# Analysis

## Description

I am going to create an advanced game of Poker involving Artificial intelligence for a single player to play against. The player will use an account with a stored amount of money allowing them to maintain their winnings over multiple sessions of playing and compare to others playing on the same system.

The game will have between three and seven AI players with each acting differently according to their circumstances and a difficulty setting. The AIs will be designed to consider many different variables when deciding how it will act in each scenario such as folding or bluffing. The AIs will be designed to all act slightly differently and the chance of them choosing certain options will be swayed by how they are generated.

The game will allow for people who enjoy Poker to play the game without suffering from any financial repercussions as the money they bet is fictional. Furthermore, people too busy for games like Poker will be able to play without having to organise an event with others and still get the experience of playing against other people.

This problem involves the use of many AIs for a person to play the game alone meaning that a computer is required to operate the other characters that the user will play against. The secure storage of the players details and score will require encryption algorithms which cannot reasonably be completed by a person.

## Research

### Graphical user interface, application Description automatically generatedPrimary research

To help with my specification I sent a questionnaire to some of my potential end users. Here are the results:

Which Games Have Been Played:

* 70% of users had played Poker
* 50% of users had played Blackjack
* 10% of users had played War
* 20% of users had played Eights

Favourite Card games:

* 50% said Blackjack was their favourite
* 30% said Poker was their favourite
* 10% said Eights was their favourite
* 10% said solitaire was their favourite

Single Choice Questions:

60% of users wanted a high score list, 30% were unsure and 10% didn’t.

40% of users wanted points to transfer between games, 30% were unsure and 30% didn’t.

20% wanted a tutorial, 60% wanted a rule list, 20% wanted neither.

Other Suggestions:

Card table backdrop and multiplayer.

A card table background can be included by lining up the cards in each game to a specific background image that is present during game.

Multiplayer would require a server or other method of connecting the players together if playing online which is too difficult for me to do at this time. Local multiplayer would be possible but quite difficult and would make some games very slow and boring so won’t be included.

### Secondary Research

As my project will contain many different card games, I decided to research each card game separately to gain a clear understanding of the various versions of each.

For *War*, I found a version on cardgames.io. This uses a very simple and vibrant graphics style with bright greens and blues for the background with whites and yellows for interactive areas. This contrast creates a childish look and so I have decided that I will likely avoid using a design based on this.

This game is a simple game and only needs the two players to be able to draw cards from a shuffled deck. This achieves this very easily with a large “deal” button which I will also make use of as it would be simple to implement and would be fully functional for this game.

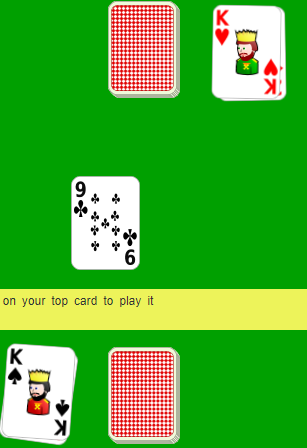
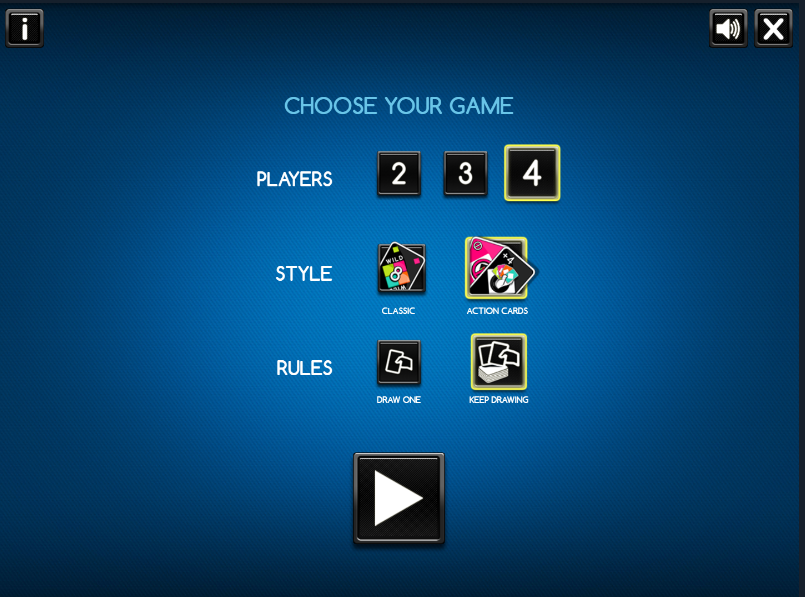
The game also has several buttons at the top of the screen that allow the player to restart the game, read the rules and change the rules. This design will be helpful to allow my version of the game to save or leave the game as well as reading rules or restarting, so I will use a similar system to this in my games.

