Project Report

On

BLACKJACK CASINO GAME

Submitted by

Dhairya Sachdeva 20103098

Ananya Kapoor 20103104

DECLARATION

We hereby declare that the project which is submitted as a mini project for the 3rd semester in DATA STRUCTURE AND ALGORITHM in Computer Science department of JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY(JIIT) Sec-62, Noida is an authentic record of our genuine work done under the guidance of Mr. Vivek Kumar Singh and Mr. Manish Kumar Thakur.

MEMBERS:

1. Dhairya Sachdeva (20103098)
2. Ananya Kapoor (20103104)

INTRODUCTION

We aimed to create the simulation of a casino game known as Blackjack or 21.

It uses decks of 52 cards and descends from a global family of casino games known as Twenty-One.

Blackjack players do not compete against each other. The game is a comparing card game where each player competes against the dealer.

The aim of blackjack is to finish the game with a higher total than that of the dealer, without exceeding 21. Going over 21 is commonly known as ‘busting’ and means an automatic loss.

The main aim of this project was to simulate a casino and create a Blackjack game which not only allows players to learn and improve their knowledge of the game and card counting while they play, but also help the player to make correct betting decisions in order to win as much as possible.

This led to the inclusion of two game modes in our program: Casino mode and Just for Fun.

DATA STRUCTURES USED

The implementation of the Blackjack Casino Game uses the following data structures:

1. Stacks (STL Library)
2. Queues (STL Library)
3. Vectors (STL Library)

Why these data structures?

1. Stacks: As we know that the stack data structure uses the concept of LIFO (last in first out), which perfectly illustrates the working of a 52-card deck. In a stack only the top element is popped or removed, similarly in a deck of cards the dealer always deals the top card to the player.

2. Queues: As we know that the queue data structure uses the concept of FIFO (first in-first out), which adequately illustrates the player and dealer hands after the cards have been dealt by the dealer. The new card is always pushed back into the queue and we are also able to retrieve the front card of the hand i.e., the front of the queue.

3. Vectors: Vectors are same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container. Vector elements are placed in contiguous storage so that they can be accessed and traversed using

iterators. In vectors, data is inserted at the end. Inserting at the end takes differential time, as sometimes there may be a need of extending the array. Removing the last element takes only constant time because no resizing happens. Inserting and erasing at the beginning or in the middle is linear in time. In the code given below, we have converted stacks and queues to vectors in some places because of the vast library it has and the numerous in-built functions which enables us to perform numerous tasks in linear time unlike the stacks and queues.

