**RUTGERS UNIVERSITY**

**COMPUTER SCIENCE**

**520**

**EXAM 1**

|  |  |
| --- | --- |
| **Author:**  Shobhit Singh (**ss4363**) | **Supervisor:**  Prof. Wes Cowan |

A picture containing logo

Description automatically generated

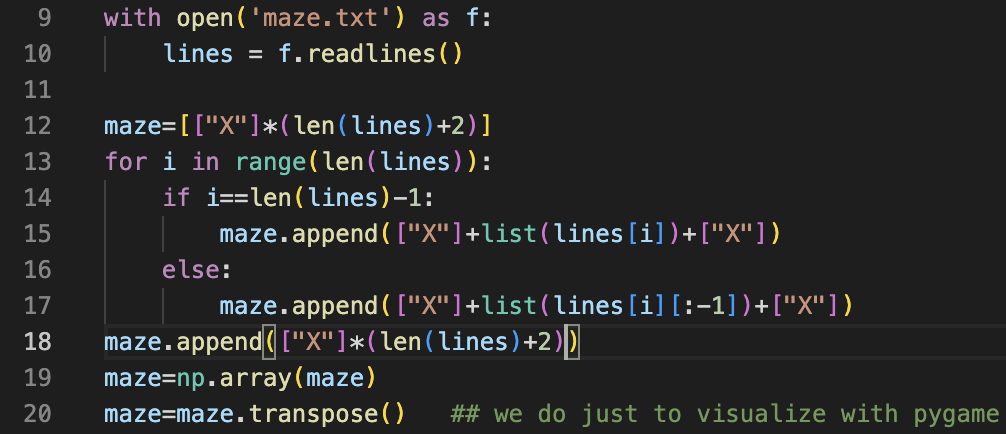
December 20, 2022

**INTRODUCTION:**

The goal is to find least number of sequence of commands that lets us know for sure where a drone is inside a maze-like network.

1. **ENVIRONMENT**

To generate the environment, we read a text file that has the sequence of “X” and “\_” that represents the walls and path of the internal networks on NNPS. We first read through this file and generate a 2-dimentional matrix that represents this path. The following code



Note that doing a transpose makes the maze representation inaccurate but it is needed for the visualization with pygame. The actual matrix representation of the network is without transpose

***Q1.* Before you do anything, what is the probability that the drone is in the top left  
corner? Why?**

Since we don’t have any information about the position of the drone, we can assume the likelihood of it being in a position as a normal distribution. So, we can assign equal probability for the likelihood of the drone for all possible locations. Once we have the environment ready, we can calculate all possible location using the below code.

The value comes up to be 199. So, all possible location has a probability of 1/199 for the drone to be in that location in initial state.

***Q2. What are the locations where the drone is most likely to be? Least likely to be?  
How likely is it to be in all the other locations? Indicate your results visually.***

In general, we initialize all location as 1/199 and when we give a move command day “down”, some drones that have an open space downward, moves down and some that have the path down blocked, stays at the same place. It is possible, that for a cell, the path is blocked and the original probability of a drone being there remains the same and its neighbor cell(upward in our case) moves over the cell adding the total probability of both cell to that cell.

So, in conclusion the drones move over one another on certain moves adding the probabilities of both cell. The resulting probabilities of the drones after the down movement is given below.

Graphical user interface, chart

Description automatically generated

Here, all the grey locations have a probability on 0, the orange locations have a probability of 1/199, and all the brown locations have a probability of 2/199.

