Casino C++

Nojus Cininas, Rokas Makaveckas, Nikita Savickis

1 Casino 1

1.1 How to play . . . 1

1.2 Program working documentation . . . 2

2 Class Index 3

2.1 Class List . . . 3

3 File Index 5

3.1 File List . . . 5

4 Class Documentation 7

4.1 BJ Struct Reference . . . 7

4.1.1 Detailed Description . . . 7

4.1.2 Constructor & Destructor Documentation . . . 7

4.1.2.1 BJ() . . . 7

4.1.2.2 ~BJ() . . . 8

4.1.3 Member Function Documentation . . . 8

4.1.3.1 calculateScore() . . . 8

4.1.3.2 dealersMove() . . . 8

4.1.3.3 giveCards() . . . 8

4.1.3.4 printCard() . . . 9

4.1.3.5 rndCard() . . . 9

4.1.3.6 showCards() . . . 9

4.1.3.7 startGame() . . . 9

4.2 Dice Struct Reference . . . 10

4.2.1 Detailed Description . . . 10

4.2.2 Constructor & Destructor Documentation . . . 10

4.2.2.1 Dice() . . . 10

4.2.2.2 ~Dice() . . . 10

4.2.3 Member Function Documentation . . . 11

4.2.3.1 startGame() . . . 11

4.3 DorN Struct Reference . . . 11

4.3.1 Detailed Description . . . 11

4.3.2 Constructor & Destructor Documentation . . . 12

4.3.2.1 DorN() . . . 12

4.3.2.2 ~DorN() . . . 12

4.3.3 Member Function Documentation . . . 12

4.3.3.1 startGame() . . . 12

4.4 Hand Class Reference . . . 13

4.4.1 Detailed Description . . . 13

4.4.2 Constructor & Destructor Documentation . . . 14

4.4.2.1 Hand() . . . 14

4.4.3 Member Data Documentation . . . 14

4.4.3.1 aces . . . 14
4.4.3.2 altScore . . . 14
4.4.3.3 end . . . 14
4.4.3.4 n . . . 14
4.4.3.5 nCards . . . 14
4.4.3.6 score . . . 15
4.5 Login Struct Reference . . . 15
4.5.1 Detailed Description . . . 15
4.5.2 Member Function Documentation . . . 15
4.5.2.1 changeCSV() . . . 15
4.5.2.2 getPlayer() . . . 16
4.5.2.3 printList() . . . 16
4.6 Player Class Reference . . . 16
4.6.1 Detailed Description . . . 17
4.6.2 Constructor & Destructor Documentation . . . 17
4.6.2.1 Player() [1/2] . . . 17
4.6.2.2 Player() [2/2] . . . 17
4.6.3 Member Function Documentation . . . 17
4.6.3.1 decreaseBalance() . . . 17
4.6.3.2 getBalance() . . . 17
4.6.3.3 getLosses() . . . 17
4.6.3.4 getName() . . . 18
4.6.3.5 getWins() . . . 18
4.6.3.6 increaseBalance() . . . 18
4.6.3.7 incrementLosses() . . . 18
4.6.3.8 incrementWins() . . . 18
4.6.3.9 setBalance() . . . 18
4.6.3.10 setLosses() . . . 18
4.6.3.11 setName() . . . 19
4.6.3.12 setWins() . . . 19

5 File Documentation **21**

5.1 BJ.cpp . . . 21
5.2 BJ.h . . . 24
5.3 Dice.cpp . . . 24
5.4 Dice.h . . . 25
5.5 DorN.cpp . . . 25
5.6 DorN.h . . . 27
5.7 Hand.cpp . . . 27
5.8 Hand.h . . . 27
5.9 Login.cpp . . . 27
5.10 Login.h . . . 28

5.11 main.cpp 29
5.12 Player.cpp 29
5.13 Player.h 30

Index **31**

Chapter 1

Casino

Simple practising tool meant for fun and as a proof of concept

1.1 How to play

At launch the program prompts you to enter your Username, as this is still a proof of concept, there is no way of registering or creating an instance of youself, thus you have to choose 1 from 1000 available in the `player↩ Data.csv` file.

After successful login you can select one of the following options:

1. Double or Nothing (DorN)

   Simple coinflip game where you can select heads or tails and enter your desired bet amount.

2. BlackJack (BJ)

   Starts a BlackJack game and lets you enter your bet amount

3. Poker

   NOT IMPLEMENTED

4. Dice

   Dice game where you can select tie, red or blue, whichever die rolls the highest wins or it's a tie

5. Roulette

   NOT IMPLEMENTED

6. Get your balance report

   Writes out currently available balance to the player.

1.2 Program working documentation

The full documentation is available in the `dokumentacija.pdf` file with descriptions.

