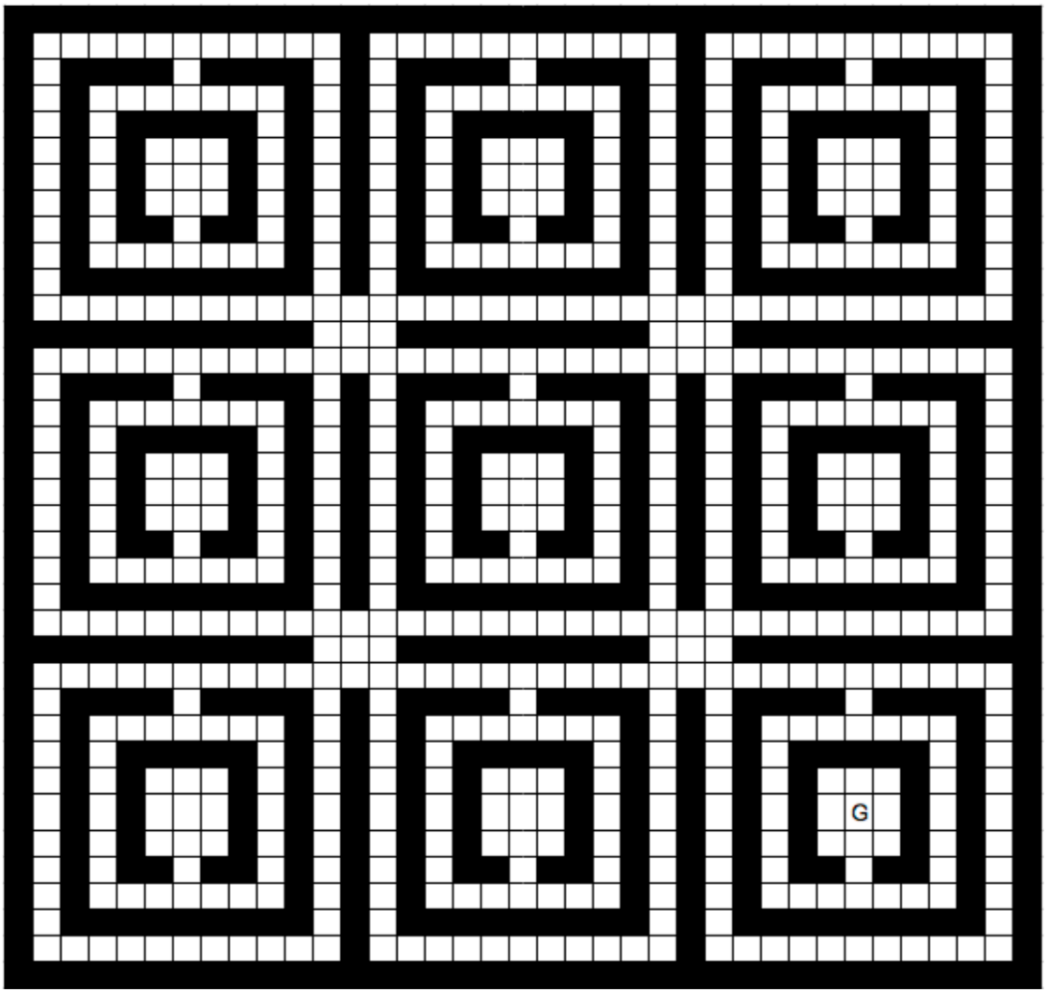

Localization

CS 520 Final Question 1

Spring 2019

Xuenan Wang

NET ID: xw336

LOCALIZATION

a) You are somewhere in the above maze, with no prior knowledge. What is the probability you are at G?

```
[🍀Xuenan👉~/Documents/GitHub/CS520_Final/question 1🧐: python localization.py    ]
map:
 [['1' '1' '1' ... '1' '1' '1']
 ['1' '0' '0' ... '0' '0' '1']
 ['1' '0' '1' ... '1' '0' '1']
 ...
 ['1' '0' '1' ... '1' '0' '1']
 ['1' '0' '0' ... '0' '0' '1']
 ['1' '1' '1' ... '1' '1' '1']]
shape of map: (37, 37)
error: G
There are 694 zeros and 674 ones in map.
There are 1 errors.
```

Figure 1. Running result of loading data file.

I loaded the given data file(Maze.txt) and counted the number of zeros and ones, there are one exception, which is a 'G'. Therefore, total white cells number should be 694+1, which is 695. With no prior knowledge, the probability of being at G is $\frac{1}{695}$.

b) Argue that there is a finite sequence of moves that without knowing where you start, and without any feedback on your moves will result in you ending at location G with complete probability/certainty. Hint: What if you knew you started either at the top left or bottom right corner, but didn't know which?

