GHOST MAZE PROJECT

Himanshu Pahwa

1) **The environment**

The creation of the matrix was done using numpy.random so as to assign probabilities to each cell as blocked or unblocked.
The probability of a cell to be blocked was taken as 0.28 and for it to be unblocked the probability was taken as 0.72.

To check whether the matrix generated is acceptable or not, we ran the Depth First Search algorithm to find a path from start node to end node.
If a path existed and if the starting and ending cell were unblocked, then it was considered an accepted maze.
All the accepted mazes were stored in a csv file so that the mazes need not be generated again and again.

Depth First Search was chosen because it can verify if the maze path existed from start node to end node using minimum space.
It could also have been achieved by BFS but since both DFS and BFS have the same time complexity, DFS saves on the space complexity.

Total number of mazes considered by us is 50 mazes of 51*51.

2) **Ghosts**

Spawning of the ghosts was done by visiting each cell of the maze and with a probability of 0.5 it was either probable to be assigned or not to that cell.
If the ghost was probable to be assigned to that cell, a path check from the probable cell to the start node was done using DFS to check that the ghost is not walled off.

Here I've classified the state of the ghost neighbors in the following ways ways -
0 - Empty Cell
1 - Blocked Cell
2 - Ghost on an empty cell
3 - Ghost on a blocked cell

```
import random
ghostparent = []
ghostchild = []
def ghost_movement(Matrix):

    for i in range(0,5):
        for j in range(0,5):
            if Matrix[i,j] == 2 or Matrix[i,j]==3:
                choice = [(i,j+1),(i,j-1),(i+1,j),(i-1,j)]
                x = random.choice(choice)
                while x[0]<0 or x[1]<0 or x[0]>4 or x[1]>4:
                    x = random.choice(choice)
                ghostparent.append((i,j))
                ghostchild.append(x)

#     Now, since i have all the locations of the selected neighbours, I categorize the movement into three parts,
#     1) Selected neighbour is empty
#     2) Selected neighbour is blocked
#     3) Selected neighbour is ghost

    for i in range(0,len(ghostparent)-1):
        if Matrix[ghostchild[i]]==0:
            Matrix[ghostchild[i]]=2
            if Matrix[ghostparent[i]]==2:
                Matrix[ghostparent[i]]=0
            elif Matrix[ghostparent[i]]==3:
                Matrix[ghostparent[i]]=1

        elif Matrix[ghostchild[i]]==1:
            move=[1,3]
            Matrix[ghostchild[i]]=np.random.choice(move, p=(0.5,0.5))
            if Matrix[ghostchild[i]] == 3:
                if Matrix[ghostparent[i]]==2:
                    Matrix[ghostparent[i]]=0
                elif Matrix[ghostparent[i]]==3:
                    Matrix[ghostparent[i]]=1
        elif Matrix[ghostchild[i]]==2:
            pass
        elif Matrix[ghostchild[i]]==3:
            pass
    print(ghostparent)
    print(ghostchild)
    print(Matrix)
```

The probability condition was also done by the numpy.random library in python.

Movement of the ghosts was done by a getchildren function which returns the neighbors of the ghosts to step on.
We randomly pick a child using numpy. random and see if the neighbor picked was blocked or not.
If blocked then the ghost moves in that block with a probability of 0.5, if unblocked ghost moves to that block with a probability of 1.

To check if our agent was killed or not, our agent will take a step then our ghost will take a step and if the agent and the ghost meet then our agent dies.

1) **<u>Agent 1</u>**

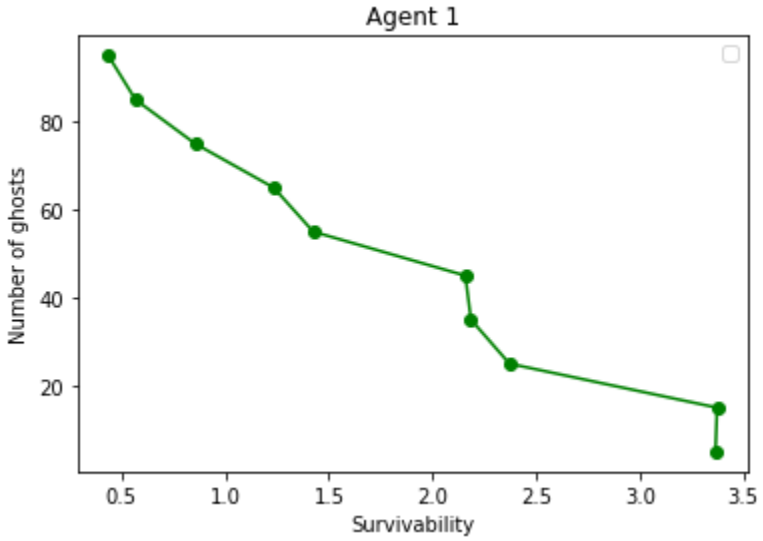
Planning → The planning of the path for agent 1 was done using BFS, this was done as BFS will give the shallowest solution for reaching the goal.

