Project 1

Title

Blackjack v8.0

An asinine recreation of the Blackjack minigame
from Clover Studio's God Hand (2006)
for the Sony PlayStation 2.

Course

CSC/CIS-5

Section

40404

Due Date

February 3, 2024

Author

Noah Garthwaite

(a.k.a. Daisy Garthwaite)

(a.k.a. Dahlia Meat)

(a.k.a. Darlene Petal)

(a.k.a. Dippy)

(hey buddy, back off)

Table of Contents

Introduction

or:

"Why are you acting like this?"

One may ask: "What does an 'asinine recreation' of a Blackjack minigame entail?" Let's say, hypothetically, for the sake of argument, One did ask. In response, I would say: "Well, One, allow me to feed you some context as to Dr. Lehr's expedited 6-week winter semester CSC/CIS-5 class."

This project was supposed to be due three days ago until it was to be due six hours from now, today - until today, where it's now due to be due tomorrow. This means not only was I able to leave my desk for a full ninety minutes this afternoon and speak to my life partner for the first time since the start of this class, but I also no longer need to rush this write-up! So, I'm going to have fun with this; because although this project was incredibly engaging, it was a torturous amount of work over such a short development period that I need to cope with humour lest I be institutionalized.

I have about a year's worth of experience with the C programming language, which is to say I took Harvard's free online self-paced 10-week CS50 course over a period of 52 weeks and proceeded to not touch C again until this January. I loved working with C, however. Once I finally understood pointers enough to only segfault four times per line of code added, I was hooked on the nitty-gritty of static programming. I am a twenty-five year-old writer going back to school with a new major and aspirations of breaking out into a career as a video games programmer. The knowledge I hope to gain from working in such a hugely rewarding and thoroughly labour-humane industry will fuel my pipe-dream independently-developed video game *YOU MUST LOVE THE VOID*.

This class, therefore, was an exciting prospect. C++ is an industry standard for AAA video games made in engines such as Unreal, and can also be used for customization of modules in smaller-scope engines such as Godot. I had never touched C++ before, so I looked forward to being forced to use it by my own hands now that I am in my mid-twenties and the only person I risk disappointing with anything less than a 4.0 is myself and everyone I want to provide for and my Mom and God.

As I suspected: programming a game, even one as simple as Blackjack, was addicting. Because I meticulously log all of my personal productivity hours, I can confidently say that at the time of writing this sentence, I have spent 2,822 minutes on this project over the course of seven days. That's just over 47 hours. During our first lecture, Dr. Lehr stated that this class is "a full-time job". If this is true: Mark, you owe me overtime.

Though, 47 hours may seem like quite a lot of time for someone with a familiarity of C to make a simple Blackjack game in C++. Well, for one, I'm a perfectionist to a serious detriment, so leave me alone. Secondly, this project was forced to be severely limited in architecture.

See, this project was assigned before we as a class had been taught even the most fundamental of programming data structures (e.g. arrays) or subroutines (e.g. programmer-defined functions). In our homework, I routinely found myself using these paradigms without even thinking. My life is too short to hard-code ten user-input integers rather than declaring one array and filling it with a for-loop. Wouldn't that be silly? Wouldn't that be silly? Wouldn't tha

Limitations

or:

"How I Learned To Stop Worrying

and Program Like YandereDev"

```
// variable declarations    // used for storing...
float           tempFlt,    // temporary float values
                usrCash,    // money in user's wallet
                sumCash,    // sum of all money won or lost
                avgDvia,    // radicand for sd of average money won/lost
                betAmnt,    // various floats tied to user bets
                sumBets,    // sum of all user bets
                betDvia;    // radicand for sd of average user bet
bool            validIn,    // whether user input is valid
                crdUniq,    // whether generated card was already made
                hndOver,    // whether hand will end after current loop
                usrStnd,    // whether user is done drawing cards
                dlrShow,    // whether to show dealer's hidden card
                dlrSoft,    // whether dealer has a soft ace
                usrSoft,    // whether user has a soft ace
                usrWon;     // whether user won the current hand
unsigned char   crdFace,    // ascii value of card's face value
                crdsDlt,    // no. of total cards drawn in current hand
                dlrCrds,    // no. of cards currently in dealer's hand
                usrCrds,    // no. of cards currently in user's hand
                usrInpt;    // one-character decision input from user
unsigned short  randVal,    // random value generated for card id
                readVal,    // reading card ids from crdVals.dat
                dlrHand,    // total value of the dealer's cards
                usrHand,    // total value of the user's cards
                usrWins,    // total number of hands won by user
                usrStrk,    // current user win-streak
                bstStrk,    // best user win-streak from this session
                roundNo;    // total no. of rounds played this session
string          tempStr;    // any string (all strings are temporary)

// file i/o objects         // used for...
fstream         outVals,    // writing card ids to crdVals.dat
                outStrs,    // writing card strings to crdStrs.dat
                outCash,    // writing $ won/lost per hand to hndData.dat
                outBets,    // writing $ user bet per hand to betData.dat
                inVals,     // reading card ids from crdVals.dat
                inStrs,     // reading card strings from crdStrs.dat
                inCash,     // reading $ won/lost per hand from hndData.dat
                inBets;     // reading $ user bet per hand from betData.dat
```

The most complicated concept allowed was reading/writing files via fstream. This was agony! I'm in hell.

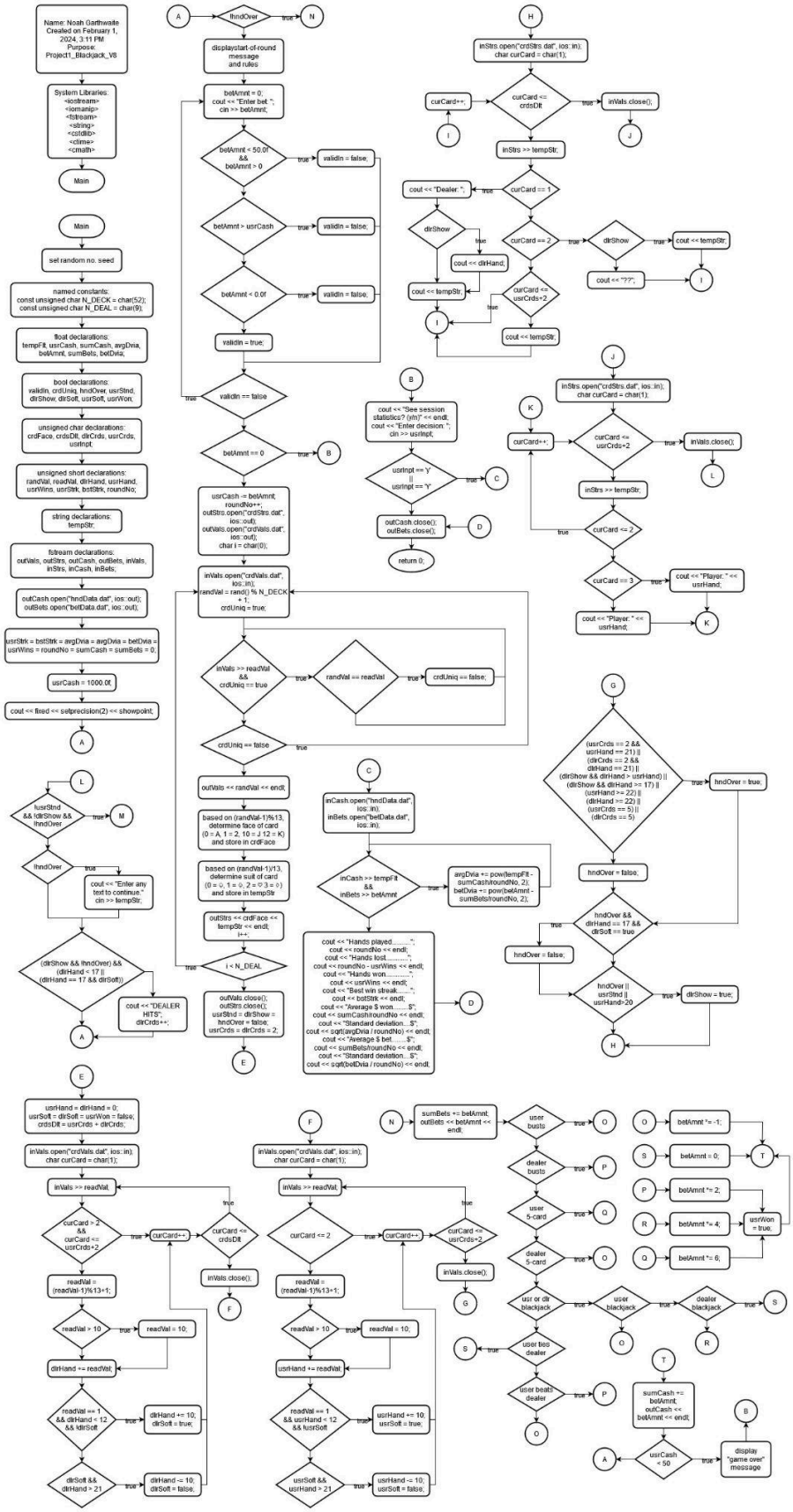
No arrays, no vectors, no programmer-defined functions, no call-by-reference... Essentially, just a handful of libraries, if-else statements, loops, basic operators, and file i/o. I'll admit, it makes for a quite stunning variable declaration block, but this is no way to live. That was a prominent purpose for this assignment, according to our professor - to teach us how difficult programming can be if you aren't willing to learn more. In a later section, I will document what I consider to be some clever workarounds with regards to these limitations, as a game of Blackjack has quite a few moving parts that I was able to condense into a program that only requires 29 variables and 4 .dat files.

