Simple Blackjack

Card Game

Author: Danielle Carlos

CSC 5

Winter 2019

Section: 40136

Table of Contents

**Introduction:**

Blackjack is a standard casino card game that can be played with as many as two to four people. The game is very simple: Whoever has the value closest to twenty one wins! Each player is dealt two cards, including the dealer. Then each player decides if they want to draw additional cards to try to get closer to twenty one or not. After this phase ends, the dealer hits until they hit seventeen and all players compare their cards if they do not have values over twenty one. Whoever has the closest value wins!

The logic of the game was straightforward and simple enough to follow. Despite its simple logic blackjack actually is very strategic, especially when bets are in play. While my program does not have a betting system it does have an earning system where you can only earn money for wins and draws. I wanted to make the base card game as close as possible for this iteration of the project.

**The Rules of the Game:**

**Objective:** To get as close to 21 as possible without busting

**Some Game Terms:**

**Hit:** The phrase to draw a card.

**Bust:** When the value of your hand goes over 21. Automatic loss.

**Blackjack:** When you get the golden starting hand of an Ace and a value 10 card. In my game it is an automatic win regardless of whatever the dealer has.

**How to Play:**

1. This game is between the player and the computer. Both will be given two cards to start off with.
2. If you get blackjack, the round ends with an automatic win. Otherwise the player will be asked to hit.
3. If the player decides to hit they will draw until they decide to stop or they bust.
4. The dealer reveals their hand after the player is done drawing cards. They will draw until they have a value of 17 or higher.
5. Whichever value is closest to 21 is the winner!

**Development of the Game:**

**What had come Before:**

The hardest part I had in translating the game’s rules into code was creating a “flexible” ace. While the understanding of the ace’s value is easy in reality, in computer logic it required many checks and the calculation of a fixed value.

In my previous project this had been handled by a variety of variables. However, in this iteration of the project the calculation of the ace was easily done due to the availability of arrays and, more importantly, vectors. Vectors allowed me to create a “flexible” array rather than having to worry about having static arrays and their size. It made the calculation much easier while also simplifying how the program kept track of what each player had in their hand.

**From Then to Now:**

In this iteration of the project, however, my goal was to equalize the odds of the card draw. In my previous project it had been handled by an rng generator that had no checks on how many of a card one had, so one could be pulling multiple aces and other values that would simply be impossible if only playing with a single deck of cards. I had also added a stats check which told the player how many times they’ve won or lost and how many times the deck has shuffled (in case anyone is feeling particularly cheated).

**A “Unique” Problem:**

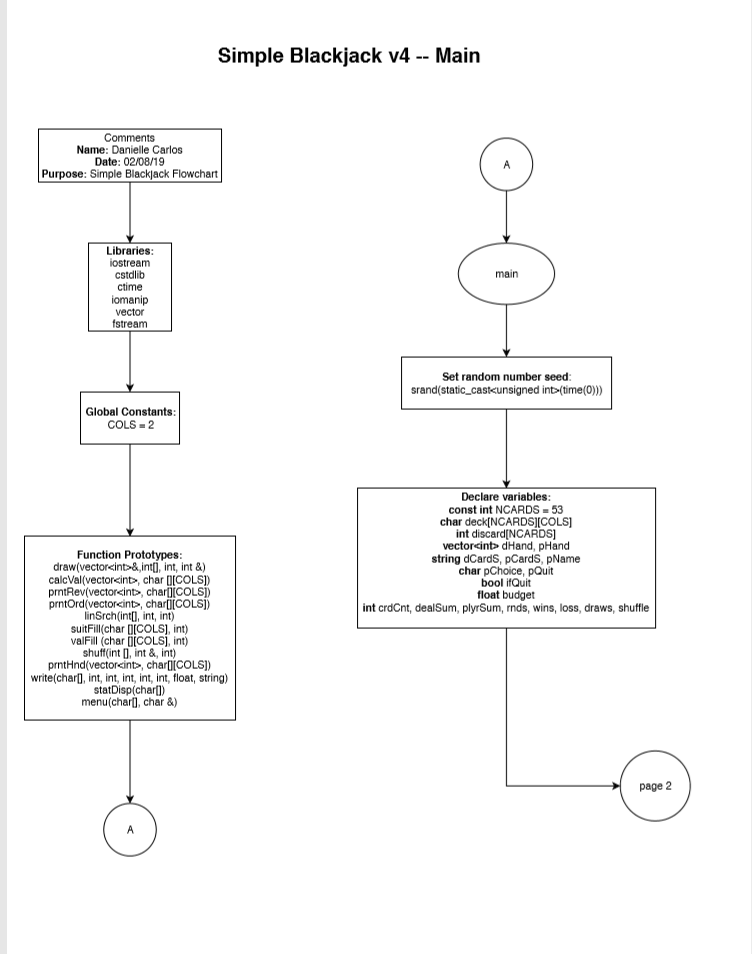
One of the difficulties of implementing a 52 card deck was ensuring “unique” pulls. Since the rng generator can pull anything within the range twice, I managed to simulate the “unique” pull by creating an array that would contain an index of the cards already “pulled” from the static array where the cards and values were held. Then every time a card was drawn it would have a reference for what had already been “pulled”, simulating a “unique” card draw.

