**Project 1 Roulette Game**



**By - Ria Shiv**

**Course - CIS 17C**

**Section – 48038**

**Date - 11/02/2016**

**Introduction**

This program is a digital version of roulette which is usually played in casinos. It is a very famous game played all year round. This game is very popular that people would want to enjoy it at home on their computers or laptops. Therefore, I coded a version of this game.

This is a betting game. The roulette table consists of cells from 1-36. Each cell has a number and a color(Red or Black) associated with it. On the side of the betting table, there is a ball rolling that generates number randomly. If your betting scenario meets the ball that appears you win.

There are four betting scenarios :-

* Odd and Even Bets – Bet if the ball is odd or even – returns double the money betted
* Black and Red Bets – Bet if the ball lands on the same color as the cell – returns double the money betted
* Single Number Bets – Bet on the exact number the ball will land – returns 37 times the money betted
* Row Bets- Bet if the ball will land on the 1st , 2nd or 3rd row – returns 3 times the money betted.

**Summary**

This program contains about 750 line of code. It has 5 header files and 6 cpp files including the main. This program has an extensive use of the STL library including lists, set, iterators and map. In addition, this program has an extensive use of the concepts of object oriented programming such as encapsulation, re usability of code and modular programming.

Some of the challenges, I faced was initially brainstorming on how to connect all the classes together to make them run into one big program. One of changeling parts was debugging and testing that the program works for all conditions. It took one and a half weeks to code the whole program and debug it to make sure its running.

**Code Description and Concepts**

The code starts with asking the user if he wants to play or quit. If the user says yes to playing then the roulette table is created with all the cells on the table and their colors represented by the letter R and B. Then the game options are shown. The user can pick either play himself or have the computer to play for him.

**Option 1: Play Himself-** When the user presses this option, cell map is made in the memory that contains information of each individual cell including value and the color it is. This uses the concepts of map. While creating each element of the map, an object of Cell was created which assigned the value and the color to the cell. This happens becaue the cell map is made in Game constructor. The user is prompted to enter his name and the balance he wants to put in his account. Then a sentinel loop is used to play the game as many times as the user wants to. First of all inside the loop it checks if the balance is not zero. If its zero it prompts cannot play game. Otherwise it goes for the playing part. The user is prompted to choose the type of bet they would want to do. Once the value is set in the betID, the user is prompted the money he wants to put on the bet. E.g even , red, 56 etc. The options are stored as a set in the memory. play() function, the betting money is subtracted from the balance. Then the ball is rolled using the game object to get the value of the cell on the wheel. If the balance goes below 10 then the program stops or if there 1000 iterations then also the program stops and the balance is printed at the end. Then the evaluateBet() function is called from within play(). The winning cell is sent as a parameter. In this an iterator, is used to evaluate bet on every option. An object of playerOption with each play. In playerOptions each option is stores the bet for a particular type. E.g Red black, odd even etc. It also evaluates the winning for bet amount and the option selected. e.g 10 on red. The money won or lost is evaluated back in playerPlay. The balance is updated. Each of the play is stored in a list in the memory and the show analytics function is called that prints a history of the whole play.

**Option 2: Computer play for him** – When you press on this option, a cell map is made in the memory that contains information of each individual cell including value and the color it is. This uses the concepts of map. **.**This happens becaue the cell map is made in Game constructor. While creating each element of the map, an object of Cell was created which assigned the value and the color to the cell. Then an object of Player is created. The constructor receives the argument of name, balance and Game object. In the computer play, the name is assigned “Ria” , balance is assigned 1000 and the game object declared initially is passed. Using the player play object the playComputer() function is called. A loop in run until i< 1000. The variable i is incremented in the program by i++. A modval variable is used that is i%4 hence it gives I values 0 1 2 3 based on each turn. This i is used for entering the choice on the type of bet. Based on the value of i, the variable choice is also assigned a random value to pic the bet option.

e.g if i = 0 then the bet is an Odd and Even type. Therefore choice is randomly assigned 1 or 2( odd or even).

The computerPlayerPlay() function is called that has three parameters the bet amount, the type of bet and the choice on the bet. All these values are assigned to the specific variables and returned to the playerPlay object. Then play() function called and playerPlay object is sent as an argument. In the play() function, the betting money is subtracted from the balance. Then the ball is rolled using the game object to get the value of the cell on the wheel. If the balance goes below 10 then the program stops or if there 1000 iterations then also the program stops and the balance is printed at the end. Then the evaluateBet() function is called from within play(). The winning cell is sent as a parameter. In this an iterator, is used to evaluate bet on every option. An object of playerOption with each play. In playerOptions each option is stores the bet for a particular type. E.g Red black, odd even etc. It also evaluates the winning for bet amount and the option selected. e.g 10 on red. The money won or lost is evaluated back in playerPlay. The balance is updated. Each of the play is stored in a list in the memory and the show analytics function is called that prints a history of the whole play.

**Flow Chart**

START

s

If gameOptions== 1

If gameOptions== 2

true

false

false

false

true

End

Declare and read gameOptions

Declare and read ch

playOption() is called

true

PrintGameTable() is called

If ch == ‘Y’

true

false

end

true

Balance<10

i = i+ 1

playerplayList is added at the end

Initialize player play object with the bet type, bet and wager

random number chosen (1,3 )

random number chosen (1, 36 )

random number chosen (1,2 )

random number chosen (1,2 )

true

true

true

true

If modVal == 3

false

false

false

If modVal == 2

If modVal == 1

If modVal == 0

While I <1000

Balance is updated according to the lost or won

moneyWon +=winningAmount

it != playerOptions.end()

playerPlay->evaluateBet is called

Game->rollball() is called and a random number is inserted in currentBallPosition

Subtract the amount of money betted

Winning = 0

isWinner == true

false

true

Winning = wager\*3

isWinner = cell->getValue() >=25 && cell->getValue() <= 36

isWinner = cell->getValue() >= 13 && cell->getValue() <= 24

If first row

isWinner = cell->getValue() >= 1 && cell->getValue() <= 12

If third row

false

If second row

false

false

Winning = 0

true

Winning = wager\*36

If cell value== betOption

false

isWinner = !cell->isEven

isWinner = cell->isEven

true

If even Bet

isWinner = !cell->isRed

Winning = 0

true

Winning = wager \*2

false

isWinner ==True

false

isWinner = cell->isRed

true

If red Bet

false

false

true

true

true

true

false

If betTypeId ==ROW

If betTypeId ==SINGLE

If betTypeId ==EVEN\_ODD

If betTypeId ==RED\_BLACK

The user is prompted to enter the wager for betting

The user is made to enter the bet type

Player object calls play()

If opt == ‘S’

false

false

End

true

false

true

true

If player->getBalance() < 1

If opt == ‘Y’

continueGame == true

Get the name and balance from the user

value after handling the exception

false

true

showAnalytics in the class playerOptions Is called which displays everything on the console

