Project 1: Ghosts in a Maze

Nachiketa Prasad | np916

Anjali Menghwani | am3370

1. Introduction

1. Problem Statement

The maze is represented through a two-dimensional array in the form of cells that can be either blocked or unblocked i.e., traversable. The agent starts at the top-left corner which is (0, 0). The goal of the agent is to reach the bottom right corner i.e.,

(Grid size – 1, Grid size – 1), preferably through the least-cost path.

The Agent is allowed to move in the 4 Cardinal directions (North, East, South, West). It has complete information about the environment at all times and it tries to traverse through the map in the most optimum way possible given its situation.

2. Environment

The environment or the maze is a 2D-integer grid with values ranging from Zero to One and each element of the grid consists of a cell object. The 1s in the grid represent the walls, which have a probability of 0.28 of being generated for each cell.

After generation of such grid/map, it is passed through a validator which checks if there exists a path from the origin to the goal node.

2. Algorithm & Mechanics

2.1. A* Algorithm

We used A* Algorithm to solve our path-finding problem by checking if there exists a path from the Origin to Goal in each map to traverse through each cell.

The heuristic for the algorithm is defined as follows:

f(n) = (size of the path taken till the current node) + (Manhattan distance from current node to goal)

It is a complete and optimal algorithm.

2.2. Manhattan Distance

The Manhattan distance is the norm of distance between any two points in a plane.

In our case, the Manhattan distance is norm of distance between any two cells.

3. <u>Project Design</u>

3.1. Architecture

We have the following packages and classes in our project:

1. Controller
    a. Controller
2. AgentService
    a. IMapTraversal.java
    b. MapTraversal.java
3. AgentUtility
    a. AgentTraversalThread.java
    b. CommonTraversalMethods.java
4. MapDTO

    a. MapReaderDTO.java

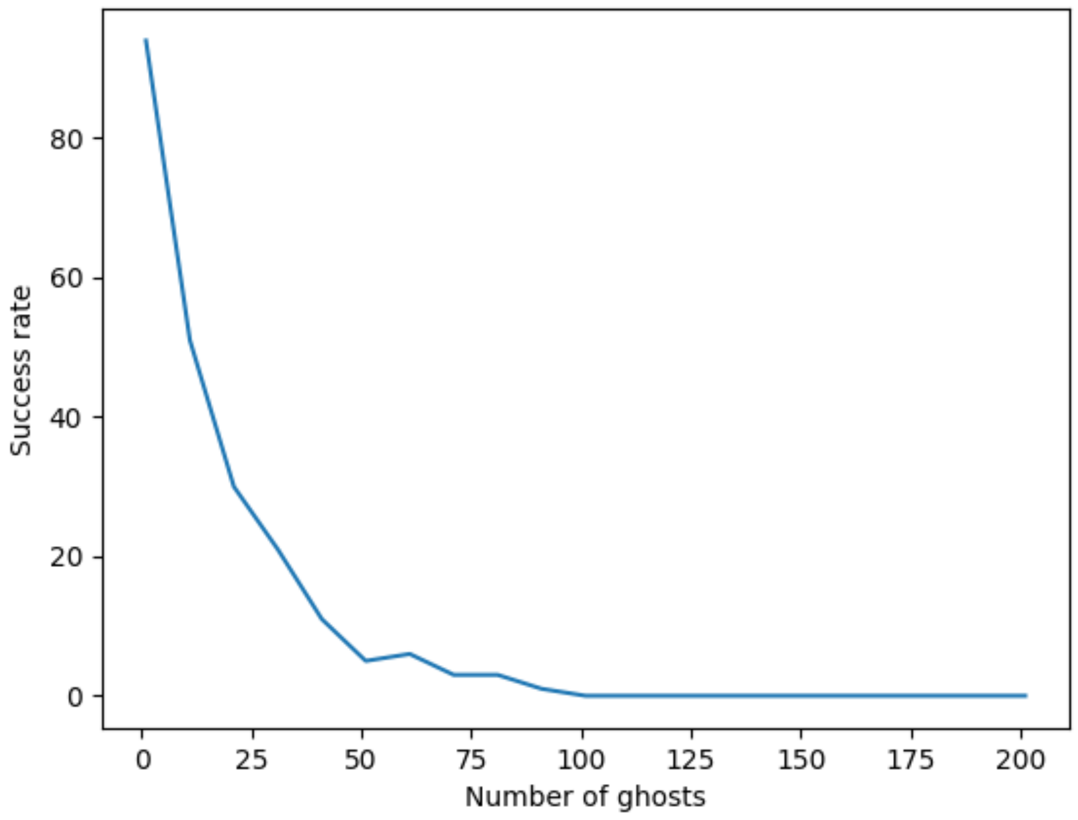
5. MapModel
    a. Coordinates.java
    b. GhostInRange.java
    c. Map.java
    d. Node.java
6. MapService
    a. IMapGenerator.java
    b. MapGenerator.java
    c. MapGeneratorThread.java
7. MapUtility
    a. Agent.java
    b. CommonlyUsedMethos.java
    c. CommonNodeMethods.java
8. ResultantDTO
    a. ResultantData

