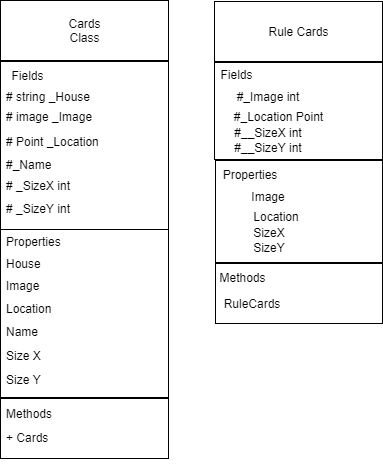
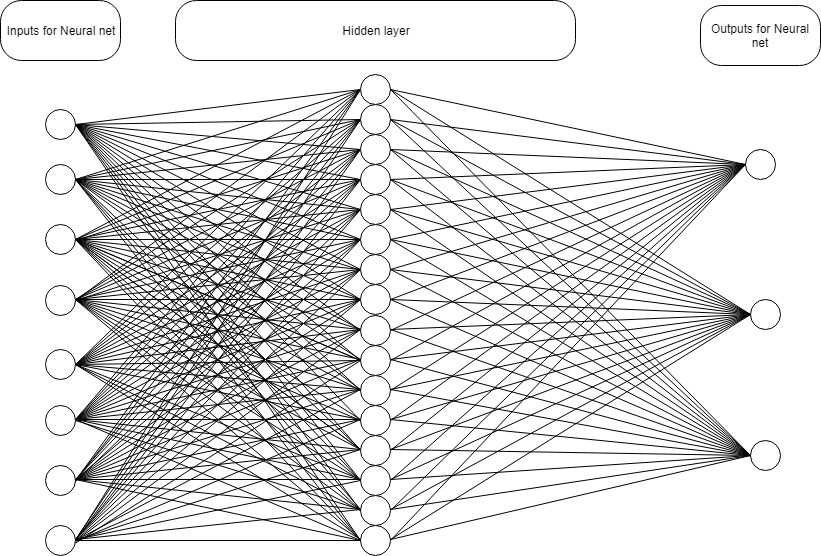
Design

In this section I will be documenting the design process of my Project, This will allow me the opportunity to show my preliminary designs and how the program will work overall, This allows me to gage how my program should be designed for maximum efficiency towards the program and how the user interface can be used so that everyone will be able to play no matter their prior experience level and skill of this classic card game. This documentation will allow me to justify what my project will be able to do and the way in which I have completed the task in doing so. This section’s purpose is to explaining all of the processes of the program and how it shall function according to the smart objectives, which I have made in the analysis section. Below explains in detail what I am hoping to achieve and how I shall go about doing this, with various tables and Diagrams showing what my program will be able to do as well as feed back from outside third-party organisations in which may be the end users and clients in which this project was designed for.

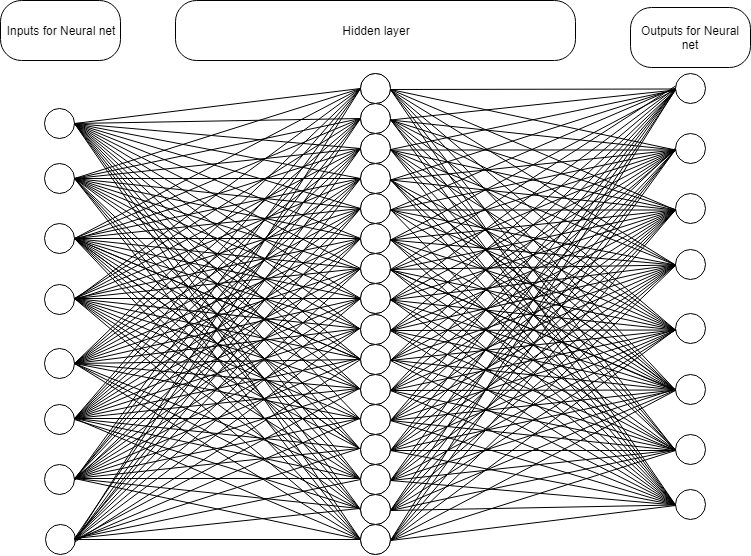
OOP Design (For Cards but not for the Neural Network Class)