The working principle is shown it he picture below:

TODO list:

- Create games
  - ~~Black Jack~~
  - Poker
  - ~~Dices~~
  - Roulette
  - ~~Double or Nothing~~
- ~~Add currence/balance~~
- ~~Add the ability to save your balance for later log in~~
- **GUI :)**

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

BJ 7
Dice 10
DorN 11
Hand 13
Login 15
Player 16

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

BJ.cpp 21
BJ.h 24
Dice.cpp 24
Dice.h 25
DorN.cpp 25
DorN.h 27
Hand.cpp 27
Hand.h 27
Login.cpp 27
Login.h 28
main.cpp 29
Player.cpp 29
Player.h 30

Chapter 4

Class Documentation

4.1 BJ Struct Reference

Public Member Functions

- BJ (Player &player)

  Constructs a new instance of the BJ class.
- void startGame (Player &player)

  Starts the BlackJack game.
- void rndCard (int[ ], Hand &)
- void giveCards (int[ ], Hand &, Hand &)
- void showCards (Hand &, Hand &)
- void printCard (int)
- void calculateScore (Hand &)
- void dealersMove (int[ ], Hand &, Hand &)

4.1.1 Detailed Description

Definition at line 8 of file BJ.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 BJ()

```
BJ::BJ (
            Player & player)
```

Constructs a new instance of the BJ class.

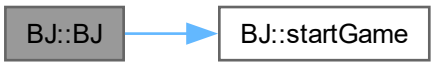
This constructor initializes a new instance of the BJ class and starts the game for the specified player.

Parameters

player	The player object for whom the game is being started.

Definition at line 17 of file BJ.cpp.

Here is the call graph for this function:



4.1.2.2 ~BJ()

```
BJ::~BJ ()
```

Definition at line 22 of file BJ.cpp.

4.1.3 Member Function Documentation

4.1.3.1 calculateScore()

```
void BJ::calculateScore (
            Hand & unknown)
```

Definition at line 207 of file BJ.cpp.

4.1.3.2 dealersMove()

```
void BJ::dealersMove (
            int cards[],
            Hand & client,
            Hand & dealer)
```

Definition at line 248 of file BJ.cpp.

4.1.3.3 giveCards()

```
void BJ::giveCards (
            int cards[],
            Hand & client,
            Hand & dealer)
```

Definition at line 127 of file BJ.cpp.

4.1.3.4 printCard()

```
void BJ::printCard (
            int card)
```

Definition at line 172 of file BJ.cpp.

4.1.3.5 rndCard()

```
void BJ::rndCard (
            int allCards[],
            Hand & unknown)
```

Definition at line 135 of file BJ.cpp.

4.1.3.6 showCards()

```
void BJ::showCards (
            Hand & client,
            Hand & dealer)
```

Definition at line 150 of file BJ.cpp.

4.1.3.7 startGame()

```
void BJ::startGame (
            Player & player)
```

Starts the BlackJack game.

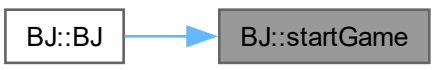
This function allows the player to play the BlackJack game. It takes a reference to a Player object as a parameter. The function prompts the player to enter their bet, deals the cards, and allows the player to make choices (hit or stand). After the player's turn, the function determines the outcome of the game and updates the player's balance accordingly. The game continues until the player runs out of balance or chooses to quit. At the end of the game, the function displays the total wins and losses of the player.

Parameters

player	The Player object representing the player.

Definition at line 37 of file BJ.cpp.

Here is the caller graph for this function:



The documentation for this struct was generated from the following files:

- BJ.h
- BJ.cpp

4.2 Dice Struct Reference

Public Member Functions

- Dice (Player &player)

  Constructs a new instance of the Dice class.
- void startGame (Player &player)

4.2.1 Detailed Description

Definition at line 4 of file Dice.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Dice()

```
Dice::Dice (
            Player & player)
```

Constructs a new instance of the Dice class.

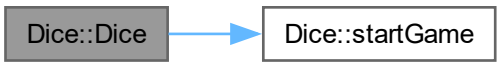
This constructor initializes a new instance of the Dice class and starts the game for the specified player.

Parameters

player	The player object for whom the game is being started.

Definition at line 16 of file Dice.cpp.

Here is the call graph for this function:



4.2.2.2 ~Dice()

```
Dice::~Dice ()
```

Definition at line 20 of file Dice.cpp.

4.2.3 Member Function Documentation

4.2.3.1 startGame()

```
void Dice::startGame (
            Player & player)
```

Starts the dice game.

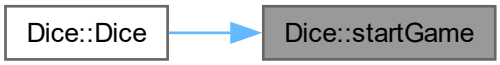
This function allows the player to play the dice game. It takes a reference to a Player object as a parameter. The player is prompted to enter their bet and choose what to bet on (red, blue, or tie). The dice are rolled and the outcome is determined. If the player wins, their balance is increased and the winnings are displayed. If the player loses, their balance is decreased and a message is displayed. The player can choose to play again or quit the game.

Parameters

player	The player object for whom the game is being started.

Definition at line 34 of file Dice.cpp.

Here is the caller graph for this function:



The documentation for this struct was generated from the following files:

- Dice.h
- Dice.cpp

4.3 DorN Struct Reference

Public Member Functions

- DorN (Player &player)

  Constructs a new instance of the DorN (Double or nothing) class.
- void startGame (Player &player)

4.3.1 Detailed Description

Definition at line 7 of file DorN.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 DorN()

```
DorN::DorN (
            Player & player)
```

Constructs a new instance of the DorN (Double or nothing) class.