Not only that, but this game is a completely accurate recreation of the house rules of Blackjack as found in the casino Blackjack minigame from Clover Studio's *God Hand* (2006) for the Sony PlayStation 2, aside from the minimum bet. In another section, I will explain what those rules are.

For now, know that my program:

- Grants the user with a wallet they may bet from
- Tracks the user's play statistics for them to view
- Accurately simulates drawing cards from one pool
- Is 100% bug-free!*

Furthermore, I was tasked with creating a flowchart detailing this program. It took 512 minutes! That's just over 8.5 hours. The flowchart is far too big to be readable within this document, because I was under the impression this assignment was due tonight, and I wouldn't have time to tidy it up into small pieces that could fit on 8.5x11 pages.


Name: Noah Garthwaite
Created on February 1, 2024, 3:11 PM
Purpose:
Project1_Blackjack_V8
System Libraries:
<iostream>
<iomanip>
<fstream>
<string>
<cstdlib>
<ctime>
<cmath>
Main
Main
set random no. seed
named constants:
const unsigned char N_DECK = char(52);
const unsigned char N_DEAL = char(9);
float declarations:
tempFlt, usrCash, sumCash, avgDvia, betAmnt, sumBets, betDvia;
bool declarations:
validIn, crdUniq, hndOver, usrStnd, dlrShow, dlrSoft, usrSoft, usrWon;
unsigned char declarations:
crdFace, crdsDlt, dlrCrds, usrCrds, usrInpt;
unsigned short declarations:
randVal, readVal, dlrHand, usrHand, usrWins, usrStrk, bstStrk, roundNo;
string declarations:
tempStr;
fstream declarations:
outVals, outStrs, outCash, outBets, inVals, inStrs, inCash, inBets;
outCash.open("hndData.dat", ios::out);
outBets.open("betData.dat", ios::out);
usrStrk = bstStrk = avgDvia = avgDvia = betDvia = usrWins = roundNo = sumCash = sumBets = 0;
usrCash = 1000.0f;
cout << fixed << setprecision(2) << showpoint;
A
hndOver
true
N
displaystart-of-round message and rules
betAmnt = 0;
cout << "Enter bet.";
cin >> betAmnt;
betAmnt < 50.0f && betAmnt > 0
true
validIn = false;
betAmnt > usrCash
true
validIn = false;
betAmnt < 0.0f
true
validIn = false;
validIn = true;
validIn == false
true
betAmnt == 0
true
B
usrCash = betAmnt;
roundNo++;
outStrs.open("crdStrs.dat", ios::out);
outVals.open("crdVals.dat", ios::out);
char i = char(0);
inVals.open("crdVals.dat", ios::in);
randVal = rand() % N_DECK + 1;
crdUniq = true;
inVals >> readVal && crdUniq == true
true
randVal == readVal
true
crdUniq = false;
crdUniq == false
true
outVals << randVal << endl;
based on (randVal-1)%13, determine face of card (0 = A, 1 = 2, 10 = J, 12 = K) and store in crdFace
based on (randVal-1)/13, determine suit of card (0 = ♢, 1 = ♤, 2 = ♡, 3 = ○) and store in tempStr
outStrs << crdFace << tempStr << endl;
i++;
i < N_DEAL
true
outVals.close();
outStrs.close();
usrStnd = dlrShow = hndOver = false;
usrCrds = dlrCrds = 2;
E
B
cout << "See session statistics? (y/n)" << endl;
cout << "Enter decision: ";
cin >> usrInpt;
usrInpt == 'Y' || usrInpt == 'Y'
true
C
outCash.close();
outBets.close();
D
return 0;
C
inCash.open("hndData.dat", ios::in);
inBets.open("betData.dat", ios::in);
inCash >> tempFlt && inBets >> betAmnt
true
avgDvia += pow(tempFlt - sumCash/roundNo, 2);
betDvia += pow(betAmnt - sumBets/roundNo, 2);
cout << "Hands played.....";
cout << roundNo << endl;
cout << "Hands lost.....";
cout << roundNo - usrWins << endl;
cout << "Hands won.....";
cout << usrWins << endl;
cout << "Best win streak.....";
cout << bstStrk << endl;
cout << "Average $ won.....$";
cout << sumCash/roundNo << endl;
cout << "Standard deviation...$";
cout << sqrt(avgDvia / roundNo) << endl;
cout << "Average $ bet.....$";
cout << sumBets/roundNo << endl;
cout << "Standard deviation...$";
cout << sqrt(betDvia / roundNo) << endl;
D
H
inStrs.open("crdStrs.dat", ios::in);
char curCard = char(1);
curCard <= crdsDlt
true
inVals.close();
curCard++;
I
J
inStrs >> tempStr;
curCard == 1
true
cout << "Dealer: ";
dlrShow
true
cout << dlrHand;
cout << tempStr;
curCard == 2
true
dlrShow
true
cout << tempStr;
cout << "??";
curCard <= usrCrds+2
true
cout << tempStr;
J
inStrs.open("crdStrs.dat", ios::in);
char curCard = char(1);
curCard <= usrCrds+2
true
inVals.close();
curCard++;
K
inStrs >> tempStr;
curCard <= 2
true
curCard == 3
true
cout << "Player." << usrHand;
cout << "Player." << usrHand;
K
L
usrStnd && !dlrShow && !hndOver
true
M
!hndOver
true
cout << "Enter any text to continue.";
cin >> tempStr;
(dlrShow && !hndOver) && (dlrHand < 17 || (dlrHand == 17 && dlrSoft))
true
cout << "DEALER HITS";
dlrCrds++;
A
G
(usrCrds == 2 && usrHand == 21) || (dlrCrds == 2 && dlrHand == 21) || (dlrShow && dlrHand > usrHand) || (dlrShow && dlrHand >= 17) || (usrHand == 22) || (dlrHand == 22) || (usrCrds == 5) || (dlrCrds == 5)
true
hndOver = true;
hndOver = false;
hndOver && dlrHand == 17 && dlrSoft == true
true
hndOver = false;
hndOver || usrStnd || usrHand>20
true
dlrShow = true;
H
E
usrHand = dlrHand = 0;
usrSoft = dlrSoft = usrWon = false;
crdsDlt = usrCrds = dlrCrds;
inVals.open("crdVals.dat", ios::in);
char curCard = char(1);
inVals >> readVal;
curCard > 2 && curCard <= usrCrds+2
true
curCard++
curCard <= crdsDlt
true
inVals.close();
F
readVal = (readVal-1)%13+1;
readVal > 10
true
readVal = 10;
dlrHand += readVal;
readVal == 1 && dlrHand < 12 && !dlrSoft
true
dlrHand += 10;
dlrSoft = true;
dlrSoft && dlrHand > 21
true
dlrHand -= 10;
dlrSoft = false;
F
inVals.open("crdVals.dat", ios::in);
char curCard = char(1);
inVals >> readVal;
curCard <= 2
true
curCard++
curCard <= usrCrds+2
true
inVals.close();
G
readVal = (readVal-1)%13+1;
readVal > 10
true
readVal = 10;
usrHand += readVal;
readVal == 1 && usrHand < 12 && !usrSoft
true
usrHand += 10;
usrSoft = true;
usrSoft && usrHand > 21
true
usrHand -= 10;
usrSoft = false;
N
sumBets += betAmnt;
outBets << betAmnt << endl;
user busts
true
O
dealer busts
true
P
user 5-card
true
O
dealer 5-card
true
O
usr or dlr blackjack
true
user blackjack
true
dealer blackjack
true
S
user ties dealer
true
S
user beats dealer
true
P
O
O
betAmnt *= -1;
S
betAmnt = 0;
T
P
betAmnt *= 2;
R
betAmnt *= 4;
usrWon = true;
Q
betAmnt *= 6;
T
sumCash += betAmnt;
outCash << betAmnt << endl;
A
usrCash < 50
true
B
display "game over" message

