**Project 2**

Title

**Roulette 2**

Course

**CSC-5 Winter 22**

Due Date

**February 12th, 2022**

Author

**Zachary Lyman**

**1.Introduction**

This is the second iteration of a simulated game of American roulette. Roulette is a game of chance played with a wheel comprised of 38 spaces. 36 of these spaces are numbered 1 through 36 and two are labeled 0 and 00. Numbers 1 through 36 have an alternating color of either black or red. 0 and 00 are colored green. A ball is spun around the wheel and stops randomly on one of the 38 spaces. Players are free to bet in several ways where they think the ball will stop. A player is free to make multiple bets for 1 round of roulette. A round of roulette ends and bet results are determined when the ball stops on 1 of the wheel’s spaces. Here is a picture of an American roulette wheel for reference.

A picture containing object, roulette, green

Description automatically generated

There are several different bets one could make while playing roulette. For my game I will only focus on a few of these types of bets form simplicity sake. For my game I will allow the player to make straight bets, red or black bets, odd or even bets and finally, high or low bets. Straight bets are made by betting on a single number, and if the ball lands on that number the player will win a payout ratio of 35:1. The remaining bets all have about a 50/50 chance of landing, and pay out at a ratio of 1:1. Many casinos have a minimum bet requirement for placing bets, and for my game I will enforce a $5.00 minimum bet.

**2.Planned changes from first build of the game.**

In the first build of the game, players could only play one bet type at a time. In the real world game, you can make multiple bets on a single spin. For the new version of the game, I intended to give the player the option to place multiple bets, mirroring the real world game. Players can also leave their bets on the table, or let it ride, as they are winning for multiple spins. Often players tend to repeat their bets as well, hoping to increase their odds of winning. This is something I also hoped to add to the game. I also wanted to add 3 new bet types players could make, the column bet, dozens bet and the corner bet. I was unable to add these 3 new bets however due to the time restraint of the project but hope to add them as I intend to continue working on this game for further practice. Some further functionality add to the game was the ability for the program to log the users bets and wins and print them out as a comprehensive list at the end of the program. There were also some changes to the base mechanic of the game which is the wheel spin as you will see in my further documentation.

**3.New Roulette Wheel Prototype Version 1**

The first step of improving my game is building a new algorithm to simulate the spin of a roulette wheel. To make it easier to develop and test this new wheel spin I decided to develop it first separately from my code from project 1. My goal for this new wheel spin was to store the numbers in order as they appear on a real-world wheel into an array. I decided to use a 2-dimensional array so I could also store a number value that would relate to the color of the corresponding number stored in the same column index of the array. I then needed a way to relate a random number to the index of the array in order to choose a number and color from the array. To do this, I decided on an approach that involved a little bit of math to somewhat simulate a realistic wheel spin.

Graphical user interface, text

Description automatically generated

In order to simulate how a real number is chosen on a roulette wheel by spinning around the wheel, I first related the number of spaces on the wheel with the degree measurement of a wheel. Dividing 360 degrees by 38 spaces give you a value of 9.47. This means that each space on the wheel takes up around 9.47 degrees of the wheel. I then simulated a spin by having the random number generator give me a number between 1 and 720. I did this so I could have an abstract ball potentially rotating around the wheel a full 2 rotations. To relate the randomly generated number back to the wheel I first subtracted the number by 360 if it was larger then 360. In all honesty I probably could have skipped this and just chose a number between 1 and 360 and it probably would have worked all the same. The next step is to divide the random number generated by the value 9.47. This will determine how far along the wheel in spaces that the abstract ball travels along the wheel before stopping. This operation does not return an int and I need a whole number to relate to the index of the array. I decided to use the floor function of the math library to round down the number. In theory, when I run a loop that iterates through the array and stops at the number I derive with my formula, I should be pointed to the theoretical space that the ball would stop if it traveled x number of degrees with the degrees being the random number.

Graphical user interface, text

Description automatically generated

The last step before testing my algorithm is to assign variables to hold the numbers pulled from the array and to assign a color to them. I did this by utilizing a simple switch case to evaluate the number pulled from the second row of the array and determine if the number was green, red, or black depending on if the number pulled was 0, 1 or 2 respectively. I moved my color and number assigning statements into functions as well as my wheel spin function and printed the results. Without testing for statistical accuracy, I was quite pleased with how my new wheel prototype performed and moved on to the next step.

**4.New Wheel Prototype Version 2(Roulette 2 Version 2)**

Note: The title of this program is incorrect, as I initially assumed that this would end up being the basis of my Roulette 2 code. However, I realized I would need a new protype program to develop my menu independently so This ended up remaining as the Version 2 Wheel Prototype.