The object for playerOptions is created

it != playerOptions.end()

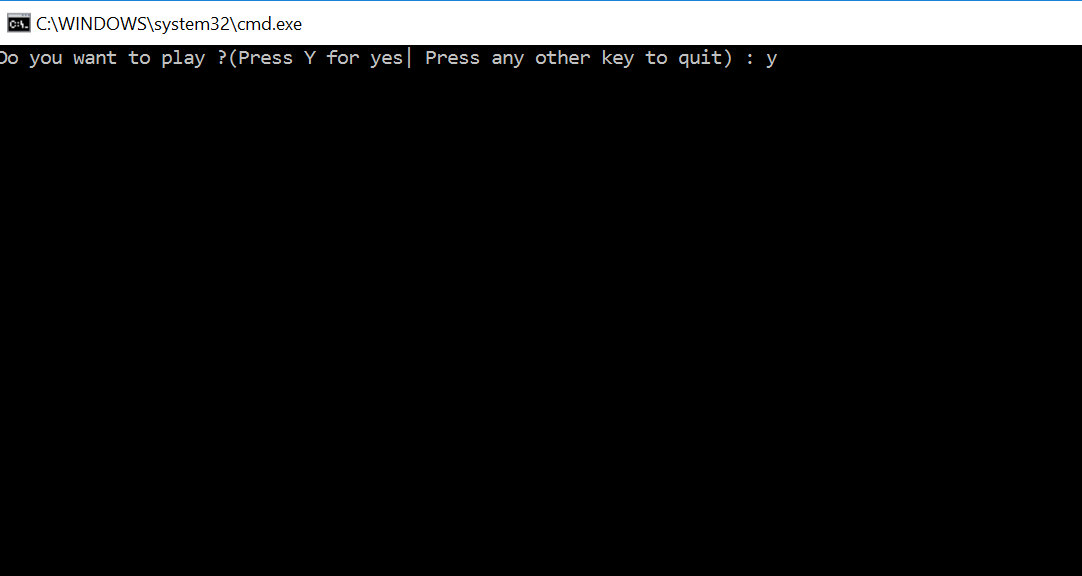
Then the option is prompted and the values for the bet are put in

The wager assigned a value for the bet

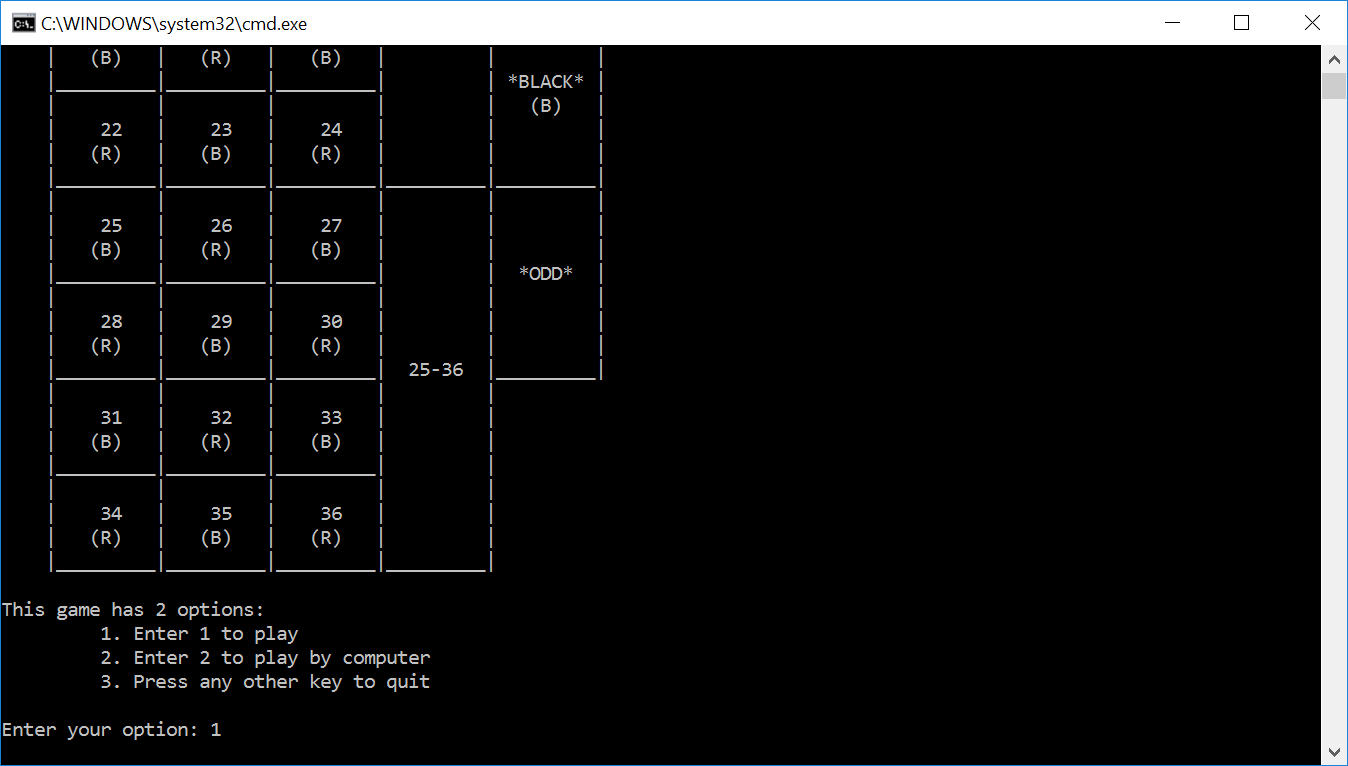
Display option is called and betOption is intialized with a value after handling the exception

