Bayesian Agents for Stochastic Pursuit in Maze Environments

Krishaan Chaudhary & Zuhayr Rashid

April 2025

Contents

1 Introduction

This project involves designing 4 different bots for a bot to capture a rat in a ship that looks like a maze. The goal of the bot consists of 2 phases. First, it localizes its position by sensing around and cross referencing blocked cells and possible movements with a map of the maze. Then, it uses a rat sensor which pings if the rat is near and does not ping if the rat is far to determine where the rat is and plan a path to capture it.

We are working with 30×30 grids.

For the purposes of visualization:

- Black squares represent closed cells (walls)
- White squares represent open cells (cells the bot can move into)
- Blue square represents the cell the bot occupies
- Brown square represents the initial cell of the rat

We tested 2 approaches for 2 cases:

1. **Case 1: Rat is Stationary**
   (a) **Bot 1:** Baseline Algorithm
   (b) **Bot 2:** Improved Algorithm
2. **Case 2: Rat is moving randomly to adjacent open cell**
   (a) **Bot 1:** Baseline Algorithm
   (b) **Bot 2:** Improved Algorithm

2 Bot 1 Baseline - Stationary Rat

This section details the design and implementation of the Baseline Bot 1 in the stationary rat case. We focus on the algorithm and probabilistic knowledge base.

2.1 Algorithm and Probabilistic Knowledge Base Updates

Phase 1: Localize the bot's initial position using the following strategy

1. **Initialization:**
   - Initialize the bot's position (r, c) and load the empty ship map.
   - Generate a set of all possible bot locations on the ship.
   - Construct a `neighbor_map` where each cell maps to its number of blocked neighboring cells.
2. **Observation:** At each step, the bot senses the number of blocked neighbors at its current position. Possible positions are filtered to those matching this sensed count.
3. **At each Iteration:**
   - Bot attempts to move in a random direction.
   - If the move succeeds:
     - Eliminate candidate positions from which that move would have been blocked.
     - Shift all remaining possible positions in the direction of the move.
   - If the move fails:
     - Restrict candidates to only those from which that move would have failed.
4. **Termination:** This loop continues until only one candidate position remains, indicating the bot's exact location has been identified.

Throughout the process, we track the number of bot movements, blocked neighbor detections, and total timesteps taken to localize.

Phase 2: Updating Probabilistic Knowledge Base and Exploration

Once the bot has localized its position, it initializes a uniform belief distribution over all open cells for the rat's location:

$$P_{\text{init}}(r, c) = \frac{1}{N} \quad \text{for all open } (r, c)$$

where N is the number of open cells.

Iterative Search Process

At each timestep:

1. **Ping** the environment:
   - `ping(info, alpha)` returns:
     - `'Found'` if the bot is on the rat.
     - `'True'` if a ping is received (rat is nearby).
     - `'False'` otherwise.
2. **Bayesian Update** of the belief $P(r, c)$ based on the ping result:

   Let $dist(K, L)$ = Manhattan distance between position K and L

   Let B = current position of bot
   - If ping was heard (`True`):

     set probability of current cell (cr, cc) to 0 since rat is not here.

     Loop through every open cell X_i = (r,c):

$$P(\text{rat in } X_i \mid \text{bot heard ping at } B) = \frac{P(\text{rat in } X_i \text{ and bot heard ping at } B)}{P(\text{bot heard ping at } B)}$$

$$= \frac{P(\text{rat in } X_i) \cdot P(\text{bot heard ping at } B \mid \text{rat in X})}{P(\text{bot heard ping at } B)}$$

$$= \frac{P(\text{rat in } X_i) \cdot e^{-\alpha(dist(X_i, B)-1)}}{P(\text{bot heard ping at } B)}$$

$$= \frac{P(\text{rat in } X_i) \cdot e^{-\alpha(dist(X_i, B)-1)}}{P(\text{bot heard ping at } B \text{ and rat in } X_1) + ... + P(\text{bot heard ping at } B \text{ and rat in } X_N)}$$

$$= \frac{P(\text{rat in } X_i) \cdot e^{-\alpha(\text{dist}(X_i, B)-1)}}{\sum_{j=1}^{N} P(\text{rat in } X_j) \cdot e^{-\alpha(\text{dist}(X_j, B)-1)}}$$

- If ping was not heard (`False`):

  set probability of current cell (cr, cc) to 0 since rat is not here.

  Loop through every open cell X_i = (r,c):

$$P(\text{rat in } X_i \mid \text{bot did not hear ping at } B) = \frac{P(\text{rat in } X_i \text{ and bot did not hear ping at } B)}{P(\text{bot did not hear ping at } B)}$$

$$= \frac{P(\text{rat in } X_i) \cdot P(\text{bot did not hear ping at } B \mid \text{rat in X})}{P(\text{bot did not hear ping at } B)}$$

$$= \frac{P(\text{rat in } X_i) \cdot e^{-\alpha(dist(X_i,B)-1)}}{P(\text{bot did not hear ping at } B)}$$

$$= \frac{P(\text{rat in } X_i) \cdot (1 - e^{-\alpha(dist(X_i,B)-1)})}{P(\text{bot did not hear ping at } B \wedge \text{bot in } X_1) + ... + P(\text{bot did not hear ping at } B \wedge \text{bot in } X_N)}$$

$$= \frac{P(\text{rat in } X_i) \cdot (1 - e^{-\alpha(\text{dist}(X_i,B)-1)})}{\sum_{j=1}^{N} P(\text{rat in } X_j) \cdot (1 - e^{-\alpha(\text{dist}(X_j,B)-1)})}$$

