PokerMasters

I.E.S. San Vicente

San Vicente del Raspeig (Alicante)

2017/2018

Teachers:

José Ignacio Cabanes / Ignacio Iborra

Student:

César Martín Sogorb

**1. Introduction**

**Project name**

PokerMasters

**Made by**

César Martín Sogorb

**Short description of the Project**

The project is a game of Poker “Texas Hold’em” in console with multiplayer mode in which you can play in local or network. The games will be from 2 to 10 players.

**2. Functionality of the Project**

Upon accessing the program, a welcome will be displayed on the screen, a user name will be requested with which you will be identified with the other users and the different game modes:

-Play in local

-Play in network

-Game rules

-Credits

-High Scores

-Exit game

The game of poker will be explained in the rules of the game.

In the local play mode, the user can play in the same network as the adversaries, settings the local network in which the game will be played.

In network play mode, the user can choose the network by settings the network in which the game will be played.

The rules of the game will be for people who do not know how to play or who have any questions at the time of playing that will possess the following:

- Hold'em is usually played using small blind ("small blind") and big blind ("big blind") bets. These bets are called "blinds" since the player bets without having seen any of the cards on the table.

The small blind is set by the player to the left of the dealer, and is equal to half of the big blind.

The big blind is provided by the player to the left of the previous one and is equivalent to the minimum bet. In tournaments, the amount of the blinds increases as the tournament progresses.

2 cards are dealt to each player face down, your cards should not be shown to anyone else, the bets are placed on the big and small blind to whom they touch.

After the first phase, 1 card of the deck is burned and 3 cards are opened with which the players who have paid to enter this phase must form the best combination and decide whether to pass or increase if everyone passes the turn if someone goes up the bet those who want to continue to Turn must pay the amount.

If the player wishes to leave the game, the chips will be lost.

You must get the best hand of the following:



The winner ends when a player obtains the chips of all the players.

Within the game, there will be 3 options that will be raise, check / call and fold, in the option to raise will also be the amount you want to raise.

**3. Screen prototype**

The game screen will look like this:

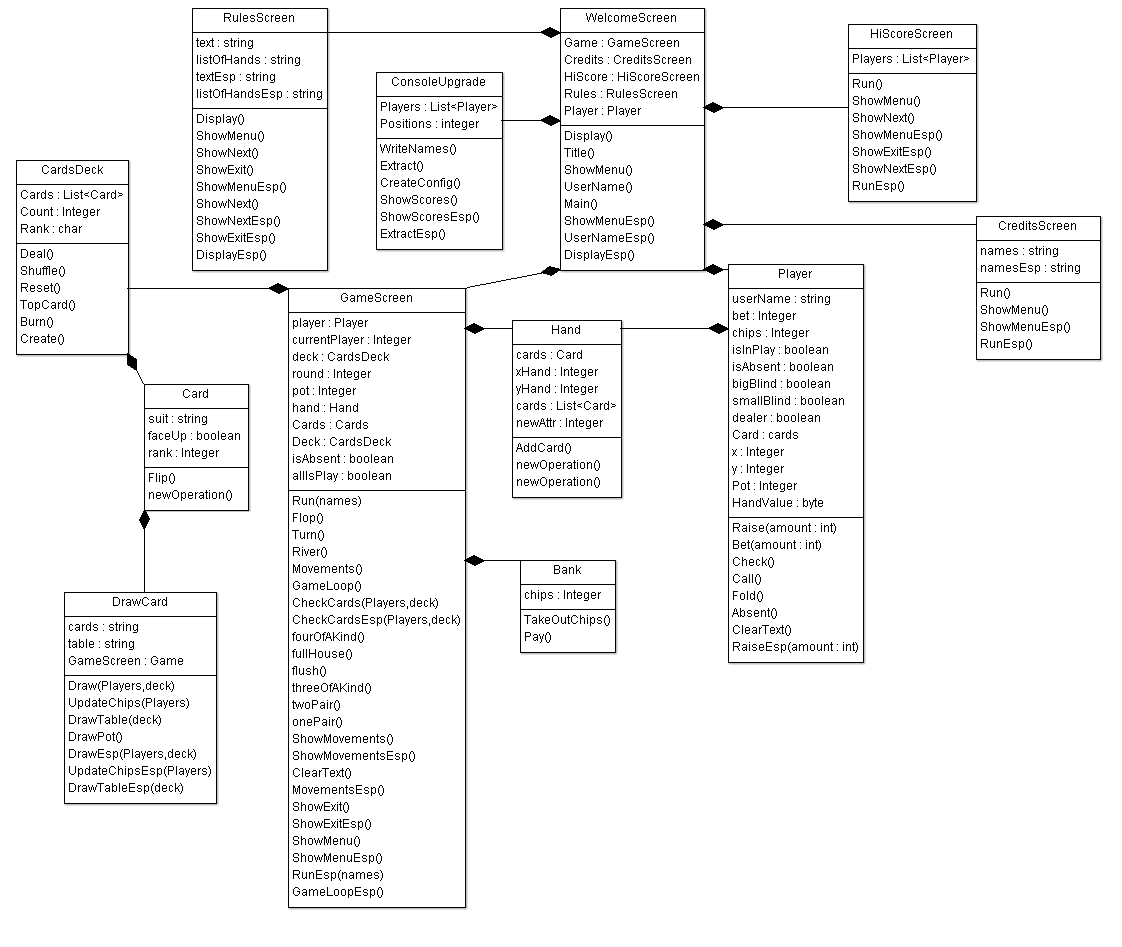


**4. Analysis**

**4a Requisites**

|  |  |
| --- | --- |
| **Requisite** | **Date achieved** |
| That the program shuffles and distributes different cards to each of the players. | Done |
| That the program allows the player to bet, place bets. | Done |
| That the program allows to play in local. | Done |
| That the program allows to play in network. |  |
| That the program performs the distribution of letters in the correct order. | Done |
| Show the actions performed in the information table. | Done |
| That the program allows the bank to perform actions with the chips. |  |
| Perform the absent mode correctly. |  |
| That the program allows a player who has lost the chips to re-engage the game. |  |
| That the visual screens work correctly. | Done |

**4c. Classes diagram**



**5. Initial planning and expected deliveries**

5a. Expected deliveries

* **Version 1:** Start screen with username input that the player will have to identify himself, where the different game modes that the player wanted will come out. And according to what you have chosen, the list of games that the player can access.

Make skeletons of someone classes.

* **Version 2:** Screen of the board, where will be the players who are in the game in each position with their username, chips and their cards.

We will create the rest of the classes so that they will be like the class diagram, we will use a list/stack for the cards that exist in the game, in the class Cards so that there will be a deck of cards where the cards will be taken and distributed to the players, the created classes will be player, deck of cards and cards with corresponding methods.

* **Version 3:** Console Upgrade. To simplify tasks, what we will do is create the Console Upgrade class, which is about positioning the users placed on the game table or introducing the player's characteristics on the welcome screen.
* **Version 4:** Implementation of the letter class with its methods and the deck class.

Implement the method to distribute, taking into account that there are 2 cards to each player, and taking into account that when all have been dealt, we will burn a card and put 3 in the middle, another will be burned and another card will be dried to the middle, the last one will be burned and the last card will be drawn.

* **Version 5:** Chips and Dealer. Make the bet, each player has some chips, when you make the bet you must remove the chips you have bet and put in the middle.

Choose Dealer that is the player with the highest card of:   
Spades > Hearts > Clubs > Diamonds

Cards distribution. The first time it is dealt knowing who the Dealer is, it is dealt blindly small, big blind, under the gun, to the dealer.

And the one who goes after the big blind speaks (The last one to speak is the big blind)

With the 3 cards in the middle (flop), the last one to speak is the Dealer.

Blind big and small blind and dealer, where the player to the left of the big blind is the first to start playing, and they are choosing each player whether to bet or not. The big blind is the minimum bet that must be made and the small blind is half the big blind.

* **Version 6:** Perform the 3 buttons to fold, call or check and raise, when the blinds have been made, the one to the left of the big blind will begin. You must choose between fold, call (in case of not being big blind) or fold, or raising the bet.

If you raise the bet you must pass another round to see if they match or not, if they do not equal they must leave the round.

We will implement a method that will increase the blinds by time of the game so that from time to time the blinds will increase.

* **Version 7:** Each player has a bet execution time that time will be 30 seconds with a beep or sound. (Can the game window be superimposed on everything the user has?) And screen notice.

We will create the method to request time that will be that the player having time to execute the play, may have extra time by pressing a button that will only be available to each player for 3 times in each game.

* **Version 8:** Logic of the cards, the best card is the AS, followed by K, Q, J, and from 10 to 2.

Know which is greater than which and in that case that wins.

Carry out a letter order:



What we will do here is create a method so that we will compare the cards that are in play when the bet is over. And according to who wins, the pot will be taken.

* **Version 9:** Logic of the cards v2, if a player makes "All in" the other players must match with the same money corresponding to the "All in" and distribute the pot among the winning parties. And the proportional part of the amount that is the "All in" will be distributed.

When you draw the 5 cards from the middle, you must show the cards of each player that keeps betting, and if you have not reached the 5 cards but you have won one before, you can decide whether to show them or not. (Know who has won or not).

For this we will implement a method that what will do is compare the cards of those who made the all in and distribute the pot among the players who have done all in. With a boolean we will allow all in or not, you can’t bet normally as in the other moves.