4. <u>**Implementation**</u>

We have implemented our project in Java as a Maven Project.

4.1. Agent 1

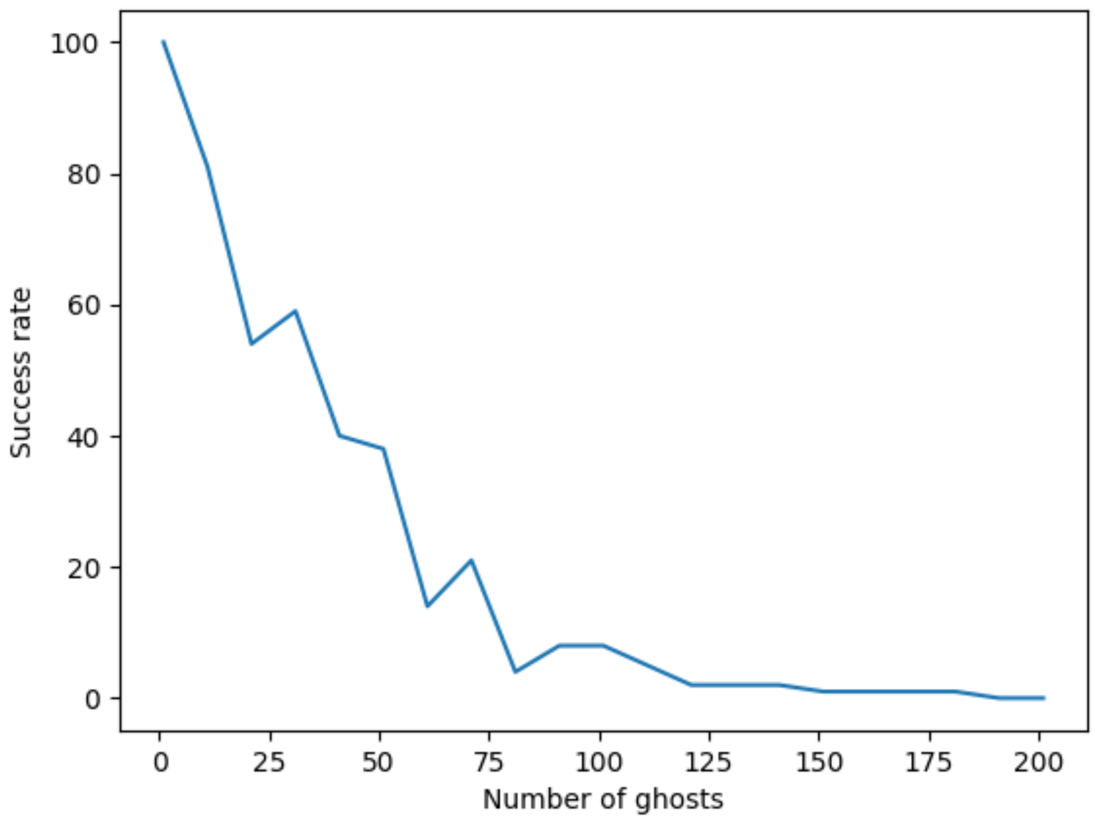
Agent 1 doesn't take the positions of ghosts while planning the path. It simply calculates a path from the start node to the goal node then follows this path.