CODE

```
#include <math.h>
#include <list>
#include <stack>
#include <queue>
#include <vector>
#include <iterator>
#include <string.h>
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>              //Used for srand((unsigned) time(NULL)) command
#include <process.h>           //Used for system("cls") command
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <ctime>
#include <limits>
#define spade 06               //Used to print spade symbol
#define club 05                //Used to print club symbol
#define diamond 04             //Used to print diamond symbol
#define heart 03
//Declare namespace
using namespace std;

//Prototype functions.
int getRandomNumber(int low, int high);
int CardValue(int card);
int getTopCard();
```

```
int getHandValue(queue<int> hand);
bool playAnotherHand();
bool userWantsToDraw();
stack<int> initializeDeck(stack<int> deck);
void dumpDeck(stack<int> deck, int size);
stack<int> shuffle(stack<int> deck);
void ShowCard(int card);
void showCards(queue<int> hand, bool hideFirstCard);
void whoWins(queue<int> pHand, queue<int> dHand);
void checkBust(queue<int> pHand, queue<int> dHand);
void blackJack(queue<int> pHand, queue<int> dHand);
void naturalBlackJack(queue<int> pHand, queue<int> dHand);
void playOneHand();
void scoreBoard(queue<int> pHand, queue<int> dHand);
void backdoorKenny(queue<int> hand);
queue<int> checkSoftOrHard(queue<int> pHand);
void softOrHardAI(queue<int> dHand, queue<int> pHand);
void showRules();
queue<int> addToHand(queue<int> hand, int cardToAdd);
void hitUntilStand(queue<int> dHand, stack<int> deck, queue<int> pHand);
void dumpDeck(stack<int> deck);

void choose_simulation_mode();
void menu();
void show_balance_and_chips();
void set_balance();
int checkChips();
void Exchange_chips();
void Exit_menu();
void alt_exit();
void update_bet(float multiplier);
void Place_bet();
```

```
queue<int> change_value_of_ace(queue<int> pHand, bool change_Ace, int position_of_Ace);

//Declare global Variables
int topCard = 0;
float win = 1.00;
int level;
stack<int> deck;
queue<int> Global_dHand;
int player_chips = 0;
long int player_balance = 0;
char mode_choice;
int g=1, bet=0;


int main()
{
   system("Color 0b");
          choose_simulation_mode();
          return 0;
}


/* ================================================
Name: set_balance
Desc: For players to enter casino with a fixed amount of cash.
Parameters:
  None.
Return:
  void.
================================================ */
void set_balance()
{
```

```
   long int input_balance;
   system("cls");
   cout<<"\t\tThis is for Players to Define how much cash they want to enter with in the Casino..."<<endl;
   cout<<"\t\tEnter Money: $ ";
   cin>>input_balance;
   ::player_balance += input_balance;
   cout<<endl<<endl;
   if(::player_balance  != 0)
   {
      cout<<"\t\tPress any key to go to Casino Menu..."<<endl;
      system("pause");
      menu();
   }
   else
   {
      cout<<"\t\tCannot proceed with $0..."<<endl;
      cout<<endl<<endl;
      system("pause");
      set_balance();
   }

}


/* ================================================
Name: show_balance_and_chips
Desc: Displays the balance cash and chips of the player in the casino.
Parameters:
  None.
Return:
  void.
================================================ */
```

```
void show_balance_and_chips()
{
   cout<<"\n\t\tMoney Balance = $ "<<::player_balance;
   cout<<endl;
   cout<<"\t\tChips Balance = "<<::player_chips<<" Chips";
   cout<<endl;
}



/* ================================================
Name: choose_simulation_mode
Desc: Menu for selecting the simulation the user wants to experience.
Parameters:
  None.
Return:
  void.
================================================ */
void choose_simulation_mode()
{
   system("cls");
   int mode_choice;
   cout<<"\n\n\n\n\t\t\tSelect the Mode of Simulation you want!"<<endl;
   cout<<"\n\n\n\t\tOptions are: 1. Casino Mode (Includes Betting)"<<endl;
   cout<<"\t                    2. Just For Fun (Does Not Include Betting)"<<endl;
   cout<<"\t\tChoose an option to proceed: ";
   cin>>mode_choice;
   switch(mode_choice)
   {
      case 1:
         ::mode_choice = 'C';
         set_balance();
         break;
```

```
    case 2:
        ::mode_choice = 'F';
        playOneHand();
        break;

    default:
        cout<<"This option is not available"<<endl<<"Please try again!"<<endl;
        choose_simulation_mode();
        break;
  }
}


/* ================================================
Name: menu
Desc: Menu for the casino simulation mode.
Parameters:
  None.
Return:
  void.
================================================ */
void menu()
{
  system("cls");
  printf("\n");
  printf("\n");
  printf("\n");
  printf("\n          222           111                    ");
  printf("\n         222 222         11111                    ");
  printf("\n        222   222        11 111                    ");
  printf("\n          222           111                    ");
```

```
printf("\n          222          111                  ");
printf("\n");
printf("\n%c%c%c%c%c    %c%c        %c%c       %c%c%c%c%c    %c    %c          ", club, club, club, club, club, spade, spade, diamond, diamond, heart, heart, heart, heart, heart, club, club);
printf("\n%c    %c    %c%c        %c  %c     %c    %c   %c   %c          ", club, club, spade, spade, diamond, diamond, heart, heart, club, club);
printf("\n%c    %c    %c%c        %c    %c    %c        %c  %c          ", club, club, spade, spade, diamond, diamond, heart, club, club);
printf("\n%c%c%c%c%c%c    %c%c        %c %c%c %c     %c        %c %c          ", club, club, club, club, club, spade, spade, diamond, diamond, diamond, diamond, heart, club, club);
printf("\n%c    %c    %c%c        %c %c%c%c%c %c    %c        %c%c %c          ", club, club, spade, spade, diamond, diamond, diamond, diamond, diamond, diamond, heart, club, club, club);
printf("\n%c    %c    %c%c        %c     %c    %c        %c    %c          ", club, club, spade, spade, diamond, diamond, heart, club, club);
printf("\n%c    %c    %c%c        %c      %c    %c    %c    %c    %c          ", club, club, spade, spade, diamond, diamond, heart, heart, club, club);
printf("\n%c%c%c%c%c%c%c    %c%c%c%c%c%c%c%c    %c      %c    %c%c%c%c%c%c    %c    %c          ", club, club, club, club, club, club, club, spade, spade, spade, spade, spade, spade, spade, diamond, diamond, heart, heart, heart, heart, heart, club, club);
printf("\n");
printf("\n                21                          ");

printf("\n    %c%c%c%c%c%c%c%c%c    %c%c        %c%c%c%c%c%c    %c    %c          ", diamond, diamond, diamond, diamond, diamond, diamond, diamond, diamond, heart, heart, club, club, club, club, club, spade, spade);
printf("\n       %c%c       %c  %c     %c    %c   %c   %c          ", diamond, diamond, heart, heart, club, club, spade, spade);
printf("\n       %c%c       %c    %c    %c        %c  %c          ", diamond, diamond, heart, heart, club, spade, spade);
printf("\n       %c%c       %c %c%c %c     %c        %c %c          ", diamond, diamond, heart, heart, heart, heart, club, spade, spade);
printf("\n       %c%c       %c %c%c%c%c %c    %c        %c%c %c          ", diamond, diamond, heart, heart, heart, heart, heart, heart, club, spade, spade, spade);
printf("\n       %c%c       %c     %c    %c        %c    %c          ", diamond, diamond, heart, heart, club, spade, spade);
printf("\n    %c  %c%c     %c      %c    %c    %c    %c    %c          ", diamond, diamond, diamond, heart, heart, club, spade, spade);
```

```
   printf("\n     %c%c%c     %c      %c    %c%c%c%c%c%c    %c     %c          ", diamond, diamond, diamond,
heart, heart, club, club, club, club, club, spade, spade);
   printf("\n");
   printf("\n       222              111                 ");
   printf("\n      222               111                 ");
   printf("\n     222                111                 ");
   printf("\n    222222222222222    111111111111111                ");
   printf("\n    2222222222222222   11111111111111111                  ");
   printf("\n");
   printf("\n");

   int menu_choice, second_menu_choice;
   cout<<"\t\t\t\tWelcome to BLACKJACK tables !!"<<endl;
   cout<<"\t\t1. To Play Blackjack"<<endl;
   cout<<"\t\t2. Exchange chips"<<endl;
   cout<<"\t\t3. Set Balance (Money you want to enter with)"<<endl;
   cout<<"\t\t4. To View Money & Chip Balance"<<endl;
   cout<<"\t\t5. Rule Book"<<endl;
   cout<<"\t\t6. Exit Casino"<<endl;
   cout<<"\n\n\tChoose an option to proceed: ";
   cin>>menu_choice;
   switch(menu_choice)
   {
      case 1:
         if(::player_chips < 500)
         {
            cout<<"You don't have enough chips to play BLACKJACK (Min. Chips required = 500)"<<endl;
            cout<<"1. Go to Main Menu"<<endl;
            cout<<"2. Buy Chips"<<endl;
            cout<<"3. Exit Casino"<<endl;
            cout<<"Choose an option to proceed: ";
            cin>>second_menu_choice;
```

```
        switch(second_menu_choice)
        {
            case 1:
                menu();
                break;
            case 2:
                Exchange_chips();
                break;
            case 3:
                alt_exit();
                break;
            default:
                cout<<"This option is not available!!"<<endl<<endl;
                menu();
                break;
        }
    }
    else if(::player_chips >= 500)
    {
        playOneHand();
    }
    break;

case 2:
    Exchange_chips();
    break;

case 3:
    set_balance();
    break;

case 4:
```

```
            show_balance_and_chips();
            system("pause");
            menu();
            break;

        case 5:
            showRules();
            system("pause");
            menu();
            break;

        case 6:
            alt_exit();
            break;

        default:
            cout<<"This option is not available"<<endl<<"Please try again!"<<endl;
            menu();
            break;
    }
}


/* ================================================
Name: update_bet
Desc: Updates the initial bet of the player in the casino
      simulation mode.
Parameters:
   float - multiplier.
Return:
   void.
================================================ */
```

```
void update_bet(float multiplier)
{
   if(multiplier == 3) //backdoor kenny
   {
      cout<<"Backdoor Kenny..."<<"Payout = 3 * Initial Bet"<<endl;
      ::player_chips = ::player_chips + (3 * ::bet);
      cout<<"Player Chips Updated..."<<::player_chips<<endl;
      ::bet = 0;
   }
   else if(multiplier == 2) // normal win without blackjack
   {
      cout<<"Win..."<<"Payout = 2 * Initial Bet"<<endl;
      ::player_chips = ::player_chips + (2 * ::bet);
      cout<<"Player Chips Updated..."<<::player_chips<<endl;
      ::bet = 0;
   }
   else if(multiplier == 1)
   {
      cout<<"Draw..."<<"Both Player and Dealer have equal hands..."<<endl;
      cout<<"Payout = Initial Bet"<<endl;
      ::player_chips = ::player_chips + ::bet;
      cout<<"Player Chips Updated..."<<::player_chips<<endl;
      ::bet = 0;
   }
   else if(multiplier == 0) // loss
   {
      cout<<"Loss..."<<"No Payout, Lost the chips placed on Bet..."<<endl;
      ::bet = 0;
   }
   else if(multiplier == 2.5) //
   {
      cout<<"Blackjack Win..."<<"Payout = 2.5 * Initial Bet"<<endl;
```

```
      ::player_chips = ::player_chips + (2.5 * ::bet);
      cout<<"Player Chips Updated..."<<::player_chips<<endl;
      ::bet = 0;
   }
}



/* ================================================
Name: Place_bet
Desc: For player in the casino simulation mode to place a
     bet before playing a round of blackjack.
Parameters:
  None.
Return:
  void.
================================================ */
void Place_bet()
{
   int choice, choice1;
   show_balance_and_chips();
   if(::player_chips ==0)
   {
      cout<<"\n\n\n\t\tYou don't have any chips to place the bet..."<<endl;
      cout<<"\t\t1. To buy more chips"<<endl;
      cout<<"\t\t2. To Check-in more cash and update your Money Balance"<<endl;
      cout<<"\t\t3. To go to Casino Menu"<<endl;
      cout<<"\t\t4. To Exit Casino"<<endl;
      cout<<"\t\tChoose an option to proceed: ";
      cin>>choice;

      switch(choice)
      {
```

```
        case 1:
            Exchange_chips();
            break;

        case 2:
            set_balance();
            break;

        case 3:
            menu();
            break;

        case 4:
            alt_exit();
            break;

        default:
            cout<<"\n\tThis option is not available"<<endl<<"\tPlease try again!"<<endl;
            system("pause");
            menu();
            break;

    }
}
else
{
    cout<<"Place the Bet...(Max Bet can be "<<::player_chips<<") : ";
    cin>>::bet;
    if(::bet <= ::player_chips)
    {
        ::player_chips -= ::bet;
        cout<<"Bet Placed of "<<bet<<"Chips..."<<endl;
```

```
}
else if(::bet > ::player_chips)
{
   cout<<"\n\n\n\t\tYou don't have any chips to place the bet..."<<endl;
   cout<<"\t\t1. To Try Again (Play Blackjack)"<<endl;
   cout<<"\t\t2. To buy more chips"<<endl;
   cout<<"\t\t3. To Check-in more cash and update your Money Balance"<<endl;
   cout<<"\t\t4. To go to Casino Menu"<<endl;
   cout<<"\t\t5. To Exit Casino"<<endl;
   cout<<"\t\tChoose an option to proceed: ";
   cin>>choice1;

   switch(choice1)
   {
      case 1:
         playOneHand();
         break;

      case 2:
         Exchange_chips();
         break;

      case 3:
         set_balance();
         break;

      case 4:
         menu();
         break;

      case 5:
         alt_exit();
```

```
            break;

        default:
            cout<<"\n\tThis option is not available"<<endl<<"\tPlease try again!"<<endl;
            system("pause");
            menu();
            break;
      }
    }
  }
}


/* ================================================
Name: alt_exit
Desc: exit menu for the player to leave the simulation of the blackjack
    game. (reminds player to retrieve winnings iff in casino simulation)
Parameters:
  None.
Return:
  void.
================================================ */
void alt_exit()
{
  char yes_or_no;
  int exit_or_menu;
  system("cls");
  if(::mode_choice == 'c' || ::mode_choice == 'C')
  {
    cout<<"1. To Exit the Casino..."<<endl;
    cout<<"2. Go to Casino Menu..."<<endl;
    cout<<"3. Trade-In Chips for Cash ($)..."<<endl;
```

```
    }
    else
    {
       cout<<"1. To Exit the Casino..."<<endl;
       cout<<"2. Go to select Simulation Mode..."<<endl;
    }

    cout<<"\n\n\tChoose an option to proceed...";
    cin>>exit_or_menu;

    switch(exit_or_menu)
    {
       case 1:
         if(::mode_choice == 'c' || ::mode_choice == 'C')
         {
            if(::player_chips != 0)
            {
               show_balance_and_chips();
               cout<<"\n\n\tAre you sure you want to exit without trading in your chips for cash($) ??
(Y/N)"<<endl;
               cout<<"\tIf YES: All Chips you own will go to the Casino..."<<endl;
               cout<<"\tIf NO:  Trade-In your chips for Cash($) before you Exit..."<<endl;
               cout<<"\n\tChoose an option to proceed: ";
               cin>> yes_or_no;
               if(yes_or_no == 'Y' || yes_or_no == 'y')
               {
                  cout<<"\n\n\tALL CHIPS LOST..."<<endl;
                  ::player_chips=0;
                  Exit_menu();
               }
               else if(yes_or_no == 'N' || yes_or_no == 'n')
```

```
        {
            Exchange_chips();
        }
        else
        {
            cout<<"\n\n\tThe option you chose is not available !!"<<"Please Try Again !!"<<endl;
            cout<<endl<<endl;
            alt_exit();
        }
      }
      else
      {
        Exit_menu();
      }
    }
    else
      Exit_menu();
    Exit_menu();
    break;

  case 2:
    if(::mode_choice == 'c' || ::mode_choice == 'C')
      menu();
    else
      choose_simulation_mode();
    break;

  case 3:
    if(::mode_choice == 'c' || ::mode_choice == 'C')
      Exchange_chips();
    else
      cout<<"";
```

```
        break;

    default:
        cout<<"This option is not available"<<endl<<"Please try again!"<<endl;
        menu();
        break;
  }
}


/* ================================================
Name: Exit_menu
Desc: Menu for exchange of cash to chips and vice versa.
Parameters:
  None.
Return:
  void.
================================================ */
void Exit_menu()
{
  system("cls");
  system("Color 0c");
  cout<<"\n\n\t\t\tThanks for Playing Blackjack!!";
  cout<<"\n\n\t\t\tProject made by: ";
  cout<<"\n\t\t\t 20103098 Dhairya Sachdeva B4";
  cout<<"\n\t\t\t 20103104 Ananya Kapoor B4";
  cout<<endl;
  exit(0);
}


/* ================================================
```

```
Name: Exchange_chips
Desc: Menu for exchange of cash to chips and vice versa.
Parameters:
  None.
Return:
  void.
================================================= */
void Exchange_chips()
{
   system("cls");
   int exchange_choice;
   int amount_of_chips;
   cout<<"\t\t1 $ = 1 chip"<<endl;
   show_balance_and_chips();
   cout<<endl<<endl<<endl;
   cout<<"\t\t1. To exchange money for chips"<<endl;
