be efficient and flexible for most, if not all, uses and, importantly, did not require an understanding of their structure to use, given the simple get methods created.

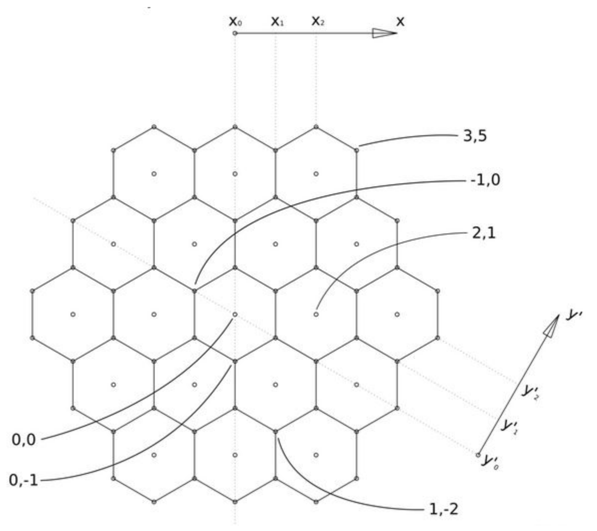


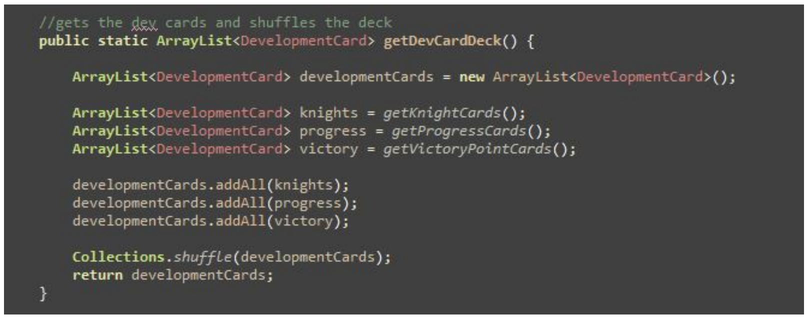
Figure 5: The coordinate system used in the inter-group protocol

## 5.2 Initial Setup

The initial setup consisted of creating a randomised board, making resource and development card decks and shuffling them, and getting initial road and settlement placement.

To create a board state, an empty board of hexes was made. A random arrangement of terrains was created, which was stored in an ArrayList. This allowed the terrains to be looped through, and each hex on the board was given one of these terrains. Once the terrains were set, the values of each hex had to also be randomly set. To do this two shuffled ArrayLists were created, one holding red values (that in the rules of Catan cannot be placed adjacent to one another) and one of the ordinary values. Then a random empty hex is chosen (by cloning the hex ArrayList into a new shuffled ArrayList and iterating), and a red value is placed in that hex, and ordinary values are then placed in all neighbouring hexes. This process is repeated till the red values are exhausted, then the rest of the ordinary values are placed. One of these ordinary values is 7, representing the robber's location, and then the desert tile swaps its terrain value with the terrain value of the tile that seven was placed over, to create a starting position for the robber.

Creating the decks was very simple. They involved first creating an ArrayList of the card type, and then adding the correct number of cards to each. For the resource card decks, this was 19 cards for each type of resource. For development cards, there were 14 "Knight" cards, 5 "Victory Point" cards, 2 "Road Building" cards, 2 "Year of Plenty" cards and 2 "Monopoly" cards. After this, the ordering of the ArrayList can be easily randomised by calling the shuffle method. We could have implemented this by having an infinite deck, but we felt that we should keep the game's rules as similar as possible, and so this included the same number of cards.



```
//gets the dev cards and shuffles the deck
public static ArrayList<DevelopmentCard> getDevCardDeck() {

    ArrayList<DevelopmentCard> developmentCards = new ArrayList<DevelopmentCard>();

    ArrayList<DevelopmentCard> knights = getKnightCards();
    ArrayList<DevelopmentCard> progress = getProgressCards();
    ArrayList<DevelopmentCard> victory = getVictoryPointCards();

    developmentCards.addAll(knights);
    developmentCards.addAll(progress);
    developmentCards.addAll(victory);

    Collections.shuffle(developmentCards);
    return developmentCards;
}
```

Figure 6: Creating the deck of development cards

The last part of the setup is player setup. The players are stored in an ArrayList which makes it easy to iterate through in terms of resource allocation and turns. We allow the players to decide on their token colour by a simple user interaction. Obviously, there are error checks in place in case to ensure each player chose different colours. A player order also has to be decided, which involves each player rolling the dice. The Array List order is then altered by highest dice roll using a simple bubble sort. Now, the players can place their first roads and settlements in the correct order. This involves iterating through the players in the forwards direction for their first road/settlement and then iterating in the backwards direction for the placement of their second road/settlement, which preserves the order of placement from the board game. After each second settlement is placed, the player gets a starting hand of resource cards. This involves checking the hexes that are adjacent to the settlement, and adding the resource card corresponding to that hex to the player's hand.

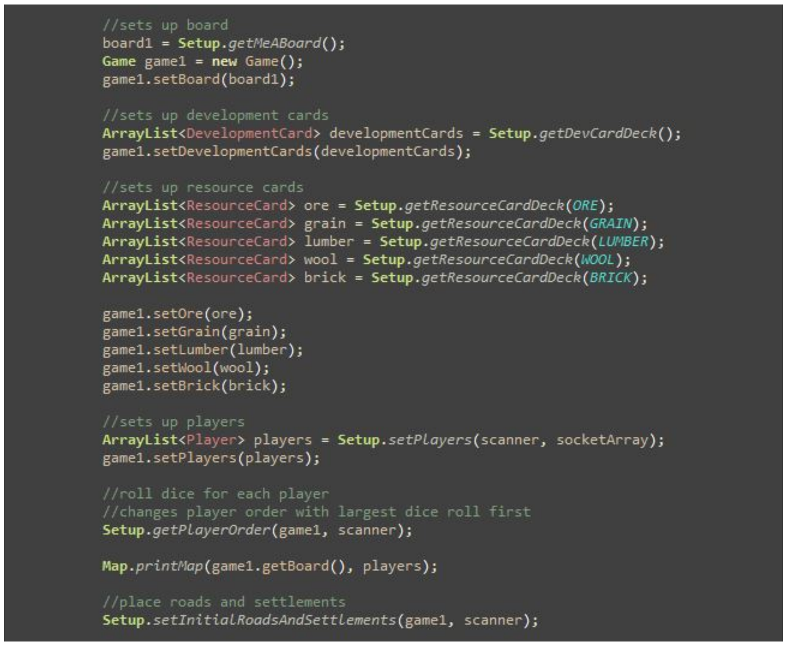

```
//sets up board
board1 = Setup.getMeABoard();
Game game1 = new Game();
game1.setBoard(board1);

//sets up development cards
ArrayList<DevelopmentCard> developmentCards = Setup.getDevCardDeck();
game1.setDevelopmentCards(developmentCards);

//sets up resource cards
ArrayList<ResourceCard> ore = Setup.getResourceCardDeck(ORE);
ArrayList<ResourceCard> grain = Setup.getResourceCardDeck(GRAIN);
ArrayList<ResourceCard> lumber = Setup.getResourceCardDeck(LUMBER);
ArrayList<ResourceCard> wool = Setup.getResourceCardDeck(WOOL);
ArrayList<ResourceCard> brick = Setup.getResourceCardDeck(BRICK);

game1.setOre(ore);
game1.setGrain(grain);
game1.setLumber(lumber);
game1.setWool(wool);
game1.setBrick(brick);

//sets up players
ArrayList<Player> players = Setup.setPlayers(scanner, socketArray);
game1.setPlayers(players);

//roll dice for each player
//changes player order with largest dice roll first
Setup.getPlayerOrder(game1, scanner);

Map.printMap(game1.getBoard(), players);

//place roads and settlements
Setup.setInitialRoadsAndSettlements(game1, scanner);
```

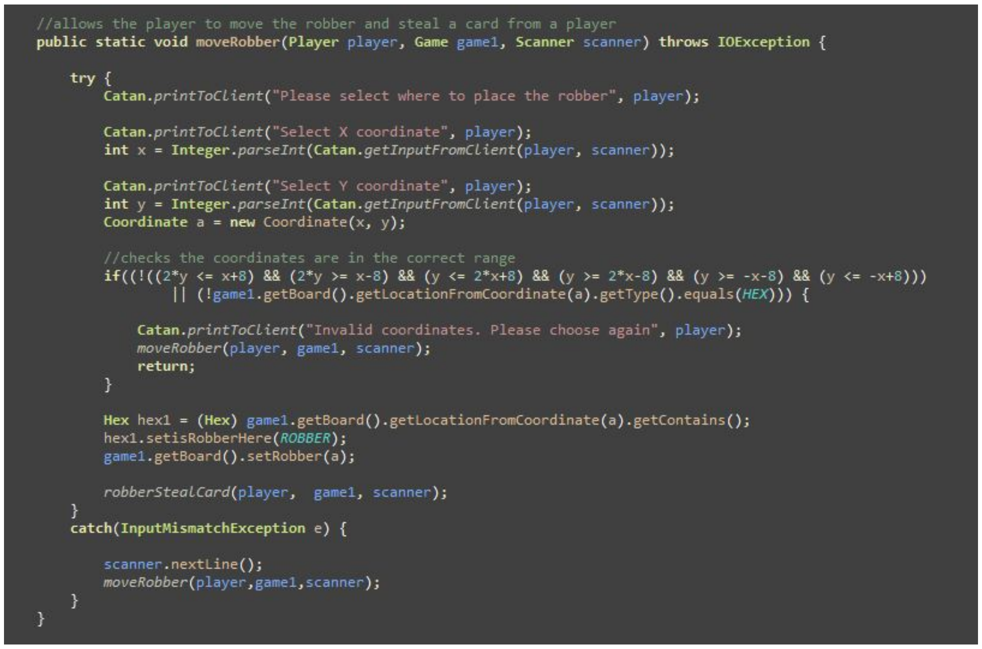
Figure 7: The order of initial setup in our game logic

### 5.3 Player Turn

To allow players to take a turn, we created a separate class, "Turn", from which all other Turn actions could be called. These actions included rolling the dice, moving the robber, resource allocation, building and buying objects, playing development cards and trading.

Dice rolling was very simple. Since the board game has two 6-sided dice, our program generated two random numbers between 1 and 6. This preserved the probabilities of rolling certain numbers, as some "Catan" strategies rely on using high probability values to receive high quantities of resources.

If a 7 was rolled, the player would have to move the robber. We modelled the robber as a coordinate, which specified the current hex that it was occupying. Although discarding cards was quite simple to implement, the nature of our implementation made the action not user-friendly. It would require a player to select 1 card to remove, and this would be repeated until half of their cards were removed. The current player was then prompted for coordinates to move the robber, which were checked to ensure they corresponded to a legal hex. If so, the robber would be moved, and the player could chose a player to steal a card from. This player had to have a settlement adjacent to a the hex, and so appropriate checks were made to ensure this happened. We also had to implement the condition that you could not steal from yourself, as that would not have been a legal move. Since the player did not choose which card to steal, it made this slightly easier since we could just transfer a random resource card between player hands.

