Interdimensional Panopoly

Final Report

Bowl of Milk

Brian Emmett 15352991

Chloe Jolliffe 15364626

Cian Kelly 15386256

Tomás McPhillips 15382091

Table of Contents

Introduction

One of the principle aims of this project was to reproduce the board-game version of monopoly in the form of a Java program. The other aims of this project centred around the idea of removing the repetitive nature of monopoly by introducing procedurally generated content that would be populated from a given NOC[1] list. In short, the core objective was to increase the replayability of the game.

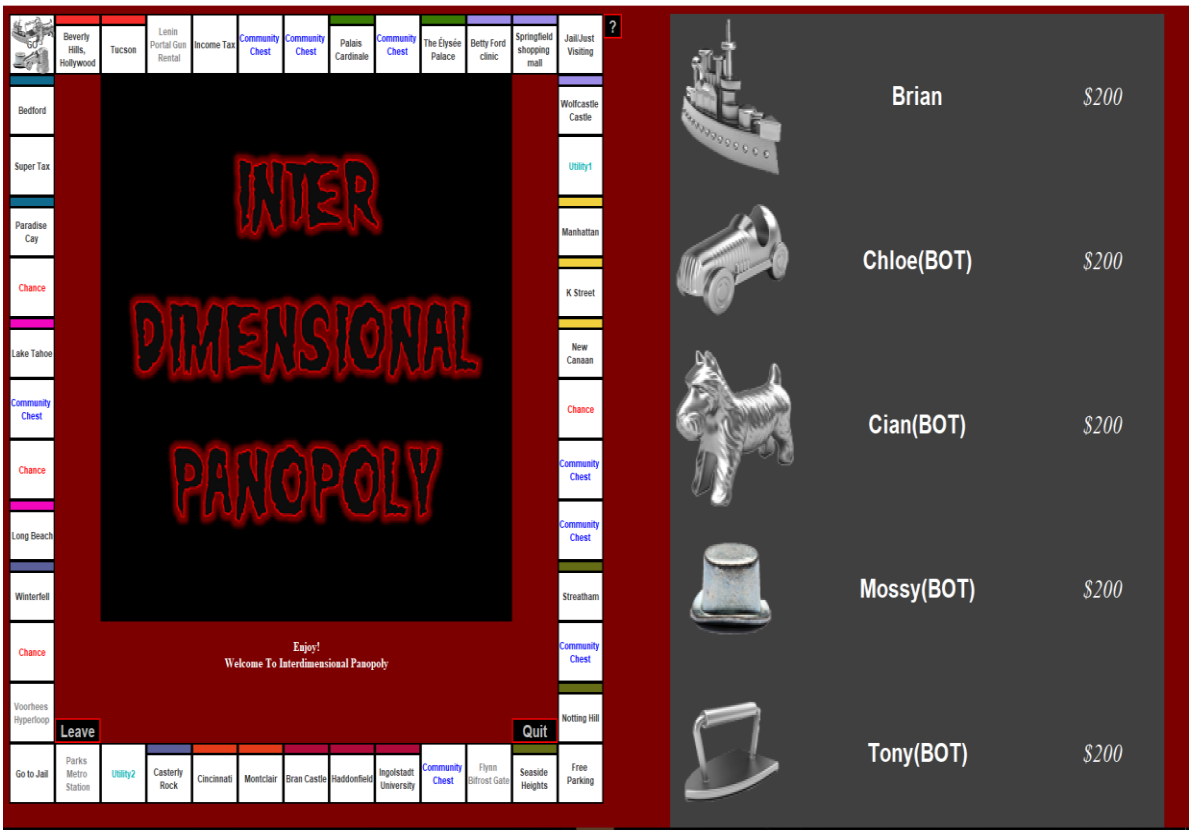
To achieve these aims, the group firstly set out to reproduce the game in its standard format as a Java based program and to then add certain features to remove the game's inherent monotonous nature. At the onset of the project the group outlined the key features it wished to add to achieve the core objective of the project. These features that were planned from the output are the following: to produce a board where everything is procedurally generated from the number of properties, colour groups, chance squares, community chest squares, stations and utilities. The names of these properties, stations and utilities all of which are randomly taken from the characters of the NOC list at the onset of the game were also altered. Another core feature being that of the chance and community chest cards which again will all be events inspired by the characters and their associated traits taken from the NOC list, the consequences of the cards also will also make sense within the context of the game as opposed to the traditional version where the cards and their related consequences are often far from complicit with each other.

