520 : GHOSTS IN THE MACHINE

A Project Report by:
- Lokesh Kodavati (bk576)
- Saransh Sharma (ss4368)

LINK TO COLAB NOTEBOOK [Scrollable Outputs]

Environment Setup

We have used the **NumPy** library to make the *grid(skeleton)* for our maze.
We have used the **Random** library to implement probabilistic logics for spawning of walls and ghosts.
We use different numbers as entries in the grid to denote different entities (OPEN=0, WALL=1, GHOST=2, PATH=7) and use mathematical operations to update and manipulate them.

Design Choices

We have taken an Object Oriented Approach using Python (v3.7) for our implementation of the given task, this OOP approach helps us in easy refactoring of the code if required and also makes adding attributes, properties and functions easier down the line. The code is also highly functional and as such consists of a number of functions which improve ease of implementation and readability.

There is 1 major class that has been implemented, namely the ***"Maze"*** Class which contains all the base attributes of the maze like its dimensions, wall spawning logic and logic that gives it a square *shape* for easy visualisation and understanding & it provides the maze object itself with an initial path to follow.

- It also contains various helper functions for:

- Checking validity of the maze
- Checking validity of moves/child nodes
- Checking if we died
- Checking if we reached goal
- Spawning ghosts in the maze
- Moving the spawned ghosts
- Getting location of cells
- Cloning the maze in its current state
- Maze Visualisation [White=Open, Black=Walls, Yellow=Ghost(s), Red=Ghost(s) in wall, Purple=Path Taken, Orange=Backtracking]

Algorithms Used

- **Depth First Search**
  - Traverses from the starting and visits nodes going away depth-wise from the root node and when it exhausts depth search for a particular node, it backtracks to the most recent node with unexplored child nodes and repeats the process.
  - The data structure that we use for the *Fringe* in this case is a **Stack** as we want to explore newer nodes first and stack is a **LIFO** (Last In First Out) structure.
  - It generally has a smaller memory footprint than BFS.
  - If a path exists, DFS does not guarantee to find the shortest/most optimal path, but it guarantees that the path that it does find, it does in lesser time.
  - Generally useful where we want to find ***a*** solution and in less amount of time.

- **Breadth First Search**
  - Traverses nodes at equal depth from a parent node before moving forward in depth (level by level) and in this way, at some time, the fringe will contain all nodes that are equidistant from the root and popping these nodes, at some time the fringe will contain all nodes 1 level down.

- The data structure that we use for the *Fringe* in this case is a **Queue** as we want to explore the oldest nodes first and it's a **FIFO** (First In First Out) structure.
- It generally has a bigger memory footprint than DFS.
- If a path exists, and given BFS' property to explore all nodes on the same level first before going deeper, it guarantees that the path it finds is the **optimal/shortest** one.

- **A* Search**
  - Uses **Priority Queue** as the *Fringe* with its priority being a function of the actual cost till that point and the estimated cost till the goal node.
  - The estimated cost till the goal node is called the Heuristic (e.g. It can be as simple as the city block distance between 2 points also known as the Manhattan Distance)
  - We take

    $$\boldsymbol{F(n) = G(n) + H(n)}$$

    {Where G is the actual cost and H is the heuristic cost or the estimated cost}

  - A* with an Optimistic/Admissible heuristic always gives an optimal path
  - An Optimistic Heuristic (H) is one that follows:

    $$\boldsymbol{H(n) \leq C*(n, G)}$$

    {Where C* is the true minimal cost}

  - It is more memory efficient than BFS because it expands lesser nodes (as it has an estimate guiding it or biasing it towards the goal, essentially enabling it to prune out paths with less likelihood of success if followed.
  - Useful and replaceable with BFS wherever it is possible to direct the search towards something (i.e. a *Goal*)

