Paul Min
07/10/18
CS-162 Summer 2018

Description: This is my design and reflection document for lab 3. In it I outline my approach to coding the die war game.

Design

Programming Style:

I will follow the coding style guidelines provided on Canvas. I will use comments to help document my code. I will check for memory leaks. I will make sure to de-allocate any dynamic memory that was allocated. I will validate input when necessary. I will make sure to submit all my files in a zip archive.

Game Class:

Menu

In my game class, I plan on implementing a displayStartingMenu function. The displayStartingMenu function will prompt the user to select between two choices: 1) Play Game or 2) Exit Game. I also plan on having a validateStartingMenuChoice function that will check to see if the user enters 1 or 2. If the user inputs anything other than 1 or 2, it will display “invalid input. Please enter 1 or 2”.

Rounds

I plan on having a promptNumRounds function that will also the user how many rounds the game should have. I will also have a validateNumRounds function that will check to see if the user enters an integer between 1 and 100. I plan on capping the rounds at a 100, because I believe that is a large enough sample size to test both loaded and normal die rolls.

Die Type

I will have a promptDiePlayer1 and promptDiePlayer2 functions to prompt what type of die each player should have. It will ask the user to enter either a ‘L’ for a loaded die or ‘N’ for a normal die. I plan on having a validateDieType function that will check to see if the user inputs either a ‘L’ or ‘l’ or ‘N’ or ‘n’. All other input is rejected and the user is prompted to enter a char ‘L’ or a char ‘N’.

Number of Sides

I will have two separate functions to prompt players 1 and 2 for the number of sides their die should have. The functions will have a cout statement stating “Player 1, please enter an integer number of sides your die will have:”. I plan on having a validateNumSides function that

will check to see if the input is at least 1 and at most 100. If the user input does not fall in that range, it will prompt the user stating “invalid input, please enter an integer between ‘1’ and ‘100’.

Detailed Round Results

I will have a roundResult function that accepts the following variables as parameters: p1Name, p2Name, p1RollValue, p2RollValue, nSidesP1, nSidesP2, p1Wins, p2Wins, ties, rounds. After each round, I will output what kind of die each player used, how many sides each die had, what each player rolled, and announce who won the round or if the round resulted in a tie. The detailed round results also kept track of how many wins each player had. If player1 rolls a higher number than player 2 then, player 1 wins the round and player 1’s wins are incremented. If player 2 rolls a higher number than player1, player 2 wins the round and player 2’s wins are incremented. If both players roll the same number, the round is a tie, and the tie variable is incremented. I make the beginning of each round noticeable by couting a line of asterisks.

Final Score and Winner of Game

After the loop goes through the number of rounds that are specified, I will announce in the game::play() function how many rounds player 1 won, how many rounds player 2 won. If player 1 has more wins than player 2, then I will cout that player 1 wins the game. Else if player 2 has more wins than player 1, then I will cout that player 2 wins the game. Else if player 1 and player 2 have the same number of wins, the game results in a tie.

Die Class

Data Members and Member Functions

For the die class, I plan on using protected data members. I will have the following: Int sides, int lastRoll, and unsigned seed as protected data members.

I will also include the following public member functions:

Die() which is a constructor that initializes lastRoll to 0, seed = time(0); and srand(seed);

Int Die::roll() which will generate a random number between 1 and the number of sides and return that number.

Void Die::setSides(int) which will set the Die’s side with the int argument that is passed into it. The int argument will come from the validateNumSides() function in the game class.

LoadedDie Class

Biased rolling

The LoadedDie class inherits the Die class. I plan on calling the Die::roll() function and incrementing it by 1 if the roll < sides.

Menu

Because project 1 required us to create reusuable menu function, I was able to largely reuse my menu. I tweaked a few of the menu prompts, but other than that they were nearly identical to the langton's ant menu.