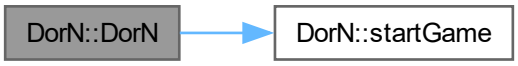
This constructor initializes a new instance of the DorN class and starts the game for the specified player.

Parameters

player	The player object for whom the game is being started.

Definition at line 16 of file DorN.cpp.

Here is the call graph for this function:



4.3.2.2 ~DorN()

```
DorN::~DorN ()
```

Definition at line 21 of file DorN.cpp.

4.3.3 Member Function Documentation

4.3.3.1 startGame()

```
void DorN::startGame (
            Player & player)
```

Starts the Double or Nothing game.

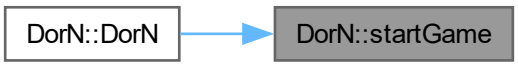
This function allows the player to play the Double or Nothing game. The player is prompted to enter their bet and choose heads or tails. The outcome is determined randomly, and the player's balance is updated accordingly. The game continues until the player's balance reaches zero or the player chooses to exit. At the end of the game, the total number of wins and losses is displayed.

Parameters

player	The player object for whom the game is being started.

Definition at line 36 of file DorN.cpp.

Here is the caller graph for this function:



The documentation for this struct was generated from the following files:

- DorN.h
- DorN.cpp

4.4 Hand Class Reference

Public Member Functions

- Hand ()

  Constructs a new instance of the Hand class. used to keep track of the player's hand in the game of BlackJack.

Public Attributes

- int nCards [10]
- int aces
- int score
- int altScore
- int n
- bool end

4.4.1 Detailed Description

Definition at line 6 of file Hand.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Hand()

```
Hand::Hand ()
```

Constructs a new instance of the Hand class. used to keep track of the player's hand in the game of BlackJack.

Definition at line 7 of file Hand.cpp.

4.4.3 Member Data Documentation

4.4.3.1 aces

```
int Hand::aces
```

Definition at line 10 of file Hand.h.

4.4.3.2 altScore

```
int Hand::altScore
```

Definition at line 12 of file Hand.h.

4.4.3.3 end

```
bool Hand::end
```

Definition at line 14 of file Hand.h.

4.4.3.4 n

```
int Hand::n
```

Definition at line 13 of file Hand.h.

4.4.3.5 nCards

```
int Hand::nCards[10]
```

Definition at line 9 of file Hand.h.

4.4.3.6 score

```
int Hand::score
```

Definition at line 11 of file Hand.h.

The documentation for this class was generated from the following files:

- Hand.h
- Hand.cpp

4.5 Login Struct Reference

Public Member Functions

- tuple< std::string, double, int, int > getPlayer ()
  Reads the player data from the CSV file.
- void changeCSV (double bet, int additionW, int additionL)
- void printList ()

4.5.1 Detailed Description

Definition at line 10 of file Login.h.

4.5.2 Member Function Documentation

4.5.2.1 changeCSV()

```
void Login::changeCSV (
            double bet,
            int additionW,
            int additionL)
```

Changes the CSV according to the input of +- bet which reduces or increases the Player's account balance. Additionaly increments wins or loses in the csv file.

Definition at line 71 of file Login.cpp.

Here is the call graph for this function:

4.5.2.2 getPlayer()

```
tuple< std::string, double, int, int > Login::getPlayer ()
```

Reads the player data from the CSV file.

This function reads the player data from the CSV file and stores it in the respective vectors. the Player is prompted to enter their username, if the username is not in the names vector the code repeats until a recognised username is entered.

Definition at line 17 of file Login.cpp.

4.5.2.3 printList()

```
void Login::printList ()
```

Replaces the current csv file with an updated one using vectors of names, balance, wins and losses.

Definition at line 82 of file Login.cpp.

Here is the caller graph for this function:

The documentation for this struct was generated from the following files:

- Login.h
- Login.cpp

4.6 Player Class Reference

Public Member Functions

- Player ()
  Player class to store player data of balance, wins and losses.
- Player (std::string initialName, int initialBalance, int initialWins, int initialLosses)
- int getBalance () const
- int getWins () const
- int getLosses () const
- const std::string & getName () const
- void setName (const std::string &newName)
- void setBalance (int newBalance)
- void setWins (int newWins)
- void setLosses (int newLosses)
- void increaseBalance (int amount)
- void decreaseBalance (int amount)
- void incrementWins ()
- void incrementLosses ()

4.6.1 Detailed Description

Definition at line 6 of file Player.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 Player() [1/2]

```
Player::Player ()
```

Player class to store player data of balance, wins and losses.

Definition at line 7 of file Player.cpp.

4.6.2.2 Player() [2/2]

```
Player::Player (
            std::string initialName,
            int initialBalance,
            int initialWins,
            int initialLosses)
```

Definition at line 10 of file Player.cpp.

4.6.3 Member Function Documentation

4.6.3.1 decreaseBalance()

```
void Player::decreaseBalance (
            int amount)
```

Definition at line 58 of file Player.cpp.

4.6.3.2 getBalance()

```
int Player::getBalance () const
```

Definition at line 13 of file Player.cpp.

4.6.3.3 getLosses()

```
int Player::getLosses () const
```

Definition at line 23 of file Player.cpp.

4.6.3.4 getName()

```
const std::string & Player::getName () const
```

Definition at line 28 of file Player.cpp.