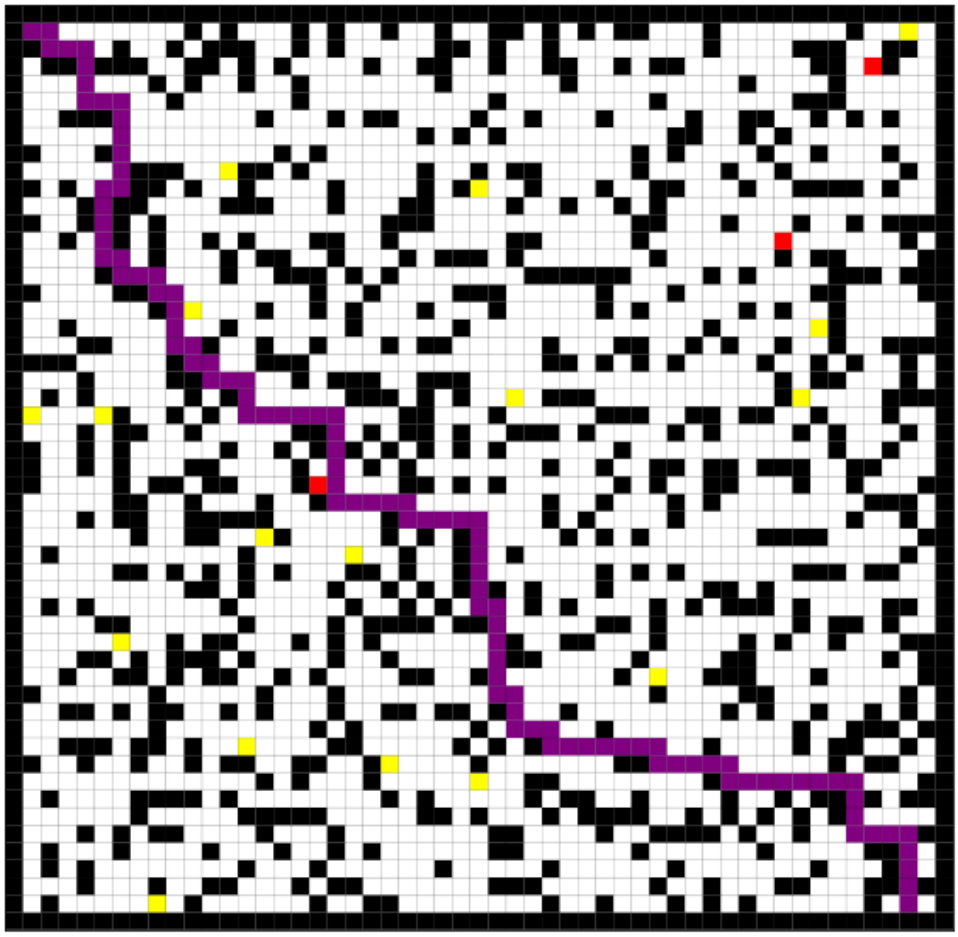
Agents

Agents are our way of solving the maze via use of search algorithms and adding extra *intelligence* on top of those.

“ The agent is going to start in the upper left corner, and attempt to navigate to the lower right corner. The agent can move in the cardinal directions (up/down/left/right), but only between unblocked squares, and cannot move outside the 51x51 grid. At any time, the agent can ‘see’ the entirety of the maze, and use this information to plan a path. ”

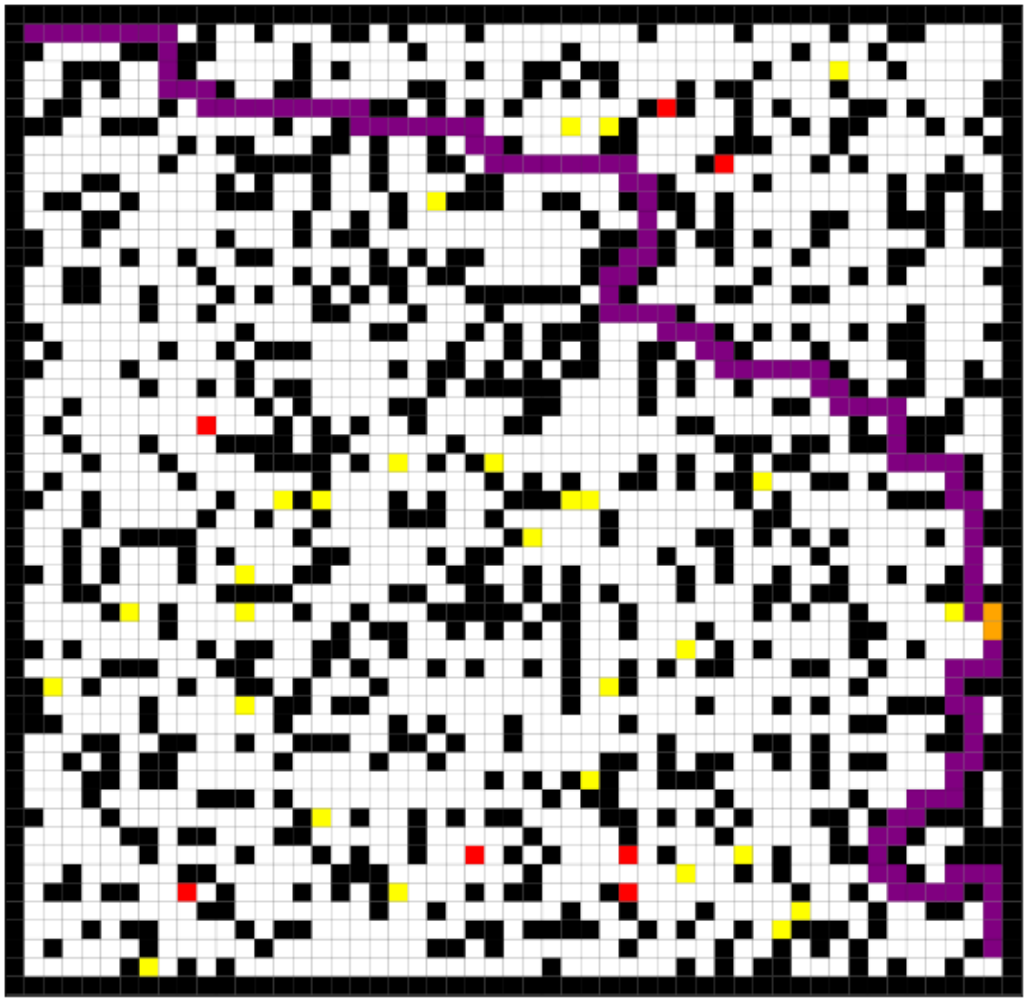
AGENT-1

- For Agent-1 we simply used our implementation of BFS to get(*plan*) the shortest path and tried to run on it (*execute*)
- During the planning, we only account for the walls and not the ghosts and so once we start following the given path, there is a high likelihood that the agent dies because a ghost came in its path.