Version 2 of my new wheel prototype is where I tested my wheel for statistical accuracy. I copied by code from project 1 that I used to test my previous wheel spin’s accuracy and replaced my old wheel spin with my new one. I had to re-assign some variables and test cases, but for the most part it was a drop in application of the code. I initially had problems with my results. After some further tinkering with my wheel spin algorithm, I was able to get a set of statistics that I was ok with.

Text

Description automatically generated

The real world win percentage for each of the 50/50 bets should be 47.37%, and for single number bets they should be 2.6%. As you can see, my statistics slightly less than ideal, especially when compared to my initial wheel spin, however being that there was only so much time to complete the project it ended up being good enough to move forward to the next phase.

**5.Menu Prototype Version 1**

The next phase of my project was to create a new menu system that would utilize functions. This was a simple task as it was mostly just copying the menu code from project 1 and implementing it into a set of functions. I wanted a way to display the menu at different points of a players game, so I decided to create a function to simple display the menu called dsplMnu. I then created a function called choice which when called is used to prompt and gather the players inputs relating to the menu. I then decided to create two vectors that would store a user’s input and could be read back within the main function. The idea here was to have the choice function loop, allowing the user to input multiple selections that are stored into the vectors. Once the loop is broken, the main function then can use a for loop that will iterate through each input that was stored in the vectors. I had some problems with my loop not behaving and ultimately this version became more of a sketch pad for the next version.

Graphical user interface, text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

**6.Menu Prototype Version 2**

For version 2 of my menu, I first tackled my loop problem. Once I had that squared away, I then decided to create two new functions that would display my two vectors. I intended to adapt these two functions later to add functionality to my code. To test my code, I simply read in several selections and bets, then called the display functions in main to make sure that my vectors where properly storing and outputting my input. Everything appeared to be working so I moved on to adding my new wheel spin and menu into my code from project 1.

Graphical user interface

Description automatically generated

**7.** **Roulette 2 Version 3**

Version 3 of Roulette 2 started off as a copy and paste of my code from project 1. Because of my previous prototyping I now had working menu and wheel spin functions to integrate into project 1’s code. The first steps were mostly copying and pasting. I removed the original menu and wheel spin code and replaced them by calling their respective functions. I then needed to replace some of project 1’s variables to match with my newly written code. This was a monotonous process, but it is also where I got a lot of my ideas for how I would implement new functionality to my game.

Once I integrated my prototypes, I then had to come up with a way to have my code evaluate as many numbers of bets stored in my bet vectors before prompting for new input. I also needed this to happen for a single wheel number, as the idea is to have the user make multiple bets for a single wheel spin. I did this by placing a for loop inside of my pre-existing do-while loop. The for loop would re-evaluate wins and losses and update the check conditions for the wins and losses by reading data from my two bet vectors. I then realized that I needed to add another vector to now hold a user input for single number bets. I created a function called snglNum to accomplish this by calling it from within the choice function to prompt the user to input a number selection into my new vector. I then created a new function to help me print data from a file which is called from the choice functions

After cleaning up and double checking some of my code I was able to get it to compile only to find that I had a major error. When inputting multiple bets, the user enters ‘n’ or ‘N’ to exit out of the selection loop and move back into main. Instead, when exiting the loop, the program crashed and returned a value of 1. After some research, I found that error code 1 usually meant that there was a small error in the code. After trying to fix the problem, I then would get a return of 2 or 139. I spent about 6 hours across two days trying to fix this problem. I had to add test outputs across different parts of my code and test different inputs to track it down because it ended up being not obvious. There ended up being an error in my choice function, a single line of code the kept me from entering a number into one of my vectors. I suspect that when I exited out of my choice function and attempt to read from my vector within main, the program fell apart because I was trying to pull from an empty vector. Either way, fixing this error bridges me to the final version of project 2.

Text

Description automatically generated with medium confidence

A screenshot of a computer screen

Description automatically generated with medium confidence

This is where my error was located.

**8.Roulette 2 Version 4**

After struggling to fix my coding errors in version 3 I became hard pressed for time and had to abandon some of my plans for project 2. I decided to skip adding the 3 additional bets as well as skip adding a feature that lets the user keep their bets for re-spins. I had some more features that were needed to fulfill the projects requirements so It became a game of time management.