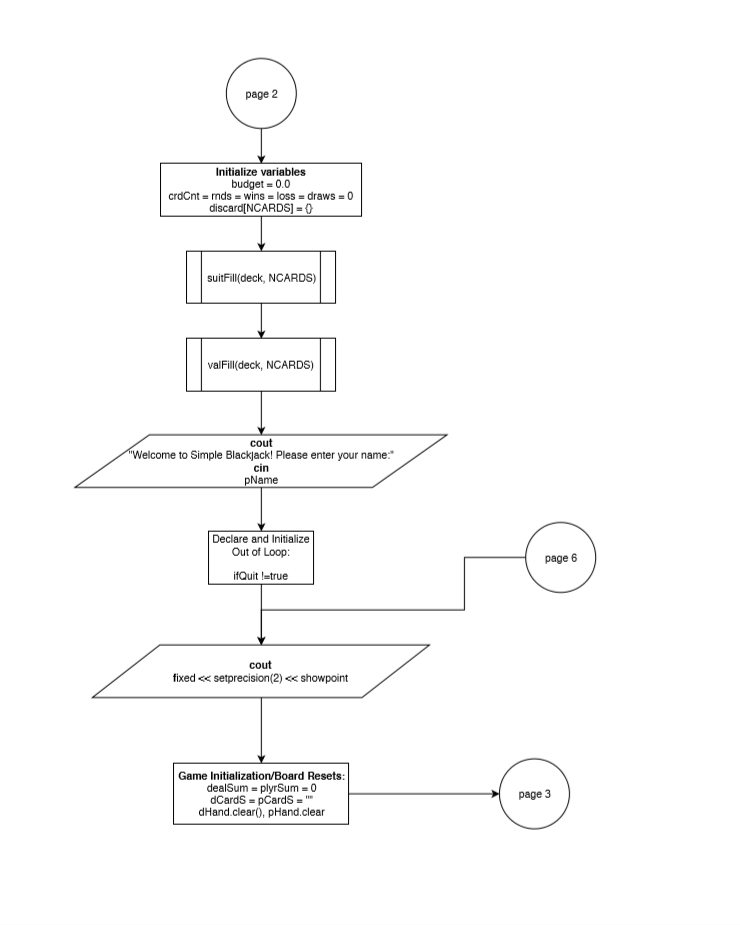
**Similarities to the Game:**

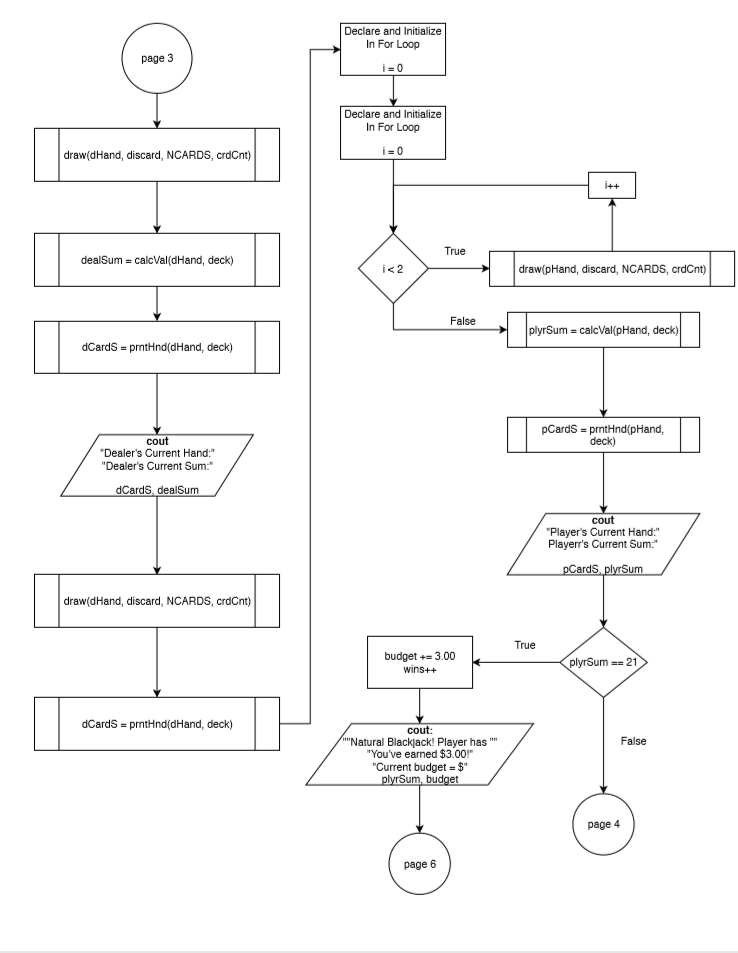
1. Win/Lose conditions are the same. Over twenty one is a loss, and closest to twenty one wins.
2. The dealer’s first initial card is hidden from the player, much like how it is in real life casinos.
3. The deck now is only played with 52 cards rather than an infinite amount