   cout<<"\t\t2. To exchange chips for money"<<endl;
   cout<<"\t\t3. To go back to Main Menu"<<endl;
   cout<<"\n\tChoose an option to proceed: ";
   cin>>exchange_choice;

   switch(exchange_choice)
   {
      case 1:
         cout<<"\n\tEnter the amount of chips you want to buy: ";
         cin>>amount_of_chips;
         if(amount_of_chips <= ::player_balance)
         {
            ::player_chips+=amount_of_chips;
            ::player_balance-=amount_of_chips;
            show_balance_and_chips();
            cout<<endl;
```

```
        Exchange_chips();
    }
    else
    {
        cout<<"\n\tYou don't have enough balance to buy this amount of chips !"<<endl;
        cout<<"\tPlease Try Again!"<<endl;
        system("pause");
        Exchange_chips();
    }
    break;

case 2:
    cout<<"\n\tEnter the amount of chips you want to trade for money: ";
    cin>>amount_of_chips;
    if(amount_of_chips <= ::player_chips)
    {
        ::player_chips-=amount_of_chips;
        ::player_balance+=amount_of_chips;
        show_balance_and_chips();
    }
    else
    {
        cout<<"\n\tYou don't have enough chips to trade-in!"<<endl;
        cout<<"\tPlease Try Again!"<<endl;
        system("pause");
        Exchange_chips();
    }
    break;

case 3:
    menu();
    break;
```

```
        default:
            cout<<"\n\tThis option is not available"<<endl<<"\tPlease try again!"<<endl;
            Exchange_chips();
            break;
    }
}


/* ================================================
Name: checkChips
Desc: Returns the player's chips.
Parameters:
   None.
Return:
   int - c.
================================================ */
int checkChips()
{
    int c = :: player_chips;
    return c;
}

/* ================================================
Name: showRules
Desc: Displays the game rules
Parameters:
   None.
Return:
   void.
================================================ */
void showRules()
```

```
{
   system("cls");
        //Display rules and information in a visually pleasing graphic.
        cout << "////////////////////////////////////////////////////////" << endl;
        cout << "////       Welcome to the game of Twenty One!      /////" << endl;
        cout << "//// ____________________________________________  /////" << endl;
        cout << "//// | Rules:                                | /////" << endl;
        cout << "//// | o Aces can be either 1 or 11 points.       | /////" << endl;
        cout << "//// | o Dealer wins ties unless player has a     | /////" << endl;
        cout << "//// |   blackjack.                           | /////" << endl;
        cout << "//// |                                       | /////" << endl;
        cout << "//// | Winnings -                             | /////" << endl;
        cout << "//// | Winnings are based on the weight of a given | /////" << endl;
        cout << "//// | win or loss:                           | /////" << endl;
        cout << "//// | o Your score beats Dealer's score -        | /////" << endl;
        cout << "//// |   1:1 payout ratio.                      | /////" << endl;
        cout << "//// |   (100% increase of your initial bet.)     | /////" << endl;
        cout << "//// | o Dealer's score beats your score -        | /////" << endl;
        cout << "//// |   1:1 loss ratio.                        | /////" << endl;
        cout << "//// |   (100% decrease of your initial bet.)     | /////" << endl;
        cout << "//// | o Blackjack -                          | /////" << endl;
        cout << "//// |   3:2 payout ratio.                      | /////" << endl;
        cout << "//// |   (150% increase of your initial bet.)     | /////" << endl;
        cout << "//// | o Natural Blackjack (A in two. Auto win) -  | /////" << endl;
        cout << "//// |   3:2 payout ratio.                      | /////" << endl;
        cout << "//// |   (150% increase of your initial bet.)     | /////" << endl;
        cout << "//// | o Backdoor Kenny (Face then A in two) -    | /////" << endl;
        cout << "//// |   1:4 payout ratio.                      | /////" << endl;
        cout << "//// |   (25% increase of your initial bet.)      | /////" << endl;
        cout << "//// | o Push (Tie) -                         | /////" << endl;
        cout << "//// |   i.  1:1 payout if in your favor.         | /////" << endl;
        cout << "//// |      (100% increase of your initial bet.)  | /////" << endl;
```

```
        cout << "//// |   ii. 0:1 payout if in Dealer's favor.      | /////" << endl;
        cout << "//// |      (No loss, no gain.)                | /////" << endl;
        cout << "//// |                                         | /////" << endl;
        cout << "//// | These odds determine your winnings         | /////" << endl;
        cout << "//// | multiplier. If you are in a betting mood,   | /////" << endl;
        cout << "//// | you can place a bet and multiply it by your | /////" << endl;
        cout << "//// | winnings multiplier at the end of the game  | /////" << endl;
        cout << "//// | to see how much you win. Feel free to give  | /////" << endl;
        cout << "//// | me your negative winnings!                | /////" << endl;
        cout << "//// |_____| /////" << endl;
        cout << "////////////////////////////////////////////////////" << endl;
        cout << "////            Level of Difficulty             /////" << endl;
        cout << "//// _____ /////" << endl;
        cout << "//// | You may choose to play with a beginner or   | /////" << endl;
        cout << "//// | expert Dealer:                          | /////" << endl;
        cout << "//// | o (Beginner) - Dealer stands at a soft 17   | /////" << endl;
        cout << "//// |              and has no AI.              | /////" << endl;
        cout << "//// | o (Expert)   - Dealer hits at a soft 17     | /////" << endl;
        cout << "//// |              and has AI.                | /////" << endl;
        cout << "//// |_____| /////" << endl;
        cout << "////////////////////////////////////////////////////" << endl;
        system("pause");
        menu();
}


/* ================================================
Name: playOneHand
Desc: Main game logic.
Parameters:
  None.
Return:
```

```
  void.
================================================= */
void playOneHand()
{
        //Declare local variable
        char Play = 'N';
        char temp;

        //Start the game at least once and repeat while Player wants to.
        do
        {
           ::g=1;
           //cleaning global dealer's hand
           while(!::Global_dHand.empty())
    {
       ::Global_dHand.pop();
    }

                //Declare, initialize and shuffle the deck.
                stack <int> deck;
                deck=initializeDeck(deck);
                ::deck=deck;
                deck=shuffle(deck);
    ::deck=deck;
                //Declare and initialize Player and Dealer hands.
                queue <int> pHand;
                queue <int> dHand;
    system("cls");

    //place bet
                if(::mode_choice == 'c' || ::mode_choice == 'C')
    {
```

```
    Place_bet();
}
else
{

}
            //Deal first round
            pHand = addToHand(pHand, getTopCard());
            dHand = addToHand(dHand, getTopCard());
            pHand = addToHand(pHand, getTopCard());
            dHand = addToHand(dHand, getTopCard());
            deck=::deck;
            ::Global_dHand=dHand;

            //Display an explanation  of what is happening
            cout << "////////////////////////////////////////////////////////" << endl;
            cout << "\n" << endl;
            cout << "The Dealer shuffled and dealt you each your cards. ";
            cout << "\n" << endl;

            //Check for an automatic win.
            naturalBlackJack(pHand, dHand);
            blackJack(pHand, dHand);

            //Display Player and Dealer hands
            cout << "Your hand:" << endl;
            showCards(pHand, false);
            cout << "\n\nDealer's hand:" << endl;
            showCards(dHand, true);

            //If Player has an Ace, see if Player want's to have a soft or hard hand.
            pHand = checkSoftOrHard(pHand);
```

```
        //Check if user wants to hit
        while (g!=0)
        {
           if(userWantsToDraw() == true)
           {
              //Deal Player a card
                    //and display an explanation  of what is happening
                    pHand = addToHand(pHand, getTopCard());
                    deck=::deck;
                    cout << "The Dealer dealt you another card.\n" << endl;

                    //Display Player's updated hand
                    cout << "Your hand:" << endl;
                    showCards(pHand, false);

                    //If Player has an Ace, see if Player want's to have a soft or hard hand.
                    pHand = checkSoftOrHard(pHand);

                    //Check to see if anyone lost
                    checkBust(pHand, dHand);

                    //Check to see if anyone won
                    blackJack(pHand, dHand);
           }
           else
break;
        }

        //Dealer hits until at a soft 17
        hitUntilStand(dHand, deck, pHand);
        cout << endl;
```

```
                dHand=::Global_dHand;

                //Check to see if anyone lost
                checkBust(pHand, dHand);

                //Check to see if anyone won
                blackJack(pHand, dHand);

                //Compare scores and determine winner
                whoWins(pHand, dHand);
                cout << endl;

                //Display updated winnings multiplier
                cout << "Winnings multiplier: " << win << endl;
        }
        while(playAnotherHand());
        //menu();
}


/* ================================================
Name: dumpDeck
Desc: Loop through the deck array and print each value.
Parameters:
  deck stack<int> - System generated.
Return:
  void
================================================ */
void dumpDeck(stack<int> deck)
{
   stack<int> temp_deck=deck;
   int i=1;
```

```
	//loop through the deck array and print each value.
	while(!temp_deck.empty())
	{
		cout << i << ".) " << temp_deck.top() << endl;
    temp_deck.pop();
	}
}



/* ================================================
Name: initializeDeck
Desc: Create the cards background values so that suit
	is in the 100s place and rank is in the 10s place.
Parameters:
  deck stack<int> - System generated.
Return:
  stack<int> - returns the deck after updating it
================================================ */
stack<int> initializeDeck(stack<int> deck)
{
	//Declare local variables.
	int Rank = 101;
	int i = 1;

	//Hearts
	for (i = 1 ; i<=13; i++)
	{
		deck.push(Rank++);
	}

	//Diamonds
	Rank = 201;
```

```
  i = 1;
        for (i = 1 ; i<=13; i++)
        {
              deck.push(Rank++);
        }

        //Clubs
        Rank = 301;
  i = 1;
        for (i = 1 ; i<=13; i++)
        {
              deck.push(Rank++);
        }

        //Spades
        Rank = 401;
  i = 1;
        for (i = 1 ; i<=13; i++)
        {
              deck.push(Rank++);
        }
        return deck;
}


/* ================================================
Name: shuffle
Desc: Randomly rearranges the cards in the given
52-card deck.
Parameters:
  deck stack<int> - System generated.
Return:
```

```
  stack<int> - returns the deck after updating it
================================================== */
stack<int> shuffle(stack<int> deck)
{
   //converting stack deck into vector deck
   vector<int> temp_vec_deck;
   stack<int> temp_stack_deck=deck;
   while(!temp_stack_deck.empty())
   {
      temp_vec_deck.push_back(temp_stack_deck.top());
      temp_stack_deck.pop();
   }

         //Loop through deck an absurd amount of loops.
         for(int i = 0; i < 500; i++)
         {
                  //Define local variables
                  int R1 = getRandomNumber(0, temp_vec_deck.size()-1);
                  int R2 = getRandomNumber(0, temp_vec_deck.size()-1);
                  swap(temp_vec_deck[R1],temp_vec_deck[R2]);

                  //Clone first card for safe keeping
                  //Replace first card with a new card
                  //Replace the new card with the old card clone
         }
         for(int j=temp_vec_deck.size()-1;j>=0;j--)
   {
      temp_stack_deck.push(temp_vec_deck[j]);
   }
   deck=temp_stack_deck;
   return deck;
}
```

```
/* ================================================
Name: ShowCard
Desc: Displays the given card's rank and suit.
Parameters:
  card (int) - System generated.
Return:
  void
================================================ */
void ShowCard(int card)
{
	//Show nothing for non cards (ie.0)
	if(card == 0)
	{
		cout << "";
	}
	//Define Ranks.
	else
	{
		switch(card % 100)
		{
			case 1:
				cout << "A";
				break;
			case 11:
				cout << "J";
				break;
			case 12:
				cout << "Q";
				break;
			case 13:
```

```
                        cout << "K";
                        break;
case 14:
   cout  << "A";
   break;

                //For non-face cards, just use their 10s value as rank.
                default:
                        cout << card % 100;
        }
}

//Show nothing for non cards (ie.0)
if(card == 0)
{
        cout << "";
}
//Define Suits.
else
{
        //Hearts
   if((card >= 101) && (card <=114))
        {
                cout << char(heart);
        }

   //Diamonds
        else if ((card >= 201) && (card <= 214))
        {
                cout << char(diamond);
        }
```

```
                //Clubs
                else if ((card >= 301) && (card <= 314))
                {
                        cout << char(club);
                }

                //Spades
                else if ((card >= 401) && (card <= 414))
                {
                        cout << char(spade);
                }
        }
}

/* ================================================
Name: ShowCards
Desc: Displays the given card?s rank and suit.
Parameters:
  hand queue<int> - Player or Dealer hand
  hideFirstCard (bool) - Programmer generated.
Return:
  void
================================================ */
void showCards(queue<int> hand, bool hideFirstCard)
{
   //temp queue deck
   queue<int> temp_deck=hand;

        //Hide dealer's first card if true.
        if(hideFirstCard)
        {
```

```
                cout << "** ";
        }


        //Show dealer's first card if false.
        else
        {
                ShowCard(temp_deck.front());
                cout << " ";
        }
   temp_deck.pop();
        //Display all the cards in the deck or hand
        //by showing their rank and suit graphic.
        while(!temp_deck.empty())
        {
                //Show cards
                if(temp_deck.front()!= 0)
                {
                        ShowCard(temp_deck.front());
                        cout << " ";
                        temp_deck.pop();
                }

                //Show nothing for non cards (ie. 0).
                else
                {
                        cout << "";
                }
        }
}


/*================================================================
```

```
 Function:    getRandomNumber
 Description: returns a random number between given low and high
              values, inclusive.
              Note: include cstdlib (for rand) and ctime (for time).
 Arguments:
              low (I) - The lowest number to be generated
              high (I) - The highest number to be generated (must be > low)
 Return value:
              A random number between low and high (inclusive)
==============================================================*/
int getRandomNumber(int low, int high)
{
        static bool firstTime=true;
        int randNum;

        //if first time called, seed random number generator
        if (firstTime) {
                srand( static_cast<unsigned int>(time(NULL)) );
                firstTime=false;
        }

        //generate random number between given low and high values (inclusive)
        randNum = rand() % (high-low+1) + low;

        return randNum;
}


/* ================================================
Name: CardValue
Desc: Returns a given card's value.
Parameters:
```

```
  card (int) - Any one given card.
Return:
  int - The given Card's Value.
================================================= */
int CardValue(int card)
{
	//Declare local variable
	int cardVal;

	//Get the card's background 10s value and assign the card a point value
	switch(card % 100)
	{
		case 1:
			cardVal = 11;
			break;
		case 11:
		case 12:
		case 13:
			cardVal = 10;
			break;
    case 14:
			cardVal = 1;
			break;
		//For non-face cards, just use their background 10s value as point value.
		default:
		cardVal = (card % 100);
	}
	return cardVal;
}


/* ================================================
```

```
Name: getTopCard
Desc: Returns the top card off the deck.
Parameters:
  None.
Return:
  int - The given Card's Value.
================================================= */
int getTopCard()
{
        //Loop through the deck