**Outputs**

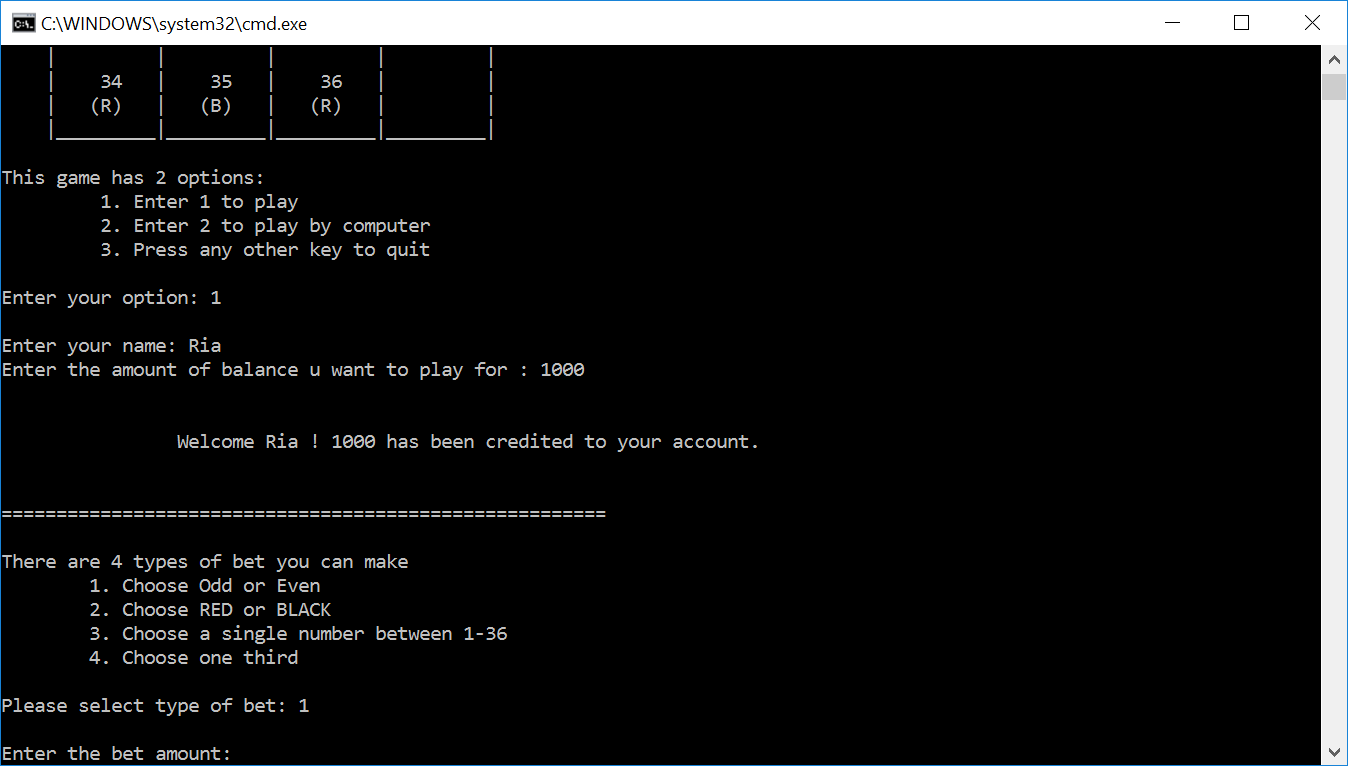
Ask the user if he/she wants to play



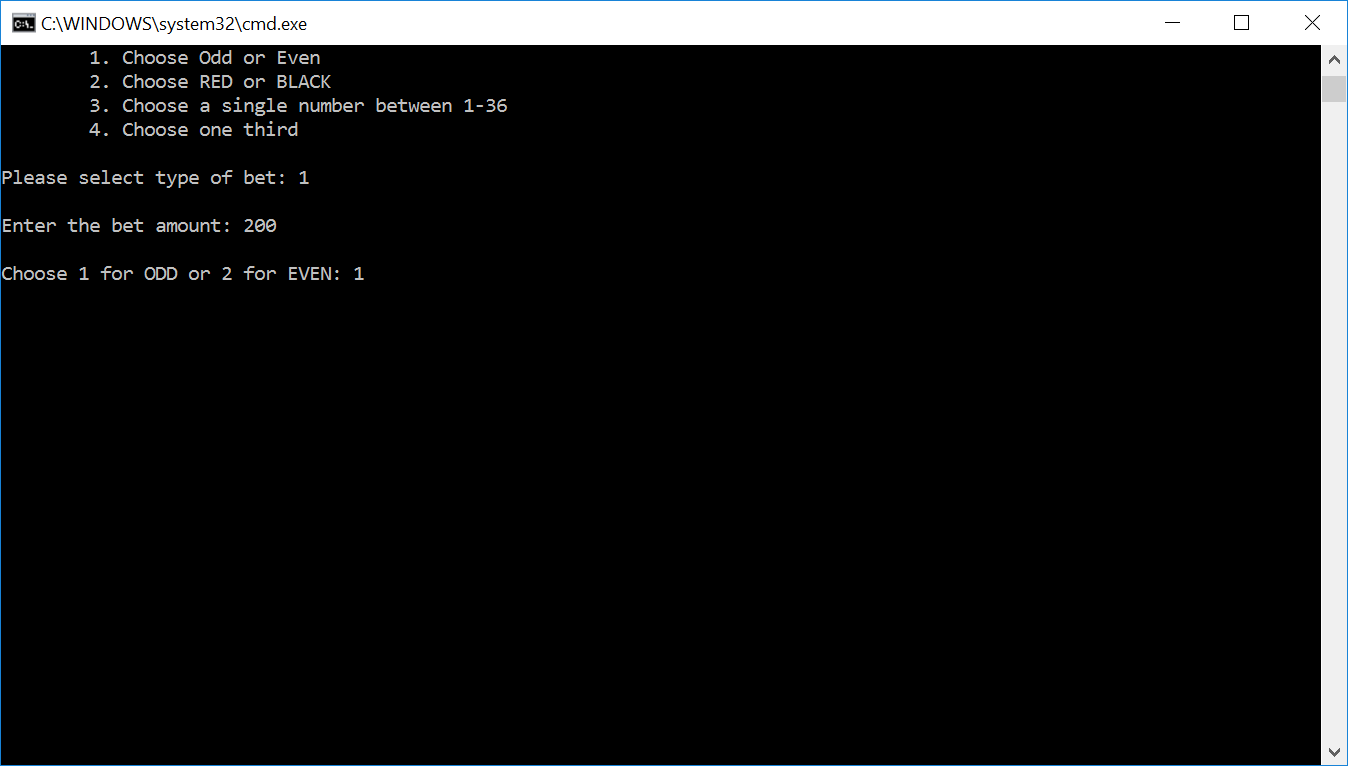
Prints out the roulette table and ask if the user wants to play or the user wants the computer to play. Press one for the user to play.