Figure :Simple example of “War”

The card style that the example used is very simplistic and works with the general style however, I don’t believe it would fit with the other games this will be alongside and so will avoid using a card style similar to the example.

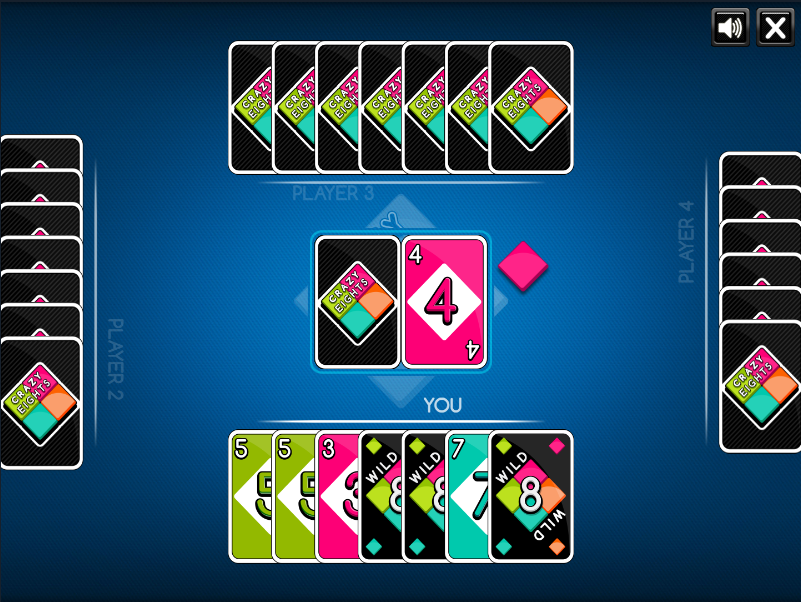
Figure 3:Cards from “War”

For *Eights*, I found a version on coolmathsgames.com. This uses a modern style of graphics with two colours dominating the screen and the rest being only accents to the black and blue. However, this graphics style seems quite complex and difficult to achieve, as the graphics would have to all be produced just for this. While this could be achieved, I believe that there are alternative methods that could provide a similar result.

This game is quite complex and has several different available rule sets, as such multiple options must be given. This version uses the options of player number (AIs), the style of the cards and the method used to draw cards. These options would be simple to implement as well as providing a decent variety of experiences, and so will be used.

Figure 4:Menu of "Crazy Eights"

*Eights* is also played as a multiplayer game and to create this effect in my version, I will have to create AIs to play against the user. This will pose a challenge due to the many considerations to be made for each turn of play.

In the game you can select your cards and the two buttons in the top-right corner of the screen. I believe that this is insufficient for my game as you will have to be able to leave and save your progress at any time in the game. However, the mute button could be useful and so I will make use of that. Also, in *tkinter* it will be very hard to turn each card into a functional button mid-way through the game and so I will have to find an alternative method of selecting them.

The card style of this game is based on colours and any special effects that they may have on the game. This is not helpful as I intend to use the same deck of cards for all the games and this style will be incompatible with some of the others.

Figure 5:game of "Crazy Eights"