We consider the whole map as separate 9 parts. Each part is shown as Figure 2.

Since we have no idea about whether or not each move is successful, we can not design an algorithm for any specific positions, which means that the algorithm we need should treat every cell equally and we have to assume that every cell is possible. However, there's a pattern in this map. Firstly, this map can be seen as 9 equal parts basically. There are 3 components in each part. Yellow part is **part A**, blue part is **part B** and green part is **part C**, and the remaining white part, we temporally call it **part D**.

It's easy to observe that if the starting point is located in **part A**, then all we need is to make the bot keep going down and adjust to left or right and continue going down, then the bot must be in point α shown in Figure 2. For **part B**, it's also easy to show that all we need is to make the bot go lest or right and then continue going up, after that, the bot will be located in left/right upper corner of part B. Then we make bot go right/left 3 cells and go up 2 cells. After these, the bot will be in position β in Figure 2.

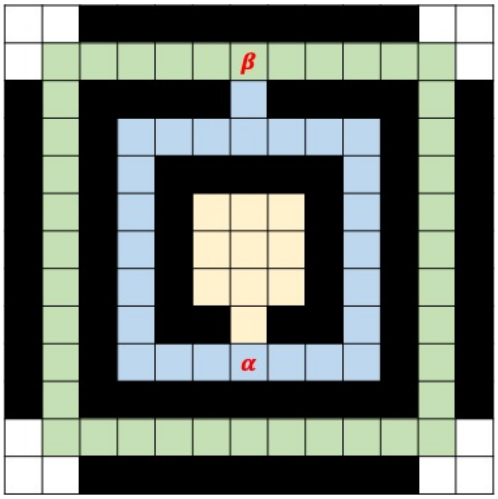

Figure 2. One part of the map.

Next, we need to make the bot be out of the single part shown in Figure 2. There is a problem: exit of each one of the total nine parts is different with each others, which means that we can no longer using the same idea to deal with this problem. We have to think the bigger picture. As shown in Figure 3, target point G is located in the bottom right corner part(part 9) and we have to make sure the bot at least end up in that part. So the basic directions are going right and going down. Before this, we have managed to make sure the bot is in β points now. Then let's assume that the bot is located in part 4,5,6,7,8 and 9, which means that the exit of part C is in the upper right corner. We make bot keep going right and then keep going down. The bot will be in the positions of red dots in Figure 4-A. Then we make bot go up exactly 10 cells and keep going right. The bot will be in the positions of red dots in Figure 4-B. After this move, we successfully reduced the possible positions of bot to 3. We continue make bot go left exactly 10 cells and keep going down. After this, there is only one possible position of bot, as shown in Figure 4-C, which means wherever the starting point is, we make the bot be fixed in this point and we now can navigate this bot go to target position G with known move.

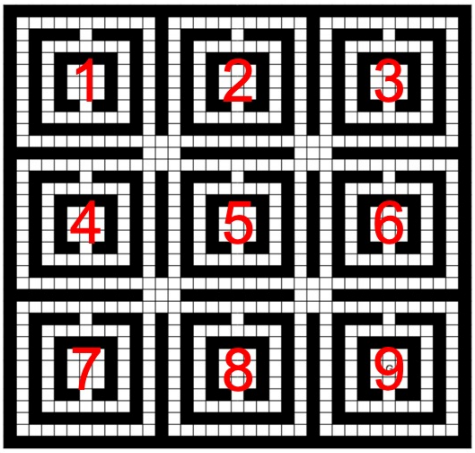

Figure 3. Nine sub-parts of the map.

Therefore, obvious there is a finite sequence of moves that without any information of starting point and feedback of movement will result ending in target G. We basically have draw this movement above. Of course, we still have to prove this.

