2019

Name: David Terence-Abanulo Candidate Number: 3651 Centre ID: 12290

Woodhouse College

10/14/2019

NEA ANALYSIS FOR UNO PROJECT

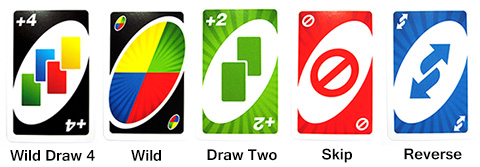


**Analysis**

**1.1-Introduction**

As a student, I find myself often looking for a quick game to pass the time when taking a break from revising and usually that game ends up being UNO. Therefore, my project for my NEA is to create a fully functional 2D shedding card type game that can be accessed as an executable file and run like an application on windows. This application would contain a single player function where the user can play against an AI and a multiplayer function with which up to 4 players can play locally (connected to the same server) or play against each other with the addition of an AI to fill empty spaces (if less than 4 players join the server).

The game itself works with up to 4 players, each player is dealt a random set of cards from the UNO deck, players then take turns placing a card from their hand into to central pile, a card a player places on the pile must be the same number or colour as the card on the top of the pile, however players can always use action cards in their hand (including skip player, +2 cards to consecutive player, +4 to a consecutive player etc.)

[[1]](#endnote-2)

Action cards in UNO

**1.2 – Outline of tasks (both programming and documentation)**

|  |  |  |
| --- | --- | --- |
| **Task** | **Priority (out of 5)** | **Due date** |
| Introduction | 1 | 15/09/19 |
| Analysis of similar systems | 2 | 20/09/19 |
| Analysis of existing system | 3 | 22/09/19 |
| MVC implementation analysis | 3 | 1/09/19 |
| Questionnaire | 3 | 6/10/19 |
| Prototype (deck of cards) with annotations | 4 | 8/10/19 |
| Researching AI algorithms to implement in python | 4 | 10/10/19 |
| Proposed Solution (include multiplayer solution) | 4 | 12/10/19 |
| **Analysis Completed & Printed** | **5** | **14/10/19** |
| **To Be Updated** |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**1.3 -Description of Existing Application**

UNO is a lot more established and well-known in comparison to other shedding type card games with many variations of the game available on Android, IOS, Xbox, PS4 and PC (windows). I will be focusing on analysing the windows version of UNO which is an application available on the mainstream digital application store, Steam. This is version which is developed by Ubisoft is the most popular online rendition of UNO to date with an estimated 500,000-1,000,000 owners of the game and the highest number of players online at once totalling up to 2,309 and up to 115,000 viewers on live streaming platforms such as Twitch.

[[2]](#endnote-3)

Line graph depicting the fluctuation of Players and Viewers of the game since its release in January 2017 till date

This version of UNO works essentially the same way as the original does, with each player given an equal number of cards from the deck and the remainder placed as a draw pile. Players must match the colour or number of a card placed down in the discard pile or use action cards randomly distributed amongst them (including, skip, reverse, +2, +4, and wildcards) to reduce the cards in their hand to 0 first. Once a player has one card remaining, they must call ‘UNO!’ when placing their final card in their hand, if this is not done then the player must draw another card from the draw pile.



2v2 option (multiplayer)

User account name and profile picture (there is a database that holds this information

Displays what card has just been played

In game Voice Chat

**[[3]](#endnote-4)**

Themed Cards

Colourful GUI to draw the user in-aesthetically pleasing

Number of cards each player has left



Score

**[[4]](#endnote-5)**

**Strengths (according to User reviews on steam)**

* Colourful and exciting GUI really enhances the user’s game experience, arguably better than playing in real life
* Good game to play online with friends through multiplayer
* Themed cards add an interesting twist to the gameplay
* Offers controller support
* Available in multiple languages

**Weaknesses (according to User reviews on Steam)**

* Game is not worth $10
* A max of only 4 players allowed to play in one game
* No support for rebinding keys to user’s preference
* Noticeable latency, poor online connection during multiplayer
* If you don’t have friends that play the game, then your experience with single player AI would be very dull, and searching for an online match takes too long
* Game frequently crashes
* Poorly optimised for users on different

This version of UNO works exactly as a normal game of UNO should and attracts users with its vibrant colours and easy to use GUI however the multiplayer aspect (which seems to be the most sought-after feature from users) is very poor due to latency severe latency issues and game crashes. This leads me to believe that this version of UNO is poorly optimised for networking. Furthermore, this game is only available on Windows OS and its minimum requirement to run is Intel Core i3 530 2.93GHZ which leaves out a considerable number of PCs on the market.

Based on this current version I plan to create a similar GUI game display, with players situated in the corners of the screen and only being able to view their own cards and the last placed card in the middle with the addition of a score counter next to each player that will increase or decrease based on cards played and whoever wins the game. I aim for my game in multiplayer to have a room/lobby that players can join and launch a game together across the same network server. Since a common issue is latency, I will endeavour to reduce latency in my game. I also will ensure my game can be run on majority of modern computers and laptops which includes devices running windows and MacOS. This version of UNO also requires you to have the Steam Engine installed on your desktop to play whereas my version will simply require python.

**1.4-Analysis of Two Similar Card Games**

As UNO is a shedding type card game, I have found another two games that are similar to it:

**Boom-O (created in 2001):** In Boom-O each player is dealt 7 cards and 3-time bomb cards that represent ‘lives. Players are given cards that can either increase or decrease the timer. The main objective is for the player to put down one card per turn until they have no cards in their hand while trying to keep the timer total below 60 seconds. If a player can’t play the correct card, they must turn over one of their bomb cards, losing a life. Once a player clears their hand, all other players must turn over their bomb cards and the survivor wins.

****

Bombs card (3 per player)

Discard pile

Draw pile

[[5]](#endnote-6)

**[[6]](#endnote-7)**

**Strengths**

* The element of a timer and ‘lives’ implemented into the card game adds a sense of intensity and urgency when playing the game, this allows the game to not last too long but be a fun game to play with friends
* The game is heavily reliant on the aspect of luck which means everyone has an equal opportunity of winning, no matter how experienced player(s) are
* Unique action cards such as ‘trade hands’ and ‘double play’ can completely change the course of gameplay in an unexpected way which makes playing the game more fun

**Weaknesses**

* The game is heavily reliant on the aspect of luck which means there’s no real strategy to learn or create.
* There is no score counter in the game which could serve as another method of winning, e.g. if a player score reaches 100, they automatically win. Scores could be based on cards added to the deck.
* The rules can get quite confusing as players have to keep track of both the timer and the number of cards, they have left in their hand

**Craits (created in the 1970s):** In Craits a standard card of 52 cards is used instead of a specialised pack in UNO and Boom-O. There are 15 hands and in the first hand 8 cards are dealt per player, this amount decreases by 1 after every hand till 1 card is dealt to each player and then incremented by 1 until once again 8 cards are dealt to each player. Like UNO, each player takes turns in placing a card from their hand on to the pile with according matching suit or rank (this excludes an eight or nine which are wild cards). Furthermore, if a two is placed on the pile at any point ‘the count’ begins in which each player must place an Ace or a two which increments the count by 1, once a player fails to place an Ace or a two they must draw a number of cards equal to the count e.g. if the count = 10 then the player must draw 10 cards from the deck.

**[[7]](#endnote-8)**

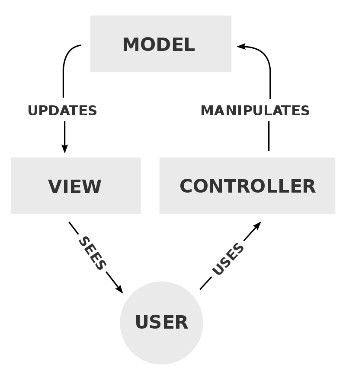
**Strengths**

* The fact that this game utilises a standard deck of 52 cards means that it is more convenient and cheaper to play as one does not need to buy a specialised deck

**Weaknesses**

* The game can get repetitive and boring quite quickly as there is no scoring system for each player
* Players may find it difficult to remember the special attributes of each card as they are just normal playing cards, this may end up confusing gameplay

**Conclusion:** Boom-O and Craits are very similar to UNO in the fundamental fact that the main objective of the game is for the player to reduce their hand to zero cards before their opponents. Boom-O is faster paced and this may appeal more to children as games do not last very long and are more intense, on the other hand Craits is a much more slow and steady game that doesn’t really have any unique aspects to it as a shedding card game but this may allow for a more mathematical, strategic approach when being played.