* **Version 10:** When a player has no chips, he can re-start if he has the minimum chips to return to the game. By eliminating from your account the total chips you have, the player will be given a number of chips for each start of the game. (User configuration .txt)

The Bank class will be created so that we can take out chips for a price that we will have to pay later.

* **Version 11:** Informative table of actions performed. So the actions performed will be written in a box in case a player has not seen the play can see who has won and what cards.
* **Version 12:** Credit and game rules screen, where the characteristics of the game of poker, how to play and the scoreboard will be shown.
* **Version 13:** Absent mode, is that if the player does not perform any execution during the 30 seconds of its necessary time, the absent mode will be assigned so that the minimum bets will be automatically made and the cards will be dealt as if they were playing.
* **Version 14:** Local mode, to be able to create a game (host), where the parameters of the game that will be created will be displayed, regardless of the number of players, starting money and creating the game. Join heading where the IP address and port where you want to connect and connect.
* **Version 15:** Create server where the game will be played and players will play this game. (Create server with my desktop computer at home and connect from class to my home computer).

It will be done by C # with the firewalls removed to prevent possible connection failures. The server will remain on so that clients will connect to the IP provided by the server.

**5b. Real deliveries**

* Version 1 (May 14): The skeletons of all the classes with their functions have been created and the welcome screen has been created where the user enters his name with which he will play and will show him the menu where he will choose the desired option and access the chosen mode.
* Version 2 (May 15): Correction when establishing the user's name.

Set the player's position on the board with his chips and his cards.

We have made the cards and the deck of cards, but it does not work correctly.

* Version 3 (May 16):The sample of the username has also been corrected when it is in game.

CardDecks and Card classes have been corrected because I have not I know how to use the enum well and I have investigated in order to fix it.

ConsoleUpgrade class has been created with the Position and WriteNames methods to position the players and write the characteristics of the player on the welcome screen. Created the structure to choose the movements

* Version 4 (May 17): Now the deck can draw the cards to the player and to the table, deleting them from the list and showing them in the screen also added the table picture.
* Version 5 (May 18): I tried to established the pot and chips with each player and we can show the pot in game table.
* Version 6 (May 21): The number of players with their respective positions has been established. An index has been established to know what number of player is playing.

The bet to deduct chips from players has been established.

* Version 7 (May 22): The number of players between 1 and 6 has been limited.

A panel has been drawn for the different options that the player has.

Fixed the pot and chips of players at the time of betting.

The screen notice about the game could not be made.

* Version 8 (May 23): The updates of the cards have been fixed at the time of the shifts.

Established to shuffle the cards has been established.

Created the order of the turns until the flop.

* Version 9 (May 24): The game loop has been established for the time of the turns, that when all the players fold they leave the game. Realized the blinds but does not bet automatically. The dealer has been eliminated since it gave problems and the big blind will be a dealer too. The update of the chips after betting has been corrected.

I have solved all the visual aspect so that the user can go through all the menus correctly.

* Version 10 (May 25): Correction to update the pot and not the table with the cards on the table. Visual form arranged with your returns to the menus, but there is a problem

when you choose a screen and you want to return to exit the game.

Established the file script config.txt with the logs. Started a list for card clashes to know who wins or who does not.

* Version 11 (May 28): Fixed the order of the shifts. Established the result of couple, double pair, trio and poker. Configuration of files established in the Console Upgrade.
* Version 12 (May 29): Screen of rules game and credits game created.

Heritage established in the classes primary class is ConsoleUpgrade.cs and the inheritance is DrawCard.cs, GameScreen.cs

* Version 13 (May 30): Established mode absent. Correction of automatic payment of blinds. Established winner for 2 players.
* Version 14 (May 31): Established the languages, english and spanish. Established the card check to know the winner, some verification is missing.
* Version 15 (Jun 1): Correction of visual errors.

**6. File Formats**

6a. Plain files format

The formats of the files will be as follows, each time a player is logged in, their data will be saved in a file called logs.txt with their username, tabs, log date and two player coordinates.

Example: cesar@20000@01/06/2018 17:30:12@0@0

Where @ is delimeter.

Once logged in the players will be saved in another file called hiScores.txt in a different way than in logs.txt and can be displayed in the high scores menu.

Example:

cesar

20000

01/06/2018 17:30:12

0

0

Where @ is replaced by line breaks.

**7. Problems found and solutions**

When you access to play in local and return to the menu does not let out correctly.

If you access the other windows, it works.

I used this method to shuffle the cards:

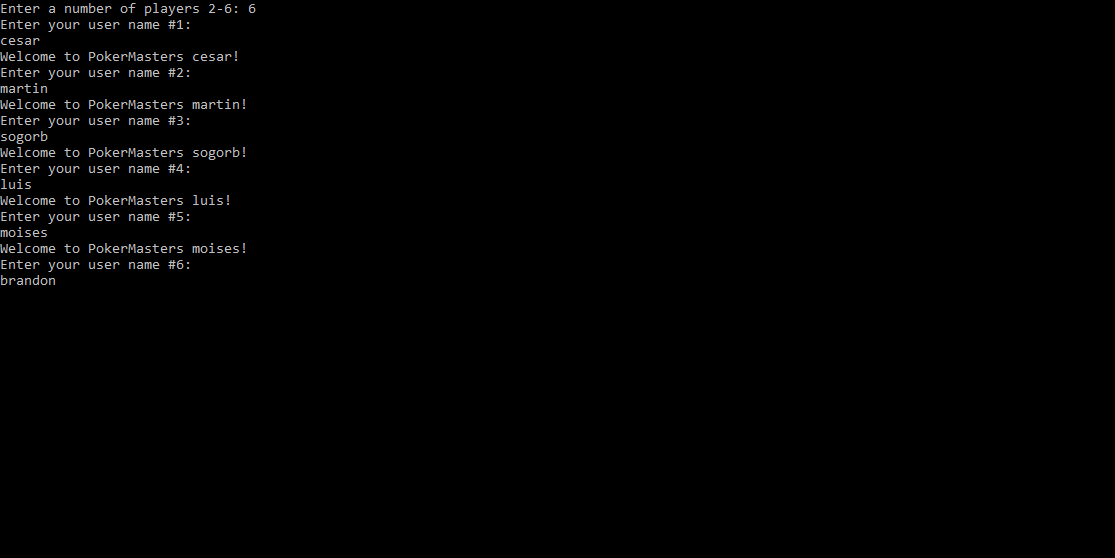
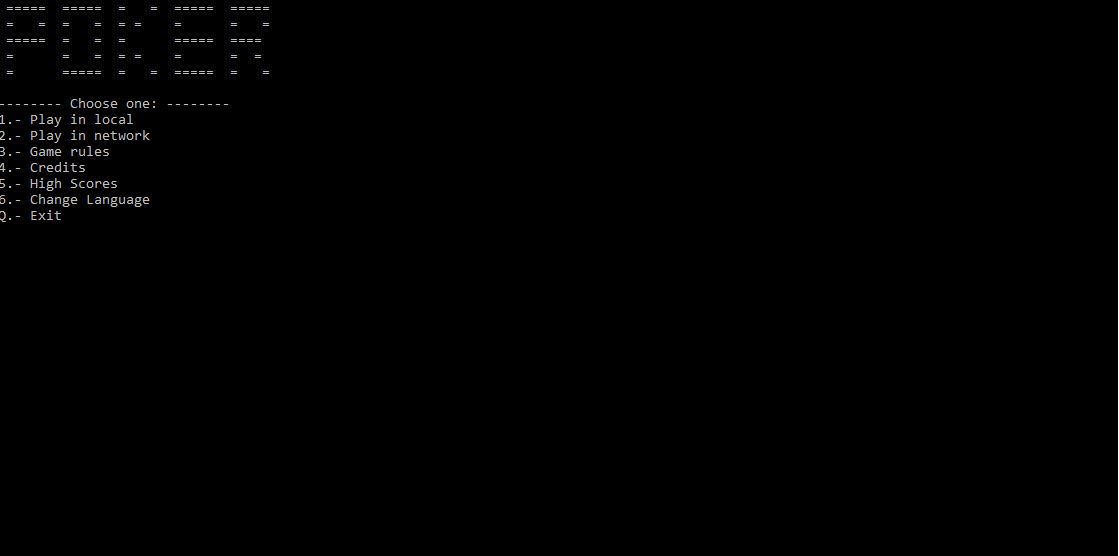
https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates\_shuffle

**8. Improvements or restrictions to the starting design**

He changed version 10 to 9.

