Paul Leone

CS 162

7/7/2019

Lab 3 Test Design and Reflection

Test Plan

displayMenu():

Test Case	Expected Outcome	Actual Outcome
Input letters	Validation loop requires re-entry of valid input	Input validation loop prints out statement requiring user to input an integer value
Input symbols	Validation loop requires re-entry of valid input	Input validation loop prints out statement requiring user to input an integer value
Input higher integer than listed	Validation loop requires re-entry of valid input	Input validation loop prints out statement requiring user to select from the options given
Input lower integer than listed	Validation loop requires re-entry of valid input	Input validation loop prints out statement requiring user to select from the options given
Empty integer	Validation loops requires user to enter a valid integer	Input validation loop prints out a statement requiring the user to not leave the input blank
Correct input (user selects option 1)	Moves on to play the War Game	Moves on to play the War Game and prompts the user for the number of rounds (Rounds() function)
Correct input (user selects option 2)	Quits the game	Quits the game

Rounds():

Test Case	Expected Outcome	Actual Outcome
Input letters	Validation loop requires re-entry of proper input	Validation loop prompts user to enter an integer
Input special characters	Validation loop requires re-entry of proper input	Validation loop prompts user to enter an integer
Input negative numbers	Validation loop requires re-entry of proper input	Validation loop prompts user to enter a positive integer
Empty integer	Validation loop requires re-entry of proper input	Validation loop prompts user to not leave answer blank
Correct input	Continues the program and moves on to the next function to play the game	Continues the program and moves on to the next function

		to find the number of sides each player's dice has. dieType()

dieType():

Test Case	Expected Outcome	Actual Outcome
Prompts user and displays 4 different options for the type of dice for each player	Prompts user and displays 4 different options of dice type for each player	Prompts user and asks for the type of dice for each player from the given options
Input letter	Validation loops prompts user to re-enter valid input	Prompts user to enter an integer value
Input special character	Validation loop prompts user to re-enter valid input	Prompts user to enter an integer value
Input negative number	Validation loop prompts user to re-enter valid input	Prompts user to enter an integer from the given options
Input number greater than the options given	Validation loop prompts user to re-enter valid input	Prompts user to enter an integer from the given options
Empty input	Validation loop prompts user to re-enter valid input	Prompts user to not leave the answer blank
Correct input	Continues the program to the next function to play the game	The integer is selected, and the players are assigned the type of dice. User is prompted for the number of sides on each of the player's dice.

Sides():

Test Case	Expected Outcome	Actual Outcome
User is prompted with the number of sides on the die for each player	User is prompted for player one's dice sides. After input player two's dice sides is prompted	User is prompted for player one's dice sides. After input player two's dice sides is prompted
Input letters	Validation loop is executed and asks for re-entry of proper input	Prompts user to enter an integer value
Input special characters	Validation loop is executed and asks for re-entry	Prompts user to enter an integer value
Input number less than 1	Validation loop is executed and asks for re-entry	Prompts user to enter an integer between 1-20
Input number greater than 20	Validation loop is executed and asks for re-entry	Prompts user to enter an integer between 1-20
Empty integer	Validation loop is executed and asks for re-entry	Prompts user that the answer cannot be left blank
Correct integer	Program takes in players' information and begins the game	Correct information is inputted. Program begins the game displays the results with the proper functions to execute.

game():

Test Case	Expected Outcome	Actual Outcome
Function takes the information taken in from the Sides() function and sets the number of sides of each die to each player	Information is collected from the Sides() function to assign the number of sides on each die	The number of sides of each die is assigned to each of the players

Score():

Test Case	Expected Outcome	Actual Outcome
Function takes count of the score of each player and increments depending on the number of rounds	Player is assigned a point each round if their roll was higher than the other players	Player is given a point and points are incremented if they won the round

roundWinner():