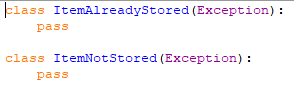
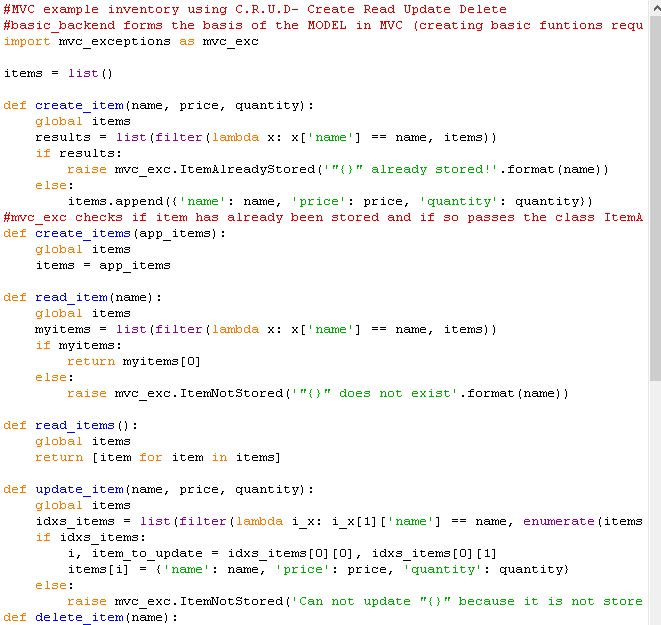
**1.5 Implementation Research**

**Model-View-Controller**

The fundamental structure of my UNO game will utilise an MVC which will allow me to manage the internal coding mechanics of the game and the design of the game in a more separate and effective manner. MVC stands for Model-View-Controller which is a software design pattern used for creating a user interface. The View is what the user sees and interacts with on the screen, the Controller is core code that runs in the background, processing inputs from the user and the Model essentially keeps a record of all the changes performed by the controller and updates the View accordingly for the user. [[8]](#endnote-9)

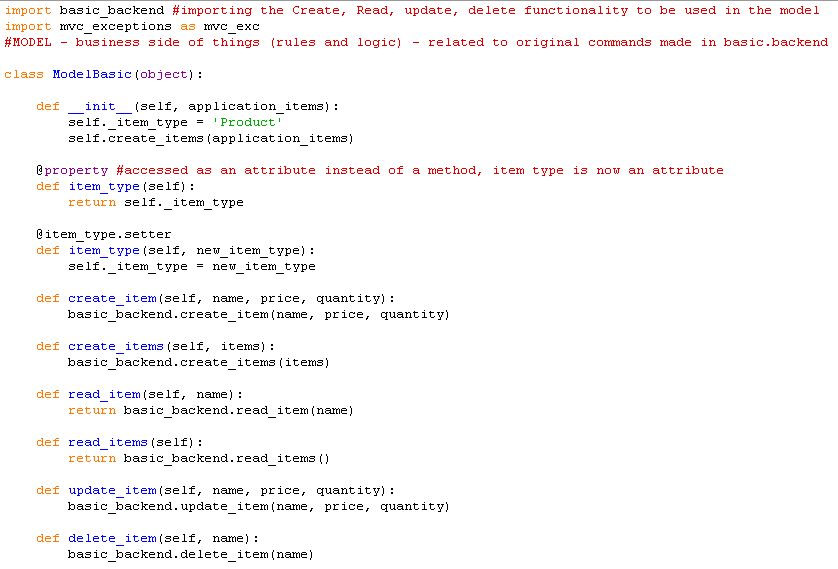
For instance, in my UNO game, the Controller would be the fundamental source code that controls how the cards are shuffled and evenly distributed amongst players, deciding which cards have special attributes and the effect on other players, keeping scores for each player, keeping track of which cards have been discarded and how many cards each player has left. This information would be passed on to a Model and then displayed visually to the user using Pygame.

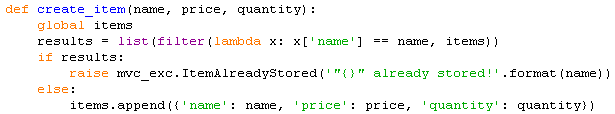
To learn how an MVC works in python and as a result how to implement my own one from scratch I have followed an online tutorial to create a mock inventory of a supermarket, in which the user can view what items are in stock, add or remove an item from stock and update the price or quantity of an item in stock. First, I had to create basic commands that could be done on a supermarket inventory (such as create, read, update and delete).

**** [[9]](#endnote-10)

In addition to creating the basic functions I implemented error-handling using try and exception: methods to be used if a user tries performing an invalid action such as trying to update an item that doesn’t exist

After making these basic functions I imported them into the main MVC program, uses the functions made is basic\_backend.py to formulate a model of the inventory, and a controller that takes user input from the view and refers to the model to make changes to the view.



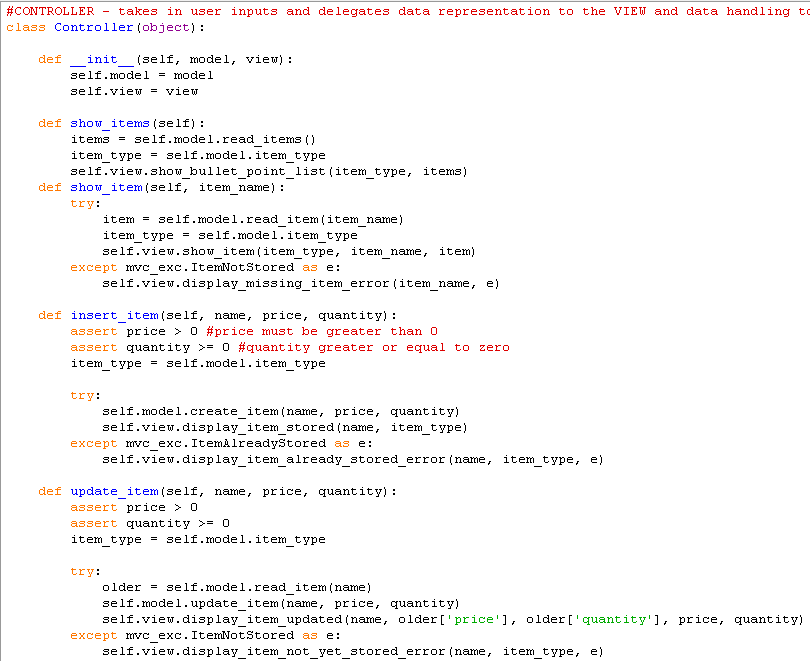
****

Above is a snippet of the model aspect of my MVC implementation. As you can see the basic functions defined in the basic\_backend file are used here

****

Here is the view aspect of an MVC model, as you can see, I created text-based images that would appear on the screen if the user attempted to perform certain commands e.g. creating an item that already exists

bvcccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

****

And finally, here is the ‘controller’ aspect of an MVC model. This is arguably the most essential part as it enables the main program to function by taking inputs from the view module, processing them based off the basic functions used in the model module and getting a result which is then outputted back to the view and the model is updated for the next input, this is done by inheritance and polymorphism. The controller essentially links the model and the view together.

bvcccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

**AI (Maths & Algorithms)**

**Maths and Complexity behind UNO**

**Maths & Monte Carlo** - Uno is essentially a game of luck and involves little to know strategy or maths. Which would make it difficult to create an AI. The AI in my game should be able to analyse playing patterns of each player and play cards accordingly to win (which may include predicting what cards a player has in their hand based on what cards they already played).