**Neural Network Design 1**

****

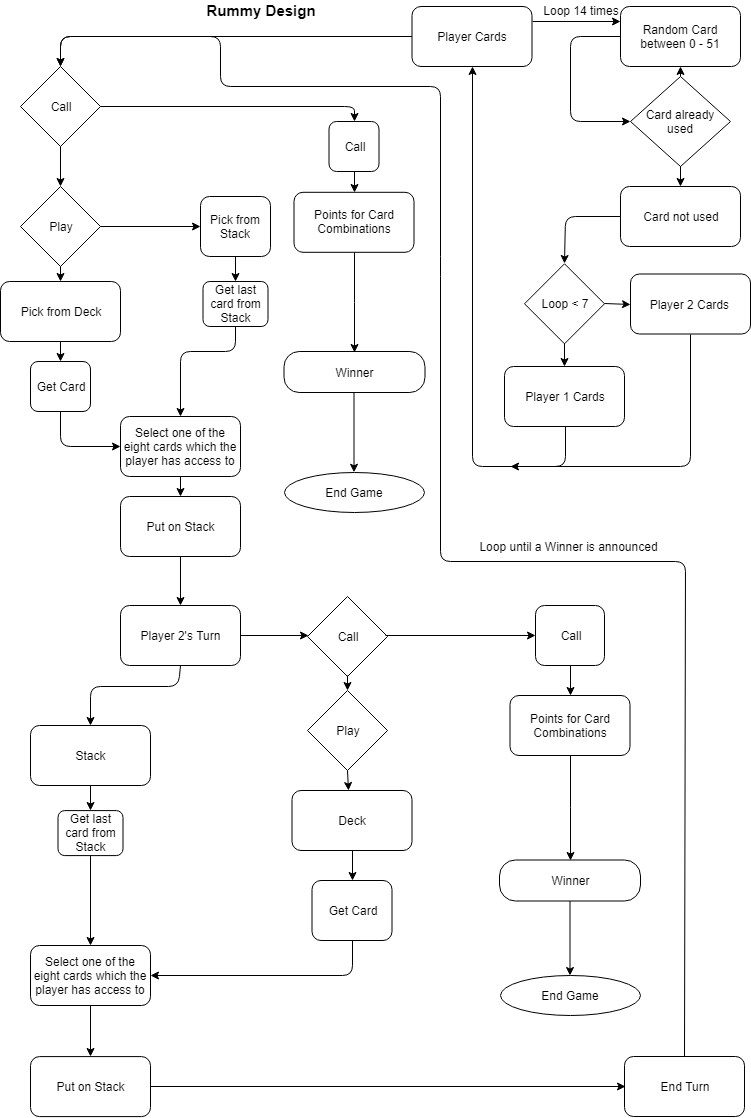
**Neural Network Design 2**

****

**ISPO chart**

|  |  |  |
| --- | --- | --- |
| **IPSO** | **Program Section** | **Item** |
| **Input** | Declare Cards | Deck Of Cards (DOC) elements declared which the name and House of the card in question, along with the image added to the elements of the array.  DOC[i].Name  DOC[i].House  DOC[i].Image |
| DecPlayerCards | The players get given 14 random numbers between 0 and 51, these are connected to the numbers of the element in the DOC array. There are 7 card numbers for each player.  P1No[i]  P2No[i] |
| Player Cards | The card numbers from the DecPlayerCards are fed through the DOC array to give the players hands  P1Dis[i].Name  P1Dis[i].House  P1Dis[i].Image  P1Dis[i].Loaction (X and Y)  P2Dis[i].Name  P2Dis[i].House  P2Dis[i].Image  P2Dis[i].Loaction (X and Y) |
| Declare Deck | The Card numbers in the players hands are filtered out of a new list for the Deck which Originally starts with 52 cards and removes the players card numbers until the number becomes 38.  Deck[j].Remove (P1No[i])  Deck[j].Remove (P2No[i]) |
| Deck Cards | Declares the Deck Location  DeckDis[i].Location (X and Y) |
| Stack Cards | Makes the last card of the Deck equal to the stacks First card. Then Sets the stacks location.  Stack[i].Add(Deck[Deck.Count])  StackDis[i].Add(DeckDis[Deck.Count])  StackDis[i].Location (X and Y) |
| Combo Start | Selects the players starting Cards and limbo card and displays them in a combo box |
| **Processing** | Combo Decision | Selects the players Cards and the limbo card and Displays then in a combo box |
| Check Deck | This checks to see if the New card selected from the deck is actually in the deck and if it is not been taken out and used un the stack. |
| Call | This selects both players cards, then using the rules of rummy allocates the points to each player with valid card scoring combinations |
| Pick from Deck | This allocated a random card from the available deck and adds it to the limbo card of the current players turn. |
| Pick From stack | This allocates the last card which was put on the stack list stack. For the player to use as a limbo card. |
| Puck On Stack | This uses the combo box for the selected card the card moves from either the limbo card or one card from the players hand and moves to the stack, if the card is one of the players hand cards then the limbo card will then move to the spot of card which was added to the stack. |
| **Storage** | NNStorage | This will store the number indicating the biases and weights of each neuron and each connection |
| **Output** | Call | This allocates points to the players for the specific card combination; The points then added to the player’s total points. The player with the highest points wins and a display message will appear displaying the number of points as well as the number of turns it took to complete the game. |