Moving Agent → When we get a path from BFS, Agent 1 starts moving, taking one step at a time on the path planned by BFS. Note that the path received from BFS would have been unblocked.
Agent 1 is not taking the ghosts into account. Hence, it keeps on moving until it reaches the goal or a ghost kills it.

Survivability → Survivability of agent 1 drastically decreased as the number of ghosts periodically increased

Agent 1 is extremely efficient as it takes the least time to run but because it does not take ghosts into account it dies very easily.

Given below is the graph that represents the situation of Agent 1.



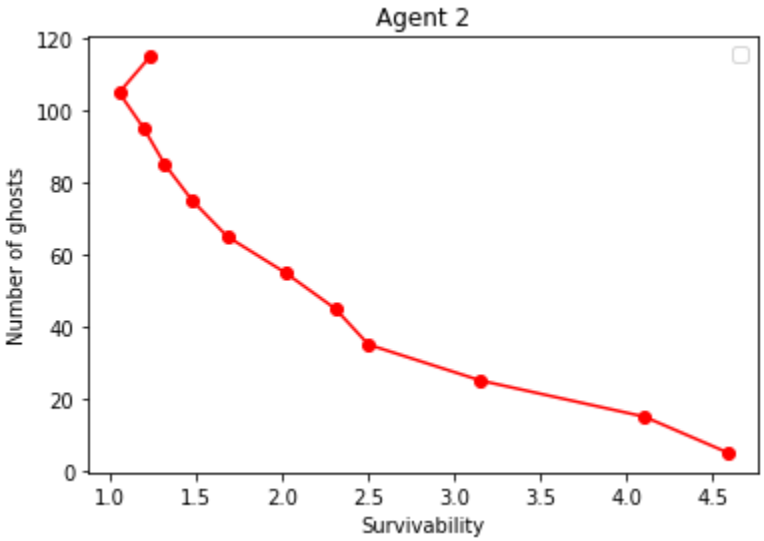
(For getting the graph we ran the survivability function for each agent and stored them to plot them later)

2) **Agent 2:**

Planning and Moving → The planning of the path is done by BFS as it will give the shallowest path to the goal.
When Agent 2 receives a path from BFS it takes a step and checks whether the following path contains a ghost or not.
If yes, then ghost 2 replans otherwise it keeps on moving the path generated.

The replaning is also done using BFS.

Survivability →



(*For getting the graph we ran the survivability function for each agent and stored them to plot them later)*

Our agent 2 works as expected, with the minimum number of ghosts, the survivability of Agent 2 goes to around 50%, with maximum number of ghosts, survivability reduces to 10%.

```
agent3(0,0,Matrix)
```

```
Result
(0, 0)
[(4, 4), (3, 4), (3, 3), (3, 2), (3, 1), (3, 0), (2, 0), (1, 0), (0, 0)]
(1, 0)
(2, 0)
replaning......
[(4, 4), (3, 4), (3, 3), (3, 2), (3, 1), (3, 0), (2, 0)]
(3, 0)
replaning......
[(4, 4), (3, 4), (3, 3), (3, 2), (2, 2), (1, 2), (1, 1), (1, 0), (2, 0), (3, 0)]
(2, 0)
replaning......
[(4, 4), (3, 4), (3, 3), (3, 2), (3, 1), (3, 0), (2, 0)]
(3, 0)
replaning......
[(4, 4), (3, 4), (3, 3), (3, 2), (3, 1), (3, 0)]
(3, 1)
replaning......
[(4, 4), (3, 4), (3, 3), (3, 2), (3, 1)]
(3, 2)
(3, 3)
(3, 4)
(4, 4)
survived
Survive
[(4, 4), (3, 4), (3, 3), (3, 2), (3, 1), (3, 0), (2, 0), (1, 0), (0, 0)]
(1, 0)
killed
Kill
[(4, 4), (3, 4), (3, 3), (3, 2), (2, 2), (1, 2), (1, 1), (0, 1), (0, 0)]
(0, 1)
(1, 1)
replaning......
[(4, 4), (3, 4), (3, 3), (3, 2), (2, 2), (1, 2), (1, 1)]
(1, 2)
```

3) **Agent 3:**

Agent 3 is purely based on survivability of the path originated from the children and not the shortest path hence it performs very poorly.