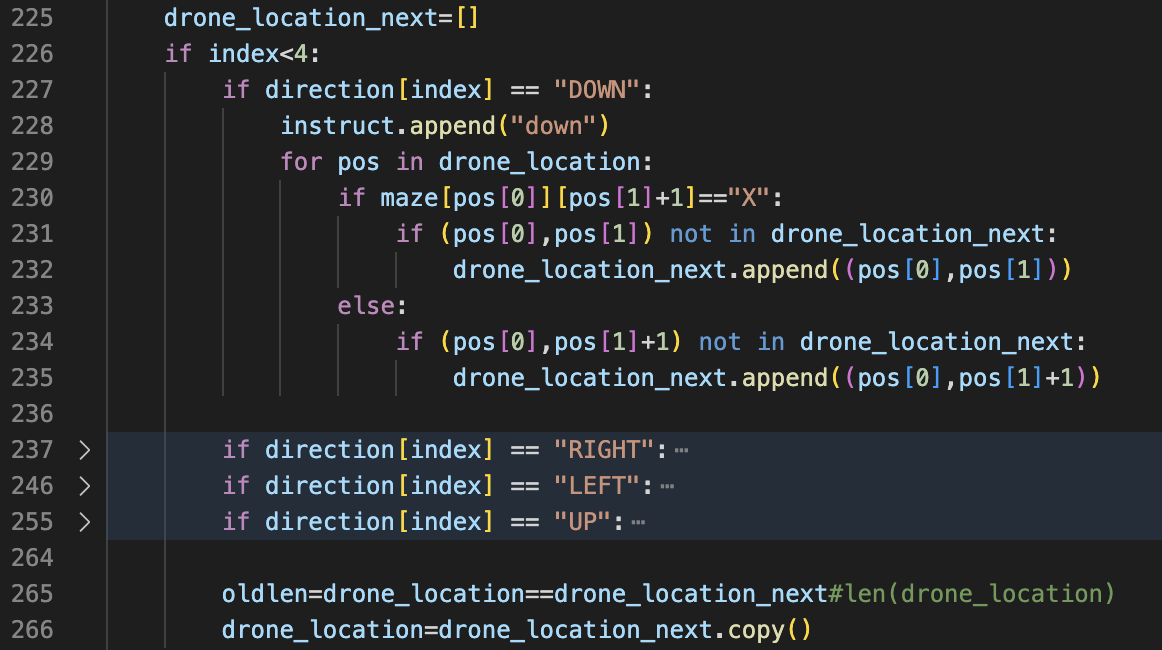
Notice that that number of grey and brown locations are equal. So, the total probability still remains 1.

1. **IMPLEMENTATION (shortest sequence to know the drone location**

We have already established that at every move, some probable drone location overlap. It should also be noted, that once they do overlap, the total number of these probable location are reduced, they never increase. This is because the moves have the same effect on every drone. So, if we keep making moves that enable reduction of these location, after N commands, it will become 1 which is our goal state. We also need to minimize the value of N.

We will divide this implementation into 3 parts:

1. First, we need to make moves that maximize the reduction of possible drone location. We simply execute moves until the no. of locations stop decreasing or no further changes happen. To achieve this, we use the following code



We have a direction variable initialized as *direction = [“UP”, “RIGHT”, “DOWN”, “LEFT”]* and we iterate through each of the command till it stops reducing probable location(sometimes, even if it does not change the number of elements, it starts doing so if we keep going. So we take no change as condition). If it does stop, we increment the value if index and try the next move to check for changes until all moves get’s tried once.

*IMPORTANT NOTE: The if-condition for other directions have been closed to fit. It should be visible that i incremented the second index as plus one for “down” and not the first index which is normally done. This is because we are working with a transpose matrix as our pygame visualization flips it. So even the index is correct. Please be vary that the same is true for a lot of places. It is correct.*

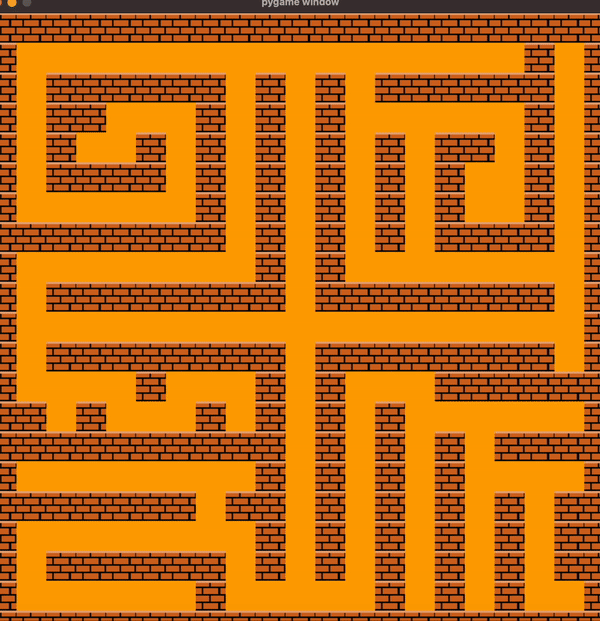
The part to increment the index is below:

Text

Description automatically generated

We also need to remove the last move from instruction list as we know it didn’t make any changes to the locations(line 306)

After implementing the above part, the graph simulates and looks like below.

****

As we can see, we have already reduced the possible locations(Orange blocks) a lot.

1. The next step is to move the these probable location to 1 same location. We can use the astar algorithm from project 1 to find and move them to one location but that comes in step 3. In step 2, we perform a local version of the same. Instead of moving all points to a far away distance one by one, we first try to combine few probable location locally so only need to cover the long distance once. The concept is demonstrated below.

It is clear that the right case travels less distance

To do this, we define some local zones and divide the maze/tunnel into smaller parts. We first aim to combine all probable location in these zones before doing so globally.