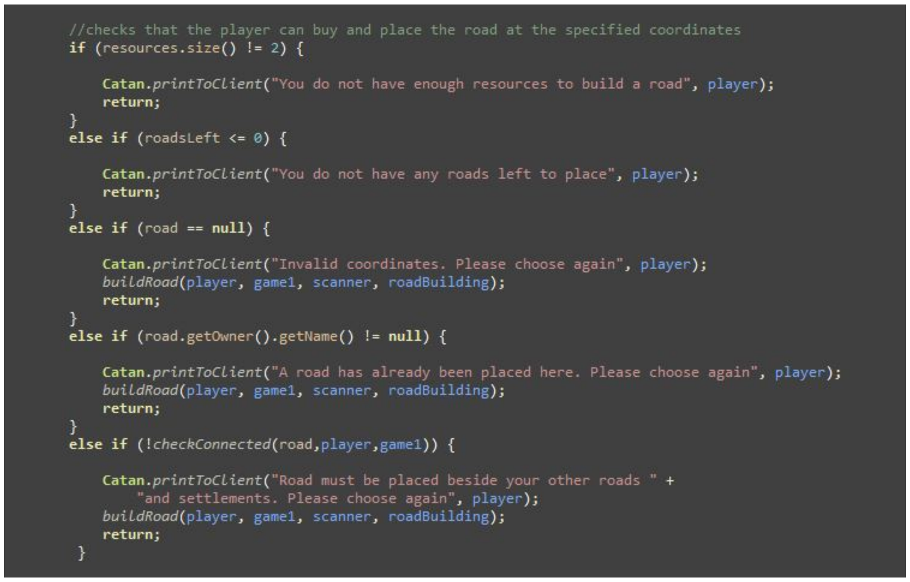


```java
//allows the player to move the robber and steal a card from a player
public static void moveRobber(Player player, Game game1, Scanner scanner) throws IOException {

    try {
        Catan.printToClient("Please select where to place the robber", player);

        Catan.printToClient("Select X coordinate", player);
        int x = Integer.parseInt(Catan.getInputFromClient(player, scanner));

        Catan.printToClient("Select Y coordinate", player);
        int y = Integer.parseInt(Catan.getInputFromClient(player, scanner));
        Coordinate a = new Coordinate(x, y);

        //checks the coordinates are in the correct range
        if((!((2*y <= x+8) && (2*y >= x-8) && (y <= 2*x+8) && (y >= 2*x-8) && (y >= -x-8) && (y <= -x+8)))
                || (!game1.getBoard().getLocationFromCoordinate(a).getType().equals(HEX))) {

            Catan.printToClient("Invalid coordinates. Please choose again", player);
            moveRobber(player, game1, scanner);
            return;
        }

        Hex hex1 = (Hex) game1.getBoard().getLocationFromCoordinate(a).getContains();
        hex1.setisRobberHere(ROBBER);
        game1.getBoard().setRobber(a);

        robberStealCard(player,  game1, scanner);
    }
    catch(InputMismatchException e) {

        scanner.nextLine();
        moveRobber(player,game1,scanner);
    }
}
```

Figure 8: Getting user coordinates before the robber is moved

Resource allocation turned out to be a very difficult part of implementation, since there are many edge cases that need to be introduced, like if the robber was on a hex then no resources could be collected from there. Another edge case, for example, was that there needed to be a check to ensure that the bank had enough of a resource to give all the players that required it the correct amount. If the bank could not do this, no player would get this resource. Although: if only one player required the resource and the bank still did not have the correct amount, the bank would give out as much as they could to this player. To code this, an Array List was added to the player class, "newResourceCards", which would keep track of what resources the player should receive on the turn. Any resources the bank did not have enough of was also added to an ArrayList, "illegalResources". A check was introduced to count how many players needed each illegal resource. If more than one would collect it, those cards were removed from their "newResourceCards". Only after this check was completed would all the cards in "newResourceCards" be added to their hand.

Building roads, settlements and cities followed a similar method to the one used in the setup phase, although extra checks were needed. This is due to the fact that building was now not free, and the player needed the correct resource cards. For each type of building object, the player's resource card deck was looked at, to ensure they had all the required cards. If so, they were removed from their hand, allowing them to build their road/settlement/city. The player then chose where they would like to place their purchased building, which has be be a legal space. If illegal coordinates were selected, an error message would be printed, so the player could easily see what was wrong. The program then prompted them to select correct coordinates. Error messages could be that an object was already placed there, that settlements had to be two spaces away from another player's settlement, if they didn't have any settlements they could turn into a city etc.. Once a building had been placed, a message was sent to all other players notifying them of the players action.



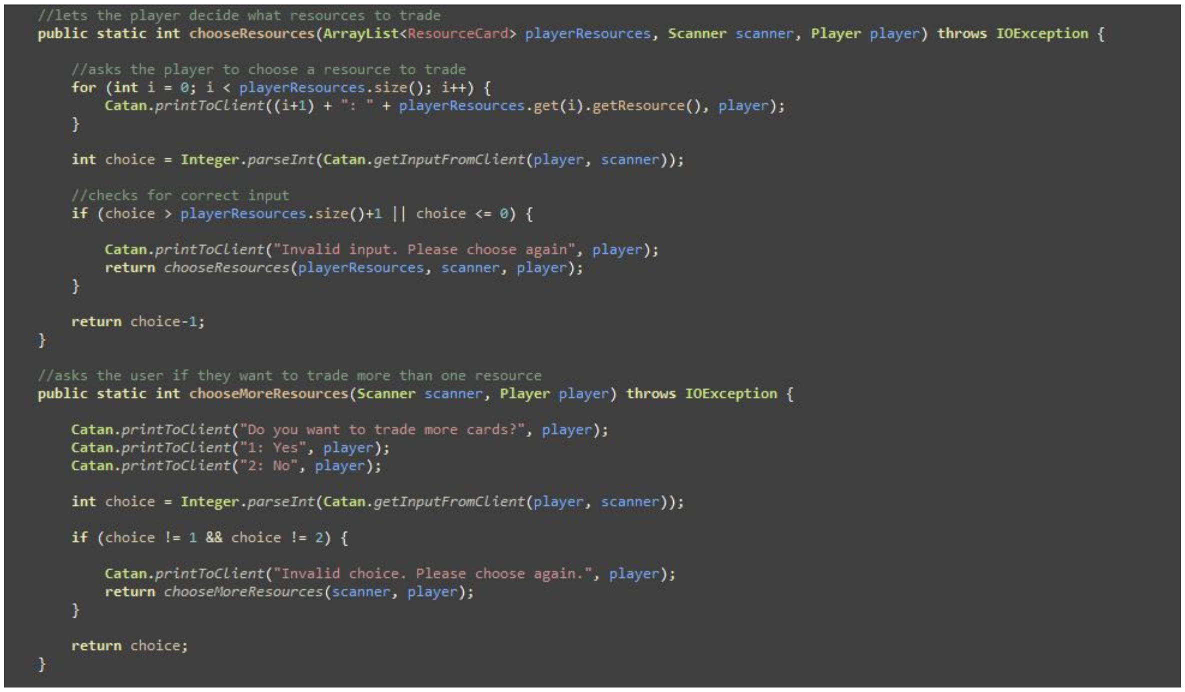
```
//checks that the player can buy and place the road at the specified coordinates
if (resources.size() != 2) {

    Catan.printToClient("You do not have enough resources to build a road", player);
    return;
}
else if (roadsLeft <= 0) {

    Catan.printToClient("You do not have any roads left to place", player);
    return;
}
else if (road == null) {

    Catan.printToClient("Invalid coordinates. Please choose again", player);
    buildRoad(player, game1, scanner, roadBuilding);
    return;
}
else if (road.getOwner().getName() != null) {

    Catan.printToClient("A road has already been placed here. Please choose again", player);
    buildRoad(player, game1, scanner, roadBuilding);
    return;
}
else if (!checkConnected(road,player,game1)) {

    Catan.printToClient("Road must be placed beside your other roads " +
        "and settlements. Please choose again", player);
    buildRoad(player, game1, scanner, roadBuilding);
    return;
 }
```

Figure 9: The error checking user when a player wishes to purchase a road

Buying development cards was implemented in the exact same way, although there were less edge cases that needed to be checked. Once a card had been purchased, the type of the card was checked by the program. If it was a victory point card, the points were automatically added to the player's score, and only the current player would be notified. If any other card was drawn, this would be added to the player's hand as a hidden card, meaning they could not play it on the current turn. At the end of each turn, all newly purchased development cards were changed to be visible, so that the player could use them on their next turn.

If the player wished to play a development card on their turn, a check would be made to ensure at least one visible card in their hand. They would then be asked to select one of their cards, which would be removed from their hand, and the relevant method for this card would then be called. If a “Knight” card was chosen, the player’s army size would be incremented and it would be checked if they should receive the largest army card. They would then be able to complete the robber’s actions. “Road Building” cards would call the method to build a road twice, though checks for resources would not be performed. If the player wished to play a “Year of Plenty” card, they would be prompted to pick two resource cards from the bank, which would be checked to ensure there were enough resources for them. Otherwise, they could not play the card, or would have to pick another resource. “Monopoly” cards required players to choose a resource, and every player would lose their resources of this type and they would be given to the current player, which was very simple to implement. An implementation for playing a victory point card was also written, although since these cards are immediately played, there should never be any instance that it would be used.

Trading proved to be a large part of our implementation, since we needed to try and preserve the social aspect as much as possible. We decided we wanted to implement a “accept, reject, counter-offer” strategy for player-to-player trade, since this would be how deals were made during the real board game. To create a trade, the player would first have to select which player to trade with, as well as the cards they would like to give. Since players did not know what were in each others’ hands, we kept these hidden by asking the player to select which resources they would like to receive, regardless of whether the other player possessed them or not. If the player did not have the cards, the trade as immediately rejected, which still preserved the secrecy of the resource card hands. Otherwise, the trade would be passed to the other player to view, before they selected if they would like to accept. If the player accepts, the trade gets carried out, with resource cards changing hands and each player is notified that a trade took place. If they reject, both parties get notified of the rejection and play continues. Otherwise, the player can instantiate a counter-offer. This involves running through the trade logic again, but with both players switching roles. If a trade was accepted, all players were notified of the players who traded, but not the specific details.



```java
//lets the player decide what resources to trade
public static int chooseResources(ArrayList<ResourceCard> playerResources, Scanner scanner, Player player) throws IOException {

    //asks the player to choose a resource to trade
    for (int i = 0; i < playerResources.size(); i++) {
        Catan.printToClient((i+1) + ": " + playerResources.get(i).getResource(), player);
    }

    int choice = Integer.parseInt(Catan.getInputFromClient(player, scanner));

    //checks for correct input
    if (choice > playerResources.size()+1 || choice <= 0) {

        Catan.printToClient("Invalid input. Please choose again", player);
        return chooseResources(playerResources, scanner, player);
    }

    return choice-1;
}

//asks the user if they want to trade more than one resource
public static int chooseMoreResources(Scanner scanner, Player player) throws IOException {

    Catan.printToClient("Do you want to trade more cards?", player);
    Catan.printToClient("1: Yes", player);
    Catan.printToClient("2: No", player);

    int choice = Integer.parseInt(Catan.getInputFromClient(player, scanner));

    if (choice != 1 && choice != 2) {

        Catan.printToClient("Invalid choice. Please choose again.", player);
        return chooseMoreResources(scanner, player);
    }

    return choice;
}
```

Figure 10: How the player chooses which resources to trade when trading with another player

Trading with the bank had to be implemented very differently, as there are many different types of bank trading depending on what ports a player settles near. This form of trading involved finding what kinds of bank trade a player could perform, before asking them how they would like to trade. Once the player had chosen, the appropriate method would then be called. Standard trade involved a player trading 4 resource cards of the same resource type to receive 1 new resource. The player would choose what type of resource they wished to give away, and what one they would like to receive. It was then checked that the player had 4 of the resource to give away and that they were not giving away the same resource for the one they were receiving. It also was checked that the bank contained the resource the player wanted. Once everything was correct, the trade took place, and all players were notified. The implementation for "standard" ports was similar, except the trade was 3:1 instead of the usual 4:1. For "special" ports, the player could only give away a specific resource for a 2:1 trade, and so less user input was required. The rest of the implementation was identical however.

There were other actions that we allowed players to perform on their turn. These included printing the map, viewing their resource and development card hands, finding the length of their longest road and counting their victory points. These were added as we felt it would make the game a lot easier to play as users could not physically view the current state of the board of their hands as you normally would with a board game. It can also be difficult to remember your score or how long your roads are, so implementing these functions proved to be useful.

