My original design included a Dice Class, a Loaded Dice Class, a Player class, and a Menu Class. The dice class included an integer N that held the number of sides of the dice, a default and custom constructor as well as a function that rolled the dice. The loaded dice class inherited dice side variable as well as the roll dice function, except that it overloaded the function to produce the increased chances of rolling a high number. The player class would hold the names, scores and types of dice the player used, as well as having getter and setter functions for those variables. The menu class consisted of functions to display the main menu, and the game class included all other functions needed to play the game, such as the menu for the game play options (dice, dice sides, number of rounds, etc) as well as functions to print the round results and display the final winner.

While I was implementing the game, I made several changes to the classes I had originally designed. The dice and loaded dice class were unchanged, but I ended up removing the Person class as well as combining the menu and game classes. I had found the person class slightly unnecessary as I decided not to ask for the player's names, instead just calling them Player 1 and Player 2, as well as the fact that the player class did not have any of its own functions other than getters and setters. Instead, I decided to make the remaining Person variables (types of dice, score, etc) part of the game class so that class's functions could access the variables directly. Since the menu class only contained one variable, that would be utilized by the game class anyway, and one function declaration, I decided to move those functions into the Game Class.

The biggest challenge for me during this lab was data validation. I had toyed with it a bit in the previous labs, but the major hurdle was figuring out how to handle a string input. I found that with my previously used methods, if the user entered a string and the string happened to contain a character that passed the data validation, the game would proceed in unexpected ways. It took a bit of trial and error, but I eventually figured out how to accept a string (getLine) and then parse the string into the appropriate variable (atoi() to convert integer or indexing the string). I also ran into a few logic errors that are explained in more detail after the test table.

I also went back and forth on my method of loading the loaded dice. I knew that I wanted to make the change of rolling high 1 in 2 so that in a game of loaded dice vs. regular dice, the loaded dice wouldn't always win. However, I first tried making the loaded dice have a 50% chance of rolling in the upper third of it's limit. After a few tests, I felt that that wasn't loaded enough, so I changed it to a 50% chance of rolling in the upper quarter. Still displeased, I finally landed on a 50% chance of rolling one of the top two numbers in its range (5 or 6 on a six sided dice).

Test Case	Inputs	Expected Outcome	Actual Outcome
Correct character Input Main Menu	Y or y	Game proceeds and user is prompted for dice variables	Error message prints$_1$
Correct character Input Main Menu	Q or q	Game proceeds and user is prompted for dice variables	Error message prints
Incorrect Main Menu Input	5, W, !, Yes., Quote	Error Message Prints, user repromoted until correct input is received	Error Message Prints, user repromoted until correct input is received

Correct Round Input	10	Game proceeds	Game proceeds and asks for dice type
Incorrect Round Input	0 ,-1, Word, 01	Error Message Prints, user repromoted until correct input is received	Error Message Prints, user repromoted until correct input is received
Correct Dice Type Input	L, l, R,r	Game proceeds	Error Message Prints$_2$
Correct Dice Side input	5,10	Game proceeds	Error Message Prints$_3$
Incorrect Dice Side Input	0, -1, Q, Sides	Error Message Prints, user repromoted until correct input is received	
Play a game	Dice 1 – Regular, 5 sides Dice 2 – Loaded, 5 sides Rounds – 10	Rolls generate random numbers, loaded dice rolls higher on average, whichever dice wins has it's score incremented by one, ties result in no score increase, and the winner is correctly declared.	The loaded dice did on average roll higher, but the score was calculated by adding the dice rolls to the players score, so the winner was not accurately calculated.
Play a ridiculous game to see if the game crashes from high input values	Dice 1 Loaded 1000 sides Dice 2 – Regular, 1000 sides Rounds - 1000	Rolls generate random numbers, loaded dice rolls higher on average, whichever dice wins has it's score incremented by one, ties result in no score increase, and the winner is correctly declared (the final score should always be equal to or less than the total number of rounds).	Game did not crash, the final scores added together equaled 999, which means there was one tie, and the loaded dice player won by a significant margin.

1. This error was caused by an incorrect OR function. I originally the statement
 while(!cin || !(playChoice == 'Y')||!(playChoice == 'Q'))
{ Error Statement}

This meant that if I entered Y, since that even though it was a correct input, it would still trigger the not Q part of the statement (and vice versa). I thus changed it to

while(!cin || !(playChoice == 'Y'|| playChoice == 'Q'))
{Error Statement}

2. This was the same issue as the menu choice, and the same correction fixed the issue.

3. I had written while(sidechoice1 > 1) when it should have been while (sidechoice1 < 1).