4.2. Agent 2

Agent 2 first calculates a path from the start node to the goal node and after the first move, if there are any ghosts in the pre-calculated path, it re-calculates a new path without ghosts and then so on. It basically uses A* recursively if a ghost is observed in the pre-calculated path.

In case, any path without ghosts is not possible, it simply moves away from the nearest ghost and then continues its process of recalculating the path as in this case, the path contained just one move of moving away from the nearest ghost.



The performance of Agent 2 is significantly better than Agent 1 because, as expected, it takes steps based on positions of ghosts at any given instance.

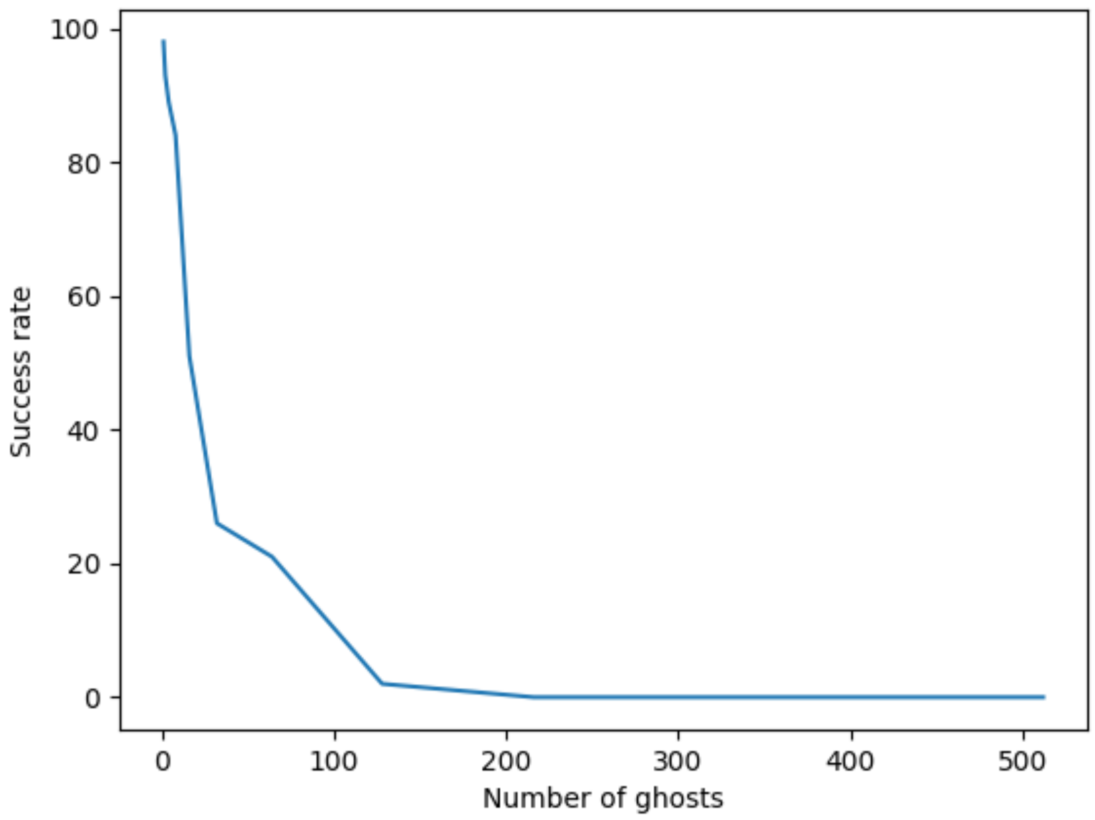
4.3. Agent 3

Agent 3 bootstraps Agent 2 by simulating it for deciding on its next move from the given node at any given instance.