As players could perform as many actions as they wished on their turn, an “endTurn” method had to be implemented, which created a new turn for the next player.

## 5.4 Longest Road Algorithm

The most complicated game rule, from a coding perspective, is probably the longest road calculation. Because of this, we feel that it is beneficial to detail a little of its implementation.

Every time someone places a road, we must check to see if this newly added road has affected the length of their longest road overall. The function used has two main cases: the new road is connected to the user’s existing roads by one intersection; the road is connected to existing roads at both ends and, in doing so, connects two independent road systems or finishes a cycle. If the new road is only joined at one end, then a recursive function is called, which carries in its parameters the distance from the start of the current road and a list of points it has visited. During each recursive step, it calls itself once again for each unvisited road branching out from its current position, incrementing the road’s total length each time. When there are no more possible unvisited roads, it returns the final distance. Higher calls then return the greatest distance found by the recursive calls, and so the final result will be the longest non-cyclical path from that point. If the new road is joined at two ends, then we initially begin by assuming that we are connecting two separate road systems, and just call the previous method twice, branching out from both ends of the new road (ideally just add the result together). Then, to account for the possibility that we are completing a loop, we mark off each location uniquely for each “side” of the road the branch originated from, and if we ever reach a place that another branch has been with a new branch that has originated from a different “side” of the original road, we then know we have a loop. When this happens we know that we must instead simply take the largest length value returned, and not add them together. There are a few details not related to the overall strategy which have been skipped over here (such as the fact that you must be suitably prohibited from traversing the newly placed road in loop situations; that the branch does not so much store a list of points visited, as it stores a list of “branchIDs” it used to have, where a new one is generated each step; each point stores a list of “branchIDs” that have landed on itetc.), but these can, of course, be deduced from the code itself.

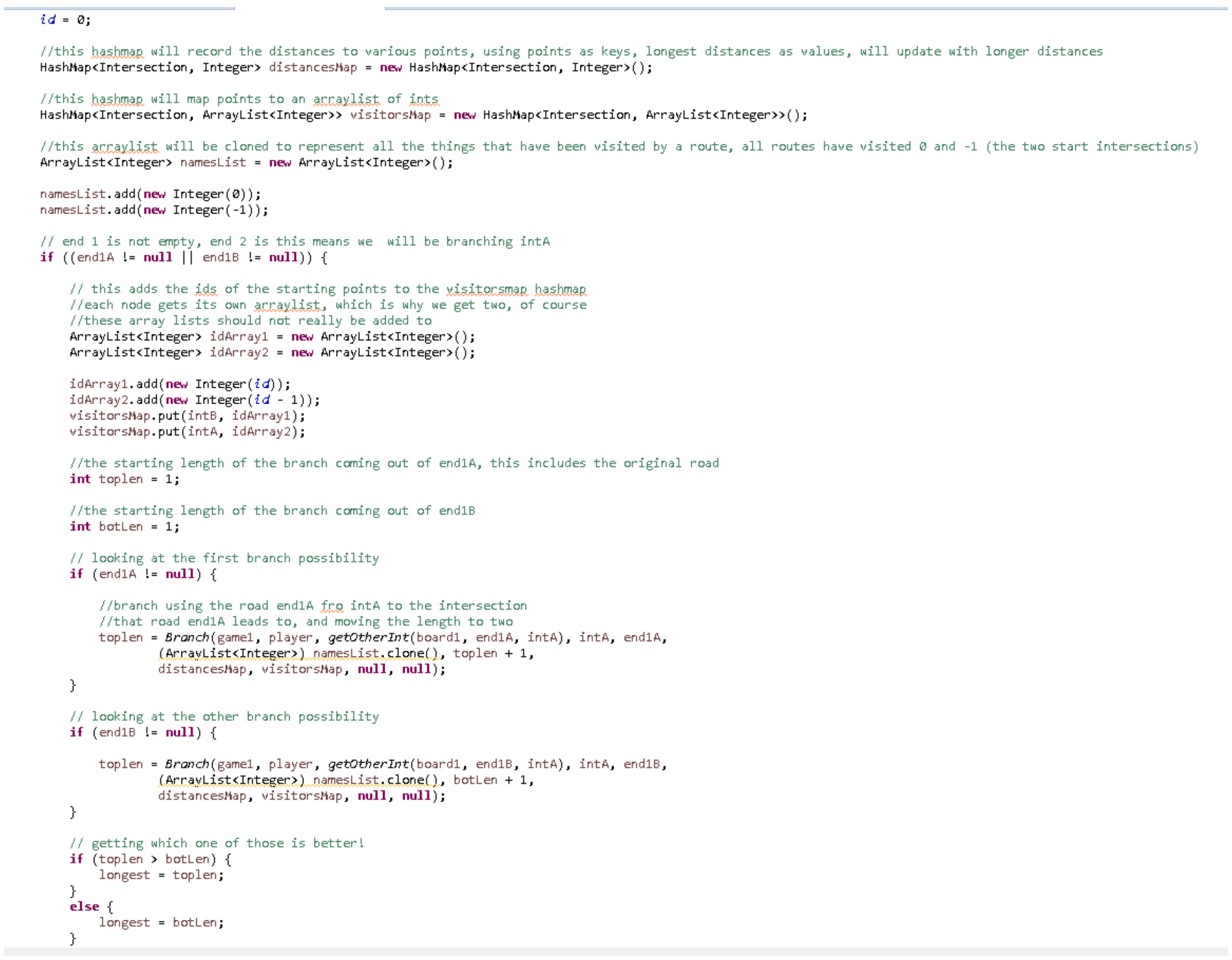
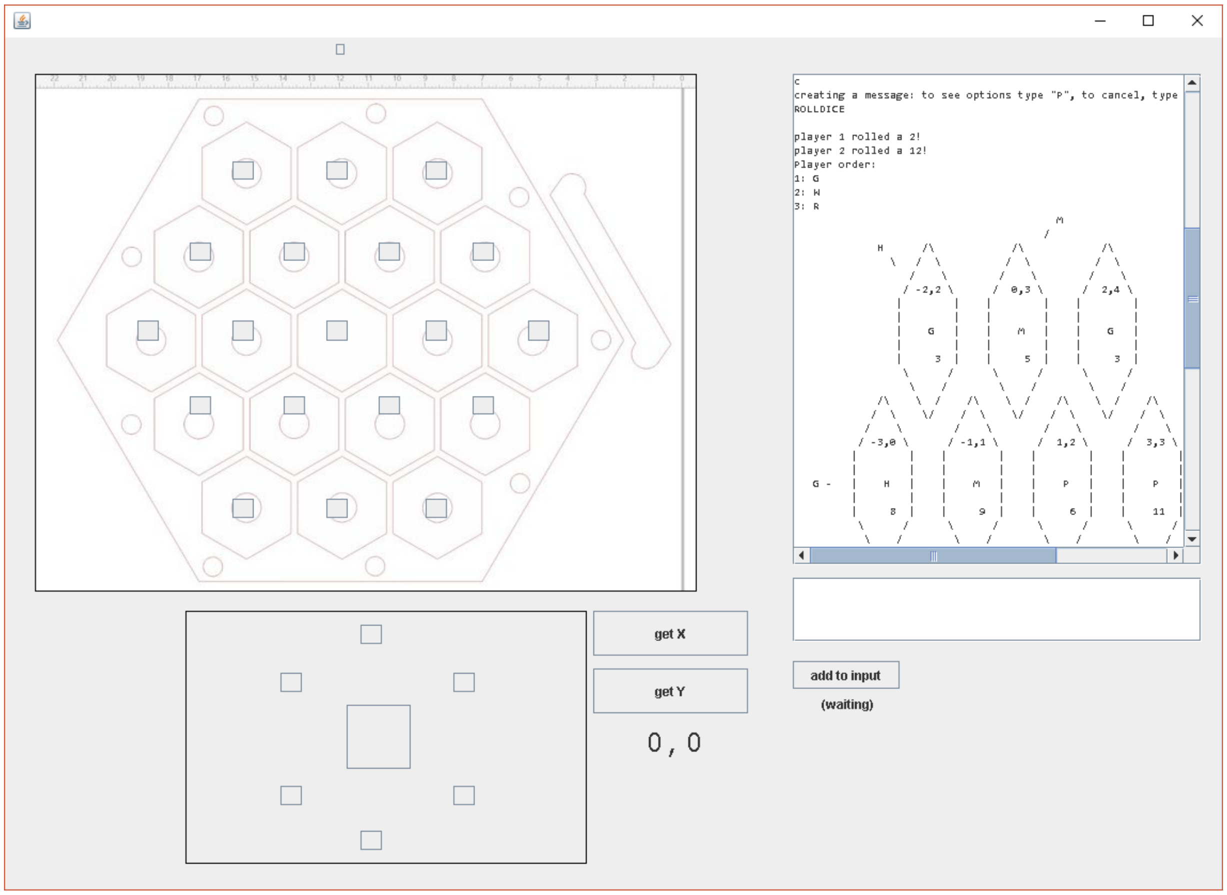

```
id = 0;

//this hashmap will record the distances to various points, using points as keys, longest distances as values, will update with longer distances
HashMap<Intersection, Integer> distancesMap = new HashMap<Intersection, Integer>();

//this hashmap will map points to an arraylist of ints
HashMap<Intersection, ArrayList<Integer>> visitorsMap = new HashMap<Intersection, ArrayList<Integer>>();

//this arraylist will be cloned to represent all the things that have been visited by a route, all routes have visited 0 and -1 (the two start intersections)
ArrayList<Integer> namesList = new ArrayList<Integer>();

namesList.add(new Integer(0));
namesList.add(new Integer(-1));

// end 1 is not empty, end 2 is this means we  will be branching intA
if ((end1A != null || end1B != null)) {

    // this adds the ids of the starting points to the visitorsmap hashmap
    //each node gets its own arraylist, which is why we get two, of course
    //these array lists should not really be added to
    ArrayList<Integer> idArray1 = new ArrayList<Integer>();
    ArrayList<Integer> idArray2 = new ArrayList<Integer>();

    idArray1.add(new Integer(id));
    idArray2.add(new Integer(id - 1));
    visitorsMap.put(intB, idArray1);
    visitorsMap.put(intA, idArray2);

    //the starting length of the branch coming out of end1A, this includes the original road
    int toplen = 1;

    //the starting length of the branch coming out of end1B
    int botLen = 1;

    // looking at the first branch possibility
    if (end1A != null) {

        //branch using the road end1A fro intA to the intersection
        //that road end1A leads to, and moving the length to two
        toplen = Branch(game1, player, getOtherInt(board1, end1A, intA), intA, end1A,
                (ArrayList<Integer>) namesList.clone(), toplen + 1,
                distancesMap, visitorsMap, null, null);
    }

    // looking at the other branch possibility
    if (end1B != null) {

        toplen = Branch(game1, player, getOtherInt(board1, end1B, intA), intA, end1B,
                (ArrayList<Integer>) namesList.clone(), botLen + 1,
                distancesMap, visitorsMap, null, null);
    }

    // getting which one of those is better!
    if (toplen > botLen) {
        longest = toplen;
    }
    else {
        longest = botLen;
    }
```