Input Validation Functions

I will have the following input validation functions:

- validateStartingMenuChoice
    - This function will check to see if the user input is either a 1 or a 2.
    - If the input is anything other than a 1 or 2 it will prompt the user stating invalid input, and ask the user to choose between '1' or '2'.
    - Floating point values are accepted, but they will be truncated. The truncated values will be cleared before accepting new input.
- validateNumRounds
    - This function will check to see that the user enters an integer between '1' and '100'
    - If the input is not between 1 and 100 it will prompt the user stating invalid input, and ask the user to choose an integer 1 through 100.
    - Floating point values are accepted, but they will be truncated. The truncated values will be cleared before accepting new input.
- validateDieType
    - This function will check to see that the user enters a char that is either a 'L' or 'l' or 'N' or 'n'.
    - If the input is anything other than those values above, then the function will prompt the user stating "invalid input. Please enter a char 'L' or a char 'N'.
    - You will be able to enter ln, but the function will only accept the first character, and discard the second character.
- validateNumSides
    - This function will check to see that the user input is between the integer 1 and the integer 100.
    - If the input is not within 1-100, the function will prompt the user stating "invalid input. Please enter an integer between '1' and '100'.
    - Floating point values are accepted, but they will be truncated. The truncated values will be cleared before accepting new input.

Test table

Test Case	Input Values	Driver Function	Expected Outcomes	Observed Outcomes
Input too low	Input < 1	validateStartingMenuChoice()	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’
Input too high	Input > 2	validateStartingMenuChoice()	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’	Invalid input. Please choose between the integer ‘1’ or the integer ‘2’
Input in correct range	2	validateStartingMenuChoice()	The End.	The End.
Input in correct range	1	validateStartingMenuChoice()	Enter an integer for the number of rounds the game should have:	Enter an integer for the number of rounds the game should have:
Input too low	Input < 1	validateNumRounds()	Invalid input. Please enter an integer between ‘1’ and ‘100’	Invalid input. Please enter an integer between ‘1’ and ‘100’
Input too high	Input > 100	validateNumRounds()	Invalid input. Please enter an integer between ‘1’ and ‘100’	Invalid input. Please enter an integer between ‘1’ and ‘100’
Input in correct range	Input >= 1 && Input <= 100	validateNumRounds()	Prompt player 1 for what type of die they would like.	Prompt player 1 for what type of die they would like.
Input not a valid char	Input not ‘L’ or ‘l’ or ‘N’ or ‘n’	validateDieType()	Invalid input. Please enter a char ‘L’ or a char ‘N’	Invalid input. Please enter a char ‘L’ or a char ‘N’

Correct input	Input is a char ‘L’ or ‘l’ or ‘N’ or ‘n’	validateDieType()	Depending on which player is prompted and what die they choose the output would be: Player 1 die type is: N Player 1 die type is: L Player 2 die type is: N Player 2 die type is: L	Depending on which player is prompted and what die they choose the output would be: Player 1 die type is: N Player 1 die type is: L Player 2 die type is: N Player 2 die type is: L
Input too high or too low	Input > 100 or Input < 1	validateNumSides()	Invalid input. Please enter an integer between ‘1’ and ‘100’	Invalid input. Please enter an integer between ‘1’ and ‘100’
Input in correct range	Input >= 1 && Input <= 100 I.e. 50	validateNumSides() promptSidesPlayer1()	Player 1’s die has 50 sides	Player 1’s die has 50 sides
Input in correct range	Input >= 1 && Input <= 100 I.e. 60	validateNumSides() promptSidesPlayer2()	Player 2’s die has 60 sides Calls play() Displays detailed round results and final score and winner	Player 2’s die has 60 sides Calls play() Displays detailed round results and final score and winner

Reflection

In my original design, they way I was creating a biased roll was by calling the Die::roll() function from the LoadedDie::roll() function. After calling the Die::roll() function, I checked to see if the roll < sides. If the roll was less than the number of sides, I incremented the roll by 1. However, this design was not biased enough for a die with a great number of sides.

To make the LoadedDie::roll() function product a more biased result, I decided to increment the roll based off the difference between the number of sides and the result of the roll. For example, if Die:roll() resulted in a roll of 50 on a die that has a 100 sides, then it would increment the roll by 40. The roll would now be a 90 instead of a 50. That's an 80% increase which definitely makes the roll more biased. Had I not implemented this, a 50/100 roll would have only increased to 51, which is a 2% increase.

One problem that I encountered was with my input validation functions. I realized that if a user enters "ln" for the type of die player 1 should have, then the 'n' would be stuck in the buffer and automatically assign that as the type of die for player 2. To address this issue, I implemented cin.clear and cin.ignore even after valid input. Doing this cleared the buffer after every input.

Overall, this lab wasn't too bad because a lot of the menu functions had been pre-made due to creating reusable menu functions in project 1. I also implemented random placement for project 1, so I was familiar with generating random values for the die roll. Using inheritance, allowed me to make a LoadedDie class really quickly, because it already inherited the member variables and member functions from the Die class.