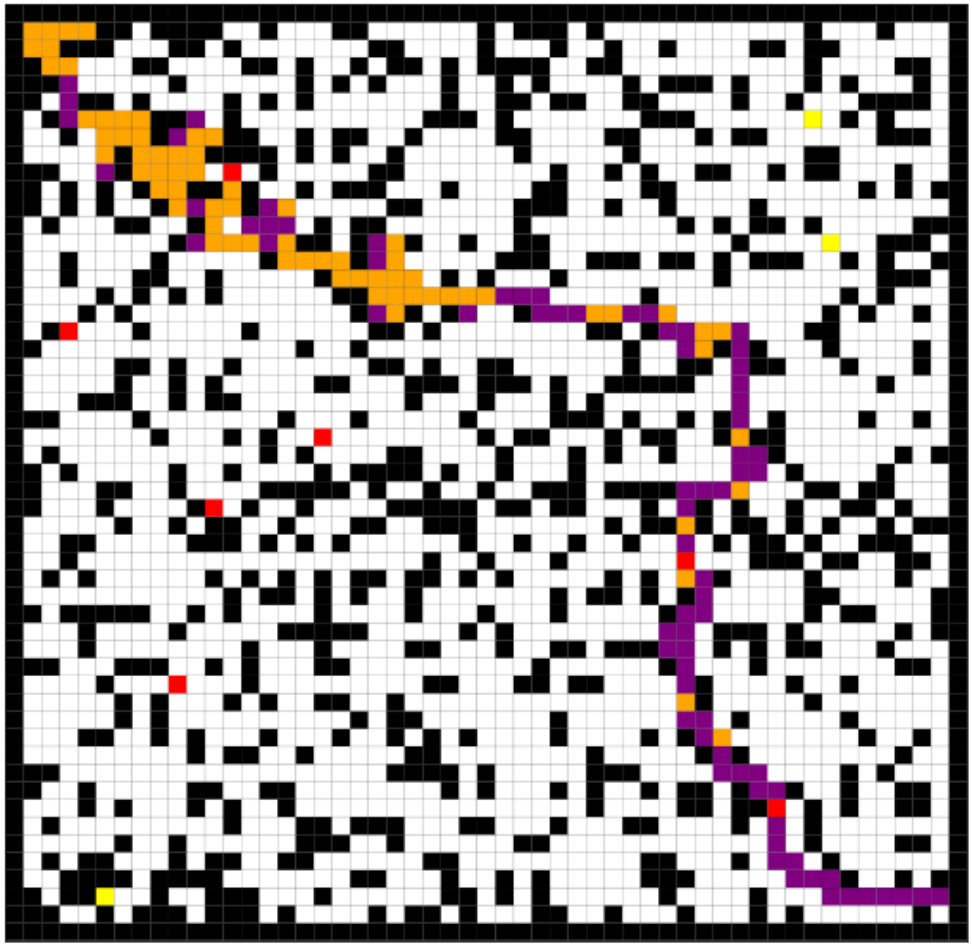
AGENT-2

- We pass an initial path to the agent that takes into account the maze (Grid with walls).
- Agent-2 surveys the path (since agents are all knowing), and if it doesn't see any ghosts on its pre-planned path, it decides to move forward with the path – but, if at any point (after any step) it sees that the path is blocked anywhere by a ghost, agent-2 replans (using specified search algorithm)
  - *Note: We don't replan at a point if no ghost has entered the path since the last point we planned(or re-planned) the path.*
- If agent-2 decides to re-plan (when it sees a ghost blocking the path) but no path at that point exists, it decides to move away 1 step from the nearest ghost (calculated using Manhattan Distance) and then re-plan again.