Figure 11: A part of how the longest road algorithm was implemented

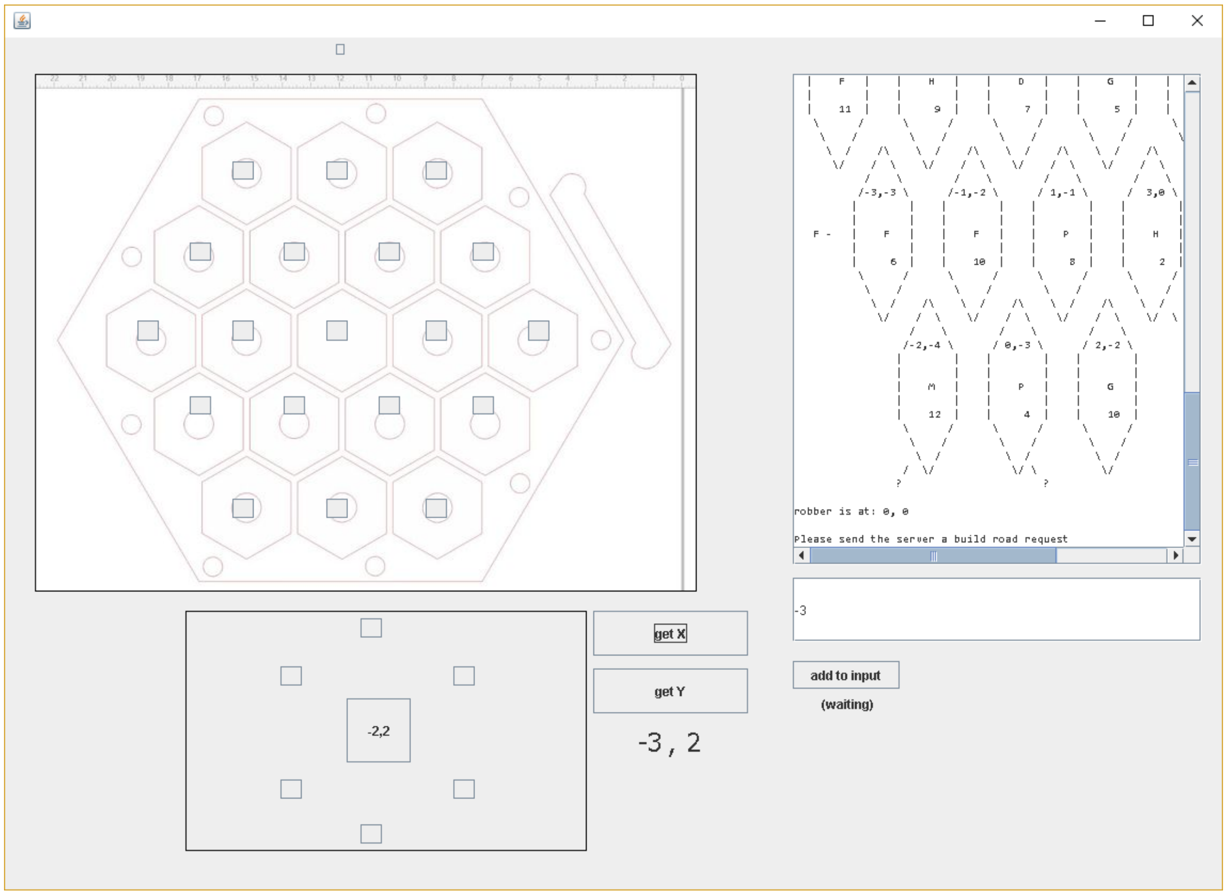
## 6 User Interface

We quickly created a text-based User Interface (UI) to give a display of the board state. We felt that this was essential in order to test our game from the beginning. It was also essential for allowing the ability to produce functioning project versions for each project milestone which could stand in their own right. Because of this, after the first semester of work on this practical, we had a reduced, but playable, “hotseat” implementation of Catan that could be played as a game. This text-based implementation of Catan was kept, and allowed us to incrementally introduce more advanced client/server communications without losing functionality in our most current version (i.e. we could use some String based elements of the inter-group protocol to convey our endemic protocol of String based messages). This kind of text based User Interface also allowed us to explain to the player easily what is expected from them and what stage in their action they are at. The main drawback of such a system is that selecting board locations is not intuitive or simple, and requires one to grasp the coordinate system used fairly well. Our GUI (Graphical User Interface) was created to specifically remedy this drawback in particular, as the rest of the original system worked well. The GUI is a "Java.swing" object, created using the NetBeans IDE, and some custom back-end Java. It allows a user to select a hex on a reference sheet, then select either the hex itself or indicate one of its adjacent intersections, all in a separate frame. Once a coordinate has been indicated in this way, the "getX" and "getY" buttons allow the user to enter these coordinates into the input field for when they are prompted to give specific coordinates. The image for the reference sheet can be found at [1].



```
c
creating a message: to see options type "P", to cancel, type
ROLLDICE

player 1 rolled a 2!
player 2 rolled a 12!
Player order:
1: G
2: W
3: R
                                         M
                                       /
             H      /\            /\            /\
               \   /  \          /  \          /  \
                  /    \        /    \        /    \
                 / -2,2 \      /  0,3 \      /  2,4 \
                |        |    |        |    |        |
                |        |    |        |    |        |
                |    G   |    |    M   |    |    G   |
                |        |    |        |    |        |
                |     3  |    |     5  |    |     3  |
                 \      /      \      /      \      /
                  \    /        \    /        \    /
             /\    \  /    /\    \  /    /\    \  /    /\
            /  \    \/    /  \    \/    /  \    \/    /  \
           /    \        /    \        /    \        /    \
          / -3,0 \      / -1,1 \      /  1,2 \      /  3,3 \
         |        |    |        |    |        |    |        |
         |        |    |        |    |        |    |        |
   G -   |    H   |    |    M   |    |    P   |    |    P   |
         |        |    |        |    |        |    |        |
         |     8  |    |     9  |    |     6  |    |     11 |
          \      /      \      /      \      /      \      /
           \    /        \    /        \    /        \    /
```

Figure 12: User Interface



```
  |    F   |    |    H   |    |    D   |    |    G   |    |
  |        |    |        |    |        |    |        |    |
  |    11  |    |     9  |    |     7  |    |     5  |    |
   \      /      \      /      \      /      \      /      \
    \    /        \    /        \    /        \    /        \
     \  /    /\    \  /    /\    \  /    /\    \  /    /\
      \/    /  \    \/    /  \    \/    /  \    \/    /  \
           /    \        /    \        /    \        /    \
          /-3,-3 \      /-1,-2 \      / 1,-1 \      /  3,0 \
         |        |    |        |    |        |    |        |
         |        |    |        |    |        |    |        |
   F -   |    F   |    |    F   |    |    P   |    |    H   |
         |        |    |        |    |        |    |        |
         |     6  |    |    10  |    |     8  |    |     2  |
          \      /      \      /      \      /      \      /
           \    /        \    /        \    /        \    /
            \  /    /\    \  /    /\    \  /    /\    \  /
             \/    /  \    \/    /  \    \/    /  \    \/  \
                  /    \        /    \        /    \
                 /-2,-4 \      / 0,-3 \      / 2,-2 \
                |        |    |        |    |        |
                |        |    |        |    |        |
                |    M   |    |    P   |    |    G   |
                |        |    |        |    |        |
                |    12  |    |     4  |    |    10  |
                 \      /      \      /      \      /
                  \    /        \    /        \    /
                   \  /          \  /          \  /
                 /  \/            \/ \          \/
                ?                      ?

robber is at: 0, 0

Please send the server a build road request
```

Figure 13: User Interface adding coordinates

## 7 Collaboration

As our final game was required to both support multi-player gameplay across multiple machines and host external AIs on our board, it was necessary for all groups, both Joint and Single Honours, to agree on appropriate protocols for communication between each team's games. In order to do this, a regular inter-group meeting was organised by one of the other teams' members, with each group having a representative present in order to discuss the best options for their group and to relay decisions made by the groups back to their own group members.

Due to the fact that all members of our group are Joint Honours students, we unfortunately had very little networking experience prior to beginning this project. Because of this, our representative at the inter-group meetings acted mainly as a messenger, relaying the other groups' decisions, rather than helping to shape the protocols themselves. These meetings were, however, still crucial to our game development, as it helped develop our understanding of web sockets, Protocol Buffers etc., without which we would not have been able to complete our game implementation fully.

The inter-group meeting also decided on the coordinate system, as this was an essential part of the messages that would need to be sent between players to ensure towns, roads etc. were consistently placed across all boards where the player placing them intended. All protocol and coordinate decisions were written up by one of the other groups' representatives and uploaded to a central GitLab repository where each teams' members could access the relevant information, ensuring there were clear, accessible guidelines on how to approach the relevant tasks.

To help organise these inter-group meetings and keep all members of groups updated on what had been discussed and decided, a channel for the module was created on Slack, accessible to all group members. This was also where links to new GitLab postings were circulated.

## 8 Methodology

While we did not stick to a strict development technique throughout this project, we did model our approach on SCRUM development, having had experience of it during our first practical for CS3051 (Software Development) in Semester 1.

The main difference between strict SCRUM development and our approach was we did not have daily stand-up meetings and our dates to have sections of the project completed by were not as strict as with SCRUM. We did, however, split development into smaller sub-sections which we then assigned to individual group members or pairs, keep track of all areas of the project that had been completed, were in progress, and still needed tackling through a Trello board, and we had regular meetings, often bi-weekly, both with and without our project supervisor to make sure we were all on track with completing our assigned sections. A Trello board was used as we had made use of one during our previous SCRUM practical as an online alternative to a physical SCRUM board, meaning we could access it anywhere, anytime and make instant updates as we progressed through the various tasks. We decided to use a Trello board again here for many of the same reasons, especially since this was a much larger scale project, with more sections needing completed throughout the year. We also had a section of the board dedicated to storing links to helpful websites, diagrams etc., such as a diagram of the coordinate system, our GitHub repository, progress reports in Google Docs etc. (see Figure 14).

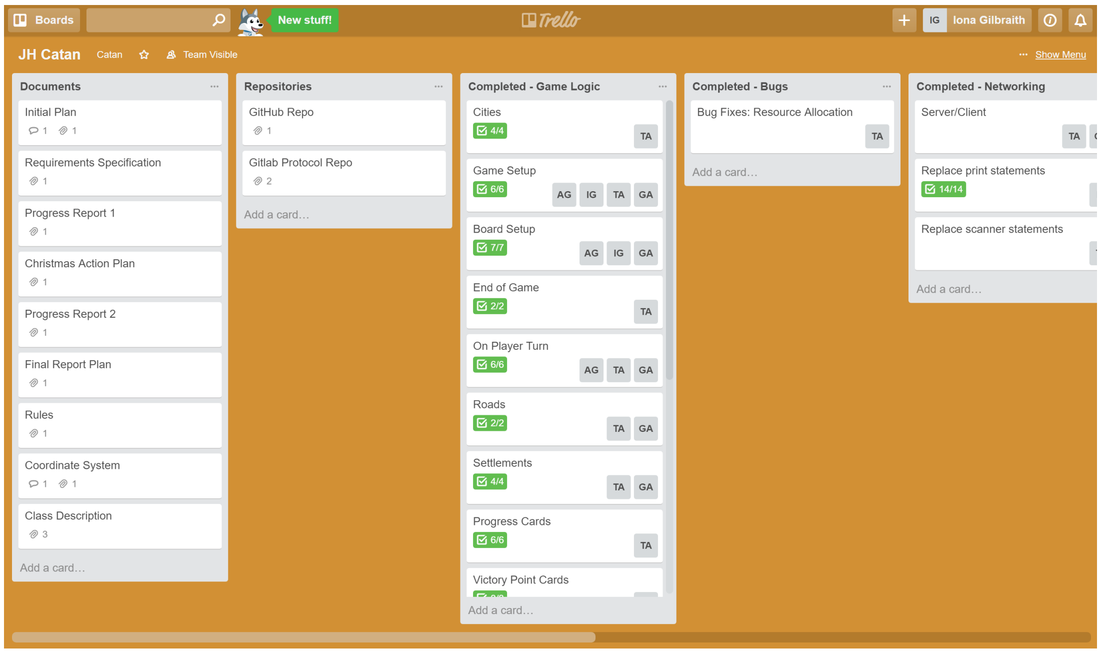


Figure 14: A section of our Trello board used throughout development, displaying file and repository links, as well as sections detailing completed work

While we did not have regular in-person meetings in the style of daily stand-up meetings, we did stay in regular contact with other team members using the Facebook Messenger instant messaging service. This allowed us to ask questions relating to work another team member had done, remind people of meetings, deadlines etc., and meant we did not have to try to organise as many in-person meetings, as this proved difficult due to our varying timetables and availability. We decided to use Messenger as it was a free service we all already had access to and, as it was instant, could update each other of changes, reminders etc. immediately, with notifications coming straight through to our phones, laptops etc. ensuring we never missed an important update.

