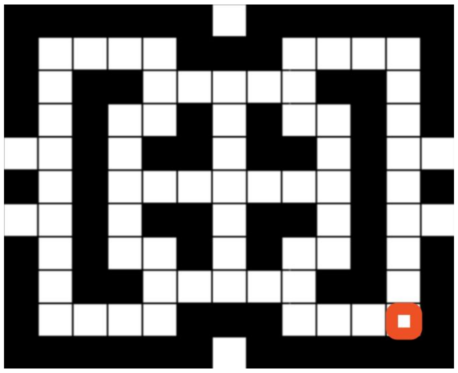
Zhuohang Li(zl299)

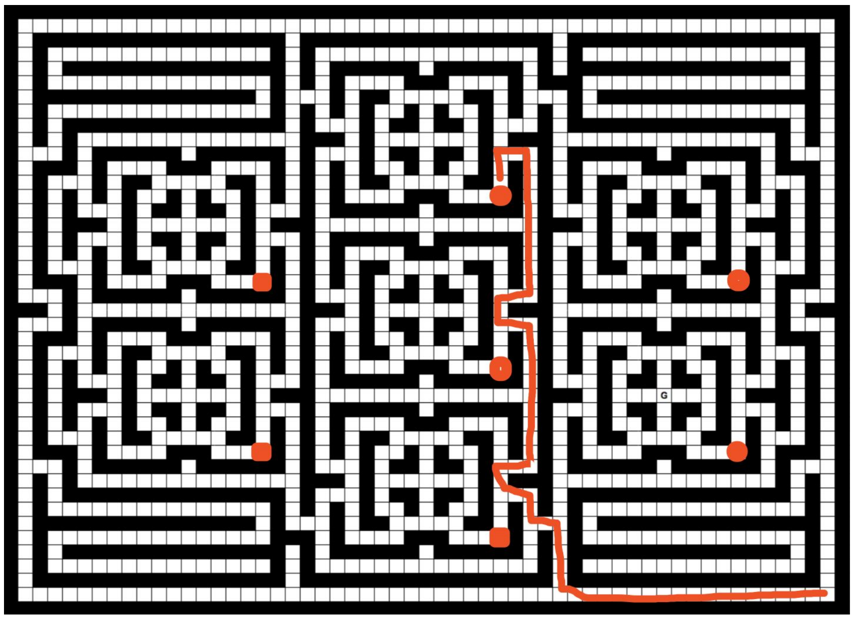
CS 520 Final: Question 1 - Localization

a. Given the fact that:1.the bot cannot be placed on the wall, 2. the probability of placing the bot in any reachable cell is equal, and therefore the probability that the bot is at G is:
1 / number of reachable cells, which is $1/1239 = 8.071 \times 10^{-4}$

b. **Strategy 1 - Manually track:**
First I tried to do it manually, just to get a rough idea of how much steps it may take. Noticed that the maze is composed of many similar patterns as is shown below:



So I first try to move the bot from all possible position within the pattern to its bottom right corner (marked as red) with the following sequence:
R3DR4DRD4LD3R3UR5DR4D2LD2R4



And with the following sequence I was able to move the bot from all red points to the bottom right of the map:

U3R2D10L2D2R2D10L2D2R2D2R2D3L14D4R14U3R2D8R18

Finally, I just need to move the bot from the bottom right to the goal point with:

L18U8R14U5L2D3L3UL2U3

The final sequence is:

RRRDRRRRDRDDDDLDDDRRRURRRRRDRRRRDDLDDRRRRUUURRDDDDDDDDDDLLDDRR
DDDDDDDDDDLLDDRRDDRRDDDLLLLLLLLLLLLLLDDDDRRRRRRRRRRRRRRUUURRDDD
DDDDDRRRRRRRRRRRRRRRRRRLLLLLLLLLLLLLLLLLLUUUUUUUURRRRRRRRRRRRRRUU
UUULLDDDLLLULLUUU