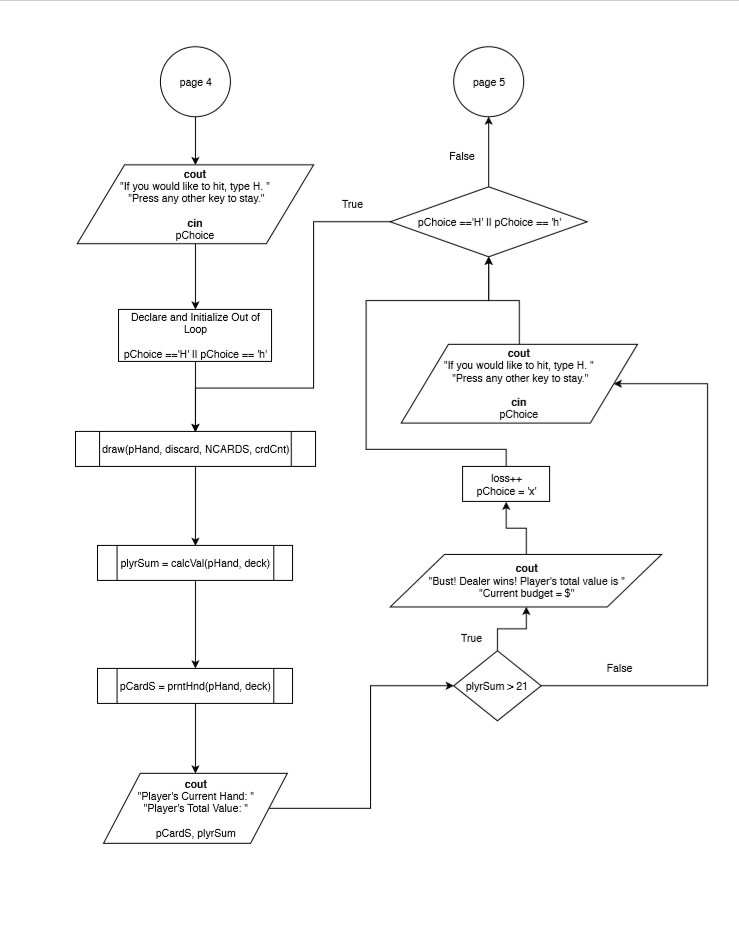
**Differences from the Game:**

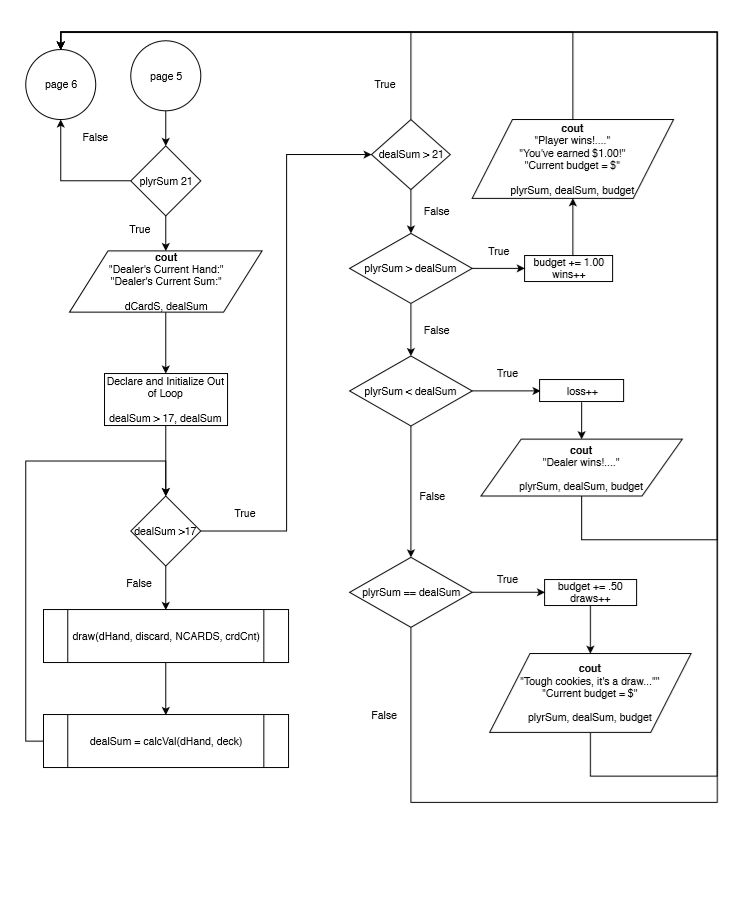
1. Bets are not in place. You can only earn money so it’s an ideal version of blackjack for anyone addicted to playing card games but hate losing money because of gambling.**On the Technical Side--The Flowchart and Logic Behind the Game:**

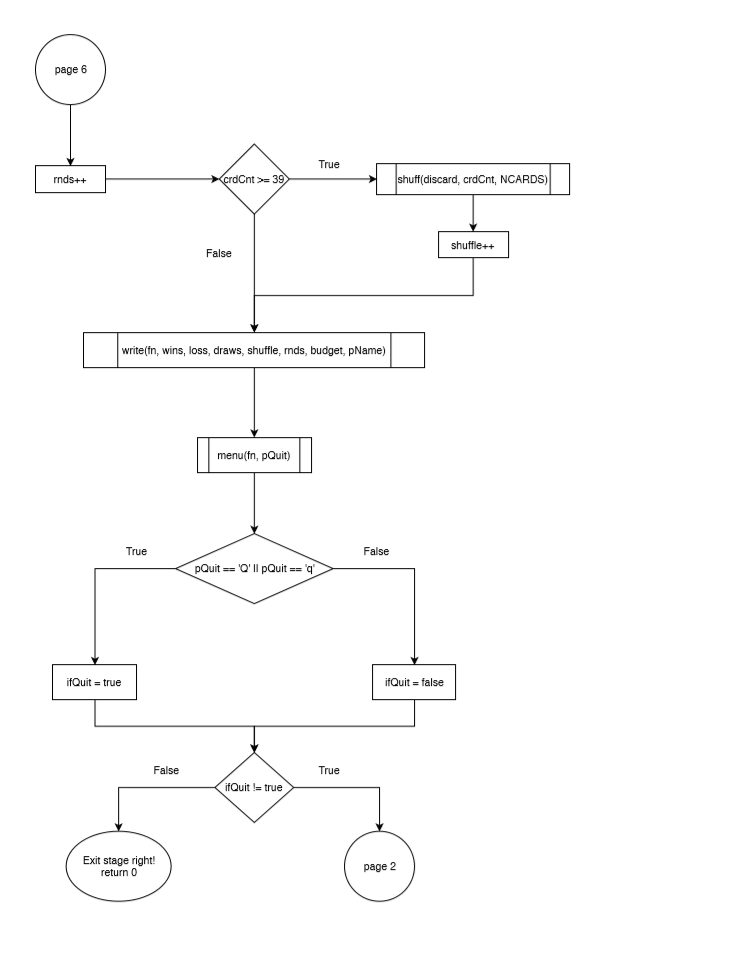
****

****

****

****

****

****

**\*\*\*This flowchart is only for main. The full flowchart is sixteen pages long and is in a separate file in the folder.**