Agent 3 basically simulates Agent 2 for each of the possible directions it can move to. The simulation for each of the directions is done 10 times and it returns the number of times Agent 2 gets success. Based on which direction's simulation of Agent 2 gives the highest number of successes, agent 3 decides its next move. That is to say that agent 3 moves in the direction of maximum number of successes achieved by agent 2.

Now, if the number of successes is same for two or more directions, agent 3 compares between these paths to check which of them corresponds to the direct path to the goal node, then chooses that move.

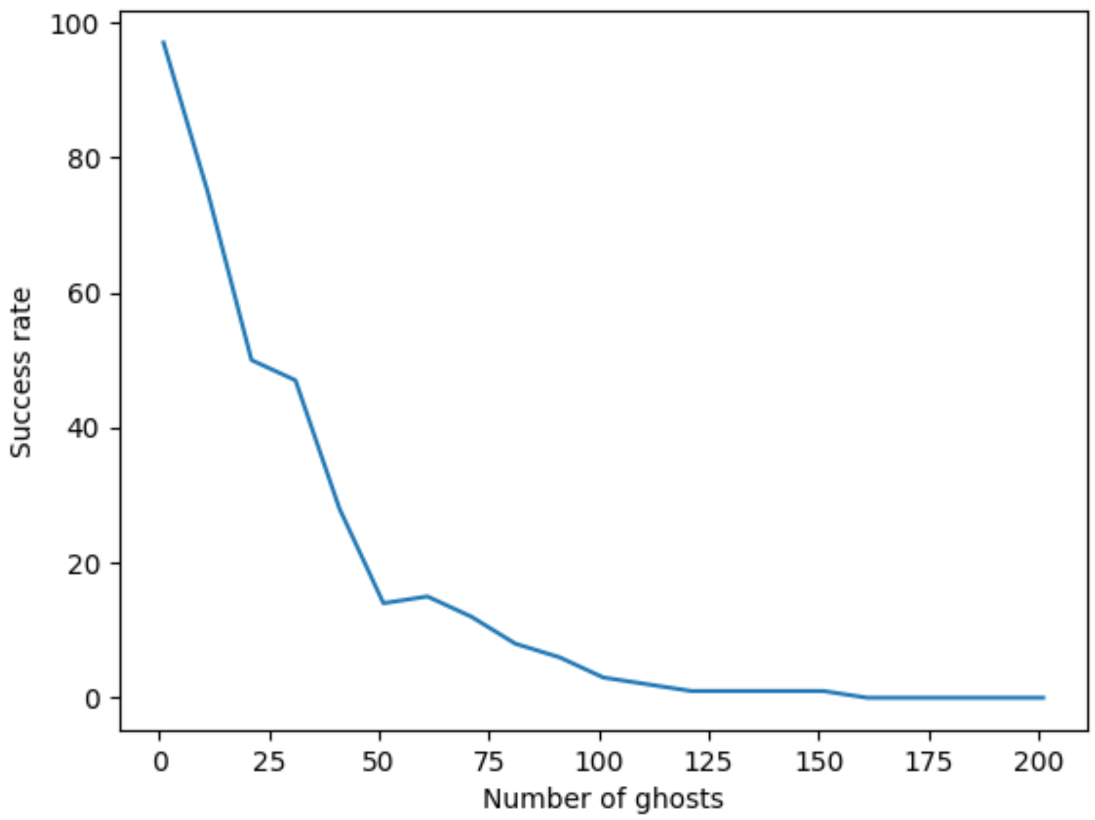
However, if the direct path does not correspond to any of the preferred moves, then agent 3 simply recalculates a new path and follows the entire process again.



4.4. Agent 4

Agent 4 works by first calculating a path from start node to the goal node and then to decide its next move from any node it considers a range which is (1/4) th of the distance from the goal node to its current position.

In the considered range, agent 4 checks for any ghosts in that range which are in its pre-calculated path (in the range). If there are no ghosts in its pre-calculated path, it just continues its path. Otherwise, it re-calculates its path to the goal node. Then for its next move, it simply repeats its process.