Another feature proposed by the group was the introduction of questions populated by information from the NOC list to allow a jailed player to leave jail, this feature would bring new life to the art of escaping jail as opposed to just getting lucky by rolling a double as the traditional version of the game would demand. The last feature the group proposed, dealing with gameplay, was an idea to dramatically shorten the average time taken to complete a game of monopoly, the group aimed to achieve this by bringing about the apocalypse. The idea was proposed to introduce a card which would predict the end of the world and set in motion an in-game timer and when said timer expires, the game ends, with the victor being the player with the greatest sum total of cash and assets.

[1] https://github.com/prosecconetwork

In terms of creating a Java based version of monopoly the group also set out to introduce the concept of peer-to-peer networking, artificial intelligence and a mobile version of the game to further bring the game into the modern era and expand its possible platforms and playability.

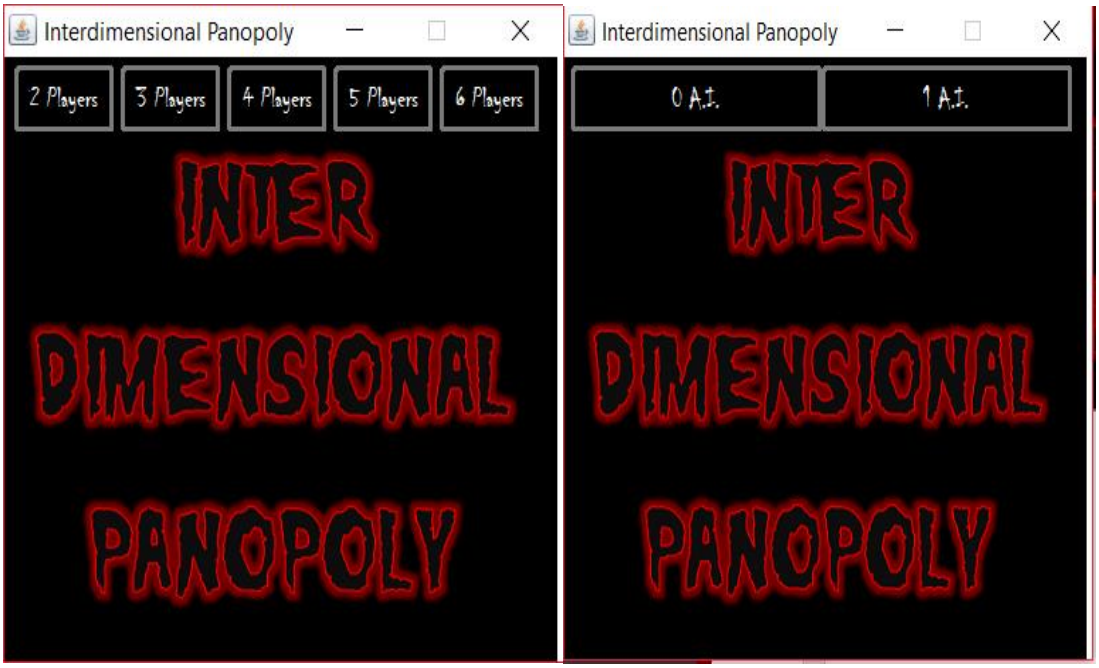
The final project displays all of the proposed features mentioned minus the artificial intelligence aspect which was dropped in favour of decision tree based bots and the Android version of the game which was scrapped due to the steep learning curve involved and the pressing time-constraints of the project. Added to the core objectives of the group, are aspects to bring a sense of the modern world and the fictional components of the NOC list into focus on the board, such as changing stations to more modern forms of transport both tangible and fictional. This was also done with the utilities, modernising them from the dated waterworks etc. to more modern utilities found in today's world. The group feels that the finished product detailed in this report accomplishes all the goals listed at the onset of this project and displays aspects of all the topics covered during the course of the module and implements numerous aspects encountered during the course.