**PsuedoCode:**

*//Set the random number seed*

*//Declare all variables*

*//Game begins here:*

*//Game initializes/Board Resets*

*//Calculation for Dealer's Initial Hand*

*//Draw initial card for dealer*

*//Obtain initial value*

*//Display Dealer's Initial Card*

*//Dealer Draws Second Card*

*//End of Dealer Initial calculations*

*//Begin Player Initial calculations:*

*//Player draws two initial cards*

*//Calculate the value of the player's hand*

*//Display of Player's Initial Total:*

*//Win Condition 1: Calculating if Player has Natural BlackJack*

*//If true*

*// Add 3.00 to budget, add to wins, skip to end of round*

*//Hit Phase Begins if Player has no Natural Blackjack*

*//Player Hit Phase Begins*

*//"Draw the card"*

*//Calculation of player's current hand*

*//Display Player's Current Value:*

*//Bust Check:*

*//If true:*

*//Make player choice false to end hit phase*

*//Else:*

*//Prompt user if they would like to hit or stay*

*//Repeat if they choose to hit*

*//Else if they stay:*

*//End of do-While Player Phase Loop*

*//End of Player Phase - Dealer Phase Begins Conditionally*

*//If player's sum is less than 21*

*//Reveal dealer's hand to player*

*//If dealer's hand value is less than 17*

*//Dealer draws card*

*//Dealer's hand is calculated*

*//If dealer's value is less than 17 hit again*

*//Else*

*End of While Loop*

*//Sort cards for presentation:*

*//Display Dealer's Full Hand/Player's Hand*

*//Bust Check for Dealer:*

*//If true:*