Sure, I could go back and rewrite some of the comically vertical pieces of this flowchart; but Project 2 is due in one week, and I would rather be hunted for sport than to spend one more hour of my life in draw.io that I don't have to. Look at that thing! I came into this class to learn C++, and I left making information graphics that look like motherboards. I've hardly learned any concepts specific to C++, doomed to make diagrams and neutered card games.

Though, truthfully, I don't think this assignment would have been nearly as engaging if these limitations weren't imposed upon the project - I wanted to think outside of the box in order to optimize the game despite these limitations. There may be some chunks of code that are faster or less memory intensive than Project 2, which will allow the usage of the aforementioned architectures barred from use in this project, but there's no way to know yet! I hope to test out the speed of the program while implementing a learning algorithm in the future that finds the best strategy, as I'm especially curious to how the house rules of God Hand (2006)'s Blackjack affect the typical house edge.

Before I show off some snippets of code that I consider to be well-designed, touch on the biggest issues faced when developing this project, and break down a gameplay demo; let's explain the basic premise of Blackjack and the house rules present in both God Hand (2006) and this specific Blackjack program, for posterity. Likewise, for posterity, the jokes stop here. I want an A. Likewise, I just looked up what "for posterity" means, and it doesn't mean what I thought it meant. I thought it meant "just to be safe", or "for clarity's sake". Anyway.

The Rules

or:

"Explaining Blackjack

to My Friends Reading This"

The goal of a round of Blackjack is to beat The Dealer. Each round of play is a new "hand", as The Player and The Dealer discard all cards from the previous "hand" and are dealt new cards to their respective "hand". In this sense, a "hand" in Blackjack is a dual-purpose word meaning both a round of play and the cards a participant has during a round. The Player's hand is a collection of cards aiming to be better than The Dealer's. However, The Dealer cannot exactly "aim" to do anything, as The Dealer must play in a strict, robotic fashion. When playing Blackjack in a casino with multiple Players and one Dealer, all players individually face The Dealer and are not competing against one another.

Cards in Blackjack are worth the following:

- 2–10: Face value
- Jack, Queen, or King: 10
- Ace: 1 or 11

All card values in a participant's hand are added together. The highest total a participant may have is 21 – anything over is a "bust", an automatic loss.

If a hand contains an Ace, it may be worth 1 or 11 based on context. If a hand would not bust with an Ace valued at 11, it is worth 11 and is considered a "soft" Ace. Otherwise, the Ace is worth 1 and is considered a "hard" Ace. An Ace can change from hard to soft at any point, depending on whichever would benefit the hand most. I'll show off how and why the program determines whether a Hard Ace or Soft Ace is best for a hand in the later Gameplay Demo.

A round of this Blackjack game consists of up to five phases:

1. The Player places a bet for the hand.
2. The Player and The Dealer are dealt two cards each. The Dealer's first card is shown face-up, and the second is kept face-down.
    a. If The Dealer and/or The Player are dealt a 10-value card and an Ace in this first phase; this is a Blackjack, and the hand is over.
3. The Player takes their turn(s). If it is their first turn (and they have enough money), they have the option to "Double Down". This doubles their current bet, gives them one additional card, and begins Phase 4.
    a. If their hand totals over 21, the hand is over.
    b. If they collected five cards without busting, the hand is over.
    c. If they don't want another card, they stand. The Dealer reveals their hand and Phase 4 begins.
    d. If they do want another card, they hit. Phase 3 restarts.
4. The Dealer takes their turn(s).
    a. If their hand totals over 21, the hand is over.
    b. If The Dealer collects five cards without busting, the hand is over.
    c. If their total is better, the hand is over.
    d. If their hand totals 17 or higher, they stand, and the hand is over.
        i. Unless, however, they have a total of 17 and a soft ace, in which case they must hit.
    e. Otherwise, they must hit. Phase 4 restarts.

5. Once the hand is over, the winner is determined, and the player decides if they wish to play another hand (if they have enough money. All cards are collected by The Dealer and deck is reshuffled.
    a. If someone achieved Blackjack, they win.
        i. Unless both did, in which there is a tie.
    b. Otherwise, if someone busted, they lose.
    c. Otherwise, if someone achieved Five Card, they win.
    d. Otherwise, if someone's total is higher, they win.
    e. Otherwise, there is a tie.
    f. If there is a tie, The Player is returned their bet.
    g. If The Player won via Blackjack, they are returned their bet plus triple the bet amount.
    h. If The Player won via Five Card, they are returned their bet plus quintuple the bet amount.
    i. If The Player won via neither Blackjack nor Five Card, they are returned their bet plus the bet amount.s
    j. If The Player loses, their bet is not returned.

These are the rules of Blackjack as present in God Hand (2006). This is a 1-deck game of Blackjack, because I have never observed two cards of the same suit and face appear in God Hand (2006). There is no splitting of pairs or insurance in God Hand (2006)'s Blackjack, so there is no splitting of pairs or insurance in this game. If you don't know what any of that means, good news: You don't need to!

Requirements

or:

"Why This Took So Long"

As stated earlier, this project was not permitted to use anything that hadn't been taught in class when the project was first assigned, which was difficult! This project was also required to use a little bit of everything we were taught up until that point. This includes using at least one feature from each required library. These libraries and how they are used are as follows:

```
// system libraries // used to...
#include <iostream> // get user input and display outpurs
#include <iomanip>  // format dollar amount outputs
#include <fstream>  // handle session stats & store cards for current hand
#include <string>   // handle "continue" input & suit ascii
#include <cstdlib>  // randomize cards drawn
#include <ctime>    // set the random number seed
#include <cmath>    // calculate standard deviation stats
```

All of these libraries were present before I was aware that they were required, except for <cmath>. To implement that library, I added several variables and two more .dat files that track the user's play statistics for optional output at the end of a session – that way, I could use the pow() and sqrt() function of <cmath> to calculate standard deviations. Additionally, <iostream>'s cout & cin are used; <iomanip>'s fixed, setprecision(), and showpoint are used; <fstream>'s... fstreams are used; <strings>'s strings are used; <cstdlib>'s rand() and srand() are used; and <ctime>'s time() is used. All other requirements were documented in the checkoff sheet, which can be found in the project's .zip file. So, with the requirements in mind, what should the code look like?

