Andrew Clos
CS-162 - 400
Lab 3 Design, Test Table, and Reflection of War Game

Design:

Begin by prompting user to quit or play. Then display menu options for the user to establish game parameters (# of rounds, type of dice for each player, number of dice sides for each player). Will be able to reuse much of the menu functions from Project 1's assignment, and input validation for integers should be a complete carry over as a Class from prior assignments. Will experiment with adding Boolean (yes/no T/F) entry and develop an input validation method for this.

The game is "round" based and requires certain results to be outputted to the console after each round – a print function will likely be needed to handle this. End of game will need to output the final score count for each player and announce a winner (separate print function for end of game). It will be good to have the game stay alive in the main menu via a while loop, where this can return to the top and ask the user if they'd like to play again after the first game is run.

Removing as much as possible from the main method is a design decision. I am not entirely sure this is always best practice, but I wanted my main method to hold almost nothing in regards to "important" code or functions of the game itself. This seems like it would have better portability or at least be easier to add my Classes and functions to a project if I was working as part of a much larger team.

Classes:

Game: The game will contain the menu and collect data entry from the user to set up the parameters of the game (rounds, dice status, etc). These will be passed as an array of integers to other functions as needed.

Key functions:

mainMenu: Game lives here, prompts user to play or quit, calls the other methods in proper order.

paramMenu: Collect user entry of game parameters

showParams: Print the legend of the game parameters (Rounds, Player 1 Dice, Player 2 Dice, Player 1 Sides, Player 2 Sides), and then print the current user entered values right below it.

printDice: Seemingly simple print function that prints ASCII "art" dice out with the game menu. Will use the random function though and have counter variables *during testing* to determine if the random functions are truly random.

beginGame: Initialize the Die objects, make the rolls of the dice and keep tally of the score. Per specs, will also need to print out these scores for each round.

endGame: Print the final score and declare a winner.

Die: Since this is a parent class, it will need to have a virtual rollDice function because the LoadedDie class will also have a slightly different version of that function. It will also need a virtual destructor. It will contain get and set methods for the number of sides and take the number of sides as an integer when called.

LoadedDie: This derived class will inherit the attributes of the Die class and its main difference will be in its roll dice function. In order for a LoadedDie to consistently (but not always) return higher rolls than a normal Die, this roll function will get to roll again (up to two times) if it rolls something below half of the max possible roll. For example, on a six sided LoadedDie. If a one, two or three is rolled, it rolls again. If that second roll results in three or less, it will roll a third time.

Test Plan:

Test Category	Description of Test	Expected Results / Handling
Loaded dice testing	P1: yes, P2: yes P1: no, P2: yes P1: no, P2: no P1:yes, P2: no	Should properly handle this by calling functions from either Die or Loaded Die. Test by running 100 rounds (with 6 sided dice), averaging total score to make sure average rolls are around 3.5 for normal and about 4.5 for loaded. -Will use "step into" debugging function in Visual Studio to watch which functions are called.
Numbers of dice sides	Different numbers for each player Same number of sides for each player One sided dice for one or both players.	For normal dice, the average should be (numSides + 1) / 2, therefore, run 100+ rounds and examine average rolls.
"Fairness" testing	Given two dice of the same type and same number of sides, does one player win more often?	-This should be a 50/50 split over time, and the game's "run the same simulation again" option will be very helpful for running through many iterations and logging results.
Input validation: Boolean entry (new for me for this assignment	1. Enter symbols: '%', '&', ''' '"', space, etc. both by themselves and sandwiched between valid input.	-Should output error message that the character is not allowed, ask user to try again. -It should reject the entry, post message and ask user to try again.

Andrew Clos
CS-162 - 400
Lab 3 Design, Test Table, and Reflection of War Game

	2. Enter allowable char/chars next to invalid ones	-In the case of a space character "y y" it may read this as a yes/true.
Input validation: is an integer	1. Entering leading zeros, any other non 1-9 char in the first position or any non 0-9 in subsequent positions. 2. '2.1' float / double handling. 3. Adhere's to size boundaries of integers. If limit is set too high or not at all.	-Explain special case of leading 0 not being allowed with cout message. -Must make sure entry isn't being truncated. If user enters '2.1,' we cannot assume they meant '2'!! -If not establishing an upper limit, warn user that their entry is above size of signed int (32,767), etc.
Input validation: adheres to limits	1. Make sure the optional limit checking feature of input validation is working by entering values higher or lower than limits. 2. Enter 0's, negative numbers. 3. Check numbers that are equal to the bounds of the limits. 4. Check numbers 1 greater or less than the limits.	-if limits are exceeded, program may through a segmentation fault -could indicate a logic or even syntax issue (variable swapping).
Random Function	1. Is it truly random – will test by running 100+ round games and looking at average values to make sure they are not skewing in a certain direction.	Expect to see the average dice roll = (1 + max sides) / 2 when large numbers of rounds are run.
Memory Leaks	1. Valgrind with -v options. 2. Test different combinations of Loaded dice and normal dice. 3. Test case of quitting right away	-need to find out if I am deleting things properly with derived classes / virtual function and destructor calls. Expect some leaks at first!

Reflection:

Andrew Clos
CS-162 - 400
Lab 3 Design, Test Table, and Reflection of War Game

Bool validation: I added Boolean input validation to my Input Validation class since I wanted this program to ask the user for "Yes or No" and not use a "1" or "2." I adapted my integer validation to create boolean input validation and proved to be similar and slightly simpler. I wanted TRUE to allow the input of "Yes", "YES", "Y", "y", "T", or "t" and FALSE to allow "No", "NO", "N", "n", "F", "f," so I implemented the convert-to-lower-case function of my string validation in to the bool validate method as well.

I moved more of the program to the Class functions and removed almost everything from main. In previous assignments, I had while loops running in the main method and was creating arrays there to "feed" the functions. I decided to have the mainMenu function of the Game class control more of the game and keep running with a while loop there. That way, fewer individual functions needed to be called from the main method. This feels cleaner overall, but I am not sure if this is best practice for coding or not.

As a new programmer, this was my first time ever using inheritance and I learned a lot. I found one of the TA's forum posts extremely helpful regarding the need to use virtual functions and constructors – Thank you Thomas! I also struggled a bit with memory leaks and played around with delete and pointing to NULL. Finally, I was able to build a leak free program when I realized I had my delete's in the wrong place. Simply moving them to the end of the same function where the dynamic objects were created solved the problem I was having.