TOP TRUMPS

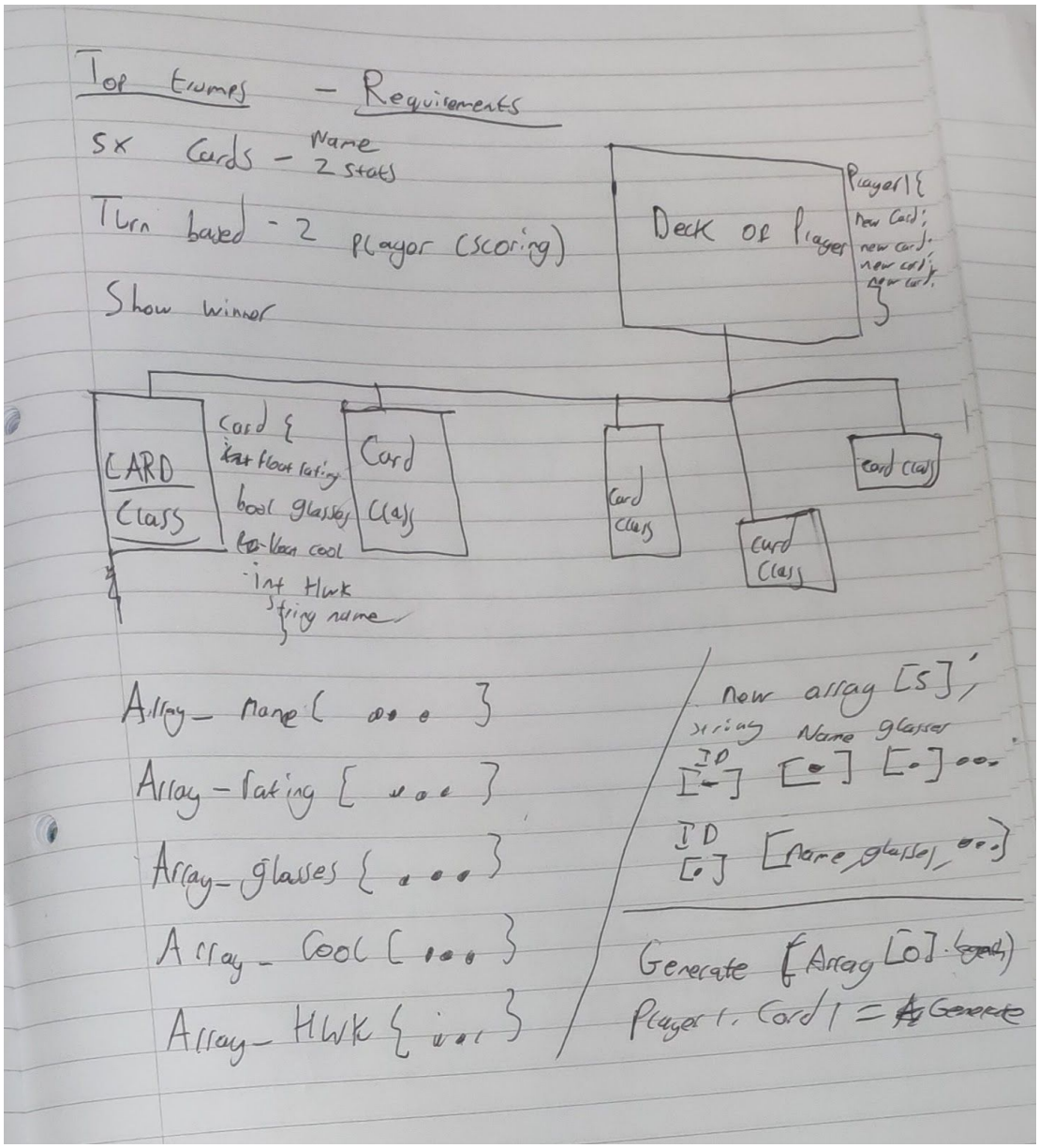
By Brijesh Savjani

Contents Page

Introduction

Top trumps is a game that involves 2 people. Each person has a hand of 5 cards. They can choose a category on a card. The value of this category is compared to the other player's value. Whichever player wins gets the other players' card. The game ends when one player has all the cards.