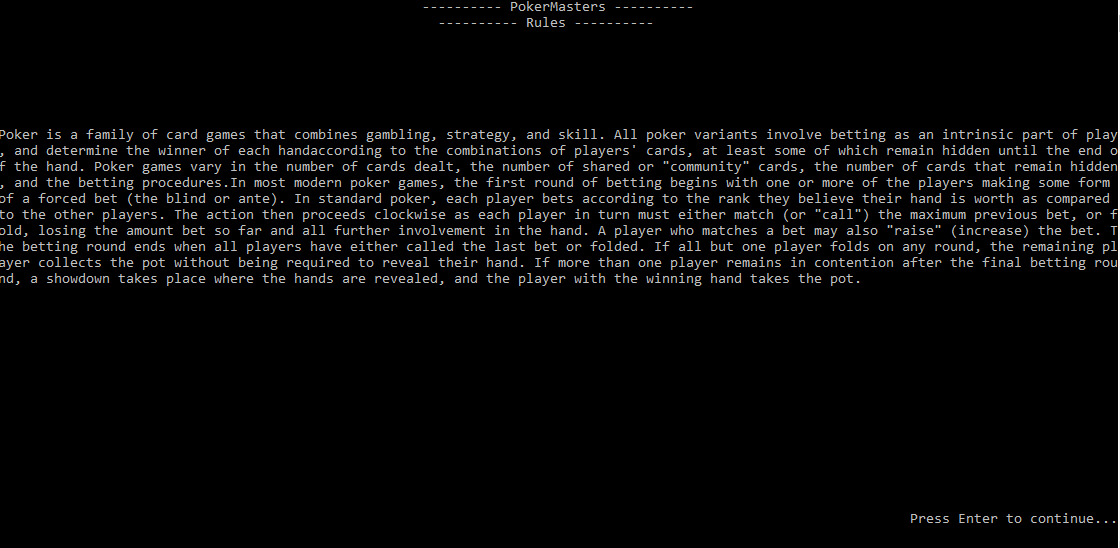
Some of the versions have been replaced by others since I have had problems when doing some activity, because if I did not finish the game.

**9. Screenshots of the final project**



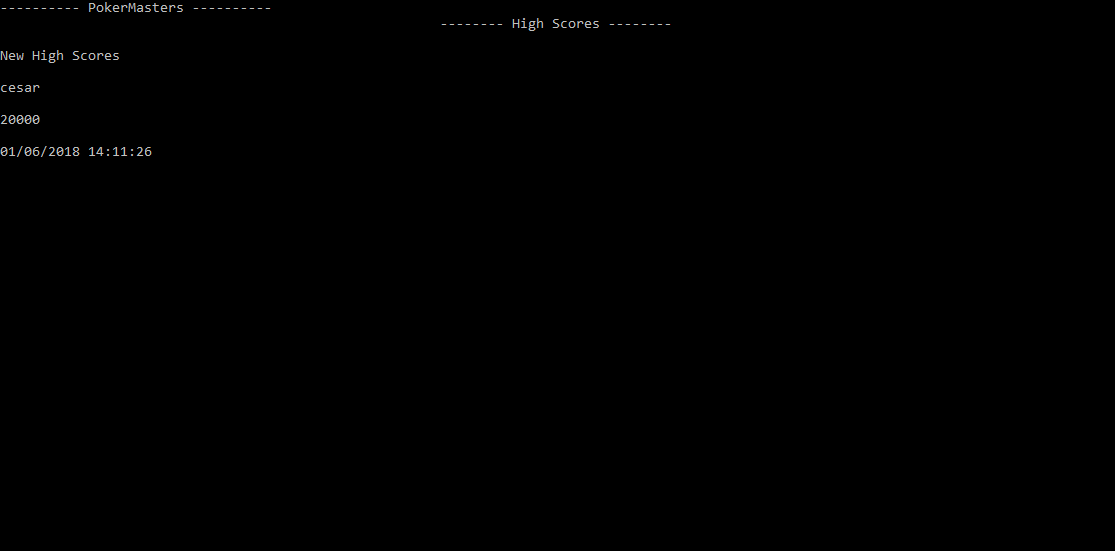










****

**10. Source code of the final project**

using System;

using System.Collections.Generic;

public class WelcomeScreen

{

private List<Player> Players { get; set; }

static void Title()

{

Console.WriteLine(" ===== ===== = = ===== ===== ");

Console.WriteLine(" = = = = = = = = = ");

Console.WriteLine(" ===== = = = ===== ==== ");

Console.WriteLine(" = = = = = = = = ");

Console.WriteLine(" = ===== = = ===== = = ");

Console.WriteLine();

}

// Show menu with the differents options

static void ShowMenu()

{

Console.WriteLine("-------- Choose one: --------");

Console.WriteLine("1.- Play in local");

Console.WriteLine("2.- Play in network");

Console.WriteLine("3.- Game rules");

Console.WriteLine("4.- Credits");

Console.WriteLine("5.- High Scores");

Console.WriteLine("6.- Change Language");

Console.WriteLine("Q.- Exit");

Console.WriteLine();

}

static void ShowMenuEsp()

{

Console.WriteLine("-------- Elige una: --------");

Console.WriteLine("1.- Jugar en local");

Console.WriteLine("2.- Jugar en red");

Console.WriteLine("3.- Reglas del juego");

Console.WriteLine("4.- Creditos");

Console.WriteLine("5.- Puntuaciones");

Console.WriteLine("6.- Cambiar idioma");

Console.WriteLine("Q.- Salir");

Console.WriteLine();

}

// We asked the user his name of the game

public string[] UserName()

{

string nPlayers;

do

{

Console.Write("Enter a number of players 2-6: ");

nPlayers = Console.ReadLine();

} while (nPlayers != "2" && nPlayers != "3" && nPlayers != "4" &&

nPlayers != "5" && nPlayers != "6");

int nPlayer = Convert.ToInt32(nPlayers);

string[] name = new string[nPlayer];

for (int i = 0; i < nPlayer; i++)

{

do

{

Console.WriteLine("Enter your user name #" + (i + 1) + ": ");

name[i] = Console.ReadLine();

} while (name[i] == "");

Console.WriteLine("Welcome to PokerMasters {0}!", name[i]);

}

return name;

}

public string[] UserNameEsp()

{

string nPlayers;

do

{

Console.Write("Introduce un numero de jugadores 2-6: ");

nPlayers = Console.ReadLine();

} while (nPlayers != "2" && nPlayers != "3" && nPlayers != "4" &&

nPlayers != "5" && nPlayers != "6");

int nPlayer = Convert.ToInt32(nPlayers);

string[] name = new string[nPlayer];

for (int i = 0; i < nPlayer; i++)

{

do

{

Console.WriteLine("Introduce tu nombre de usuario #" + (i + 1) + ": ");

name[i] = Console.ReadLine();

} while (name[i] == "");

Console.WriteLine("Bienvenido a PokerMasters {0}!", name[i]);

}

return name;

}

// The user enter the name player with will be play

public void Display()

{

GameScreen Game;

CreditsScreen Credits;

HiScoreScreen HiScore;

RulesScreen Rules;

string option;

bool exit = false;

do

{

Console.Clear();

Title();

ShowMenu();

option = Console.ReadLine();

switch (option.ToUpper())

{

case "1":

Console.Clear();

Game = new GameScreen();

string[] name = UserName();

Game.Run(name);

Console.Clear();

break;

case "2":

Console.Clear();

Console.WriteLine("Not available yet...");

Console.Clear();

Display();

break;

case "3":

Console.Clear();

Rules = new RulesScreen();

Rules.Display();

Console.Clear();

break;

case "4":

Console.Clear();

Credits = new CreditsScreen();

Credits.Run();

Console.Clear();

break;

case "5":

Console.Clear();

HiScore = new HiScoreScreen();

HiScore.Run();

Console.Clear();

break;

case "6":

DisplayEsp();

break;

case "Q":

exit = true;

Console.WriteLine("See you soon!");

Console.WriteLine();

break;

default:

Console.Clear();

Title();

ShowMenu();

Console.WriteLine();

Console.WriteLine("Enter a correct option!");

option = Console.ReadLine();

break;

}

} while (!exit);

}

public void DisplayEsp()

{

GameScreen Game;

CreditsScreen Credits;

HiScoreScreen HiScore;

RulesScreen Rules;

string option;

bool exit = false;

do

{

Console.Clear();

Title();

ShowMenuEsp();

option = Console.ReadLine();

switch (option.ToUpper())

{

case "1":

Console.Clear();

Game = new GameScreen();

string[] name = UserNameEsp();

Game.RunEsp(name);

Console.Clear();

break;

case "2":

Console.Clear();

Console.WriteLine("No disponible todavía...");

Console.Clear();

DisplayEsp();

break;

case "3":

Console.Clear();

Rules = new RulesScreen();

Rules.DisplayEsp();

Console.Clear();

break;

case "4":

Console.Clear();

Credits = new CreditsScreen();

Credits.RunEsp();

Console.Clear();

break;

case "5":

Console.Clear();

HiScore = new HiScoreScreen();

HiScore.RunEsp();

Console.Clear();

break;

case "6":

Display();

break;

case "Q":

exit = true;

Console.WriteLine("Hasta pronto!");

Console.WriteLine();

break;

default:

Console.Clear();

Title();

ShowMenu();

Console.WriteLine();

Console.WriteLine("Introduce una opcion correcta!");

option = Console.ReadLine();

break;

}

} while (!exit);

}

// Main loop of the welcome screen

public static void Main()

{

// Established Windows Size

Console.SetWindowSize(140, 35);

WelcomeScreen Welcome = new WelcomeScreen();

Welcome.Display();

}

}

using System;

using System.Collections.Generic;

public class GameScreen : ConsoleUpgrade

{

WelcomeScreen Welcome = new WelcomeScreen();

public Player Player { get; set; }

private Hand Hand { get; set; }

private CardsDeck Deck { get; set; }

private int round = 0;

public int pot;

public int Index { get; set; }

public Card[] Cards = new Card[5];

public bool isAbsent;

public bool allIsPlay;

public GameScreen()

{

Players = new List<Player>();

Index = 0;

}

public void Flop(CardsDeck deck)

{

// Burn

deck.Burn(deck);

Random r = new Random();

int number = r.Next(0, deck.Count);

Card card1 = deck.Cards[number];

deck.Cards.RemoveAt(number);

Console.SetCursorPosition(51, 15);

Console.Write(card1.Rank);

