Reinforcement & Applications

Andey Robins & Dr Borowczak

April 20, 2023

Outline

- Monty Hall Simulations
- Giant Rat Breeding
- Fermi Paradox
- The Next Steps

Today's Plan

We'll be exploring applications and uses of python!

Impractical Python Projects

Projects

1. Monty Hall Problem
2. Breed Bulldog sized Rats
3. Model the Fermi Paradox

Monty Hall Problem

Monty Hall Problem

A showcase of the Monty Hall problem

1. Contestants are shown three doors
2. The contestant chooses a door
3. One of the unselected doors is opened to reveal a goat
4. The contestant can switch doors or keep their selection
5. If they end up opening the door with the car, they win!

Optimal Strategy

1. Choose a door
2. After a door is revealed, switch your door
3. Win 2/3 of the time.

The Controversy

A popular "Ask column" provided the correct solution (i.e. switch) as an answer in 1990, but was met with major push back from the general public.

We'll experimentally prove the answer by simulating many many instances of the problem.

Simulation

```
for i in range(1000):
  doors = setup_doors()
  choice = choose_door()
  revealed_door = reveal_door(doors, choice)

  if switch_wins(doors, revealed_door, choice):
    switch_win_count += 1

  if stay_wins(doors, revealed_door, choice):
    stay_win_count += 1
```

```python
def setup_doors():
  car_idx = random.randint(0, 2)
  doors = ["goat", "goat", "goat"]
  doors[car_idx] = "car"
  return doors
```

```
def choose_door():
  return random.randint(0, 2)
```

Why did we put this in a function?

```python
def reveal_door(doors, choice):
  if choice == 0:
    if doors[1] == "car":
      return 2
    else:
      return 1
  elif choice == 1:
    if doors[0] == "car":
      return 2
    else:
      return 0
  else: # choice == 2
    if doors[0] == "car":
      return 1
    else:
      return 0
```

```python
def switch_wins(doors, revealed, choice):
  doors_idxs = [0, 1, 2]
  if revealed > choice:
    doors_idxs.pop(revealed)
    doors_idxs.pop(choice)
  else:
    doors_idxs.pop(choice)
    doors_idxs.pop(revealed)
  return doors[doors_idxs[0]] == "car"
```

```python
def stay_wins(doors, revealed, choice):
  return doors[choice] == "car"
```

Simulation Results

```
$ python3 monte.py
Welcome to the Monty Hall Solution Simulator
The game will be run 1000 times for staying and switching.
Beginning simulation...
Finished simulation. Results to follow.
The win rate of staying is 0.334
The win rate of switching is 0.666
```

Monty Hall Math

A visual demonstration of the answer to the Monty hall problem

Giant Rats

Genetically Breeding Giant Rats

I feel like a mad scientist today, and I've decided that I'd like to create a race of super-mutant, giant rats. To see if it's even feasible, I'd first like to mode the outcomes in python to determine if we can actually do this (because that's the only challenge we need to overcome in this scenario).

Strategy

The Rat Breeding Strategy

Information about the brown rat

Parameters for our program

Simulation

```python
def main():
  # create initial population
  generations = 0
  parents = populate(NUM_RATS, INITIAL_MIN_WT,
                     INITIAL_MAX_WT, INITIAL_MODE_WT)
  pop_fitness = fitness(parents, GOAL)

  # create new generations
  while pop_fitness < 1 and generations < GENERATION_LIMIT
    selected_rats = select(parents, NUM_RATS)
    children = breed(selected_rats, LITTER_SIZE)
    children = mutate(children, MUTATE_ODDS,
                      MUTATE_MIN, MUTATE_MAX)
    parents = selected_rats + children
    pop_fitness = fitness(parents, GOAL)
    generations += 1
```

```
def populate(num_rats, min_wt, max_wt, mode_wt):
  rats = []
  while num_rats > 0:
    rats.append(random.triangular(min_wt, max_wt, mode_wt))
    num_rats -= 1
  return rats
```

```python
def fitness(population, goal):
  avg = statistics.mean(population)
  return avg / goal
```

```python
def select(population, num_to_retain):
  sorted_population = sorted(population)
  return sorted_population[-num_to_retain:]
```

```
def breed(rats, litter_size):
  small_pop = rats[int(len(rats)/2):]
  random.shuffle(small_pop)
  large_pop = rats[:int(len(rats)/2)]
  random.shuffle(large_pop)

  children = []
  for rat1, rat2 in zip(small_pop, large_pop):
    for child in range(litter_size):
      if rat1 < rat2:
        child = random.randint(int(rat1), int(rat2))
      else:
        child = random.randint(int(rat2), int(rat1))
      children.append(child)

  return children
```

```python
def mutate(children, mutate_odds, mutate_min, mutate_max):
  for idx, rat in enumerate(children):
    if mutate_odds >= random.random():
      children[idx] = round(rat *
                  random.uniform(mutate_min, mutate_max))
  return children
```