  Normalize $P(r, c)$ such that:

$$\sum_{r,c} P(r, c) = 1$$

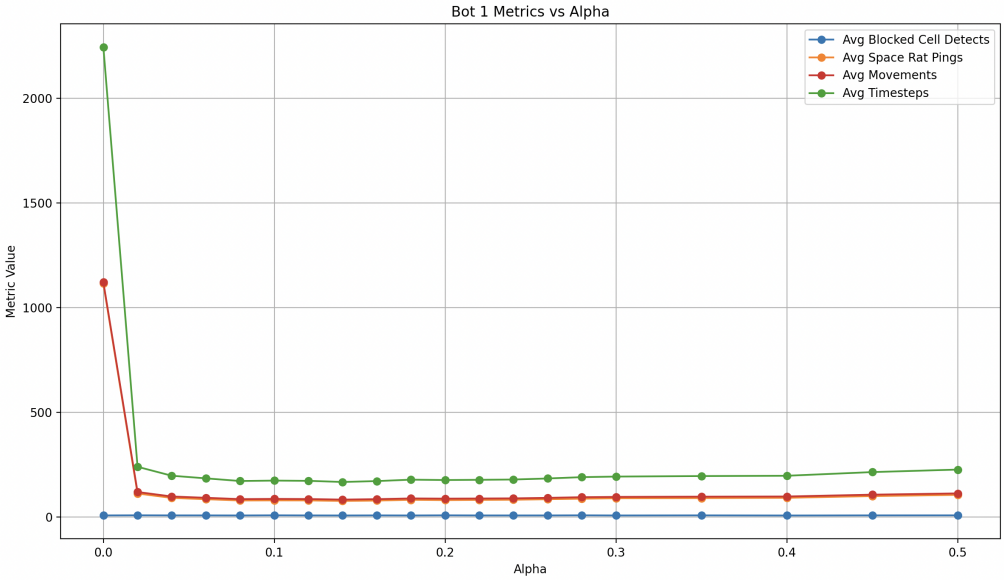
  by adding up all nonzero probabilities, and dividing the probability in every cell by that value.

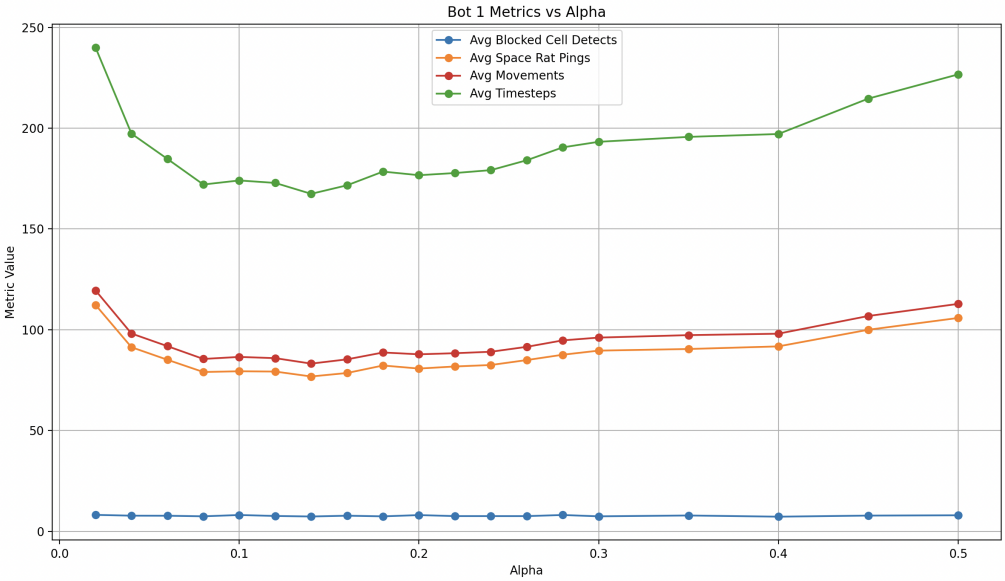
3. **Select Target Cell** with highest $P(r, c)$, and plan shortest path using A* from the current bot location.

4. **Traverse Path** step-by-step:

   - After each move, ping again and update the belief map.
   - If the rat is found, return all counter metrics.

2.2 Data and Analysis





Data Generation Procedure:

To evaluate the performance of various bot strategies under different detection sensitivities, we simulated both stationary and moving rat scenarios on ship-like grid environments. The following methodology was employed:

- **Ship Generation:** A total of 20 unique ships were generated using the `init_ship(30)` function, each representing a 30×30 grid environment.
- **Alpha Values:** Bots were tested under a range of α values:

$$\alpha \in \{0.00, 0.02, 0.04, \ldots, 0.50\}$$

- **Simulation Runs:** For each α value and each ship instance, 10 independent simulation runs were performed, totaling 100 runs per α.
- **Metrics Stored:** Each bot returns the following metrics per run: number of blocked cell detections, number of space rat pings, number of movements, and total timesteps to capture. These are averaged across all runs per α value, and then they are used for plotting.

Analysis of Results:

We plot a graph of how many timesteps it takes for various phases in the ratfinding process for various alpha values.

The average number of blocked cell detects, which is reflective of the bot localization process tends to be small regardless of alpha value, around 7-8 timesteps because this process does not use ping information in any way.

At $\alpha = 0$, we notice a very large amount of timesteps required for the ratfinding process, specifically from the movement and ping actions. In order to understand why this happens, we can see that in this case, the probability of hearing a ping in any given cell $= e^{-0(x-1)} = 1$. This means we will hear a ping regardless of where we are. This confuses the bot because it always thinks the rat is nearby, and its probabilistic knowledge base does not provide it any useful information on how to differentiate unvisited cells. It plans a path to every open cell from left to right, top to bottom on the grid. Developing a separate approach to handle the $\alpha = 0$ case separately is definitely a consideration for the improved bot.