(board design)

Actual Design

This section on actual design consists of a full rundown from start to finish of what a user may encounter during the course of playing a game of Interdimensional Panopoly. The game starts by means of an introductory screen where the player is asked to select the number of players required for the game, from between 2 and 6. Next the user is asked how many A.I or bots the player wishes to play against. The final introductory tab asks each player to enter their name and to select their icon from one of the six standard icons available and to confirm their decision. The display on the right-hand side of the window displays the players, their balance and indicates the players whose turn it currently is. It will also indicate if a particular player is currently in Jail.



Once past the introductory screens, the user is presented with a game board (one for each human player). From here the player uses the labelled buttons to interact with the Panopoly game. Any questions on how to play the game can be answered by reading the help file accessed through the help (?) button. To start the game, press the roll button, to end the player's turn push the end button. The roll button will then be displayed to the screen of the player whose turn it currently is. This player and this player only will have the interactive buttons such as buy, mortgage, build, etc. during their turn, however all players have the option to leave the game at any time, by way of the leave button. Should a player own an entire colour group they will be permitted to build houses or hotels on the property increasing its rental value. This conforms with the traditional rule of even build, whereby there cannot be more than one house worth of difference between any two properties of that group. Players can trade with other players by selecting the player you wish to trade with clicking the trade button and selected the property to be traded.

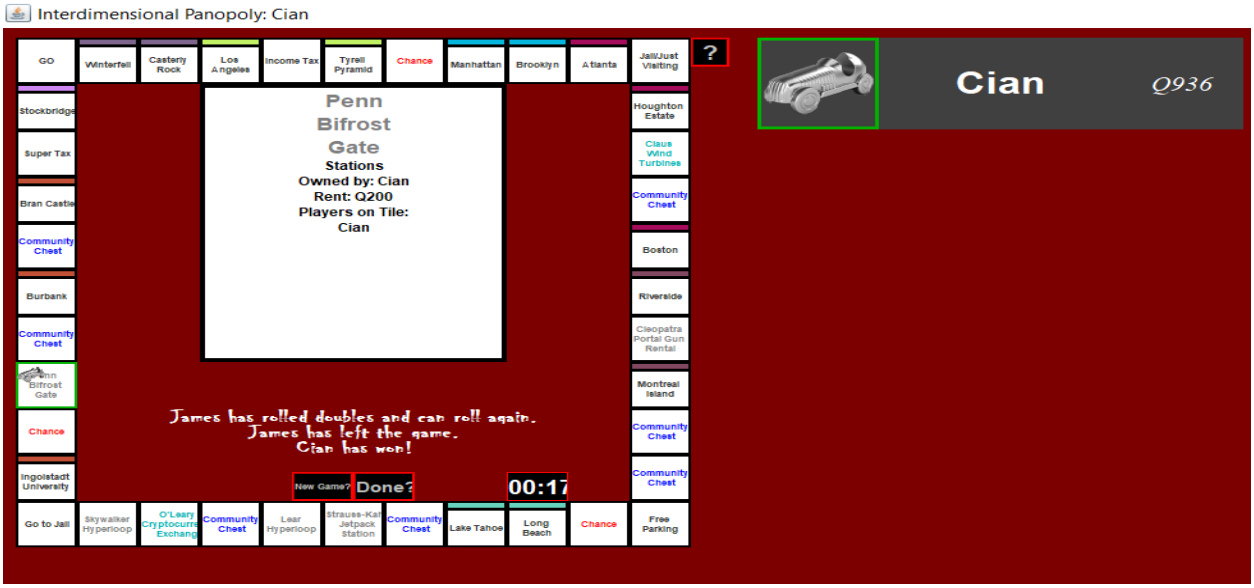
(two player game boards side-by-side)

Should a player land on an unowned property and elect not to buy said property, this property will then go to auction where all players will have the option to bid within the 15 seconds time frame using the bid button that will appear. Bids are made in instances of 10% of the properties overall price. The player with the highest bid will at the conclusion of the auction attain the property. To build or manage a property a player can click on the property square and perform the actions available. If a player is sent to jail they must answer a question correctly in order to free themselves of their confinement, if said player does not correctly answer the question they

will remain in jail indefinitely, whereas if they get the question correct they will be freed from jail and their turn will end.

Should a player land on an property which is already owned, the rent is calculated and automatically deducted from the player's balance and paid to the owner of that property. If a player lands on a tax square, they cost of the rent or tax is automatically deducted from the players balance. Go to Jail performs as in normal Monopoly, any player that lands here is automatically sent to Jail and their turn ends.