Console.SetCursorPosition(51, 16);

Console.Write(card1.Suit);

number = r.Next(0, deck.Count);

Card card2 = deck.Cards[number];

deck.Cards.RemoveAt(number);

Console.SetCursorPosition(56, 15);

Console.Write(card2.Rank);

Console.SetCursorPosition(56, 16);

Console.Write(card2.Suit);

number = r.Next(0, deck.Count);

Card card3 = deck.Cards[number];

deck.Cards.RemoveAt(number);

Console.SetCursorPosition(61, 15);

Console.Write(card3.Rank);

Console.SetCursorPosition(61, 16);

Console.Write(card3.Suit);

Cards[0] = card1;

Cards[1] = card2;

Cards[2] = card3;

}

public void Turn(CardsDeck deck)

{

// Burn

deck.Burn(deck);

Random r = new Random();

int number = r.Next(0, deck.Count);

Card card1 = deck.Cards[number];

deck.Cards.RemoveAt(number);

Console.SetCursorPosition(66, 15);

Console.Write(card1.Rank);

Console.SetCursorPosition(66, 16);

Console.Write(card1.Suit);

Cards[3] = card1;

}

public void River(CardsDeck deck)

{

// Burn

deck.Burn(deck);

Random r = new Random();

int number = r.Next(0, deck.Count);

Card card1 = deck.Cards[number];

deck.Cards.RemoveAt(number);

Console.SetCursorPosition(71, 15);

Console.Write(card1.Rank);

Console.SetCursorPosition(71, 16);

Console.Write(card1.Suit);

Cards[4] = card1;

}

// Check of cards to know who is the winner of each round

public void CheckCards(List<Player> Players, CardsDeck deck)

{

List<Card> check = new List<Card>();

for (int i = 0; i < Players.Count; i++)

{

check.Add(Players[i].cards[0]);

check.Add(Players[i].cards[1]);

check.Add(deck.Cards[0]);

check.Add(deck.Cards[1]);

check.Add(deck.Cards[2]);

check.Add(deck.Cards[3]);

check.Add(deck.Cards[4]);

//1 Royal flush

/\*if (royalFlush())

Players[i].HandValue = 1;

//2 Straight flush

else if (straightFlush())

Players[i].HandValue = 2

//3 Four of a kind

else \*/if (fourOfAKind(check))

Players[i].HandValue = 3;

//4 Full House

else if (fullHouse(check))

Players[i].HandValue = 4;

//5 Flush

else if (flush(check))

Players[i].HandValue = 5;

//6 Straight

/\* else if (straight(check))

Players[i].HandValue = 6;\*/

//7 Three of a kind

else if (threeOfAKind(check))

Players[i].HandValue = 7;

//8 Two pair

else if (twoPair(check))

Players[i].HandValue = 8;

//9 One pair

else if (onePair(check))

Players[i].HandValue = 9;

//10 High card

else

Players[i].HandValue = 10;

if (Players[0].HandValue < Players[1].HandValue)

{

Players[0].Chips += pot;

Console.SetCursorPosition(110, 13);

Console.WriteLine(Players[0].UserName + " wins!");

}

else if (Players[1].HandValue < Players[0].HandValue)

{

Players[1].Chips += pot;

Console.SetCursorPosition(110, 13);

Console.WriteLine(Players[1].UserName + " wins!");

}

else

{

int newPot = pot / 2;

Players[0].Chips = pot;

Players[1].Chips = newPot;

Console.SetCursorPosition(110, 13);

Console.WriteLine("Deuce!");

}

}

}

public void CheckCardsEsp(List<Player> Players, CardsDeck deck)

{

List<Card> check = new List<Card>();

for (int i = 0; i < Players.Count; i++)

{

check.Add(Players[i].cards[0]);

check.Add(Players[i].cards[1]);

check.Add(deck.Cards[0]);

check.Add(deck.Cards[1]);

check.Add(deck.Cards[2]);

check.Add(deck.Cards[3]);

check.Add(deck.Cards[4]);

//1 Royal flush

/\*if (royalFlush())

Players[i].HandValue = 1;

//2 Straight flush

else if (straightFlush())

Players[i].HandValue = 2

//3 Four of a kind

else \*/

if (fourOfAKind(check))

Players[i].HandValue = 3;

//4 Full House

else if (fullHouse(check))

Players[i].HandValue = 4;

//5 Flush

else if (flush(check))

Players[i].HandValue = 5;

//6 Straight

/\* else if (straight(check))

Players[i].HandValue = 6;\*/

//7 Three of a kind

else if (threeOfAKind(check))

Players[i].HandValue = 7;

//8 Two pair

else if (twoPair(check))

Players[i].HandValue = 8;

//9 One pair

else if (onePair(check))

Players[i].HandValue = 9;

//10 High card

else

Players[i].HandValue = 10;

if (Players[0].HandValue < Players[1].HandValue)

{

Players[0].Chips += pot;

Console.SetCursorPosition(110, 13);

Console.WriteLine(Players[0].UserName + " gana!");

}

else if (Players[1].HandValue < Players[0].HandValue)

{

Players[1].Chips += pot;

Console.SetCursorPosition(110, 13);

Console.WriteLine(Players[1].UserName + " gana!");

}

else

{

int newPot = pot / 2;

Players[0].Chips = pot;

Players[1].Chips = newPot;

Console.SetCursorPosition(110, 13);

Console.WriteLine("Empate!");

}

}

}

//fourOfAKind

private static bool fourOfAKind(List<Card> check)

{

for (int i = 0; i < 4; i++)

{

int cont = 0;

for (int j = 0; j < check.Count; j++)

{

if (check[i].Rank == check[j].Rank)

cont++;

if (cont == 4)

return true;

}

}

return false;

}

//fullHouse

private static bool fullHouse(List<Card> check)

{

for (int i = 0; i < 4; i++)

{

int cont = 0;

for (int j = 0; j < check.Count; j++)

{

if (check[i].Rank == check[j].Rank)

cont++;

if (cont == 3)

return true;

}

}

return false;

}

//flush

private static bool flush(List<Card> check)

{

for (int i = 0; i < 4; i++)

{

int cont = 0;

for (int j = 0; j < check.Count; j++)

{

if (check[i].Suit == check[j].Suit)

cont++;

if (cont == 5)

return true;

}

}

return false;

}

//threeOfAKind

private static bool threeOfAKind(List<Card> check)

{

for (int i = 0; i < 4; i++)

{

int cont = 0;

for (int j = 0; j < check.Count; j++)

{

if (check[i].Rank == check[j].Rank)

cont++;

if (cont == 3)

return true;

}

}

return false;

}

//twoPair

private static bool twoPair(List<Card> check)

{

for (int i = 0; i < 4; i++)

{

int cont = 0;

for (int j = 0; j < check.Count; j++)

{

if (check[i].Rank == check[j].Rank)

cont++;

if (cont == 2)

return true;

}

}

return false;

}

//onePair

private static bool onePair(List<Card> check)

{

for (int i = 0; i < 4; i++)

{

int cont = 0;

for (int j = 0; j < check.Count; j++)

{

if (check[i].Rank == check[j].Rank)

cont++;

if (cont == 1)

return true;

}

}

return false;

}

public void ShowMovements()

{

Console.SetCursorPosition(0, 30);

Console.WriteLine("1.FOLD / 2.CHECK / 3.CALL / 4.RAISE");

Console.WriteLine("Press Z for Absent mode");

Console.Write(Players[Index].UserName + ", enter a option: ");

}

public void ShowMovementsEsp()

{

Console.SetCursorPosition(0, 30);

Console.WriteLine("1.RETIRARSE / 2.PASAR / 3.IGUALAR / 4.SUBIR");

Console.WriteLine("Pulsa Z para el modo Ausente");

Console.Write(Players[Index].UserName + ", introduce una opcion: ");

}

public void ClearText()

{

Console.SetCursorPosition(0, 31);

Console.WriteLine(new string(' ', 100));

Console.SetCursorPosition(0, 32);

Console.WriteLine(new string(' ', 100));

Console.SetCursorPosition(0, 33);

Console.WriteLine(new string(' ', 100));

}

// Created the structure to choose the movements

public void Movements()

{

string move;

ShowMovements();

move = Console.ReadLine();

switch (move.ToUpper())

{

case "1":

Players[Index].Fold();

break;

case "2":

Players[Index].Check();

break;

case "3":

Players[Index].Call();

Players[Index].Chips -= Players[Index - 1].Pot;

pot += Players[Index - 1].Pot;

break;

case "4":

pot += Players[Index].Raise();

break;

case "Z":

isAbsent = Players[Index].Absent();

break;

case "Q":

Console.Clear();

Welcome.Display();

break;

default:

ClearText();

Movements();

break;

}

for (int i = 0; i < Players.Count - 1; i++)

{

if (move != "1")

{

allIsPlay = true;

}

}

}

public void MovementsEsp()

{

string move;

ShowMovementsEsp();

move = Console.ReadLine();

switch (move.ToUpper())

{

case "1":

Players[Index].Fold();

break;

case "2":

Players[Index].Check();

break;

case "3":

Players[Index].Call();

Players[Index].Chips -= Players[Index - 1].Pot;

pot += Players[Index - 1].Pot;

break;

case "4":

pot += Players[Index].RaiseEsp();

break;

case "Z":

isAbsent = Players[Index].Absent();

break;

case "Q":

Console.Clear();

Welcome.DisplayEsp();

break;

default:

ClearText();

MovementsEsp();

break;

}

for (int i = 0; i < Players.Count - 1; i++)

{

if (move != "1")

{

allIsPlay = true;

}

}

}

public void ShowExit()

{

Console.SetCursorPosition(119, 32);

Console.WriteLine("Press Q to go back...");

}

public void ShowExitEsp()

{

Console.SetCursorPosition(119, 32);

Console.WriteLine("Pulsa Q para volver...");

}

public void ShowMenu()

{

Console.SetCursorPosition(53, 0);

Console.WriteLine("---------- PokerMasters ----------");

Console.SetCursorPosition(53, 1);

Console.WriteLine(" ---------- Play Local ----------");

Console.WriteLine();

}

public void ShowMenuEsp()

{

Console.SetCursorPosition(53, 0);

Console.WriteLine("------------ PokerMasters ----------");

Console.SetCursorPosition(53, 1);

Console.WriteLine(" ---------- Jugar en local ----------");

Console.WriteLine();

}

public void Run(string[] names)

{

Deck = new CardsDeck();

Console.Clear();

ShowMenu();

// Position for username #1 with name in game and chips

Positions[] positions = new Positions[6];

positions[0].X = 20;

positions[0].Y = 8;

positions[1].X = 60;

positions[1].Y = 8;

positions[2].X = 100;

positions[2].Y = 8;

positions[3].X = 20;

positions[3].Y = 27;

positions[4].X = 60;

positions[4].Y = 27;

positions[5].X = 100;

positions[5].Y = 27;

for (int i = 0; i < names.Length; i++)

{

Player player = new Player(names[i]);

player.UserName = names[i];

Players.Add(player);

}

// Draw Game table and cards

DrawCard.Draw(Players, Deck);

DrawCard.UpdateChips(Players);

DrawCard.DrawTable(Deck);

// Create logs of players

CreateConfig(Players);

// Loop of game

GameLoop();

}

public void RunEsp(string[] names)

{

Deck = new CardsDeck();

Console.Clear();

ShowMenuEsp();

// Position for username #1 with name in game and chips

Positions[] positions = new Positions[6];

positions[0].X = 20;

positions[0].Y = 8;

positions[1].X = 60;

positions[1].Y = 8;

positions[2].X = 100;

positions[2].Y = 8;

positions[3].X = 20;

positions[3].Y = 27;

positions[4].X = 60;

positions[4].Y = 27;

positions[5].X = 100;

positions[5].Y = 27;

for (int i = 0; i < names.Length; i++)

{

Player player = new Player(names[i]);

player.UserName = names[i];

Players.Add(player);

}

// Draw Game table and cards

DrawCard.DrawEsp(Players, Deck);

DrawCard.UpdateChipsEsp(Players);

DrawCard.DrawTableEsp(Deck);

// Create logs of players

CreateConfigEsp(Players);

// Loop of game

GameLoopEsp();

}

public void GameLoop()

{

bool exit = false;

int nIsAbsent;

do

{

do

{

ShowExit();

for (int i = 0; i < Players.Count - 1; i++)

{

//1 Check user input

Movements();

// Payment of blinds

if (Players[i].bigBlind)

{

int big = 200;

Players[i].Chips -= big;

pot += big;

}

if (Players[i].smallBlind)

{

int small = 100;

Players[i].Chips -= small;

pot += small;

}

if (!isAbsent)

{

//2 Movements

if (Index < Players.Count - 1)

{

Index++;

// Sound to inform about the turn of another player

Console.Beep(600, 1000);

//Update Pot

DrawCard.DrawPot(pot);

DrawCard.UpdateChips(Players);

}

else

{

Index = 0;

Console.Beep(600, 1000);

//Update Pot

DrawCard.DrawPot(pot);

DrawCard.UpdateChips(Players);

}

}

else

{

nIsAbsent = Index;

if (Index < Players.Count - 1)

Index++;

else

Index = 0;

}

DrawCard.UpdateChips(Players);

}

// 3 firsts cards of the middle

if (allIsPlay)

{

Flop(Deck);

allIsPlay = false;

for (int i = 0; i < Players.Count; i++)

{

Movements();

if (!isAbsent)

{

//2 Movements

if (Index < Players.Count - 1)

{

Index++;

// Sound to inform about the turn of another player

Console.Beep(600, 1000);

//Update Pot

DrawCard.DrawPot(pot);

DrawCard.UpdateChips(Players);

}

else

{

Index = 0;

Console.Beep(600, 1000);

//Update Pot

DrawCard.DrawPot(pot);

DrawCard.UpdateChips(Players);

}

}

else

{

nIsAbsent = Index;

if (Index < Players.Count - 1)

Index++;

else

Index = 0;

}

}

}

else

{

allIsPlay = false;

}

if (allIsPlay)

{

Turn(Deck);

for (int i = 0; i < Players.Count; i++)

{

Movements();

if (Index < Players.Count - 1)

{

Index++;

Console.Beep(600, 1000);

DrawCard.DrawPot(pot);

DrawCard.UpdateChips(Players);

}

else

{

Index = 0;

Console.Beep(600, 1000);

//Update Pot

DrawCard.DrawPot(pot);

DrawCard.UpdateChips(Players);

}

}

}

else

{

allIsPlay = false;

}

// The last card of the middle

if (allIsPlay)

{

River(Deck);

for (int i = 0; i < Players.Count; i++)

{

Movements();

if (Index < Players.Count - 1)

{

Index++;

Console.Beep(600, 1000);

DrawCard.DrawPot(pot);

DrawCard.UpdateChips(Players);

}

else

{

Index = 0;

Console.Beep(600, 1000);

//Update Pot

DrawCard.DrawPot(pot);

DrawCard.UpdateChips(Players);

}

}

}

else

{

allIsPlay = false;

}

} while (!allIsPlay);

for (int timesToShuffle = 0; timesToShuffle < 10; timesToShuffle++)

{

Deck.Shuffle();

}

// Update cards after the turn

CheckCards(Players, Deck);

DrawCard.UpdateChips(Players);

DrawCard.DrawTable(Deck);

DrawCard.Draw(Players, Deck);

//DrawCard.DrawResult(Players);

pot = 0;

Index = 0;

Deck.Reset();

round++;

} while (!exit);

}

public void GameLoopEsp()

{

bool exit = false;

int nIsAbsent;

do

{

do

{

ShowExitEsp();

for (int i = 0; i < Players.Count - 1; i++)

{

//1 Check user input

MovementsEsp();

// Payment of blinds

if (Players[i].bigBlind)

{

int big = 200;

Players[i].Chips -= big;

pot += big;

}

if (Players[i].smallBlind)

{

int small = 100;

Players[i].Chips -= small;

pot += small;

}

if (!isAbsent)

{

//2 Movements

if (Index < Players.Count - 1)

{

Index++;

// Sound to inform about the turn of another player

Console.Beep(600, 1000);

//Update Pot

DrawCard.DrawPot(pot);

DrawCard.UpdateChipsEsp(Players);

}

else

{

Index = 0;

Console.Beep(600, 1000);

//Update Pot

DrawCard.DrawPot(pot);

DrawCard.UpdateChipsEsp(Players);

}

}

else

{

nIsAbsent = Index;

if (Index < Players.Count - 1)

Index++;

else

Index = 0;

}

DrawCard.UpdateChipsEsp(Players);

}

// 3 firsts cards of the middle

if (allIsPlay)

{

Flop(Deck);

allIsPlay = false;

for (int i = 0; i < Players.Count; i++)

{

MovementsEsp();

if (!isAbsent)

{

//2 Movements

if (Index < Players.Count - 1)

{

Index++;

// Sound to inform about the turn of another player

Console.Beep(600, 1000);

//Update Pot

DrawCard.DrawPot(pot);

DrawCard.UpdateChipsEsp(Players);

}

else

{

Index = 0;

Console.Beep(600, 1000);

//Update Pot

DrawCard.DrawPot(pot);

DrawCard.UpdateChipsEsp(Players);

}

}

else

{

nIsAbsent = Index;

if (Index < Players.Count - 1)

Index++;

else

Index = 0;

}

}

}

else

{

allIsPlay = false;

}

if (allIsPlay)

{

Turn(Deck);

for (int i = 0; i < Players.Count; i++)

{

MovementsEsp();

if (Index < Players.Count - 1)

{

Index++;

Console.Beep(600, 1000);

DrawCard.DrawPot(pot);

DrawCard.UpdateChipsEsp(Players);

}

else

{

Index = 0;

Console.Beep(600, 1000);

//Update Pot

DrawCard.DrawPot(pot);

DrawCard.UpdateChipsEsp(Players);

}

}

}

else

{

allIsPlay = false;

}

// The last card of the middle

if (allIsPlay)

{

River(Deck);

for (int i = 0; i < Players.Count; i++)

{

MovementsEsp();

if (Index < Players.Count - 1)

{

Index++;

Console.Beep(600, 1000);

DrawCard.DrawPot(pot);

DrawCard.UpdateChipsEsp(Players);

}

else

{

Index = 0;

Console.Beep(600, 1000);

//Update Pot

DrawCard.DrawPot(pot);

DrawCard.UpdateChipsEsp(Players);

}

}

}

else

{

allIsPlay = false;

}

} while (!allIsPlay);

for (int timesToShuffle = 0; timesToShuffle < 10; timesToShuffle++)

{

Deck.Shuffle();

}

// Update cards after the turn

CheckCardsEsp(Players, Deck);