As our approach was not a strict SCRUM development, we did not have defined roles such as Scrum Master and Product Owner. We did, however, have a "Team Coordinator", with this role being carried out by Taylor, who acted as a sort of hybrid of these two roles, helping decide which tasks were of most importance at the current development stage, organising meetings with our project supervisor, attending inter-group meetings, and ensuring we kept the record of our progress up to date on the shared Trello board so all team members could quickly tell what stage of development we were currently at.

This approach, while still offering a little more flexibility than strict SCRUM development, ensured that we were constantly making steady progress towards our final goals, and that each group member was contributing their share of work towards our final implementation.

When we first started our implementation of the game rules and logic required for gameplay, we organised a couple of paired programming sessions to get the basic structure and major methods etc. laid out. Paired programming proved very useful at the start of development, as it not only allowed two people to bring their own perspectives and ideas to a task, but also mean that there was more than one person who had worked on these sections. While this shared understanding of the code through paired programming was easier to maintain towards the start of development, it was not sustainable, as it required multiple group members to be able to commit a period of time when they were all free, which became increasingly difficult as the year and semester went on. Even though we carried out less paired programming later in development, we still tried to ensure that all group members looked over work done by the other members, even if just at the structural level, o that it was not just one person with knowledge of how a feature worked. This shared knowledge was essential in the development of our solution, as it made adding and integrating new features of the game quicker and easier.

In order for each team member to be able to access the latest version of the code on their own machine we created a collaborative GitHub repository, where all code was uploaded and stored, with each team member updating the repository each time they made a change or addition to the code. GitHub not only allowed us to have an easily accessible, up-to-date version of all code written so far, but also acted as version control, meaning we could revert to previous versions of the code if needed. This helped protect against accidental code deletion, unsuccessful code branch merges etc..

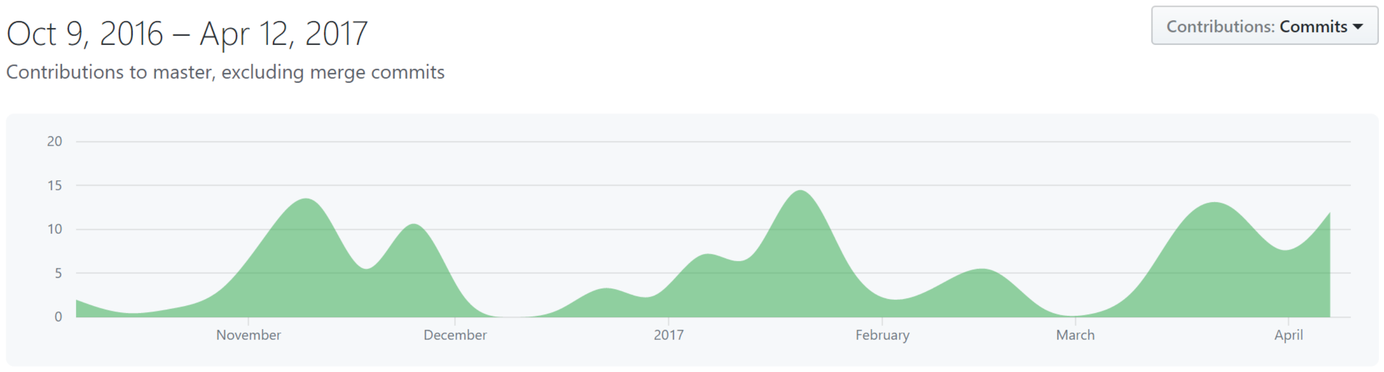
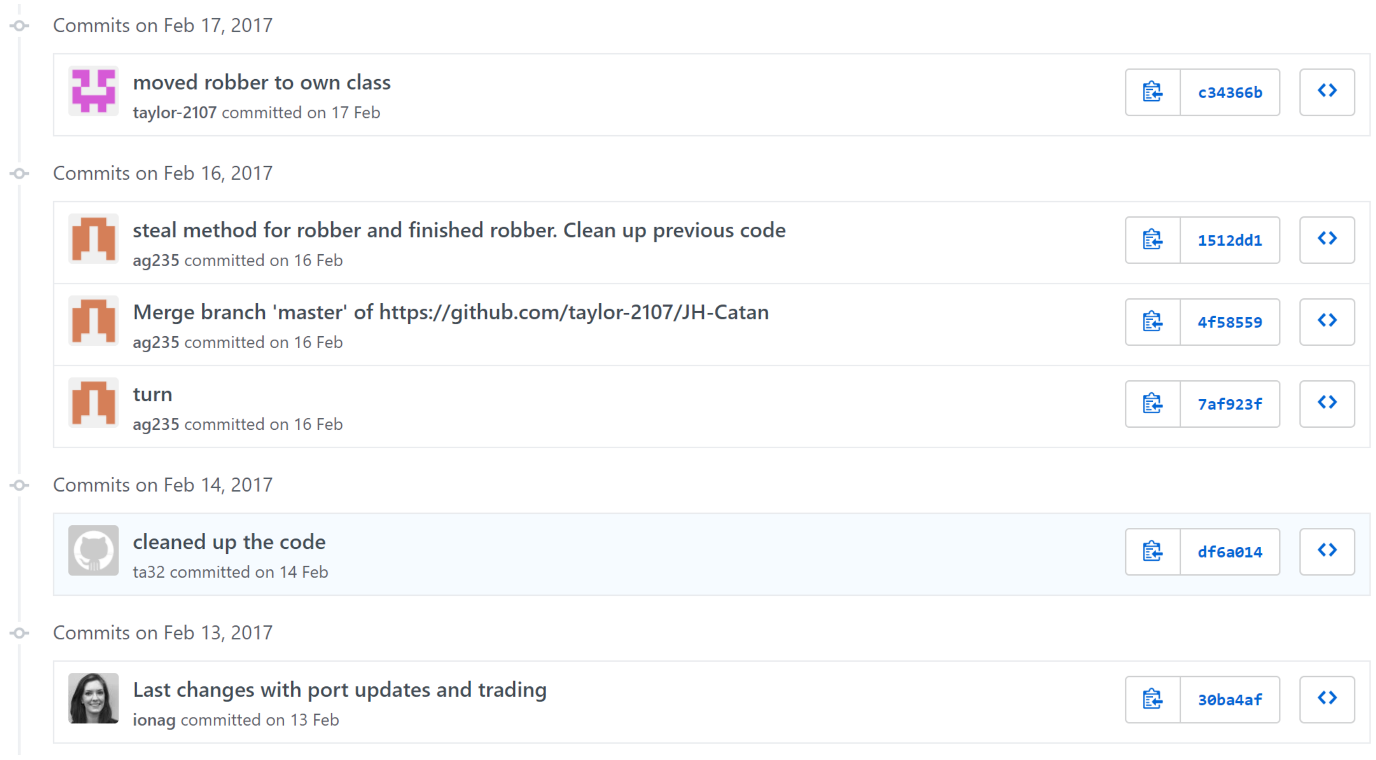


Figure 15: Graph showing our frequency of commit rates to our GitHub repository


Commits on Feb 17, 2017

moved robber to own class
taylor-2107 committed on 17 Feb
c34366b

Commits on Feb 16, 2017

steal method for robber and finished robber. Clean up previous code
ag235 committed on 16 Feb
1512dd1

Merge branch 'master' of https://github.com/taylor-2107/JH-Catan
ag235 committed on 16 Feb
4f58559

turn
ag235 committed on 16 Feb
7af923f

Commits on Feb 14, 2017

cleaned up the code
ta32 committed on 14 Feb
df6a014

Commits on Feb 13, 2017

Last changes with port updates and trading
ionag committed on 13 Feb
30ba4af

Figure 16: A sample of some of our GitHub commits, showing who made each commit, what date it was committed, along with a summary of what each commit consisted of

As with most of the tools we decided to use throughout our development (Trello, GitHub etc.), when it came to writing our intermittent progress reports, as well as this final report, we chose platforms that allowed us to collaborate across multiple machines and users. As our progress reports were not formally assessed we decided simply to use Google Docs to collaboratively write and check/edit them, as it allows for many of the same features as Microsoft Word. However, for this final report using LaTeX, we used ShareLatex, an online, collaborative document editor like Google Docs tailored to creating LaTeX documents.

## 8.1 Timeline Progression

When working on an advanced project that is of significant length, it is important to strategize an estimated plan that the software engineer, or the team, will attempt to follow. However, it is also important to note that although deadlines are set before programming begins, impediments will be faced throughout the time period and not all parts of a project can be foreseen. This section will provide a summary of the teams original estimated time frame and the time frame that was actually achieved. Each "objective" or "goal" will be referred to as a milestone, which will be given a number 1-8. The timelines will also provide markings for when certain papers were required to be submitted or when demo's were arranged. Figure 17 and 18 below are a comparison of the estimated milestone time period and the actual achieved milestone periods.

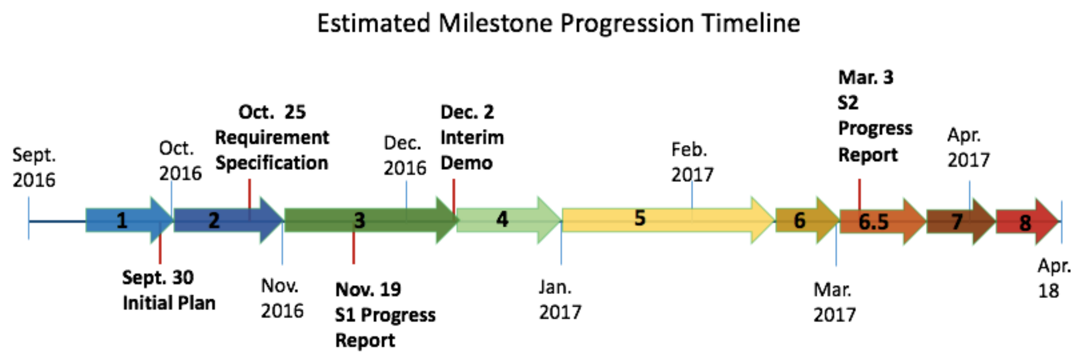


Figure 17: Predicted Milestones Within the Project Timeline

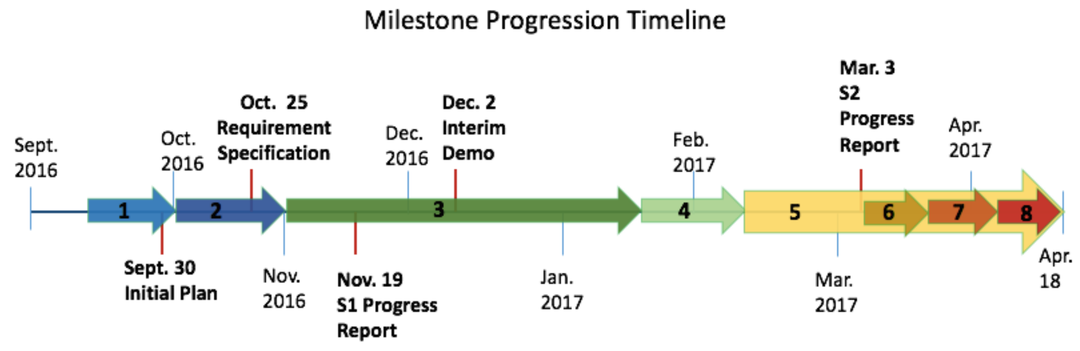


Figure 18: Milestones Achieved Within the Project Timeline

**Milestone Expanded Definitions**

**Milestone 1 : Initial Planning**

During this time the group started to familiarize itself with the specifications and with the Catan game. The rules of the game were inspected and the group met several times to play the physical game board. Once the rules were discovered, an initial rough plan for the project ahead was submitted.

