CS 520 - Project 1: Ghosts in the Maze

Akanksha Reguri (ar2085), Khyati Doshi (kbd57)

October 14, 2022

1 Abstract

The problem statement is to design five different types of agents in a maze (51*51 grid) consisting of ghosts. The agent dies if the agent and the ghost are in the same cell. Each agent comes with it's own capabilities and it's goal is to reach the bottom right corner of the maze from the top left corner.

2 The Environment

A 51*51 grid (maze) with blocked cells (numbered as 1) and unblocked cells (numbered as 0) was created. Blocked and unblocked cells are assigned with a probability of 0.28 and 0.72 respectively. The first cell (left top corner) and the last cell (bottom right corner) will always be unblocked.

In order to store neighbors for a cell, say (x, y), storing (x+1, y), (x, y+1), (x-1, y) and (x, y-1) would be both space-consuming and code complexity increases. So we came up with an approach where cells are numbered from 0 to (n*n)-1. Say for a 3*3 grid numbering looks like following:

0 (0,0)	1 (0,1)	2 (0,2)
3 (1,0)	4 (1,1)	5 (1,2)
6 (2,0)	7 (2,1)	8 (2,2)

Figure 1: grid

Storing the above table would again consume space. So we are not storing the above numbering in any data structure. Instead we are extracting them based on cell's row and column based on the formula :
Cell number = cell_row*n + cell_column
n is no of columns here. The above extraction would be of constant time.

In order to store the neighbors, a neighbour data structure was created which would be a dictionary with cell_no as key and value as list of neighboring unblocked cells : cell_no : [neighboring_cells].

Once an environment is created, a Maze Quality check(there exists a path from start cell to end goal) would be performed. If the quality check fails(no path from start cell to end goal), environment would be re-created.
An environment would be created if and only if:

- There exists a path from the start cell to the end goal. Bi-directional BFS(Breadth-first search) has been implemented to find the path.

- There exists a path from every ghost to the top left cell. At the time of spawning of ghosts, only this path would be checked and the ghost in a particular location would be spawned if there exists a path from ghost location to start cell.

An environment consists of the following unique entities:

- Blocked and Unblocked cells in the grid.
- Shortest path from start cell to end goal
- Birth positions of ghosts based on the number of ghosts (we can vary the number of ghosts as per our use case.).
- Neighbor_map data structure consisting of neighbor locations for each and every cell.

Which Algorithm is most useful for checking of these paths. Why?
A simple DFS/BFS should be sufficient to check the existence of path in the maze. Both BFS and DFS have the same run time complexity. But DFS offers good space complexity and BFS provides the optimal path. So it is a trade-off. Shortest/Optimal path is not really needed for checking these paths, so DFS is useful since it offers good space complexity.
But in the design we implemented, we intended to give the path found in the creation of the environment to Agent 1, thus avoiding multiple calls when ran on huge number of mazes. So the shortest path is a requirement for sure. To get the shortest path Bi-directional BFS/Djkstra's are the choices considered and since there is no weight to the edges, Bi-Directional BFS has been chosen. Also, it offers better computational complexity to BFS.

3 Agents

3.1 Agent 1

Design: Agent 1 takes a pretty straight forward approach. The path of the created environment(which was extracted through Bi-Directional BFS) would be given to Agent 1. Agent 1 follows the path ignoring the changes in the environment(movement of the ghosts). If the ghost enters agent_1's cell or if agent_1 enters a cell consisting of ghosts then agent_1 dies.

Challenges faced: None
Survivability: Agent 1 survivability went to zero when the ghost count is 60(Analysis on 100 mazes)
Run Time: Agent 1 runs in milli or micro seconds(Zero seconds).
Analysis: You Lose the game easily if you dont maintain adaptability to the environment.

3.2 Agent 2

Design: Agent 2 takes a fairly straightforward approach but unlike Agent 1, Agent 2 looks through the maze and plans the path to the end goal considering the changes in the environment(movement of ghosts). At every timestamp before taking the next step, Agent 2 would re-plan the path from it's current position to the end goal. The A* Search algorithm is used for path-planning. A* search algorithm would run removing ghost locations from the neighbor list values of any other cell. This is equivalent to considering ghost location as blocked cell, so that agent won't plan a path containing a ghost cell. If there comes a scenario where there exists no path from the current state to the end goal, Agent 2 tries to move away from the nearest visible ghost(ghost not in walls). Utilized A* search to find the nearest visible ghost, since it would give the shortest path between ghost and agent. Agent 2 moves to one of it's neighbors (which is farther from the nearest ghost than from it's current position). If moving to any of it's neighbor locations is moving towards the ghost, then agent stays in the current state only.