DrawCard.UpdateChipsEsp(Players);

DrawCard.DrawTableEsp(Deck);

DrawCard.DrawEsp(Players, Deck);

//DrawCard.DrawResult(Players);

pot = 0;

Index = 0;

Deck.Reset();

round++;

} while (!exit);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

public class Player

{

public string UserName { get; set; }

public int Bet { get; set; }

public int Chips { get; set; }

public Card[] cards;

public int X { get; set; }

public int Y { get; set; }

public int Pot { get; set; }

// Atributes to asign

public bool bigBlind;

public bool smallBlind;

public byte HandValue;

public bool dealer;

public Player()

{

cards = new Card[2];

Chips = 20000;

}

public Player(string userName)

{

UserName = userName;

cards = new Card[2];

Chips = 20000;

}

public void ClearText()

{

Console.SetCursorPosition(0, 31);

Console.WriteLine(new string(' ', 100));

Console.SetCursorPosition(0, 32);

Console.WriteLine(new string(' ', 100));

Console.SetCursorPosition(0, 33);

Console.WriteLine(new string(' ', 100));

}

public int Raise()

{

int amount;

string nAmount;

bool correct = false;

do

{

Console.Write("Chips to raise: ");

nAmount = Console.ReadLine();

// Introduced the tryparse so that it does not crash when

//it is converted from string to int

correct = int.TryParse(nAmount, out amount);

if (correct)

{

amount = Convert.ToInt32(nAmount);

Chips = Chips - amount;

Pot = amount;

}

else

{

correct = false;

}

} while ((!(amount <= Chips)) || (!correct));

ClearText();

return amount;

}

public int RaiseEsp()

{

int amount;

string nAmount;

bool correct = false;

do

{

Console.Write("Chips to raise: ");

nAmount = Console.ReadLine();

correct = int.TryParse(nAmount, out amount);

if (correct)

{

amount = Convert.ToInt32(nAmount);

Chips = Chips - amount;

Pot = amount;

}

else

{

correct = false;

}

} while ((!(amount <= Chips)) || (!correct));

ClearText();

return amount;

}

public int Call()

{

Console.WriteLine("You called.");

ClearText();

return Pot;

}

public void Check()

{

Console.WriteLine("You checked.");

ClearText();

}

public void Fold()

{

Console.WriteLine("You folded.");

ClearText();

}

public bool Absent()

{

Console.WriteLine("You are absent.");

ClearText();

return true;

}

}

using System;

public class Card

{

public Suit Suit { get; set; }

public char Rank { get; set; }

public Card(Suit suit, char rank)

{

Suit = suit;

Rank = rank;

}

public void Flip()

{

// To do

}

}

using System;

using System.Collections.Generic;

public class CardsDeck

{

public List<Card> Cards { get; set; }

public int Count { get { return Cards.Count; } }

// Generate List to save the cards

public CardsDeck()

{

Cards = new List<Card>();

Create();

}

// Generate the create method to set each card in the deck

// and assign each of the suits and numbers.

public void Create()

{

Suit[] Suits = new Suit[4];

Suits[0] = Suit.C;

Suits[1] = Suit.D;

Suits[2] = Suit.H;

Suits[3] = Suit.S;

// Assign each letter a suit and a rank

for (int suitIndex = 0; suitIndex < 4; suitIndex++)

{

for (int rankIndex = 1; rankIndex <= 13; rankIndex++)

{

switch (rankIndex)

{

case 10:

Cards.Add(new Card(Suits[suitIndex], 'X'));

break;

case 11:

Cards.Add(new Card(Suits[suitIndex], 'J'));

break;

case 12:

Cards.Add(new Card(Suits[suitIndex], 'Q'));

break;

case 13:

Cards.Add(new Card(Suits[suitIndex], 'K'));

break;

default:

Cards.Add(new Card(Suits[suitIndex],

Convert.ToChar(rankIndex.ToString())));

break;

}

}

}

}

public void Deal()

{

// To do

}

// Reset list of cards (deck of cards)

public void Shuffle()

{

Random r = new Random();

for (int i = Cards.Count - 1; i > 0; i--)

{

int n = r.Next(i + 1);

Card temp = Cards[i];

Cards[i] = Cards[n];

Cards[n] = temp;

}

}

public void Reset()

{

Create();

}

public void TopCard()

{

// To do

}

public void Burn(CardsDeck deck)

{

Random r = new Random();

int number = r.Next(0, deck.Count);

deck.Cards.RemoveAt(number);

}

}

using System;

using System.Collections.Generic;

public class Hand

{

// A list of cards has been made in the hand class and the AddCard method

// has been created to add cards to the hand

protected List<Card> cards = new List<Card>();

public List<Player> Players { get; set; }

public void AddCard(Card card)

{

cards.Add(card);

}

}

public enum Suit

{

S = 3,

H = 2,

C = 1,

D = 0

}

using System;

public class Bank

{

private int chips;

public void TakeOutChips()

{

// To do

}

public void Pay()

{

// To do

}

}

using System;

class RulesScreen

{

private string text = "Poker is a family of card games that combines " +

"gambling, strategy, and skill. All poker variants involve betting " +

"as an intrinsic part of play, and determine the winner of each hand" +

"according to the combinations of players' cards, at least some of " +

"which remain hidden until the end of the hand. Poker games vary in " +

"the number of cards dealt, the number of shared or \"community\" " +

"cards, the number of cards that remain hidden, and the betting " +

"procedures." +

"In most modern poker games, the first round of betting begins with " +

"one or more of the players making some form of a forced bet " +

"(the blind or ante). In standard poker, each player bets according " +

"to the rank they believe their hand is worth as compared to the " +

"other players. The action then proceeds clockwise as each player " +

"in turn must either match (or \"call\") the maximum previous bet, " +

"or fold, losing the amount bet so far and all further involvement " +

"in the hand. A player who matches a bet may also \"raise\" " +

"(increase) the bet. The betting round ends when all players have " +

"either called the last bet or folded. If all but one player folds " +

"on any round, the remaining player collects the pot without being " +

"required to reveal their hand. If more than one player remains in " +

"contention after the final betting round, a showdown takes place " +

"where the hands are revealed, and the player with the winning " +

"hand takes the pot.";

private string textEsp = "Poker es una familia de juegos de cartas que " +

"combina juegos de azar, estrategia y habilidad. Todas las " +

"variantes de póquer implican apuestas como una parte intrínseca " +

"del juego, y determinan el ganador de cada mano según las " +

"combinaciones de cartas de los jugadores, al menos algunas de las " +

"cuales permanecen ocultas hasta el final de la mano. Los juegos " +

"de póker varían en la cantidad de cartas repartidas, la cantidad " +

"de cartas compartidas o \"comunitarias\", el número de cartas que " +

"permanecen ocultas y los procedimientos de apuestas." +

"En la mayoría de los juegos de póquer modernos, la primera ronda " +

"de apuestas comienza con uno o más de los jugadores que realizan " +

"alguna forma de apuesta forzada(ciego o ante). En el poker " +

"estándar, cada jugador apuesta de acuerdo con el rango que cree " +

"que vale su mano en comparación con los otros jugadores.La acción " +

"luego continúa en el sentido de las manecillas del reloj ya que " +

"cada jugador a su vez debe igualar(o \"llamar\") la apuesta " +

"anterior máxima, o retirarse, perdiendo la cantidad apostada " +

"hasta el momento y toda participación adicional en la mano.Un " +

"jugador que iguala una apuesta también puede \"subir\" (aumentar) " +

"la apuesta.La ronda de apuestas finaliza cuando todos los " +

"jugadores han igualado la última apuesta o se han retirado.Si " +

"todos los jugadores menos uno se retiran en cualquier ronda, el " +

"jugador restante recoge el bote sin que se le exija revelar su " +

"mano. Si más de un jugador permanece en contención después de la " +

"última ronda de apuestas, se produce un enfrentamiento donde se " +

"revelan las manos, y el jugador con la mano ganadora toma el bote.";

private string[] listOfHands = {

"1.- Royal Flush",

"2.- Straight Flush",

"3.- Four of a kind",

"4.- Full House",

"5.- Flush",

"6.- Straight",

"7.- Three of a kind",

"8.- Two pair",

"9.- One pair",

"10.- High card",

};

private string[] listOfHandsEsp = {

"1.- Escalera Real",

"2.- Escalera de Color",

"3.- Póker",

"4.- Full",

"5.- Color",

"6.- Escalera",

"7.- Trío",

"8.- Doble pareja",

"9.- Pareja",

"10.- Carta alta",

};

public void ShowMenu()

{

Console.SetCursorPosition(53, 0);

Console.WriteLine("---------- PokerMasters ----------");

Console.SetCursorPosition(53, 1);

Console.WriteLine(" ---------- Rules ----------");

Console.WriteLine();

}

public void ShowMenuEsp()

{

Console.SetCursorPosition(53, 0);

Console.WriteLine("---------- PokerMasters ----------");

Console.SetCursorPosition(53, 1);

Console.WriteLine(" ---------- Reglas ----------");

Console.WriteLine();

}

public void ShowNext()

{

Console.SetCursorPosition(114, 32);

Console.WriteLine("Press Enter to continue...");

}

public void ShowNextEsp()

{

Console.SetCursorPosition(114, 32);

Console.WriteLine("Pulsa Intro para continuar...");

}

public void ShowExit()

{

Console.SetCursorPosition(119, 32);

Console.WriteLine("Press Q to go back...");