4.6.3.5 getWins()

```
int Player::getWins () const
```

Definition at line 18 of file Player.cpp.

4.6.3.6 increaseBalance()

```
void Player::increaseBalance (
            int amount)
```

Definition at line 53 of file Player.cpp.

4.6.3.7 incrementLosses()

```
void Player::incrementLosses ()
```

Definition at line 68 of file Player.cpp.

4.6.3.8 incrementWins()

```
void Player::incrementWins ()
```

Definition at line 63 of file Player.cpp.

4.6.3.9 setBalance()

```
void Player::setBalance (
            int newBalance)
```

Definition at line 38 of file Player.cpp.

4.6.3.10 setLosses()

```
void Player::setLosses (
            int newLosses)
```

Definition at line 48 of file Player.cpp.

4.6.3.11 setName()

```
void Player::setName (
            const std::string & newName)
```

Definition at line 33 of file Player.cpp.

4.6.3.12 setWins()

```
void Player::setWins (
            int newWins)
```

Definition at line 43 of file Player.cpp.

The documentation for this class was generated from the following files:

- Player.h
- Player.cpp

Chapter 5

File Documentation

5.1 BJ.cpp

```
00001 #include <iostream>
00002 #include <cstdlib>
00003 #include <ctime>
00004 #include "Player.h"
00005 #include "BJ.h"
00006 #include "Hand.h"
00007
00008 using namespace std;
00009
00017 BJ::BJ(Player& player)
00018 {
00019     startGame(player);
00020 }
00021
00022 BJ::~BJ()
00023 {
00024 }
00025
00037 void BJ::startGame(Player& player)
00038 {
00039     string choice;
00040     int bet;
00041     int cards[52];//All cards
00042     system("chcp 65001");
00043     system("cls");
00044     cout « "Welcome to the BlackJack game!" « endl;
00045     cout « "Your initial balance is: $" « player.getBalance() « endl;
00046     while (player.getBalance() > 0)
00047     {
00048         Hand client;
00049         Hand dealer;
00050         cout « "\nEnter your bet: $";
00051         cin » bet;
00052         system("cls");
00053         cout « endl;
00054         if (bet > player.getBalance())
00055         {
00056             cout « "You don't have enough balance to make this bet. Try again." « endl;
00057             continue;
00058         }
00059         giveCards(cards, client, dealer);
00060         while (1)
00061         {
00062             showCards(client, dealer);
00063             if (client.score > 21)
00064             {
00065                 client.end = true;
00066                 break;
00067             }
00068             cout « "Make a choice:\n";
00069             cout « "1. Hit\n";
00070             cout « "2. Stand\n";
00071             cin » choice;
00072             cout « endl;
00073             if (choice == "hit" || choice == "Hit" || choice == "1") rndCard(cards, client);
00074             else if (choice == "stand" || choice == "Stand" || choice == "2")
00075             {
00076                 client.end = true;
```

```
00077                 break;
00078             }
00079             else cout « "Invalid choice. Please enter 'hit', '1' or 'stand', '2'.\n" « endl;
00080         }
00081         if (client.score > 21)
00082         {
00083             player.decreaseBalance(bet);
00084             player.incrementLosses();
00085             rndCard(cards, dealer);
00086             showCards(client, dealer);
00087             cout « "\nSorry, you lost. Your new balance is: $" « player.getBalance() « endl;
00088             cout « endl;
00089         }
00090         else
00091         {
00092             dealersMove(cards, client, dealer);
00093             if (client.end == true)
00094             {
00095                 if (dealer.end == true)
00096                 {
00097                     player.decreaseBalance(bet);
00098                     player.incrementLosses();
00099                     cout « "Sorry, you lost. Your new balance is: $" « player.getBalance() « endl;
00100                     cout « endl;
00101                 }
00102                 else
00103                 {
00104                     player.increaseBalance(bet);
00105                     player.incrementWins();
00106                     cout « "Congratulations! You won. +" « bet « "$!\nYour new balance is: $" «
      player.getBalance() « endl;
00107                     cout « endl;
00108                 }
00109             }
00110             else
00111             {
00112                 cout « "It's a tie! Your balance remains the same: $" « player.getBalance()« endl;
00113                 cout « endl;
00114             }
00115         }
00116         if (player.getBalance() == 0)
00117         {
00118             cout « "Game over! You've run out of balance. :3" « endl;
00119             break;
00120         }
00121     }
00122
00123     cout « "Thank you for playing!" « endl;
00124     cout « "Total Wins: " « player.getWins() « endl;
00125     cout « "Total Losses: " « player.getLosses() « endl;
00126 }
00127 void BJ::giveCards(int cards[], Hand& client, Hand& dealer)
00128 {
00129     for (int i = 0; i < 52; i++) cards[i] = 0;
00130
00131     rndCard(cards, client);
00132     rndCard(cards, client);
