Simulation – Experiments and Trials

Learning Goals

- Use Monte Carlo methods to estimate the answer to a question

- Organize animated simulations to observe how systems evolve over time

Randomness

Random Functions

Most simulations use randomness in some way; otherwise, every run of the simulation will produce the same result.

Recall the random library, which we learned about early in the semester. This module included several useful functions we can use:

```
random.random() # pick a random float between 0-1

random.randint(x, y) # pick a random number in a range

random.choice(lst) # chooses an element randomly

random.shuffle(lst) # destructively shuffles the list
```

Computing Randomness

How is it possible for us to generate random numbers this way?

Randomness is difficult to define, either philosophically or mathematically. Here is a practical definition: given a truly random sequence, there is no gambling strategy possible that allows a winner in the long run.

But computers are deterministic – given an input, a function should always return the same output. Circuits should not behave differently at different points in time. So how does the random library work?

True Randomness

To implement truly random behavior, we can't use an algorithm. Instead, we must gather data from physical phenomena that can't be predicted.

Common examples are atmospheric noise, radioactive decay, or thermal noise from a transistor.

This kind of data is impossible to predict, but it's also slow and expensive to measure.

Pseudo-Randomness

Most programs instead use pseudo-random numbers for casual purposes. A pseudo-random number generator is an algorithm that produces numbers which look 'random enough'. Each number the algorithm generates acts as a starting place to generate the next one.

By calling the function repeatedly, the algorithm generates a sequence of numbers that appear to be random to the casual observer.

The number sequence generated by a pseudo-random number generator isn't truly random; if someone figures out the algorithm, they can predict the results. But it is random enough to use for casual purposes.

Monte Carlo Methods

Randomness in Simulation

Using randomness in a simulation means that the same simulation might have multiple different outcomes on the same input model. A single run of a simulation is not a good estimate of the true average outcome.

To find the truth in the randomness, we need to use probability!

Law of Large Numbers

The Law of Large Numbers states that if you perform an experiment multiple times, the average of the results will approach the expected value as the number of trials grows.

This law works for simulation as well! We can calculate the expected value of an event by simulating it a large number of times.

We call programs that repeat simulations this way Monte Carlo methods, after the famous gambling district in the French Riviera.

Monte Carlo Method Structure

If we put our simulation code in the function `runTrial()` and want to find the odds that a simulation 'succeeds', a Monte Carlo method might take the following format:

```
def getExpectedValue(numTrials):
    count = 0
    for trial in range(numTrials):
        result = runTrial() # run a new simulation
        if result == True:  # check the result
            count = count + 1
    return count / numTrials # return the probability
```

Monte Carlo Example

Every year, A school holds the Random Distance Race. The length of this race is determined by rolling two dice. What is the expected number of laps a runner will need to complete?

```
import random
def runTrial():
    return random.randint(1, 6) + random.randint(1, 6)

def getExpectedValue(numTrials):
    lapCount = 0
    for trial in range(numTrials):
        lapCount += runTrial()
    return lapCount / numTrials
```

Activity: Monte Carlo Methods

You do: what are the odds that a runner in the Random Distance Race will need to run 10 or more laps?

Write the code to run the trial. You can modify the code from the previous slide.