}

public void ShowExitEsp()

{

Console.SetCursorPosition(119, 32);

Console.WriteLine("Pulsa Q para volver...");

}

public void Display()

{

ShowMenu();

Console.SetCursorPosition(0, 17 / 2);

Console.WriteLine(text);

ShowNext();

string press = Console.ReadLine();

do

{

ShowExit();

if (press == "")

{

Console.Clear();

byte j = 12;

for (int i = 0; i < listOfHands.Length; i++)

{

Console.SetCursorPosition(62, j);

Console.WriteLine(listOfHands[i]);

j++;

}

ShowExit();

}

press = Console.ReadLine().ToUpper();

} while (press != "Q");

}

public void DisplayEsp()

{

ShowMenuEsp();

Console.SetCursorPosition(70 / 2, 17 / 2);

Console.WriteLine(textEsp);

ShowNextEsp();

string press = Console.ReadLine();

do

{

ShowExit();

if (press == "")

{

Console.Clear();

byte j = 12;

for (int i = 0; i < listOfHandsEsp.Length; i++)

{

Console.SetCursorPosition(62, j);

Console.WriteLine(listOfHandsEsp[i]);

j++;

}

ShowExitEsp();

}

press = Console.ReadLine().ToUpper();

} while (press != "Q");

}

}

using System;

using System.Collections.Generic;

using System.IO;

public class ConsoleUpgrade

{

public struct Positions

{

public int X;

public int Y;

}

public List<Player> Players { get; set; }

public ConsoleUpgrade()

{

Players = new List<Player>();

}

//Write the player's characteristics to the right of the welcome screen.

public static void WriteNames(Player player)

{

Console.SetCursorPosition(50, 2);

Console.WriteLine(player.UserName+ ", you have "+ player.Chips

+" chips");

}

//Created this method for when we use configuration file

public static void CreateConfig(List<Player> Players)

{

StreamWriter outFile;

DateTime now = DateTime.Now;

try

{

outFile = File.AppendText(@"..\..\..\configs\logs.txt");

outFile.WriteLine("/Logs PokerMasters/");

for (int i = 0; i < Players.Count; i++)

{

outFile.WriteLine(Players[i].UserName + "@" +

Players[i].Chips + "@" + now + "@" + Players[i].X

+ "@" + Players[i].Y);

}

outFile.Close();

}

catch (PathTooLongException)

{

Console.SetCursorPosition(119, 33);

Console.WriteLine("Entered path was too long.");

return;

}

catch (FileNotFoundException)

{

Console.SetCursorPosition(119, 33);

Console.WriteLine("File not found.");

return;

}

catch (IOException exp)

{

Console.SetCursorPosition(119, 33);

Console.WriteLine("Input/output error: {0}", exp.Message);

return;

}

catch (Exception exp)

{

Console.SetCursorPosition(119, 33);

Console.WriteLine("Unexpected error: {0}", exp.Message);

return;

}

Console.SetCursorPosition(119, 33);

Console.WriteLine("Extraction finished");

}

public static void CreateConfigEsp(List<Player> Players)

{

StreamWriter outFile;

DateTime now = DateTime.Now;

try

{

outFile = File.AppendText(@"..\..\..\configs\logs.txt");

outFile.WriteLine("/Logs PokerMasters/");

for (int i = 0; i < Players.Count; i++)

{

outFile.WriteLine(Players[i].UserName + "@" +

Players[i].Chips + "@" + now + "@" + Players[i].X

+ "@" + Players[i].Y);

}

outFile.Close();

}

catch (PathTooLongException)

{

Console.SetCursorPosition(119, 33);

Console.WriteLine("La ruta introducida era demasiado larga.");

return;

}

catch (FileNotFoundException)

{

Console.SetCursorPosition(119, 33);

Console.WriteLine("Fichero no encontrado.");

return;

}

catch (IOException exp)

{

Console.SetCursorPosition(119, 33);

Console.WriteLine("Error salida/entrada: {0}", exp.Message);

return;

}

catch (Exception exp)

{

Console.SetCursorPosition(119, 33);

Console.WriteLine("Error inesperado: {0}", exp.Message);

return;

}

Console.SetCursorPosition(119, 33);

Console.WriteLine("Extraído correctamente");

}

//Created this method for when use Hi Scores in HiScoresSreen.cs

public static void ShowScores()

{

StreamReader inFile;

StreamWriter outFile;

string line;

try

{

inFile = File.OpenText(@"..\..\..\configs\logs.txt");

outFile = File.CreateText(@"..\..\..\configs\hiScores.txt");

do

{

line = inFile.ReadLine();

if (line != null)

{

Extract(line, outFile);

}

} while (line != null);

outFile.Close();

inFile.Close();

}

catch (PathTooLongException)

{

Console.WriteLine("Entered path was too long.");

return;

}

catch (FileNotFoundException)

{

Console.WriteLine("File not found.");

return;

}

catch (IOException exp)

{

Console.WriteLine("Input/output error: {0}", exp.Message);

return;

}

catch (Exception exp)

{

Console.WriteLine("Unexpected error: {0}", exp.Message);

return;

}

}

public static void ShowScoresEsp()

{

StreamReader inFile;

StreamWriter outFile;

string line;

try

{

inFile = File.OpenText(@"..\..\..\configs\logs.txt");

outFile = File.CreateText(@"..\..\..\configs\hiScores.txt");

do

{

line = inFile.ReadLine();

if (line != null)

{

ExtractEsp(line, outFile);

}

} while (line != null);

outFile.Close();

inFile.Close();

}

catch (PathTooLongException)

{

Console.WriteLine("La ruta introducida era demasiado larga.");

return;

}

catch (FileNotFoundException)

{

Console.WriteLine("Fichero no encontrado.");

return;

}

catch (IOException exp)

{

Console.WriteLine("Error salida/entrada: {0}", exp.Message);

return;

}

catch (Exception exp)

{

Console.WriteLine("Error inesperado: {0}", exp.Message);

return;

}

}

public static void Extract(string line, StreamWriter file)

{

if (line.Contains("/Logs PokerMasters/"))

{

line = line.Replace("/Logs PokerMasters/", "New High Scores");

}

int positionUsername = line.IndexOf('@');

string[] usernames = line.Split('@');

string chips = line.Substring(positionUsername + 1, 5);

line = line.Replace("@", " ");

foreach (string player in usernames)

{

file.WriteLine(player);

file.WriteLine();

}

}

public static void ExtractEsp(string line, StreamWriter file)

{

if (line.Contains("/Logs PokerMasters/"))

{

line = line.Replace("/Logs PokerMasters/", "Nuevas Puntuaciones");

}

int positionUsername = line.IndexOf('@');

string[] usernames = line.Split('@');

string chips = line.Substring(positionUsername + 1, 5);

line = line.Replace("@", " ");

foreach (string player in usernames)

{

file.WriteLine(player);

file.WriteLine();

}

}

}

using System;

using System.IO;

using System.Collections.Generic;

class HiScoreScreen

{

public List<Player> Players { get; set; }

public void ShowMenu()

{

Console.WriteLine("---------- PokerMasters ----------");

Console.SetCursorPosition(53, 1);

Console.WriteLine(" -------- High Scores --------");

Console.WriteLine();

}

public void ShowMenuEsp()

{

Console.SetCursorPosition(53, 0);

Console.WriteLine("---------- PokerMasters ----------");

Console.SetCursorPosition(53, 1);

Console.WriteLine(" -------- Puntuaciones --------");

Console.WriteLine();

}

public void ShowExit()

{

Console.WriteLine("Press Q to go back...");

}

public void ShowExitEsp()

{

Console.WriteLine("Pulsa Q para volver...");

}

public void ShowNext()

{

Console.WriteLine("Press Enter to continue...");

}

public void ShowNextEsp()

{

Console.WriteLine("Pulsa Enter para continuar...");

}

public void Run()

{

string press;

do

{

ShowMenu();

ConsoleUpgrade.ShowScores();

try

{

StreamReader input =

new StreamReader(@"..\..\..\configs\hiScores.txt");

string line;

do

{

line = input.ReadLine();

if (line != null)

{

Console.WriteLine(line);

if (line.Length % 5 == 4)

{

Console.CursorVisible = false;

Console.ReadLine();

Console.Clear();

ShowMenu();

}

}

} while (line != null);

input.Close();

}

catch (PathTooLongException)

{

Console.WriteLine("Entered path was too long.");

return;

}

catch (FileNotFoundException)

{

Console.WriteLine("File not found.");

return;

}

catch (IOException exp)

{

Console.WriteLine("Input/output error: {0}", exp.Message);

return;

}

catch (Exception exp)

{

Console.WriteLine("Unexpected error: {0}", exp.Message);

return;

}

Console.WriteLine("Extraction finished");

ShowExit();

press = Console.ReadLine().ToUpper();

} while (press != "Q");

}

public void RunEsp()

{

string press;

do

{

ShowMenuEsp();

ShowNextEsp();

ConsoleUpgrade.ShowScoresEsp();

try

{

StreamReader input =

new StreamReader(@"..\..\..\configs\hiScores.txt");

string line;

do

{

line = input.ReadLine();

if (line != null)

{

Console.WriteLine(line);

if (line.Length % 5 == 4)

{

Console.CursorVisible = false;

Console.ReadLine();

Console.Clear();

ShowMenuEsp();

ShowNextEsp();

}

}

} while (line != null);

input.Close();

}

catch (PathTooLongException)