Challenges faced:

- What would be the nearest visible ghost? - Ghost nearer in cell position but farther in distance or nearer in the distance even though it is farther in cell position.
  The nearest visible ghost is the ghost which is the first visible ghost to the Agent when it looks

through the maze. So it would be the one which is nearer in the distance even though it is farther in cell position. (Prof. Cowan gave a clarity on this).

- What does the agent do if all of it's moves(neighboring states), it can choose are moving towards the nearest ghost? Agent chose to stay in the same place. Agent takes a move if and only if the neighboring states are better than it's current state in terms of distance from the nearest visible ghost, that is neighboring state distance to nearest ghost should be at least x+1, if current state distance is x in order for agent to chose them. This distance is calculated through A* search again, since we are considering the nearest visible ghost as one that is nearer in distance.

Survivability: Agent 2 survivability went to zero when the ghost count is 100(Analysis on 100 mazes).
Run Time: Agent 2 takes a minimum run time of 0 sec and maximum run time of 20 sec. This maximum run time is seen in the scenarios where agent was escaping (moving away) from the nearest visible ghost. The average run time for each simulation was around 13 sec(Analysis on 100 mazes)
Analysis: Agent 2 with it's smart re-planning behavior was dying(survivability to zero) at 100 ghosts while the agent 1 was dying at 60 only.
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?
To make the searches as efficient as possible A* search algorithm has been implemented since for an admissible and consistent heuristic, A* searches only those nodes where priority(node) is less than true minimal cost.
No there is no need to re-plan always. At timestamp t, Agent 2 can go ahead with the old plan if there are no ghosts blocking it's path. The new plan would be the same as the old plan when there are no ghosts blocking the old plan. If the ghosts block the old plan, then Agent 2 needs to re-calculate. Below Optimised version of Agent 2 explains more about it.

3.3 Agent 2 Optimised

This is an optimized version of Agent 2. This optimized version doesn't re-plan the path at every timestamp. Before re-planning the path it checks if there came any ghosts in it's already planned path. If any ghost is blocking the path to the end goal, it re-plans the path, else it continues with the same path. The reason for this optimization is that if say if Agent 2 re-plans at every timestamp, the new plan would be the same old plan in case of no ghosts blocking the old plan. Except for this path re-planning strategy, everything else is similar to Agent 2.
Survivability: Agent 2 survivability went to zero when the ghost count is 110(Analysis on 100 mazes)
Run Time: Agent 2 Optimised takes a minimum run time of 0 sec and maximum run time of 16 sec. This maximum run time is seen in the scenarios where agent was escaping (moving away) from the nearest visible ghost. The average run time for each simulation was around 8 sec(Analysis on 100 mazes)
Other Performance metrics:

- NoOfReplans - How the number of path re-plans varied with the number of ghosts. How did it affect survivability?

  As the ghost number in the gird increase we could observe that the re-plans required by the agent also increased. This is logical because more the number of ghosts, more will the path of the Agent be blocked and thus Agent will have to re-plan its path multiple times. This behavior is irrespective of the Agent reaching the end goal alive or dying somewhere in between.

- NoOfmoveAwayFromVisibleGhost - How many times did the agent move away from the nearest visible ghost and how did the count vary with the number of ghosts. How did this affect survivability?

  The data collected for number of times the agent moved away from the nearest visible ghost when

  1. **agent is alive**: This data didn't show any trend. This seems credible because as number of ghosts increases, number of path re-plans increases and at every re-plan agent plans a path