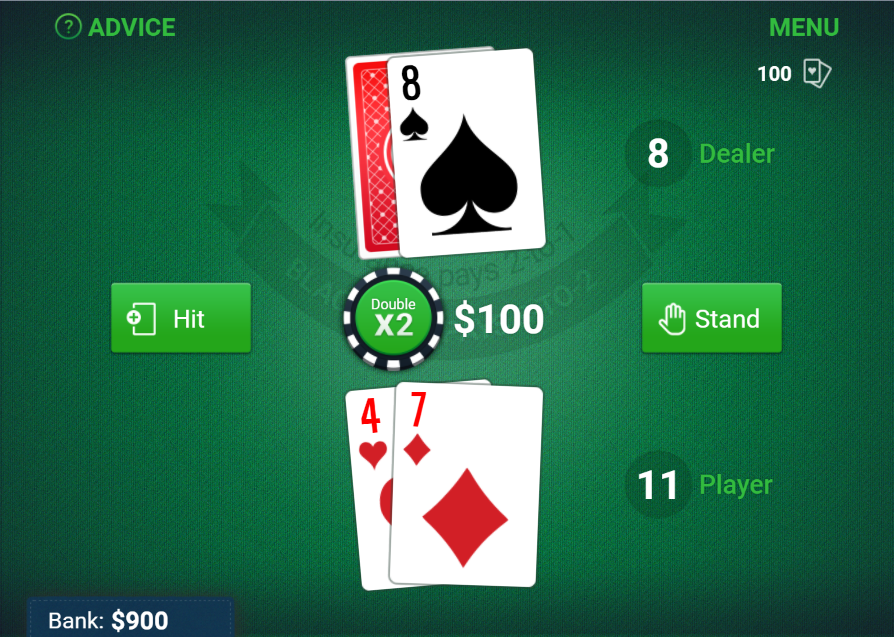
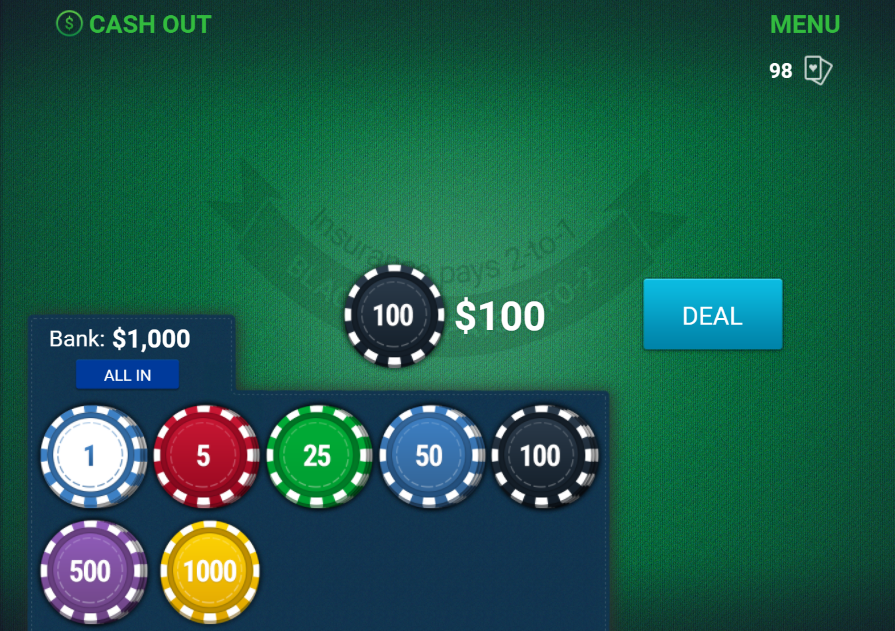
For *Blackjack*, I found a version on The Washington Post. This uses a more casino style of graphics with the background being based of off an actual casino table. This style would work for the games I intend to create; however, it may be too specific and could have variations for each game.

Figure 6:game of "Blackjack"

This game mechanics such as proper betting as well as an opposing player, the dealer. In the example, the bets range from $1 to $1000 which I think will be too high for the game I will producing, especially as its score may be combined with other games that the user plays.

*Blackjack* AIs are usually made by simply aiming for a hand score of seventeen which will make creating a dealer somewhat easier than in the other games.

Figure 5:Betting of "Blackjack"

In the game there are five available buttons: hit, stand, menu, double and advice. The menu and advice will not be included as they would be obsolete with the system I am planning to use of many buttons above the game. Stand and hit act as the methods the user must play the game by drawing or stopping and so these will be vital. Double allows the user to double their bet on for this round if they are feeling confident, but I will not be including this as it will become too easy to gain money in this game.

The card style of this game is a classic deck with the suit making up most of the card face and a red back when concealed from the player. This card style is appropriate for all the games involved as it is very easily recognisable. Furthermore, as my stakeholders are generally older, it is possible for the user to have poor eyesight and so making the information on the card very clear and large could help deal with this potential problem.



For *Poker* I found a version of arkadium.com. This uses a casino graphics style with everything happening within the table. This is simple enough that it could be used for any of the games I am making so I will create variants for each of the games using different colours.

*Poker* will be very complex to produce due to the number of AIs and number of hand combinations that must be constantly considered. Also, *Poker* requires a minimum of three players to play the game meaning good AIs will be extremely important for the game to run smoothly.

Figure 8: game of "Poker"

Figure 7:early game of "Poker"

In this game there are five buttons available to the user: Fold, check/call, bet, all-in and menu (top-left). Once again, the menu button will be ignored as my design will incorporate its functions into the UI regardless. The remaining buttons are vital for the user to play the game as they require the ability to change their bets or leave the play after a bet has been changed.

The card style of this game is almost the same as the one from the *blackjack* game but with slight adjustments to the sizes and contrasts of the images on the card face.

