Game of Blackjack

By Suhan Ree

1. Introduction

In this document, I describe the rules of the blackjack, chosen for this coding challenge, and show briefly how the code was implemented.

In the rules, the game is played by two players, a player (a user) and a dealer (played by the computer), and most basic actions are implemented except splitting. One thing to note is that I chose “S17” (soft 17) rule, which determines when the dealer stops hitting. For example, if the dealer’s hand has an ace and a 6, which is called the “soft 17” because its value can be interpreted as either 7 or 17, the S17 rule says that this hand should be regarded as 17 and that the dealer should stop hitting, since, in the blackjack, the dealer should stop if the dealer’s hand becomes 17 or greater. If the value is greater than 17, the dealer has to stop hitting even if the dealer has an ace and uses as 11.

To implement this game, I chose C++, and made 4 classes: Card, Decks, Hand, and Game. The Card class represents each card, and the Decks and Hand classes represent groups of cards. A Decks object contains all cards used in the game, and a Hand object is for cards in a hand of a player or a dealer at a given moment. The Game class manages the flow of the game, while dealing with text-based user interactions using 5 stages.

2. Rules of the game

There are two players: a dealer, played by a computer, and a player, played by a user. The game can be played as many rounds as the player can or wants, and the winner is determined each round. At the beginning of the game, the player chooses how many decks are used for the game, where each deck consists of 52 cards, 13 for each suit (Club, Spade, Heart, and Diamond); here the number of decks can be 1 or 2 or 4. You, the player, start with 100 chips and can bet at least 1 chip each round. The maximum number of chips a player can bet at each round is set at 5 chips here. The dealer is assumed to have 10,000 chips in the beginning. If either the player or the dealer loses all chips, the game ends automatically.

At each round, the objective of the player is to win the bet by creating a card total that is higher than the value of the dealer's hand, but not exceeding 21 (called, "busting"). The value of a hand is determined by summing over values of all cards in a hand: 2~10 have the same values as the face values, while J, Q, and K (face cards) are counted as 10 and an ace, A, can be counted as either 1 or 11. The suits of the cards don't have any meaning.

Once the amount of the bet is chosen for each round, two cards are dealt at the beginning of the round: both cards of the player are revealed, while only one card is revealed for the dealer. The player has two options: Hit or Stand.

(1) Hit: Take another card from the dealer. If the player's hand is not busted by exceeding 21, the player has another chance to choose to hit or stand,
(2) Stand: Take no more card. Then, the player's value is determined by summing over all cards in the hand (A can be either 1 or 11, whichever is better).

If the player gets busted by exceeding 21, the dealer wins. If the player choose to stand at a value 21 or lower, the dealer should hit until the value is 17 or greater (the ace, A, is counted as 11 as long as the sum is less than 21, even when the sum becomes 17, which is called "S17" rule). If the dealer gets busted, the player wins. If both are not busted, the winner is determined by comparing values; the player wins if the player's value is greater, and the dealer wins if the dealer's value is greater. If tied, the bet is returned to the player.

If the first two cards has the value 21 by having an ace and a 10-valued card (10 or J or Q or K), it's called the "Blackjack" and wins every hand except another blackjack (it's a tie if both get the blackjack). The value of an ace can be either 1 or 11, and it is not hard to determine which value to use: we choose 11 as long as the resulting sum doesn't exceed 21.

3. Implementation

The language of choice is C++, which I am most familiar with, for this challenge (only using features supported by C++03). First, I describe classes I made for this program; second, I explain how the flow of the game is managed; third, I show how to run the code and how to play the game using this program briefly; and lastly, I describe how each card is represented during the game.

3.1. Classes

Card class

This class represents a card with a number (rank) and a suit.

`Card(int num , char suit)` : constructor (**`num`**: value of the card, 1~13; **`suit`**: suit of the card, 'c', 's', 'h', and 'd', representing club, spade, heart, and diamond, respectively).

`int getValue()` : returns the value of the card (return values: 1~13).

`string getRank()` : returns the rank of the card (return values: A, 2, 3, …, 9, 10, J, Q, K, as string objects).

`char getSuit()` : returns the suit of the card (return values: 'c', 's', 'h', and 'd', as characters).

Decks class

This class represents cards of decks to be played in the game of blackjacks. The player can choose the number of decks to be used in the game (only 1 or 2 or 4 is allowed here).

`Decks(int nDeck)` : constructor (**`nDeck`**: number of decks to be used, 1 or 2 or 4).

`void create(int nDeck)` : create the cards using **`nDeck`** decks.

`void shuffle()` : shuffle all cards and set the current card as the first card.

`Card deal()` : returns a Card object pointed currently, and set the pointer to the next card.

`void print()` : writes all cards of given decks in the currently shuffled state on the screen.

Hand class

This class represents a hand (currently held cards) of a player or a dealer.

`Hand()` : constructor, creates a hand with no card.

`void addCard(Card card)` : add a card given as an argument to the hand.