Pseudocode

or:

"Quasicodo"

In plain English, the program should behave as so:

```
INCLUDE LIBRARIES
SET RANDOM SEED
DECLARE VARIABLES & CONSTANTS
OPEN/CREATE THE SESSION-STATS DATA FILE WRITERS
INITIALIZE VARIABLES
SET DECIMAL PRECISION
BEGIN GAMEPLAY LOOP
    DISPLAY NEW-HAND MESSAGE & RULES
    LET USER BET OR END THE GAME
    IF ENDING THE GAME:
      ASK TO DISPLAY STATS
      IF STATS REQUESTED:
            OPEN SESSION DATA FILE READERS
            CALCULATE STATS & DISPLAY THEM
      CLOSE SESSION DATA FILE READERS
      CLOSE THE PROGRAM
    REMOVE BET FROM THEIR WALLET
    BEGIN NEW HAND
      OPEN/CREATE THE CARD DATA FILE WRITERS
      GENERATE RANDOM CARD ID
            IF CARD ID ALREADY IN CARD ID FILE:
                  GENERATE A NEW ONE & CHECK AGAIN
            WRITE THE CARD ID TO THE CARD ID FILE
            CALCULATE THE CARD SUIT & FACE FROM THE ID
            WRITE THE CARD LOOK TO THE STRING DATA FILE
            REPEAT UNTIL 9 UNIQUE CARDS ARE GENERATED
      CLOSE THE CARD DATA FILE WRITERS
```

```
ASSIGN TWO CARDS EACH TO DEALER AND PLAYER
IF EITHER GOT BLACKJACK:
      END THE HAND
SHOW DEALER'S FIRST CARD, HIDE DEALER TOTAL
SHOW PLAYER'S CARDS AND TOTAL
BEGIN PLAYER'S TURN
      IF PLAYER BUSTED OR GOT FIVE CARD:
            END THE HAND
      IF PLAYER CAN'T HIT:
            END THEIR TURN
      DISPLAY DECISIONS AND PROMPT PLAYER INPUT
      IF PLAYER CAN'T DOUBLE DOWN:
            DON'T DISPLAY DOUBLE DOWN OPTION
      IF USER HITS:
            ASSIGN A NEW CARD AND REPEAT TURN
      IF USER STANDS:
            END THEIR TURN
      IF USER DOUBLES DOWN:
            IF THEY CAN DOUBLE DOWN:
                  TAKE MONEY NEEDED FROM WALLET
                  ASSIGN A NEW CARD
                  IF THEY BUSTED:
                        END THE HAND
                  IF THEY DIDN'T BUST:
                        END THEIR TURN
BEGIN DEALER'S TURN
      IF DEALER BUSTED:
            END THE HAND
```

```
        IF DEALER GOT FIVE CARD OR BEAT PLAYER:
            END THE HAND
        IF DEALER HAS MORE THAN 17 OR HARD 17:
            END THE HAND
        IF DEALER HAS LESS THAN 17 OR SOFT 17:
            ASSIGN A NEW CARD AND REPEAT TURN
    DETERMINE WHO WON
    IF PLAYER DIDN'T LOSE:
      GIVE PLAYER BET BACK
      IF PLAYER WON:
            CALCULATE WINNINGS AND AWARD TO PLAYER
    LOG HAND STATS TO SESSION DATA FILES
    IF USER LACKS MONEY TO PLAY NEW HAND:
      DISPLAY GAME OVER
      ASK TO DISPLAY STATS
      IF STATS REQUESTED:
            OPEN SESSION DATA FILE READERS
            CALCULATE STATS & DISPLAY THEM
      CLOSE SESSION DATA FILE READERS
      CLOSE THE PROGRAM
    REPEAT GAMEPLAY LOOP
```

If it looks silly to check at the end of a hand if the player has enough money to start a new hand rather than just checking at the beginning... That's because it is. I didn't notice that until writing this pseudocode. ...That's about sixty redundant lines of code. I don't have enough time to fix that, so... Moving on.

Highlights

or:

"The Bits That Suck the Least"

I'd like to show off some of the code from this project that I'm proud of:

```
cout << "O-----------BLACKJACK----------O" << endl;
cout << "|    DEALER MUST HIT SOFT 17   |" << endl;
cout << "|     BLACKJACK PAYS 3 TO 1    |" << endl;
cout << "|     FIVE CARD PAYS 5 TO 1    |" << endl;
cout << "O-----------*.°.+.°.*----------O" << endl;
```

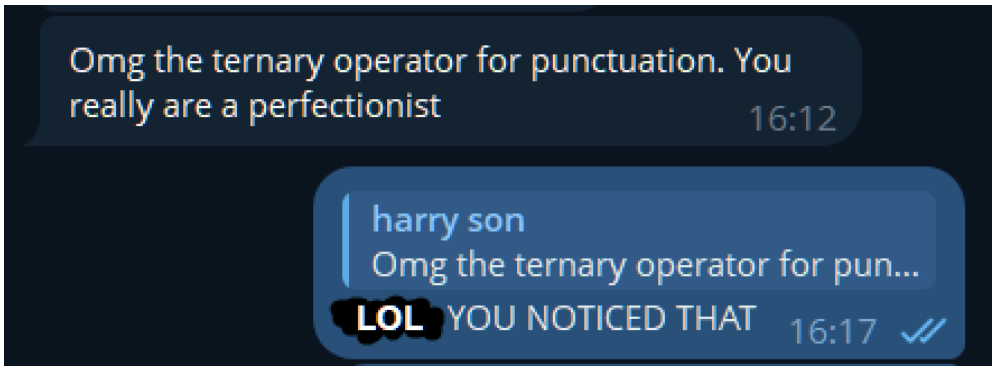
I think this part looks neat. I like making ASCII art!

```
// validate user input
if (betAmnt < 50.0f && betAmnt > 0) {
    validIn = false;
    cout << "Bet must be at least 50." << endl;
}
else if (betAmnt > usrCash) {
    validIn = false;
    cout << "You can't bet more than you have, wise guy." << endl;
}
else if (betAmnt < 0.0f) {
    validIn = false;
    cout << "You can't bet a negative amount. Nice try." << endl;
}
else validIn = true;
```

I think it's funny for a program to have passive-aggressive input validation. Some of the first programs I ever wrote would do this back when I started learning how to code.

```
cout << "You walked away with $" << usrCash;
// no need to celebrate if the user didn't win anything
(usrCash > 1000.0f) ? (cout << "!") : (cout << ".");
cout << endl;
```

Incredibly small detail I didn't think anyone would notice, but evidently:




```
// parse how to display a card's suit based on card id
switch ((randVal-1) / 13)
{
    case 0:     tempStr = "♤";
                break;
    case 1:     tempStr = "♧";
                break;
    case 2:     tempStr = "♡";
                break;
    default:    tempStr = "♢";
}
```

I opted to use ASCII card suits, and I think they look nice during gameplay.

```
/*
 * in both crdVals.dat and crdStrs.dat, the dealer's cards
 * are stored at the beginning and end of the .dat files.
 * this is because a hand follows this order of turns:
 *
 * 1. dealer + user both get two cards
 * 2. user has to stand/hit first
 * 3. dealer has to stand/hit last
 *
 * because it is unknown how many cards both the user
 * & dealer will draw each hand, and there is no way of knowing
 * whether it could be the user or dealer who wins by a
 * five-card charlie, it's easiest to simply dedicate the first
 * two lines of the .dat cards to the dealer's starting two cards
 * and then dedicate however many cards the user ends up with
 * after the dealer's first two cards.
 *
 * this also accurately simulates drawing a card from the deck
 * whenever the user or dealer hits, because it means either
 * of them could have gotten that card.
 *
 * this makes reading the .dat files slower, but i think
 * it's more fair to faithfully simulate drawing from the deck.
 */
```

An explanation as to the claim earlier that the program "accurately simulates drawing cards from one pool". I've always wondered if something like this is used when playing any form of virtual Blackjack, or if there is a discrepancy between what The Dealer can draw vs. what The Player can draw. For reasons I'm about to explain, I don't think this mechanic technically slows down the program, now that I'm thinking about it.

```
// calculate totals separately to maintain my sanity
// calculate dealer total first
inVals.open("crdVals.dat", ios::in);
for (char curCard = char(1); curCard <= crdsDlt; curCard++) {
    // read next card in the file
    inVals >> readVal;

    // skip card if it belongs to the user
    if (curCard > 2 && curCard <= usrCrds+2) continue;

    readVal = (readVal-1)%13+1;     // parse the card face
    if (readVal > 10) readVal = 10; // set to 10 if J, Q, or K
    dlrHand += readVal;

    // if card is an ace, hand is hard, and soft ace won't bust
    if (readVal == 1 && dlrHand < 12 && !dlrSoft) {
        dlrHand += 10;  // 11 (soft ace) - 1 (hard ace) = 10
        dlrSoft = true; // flag to track soft ace for later loops
    }
    // check if earlier soft ace should be hard to prevent bust
    else if (dlrSoft && dlrHand > 21) {
        dlrHand -= 10;
        dlrSoft = false;
    }
}
inVals.close();
```

I realized I could essentially cheat and use file i/o as a form of index-less array of information due to the way fstream objects work as arguments. This means there are no hard-coded variables needed to store card data! This does mean that if we need to access information at the end of the file, we need to read through the whole file (though, nine lines is trivial). However, this means the program would be no faster if The Player's cards weren't sandwiched in the middle of the file - if we just kept their cards at the end, fstream would need to read the beginning anyway.