        while(!::deck.empty())
        {
                //Clone card for safe keeping
                //Replace first card with the empty card value
                //Return the clone of the card
                topCard = ::deck.top();
                ::deck.pop();
                return topCard;
        }
}


/* ================================================
Name: addToHand
Desc: Adds the given card to an array representing a
players hand.
Parameters:
  hand queue<int> - A given hand.
  cardToAdd (int) - A card to add (always the top card from the deck).
Return:
  queue<int> - returns the hand after updating.
================================================= */
```

```
queue<int> addToHand(queue<int> hand, int cardToAdd)
{
   hand.push(cardToAdd);
   return hand;
}



/* ================================================
Name: playAnotherHand
Desc: Prompts user for input and determines if the
            user will play another hand.
Parameters:
      None
Return:
  True (bool)  - Player wants to play again.
  false (bool) - Player doesn't want to play anymore.
================================================*/
bool playAnotherHand()
{
   char choice;
         //Prompt user to see if they would like to play another hand.



         cout << endl << "\nWould you like to play another hand? (Y/N) ";
         cin >> choice;
         fflush(stdin);
         cout << "\n" << endl;

         //Go back to the main game logic function to restart if they do.
         if(choice == 'y' || choice == 'Y')
         {
                  playOneHand();
```

```
                return true;
        }
        //If they don't, exit the program.
        else if(choice == 'n' || choice == 'N')
        {
           alt_exit();
                return false;
        }
        else
  {
     cout<<"Entered option is not available!!"<<endl<<"Try Again!"<<endl;
     playAnotherHand();
  }
}


/*================================================================
 Function:    userWantsToDraw
 Description: Determines if user wants to hit.
 Parameters:
      None
 Return value:
                                True (bool) - Start loop.
                                False (bool) - Skip loop.
================================================================*/
bool userWantsToDraw()
{
   char draw;
        //Prompt user to see if they would like to enter a new time.
        cout << endl << "\nWould you like to hit or stand? (H/S) ";
        cin >> draw;
        fflush(stdin);
```

```
        cout << "\n";

        //If they do, return true and start the loop to draw another card.
        if(draw == 'h' || draw == 'H')
        {
                        return(true);
        }
        //If they don't, skip the loop.
        else if(draw == 's' || draw == 'S')
        {
           ::g=0;
                return(false);
        }
        else
  {
     cout<<"Entered option is not available!!"<<endl<<"Try Again!"<<endl;
     userWantsToDraw();
   }
}


/* ================================================
Name: naturalBlackJack
Desc: Checks to see if a given hand has a blackjack.
Parameters:
  pHand queue<int> - Player's hand.
  dHand queue<int> - Dealer's hand.
Return:
  void
================================================ */
void naturalBlackJack(queue<int> pHand, queue<int> dHand)
{
```

```
        //Define local variables
        int playerScore = getHandValue(pHand);
        int dealerScore = getHandValue(dHand);

        //If Player has blackjack and Dealer doesnt
        if((playerScore == 21) && (dealerScore != 21))
        {
                //Display message, compute new winnings multiplier, ask to play another hand.
                scoreBoard(pHand,  dHand);
                cout << "\n";
                cout << "Natural Blackjack! You win a 3:2 payout.";
                win = win + 1.5;
                backdoorKenny(pHand);
                if(::mode_choice == 'c' || ::mode_choice == 'C')
    {
       if(::bet != 0)
         update_bet(1.5);
     }
                playAnotherHand();
        }
}


/* ================================================
Name: blackJack
Desc: Checks to see if a given hand has a blackjack.
Parameters:
  pHand queue<int> - Player's hand.
  dHand queue<int> - Dealer's hand.
Return:
  void
================================================ */
```

```
void blackJack(queue<int> pHand, queue<int> dHand)
{
	//Define local Variables
	int playerScore = getHandValue(pHand);
	int dealerScore = getHandValue(dHand);

	//If Player has blackjack but Dealer doesn't.
	if((playerScore == 21) && (dealerScore != 21))
	{
		//Display message, compute new winnings multiplier, ask to play another hand.
		cout << "\n\n";
		scoreBoard(pHand,  dHand);
		cout << "\n";
		cout << "Blackjack! You win a 3:2 payout." << endl;
		win = win + 1.5;
		cout << "\n";
		cout << "Winnings multiplier: " << win << endl;
		backdoorKenny(pHand);
		if(::mode_choice == 'c' || ::mode_choice == 'C')
    {
      if(::bet != 0)
        update_bet(2.5);
    }
		playAnotherHand();
	}
	//If both Player and Dealer have blackjack.
	else if((playerScore == 21) && (dealerScore == 21))
	{
		//Display message, compute new winnings multiplier, ask to play another hand.
		scoreBoard(pHand,  dHand);
		cout << "\n";
		cout << "The Dealer and you both got Blackjack. Push in your favor at 1:1 payout!" << endl;
```

```
                win++;
                cout << "\n";
                cout << "Winnings multiplier: " << win << endl;
                if(::mode_choice == 'c' || ::mode_choice == 'C')
    {
       if(::bet != 0)
          update_bet(1);
    }
                playAnotherHand();
        }
}


/* ================================================
Name: getHandValue
Desc: Computes and returns the value of the given hand.
Parameters:
  pHand queue<int> - Player's hand.
Return:
  addCardValues (int) - Value of the given hand.
================================================ */
int getHandValue(queue<int> hand)
{
   queue<int> temp_hand=hand;
        //Declare local variables
        int addCardValues = 0;

        //Loop through the hand and add up the card values
        while(!temp_hand.empty())
        {
                addCardValues = addCardValues + CardValue(temp_hand.front());
                temp_hand.pop();
```

```
        }

        //Give the sum of the card values as the hand value
        return addCardValues;
}



/* ================================================
Name: hitUntilStand
Desc: Background logic to make the dealer hit until a soft 17.

Parameters:
  pHand queue<int> - Player's hand.
  dHand queue<int> - Dealer's hand.
  deck stack<int>  - The game deck.
Return:
  void
================================================ */
void hitUntilStand(queue<int> dHand, stack<int> deck, queue<int> pHand)
{
  queue<int> temp_dHand = dHand;
  queue<int> temp_pHand = pHand;
  int i=0;
        //Loop through Dealer's hand
        while(!temp_dHand.empty())
        {
                //If Dealer must hit
                if(getHandValue(dHand) < 17)
                {
                        //If Player chose Beginner level dealer stands at soft 17
                        //...add a card to the hand
                        //softOrHardAI(dHand,pHand);
```

```
	dHand = addToHand(dHand, getTopCard());
	deck=::deck;
	::Global_dHand=dHand;
		}

		//Display how many cards Dealer hit if Dealer has 17 or more points...
		else if(getHandValue(dHand) >= 17)
		{
			//Dealer didn't hit to get to 17
			if(i == 0)
			{
				cout << "The Dealer stands." << endl;
				break;
			}
			//Dealer hit once to get to 17
			else if(i == 1)
			{
				cout << "The Dealer hit a card and stands." << endl;
				break;
			}
			//Dealer hit more than once to get to 17
			else
			{
				cout << "The Dealer hit " << i << " cards and stands." << endl;
				break;
			}
		}

		//Run dealer AI again if player chose to play Expert
		//This is called again for the sake of redundancy.
		//if(level == 1);
++i;
```

```
   temp_dHand.pop();
        }
}


/* ================================================
Name: scoreBoard
Desc: Displays a simple score board with both the
        cards in each hand and the total points for
        each hand.
Parameters:
  pHand queue<int> - Player's hand.
  dHand queue<int> - Dealer's hand..
Return:
  void
================================================ */
void scoreBoard(queue<int> pHand, queue<int> dHand)
{
   cout<<"Scoreboard: "<<endl;
        //Display Player's cards/hand value
        cout << "Player hand: ";
        showCards(pHand, false);
        cout << " ("<< getHandValue(pHand) << "pts)."<<endl;

        cout <<"V.S." << endl;

        //Display Dealer's cards/hand value
        cout << "Dealer hand: ";
        showCards(dHand, false);
        //testshow(dHand, false);
        cout << " ("<< getHandValue(dHand) << "pts)."<<endl;
}
```

```
/* ================================================
Name: checkSoftOrHard
Desc: Asks Player if they would like their Ace to
        count as 1 or 11 points and then computes those
        changes.
Parameters:
  pHand queue<int> - Player's hand.
  dHand queue<int> - Dealer's hand.
Return:
  queue<int> - returns player hand after updating.
================================================ */
queue<int> checkSoftOrHard(queue<int> pHand)
{
   queue<int> temp_pHand=pHand;
   int i=0;
        //check cards in hand
        while(!temp_pHand.empty())
        {
                //Define local variables.
                int checkAce = CardValue(temp_pHand.front());
                int softOrHard;

                //If card in hand is an Ace prompt Player for input
                if(checkAce == 11)
                {
                        cout << "\n";
                        cout << "\nWould you like your position"<<i+1<<" Ace to count as 1 or 11 points?
(1/11): ";
                        cin >> softOrHard;
```

```
                //If Player chooses 1
                if(softOrHard == 1)
                {
                        //if(checkAce == 11)
                        pHand = change_value_of_ace(pHand, true, i);
                }

                //If Player chooses 11
                else if(softOrHard == 11)
                {
                        //if(checkAce == 1)
                        {
                                pHand = change_value_of_ace(pHand, false, i);
                        }
                }

                //If player doesn't input 1 or 11
                else if (softOrHard != 1 || softOrHard != 11)
                {
                        //Clears input error flags and removes everything currently in the input
buffer.
                        cin.clear();
                        cin.ignore(numeric_limits<streamsize>::max(), '\n');

                        //Display error message and restart the function
                        cout << "\nPlease enter the number 1 or 11." << endl;
                        checkSoftOrHard(pHand);
                }
        }
        else if(checkAce == 1)
    {
```

```
        cout << "\n";
                cout << "\nWould you like your "<<i+1<<"th Ace to count as 1 or 11 points? (1/11): ";
                cin >> softOrHard;



                //If Player chooses 1
                if(softOrHard == 1)
                {
                        //if(checkAce == 11)
                        pHand = change_value_of_ace(pHand, false, i);
                }

                //If Player chooses 11
                else if(softOrHard == 11)
                {
                        //if(checkAce == 1)
                        {
                                pHand = change_value_of_ace(pHand, true, i);
                        }
                }

                //If player doesn't input 1 or 11
                else if (softOrHard != 1 || softOrHard != 11)
                {
                        //Clears input error flags and removes everything currently in the input
buffer.
                        cin.clear();
                        cin.ignore(numeric_limits<streamsize>::max(), '\n');

                        //Display error message and restart the function
                        cout << "\nPlease enter the number 1 or 11." << endl;
                        checkSoftOrHard(pHand);
```

```
                        }
    }
  temp_pHand.pop();
  ++i;
        }
        return pHand;
}


/* ================================================
Name: check Bust
Desc: See Player or Dealer bust 21
Parameters:
  pHand queue<int> - Player's hand.
  dHand queue<int> - Dealer's hand.
Return:
  void
================================================ */
void checkBust(queue<int> pHand, queue<int> dHand)
{
        //Define local variables.
        int playerScore = getHandValue(pHand);
        int dealerScore = getHandValue(dHand);

        //Check if Player busts.
        //Display message, compute new winnings multiplier, ask to play another hand.
        if(playerScore > 21)
        {
                cout << "You bust with " << getHandValue(pHand) << " points." << endl;
                cout << "\n" << endl;
                win--;
                cout << "Winnings multiplier: " << win << endl;
```

```
                if(::mode_choice == 'c' || ::mode_choice == 'C')
    {
        update_bet(0);
    }
                playAnotherHand();
        }


        //Check if Dealer busts.
        //Display message, compute new winnings multiplier, ask to play another hand.
        else if(dealerScore > 21)
        {
                scoreBoard(pHand,  dHand);
                cout << "\n" << endl;
                cout << "The Dealer went bust. You Win!" << endl;
                cout << "\n" << endl;
                win++;
                cout << "Winnings multiplier: " << win << endl;
                if(::mode_choice == 'c' || ::mode_choice == 'C')
    {
        update_bet(2);
    }
                playAnotherHand();
        }
}


/* ================================================
Name: backdoorKenny
Desc: Checks to see if a given hand has a Backdoor Kenny.
            Any card valued at 10 points followed by an Ace as
            the first two cards in the deck.
Parameters:
```

```
  pHand queue<int> - Player's hand.
  dHand queue<int> - Dealer's hand.
Return:
  void
================================================= */
void backdoorKenny(queue<int> hand)
{
   queue<int> temp_hand=hand;
   vector<int> temp_vec_hand;
   //converting queue to vector
   while(!temp_hand.empty())
   {
      temp_vec_hand.push_back(temp_hand.front());
      temp_hand.pop();
   }

         //If index 0 is a 10 and index 1 is an ace
         if((CardValue(temp_vec_hand[0])%100 == 10)&&(CardValue(temp_vec_hand[1])%100 == 11))//I
did not use 1 as an argument because 11 is the default

                              //value and this happens before the user can choose.
         {
                  //Display message and compute new winnings multiplier.
                  cout << "\n\n";
                  cout << "You pulled a Backdoor Kenny!\n" << endl;
                  cout << "Win an additional 3:1 payout\n" << endl;
                  win = win + .25;
                  cout << "Winnings multiplier: " << win << endl;
                  if(::mode_choice == 'c' || ::mode_choice == 'C')
      {
         update_bet(3);
      }
```

```
        }
}


/* ================================================
Name: whoWins
Desc: Determines the winner based off of hand point
value comparison.
Parameters:
  pHand queue<int> - Player's hand.
  dHand queue<int> - Dealer's hand.
Return:
  void
================================================ */
void whoWins(queue<int> pHand, queue<int> dHand)
{
        //Declare local variables
        int playerScore = getHandValue(pHand);
        int dealerScore = getHandValue(dHand);

        //Display the scoreboard
        scoreBoard(pHand, dHand);

        //win tree
        //If Player scores less than 22
        if( (playerScore < 22)
                //If Player's score is better than Dealer's score
                && (playerScore > dealerScore)
                //If Player's score is under 22 but Dealer's score is over 21
                || ( (dealerScore > 21)
                        && (playerScore < 22) ) )
        {
```

```
            //Display message and compute new winnings multiplier
            cout << "\n";
            cout << "You win!" << endl;
            win++;
            if(::mode_choice == 'c' || ::mode_choice == 'C')
{
   update_bet(2);
}
    }
    //If you don't win, you lose or tie.
    else
    {
            //Tie
            //Display message
            if(playerScore == dealerScore)
            {
                    cout << "\n";
                    cout << "Push in the Dealer's favor. 1:1 payout." << endl;
                    if(::mode_choice == 'c' || ::mode_choice == 'C')
   {
      update_bet(1);
   }
            }

            //loose
            //Display message and compute new winnings multiplier
            else
            {
                    cout << "\n";
                    cout << "You lose." << endl;
                    win--;
                    if(::mode_choice == 'c' || ::mode_choice == 'C')
```