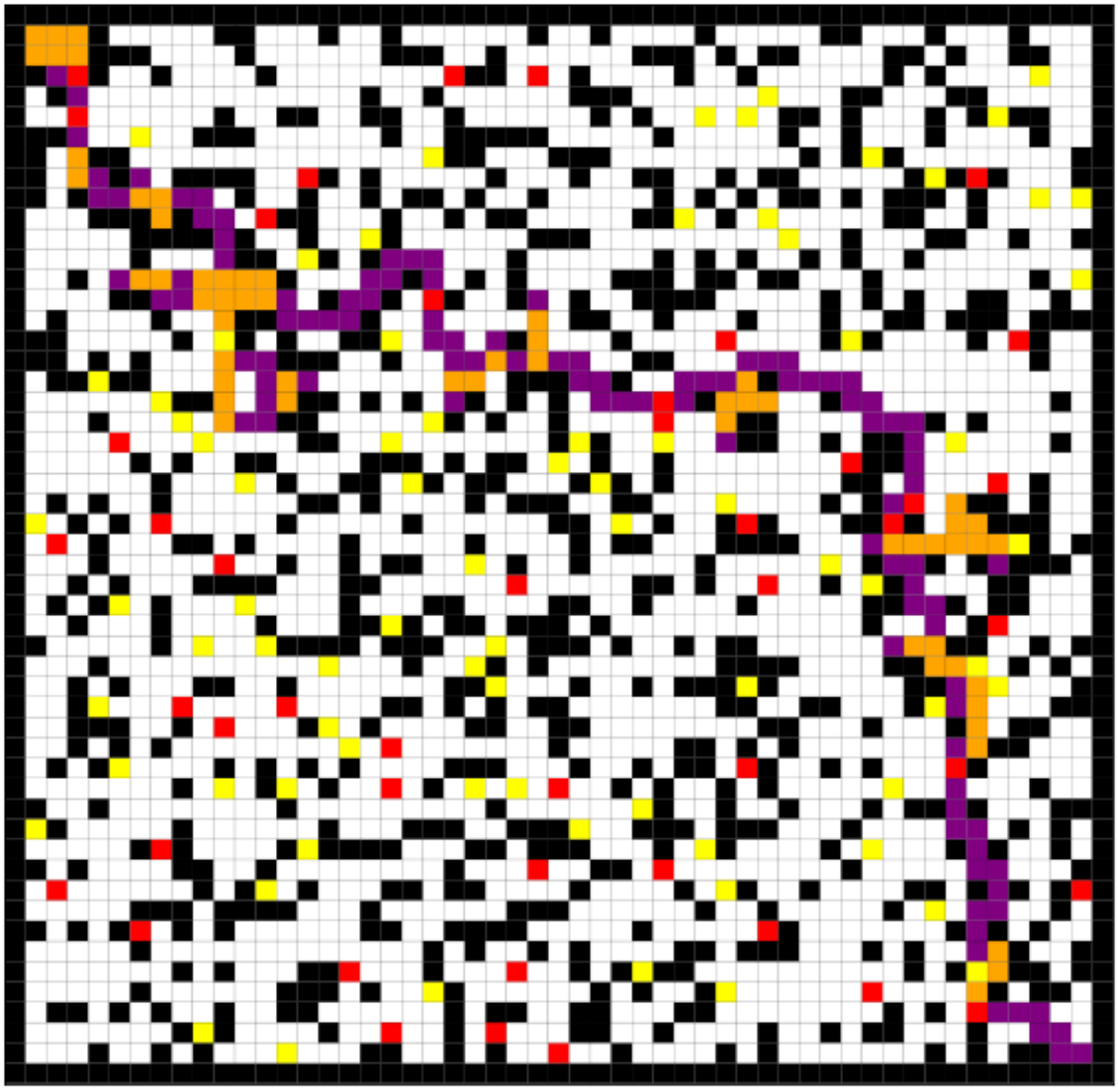
AGENT-3

- Agent-3 bootstraps agent-2 to find a path through the maze
- It is given an initial optimal path to follow (according to the initial conditions of the maze environment
- It looks at its children and sees that there are a maximum of 5 possible steps to take at any point and for each valid move out of those possible steps, it runs agent-2 simulations to find what the best probabilistic way to move forward is, according to number of successful paths that agent-2 found via any of the possible moves
    - If agent-3 finds that the success rate for agent-2 simulations in multiple directions is the same (and maximum at that point) it chooses the move that gave the shorter path
        - We break the tie randomly after this point
    - If agent-3 finds that the success rate for agent-2 simulations in all directions is 0 then it defaults to the naïve agent-2 behaviour.
- We use Manhattan Distance as the heuristic as it is optimistic for our case of grid searching



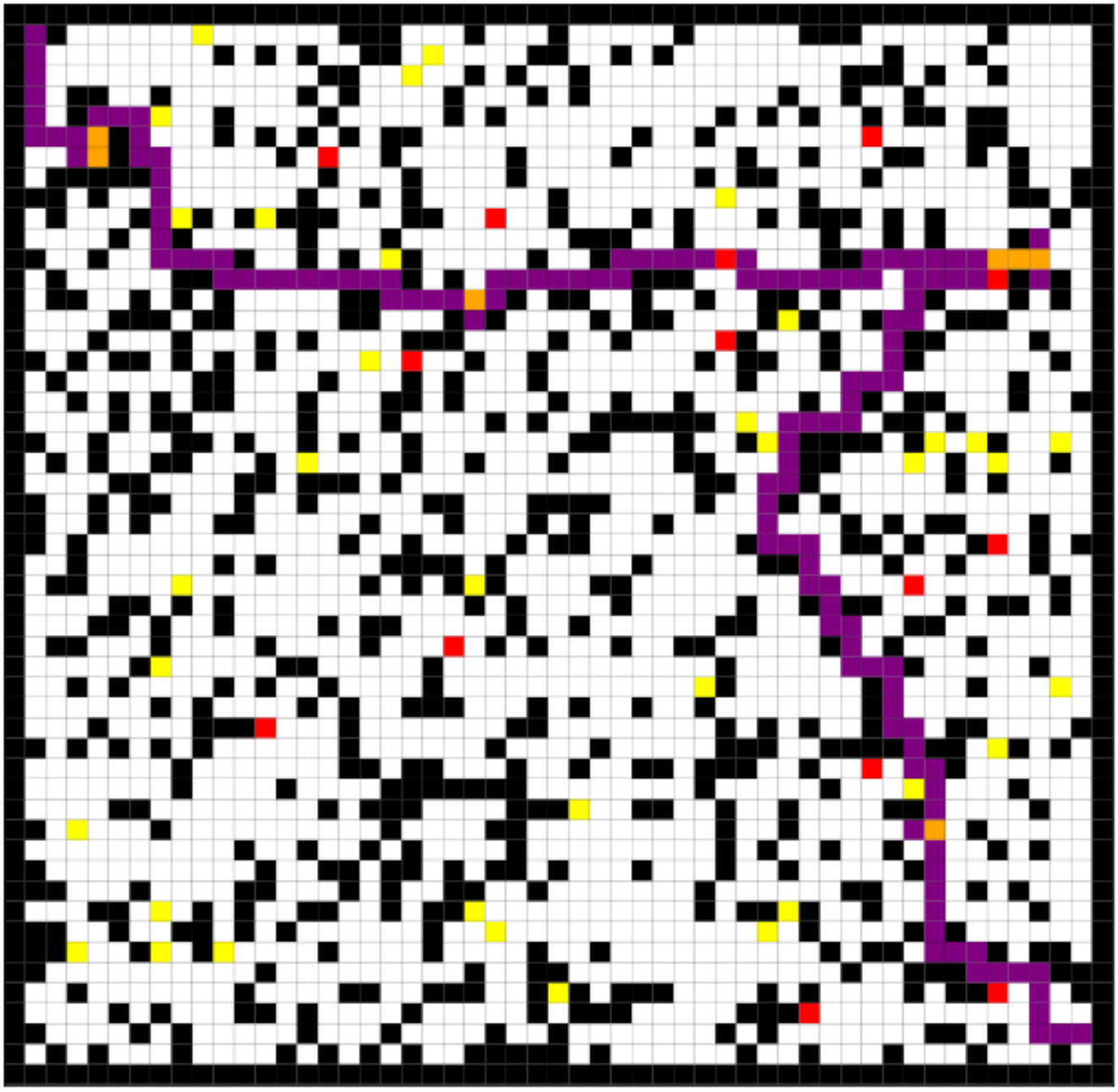
AGENT-4

- Agent-4 is an upgraded agent-2 where we make 3 major changes:
  - We increased the size(boundary) of our agent and ghosts to look like a ‘ **+** ’ , meaning that the immediate children of ghosts and agents are now also treated like parts of the ghosts and agent respectively. This essentially increases the distance that the agent maintains with ghosts and is therefore less likely to come into close contact with ghosts which in turn leads to a higher chance of survivability.
  - Also, we replan when the extended bodies intersect (when the 2 ‘ **+** ’s intersect)



AGENT-5

- This is a modified agent-4 :
  - We don’t account for ghosts in walls



Analysis

***Q** : "What algorithm is most useful for checking for the existence of these paths? Why?"*

A : Depth First Search (DFS) is the best suited algorithm to check for the existence of paths as it is the fastest algorithm from the pool of DFS, BFS, A* at finding *a* path not the optimal path and also has a smaller memory footprint(which essentially checks whether the generated maze is valid or not).

