Name: Christopher Matian
Date: 10/14/2017
OID: 933308644

Original Design:

The design for my program was rather simple. I pulled a lot of what I learned from Project 1 and leveraged it here for lab 3. I started off by pegging down what kind of functions I wanted to be used for the main menu of the program. For this, I opted to use the validation function I had produced for Project 1, and created 2 additional functions. One function would control a program exit message and the other would return whether or not the die was loaded or regular (this was purely made for menu display).

I more or less used the same concept I had implemented for Project 1. Pass in all of the details to the Game constructor and initialized the Die objects within that same constructor. This allowed me to easily *morph* the objects into their relative class types (both were initialized as Die objects but I would change either of them to loadedDie objects if a certain criteria was met).

Changes in the Design:

A lot of my design ended up executing smoothly and how I expected it to. I had three hitches altogether, one of which was related to copy/pasting (internally) a **cout** statement for the second die object, another relating to a memory leak, and the final one relating to how I would set the sides for each player's dice. More details on all of these hitches will be in the final reflection below.

Altogether, very little was changed in the final program. I opted to keep the menus within main.cpp and let the game class control assignment of initial variables and also generate the Die objects that would be assigned to each of the two players. So, really, very little did change in implementation. I was personally surprised with this lab because from beginning to end I managed to implement the entire thing without straying from the plan I had written down and visualized ahead of time, and ran into almost zero issues that needed more than 5 minutes of debugging. Probably the most smooth assignment I've had so far.

Reflection and Design Problems/Challenges:

As previously mentioned I ran into a few hitches - see below:

1) The cout print issue
- A really stupid and minor oversight on my part. I had essentially been copy/pasting cout blocks and changing the values to correspond with their related variables. In a few cout statement blocks I forgot to change the variables to output the dice results for player 2, so I ended up with values that were identical each round. A quick scan of the code helped me determine what the issue was in the end.

2) Memory Leak
- The mantra of "*for every new there must be a delete"* rang true. I had created 2 new Die objects in my Game constructor, but forgot to delete them in my run( ) function. Simply adding ***delete d1*** and ***delete d2*** for each of the objects fixed the issue right away. I confirmed the results on FLIP by using valgrind.

3) Player Die Facing
- This part of the assignment was rather interesting because I had to account for the fact that the player's could determine how many sides their own die had. It was essentially unique to each player. I pored over this for a little while and finally I determined that since I was using inheritance and objects, I could just store the side value in the object (when it's created). It was a rather silly thing to get hung up over but the concept of Objects/Classes is still rather murky for me.

Test Plan: (10 rounds max)

Test Case	**Input Values**	**Driver Functions**	**Expected Outcome**	**Observed Outcome**
Game Run #1	Player 1 = Regular Die with 6 sides. Player 2 = Loaded Die with 6 sides.	void Game::run()	Player 2 should win with a 50% chance of rolling the max value of his/her die.	Player 1 Scores: 55 Player 2 Scores: 68 PLAYER 2 WINS
Game Run #2	Player 1 = Loaded Die with 10 sides. Player 2 = Loaded Die with 4 sides.	void Game::run()	Player 1 should win with a higher face value on his/her die.	Player 1 Scores: 62 Player 2 Scores: 34 PLAYER 1 WINS
Game Run #3	Player 1 = Regular Die with 10 sides. Player 2 = Regular Die with 10 sides.	void Game::run()	Either player can win this match with an even matchup.	Player 1 Scores: 47 Player 2 Scores: 45 PLAYER 1 WINS
Game Run #4 - Testing with higher values	Player 1 = Loaded Die with 400 sides. Player 2 = Regular Die with 300 sides.	void Game::run()	Player 1 should win with a high-value loaded die.	Player 1 Scores: 2735 Player 2 Scores: 1205 PLAYER 1 WINS
Game Run #5	Player 1 = Regular Die with 20 sides. Player 2 = Loaded Die with 8 sides.	void Game::run()	Player 1 should win with the higher die value offsetting the fact that player 2 is using a loaded die with 8 sides.	Player 1 Scores: 134 Player 2 Scores: 59 PLAYER 1 WINS