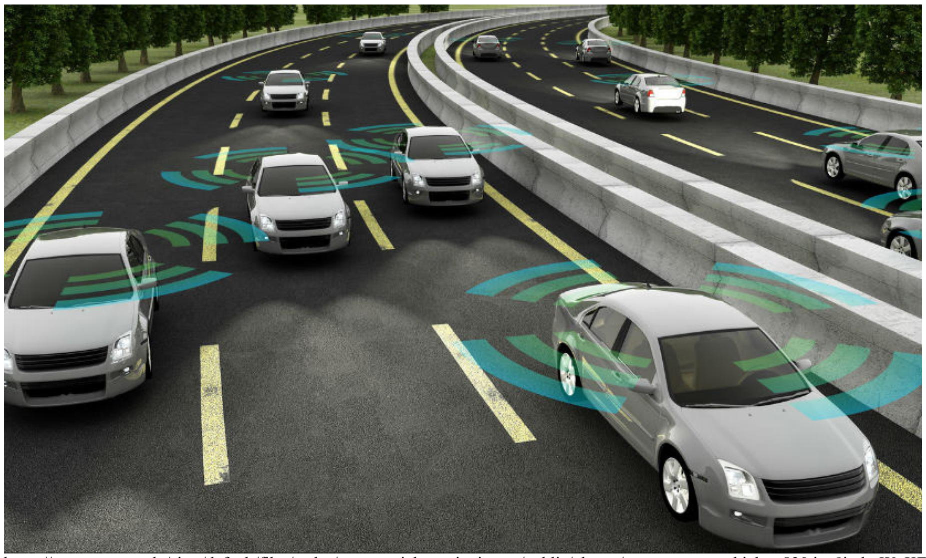
CSCI561 Fall 2018 Foundations of Artificial Intelligence

Homework 3 – CORRECTED, CLARIFIED, AND DEADLINE EXTENDED

Due December 7, 2018 23:59:59
BONUS: 20 points if submitted by November 30, 2018 23:59:59
EXTRA BONUS: 40 points if submitted by November 23, 2018 23:59:59


https://news.utexas.edu/sites/default/files/styles/news_article_main_image/public/photos/autonomous_vehicles_830.jpg?itok=WvKF9fpF

Problem Description:

You are the CTO of a new startup company, SpeedRacer, and you want your autonomous cars to navigate throughout the city of Los Angeles. The cars can move North, South, East, or West. The city can be represented in a grid, as below:

0,0	1,0	2,0	3,0	4,0
0,1	1,1	2,1	3,1	4,1
0,2	1,2	2,2	3,2	4,2
0,3	1,3	2,3	3,3	4,3
0,4	1,4	2,4	3,4	4,4

There will be some obstacles, such as buildings, road closings, etc. If a car crashes into a building or road closure, SpeedRacer has to pay $100. You know the locations of these, and they will not change over time. You also spend $1 for gas each time you move. The cars will start from a given SpeedRacer parking lot, and will end at another parking lot. When you arrive at your destination parking lot, you will receive $100. **Your goal is to make the most money**[1] over time with the greatest likelihood. Your cars have a faulty turning mechanism, so they have a chance of going in a direction other than the one suggested by your model. They will go in the correct direction 70% of the time (10% in each other direction, including along borders).

The first part of your task is to design an algorithm that determines where your cars should try to go in each city grid location given your goal of making the most money. Then, to make sure that this is a good algorithm when you present it to the rest of your board, you should simulate the car moving through the city grid. To do this, you will use your policy from your start location. You will then check to see if the car went in the correct direction using a random number generator with specific seeds to make sure you can reproduce your output. You will simulate your car moving through the city grid **10 times** using the random seeds 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. You will report the mean over these 10 simulations **as an integer after using the floor operation** (e.g., numpy.floor(meanResult)). An example of this process is given in detail below.