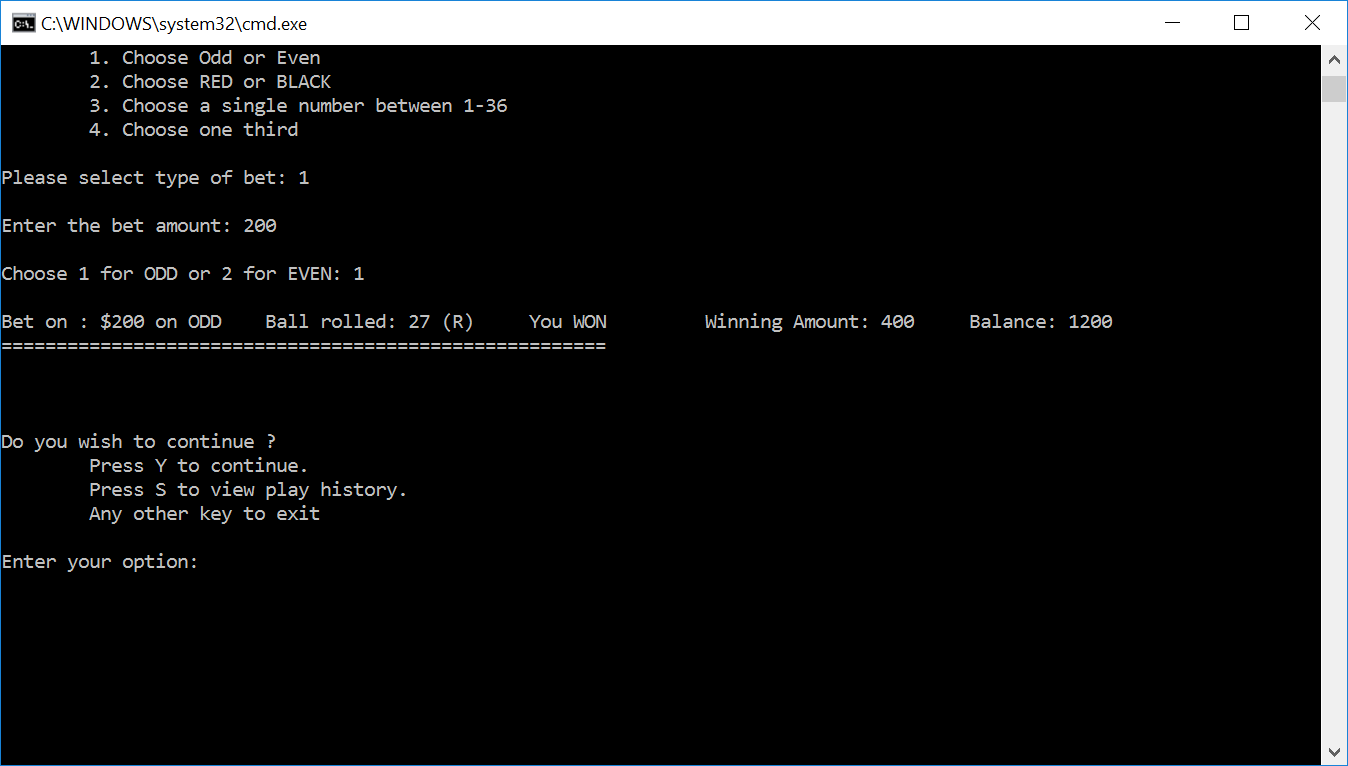
Ask the user to enter the name and the amount he wants in their balance ($1000 in my case). The user can choose from 4 types of betting scenarios. I chose one on ODD and Even.



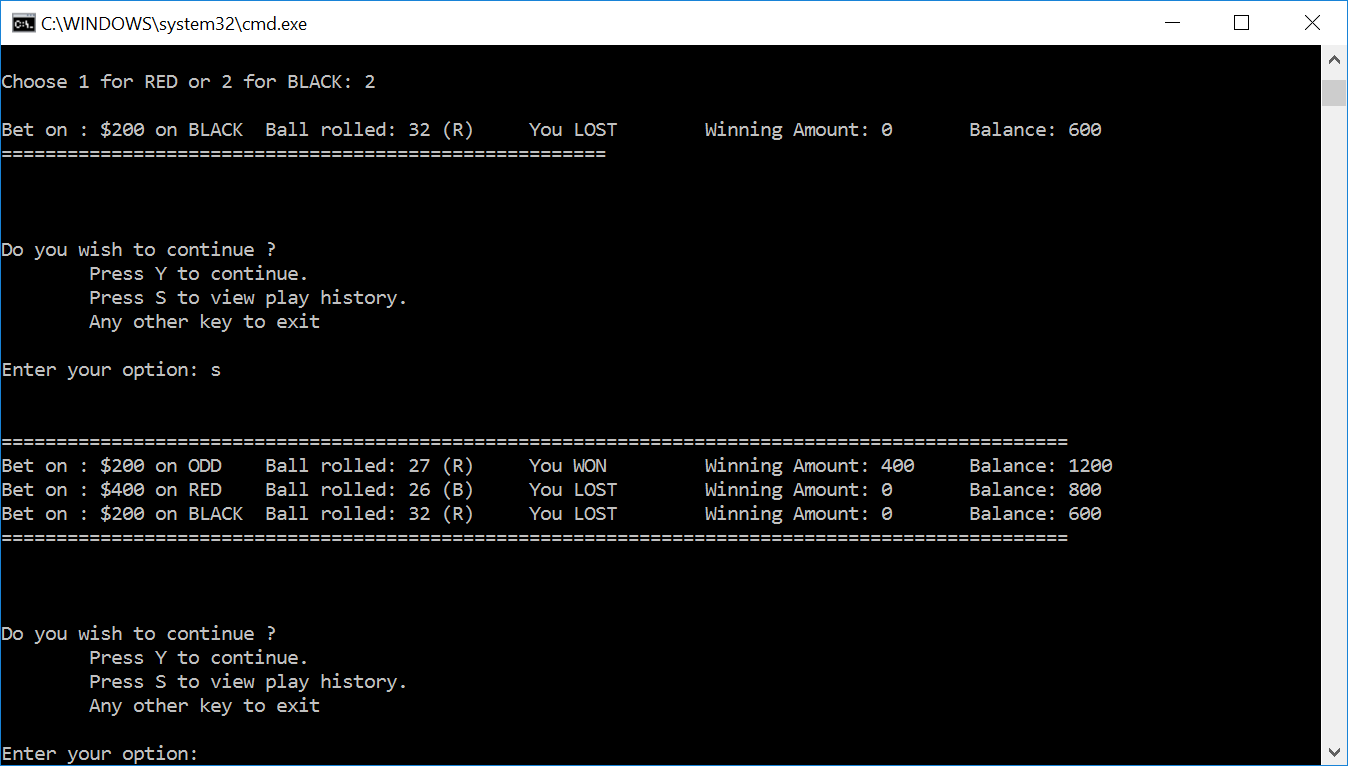
The betting amount is the amount the user wants to bet for on the particular scenario ($200). This cannot be beyond the balance, exception handled in case its beyond. The user enter 1 to bet that it would be ODD.



The randomly ball rolled lander on a 27 which made me win twice the amount of money as I betted and the information is displayed updating my balance. The menu below shows options in which the user can keep playing or can see the betting history



The following examples shows the betting history of 3 games.