*//Add Money to Budget*

*//Skip to end of blackjack round*

*//Else:*

*//Dealer Hit Phase Ends-*

*//Comparison Phase Begins if player and dealer have not busted*

*//End of blackjack round*

*//Add to round counter*

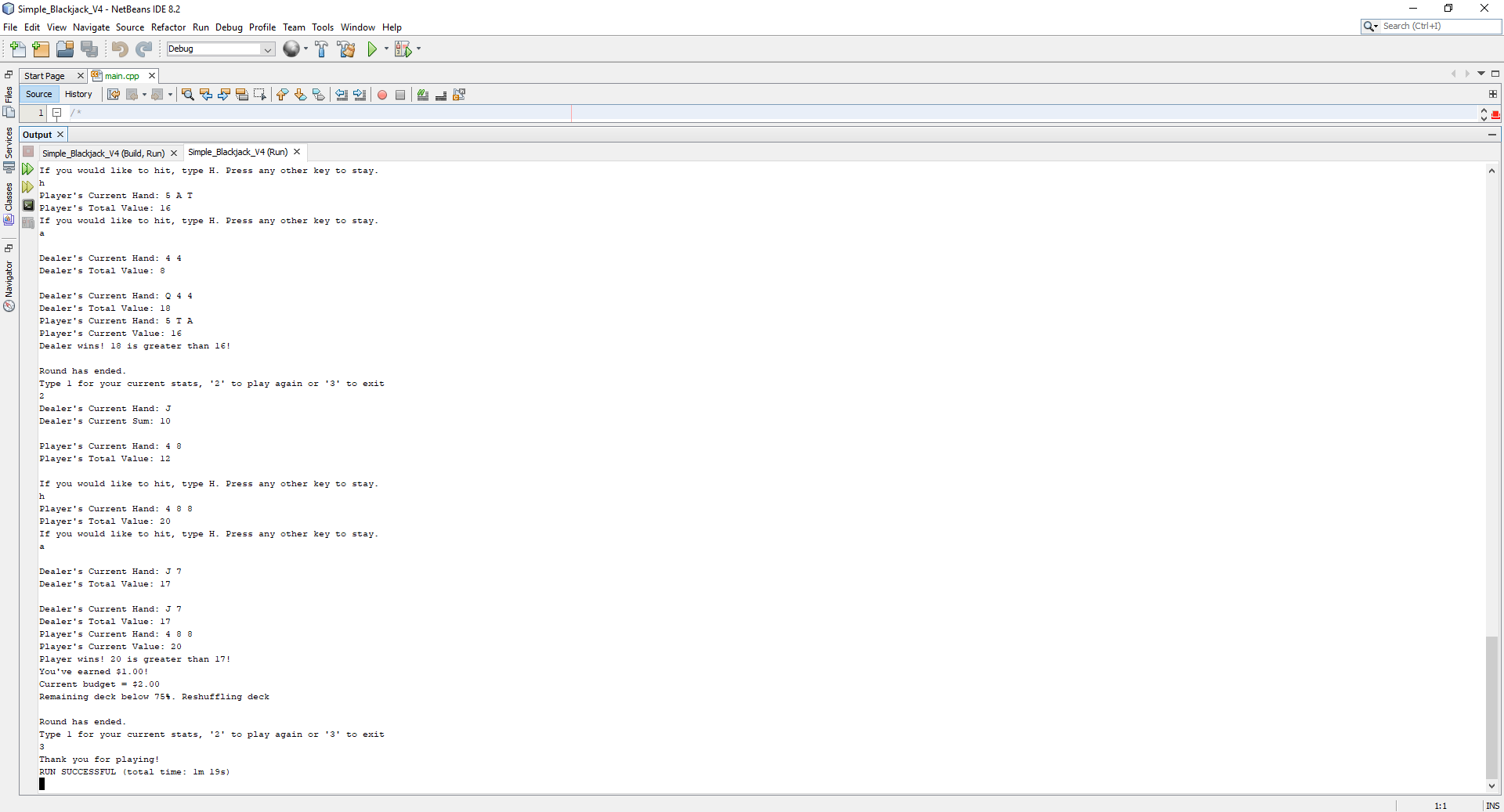
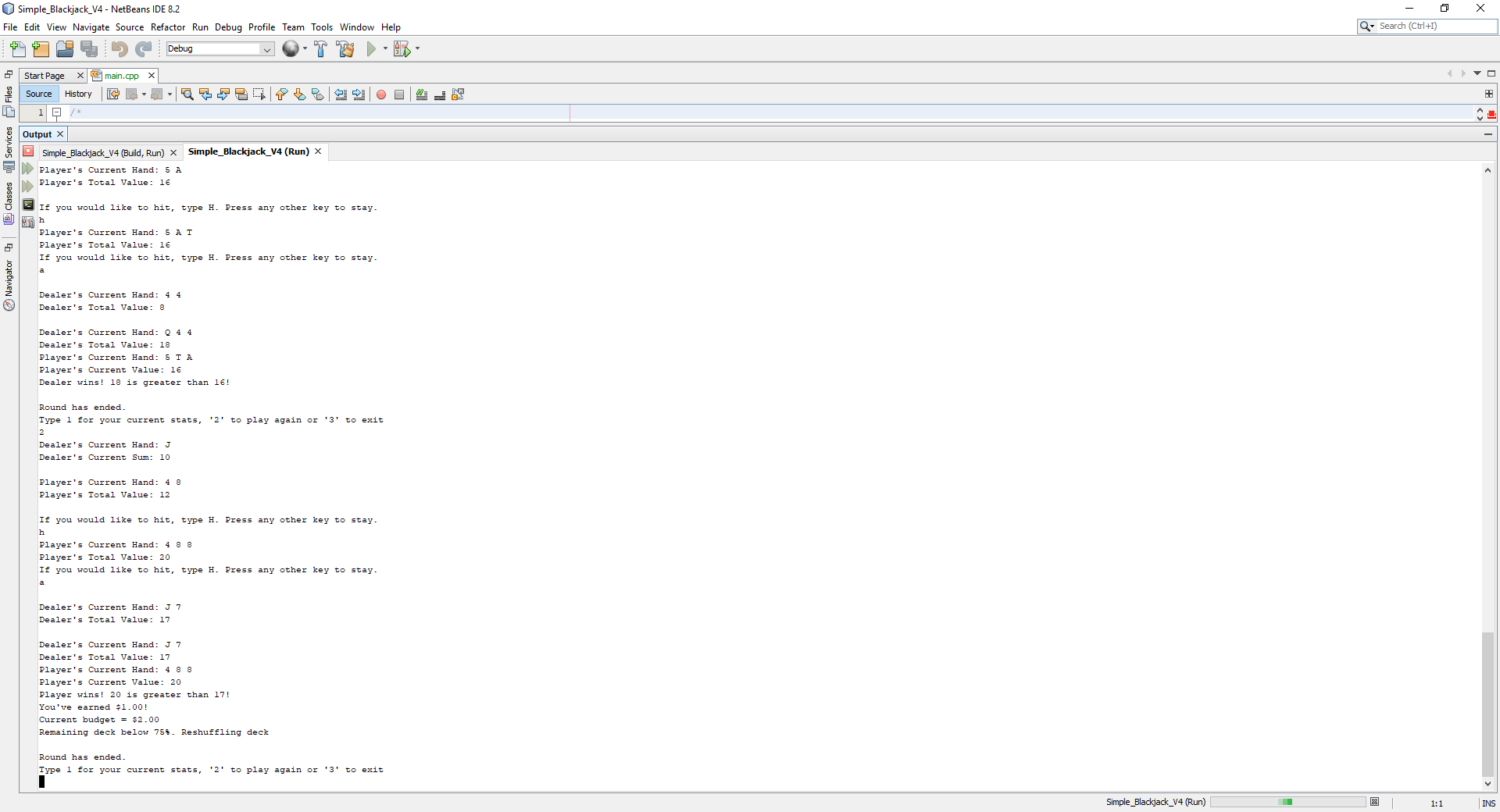
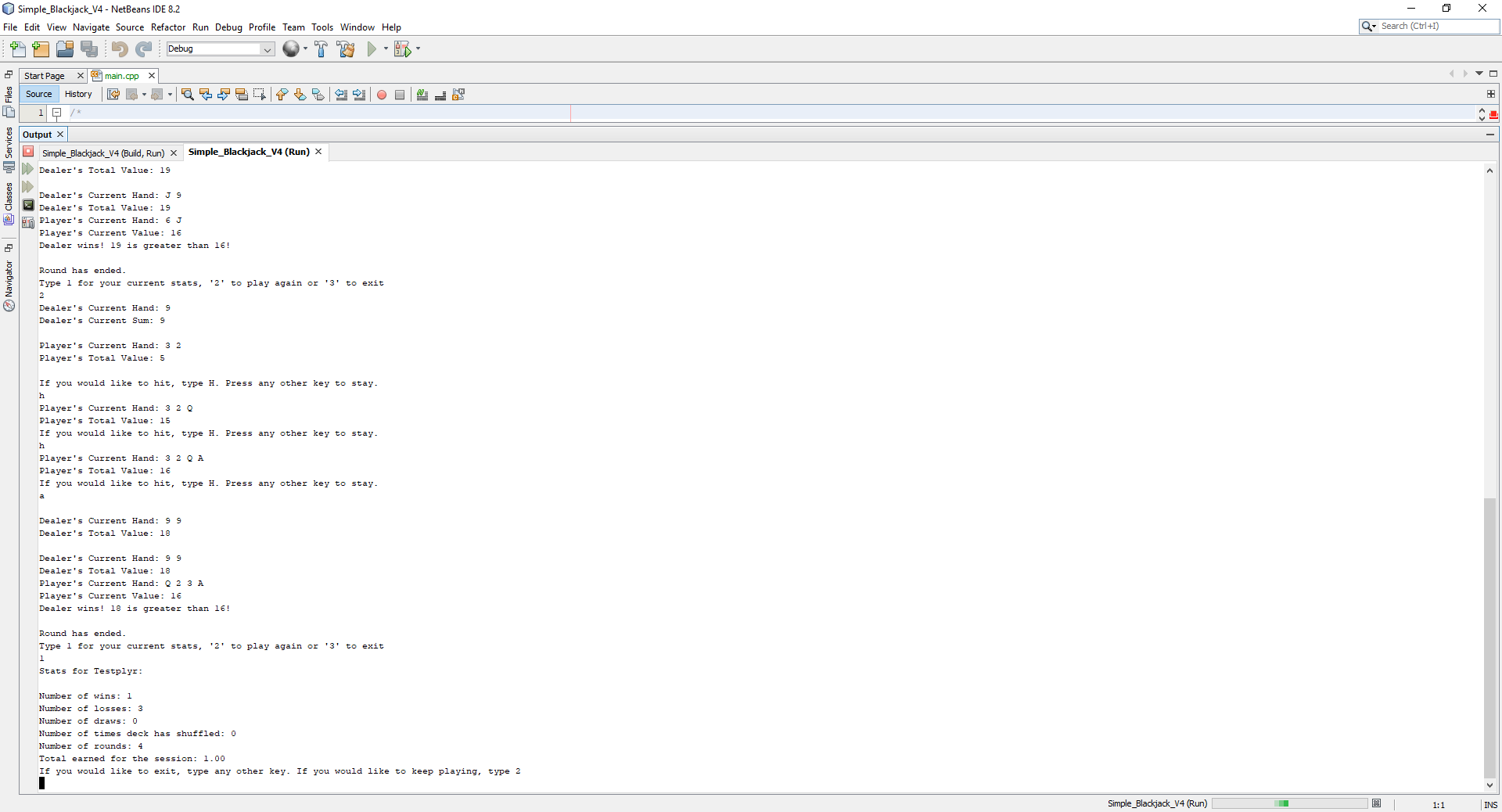
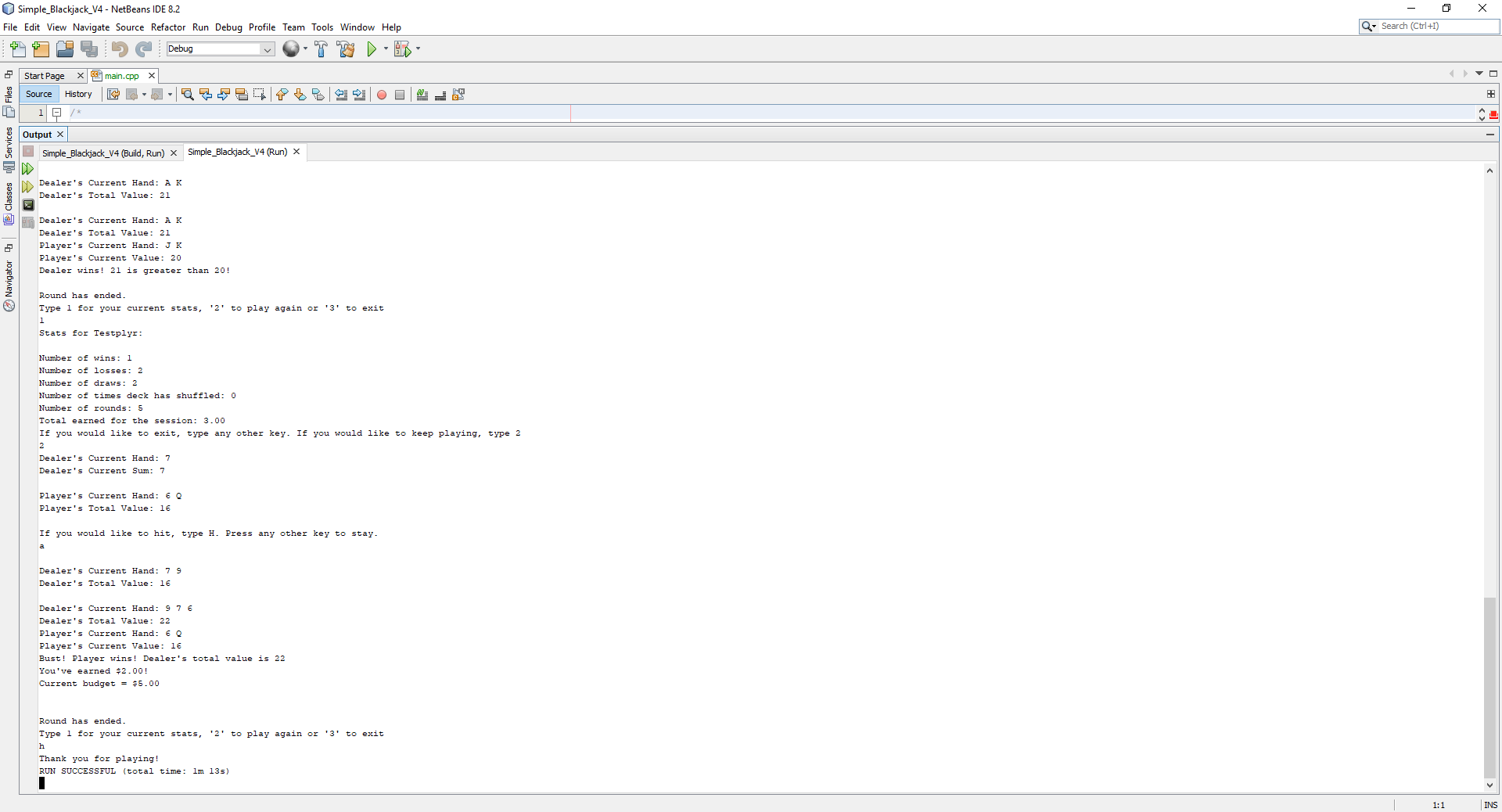
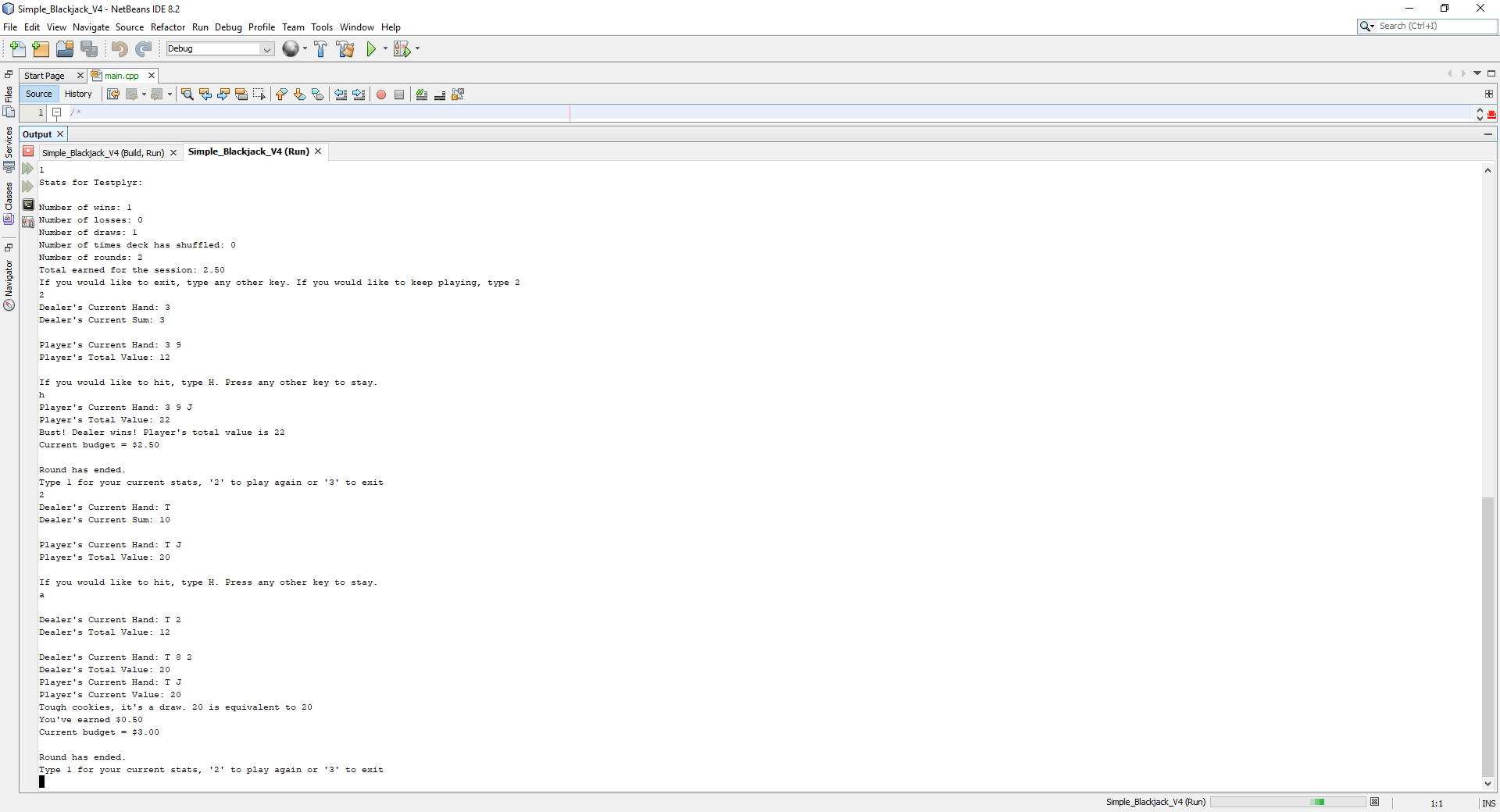
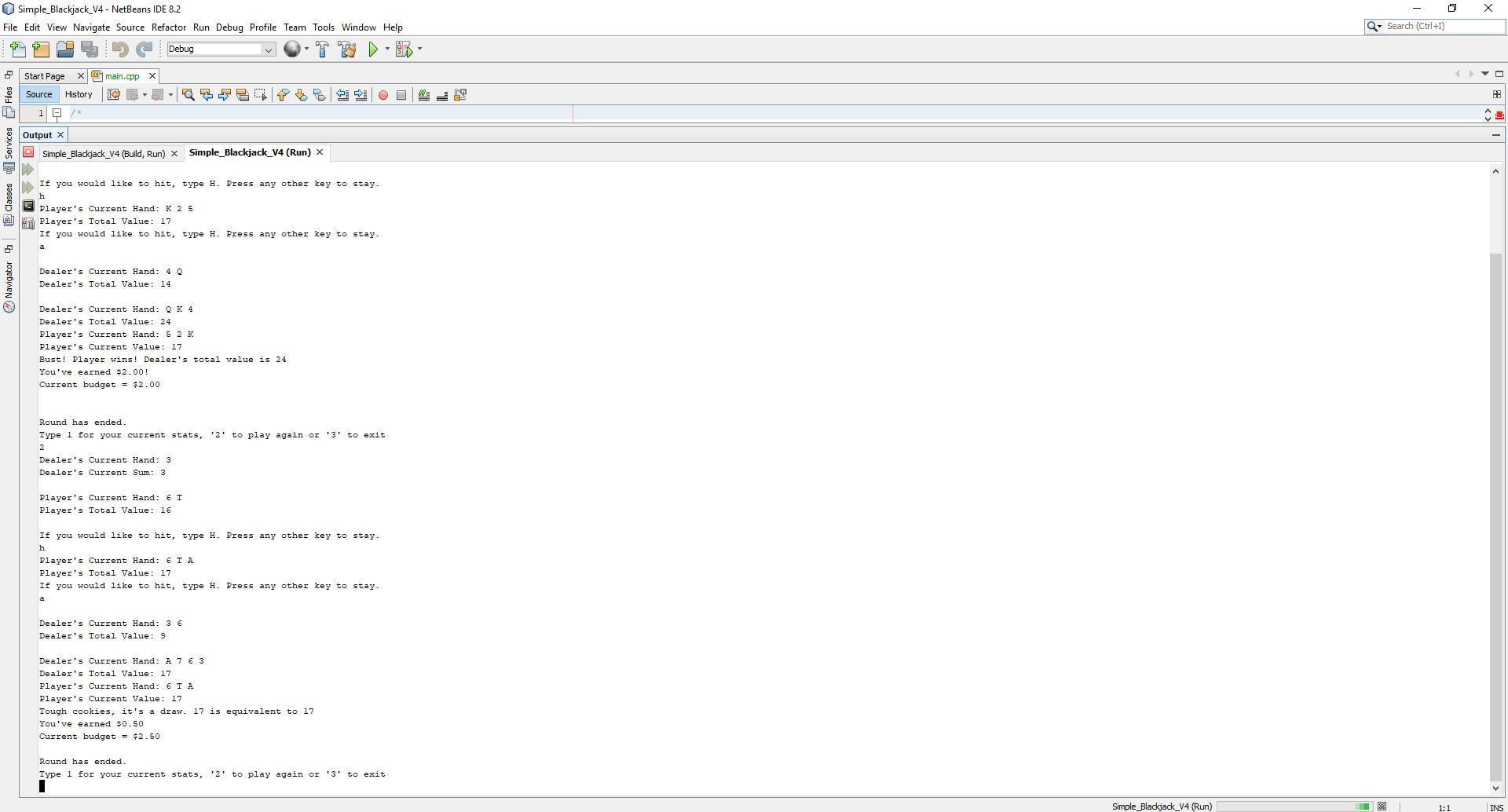
*//Check if deck is below 75%*