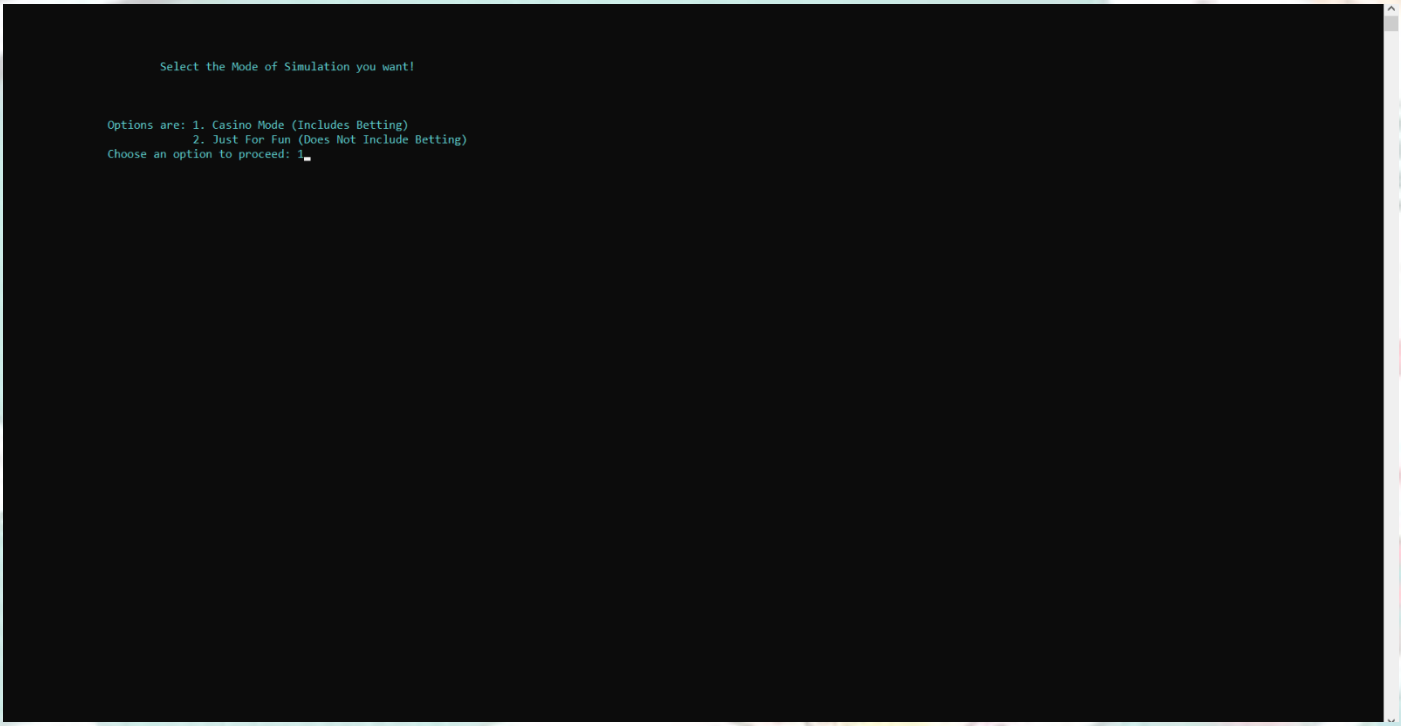
```
        {
           update_bet(0);
        }
                }
         }
}


/* ================================================
Name: checkSoftOrHardAI
Desc: Logical test based on if Dealer has an Ace.
           If Dealer does have an Ace Dealer should see if
           Dealer can make Dealer's hand 21, if not make
           Dealer's hand beat Player's hand with out
           going over 21.
Parameters:
  pHand queue<int> - Player's hand.
  dHand queue<int> - Dealer's hand.
Return:
  void
================================================ */
void softOrHardAI(queue<int> dHand, queue<int> pHand)
{
   queue<int> temp_dHand=dHand;
   queue<int> temp_pHand=pHand;
         //check cards in hand
         while(!temp_dHand.empty())
         {
                  //Find an Ace
                                    //...change it to 1 point if that will beat player...
                                    if (getHandValue(dHand) - CardValue(temp_dHand.front()) + 1 >
getHandValue(pHand))
```