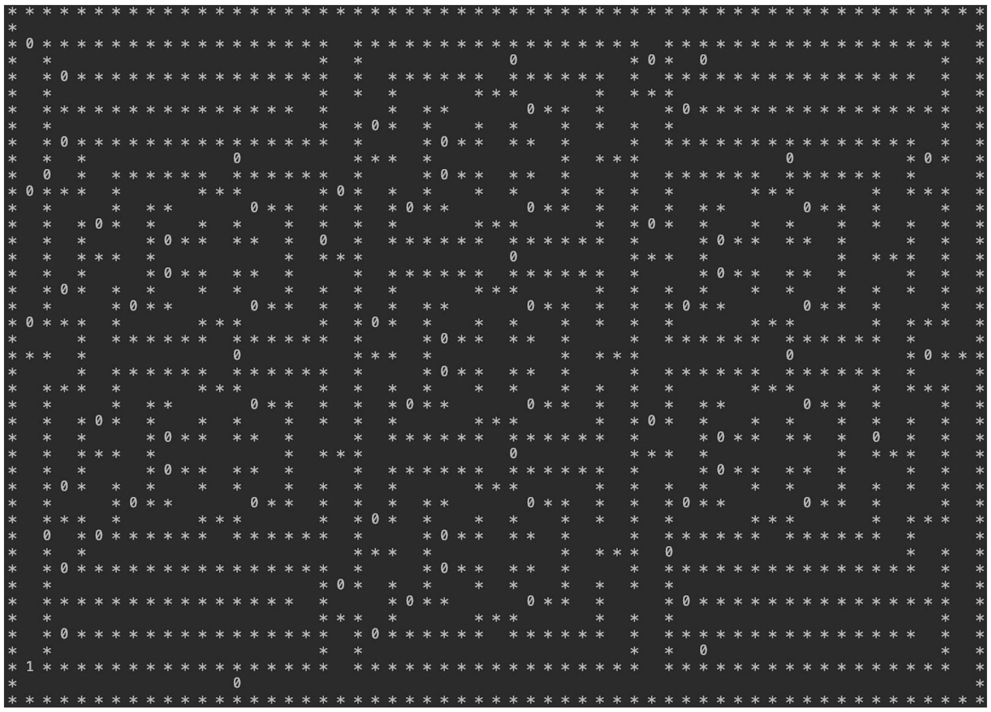
---

which is totally 207 moves and the probability that the bot is at G after this sequence of moves is 0.65213.

Strategy 2 - Automatically Run

Clearly, manually tracking the sequence to move the bot from all possible positions to the goal point is difficult and trivial, therefore I need an algorithm to do the work for me.

