Monte Carlo Simulations – Assignment03

A Monte Carlo simulation is a mathematical technique used to estimate the possible outcome(s) of an uncertain event. The Monte Carlo simulation technique involves creating a model with a set of possible results and then determining multiple outcomes based on random input. Ultimately all of those outcomes are aggregated into results for evaluation.

If you have ever played the board game Monopoly you should have a sense of this. The game Monopoly is a model. Each game of Monopoly has a result (a winner). The result is based both on the player's strategies and random input (dice rolls). If you play one hundred games of Monopoly you can aggregate the results and observe things that are generally true like "the player that develops the most real-estate properties in the first three rounds tends to win."

The events we will simulate in this assignment are much more simple than a game of Monopoly! We will simulate rolling one or more dice of different kinds.

Instructions

1. Download assignment03.zip and put your assignment03 folder into your MIS2402 workspace.
2. Start by editing montecarlo1.html – page will simulate the roll of a single die.

Scenario 1 – Roll a single die

There are many kinds of dice. The most common die has six sides. But there are other kinds (you know this if you have ever seen the game Dungeons and Dragons being played!)

In our simulation we are going to allow for dice to have any number of sides. A roll of a six-sided die should return a random integer between 1 and 6. A roll of a twenty-sided die should return a random integer between 1 and 20. And so on.

In our simulated world, we can even have some nonsensical things, like a 1000-sided die or a 2-sided die (that would be a coin) or even a 1-sided die (I'm not sure what that would look like!)

But you can't have a zero-sided die, and you can't have a die with fractional or negative sides.

3. This first scenario is really very simple. There is no loop in this scenario.
4. In montecarlo1.html observe that a function called getRandomInt has been defined for you. This function will return a random number between 1 and x. So getRandomInt(3) might return 1, 2, or 3.
5. Bring in the function isNaturalNumber from a prior assignment.
6. Now edit the click event handler. If the number of sides specified by the user is not a natural number, put the text "Bad data. Try again." into the tag textDisplayed1. You can use a statement that looks like this:

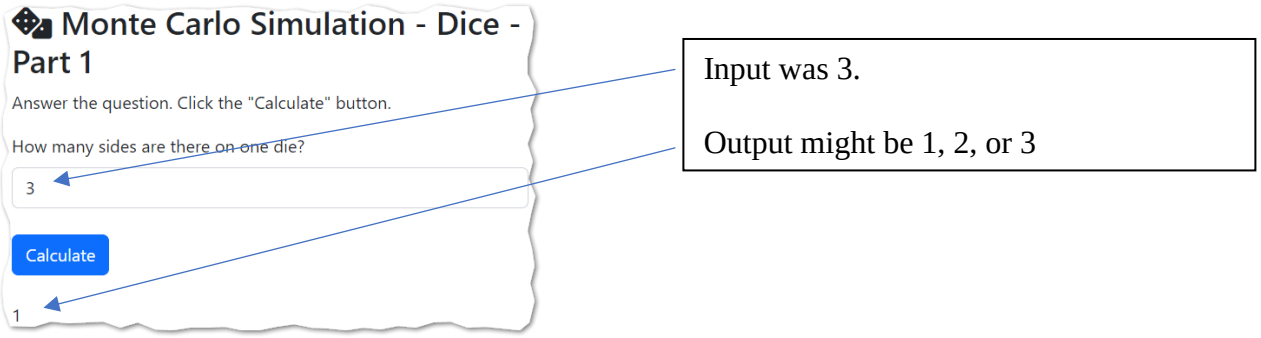
   ```
   $('#textDisplayed1').html("Bad data. Try again.");
   ```

7. If the number of sides specified by the user is a natural number, call the getRandomInt function and put the answer into the textDisplayed1 tag.

8. Notice that, in this scenario, all of the error trapping was inside the event handler. So, now you have seen that "error trapping" can occur in different parts of a solution. Either in the "event handler" portion of code, or in a function, or even a little bit in both places.

   In any assignment for this class, be sure to pay attention to where the instructions tell you to do the error trapping!

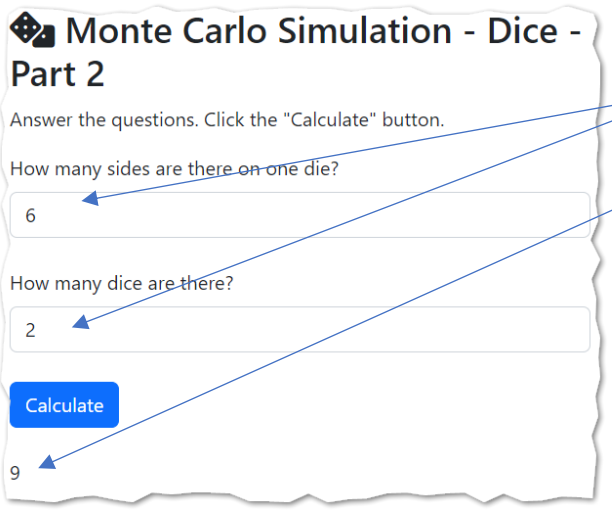
9. Test your work. You should be able to specify the number of sides and get one of the possible answers when you click calculate. If you don't provide good input, you should see "Bad data. Try again."



Scenario 2 – Roll multiple dice together

When you play Monopoly, you typically roll two dice at the same time. In other games, you might roll three or more dice, or in other games you might roll only one die. In this scenario we will extend the work we did in scenario one to simulate a roll of multiple dice.

