David An

Lab3: Reflection

It was originally easier for me to imagine implementing this game without utilizing a separate Game class actually. I tweaked my inputValidation method that I’ve used in some previous assignments to work with this one, and so the only other classes that I needed to design (until I chose to write the Game class) were the Die and LoadedDie classes. I knew the only differences between the two die were the types of die they were, and their roll() methods. Since I knew this, it was easy to write the Die class and then have the LoadedDie class inherit its behavior and then rework the roll() method to return numbers only in the latter half of the die’s range. I then actually wrote the core of the program in the main method just to work out the logic and see how it would work. After testing, I began to write the Game class using much of the code that I originally had in the main method. I decided to divide the Game class into two methods: the menu() function, and the play() function. The menu function gathered the information, while the play function used that information to simulate the game and display the results. I originally had a plan to separate the play() function into two smaller functions – one to iterate through and display the info as the game went on, and another to display the final result of the game. However, for the sake of simplicity I decided not to do that and to just combine them into the one play() function.

One of the more notable changes I had to make to my original design was creating the die objects as pointers instead of just created them as instances like I had done in the past. This allowed more flexibility in creating them conditionally, and still being able to use their specific methods. However, one problem that I ran into was that when a user wanted to use a loaded die, the rolls would not take this into account and would continue to use the regular die’s roll method instead of the loaded die’s roll method. After doing some research online, I found that I needed to use the ‘virtual’ keyword when declaring Die’s roll method.

Test cases for binary options (1 or 2)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input Values | Functions | Expected | Observed |
| Input too low | Input < 1 | Main()  inputValidation()  Game() | re-prompt for new entry | re-prompt for new entry |
| Input in correct range | 1 <= Input <= 2 | ^ | inputValidation returns input back to variable | inputValidation returns input back to variable |
| Input extreme low | Input = 1 | ^ | inputValidation returns input back to variable | inputValidation returns input back to variable |
| Input extreme high | Input = 2 | ^ | inputValidation returns input back to variable | inputValidation returns input back to variable |
| Input too high | Input > 2 | ^ | Re-prompt for new entry | Re-prompt for new entry |
| Input is incorrect type | Input = String | ^ | Re-prompt for new entry | Re-prompt for new entry |

Test cases for wider range options

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input Values | Functions | Expected | Observed |
| Input too low | Input < 1 | Main()  inputValidation()  Game() | re-prompt for new entry | re-prompt for new entry |
| Input in correct range | 1 < Input < Max (depends on prompt) | ^ | inputValidation returns input back to variable | inputValidation returns input back to variable |
| Input extreme low | Input = 1 | ^ | inputValidation returns input back to variable | inputValidation returns input back to variable |
| Input extreme high | Input = Max | ^ | inputValidation returns input back to variable | inputValidation returns input back to variable |
| Input too high | Input > Max | ^ | Re-prompt for new entry | Re-prompt for new entry |
| Input is incorrect type | Input = String | ^ | Re-prompt for new entry | Re-prompt for new entry |