In order to add a sorting feature to the game I decided to create two new vectors that would be updated at the end of each spin with the bets placed and win results of said bets. This was needed because at the end of each wheel spin cycle my vectors that store the users inputs are cleared so that they can accurately hold new user information should the player elect to play more rounds. I then used a pair of functions that would first sort the new vectors in alphabetical order, then display the results to the user at the end of the game. In order to make sure that the bet results stays with their respective bet selections, I created a bubble sort that used the information from my selection vector to sort in alphabetical order. I then placed my bet vector containing floats directly into the bubble sort function so that their index’s were sorted in the exact same way. In my testing I found that this worked quite well.

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated

I originally intended to also run some sort of a search on these newly created vectors, however I decided to abandon the idea because these new log vectors would dynamically change as the user inputs data. In order to run a search, I must know what I’m looking for and the fact that the user will enter an unpredictable selection means searching in this way just wouldn’t have been that effective. I decided to add and array of numbers in which I would search for a specific number. This had nothing really to do with the game, however it was needed to help fulfill the requirements. At the end of the game, I pose a simple question to the user. What is the answer to the ultimate question in life? This answer, at least if you read The Hitchhikers Guide to the Galaxy by Douglas Adams, is 42. I then have an array fill with 100 random numbers and run a binary search to look for the number 42. If the number is found, I display it to the user, much like the supercomputer called deep thought in Adam’s novel. If not I simply tell the user that the answer was not found.

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated

The final few things I did before calling the project done was to add a new menu item that would execute an exit function to terminate the game immediately. I also cleaned up some more of the code and fine-tuned some of the outputs to make the program more presentable and useable. I also added some more input validation when the user inputs their bets to make sure that they do not bet more then what is stored in their bank, because without it the user could bet indefinitely regardless of how much money they had left. Once these last few tasks were complete I decided that the project was good enough to turn in.

**9. Roulette 2 Version 5**

In the final version of project 2, I decided to go back in and double check a few things since the project was extended. One of my major concerns was my use of global constants that where used to define the dimensions of my 2 d array. The way I previously had my code written, these were needed in order for the code to run properly. I was nervous however, that this use case might not be approved and risked me outright failing the project. I went back in and, after looking back at our lectures on arrays and a little google searching I found a new way to use my function prototypes with the 2 d array without needing to use global variables. I’m sure that there is a better way to accomplish this then the way I ended up using but for now this will have to do. At least I was able to eliminate any global variables from my project to prevent an outright failure.

The next couple of things I did was add a new two new functions to meet our static variable and defaulted argument requirements. I had completely forgotten these and needed to add them in. One function simply keep a running tally of the number of wheel spins that occur in a single run of the program. The other function makes use of a defaulted argument to automatically display a default number to the user as well as display the total number of games played when called just before the program is exited.

**10.Conclusion**

If I had a main takeaway from working on project 2 it would be the importance of efficient debugging. I spent far too much time trying to fix the errors that were present in Version 3 and it completely killed my momentum. While I was able to eventually get the project into a presentable state, I had to sacrifice some of my original design plans. I suppose this was a valuable lesson in software development in general, as I am sure many programmers go through the same dilemma but on a much larger scale when developing software. I entered this project still unsure on how to properly write and utilize functions, but I can say that after completing this project I feel like I have a better understanding and have grown in my knowledge as a result.

**11.Pseudocode**

Libraries needed: iosteam, iomanip, string, cstdlib, ctime, fstream, cmath, vector

Function Prototypes:

Int spinWhl(int [] [38], int, int)

Int asgnNum(int [] [38], int, int)

String asgnClr(int [][38], int, int)

Void dpslMnu(float)

Bool choice(vector<unsinged char>&,vector<float>&,float,vector<int>&)

Void readlog(fstream&,string,unsigned char&)

Void sgleNum(vector<int>,unsinged char)

Unsinged char readSel(vector<unsigned char>&)

Float readBet(vector<float>&,int)

Void dspBL(vector<float>&)

Void dspSL(vector<unsigned char>&)

Void bublSrt(vector<unsigned char>&,vector<float>&)

Void fillAns(int[],int)

Void bblSrtA(int[],int)

Int binSrch(int[],int,int)

Int gameCnt(int)

Void dspGCnt(int, int lghtSp=186282)

Main

Initialize random seed, type cast to unsigned int

Declare variables: ROWS,COLSANSW, random, space, indx, sglNum, wheel, ans, betNum, select, bet, betLog, selLog, usrBank, betAmnt, sel, betSel, ballPos, logLine, color, ballLog

Initialize variables: usrBank=200, sel=’B’

Reset ball log for new game

Call functions to fill ans array and bubble sort ans array.