## Success Criteria

|  |  |  |
| --- | --- | --- |
| Reference | Requirement | Justification |
|  | Progress saving | Players can save the |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. Allow user to save progress
   1. Save name and scores
   2. Save progress in ongoing games
   3. Stored by accounts
2. Securely save all data
3. Must have multiple advanced AIs
   1. Different play styles
   2. Different styles at difficulty levels
4. General buttons for all games
   1. Sound, save, options, help, quit
   2. Displayed above game screen
5. Buttons for *poker*
   1. Check/call, fold, bet buttons
   2. Displayed near cards, bottom of screen

## Requirements

### Hardware Requirements

|  |  |
| --- | --- |
| Item | Reason |
| Mouse | The games will be controlled by the mouse, and it will be required to use the menus. |
| Keyboard | The log on system will require the user to type a username and password. |
| Monitor | The display will need to be displayed on the user’s screen for the game to be run. |
| 2GHz processor or better | This easily fulfils the minimum for python. Ursina, the engine I am using, does not have a list of system requirements so I am just using the speed of my processor as I have confirmed that this can run the engine. |
| 2GB RAM | Ursina is somewhat light in terms of code and so will not require large amounts of memory to allow it to run properly so this amount is just based of off running python normally on a computer. |
| 20MB of storage | The largest section of my project will be the graphics required for the cards, backgrounds and buttons meaning not much space will be required for the game to be stored properly. |

### Software Requirements

|  |  |
| --- | --- |
| Item | Reason |
| Windows or Linux | The game is designed for desktop computers and the Ursina engine cannot be run on MacOS. |
| Python 3.9 or newer | The game will be made in python and so will require python installed to run the game. |
| Ursina | The game will be made using Ursina as its engine so it will be needed for the game to run. |