Planning and Movement → Each node checks its neighbors and runs agent 2 on each neighbor to check the survivability rate from each child.
Agent 3 then picks the child with maximum survivability rate.
As soon as Agent 3 steps on the picked child it replans again based on the above mentioned rules.

Agent 3 is simulating Agent 2 and the extent of the simulation is till the end node.

Let the utility function be the survivability from each step.

Survivability →

```
print(agent3(0,0,Matrix))
```

```
{(0, 0): 858, (0, 1): 853, (1, 0): 812}
{(0, 0): 847, (0, 1): 882, (1, 0): 803}
{(0, 1): 870, (0, 2): 856, (1, 1): 877, (0, 0): 863}
{(1, 1): 844, (2, 1): 856, (1, 0): 785, (0, 1): 889}
{(0, 1): 863, (0, 2): 862, (1, 1): 855, (0, 0): 833}
{(0, 1): 855, (0, 2): 894, (1, 1): 872, (0, 0): 874}
{(0, 2): 877, (0, 3): 869}
{(0, 2): 874, (0, 3): 868, (0, 1): 877}
{(0, 1): 862, (0, 2): 885, (1, 1): 858, (0, 0): 843}
{(0, 2): 866, (0, 3): 882, (0, 1): 865}
{(0, 3): 883, (0, 2): 904}
{(0, 2): 906, (0, 3): 892, (0, 1): 855}
{(0, 2): 863, (0, 3): 859, (0, 1): 875}
{(0, 1): 869, (0, 2): 875, (1, 1): 851, (0, 0): 877}
{(0, 0): 859, (0, 1): 870, (1, 0): 813}
{(0, 1): 857, (0, 2): 877, (1, 1): 838, (0, 0): 844}
{(0, 2): 894, (0, 3): 901, (0, 1): 876}
{(0, 3): 894, (1, 3): 919, (0, 2): 884}
{(1, 3): 882, (1, 4): 924, (2, 3): 914, (0, 3): 867}
{(1, 4): 945, (2, 4): 965, (1, 3): 891}
{(2, 4): 972, (3, 4): 992, (2, 3): 924, (1, 4): 937}
{(3, 4): 969, (4, 4): 1000, (3, 3): 953, (2, 4): 956}
soorvived
```

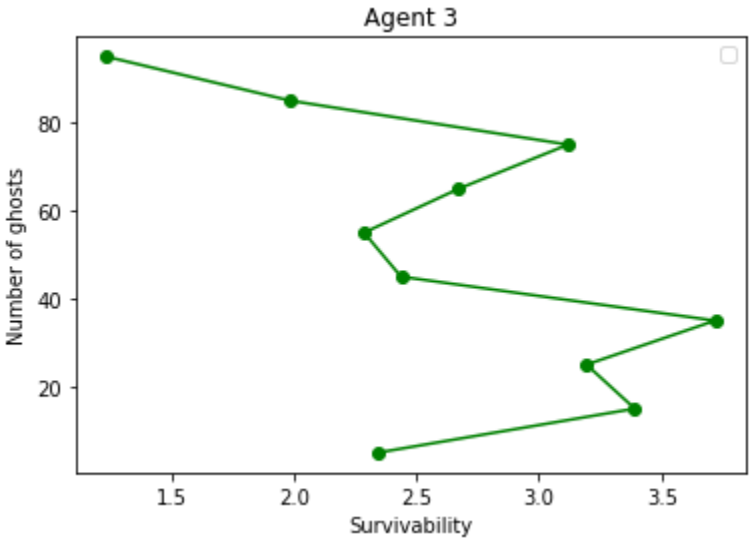
Since the ghosts are moving on each step, the utility that we are getting for each neighbor node actually doesn't provide good information to the Agent 3 to survive.

We also encounter a wiggle problem while moving Agent 3.

The Wiggle Problem: This problem originates when the survivability of 2 or more neighbors is the same.

In this case Agent 3 keeps on wiggling between 2 neighbors and can't find a solution.