1. *Representation:* First I need to figure a way to represent this problem. Inspired by Question a), I decided to represent this in a probability way. The probability of a grid is the probability of the bot being in that grid after some sequence of moves. I used a 2-dimensional matrix to keep the probability of all grids. The probability of a wall grid is always 0. And at any stage of the computation, the probability of all grids should always sum to 1 (There is no way for the bot to escape from the maze). Then I implemented functions for moving right, left, up and down. For example, if I call right(), the probability of all the grids will move to its right, except for those with a wall on its right side.
2. *Evaluation:* Next I need to find some kind of evaluation of which direction should the bot go for each step. Inspired by what I did in the manually tracking part, generally what I want is to try to narrow down the number of positions that the bot may be in. Because once I successfully made two possible positions merged into one position, they will 'stick' to each other forever. And if I managed to merge all possible positions into one single position, all I need to do next is to apply a path-finding algorithm to it and move it from wherever it is to the goal point. At each step, I let the algorithm to move towards a direction such that after this step the number of possible positions is the fewest. Based on the observation, typically the could reduce the number of possible positions from 1237 to around 80 within less than one hundred steps. But after that, this strategy failed to provide rational moves because at this stage the board is pretty sparse and you will not be able to get to any other possible position with just one move. For example, below is what the board looks like after 84 steps, '0' and '1' represent possible positions.



```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                                                               *
* 0 * * * * * * * * * * * * * * * * *   * * * * * * * * * * * * * * * * *   * * * * * * * * * * * * * * * * *   *
*   *                               *   *                 0             * 0 *   0                           *   *
*   * 0 * * * * * * * * * * * * * * *   *   * * * * * *   * * * * * *   *   * * * * * * * * * * * * * * *   *   *
*   *                               *   *   *         * * *         *   * * *                               *   *
*   * * * * * * * * * * * * * * *   *       *   * *         0 * *   *       * 0 * * * * * * * * * * * * * * *   *
*   *                               *   * 0 *   *     *   *     *   *   *   *                               *   *
*   * 0 * * * * * * * * * * * * * * *   *       * 0 * *   * *   *       *   * * * * * * * * * * * * * * *   *   *
*   *   *                 0             * * *   *               *   * * *                 0             * 0 *   *
*   0   *   * * * * * *   * * * * * *   *       * 0 * *   * *   *       *   * * * * * *   * * * * * *   *       *
* 0 * * *   *         * * *         * 0 *   *   *     *   *     *   *   *   *         * * *         *   * * *   *
*   *       *   * *         0 * *   *   *   * 0 * *         0 * *   *   *   *   * *         0 * *   *       *   *
*   *   * 0 *   *     *   *     *   *   *   *         * * *         *   * 0 *   *     *   *     *   *   *   *   *
*   *   *       * 0 * *   * *   *   0   *   * * * * * *   * * * * * *   *       * 0 * *   * *   *       *   *   *
*   *   * * *   *             *   * * *                 0             * * *   *               *   * * *   *   *
*   *   *       * 0 * *   * *   *       *   * * * * * *   * * * * * *   *       * 0 * *   * *   *       *   *   *
*   * 0 *   *   *     *   *     *   *   *   *         * * *         *   *   *   *     *   *     *   *   *   *   *
*   *       * 0 * *         0 * *   *   *   *   * *         0 * *   *   *   * 0 * *         0 * *   *       *   *
* 0 * * *   *         * * *         *   * 0 *   *     *   *     *   *   *   *         * * *         *   * * *   *
*       *   * * * * * *   * * * * * *   *       * 0 * *   * *   *       *   * * * * * *   * * * * * *   *       *
* * *   *                 0             * * *   *               *   * * *                 0             * 0 * * *
*       *   * * * * * *   * * * * * *   *       * 0 * *   * *   *       *   * * * * * *   * * * * * *   *       *
*   * * *   *         * * *         *   *   *   *     *   *     *   *   *   *         * * *         *   * * *   *
*   *       *   * *         0 * *   *   *   * 0 * *         0 * *   *   *   *   * *         0 * *   *       *   *
*   *   * 0 *   *     *   *     *   *   *   *         * * *         *   * 0 *   *     *   *     *   *   *   *   *
*   *   *       * 0 * *   * *   *       *   * * * * * *   * * * * * *   *       * 0 * *   * *   *   0   *   *   *
*   *   * * *   *             *   * * *                 0             * * *   *               *   * * *   *   *
*   *   *       * 0 * *   * *   *       *   * * * * * *   * * * * * *   *       * 0 * *   * *   *       *   *   *
*   * 0 *   *   *     *   *     *   *   *   *         * * *         *   *   *   *     *   *     *   *   *   *   *
*   *       * 0 * *         0 * *   *   *   *   * *         0 * *   *   *   * 0 * *         0 * *   *       *   *
*   * * *   *         * * *         *   * 0 *   *     *   *     *   *   *   *         * * *         *   * * *   *
*   0   * 0 * * * * * *   * * * * * *   *       * 0 * *   * *   *       *   * * * * * *   * * * * * *   *       *
*   *   *                               * * *   *               *   * * *   0                           *   *   *
*   * 0 * * * * * * * * * * * * * * *   *       * 0 * *   * *   *       *   * * * * * * * * * * * * * * *   *   *
*   *                               * 0 *   *   *     *   *     *   *   *   *                               *   *
*   * * * * * * * * * * * * * * *   *       * 0 * *         0 * *   *       * 0 * * * * * * * * * * * * * * *   *
*   *                               * * *   *         * * *         *   *   *                               *   *
*   * 0 * * * * * * * * * * * * * * *   * 0 * * * * * *   * * * * * *   *   * * * * * * * * * * * * * * *   *   *
*   *                               *   *                               *   *   0                           *   *
* 1 * * * * * * * * * * * * * * * * *   * * * * * * * * * * * * * * * * *   * * * * * * * * * * * * * * * * *   *
*                         0                                                                                     *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

3. *Random Walk*: Since the evaluation is not working after this stage, I need another strategy to keep my algorithm running. The most direct way is to just arbitrarily pick a direction. But given a sparse board like this, with a large average distance between two possible positions, it is very possible for a totally random algorithm to take tons of steps to find another convergence. So I did a little modification and make the probability of choosing the same direction as last step higher than other directions. So the algorithm will be likely to continue to walk down the same direction farther before it switches to another direction.
4. *Path-Finding*: The random process will be able to reduce the number of possible positions from around 80 to around 30. After this point, the random algorithm will also become very inefficient. I don't want to waste more steps to try out my luck so it's time to try to intentionally move things together using a path-finding algorithm. To find the possible shortest sequence, I chose to use BFS which guarantees to return the shortest path. At each step, I try to move the grid with second-largest probability to the grid with the largest probability. Of course, while I performing the sequence of moves, the

position with the largest probability will also change. So I keep doing that, until a point when the positions for the second-largest probability and the largest probability are finally overlapped and become a new position with the largest probability, and then continue to move the new second-largest to the new largest. Keep doing this, until the largest probability is greater than 0.5, then I move it to the goal point and the algorithm is finished.

5. *Result:* I gave it a few runs and the best I got is a sequence like this:

```
RLLLDDDDDDDDDLLLLLLLLLLDDDDDDDDDDLLLLLLLLLLLLLLLLLLLLLLLL
LLLLLLLLLLLLRLUDDRLUDRRRLUDURDDDRRLUDDRLUDDRUURLDRDDRRLUDR
RRLLRLUDDURRRURLUDDURRRDDLRRRRRRLUDURDDLLDDRRRRLRRRLUDDDDU
DRRRLLDDRLUDRLUDRLRLUDDURLUDDRLUDULLRRLLLLUULLLDLLLLLURLURL
UDDDDRLUDRLUDRLUDDDLLLDURLUDUDLRRLUDDRUULLDRLUDDDUUDRLUDR
LUDRLRLUDDLLLLLRLUDRLUDRRLUDRLUDRLUDDUUDDRLUDDRLLUUUUDLLLR
LUDDRLUDULLLURLUDRLUDRLUDDUDDDRDDLLLUUULLDDDDDDDDLLDDDLLLL
LLLLLLLLLLUUULLUUUUUUUUULLDDDLLLULLLLULRDDLLLUUUUULLUULLDUUU
UURRUULLUUUUUUUUUUUUUUUUUUURRRRRRRRRRRRRRRRRDDDDDDDDDD
DDLLDDRRDDDDDLLLLLLLLLLLLLLDDDLLDURRUUUUUUUURRDDDUUUUUUUUR
RRDRRRRURRRDDDRRUUUUUUUURRUUURRRRRRRRRRRRRRDDDRRDDDDDDDDR
RDDLLDDDDDDDDDDRRUUURRRDRRDDD
```

There are totally 669 steps, and the final probability of the bot end up at the goal point is 0.76997.

c. To move the bot from all possible positions to the goal point, all I need to do is to make a little modification on the algorithm I used in Question b), make it stop only when there is only one possible position left, and then move it to the goal. Here is what I got:

```
RLLLDDDDDDDDDLLLLLLLLLLDDDDDDDDDDLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
LLLLLLLRLUDDRURLUDLLLDDDLDDURLUDRLUDLLRLUDDDDURLUDDLRLUDDDLLDRUR
LUDDRRDRDLRLLRLUDUULRLUDURRLUDUURRRDRLLRLDUDDLDLLLLRRDDRDRLDDRU
LURUURDDRLUDLLRRRDRDDDLRLUDRLUDRLUDURLUDDDDDDRDLDRLUDLURLUDLDLL
LRLUDDRLUDLDLDLRLUDDDRLUDDDRRRURLUDLLLRLUDLRLUDDRLUDUURLUDUURLU
DLLRLUDLLRLUDRLUDRRUUUUUDLDDRLUDLDDRLLRLLLRLUDDURRLUDDRLUDRLUDD
DDRLUDRRLUDURLUDULDURLUDDRLUDURLUDRLUDRUDDDRLUDRLUDUUUUURLUDRL
UDRLUDDDDUUDDDRLUDURLUDUULLDDRLUDRRLUDRLUDRLUDULDUURLUDRRRDURR
LUDURRUUUUUUUUURRUULLUUUUURRRRRRRRRRRRRUUUUUUUULLLLLLLLLLLLLL
LDDDDDDDDDDDDDDDDDDDRRDDLLDDDDDDDDDDDDDDUUUUUUUUUURRUULLUULLU
```

UUUULLLLLLLLLLLLLLLLLLLLDDDDDDDDDDDDDDDDDDDDDRRDDLLDDDDDDDDDDDDDDDDD
DDDRRDDLLDDDDDDDDDDDDDDLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLRRRUUURR
DDDDDDDDDDLLDDRRDDDDDDDDDDDLLDDRRDDRRDDDDDLLLLLLLLLLLLLLLLLLLLLLL
LLLLLLLLLLLLRRRRRRRRRRRRRUUURRDDDDDLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
LLLLLDDDDLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLRRRRR
RRRRRRRRRDDDRRDDDDDDDDDDDLLDDRRDDDDDDDDDDDDDDDDDDDDDDLLLLLLLLLLLL
LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLRRRRRRRRRRRRRUUURRDDDDDLLL
LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLDDDDDDDDDDDDDDDDDDDRRRRRRRRRRRRR
UULLLLLLLLLLLLLLLUURRRRRRRRRRRRRRRUUURRDDDDDDDDDLLLLLLLLLLLLLLLLLLL
LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLDDDDDDDDLLLLLLLLLLLLLLLLLLLLLLLLLLL
LLLLLLLLLRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRUUUUUUUUUUUUURRDDDR
RRURRUUU

Length: 1364

d. 1) First, find all cells with 5 blocked cells surrounded and add them to a list. Then iteration through all the cells in the list, and let them move to its left. If it's blocked, it will stay where it was. Then check again for each cell, the number of surrounding blocked cells, if it's still equal to 5, keep it in that list, otherwise, delete it from the list. Repeat another time. Finally, we got a list of cells which we are likely to be in. The length of this list is 284, that means there are 284 potential locations before the moves, and the probability of the bot being in each cell is: 1/284 = 0.003521. The number of unique cells is 256, with 14 of them has a larger probability of 0.01056. The probability for the bot to be in the rest of the maze is 0.
Final probability:

1. For the following cells, the probabilities are 0.01056 (totally 14 cells): [5, 30], [11, 14], [11, 46], [13, 30], [17, 30], [19, 14], [19, 46], [23, 14], [23, 46], [25, 30], [29, 30], [31, 14], [31, 46], [37, 30]
2. For the following cells, the probabilities are 0.003521 (totally 242 cells): [1, 18], [1, 36], [2, 1], [2, 55], [3, 27], [4, 21], [4, 28], [4, 35], [5, 19], [5, 21], [5, 24], [5, 35], [6, 19], [6, 23], [6, 33], [6, 35], [7, 19], [7, 23], [7, 25], [7, 33], [7, 37], [8, 3], [8, 19], [8, 33], [8, 37], [8, 53], [9, 1], [9, 3], [9, 11], [9, 25], [9, 37], [9, 43], [9, 53], [9, 55], [10, 1], [10, 5], [10, 12], [10, 19], [10, 33], [10, 37], [10, 44], [10, 51], [11, 1], [11, 5], [11, 8], [11, 21], [11, 23], [11, 25], [11, 33], [11, 35], [11, 40], [11, 51], [11, 55], [12, 7], [12, 17], [12, 23], [12, 33], [12, 39], [12, 49], [12, 51], [13, 3], [13, 7], [13, 9], [13, 17], [13, 19], [13, 24], [13, 37], [13, 39], [13, 41], [13, 49], [13, 53], [14, 17], [14, 21], [14, 28], [14, 35], [14, 49], [15, 9], [15, 21], [15, 41], [16, 17], [16, 21], [16, 28], [16, 35], [16, 49], [17, 3], [17, 7], [17, 9], [17, 17], [17, 19], [17, 24], [17, 37], [17, 39], [17, 41], [17, 49], [17, 53], [18, 7], [18, 17], [18, 23], [18, 33], [18, 39], [18, 49], [18, 51],

[19, 1], [19, 5], [19, 8], [19, 21], [19, 23], [19, 25], [19, 33], [19, 35], [19, 40], [19, 51], [19, 55], [20, 5], [20, 12], [20, 19], [20, 33], [20, 37], [20, 44], [20, 51], [21, 5], [21, 25], [21, 37], [22, 5], [22, 12], [22, 19], [22, 33], [22, 37], [22, 44], [22, 51], [23, 1], [23, 5], [23, 8], [23, 21], [23, 23], [23, 25], [23, 33], [23, 35], [23, 40], [23, 51], [23, 55], [24, 7], [24, 17], [24, 23], [24, 33], [24, 39], [24, 49], [24, 51], [25, 3], [25, 7], [25, 9], [25, 17], [25, 19], [25, 24], [25, 37], [25, 39], [25, 41], [25, 49], [25, 53], [26, 17], [26, 21], [26, 28], [26, 35], [26, 49], [27, 9], [27, 21], [27, 41], [28, 17], [28, 21], [28, 28], [28, 35], [28, 49], [29, 3], [29, 7], [29, 9], [29, 17], [29, 19], [29, 24], [29, 37], [29, 39], [29, 41], [29, 49], [29, 53], [30, 7], [30, 17], [30, 23], [30, 33], [30, 39], [30, 49], [30, 51], [31, 1], [31, 5], [31, 8], [31, 21], [31, 23], [31, 25], [31, 33], [31, 35], [31, 40], [31, 51], [31, 55], [32, 1], [32, 5], [32, 12], [32, 19], [32, 33], [32, 37], [32, 44], [32, 51], [33, 1], [33, 3], [33, 11], [33, 25], [33, 37], [33, 43], [33, 53], [33, 55], [34, 3], [34, 19], [34, 33], [34, 37], [34, 53], [35, 19], [35, 23], [35, 25], [35, 33], [35, 37], [36, 23], [36, 33], [36, 35], [37, 21], [37, 24], [37, 35], [37, 37], [38, 21], [38, 28], [38, 35], [39, 27], [40, 1], [40, 55], [41, 18], [41, 36]

3. The probabilities of the rest cells are all 0.

2) Algorithm:

```
cells = a list of cells with the number of surrounding blocked cells
        equals to Y0
FOR i from 0 to n-1:
        perform action Ai
                FOR cell in cells:
                        IF number of surrounding blocked cells = Yi:
                                keep cell in list
                        ELSE:
                                Discard cell
FOR cell(i, j) in cells:
        Update prob maxtrix[i, j] += 1/Len(cells)
FOR each cell(i, j) in prob matrix:
        IF prob matrix [i, j] == maximum probability:
                Add cell(i, j) in most_likely_cell_list
RETURN most_likely_cell_list
```

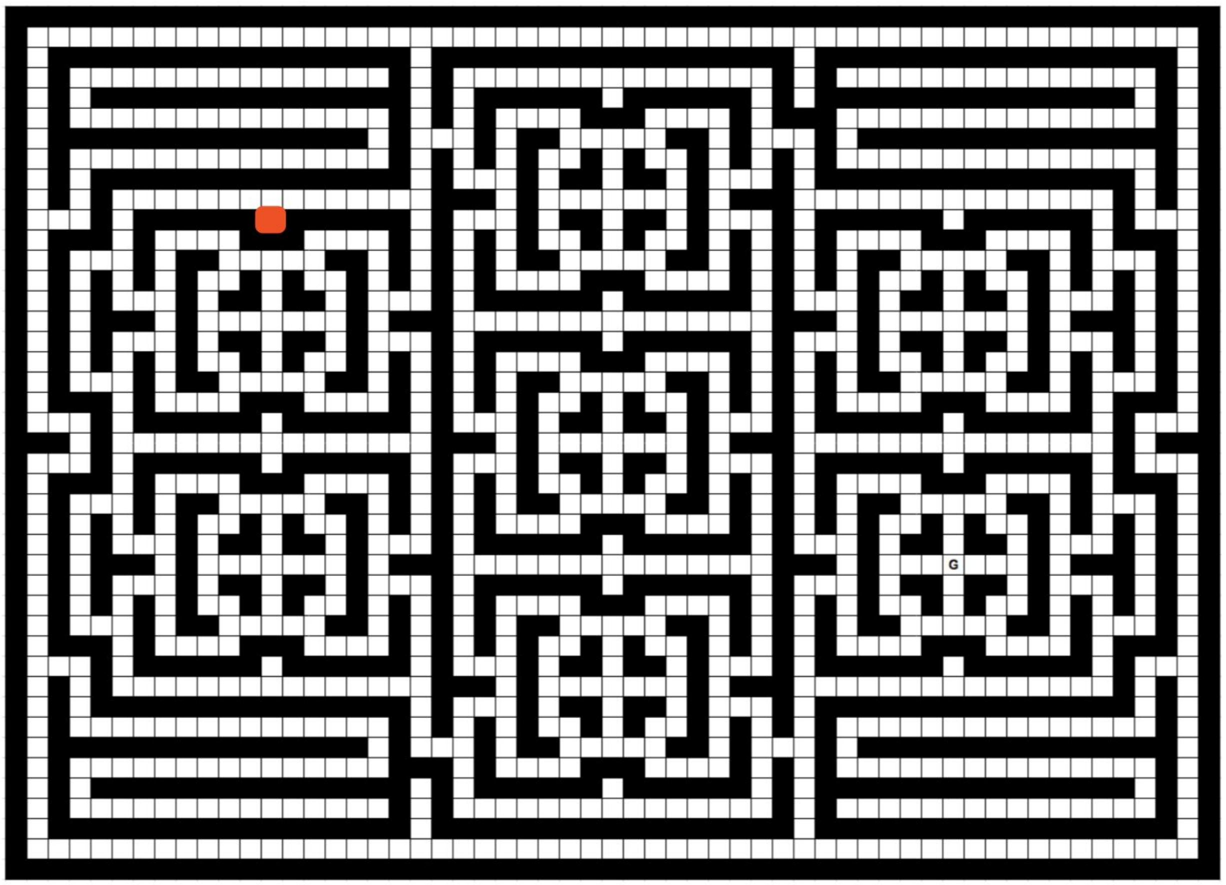
I also implemented this algorithm in *Localization.py*, the function name is most_likely_cells().

Here is the returned list of most likely cells by calling most_likely_cells([5,5,5], ‘LL’):

[5, 30], [11, 14], [11, 46], [13, 30], [17, 30], [19, 14], [19, 46], [23, 14], [23, 46], [25, 30], [29, 30], [31, 14], [31, 46], [37, 30]

Bonus:

The cell we want to use to determine where we are should be easy to reach and has as much surrounding blocked cells as possible. A potential cell for a quick localization could be [10, 12], which is marked red in the following image:



The following sequence can get us there:

RLLLDDDDDDDDDLLLLLLLLLLLDDDDDDDDDLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
LRLUDDLRLUDDDLDDDLRLUDLLLRLUDRLUDRDLLDDDRLUDDDULRLUDLLDDDDURLUDDRULLL
RRLRLUDDRLRRRUDRRRLUDRLUDUDDUDRLUDUURLUDDDLRRLUDLLRRLUDRLUDDDRRRLUDD
RDURLLULLLLLLURLUDURDRUDDLLUULDRLUDRLUDRLUDRRRLUDURLUDURLUDLDRRULRLUD
URLLLLLDLLRRUUURRRRRUURLUDRLUDDUDRRDRRRLUDDDULDRLUDRLUDRLUDDRLUDLLUU
URLUDRLUDLURLUDRDRRRRDLULUUUDURDDDDULRRLUDRRRLUURRLUDDRDURULRRLUDDRD

