Francis Hoang | CSC 5- 41202 | February 3, 2016

Project 1

Blackjack

# Introduction

This is essentially a C++ implementation of Blackjack since it seemed like something doable based off what we’ve learned so far this semester. Currently, at the time of this writing, it’s playable but it’s missing a dealer to play against, as well as money to bet with. It is simple Blackjack, currently with an infinite amount of decks so it’s theoretically possible (but highly unlikely) to get 21 Aces dealt. The player is dealt two cards, and from there they can keep drawing till they bust or choose to stand.

# Summary

This project mostly uses what we were taught, though I still have not thought up a good use for an int primitive type since Blackjack is a small numbers game, and short generally suffices for most calculations. I ended up implementing arrays since I wanted the player to know what their hand was, as well as eventually the dealer’s hand, so arrays were useful to store those values. The first big issue was handling the Ace since it could represent either a 1 or 11 that the player could change at will. I opted to have the player prompted every loop should they have an Ace, though it does interrupt gameflow a bit. In the process, I also realized I would have to reset the player point total to 0 and recalculate it again entirely instead of switching 1 and 11 to account for the unlikely scenario the player had multiple Aces which they may change as they please. I ended up repeating a lot of code due to this, which might be more smoothly handled as functions that I haven’t quite figured out how to implement just yet. How the game is ended to break the loop is somewhat inefficient as well at the moment, also a result of the recalculation of Ace since there was a period where the code would keep running even if the loss condition had already been reached. I had originally intended to use a for-loop for the drawing/hit me phase, since it would be a much smoother use of calling the values from the array than a separate counter, while the do-while was simply just to ask to play again, but I was somewhat haphazardly coding at the time so the do-while loop is the current draw phase for the moment. I may change it back to a for-loop in the future if I have time. Future plans are still to implement the dealer to add an additional win/loss condition for the player, as well as adding the gambling component of betting money. Other possible plans would be limiting the amount of decks, which would add the complication of having to account for a limited number of cards, and the ability to split your hand, which would involve having to account for another array.

# Summary v3

There was actually a V2 which basically a finished v1, and added gambling and a dealer as planned, as well as some polish, specifically in using the for-loop as originally intended. V3’s goal is mainly polishing it further by using functions to clean up the repetitive code. There’s still some code that’s repeated though, specifically the win-loss checks, which is mostly because I did not want to take the time to figure out how to make a function that would implement 4 or 5 other functions, where I would have to account for all those parameters for the sub-functions. There are also two deal functions which probably could be consolidated into one as well, the initial 2-card deal, and the hit me deal. I mostly kept both deal functions since there was enough slight variation in the output that I didn’t want to combine the two. It also conveniently serves as an example of an overloaded function though. I have replaced the hasAce Boolean with a Boolean search function which basically checks for an Ace. The Ace function may not be in its most ideal place, and I only left it there as a remnant of the old code in case I break something if I do move it. Based off the pseudocode, I should ideally place it before checking win conditions, so I don’t recall why I chose to place it in the middle, other than vaguely remembering running into loop errors when it was elsewhere. I have also created a sort function which sorts the hand of both the dealer and player. The main thing I’m missing is a 2 dimensional array, I believe. If I had more time, I assume a 2d array would be useful if I were to split the cards. Other missing features in this version of blackjack is the ability to double down, and potentially more players. I mostly didn’t add them due to lack of time, though each new player and new hand would also add a layer of complexity that I did not feel like covering just yet. I imagine making the players separate objects of a Player class might be better for that scenario to account for the various variables they share.

### Pseudocode

*Initialize*

*Ask for input from player how big their wallet is.*

*Store the initial wallet amount.*

*Do*

*Reset point and Boolean values to game start*

*Ask for input from player how much to bet*

*Subtract the bet from their wallet*

*Deal the first two cards to the player and the dealer*

*Sort and output the player’s hand and the dealer’s face card*

*Calculate the initial hand and add to the point value of player and dealer*

*Output the point value of the player’s hand*

*Check if the player has a winning hand of 21, or higher (WIN/LOSS CHECK 21)*

*End condition is true*

*If they have a hand greater than 21*

*Bust, increment the loss counter*

*Else*

*Reveal the dealer’s hole card*

*Compare hands*

*If they’re equal and it’s the player’s initial hand*

*It’s a draw*

*Return bet money to wallet*

*Increment draw counter*

*Else if it’s the dealer’s initial hand = 21 and player has 21 from 3 or more*

*Cards*

*Dealer has blackjack and wins*

*Increment loss counter*

*else If the dealer’s hand < 17*

*if it’s the player’s initial draw*

*Blackjack win*

*Bet + bet\*1.5 is added to wallet*

*Increment win counter*

*Else if it’s not the player’s initial draw*

*Dealer draws till hand is 17 or higher*

*If hands are equal*

*Draw*

*Return bet money to wallet*

*Increment draw counter*

*Else*

*Player won*

*Bet\*2 added to wallet*

*Increment win counter*

*Check if the player has an Ace (could save a lot of code if this is moved to earlier)*

*If has an ace, ask if player wants to alter value to 1 or 11*

*Recalculate point total of hand if needed*

*If changing the Ace has not changed card value to 21 or higher*

*Ask player to hit?*

*If yes, deal them a card*

*If no, player chooses to stand*

*End condition is true*

*Reveal the dealer’s hole card*

*Compare hands*

*If dealer’s initial hand is 21*

*Dealer blackjack wins*

*Increment loss counter*

*else If dealer’s hand is less than 17*

*Dealer draws cards till hand is 17 or more*

*If dealer hand > player hand*

*Player loses*

*Increment loss*

*If equal hands*

*Increment draw*

*Return bet to wallet*

*If player hand > dealer hand*

*Player won*

*Increment won*

*Add bet\*2 to wallet*

*Else if hand is now 21 or higher from changing Ace*

*End condition is true*

*(WIN/LOSS CHECK 21)*

*While end conditions are not met*

*I*

*Ask to play again*

*Player inputs answer*

*If yes*

*Check wallet amount*

*If 0, output that they’re broke and end game*

*Else repeat game*

*Output total wins, losses, and draws*

## Checklist

**Chapter 2**

int

short

float

char

bool

string

Arithmetic Operators

If-else statement

Ternary operator

Comparison operators

while loop

do-while loop

constants

**Chapter 3**

Boolean expression

Multiway if-else statement

switch statement

increment and decrement operators

for statement

break statement

**Chapter 4**

function call

random number generator

type casting

function prototypes

Boolean function

global constants and variables

overloading function

**Chapter 5**

void function

function calling function

Call by reference

call by value

mixed parameters

stubs and drivers

**Chapter 6**

input stream

line 42

line 36,37

line 39

line 40

line 41

N/A

line 85

line 66

line 118

line 65

line 245

line 51

N/A

line 122

line 82

line 258

line 69

line 56

line 427

line 34

line 34

line 34

line 20

line 24

N/A

line 20,23

line 20

line 439

line 20

line 23

line 23

N/A

N/A

output stream  
Exit statement

get function

put function

**Chapter 7**

Array

2D array

Search function

Sort function

line 46, 231

N/A

N/A

N/A

line 38

N/A

line 423

line 435