5. Computational Challenges and their Handling

As we implemented and executed Agent 3, we observed that it was taking a huge amount of time, as sequential processing wasn't efficient when we were repeatedly re-calculating paths. Therefor, it was a very time-consuming task during data collection, about 5hours.

To overcome this, we used multi-threading which reduced the time significantly to 1.5 hour.

Also, for Agent 3 we generated only 30 mazes instead of 100 to reduce the data collection time.

6. Answers to the gray box questions

- *What algorithm is most useful for checking for the existence of these paths? Why?*
  - ➢ Initially, we started off with implementing DFS for finding the path. It did find the path, however, with the increase in the maze size, the computational time increased.

    - ➢ Therefore, to overcome this limitation, we used the A* algorithm which helped in finding a path with minimal cost in terms of distance.

- *<u>Agent 2 requires multiple searches - you'll want to ensure that your searches are 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?</u>*
    - ➢ To increase the efficiency, before considering a node for exploration, we checked if the future node is blocked and if it contains a ghost. If it does, we do not consider it for exploration.
    - ➢ We do not always need to re-plan, as the ghosts which are far away from us, but in our path (currently), could move away as we move forward.
- *<u>Agent 3 requires multiple searches - you'll want to ensure that your searches are efficient as possible, so they don't take much time. Additionally, 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?</u>*
    - ➢ With each move of the agent, the arrangement of the ghosts also changes. Hereby, at any given instance, if there exists no successful path, we simply move away from the closest ghost. This ensures our survivability in that instance.
    - ➢ However, there is no guarantee of overall success or failure.

- *<u>What possible ways can you improve on the previous agents? Does each planned path really need to be the shortest possible path? How could you factor in distance to ghosts? What do they do well? What do they do poorly? Can you do it better?</u>*
    - ➢ **To improve efficiency**, we used the fact that, at any given instance, the ghosts which are significantly distance from agent's current position didn't matter at the moment. The reason being that the positions of ghosts are not fixed and keep changing with every step an agent takes. Therefore, we re-calculate our path only when we find a ghost only in $(1/4)^{th}$ of the agent's current path to the goal. This way, the efficiency increased as the number of times the path is recalculated is reduced.
    - ➢ In order **to improve survivability**: when looking for a path, we check if there exists a ghost in the agent's distance in unit radius. If it does, then we do not explore upon that node. Here, we kept just one unit radius because at any given instance, our agent is vulnerable to only the immediate ghosts. If we had taken more than one unit radius distance, we'd have further increased the survivability but at the same time the cost would have increased as the agent would have had to cover greater distance (Explore larger number of nodes at every given instance).

- Further, in this way, by reducing the cost and increasing the survivability, we managed to draw a trade-off between intelligence and efficiency

7. Comparative Analysis

When the ghosts the wall are not visible to the agents, the success rate will vary.

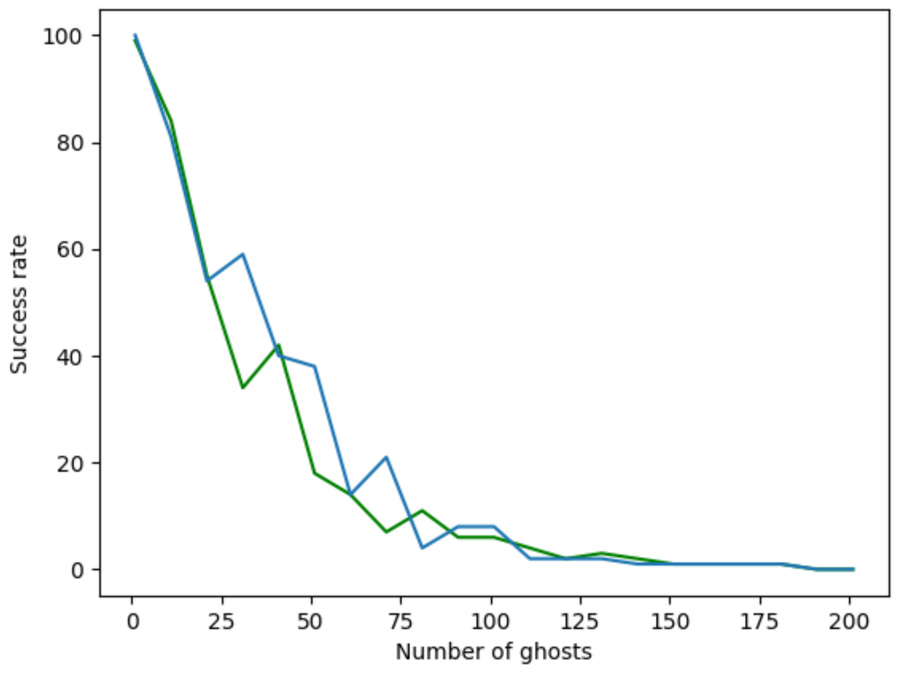
Following are the comparative graphs for Agent 2, Agent 3 and Agent 4 with complete ghosts visibility (blue) and partial ghost visibility (green).