Input: The file `input.txt` in the current directory of your program will be formatted as follows:

First line: strictly positive 32-bit integer s, size of grid [grid is a square of size sxs]

Second line: strictly positive 32-bit integer n, number of cars

Third line: strictly positive 32-bit integer o, number of obstacles

Next o lines: 32-bit integer x, 32-bit integer y, denoting the location of obstacles

Next n lines: 32-bit integer x, 32-bit integer y, denoting the start location of each car

Next n lines: 32-bit integer x, 32-bit integer y, denoting the terminal location of each car

Output:

n lines: 32-bit integer, denoting the mean money earned in simulation for each car, **integer result of floor operation**

[1]Note that although we would like to make the most money possible, you MUST use the parameters we provide to make sure everyone gets the same value. Other parameters may give better results, but you should not use them. You SHOULD use an approach we learned in class to make sure you get the same value.

Example:

Input.txt
```
3
1
1
0,1
2,0
0,0
```

Output.txt
```
95
```

For example, say you have a 3x3 grid, as follows, with 1 car in start position <u>2,0</u> (green):

99	-1	🟩 -1
-101	-1	-1
-1	-1	-1

<u>You determine that based on the locations of certain obstacles, you should move in these directions in each cell:</u>

99	←	🟩 ←
↑	↑	↑
→	↑	↑

Then, you should do simulation using this policy. Beginning at the start position, move in the direction suggested by your start policy. There is a 10% chance that you will move South, so check your direction using random generation with random seed = 0 (see code below). In this case, you actually move West, so you will receive -$1. You will now try to move West again based on your policy. With the random seed = 1, you accidentally move South. Therefore, you now have -$2. Repeat at your next locations, until you end at your terminal state. Record the total money you have at the end. Let's say that the total is $91. Then, repeat this process 9 more times. You will average $91 with the 9 other results, and report the number. If the result is 91.65093, for example, you should record the floor, 91, in your output file. <u>Here are the detailed steps for getting a value of 95 for the above input.txt, where the iteration of simulation is given before several ~'s, and the moves to reach the terminal state are provided followed by the value gained, and the total resulting score:</u>

```
0~~~~~~~~~~~~~~~~~~~~~~~~~
MOVE: (-1, 0)   VALUE GAINED: -1.0     SWERVE VALUE: 0.5488135039273248
MOVE: (0, 1)    VALUE GAINED: -1.0     SWERVE VALUE: 0.7151893663724195
MOVE: (0, -1)   VALUE GAINED: -1.0     SWERVE VALUE: 0.6027633760716439
MOVE: (-1, 0)   VALUE GAINED: 99.0     SWERVE VALUE: 0.5448831829968969
SCORE: [96.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
1~~~~~~~~~~~~~~~~~~~~~~~~~
MOVE: (-1, 0)   VALUE GAINED: -1.0     SWERVE VALUE: 0.417022004702574
MOVE: (0, 1)    VALUE GAINED: -1.0     SWERVE VALUE: 0.7203244934421581
MOVE: (0, -1)   VALUE GAINED: -1.0     SWERVE VALUE: 0.00011437481734488664
MOVE: (-1, 0)   VALUE GAINED: 99.0     SWERVE VALUE: 0.30233257263183977
SCORE: [96.0, 96.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
2~~~~~~~~~~~~~~~~~~~~~~~~~
MOVE: (-1, 0)   VALUE GAINED: -1.0     SWERVE VALUE: 0.43599490214200376
