Lab 3 Reflection

Overall, I found that this lab was easily the hardest yet. While the first two labs had simple prompts just to test knowledge of simple concepts such as pointers and fstream, this one had you test out inheritance through objects. I started this assignment by creating a Die object which contained member variables for the number of sides on the die object as well as the dieType (loaded or not loaded). The default was not loaded (basic die with no biased return). I included member functions that rolled the die and returned a random number generated between 1 and the total number of sides, a getDieType function that returned whether the die was loaded or unloaded (in the form of string), and an int function that returned the number of sides. I then created a loadedDie class that inherited from the Die base class. The changes in this class involved setting the default DieType to "Loaded" and overloading the roll() function to return upper range values on every alternate roll. If the die had an even number of sides, it would generate a number within the upper half of values. For example, an 8 side die would return a value of either 5, 6, 7, 8 on every alternate roll. An odd side die returned upper values but due to the uneven break of the even number of sides, I had the die return a number from the top half which contained less total values than the bottom half. So a 7 sided die would return 5, 6, 7 rather than 4, 5, 6, 7. I felt that this was better design for the game. This part of the lab gave me the most trouble. I didn't know how I would load the die. I initially thought that I would just return the top value on every alternative roll but felt that this made the loadedDie too biased, and led to a pointless game where the loadedDie would win everytime. I also made the Game class, which contained the two Die objects, a menu function that took in user input and initialized these two objects, a play function that looped the game while using the user inputted total rounds as a loop control varaiable, and a displayResults function which outputted the results to the screen. I used a 2D int array to store the values during the play function and output the values and determine the winner during the display results function. I felt that this was the easiest part of the lab. I still struggle with user input but I've learned new ways of testing it such as using cin.fail(). I will continue to Improve my user input test on every new assignment. This is by far my biggest problem with programming thus far.

Test Table

Menu Test	Entered B to exit game	After compiling and running, this resulted in a segmentation fault when trying to exit the game as the other member functions of the game class still ran in main. I had to include a conditional in main that checked the truth value of menuSuccess. If the player chose to exit the game, menuSuccess was set to false, and the main returned a 0 to exit the game. THis got rid of the segmentation fault
userMenu() function prompting user if they wanted to use loaded die or not.	Typed 'y' instead of 'yes' for using Loaded Die in userMenu() caused errors.	This forced me to chance the response variable from a string to a char. I also had to include cin.ignore() in order to clear the input buffer after every char extraction. An uncleared input buffer resulted in a new line character being extracted by future extractions.
Menu Test	Typed D instead of 'A' or 'B' in the first menu to start the game. Correctly caused the program to output "Invalid Response" but caused stored entered in the input buffer to input values when the loop started over.	Had to include a cin.ignore inorder to clear the input buffer.
Loaded Die Biased Test	I initially had issues with my Loaded Die class not overloading the roll() function from my Die class. I had to include tests to ensure this by making the loaded die return a concrete value.	After research, I learned that I needed to include a virtual before int roll() in the Die header page. This fixed the issue and I was able to make the loadedDie object return biased results.
Loaded Die Biased Test	I initially had the loaded die return the max value which was equal to the value for numberSides (sides of Dice).	As described above, it initially outputted the wrong value but once I included the virtual, the max value was correctly outputted each time. I then implemented a much more

		advanced method for loading the dice.
Deriving loadedDie from Die class	I initially ran into issues from deriving loadedDie from the Die class.	Research taught me the correct syntax for deriving loadedDie from the base class Die. I needed to include (: Die(int a)) in the loadedDie constructor.