Do-while loop, while sel!=’q’

Display game header

Call choice function

If choice function returns true, sel=’q’

Initialize indx=0

Simulate wheel spin

Random=random number%720

Indx=call function spinWhl

Space=call function asgnNum

Color=call function asgnClr

Loop to determine bet results

For size of select vector

Reset betAmnt=0

Read bet from bet vector [i]

Remove bet from usrBank

Read selection vector [i]

Read single number vector [i]

Input ball position into file

If ball lands on non zero space

Display ball position and user bet

Evaluate black/red bets

Store results in selLog,betLog

Evaluate even/odd bets

Store results in selLog,betLog

Evaluate high/low bets

Store results in selLog,betLog

If ball lands on zero space

Assign 0 or 00 to numbers 37 or 38, display ball position to user

If user made single number bet

Evaluate single number bet

Store results in selLog,betLog

Check is usrBank is less than 5.00

Display that they are out of funds and force end game condition

Check if end game condition is met

If met, display payout and game over

Increment I and continue for loop for i<select vector size, if not break loop

Empty select, bet and betNum vectors for new user inputs

Increase game play count by calling gameCnt fucntion

Check do-while loop end condition sel!=’q’

Continue loop for sel!=’q’, or break out of loop

Display bet log

Call bublSrt function

Call dspSL select log function

Call dspBL bet log function

Display final payout from usrBank

Display Ultimate question of life

Declare int val=42

Int ansI=call binSrch function

If binSrch does not return -1, display val and indx location of val to user

Else display that answer was not found

Call dspGCnt to display the running tally of games and speed of light to user

Return 0 to exit program

Void dspGCnt

Display the number of games played as well as the speed of light to the user

Int gameCnt

Makes use of a static variable and adds it to a game count variable when it is called

Int binSrch

Declare and initialize high low

Do-while loop

Declare middle to average of high low

Check if val is equal to middle, return middle if true

Check if val is less then middle, high now equal middle -1 if true

Else low equals middle -1

Check end of loop condition low is less than high, continue loop or break out

Return -1 if break loop

Void bblSrtA

Declare swap

Do-while loop

Swap equals false

Decrement n

For loop, I is less then n

If a[i] less than a[i+1]

Swap equals true

Declare temp equal to a[i]

A[i] now equals a[i+1]

A[i+1] now equals temp

Increment I and check end of for loop

Exit for loop

Check do-while end condition swap is true

Void fillAns

For loop I less then n

A[i] is filled with 100 random numbers

Increment I and check if loop end condition met

Break loop

Void bublSrt

Declare n equals size of selLog

Declare swap

Do=while loop

Swap equal false

Decrement n

For-loop while I less than n

If selLog[i] is less than selLog[i+1]

Swap equals true

Declare tempCh equals selLog[i]

selLog[i] equals selLog[i+1]

selLog[i+1] equals tempCh

Dcelare tempFl equals betLog[i]

betLog[i] equals betLog[i+1]

betLog[i+1] equals tempFl

increment i

check for end for loop condition

break loop

check do-while end condition swap equals true

break loop

void dspBL

for-loop I less than size of betLog

display betlog[i] with formatting

increment I

check end of loop condition

break loop

void dspSL

for-loop I less than size of selLog

display selLog[i]

increment i

check end of loop condition

break loop

void log

open text file

display previous ball positions by pulling form text file

close text file

bool choice

declare repeat, s, b, ballLog, logline,

call dsplMnu function

initialize repeat to n

do-while loop

prompt user selection input

if user inputs x, exit function to immediately exit program

check if repeat equals y

do not accept q as user selection if true

prompt for new selection input

if input is not q

if user inputs p call log function

prompt for new selection input

push user selection into select vector

If user inputs s

Prompt user to input a single number to bet on

Push user number selection to betNum vector

Else

Push 1 into betNum vector

Prompt user to input bet in dollars

Check is bet is less than 5 and if so, prompt for new dollar input

Check if bet is larger then usrBank funds, if so, prompt for new dollar input

Push bet input into bet vector

Subtract bet input form usrBank

Display to user the amount of funds remaining after placing bet

Prompt user to input more bets

If the user does not have funds to make another bet, force no

Else user enters q, return true and exit function

Check end of do-while loop condition repeat does not equal n

Break loop if met

Return false and exit function

Void dpslMnu

Display menu item list and how many funds the user has to bet with

String asgnClr

Declare colNum, color

colNum equals array[1][i]

switch case

1 case color equals black

2 case color equals red

Default case color equals green

Return color and exit function

Int asgnNum

Declare space

Space equals array[0][i]

Return space and exit function

Int spinWhl

Declare indx

Indx equals random%38

Return floor function of indx and exit funciton