Agent 3 performs worse than Agent 2 on account that the utility we are getting for each node is actually not helping the case of survivability in the longer run since the ghosts are moving on each step. In addition to this the wiggle problem tends to make Agent 3 worse than Agent 2.



(*For getting the graph we ran the survivability function for each agent and stored them to plot them later)*

Since Agent 3 showed very random results so we could not get a proper pattern for it as it behaved randomly at each number of ghosts. It was actually improving somewhat with an increase in the number of ghosts since more number of ghosts took Agent 3 out of the wiggle problem.

Q. What algorithm is most useful for checking for the existence of these paths? Why?

I believe that DFS is the most suitable algorithm for checking the existence of these paths. As we've seen in class, though BFS has a probability of providing us with a more optimum solution, DFS is faster and gives a solution. This might not necessarily be the best solution for the maze but it does fulfill the purpose

of this section of the project i.e. checking the existence of a path from start to goal node.

Q. Agent 2 requires multiple searches - you'll want to ensure that your searches are as efficient as possible so they don't take much time. Do you always need to replan? When will the new plan be the same as the old plan, and as such you won't need to recalculate?

Agent 2 calculates the path based on the path clarity. As long as there is no ghost in its path, agent 2 does not object. But when and as the ghost appears in the path, agent 2 replans and looks for another path. So to answer the question, no we do not need to replan always. We only need to replan when the path is not clear i.e. a ghost is on the planned path.

Q. If Agent 3 decides there is no successful path in its projected future, what should it do with that information? Does it guarantee that success is impossible?

Simple answer, no. It does not guarantee that success is impossible. Whenever agent 3 decides there is no successful path in its projected future, it either stays there and lets the ghosts move in the following timesteps or it tends to move away from the nearest ghost.

4) **AGENT 4**

Agent 4 was an open strategy. We decided to fix one problem that we were facing in most of the previous agents, especially agent 2. Though agent 2 replanned its path whenever there was a ghost in the path, it was in vain. In most cases, by the time the agent reached that particular block, the ghost was long gone. Our focus with agent 4 was completely on survivability and not on the time or number of steps it takes the agent to reach the goal state.

Our strategy with agent 4 was to take the immediate step carefully. We started with planning the shortest path but unlike agent two, did not walk it blindly. We analyzed every single step we took with agent 4 and this is how - We checked the neighbors of the potential child and looked for any ghost in or out of the

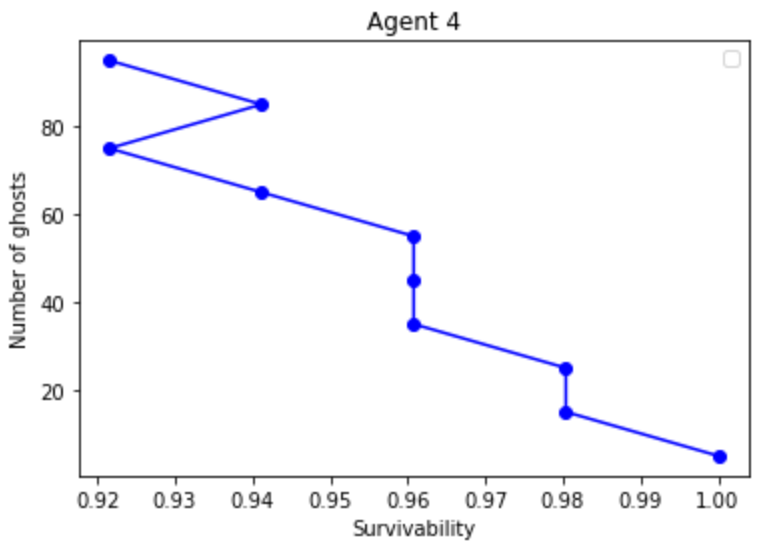
wall. If a ghost was present adjacent to the potential child, we would replan our path, if not, we would continue on that path and repeat the process.

```
(32, 31)
(33, 31)
(34, 31)
(34, 32)
(35, 32)
(35, 33)
(35, 34)
(35, 35)
(35, 36)
(35, 37)
(35, 38)
(35, 39)
(35, 40)
(36, 40)
[(50, 50), (50, 49), (49, 49), (48, 49), (48, 48), (47, 48), (46, 48), (45, 48), (44, 48), (43, 48), (42, 48), (42, 47), (42, 4
6), (41, 46), (40, 46), (40, 45), (40, 44), (40, 43), (40, 42), (40, 41), (39, 41), (38, 41), (38, 40), (37, 40), (36, 40)]
(37, 40)
(38, 40)
(38, 41)
(39, 41)
(40, 41)
(40, 42)
(40, 43)
(40, 44)
(40, 45)
(40, 46)
(41, 46)
(42, 46)
(42, 47)
(42, 48)
```