***Q** : "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?"*

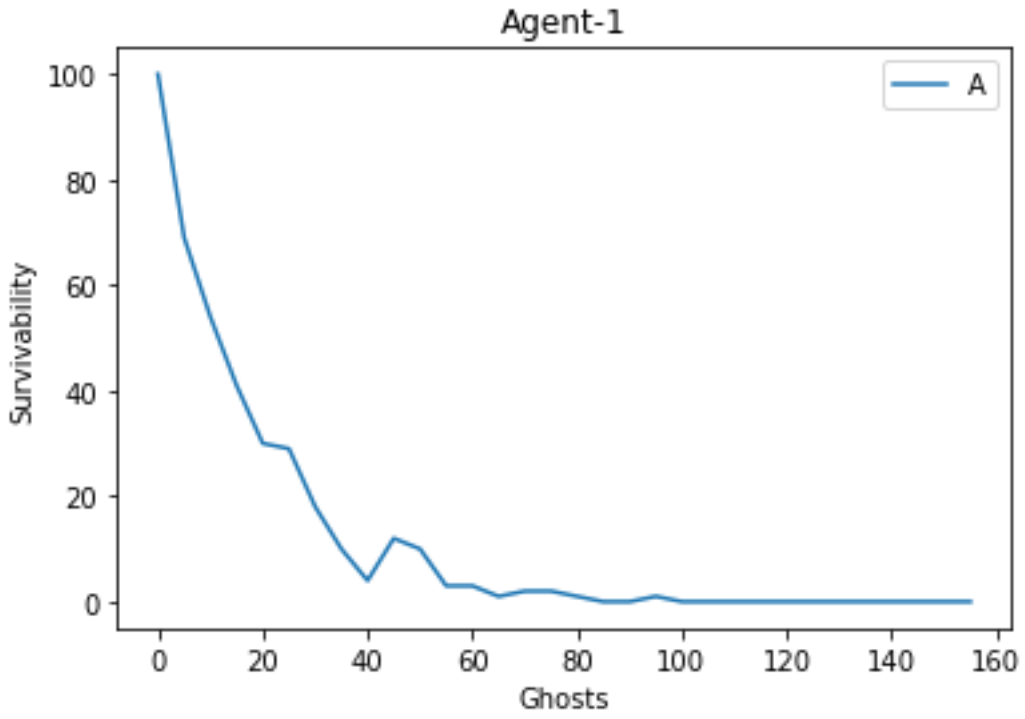
A : No, we don't need to replan every time. Whenever there is no ghost in the planned path, the path remains the same.

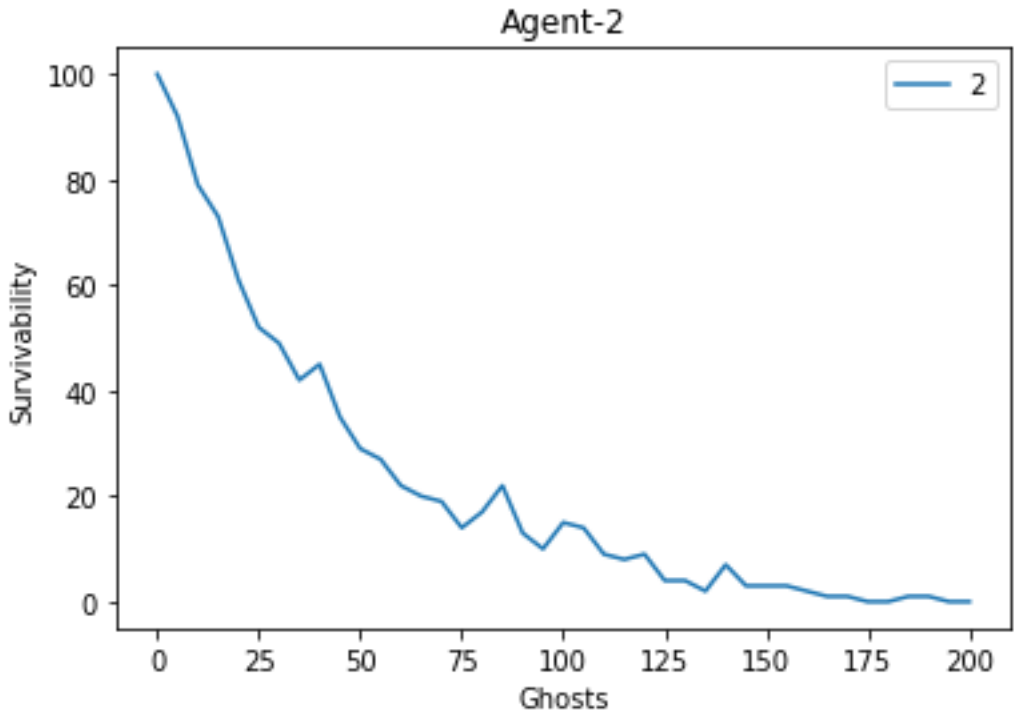
***Q** : "Agent 3 requires multiple searches - you'll want to ensure that your searches are efficient as possible so they don't take much time. Additionally, if Agent 3 decides there is no successful path in its projected future, what should it do with that information? Does it guarantee that success is impossible?"*