Agent 1

Since Agent one doesn't considers the position of the ghosts throughout the traversal, visibility of ghosts wouldn't change the outcome.

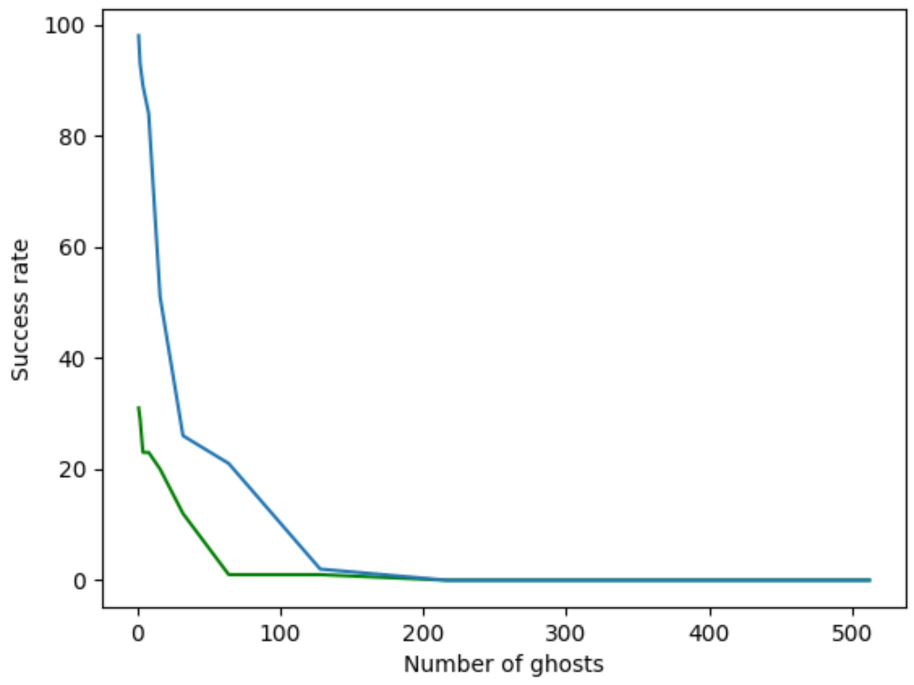
Agent 2

When the number of shots are approximately between 25 to 150, the performance of agent 2 is better with complete visibility of ghosts. However, as the number of ghosts increase, there is no significant variance in the two conditions.



