**Project 1**

Title

**Roulette v4**

Course

**CSC-5**

Due Date

**February 3, 2022**

Author

**Zachary Lyman**

**1. Introduction**

This is a simulation of the popular casino game roulette. I have modeled my game off of the American version of roulette which uses a wheel of 38 numbers. Roulette is a game of chance where a player will make bets on where they think a spinning ball will stop on a spinning wheel. The wheel is divided into 38 equal spaces. 36 of the spaces are marked with the numbers 1-36. Each of these numbers are colored either black or red. The colors alternate for the most part, but there is a break in the pattern. There are also two zero spaces marked with the numbers 0 and 00 which are colored green. Here is an example of the standard American roulette wheel.

A picture containing object, roulette, green

Description automatically generated

Players are free to bet on where the ball will stop in a variety of ways. In my simplified version of roulette, I will limit these to only the most basic bets a person can do. These bets include black or red numbers, even or odd numbers, high or low numbers, and single numbers. Players who win are paid out in an amount that is almost proportional to the odds of the bet they are placing. For instance, if a player were to bet on a even number, they would have an approximate 47.4% chance of winning. This bet would be reflected with a 1:1 payout ratio, with the player receiving twice their initial bet back. The odds for a single number bet are approximately 2.6% which are reflected with a 35:1 payout ratio. Players can make new bets once the results of the current spin are resolved, and are free to let their bets ride, meaning that they keep their bets on the table until a loss. Like many casino games, the game only ends when the player leaves, either by choice or a lack of funds. Many casinos also have a standard minimum bet for table games.

**2. Version 1**

I began my project by simulating the spin of a roulette wheel. To do this, I utilized the ctime and cstdlib libraries in order to generate random numbers. Once I initialized a random seed type casted to an unsigned int, I need to declare some variables. Since the main purpose of this first version was to simulate a roulette wheel, I decided write it as a probability calculator. I made a variable called nGames that I could assign a number of games to so I could simulate a large number of wheel spins. I named two constant integers called NUMMIN and NUMMAX which where initialized to 1 and 38 to represent the spaces on a roulette wheel. I had a unsinged integer called random which would hold the random number to be generated, and a number of integers to hold the wins and losses for each type of bet I planned to implement. In order to run a large number of test I used a for loop with an integer named games which would run as long as games was less then the number of games I wished to simulate. I then assigned random to the random number generated, making sure that I would only receive random numbers between 1 and 38. I then need to figure out how to assign each number a color and how to deal with the 0 spaces since my random number could not generate a 0 or 00.

In order to assign colors to my numbers, I decided to use a switch case statement. Using an image of an American roulette wheel for reference, I assigned a color to each number as they corresponded to the image. In order to assign a 0 or 00 to the wheel, I decided to use a string variable called ballPos which would relate to the random numbers 37 and 38. I then used a pair of nested if statements to print the position that the ball landed on. I used Booleans to help me achieve this, which were initialized in the switch case mention previously. In order to properly display where the ball would land, I need my first nested if statement to evaluate only for the numbers 1-36, and my second for the numbers 37 and 38. It is in this second nested if statement that I used the string workaround to store 0 and 00 to the numbers 37 and 38. When running this program, I was successful simulating the spin of a roulette wheel as I was able to output my desired numbers and associated colors. Here is a screen shot of the code with an example output.

**A computer screen capture

Description automatically generated with medium confidence**

This is as far as I got with this version. In the next version I plan to test the probabilities of my wheel and bets to make sure they match with what is expected.

**3. Version 2**

My primary goal in version 2 was to test the probability of my wheel spin. To do this, I needed to evaluate the results of the spin with the bets I planned on implementing. To do this I would need to expand my if statements. Starting with my first if statement which evaluated the numbers 1-36 I made a new series of if statements. I started with comparing black and red numbers first, as these seemed to be the most straight forward ones to evaluate. I nested an if statement that would apply if the ball landed on a black number. Inside of this statement I would increment a variable called wBlk by 1 representing a win on a black number. I followed this by incrementing a variable called lRed by one which would represent a loss on a red number. I then repeated this if statement but changed it so that it would apply if a red number was generated. I then swapped the win loss variables to a red win and a black loss and incremented as before. In order to make sure that my code was working I then displayed output to the console showing the number of wins and losses for black and red. I then calculated the number of wins and losses by dividing the number of wins or losses by the total number of games and multiplying the result by 100. In my testing, I found that I was getting a percentage of 47.39% black wins and 47.36% red wins. I used the following linked website as a criteria for my percentages.

