Clinton Hawkes
CS162-400
Lab 3 Design Document

Design:

Program will have a main function, a Game class, a Die class, a LoadedDie class, and an input validation function.

1) Main function – only used to seed the rand() function, create a Game object and call the Game::run() function to start the game.
   - A) Create a bool variable "State" to hold the current state of the game
   - B) Call the menu function to display the main menu
     - I. Ask user if they want to begin the war games or quit
       - a) If begin war games, return true to the run function
       - b) If quit, return false to the run function. (game will end)
   - C) Create a while loop that loops until the state variable is false
     - I. Ask user how many rounds they want to play
       - a) Set numRounds variable equal to input
     - II. Ask user what type of die they want player 1 to have (reg or loaded)
       - a) Set player 1 die type to user input
     - III. Ask user what type of die they want player 2 to have (reg or loaded)
       - a) Set player 2 die type to user input
     - IV. Ask user how many sides player 1 die should have
       - a) Set number of sides for player 1 die to user input
     - V. Ask user how many sides player 2 die should have
       - a) Set number of sides for player 2 die to user input
     - VI. Dynamically create die for player 1 and player 2 according to user spec
     - VII. Create loop that iterates until numRounds is reached
       - a) Call Game::match function
         - i. Function will simulate player 1 and 2 rolling die. Highest value wins round.
     - VIII. Overall winner of the game is calculated
     - IX. Total rounds won by each player will be printed to console. Overall winner will be announced.
     - X. Die object for player 1 and 2 will be deleted. Pointers for these objects set to NULL.
     - XI. Game is ended by setting "state" variable to false.

Game Class: Used to implement the War Games. Main() takes a back seat and uses the Game class to call all supporting functions to play the game.

Variables:	long numrounds	Functions:	Game()
	p1Score		mainMenu()
	p2Score		run()
	Die* p1Die		match()
	Die* p2Die		
	string p1DieType		
	p2DieType		
	winner		

I. Game::mainMenu()

I. Used as main menu to ask user if they want to play or quit

I. If they want to play, true is returned to the calling function

II. If they want to quit, false is returned to the calling function

II. Game::run() - used to call all functions require to play the game.

I. This function is used as the main() has been used in prior assignments

II. Calls all functions needed in game

III. Creates all objects used by game (except the initial game object)

III. Game::match() - used to simulate a single match between players.

I. Calls player 1 rollDie function to get random value

II. Calls player 2 rollDie function to get random value

III. Compares values and calculates a winner for the round.

Die Class: Used to store info about a Die object and simulate the rolling of a die.

Variables:	long N	Functions:	Die()
			~Die()
			getN()
			setN()
			rollDie()

1) Die::getN() - getter for the N variable

A) Returns the N value to the calling object

2) Die::setN() - setter for the N variable

A) Sets the N variable

3) Die::rollDie() - simulates the player rolling a die

A) Returns a random number between 1 and N to the calling object

LoadedDie Class: Used to store info about a LoadedDie object and simulate the rolling of a loaded die.

Variables:	long count	Functions:	LoadedDie()
			~LoadedDie()

rollDie()

4) Die::rollDie() - simulates the player rolling a loaded die
   A) Generates a random number between 1 and N, adds the random number to N and then calculates the average. This is how the die gets it's "loaded" characteristic.
      I. If current round count is even, a number from the "loaded" die is returned
      II. If current round count is odd, a "regular" random number is returned.

Menu Function: The menu functions print out greetings, instructions and options to the user. The functions will get input from the user and return the appropriate bool to the main().

Validation Function: The validation function will receive input from the user, check that the input is the correct data type, check if the input is within the specified range, and if it fails these test, the user will be prompted to enter a valid input. The validation functions will return a value of the indicated type back to the main() after all the tests are passed.

Test Plan for integerValidation()

Test Case	Input Value	Expected Result	Observed Result
Char	A, c, +, .	"Please enter a valid integer."	"Please enter a valid integer."
String	Four, .five, yes	"Please enter a valid integer."	"Please enter a valid integer."
Float	6.31, 07.0, 8.8	"Please enter a valid integer."	"Please enter a valid integer."
< Min (set at 1)	0, -1, -5	"Please enter a valid integer."	"Please enter a valid integer."
> Max (set at 1000)	1001, 9999	"Please enter a valid integer."	"Please enter a valid integer."
Lower limit (min=1)	1	Function accept and return value to main()	Function accepted 5 and returned 5 to main()
Upper limit (max=1000)	1000	Function accept and return value to main()	Function accepted 200 and returned 200 to main()