For $0 < \alpha \leq 0.2$, we notice that it takes the bot around 200 timesteps to find and catch the space rat. This range is the best for bot performance because it is reasonably likely for pings to happen anywhere up till 15-20 units of Manhattan Distance away from the rat. This range captures a lot of cells on the board. As a result, the result of a ping at various places on the ship is quite informative for this range in terms of the bot's ability to update its probabilistic knowledge base.

For $\alpha > 0.2$, we notice that the number of timesteps once again increases somewhat. This makes intuitive sense because for alpha values in this range, it is only likely to hear pings if we are 1 or 2 units of Manhattan Distance away from the rat. This means that the bot is in many situations where its probabilistic knowledge base tells it to plan paths really far away from its current position, contributing to a journey taking overall more timesteps to find the rat.

3 Bot 2 Improved - Stationary Rat

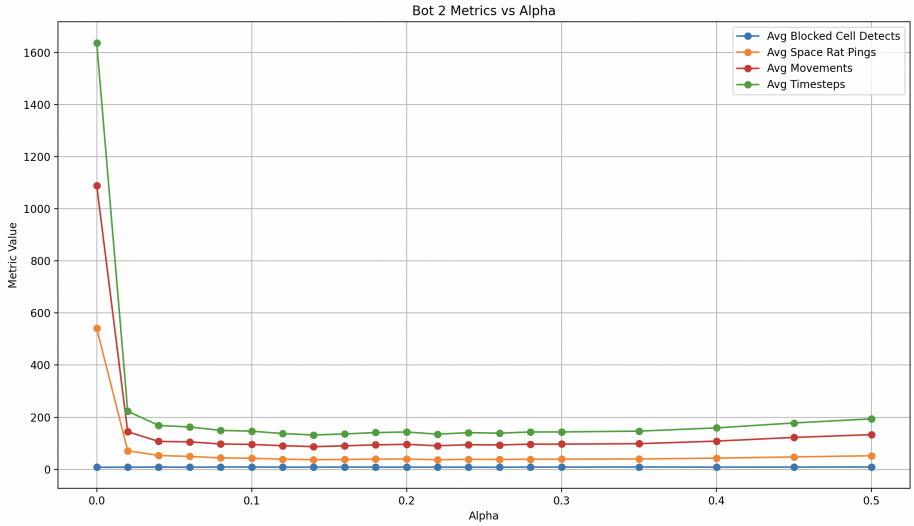
This section details the design and implementation of Bot 2. We focus on the algorithm and probabilistic knowledge base.

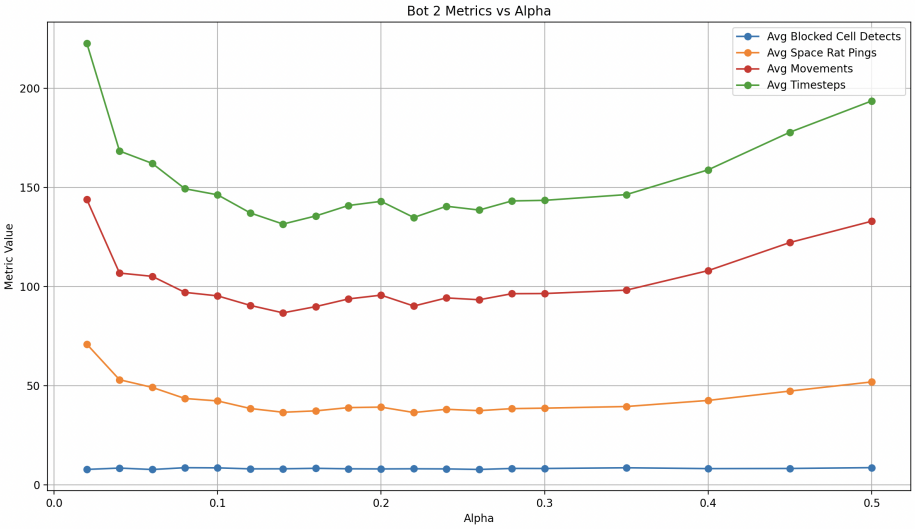
3.1 Algorithm and Probabilistic Knowledge Base Updates

There are 4 key ways that Bot 2 improves upon Bot 1. While the procedure to calculate and update the probabilistic knowledge base remains the same as Bot 1, how Bot 2 makes use of it is different.