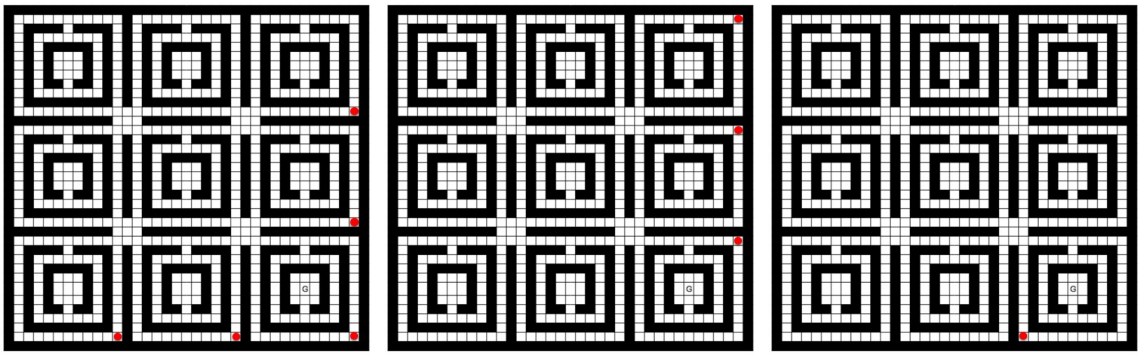

Figure 4. Possible positions of bot in map(A: left, B: middle, C: right).

We assumed the starting point is in part A above. Actually, there's no difference if the starting point is not in part A. Because the only possible situation that this sequence of moves is not working is that the bot goes into part A or part B when we want it to going out. But the only way of going into part A is going upward, when the bot is in part B, he will only go down and then the sequence will guide it go out of part B. Same thing happens when starting point is in part C and D. Therefore, wherever the starting point is, this sequence can always guide bot go the exact same point, which is shown in Figure 4-C, and from there, every movement is crystal clear to us.

c) How could you find such a sequence? How could a computer find such a sequence?

We basically have found such a sequence above. For computer, the key idea is trying to use the walls in maze. For this question, the only difference between going in to an empty cell and a wall is that your position will change if that is an empty cell. Hence, we can use walls to stop bot from going into somewhere we don't want it to be and in this case, we make bot waiting by continually going into wall to wait other possibility goes away.

d) Write an algorithm to find the shortest sequence of moves you can to reach G independent of where you begin and without feedback. Describe your algorithm in detail, including any design choices you made. What is the sequence of moves?

As explained above, I will write this algorithm in four steps.

STEP 1: Down*4+Left*1+Down*4+Right*1+Down*4

STEP 2: Right*3+Up*6+Left*3+Up*2

STEP 3: Right*30+Down*24+Up*10+Right*24+Left*10+Down*35

STEP 4: Up*10+Right*5+Down*2+Right*3+Down*6+Left*3+Up*3

Explanation: After STEP 1, the bot will be in position α in Figure 2; after STEP 2, the bot will be in position β in Figure 2; after STEP 3, the bot will be in positions of red dots in Figure 4-C and after STEP 4, the bot will reach the target position G.

e) Suppose that after each move, you receive an observation / feedback of the form Y_t = the number of blocked cells surrounding your location. Let Y_0 be the number of blocked cells surrounding your starting location. Again, you get no feedback if the move was successful or not, simply the number of blocked cells surrounding your current location.

e.1) You initially observe that you are surrounded by 5 blocked cells. You attempt to move LEFT. You are surrounded by 5 blocked cells. You attempt to move LEFT. You are surrounded by 5 blocked cells. Indicate, for each cell, the final probability of you being in that cell.

There are 211 cells that has an observation of Yt = 5. For after 3 times moving LEFT, there are 128 cells satisfy the condition. These are shown as Figure 5.

```
ctr_5: 211 len(coordinate_array): 422
Number of satisfied points: 128.0
```

Figure 5. Running result of e.1.

e.2) Write an algorithm to take a sequence of observations $\{Y_0,Y_1,\ldots,Y_n\}$ and a sequence of actions $\{A_0,A_1,\ldots,A_{n-1}\}$ and returns the cell you are most likely to be in.

Assume we have a sequence of observations {5, 2, 5, 5, 5, 6} and a sequence of actions {Down, Down, Right, Right, Right}. We search the whole map, and found 9 satisfied points. Results are showed as Figure 6. Target cells and moves are shown in Figure 7.

```
init_array: [ 1.  6.  1. 18.  1. 30. 13.  6. 13. 18. 13. 30. 25.  6. 25. 18. 25. 30.]
target_array: [ 3.  9.  3. 21.  3. 33. 15.  9. 15. 21. 15. 33. 27.  9. 27. 21. 27. 33.]
number of target points: 9.0
```

Figure 6. Running result of e.2.

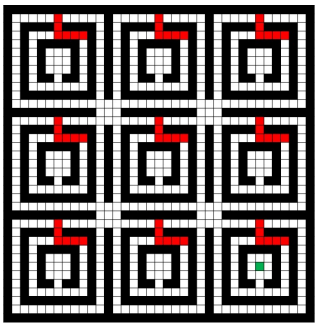

Figure 7. Movements in Map.