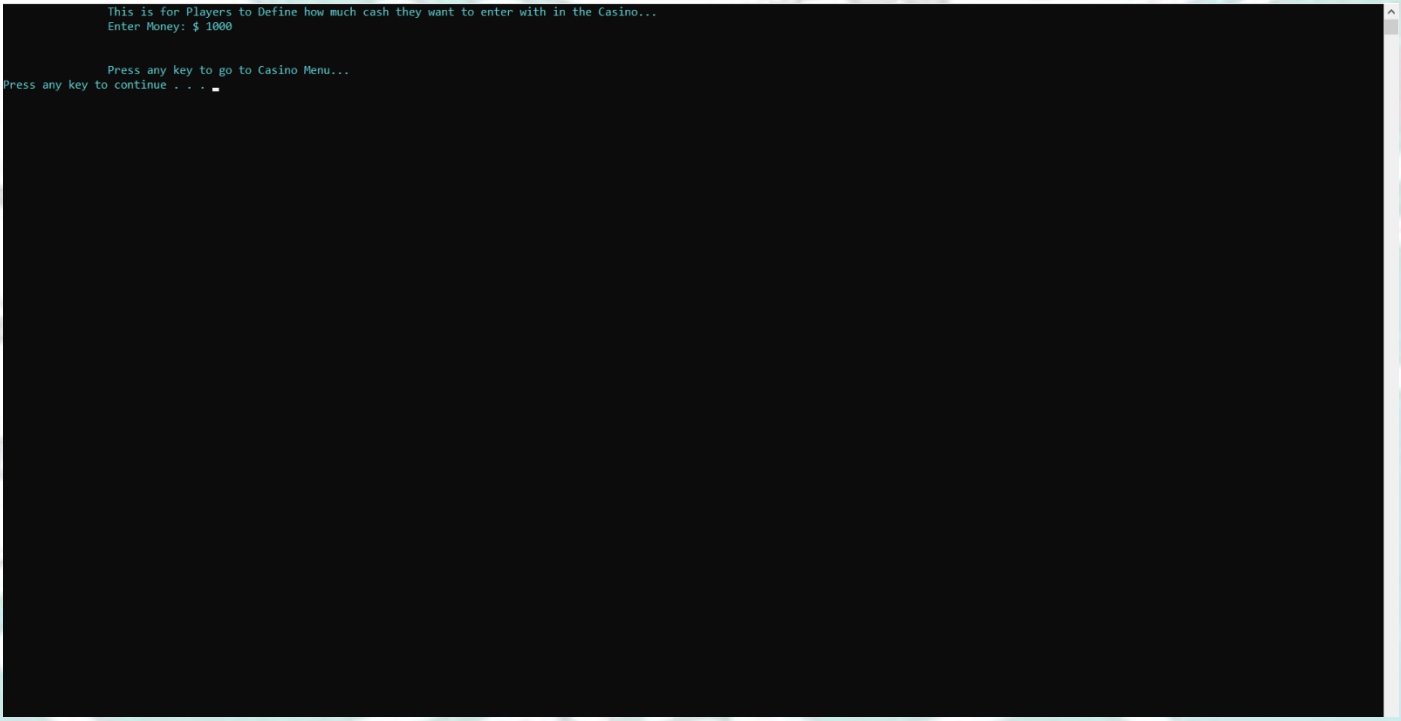
```
                {
                        //...and not bust.
                        if (getHandValue(dHand) - CardValue(temp_dHand.front()) + 1 < 22)
                        {
                                temp_dHand.front() = temp_dHand.front() + 13;
                        }
                }

                //Else if hand is a bust at 11, make it 1
                else if (getHandValue(dHand) > 21)
                {
                        temp_dHand.front() = temp_dHand.front() + 13;
                }
        //}

        //Else Ace is worth 1 point.

                //Change it to 11 points if that will beat player...
                if (getHandValue(dHand) - CardValue(temp_dHand.front()) + 11 >
getHandValue(pHand))
                {
                        //...and not bust.
                        if (getHandValue(dHand) - CardValue(temp_dHand.front()) + 11 < 22)
                        {
                                temp_dHand.front() = temp_dHand.front() ;//- 13;
                        }
                }
        }
   temp_dHand.pop();
        //}
}
```

