#12: Randomness and Monte Carlo Methods

SAMS SENIOR CS TRACK

Last Time

Use **side-scrolling** to recognize the difference between the model and the view.

Use a **time loop** to create animations in interactive graphics.

Today's Learning Goals

Understand how programs can compute **random** or **pseudo-random** numbers computationally

Recognize when **simulations** can be used to learn about systems

Use **Monte Carlo methods** to solve problems

Randomness

Deterministic or Not?

So far, we've mostly written **deterministic** algorithms- a given input will always produce the same output.

Some algorithms are non-deterministic, or **random**; the output may change.

Question: what are some examples of problems where we want an algorithm to include randomness?

Features of Randomness

It’s hard to directly define randomness, because… it’s random.

However, a truly random sequence should **not be predictable**. There can’t be any patterns.

Algorithms are all about patterns! How can a computer generate randomness?

Pseudo-Random Numbers

Python's built-in random library uses an algorithm (the Mersenne Twister) to generate a sequence of **psuedo-random numbers**.

These numbers are not truly random, but they are unpredictable enough for everyday use (like Tetris).

It still isn't safe to use most psuedo-random number generators for security purposes- if you can guess the algorithm, it's no longer random. The best way to generate truly random sequences is to gather **physical data** (like atmospheric noise or radioactive decay) that cannot be predicted.

Python's Random Library

The random library in Python is secretly deterministic- if you know where you are in the sequence, you can predict what will come next.

Use **random.seed()** to select a specific location in the sequence.

All methods in the random library start from **random.random()**, which generates a number from [0.0, 1.0)

Useful Random Functions

random.randint(a, b) - generates a random integer in [a, b]

random.choice(lst) - returns a random item in the iterable lst

random.shuffle(lst) - shuffles the sequences lst

Activity - how would you implement flipping a coin, rolling a die, and drawing a card?

Simulation

Simulation

Simulation is the process of modeling an **event** to determine how it turns out. It’s based on two parts:

- A **model** that describes the state of the world
- **Rules** that describe how the world changes over time

Our time-based animations are simulations!

Simulation in Many Disciplines

Simulations are used in many different fields to study different subjects.

You can find some great examples on NetLogo:

- Fireworks
- Ants
- Infection Rates
- Waves
- Fire
- Traffic

Law of Large Numbers

One simulation is just an example.

But the more simulations you run, the closer their average will be to the true expected value.

Therefore, by running lots of simulations and averaging them, we can **simulate the truth**.

Monte Carlo Methods

Monte Carlo Methods

A **Monte Carlo method** is an algorithm that uses a series of simulations to approximate the answer to a problem.

It's gambling with the accuracy of its answer!

Fun fact: a **Las Vegas** algorithm also uses randomness, but gambles with runtime, not accuracy.

Monte Carlo Algorithm Template

```
def simulate(trials):
   count = 0
   for trial in range(trials): # Run a large number of trials
       result = runTrial() # simulate a random event each time
       if result == True: # count the number of successful events
           count += 1
   return count / trials # calculate the observed outcome
```

Example: Prime Poker Hand

What are the odds that five cards drawn from a deck will sum to a prime number? Can we write a program to compute this?

Let's use Monte Carlo methods to solve this!

Time vs. Accuracy

The more trials we run, the more accurate our estimate will be.

But the more trials we run, the longer our code will take to run!

Determining how many trials to use is always a balance between these two factors.

Example: Random Distance Race

Every year, CMU holds the Random Distance Race. At the start of the race, one giant die is thrown, and racers must run that many laps. As soon as the first racer finishes those laps, a second die is thrown, and everyone must run that many more laps.

Question: how often will someone need to run 10 or more laps at the Random Distance Race?

Today's Learning Goals

Understand how programs can compute **random** or **pseudo-random** numbers computationally

Recognize when **simulations** can be used to learn about systems

Use **Monte Carlo methods** to solve problems