The Snake Problem

Letian Xu

Monte Carlo Algorithms

By Prof. Massimo DiPierro

DePaul University

2018 Spring

1. Problem

Do you know the video game “snake”? Implement a simulator for the game. The snake is a one dimensional object of length 30cm. It moves at a speed of 1cm/second. It lives in a cage of 1000cm x 1000cm. The snake can only move horitontally or vertically. At random intervals (1 every 5 seconds in average) the snake can turn left or right. If the snake intersect itself, the snake bites itself and dies. The snake starts moving at the center of the cage. I bet $1000 that the snake will bite itself before it reaches the end of the cage. Is this a good bet? Explain. (optional: can you visualize it with tkinter)

Write a short paper (10 pages), explaining the problem, your solution, answers to the above questions with explanations, and an appendix containing the code (commented and indented).

2. Solution

Basically, this problem is trying to figure out what is the chance that this snake can reach the edge of cage without bites itself. Based on the bet, we know that higher the chance, worse the bet. Since we can get the probability of the snake reaches the end of the cage before it bites itself, we can simply use the expectation value to explain whether is a good bet or not.

2.1 Expectation value:

$E[x] = \sum p(x) \cdot f(x)$

$= p(\text{bites itself}) \times 1000 - p(\text{reaches the edge}) \times 1000$

As long as the expectation value is greater than 0, we can consider this bet was a good bet because the chance that snake bites itself is higher than snake reaches the edge. Then in order to get the probability that the snake reaches the end of the cage, we need to do the Monte Carlo simulation.

3. Direction Object

Before we create the snake object, I found out that I need to specified directions for the snake object. That will help us to use direction object to separate moving directions and turning directions later.

```
class Direction():
    def __init__(self, move):
        self.move = move
        self.directions = []

up = Direction(lambda x,y: (x, y+1))
down = Direction(lambda x,y: (x, y-1))
left = Direction(lambda x,y: (x-1, y))
right = Direction(lambda x,y: (x+1, y))

up.directions.extend([left, right])
down.directions.extend([right, left])
left.directions.extend([down, up])
right.directions.extend([up, down])
```

The four direction objects are using lambda to tell the snake how to move on the coordinate axis. For example, when the snake is going up, the only coordinate will change is the value of y, and the value of x remains same until change direction. Then when the snake facing the truning point, the snake has two relative directions for every moving direction.

I use extend function to add two relative two directions in the end of self.direction list. So, the current snake object can always know what directions it can choose when the snake facing turnning point. We can move to create the body of snake.

4. Snake Object

```
class snake():
    def __init__(self):
        self.body = [(i,0) for i in range(30)]
        self.direction = left
        self.dead = False
        self.completed = False
    def move(self):
        self.body.pop()
        newHeader = self.direction.move(*self.body[0])
        self.isDead(newHeader)
        self.body.insert(0, newHeader)
        if not self.dead:
            self.isCompleted(newHeader)
    def isDead(self, newHeader):
        if newHeader in self.body:
            self.dead = True
    def isCompleted(self, newHeader):
        if any(val<=-500 or val>=500 for val in newHeader):
            self.completed = True
    def changeDir(self, dir):
        self.direction = self.direction.directions[dir]
```

In the snake class, we use a python list to create body for this snake, the python list is including (0, 0), (1, 0) ...... (28, 0), (29, 0). Every dot in the list is one part of the snake, dot (0, 0) is the beginning snake head, and dot (29, 0) is the snake tail. If the snake is on the moving direction, the direction object will using lambda to take a new dot and then input it to the snake's body. Also, the pop function will delete the current tail of the snake's body to maitain the length of the snake, which is 30.

```
def isDead(self, newHeader):
    if newHeader in self.body:
        self.dead = True
def isCompleted(self, newHeader):
    if any(val<=-500 or val>=500 for val in newHeader):
        self.completed = True
```

For us to reach the answer of the question, we need to set a binary value to specify whether the snake bites itself or it successfully escaped. The actual situation of the snake bites itself means the new snake head input a new dot that existed in the body list.So, when snake is moving, it will check whether the snake bites itself or not. If the newheader in the body list, then the self.dead will return True.

Also, When the newheader can reach the edge, which means x or y <= -500, or, x or y >=500, the game completes, and the self.completed returns True.

```
def changeDir(self, dir):
    self.direction = self.direction.directions[dir]
```

Since we already did direction object above, we can use it to change the current direction to relative right or left direction for the snake. In the directions list, there are two options, or relatvie directions to choose, for different moving directions, and we will use python provided random function to choose changing direction below.