If a player lands on a chance or community chest card, a box will appear bearing the content of the card, the consequences of the card are imposed on them immediately. Said consequences could be a change to their balance, position on the board or their incarceration status. If a player draws a doomsday card a timer starts counting down from 5 minutes, only one of which can be drawn per game. When said timer expires the game will end and the winner (player with greatest sum total of assets and balance) will be declared. Similarly, if all but one player leaves the game before its conclusion the remaining player will be declared the winner. If a player is bankrupt i.e. they cannot pay off their debts, they are unable to make any move except to leave the game. If a player leaves, the property they owned and or any houses built returns to its default state of being an unowned property with no buildings. Once a player has been declared a winner, two new buttons appear to replace all others; Done and New Game? Clicking the done button exits the program whereas the new game button launches a new version of the game with a completely fresh board.



(winner game board)

System Description

Procedural Content Generation:

Properties:

In accordance with the group's aim of having a board driven entirely by procedural content generation, all the properties must be randomly generated from the elements of the NOC list at the onset of each new game. This was achieved by leaning on the pre-existing classes that come linked with the NOC list while also expanding and morphing these existing structures to fulfil the needs of the project. In the **PersonOfInterest** class a new instance of the NOC list was created and equalled to a new instance of the **KnowledgeBaseModule** which serves to read in the NOC list and to return the first column results of any searches. A ArrayList of strings called **locations** was created to store the resulting property names taken from places listed in the NOC list. The Vector string **checkers** was created and in conjunction with the **WORLDS** instance of the **KnowledgeBaseModule** class (which takes in the domains from the Specific Domains .csv file) held its values. Using a for loop with **checkers.size** as its upper limit, the string **world** was passed into the **allPeople** vector string, which called the function **getAllKeysWithFieldValue** for the domains fieldname and with the appropriate world string passed in for each iteration. If the number of properties within the listed domain was greater than or equal to two, then the names of said properties and their attached domain world were added to the **locations** Arraylist. To remove duplicates which can occur from properties which are covered by two plus domains, the ArrayList's contents are passed through a set called **removes** before being added back into the **locations** Arraylist. **Locations** is then used at the board instantiation to generate the properties.

Given the groups free reign in changing the game, the group also elected to change the stations names to utilise the surnames of the characters from the NOC list, this was achieved in the functions **Surnames** which returned a string of a randomly selected surname which was separated using the .**split** operation. This function was then called in another function called **TransportLinks** where the surname string was appended to a random string taken from a finite string array holding transport types (e.g. monorail, Hyperloop, etc.) and the whole string returned for later use with the board related classes. The **Utility** class is almost identical to **TransportLinks**

and serves the purpose of renaming utilities with a character's surname while also updating the type of utilities seen on the board.

Most board locations are randomly generated with the exception of Tax squares and Named Locations, i.e. GO, Go to Jail etc. The other types of properties; Stations, Utilities and Investment Properties have varying levels of procedural generation. Only two Utilities are generated for each board and they are always in the same positions, however, their names are generated randomly each time using the previously mentioned functions. A variable amount of Stations are generated for each board. Investment Properties have the highest proportion of procedural content generation, the number of these properties, their prices and rents are all generated at instantiation. The board is generated in a clockwise rotation from GO, with a starting minimum and maximum price for each investment property, the actual price is then a random integer in this range. Each address is formed from **locations**, first the property addresses are split from the group, then the address is split on a comma to allow for multi-line addresses. Each of these addresses are added to an ArrayList and to a HashTable, where the key is the group i.e. the domain and the value is the address. This removes the possibility of the same address appearing multiple times. As instantiation moves around the board, different groups are selected and a maximum of three properties are created for each. The minimum and maximum price is constantly increasing as the board is generated resulting in the later properties having the highest prices.

A rent array is generated based on the actual price a property is given. This is done using a modified version of the patterns found in traditional Monopoly by Julian Wiseman[2]. Build price is set using the traditional method of a set price for each side of the board.

Cards:

Following the objectives set down at the outset of the project, the group decided to build the chance and community chest cards based on the information held in the NOC list. To enable this the pre-existing function of the **KnowledgeBaseModule** was used as a building platform. Numerous vector strings were initialised and assigned to return various different columns values (e.g. Vehicle of Choice, Opponent, Address 3). Each of these vector strings was then used in one or more functions all of which returned an object of type Card which incorporates a string.