Simulation Results

```
...
Generation 355 fitness = 0.8798
Generation 356 fitness = 0.8829
Generation 357 fitness = 0.8878
Generation 358 fitness = 0.9170
Generation 359 fitness = 0.9730
Generation 360 fitness = 1.0119

number of generations = 361
number of years = 36
```

Fermi Paradox

Fermi Paradox

Since the Universe is $\sim$13 billion years old, and even a modestly space faring civilization could explore the entire milky way galaxy in $\sim$10 million years. The radio bubbles of these groups would likely be larger than their explored space, so: *where are all the aliens?*

A spiral galaxy similar to the Milky Way

Drake Equation

A modeling of the drake equation

Constant Values

Fermi Constant Estimates

Calculating the Drake Equation

```
R_STAR = 3
F_P = 1
N_E = 0.2
F_L = 0.13
F_I = 1
F_C = 0.2
L = 10 ** 9

def drake_estimation():
  return R_STAR * F_P * N_E * F_L * F_I * F_C * L

print(int(drake_estimation()))
```

`15,600,000`

Using these estimates, they place an approximate estimation for the number of detectable civilizations at ~15.6 million.

Note: detectable in this context means that they could be detected in the MWG not they could be detected from Earth

Radio Bubbles

Assuming that an alien civilization would only be leaking incidental radio waves (i.e. not broadcasting their existence into the universe), their presence would be defined by a "radio bubble."

This is a "bubble" approximately 200 LY across centered around the civilization's home planet.

We will model the galaxy as a collection of radio cubes, a very loose estimate of a radio bubble.

Simulation

```python
def main():
  cube_count = radio_cubes_in_galaxy(RADIO_BUBBLE_RADIUS)
  civilization_count = drake_estimation()
  locations = []

  # simulation to follow
```

```python
def drake_estimation():
  return int(R_STAR * F_P * N_E * F_E * F_I * F_C * L)
```

```
def radio_cubes_in_galaxy(radio_bubble_radius):
  galaxy_volume = 3.14 * 50_000 * 50_000 * 1_000
  bubble_volume = 3.14 * (4/3) * radio_bubble_radius
  return galaxy_volume / bubble_volume
```

```python
for i in tqdm(range(civilization_count)):
  locations.append(random.randint(1, cube_count))

counts = {}
for val in tqdm(locations):
  if val in counts:
    counts[val] += 1
  else:
    counts[val] = 1

detected_civs = 0
for val in tqdm(counts.values()):
  if val > 1:
    detected_civs += val
```

Simulation Results

```
Populating galaxy
100%|==========================| 15600000/15600000
Counting civs per cube
100%|==========================| 15600000/15600000
Counting overlapping civs
100%|==========================| 15593457/15593457
The number of civs who could have
  detected each other = 13084
This is a rate of 0.0839%
```

In other words, the chance of a civilization being in a position to detect another is 1 in 1750.

Questions

The Next Steps

This has been a great start to your computer science journey, but how do you move forward from here?

- Studying/Practice
- Courses
- Learning new Languages

How to Practice

We've mentioned before, but just writing code is often the best way to practice! Fortunately, there are a lot of great places you can go to practice programming problems at various levels!

kattis.com

codingame.com

projecteuler.net

adventofcode.com

Courses

UWyo COSC Courses

How to Learn a New Language

1. Learn the basic, atomic pieces
2. Solve some basic problems
3. Learn the tools
4. Build a medium project and “release” it
5. Read about and implement best practices

```cpp
#include <iostream>
int main() {
  int max = 0;
  bool show_text = false;
  int my_nums = { 0, 2, 3, 50, 1};

  if (show_text)
    std::cout << "The array: ";

  // ....
```

```
  // ....
  for (auto num: my_nums) {
    if (num > max)
      max = num;
    if (show_text)
      std::cout << num << " ";
  }
  if (show_text) {
    std::cout << std::endl << "Max number: "
              << max << std::endl;
  }
}
```

Tools

Instead of the interpreter `python3` we use the compiler `gcc`.

Instead of pip, we use "header files."

Instead of `import` we say `include`.

Releasing Something

2030 Bloodsugar project

Best Practices

The C Book

Questions