Assuming action cards such as reverse, skip, +2, +4 and change colour are not considered. In single player mode against an AI, you and the AI are given an equal number of cards. There are 52 cards in a deck, which means each player get 13 cards. Out of the total 52 cards, there are 4 suits (red, green, yellow and blue) and each suit has 1 set of 0-9 cards and. This means there is a 10/13 chance of getting any 0-9 card of any suit and a 1/13 chance of getting a +2 or reverse or skip card. In terms of analysing which card each player has in their hand, this means that players are more likely to have numbered cards as oppose to action cards and if there are a total of 4 players then it is likely that one, or even two players may not have actions cards at all. This would be helpful if one was taking a more mathematical approach to creating an AI like the Monte Carlo method. The Monte Carlo method uses probability distribution to evaluate the success and failure of a model. I would use this in my UNO game by inputting a range of probability values for the chances of each type of card being placed on the pile after every move in the game until a user empties their hand. The Monte Carlo method would repeat this for random ranges of probability values (from most extreme to most likely) until it has an 'accurate' outcome for each situation. For instance, if the AI and the user were trying to win after a Blue 7 was placed on the pile, the AI would go through a bunch of different scenarios such as, playing a wildcard on to the pile or a Blue 0, or Red 7 etc and keep playing till it or the player empties their hand. If the AI empties their hand first (thus winning) the AI would record how many moves it took to win and whichever method took the least number of moves would be assigned a high value of probability. Eventually after going through every outcome, my AI would have a list of efficient method to win against a user. Each method could be put into a tree that my AI can traverse and based on the situation my AI would pick a tree and traverse it. This method is laborious but it suitable for a game such as UNO that doesn’t necessarily have a distinct probability attached to each move (as the game is more luck based than strategy).

**Minimax Algorithm** - As an UNO AI would essentially be searching through a set of valid moves, using the optimal moves to get to the goal which in this case is getting rid of all the cards in its hand first. This means that I can represent this as a tree algorithm, where the start node is when the first card is placed on the pile and the AI has x number of cards in their hand and the final node is when the AI has no cards in their hand. Now I must start to think about how an AI would traverse this tree algorithm from node to node in the fastest route possible.

This is done by using the Minimax algorithm which finds the optimal move for a player. It is typically used in two player turn based such as Chess and Tic-tac-toe this means it should work well for my UNO game. In a Minimax algorithm the two players are defined as either the maximiser and a minimizer where the maximiser attempts to get the highest value possible and the minimiser attempts to get the lowest value possible and they traverse a tree from state to state with each state corresponding to a defined value, these values are calculated using a heuristic function. In terms of tic-tac-toe, using the heuristic function one could calculate the value of the board based on the certain state of the game, e.g. if the maximiser wins the board then the value is set to 10 and if the minimiser wins the board then the value is set to -10. This idea is used to form the basis of a tic-tac-toe game AI by creating a findbestmove function, a minimax function which considers all the moves the AI can make and returns the best value for that move. For this algorithm to work, the AI would be the maximiser and the other player (in this case the user) would be the minimiser.

An example (which I will be following) of minimax being used to find the optimal move for an AI:

**function** minimax(board, depth, isMaximizingPlayer):

**if** current board state is a terminal state :

**return** value of the board

[[10]](#endnote-11)

**if** isMaximizingPlayer :

bestVal = -INFINITY

**for each** move in board :

value = minimax(board, depth+1, false)

bestVal = max( bestVal, value)

**return** bestVal

**else** :

bestVal = +INFINITY

**for each** move in board :

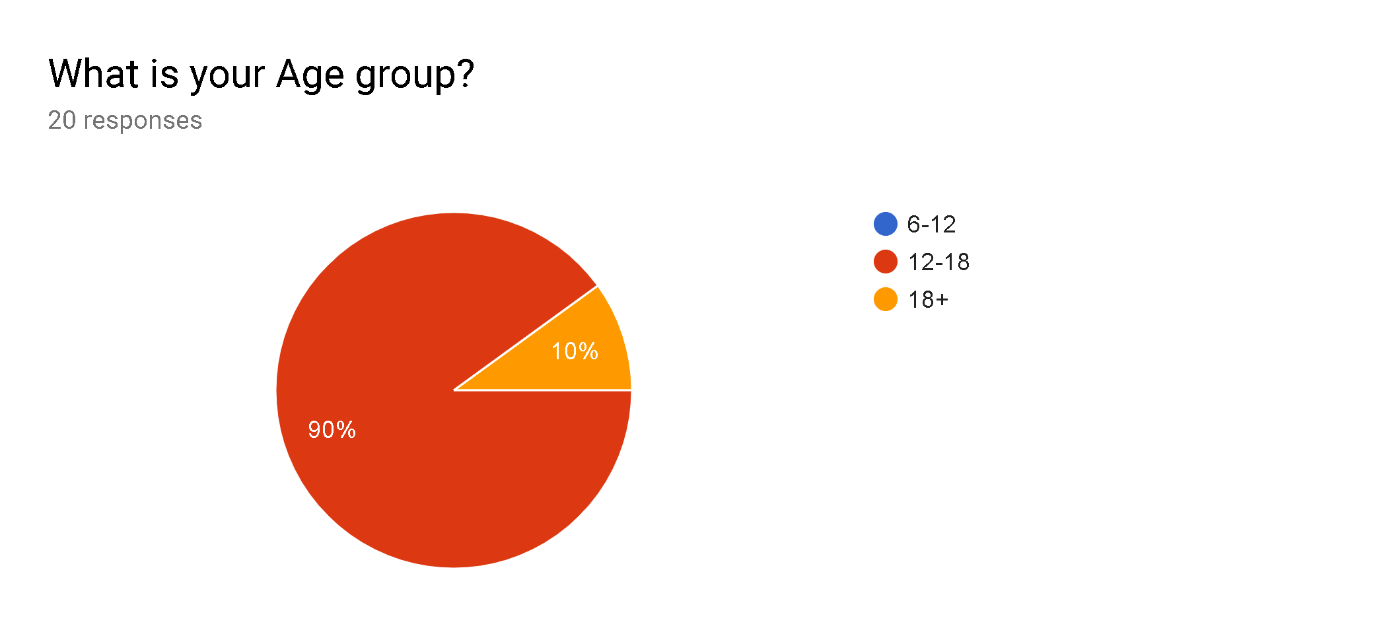
value = minimax(board, depth+1, true)

bestVal = min( bestVal, value)