**Major Variables**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Location** | **Description** |
| **game** | **Game** | **main.cpp** | **This is one of the main variables that is passed in the player object which rolls the ball** |
| **cellmap** | **map<int, Cell\*>** | **Game.h** | **This creates a cellMap in the memory** |
| **betTypeId** | **int** | **playerOption.h** | **It describes the type of Bet it is.** |
| **betOption** | **int** | **playerOption.h** | **It describes what are you betting on** |
| **WinningCell** | **Cell\*** | **PlayerPlay.h** | **It stores the information of the winning cell** |

**References**

1. TextBook
2. Cplusplus.com – for syntax and functionality of various files

**Code Documented**

**main.cpp**

/\*

\* Name : Ria Shiv

\* File : main.cpp

\* Date : 30th October 2016

\*/

//C++ libraies

#include <cstdlib>

#include <ctime>

#include <iostream>

//User made header files

#include "Game.h"

#include"Player.h"

#include"PlayerPlay.h"

#include"Cell.h"

using namespace std;

//global functions

bool playGame();

void playOptions();

void printGameTable();

void playComputer();

void play();

bool isQuit = false;

//main

int main(int argc, char\*\* argv) {

if (playGame()) //checks if user wants too play the game

{

printGameTable(); //prints the game table

playOptions(); //prints the various options to play

while (!isQuit) { //quits the game when the user wants it to end

}

}

else

{

cout << "THANK YOU!" << endl;

}

}

//Called when the user wants to play the game by himself

void play() {

string name;

int balance;

cout << "\nEnter your name: "; //get user name

while (name.empty()) {

getline(cin, name);

}

cout << "Enter the amount of balance u want to play for : "; //get user balance

cin >> balance;

cout << "\n\n\t\tWelcome " << name << "! " << balance << " has been credited to your account.\n";

Game \*game = new Game(); // game object is created

Player \*player = new Player(name, balance, game); //player option is created

bool continueGame = true;

char opt = 'Y';

while (continueGame) { // plays only until the user wants

if (opt == 'Y' || opt == 'y') {

if (player->getBalance() < 1) { //throws acception when the user is running out of money

cout << "\n\nSorry! You are running out of balance, game cannot be continue";

}

else {

player->play(); //called to play the game

}

}

else if (opt == 'S' || opt == 's') {

player->showAnalytics();

} // the analytics of the game are shown with past history

opt = 'N';

cout << "\n\nDo you wish to continue ? \n\tPress Y to continue. \n\tPress S to view play history. \n\tAny other key to exit\n\n";

cout << "Enter your option: ";

cin >> opt;

continueGame = opt == 'Y' || opt == 'y' || opt == 'S' || opt == 's';

isQuit = !continueGame;

}

}

//called when the computer wants to play

void playComputer() {

Game \*game = new Game(); //game object is created

Player \*player = new Player("Ria", 1000, game); // player object is created

player->playComputer(); // Computer object is created

}

//displays the options on what the user wants to take

void playOptions()

{

cout << "This game has 2 options:\n";

cout << "\t 1. Enter 1 to play\n";

cout << "\t 2. Enter 2 to play by computer\n";

cout << "\t 3. Press any other key to quit\n\n";

cout << "Enter your option: ";

int gameOption;

cin >> gameOption;

if (gameOption == 1) { // when the user wants to play

play();

}

else if (gameOption == 2) { // when the computer wants to play

playComputer();

isQuit = true;

}

else {

isQuit = true;

}

}

//ask intially if the user wants to play the game

bool playGame()

{

system("cls");

char ch;

cout << "Do you want to play ?(Press Y for yes| Press any other key to quit) : ";

cin >> ch;

if (ch == 'Y' || ch == 'y')

return true;

return false;

}

//prints the table

void printGameTable()

{

cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\* ROULLETE TABLE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

<< "\n \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_ "

<< "\n | | | | |"

<< "\n | 1 | 2 | 3 | |"

<< "\n | (B) | (R) | (B) | |"

<< "\n |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_| |"

<< "\n | | | | |"

<< "\n | 4 | 5 | 6 | |"

<< "\n | (R) | (B) | (R) | |"

<< "\n |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_| 1-12 |\_\_\_\_\_\_\_\_\_"

<< "\n | | | | | |"

<< "\n | 7 | 8 | 9 | | |"

<< "\n | (B) | (R) | (B) | | | "

<< "\n |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_| | \*EVEN\* |"

<< "\n | | | | | |"

<< "\n | 10 | 11 | 12 | | |"

<< "\n | (R) | (B) | (R) | | |"

<< "\n |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|"

<< "\n | | | | | |"

<< "\n | 13 | 14 | 15 | | |"

<< "\n | (B) | (R) | (B) | | |"

<< "\n |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_| | \*RED\* |"

<< "\n | | | | | (R) |"

<< "\n | 16 | 17 | 18 | | |"

<< "\n | (R) | (B) | (R) | | |"

<< "\n |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_| 13-24 |\_\_\_\_\_\_\_\_\_|"

<< "\n | | | | | |"

<< "\n | 19 | 20 | 21 | | |"

<< "\n | (B) | (R) | (B) | | |"

<< "\n |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_| | \*BLACK\* |"

<< "\n | | | | | (B) |"

<< "\n | 22 | 23 | 24 | | |"

<< "\n | (R) | (B) | (R) | | |"

<< "\n |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|"

<< "\n | | | | | |"

<< "\n | 25 | 26 | 27 | | |"

<< "\n | (B) | (R) | (B) | | |"

<< "\n |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_| | \*ODD\* |"

<< "\n | | | | | |"

<< "\n | 28 | 29 | 30 | | |"

<< "\n | (R) | (B) | (R) | | |"

<< "\n |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_| 25-36 |\_\_\_\_\_\_\_\_\_|"

<< "\n | | | | |"

<< "\n | 31 | 32 | 33 | |"

<< "\n | (B) | (R) | (B) | |"

<< "\n |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_| |"

<< "\n | | | | |"

<< "\n | 34 | 35 | 36 | |"

<< "\n | (R) | (B) | (R) | |"

<< "\n |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\n\n";

}

**Cell.h**

/\*

\* Name : Ria Shiv

\* File : Cell.h

\* Date : 30th October 2016

\*/

#pragma once

#ifndef CELL\_H

#define CELL\_H

#include <cstdlib>

class Cell

{

public:

int value; //value of cell impt

Cell(int aval, bool aisRed) //contructor

{

value = aval;

isRed = aisRed;

}

Cell() //assigns random bumber to cell btw 0 - 37

{

value = rand() % 37 + 0;

}

bool isEven(); //checks if its ODD or EVEN

bool isRed; //checks if its RED or BLACk

bool isZeroOrDoubleZero(); //checks if the number is 0 or 37

int getValue(); //returns value of cell

int getRow(); //returns the row

};

#endif

**Cell.cpp**

/\*

\* Name : Ria Shiv

\* File : Cell.cpp

\* Date : 30th October 2016

\*/

#include "Cell.h"

//checks if the cell is odd or even

bool Cell::isEven()

{

return value % 2 == 0;

}

//Checks if the number is 0 or 37

bool Cell::isZeroOrDoubleZero()

{

if (value == 0 || value == 37)

{

return false;

}

return true;

}

//returns the cell value

int Cell::getValue()

{

return value;

}

//returns the value of row for the one -third

int Cell::getRow()