*//Prompt user to repeat or quit*

*//Exit stage right or left!*

**Proof the Game Runs:**

If, for some reason, the game has broken on the way, I have provided screenshots to prove the game at least ran while on my machine.



**Concepts and Constructs Used:**

|  |  |  |
| --- | --- | --- |
| Name | Description | Location |
| cout | Prints out things/lines | Line 77 first instance, used throughout program |
| libraries | Iostream, cstdlib, ctime, iomanip, ctime, vector, fstream | Lines 10-15 |
| Varaibles/literals | Holds information for the program to use | Lines 44-65 |
| Integers | Holds numbers | Lines 44, 46-50, 55 |
| Characters | Primitive data type, ASCII | Line 56 |
| Strings | Another word for character array | Line 52, 62 |
| Floats | The preferred way of holding decimal numbers | Line 59 |
| Bools | Determines True or False | Line 58 |
| Cin | Gets input from the user | Line 78 |
| Type Casting | Transforms one data type into another | Line 41, 398 |
| If | Individual if | Line 191 |
| If-else | Will execute one or the other | Line 145-159 |
| Nesting | Putting constructs inside other constructs | Line 145-159 |
| If-else-if | Will execute one of three (or more) statements | Lines 204-225 |
| Logical Operators | Used to compare similar data types with each other | Lines 117, 145, 165 |
| Condtional Operator | Otherwise known as the ternary operator | Lines 396-397 |
| Switch | A form of stating if/else, usually used for menus | Lines 298-313 |
| Increment/Decrement | Shorthand of either adding 1 or subtracting 1 | Line 104, 199 |
| While | A type of loop | Lines 174-181 |
| Do-while | A loop that does whatever is inside its scope first then checks to loop or not | Lines 132-160 |
| For loop | Also known as the counting loop | Lines 104-106 |
| Files input/output | Reading in from files and reading out from files | Lines 317-333, 301(function call), 335-343, 237(function call) |
| Function Prototypes | Let’s the program know what the function is going to be. | Lines 24-35 |
| Pass by Value | Passing in the copy | Line 254 |
| Return | Returns whatever value the function is supposed to return | Line 267 (first instance), used more after |
| Returning Boolean | Returns true or false | Line 381, 384 |
| Static variables | Only one constant variable to define the 2D Array | Line 22 |
| Defaulted arguments |  | Line 74 |
| Pass by reference | Passing in the original to make changes to it beyond main function | Line 415 |
| Exit function | Exits the function/program | Line 312 |
| Single Dimension Arrays | Static Arrays | Line 46 |
| Parallel Arrays | Used in this program with the index as a 1D array referencing a 2D array | Line 395-396 |
| Single Dimension Arrays as Function Arguments | Used to alter the index in card draws | Line 108, 109 |
| 2 Dimensioned Arrays | Held our static deck of 52 cards | Line 45 |
| STL Vectors | The constructs that make up the player’s and dealer’s hands | Line 51, 61 |
| Passing Arrays to and From Functions |  | Line 108, 109 |
| Passing Vectors to and From Functions | Done many times to calculate values | Line 108, 109 |
| Bubble Sort | The slowest way of sorting | Line 256-264 |
| Selection Sort | Another way of sorting | Line 272-284 |
| Linear or Binary Search | Linear only, Binary required sorting | Line 378-384 |

**References:**

1. Dr. Lehr’s Lecture/Notes
2. Gaddis, Tony. *C++ Control Structures through Objects* (9th edition)(Textbook)