Planning Stages



Success Criteria

- [ ] Have a GUI
- [ ] Functioning 2 player game
- [ ] Use objects
- [ ] Card designer

Original Plan

My original plan was quite simplistic as I felt that I had a good grasp on the game and how I would go about programming it. I was planning on using the Agile Development method. My initial plan showed that the program would have a Player class. The player class would contain an array that houses all of the Cards in the Players' hand. It would also contain information like the name of the Player and the current score of the player.

I also planned to have a Card object for each of the Cards. This would contain all the categories and values of the card. I planned three different ways to create and populate the values in the cards. At this point in time I was just roughly brainstorming my ideas at this stage. The first way was to have 5 arrays (one for each category) in which the values of each card is stored so populating a card look like this:

```
//Method 1
//Declare Variables
Array Cool[5] = [5,4,3,2,5]
Array Rating[5] = [9,5,6,7,4]
Array glasses[5] = [true,false,true,true,false]
Array hwk[5] = [1,4,5,6,8]
Array name[5] = [Name1,Name2,Name3,Name4,Name5]

While i <= Cool.Legnth():
    Player.hand[i] = new Card(name[i],Cool[i],Rating[i],Glasses[i],hwk[i])

//Method 2
Array Data [5,5,5,5,5]

Data[0][0] = Name
Data[0][1] = Cool
//... Continues going like this for very value in every card
```

```
//Method3
Array Card_data[5,5]

Array[0] = “Name1”
Array[0][1] = “5,9,true,1”

Array[1] = “Name2”
Array[1][1] = “4,7,true,3”
//...Continues going on like this
```

Later Planning

However, the first thing I wanted to work on was actually the shuffling sequence as this seemed like a more challenging and time consuming thing and I felt that if I could complete this first I would be well on my way to completing the project. I planned out my Shuffling sequence with the following pseudocode using Paint:

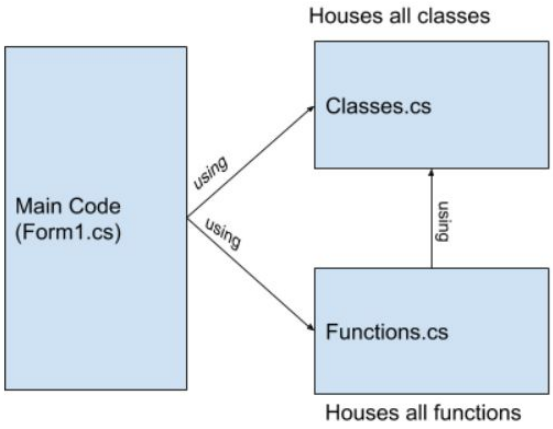
```
//MainDeck
ClassArray[] = new ClassArray;
//Players' Hand
Class[] DeckP1 = new Class[5];
Class[] DeckP1 = new Class[5];

for card in ClassArray{
  class tmp_var = card;    ClassArray.legnth()
    ClassArray[rnd.Next(0,10)] = tmp_var;
    ClassArray.remove(tmp_var];
      DeckP1[i] = tmp_var;
     i = i+1;
     }
  j i = 0;
     if (j < 5){
      DeckP1[ij] = tmp_var;
     i = i+1;
     }
```

The code firstly initialises one big array with all the cards in. And then initialises the individual Player Hands as an array. I then declared a ‘for loop’ so that it would once for every card available in the large deck. It would then choose a random card and remove it from the main deck so it can’t be added twice. It would then add the card to the individual player decks.

The Program

Structure

To help you understand my program I have made the following diagram to show you the structure.