MOVE: (-1, 0)   VALUE GAINED: 99.0     SWERVE VALUE: 0.025926231827891333
SCORE: [96.0, 96.0, 98.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
3~~~~~~~~~~~~~~~~~~~~~~~~~
MOVE: (-1, 0)   VALUE GAINED: -1.0     SWERVE VALUE: 0.5507979025745755
MOVE: (0, 1)    VALUE GAINED: -1.0     SWERVE VALUE: 0.7081478226181048
MOVE: (0, -1)   VALUE GAINED: -1.0     SWERVE VALUE: 0.2909047389129443
MOVE: (-1, 0)   VALUE GAINED: 99.0     SWERVE VALUE: 0.510827605197663
SCORE: [96.0, 96.0, 98.0, 96.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
4~~~~~~~~~~~~~~~~~~~~~~~~~
MOVE: (1, 0)    VALUE GAINED: -1.0     SWERVE VALUE: 0.9670298390136767
MOVE: (-1, 0)   VALUE GAINED: -1.0     SWERVE VALUE: 0.5472322491757223
MOVE: (1, 0)    VALUE GAINED: -1.0     SWERVE VALUE: 0.9726843599648843
MOVE: (0, 1)    VALUE GAINED: -1.0     SWERVE VALUE: 0.7148159936743647
MOVE: (0, -1)   VALUE GAINED: -1.0     SWERVE VALUE: 0.6977288245972708
MOVE: (-1, 0)   VALUE GAINED: -1.0     SWERVE VALUE: 0.21608949558037638
MOVE: (1, 0)    VALUE GAINED: -1.0     SWERVE VALUE: 0.9762744547762418
MOVE: (-1, 0)   VALUE GAINED: -1.0     SWERVE VALUE: 0.006230255204589863
MOVE: (-1, 0)   VALUE GAINED: 99.0     SWERVE VALUE: 0.25298236238344396
SCORE: [96.0, 96.0, 98.0, 96.0, 91.0, 0.0, 0.0, 0.0, 0.0, 0.0]
5~~~~~~~~~~~~~~~~~~~~~~~~~
MOVE: (-1, 0)   VALUE GAINED: -1.0     SWERVE VALUE: 0.22199317108973948
MOVE: (0, -1)   VALUE GAINED: -1.0     SWERVE VALUE: 0.8707323061773764
MOVE: (-1, 0)   VALUE GAINED: 99.0     SWERVE VALUE: 0.20671915533942642
SCORE: [96.0, 96.0, 98.0, 96.0, 91.0, 97.0, 0.0, 0.0, 0.0, 0.0]
6~~~~~~~~~~~~~~~~~~~~~~~~~
MOVE: (0, -1)   VALUE GAINED: -1.0     SWERVE VALUE: 0.8928601514360016
MOVE: (-1, 0)   VALUE GAINED: -1.0     SWERVE VALUE: 0.3319798053011772
MOVE: (0, -1)   VALUE GAINED: -1.0     SWERVE VALUE: 0.8212291230578318
MOVE: (-1, 0)   VALUE GAINED: 99.0     SWERVE VALUE: 0.0416966257252499
SCORE: [96.0, 96.0, 98.0, 96.0, 91.0, 97.0, 96.0, 0.0, 0.0, 0.0]
7~~~~~~~~~~~~~~~~~~~~~~~~~
MOVE: (-1, 0)   VALUE GAINED: -1.0     SWERVE VALUE: 0.07630828937395717
MOVE: (0, 1)    VALUE GAINED: -1.0     SWERVE VALUE: 0.7799187922401146
MOVE: (0, -1)   VALUE GAINED: -1.0     SWERVE VALUE: 0.4384092314408935
MOVE: (0, 1)    VALUE GAINED: -1.0     SWERVE VALUE: 0.7234651778309412
MOVE: (0, 1)    VALUE GAINED: -1.0     SWERVE VALUE: 0.9779895119966027
MOVE: (0, -1)   VALUE GAINED: -1.0     SWERVE VALUE: 0.5384958704104337
MOVE: (0, -1)   VALUE GAINED: -1.0     SWERVE VALUE: 0.5011204636599379
MOVE: (-1, 0)   VALUE GAINED: 99.0     SWERVE VALUE: 0.07205113335976154
SCORE: [96.0, 96.0, 98.0, 96.0, 91.0, 97.0, 96.0, 92.0, 0.0, 0.0]
8~~~~~~~~~~~~~~~~~~~~~~~~~
MOVE: (0, -1)   VALUE GAINED: -1.0     SWERVE VALUE: 0.8734294027918162
MOVE: (1, 0)    VALUE GAINED: -1.0     SWERVE VALUE: 0.968540662820932
MOVE: (0, -1)   VALUE GAINED: -1.0     SWERVE VALUE: 0.86919454021392
MOVE: (-1, 0)   VALUE GAINED: -1.0     SWERVE VALUE: 0.530855691555599
MOVE: (-1, 0)   VALUE GAINED: 99.0     SWERVE VALUE: 0.2327283279772907
SCORE: [96.0, 96.0, 98.0, 96.0, 91.0, 97.0, 96.0, 92.0, 95.0, 0.0]
9~~~~~~~~~~~~~~~~~~~~~~~~~
MOVE: (-1, 0)   VALUE GAINED: -1.0     SWERVE VALUE: 0.010374153885699955
MOVE: (-1, 0)   VALUE GAINED: 99.0     SWERVE VALUE: 0.5018745921487388
SCORE: [96.0, 96.0, 98.0, 96.0, 91.0, 97.0, 96.0, 92.0, 95.0, 98.0]
AVERAGE SCORE: 95.5
FLOORED AVERAGE SCORE: 95.0
```