{

if (value >= 1 && value <= 12)

return 1;

else if (value > 12 && value <= 24)

return 2;

else

return 3;

}

**Player.h**

/\*

\* Name : Ria Shiv

\* File : Player.h

\* Date : 30th October 2016

\*/

#pragma once

#ifndef PLAYER\_H

#define PLAYER\_H

#include<string>

#include<iostream>

#include <list>

#include"PlayerPlay.h"

#include"Game.h"

using namespace std;

class Player

{

private:

string name; //To store the name of the player

int balance; //To store current balance

int betId; //To store the type of bet

std::list<PlayerPlay\*> playPlayerList; //stores historical plays

std::list<PlayerPlay\*>::iterator playPlayerIt; // used to iterate through the playerPlaylist

Game \*game;

public:

void setName(); //to set the name of the player

void play(); //called to play the game when the player plays the first time

void play(PlayerPlay\* playerPlay); //called every other time the layer plays the game

void playComputer(); // called when the player opts for the computer to play

void chooseOption(); //chooses option on which type of bet is done

void showAnalytics(); //shows the history of all the play

int getBalance(); //returns the balance left

PlayerPlay\* computerPlayerPlay(int amount, int inputBet, int inputBetOption); //called when opted for the computer to play

Player(string aname, int abalance, Game \*agame) //player constructor

{

name = aname;

balance = abalance;

game = agame;

}

};

#endif

**Player.cpp**

/\*

\* Name : Ria Shiv

\* File : Player.cpp

\* Date : 30th October 2016

\*/

#include "Player.h"

#include<stdlib.h>

#include<ctime>

//called when the computer is opted to play

void Player::playComputer() {

int i = 0;

cout << "\*\*\*\*\* Starting Balance : $" << balance;

while (i < 1000) {

PlayerPlay \*playerPlay = NULL;

int modVal = i % 4; //chooses a number btw 0-3 based on i

if (modVal == 0) {

int choice = game->randomNumner(1, 2);

playerPlay = computerPlayerPlay(10, 1, choice); // Play ODD EVEN

}

else if (modVal == 1) {

int choice = game->randomNumner(1, 2);

playerPlay = computerPlayerPlay(10, 2, choice); // Play Red Black

}

else if (modVal == 2) {

int choice = game->randomNumner(1, 3);

playerPlay = computerPlayerPlay(10, 4, choice); // Play One Third

}

else {

int choice = game->randomNumner(1, 36);

playerPlay = computerPlayerPlay(10, 3, choice); // Play Number

}

play(playerPlay); //called to play the game

playPlayerList.insert(playPlayerList.end(), playerPlay);

i += 1;

if (balance < 10) {

break;

}

}

showAnalytics(); //shows the histroy of the play

cout << "\n\n\*\*\*\*\* Balance at the end of game : $" << balance;

}

// computer player is called and the object is intialized with amount the betting amount and option

PlayerPlay\* Player::computerPlayerPlay(int amount, int inputBet, int inputBetOption) {

PlayerPlay\* playerPlay = new PlayerPlay(amount, inputBet);

playerPlay->inputBallBet(inputBetOption);

return playerPlay;

}

//In this part the player chooses the option on which he wants to bet on

void Player::chooseOption() {

betId = 0;

cout << "\n\nThere are 4 types of bet you can make \n";

cout << "\t1. Choose Odd or Even\n";

cout << "\t2. Choose RED or BLACK\n";

cout << "\t3. Choose a single number between 1-36\n";

cout << "\t4. Choose one third\n";

cout << "\nPlease select type of bet: ";

while (betId < 1 || betId > 4) { //bet id is assigned with the exception being handled

cin >> betId;

}

}

//this is callled to play the game

void Player::play() {

cout << "\n\n=======================================================";

chooseOption(); //chooses option from the list

PlayerPlay \*playerPlay = new PlayerPlay(betId); //object of player play is created

playerPlay->setRemainingBal(balance); //balance is set

playerPlay->takeWager(); //this function makes the user assign the wager for the bet

playerPlay->inputBallBet(); //bet on type is inputted

play(playerPlay);// called to play the real game

playPlayerList.insert(playPlayerList.end(), playerPlay);//inserts object on the memory list

playerPlay->showAnalytics(); // shows betting info

cout << "\n=======================================================\n\n";

}

void Player::play(PlayerPlay\* playerPlay) {

balance = balance - playerPlay->getTotalWager();// the betting balance is subtracted

game->rollBall();// ball is rolled to a random position

playerPlay->evaluateBet(game->cellAtBallPosition()); // the bet value is evaluated

balance = balance + playerPlay->winning(); // the winning amount is added

playerPlay->setRemainingBal(balance);// the balance is set back

}

//sets your name in the program

void Player::setName()

{

cout << "Enter your name ";

getline(cin, name);

}

//the list is displayed with all the values of the previous plays

void Player::showAnalytics() {

cout << "\n\n=================================================================================================";

//iterates through the list

for (playPlayerIt = playPlayerList.begin(); playPlayerIt != playPlayerList.end(); ++playPlayerIt) {

PlayerPlay\* playerPlay = dynamic\_cast<PlayerPlay\*> (\*playPlayerIt);

playerPlay->showAnalytics(); //calls playerplay function using object

}

cout << "\n=================================================================================================\n\n";

}

//balance is returned

int Player::getBalance() {

return balance;

}

**Game.h**

/\*

\* Name : Ria Shiv

\* File : Game.h

\* Date : 30th October 2016

\*/

#pragma once

#include<map>

#include<stdlib.h>

#include<set>

#include<ctime>

#include "Cell.h"

class Game

{

std::map<int, Cell\*> cellMap; // makes a cell map in the memory

int currentBallPosition; //the position of the ball

public:

void rollBall(); // rolls the ball on the circle

int randomNumner(int low, int high); //returns a random number

Cell\* cellAtBallPosition(); //Finds the cell in the map

//contrutor makes a map in the memory

Game()