Shuffling the cards

I mainly followed my pseudocode plan on this except I used a switch case as it looked neater. The whole thing has been put in a try,catch loop.Here is my commented code:

```
//Total Number of loops left = k
int k = Card_Deck.Count();
//Total number of cards dealt
int i = 0;
while (k >= 0)
{
   //Select Random Card
    Card tmp_var = Card_Deck[rnd.Next(0, Card_Deck.Count())];
    //Remove selected random card so it can't be picked twice
    Card_Deck.Remove(tmp_var);
    //Update amount of loops
    k = k - 1;
    //Switch case to Deal cards into player hands
    switch (i)
```

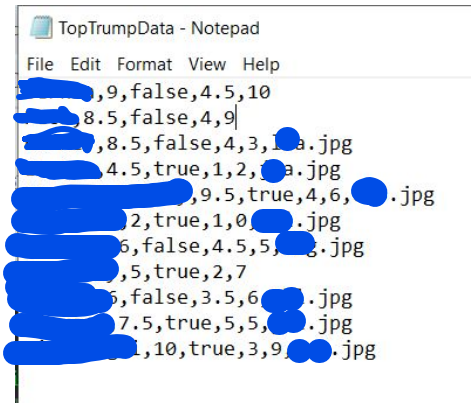
```
                    {
                        //Put in Player1 Hand
                        case < 5:
                            Player1.hand.Add(tmp_var);
                            i = i + 1;
                            break;
                        //Put in Player2 Hand
                        case < 10:
                            Player2.hand.Add(tmp_var);
                            i = i + 1;
                            break;

                    }
}
```

Creating the cards

I originally did plan to create my cards using the methods shown in the plans above. However as I had more time to ponder upon these plans, I realised that it would be better to just create my cards from an external file incase I wanted to add a card designer in the future. Furthermore, using an external file was more simple and elegant.

```
    //Open File to read from
     var Card_Data = new StreamReader("TopTrumpData.txt");
     string line;
     string img_location;
     //Creates a card for each line in document
     //Every line in file
     while((line = Card_Data.ReadLine()) != null)
     {
         //Split lines into individual properties
         var properties = line.Split(",");
         try
         {
           //Set image location
           img_location = properties[5];
         }
         catch
         {
              //catch if no image provided
```

```
                    img_location = "default.png";
                }
                 //Add card to card deck
                Card_Deck.Add(new Card(properties[0], properties[1], properties[2],
properties[3], properties[4], img_location));

            }
```



<-- File used for data

Compare the two cards

```
public static bool CompareCards(string catergory, Player Player1, Player Player2)
        {//Catergory is the category selected by the player
            float a;
            float b;
           if (catergory == "Glasses")
            {
                //Boolean → Float
                a = Glasses(Player1);
                b = Glasses(Player2);
            }
            else
            {
                //If not glasses parse value as float
                a = float.Parse(Player1.hand[0].GetProperty(catergory));
                b = float.Parse(Player2.hand[0].GetProperty(catergory));
            }
```

```
            if(a > b)
            {
                //Give Player 2s' Card to Player 1
                SwapCards("2->1", Player1, Player2);
                //Alert user to win
                MessageBox.Show("Player1");
                return true;
            }
            else if (a < b)//Other directioi.e. Player 1 gives to Player 2)
            {
                SwapCards("1->2",Player1,Player2);
                MessageBox.Show("Player2");
                return true;
            }
            else
            {//
                MoveToBack(Player1);//Move Players' card to the back
                MoveToBack(Player2);
                return false;
            }

        }
//^ If it's a draw then it Moves Cards to back and continues game
```

The compare cards function is responsible for Comparing Player 1's card and Player2's card and determining which is better. It then calls the appropriate functions for replacing the players cards. This is the longest function in length as it's function is the most complex.However, it has been decomposed into a small part already.