{

Console.WriteLine("La ruta introducida era demasiado larga.");

return;

}

catch (FileNotFoundException)

{

Console.WriteLine("Fichero no encontrado.");

return;

}

catch (IOException exp)

{

Console.WriteLine("Error salida/entrada: {0}", exp.Message);

return;

}

catch (Exception exp)

{

Console.WriteLine("Error inesperado: {0}", exp.Message);

return;

}

Console.WriteLine("Extraído correctamente");

ShowExitEsp();

press = Console.ReadLine().ToUpper();

} while (press != "Q");

}

}

using System;

using System.Threading;

public class CreditsScreen : WelcomeScreen

{

public void ShowMenu()

{

Console.SetCursorPosition(53, 0);

Console.WriteLine("---------- PokerMasters ----------");

Console.SetCursorPosition(53, 1);

Console.WriteLine(" ---------- Credits ----------");

Console.WriteLine();

}

public void ShowMenuEsp()

{

Console.SetCursorPosition(53, 0);

Console.WriteLine("---------- PokerMasters ----------");

Console.SetCursorPosition(53, 1);

Console.WriteLine(" ---------- Creditos ----------");

Console.WriteLine();

}

protected string[] names = {

"Producer and Project Director: César Martín",

"Lead Programmer and Assistant Director: César Martín",

"Programmers: César Martín",

"Designer: César Martín",

"Interface: César Martín",

"Textures: César Martín"

};

protected string[] namesEsp = {

"Producido y Director del proyecto: César Martín",

"Líder programador y asistente director: César Martín",

"Programador: César Martín",

"Diseñador: César Martín",

"Interfaz: César Martín",

"Texturas: César Martín"

};

public void Run()

{

ShowMenu();

for (int i = 0; i < names.Length; i++)

{

Console.Clear();

int x = 140;

int y = 35;

Console.CursorVisible = false;

Console.SetCursorPosition((x / 2) - (names[i].Length / 2),

y / 2);

Console.WriteLine(names[i]);

Thread.Sleep(2000);

}

}

public void RunEsp()

{

ShowMenu();

for (int i = 0; i < namesEsp.Length; i++)

{

Console.Clear();

int x = 140;

int y = 35;

Console.CursorVisible = false;

Console.SetCursorPosition((x / 2) - (namesEsp[i].Length / 2),

y / 2);

Console.WriteLine(namesEsp[i]);

Thread.Sleep(2000);

}

}

}

using System;

using System.Collections.Generic;

public class DrawCard : ConsoleUpgrade

{

private GameScreen Game { get; set; }

public static void Draw(List<Player> Players, CardsDeck deck)

{

Positions[] positions = new Positions[6];

positions[0].X = 20;

positions[0].Y = 5;

positions[1].X = 60;

positions[1].Y = 5;

positions[2].X = 100;

positions[2].Y = 5;

positions[3].X = 20;

positions[3].Y = 25;

positions[4].X = 60;

positions[4].Y = 25;

positions[5].X = 100;

positions[5].Y = 25;

//Generator for random numbers

Random r = new Random();

// Positions for players, with username and chips, and cards

for (int i = 0; i < Players.Count; i++)

{

int number = r.Next(0, deck.Count);

Players[i].cards[0] = deck.Cards[number];

deck.Cards.RemoveAt(number);

number = r.Next(0, deck.Count);

Players[i].cards[1] = deck.Cards[number];

deck.Cards.RemoveAt(number);

string[] cards = { " --- --- ",

"| "+Players[i].cards[0].Rank+" || "

+Players[i].cards[1].Rank+" |",

"| "+Players[i].cards[0].Suit+" || "

+Players[i].cards[1].Suit+" |",

"| || |",

" --- --- "};

for (int j = 0; j < cards.Length; j++)

{

Console.SetCursorPosition(positions[i].X, positions[i].Y + j);

Console.WriteLine(cards[j]);

}

Console.SetCursorPosition(positions[i].X, positions[i].Y - 2);

Console.WriteLine("Name: " + Players[i].UserName);

/\*Console.SetCursorPosition(positions[i].X, positions[i].Y - 1);

Console.WriteLine("Chips: " + Players[i].Chips);\*/

}

// Big Blind

Random rand = new Random();

int bigBlind = rand.Next(0, Players.Count - 1);

Console.SetCursorPosition(positions[bigBlind].X + 14,

positions[bigBlind].Y + 2);

Console.WriteLine("B");

Players[bigBlind].bigBlind = true;

// Small blind

bigBlind = rand.Next(0, Players.Count - 1);

Console.SetCursorPosition(positions[bigBlind + 1].X + 14,

positions[bigBlind + 1].Y + 2);

Console.WriteLine("S");

Players[bigBlind + 1].smallBlind = true;

}

public static void DrawEsp(List<Player> Players, CardsDeck deck)

{

Positions[] positions = new Positions[6];

positions[0].X = 20;

positions[0].Y = 5;

positions[1].X = 60;

positions[1].Y = 5;

positions[2].X = 100;

positions[2].Y = 5;

positions[3].X = 20;

positions[3].Y = 25;

positions[4].X = 60;

positions[4].Y = 25;

positions[5].X = 100;

positions[5].Y = 25;

//Generator for random numbers

Random r = new Random();

// Positions for players, with username and chips, and cards

for (int i = 0; i < Players.Count; i++)

{

int number = r.Next(0, deck.Count);

Players[i].cards[0] = deck.Cards[number];

deck.Cards.RemoveAt(number);

number = r.Next(0, deck.Count);

Players[i].cards[1] = deck.Cards[number];

deck.Cards.RemoveAt(number);

string[] cards = { " --- --- ",

"| "+Players[i].cards[0].Rank+" || "

+Players[i].cards[1].Rank+" |",

"| "+Players[i].cards[0].Suit+" || "

+Players[i].cards[1].Suit+" |",

"| || |",

" --- --- "};

for (int j = 0; j < cards.Length; j++)

{

Console.SetCursorPosition(positions[i].X, positions[i].Y + j);

Console.WriteLine(cards[j]);

}

Console.SetCursorPosition(positions[i].X, positions[i].Y - 2);

Console.WriteLine("Nombre: " + Players[i].UserName);

/\*Console.SetCursorPosition(positions[i].X, positions[i].Y - 1);

Console.WriteLine("Chips: " + Players[i].Chips);\*/

}

// Big Blind

Random rand = new Random();

int bigBlind = rand.Next(0, Players.Count - 1);

Console.SetCursorPosition(positions[bigBlind].X + 14,

positions[bigBlind].Y + 2);

Console.WriteLine("G");

Players[bigBlind].bigBlind = true;

// Small blind

bigBlind = rand.Next(0, Players.Count - 1);

Console.SetCursorPosition(positions[bigBlind + 1].X + 14,

positions[bigBlind + 1].Y + 2);

Console.WriteLine("P");

Players[bigBlind + 1].smallBlind = true;

}

public static void UpdateChips(List<Player> Players)

{

Positions[] positions = new Positions[6];

positions[0].X = 20;

positions[0].Y = 5;

positions[1].X = 60;

positions[1].Y = 5;

positions[2].X = 100;

positions[2].Y = 5;

positions[3].X = 20;

positions[3].Y = 25;

positions[4].X = 60;

positions[4].Y = 25;

positions[5].X = 100;

positions[5].Y = 25;

for (int i = 0; i < Players.Count; i++)

{

Console.SetCursorPosition(positions[i].X, positions[i].Y - 1);

Console.WriteLine("Chips: " + Players[i].Chips.ToString("00000"));

}

}

public static void UpdateChipsEsp(List<Player> Players)

{

Positions[] positions = new Positions[6];

positions[0].X = 20;

positions[0].Y = 5;

positions[1].X = 60;

positions[1].Y = 5;

positions[2].X = 100;

positions[2].Y = 5;

positions[3].X = 20;

positions[3].Y = 25;

positions[4].X = 60;

positions[4].Y = 25;

positions[5].X = 100;

positions[5].Y = 25;

for (int i = 0; i < Players.Count; i++)

{

Console.SetCursorPosition(positions[i].X, positions[i].Y - 1);

Console.WriteLine("Fichas: " + Players[i].Chips.ToString("00000"));

}

}

// Draw a table game with cards.

public static void DrawTable(CardsDeck deck)

{

string[] table =

{

" -------------------------------------------- ",

" / \\ ",

"| --- --- --- --- --- |",

"| | || || || || | POT: |",

"| | || || || || | |",

"| | || || || || | |",

"| --- --- --- --- --- |",

" \\ / ",

" -------------------------------------------- ",

};

for (int i = 0; i < table.Length; i++)

{

Console.SetCursorPosition(45, 12+i);

Console.WriteLine(table[i]);

}

}

public static void DrawTableEsp(CardsDeck deck)

{

string[] table =

{

" -------------------------------------------- ",

" / \\ ",

"| --- --- --- --- --- |",

"| | || || || || | BOTE: |",

"| | || || || || | |",

"| | || || || || | |",

"| --- --- --- --- --- |",

" \\ / ",

" -------------------------------------------- ",

};

for (int i = 0; i < table.Length; i++)

{

Console.SetCursorPosition(45, 12 + i);

Console.WriteLine(table[i]);

}

}

public static void DrawPot(int pot)

{

Console.SetCursorPosition(80, 16);

Console.WriteLine(pot.ToString("000000"));

}

//TO DO

/\*

public static void DrawResult(List<Player> Players)

{

if (Players.Count == 2)

{

}

}\*/

}