Code demonstrating how to determine the direction your car turns during simulation is given below. <u>You should use this directly without edits.</u>

```
orientations = NORTH, SOUTH, EAST, WEST = [(1, 0), (0, -1), (-1, 0),
(0, 1)]
turns = LEFT, RIGHT = (+1, -1)

def turn_heading(heading, inc, headings=orientations):
    return headings[(headings.index(heading) + inc) % len(headings)]

def turn_right(heading):
    return turn_heading(heading, RIGHT)

def turn_left(heading):
    return turn_heading(heading, LEFT)

for i in range(len(cars)):
    for j in range(10):
        pos = cars[i]
        numpy.random.seed(j)
        swerve = numpy.random.random_sample(1000000)
        k=0
        while pos != ends[i]:
            move = policies[i][pos]
            if swerve[k] > 0.7:
                if swerve[k] > 0.8:
                    if swerve[k] > 0.9:
                        move = turn_right(turn_right(move))
                    else:
                        move = turn_right(move)
                else:
                    move = turn_left(move)
            k += 1
```

Guidelines

This is a programming assignment. You are provided sample input and output files. Please understand that the goal of these samples is to check that you can correctly parse the problem definitions, and generate a correctly formatted output. The samples are very simple and it should not be assumed that if your program works on the samples it will work on all test cases. There will be more complex test cases and it is ***your task to make sure that your program will work correctly on any valid input***. You are encouraged to try your own test cases to check how your program would behave in some complex special case that you might think of. Since **each homework is checked via an automated A.I. script**, your output should match the specified format *exactly*. Failure to do so will most certainly cost points. The output format is simple and examples are provided. You should upload and test your code on vocareum.com, and you will submit it there.

Grading

Your code will be tested as follows: Your program must not require any command-line argument. It should read a text file called "input.txt" in the current directory that contains a problem definition. It should write a file "output.txt" with your solution to the same current directory. Format for input.txt and output.txt is specified below. End-of-line character is LF (since Vocareum is a Unix system and follows the Unix convention).

The grading A.I. script will

- Create an input.txt file, delete any old output.txt file.
- Run your code.
- Test your output.txt file

Academic Honesty and Integrity

All homework material is checked vigorously for dishonesty using several methods. All detected violations of academic honesty are forwarded to the Office of Student Judicial Affairs. To be safe you are urged to err on the side of caution. Do not copy work from another student or off the web. Keep in mind that sanctions for dishonesty are reflected in *your permanent record* and can negatively impact your future success. As a general guide:

- **Do not copy** code or written material from another student. Even single lines of code should not be copied.
  Do not collaborate on this assignment. The assignment is to be solved individually.
  Do not copy code off the web. This is easier to detect than you may think.
- **Do not share** any custom test cases you may create to check your program's behavior in more complex scenarios than the simplistic ones considered below.
  Do not copy code from past students. We keep copies of past work to check for this.
- **Do** ask the professor or TA if you are unsure about whether certain actions constitute dishonesty. It is better to be safe than sorry.