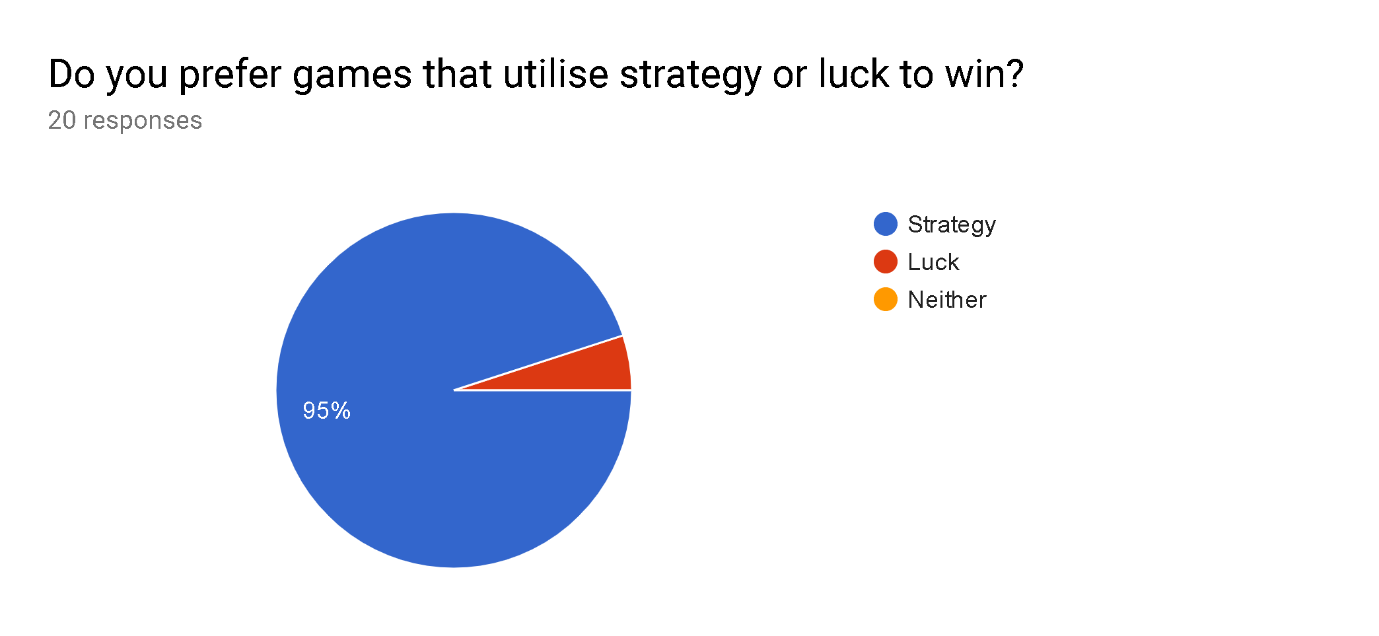
**return** bestVal

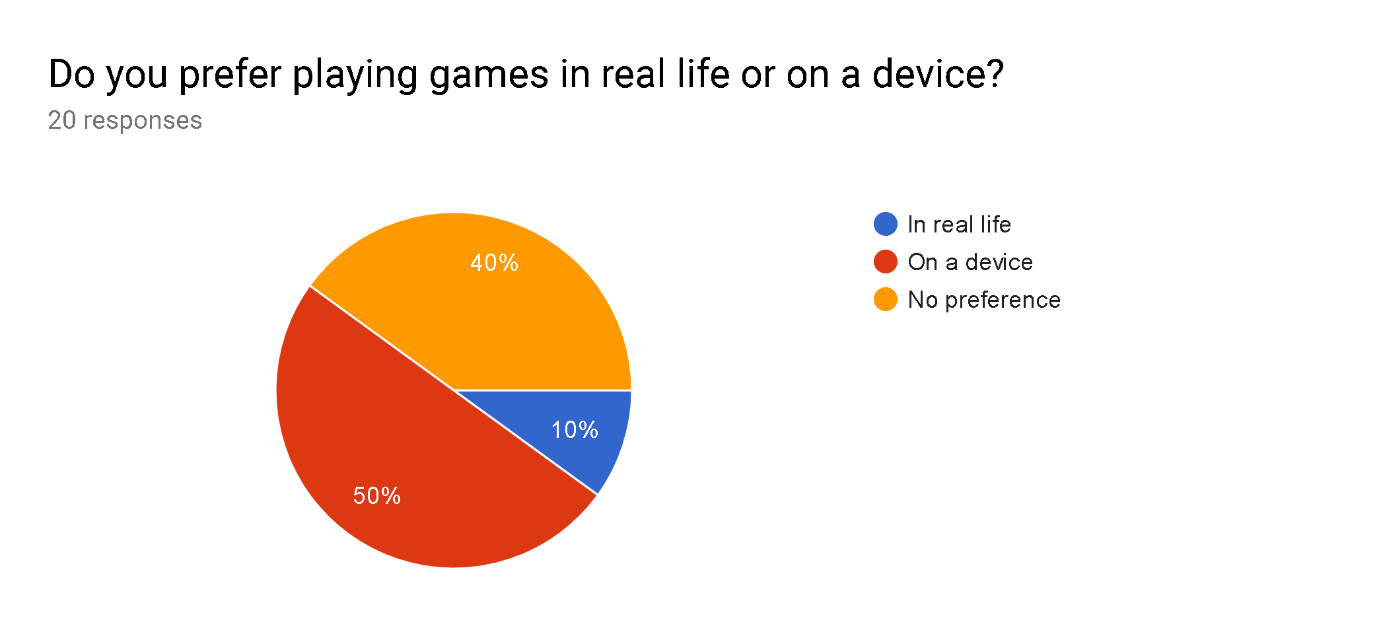
**1.6 Analysis of Responses from Questionnaire**

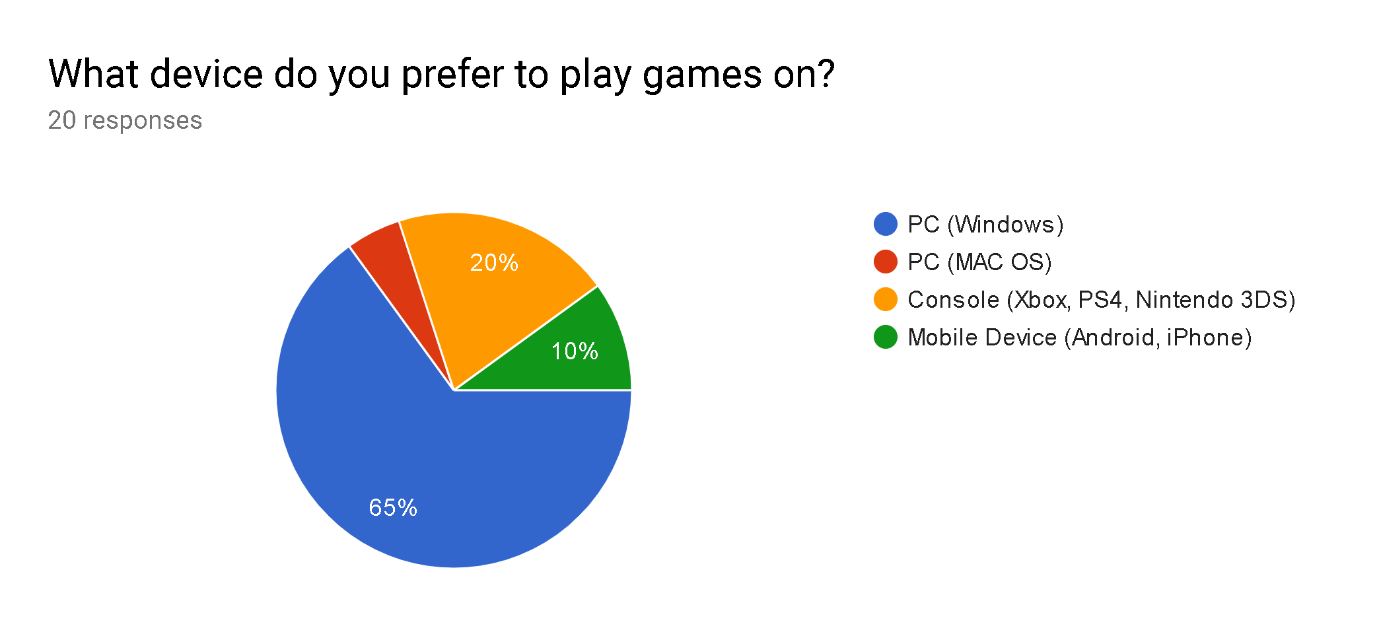
On the 24th September I sent out a gaming-based questionnaire to learn more about my potential users and their views on certain aspects of gaming. These were their responses:



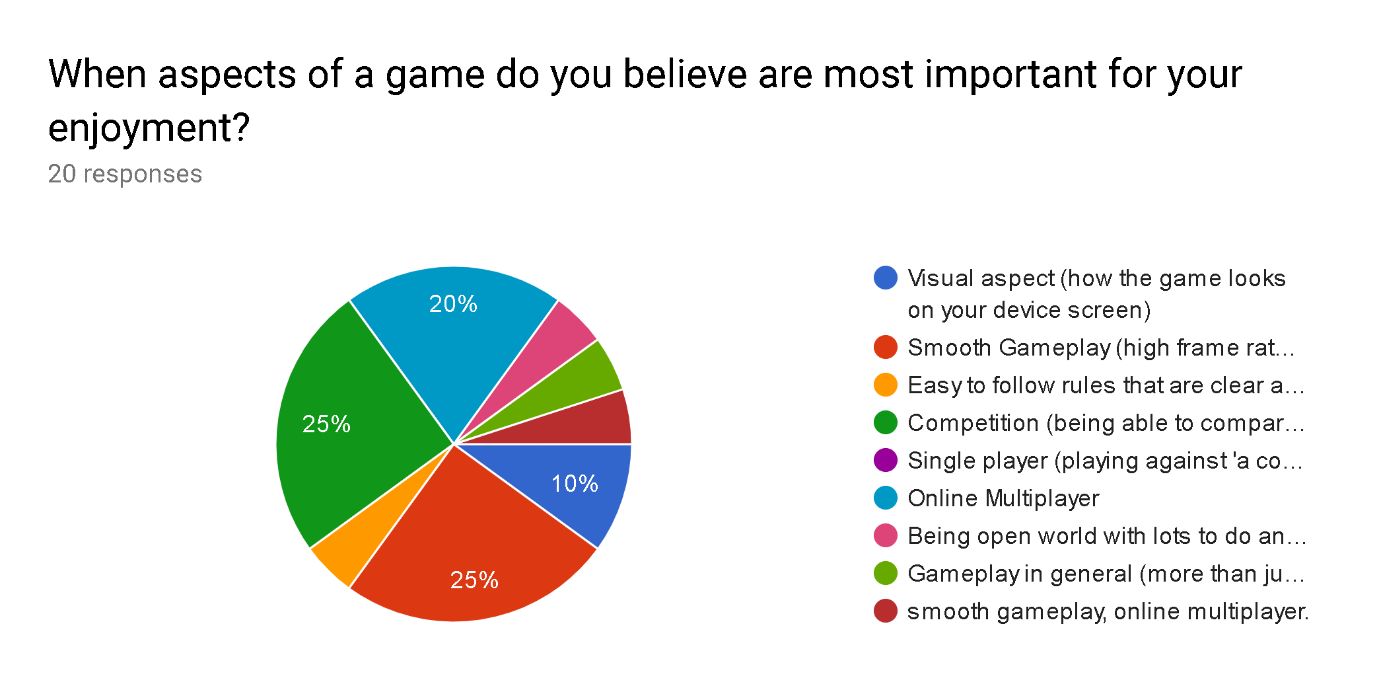
A significant of my potential users will be teenagers between the age of 12-18, this means I can assume that my users will be familiar with the technology used (in this case a computer or laptop) and as majority of my users are in their adolescence they should be able to catch on to the rules of UNO at a much faster rate than someone older which means I have to won’t focus on explicitly detailing the rules of the game.

A resounding 95% of responses said they prefer a game that involves strategy. This is good because strategy in a UNO helps make the game more interesting and addictive because users will constantly be thinking of new ways to play certain cards at certain times to ensure they win more games. To motivate users, I intend to create a function that gives users extra points based on how quickly they manage to win a game, urging them to think of fast UNO strategies and giving them an incentive to keep playing to try beat their best time. Furthermore, with the implementation of an AI users will be able to see how their strategies stack up against a ‘computer’s’.



This question received mixed responses, but majority said they prefer playing games on a device. This is probably due to the added convenience playing on a device has, as users can play anywhere at any time and with any of their friends. My UNO game will be limited to being played on a desktop but will allow users to play with their friends across a LAN network (to reduce potential latency issues)

Based on the responses from this questions, majority of my potential users prefer to use PC as oppose to a console or mobile device. In addition to this 93% of those who chose PC as their preference, favour Windows OS over MAC OS for gaming. This means that I will focus on making sure my game can run on windows OS. However, a secondary focus in the future would be trying to make my UNO game compatible with MAC OS users as this was the

second most popular choice. 

The responses for this question were quite evenly spread with the two most popular choices being ‘Competition (being able to compare you score with others, win: lose ratio etc)’ and ‘Smooth gameplay (high frame rate and reduced latency). This means that for my UNO project, I must focus on ensuring each user has a score that remains on their profile and potentially allows users to compare scores with their friends as well as making sure the game runs smoothly on python, which may mean implementing a time.sleep module in python and the clock module from pygame to make my blocks of code run in a more sequential manner.



- ‘too many rules’ – this means that when I add a ‘help’ function for users I will make sure it’s not too forced so those who are experienced in gaming or already know the rules of UNO can quickly avoid it and get straight into the game

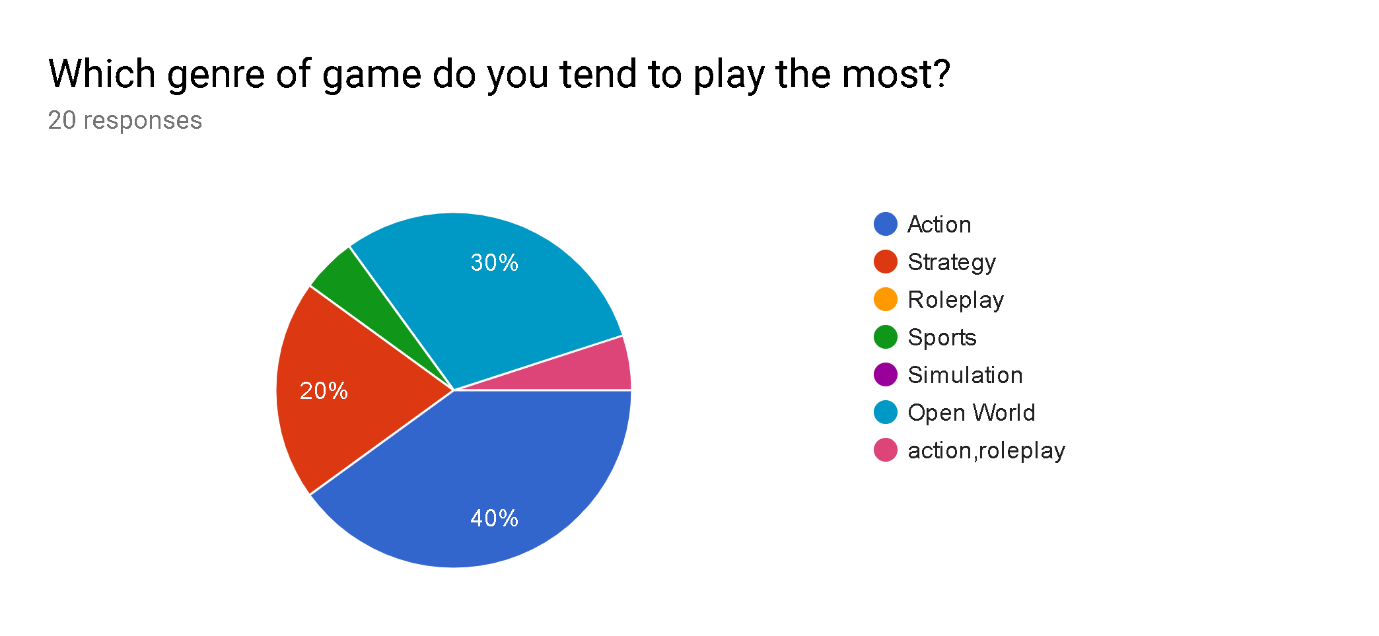
- ‘turn based, top down, being like Dota gameplay wise’ – this is good because UNO is already a turn-based game, but UNO is not a top down game like Dota

- ‘games that don’t have an in-game text chat or voice chat’ – this isn’t a priority for my project however if given more time on the project I could implement a ‘say’ function where players can type ‘Say’ following by what they wish to communicate.

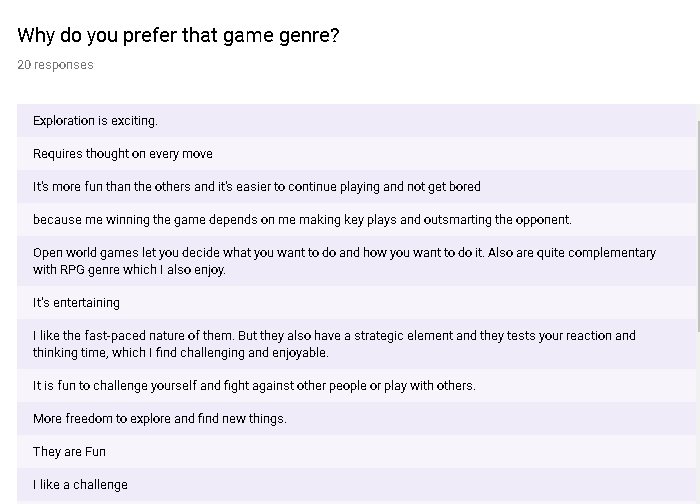
- ‘when there is a lack of choice of how to play the game and progress’ – UNO is a very versatile game in that there are no set ways to play it, if the user clears all their cards first the method they choose to do so is up to the user. However, there is a lack of progress in UNO, I could attempt to implement an advancement to the basic scoring system, in which a certain score equates to a milestone and with each milestone users unlock more features, such as card themes etc.

- ‘frame drops and lag’ – my game will likely not have any lags or frame drops as It will be running on python which is optimised for most windows devices.