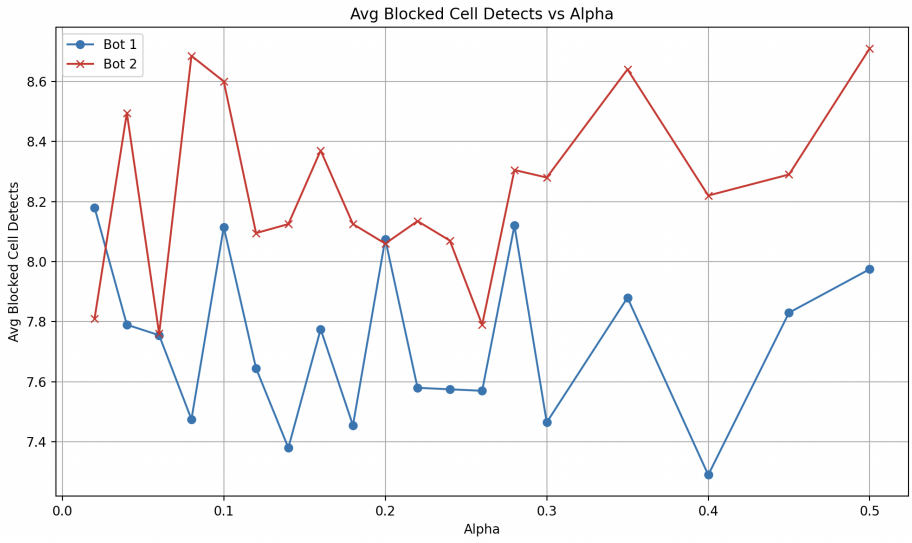
1. **Avoid Ping on 0 Probability Cells**: Bot 2 will not ping if it is at a cell that has been determined by the probabilistic knowledge base to definitely not contain the rat. The motivation behind this is that pinging a cell that definitely does not have the rat does not provide much marginal benefit in terms of updating our knowledge base. Avoiding a ping when it does not provide much marginal value saves us a timestep in the overall procedure.
2. **Do not Over-commit to Long Paths**: A problem we noticed with Bot 1 is that it sometimes went down really long paths to the highest probability cell, a cell that may not even have a high probability by the time the bot reaches it due to the intermediate pings along the way. Bot 2 has a mechanism to counter this problem. If the path to the highest probability cell is more than 15 steps, it will only move 15 steps along this path before stopping and planning a new path to the new best cell based on its pings along the 15 steps. It will then repeat this process until the rat is found. 15 was determined to be an effective value through experimentation.
3. **Modified A* Heuristic that Favors High Rat Probability Cells**: Another modification that improved the performance of this bot is as follows. We modified the A* shortest path finding algorithm Bot 2 uses when calculating the shortest path from its current position to the cell with the highest rat probability. We decrement the cost of visiting a cell by a scaling factor, k, times the probability that the cell contains a rat. The intuition behind this is that the rat should be encouraged to check a lot of cells of high rat probability along the way to checking the cell with the highest probability. Experimentally, we found 1000 to be a good value for k.
4. **Handling the Special $\alpha = 0$ case with DFS**: We noticed that Bot 1 struggled a lot in the $\alpha = 0$ case where it would plan a path to every single cell and visit it because for this α value it would hear a ping everywhere, rendering its knowledge base only effective in narrow down its search space by eliminating visited cells. The way it planned paths to the each cell was quite inefficient as well. In this case, the only effective strategy is to visit and ping each cell, which we did using a DFS approach.

