Dice Wars Biased Dice Project Plan

Problems

Must design and solve for:

- Menu function containing available options to select from – 1 to start 2 to exit
- Input validation functions for each user input – input validation class?
    - User inputs needed for:
        - Choice to play or exit the game
        - Number of rounds to play
        - Type of die
            - Normal or Loaded
        - Sides for each die
- Two-player game in which each player can select their type of die
    - How the game changes when player selects loaded die
        - Bias the number to beat the other player's score
- Create a Die class with a default constructor for 6 sides on a normal die
    - Overloaded constructor for user input number of sides
    - Return function for returning the number of sides on the die
    - Roll dice function to generate a random number between 1 and the number of sides on the die – input by the player
        - Return the random number value
    - Get type of die function that returns the type of die used by the player
        - 1. Loaded
        - 2. Normal
- Create a child class from the Die class called loaded die class
    - Default constructor contains 6 sides on die
    - Overloaded constructor to initialize loaded die as type of die
    - Call roll dice function from the parent Die class
        - Assign the value, change conditions, and make the number larger
        - Return the value from the loaded die class
- Make a game class to introduce the logic of the game
    - Default constructor with 3 rounds minimum and initialize players
    - Create limits for number of rounds in game
    - How to keep score?
    - Functions for
        - Menu
        - Number of rounds to play
        - Player inputs
        - Roll dice
        - Get scores – show winner
- In main function, seed the srand() function to time(0) as recommended on various message boards (learncpp.com/, cplusplus.com/, stackoverflow.com/)
- Include all of the necessary libraries to complete the program:
    - For random numbers must include – cstdlib and ctime

Design

- **Create Die class**
  - Default constructor initialize values of side number and type of die
  - Overloaded constructor passing the argument for the number of sides
    - Use "this->sides = sides" syntax
    - Default value of die is 2, equal to a 'Normal die'
  - Make a return function that will return the number of die sides the user enters
  - Make a return function that returns the random value generated by the rand() function between 1 – # of sides
    - rand() % sides + 1 to ensure random number generate does not exceed the number of sides on the die…source code inspired by Gaddis textbook and stackoverflow.com/
  - Make return function that returns the type of die the player selects
    - 1. Loaded
    - 2. Normal

- **Create a loadedDie class – child class of parent Die class**
  - Default constructor to initialize sides on die to 6
  - Overloaded constructor with inheritance from the Die class for the type of die
  - Make return function to return biased number for loaded die
    - Initialize a value to the rollDice function located in parent class
    - Set conditions to bias results of the random integer output
      - If number is equal to 1, return a higher number
      - If number is equal to 2, return a higher number
    - Return the value from the loadedDie class

- **Create a Game class**
  - Game class contains logic of how the game is run and played by both players
  - To store the values and scores of each player, use an array of at most 50 rounds for both players – 2D array (dynamic?)
  - Default constructor containing minimum number of rounds set to 3
    - Initialize player1 and player2 values
  - Make a menu function to display the available options during the game
    - Main menu – 1. Play 2. Exit
    - In-Game menu for die selection – 1. Normal 2.Loaded
  - Create function to ask for and validate input for the number of rounds the game will be played
  - Make function to get user inputs for the type of die to use and the number of sides for each die (loaded and/or normal die)
    - Validate all user-inputs with function calls to the isValidInput class passing the user-inputs as parameters
  - Include logic to create dynamic variables – array? Player1? Player2?
    - Use 'new' Die – for call to Die class if the user selects a normal die
    - Use 'new' loadedDie – for call to loadedDie class if user select loaded die

- Make a function to play the game
  - For loop to run until the number of user-inputted rounds is completed
  - Use array to store values from each roll for each player
    - roundScore[i][1] – for player 1
      - call function to roll dice from either parent of child class
    - roundScore[i][2] – for player 2
      - call function to roll dice from either parent of child class
- Make a function to get scores from the game
  - Initialize the player1 and player2 points to 0.
    - Also include no. of tie games when player1score = player2score
  - Use logic similar to how the game was played
    - If(roundScore[i][1] > roundScore[i][2]) – Add point to P1 score
    - If(roundScore[i][1] < roundScore[i][2]) – Add point to P2 score
    - If(roundScore[i][1] == roundScore[i][2]) – Add point to Tie score
  - Use if statement to deduce winner from the scores
    - If(player1Score > player2Score) – P1 is the winner
    - If(player1Score < player2Score) – P2 is the winner
    - Rare scenario for tie based on loadedDie class, but it is possible for a low number of rounds (i.e. 3-5 rounds)
      - Include a condition for tie games
      - If(player1Score == player2Score) – tie game
- Make a game summary function to display the results and outcomes of game
  - Show:
    - How many rounds were played
    - The type of die used by player 1
    - The type of die used by player 2
    - The number of sides on die used by player1
    - The number of sides on die used by player1
    - Total score for player1
    - Total score for player2
    - Number of tied games
    - Winner of the game