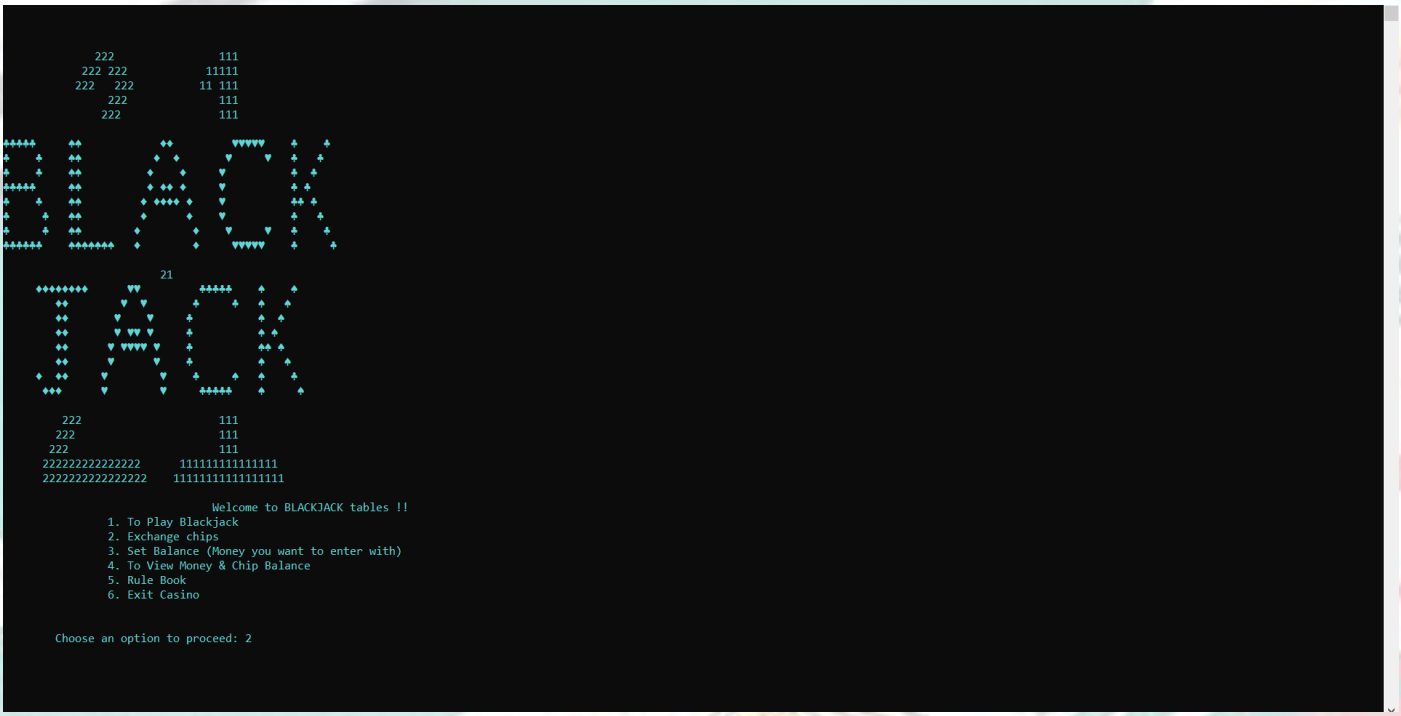
```
/* ================================================
Name: change_value_of_ace
Desc: Used to change the value of Ace depending on
      the input given by the user in the softOrHard function
      for the player's hand.
Parameters:
   pHand queue<int> - Player's hand.
   change_Ace (bool) - True if value needs to be changed else false.
   position_of_ace (int) - position of the Ace in the Player's hand.
Return:
   queue<int> - returns the player hand after updating
              the value of Ace(s).
================================================ */
queue<int> change_value_of_ace(queue<int> pHand, bool change_Ace, int position_of_Ace)
{
    queue<int> temp_hand ;
    if(change_Ace == true)
    {
       for(int i = 0; i < position_of_Ace; i++)
       {
          temp_hand.push(pHand.front());
          pHand.pop();
       }

       //changing value of i th Ace from 11 to 1
       if(CardValue(pHand.front()) == 11)
       {
          temp_hand.push(pHand.front() + 13);
          pHand.pop();
       }
       else if(CardValue(pHand.front()) == 1)
```

```
    {
      temp_hand.push(pHand.front() - 13);
      pHand.pop();
    }
    else
      cout<<"";

    while(!pHand.empty())
    {
      temp_hand.push(pHand.front());
      pHand.pop();
    }
    pHand = temp_hand;
  }
  else
  {
    cout<<"";
  }

  return pHand;
}
```