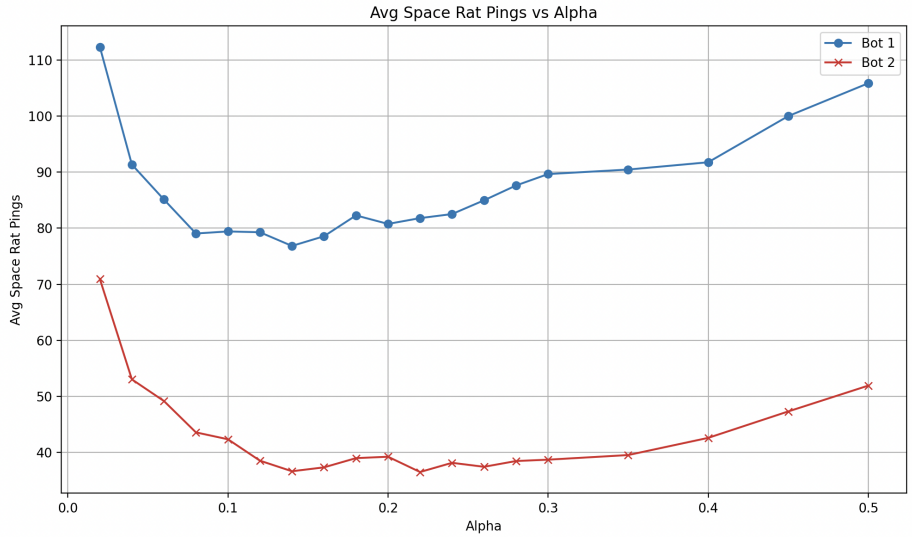
3.2 Data and Analysis

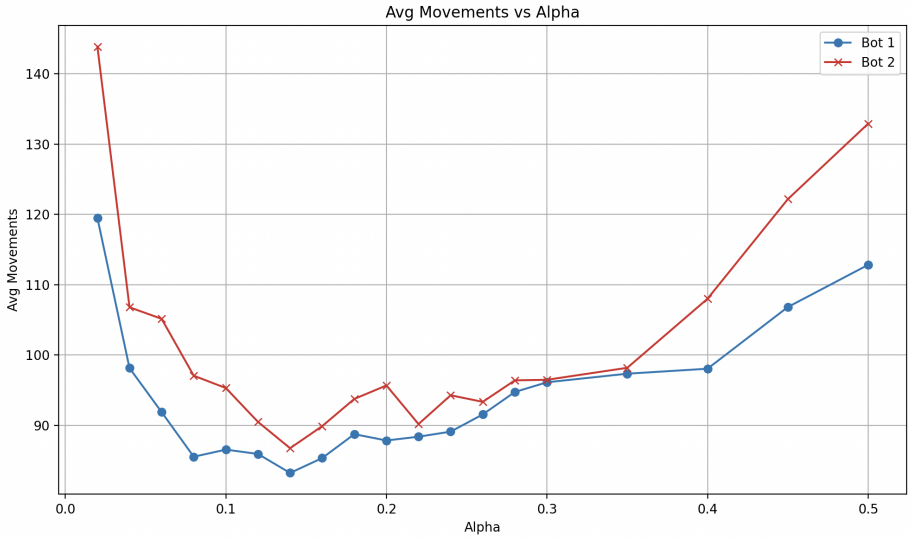


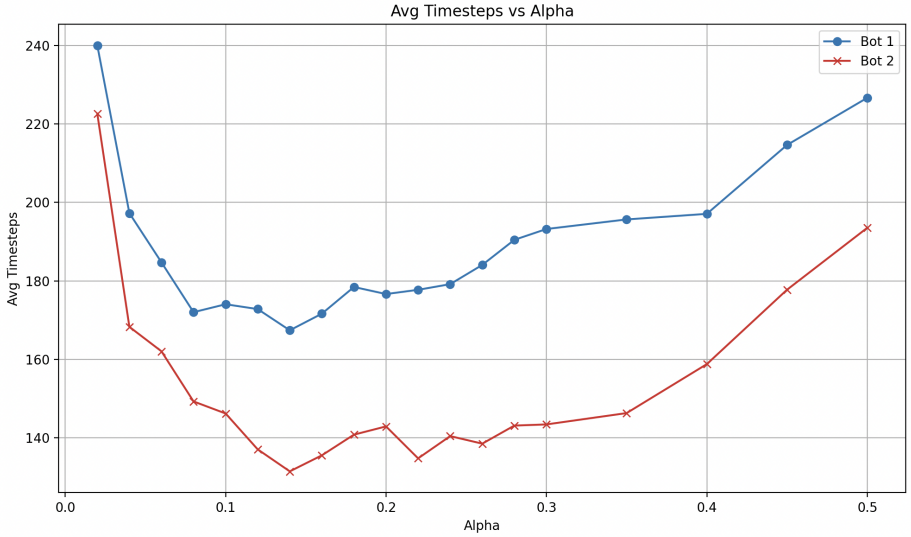


3.3 Comparing Bot 1 and Bot 2 Performance






Avg Movements vs Alpha
Bot 1
Bot 2
Avg Movements
Alpha
140
130
120
110
100
90
0.0
0.1
0.2
0.3
0.4
0.5


Avg Timesteps vs Alpha
Bot 1
Bot 2
Avg Timesteps
Alpha
240
220
200
180
160
140
0.0
0.1
0.2
0.3
0.4
0.5

Analysis of Results:

For $\alpha = 0$, the DFS approach helped reduce timesteps significantly from around 2300 to around 1600 timesteps. Still, this is a special case where the rat sensor is very uninformative, so there is not much the Bot 2 can do better than just searching the entire space and pinging unvisited cells.

The overall trends and patterns in the graph show that when α is around .05-.20, Bot 2 does significantly better than Bot 1. This suggest that Bot 2 is using the information from the ping sensor more effectively than Bot 1. This fits in with our changes; recalculating more frequently and prioritizing high probability cells in our A* aims to use our information more than the default Bot 1. Outside this range, pinging gives less information, and the difference between Bot 1 and Bot 2 decreases.

To compare the actions more specifically, we notice that since Bot 1 and Bot 2 use the same approach for initial localization, there is not a significant difference in the time steps for that graph.

In the second graph, we can see that the main improvement for Bot 2 is the amount of times it has to ping. This can be explained by 2 reasons: the first is that we avoid pinging cells that have 0 probability of containing the rat, and the other is that an overall improved approach will require less pinging, since we find the rat quicker.

While Bot 2 does have more movements on average than Bot 1 as seen in the third graph, these are not significant enough to undo the gains in efficiency by more efficient pinging, as seen in the overall improvements in the fourth graph.

3.4 Discussion of Other Approaches

1. **Triangulation**: This approach involved pinging many times at the bot's initial location to narrow a search radius for the rat, which helped narrow down the search space.

   We tried several follow up techniques: one was to move to the most probable location of the rat and then repeat this procedure. Another was to move to the most uncertain location (defined as location with rat probability closest to mean rat probability across all cells) and then ping several times again. Another was to combine this along with distance to target cell to make a better decision that weighs the information gain of a cell and its distance away from the current bot position.

   The problem with this approach was that it would take too many pings to meaningfully narrow the search space, which would cause the overall approach to be less efficient in terms of total number of timesteps.

2. **Periodic Pinging**: We also tried an approach where we pinged several times at one position along a planned path. For example, ping 4 times every 5 moves we make. However, even after playing around with many different values of ping frequency and intermediate non-ping moves, include an approach that dynamically changed them based on the entropy of our rat probability map, this framework provided marginal, if not no improvements to our timestep count.

4 Bot 1 Baseline - Moving Rat

This section details the design and implementation of the Baseline Bot 1 in the moving rat case. We focus on the algorithm and probabilistic knowledge base.

4.1 Algorithm and Probabilistic Knowledge Base Updates

Phase 1: Localization

This phase is the same as Bot 1 Baseline - Stationary Rat.

Phase 2: Rat Finding

Once the bot's position is known, it searches for the moving rat using probabilistic pings and Bayesian updates.