00133     rndCard(cards, dealer);
00134 }
00135 void BJ::rndCard(int allCards[], Hand& unknown)
00136 {
00137     int card;
00138     while (1)
00139     {
00140         card = rand() % 52;
00141         if (allCards[card] == 0)
00142         {
00143             allCards[card] = 1;
00144             unknown.nCards[unknown.n] = card;
00145             unknown.n++;
00146             break;
00147         }
00148     }
00149 }
00150 void BJ::showCards(Hand& client, Hand& dealer)
00151 {
00152     system("cls");
00153     cout « "Dealers's cards:\n";
00154     for (int i = 0; i < dealer.n; i++)
00155     {
00156         printCard(dealer.nCards[i]);
00157         cout « " ";
00158     }
00159     if (client.end == false)cout « "X" « endl;
00160     else cout « endl;
00161     calculateScore(dealer);
00162     cout « "Your cards:\n";
```

```
00163     for (int i = 0; i < client.n; i++)
00164     {
00165         printCard(client.nCards[i]);
00166         cout « " ";
00167     }
00168     cout « endl;
00169     calculateScore(client);
00170     cout « endl;
00171 }
00172 void BJ::printCard(int card)
00173 {
00174     if (card / 13 == 0)//
00175     {
00176         if (card % 13 < 8) cout « "" « card % 13 + 2;
00177         else if (card % 13 == 9) cout « "J";
00178         else if (card % 13 == 10) cout « "Q";
00179         else if (card % 13 == 11) cout « "K";
00180         else cout « "A";
00181     }
00182     else if (card / 13 == 1)//
00183     {
00184         if (card % 13 < 8) cout « "" « card % 13 + 2;
00185         else if (card % 13 == 9) cout « "J";
00186         else if (card % 13 == 10) cout « "Q";
00187         else if (card % 13 == 11) cout « "K";
00188         else cout « "A";
00189     }
00190     else if (card / 13 == 2)//
00191     {
00192         if (card % 13 < 8) cout « "" « card % 13 + 2;
00193         else if (card % 13 == 9) cout « "J";
00194         else if (card % 13 == 10) cout « "Q";
00195         else if (card % 13 == 11) cout « "K";
00196         else  cout « "A";
00197     }
00198     else //
00199     {
00200         if (card % 13 < 8) cout « "" « card % 13 + 2;
00201         else if (card % 13 == 9) cout « "J";
00202         else if (card % 13 == 10) cout « "Q";
00203         else if (card % 13 == 11) cout « "K";
00204         else  cout « "A";
00205     }
00206 }
00207 void BJ::calculateScore(Hand& unknown)
00208 {
00209     unknown.score = 0;
00210     unknown.altScore = 0;
00211     unknown.aces = 0;
00212     for (int i = 0; i < unknown.n; i++)
00213     {
00214         int card = unknown.nCards[i];
00215         if (card % 13 < 8)
00216         {
00217             unknown.score += (card % 13) + 2;
00218             unknown.altScore += (card % 13) + 2;
00219         }
00220         else if (card % 13 == 12)
00221         {
00222             unknown.altScore += 11;
00223             unknown.score += 1;
00224             unknown.aces++;
00225         }
00226         else
00227         {
00228             unknown.score += 10;
00229             unknown.altScore += 10;
00230         }
00231     }
00232     while (1)
00233     {
00234         if (unknown.aces > 0)
00235         {
00236             if (unknown.altScore > 21)
00237             {
00238                 unknown.aces--;
00239                 unknown.altScore -= 10;
00240             }
00241             else break;
00242         }
00243         break;
00244     }
00245     if (unknown.aces == 0) cout « "Score: " « unknown.score « endl;
00246     else cout « "Score: " « unknown.score«"/"« unknown.altScore « endl;
00247 }
00248 void BJ::dealersMove(int cards[], Hand& client, Hand& dealer)
00249 {
```

```
00250     rndCard(cards, dealer);
00251     showCards(client, dealer);
00252     if (client.altScore<22 && client.altScore>client.score) client.score = client.altScore;
00253     while (dealer.score < 17 && (dealer.altScore < client.score) && (dealer.score < client.score))
00254     {
00255         rndCard(cards, dealer);
00256         showCards(client, dealer);
00257         if (client.altScore<22 && client.altScore>client.score) client.score = client.altScore;
00258     }
00259     if (dealer.altScore<22 && dealer.altScore>dealer.score) dealer.score = dealer.altScore;
00260     if(dealer.score>21) {}
00261     else if (dealer.score > client.score)
00262     {
00263         dealer.end = true;
00264     }
00265     else if (dealer.score == client.score)
00266     {
00267         client.end = false;
00268     }
00269 }
00270
00271
00272
00273
```

5.2 BJ.h

```
00001 #pragma once
00002 #include <iostream>
00003 #include <cstdlib>
00004 #include <ctime>
00005 #include "Player.h"
00006 #include "Hand.h"
00007
00008 struct BJ
00009 {
00010 public:
00011     BJ(Player& player);
00012     ~BJ();
00013     void startGame(Player& player);
00014     void rndCard(int[], Hand&);
00015     void giveCards(int[], Hand&, Hand&);
00016     void showCards(Hand&, Hand&);
00017     void printCard(int);
00018     void calculateScore(Hand&);
00019     void dealersMove(int[],Hand&, Hand&);
00020 };
00021 #pragma once
```