Below is the Glasses function it calls upon:

Glasses()

```
        private static float Glasses(Player player)
        {
            if (player.hand[0].GetProperty("Glasses") == "True")
            {//If it's true return 1
                return 1;
            }
            else
            { //Else it must be false so return 0
                return 0;
}
```

```
        }
```

Swapping Cards

```
private static bool SwapCards(string swap_direction, Player Player1,Player Player2)
       {
           //Declare temporary Variable
           Card tmp;
           switch (swap_direction)
           {
               case "1->2"://Moving from player 1 to 2
                   tmp = Player1.hand[0];//Set First card from P1 as temp
                   Player1.hand.Remove(tmp);//Remove first card
                   Player2.hand.Add(tmp);//Give to player2
                   MoveToBack(Player1);//Rotate cards
                   MoveToBack(Player2);
                   return true;
               case "2->1"://Same but otherway around
                   tmp = Player2.hand[0];
                   Player2.hand.Remove(tmp);
                   Player1.hand.Add(tmp);
                   MoveToBack(Player1);
                   MoveToBack(Player2);
                   return true;

           }
           return false;//If any errors
       }
```

This is a slightly more janky function then others. It is responsible for Swapping the cards depending on who won the previous round as judged by the CompareCards function. I did not physically plan this out but instead just thought it up when programming due to the limited timeframe of this project. Hence, the small inefficiencies in this function.It relies on the MoveToBack function to put the cards at the front of the hand to the back so the Players can't keep using the same winning card.

MoveToBack()

```
        private static void MoveToBack(Player player)
        {
            //Creates a temporary variable and sets it to first card
            Card tmp = player.hand[0];
            //Removes first card
            player.hand.Remove(tmp);
            //Adds it to the end
            player.hand.Add(tmp);

        }
```

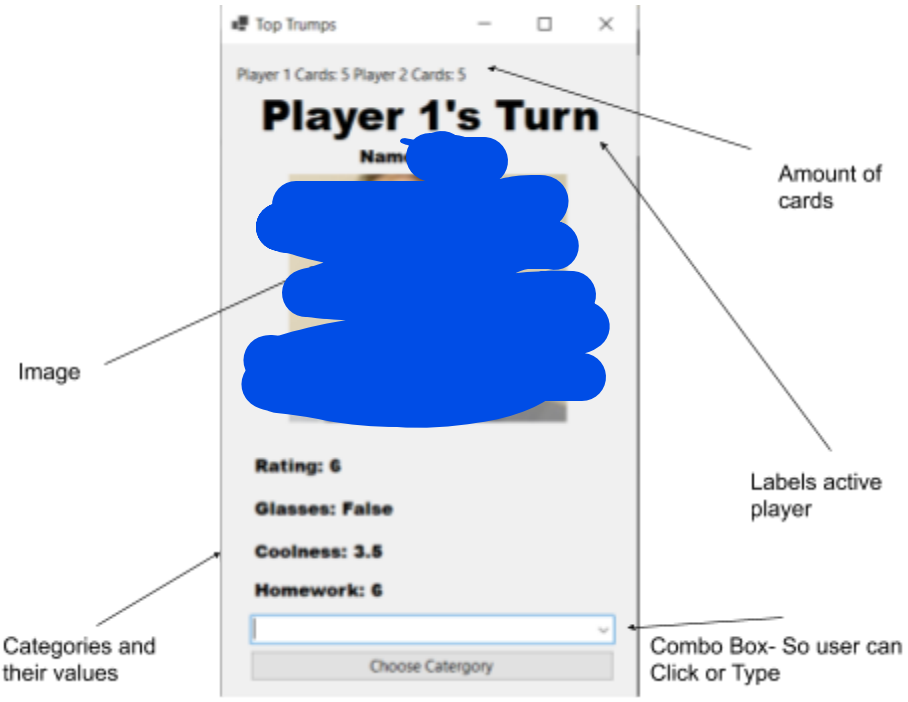
Check Win

The CheckWin functions is a very simple function that just checks if any of the users have won and if they have then it will alert the user to their victory so the game can be restarted or cancelled.

```
        public static string CheckWin(Player Player1,Player Player2)
        {
            if(Player1.hand.Count == 0)//If player 1 has won
            {
                MessageBox.Show(Player1.GetName() + " WINS!!!!!" + "with a score of: "
+Player1.Score);//Tell them
                return "Player1";
            }
            else if (Player2.hand.Count==0){//Otherwise if Player 2 has one
                MessageBox.Show(Player2.GetName() + " WINS!!!!!" + "with a score of: " +
Player2.Score);//Tell them
                return "Player2";
            }
            return null;//Otherwise carry on
        }
```