- ‘pay-to-win’ – My rendition of UNO will not include any pay to win aspects.

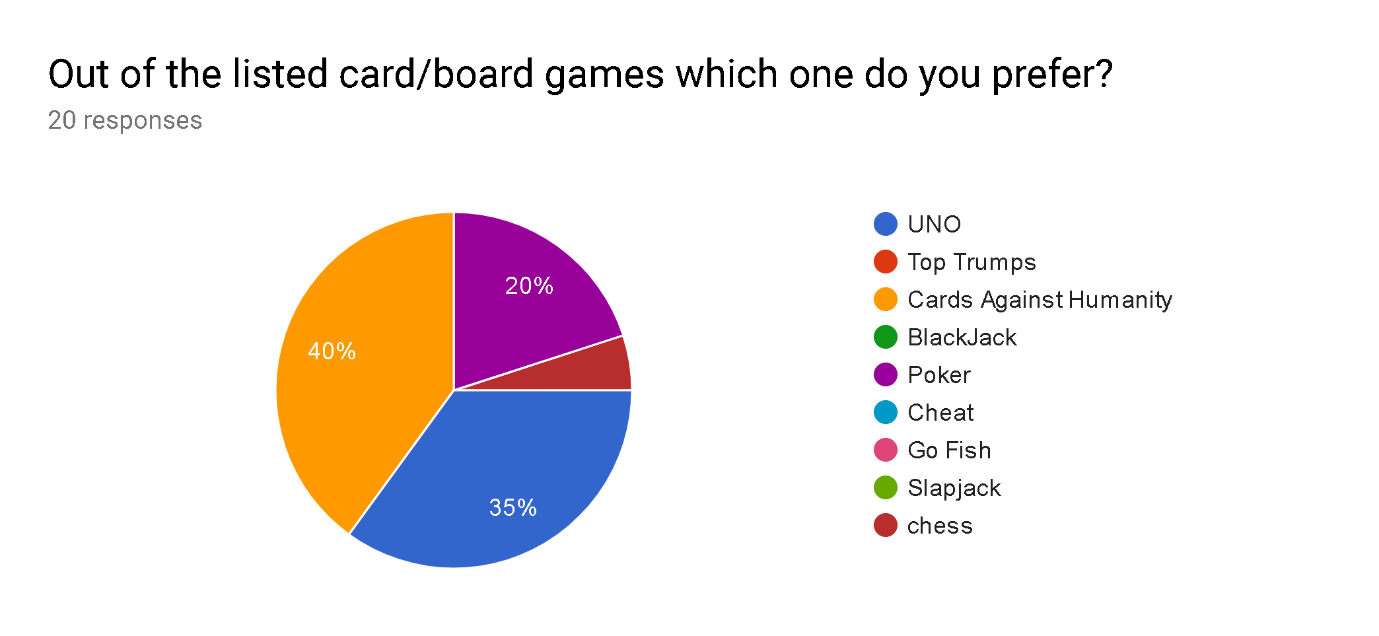


The most popular choices were ‘Action’, ‘Strategy’ and ‘Open world’. While ‘open world’ doesn’t apply to my application at all. I will aim to make my version of UNO fast paced to give users a sense of action and users will be able to discover and implement their own winning strategy

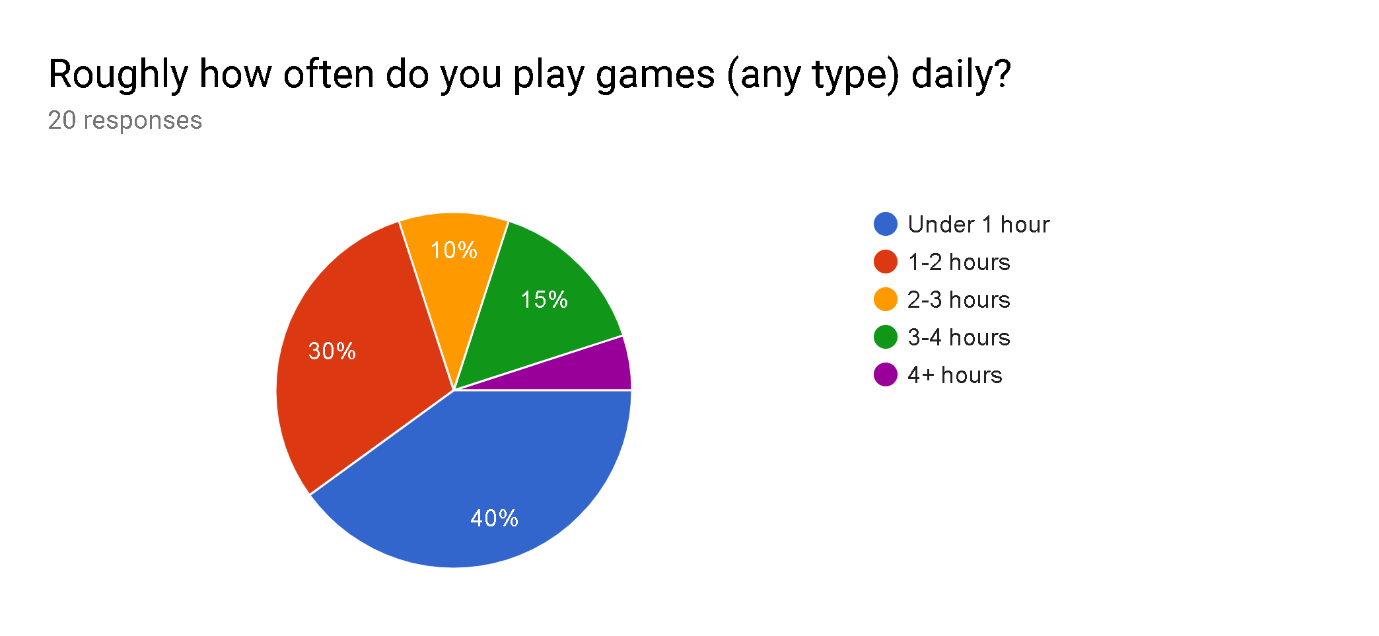


- ‘requires thought on every move’ & ‘because me winning the game depends on me making key plays and outsmarting the opponent’ – this is true with UNO as you have to think about every move to enable to get rid of all your cards first

‘it is fun to challenge yourself and fight against other people or play with others’ – this means I will have to focus on making sure the AI I create for single player mode is challenging enough for the user to stay engaged during gameplay, furthermore a multiplayer option against other users (whether on the same computer or across a network) is key to my project’s success



UNO is the second most popular card behind cards of humanity this means that my rendition of UNO has the potential to become popular with a lot of users as the game is already well established and liked.



A significant number of my potential clients only played games from ‘under an hour’ up to ‘2 hours’ per day. This means that most of my users may not have that much experience in gaming which means that a ‘help’ screen with a list of instructions may be necessary, so all users understand how to play UNO. However, this low average of gameplay per day may be due to the fact that it is school term time so gameplay overall will be reduced due to school priorities, because of this fact it will be better if the games are fast-paced and short so users don’t spend too long gaming and have enough time to allocate to other daily activities.

**1.7 Proposed Solution & Objectives**

As this project is to be done over several months below, based on responses from my questionnaire and my own research I have made a rough outline of the objectives I’ve set for myself to complete over the duration of my NEA, this will form my proposed solution to creating a rendition of UNO using python and pygame.