**Flow Chart**

****

Data Dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| Data Item | Data type | Validation | Sample Data |
| DOC | Cards[51] |  | “Ace”, “Hearts”, Properties.Resources.ACEHEARTS,(100,100) |
| RNG | Random |  | 20 |
| P1Cards | Cards[7] |  | “Ace”, “Hearts”, Properties.Resources.ACEHEARTS,(80,250) |
| P2Cards | Cards[7] |  | “Ace”, “Hearts”, Properties.Resources.ACEHEARTS,(80,20) |
| Back | Cards |  | “”,””,Properties.Resources.BackCard,(120,130) |
| LimboCard | Card |  | “Ace”, “Hearts”, Properties.Resources.ACEHEARTS,(400,250) |
| DeckDisplay | List<Cards> |  | “Ace”, “Hearts”, Properties.Resources.ACEHEARTS,(120,130) |
| StackDisplay | List<Cards> | If StackDisplay.count = 38 Then  For int i = 0; I < 38;i++;  StackDisplay.RemoveAt(i);  StackNo.RemoveAt(i); | “Ace”, “Hearts”, Properties.Resources.ACEHEARTS,(200,130) |
| NameHouse | String[8] |  | “Ace Hearts” |
| P1No | Int[7] |  | 2 |
| P2No | Int[7] |  | 50 |
| Deck | List<int> |  | 49 |
| Stack | List<int> | Same as Stack Display | 27 |
| PriorCardName | string | If Current CardName == Prior Card Name Then  PickFromDeck = False; | “Two Hearts” |
| Limbo | int |  | 0 |
| NumberOfMoves | int |  | 30 |
| P1Points | Float |  | 6f |
| P2Points | Float |  | 10f |
| PickFromDeck | bool |  | False |
| DeckValidation | bool |  | True |
|  |  |  |  |

**Research**

I have been in contact with different companies which ass specialise with gaming and software development. This includes companies such as Rummycircle who are famous as being an online rummy game where you the user can either play against others online of against the computer which I suspect is a form of neural network. This is what I am hoping to make to function as player 2 in the game.

I have also conducted various interviews with various people who have an interest in Rummy. I have shown the preliminary designs to many people asking weather the interface is user friendly and if the interface is effective in minimalizing the options of the user. As my research and questionnaires from the analysis section have shown that many people would like to see a more minimalistic setting, when shown pictures of various online rummy games. They suggested that the games interface has more buttons and various different items in order to seem professional, however this (from my research anyway) has had the opposite effect making the designing seem crammed and this makes the user less able to concentrate on the game rather that the mass of buttons surrounding the display. This in my and other opinions takes away from the immersive effect of what the game is meant to be, thus is this situation less is more. This is shown by the questionnaire below.

**Algorithm Design** (Pseudo Code and Structured English)

**Cards Class Methods**

\_SizeY ← 60;

\_SizeX ← 85;

\_Name ← Name;

\_House ← House;

\_Location ← new point (120,130);

This has changes the size of the image of each card to fit the size and design; it also declares the \_Name to Name, which is the variable that changes the name of the array for the DOC mentioned later. This is the same for house. The location id declared to be the base point which if the deck point; this makes the change that I do not have to change the location for the deck cards.

