Project 1: Muster

Austin Ngo

ango26@student.mtsac.edu

Prof. D. Atanasio

Introduction

Muster is a two-player dice rolling game where both players roll six 6-sided dice to try and reach 1000 points first. If both players reach 1000 points on the same turn, they continue until one of them has more points than the other at the end of the turn. At the beginning of each turn, each player is prompted for how many times they would like to shake the dice. The game will roll the dice this many times, but only the last set of rolls will be counted. The scoring system is broken down as such: If a player rolls a straight flush (1-2-3-4-5-6) that player earns 300 points. If a player rolls 3 pairs of numbers (including 4 of a kind and 1 pair) they will earn 150 points. If a player rolls at least 3 of a single number, they earn whatever number they rolled multiplied by 10 (3 sixes earns 60 points, 3 fours earn 40 points, etc.). If a player rolls at least 1 five then they earn 5 points, and if they roll at least 1 one then they earn 2 points. A player is only allowed to earn points once for their roll, so they will be rewarded with the highest possible single instance of points for their roll, rather than the most amount of points added together. Once a player has exceeded 1000 points as well as the other player's score, they win. When the game is over the players will be prompted to play again, restarting the game. If they choose to end the game, it closes with "---- Hasta La Vista Baby ----".

Design and Implementation

I began writing this program using the random number generator function we previously used in class. This function utilizes the random header, but because the function we created was not originally seeded, I had to seed it using time 0 as the seed, which sets the seed of the random function to the current time of the host computer. After implementing this, I wrote out the main function of the program. Because this program is supposed to be highly modularized, the main function should only consist of prompting the user to start the game and the function calls necessary to print the rules and begin playing the game. I also implemented the exit message in the main function.

Following the main function would be the function to actually play the game, aptly named *play*. I understand that functions should be 10 lines or less, but I truly struggled on reducing the size of my play function as well as the scoring algorithm. I initially wrote the play function out to simulate the game, then I refactored it into as many individual functions as I could. I tried to avoid writing functions with only a single line, since I believed they would be

redundant since I could just write a single line of code in its place. One issue I ran into while running my game was the scores not being tracked properly. I initially only had one variable keeping score for each player, so when I would print the score they earned for that turn it would actually print their total score as the amount they earned for that turn. The solution for this was to implement a temporary variable that holds the score that the player earned for that round, and add that value to the total score of the player after they rolled.

Another issue I encountered was the roll values giving weird results for the second player. The first player would roll the dice and it would count the score without any issue, but when the second player would roll the results of the roll became jumbled, likely because I was using the same function and variables to roll for each player, so when the second player would roll it would combine their rolls with the initial rolls from the first player. The solution to this was to reset the variables and the arrays that had held the roll results for the first player by setting all of them equal to 0, then letting the second player roll. A second set of variables and arrays can be used for the second player, but I felt that it overcomplicated my program by adding to many parameters for the functions that had to do with dice rolling, keeping score, and printing the results after each turn.

Possibly the largest function in this program is the scoring function, I could not figure out how to make it any smaller, without breaking it into pieces and splitting it into smaller functions, I did not want to try this since it seemed like a nightmare. I initially tried writing the portion that scores the 3 pairs using if else statements, but after writing down each different possible combination of pairs I quickly realized that this was not feasible. After some thinking I came up with a much more elegant solution which was to iterate the array which stores the frequency of the player's roll in a for loop, if any number had a frequency of 2 it would increment a counter, if the counter hit 3 then that indicates that there must be 3 pairs, awarding the player 150 points and breaking out of the loop.

One rather embarrassing but easy to fix mistake I made was using the comparison operator "==" instead of the assignment operator "=" when assigning the scores in the scoring function, the amount of time I spent trying to find this mistake is concerning. The worst part was I had written it properly for the first few if statements, but I encountered some late-night synaptic flatulence when writing the last few if statements for this function.

The most complex part of this program is easily the scoring algorithm, simply because there are many conditions for scoring points, which must be written in a specific order such that it will always award the player the highest possible points for their roll. This meant that the conditions for the highest points must be evaluated first, and the lowest possible points are evaluated last. It is possible to score 0 points on a turn, in a game with a max score of 1000 games can last upwards of 50 rounds, so perhaps adding additional ways of earning points would shorten the game significantly.

If I were to pick my favorite part of this program/assignment it would be the dice rolling function that I wrote. I chose to use two arrays instead of a bunch of variables or just a single array to keep track of the rolls to make scoring (in my opinion) easier. The rolling function iterates a for loop based on the user input “shakes”, which iterates another nested for loop which stores the latest roll results in an array called rollResult. This allows the user to only “shake” the cup once but will still result in a roll for each die. The second for loop resets the array which counts the frequency of rolls from rollResult, called dieTotal. The third for loop iterates rollResult and increments each element in dieTotal if it appeared in rollResult. I spent quite a bit of time figuring out the simplest and best way to count not only the results of the roll but also a way to store the roll frequencies into an array.