## Limitations

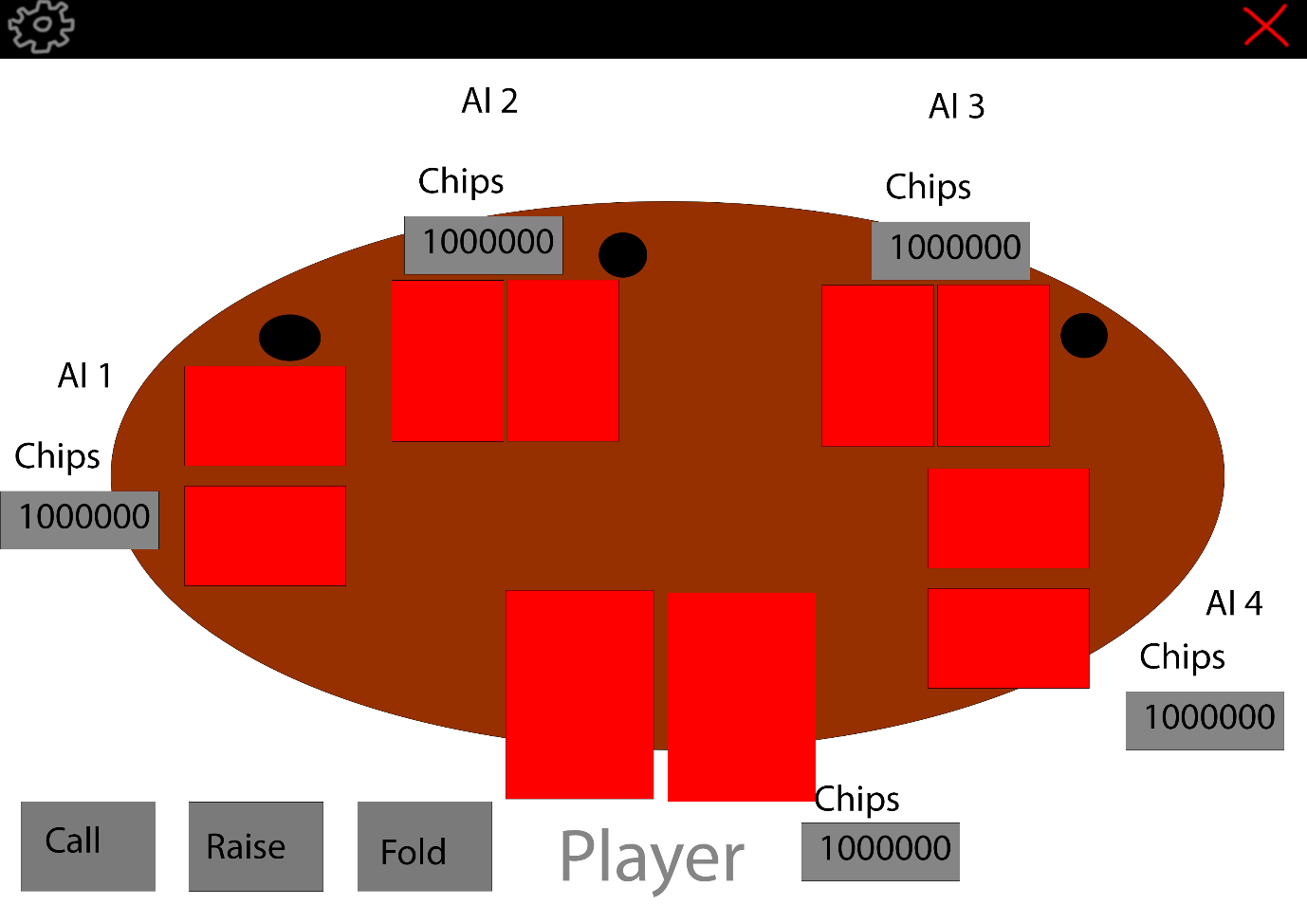
## Features

1. **User accounts** are a major feature of my program and will allow the user to store their progress in their games as well as retaining and comparing scores.
   1. Users should be able to select their username from a dropdown to allow them to log in faster.
   2. Creating a new user account should be easy and fast as to not diminish from the user’s time in the game.
2. **Game Settings** once in the game the user should be able to select any settings required for that game
   1. The user should be able to load any saved games of that type with no problems carrying on from where they left off.
   2. The user should be able to choose from multiple graphical and gameplay related settings.
3. **Debug Mode** should be able to select users to allow them to test features of the game as to find problems with the game.
   1. Debuggers should be able to modify scores and saves without having to play the game.
   2. This will only be used for testing purposes and would be considered cheating so shouldn’t be shown on score boards.
4. **Score count** should be displayed to the user from the main menu showing how many points they currently have in their save.
   1. This will be the same count used on score boards when comparing to other users.

# Design

## Screen Design

When the player opens the game, they will be asked to either log into an existing account or to register a new account. Each option will offer them with a username option and a password. Once both have been inputted they can press the relevant button which will access a file that stores that data. In order to ensure security, the passwords will be hashed before being stored and when logging in the password will be hashed before being checked so that the stored password is never in an insecure form.

Once in the game the player will be presented with a table with each of the players around the table. Their chip counter will be displayed near each of the players as well as their name and if they are out of the game. In the corner they will have buttons that allow them to make actions in the game such as raising the current bet. Black counters will be used to display who is the current Dealer and who are the current Blinds. In both top corners will be buttons that let the player close the game easily and access the options like difficulty.

## Classes

|  |  |
| --- | --- |
| Class name | Card |
| Inheritance | None |
| Methods | make\_face – checks if the would be a face card and defines it. |
| Attributes | value - integer  suit - string  face - string |
| Justification | This class will contain the value, suit and face of the card. As the game will have 52 cards in use at a time using a class will prevent me from using a list for each one using a large amount of memory. |
| Validation | Card’s attributes must remain the same types. suit must be one of the parts of the string Suits. face must only be defined when the cards value is a face card. |

|  |  |
| --- | --- |
| Class name | Deck |
| Inheritance | None |
| Methods | shuffle – shuffles the deck during its creation  draw – returns the first item in the deck and removes it from the deck |
| Attributes | cards - list  no\_cards - integer  suits - string |
| Justification | This class will contain and control the cards that the game will use. The game will take cards by using the draw method which ensures that only a single card is given. |
| Validation | The Deck will recreate the list of cards if it has drawn every card so it can’t try to draw more cards than exist. |

|  |  |
| --- | --- |
| Class name | My\_Button |
| Inheritance | Ursina.Button |
| Methods | on\_click – redefined from inheritance, allows the button to react to being clicked on by activating a function. |
| Attributes | text – string, what the button will say over itself  color – Ursina.color, what the default colour of the button will be  texture – string linking to image file, image that the button displays as its background  highlight\_color – Ursina.color, colour of the button when the mouse is hovering over the button.  pressed\_color – Ursina.color, colour of the button when clicked on.  position – tuple, x, y and z coordinates of the button  scale – tuple, size of the button |
| Justification | Allows for me to create a version of the Ursina Button class with default values which every button I make will contain such as its look and general size. |
| Validation | The position and scale as tuples mean’s that they won’t move or change once the button is made. Texture can only be an image that I have placed into a specific folder near the code meaning it cant be changed into many options. |

|  |  |
| --- | --- |
| Class name |  |
| Inheritance |  |
| Methods |  |
| Attributes |  |
| Justification |  |
| Validation |  |

## System Diagram