At each timestep:

1. **Ping the Environment**:
   - `ping(info, alpha)` returns:
     - `'Found'` if the bot is at the rat's position ($bot\_r = rat\_r, bot\_c = rat\_c$).
     - `'True'` if a ping is received (rat is nearby), with probability $e^{-\alpha(dist-1)}$, where $dist = |bot\_r - rat\_r| + |bot\_c - rat\_c|$.
     - `'False'` otherwise.
2. **Rat Moves**:
   - The rat moves to a randomly chosen adjacent open cell, updating the ship state as in Phase 1.
3. **Bayesian Update** of the belief $P(r, c)$ based on the ping result:

   Let $dist(K, L)$ = Manhattan distance between positions K and L, and $B = (bot\_r, bot\_c)$ be the current bot position.
   - Set $P(bot\_r, bot\_c) = 0$ since the rat is not at the bot's position (unless `'Found'`).
   - If ping was heard (`True`):

     $$P(\text{rat in } X_i \mid \text{bot heard ping at } B) = \frac{P(\text{rat in } X_i) \cdot e^{-\alpha(dist(X_i,B)-1)}}{\sum_{j=1}^{N} P(\text{rat in } X_j) \cdot e^{-\alpha(dist(X_j,B)-1)}}$$

     where $X_i = (r, c)$ is an open cell, and N is the number of open cells.
   - If ping was not heard (`False`):

     $$P(\text{rat in } X_i \mid \text{bot did not hear ping at } B) = \frac{P(\text{rat in } X_i) \cdot (1 - e^{-\alpha(dist(X_i,B)-1)})}{\sum_{j=1}^{N} P(\text{rat in } X_j) \cdot (1 - e^{-\alpha(dist(X_j,B)-1)})}$$

Normalize $P(r, c)$ such that:

$$\sum_{r,c} P(r, c) = 1$$

by adding up all nonzero probabilities, and dividing the probability in every cell by that value.

4. **Update Rat Movement Probabilities**:

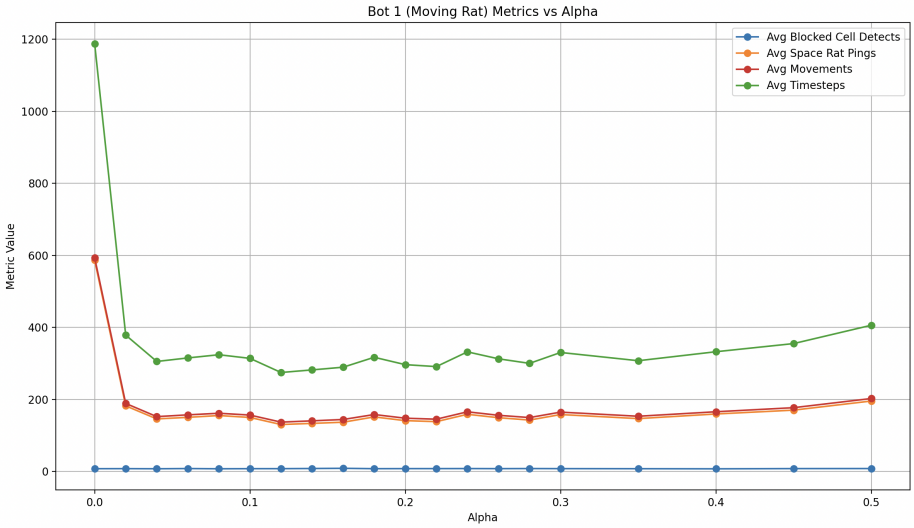
   - Using the transition matrix T, compute the new probability distribution:
   $$P_{\text{new}}(nr, nc) = \sum_{(r,c)} P(r, c) \cdot T[(r, c)][(nr, nc)]$$
   where $T[(r, c)][(nr, nc)] = \frac{1}{k}$ if (nr, nc) is an open neighbor of (r, c), and k is the number of open neighbors.

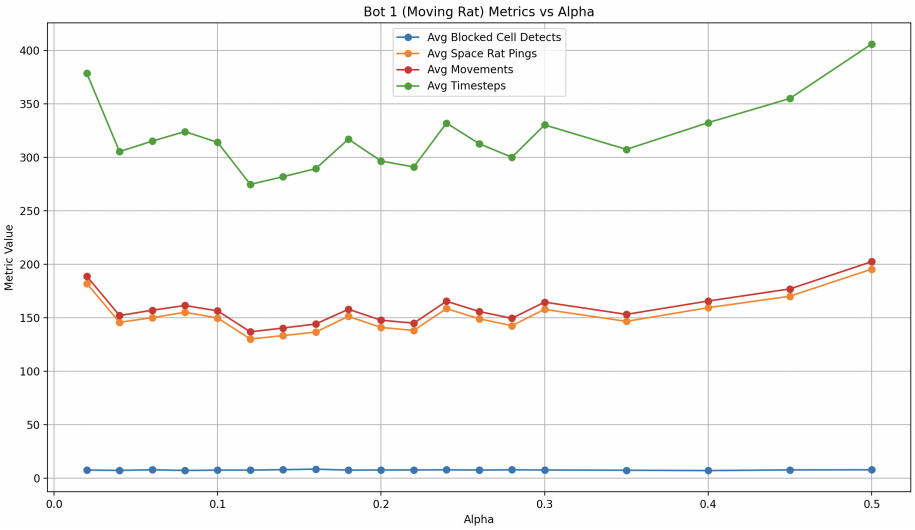
5. **Select Target Cell**: Identify the cell with the highest $P(r, c)$.

6. **Plan and Traverse Path**:

   - Use A* to compute the shortest path from the bot's current position to the target cell.
   - For each step along the path:
     - Move the bot, updating the ship state.
     - The rat moves to a random adjacent open cell.
     - Ping again and update $P(r, c)$ as above.
     - If `'Found'`, return metrics: number of blocked cell detects, pings, movements, and timesteps.

4.2 Data and Analysis





Compared to the Baseline Bot 1 in the stationary rat case, we notice that the Baseline Bot 1 in the moving rat case takes less steps when $\alpha = 0$, but 100-150 steps more for nonzero α values.

There is no noticeable increase in the initial localization phase, which is reasonable since the movement of the rat does not affect this part.

It is reasonable to see an increase in the amount of time steps needed to move and ping because the random movement of the rat can cause the rat to move into spaces where the bot has already checked. The random movement also adds to the uncertainty of the rat's position, thus weakening the ability of the probabilistic knowledge base to narrow down locations as easily as it did in the stationary case.