The below code implements the local concept:

Text

Description automatically generated

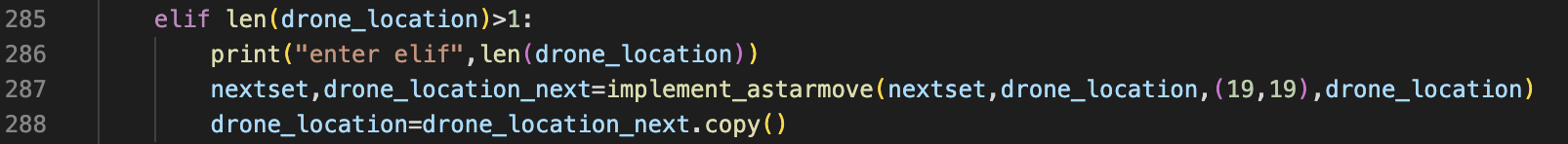
Since there are 4 zones in our case, once all local zones has one possible location of drones, we are left with 4 drone locations. So, we implement this local optimization till we have 4 total locations remaining.

We have mentioned the point of local convergence in Line 270 but this can also be calculated via code. Given we have a view of the map beforehand, we can specify these points in the interest of time(I mean the NNPS is being held under ransom). We separate the drone locations into 4 parts based on which zone they lie in and move them to the local converge location using “implement\_astarmove” function the function simply moves the locations using one move at a time similar to how we did in part 1. The move sequence is determined by astar path similar to project 1. The function is well commented in the code.

We run it till we have just 1 location for a node.

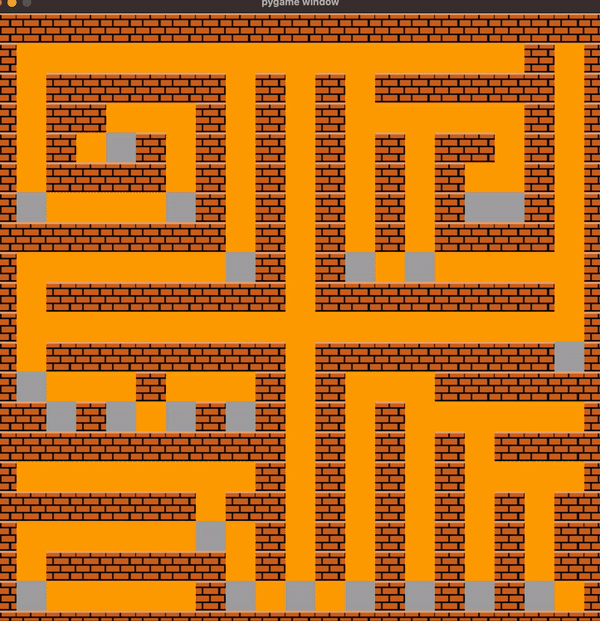
*Note: The code will perform for any dimension of graph as long as we specify the zones correctly*

1. Once we have all the local solutions, we can combine them to one location globally. This saves a lot of moves than moving without local solutions. The global solution gets implemented as follows:



We are converging all locations to (19,19) till total locations are more than 1. Once total locations become 1, our goal state is received, and we are sure the drone is there.

The Final entire simulations look like below:



The final sequence of commands came out to be

['up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'up', 'right', 'right', 'down', 'down', 'down', 'left', 'left', 'left', 'left', 'left', 'up', 'up', 'up', 'up', 'up', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'down', 'down', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'right', 'up', 'left', 'left', 'down', 'left', 'left', 'up', 'left', 'left', 'left', 'up', 'up', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'right', 'left', 'down', 'down', 'right', 'down', 'down', 'right', 'right', 'right', 'right', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'right', 'right', 'down', 'right', 'right', 'down', 'down', 'right', 'right', 'down', 'down', 'down', 'down', 'right', 'up', 'up', 'up', 'left', 'left', 'left', 'left', 'left', 'up', 'up', 'left', 'left', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'right', 'right', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'right', 'right', 'down', 'right', 'right', 'down', 'down', 'right', 'right', 'down', 'down', 'down', 'down', 'right', 'up', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'right', 'right', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'right', 'right', 'down', 'right', 'right', 'down', 'down', 'right', 'right', 'down', 'down', 'down', 'down', 'right']

This is a total of **290** commands. We can further optimize this by adjusting the number of zones, converge points, and the zone locations.

*Note: skipping step 2 and converging all lone point to (11,19) is performing better with 276 commands. But the effect of local converging will show exponentially better results as the maze dimension’s increase so I am keeping it.*

*Note: you can use the check.py file to simulate using the above sequence to verify the correctness.*

***Q4.* Find a 19\*19 reactor hat has the longest possible sequence of commands needed to locate the drone that you can find.**

To Solve for this, we need to reverse engineer our method of finding the shortest sequence. We need to think what maze minimizes the progress we make at each 3 parts of our above solution.