Programming aspect (strongly based around MVC):

**MODEL + CONTROLLER**

Module – Singleplayer

-register account (SQL)

-enter username, email and password

-log in (SQL)

-retrieve user data and display profile (e.g name, date account was created, wins and losses, highest score)

-load **AI**

**START GAME**

Module – Multiplayer

-log in (SQL)

-retrieve user data and display profile (e.g name, date account was created, wins and losses, highest score)

-join or create a lobby

-wait till every player in lobby is ‘ready’ if all players ready, start game

**START GAME**

Module – **start game**

-initialise deck

-distribute cards to players

-when card clicked card placed on pile and update model (event management in pygame)

-if player places card on pile that doesn’t match colour or number, card is returned to player hand and they draw a card

-draw a card from pile

-for every correctly placed card score is incremented by 10

-for every incorrectly placed card/draw from pile, score is decreased by 5

-if player(s) has one card left, press enter to call uno, if uno not called draw card from pile + decrease score by 5

-if player has no cards left, they win. Score +100

-if score is above user highscore, score updated on their profile

-user pressed Q to leave game

Module – **AI**

-define constant rules for ai to follow

-implement Monte Carlo to calculate values and create a tree based on these values that the AI will traverse to win

**VIEW (PYGAME)**

Image list

-individual uno cards

-player icons

Modules/functions

-addscreen function

* I will attempt to add a login system that allows the user to create an account
* Each user account will store total time played, total games played, total points and a win to lose ratio
* There will be a ranking system against other registered users that corresponds to a user’s score and will allow them to compare their ranking against friends.
* When my project is completed, I will test it amongst my family and peers to collect a range of responses with varying ages as a factor to determine the general success of my project as a fully functioning game application.
* For my project multiplayer is quite an important aspect however due to time constraints It may not too possible to implement this fully as I will have to pick up a lot of new coding skills (such as learning how sockets work). If I am unable to make an online multiplayer function, I will limit multiplayer to one computer, meaning users will have to physically take turns or control their avatar using different sections of a keyboard (e.g. one user playing on arrow keys and the other on WASD).
* I will implement an AI using a Monte Carlo method, which based on a list of available moves I create in UNO, I will give each move a value based on each situation and create a tree (a compilation of moves) for each common situation e.g., if a +2 is placed on the pile, the AI will place a +2 or +4 to avoid having to draw 2 more cards, this move will be set to the highest value as it is the best move in that situation. (Prototypes will be included in documentation after further research).
* Create a help function that gives the user a run down of the Uno game and how it works but I will make sure this function is only called if the user needs it to not hinder experienced players
* For the view aspect of my project I will be focusing on using pygame to create screens such as the start up screen, lobby screen and main game screen. I will import default avatar images for each player added to the lobby. I may also add a profile screen, in which the user can view theirs and other players wins, loses and high score as well as total games played (scores will also be used in creating a ranking system for players to add to the competitive nature of my game)
* I will implement an MVC to link my pygame view and python model and controller together

**1.8 OOP Paradigm Prototype v1**

Since UNO is a very complicated game to code, I have decided to format my code in OOP, so it is easier for me to step through my code when I encounter an error and easier for my NEA Supervisor to analyse my code if necessary. I have started off by creating the fundamentals of an UNO game in which my code allows the programmer to create a deck, shuffle the deck, draw cards from the deck and initialise players. I have also created an SQL database in which users can create an account or log into an existing account. Currently I am attempting to connect the SQL database to class Player: so, once the user logs in, they are initialised as a player when they start playing UNO, I will include this in my documented design.

Below is my deck of cards code in OOP:

import random

class Card:

def \_\_init\_\_(self, suit, number):

self.suit = suit

self.number = number

def \_\_unicode\_\_(self):

return self.showcard()

def \_\_eq\_\_(self, other):

if self.suit == other.suit and self.number == other.number:

return True

return False

def \_\_repr\_\_(self):

return self.showcard()

def showcard(self):

if self.number == 10:

number = "Reverse"

elif self.number == 11:

number = "Skip"

elif self.number == 12:

number = "+2"

else:

number = self.number

print("{} {}".format(self.suit, number))

class Deck:

def \_\_init\_\_(self):

self.cards = []

self.builddeck()

def builddeck(self):

for s in ["Yellow","Red","Blue","Green"]:

for n in range(0,13):

self.cards.append(Card(s, n))

def showdeck(self):

for card in self.cards:

card.showcard()

def shuffle(self):

for i in range(len(self.cards) - 1, 0, -1):

random.shuffle(self.cards)

def drawcard(self):

return self.cards.pop()

def discard(self):

return self.hand.pop()

class Player:

def \_\_init\_\_(self, u\_name\_l):

self.hand = []

self.pile = []

self.name = u\_name\_l

def draw(self, deck, x):

for i in range(x):

self.hand.append(deck.drawcard())

#return self #allows player draw multiple cards

def showhand(self):

print("\nYour hand is")

for card in self.hand:

card.showcard()

def showpile (self):

print("'\nCards on the pile are")

for card in self.pile:

card.showcard()

def discard(self): #remove random card - could be used in 'AI'

discard\_card = input("What card do you want to discard? ").split(' ')

suit = discard\_card[0]

if discard\_card[1] == 'Reverse':

discard\_card[1] = 10

number = int(discard\_card[1])

discard\_card = Card(suit, number)

elif discard\_card[1] == 'Skip':

discard\_card[1] = 11

number = int(discard\_card[1])

discard\_card = Card(suit, number)

elif discard\_card[1] == '+2':

discard\_card[1] = 12

number = int(discard\_card[1])

discard\_card = Card

if discard\_card in self.hand:

self.hand.remove(discard\_card)

self.pile.append(discard\_card)

deck = Deck()

#deck.shuffle()

Player1 = Player('David')

#Player1.draw(deck, 5)

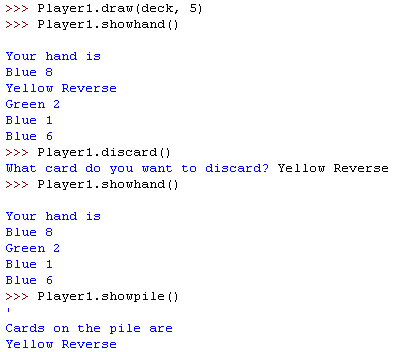
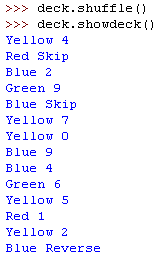
#Player1.showhand()

#Player1.discard()

#Player1.showhand()

#Player1.showpile()

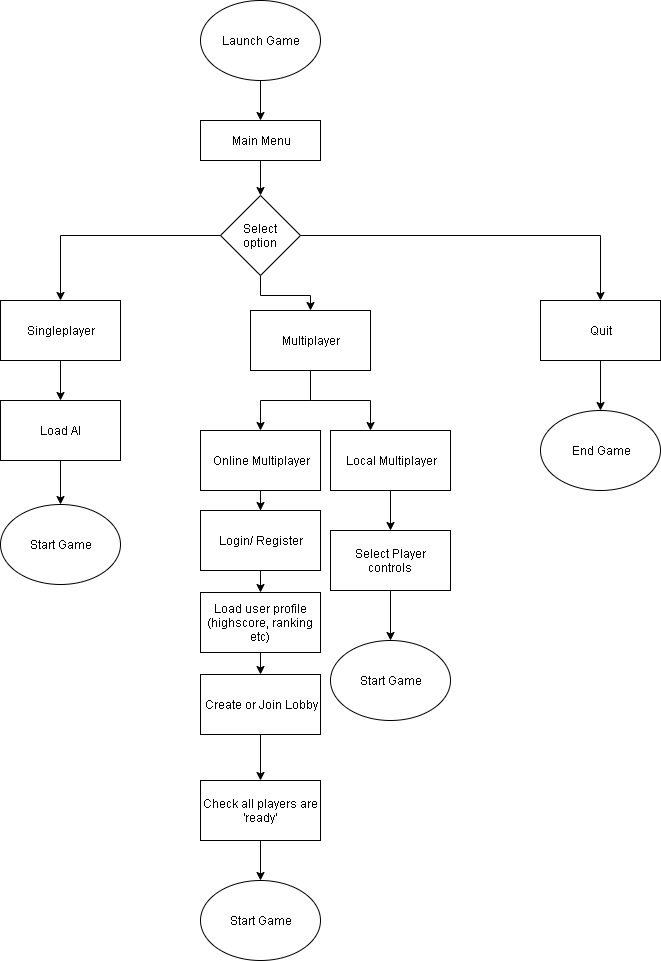
Below is the output of my code, I tested all the necessary functions to ensure it works correctly. So far, my code allows a created player to X number of cards from the main deck (which can be shuffled or unshuffled), see what cards are in their hand, discard a card in their hand onto a pile. However, I still must create an addplayer function which takes user that can logged into an SQL database and add them to the lobby/game. I also have to make the pile accessible to all players before I can continue with making the rest of my UNO game.