5.3 Dice.cpp

```
00001 #include "Dice.h"
00002 #include "Player.h"
00003 #include <iostream>
00004 #include <cstdlib>
00005 #include <ctime>
00006
00007 using namespace std;
00008
00016 Dice::Dice(Player& player){
00017     startGame(player);
00018 }
00019
00020 Dice::~Dice(){
00021 }
00022
00034 void Dice::startGame(Player& player){
00035     int bet;
00036     string choice;
00037     bool done = false;
00038
00039     cout « "\nWelcome to the dice game!" « endl;
00040     cout « "Your initial balance is: $" « player.getBalance() « endl;
00041     while (player.getBalance() > 0 && !done)
00042     {
00043         cout « "\nEnter your bet: $";
00044         cin » bet;
00045         if (bet > player.getBalance())
00046         {
```

```
00047             cout « "You don't have enough balance to make this bet. Try again." « endl;
00048             continue;
00049         }
00050         cout « "Choose what to bet on: red, blue or tie:" « endl;
00051         cin » choice;
00052 
00053         int red = rand() % 6 + 1;
00054         int blue = rand() % 6 + 1;
00055         cout « "The dice rolled: red(" « red « ") and blue(" « blue « ")" « endl;
00056         if (red == blue && choice == "tie")
00057         {
00058             player.increaseBalance(bet * 5);
00059             player.incrementWins();
00060             cout « "Congratulations! You won. +" « bet * 5 « "$!\nYour new balance is: $" «
      player.getBalance() « endl;
00061         }
00062         else if (red > blue && choice == "red")
00063         {
00064             player.increaseBalance(bet);
00065             player.incrementWins();
00066             cout « "Congratulations! You won. +" « bet « "$!\nYour new balance is: $" «
      player.getBalance() « endl;
00067         }
00068         else if (red < blue && choice == "blue")
00069         {
00070             player.increaseBalance(bet);
00071             player.incrementWins();
00072             cout « "Congratulations! You won. +" « bet « "$!\nYour new balance is: $" «
      player.getBalance() « endl;
00073         }
00074         else
00075         {
00076             player.decreaseBalance(bet);
00077             player.incrementLosses();
00078             cout « "Sorry, you lost. Your new balance is: $" « player.getBalance() « endl;
00079         }
00080         if (player.getBalance() == 0)
00081         {
00082             cout « "You have no more balance to play with." « endl;
00083             break;
00084         }
00085 
00086         cout « "Do you want to play again? (yes/no): ";
00087         while(1){
00088             string answer;
00089             cin » answer;
00090             if(answer == "yes"){
00091                 done = false;
00092                 break;
00093             }else if(answer == "no"){
00094                 done = true;
00095                 break;
00096             }else {
00097                 cout « "Invalid input. Type \"yes\" or \"no\": ";
00098             }
00099         }
00100     }
00101 }
```

5.4 Dice.h

```
00001 #include "Player.h"
00002 #include <iostream>
00003 
00004 struct Dice{
00005     public:
00006         Dice(Player& player);
00007         ~Dice();
00008 
00009         void startGame(Player& player);
00010 };
```

5.5 DorN.cpp

```
00001 #include <iostream>
00002 #include <cstdlib>
00003 #include <ctime>
00004 #include "Player.h"
00005 #include "DorN.h"
```

```
00006
00007 using namespace std;
00008
00016 DorN::DorN(Player& player)
00017 {
00018     startGame(player);
00019 }
00020
00021 DorN::~DorN()
00022 {
00023 }
00024
00036 void DorN::startGame(Player& player)
00037 {
00038     string choice;
00039     int bet;
00040     cout « "\nWelcome to the Double or Nothing game!" « endl;
00041     cout « "Your initial balance is: $" « player.getBalance() « endl;
00042     while (player.getBalance() > 0)
00043     {
00044         cout « "\nEnter your bet or -1 to exit: $\n";
00045         cin » bet;
00046         if (bet == -1){
00047             return;
00048         }
00049         else if (bet > player.getBalance())
00050         {
00051             cout « "You don't have enough balance to make this bet. Try again." « endl;
00052             continue;
00053         }
00054         cout « "Choose heads or tails:";
00055         cin » choice;
00056         int outcome = rand() % 2;
00057         if (choice == "heads")
00058         {
00059             if (outcome == 1)
00060             {
00061                 player.increaseBalance(bet);
00062                 player.incrementWins();
00063                 cout « "Congratulations! You won. +" « bet « "$!\nYour new balance is: $" «
      player.getBalance() « endl;
00064             }
00065             else
00066             {
00067                 player.decreaseBalance(bet);
00068                 player.incrementLosses();
00069                 cout « "Sorry, you lost. Your new balance is: $" « player.getBalance() « endl;
00070             }
00071         }
00072         else if (choice == "tails")
00073         {
00074             if (outcome == 0)
00075             {
00076                 player.increaseBalance(bet);
00077                 player.incrementWins();
00078                 cout « "Congratulations! You won. +" « bet « "$!\nYour new balance is: $" «
      player.getBalance() « endl;
00079
00080             }
00081             else
00082             {
00083                 player.decreaseBalance(bet);
00084                 player.incrementLosses();
00085                 cout « "Sorry, you lost. Your new balance is: $" « player.getBalance() « endl;
00086             }
00087         }
00088         else
00089         {
00090             cout « "Invalid choice. Please enter 'heads' or 'tails'." « endl;
00091         }
00092         if (player.getBalance() == 0)
00093         {
00094             cout « "Game over! You've run out of balance." « endl;
00095             break;
00096         }
00097     }
00098
00099     cout « "Thank you for playing!" « endl;
00100     cout « "Total Wins: " « player.getWins() « endl;
00101     cout « "Total Losses: " « player.getLosses() « endl;
00102 }
```

