Maliha Syed

CS162

Final Project – Quest to Save Middle-Earth

**Design**

1. Input Validation:
2. For integers with min and max
3. For string, only allows directions “n”, “s”, “e”, “w”
4. Menu:
5. Show the menu choices
6. Accept user choice after validation
7. Die:
8. Member variable number of sides
9. Member variable type, set to “normal”
10. Default constructor with sides set to 6
11. Destructor
12. Constructor with integer parameter
13. Get sides that returns the number of sides
14. Get type that returns normal
15. Roll dice that implements a random number generator from 1 to the number of sides of the dice
16. Loaded Die:
17. Inherits number of sides from Die
18. Member variable type, set to “loaded”
19. Default constructor inherited from Die
20. Destructor
21. Constructor for number of sides using base class function
22. Get type that overrides base class and returns “loaded”
23. Roll dice function that overrides base class definition, uses random generator to generate a number from 1 to half of the number of sides, then adds half the number of sides. E.g. a 6-sided loaded die will only yield numbers from 4-6
24. Player:
25. Member variable name
26. Member variable score
27. Default constructor
28. Destructor
29. Update score function that adds 1 to the player’s score
30. Get score getter that returns the player’s score
31. Game:
32. Member variables player1 and player2 from Player class
33. Member variable integer rounds
34. Member variables integers roll1 and roll2
35. Member variables pointers to Die die1 and die2
36. Default constructor that initializes the two Player objects
37. Destructor
38. Set rounds setter that initializes the rounds to a user input
39. Choose Dice that asks user what type of die they want
40. Choose sides that allows user to set the number of sides
41. Play game function should run the simulation by calling the menu functions, and calling the game helper functions in the correct order to set up each player’s dice, then call roll dice for a number of times equal to the rounds and display the results after each round. It should then display the final result.
42. Main:
43. Instantiate a Quest object
44. Start playing the game

**Test Plan**

|  |  |  |
| --- | --- | --- |
| Test Scope | Description | Expected Result |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Reflection**

The main changes I made to this program from the design:

* Decided against having a player name to keep the program more simple, but this could have been added fairly easily
* Added helper functions that displayed the rounds results and final result
* Choosing the sides of the dice was left in the play function, although in hindsight this would have been better as a helper function due to code repeat

Although the logic of this program was pretty simple, it was the first time I implemented inheritance using C++. I realized after some searching and discussing, that using virtual for the base functions that needed overriding worked in being able to access the Loaded Die’s specific functions like rollDice. I also explicitly added the override keyword to the functions in the derived class that were overriding the base class functions.

For the LoadedDie rollDice function, the random number generator actually worked the way I wanted but kind of accidently. I wanted the random generator to generate a range of numbers of half the number of sides, so 1-3 for a 6-sided die. Then take that number and add half the number of sides, so the range for the generator would be 4-6. But this wouldn’t work correctly for an odd sided die, however because I forgot that the modulo operator is higher than division in priority, I didn’t use parentheses. This actually resulted in a correct range for an odd sided dice, so a range of 4-7 for a 7-sided die.

This lab I think was good practice for project 2, and although I felt a little lost at the start, I feel more confident about how variables and functions should be coded/behave with parent and child classes.