C++ and Object-Oriented Programming

Report: C++ Generated Game of Blackjack

Ms. Srinidhi Rajagopalan

Royal Holloway University of London

1. Introduction:

The closest version of the Blackjack, that we know now dates to the 17th century. It is one of the most popular gambling games played in almost every casino, which is attributed to its simplicity and the superior control the player has on the outcome over the house, compared to other casino games [2].

The objective of the game is simple,

(1) The player plays against the dealer. Both the player and the dealer are dealt two cards per game.

(2) The numbered card has the same value as their numbers, whereas the face cards: jack, queen and king all have 10 points.

(3) Both the cards need to have a total card value, equal to or as close to the value 21, such that the player has more than the dealer, but less than 21.

(4) It should be noted, that only one of the dealer's card will be face up, so then the player must make the decision to hit/stick, based on the value of his/her cards and the dealer's face up card. (To hit means to take another card, to add to your total card value and to stick means to stick with the cards you already have). The player can do this as many times as he/she wants.

(5) Once the decision to stick is made, then it's the dealer's turn to make similar decisions until, the best hand to win is reached. (Here, Hand refers to the set of cards the player/dealer has).

(6) If both the player and the dealer have Blackjack (that is, a total card value of 21) then the result is a push (tie) [1].

In this C++ generated game of Blackjack, the player gambles with money on his/her hand. Along with the basic rules, some additional options such as soft17, doubling down, insurance and surrendering have been included. The game can be played by any number of human players, against only the dealer or along with any number of computer-generated players with three playing strategies: Alex, Sidney and Basic. Basic is a player who displays the basic strategy which is described in the project script. The game is fully automated and does not require the adjustment of any feature. This version of Blackjack is written to be played with a 1:1 bet, that is the winner wins the same amount he/she bets. However, an option to switch to 6:5 bet is also made available.

2. Code Breakdown

The game has been built on five pillars (files) that is necessary to build the game: Card, deck, hand, player and house. Each of them holds a key feature of the game. The major aspects of each file are described as follows.

a. Card

Card is a header file that holds the features of a single card. Each card is described by the suit which is stored in the variable "num" and/or the face value which is stored in "suit". Both the variables are set to a "this" pointer, of the same name to increase readability of the code.

For accessing the cards, an if statement that uses indices from 1 to 13 is used. The face cards jack, queen, king and ace are accessed specially by a number to string function (to_string).

Number to string function:

card += to_string(num);

This function returns the string for the specific value (number) [3].

b. Deck

The deck contains all the 52 cards split into 4 suits: spade, heart, clover and diamond. Each suit has 13 cards that runs from ace to 10 and jack, queen and king. Each of the suits are initialized in a for loop with a vector function (push_back), which adds a new card to achieve a set of 13 that is saved in the vector "cards" [4]. Two functions are used to control the deck: shuffle and get card. The shuffle function uses (random_shuffle) which is a standard library function that is used to randomly shuffle the elements between a given set and takes the first and last element in your set as arguments [5]. In this case, cards.begin () and cards.end () are the two functions that returns the first and last element respectively [6]. The variable "nextCardIndex" is used to store the index of the next card that increments it like an iterator to access the next card in the deck.

c. Hand

Each hand is set to have two cards. The Hand file contains functions that are used to "insert" a card into a hand, count the value of the hand and check for blackjack. Here, the hand is initialized using a vector function clear (). This function empties the vector container thus, initializing it with a size zero [7]. The insert function also checks for Blackjack. This is done by using the logical

operators “AND” and “OR”. The if statement checks it twice, that is, for both arrangement options for a hand.

d. Player

Player contains all the options that are provided for each player and strategies for the computer players. The functions for the class Player include the following:

i. Checking if the player wants to play under Soft17
   The Soft17 rule states that, when the dealer has two cards whose total value sums to 17 and one of the cards is an ace then the dealer must stick and cannot hit. This feature can be turned on or off before the beginning of the game.
ii. Checking if the player wants to bet according to the 6:5 rule instead of the default 1:1
   The 6:5 rule mandates that for every 6$ the player bets and wins, he/she gets 5$ back. This is not a great deal as it benefits the house. So, the decision to turn it on is again left to the user.
iii. The kind of player
   There can be any number of human player and three kinds of computer players.
iv. Setting a name for the human players
v. Getting the bet money
   The human player can bet anywhere between 1-10$ at a time. If the human player bets more than 10$ then the bet is set to 10$. Whereas the computer players will always bet only 5$.
vi. Once the hand is given, it checks if the human player wants to: double down/insure/surrender
   Doubling down: After a player has bet, he/she will receive two cards. If the player chooses to double down, then the bet money will be doubled down and another card will be dealt.
   Insurance: When the dealer’s face up card is an ace then the dealer offers all the players insurance. That is, they can insure half of their bet money. Now, in the case of a Blackjack, the user will still get back half of the bet money instead of losing it all.
   This is easily achieved by setting the following arithmetic methods:

Insurance = (int)round(bet_money/2.0);

Bank_roll -= insurance;

Surrender: Surrendering is an option, where the player can choose after being dealt with the first round of cards. This would mean that the player does not have to hit/stick and gets half of the bet back. If the bet money is an odd number, it is rounded off to the nearest whole number. [8]

Bank_roll += (int)round(bet_money/2.0);

vii. The make decision function finds out if the players want to hit/stick

1. The human player can take their decisions autonomously.
2. Alex plays it quite recklessly. He hits whenever his score is less than 21 and stick only if it is over. Whereas Sidney plays it safe. She hits, only her score is less than 11 otherwise, she sticks.
3. The Basic player has a good strategy:

(a) If Basic has been dealt with a total card value greater than or equal to 17, then he sticks regardless of the dealer's hand.

(b) If it is greater than 13 and the dealer's hand is less than or equal to 6, then he sticks.

(c) If the score is equal to 12 and dealers' card is greater than or equal to 4 but less than or equal to 6, the he sticks.

(d) Otherwise, Basic will hit. This strategy has been set for Basic using a simple if else nest.

viii. Once the players stick and the dealer plays and sticks, the scores are settled and the winner get their bets back, while the losers lose their whole bet money or half if they had insured it in case of an ace with the dealer. The amount the player has is displayed as bankroll + bet money + won money or bankroll (with the bet money subtracted)

ix. The print function is written elaborately with the standard printing function to make display the game as close to a real-world console as possible to make it easy to play the game.

e. House

The house file establishes how the dealer plays. Its golds the following features:

i. The get card function here will show only card indexed 1. The card indexed 0 will be displayed as XX.

ii. The dealer plays by soft17 rule if it is enabled in the beginning of the game.

iii. Otherwise, the dealer always hits if it is less than 17, and always sticks if it is greater or equal to 17.

These decision-making functions are created with simple logic statements and if else nests.

3. The Game

The game is the main file that runs together all the main five files. The variable TEST MODE is set to run by default mode. However, to test the computer players individually and see them play, the user may change the value of TEST MODE to 1 for Sidney, 2 for Alex and 3 for Basic.

The getYN function gets the yes or no answers for the various options posed to the player. The srand () function is used to deal a new deck each time [9]. This pseudo random function ensures that it is new every time, the seed argument is set to time, that is, the time the game is played.

```
------------- Begin Game -------------
Do you enable Soft-17? (y/n)
n
Do you enable 6:5 Blackjack? (y/n)
n
How many human players?
1
How many computer players?
0
Human player 1 name:Srinidhi
 How much do you want to bet? (1-10)
10
Srinidhi bet $10
---------------- Deal ----------------
Let's deal some cards!
Dealer:         XX DK
Srinidhi:       D9 D5 (14)
------------------------------------
Srinidhi:       D9 D5 (14)
Do you want to enable Doubling Down? (y/n)
n
Do you want to surrender? (y/n)
n
Do you want to hit(y/n)? y
Srinidhi hits!
```

Fig 1. The Blackjack Console

4. Conclusion

The program has the following lack of features/ problems:

1. Sometimes, the total card value of the player's hand seems to be calculated wrong, though this is a rare occurrence and mostly happens when the user enters zero human players and watches the computer players play. One way to use it is to change the variable TEST MODE or to handle the error within the code.
2. When a computer player pushes, it does not show what the dealer's face down card is, this makes it difficult to judge, as the user is not able to see the dealer's cards.
3. It'd be useful if the global variable TEST MODE in Game.h was made into an option for the user to pick during run time as an input.
4. The game is written such that each player plays against the dealer and not against each other, it could be rearranged such that it can be played among the human/computer players.
5. The options the player has, is asked repeatedly making it tire for the user to play. Introducing stationary buttons for surrendering, doubling down, insuring etc. would make the game easier and comfortable to play with
6. Once the current code is perfected, the next step would be to introduce more players like Basic but with more elaborate strategies and card counting.

High-low is a separate game that was created before Blackjack to create a basic structure. Due to lack of time the code was not very well written. The deck does not shuffle right as it does not use a pseudo random function. This means it poses the same card each time the user plays.

Reference

1. Chongwu, R. (2017). Blackjack and probability, Maths 190S-Hubert Bay.

2. Scarne, J. (1986). Scarne's new complete guide to gambling (reprint). Simon & Schuster.

3. <http://www.cplusplus.com/reference/string/to_string/>

4. < http://www.cplusplus.com/reference/vector/vector/push_back/>

5. < http://www.cplusplus.com/reference/algorithm/random_shuffle/>

6. < http://www.cplusplus.com/reference/iterator/begin/>

7. < http://www.cplusplus.com/reference/vector/vector/clear/>

8. http://www.cplusplus.com/reference/cmath/round/

9. < http://www.cplusplus.com/reference/cstdlib/srand/>