10. Start by editing montecarlo2.html – this page will simulate the roll of 1 or more dice. You don't know in advance how many dice are in a roll, so you will need to write a loop in this scenario.
11. Before we get to the tricky part, bring in isNaturalNumber from scenario 1.
12. Complete the function getRoll. The getRoll function should contain a "**for" loop** that sums up 1 or more randomly generated integers. For example, rolling 2, six-sided dice should `return` a value between 2 and 12 (inclusive).
13. Complete the click event handler in montecarlo2.html (it will be very similar to what you did in scenario one!) Here's what the click event handler code should do…
    a. If `sides` is not a natural number, return "Bad data. Try again."
    b. If `dice` is not a natural number, return "Bad data. Try again."
    c. If `dice` and `sides` are both natural numbers, then call getRoll and put the result of getRoll into the textDisplayed1 tag.
14. Test your work.



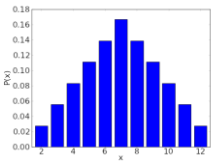
Input was 6 and 2.

Output might be 2 through 12 with 7 being the most frequently occurring outcome.

Test other scenarios as well!

Scenario 3 – Simulate multiple rolls

Now we will (at last) run a Monte Carlo simulation. In this scenario we will simulate multiple rolls, over and over again, and determine the average outcome.



(This is just an example you don't need to make this picture!)

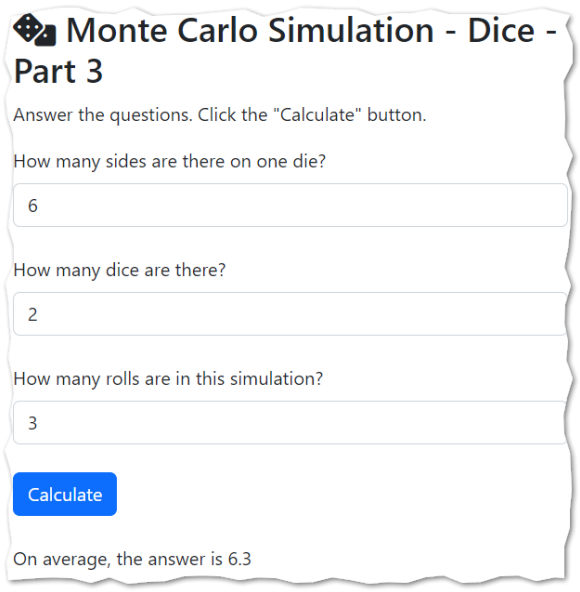
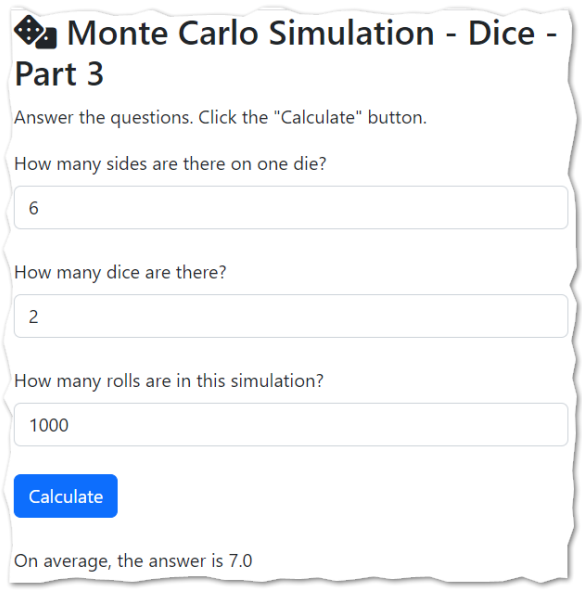
15. Start by editing montecarlo3.html.
16. Before we get to the tricky part, bring in isNaturalNumber and getRoll from scenarios 1 and 2.
17. Complete the function runSimulation. The runSimulation function should contain a "**for" loop** that sums up 1 or more results from the getRoll function. When the loop is done, use the sum to compute the average value of the roll, and return the average, rounded to 1 decimal place.
18. Complete the click event handler in montecarlo3.html (it will be very similar to what you did in scenario two!) Here's what the click event handler code should do…
    a. If `sides` is not a natural number, return "Bad data. Try again."
    b. If `dice` is not a natural number, return "Bad data. Try again."
    c. If `rolls` is not a natural number, return "Bad data. Try again."
    d. If `dice, sides` and `rolls` are all natural numbers, then call runSimulation and put the result into the textDisplayed1 tag with some extra text. Like this:

        "On average, the answer is 99.9"

        Of course, the value 99.9 is the number returned by the runSimulation function.
19. Test your work. You should notice that the more rolls you average, the closer your answer will get to the theoretical expected output.

    However, even with 1000 rolls, there's no guarantee that you will get exactly the expected result each time!

CONTINUED

Three rolls	1000 rolls
	

When you are done…

20. Be sure to upload both all three html files to assignment 3 on canvas.

How will this assignment be graded?

Scenario 1 work is worth 30 points. Scenario 1 needs to run. If it does not run at all you lose all 30 points.

Scenario 2 work is worth 30 points. Scenario 2 needs to run. If it does not run at all you lose all 30 points.

Scenario 3 work is worth 40 points. Scenario 3 needs to run. If it does not run at all you lose all 40 points.

Assuming your work runs – any other point deductions will be assigned in 5-point increments.

Things you could lose points for:

- Missing / incomplete error trapping
- Solving the problem without loops / or with unexpected language elements not taught in class
- Renaming functions
- Miscalculations
- Forgetting to round your answer in scenario 3
- Forgetting to put your answer in a sentence in scenario 3
- Any bug / error that causes the program return incorrect results