[2] http://www.jdawiseman.com/papers/trivia/monopoly-rents.html

Depending on the type of card, other variables may have been introduced as necessary, such as if the objective of the card was to change the players position, the number of squares traversed would be randomly selected or if the purpose of the card was to fine or reward a person then this variable too was randomly selected, all of this only further added to the randomness of the game. In terms of the card type themselves, there are 6 different card types: one that rewards a player with money, one that fines or punishes a player's balance, one that sends a player to jail and its opposite that allows the player to get out of jail for free, and the final 2 types change the player's position, one dealing with forward movement and one with backward movement. In certain circumstances the other traits of a selected character could also be appended to a string, an example of this being randomly selecting a player's opponent. Within these 6 categories there exists 21 different card types, each with a minimum of 5! possible different outputs, making the chance of a repeat card appearing in the same game or in back-to-back games a statistical improbability.

When a player lands on a Chance or Community Chest square, a card is randomly drawn. Each type can be drawn with equal probability except for the Doomsday card which will be discussed later. The card's consequence is performed automatically and the string is passed back to the GUI.

Questions:

The group decided to replace the random escape from jail process with a question allowing the player to win their freedom with their knowledge as opposed to by pure chance. The **PersonOfInterest** class is used again in conjunction with the **Question** function. This function returns a string array with one of four questions which is decided by a random variable, the correct answer and three wrong answers. For each of the question types a vector string called **question** calls the **getAllKeys** method which returns all the values from a column, from this column a random option is chosen and input into the question string. Next the answers both correct and incorrect were gathered. The correct answer was obtained using the new method **getAllKeysWithFieldValueNon** which was assigned to the first element of the string array **qandA,** if multiple answers are correct, one is randomly selected. Whereas the incorrect answers were attained using the method **getAllKeysWithoutFieldValu**e which gets all the elements from a column not associated with the question value, these values were then assigned to the remaining value of the qandA string array before it's returned. The order of the answers are then jumbled, once the **Questions** function is called in another part of the implementation of the game.

Doomsday:

A key feature for our project was to interject a sense of urgency into the game once a card is drawn to increase the games excitement level and to shorten the games average playing time. This card was again generated from the NOC list in the following manner. As in the other card methods, the **Doomsday** method uses a vector string, in this case **villains,** which takes in all characters who have villain as a trait listed under the heading category. A random value is taken from these inputs and appended to a string along with the method they will use to bring about the doomsday scenario, this is then returned as an object of type Card for implementation with devastating consequences in other areas of the program.

Once a Doomsday card is drawn, no other Doomsday events can occur. The Doomsday card calls a method **startCountdown()** which creates a timer that ticks every second. This updates a Doomsday clock which appears in the lower right of the Panopoly board. Once this timer reaches0, the player with the highest net worth is declared the winner and the game ends.

(example of a property card)

GUI

With regards to the graphical user interface (GUI), many design decisions were made to keep a consistent look and feel about the board. A black, scarlet and grey colour scheme was decided upon and kept consistent throughout. Similar borders and such are used throughout. The use of the Chiller font resulted in an accidental generally Halloween look. This ended up influencing the whole design and look of the board as it played in well with the colour scheme.

The **GUIButton** class was influenced by the factory design pattern, which was used to give each button within the board the same look throughout. Each button is kept mostly the same size but given a distinct name, actions and widths. All buttons keep a consistent height.

The GUI object was designed for a networking or mobile development approach. Each GUI object is associated to a player and the panopoly, in a client and server approach. The GUI was designed to be highly relative so that objects could be moved or shaped to fit into a display of smaller or different proportions. The final layout of the GUI having multiple game windows for human players on the same machine is a result of attempting to make use of networking. The preferred look of the board would have one window per player at each computer. The closest to this final look is a game with a full AI and a singular player.

The approach was taken to restrict the player from making invalid moves, by restricting their access to buttons in the GUI. This was to speed up the game and provide the player with an overall quicker easier experience. To also assist with this, the GUI will provide the player with as much relevant information about each location it visits through the use of the **locationWindow** JLabel.