{

cellMap.insert(std::pair<int, Cell\*>(0, new Cell(0, false)));

cellMap.insert(std::pair<int, Cell\*>(1, new Cell(1, true)));

cellMap.insert(std::pair<int, Cell\*>(2, new Cell(2, false)));

cellMap.insert(std::pair<int, Cell\*>(3, new Cell(3, true)));

cellMap.insert(std::pair<int, Cell\*>(4, new Cell(4, false)));

cellMap.insert(std::pair<int, Cell\*>(5, new Cell(5, true)));

cellMap.insert(std::pair<int, Cell\*>(6, new Cell(6, false)));

cellMap.insert(std::pair<int, Cell\*>(7, new Cell(7, true)));

cellMap.insert(std::pair<int, Cell\*>(8, new Cell(8, false)));

cellMap.insert(std::pair<int, Cell\*>(9, new Cell(9, true)));

cellMap.insert(std::pair<int, Cell\*>(10, new Cell(10, false)));

cellMap.insert(std::pair<int, Cell\*>(11, new Cell(11, false)));

cellMap.insert(std::pair<int, Cell\*>(12, new Cell(12, true)));

cellMap.insert(std::pair<int, Cell\*>(13, new Cell(13, false)));

cellMap.insert(std::pair<int, Cell\*>(14, new Cell(14, true)));

cellMap.insert(std::pair<int, Cell\*>(15, new Cell(15, false)));

cellMap.insert(std::pair<int, Cell\*>(16, new Cell(16, true)));

cellMap.insert(std::pair<int, Cell\*>(17, new Cell(17, false)));

cellMap.insert(std::pair<int, Cell\*>(18, new Cell(18, true)));

cellMap.insert(std::pair<int, Cell\*>(19, new Cell(19, true)));

cellMap.insert(std::pair<int, Cell\*>(20, new Cell(20, false)));

cellMap.insert(std::pair<int, Cell\*>(21, new Cell(21, true)));

cellMap.insert(std::pair<int, Cell\*>(22, new Cell(22, false)));

cellMap.insert(std::pair<int, Cell\*>(23, new Cell(23, true)));

cellMap.insert(std::pair<int, Cell\*>(24, new Cell(24, false)));

cellMap.insert(std::pair<int, Cell\*>(25, new Cell(25, true)));

cellMap.insert(std::pair<int, Cell\*>(26, new Cell(26, false)));

cellMap.insert(std::pair<int, Cell\*>(27, new Cell(27, true)));

cellMap.insert(std::pair<int, Cell\*>(28, new Cell(28, false)));

cellMap.insert(std::pair<int, Cell\*>(29, new Cell(29, false)));

cellMap.insert(std::pair<int, Cell\*>(30, new Cell(30, true)));

cellMap.insert(std::pair<int, Cell\*>(31, new Cell(31, false)));

cellMap.insert(std::pair<int, Cell\*>(32, new Cell(32, true)));

cellMap.insert(std::pair<int, Cell\*>(33, new Cell(33, false)));

cellMap.insert(std::pair<int, Cell\*>(34, new Cell(34, true)));

cellMap.insert(std::pair<int, Cell\*>(35, new Cell(35, false)));

cellMap.insert(std::pair<int, Cell\*>(36, new Cell(36, true)));

cellMap.insert(std::pair<int, Cell\*>(37, new Cell(37, false)));

}

};

**Game.cpp**

/\*

\* Name : Ria Shiv

\* File : Game.cpp

\* Date : 30th October 2016

\*/

#include "Cell.h"

#include <map>

#include "Game.h"

// rolls the ball to find a random number and assigns it to currentBallPosition

void Game::rollBall()

{

currentBallPosition = randomNumner(1, 36);

}

// returns a random number

int Game::randomNumner(int low, int high) {

srand(time(NULL)); //seed value

if (low > high) return high;

return low + (std::rand() % (high - low + 1));

}

//returns the value of the cell position at the currentBallPosition

Cell\* Game::cellAtBallPosition()

{

std::map<int, Cell\*>::iterator it;

it = cellMap.find(currentBallPosition);

return it->second;

}

**PlayerOption.h**

/\*

\* Name : Ria Shiv

\* File : PlayerOption.h

\* Date : 30th October 2016

\*/

#pragma once

#include<string>

#include<iostream>

#include<set>

#include "Cell.h"

using namespace std;

class PlayerOption

{

private:

int betTypeId, betOption; //stores the type of bet and what is betted on

int wager, winnings; //stores the amount of money betted and money won

enum BetType { EVEN\_ODD = 1, RED\_BLACK = 2, NUMBER = 3, ONE\_THIRD = 4 }; //enum for type of bet

enum BetEvenOddOpt { ODD = 1, EVEN = 2 }; //enum for odd and even

enum BetRedBlackOpt { RED = 1, BLACK = 2, }; //enum for red and black option

enum BetSetsOpt { OT\_1\_12 = 1, OT\_13\_24 = 2, OT\_25\_36 = 3 }; // ONE\_THIRD

void evaluateRedBlack(Cell\* cell); // to evaluate red and black type bet

void evaluateOddEven(Cell\* cell); // to evaluate odd and even type bet

void evaluateSingleNumber(Cell\* cell); //to evaluate single number bet

void evaluateOneThird(Cell\* cell); //to evaluate one third bet

public:

void displayOptions(); //displays the menu on the bet option for each type of bet

void isValidOption(); //checks if the options inputted are valid or not

void evaluateBet(Cell\* cell); // sees which type of bet is made to call the specific function

int winningAmount(); //returns the amount won

void showAnalytics(Cell\* cell); //shows the analytics of the bets

bool isWinner; //stores a bool value depending uponn if the person won or not

string selectedBet(); //assigns value to the bet selected depending on the type

void setWager(int amount);//sets value to the wager variable

PlayerOption(int \_betTypeId) { //contructor

betTypeId = \_betTypeId;

betOption = 0;

wager = 0;

winnings = 0;

}

PlayerOption(int \_betTypeId, int \_betOption) { //overloaded constructor

betTypeId = \_betTypeId;

betOption = \_betOption;

wager = 0;

winnings = 0;

}

};

**PlayerOption.cpp**

/\*

\* Name : Ria Shiv

\* File : PlayerOption.cpp

\* Date : 30th October 2016

\*/

#include "PlayerOption.h"

#include<stdlib.h>

//called to evaluate bet on the cell

void PlayerOption::evaluateBet(Cell\* cell){

switch (betTypeId){ //the corresponding evaluate bet function is called based on the type of bet

case RED\_BLACK: //for red black bet

evaluateRedBlack(cell);

break;

case EVEN\_ODD: // for odd and even bet

evaluateOddEven(cell);

break;

case NUMBER: // for single number bet

evaluateSingleNumber(cell);

case ONE\_THIRD: //for one third row bet

evaluateOneThird(cell);

break;

}

}

//Callled when the betting type is red black

void PlayerOption::evaluateRedBlack(Cell\* cell){

switch (betOption){

case RED:

isWinner = cell->isRed;

break;

case BLACK:

isWinner = !cell->isRed;

break;

}

winnings = isWinner ? wager \* 2 : 0; //winning is given value based on isWinner

}

//called when the bet is an odd and even type

void PlayerOption::evaluateOddEven(Cell\* cell){

switch (betOption){

case EVEN:

isWinner = cell->isEven();

break;

case ODD:

isWinner = !cell->isEven();

break;

}

winnings = isWinner ? wager \* 2 : 0; //winning is given value based on isWinner

}

//called when the bet is single number type

void PlayerOption::evaluateSingleNumber(Cell\* cell){

if (betOption == cell->getValue())

isWinner = true;

else

isWinner = false;

winnings = isWinner ? (wager \* 36) : 0; //winning is given value based on isWinner

}

//called when the bet is on one third of the row

void PlayerOption::evaluateOneThird(Cell\* cell){

switch (betOption)