GUI Stuff & Game Logic (Form1.cs)

GUI Stuff



Writing to the labels

```
string[] values;
            if (turns%2 != 0) // If there's an odd amount of turns
            {
                label1.Text = "Player 1s' Turn";
                values = functions.LoadInformation(Player1); //Load information for P1
                Player1.Score = turns - Player2.Score; //Amount of turns played = score
            }
            else
            {
                label1.Text = "Player 2s' Turn";
                values = functions.LoadInformation(Player2);//Load information for p2
                Player2.Score = turns - Player1.Score; //Amount of turns played
            }
            pictureBox1.Load(values[5]);
```

```
            label2.Text = values[0];
            label3.Text =  values[1];
            label4.Text =  values[2];
            label5.Text =  values[3];
            label6.Text = values[4];//Display all values
            label8.Text = "Player 1 Cards: " + Player1.hand.Count() + " Player 2 Cards: " +
Player2.hand.Count();
            functions.CheckWin(Player1, Player2);//CheckWin
```

This code updates the UI to match the current card and the current player that is playing.

LoadInformation()

```
        public static string[] LoadInformation(Player player)
        {
            String[] values = new string[6];//Fetches all information as seen
            values[0] = "Name: " + player.hand[0].GetProperty("Name");
            values[1] = "Rating: " + player.hand[0].GetProperty("Rating");
            values[2] = "Glasses: " + player.hand[0].GetProperty("Glasses");
            values[3] = "Coolness: " + player.hand[0].GetProperty("Cool");
            values[4] = "Homework: " + player.hand[0].GetProperty("HWK");
            values[5]= player.hand[0].GetProperty("Image");
            return values;
        }
```

The code for Loading all the Card information is quite simple and just uses the GetProperty method that is found in the class itself. It does not need to to take in a Card index because it will always take the top card because that will always be the card in use.

Classes

```
public class Card
    {
        //Properties
        private string Name;
        private float Rating;
        private bool Glasses;
        private float Coolness;
        private int HWK;
        private string Image;
        //Constructors
```

```
        public Card(string name, string rating, string glasses, string cool, string
hwk,string image)
        {
            Name = name;
            Rating = float.Parse(rating);
            Glasses = bool.Parse(glasses);
            Coolness = float.Parse(cool);
            HWK = int.Parse(hwk);
            Image = image;
        }
        public string GetProperty(string item) //GetProperty Method
        {
            switch (item) //Switch cases
            {
                case "Name":
                    return Name;
                case "Rating":
                    return Rating.ToString();
                case "Glasses":
                    return Glasses.ToString();
                case "Cool" or "Coolness":
                    return Coolness.ToString();
                case "HWK":
                    return HWK.ToString();
                case "Homework":
                    return HWK.ToString();
                case "Image":
                    return Image;
            }
            return "error"; //No Category
        }
```

```
public class Player
        {
            public List <Card> hand = new List <Card>(); //Personal Hand
            public int Score;
            string name;
            public string GetName()//Return Name
        {
            return name;
```

```
        }
```

Evaluation

I feel all in all my project was successful in meeting most of its success criteria. I felt that parts of the program weren't as elegant as I would like but they were all entirely functional.

I feel as a whole the Program function well and made good use of Problem decomposition to break down big blocks of code into smaller more efficient bits of code with targeted functions. I also felt I made good use of algorithmic thinking in Planning stage.

I felt that the only limiting factor in my project was time and had I have had time then it would've been much better. I did not manage to complete the Card Designer because of this but everything else in success criteria was met and the base of Card Designer was built so the only thing that needed top be added was a GUI and some code to write text to the Data text file.

Furthermore, I felt that my program should have made use of a structured file type like JSON or XML. However, it didn't because parsing JSON in C# is overcomplicated. However, in the future I could change it too work off XML.