Android

This interdimensional panopoly project took a lot research into the areas of multiplayer board games and the android approach to such a game. The idea for the game was to have a procedurally generated board in a range of squares. The options of both android native development and **libGDX** graphics game development engine development were researched.

Android native development was a slow procedure and posed the difficulty of the entire group working on android development due to not everyone having an android phone. Android gradle scripts could not be executed on computers and the desktop version of the game could not be developed in the android studio framework and had to be worked on in external GitHub

repositories. The approach taken with android native was to generate a board and to use enumerators and game states. The use of transitions animations was used to move players, but the ability to make randomly generated squares clickable proved challenging.

LibGDX development was explored and had a much better game development environment. This framework allowed cross platform development for android, ios and desktop. The framework was a steep learning curve with many new skills being learned to make a good user interface. The GUI consisted of an actor a procedurally generated board which was put on a stage (The based layer to add actors too). Sprites (player icons) were actors added to the stage and many actors were added. The Game used the scene2D framework and JSON files to make a presentable GUI. Learning a graphics framework was difficult and issues posed with input multiplexers for layers, viewports which worked correctly with JSON pop ups. The development of the GUI in java native using swing was a much more doable option due to its familiarity.

Networking

The game was designed in such a way which would allow multiple users to have a user interface. The game panopoly is optimized for a turned based game which was soon realised as development continued. The using of TCP/IP was a viable option for this project. The making of a server with multiple clients was the approach taken for this project. The board was randomly generated for each game the implementation of clients where each had the same GUI over a network became an impossibility. The same panopoly object would have been needed for each clients and TCP is not suited for passing large objects such as a game. Had this hurdle been overcome in the limited time frame networking would have been achieved in full due to how the group designed the GUI.