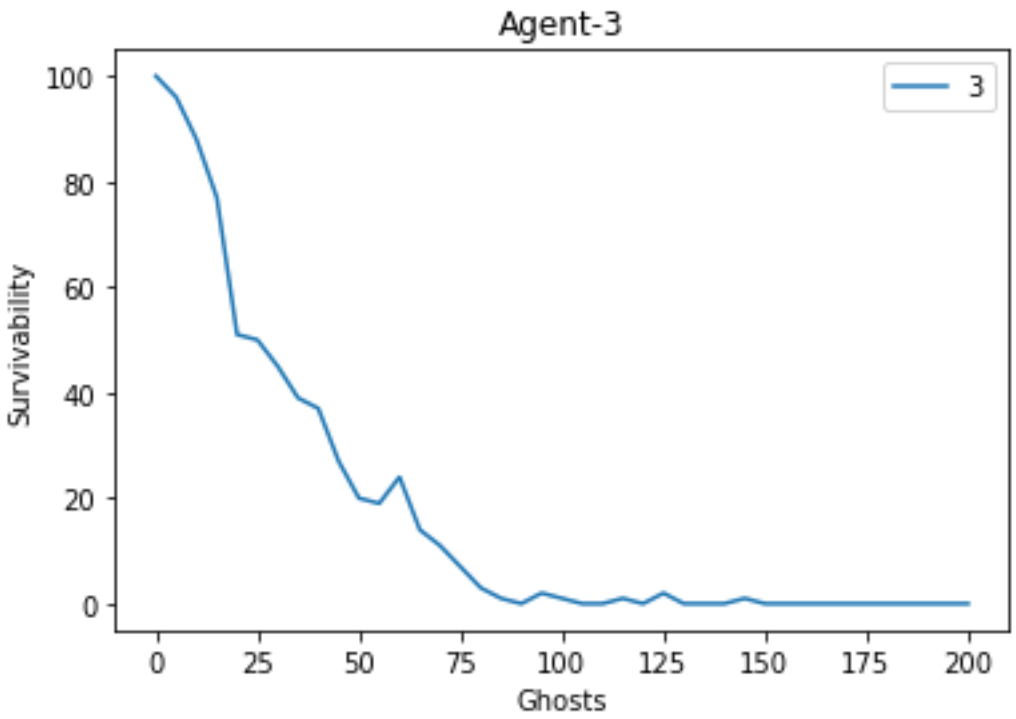
A : If there is no successful path in the near projected future – meaning that all 4 adjacent cells have equally low probability of success, we also run simulations for staying in the same position(which is our 5^{th} direction) and it might choose to stay there and then check for a path in the next step.

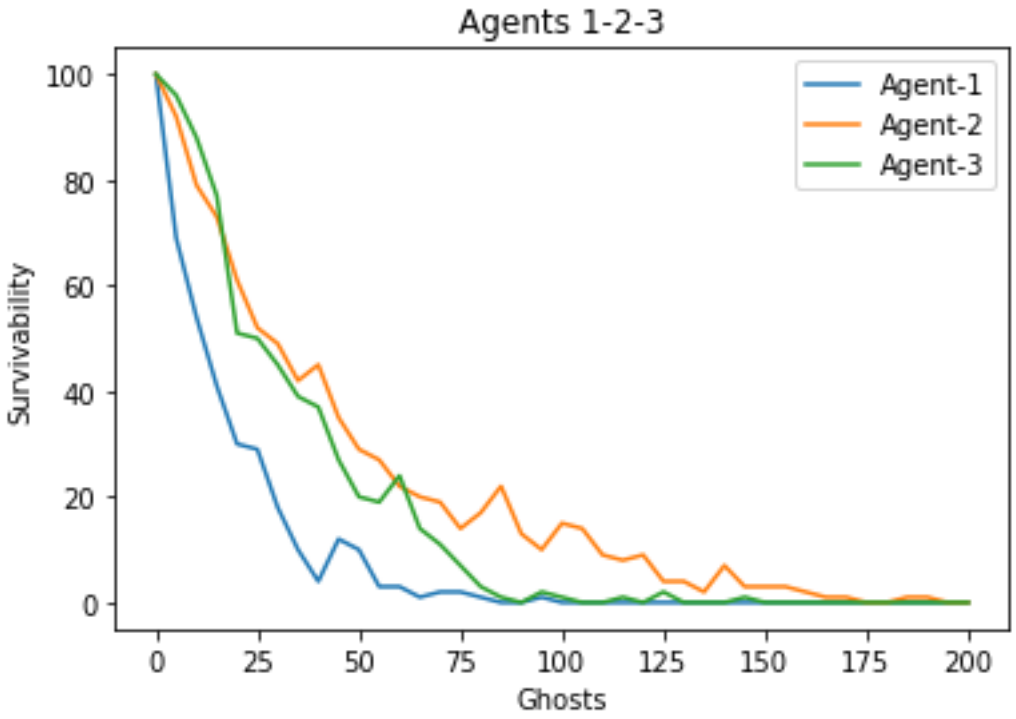
If it gets equally bad probability for all 5 possible moves, agent-3 defaults to agent-2 behaviour, but it doesn't necessarily mean that it is impossible to succeed, it means that the probabilistic measures that we took generally suggest that results will be poor but when we actually decide to move, the situation can be different and we can have a successful run. This is a case of over-exploitation of agent-2 rather than a balance with exploration.

AGENT-1 *vs* AGENT-2 *vs* AGENT-3






Agent-3
3
Survivability
Ghosts
0
20
40
60
80
100
0
25
50
75
100
125
150
175
200


Agents 1-2-3
Agent-1
Agent-2
Agent-3
Survivability
Ghosts
0
20
40
60
80
100
0
25
50
75
100
125
150
175
200

- Our agents converge to near 0 survivability at :
  - Agent-1 : 90 Ghosts
  - Agent-2 : 190 Ghosts
  - Agent-3 : 125 Ghosts

- Overall,

$$\boldsymbol{Agent\,2 > Agent\,3 > Agent\,1}$$

- Our agent-2 is generally better at surviving than agent-1 and agent-3
  - *Note: There is an outlier opposing this statement for around 15 ghosts but that can be explained by the fact that we are collecting data for only 100 mazes for some number of ghosts and would become an inlier if the number of simulations was to be increased.*
- Agent-3 is always better than Agent-1 and almost equivalent to agent-2 till about 35 ghosts but steadily becomes worse at larger number of ghosts.