5. Simulation

After imitating the snake and controling the direction, at this step, we need some random number to tell the snake when it can change the direction. Then we need to do Monte Carlo simulation.

```
def simulate():
    currSnake = snake()
    while True:
        gap = int(ceil(expovariate(0.2)))
        for i in range(gap):
            currSnake.move()
            if currSnake.dead:
                return 0
            elif currSnake.completed:
                return 1

        currSnake.changeDir(choice([0, 1]))
```

In the while loop of the simulation, we need a gap between two turning points, and the snake will keep following the moving direction in this gap distance until change direction. Since this is a random time interval problem, then we can use the exponential distribution and exponential probability formula to input random number as different gap distances.

From the exponential probability formula:

$$p(x) = \lambda \cdot e^{-\lambda x}$$

Cumulative Density Function will be:

$$F(x) = 1 - e^{-\lambda x}$$

So, we can get the inverted formula to get an exponential random value. This will be every gap distance that the snake move until next change direction(If we take F(x) as u):

$$x = -\frac{1}{\lambda} \cdot \ln u$$

Since the problem gives us that "At random intervals (1 every 5 seconds in average) the snake can turn left or right", we can find out that λ = 0.2. Based on the exponential random value we got, we can know the time interval, which means how many centimeters that a snake moves on its speed until next change direction.

At the same time, we also check whether the snake bites itself or reaches the edge, and if any one of them happened return 0 or 1. Then when gap ends and no bite or reach happened, the snake will turn its direction thourgh random choice. It will give each postion's direction in the self.directions list. For example, if the snake is moving up, when it needs to make a turn, it has 50 percent possibility go left and 50 percent possibility go right. Because we already set up two relative turnning directions for every moving direction, the changeDir will let the snack randomly choose the relative two turnning options for it.

This simulate function will keep repeating moving forward and changing direction until the snake bites itself or reaches the edge.

5.1 Simulation Many

Theoretically, the snake do have the chance for reaching the edge before it bites itself, although the chance will be very small. If we imagine how the snake escape, for total distance is 500, and average gap is 5, the snake needs to make average 100 times changing direction, and every two same turning direction will make it harder to reach the edge. The most ideal way for the snake reach the edge is that the snake will make every changing direction are different with its last changing direction. So, it will be $\frac{1}{2^{100}}$ for the ideal chance.

However, since we have the simulation technique, we can use the simulate_many to simulate a giant number of testing times.

```
def simulate_many(time):
    total = 0
    for i in range(time):
        total += simulate()
    return total / time
```

We can use this function to control the total times of test, and be able to get the probability of that snake reaching the edge before it bites itself.

```
#### simulate 100000 times

print(simulate_many(100000))
```

```
0.0
```

Then, for 100000 times simulation, the probability that the snake reaches the edge is 0.

Based on that, we can get the expectation value:

$$p(x) = \sum p(x) \cdot f(x)$$

$$= 100\% \times 1000 - 0\% \times 1000$$

$$= 1000$$

6. Conclusion

It's a very good bet because you literally have zero chance to lose.

7. Full Code:

```
#### import modules
from random import *
from math import ceil

class Direction():
    def __init__(self, move):
        self.move = move
        self.directions = []

#### moving directions
up = Direction(lambda x,y: (x, y+1))
down = Direction(lambda x,y: (x, y-1))
left = Direction(lambda x,y: (x-1, y))
right = Direction(lambda x,y: (x+1, y))

#### relative directions
up.directions.extend([left, right])
down.directions.extend([right, left])
left.directions.extend([down, up])
right.directions.extend([up, down])

#### define the snake object
class snake():
    def __init__(self):
        self.body = [(i,0) for i in range(30)]
        self.direction = left
        self.dead = False
        self.completed = False
    def move(self):
        self.body.pop()
        newHeader = self.direction.move(*self.body[0])
        self.isDead(newHeader)
        self.body.insert(0, newHeader)
        if not self.dead:
            self.isCompleted(newHeader)
    def isDead(self, head):
        if head in self.body:
            self.dead = True
    def isCompleted(self, head):
        if any(val<=-500 or val>=500 for val in head):
            self.completed = True
    def changeDir(self, dir):
        self.direction = self.direction.directions[dir]
```

```
#### simulate once
def simulate():
    currSnake = snake()
    while True:
        gap = int(ceil(expovariate(0.2)))
        for i in range(gap):
            currSnake.move()
            if currSnake.dead:
                return 0
            elif currSnake.completed:
                return 1

        currSnake.changeDir(choice([0, 1]))

#### simulate many
def simulate_many(time):
    total = 0
    for i in range(time):
        total += simulate()
    return total / time
```