{

case OT\_1\_12:

isWinner = cell->getValue() >= 1 && cell->getValue() <= 12;

break;

case OT\_13\_24:

isWinner = cell->getValue() >= 13 && cell->getValue() <= 24;

break;

case OT\_25\_36:

isWinner = cell->getValue() >= 25 && cell->getValue() <= 36;

break;

}

winnings = isWinner ? wager \* 3 : 0; //winnings is given the amount based on the value of isWinner

}

//returns the winning amount

int PlayerOption::winningAmount(){

return winnings;

}

//displays the option of bets that should have been chosen for each type

void PlayerOption::displayOptions(){

switch (betTypeId)

{

case EVEN\_ODD:

cout << "\nChoose 1 for ODD or 2 for EVEN: ";

break;

case RED\_BLACK:

cout << "\nChoose 1 for RED or 2 for BLACK: ";

break;

case NUMBER:

cout << "\nEnter a number between 1-36: ";

break;

case ONE\_THIRD:

cout << "\nCHOOSE 1 for 1-12, 2 for 13-24 OR 3 for 25-36: ";

break;

}

while (betOption == 0){

cin >> betOption;

isValidOption(); //checks for the validity of the option exception handler

}

}

void PlayerOption::isValidOption(){ //checks if the option entered is valid or not

bool isError = false;

switch (betTypeId)

{

case EVEN\_ODD:

case RED\_BLACK:

isError = betOption != 1 && betOption != 2;

break;

case ONE\_THIRD:

isError = betOption != 1 && betOption != 2 && betOption != 3;

break;

case NUMBER:

isError = betOption < 1 || betOption > 36;

break;

}

if (isError){ //error displayed when it is not valid

cout << "@@@@ Error: Please select valid option: ";

betOption = 0;

}

}

//This assign the value to the betOption basd on various betting types

string PlayerOption::selectedBet(){

switch (betTypeId)

{

case EVEN\_ODD:

return betOption == 1 ? "ODD" : "EVEN";

break;

case RED\_BLACK:

return betOption == 1 ? "RED" : "BLACK";

break;

case NUMBER:

return to\_string(betOption);

break;

case ONE\_THIRD:

return betOption == 1 ? "1-12" : (betOption == 2 ? "13-24" : "25-36");

break;

}

return "";

}

//sets the value to the wager

void PlayerOption::setWager(int amount){

wager = amount;

}

//shows the analytics of the historical plays

void PlayerOption::showAnalytics(Cell\* cell){

cout << "\n" << "Bet on : $" + to\_string(wager) + " on " << selectedBet() + "\t" << "Ball rolled: " << (cell->getValue()) <<(cell->isRed ? " (R)" : " (B)") << "\t";

cout << "You " << (isWinner ? "WON\t" : "LOST") << "\tWinning Amount: " << winningAmount();

}

**PlayerPlay.h**

/\*

\* Name : Ria Shiv

\* File : PlayerPlay.h

\* Date : 30th October 2016

\*/

#pragma once

#ifndef PLAYERPLAY\_H

#define PLAYERPLAY\_H

#include "Cell.h"

#include "PlayerOption.h"

#include<set>

#include "Game.h"

using namespace std;

class PlayerPlay

{

std::set<PlayerOption \*> playerOptions; //a set of playerOptions is created

public:

int remainingBal; //the balance left

int moneyWon = 0; //the money won during the bet

int wager = 0; //the balance of the bet

int choice; //the type of bet used

Cell\* cell; //an object of cell is created

PlayerPlay(int \_wager, int achoice) //constructor

{

wager = \_wager;

moneyWon = 0;

choice = achoice;

}

PlayerPlay(int achoice) //overloaded contructor

{

choice = achoice;

moneyWon = 0;

wager = 0;

}

void setRemainingBal(int amount); //updates the amount of the balance

void showAnalytics(); //used to show the historical plays

Cell\* getWinningCell(); //returns the winning cell

void evaluateBet(Cell \*winningCell); //iterates through the options to call the bet amount won

PlayerOption\* inputBallBet(int betNumber); //this is used to enter the bet depending on the type

PlayerOption\* inputBallBet(); //overloaded functions

int getTotalWager(); //returns the total wager that users wants to bet

void takeWager(); //enters the wager from the user

int winning(); //returns the money won

};

#endif

**PlayerPlay.cpp**

/\*

\* Name : Ria Shiv

\* File : PlayerPlay.cpp

\* Date : 30th October 2016

\*/

#include<iostream>

#include "PlayerPlay.h"

using namespace std;

//makes the user enter the wager to bet

void PlayerPlay::takeWager()

{

cout << "\nEnter the bet amount: ";

while (wager == 0) {

cin >> wager;

if (wager > remainingBal) { //handles exception if the bet is greater than the balance

wager = 0;

cout << "\n@@@Error:: Not enough balance. Current balance is $" << remainingBal << "\n";

}

}

}

//returns the money won

int PlayerPlay::winning()

{

return moneyWon;

}

//evaluates the bet if the person won or not

void PlayerPlay::evaluateBet(Cell \*winningCell)

{

cell = winningCell;

std::set<PlayerOption \*>::iterator it;

for (it = playerOptions.begin(); it != playerOptions.end(); ++it) //iterates through the options

{

PlayerOption \*playerOption = dynamic\_cast<PlayerOption\*>(\*it);

playerOption->evaluateBet(winningCell); //the winning cell is passed to evaluate the bet in playerOptions

moneyWon += playerOption->winningAmount();

}

}

//returns the wager of the bet

int PlayerPlay::getTotalWager()

{

return wager;

}

//sets the value of the bet depending on the type

PlayerOption\* PlayerPlay::inputBallBet(int number) {

PlayerOption\* playerOption = new PlayerOption(choice, number); //passes the number to playerOptions

playerOption->setWager(wager);

playerOptions.insert(playerOption);

return playerOption;

}

//overloaded function that ask the user for the type of bet the user wants

PlayerOption\* PlayerPlay::inputBallBet() {

PlayerOption\* playerOption = new PlayerOption(choice);

playerOption->displayOptions(); //ask the user to enter the type of bet

playerOption->setWager(wager);

playerOptions.insert(playerOption);

return playerOption;

}

//returns the winning cell

Cell\* PlayerPlay::getWinningCell() {

return cell;

}

//represents the historical plays

void PlayerPlay::showAnalytics() {

std::set<PlayerOption \*>::iterator it;

for (it = playerOptions.begin(); it != playerOptions.end(); ++it)//iterates through the options

{

PlayerOption\* playerOption = dynamic\_cast<PlayerOption\*>(\*it);

playerOption->showAnalytics(getWinningCell());

cout << "\tBalance: " << remainingBal;

}

}

//updates the balance that remains after each play

void PlayerPlay::setRemainingBal(int amount) {

remainingBal = amount;

}