5.6 DorN.h

```
00001 #pragma once
00002 #include <iostream>
00003 #include <cstdlib>
00004 #include <ctime>
00005 #include "Player.h"
00006
00007 struct DorN
00008 {
00009 public:
00010     DorN(Player& player);
00011     ~DorN();
00012     void startGame(Player& player);
00013 };
```

5.7 Hand.cpp

```
00001 #include "Hand.h"
00002 #include <string>
00007 Hand::Hand() {
00008     aces = 0;
00009     score = 0;
00010     altScore = 0;
00011     n = 0;
00012     end = false;
00013     for (int i = 0; i < 10; i++) nCards[i] = -1;
00014 }
```

5.8 Hand.h

```
00001 #ifndef HAND_H
00002 #define HAND_H
00003
00004 #include <string>
00005
00006 class Hand{
00007 public:
00008     Hand();
00009     int nCards[10];
00010     int aces;
00011     int score;
00012     int altScore;
00013     int n;
00014     bool end;
00015 };
00016
00017 #endif
```

5.9 Login.cpp

```
00001 #include "Login.h"
00002 #include <fstream>
00003 #include <iostream>
00004 #include <sstream>
00005 #include <vector>
00006 #include <tuple>
00007 using namespace std;
00008
00017 tuple<std::string, double, int, int> Login::getPlayer()
00018 {
00019     string name;
00020     ifstream fin("playerData.csv");
00021     if (!fin)
00022     {
00023         cerr « "Failed to open file" « endl;
00024         exit(-1);
00025     }
00026
00027     string line;
00028     getline(fin, line);
00029
00030     while (getline(fin, line))
00031     {
00032         stringstream ss(line);
```

```
00033         string token;
00034         vector<string> data;
00035         while (getline(ss, token, ','))
00036         {
00037             data.push_back(token);
00038         }
00039         if (data.size() == 4)
00040         {
00041             names.push_back(data[0]);
00042             balance.push_back(stod(data[1]));
00043             wins.push_back(stoi(data[2]));
00044             losses.push_back(stoi(data[3]));
00045             counter++;
00046         }
00047     }
00048 
00049     while(true){
00050         cout « "Enter your username:" « endl;
00051         cin » name;
00052 
00053         for(int i = 0; i < counter; i++){
00054             if(names[i] == name){
00055                 fin.close();
00056                 cout « "User found!" « endl;
00057                 playerNum = i;
00058                 return std::make_tuple(names[i], balance[i], wins[i], losses[i]);
00059             }
00060         }
00061         cout « "User not found. try again" « endl;
00062     }
00063     exit(-1);
00064 }
00065 
00071 void Login::changeCSV(double bet, int additionW, int additionL){
00072     balance[playerNum] = bet;
00073     wins[playerNum] = additionW;
00074     losses[playerNum] = additionL;
00075     printList();
00076 }
00077 
00082 void Login::printList(){
00083     fstream fin, fout;
00084     fout.open("Temp.csv", ios::out);
00085     fin.open("playerData.csv", ios::in);
00086     fout « "Player,Balance,Wins,Loses" « endl;
00087     for(int i = 0; i < counter; i++){
00088         fout « names[i] « "," « balance[i] « "," « wins[i] « "," « losses[i] « endl;
00089     }
00090     fin.close();
00091     fout.close();
00092 
00093     remove("playerData.csv");
00094     rename("Temp.csv", "playerData.csv");
00095 }
```

5.10 Login.h

```
00001 #pragma once
00002 #include <iostream>
00003 #include <cstdlib>
00004 #include <ctime>
00005 #include <vector>
00006 #include "Player.h"
00007 
00008 using namespace std;
00009 
00010 struct Login
00011 {
00012 private:
00013     int counter = 0;
00014     int playerNum;
00015     vector<string> names;
00016     vector<double> balance;
00017     vector<int> wins;
00018     vector<int> losses;
00019 public:
00020     tuple<std::string, double, int, int> getPlayer();
00021     void changeCSV(double bet, int additionW, int additionL);
00022     void printList();
00023 };
```