In the $\alpha = 0$ case the bot's probabilistic knowledge base does not help it much, as described earlier. A potential reason why it is doing better in the case with random rat movement is because the rat is randomly moving into the path of the bot. The bot's search process is not exploring efficiently because of its over reliance on the probabilistic knowledge base. The approach of the bot in this case is to make a journey to every open cell left to right because of the way its probabilistic knowledge base is structured. It is more likely for a moving rat to randomly move into one of the bot's planned paths, than it is for the bot to stumble upon a stationary rat using its planned paths.

5 Bot 2 Improved - Moving Rat

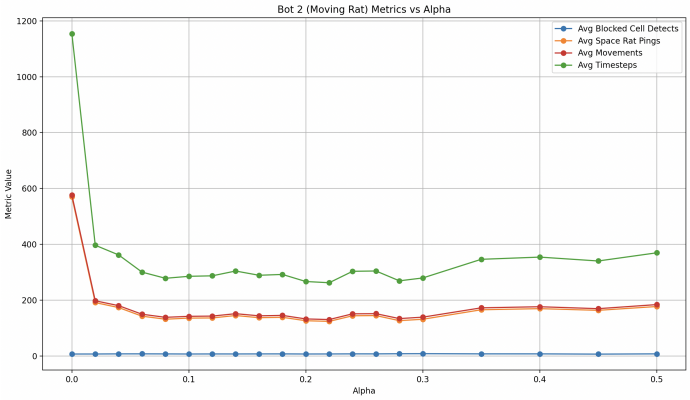
We now test Bot 2 in this new Moving Rat Case.

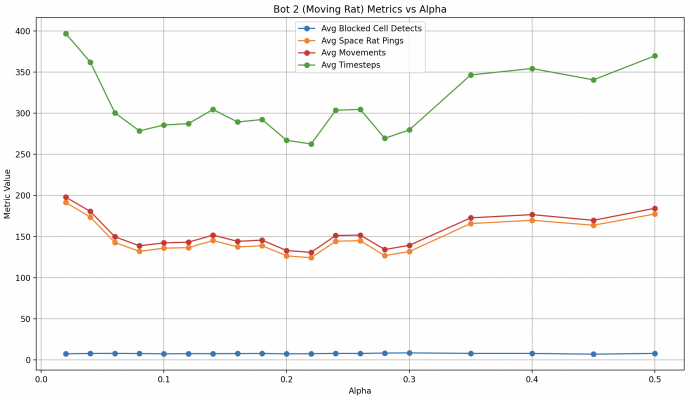
5.1 Algorithm and Probabilistic Knowledge Base Updates

Original Approach:

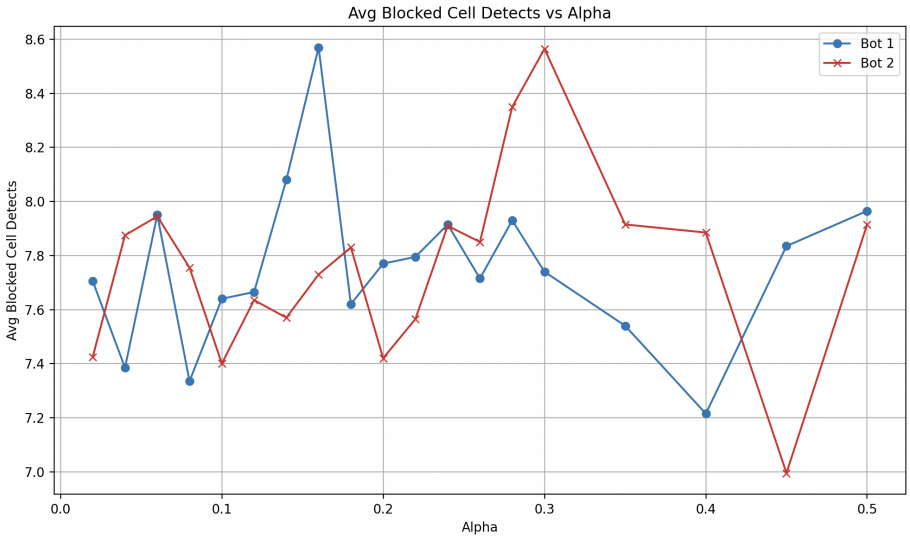
If we keep the overall framework of Bot 2 in the moving case as the same as the one in the stationary case, except for the new probabilistic knowledge base update rules (described in Bot 1 Moving Case), we can test this version to see if our Bot 2's strategy helps it better find the rat when it is moving. We also exclude the DFS approach for $\alpha = 0$.