(this shows a snippet of replanning in agent 4)

By running our algorithm with a different number of ghosts, we came to a very important analysis. Since our agent focusses completely on the ghost positions and not much on the efficiency of the path, the survivability doesn't diminish by much with the increase in the number of ghosts. In very few cases where the ghost concentration is very high around the agent, and it misses the position of a ghost, it dies. But in most cases the agent survives.

Agent 4 was a modification of Agent 2, so at its core, it used BFS for finding the shortest path during planning and replanning.



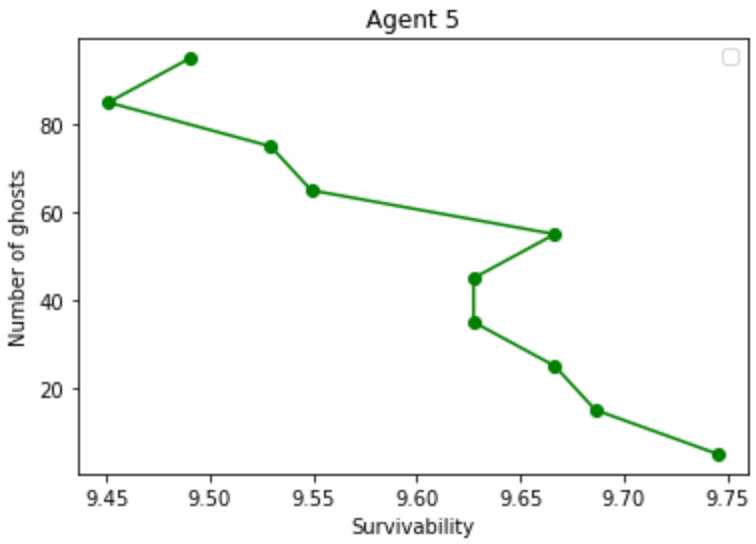
(*For getting the graph we ran the survivability function for each agent and stored them to plot them later)*

5) **Agent 5:**

Agent 5 was very similar to agent 4. The only noticeable difference was agent 5's lack of knowledge of the presence of ghosts in the walls. Though that seemed like a major difference, it didn't make the expected impact. Our analysis of the situation of why it was happening was as follows: As we know agent 4 and 5 both consider immediate neighbors as their priority and the fact whether a ghost is present in the vicinity or not. Why we think it didn't make a big difference is because the chances of a ghost appearing in the wall in an adjacent cell of the immediate step are pretty low.

Also we noticed that since it's so dependent on the ghosts, a few situations might be favorable for the agent even in a high no of ghost situation. But overall this agent came out to be very similar to Agent 4.

Given below is the graph that depicts the trend of survivability for this agen.



(*For getting the graph we ran the survivability function for each agent and stored them to plot them later)*

Q. How did Agents 1, 2, and 3, compare? Was one always better than others, at every number of ghosts?

Agent 1 did not perform very well since it did not take ghost into account and just took the next step.

Agent 2 improved on this and replanned the path when it encountered a ghost in its path. Hence it's survivability increased.

Agent 3 could not improve on agent 2 because the utility we were assigning to each node did not prove to be sufficient to save the agent from the ghost in the near future. Agent 3 also encountered the wiggle problem because of which it could actually not reach the path.

Q. How did your Agent 4 stack up against the others? Why did it succeed, or why did it fail?

Agent 4 behaved sufficiently well in front of other agents since it was prioritizing survivability at each step and not in the planned path in the future.
It was checking for a ghost in each neighbor of its child and then decided the best child to step on.

→ We did not redo the agents 1,2,3 because the implementations we took forward did not involve the survivability of the agent to be measured according to ghost in the walls.

Bottlenecks

→ Bottleneck in Agent 2:

There may be some cases when BFS can't find a path for Agent 2 to run in the current scenario until the position of ghosts changes. If BFS can't find a path for agent 2 to run then Agent 2 tends to move away from the ghost.

→ Bottleneck in Agent 3

The utility provided to each child did not fair well for the Agent as the ghosts were moving at each step and it also encountered the wiggle problem