**Rule Cards Class Methods**

\_SizeY ← 200;

\_SizeX ← 200;

This declares the size of the rule cards. This is used in the showing the rule pictures and teaching players how to play.

**Declare Cards**

Sub DeclareCards(Cards[] DeckofCards)

For I ← 0 to 52

If I < 13 Then

DeckofCards[i].House ← “Clubs”;

Else if (I >= 13 AND I < 26) Then

DeckOfCards[i].House ← “Hearts”;

Else if (I >= 26 AND I < 39) Then

DeckOfCards[i].House ← “Diamonds”;

Else if (I >= 39 AND I < 52) Then

DeckOfCards[i].House ← “Spades”;

End If

Int MODNo ← I Mod 13;

If ModNo ← 0 Then

DeckOfCards[i].Name ← “Ace”;

Else If ModNo ← 1 Then

DeckOfCards[i].Name ← “Two”;

Else If ModNo ← 2 Then

DeckOfCards[i].Name ← “Three”;

Else If ModNo ← 3 Then

DeckOfCards[i].Name ← “Four”;

Else If ModNo ← 4 Then

DeckOfCards[i].Name ← “Five”;

Else If ModNo ← 5 Then

DeckOfCards[i].Name ← “Six”;

Else If ModNo ← 6 Then

DeckOfCards[i].Name ← “Seven”;

Else If ModNo ← 7 Then

DeckOfCards[i].Name ← “Eight”;

Else If ModNo ← 8 Then

DeckOfCards[i].Name ← “Nine”;

Else If ModNo ← 9 Then

DeckOfCards[i].Name ← “Ten”;

Else If ModNo ← 10 Then

DeckOfCards[i].Name ← “Jack”;

Else If ModNo ← 11 Then

DeckOfCards[i].Name ← “Queen”;

Else If ModNo ← 12 Then

DeckOfCards[i].Name ← “King”;

End If

Next I

End For

* Iterates for the 52 elements of the array and gives them the images of the different 52 cards.

End Sub

**Justification:**

This subroutine declares the cards name, house and image. Through many different trials of defining the cards, this has been the most effective and efficient in terms of the number of lines the subroutine will be. At the end of it there is some structured English, this section defines the image of each card.

**Declare Player Cards**

Sub DecPlayerCards(int[] P1No, int[] P2No)

List[] CardsAvailable ← new List[n];

Int PlayerCard;

Random RNG ← new Random();

For i ← 0 to 52

CardsAvailable.Add(i);

Next i

End For

For I ← 0 to 7

PlayerCard ← RNG(CardsAvailabel.count);

Player1No ← CardsAvailable[PlayerCard];

CardsAvailable.RemoveAt(PlayerCard);

Next i

End For

For I ← 0 to 7

PlayerCard ← RNG(CardsAvailabel.count);

Player2No ← CardsAvailable[PlayerCard];

CardsAvailable.RemoveAt(PlayerCard);

Next i

End For

End Sub

**Justification:**

This separates the players 7 starting cards numbers from the deck; it also splits and moves to cards reference number to each player’s hand. This then validates the previous lines of code by removing them from the deck’s numbers so that the cards numbers cannot be duplicated meaning that the player has a cards which are unique and cannot be picked from the deck or in the other players hand.

**Player Cards**

Players Cards (Cards[] DOC, Cards[] P1Cards, int[] P1No, int[] P2No, Cards[] P2Cards)

For i = 0 to 2

For CD = 0 to 8

If I = 0 Then

P1Cards[CD] ← DOC[P1No[CD]]

P1Cards[CD].Location ←(30+ (CD \* 50), 250)

Next i

Else if

If I = 1 Then

P2Cards[CD] ← DOC[P2No[CD]]

P2Cards[CD].Location ← (30+ (CD \* 50), 20)

End If

End For

End For

End Sub

**Justification:**

This moves the data from the previous subroutine and uses the data to give the players hand the location, and the card details of the DOC, which is the base array, which defines the cards house, mage and name. This is the most efficient way which I could code this section.