Demonstration

or:

"Blackjack Gameplay Walkthrough

FULL GAME (4K 60FPS) No Commentary"

```
O-----------BLACKJACK-----------O
|     DEALER MUST HIT SOFT 17    |
|      BLACKJACK PAYS 3 TO 1     |
|      FIVE CARD PAYS 5 TO 1     |
O-----------*.°.+.°.*-----------O

You have: $1000.00
How much would you like to bet?

Minimum bet is 50.
Input a bet of 0 to walk away
with your current cash.

Enter bet: 50

Dealer:
A♤  ??

Player: 13
3♡  T♤

Pot total: $50.00
You have: $950.00
```

The Player opens the game and is given $1000.00 to start. They bet $50 to start a hand, and are given two cards: a 3 of Hearts and a Ten of Clubs. They can see The Dealer has an Ace of Spades.

```
What would you like to do?
Type S to Stand
Type H to Hit
Type D to Double Down
Enter decision: h

Dealer:
A♤  ??

Player: 17
3♡  T♤  4♡

Pot total: $50.00
You have: $950.00

What would you like to do?
Type S to Stand
Type H to Hit
Enter decision: h

Dealer: 15
A♤  4♤

Player: 23
3♡  T♤  4♡  6♤

YOU BUST
DEALER WINS
```

The Player hits and receives a Four of Hearts, bringing their total to 17. Thinking The Dealer might have a seven or better, they hit and receive a six of spades, causing them to bust. The program shows who won and how they won.

```
O-----------BLACKJACK-----------O
|     DEALER MUST HIT SOFT 17    |
|      BLACKJACK PAYS 3 TO 1     |
|      FIVE CARD PAYS 5 TO 1     |
O-----------*.°.+.°.*-----------O

You have: $950.00
How much would you like to bet?

Minimum bet is 50.
Input a bet of 0 to walk away
with your current cash.

Enter bet: 50.50

Dealer:
7♤  ??

Player: 15
5♢  K♢

Pot total: $50.50
You have: $899.50
```

The Player is greeted with the opening message again. They decide to wager $50.50 this time, just to see what would happen or something. The program accepts the amount and puts it in the pot before dealing a new hand.

```
What would you like to do?
Type S to Stand
Type H to Hit
Type D to Double Down
Enter decision: h

Dealer: 16
7♤  9♤

Player: 22
5♢  K♢  7♡

YOU BUST
DEALER WINS
```

The Player hits, and they bust again.

```
O-----------BLACKJACK----------O
|    DEALER MUST HIT SOFT 17   |
|     BLACKJACK PAYS 3 TO 1    |
|     FIVE CARD PAYS 5 TO 1    |
O----------*.°.+.°.*----------O

You have: $899.50
How much would you like to bet?

Minimum bet is 50.
Input a bet of 0 to walk away
with your current cash.

Enter bet: 99.50
```

The opening message greets them again. They decide to bet $99.50 this time, probably because the dollar amount was bugging them.

```
Dealer:
9♦  ??

Player: 10
7♠  3♦

Pot total: $99.50
You have: $800.00

What would you like to do?
Type S to Stand
Type H to Hit
Type D to Double Down
Enter decision: d

Dealer: 12
9♦  3♠

Player: 14
7♠  3♦  4♠

Enter any text to continue.
```

The Player is dealt a Seven and a Three.

The Player decides to double down. They draw a Four, because of course of course they would. The program pauses and waits for The Player to prompt it to continue with the next turn, like it's rubbing it in.

```
DEALER HITS

Dealer: 18
9♢  3♤  6♤

Player: 14
7♤  3♢  4♤

DEALER STANDS
DEALER WINS
```

The player then expectedly loses after The Dealer hits and receives a Six, bringing their total to 18, which they stand on.

```
O-----------BLACKJACK----------O
|    DEALER MUST HIT SOFT 17   |
|     BLACKJACK PAYS 3 TO 1    |
|     FIVE CARD PAYS 5 TO 1    |
O-----------*.°.+.°.*----------O

You have: $700.50
How much would you like to bet?

Minimum bet is 50.
Input a bet of 0 to walk away
with your current cash.

Enter bet: 200.50
```

The Player forgot that doubling down would mean betting an extra fifty cents last turn, so they enter $200.50 to try and make things right for once. If they were confident, they would have wagered $299.50, but they didn't.

```
What would you like to do?
Type S to Stand
Type H to Hit
Enter decision: h

Dealer:
T♤  ??

Player: 20
2♤  Q♢  3♡  5♡

Pot total: $200.50
You have: $500.00

What would you like to do?
Type S to Stand
Type H to Hit
Enter decision: s

Dealer: 20
T♤  J♡

Player: 20
2♤  Q♢  3♡  5♡
```

The Player hits twice and stands on a 20, feeling confident.

The Dealer reveals a 20.

```
O-----------BLACKJACK----------O
|    DEALER MUST HIT SOFT 17   |
|     BLACKJACK PAYS 3 TO 1    |
|     FIVE CARD PAYS 5 TO 1    |
O-----------*.°.+.°.*----------O

You have: $700.50
How much would you like to bet?

Minimum bet is 50.
Input a bet of 0 to walk away
with your current cash.

Enter bet: 700.50

Dealer:
3♤  ??

Player: 14
9♡  5♧

Pot total: $700.50
```

The Player's bet is returned to them. Thinking this $0.50 has cursed this session, they wager everything in a fugue state.

```
Dealer:
3♤  ??

Player: 14
9♡  5♤

Pot total: $700.50
You have: $0.00

What would you like to do?
Type S to Stand
Type H to Hit
Enter decision: h

Dealer: 14
3♤  A♢

Player: 21
9♡  5♤  7♢

Enter any text to continue.
```

The Player is dealt a 14, and they decide to hit.

The Player, on their knees, is spared by the program.

```
O-----------BLACKJACK----------O
|    DEALER MUST HIT SOFT 17   |
|     BLACKJACK PAYS 3 TO 1    |
|     FIVE CARD PAYS 5 TO 1    |
O-----------*.°.+.°.*----------O

You have: $1401.00
How much would you like to bet?

Minimum bet is 50.
Input a bet of 0 to walk away
with your current cash.

Enter bet: 0

You walked away with $1401.00!
See session statistics? (y/n)
Enter decision: y

Hands played.....5
Hands lost.....4
Hands won.....1
Best win streak.....1
Average $ won.....$220.30
Standard deviation....$594.10
Average $ bet.....$240.10
Standard deviation....$239.71
```

The Player goes on to win the hand and end up with more money than they started with. The Player immediately hits the bricks. The program asks if The Player wants to see their session statistics before they leave, to which they say yes. The program reminds The Player of the mercy they were shown.

Conclusion

or:

"I Said the Jokes Would Stop and They Didn’t so I Should Probably Say Something Sincere Because if I Get Penalized for Trying To Be Funny I’m Going To Hate Myself"

I had a lot of fun iterating on the code for this. I ended up learning some valuable insight into how reading and writing to/from files are handled through fstream. Calling for the reader in the argument of a loop was interesting, and I was able to intuit that if I could use a logical && operator to read from two files at the same time in a loop condition, which was neat.