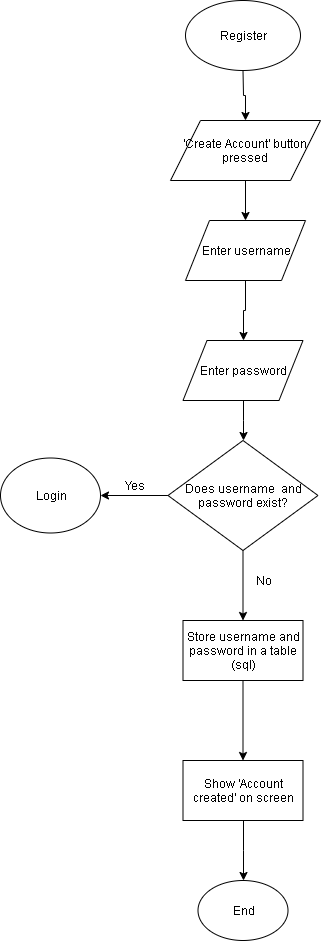


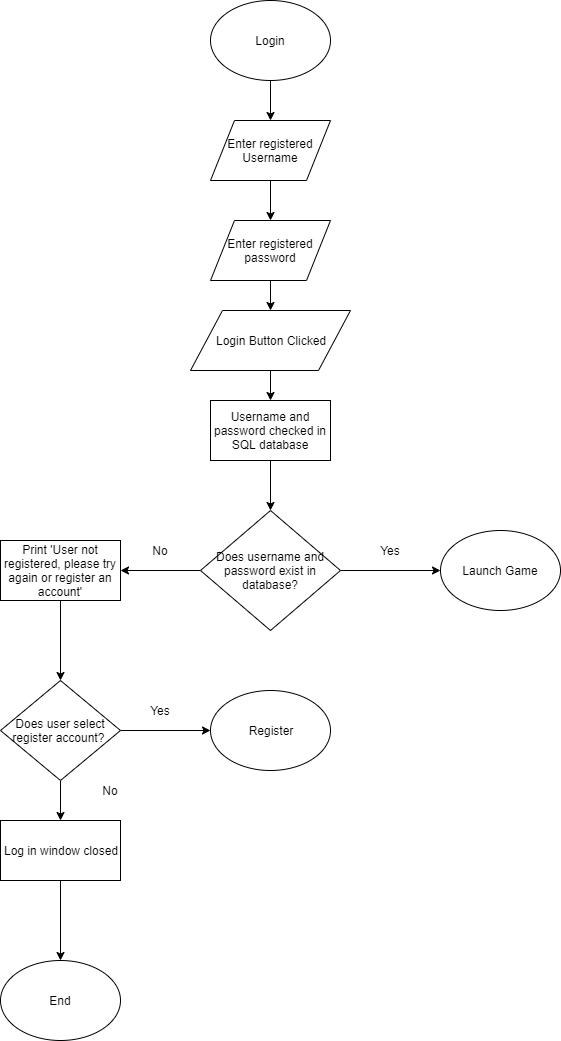
**Documented Design**

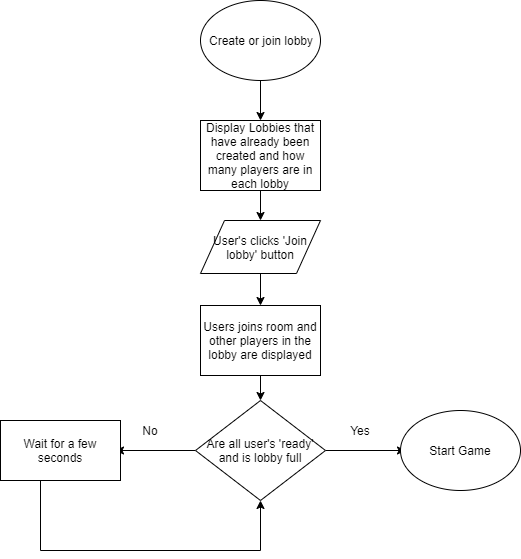
**2.1 Diagrams & Flowcharts**

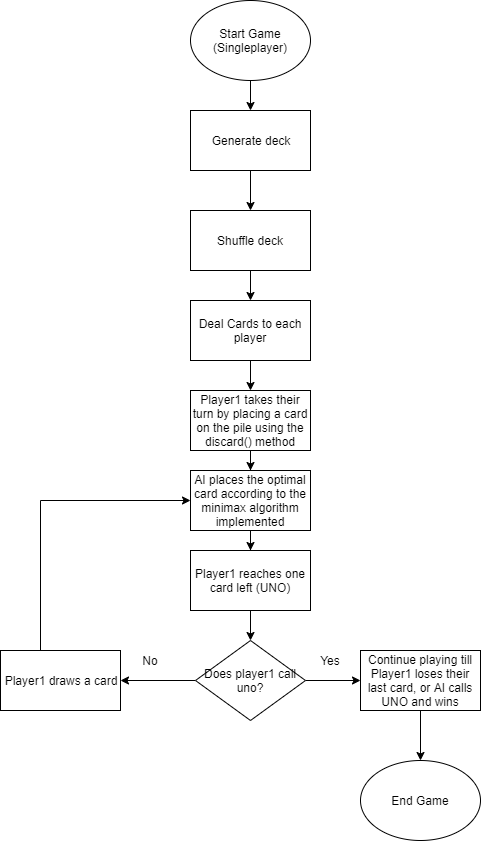
Below are various flowcharts that depict how I plan to code each significant stage of project.

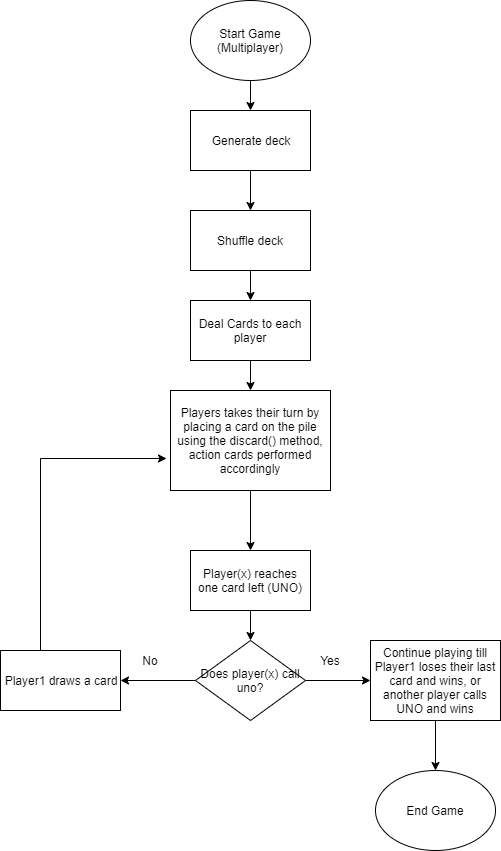




1. ****

   ****

   ****

   ****

   **Load ai diagram pending**

   **List of References**

   Below is an ordered list of citation that link to where I retrieved any image or information used in this analysis.

   <https://sea.mashable.com/entertainment/2138/its-confirmed-you-can-indeed-end-an-uno-game-with-an-action-card> [↑](#endnote-ref-2)
2. <https://steamdb.info/app/470220/graphs/> [↑](#endnote-ref-3)
3. <https://store.steampowered.com/app/470220/UNO/> [↑](#endnote-ref-4)
4. <https://store.steampowered.com/app/470220/UNO/> [↑](#endnote-ref-5)
5. <https://tesera.ru/images/items/855783/f00fc229-565a-400e-b9cf-9a9152ffd067.pdf> [↑](#endnote-ref-6)
6. <https://tesera.ru/images/items/855783/f00fc229-565a-400e-b9cf-9a9152ffd067.pdf> [↑](#endnote-ref-7)
7. <https://en.wikipedia.org/wiki/Craits> [↑](#endnote-ref-8)
8. <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller> [↑](#endnote-ref-9)
9. <https://www.giacomodebidda.com/mvc-pattern-in-python-introduction-and-basicmodel/> [↑](#endnote-ref-10)
10. <https://www.tutorialspoint.com/artificial_intelligence_with_python/artificial_intelligence_with_python_gaming.htm> [↑](#endnote-ref-11)