**Farkle Project Brief**

*The Game*

Given the three scenarios for the final project, I decided to develop a program that follows a full multiplayer dice game called Farkle. It commences by displaying the rules to the user, read from a text file implemented within the program. Outlining instructions on how to play and any strategies the user may use to obtain victory. Afterwards, it becomes a turn-taking matter that involves continuously rolling dice, accumulating points, while making strategic decisions on whether to take risks on rerolling any remaining dice.

*Approach*

First and foremost, I had to develop a text file that would be read by the program to display to the user. Since it consisted of a lot of text, it is only reasonable to use a separate file to ensure proper code readability and performance. I used the fstream class to retrieve the content of the file, along with a while-loop to print every line of text the text-file contained.

Afterwards, I developed a separate player class that would be responsible for constructing player objects to iterate through, keeping track of both the players’ names, as well as their overall score as the game progresses. Within the class, contains the constructor’s declaration with setter and getter methods to retrieve the player object attributes.

Since there is a dice-rolling mechanic and point-system that needed to be written, I made the game class which would consist of all the methods and functions required to establish the main logic behind the program. Beginning with setting both private and public instances where an array of dice, dice count, and total player count would be declared at the start of the program. Two functions would handle rolling and rerolling dice for players, while another would be responsible for calculating the amount of points the player obtained during a single roll. Setter and getter methods were established to keep track of dice values and points to score according to the game’s rules.

Finally, there is one primary function that would be called to run the game, combining all the different elements together to provide the game’s loop. Prompting the main user with the number of players that would be playing, creating player objects based on that amount, and afterwards commencing the game after each player rolled a total of one thousand points in a single roll. The game’s main looping mechanic works by iterating through a vector that contains all the player’s objects. Every turn would be decided based on the index in which each player is located within the declared player vector at the start of the game’s loop. I ensured that local variables within the function’s logic were in place to keep track of temporary values such as the number of points that a player obtained during a reroll, without finalizing those points that would add onto a player’s overall score. Since the game contains an element where if a player were to roll no point values, then they’d lose all those points during the round and pass their turn onto the next player. The game’s loops are primarily controlled by Boolean values that would change depending on the end-game condition. One player needs to acquire a total of ten thousand points for there to be a potential winner. Once the winner is decided, the Boolean value is immediately set to end the program prompting a winner onto the console.

Two separate classes were used to outline the game’s logic with the main function containing the separate functions that would run the game until a player is declared a winner.

*Debugging*

I used an iterative process to develop each function or method that would run the game. I would focus on developing one function at a time while testing its functionality by continuously placing it in the main function and running the code to see the results on the console. Any obvious errors were picked up by visual studio’s compiler notifying me of the issue at certain lines within the program. I focused on the more obvious bugs while the less obvious issues required further trial and error keeping track of each of the program’s element outputs. I personally like to use cout within different parts of the function to keep track of certain values to make sure the logic is working properly and where it begins to fail. If I see that a function’s output is not as I desire, then I make sure to look back at the structure of that part of the program and either change it, improve it, or remove it altogether if it won’t entirely contribute to the main program. There were also times where I simply needed to include certain classes beforehand, such as the string class or ensure I’m using namespace std to improve my program’s readability. I needed to address the most critical parts of my program before moving forward with development. It saved me a lot of time writing one section at a time rather than writing it all at once. I was able to quickly and easily pick up on where my problem was located and how to address it. Any doubt I had was mitigated by doing a bit of research and looking back at previous building block assignments within the course.