`int getValue()` : returns the current value of a hand (summed over card values of a hand).

`int size()` : returns the number of cards in the hand.

`bool blackjack()` : returns true if the hand is the blackjack (getting 21 with two cards); false if not.

`void removeAllCards()` : removes all cards in the hand.

`void print(bool hideFirst)` : writes all cards of the hand on the screen (if **`hideFirst`** is true, the first card of the hand is hidden; shown if false).

Game class

This class manages the game.

`Game()` : constructor, creates cards and two hands for a player and a dealer.

`void play()` : plays the game (managing flows of the game, and letting other functions interact with the player).

3.2. Flow of the game: 5 stages

To manage the flow of the game, I split the game into 5 stages, and implemented 5 private member functions in the Game class, each representing a stage. The main thing these functions do is I/O, only using standard I/O (text-based IO) for respective stages. They write information and current states of the game on the screen, and also ask inputs from the player during the game.

Stage1: Begin the game [**`char beginGame()`**]

Output:

1. a banner for the game,
2. a question of how many decks to be used,
3. beginning total chips for the player,
4. a question of next step: whether to start a new round, or to show the rules, or to quit.

Input:

1. how many decks to be used (1/2/4; default: 1),
2. next step (n/r/q; default: n) (n: new round, r: show rules, q: quit).

Stage2: Begin a round [**`void beginRound()`**]

Output:

1. showing the remaining chips of the player,
2. a question of how many chips to bet, and how many chips are bet.

Input:

1. a number of chips to bet (1~5; default: 1).

Stage3: In a round [**`char inRound()`**]

Output:

1. showing the current hands of both players (with the first card of the dealer hidden),
2. a question of next step: whether to hit, or to stand, or to show the rules, or to quit.

Input:

1. next step (h/s/r/q; default: h) (h: hit, s: stand, r: show rules, q: quit).

Stage4: End a round [**`void endRound(char input)`**]

Output:

1. showing results of the round, and how many chips are gained or lost by the player,
2. showing the remaining chips of the player,
3. a question of next step: whether to start a new round, or to show the rules, or to quit.

Input:

1. next step (n/r/q; default: n) (n: new round, r: show rules, q: quit).

Stage5: End the game [**`void endgame()`**]

Output:

1. showing the remaining chips of the player, with the ending banner.

Input: none.

3.3. How to run & How to play

To make the code easy to evaluate, I put every code into one file, called *Blackjack.C* (~600 lines). I used the gcc compiler (g++, version 4.4.7) in a CentOS virtual machine, but it should work fine in other platforms with other compilers, I assume.

Playing the game is self-explanatory once you execute the program. At first, the program asks how many decks to be played (1 or 2 or 4 is allowed here) [stage 1]. Keep in mind that, to reduce confusion, only the first character (excluding white spaces) is used in an input of a line from the player throughout the whole game, and that following characters before 'Enter' will be ignored, if any. Hence, if you type '1', '0', and 'Enter' in this case, the program will only read 1 and use it as the number of decks. Valid input characters and default values (if no character is given by just typing 'Enter' for example) are given in all cases of user inputs. If the input character is not valid, the program will ask for an input again. Then a new round begins [stage 2] by asking how many chips will be bet in this round. Only 1~5 chips can be bet in a round. Next, two cards are dealt to both players (only showing one card for the dealer), and a player is given a chance of choosing "hit" (getting one more card) or "stand" (no more card) [stage 3]. The round ends if a player get busted or a player chooses to stand. Then, the winner of the round is determined and chips are given to the winner [stage 4]. At this stage, the player can choose to quit the game [stage 5] or start a new round [going back to stage 2]. If the player chooses to quit, the remaining chips are shown and the game ends.

All possible user inputs during the game are: 'n' (new round), 'r' (displaying rules), 'h' (hit), 's' (stand), 'q' (quit), and numbers 1 to 5 (used for the number of decks, or the number of chips to bet). If the player types only 'Enter' for an input, the default value shown in the screen will be used in all cases.

3.4. Card representation in game

The ranks: A (ace), 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K.

The suits: c (club), s (spade), h (heart), d (diamond).

For example, "A(s)" stands for the spade ace, "10(d)" stands for the diamond 10, and "Q(h)" stands for the heart queen.

4. Summary

The logic of the game is not complicated, but the main point of this coding challenge seems to be how we effectively establish the user interface using only the standard IO (text-based IO) dealing with technical tradeoffs. It looked simple at first, but I realized it is much harder than implementing using a GUI-based IO. Frankly, I already have written a code for the game of blackjack before, using Python and a library for GUI (~250 lines of code in Python). This time, though, making a clean text-based user interface has been relatively harder (~500 lines of code in C++, excluding rule texts). To establish an effective user interface, I split the game into 5 stages, and each stage is implemented as a separate function in the Game class. To make the user input as little as possible, I chose a default value for every user input, so that the user can only type 'Enter' whenever a user input is needed (except to stand in a round, or to quit the game).