Test Plan for War Games

Test Case	Description	Expected Result
Die Creation	Select a regular die for both players when setting up game.	A Die object should be created for both players and the print out during the game should indicate that both players are using a regular die.
LoadedDie Creation	Select a loaded die for both players when setting up game.	A LoadedDie object should be created for both players and the print out during the game should indicate that both players are using a loaded die.
Die and LoadedDie Creation	Select a loaded die for one player and a regular die for the other player when setting up the game.	A LoadedDie object should be created for one player and a Die object should be created for the other player. The print out during the game should indicate that one player is using a

		regular die and the other player is using a loaded die.
Menu 1	1. Choose option to quit 2. Choose option to play	Option to quit exited the program. Option to play should start the game and the player should begin to see prompts for game information.
Number of Rounds	1. Set the number of rounds for the game to 1. 2. Set the number of rounds for the game to 25000. 3. Set the number of rounds for the game to 100000.	Game should run without crashing for all of these cases. Minimum and Maximum should work.
Memory Leak	1. Start and run program using valgrind. Set number of rounds to 25000, player 1 has regular die with 10 sides, player 2 has loaded die with 20 sides. 2. Start and run program using valgrind. Set number of rounds to 10000, player 1 has loaded die with 30 sides, player 2 has regular die with 10 sides. 3. Start program and immediately quit.	There should be no memory leaks for any of these cases.
Die Size	1. Set player 1 die to 1 side and player 2 die to 1 side. 2. Set player 1 die to 50 sides and player 2 die to 50 sides. 3. Set player 1 die to 1000 and set player 2 die to 1000.	Program should function normal without crashing when number of sides for each player’s die is set to the min, max, or in between.
Loaded Die Size	1. Set player 1 loaded die to 1 side and player 2 loaded die to 1 side. 2. Set player 1 loaded die to 50 sides and player 2 loaded die to 50 sides. 3. Set player 1 loaded die to 1000 sides and player 2 loaded die to 1000 sides.	Program should function normal without crashing when number of sides for each player’s loaded die is set to the min, max, or in between.
Die vs LoadedDie (Outcome)	1. Player 1 has loaded die, Player 2 has regular die. Set number of rounds to 100 and make each player’s die have 10 sides.	If one player has a loaded die, and the other player has a regular die, the player with the loaded die should win more rounds on average (for a decent sized number of rounds).
Die vs Die Loaded vs Loaded (Outcome)	1. Player 1 has a regular die, Player 2 has a regular die. Set the number of rounds to 100 and make each player’s die have 10 sides. 2. Player 1 has a loaded die, Player 2 has a loaded die. Set the number of rounds to 100 and make each player’s die have 10 sides.	If both players have the same type of die (die vs die || loaded vs loaded) the game should be evenly matched and the winner should be random.
Winner Calculation	1. Set number of rounds to 100. Each player has the same type of die.	For each round won, the player’s overall win count should be incremented. After all the rounds have been played, the player with the

	1. Set number of rounds to 100. One player has a regular die and the other player has a loaded die.	highest win count should be announced as the winner.
Screen Printing (round)	1. Number of rounds is 1000 2. Player 1 has a loaded die 3. Player 2 has a regular die 4. Player 1 die has 6 sides 5. Player 2 die has 8 sides	First, the current round number should print. Then the program should print each player's type of die, the number of sides on their die, and the number they rolled. Last, the winner of the current round should print.
Screen Printing (final result)	1. Number of rounds is 1000 2. Player 1 has a loaded die 3. Player 2 has a regular die 4. Player 1 die has 6 sides 5. Player 2 die has 8 sides	Total number of wins for each player (and ties) should be correctly calculated and sum to the total number of rounds (1000). The player with the highest number of wins should be announced as the overall winner.

Reflection:

When I first sat down to begin the planning process for this assignment, I did not know where to begin. I thought and pondered for a couple hours, but nothing came. Frustration then set in and I knew I was not going to get anything done while I was in that state of mind. That was the first coding assignment that has left me so stressed that I had to take a break for a day to gather my thoughts. Luckily, I returned to the drawing board a day later and the ideas began to flow. I believe I was so lost, because I have never implemented an entire program inside a class. I did not know how to begin if I wasn't able to use the main function for implementation. Once I understood that a function inside a class can behave just like the main function, I was good to go. I believe having these types of "breakthroughs" is the best way to learn. Now that I have been working on this assignment and Project 2 for a week, implementing everything inside a class is no different than using the main(). For me, it's one of those situations where I can't believe I struggled so much while learning how to do it.

After that initial problem I had getting started, the assignment seemed very straightforward. I got my Die and LoadedDie classes setup. I used the input validation function and the menu function I had already created in the first project (this was very helpful). Everything came together quicker than I expected. There really weren't any major issues in the later stages. There were a few 'rookie' mistakes that were made like incrementing the round counter after the return statements were already issued and wondering why the counter isn't working, but nothing major.

I guess last obstacle I experienced was writing this reflection. I have never excelled at writing reflections, and it makes it even more difficult when I don't have much to talk about. It takes me hours to write these reflections! I can code an assignment in 5 or so hours, and then the reflection takes just as long. I guess documenting code is going to be a big part of any developer job, so I am glad it is required to write these reflections for the assignments. It is forcing me to become more comfortable with writing reflections and more proficient at documenting my code.

Changes:

One change I made from the original design was making the int variables in my Game class long int variables. This allowed me to accommodate large numbers of rounds for the war games. I have the max number of rounds set to 100000, so a long int was a must.

Not a lot of changes were made from the original. After stressing and planning for the first couple days, I didn’t find much to improve upon. I really only changed the way information printed out on the screen and such. I added a round counter to make it easier for my TA to tell if the correct number of rounds is being printed, and I printed out the total number of ties at the end as well. I think it looks good and runs without errors.

This assignment came in really handy while working on the Zoo project. There are many parts that are setup in the same manner and I am not stressing nearly as much while doing that project. I learn so many new things in this class each week that makes the next assignment so much easier. I’m sure I will continue to struggle at times, but I will make it through to the end. Bring on the next struggle.