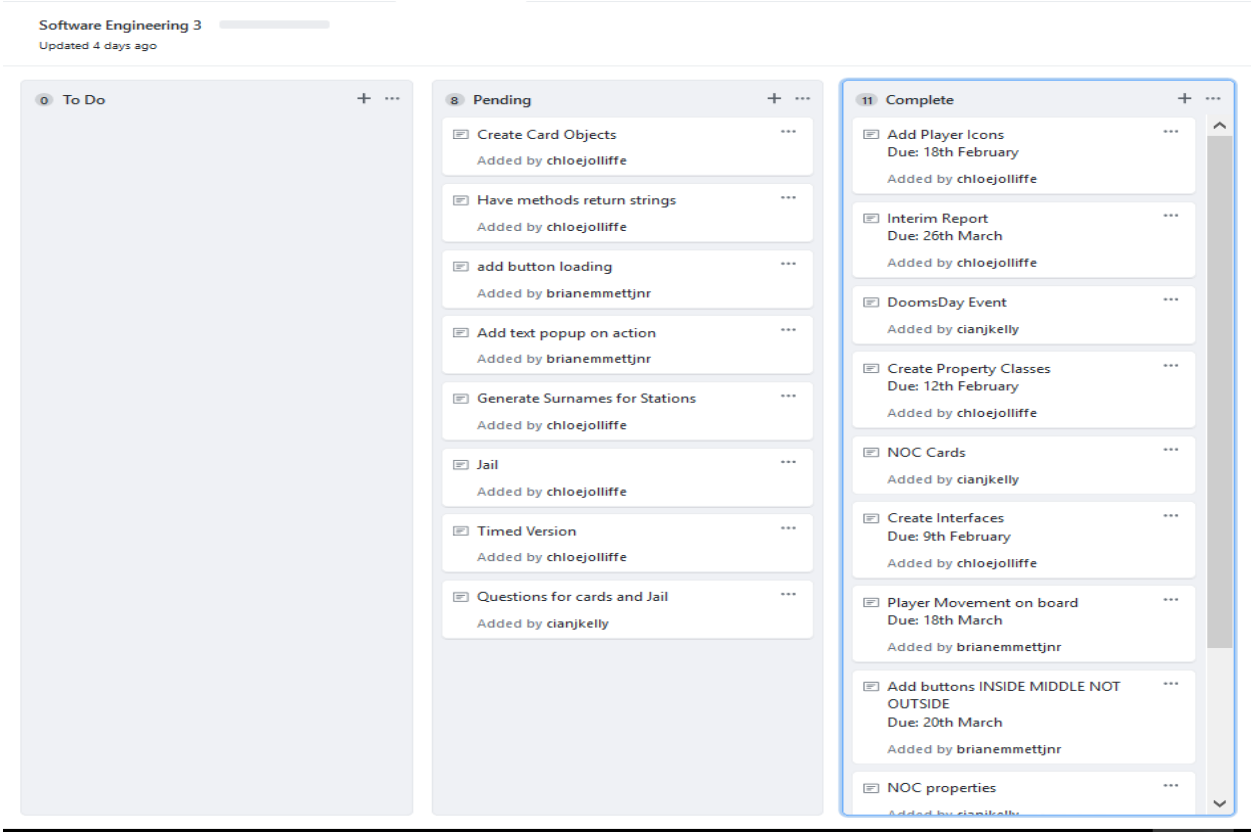
Bots & AI

Game playing bots are used to simulate a player playing the game. These bots must make decisions based on the current game state. Bots were a feature introduced when a playable game was available. Each bot extended player as it would have the same functionality as a player. The option at start-up is given to specify whether you want to play against bots or actual players. If bots are selected, they do not have a GUI but play in the background using the **InteractionAPI**. At the exchange of each player, the player class would reflect on itself and see if it was an instance of a bot and if so, the **makeDecision** functioned was called. The **InteractionAPI** allowed the

player to access methods which could make changes to the state of the game and the GUI. The process for making a decision for the bots relied on buttons being visible presenting what options were available for them. If certain criteria was met the game bot would make its best possible, move in the game situation that was available to it. The **InteractionAPI**, **GameBot** and GUI were all adapted in tandem to give a game playing experience which was successfully provided by a computer player.

Team Analysis

At the onset of the project the team divided the workload into the following areas: Procedural content generation populated by the NOC list, User interface and the board, General gameplay, Networking, AI and android applications. Each of these areas were then evaluated to ensure they were achievable within the time frame, before being assigned to a member of the group based on the group member's interest and strengths. In keeping with the techniques learned during the duration of the course the group then began development of the project following the guidelines defined by the Agile software development life cycle.



(example of a scrum board for whole project)

However, despite the division of the work into manageable workloads and the assigning of group members to that group the team still remained closely connected through weekly scrum meetings with a rotating scrum master. This allowed the group to have a good idea of the overall progress of the project during its progression, while also allowing the group to aid or suggest new

paths to each other should a member become bogged down with their workload. The group also utilised pair programming when two aspects of the project were closely related.

During the course of the project the group used all available tools to aid in the development of the monopoly game, such tools include but are not limited to: GitHub (where the group corroborated our efforts while also utilising the scrum board like feature that GitHub offers, Discord (which was used as a means of communication and screen-sharing, should a group member not be available to meet with the group in person) and Messenger (again used for short quick communication between group members).

Evaluation of Success

At the outset of the project the group was tasked with making monopoly great again. The task was set to remove the monstrous nature of the game and to drag it into the modern age. The core objectives of the project were all achieved. From a creative standpoint the group went to great lengths to put their own personal spin on the game of monopoly and to incorporate as much procedural content generation as possible. This is evident given the game board is being entirely generated using procedurally generated content. This procedural generation ranges from the number of squares of each type to their name, price etc. Cards are also generated in this manner making each game unique. The introduction of questions in place of a dice roll to escape jail, also brings a new and exciting aspect to the game, rewarding a player for their knowledge. Lastly the Doomsday event is a brand new feature to the game. Once the doomsday card is drawn the games pace quickens exponentially, due to the time constraints that are introduced. This pressure placed on the players increases their enjoyment of game while also simultaneously dropping the average time spent playing monopoly. Both of these only serve to further increase the replayability of the game and reduce its monotonous nature.

The team also wished to expand the reach of our game beyond that of a java program by implementing features such as artificial intelligence, networking and an Android version of the game. However, despite extensive research on all of these areas only artificial intelligence could be achieved in the form of bots that play the game. This is due to many factors including the very steep learning curve that comes with the learning of how to implement such techniques (which was detailed in the system description), coupled with the extreme time-constraints the team was put under given their other workloads and commitments. Due to the heavy research load one of

the group members was unable to contribute as many commits as the others, however, the group member contributed their full share to the completion of the project.

Overall the team believes they fulfilled the vast majority of what they set out to achieve and believe the final product fully shows their creativity and commitment to the task at hand.