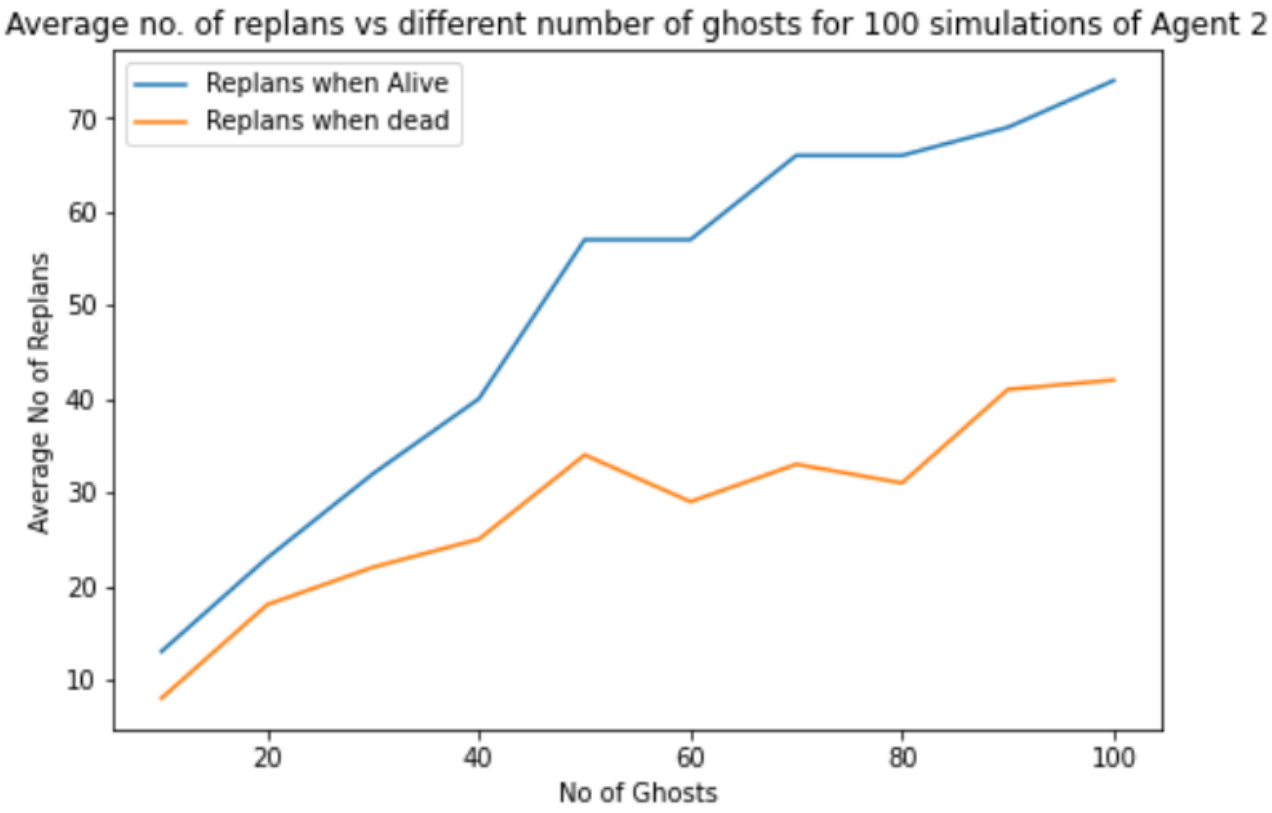


Figure 2: Re-plans vs Number of Ghost

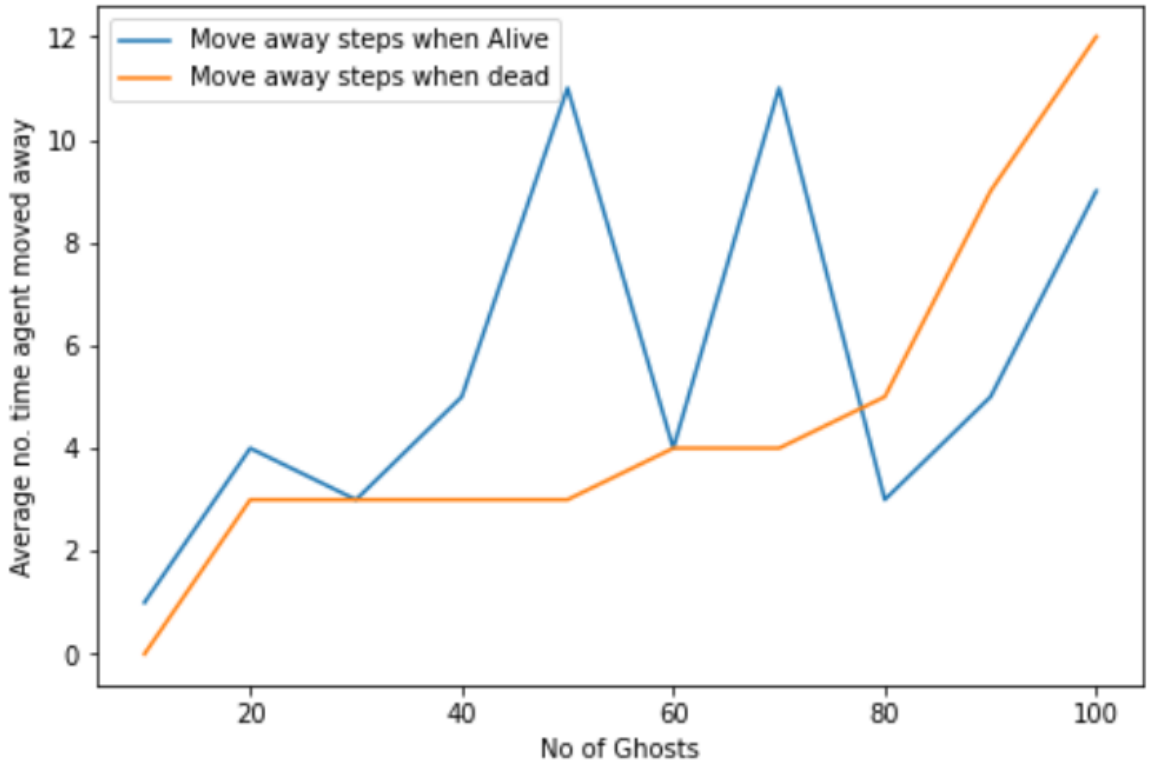


Figure 3: Avg no. of times agent moved away vs different number of ghosts for 100 simulations of Agent 2 optimised

in such a way that is not having any ghosts. Thus the probability of an agent not finding any path to end goal might decrease and even though if agent encounters a scenario where there is no path to the end goal, agent moves away from the nearest visible ghost and if the agent is alive, that means it had found path to goal node in sometime. Thus number of move always is dependent on ghost locations which is randomised and we can't conclude anything on this.

2. **agent is dead**: Here we see as number of ghosts increases, number of move aways increases. As number of ghosts increases path re-plans increases but here the agent is dead that means it's re-planning strategy is not winning over the distribution of ghosts and agent always need to run away from the nearest visible ghost to save its life. Thus it seems logical that in case of dead scenarios, agent moves away from the nearest visible ghost more and more trying to save it's life but is being dead at the end since it's re-planning strategy is failing due to distribution of ghosts.

3.4 Agent 3

Design: Agent 3's approach isn't straightforward. Agent 3 simulates agent 2 from each of the possible moves it can take including it's current position. Based on the success rates(Number of times Agent 2 is alive), Agent 3 decides on the move that has the highest success rate. In case of multiple moves having the same success rate, Agent 3 goes ahead with the goal oriented behaviour(moving in the direction of the goal).
Challenges faced:

- **Recursion Problem**: Initially, Agent 1 and 2 have been implemented in recursion, instead of iteration and it was working well. Hence followed the same recursion approach for Agent 3 as well, but Agent 3 broke saying maximum recursion depth reached. Then to maintain the code uniformity each of the agent was implemented back in the iterative approach. With an iterative approach, Agent 3 worked well.