**Milestone 2 : Pseudocode and Requirement Analysis**

This process was involved with taking apart each section of the game, piece by piece, and creating pseudocode logic that the developers can base their programming logic off of. This milestone constructed concrete requirements and general logic of how the requirements are to be implemented.

**Milestone 3 : Logic Implementation**

Once the pseudocode is complete, the logic is implemented. The team members are to divide the different game logic groups between themselves and complete the code in concrete terms.

**Milestone 4 : Testing Game Logic**

Milestone 4 is dedicated to testing the implementation of the game logic and fixing any bugs that are contained. The team is supposed to roughly have a working game after this point is completed.

**Milestone 5 : Networking**

This section involved research of networking, client-server models, and its implementation. The group is to decided which part of the program is to act as the server implementation and which part is to act as the client implementation. The communication method between the two is to be decided as well.

**Milestone 6 : Protocol**

The inter-team committee protocol, that was written, at this stage is to be embedded into the code.

**Milestone 7 : Bug Fixes and Testing 2**

All bugs must be found and fixed. The connection between the networking and the game logic should be tested. The implementation of the overall project should be pretty much wrapped up.

**Milestone 8 : Project Report and Final Touches**

The project should be documented and assessed. Any details that are missing should be completed and final touches added.

***Milestone 6.5* : User Interface***

An appealing aesthetic should be created for the user interface. The user should be able to see graphics and the status of their game in an organized manner.

**Comparison of Predicted vs Actual Milestone Time Periods**

When examining the two timelines, within this section, it is evident that some objectives took a longer time to reach that originally expected. For example, the implementation of the code logic was a lengthier processes than the group originally thought because of the intricate interweaving rules between players and card manipulation tactics. Networking turned out to be a bit more tricky than originally anticipated, as well. Thus, the group extended its implementation until the end of the time period, while working on other aspects simultaneously. Finally, the group originally intended to create a more sophisticated user interface. However, with the time crunch and unexpected impediments, this milestone was scratched from the final timeline, especially since it was not crucial for the Joint Honours students.

# 9 Testing

In order to test the program the team decided it would be best to organize official testing days. During these testing days, a time period of 3-4 hours was allocated and all the members would go through each part of the program together, fixing all bugs and checking for accuracy. For our first test day, we played our implementation of the game, whilst also playing the board game in tandem, allowing us to correctly determine if the game logic was correct. A total of 2 official testing days were allocated, however testing was still implemented throughout the process individually and in pairs. In addition to these, we used mocked client input, which was read in from a text file in order to test features located further into the gameplay.

Figure 19: Testing: Our First Testing Day

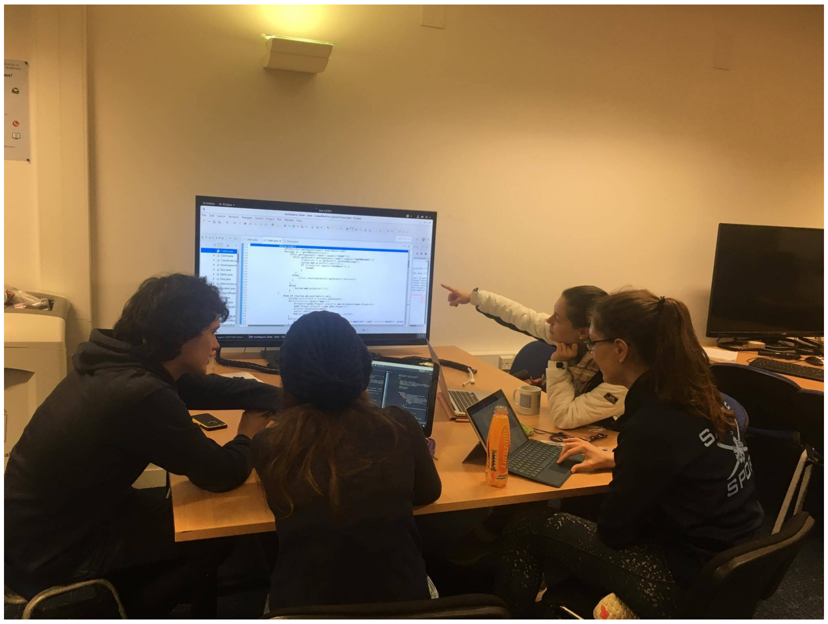
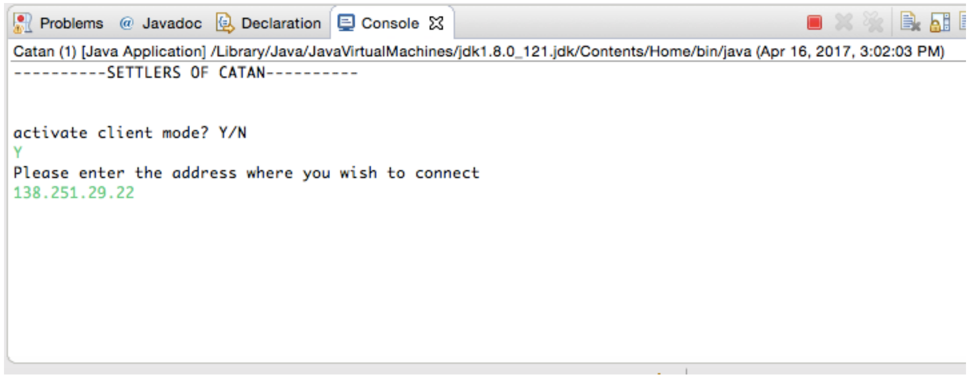


Figure 20: Testing: Official Testing Day

When the Catan game is played the console returns a prompt that asks a user whether they are a client or a server. As explained in the client server implementation above, only the first player, player one, should choose no for this option. Other players must choose "Y", yes, to be a client and then submit the IP address of the server that they are connecting to. This functionality is visible in Figure 21. The player will act as a client in this instance.

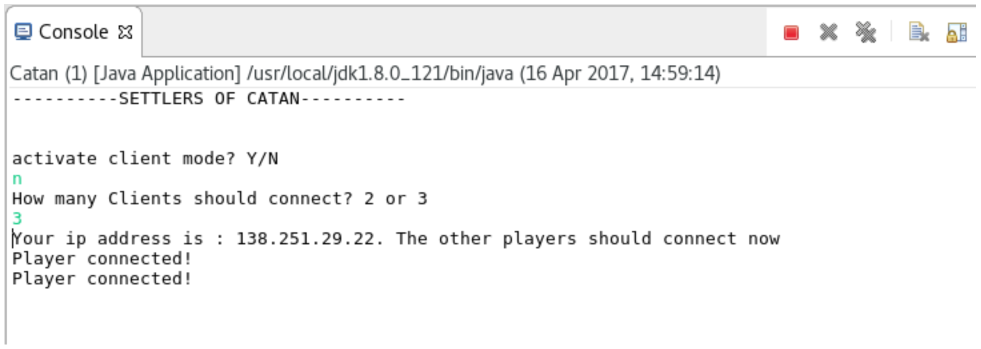


```
Catan (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/bin/java (Apr 16, 2017, 3:02:03 PM)
----------SETTLERS OF CATAN----------


activate client mode? Y/N
Y
Please enter the address where you wish to connect
138.251.29.22
```

Figure 21: Testing: Adding player to the game (if you are not the player hosting the game)

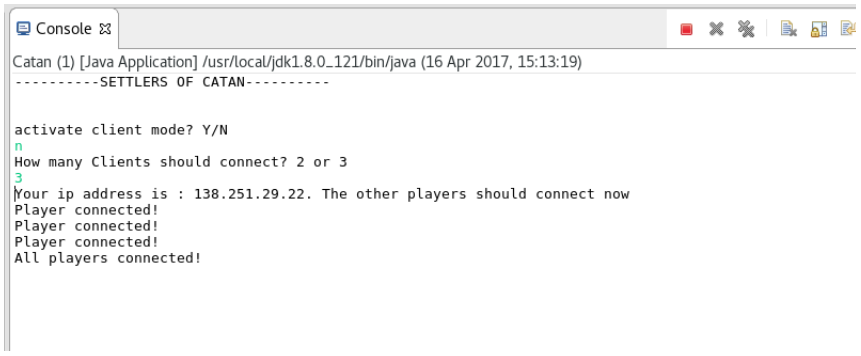
When a player acts as a server, they are able to see when other players are connected. The player will see the message "Player Connected!". Figure 22 shows the console when a player connects.

As seen in Figure 23 the player acting as a server will be prompted choose the number of players in the game. When the number is reached the console will say “All players connected!”.



```
Catan (1) [Java Application] /usr/local/jdk1.8.0_121/bin/java (16 Apr 2017, 14:59:14)
----------SETTLERS OF CATAN----------


activate client mode? Y/N
n
How many Clients should connect? 2 or 3
3
Your ip address is : 138.251.29.22. The other players should connect now
Player connected!
Player connected!
```

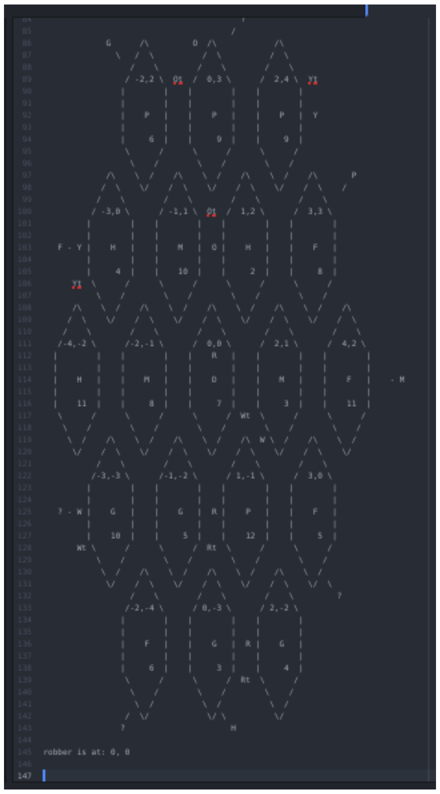
Figure 22: Testing: Adding player to the game and checking if other players are connected



```
Catan (1) [Java Application] /usr/local/jdk1.8.0_121/bin/java (16 Apr 2017, 15:13:19)
----------SETTLERS OF CATAN----------


activate client mode? Y/N
n
How many Clients should connect? 2 or 3
3
Your ip address is : 138.251.29.22. The other players should connect now
Player connected!
Player connected!
Player connected!
All players connected!
```

Figure 23: Testing: When all players are connected, a signal is returned to continue game

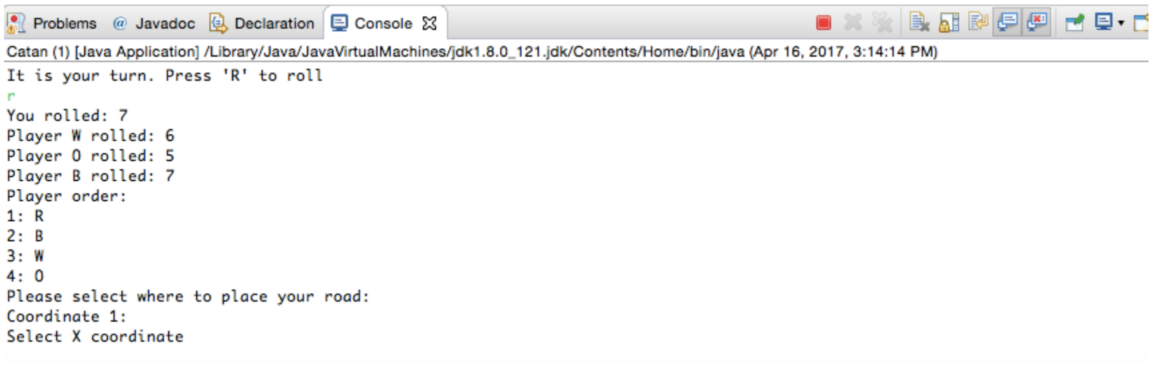
Figure 24 is a representation of the the user interface looks for this game initially when we were testing on a text based console. All the players and resources are present in the graph. When roads and settlements are built, they are also represented on this UI.



```
robber is at: 0, 0
```