* In part 1, we basically shake the maze structure to combine all possible locations that are adjacent to each other or in a straight line. To minimize the reduction of locations in this case, we need to make a maze with less to know long parallel paths. This is very important as the more the number of stand-alone locations that are left after this part, the more steps we need to converge them.
* In part 2, we locally converge the points. To make this inefficient, we need to make sure that 2 zones don’t have similar configurations. Each zone needs to be converged separately and should not be the case that converging one zone reduces the no. of locations in other zones simultaneously.
* In part 3, we converge the local optimized locations. We need to make sure that these distances are long. Each zone should not have multiple paths leading outside of it. It should have one outlet that makes the distance longest when converging.

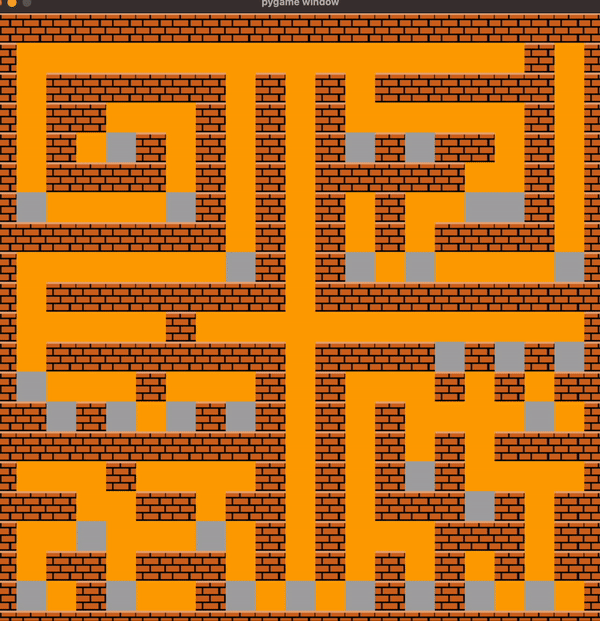
The graph that we are using, has a lot of long paths and multiple paths leading to places. We need to change according to these 2 conditions. Also, when simulating local zones, if the locations of other zones start to converge as well, we need to alter those parts of zone as well. We also need to make sure the graph stays well connected and there are no blocked areas or else the locations can never converge.

So, we can write a code that simulates for the shortest path but alters the maze accordingly if it encounters the conditions mentioned above. We start with the maze provided to us and make changes to it until we get a larger sequence to know the exact location of drone.

We stop at a sequence of length 539 with below instructions:

['up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'left', 'up', 'up', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'down', 'down', 'right', 'right', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'left', 'left', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'left', 'left', 'up', 'up', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'up', 'right', 'right', 'down', 'down', 'down', 'left', 'left', 'left', 'left', 'left', 'up', 'up', 'up', 'up', 'up', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'up', 'left', 'left', 'left', 'up', 'up', 'up', 'up', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'right', 'right', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'right', 'left', 'down', 'down', 'right', 'down', 'down', 'right', 'right', 'right', 'left', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'right', 'right', 'down', 'down', 'right', 'right', 'right', 'right', 'right', 'down', 'down', 'down', 'left', 'left', 'left', 'down', 'down', 'right', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'right', 'right', 'down', 'down', 'right', 'right', 'right', 'right', 'right', 'down', 'down', 'down', 'left', 'left', 'left', 'down', 'down', 'right', 'up', 'up', 'up', 'left', 'left', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'left', 'left', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'right', 'right', 'down', 'down', 'right', 'right', 'right', 'right', 'right', 'down', 'down', 'down', 'left', 'left', 'left', 'down', 'down', 'right', 'up', 'up', 'up', 'left', 'left', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'left', 'left', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'right', 'right', 'down', 'down', 'right', 'right', 'right', 'right', 'right', 'down', 'down', 'down', 'left', 'left', 'left', 'down', 'down', 'right', 'left', 'left', 'left', 'left', 'left', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'right', 'right', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'right', 'right', 'down', 'right', 'right', 'down', 'down', 'right', 'right', 'down', 'down', 'down', 'down', 'right', 'up', 'up', 'up', 'left', 'left', 'left', 'left', 'left', 'up', 'up', 'left', 'left', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'right', 'right', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'right', 'right', 'down', 'right', 'right', 'down', 'down', 'right', 'right', 'down', 'down', 'down', 'down', 'right', 'up', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'left', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'right', 'right', 'up', 'up', 'up', 'up', 'up', 'up', 'up', 'right', 'right', 'down', 'right', 'right', 'down', 'down', 'right', 'right', 'down', 'down', 'down', 'down', 'right', 'left', 'left', 'left', 'down', 'down', 'right', 'right', 'down', 'down', 'down', 'down', 'right', 'up', 'right', 'right', 'down', 'down', 'down', 'down', 'right']

The final simulation of the altered maze is below:

****