- **Agent 3 performing worse than Agent 2**: My Agent 3 was dying in the scenarios where Agent 2 was alive. After debugging to the core, we found a few scenarios where in case of no path to the end goal, Agent 2 was moving away from the nearest visible ghost, waiting for the ghosts to clear the path, re-planning the path, and giving a good success rate for that move. Agent 3 was choosing one of such moves and was ultimately being killed by the ghost in no-time.
  Possible solutions: When there is an ambiguity if Agent 3 should also move away from the nearest visible ghost? But that wasn't mentioned in the problem. (or) Shouldn't we not consider the success rate of Agent 2 when it was moving away from the end goal?(This may not work because there could be potential paths which may move away from end goal to get to the end goal at the end).
  Final Resolution: Dr.Cowan clarified saying it is a corner case and Agent 3 also should move away from the nearest visible ghost in case of no path to the end goal.

- **Wiggle Problem**: Wiggle problem is a scenario where Agent is continuously struck between 2 states and is not coming out of that infinite loop. Wiggle Problem occurs when Agent 3 at timestamp t assumed that the right neighbor is better of all it's neighboring states and at the timestamp t+1 assumed that the left neighbor was better. (valid for up and down cells as well instead of left and right)
  Debugging leads: Wiggle problem isn't caused because of one unique or particular scenario. Below mentioned attributes helped in making progress over the wiggle problem.

  1. **Number of times Agent 2 is simulated**: This hyper parameter played a vital role in affecting the time for which Agent 3 was wiggling. Initially when testing with three simulations, success rates weren't being captured accurately, and hence wiggling was huge. By increasing the simulations to seven, the amount of wiggle(the number of times agent 3 would wiggle) was reduced. With 5 simulations as well, wiggling was less.
  2. **Agent 3 moving away from ghost**: One solution of agent 3 moving away from the nearest visible ghost in case of no path scenario fixed the wiggle problem of agent 3 as well.