Test Cases	Input Values	Functions	Expected Outcome	Observed Outcome
Value too low	choice < 1, rounds < 3, loaded die & normal die < 1, sides < 3	Main() function for choice, getRounds() for rounds, getPlayerInputs() for type of die and number of ides on die…	Choice input validation class member function will signal unacceptable values and prompt for new values. Rounds input validation class member function will prompt user for valid rounds input between 3 – 50. Die type input validation class member function will take only values 1 or 2 for loaded and unloaded die. Sides number input validation class member function will only accept values b/w 3 – 12.	.choice < 1: Value is tossed back and the user is asked to reenter acceptable value rounds < 3: Value is tossed and will not continue until rounds input of 3 – 50 is entered loaded die & normal die only accepts values 1 or 2, and sides < 3: Value is tossed and user must reenter value between 3 – 12.
Value of 0	choice = 0, rounds =0, loaded die = 0 normal die = 0, sides =0	Main() function for choice, getRounds() for rounds, getPlayerInputs() for type of die and number of ides on die…	Class member functions from isValidInput for choice, rounds, type, and sides will signal to the user if values are out of range. Currently, the user may not enter 0 for any of these variables, so the functions should ask the user for acceptable values or ranged inputs	The program will repeat each input statement until the user enters valid inputs for choice, rounds, type of die, and the number of sides on each player's die. It completes this input validation check using the isValidInput class created to validate all user inputs
Value in accepted range	choice = 1 to play, or 2 to exit, rounds = 3 - 50, loaded die = 2 normal die = 1, sides = 3 - 12	Main() function for choice, getRounds() for rounds, getPlayerInputs() for type of die and number of sides on die…	IsValidInput class member functions do not flag any invalid inputs and the program runs as designed	The program runs and does not contain any logic errors
Value too high	choice = 3, rounds =76, loaded die = 4 normal die = 3, sides =15	Main() function for choice, getRounds() for rounds, getPlayerInputs() for type of die and number of ides on die…	Choice input validation class member function will signal unacceptable value and prompt for acceptable value. Rounds input validation class member function will prompt user for valid rounds input between 3 – 50 Die type input validation class member function will take only values 1 or 2 for loaded and unloaded die. Sides number input validation class member function will only accept values b/w 3 – 12	.choice > 2: Value is tossed back and the user is asked to reenter acceptable value rounds > 50: Value is tossed and will not continue until rounds input of 3 – 50 is entered loaded die & normal die only accepts values 1 or 2. User must enter 1 or 2 sides > 12: Value tossed & user reenters value 3 – 12.
Negative values	choice = -1, rounds = -3, loaded die = -4 normal die = -21, sides = -7	Main() function for choice, getRounds() for rounds, getPlayerInputs() for type of die and number of ides on die…	Same as if values entered were too high or too low All input validation class member functions will toss values until valid integers are entered within accepted ranges of values	All input validation class member functions are called to ensure acceptable values are inputted from user
Garbage value with characters	choice = %, rounds =;[w, loaded die = *7 normal die = [{], sides =#gf	Main() function for choice, getRounds() for rounds, getPlayerInputs() for type of die and number of ides on die…	Symbols and values that cause the program to crash will call each input validation member function. Since each isValidInput member function uses cin.fail(), the user must reenter valid inputs for each variable	All invalid characters or symbols signal the isValidInput class member functions and prompt the user to enter valid inputs for each variable

Reflection

This lab assignment introduced the concepts of inheritance that tested the ability to build a child class for loaded die from the parent die class. In my original project plan, I began making the Game class initially. I realized after getting the menu and each of the variables—rounds, die type, and sides—planned and coded within the Game class I was lacking the logic needed from the Die class. In response, I transitioned into a new plan after reading various questions and answers from the group threads. As a result, I modified my initial plans by planning and coding the Die class initially. In doing so I was able to set the logic of the die and how to return a randomized number. For this syntax I utilized the Gaddis textbook and various online resources to ensure the proper seed value and constraints of the rand() function not to exceed the number of user-inputted sides.

Additionally, this program introduced the concept of virtual functions. I found the syntax for the virtual function in chapter 15 of the Gaddis textbook. By planning and implementing a virtual function I was allowing the Die class member function for rolling the dice to be redefined, or overridden, in the child loaded Die class. I found this challenging to implement initially since the virtual function is only declared in the source header file, so I corrected for this error upon initial use. The loaded Die class syntax was aided by the Gaddis textbook and multiple online sources to solidify how the child class uses inheritance to redefine the functions from the parent class. The other challenge was to plan the logic for the biased dice necessary for becoming 'loaded' when the player uses them.

Although I planned several different methods to solve the logic to make the loaded Die class dice weighted, I avoided using the switch statement since I thought it would take up more lines of code. I started with a small line of source pseudocode to plan and implement a less verbose solution. I was able to initialize the random integer value from the parent class roll dice member function. If the random integer from the function rolled a value of one, then the child class redefinition of the roll dice function would return the number of sides. Alternately, if the value were two, then the child class function would add one and return the value. By using this solution to solve the biased die problem I was able to get near 100% expected results in the outputs of the game summary. The only issue I found when using this method over others was the tendency to have tie games in games of 3-5 rounds. Due to statistical sampling sizes with low sides numbers and a small number of rounds this makes sense. However, I did not have any trouble with the logic at low numbers. The game still performed even with low value inputs.

Lastly, this lab also gave me an opportunity to create a validation input class to ensure input values by the user. Along with the menu functions, I will be able to recycle code in future assignments using what I planned and implemented for this lab assignment. This lab inspired me to read more on the web about inheritance, how to implement virtual functions, and syntax source ideas for rand() function use, among other small seemingly minor details about parent and child class implementation.