OUTPUT SCREENSHOTS






Welcome to BLACKJACK tables !!
1. To Play Blackjack
2. Exchange chips
3. Set Balance (Money you want to enter with)
4. To View Money & Chip Balance
5. Rule Book
6. Exit Casino
Choose an option to proceed: 2


1 $ = 1 chip
Money Balance = $ 1000
Chips Balance = 0 Chips
1. To exchange money for chips
2. To exchange chips for money
3. To go back to Main Menu
Choose an option to proceed: 1
Enter the amount of chips you want to buy: 1000

```
               Money Balance = $ 0
               Chips Balance = 1000 Chips
Place the Bet...(Max Bet can be 1000) : 1000
```

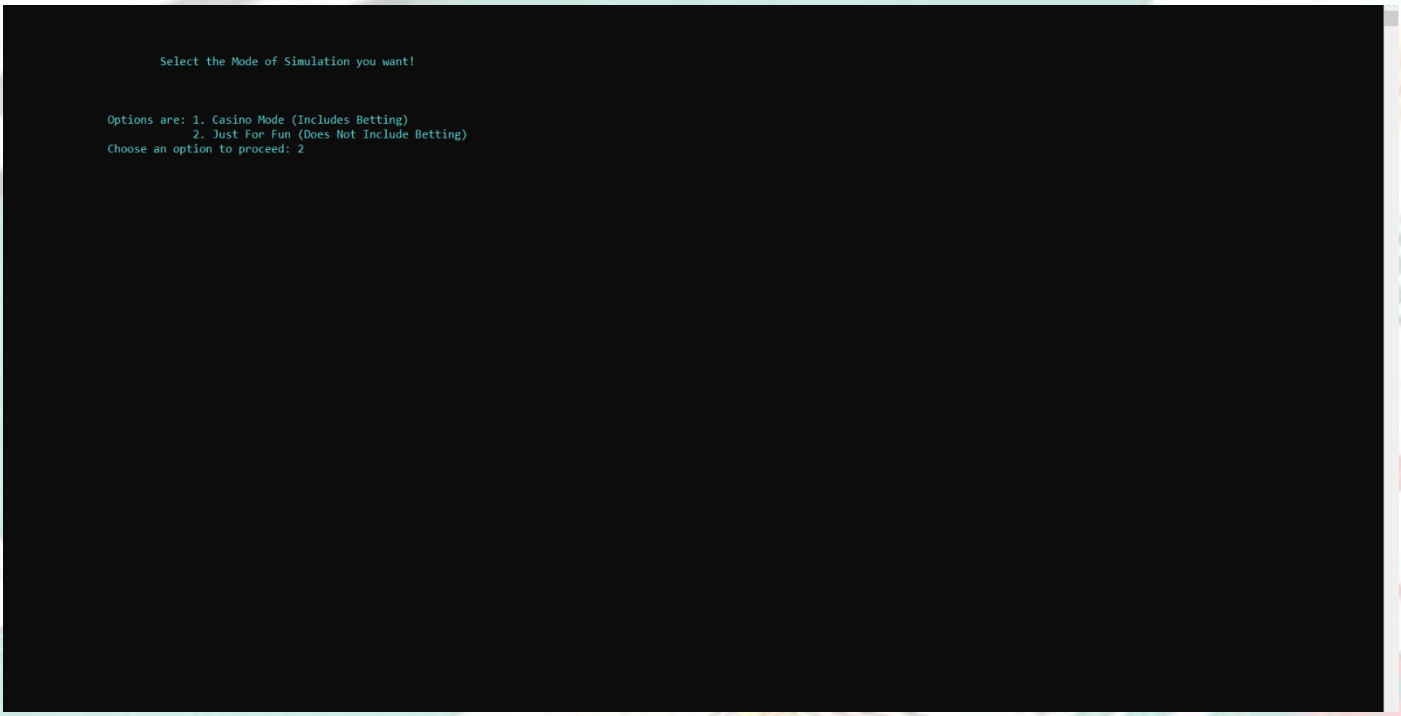
```
               Money Balance = $ 0
               Chips Balance = 1000 Chips
Place the Bet...(Max Bet can be 1000) : 1000
Bet Placed of 1000Chips...
////////////////////////////////////////////////////////////


The Dealer shuffled and dealt you each your cards.

Your hand:
Q♠ J♠

Dealer's hand:
** 2♦

Would you like to hit or stand? (H/S) s


Scoreboard:
Player hand: Q♠ J♠  (20pts).
V.S.
Dealer hand: 5♣ 2♦ 3♥ J♥  (20pts).

Push in the Dealer's favor. 1:1 payout.
Draw...Both Player and Dealer have equal hands...
Payout = Initial Bet
Player Chips Updated...1000

Winnings multiplier: 1


Would you like to play another hand? (Y/N) y
```

```
               Money Balance = $ 0
               Chips Balance = 1000 Chips
Place the Bet...(Max Bet can be 1000) : 1000
Bet Placed of 1000Chips...
////////////////////////////////////////////////////////////


The Dealer shuffled and dealt you each your cards.

Scoreboard:
Player hand: A♠ 10♣  (21pts).
V.S.
Dealer hand: 3♦ 4♣  (7pts).

Natural Blackjack! You win a 3:2 payout.

Would you like to play another hand? (Y/N)
```

```
               Money Balance = $ 0
               Chips Balance = 1000 Chips
Place the Bet...(Max Bet can be 1000) : 1000
Bet Placed of 1000Chips...
////////////////////////////////////////////////////////////


The Dealer shuffled and dealt you each your cards.

Your hand:
J♠ J♦

Dealer's hand:
** 10♣

Would you like to hit or stand? (H/S) s

The Dealer hit a card and stands.

Scoreboard:
Player hand: J♠ J♦  (20pts).
V.S.
Dealer hand: 2♣ 10♣ A♠  (23pts).


The Dealer went bust. You Win!


Winnings multiplier: 3.5
Win...Payout = 2 * Initial Bet
Player Chips Updated...2000


Would you like to play another hand? (Y/N) y
```

```
                Money Balance = $ 0
                Chips Balance = 10000 Chips
Place the Bet...(Max Bet can be 10000) : 1000
Bet Placed of 1000Chips...
////////////////////////////////////////////////////////


The Dealer shuffled and dealt you each your cards.

Your hand:
5♦ 3♠

Dealer's hand:
** 7♦

Would you like to hit or stand? (H/S) h

The Dealer dealt you another card.

Your hand:
5♦ 3♠ 9♥

Would you like to hit or stand? (H/S) h

The Dealer dealt you another card.

Your hand:
5♦ 3♠ 9♥ A♣

Would you like your position4 Ace to count as 1 or 11 points? (1/11): 1


Would you like to hit or stand? (H/S) h

The Dealer dealt you another card.

Your hand:
5♦ 3♠ 9♥ A♣ 8♠

Would you like your 4th Ace to count as 1 or 11 points? (1/11): 1
You bust with 26 points.


Winnings multiplier: 1.5
Loss...No Payout, Lost the chips placed on Bet...


Would you like to play another hand? (Y/N)
```

```
                    Thanks for Playing Blackjack!!

                    Project made by:
                     20103098 Dhairya Sachdeva B4
                     20103104 Ananya Kapoor B4

Process returned 0 (0x0)   execution time : 331.951 s
Press any key to continue.
```


Select the Mode of Simulation you want!
Options are: 1. Casino Mode (Includes Betting)
2. Just For Fun (Does Not Include Betting)
Choose an option to proceed: 2


//
The Dealer shuffled and dealt you each your cards.
Your hand:
3♥ 5♠
Dealer's hand:
** 7♥
Would you like to hit or stand? (H/S) h
The Dealer dealt you another card.
Your hand:
3♥ 5♠ 2♥
Would you like to hit or stand? (H/S) h
The Dealer dealt you another card.
Your hand:
3♥ 5♠ 2♥ K♣
Would you like to hit or stand? (H/S) s
The Dealer hit a card and stands.
Scoreboard:
Player hand: 3♥ 5♠ 2♥ K♣ (20pts).
V.S.
Dealer hand: 5♥ 7♥ J♣ (22pts).
The Dealer went bust. You Win!
Winnings multiplier: 2
Would you like to play another hand? (Y/N)

```
////////////////////////////////////////////////////////////


The Dealer shuffled and dealt you each your cards.

Your hand:
10♠ 5♦

Dealer's hand:
** 7♠

Would you like to hit or stand? (H/S) h

The Dealer dealt you another card.

Your hand:
10♠ 5♦ 7♥ You bust with 22 points.


Winnings multiplier: 1


Would you like to play another hand? (Y/N) n
```

```
////////////////////////////////////////////////////////////


The Dealer shuffled and dealt you each your cards.

Scoreboard:
Player hand: K♠ A♣  (21pts).
V.S.
Dealer hand: 9♥ 2♦  (11pts).

Natural Blackjack! You win a 3:2 payout.

You pulled a Backdoor Kenny!

Win an additional 3:1 payout

Winnings multiplier: 0.75


Would you like to play another hand? (Y/N)
```

CONCLUSION

Our program is a simple simulation of a casino game known as blackjack which we have developed using C++ language and data structures which include stacks, queues and vectors.

The basic objective behind doing this project was to get knowledge about C++ programming language and data structure implementation into a simple card game.

The game can not only be played for entertainment using the Just for Fun mode, but can also allow learning of blackjack and card counting, using the Casino mode, which allows players to place bets therefore simulating real-world games.

REFERENCES

1. https://www.geeksforgeeks.org/data-structures/

2. https://stackoverflow.com/questions/2301555/learning-algorithms-and-data-structures-fundamentals

3. Slides Provided in Classroom.