  It is not that the wiggle problem got eradicated completely, Agent 3 would still wiggle a little here and there(completely possible based on the location of ghosts), but at the end it would resolve it's path to the goal. With the above fixes, the amount of wiggle time of agent 3 was reduced significantly.
  Initial wiggle time: 30 mins
  Now wiggle time: 5 mins

Survivability: Agent 3 survivability went to zero when the ghost count is 120(Analysis on 50 mazes).
Run Time: Agent 3's average run time is 25 mins per simulation (Analysis on 50 mazes)
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?
To make the searches efficient as possible, A* search algorithm(being used in Agent 2) implemented was used here as well. If there is no successful path in the projected future, Agent 3 mimics Agent 2 and moves away from the nearest visible ghost. It doesn't guarantee that success is impossible. Agent 3 will keep moving away from the nearest visible ghost, saving it's life till there is some path in the projected future. Once the path is found, Agent 3 then try moving towards the goal again. So there is a probability that Agent 3 might still be successful.

3.5 Agent 4

Design: Agent 4 works in a slightly complex way than other agents. Just like Agent 3, Agent 4 also simulates agent 2 from each of the possible moves it can take including it's current position. Additionally to the success rates(Number of times Agent 2 is alive), Agent 4 would also calculate the safety distance - the proximity of each ghost to the path that the respective move decides on. Based on this Agent 4 selects the move, whose path leading to the end goal is having the highest average safety distance from the ghost and higher survivability rate of agent 2. In case where there are multiple

choices with the same success rate and ghost proximity, Agent 4 makes a choice of move with goal oriented behaviour(moving in the direction of the goal).

Safety_distance = $\sum$(min(dist(ghost_location, path))) for all ghosts / Number of ghosts

dist(ghost-location, path) = min manhattan distance(ghost-location, path-cell) for every cell in the path

Proximity of the ghosts to the path: Proximity of the ghosts to the path, is calculated based on the Manhattan distance of each ghost to the nearest cell on the path that the agent selected. The farther the ghosts are from the agent's path, on an average more is the path preferred.

Challenges faced:

- **Multiple strategies for agent 4**:
  We had 2 other strategies for Agent 4 which we were considering:
  - **Strategy 1**: For each cell,the probability of a ghost acquiring the cell is calculated. Based on the probability for each cell we trace the path available to the goal state. With this approach we faced Wiggle problem and the Agent was stuck in between 3 to 4 block for long time in few scenarios, and in few other scenarios Agent 4 was not winning over agent 3. So we scaled up the idea of calculating the proximity of ghosts to the cell to the entire path and that worked well.
  - **Strategy 2**: For all the choices we simulate from agent 2, instead of just depending upon the number of time Agent 2 survived on the path we also take into consideration how many time it re-planned the path. Based on this the Agent 4 moves on the path which has fewer re-plans and maximum times agent 2 survived on the path. This approach failed as the performance of Agent 2 and 3 was better as compared to Agent 4.
- **Issues with the current implementation**: In cases where the ghost count was increased to 130 and more, Agent roams around a few cells for about 25 minutes. We discussed this with prof. Cowan and decided that in this case the maze should be considered unsolvable as the Agent neither dies nor reaches the goal state. In these scenarios, Agent 3 to 1 were dying. Thus Agent 4 is better than rest of the agents.

Survivability: Agent 4 survivability went to zero when the ghost count is 130(Analysis on 50 mazes).
Run Time: Agent 4's average run time is 30 mins per simulation (Analysis on 50 mazes)

3.6 Agent 5

This agent loses sight of ghosts when they are in the walls. This affects the performance of the agents in the following way:

- **Agent 1**: This has no effect on Agent 1, since it doesn't care about the environment and blindly follows the path.
- **Agent 2**: Losing sight of ghosts in the walls doesn't affect Agent 2 in any way. Two reasons are:
  - **Re-Planning**: Ghosts in the walls are always obstacles to Agent 2 because of the walls. Even though agent 2 loses sight of ghosts in the walls, it's re-planning behaviour wont get affected because it would still consider the walls as obstacles and will plan accordingly
  - **Run away**: Agent 2 moves away from the nearest visible ghost. So even though ghosts in walls are not visible, this running away behavior is not going to change.
- **Agent 3**: Agent 3 simulates agent 2 and makes the decision based on the success rates of Agent 2. Agent 2 is not being affected, hence there is no affect on agent 3 planning as well. Agent 3 also moves away from the nearest visible ghost(similar to agent 2) in case of no path to the end goal.