**Declare Deck**

Sub DeclareDeck (List<int> Deck, ref int[] P1No, ref int[] P2No)

Int[] Player12Combined ← new int[14];

For I ← 0 to 14

If I <7 Then

Player12Combined[i] = P1No[i];

Else if I >= 7 Then

Player12Combined[i] = P2No[i- 7];

End If

Next I

End For

For I ← 0 to 52

Deck.Add(I);

Next I

End For

For I ← 0 to 14

For J ←0 to 52

If Player12Combined[i] ← J Then

Deck.Remove(J);

End If

Next J

End For

Next I

End For

End Sub

**Justification:**

This section makes the deck between the two players. This validates the cards and makes sure that the cards, which once added to the deck, will only be the cards, which are not currently used.

**Deck Cards**

Sub DeckCards (Cards[] DOC, List<int> Deck, List<Cards> DeckDis)

For I ← 0 to Deck.Count

DeckDis.Add(Deck[I]);

Next I

End For

End Sub

**Justification:**

This section adds the decks card numbers to the list of deck display using the number as a reference to the DOC. This is efficient as it uses previous variables to use as reference to add the cards to the deck.

**Stack Cards**

Sub StackCards (List<int> Stack, List<Cards> StackDis, List<Cards> DeckDis, List<int> Deck, Cards[] DOC, Random RNG)

Int RNGCount ← 0;

Int CurrentStack ← 0;

RNGCount ← RNG.Next(Deck.Count);

CurrentStack ← Deck[RNGCount];

Stack.Add(CurrentStack);

Deck.Add(CurrentStack);

StackDis.Add(DOC[CurrentStack]);

DeckDis.Remove(DOC[CurrentStack]);

StackDis.ToArray();

For I ← 0 to StackDis.Count

StackDis[i].Location ← new Point(200,130);

Next I

End For

StackDis.ToList();

End Sub

**Justification:**

This subroutine adds the cards used to the current stack, this also defined the location for the deck for the cards. This subroutine also uses a random number from the deck and at the start adds it to the stack. When the player takes the card from the deck, the card is removed from both deck lists. This was done to be able to track where any current card is at on specific time; it also is used to stop the program from using the same card multiple times.

**Check Deck**

Sub CheckDeck ()

For I ← 0 to Stack.Count

Do

If Deck[ElementLim ← Stack[I]] Then

ElementLim ← RNG.Next(Deck.Count);

CheckDeck(Stack, Deck, ElementLim);

End If

While Deck[ElementLim] ← Stack[I]]

Next I

End For

End Sub

**Justification:**

The check deck subroutine is designed to be a recursive subroutine as if the random number used from the deck has already been used and is in the stack, then the subroutine will pick another number and validate it. This is because if the number and card has been added to the stack then to validate the number of cards to then shuffle the deck, the deck has to have all 38 opening cards. If the stack has 38 cards then the subroutine then shuffles the cards and the cards and then moved from the stack apart from the last card added.

**Call**

Sub ScoringRules

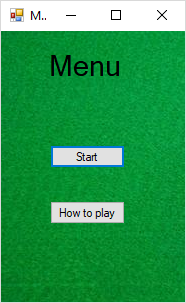
* The Program iterates through all of the Player1’s Cards
* If 3 or 4 of the Players Hand in a Sequence (Consecutive order)
* Then Player1Points Equals Player1 Points plus the Collective points of the three or four cards.
* If 3 or 4 of the of the players hand is then same number or face of the card Then
* Player1Points Equals Player1Pointsplus the other three or four cards, selected.
* Cards then added to an array for future detection of the cards if they are used in future scoring.
* The Program iterates through all of the Player2’s Cards
* If 3 or 4 of the Players Hand in a Sequence (Consecutive order)
* Then Player1Points Equals Player2 Points plus the Collective points of the three or four cards.
* If 3 or 4 of the of the players hand is then same number or face of the card Then
* Player2Points Equals Player1Pointsplus the other three or four cards, selected.
* Cards then added to an array for future detection of the cards if they are used in future scoring.