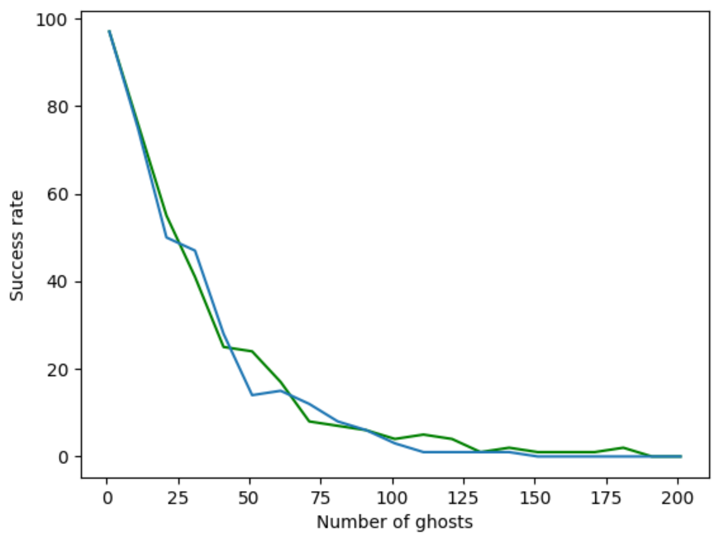
● Agent 3

When the number of ghosts are less than about 125, the performance of agent 3 is significantly better with complete visibility. However, the success rate becomes constant with respect to visibility of ghosts with increase in their numbers.

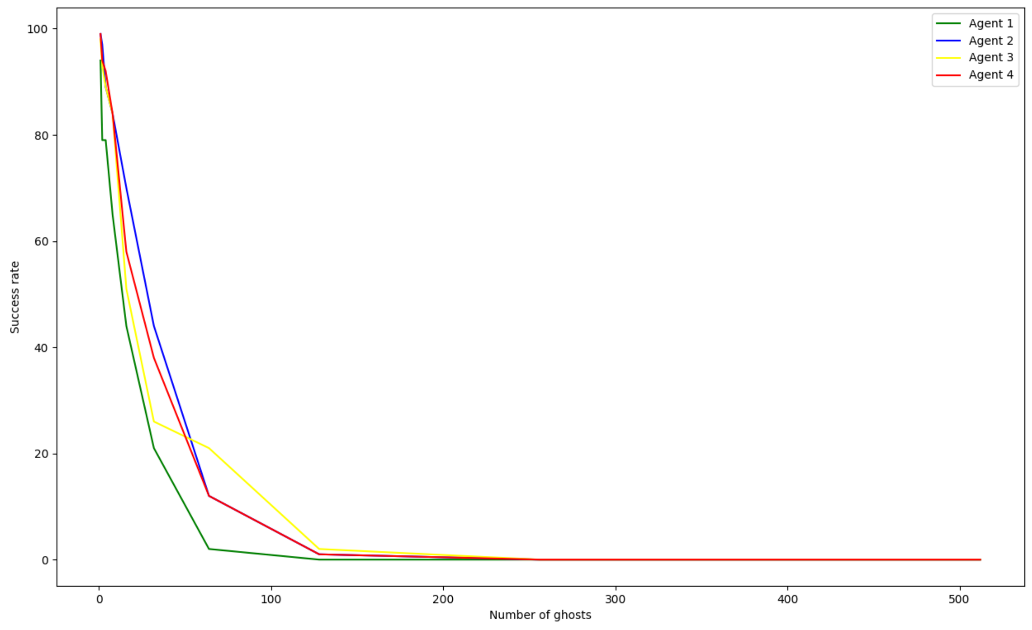


● Agent 4

For, number of ghosts less than 10, the performance of agent 4 is almost same for complete and partial visibility of ghosts but for the larger number of ghosts, surprisingly, the performance is slightly better in case of partial visibility. Also, for the number of ghosts between 25 and 125, roughly, there is no specific trend in the nature of success rate.



Comparative graph for Agent 1, Agent 2, Agent 3 and Agent 4



We can conclude that Agent 2 performs significantly better than Agent 1.

Agent 3, theoretically speaking, could potentially perform better than agent 2 given enough time for simulation and data collection. But here, we have limited the potential of what agent 3 could do by limiting the number of simulations which causes agent 3 to perform relatively poor than agent 2.

We aren’t able to analyze why agent 4 performs worse than agent 2. The reason could potentially be the arrangement of ghosts during initialization. It is highly possible that the position of ghosts was already close to the position of the agent, which made the agent vulnerable and thus dying earlier than expected.