The hardest part, hands down, was trying to program logic for handling hard/soft Aces. I attempted many times to brainstorm my own ideas for it, but I ended up getting tangled in my code and struggling to make it work in all conditions without bloating the code. I tried flags called "uHasAce" & "dHasAce", but it wasn't effective in figuring out when a hand should stop being soft and start being hard, and vice versa. I also tried an "usrAces" & "dlrAces" count, but having multiple Aces ended up still causing bugs. After deciding to research a good way to program this, a stackoverflow thread came up in which the accepted answer from an Unknown user stated:

"Just treat each ace as 11. Then while the value is over 21, subtract 10 from your total for each ace in your hand."

In a comment to that answer, Lee Daniel Crocker replied:

Amazing how the wrong answer gets voted up to the top. As subsequent correct answers have noted, only one ace is ever counted as 11, so it is much more efficient to simply count every ace as 1, then, if you found an ace, and the total < 12, add 10. No need to keep track of how many aces, and only one extra add needed. If you did add 10, you can set a "soft" flag if you need to track whether a total is hard or soft.

This ended up being the exact insight I needed. Using booleans named "usrSoft" & "dlrSoft" was much easier, and I believe my program to now be bug-free (aside from certain input validation methods to account for edge cases).

These issues with aces would have arisen even without the limitations on the scope of this project's architecture, so I'm happy to know exactly how to implement that in Project 2. The limitations made this into a kind of logic puzzle, which is one of the biggest appeals programming has, to me. It may comes as a surprise, but I enjoy taking creative approaches when I can, so this was a welcome breath of fresh air from all of the rather clinical nature of most computer science homework. Even the flowchart, be it my enemy, was occasionally relaxing once I started getting more creative with the structure of my flowcharts toward the end. I attempted to leave as little space empty as possible without making it too cluttered to navigate. The full flowchart can be seen up close here:

https://drive.google.com/file/d/1SI9KK-pu0oLjJvOUO6V1T16Do-gBv90i/view?usp=sharing

I'm looking forward to fixing that obvious oversight with the Game Over code block, but I imagine Project 2 may feel like busywork compared to this. Dr. Lehr, if you actually read this entire thing: I will do anything to not make another flowchart. Name your price. Please.

The Code

or:

"The Code"

```
/*
 * File:    main.cpp
 * Author:  Noah Garthwaite
 * Created on February 1, 2024, 3:11 PM
 * Purpose: Project1_Blackjack_V8
 */

// system libraries // used to...
#include <iostream> // get user input and display outputs
#include <iomanip>  // format dollar amount outputs
#include <fstream>  // handle session stats & store cards for current hand
#include <string>   // handle "continue" input & suit ascii
#include <cstdlib>  // randomize cards drawn
#include <ctime>    // set the random number seed
#include <cmath>    // calculate standard deviation stats

using namespace std;

// execution begins
int main(void) {

    srand(static_cast<unsigned int>(time(0)));  // set the random no. seed

    // named constants
    const unsigned char N_DECK = char(52);  // number of possible cards to draw
    const unsigned char N_DEAL = char(9);   // number of cards needed per hand

    // variable declarations    // used for storing...
    float           tempFlt,    // temporary float values
                    usrCash,    // money in user's wallet
                    sumCash,    // sum of all money won or lost
                    avgDvia,    // radicand for sd of average money won/lost
```

```
                betAmnt,    // various floats tied to user bets
                sumBets,    // sum of all user bets
                betDvia;    // radicand for sd of average user bet
bool            validIn,    // whether user input is valid
                crdUniq,    // whether generated card was already made
                hndOver,    // whether hand will end after current loop
                usrStnd,    // whether user is done drawing cards
                dlrShow,    // whether to show dealer's hidden card
                dlrSoft,    // whether dealer has a soft ace
                usrSoft,    // whether user has a soft ace
                usrWon;     // whether user won the current hand
unsigned char   crdFace,    // ascii value of card's face value
                crdsDlt,    // no. of total cards drawn in current hand
                dlrCrds,    // no. of cards currently in dealer's hand
                usrCrds,    // no. of cards currently in user's hand
                usrInpt;    // one-character decision input from user
unsigned short  randVal,    // random value generated for card id
                readVal,    // reading card ids from crdVals.dat
                dlrHand,    // total value of the dealer's cards
                usrHand,    // total value of the user's cards
                usrWins,    // total number of hands won by user
                usrStrk,    // current user win-streak
                bstStrk,    // best user win-streak from this session
                roundNo;    // total no. of rounds played this session
string          tempStr;    // any string (all strings are temporary)

// file i/o objects         // used for...
fstream         outVals,    // writing card ids to crdVals.dat
                outStrs,    // writing card strings to crdStrs.dat
                outCash,    // writing $ won/lost per hand to hndData.dat
                outBets,    // writing $ user bet per hand to betData.dat
                inVals,     // reading card ids from crdVals.dat
```

```
                inStrs,     // reading card strings from crdStrs.dat
                inCash,     // reading $ won/lost per hand from hndData.dat
                inBets;     // reading $ user bet per hand from betData.dat

// initialize session log fstream objects & starting variable values
outCash.open("hndData.dat", ios::out);  // outCash and outBets stay
outBets.open("betData.dat", ios::out);  // open until program closes.
usrStrk = bstStrk = avgDvia = avgDvia = betDvia = 0;
usrWins = roundNo = sumCash = sumBets = 0;  // set accumulators to 0.
usrCash = 1000.0f;  // user starts with $1000.00

cout << fixed << setprecision(2) << showpoint;  // all floats printed
                                                // are dollar amounts, so
// gameplay loop starts here                    // show two decimal places.
while (true) {

    cout << "O-----------BLACKJACK----------O" << endl;
    cout << "|    DEALER MUST HIT SOFT 17   |" << endl;
    cout << "|     BLACKJACK PAYS 3 TO 1    |" << endl;
    cout << "|     FIVE CARD PAYS 5 TO 1    |" << endl;
    cout << "O-----------*⁺°.+.°⁺*----------O" << endl;

    cout << endl << "You have: $" << usrCash << endl;
    cout << "How much would you like to bet?" << endl << endl;
    cout << "Minimum bet is 50." << endl;
    cout << "Input a bet of 0 to walk away" << endl;
    cout << "with your current cash." << endl << endl;

    // let user decide to bet or walk away
    do {
        betAmnt = 0;    // resets for every hand
        cout << "Enter bet: ";
```

```
        cin >> betAmnt;
        cout << endl;

        // validate user input
        if (betAmnt < 50.0f && betAmnt > 0) {
            validIn = false;
            cout << "Bet must be at least 50." << endl;
        }
        else if (betAmnt > usrCash) {
            validIn = false;
            cout << "You can't bet more than you have, wise guy." << endl;
        }
        else if (betAmnt < 0.0f) {
            validIn = false;
            cout << "You can't bet a negative amount. Nice try." << endl;
        }
        else validIn = true;

    } while (validIn == false);

    // display ending messages if user decides to walk away
    if (betAmnt == 0) {
        cout << "You walked away with $" << usrCash;
        // no need to celebrate if the user didn't win anything
        (usrCash > 1000.0f) ? (cout << "!") : (cout << ".");
        cout << endl;
        cout << "See session statistics? (y/n)" << endl;
        cout << "Enter decision: ";
        cin >> usrInpt;
        cout << endl;
        if (usrInpt == 'y' || usrInpt == 'Y') {
```

```
// start reading final session data now that game is over
inCash.open("hndData.dat", ios::in);
inBets.open("betData.dat", ios::in);

// standard deviation is evaluated as the square root of (e/n)
// where n = the number of values in the data set and
// where e = the sum of ((each value - mean of data set)^2)
while (inCash >> tempFlt && inBets >> betAmnt) {
    avgDvia += pow(abs(tempFlt - sumCash/roundNo), 2);
    betDvia += pow(abs(betAmnt - sumBets/roundNo), 2);
}

// stop reading final session data
inCash.close();
inBets.close();

cout << "Hands played..........";
cout << roundNo << endl;
cout << "Hands lost............";
cout << roundNo - usrWins << endl;
cout << "Hands won.............";
cout << usrWins << endl;
cout << "Best win streak........";
cout << bstStrk << endl;
cout << "Average $ won.........$";
cout << sumCash/roundNo << endl;
cout << "Standard deviation....$";
cout << sqrt(avgDvia / roundNo) << endl;
cout << "Average $ bet.........$";
cout << sumBets/roundNo << endl;
cout << "Standard deviation....$";
cout << sqrt(betDvia / roundNo) << endl;
```

```
    }
    //close remaining .dat files now that game is over
    outCash.close();
    outBets.close();

    // exit stage right
    return 0;
}
// start a new hand
else usrCash -= betAmnt;    // put user bet into the pot
roundNo++;                  // increment round counter

// start writing card data to their respective files
outVals.open("crdVals.dat", ios::out);
outStrs.open("crdStrs.dat", ios::out);

/*
 * in (this version of) blackjack, the player automatically
 * wins the hand if they are dealt 5 cards without busting.
 * this is called a "five-card charlie" and is controversial.
 * http://tinyurl.com/y6pv4mjz
 *
 * this program simulates the rules of blackjack according
 * to the blackjack minigame found in the 2006 playstation 2
 * game "GOD HAND", a game i have played a lot.
 * http://tinyurl.com/ds4nyvvk
 *
 * these are not casino rules by any means - but i find them
 * to be more fun because the player is slightly more at an
 * advantage than in a true game of casino blackjack.
 *
 * these rules mean that the maximum number of cards that
```

```
 * can be used in one hand is 9, which would occur if the
 * player stands on 4 cards before the dealer wins with
 * a five-card charlie.
 *
 * this is why N_DEAL == 9, and the loop below generates
 * 9 unique cards to use during the current hand.
 *
 * the player could cheat by reading crdStrs.dat as they
 * play, but if they're playing this game in an IDE like
 * my professor will, they could already cheat anyway.
*/

// generate nine random cards to store their id & string
for (char i = char(0); i < N_DEAL; i++) {
    do {
        inVals.open("crdVals.dat", ios::in);
        // generate random card id between 1 & 52
        // instead of 0-51 for my own sanity when calculating
        // hand totals later (A=1, 2=2, 3=3, 10=10, etc).
        randVal = rand() % N_DECK + 1;
        crdUniq = true;
        // make sure card id has not been generated yet
        while (inVals >> readVal && crdUniq == true) {
            if (randVal == readVal) crdUniq = false;
        }
        inVals.close();
    } while (crdUniq == false);
    // write current unique card id to crdVals.dat
    outVals << randVal << endl;

    // parse how to display a card's face value based on card id
    switch ((randVal-1) % 13)
```

```
{
    case 0:     crdFace = 'A';
                break;
    case 1:     crdFace = '2';
                break;
    case 2:     crdFace = '3';
                break;
    case 3:     crdFace = '4';
                break;
    case 4:     crdFace = '5';
                break;
    case 5:     crdFace = '6';
                break;
    case 6:     crdFace = '7';
                break;
    case 7:     crdFace = '8';
                break;
    case 8:     crdFace = '9';
                break;
    case 9:     crdFace = 'T';
                break;
    case 10:    crdFace = 'J';
                break;
    case 11:    crdFace = 'Q';
                break;
    default:    crdFace = 'K';
}

// parse how to display a card's suit based on card id
switch ((randVal-1) / 13)
{
    case 0:     tempStr = "♤";
```

```
                    break;
        case 1:     tempStr = "♧";
                    break;
        case 2:     tempStr = "♡";
                    break;
        default:    tempStr = "♢";
    }
    // write current unique card's face & suit to crdStrs.dat
    outStrs << crdFace << tempStr << endl;
}
// stop writing card data to their respective files
outVals.close();
outStrs.close();

// initialize starting values for current hand
usrStnd = dlrShow = hndOver = false;
usrCrds = dlrCrds = 2;    // start user & dealer with two cards

do {
    // initialize values for current turn
    usrHand = dlrHand = 0;
    usrSoft = dlrSoft = usrWon = false;
    crdsDlt = usrCrds + dlrCrds;

    /*
     * in both crdVals.dat and crdStrs.dat, the dealer's cards
     * are stored at the beginning and end of the .dat files.
     * this is because a hand follows this order of turns:
     *
     * 1. dealer + user both get two cards
     * 2. user has to stand/hit first
     * 3. dealer has to stand/hit last
```

```
 *
 * because it is unknown how many cards both the user
 * & dealer will draw each hand, and there is no way of knowing
 * whether it could be the user or dealer who wins by a
 * five-card charlie, it's easiest to simply dedicate the first
 * two lines of the .dat cards to the dealer's starting two cards