5.11 main.cpp

```
00001 #include <iostream>
00002 #include <cstdlib>
00003 #include <ctime>
00004 #include <tuple>
00005 #include <vector>
00006 #include "Player.h"
00007 #include "DorN.h"
00008 #include "Dice.h"
00009 #include "Login.h"
00010 #include "BJ.h"
00011
00012
00013 using namespace std;
00017 int main()
00018 {
00019     int selection;
00020     srand(time(0));
00021     Login login;
00022     auto nbwl = login.getPlayer();
00023     Player player(get<0>(nbwl), get<1>(nbwl), get<2>(nbwl), get<3>(nbwl));
00024     while(true){
00025         cout « "\nWelcome back " « player.getName() « "! Select your game:" « endl;
00026         cout « "1. Double or Nothing\n"
00027             « "2. BlackJack\n"
00028             « "3. Poker\n"
00029             « "4. Dice\n"
00030             « "5. Roullete\n"
00031             « "6. Get your balance report" « endl;
00032         cin » selection;
00033         if(selection == 1){DorN dorn(player); login.changeCSV(player.getBalance(), player.getWins(),
      player.getLosses()); }
00034         else if(selection == 2){BJ bj(player);login.changeCSV(player.getBalance(), player.getWins(),
      player.getLosses());}
00035         else if(selection == 3){}
00036         else if(selection == 4){Dice dice(player); login.changeCSV(player.getBalance(),
      player.getWins(), player.getLosses());}
00037         else if(selection == 5){}
00038         else if(selection == 6){cout « "\nYour balance is: " « player.getBalance() « "$\n";}
00039         else {cout « "Bad selection" « endl;};
00040     }
00041
00042     return 0;
00043 }
```

5.12 Player.cpp

```
00001 #include "Player.h"
00002 #include <string>
00003
00007 Player::Player()
00008     : balance(1000), wins(0), losses(0) {}
00009
00010 Player::Player(std::string initialName, int initialBalance, int initialWins, int initialLosses)
00011     : name(initialName), balance(initialBalance), wins(initialWins), losses(initialLosses) {}
00012
00013 int Player::getBalance() const
00014 {
00015     return balance;
00016 }
00017
00018 int Player::getWins() const
00019 {
00020     return wins;
00021 }
00022
00023 int Player::getLosses() const
00024 {
00025     return losses;
00026 }
00027
00028 const std::string& Player::getName() const
00029 {
00030     return name;
00031 }
00032
00033 void Player::setName(const std::string& newName)
00034 {
00035     name = newName;
00036 }
00037
00038 void Player::setBalance(int newBalance)
```

```
00039 {
00040     balance = newBalance;
00041 }
00042
00043 void Player::setWins(int newWins)
00044 {
00045     wins = newWins;
00046 }
00047
00048 void Player::setLosses(int newLosses)
00049 {
00050     losses = newLosses;
00051 }
00052
00053 void Player::increaseBalance(int amount)
00054 {
00055     balance += amount;
00056 }
00057
00058 void Player::decreaseBalance(int amount)
00059 {
00060     balance -= amount;
00061 }
00062
00063 void Player::incrementWins()
00064 {
00065     wins++;
00066 }
00067
00068 void Player::incrementLosses()
00069 {
00070     losses++;
00071 }
```

5.13 Player.h

```
00001 #ifndef PLAYER_H
00002 #define PLAYER_H
00003
00004 #include <string>
00005
00006 class Player {
00007 public:
00008     Player();
00009     Player(std::string initialName, int initialBalance, int initialWins, int initialLosses);
00010
00011     int getBalance() const;
00012     int getWins() const;
00013     int getLosses() const;
00014     const std::string& getName() const;
00015
00016     void setName(const std::string& newName);
00017     void setBalance(int newBalance);
00018     void setWins(int newWins);
00019     void setLosses(int newLosses);
00020
00021     void increaseBalance(int amount);
00022     void decreaseBalance(int amount);
00023     void incrementWins();
00024     void incrementLosses();
00025
00026 private:
00027     std::string name;
00028     int balance;
00029     int wins;
00030     int losses;
00031 };
00032
00033 #endif
```

Index

~BJ
    BJ, 8
~Dice
    Dice, 10
~DorN
    DorN, 12

aces
    Hand, 14
altScore
    Hand, 14

BJ, 7
    ~BJ, 8
    BJ, 7
    calculateScore, 8
    dealersMove, 8
    giveCards, 8
    printCard, 8
    rndCard, 9
    showCards, 9
    startGame, 9

calculateScore
    BJ, 8
Casino, 1
changeCSV
    Login, 15

dealersMove
    BJ, 8
decreaseBalance
    Player, 17
Dice, 10
    ~Dice, 10
    Dice, 10
    startGame, 11
DorN, 11
    ~DorN, 12
    DorN, 12
    startGame, 12

end
    Hand, 14

getBalance
    Player, 17
getLosses
    Player, 17
getName
    Player, 17
getPlayer
    Login, 15
getWins
    Player, 18
giveCards
    BJ, 8

Hand, 13
    aces, 14
    altScore, 14
    end, 14
    Hand, 14
    n, 14
    nCards, 14
    score, 14

increaseBalance
    Player, 18
incrementLosses
    Player, 18
incrementWins
    Player, 18

Login, 15
    changeCSV, 15
    getPlayer, 15
    printList, 16

n
    Hand, 14
nCards
    Hand, 14

Player, 16
    decreaseBalance, 17
    getBalance, 17
    getLosses, 17
    getName, 17
    getWins, 18
    increaseBalance, 18
    incrementLosses, 18
    incrementWins, 18
    Player, 17
    setBalance, 18
    setLosses, 18
    setName, 18
    setWins, 19
printCard
    BJ, 8
printList
    Login, 16

rndCard
BJ, 9

score
Hand, 14
setBalance
Player, 18
setLosses
Player, 18
setName
Player, 18
setWins
Player, 19
showCards
BJ, 9
startGame
BJ, 9
Dice, 11
DorN, 12