<https://www.casino.org/roulette/odds/>

As you can see from this website, the odds on a red or black number bet for American roulette is listed as 47.37%. This matches up quite nicely with my results.

Next I needed to evaluate the number of even and odd wins and losses. To do this, I copied my code for the red and black numbers and changed the if statements to evaluate if the random number was an even or odd number. I then incremented the corresponding wins and losses. I did the same for the low and high numbers as well. I then moved on to the zero numbers. To do this, I broke out of my if statement that was evaluating numbers 1-36 and made a new if statement evaluating numbers greater then 36. I then made two more if statements, one if the random number was 37 representing a 0 and the other for the number 38 representing 00. I then incremented corresponding wins and losses within these statements. I displayed the outputs for my new set of tests and the results are on the following page.

Text

Description automatically generated

**Expected vs. Observed Probability**

Red or Black

Expected: 47.37%

Observed: 47.39% and 47.36%

Odd or Even

Expected: 47.37%

Observed: 47.35% and 47.40%

High or Low

Expected: 47.37%

Observed: 47.38% and 47.37%

Single Number

Expected: 2.6%

Observed: 2.62% and 2.62%

As you can see from these results, my simulated roulette wheel which was tested at 6 million spins matched up nicely with my criteria. This is where I concluded version 2 of my project.

**4. Version 3**

Version 3 is where my project underwent the most change. I was satisfied that my probabilities matched those of the real world game and decided to abandon my probability calculator and try to make my project as interactive and game like as possible. To do this, I started with the menu.

I first displayed some basic information to the user telling them what game they were playing and how they would be interacting with the program. I then designed a menu for them to choose from. My menu needed to allow a user to make bets or quit the game. The user does this by inputting a single character relating to the option presented in the menu. The program will then ask them how much money they would like to bet. In order to implement this, I got rid of my for loop and nGames which I used previously to test a large number of games. I then created a variable called usrBank which would store the amount of money the user has to play with. I decided to make this a set amount based on a rule of thumb someone once told me about playing casino table games. The general idea is that you want to have around 20 times the minimum bet for the game. That way, you will have sufficient funds if playing the minimum bet to make enough bets to eventually win and stay playing. I decided initialize the users money to $200.00 as that is 20 times the minimum bet of $5.00. I then utilized a while loop ensuring that the users bet is at least 5 dollars, if it is not, they will need to re-enter a valid bet.

A computer screen capture

Description automatically generated with medium confidence

In order to make the game playable until the user wishes to quit, I put the menu and user selections in a do loop that would execute as long as their selection was not the sentinel defined as Q for quit. I then got rid of my variables that were used to keep track of wins and losses and replaced them with lines of code that would add to the usrBank variable for wins, simulating winning a bet. I did this for all of my previous test cases from version 2. I also added output in each test case to output a result of the get to the console. This was a little bit tricky, as I was writing a single line of output in pieces that were located throughout the code. For instance, the start of the result output would be located at the start of the if statement I used to evaluate if the random number was 1-26. Then, I had two statements evaluating if the number was red or black, which would output the corresponding color to the result output line. Then, each test case needed to output the results if the user chose those bets. A computer screen capture

Description automatically generated with medium confidence

I then needed to create a new test case representing the user betting on a single number. I did this by creating a nested if-else statement that was independent of the statements evaluating the random number between 1-36, and 37 and 38. For this bet to work, I need to prompt the user to input a number after selecting S for single number bet. I did this with a simple if statement after my input validating while statement. If the user selects S, they are then prompted to enter a number they would like to bet on. They chose a number 1-38, with 37 and 38 representing 0 and 00. The program stores this choice into a newly created variable called betNum. I tried to find a way to allow the user to simply enter 0 of 00 for those numbers, but was unable to find a work around so this would have to do. Back down in test case, I then compared their selection to the random number generated for that bet. If they won, I then added 35 time the users bet with the users bet then added that to the users bank. This is because the payout ratio for this type of bet is 35:1. I then wrote code displaying these results to the user as well.

I now needed a way for the program to terminate if the user runs out of money to bet with, as you cannot bet with negative money. Towards the end of my do-while loop I then created a test case that would evaluate if the user has at least $5.00 in their bank to bet the minimum amount. If they don’t, the program will inform them that they have insufficient funds and that the game is over. In order to cause the game to auto-terminate, I assigned the variable select with the char Q to force the end condition in the do-while loop. I also decided to add some output to the user to show them their final payout once the game terminates which is located just about the while statement that ends the loop.