**Justification:**

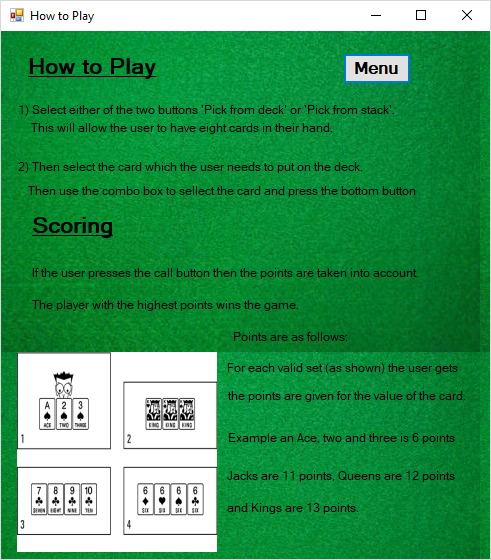
I used structured English for this subroutine as the subroutine is around 3/400 lines line. The point of this subroutine is to iterate through both players cards and to calculate based on the contents of the array what points are allocated to which player. This shown above. This is the most efficient and effective way for the player to accurately present by the rules of the game. This subroutine is arguably the most important subroutine in this game as it is responsible for the points allowing a winner of the game. This iterates through both players cards finding either 3 or 4 cards which have a connection to each other, such as having a three card run (which is three cards which follow each other consecutively).