- **Agent 4**: Agent 4 simulates agent 2 and makes the decision by calculating survivability of each of those moves. Agent 2 is not being affected, hence there is no affect on getting alive rates of Agent 2 for each of those cells. But when calculating safety distance (proximity of ghosts to each of the cells on the path), now ghosts in the walls shouldn't be considered.

Design: Agent 5 is same as agent 4 except that it doesn't have the ability to see ghosts when they are in the walls. So safety distance is calculated based on ghosts in unblocked cells only. Now because of the loss of the information, agent 5's path survivability would be relatively less than or equal to that of Agent 4. **Survivability**: Agent 5 survivability went to zero when the ghost count is 100(Analysis on 5 mazes).
Run Time: Agent 5's average run time is 20 mins per simulation (Analysis on 5 mazes)

4 Implementation

4.1 Code Organization

This entire project has been coded in python. The list of python files includes the following:

4.1.1 The Environment

environment.py consists of The_Environment class which includes the following variables:

- maze - 51*51 numpy matrix with values : 0,1.
- path - python list which stores the shortest path from start cell to the end goal. This path is fetched through bi-directional BFS.
- neighbor_data - python dictionary with key as cell numbers and value being list of neighboring states of that particular cell.
- ghost_locs - list of cells where ghosts exist.
- ghost_count - Number of ghosts, which is an optional argument to the class and can be changed based on the requirements.

python environment.py would create the great environment including the above entities which are unique to that particular environment instance. Environment would be created if and only if there exists a path to the end goal(Maze Quality check). If Maze quality check fails, create_environment function would be called recursively and this recursion stops only when there exists a path to the end goal.

4.1.2 The Agents

agent.py consists of the Agent class which is inherited from The_Environment class. This class includes functions specific for each and every agent. (Agent-1 to Agent-5).

- agent_1: This function implements Agent 1. This function follows a simple iterative approach of following the path (fetched through bi-directional BFS). Returns True if the agent is alive, else would return false if the agent died. Agent would die if ghost entered agent's cell or if agent entered ghost's cell.
  python agent.py –agent_1
- agent_2: This function implements Agent 2. It is implemented according to the design of Agent 2 discussed above. It re-plans the path at every timestamp using the A* search algorithm. In case of no path to the end goal, Agent 2 moves away from the nearest visible ghost. The nearest visible ghost is the one which is the ghost at minimum distance(This distance is calculated using A* search with current location as start cell and ghost location as end cell). Apart from returning the boolean which indicates if the agent is alive or not. The other data gathered here would be: No of times the agent moved away from the ghost and the Number of times Agent re-planned the path to understand the behavior of Agent 2.

- agent_2_optimised: This function implements Agent 2 with the optimization that path would be re-planned only in the scenarios where ghosts are blocking the planned path.
  python agent.py –agent_2

- agent_3: This function implements Agent 3 according to the design discussed above. At every timestamp Agent 3 simulates Agent 2. The number of times Agent 2 simulated can be varied as part of the function call, the default value is 5. Agent 3 simulates Agent 2 from each of the possible states it can take including it's current state. Based on Agent 2's success, assigns success rates to each of those moves and chooses the move with highest success rate(survivability rate). If the success rates of 2 or moves are same, then it chooses the move which is heading towards the goal(goal-oriented behaviour). In case if success rate of each and every move is zero, then Agent 3 continues in the state since there is no probability of success from any perspective.
  python agent.py –agent_3

- agent_4: Agent 4 implementation is similar to agent 3. It is like an enhanced version of agent 3. It simulates agent 2 from each of the possible states it can take including it's current state. The way agent 4 assigns success rates is little different from that of agent 3. Survival is the main goal for Agent 4. So along with success rate of Agent 2, we make survival oriented choice based on the location of ghosts. This survival oriented choice is decided based on safety distance metric. Safety distance of a path is the average distance of the path from all of the ghosts present in the maze calculated based on the formula mentioned above. The higher the safety distance, there is a high probability for the agent to live long on that path
  python agent.py –agent_4

- agent_5: Agent 1,2 and 3 won't be affected by losing the sight of visible ghosts as mentioned in the design. But agent 4 would be affected when calculating the safety distance. Now Agent 5(Agent 4 losing sight of ghosts in the walls) won't be able to calculate the safety distance of the path from the ghosts in the walls, and this will affect the agent's choice. A function to filter out ghost locations was written, it filter's out all the location of ghosts that are unblocked. Any ghost on the block cell are not included in the result. The result of the filter is considered for calculating safety distance of the path from the agents.
  python agent.py –agent_5

4.1.3 Algorithms

The algorithms implemented are:

- Bi-Directional BFS: This is implemented using queues, written in search_algos.py and the parameters include start cell, the end goal, and neighbor data structure. Given the start, end cells and the neighboring data information, this function would be successful in finding an optimal shortest path between the two cells.