Figure 24: Testing: User Interface of Coordinates and Board

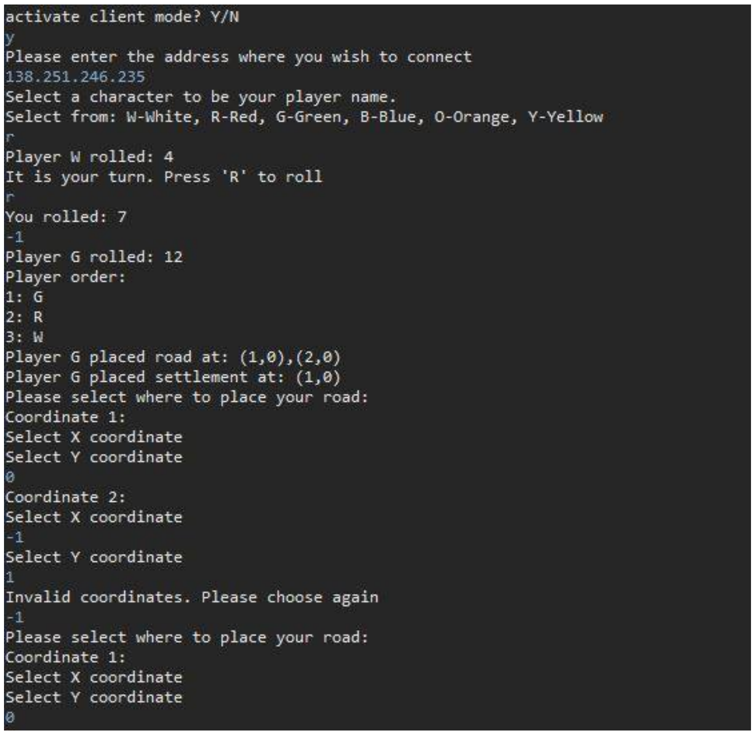
Figure 25 displays the order in which the players will take their turns. The order depends on the dice value each player rolls.



```
Catan (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/bin/java (Apr 16, 2017, 3:14:14 PM)
It is your turn. Press 'R' to roll
r
You rolled: 7
Player W rolled: 6
Player O rolled: 5
Player B rolled: 7
Player order:
1: R
2: B
3: W
4: O
Please select where to place your road:
Coordinate 1:
Select X coordinate
```

Figure 25: Testing: All player roll the dice in the beginning to set up playing order

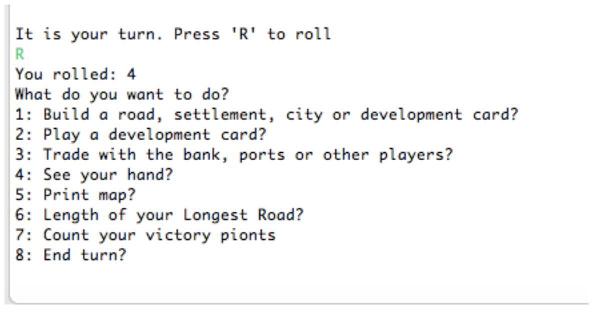
Once all of the players are connected and each player picks a color, all players roll the dice to see who obtains the highest value. Then the player order is ranked based on the highest to lowest, highest value goes first. Figure 26 represents the order of the game after all the dice are rolled. After the order is picked the user is allowed to place two roads and a settlement, two times. Figure 26 also represents this functionality being prompted in the console.



```
activate client mode? Y/N
y
Please enter the address where you wish to connect
138.251.246.235
Select a character to be your player name.
Select from: W-White, R-Red, G-Green, B-Blue, O-Orange, Y-Yellow
r
Player W rolled: 4
It is your turn. Press 'R' to roll
r
You rolled: 7
-1
Player G rolled: 12
Player order:
1: G
2: R
3: W
Player G placed road at: (1,0),(2,0)
Player G placed settlement at: (1,0)
Please select where to place your road:
Coordinate 1:
Select X coordinate
Select Y coordinate
0
Coordinate 2:
Select X coordinate
-1
Select Y coordinate
1
Invalid coordinates. Please choose again
-1
Please select where to place your road:
Coordinate 1:
Select X coordinate
Select Y coordinate
0
```

Figure 26: Testing: Initial Board Set Up Complete

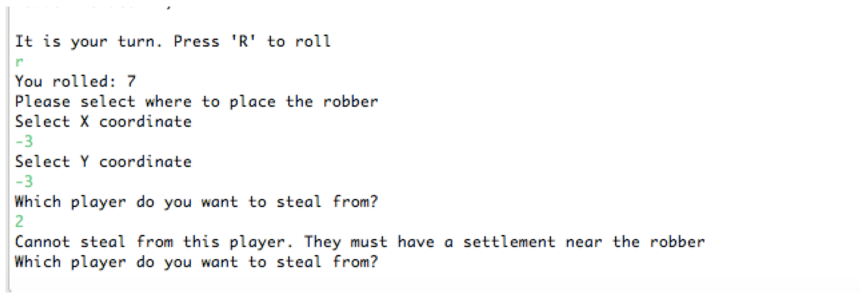
Once the initial settlements and roads are placed. The game begins. The first player rolls the dice and then is promoted with 8 options. The player will choose an option. Figure 27 displays a list of the options that the player is given.



```
It is your turn. Press 'R' to roll
R
You rolled: 4
What do you want to do?
1: Build a road, settlement, city or development card?
2: Play a development card?
3: Trade with the bank, ports or other players?
4: See your hand?
5: Print map?
6: Length of your Longest Road?
7: Count your victory pionts
8: End turn?
```

Figure 27: Testing: Player rolls the dice. Options are displayed.

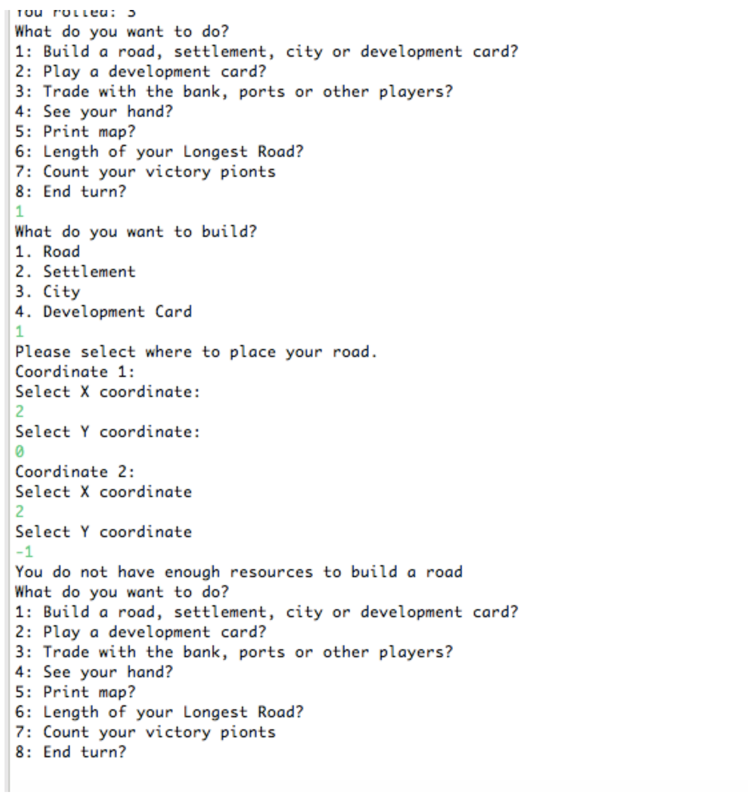
Figure 28 displays testing of the Robber. When the robber is switched position, the player that moved him can steal a card. In this instance, the player purposely tried to steal a card that was unavailable. When a card is unavailable the console rejects the “steal” and asks to choose another player and card to steal from.



```
It is your turn. Press 'R' to roll
r
You rolled: 7
Please select where to place the robber
Select X coordinate
-3
Select Y coordinate
-3
Which player do you want to steal from?
2
Cannot steal from this player. They must have a settlement near the robber
Which player do you want to steal from?
```

Figure 28: Testing: Robber switched position and card stolen

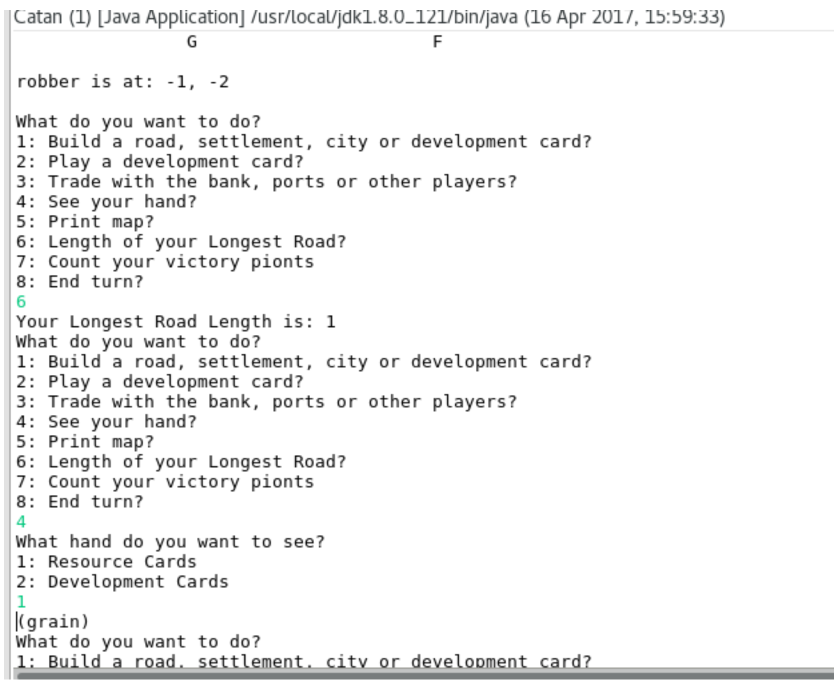
Figure 29 displays an instance of a player trying to build a road but not having enough resources to afford one. When a player chooses this option and is rejected the console returns “You do not have enough resources to build a road” and prompts the user to choose another option.



```
You rolled: 5
What do you want to do?
1: Build a road, settlement, city or development card?
2: Play a development card?
3: Trade with the bank, ports or other players?
4: See your hand?
5: Print map?
6: Length of your Longest Road?
7: Count your victory pionts
8: End turn?
1
What do you want to build?
1. Road
2. Settlement
3. City
4. Development Card
1
Please select where to place your road.
Coordinate 1:
Select X coordinate:
2
Select Y coordinate:
0
Coordinate 2:
Select X coordinate
2
Select Y coordinate
-1
You do not have enough resources to build a road
What do you want to do?
1: Build a road, settlement, city or development card?
2: Play a development card?
3: Trade with the bank, ports or other players?
4: See your hand?
5: Print map?
6: Length of your Longest Road?
7: Count your victory pionts
8: End turn?
```

Figure 29: Testing: When the resources are insufficient, player will not be able to build road or settlement

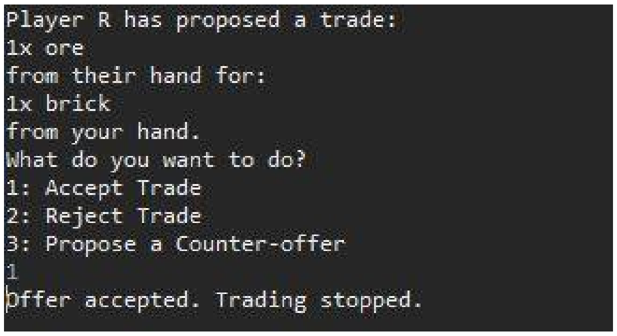
Figure 30 displays a turn cycle and what happens when a player chooses to “see their hand”.



```
Catan (1) [Java Application] /usr/local/jdk1.8.0_121/bin/java (16 Apr 2017, 15:59:33)
                G                       F

robber is at: -1, -2

What do you want to do?
1: Build a road, settlement, city or development card?
2: Play a development card?
3: Trade with the bank, ports or other players?
4: See your hand?
5: Print map?
6: Length of your Longest Road?
7: Count your victory pionts
8: End turn?
6
Your Longest Road Length is: 1
What do you want to do?
1: Build a road, settlement, city or development card?
2: Play a development card?
3: Trade with the bank, ports or other players?
4: See your hand?
5: Print map?
6: Length of your Longest Road?
7: Count your victory pionts
8: End turn?
4
What hand do you want to see?
1: Resource Cards
2: Development Cards
1
(grain)
What do you want to do?
1: Build a road, settlement, city or development card?
```