Bonus: G was chosen arbitrarily. In the no-feedback case, if you want to determine where you are as efficiently as possible, what square should you try to get to, and what moves should you take to get there?

We can do the STEP 1-3 in question d. In this way, we can make sure that no matter where the bot was initially at, after those steps, the point can only be in one cell. This procedure is showed as Figure 8.A-D.

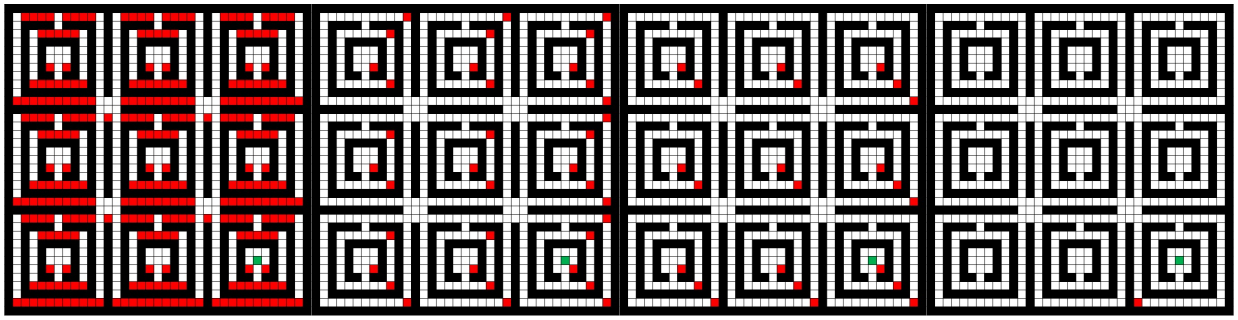

Figure 8. Movements of reducing possibility to one.

APPENDIX: SOURCE CODE

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# @Date    : 2019-05-11 17:24:05
# @Author  : Xuenan(Roderick) Wang
# @Email   : roderick_wang@outlook.com
# @Github  : https://github.com/hello-roderickwang

import numpy as np

class Map:
        def __init__(self):
                self.map_size = 37
                file = open('./data/Maze.txt', 'r')
                array = np.array([])
                for x in file:
                        for y in x:
                                if y is not '\t' and y is
not '\n':
                                        array =
np.append(array, y)
                # print('array:', array)
                if len(array) !=
self.map_size*self.map_size:
                        print('Load file length error! File
length:', len(array))
                self.map = array.reshape((self.map_size,
self.map_size))
                self.print_map()
                self.get_info()

        def print_map(self):
                print('map:\n', self.map)
                print('shape of map:', self.map.shape)

        def get_info(self):
                ctr_0 = 0
                ctr_1 = 0
                ctr_error = 0
                ctr_array = np.array(['0', '1'])
                for i in range(self.map_size):
                        for j in range(self.map_size):
                                # print('self.map[i][j]:',
self.map[i][j])
                                # print('type of
self.map[i][j]:', type(self.map[i][j]))
                                # print('type of
ctr_array[0]:', type(ctr_array[0]))
                                if self.map[i][j] ==
ctr_array[0]:
                                        ctr_0 += 1
                                elif self.map[i][j] ==
ctr_array[1]:
                                        ctr_1 += 1
                                else:
                                        ctr_error += 1
                                        print('error:',
self.map[i][j])
                print('There are', ctr_0, 'zeros and', ctr_1,
'ones in map.')
                print('There are', ctr_error, 'errors.')

class MazeSolvingBot:
        def __init__(self):
                self.map = Map()
                self.bot_position = [0, 0]

        def random_start(self):
                x = np.random.randint(1, 37)
                y = np.random.randint(1, 37)
                if self.map.map[x][y] != self.map.map[0]
[0]:
                        self.map.map[x][y] = 'S'
                        self.bot_position[0] = x
                        self.bot_position[1] = y
                else:
                        return self.random_start()

        def move(self, direction, step):
                for i in range(step):
                        if direction is 'u':
                                # print('bot_position:',
self.bot_position)
                                if
self.map.map[int(self.bot_position[0])-1]
[int(self.bot_position[1])] != self.map.map[0][0]:

self.bot_position[0] = self.bot_position[0]-1
                                if direction is 'd':
                                        if
self.map.map[int(self.bot_position[0])+1]
[int(self.bot_position[1])] != self.map.map[0][0]:

self.bot_position[0] = self.bot_position[0]+1
                                if direction is 'l':
                                        if
self.map.map[int(self.bot_position[0])]
[int(self.bot_position[1])-1] != self.map.map[0][0]:

self.bot_position[1] = self.bot_position[1]-1
                                if direction is 'r':
                                        if
self.map.map[int(self.bot_position[0])]
[int(self.bot_position[1])+1] != self.map.map[0][0]:

self.bot_position[1] = self.bot_position[1]+1

        def is_goal(self):
                if self.map.map[self.bot_position[0]]
[self.bot_position[1]] == 'G':
                        return True
                else:
                        return False

        def get_neighbor_num(self):
                ctr = 0
                if self.map.map[int(self.bot_position[0])-1]
[int(self.bot_position[1])-1] == self.map.map[0][0]:
                        ctr += 1
                if self.map.map[int(self.bot_position[0])]
[int(self.bot_position[1])-1] == self.map.map[0][0]:
                        ctr += 1
```

```
                if self.map.map[int(self.bot_position[0])+1]
[int(self.bot_position[1])-1] == self.map.map[0][0]:
                        ctr += 1
                if self.map.map[int(self.bot_position[0])+1]
[int(self.bot_position[1])] == self.map.map[0][0]:
                        ctr += 1
                if self.map.map[int(self.bot_position[0])+1]
[int(self.bot_position[1])+1] == self.map.map[0][0]:
                        ctr += 1
                if self.map.map[int(self.bot_position[0])]
[int(self.bot_position[1])+1] == self.map.map[0][0]:
                        ctr += 1
                if self.map.map[int(self.bot_position[0])-1]
[int(self.bot_position[1])+1] == self.map.map[0][0]:
                        ctr += 1
                if self.map.map[int(self.bot_position[0])-1]
[int(self.bot_position[1])] == self.map.map[0][0]:
                        ctr += 1
                return ctr

if __name__ == '__main__':
        my_map = Map()

        bot = MazeSolvingBot()

        neighbor_array = np.array([])
        for i in range(bot.map.map_size):
                for j in range(bot.map.map_size):
                        bot.bot_position = [i, j]
                        if bot.map.map[i][j] == '1':
                                neighbor_array =
np.append(neighbor_array, -1)
                        if bot.map.map[i][j] == '0':
                                neighbor_array =
np.append(neighbor_array, bot.get_neighbor_num())
                        if bot.map.map[i][j] == 'G':
                                neighbor_array =
np.append(neighbor_array, -2)
        neighbor_map =
neighbor_array.reshape((bot.map.map_size,
bot.map.map_size))
        print('neighbor_map:', neighbor_map)
        coordinate_array = np.array([])
        ctr_5 = 0
        for i in range(bot.map.map_size):
                for j in range(bot.map.map_size):
                        if neighbor_map[i][j] == 5:
                                ctr_5 += 1
                                coordinate_array =
np.append(coordinate_array, [i, j])
        # print('coordinate for neighbor_num = 5 are:',
coordinate_array)
        print('ctr_5:', ctr_5, 'len(coordinate_array):',
len(coordinate_array))
        two_left_array = np.array([])
        for i in range(int(len(coordinate_array)/2)):
                x = coordinate_array[i*2]
                y = coordinate_array[i*2+1]
                bot.bot_position = [x, y]
                bot.move('l', 1)
                if bot.get_neighbor_num() == 5:
                        bot.move('l', 1)
                        if bot.get_neighbor_num() == 5:
                                two_left_array =
np.append(two_left_array, [x, y])
        # print('After two LEFT move, coordinate for
neighbor_num = 5 are:', two_left_array)
        print('Number of satisfied points:',
len(two_left_array)/2)

        # sequence of observation: 5, 2, 5, 5, 5, 6
        # sequence of movement: Down, Down, Right,
Right, Right
        init_array = np.array([])
        target_array = np.array([])
        for i in range(int(len(coordinate_array)/2)):
                x = coordinate_array[i*2]
                y = coordinate_array[i*2+1]
                bot.bot_position = [x, y]
                bot.move('d', 1)
                if bot.get_neighbor_num() == 2:
                        bot.move('d', 1)
                        if bot.get_neighbor_num() == 5:
                                bot.move('r', 1)
                                if
bot.get_neighbor_num() == 5:
                                        bot.move('r',
1)
                                        if
bot.get_neighbor_num() == 5:

bot.move('r', 1)
                                                if
bot.get_neighbor_num() == 6:

init_array = np.append(init_array, [x, y])

target_array = np.append(target_array, bot.bot_position)
        print('init_array:', init_array)
        print('target_array:', target_array)
        print('number of target points:', len(target_array)/2)
```