Homework Rules

1. Use Python 2.7 to implement your homework assignment. You are allowed to use standard libraries only. You have to implement any other functions or methods by yourself.

2. Create a file named "hw3cs561f2018.py". When you submit the homework on labs.vocareum.com, the following commands will be executed:

   ```
   python hw3cs561f2018.py
   ```

3. Create a file named "output.txt" and print its output there. For each test case, the grading script will put an "input.txt" file in your work folder, runs your program

(which reads "input.txt"), and check the "output.txt" file generated by your code. The grading script will replace the files automatically, so you do NOT need to do anything for that part.

4. Homework must be submitted through Vocareum. Please only upload your code to the "/work" directory. **Don't create any subfolder or upload any other files**. Please refer to http://help.vocareum.com/article/30-getting-started-students to get started with Vocareum.

5. Your program must handle all test cases within a maximum runtime of **3 minutes** per test case on Vocareum.

6. It is recommended to submit your program 24 hours ahead of the deadline to avoid any submission issues on Vocareum. **Late submissions will not be graded.**

Helpful Hints:

1. **Tie breaking.** If values are the same for your available moves, choose to move in directions in this order of preference: North, South, East, West.
2. **Cars can be in the same grid cell at the same time.** There is no need to consider multi-agent coordination.
3. **Calculating expected value.** When considering future moves, make sure to give them a lesser weight, specifically 0.9. You can stop calculating based on an error factor of 0.1. (HINT: These are parameters for your algorithm.)
4. **What if a car ends up turning "off grid" during simulation?** The car should just remain in the current cell, but will spend another $1 on gas.
5. **You may use numpy** (and should use numpy where we do in the simulation).
    a. During our computation of the policy, we use numpy.float64 precision with the following parameters (you can see yours using numpy.finfo):
        i. precision = 15
        ii. resolution = 1.0000000000000001e-15
        iii. machep = -52
        iv. eps = 2.2204460492503131e-16
        v. negep = -53
        vi. epsneg = 1.1102230246251565e-16
        vii. minexp = -1022
        viii. tiny = 2.2250738585072014e-308
        ix. maxexp = 1024
        x. max = 1.7976931348623157e+308
        xi. nexp = 11
        xii. min = -max
    b. During simulation, we use numpy.float64 precision with the following parameters (you can see yours using numpy.finfo(type(swerve[k]))):
        i. precision = 15
        ii. resolution = 1.0000000000000001e-15
        iii. machep = -52

iv. eps = 2.2204460492503131e-16
v. negep = -53
vi. epsneg = 1.1102230246251565e-16
vii. minexp = -1022
viii. tiny = 2.2250738585072014e-308
ix. maxexp = 1024
x. max = 1.7976931348623157e+308
xi. nexp = 11
xii. min = -max

6. **Vocareum has a memory limit.** It is 4194304 KB.
7. **What if …?**
   a. The start position is surrounded by obstacles? This will not happen.
   b. There is more than one obstacle in a cell? This will not happen.
   c. What if the start/end location has an obstacle? This will not happen.
8. What is the value of the start state? All we have specified is that making a move is what results in the -$1 gas cost.
9. Is crashing or hitting an obstacle a terminal state? No.
10. **<u>It is very important to use the exact parameters we provide for your simulation, as different answers will be marked as incorrect.</u>**
11. **We will not give unsolvable inputs.** This means that we won't give any irregular inputs that don't conform to the format we've described in this document.
12. **Think about representing the problem.**
   a. What is a good representation of states and operators?
   b. How can you use this to simplify the problem representation?
   c. How will you evaluate the "score" of a state?
13. **Think about complexity.**
   a. How can you use the input parameters to determine which algorithm will be able to generate a solution within 3 minutes?