Figure 30: Testing: A turn cycle

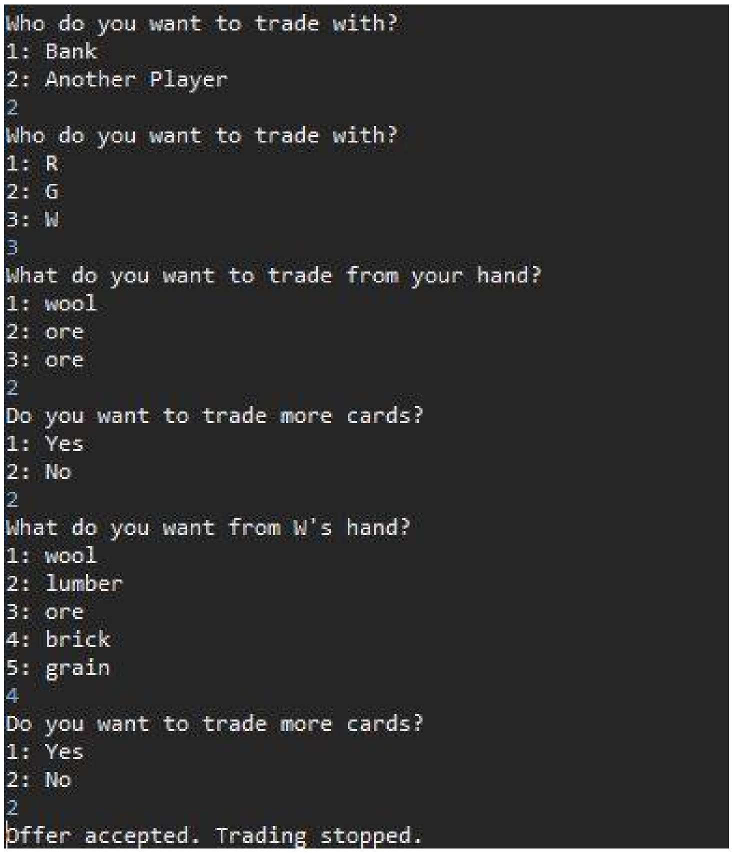
Figure 31 displays the console when a trade is proposed to a player. The player that is being presented the offer has three choices.



```
Player R has proposed a trade:
1x ore
from their hand for:
1x brick
from your hand.
What do you want to do?
1: Accept Trade
2: Reject Trade
3: Propose a Counter-offer
1
Offer accepted. Trading stopped.
```

Figure 31: Testing: Trade with Another Player

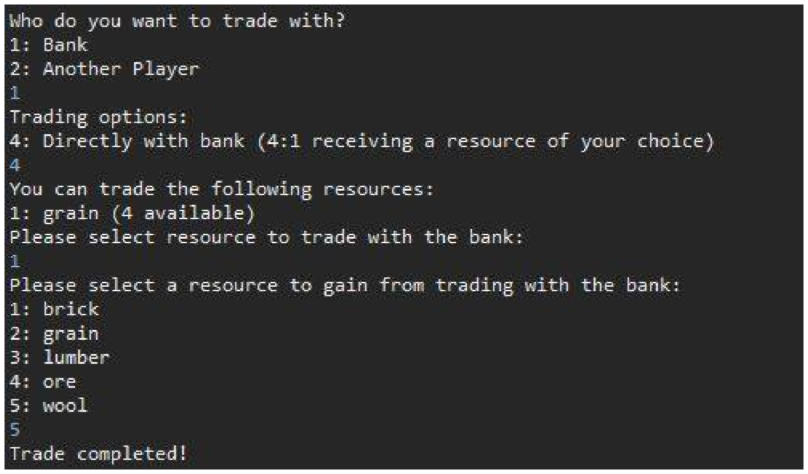
Figure 32 displays what happens when a player chooses to propose a trade (essentially what the player on the proposing side will see in the console).



```
Who do you want to trade with?
1: Bank
2: Another Player
2
Who do you want to trade with?
1: R
2: G
3: W
3
What do you want to trade from your hand?
1: wool
2: ore
3: ore
2
Do you want to trade more cards?
1: Yes
2: No
2
What do you want from W's hand?
1: wool
2: lumber
3: ore
4: brick
5: grain
4
Do you want to trade more cards?
1: Yes
2: No
2
Offer accepted. Trading stopped.
```

Figure 32: Testing: Trade with Another Player

Figure 33 displays a trade that occurs between a player and the bank, rather than a trade between two players.



```
Who do you want to trade with?
1: Bank
2: Another Player
1
Trading options:
4: Directly with bank (4:1 receiving a resource of your choice)
4
You can trade the following resources:
1: grain (4 available)
Please select resource to trade with the bank:
1
Please select a resource to gain from trading with the bank:
1: brick
2: grain
3: lumber
4: ore
5: wool
5
Trade completed!
```

Figure 33: Testing: Trade with the Bank

## 10 Evaluation and Critical Appraisal

As software engineers and team of people accomplishing a goal, it is important to reflect upon the accomplishments and shortcomings of a project. Objective and subjective evaluations are important in order to progress and learn for future endeavours. This section will attempt to provide a critical view on the achievements of the group when concerning this project. It will asses the level of communication amongst team members and compare the result of the project with similar applications available on the web.

The overall objective, as previously mentioned, of this project was to implement the game *Settlers of Catan.* Since the group obtained a project of smaller scope (since all members are Joint Honours students), the time available to coordinate the project was reduced by half. However, considering the structural and physical constraints, the group was able to communicate in an effective manner. As mentioned previously, the team was able to establish group meetings almost every week and to delegate tasks appropriately. In many cases, when groups face deadlines from other courses, long term project seem to get pushed back. However, the group impressively was able to focus some attention on this project most weeks of the year.

The team also effectively communicated with their project supervisor. She was able to see the updates and progression of the project. Meetings were scheduled either weekly or bi-weekly to talk about the time-line of the project.

In comparison to other open-source available *Settlers of Catan* games online, the game is not following the same standards for the User Interface, but the logic is complete for a three or four player game (non-extended version). Since we are a Joint Honours group, we decided as a team, it was most important to focus on the logic of the game before creating an extensive visual experience. Adding a user interface would be comparatively more trivial, if the group was given more time. As an evaluation, it is useful to compare the accomplishments of the team to other Catan games available. One of the most impressive open source Catan games available is called *Catan Universe* [6].

*Catan Universe* has a very extensive user interface, which adds to the experience of playing the game. However, it can be assumed that the advanced elements of the interface sometimes interfere with the efficiency of making movements because it adds a lot of time to the run time. Using a "MacBook Air" the program froze many times, making it highly unappealing to play for impatient people. Although, the run time does change based on the time of computer the game is being played on. Another fault of the game interface, was we found it hard to drag and move the pieces onto the board. However, this version has positive feature as well. It is highly visually appealing and it allows you to play with either other online users or bots. It also has interactive graphics and extension options. It also allows users to create a profile or even play without creating one (thus assuming it integrated a database keeping track of registered users). This section contains screen shots presenting the interface of the online game and also proof that it takes a while to load.

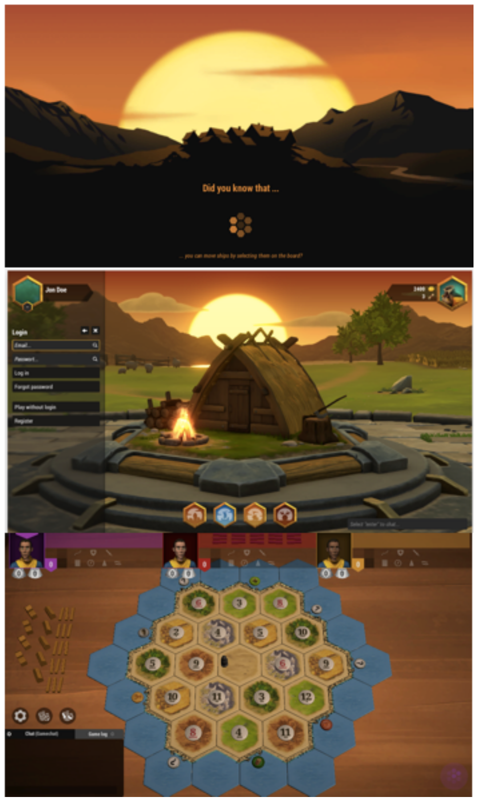


Figure 34: Catan Universe Screen Shots

Other Catan games were found online but most of them had a much more standard graphical user interface and less functionalities. Comparing our program to the other Catan games available online allows us to gauge what we could have implemented, given more time, and allows us to address the pros and cons of the system we have built. The group would focus on fixing the cons, given more time. The group would create a more appealing User Interface, but avoid too many graphics that slow down the efficiency of the game. The team would also allow for more players and a few extension packs. Since the most board games contain a social element, it would also be useful for the group to include a chat system that would allow the players to communicate with each other, either using text based or voice chat. Finally, it may be found useful to add a database keeping track of the players and the amount of times they have won the game. However there are pros in the software that we have built. The basic software logic works and can be used as a great basis for building the program further. Also, although our user interface is simplistic, it allows for efficient run time. A minimalistic interface allows the program to be stored on the server which removes the risk of the game being out of sync across clients. A history of the games is also maintained in the console so the a players game decisions can be reviewed. We can also use our program to host other groups clients easily and give extreme flexibility, as opposed to other Catan games. Overall, our project is a solid building block for the future.

# 11 Conclusion

In summary, the group recognizes that when working in teams, the elements of project implementation are both lessened and increased in burden. Working in a group requires careful planning and delegating. It also allows difficulties to be broken down and completed by several people. Thus, we believe that we learned a great deal about working as software engineers in a real world setting.

As far as the project itself is concerned, we believe that as a Joint Honours group, with limited time, we created a solid game logic platform that can later be combined with AI and a more extensive interface. We believe that we have managed to fulfill all the requirements that have been asked by us. We understand that the project is not ready to be uploaded and played online by many players, however, we believe that we have managed to complete most of the grunt work needed for an online version of *Settlers of Catan*.

Finally, we believe that we worked well as a team and were able to meet frequently and to solve problems efficiently. Given the project again, we would probably keep the same structure for our methodology. Overall, we all learned a great deal and were satisfied with the level of our work.

## 11.1 Acknowledgements

We are grateful for the time that our supervisor, Alice Toniolo, has given towards this project, as well as the patience and kindness she has extended to us throughout.

We are also grateful to Juliana Bowles, who was our supervisor at the beginning stages of the project and put in time to help us develop an initial strategy for our program.

## References

[1] Josh Baker. Laser Cut Settlers of Catan Board. http://www.instructables.com/id/Laser-Cut-Settlers-of-Catan-Board/.

[2] Google Developers. Protocol Buffers. https://developers.google.com/protocol-buffers/.

[3] The Eclipse Foundation. Eclipse. https://www.eclipse.org/.

[4] Oracle. NetBeans IDE. https://netbeans.org/.

[5] Klaus Teuber. The Settlers of Catan: Game Rules & Almanac, 1995. http://www.catan.com/en/download/?SoC_rv_Rules_091907.pdf.

[6] USM. Catan Universe. https://www.catanuniverse.com/.