Test Case	Expected Outcome	Actual Outcome
Function displays the winner of each round according the roll of the players' die	Function determines the winner of each round	Round winner is determined and assigned depending on roll. Displays round winner

Winner():

Test Case	Expected Outcome	Actual Outcome
Function that displays the final winner of the game. Calculates the total points accumulated from each player	Displays the winner after accumulating the points of each player	Displays the winner after accumulating the points of each player. Displays a tie if no one won

Test Case	Expected Outcome	Actual Outcome
This is the function that displays the type of die each player has and the number of sides. It runs the game and accumulates the score of each player and displays the winner of each round and the final winner	Proper input is taken in and runs the game. Displays the type of dice each player has and the number of sides. Round winner and final winner is displayed	Displays the type of dice each player and the number of sides each dice has for the players. Displays the round winner and final winner

Pseudocode

In Main

Randomizer seed is set

Game object is created

Run the game menu for user to select option

Use a switch function to determine direction of the program

Case 1 plays the game

Use another switch to determine the die types and run the game

Case 1: Player 1 is regular die and player 2 is loaded die

Dynamically allocate objects with smart pointers

Set the players

Set the game

Run the game

Case 2: Player 1 is loaded die and player 2 is loaded die

Dynamically allocate object with smart pointers

Set the players

Set the game

Run the game

Case 3: Player 1 is loaded die and player 2 is regular die

Dynamically allocate object with smart pointers

Set the players

Set the game

Run the game

Case 4: Player 1 is regular die and player 2 is regular die

Dynamically allocate object with smart pointers

Set the players

Set the game

Run the game

Case 2: quits the game

displayMenu()

Validates the user input and prompts user accordingly if proper input was entered

Die class

Private member variable holds the score

Protected member variable holds the number of sides

Contains public member functions that initializes the private and protected member variables

Contains function that returns a random number from the roll of the dice

LoadedDie class

Private member variable holds the score

Contains a constructor and getter and setter functions for the private member variable

Contains a function to determine loaded number of the dice

Game class

Contains the player member variables that point to the Die class

Contains the member functions Rounds(), Sides(), and dieType() that validate user input and initialize the private member variables

Score()

Increments the scores of each player

Increments score for each round depending on the number of rounds inputted

game()

sets the number of sides on each of the players dice

roundWinner()

Determines the winner of each round by incrementing the points awarded depending on the player's roll

Returns the Simulate() function to display the results

Winner()

Determines the final winner of the game.

Adds up the total points awarded to each player to determine and display the winner

Simulate()

Displays the game data from user input

Displays the type of die each player has and the number of sides each dice has

Displays the round winner

Displays the final winner

Quit()

Exits the game

Reflection

The assignments are not getting easier; however, I am seeing myself learn more every day not only from the book, but just doing the assignments as well. This assignment was difficult, but not as difficult as the previous project. Watching the lectures on how to design your programs and the proper steps to take have made it less stressful for me when it comes to figuring out the functions and variables I need when it comes to designing the program.

The original design of my program contained a few more functions than they did for the final design. I had a getter and setter function for the Rounds() function to initialize the rounds member variable to that function to receive the number of rounds from the user. However, I was having difficulty retrieving the date as the program would hold on to the number of rounds inputted from the user. I couldn't figure out why the program wasn't running the way it should have, but I figured out that those functions weren't necessary for the program to run. I figured that by having just one function that returns the rounds variable would make it easier to read and debug. I did the same thing with the dieType() function to determine the type of dice each player would have. I was having the same problem, so I removed the setter and getter functions and initialized the member variable directly in that function being used. I did however keep the setter and getter functions for sides member variable as that function was working properly. I was still unable to figure out why the program wouldn't output the data with those functions and hoping you could provide some insight for me.

One thing I did try differently was the use of input validation. I should have created its own function, however I wanted to try implementing it within the functions that ask for the information. I found that doing that was more tedious than having its own function. Creating its own function is easier and cleaner than continuously writing it into each function.