 * and then dedicate however many cards the user ends up with
 * after the dealer's first two cards.
 *
 * this also accurately simulates drawing a card from the deck
 * whenever the user or dealer hits, because it means either
 * of them could have gotten that card.
 *
 * this makes reading the .dat files slower, but i think
 * it's more fair to faithfully simulate drawing from the deck.
 */

// calculate hand totals before cards are displayed
// calculate totals separately to maintain my sanity
// calculate dealer total first
inVals.open("crdVals.dat", ios::in);
for (char curCard = char(1); curCard <= crdsDlt; curCard++) {
    // read next card in the file
    inVals >> readVal;

    // skip card if it belongs to the user
    if (curCard > 2 && curCard <= usrCrds+2) continue;

    readVal = (readVal-1)%13+1;     // parse the card face
    if (readVal > 10) readVal = 10; // set to 10 if J, Q, or K
    dlrHand += readVal;
```

```
    // if card is an ace, hand is hard, and soft ace won't bust
    if (readVal == 1 && dlrHand < 12 && !dlrSoft) {
        dlrHand += 10;  // 11 (soft ace) - 1 (hard ace) = 10
        dlrSoft = true; // flag to track soft ace for later loops
    }
    // check if earlier soft ace should be hard to prevent bust
    else if (dlrSoft && dlrHand > 21) {
        dlrHand -= 10;
        dlrSoft = false;
    }
}
inVals.close();

// calculate player total last
inVals.open("crdVals.dat", ios::in);
for (char curCard = char(1); curCard <= usrCrds+2; curCard++) {
    inVals >> readVal;

    // skip card if it's one of the dealer's starting cards
    if (curCard <= 2) continue;

    readVal = (readVal-1)%13+1;
    if (readVal > 10) readVal = 10;
    usrHand += readVal;

    if (readVal == 1 && usrHand < 12 && !usrSoft) {
        usrHand += 10;
        usrSoft = true;
    }
    else if (usrSoft && usrHand > 21) {
        usrHand -= 10;
        usrSoft = false;
```

```
    }
}
inVals.close();

// current hand should be over if...
// user has blackjack
(usrCrds == 2 && usrHand == 21) ? hndOver = true :
// dealer has blackjack
(dlrCrds == 2 && dlrHand == 21) ? hndOver = true :
// dealer beat the user
(dlrShow && dlrHand > usrHand)  ? hndOver = true :
// dealer got at least 17
(dlrShow && dlrHand >= 17)      ? hndOver = true :
// user busted
(usrHand >= 22)                 ? hndOver = true :
// dealer busted
(dlrHand >= 22)                 ? hndOver = true :
// user got five-card charlie
(usrCrds == 5)                  ? hndOver = true :
// dealer got five-card charlie
(dlrCrds == 5)                  ? hndOver = true : hndOver = false;

// however, dealer must hit again if they have a soft 17
if (hndOver && dlrHand == 17 && dlrSoft == true) hndOver = false;

// show dealer's hidden card if user can't draw more cards
// check this after checking for hand-end conditions as to
// allow the user a chance to hit continue again, or else
// the messages could display a little too fast
if (hndOver || usrStnd || usrHand>20) dlrShow = true;

// allow user to keep going for five cards if they can
```

```
if (usrCrds < 5 && usrHand == 21 && usrSoft) dlrShow = false;

// display dealer cards & total first
inStrs.open("crdStrs.dat", ios::in);
for (char curCard = char(1); curCard <= crdsDlt; curCard++) {
    // read next card in the file
    inStrs >> tempStr;

    // display dealer's total only if hidden card is revealed
    if (curCard == 1) {
        cout << "Dealer: ";
        if (dlrShow) (cout << dlrHand);
        cout << endl << tempStr << " ";
    }
    // hide dealer's hidden card if user can still hit
    else if (curCard == 2) {
        cout << (dlrShow ? tempStr : "??") << " ";
    }
    // skip if card belongs to the user
    else if (curCard <= usrCrds+2) continue;
    // show any additional cards
    else {
        cout << tempStr << " ";
    }
}
inStrs.close();
cout << endl << endl;

// display player cards & total last
inStrs.open("crdStrs.dat", ios::in);
for (char curCard = char(1); curCard <= usrCrds+2; curCard++) {
    inStrs >> tempStr;
```

```
    // skip if one of the dealer's starting cards
    if (curCard <= 2) continue;
    // display user's total and first card
    else if (curCard == 3) {
        cout << "Player: " << usrHand << endl << tempStr << " ";
    }
    // display any additional cards
    else {
        cout << tempStr << " ";
    }
}
inStrs.close();
cout << endl << endl;

// if user can still hit, let them choose if they want to
if (!usrStnd && !dlrShow && !hndOver) {
    cout << "Pot total: $" << betAmnt << endl;
    cout << "You have: $" << usrCash << endl << endl;
    cout << "What would you like to do?" << endl;
    cout << "Type S to Stand" << endl;
    cout << "Type H to Hit" << endl;
    // don't display double-down option if user can't do it
    if (usrCash >= betAmnt && usrCrds == 2) {
        cout << "Type D to Double Down" << endl;
    }

    // validate input
    do {
        cout << "Enter decision: ";
        cin >> usrInpt;
        validIn = true;
```

```
switch (usrInpt)
{
    case 's':   // indicate user is done drawing cards
    case 'S':   usrStnd = true;
                break;
    case 'h':   // give user another card
    case 'H':   usrCrds++;
                break;
    case 'd':   // if user can't double down but tries...
    case 'D':   if (usrCash < betAmnt || usrCrds != 2) {
                    validIn = false;
                }
                else {
                    // user must stand after double down
                    usrStnd = true;
                    usrCrds++;
                    //add double down to the pot
                    usrCash -= betAmnt;
                    betAmnt += betAmnt;
                }
                break;
    default:    validIn = false;
}

// display why input is invalid
if (validIn == false) {
    if ((usrInpt=='d' || usrInpt=='D') && usrCrds != 2) {
        cout << "You can only double down" << endl;
        cout << "if you have two cards.";
    }
    else if (usrInpt=='d' || usrInpt=='D') {
```

```
                    cout << "You need to have at least" << endl;
                    cout << "$" << betAmnt << " to double down.";
                }
                else {
                    cout << "Invalid input.";
                }
                cout << endl << endl;
            }

        } while (validIn == false);
        cout << endl;
    }
    // allow user to view one dealer turn at a time
    else if (!hndOver) {
        cout << "Enter any text to continue." << endl;
        cin >> tempStr;
        cout << endl;
    }

    // if it's the dealer's turn...
    if (dlrShow && !hndOver) {
        // decide if dealer needs to hit
        if (dlrHand < 17 || (dlrHand == 17 && dlrSoft)) {
            cout << "DEALER HITS" << endl << endl;
            dlrCrds++;
        }
    }
} while (!hndOver);

// hand is over, so log how much the user bet this hand
// we do this after the hand instead of before because the user
// can choose to double down during the hand, which is another bet
```

```
sumBets += betAmnt;
outBets << betAmnt << endl;

// determine who won and how
// now that we've logged how much the user bet, we can re-purpose
// betAmnt to log how much money the user won/lost this hand
if (usrHand > 21) {
    cout << "YOU BUST" << endl;
    cout << "DEALER WINS" << endl;
    betAmnt *= -1;  // make betAmnt negative to log a loss
}
else if (dlrHand > 21) {
    cout << "DEALER BUSTS" << endl;
    cout << "YOU WIN!" << endl;
    betAmnt *= 2;   // standard win pays 1 to 1
    usrWon = true;
}
else if (usrCrds == 5 || dlrCrds == 5) {
    cout << "FIVE CARDS" << endl;
    if (usrCrds == 5) {
        cout << "YOU WIN!" << endl;
        betAmnt *= 6;   // five cards pays 5 to 1
        usrWon = true;
    }
    else {
        cout << "DEALER WINS" << endl;
        betAmnt *= -1;
    }
}
else if ((usrCrds==2 && usrHand==21) || (dlrHand==21 && dlrCrds==2)) {
    cout << "BLACKJACK" << endl;
    if ((usrCrds==2 && usrHand==21) && (dlrHand==21 && dlrCrds==2)) {
```

```
            cout << "TIE" << endl;
            usrCash += betAmnt; // return pot to the user
            betAmnt = 0;    // set betAmnt to zero to log a tie
        }
        else if (usrHand == 21 && usrCrds == 2) {
            cout << "YOU WIN!" << endl;
            betAmnt *= 4;   // blackjack pays 3 to 1
            usrWon = true;
        }
        else {
            cout << "DEALER WINS" << endl;
            betAmnt *= -1;
        }
    }
    else if (usrHand == dlrHand) {
        cout << "DEALER STANDS" << endl;
        cout << "TIE" << endl;
        usrCash += betAmnt;
        betAmnt = 0;
    }
    else {
        cout << "DEALER STANDS" << endl;
        if (usrHand > dlrHand) {
            cout << "YOU WIN!" << endl;
            betAmnt *= 2;
            usrWon = true;
        }
        else {
            cout << "DEALER WINS" << endl;
            betAmnt *= -1;
        }
    }
```

```
cout << endl;

if (usrWon) {
    usrCash += betAmnt; // award player any money won
    usrWins++;          // increment user's total win count
    usrStrk++;          // increment user's current win streak count
    // if current streak is user's best, update bstStrk
    if (usrStrk > bstStrk) bstStrk = usrStrk;
}
else usrStrk = 0;

// log the total amount of money won/lost this hand
sumCash += betAmnt;
outCash << betAmnt << endl;

// if user cannot bet anymore, the game is over
if (usrCash < 50) {
    if (usrCash == 0 ) {
        cout << "You ran out of money and were" << endl;
        cout << "kicked out of the casino." << endl;
    }
    else {
        cout << "You did not have enough money" << endl;
        cout << "to continue playing and walked" << endl;
        cout << "home in shame." << endl;
    }
    cout << endl;
    cout << "GAME OVER" << endl << endl;

    // still let user choose to see their stats on a game over
    cout << "See session statistics? (y/n)" << endl;
    cout << "Enter decision: ";
```

```
cin >> usrInpt;
cout << endl;

// see line 127, code is identical
if (usrInpt == 'y' || usrInpt == 'Y') {
    inCash.open("hndData.dat", ios::in);
    inBets.open("betData.dat", ios::in);
    while (inCash >> tempFlt && inBets >> betAmnt) {
        avgDvia += pow(tempFlt - sumCash/roundNo, 2);
        betDvia += pow(betAmnt - sumBets/roundNo, 2);
    }
    cout << "Hands played..........";
    cout << roundNo << endl;
    cout << "Hands lost............";
    cout << roundNo - usrWins << endl;
    cout << "Hands won.............";
    cout << usrWins << endl;
    cout << "Best win streak........";
    cout << bstStrk << endl;
    cout << "Average $ won.........$";
    cout << sumCash/roundNo << endl;
    cout << "Standard deviation....$";
    cout << sqrt(avgDvia / roundNo) << endl;
    cout << "Average $ bet.........$";
    cout << sumBets/roundNo << endl;
    cout << "Standard deviation....$";
    cout << sqrt(betDvia / roundNo) << endl;
}
// close remaining .dat files now that game is over
outCash.close();
outBets.close();
```

```
            // exit stage left
            return 0;
        }
        cout << endl;
    }
}
```

https://github.com/darlenepetal/csc-5/tree/main/PROJ

https://github.com/darlenepetal/csc-5/blob/main/PROJ/Project1_Blackjack_V8/main.cpp

646 total lines

588 lines of code

58 whitespace lines

204 commented lines