  - *<u>Why is AGENT-3 not better ?</u>*
    - *Note: Even though Agent-3 has all the same information as agent-2, it still performs relatively poorly. One possible explanation for this could be that probabilistic decisions based on simulations are not the most efficient way to decide where to move, since agent-2 simulations are already making decisions based on probabilistic movements of ghosts – that might or might not happen when agent-3 ends up deciding – and therefore might render the data collected by agent-2 simulations to be useless.*

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

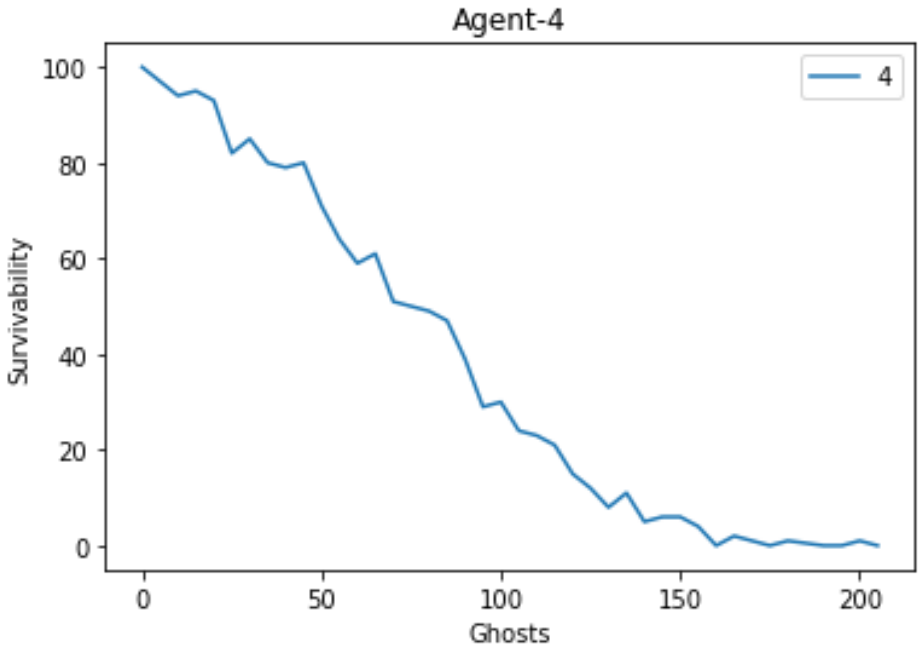
A : Agent-4 performed the best out of all the other agents as it utilises all information in a meaningful way and replans only when it's really necessary i.e., it checks it's nearby children

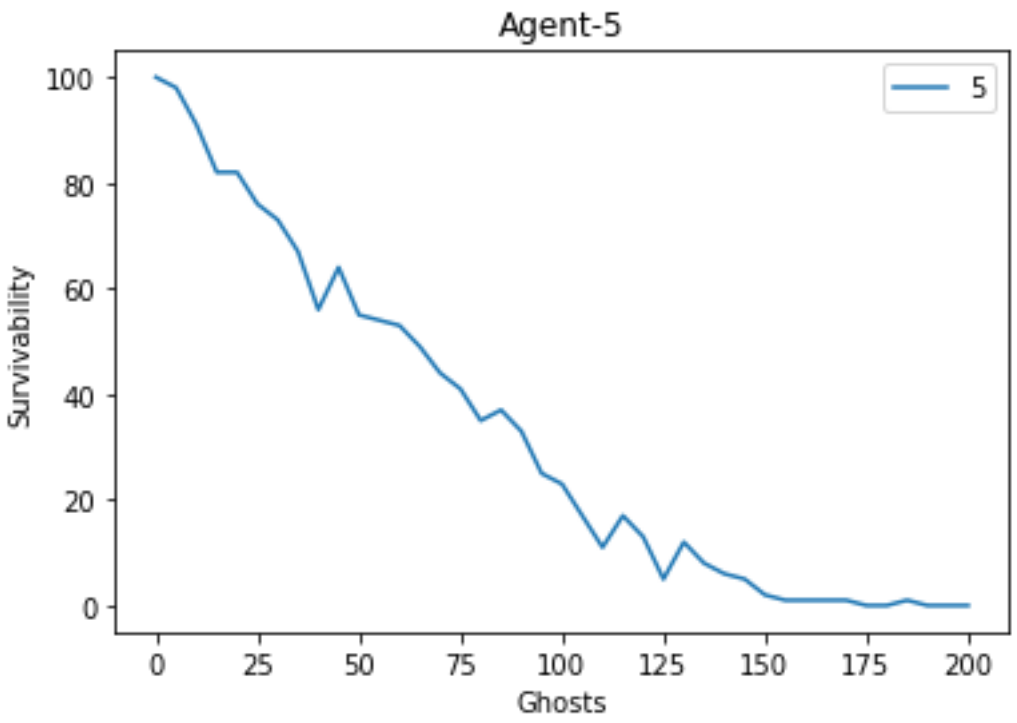
and the ghosts' nearby children, trying essentially to keep a greater minimum distance from ghosts and increases survivability