```
LUUUULRUULUDLLLLLURLUDRLUDRLUDRLUDLDDDDDDRLRRRUDLDDRRUUUDDRLUDLLRLUD
RLUDDRUDDLDDDUDRRLUDDLRRDDLLDDDDDDDDDDDDDDDDDDDDRRRRRRRRRRRUUURRDDDD
DDDDRRDDDRRRRRRRRRRRRRRRUUURRDDDDDLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
LLLDDRRDDDDDDDDDRRDDDRRRRRRRRRRRRRRRUUURRDDDDDLLLLLLLLLLLLLLLLLLLLLLL
LLLLLLLLLLDDDLLLLLLLLLLLLLLLDDDDDDDDLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLRRR
UUURRDDRRDDDDDLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLUUULLDDDDDDDDDDRRDDLL
DDDDDDDDDDDDDLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLRRRRRRRRRRRRRRDDDRRDDD
RRRRRRRRRRRRRRUUURRDDDDDLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLRRRRRRRRRRR
RRUULLLLLLLLLLLLLLLUURRRRRRRRRRRRRRUUURRDDDDDDDDDLLLLLLLLLLLLLLLLLLLLLL
LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLRRRRRRRRRRRRRRUULLLLLLLLLLLLLLUUULLDDDDDDD
DDRDRRRRDRRRUUURRDDRRDDDDDLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
LLLLLLLLUUUUUUUUUUUUUUUUUUUURRUULLUUUUUUUUUUUUUUUUUUURRRRRRRRRRRRRR
RRRDDDDDDDDDLLLLLLLD
```

Length: 1282