I ran into several issues when writing this version of my code, The large number of if statements and conditions led to many infinite and broken loops. It was quite tricky to get right. Even in this version there are some conditions that will still cause Infinite loops as well as undesirable results. I refined it as much as I could, but opted to start on version 4 where I would polish what I had, and add some more functionality to my code in the process.

**5. Version 4**

Version 4 is the final version of my project. Most of my focus on this version was ironing out the bugs with my loops and nested statements which were giving me undesired results. I had to rearrange some of my if statements in order to correct many of these errors. The biggest error I ran into was a infinite loop when trying to exit the program. This was due to several errors in my code when evaluating if the end of loop condition was met by the users input. I also decided to add some new functionality to my code by providing two new options for the user. The first would be a running log that would keep track of the previous numbers generated by the code. This simulates the boards you will often find in casinos next to roulette tables that people will often use to try and guess more “accurately”. I would do this by writing and reading to a file. The next added function would be an option for the user to repeat their current bet. This would roughly simulate a player letting their bet ride on a single bet on the table.

A screenshot of a computer

Description automatically generated with medium confidence In order to create a running log of generated numbers, I used the fstream library and created a variable called ballLog. I started by having the program open and write the current ball position to the file in both the 1-36 if statement and the 37-38 if statement. This was needed in order to write 0 and 00 into the file by taking advantage of the code I had already written. I then created a new menu item called P for view previous numbers for the user to chose from. I then created a new if statement that would evaluate when the user selects P. I need this statement to fire before asking the user for their bet otherwise this caused unwanted output in my code. Within this if statement, I had the program open my log file in the read configuration and used a while statement with the getline function to read the previous ball numbers to the user. This is where I ran into trouble. When using that standard write to file option, each time the program is called to write to the file, It overwrites the previous entry. This is pointless for a running log of numbers generated. I change my write to file to the append configuration which gave me my desired results. I cannot exactly remember if this was taught to us in class or not but this was needed to produce my desired results, so I opted to use it either way. The only problem with using append was that I would now have an endlessly growing text file that would store not only the current games numbers, but all previous games as well. In order to get around this I had the program open the text file one more time now in the output configuration and placed it outside of my loops towards the top of the code. The idea was to have the program overwrite the file every time the program is ran in order to ensure an empty log for each new run.

There was some cleaning up I need to do after implementing the log functionality of my code, such as implementing a way for the user to input new selections after outputting the log or quitting the program after viewing the log. After doing this, I then moved on the next piece of functionality.

A computer screen capture

Description automatically generated with medium confidence In order to allow a user to repeat their bet, I used a for loop which would be located before I generate my random number, but after prompting the user for input. Instead of adding a new menu item, I decided to implement new lines of code that would prompt the user if they would like to repeat their bet after they enter a valid bet amount. This is done by storing a yes no choice input by the user to a new variable called repeat. I then use an if statement if the user inputs a y that asks the user how many spins of the wheel they would like to repeat. This input is stored into a new variable called spins. The for loop I created earlier would then repeat for a number a spins plus one that are greater than the default of 1. I need to format the loop this way to ensure that the loop always loops once for all the other user inputs, and only changes the number of loops if the user changes the spin variable by electing to repeat their bets.

A computer screen capture

Description automatically generated with medium confidence

A computer screen capture

Description automatically generated with medium confidence The remainder of my work on version 4 was focused on solving bugs and errors in my code. I had a couple more infinite loops to fix and some new unwanted outputs to solve. This was really just an exercise in repetition and experimentation in order to fix these problems. I also added use of the math libraries power function in order to meet the projects criteria. While I did not need this function for my code to work, I added it by raising one of my bet values to the power of 1 in order to utilize it without changing my numbers. I’m not sure how this code will look to any who knows what they are doing. But I’m content with the functionality of my code and my outputs when tested across as many criteria I can think of. Here are some outputs to show how the code runs.

A computer screen capture

Description automatically generated with medium confidence

A computer screen capture

Description automatically generated with medium confidence

A computer screen capture

Description automatically generated with medium confidence

A computer screen capture

Description automatically generated with medium confidence

A computer screen capture

Description automatically generated with medium confidence