**Interface Designs**

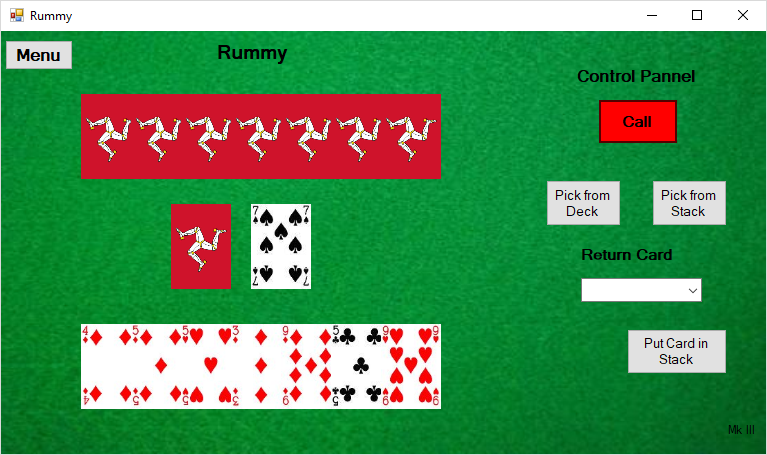
**Main Menu**



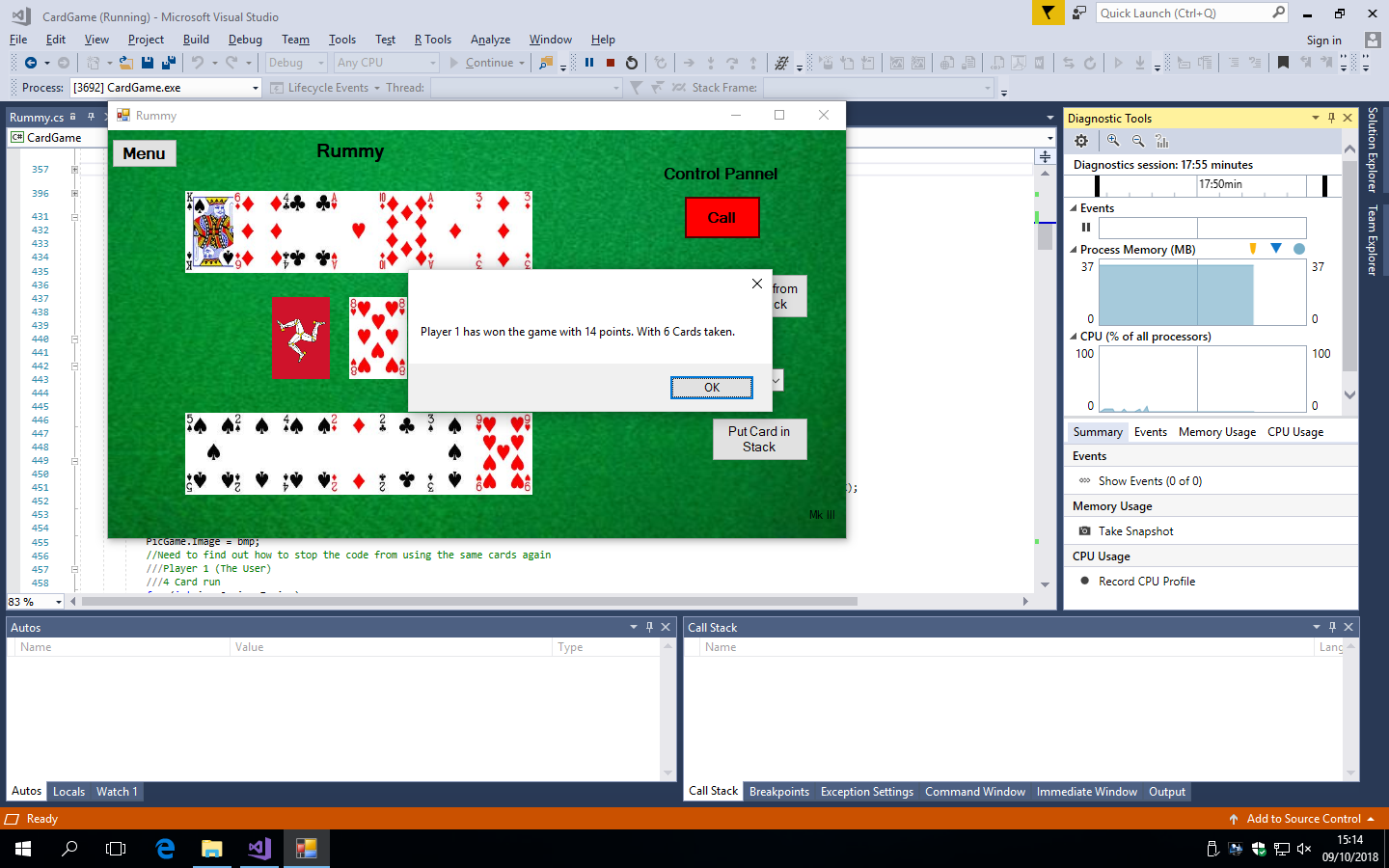
How To Play



Game Page



Pressing the Call Button to view the final scores, this after confirming the message box takes the user to the main menu page.



Questionnaire

The results show:

1. From the participants who filled out the questionnaire that some people (36%) have played rummy and or continuing to play rummy in their spare and free time. Whilst others (32%) of the participants only sometimes play rummy in their spare time. However as a result the other (32% of the) participants do not currently play rummy in there free or spare time.
2. A majority of the participants (79.2%) have considered playing rummy for entertainment.
3. There were many current flaws with the current system, which is used by the rummy community. Some involve the setting of the interface, whilst others suggest that the system is too slow, or the vast amounts of different types of adverts. Whilst others still complained with the AI, service suggesting that it is not trained properly for the user.
4. The image, which I inputted, was an interface of an online rummy game. The question asked the participant if the image design looked similar to the current service, which is used. A majority of the participants (66.7%) agreed with this question.
5. (Almost all participants answered this; this may show that even though some participants did not play rummy they recognised some of the flaws of the interface.) A majority of the participants (88%) thought that it was unnecessarily compact.
6. This question asked the user, if the interface were designed in a simpler format then would they consider playing rummy. Many of the participants (56%) said that they would consider playing the game, and 44% of participants said that there may have to be some alterations to the game and answered maybe.
7. The question asked the participants if there were an application of rummy on app stores would they consider downloading it. Many (52%) said that they would, however some (48%) stated that it would depend on the situation and if the changes that they wanted to see being implemented into it. Thus answering maybe.
8. The question asked ‘What changes would you like to see implemented into the new design of Rummy except the updating of the interface.’ Many wanted to be able to change the interface design and be able to change the rules to customise the experience of the game. However, an overwhelming number of the participants suggested that they would like to see the possibility on a dedicated page for statistics, either for the game or at the end of the game.
9. The question asked the participants does the prototype of the rummy interface in which I have created look user friendly. The majority of participants (96%) stated that it did in fact look user friendly and that they could navigate around the page.
10. The final question asked the participants is there anything, which I could add to improve the user interface. The responses suggest that I should change the colour of some of the buttons increasing the immersive effect of making the colours the same colour as some of the chips, which we usually associate with many different card games.

In conclusion, the questionnaire has increased my knowledge into what many potential end users want, this has told me what to look out for and how to make the interface the most efficient for the end user. This has also allowed be to make some changes to the Neural Network systems so that the end user can have the ability to change the strengths of the opposition, making it weak to almost professional at playing rummy.