- A* Search Algorithm: This is implemented using hashsets, written in a_star_search_algo.py and the parameters include start cell, the end goal, and neighbor data structure. Given the start, end cells, and the neighboring data information, this function would be successful in finding an efficient shortest path between the two cells. The other parameter includes ghost_locs which is an optional argument and would be passed in case of scenario where ghost coordinates would be treated as blocked cells (In the case of Agent 2).

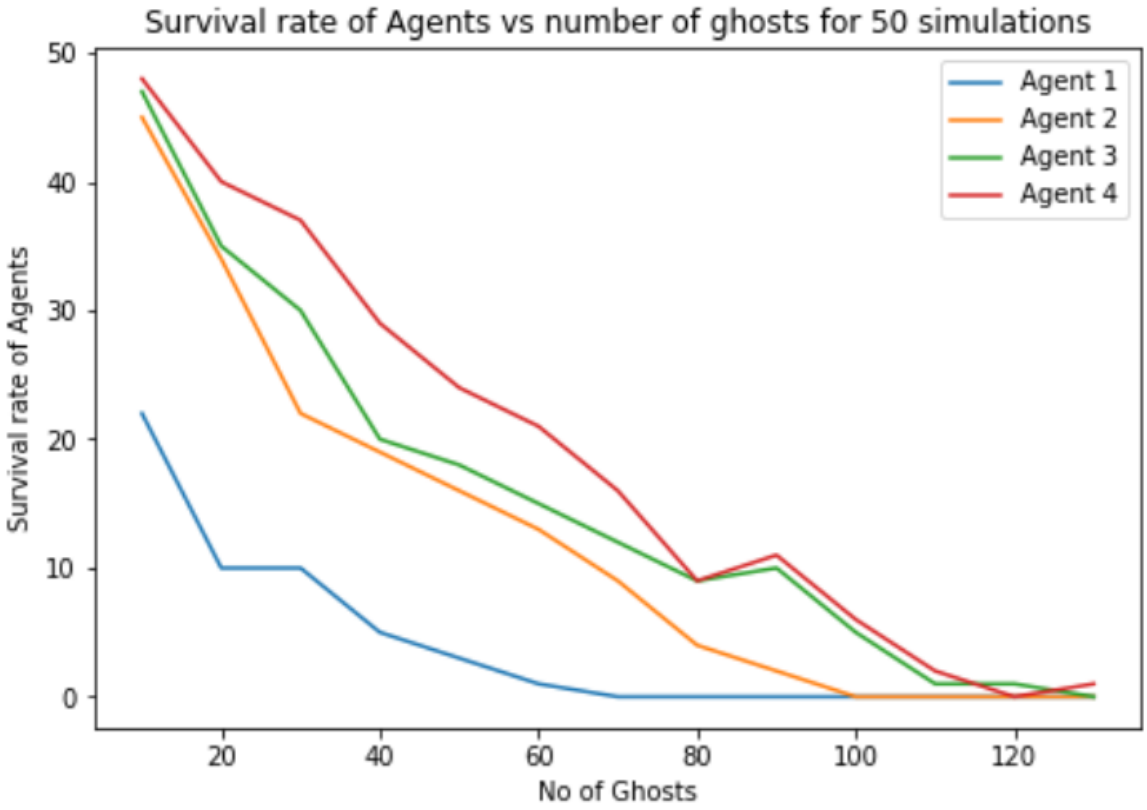
4.1.4 Utils

The other utilities for the project include:

- agent_performance.py: This python file captures the agent's performance and stores it in the csv files

- utils_.py: This file includes other utility functions required for working of other files. The functions are like nearest visible ghost, choose goal oriented choice, choose survival oriented choice etc...

4.2 Results



How did Agents 1, 2, and 3, compare? Was one always better than others, at every number of ghosts?: Yes Agent 3 is always better than Agent 2 followed by 1.