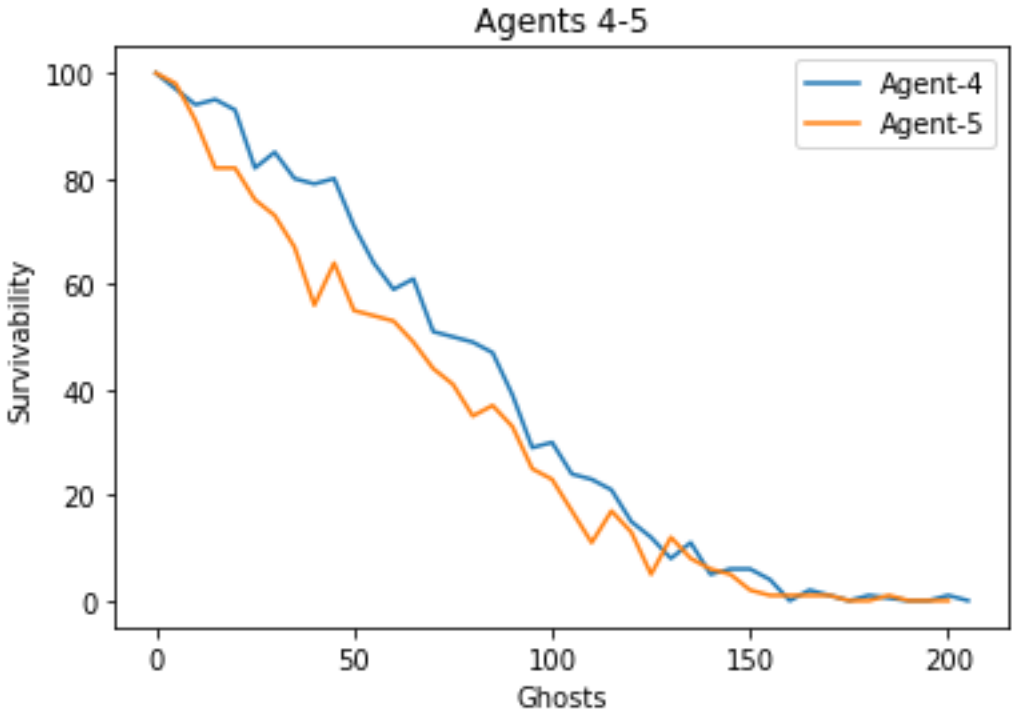
***Q** : Redo the above, but assuming that the agent loses sight of ghosts when they are in the walls, and cannot make decisions based on the location of these ghosts. How does this affect the performance of each agent? Build an Agent 5 better suited to this lower-information environment. What changes do you have to make in order to accomplish this?*

A : We are planning based on the assumption that ghosts within walls are nothing but walls, we are therefore purposefully ignoring them (so to speak).

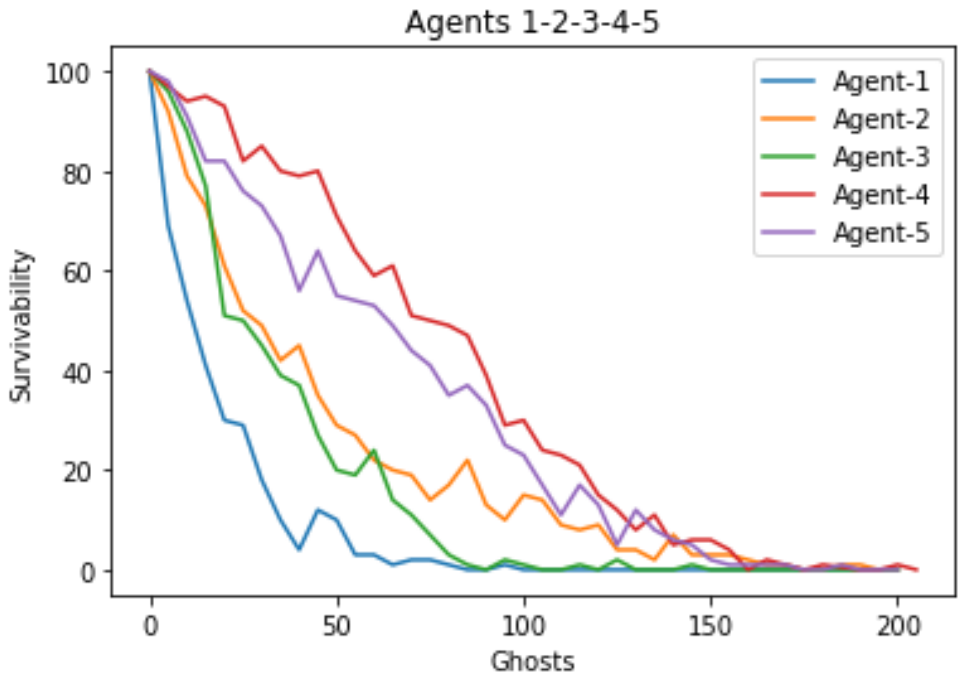
- Agent-1 Doesn't account for ghosts (doesn't care for ghosts wherever they may be).
- Agent-2 Doesn't account for ghosts inside walls and
- Agent-3 Treats ghosts inside walls as walls only for planning and replanning
- Agent-4 While running away checks for ghosts in the wall and runs away from them as well
- Agent-5 Is a change in agent-4 that doesn't allow it to account for ghosts in walls and therefore has a higher probability of dying due to ghosts in walls.




Agent-5
5
Survivability
Ghosts
0
20
40
60
80
100
0
25
50
75
100
125
150
175
200


Agents 4-5
Agent-4
Agent-5
Survivability
Ghosts
0
20
40
60
80
100
0
50
100
150
200

AGENT-1 *vs* AGENT-2 *vs* AGENT-3 *vs* AGENT-4 *vs* AGENT-5



Key Takeaways

- We work on understanding and implementing search algorithms and making them more intelligent (via our agents) and trying to have them make more informed – or intelligent – decisions.
- We observed that more information isn't always beneficial since a lot of extra information can also lead to a lot of confidently made wrong decisions(as is the case for our Agent-3) and act as *static* or *noise* for our agents.
- We must consider carefully how to explore and exploit in a balanced manner – either extreme worsens survivability.
- In our case, our Agent-4 performs the best overall – which is a result of the right amount of information that is used in a correct manner and that is exactly where agent-3 lacks (in that it has a lot of information about what the future might hold, but ends up utilising it poorly).

References

[1] Visualization : https://stackoverflow.com/questions/56614725/generate-grid-cells-occupancy-grid-color-cells-and-remove-xlabels

[2] Plotting : https://stackoverflow.com/questions/63886233/how-to-load-a-text-file-and-plot-multiple-columns-in-a-single-figure/63886440#63886440