How did your Agent 4 stack up against the others? Why did it succeed, or why did it fail?: Agent 4 succeeded in beating the rest of the agents. Agent 3 was always better than Agent 2. So now only goal is to design the agent 4 in such a way that it beats Agent 3. Clearly Agent 4 is an enhanced and intelligent version of agent 3 since it is calculating success rates based on agent 2's alive status and safety distance(proximity of ghosts to the path).

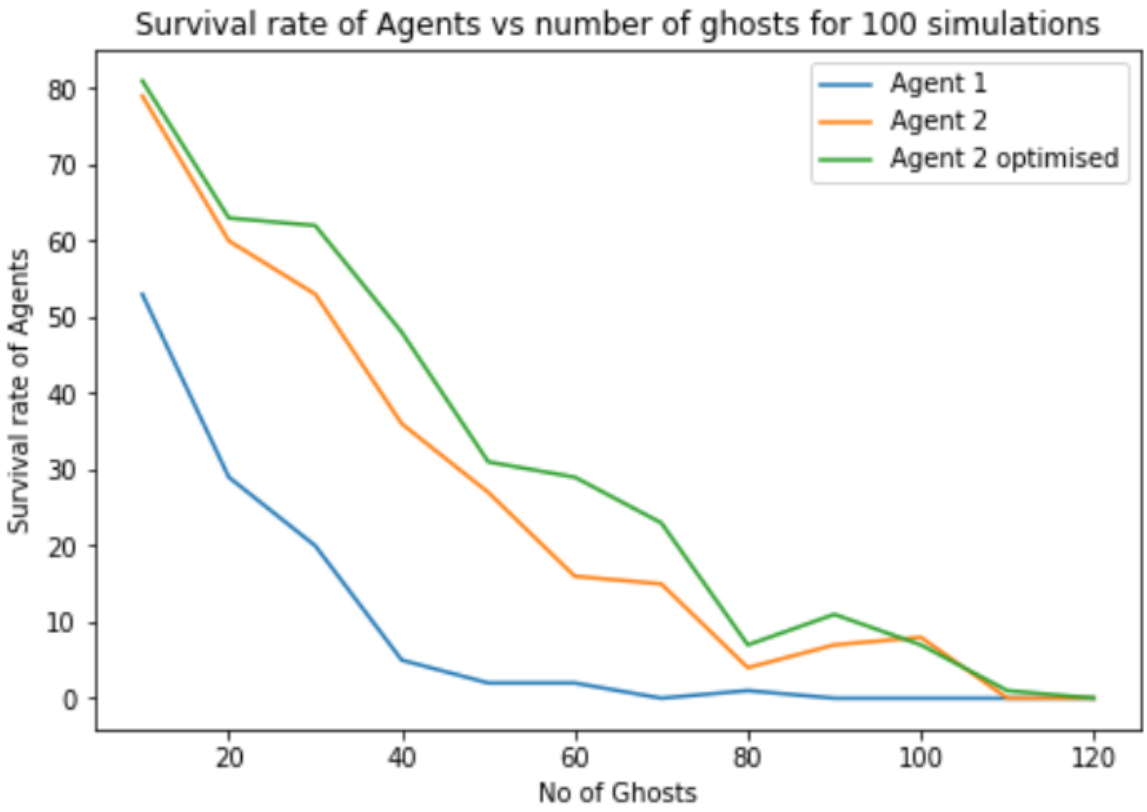
Computational bottlenecks and Challenges: Since Agents 3 and 4 are time-consuming we couldn't run them for 100 mazes. Instead, we ran them for 50 mazes, by splitting them (split by ghost count and the number of mazes) across various google colab notebooks.

A statistical chance: There are few scenarios seen where Agent 2 survived say X times for Y number of ghosts and X+e(e varied from 1 to 3) times for Y+1 number of ghosts which is not expected since we expect survivability to reduce with number of ghosts. But there is a luck factor depending on the position of ghosts for an agent to win more times with higher number of ghosts. Again in these scenarios(99.99 percent times) Agent I+1 is beating the Agent I.

4.3 Summary

Finally our Agent 4 is better than Agent 3 followed by Agent 2 and 1.

- Agent 1, (the least intelligent one), it's survivability" went to zero when there are 60 ghosts in the maze.
- Agent 2 being a little more considerate of the environment survived until 100 ghosts in the maze.
- Agent 3 which simulated Agent 2, it's survivability went to zero when ghost count is 120



- Agent 4 survival rate went to 0 when there were more than 130 ghosts.