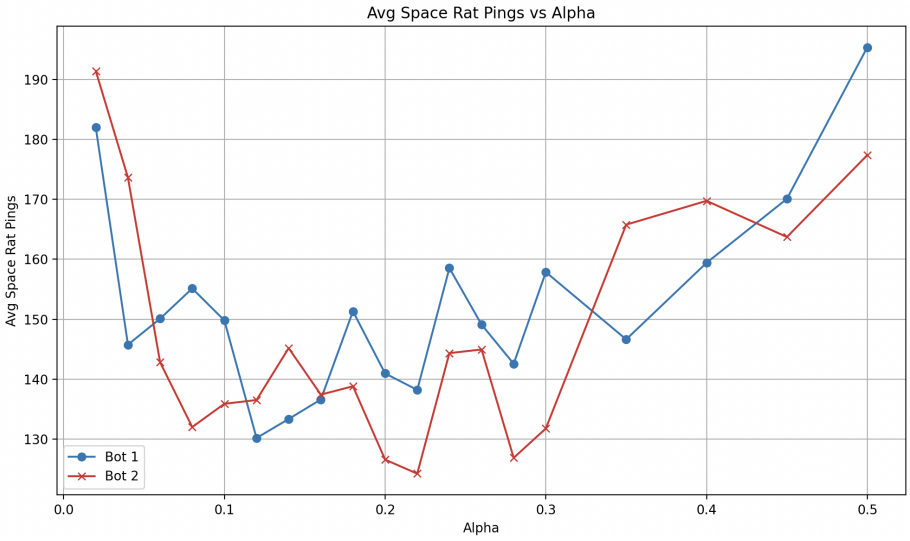
5.2 Data and Analysis

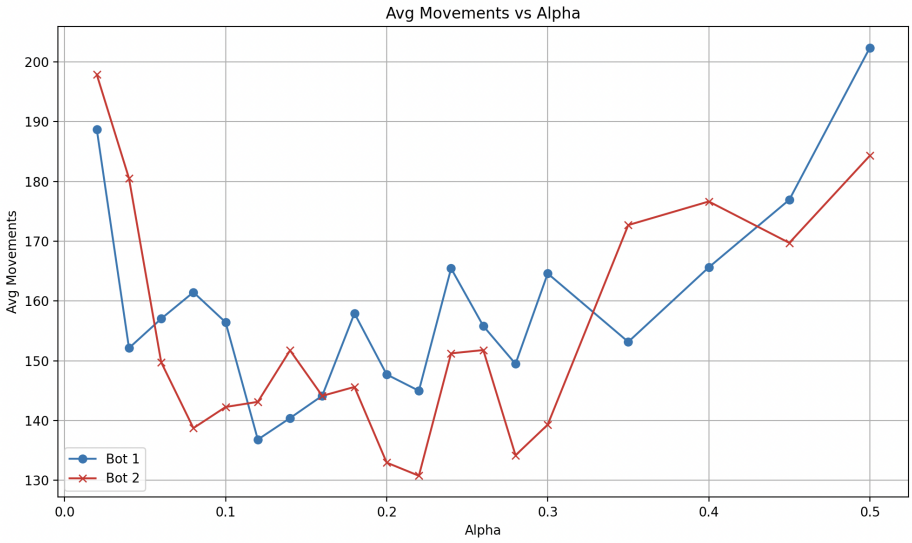


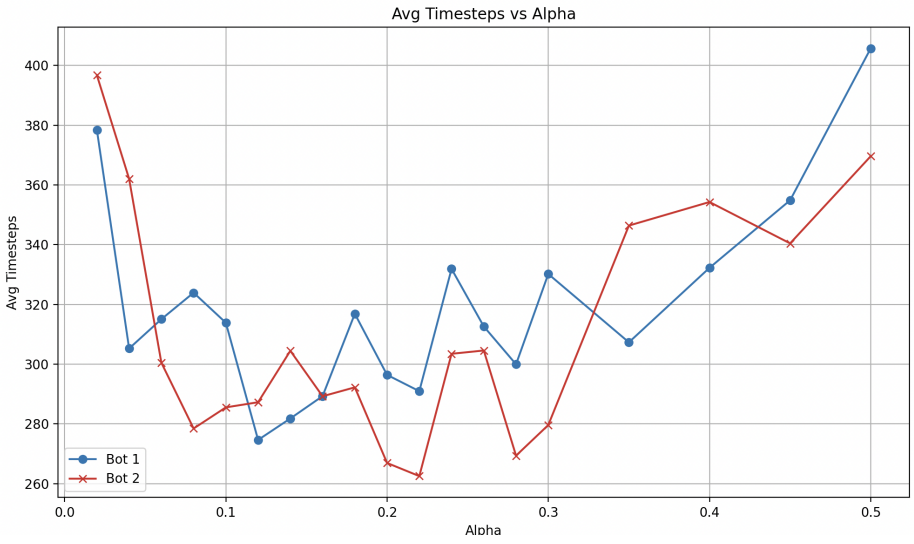


5.3 Comparing Bot 1 and Bot 2 Performance






Avg Movements vs Alpha
Avg Movements
Alpha
Bot 1
Bot 2
200
190
180
170
160
150
140
130
0.0
0.1
0.2
0.3
0.4
0.5


Avg Timesteps vs Alpha
Avg Timesteps
Alpha
Bot 1
Bot 2
400
380
360
340
320
300
280
260
0.0
0.1
0.2
0.3
0.4
0.5

Analysis of Results:

Based on the plots that compare the movements, senses, pings, and overall timesteps of both bots in the moving case, we notice that Bot 2 performs slightly better than Bot 1 for most α values of interest (0.0 - 0.2). However, there is no specific key metric that Bot 2 outperforms Bot 1 in when it comes to this moving case, as opposed to the results in the stationary case.

5.4 Further Improving Bot 2 - Moving Case

Approach

In order to modify our Bot 2 approach to improve our results for alpha values between 0 and 0.225 in the moving case, we made the following changes to our original Bot 2 design:

1. **Commit to Shorter Portion of Planned Path**: Because of the moving nature of the rat, long planned paths are not very good for catching the rat by the time we reach the end of the planned path. We reduced our path length cap parameter so that a rat will move at most 10 steps along a planned path before recalculating a new best destination.

2. **Planning for a Future Rat Position**: Bot 2 now simulates the map 1 step into the future based on the transition model, and it uses that to determine the best cell to plan a path towards. The math is outlined below. The A* algorithm used by this bot uses an adapted heuristic which rewards visiting cells that have a high future probability by subtracting a scale factor multiplied the future rat probability in that cell from its original heuristic of Manhattan Distance.

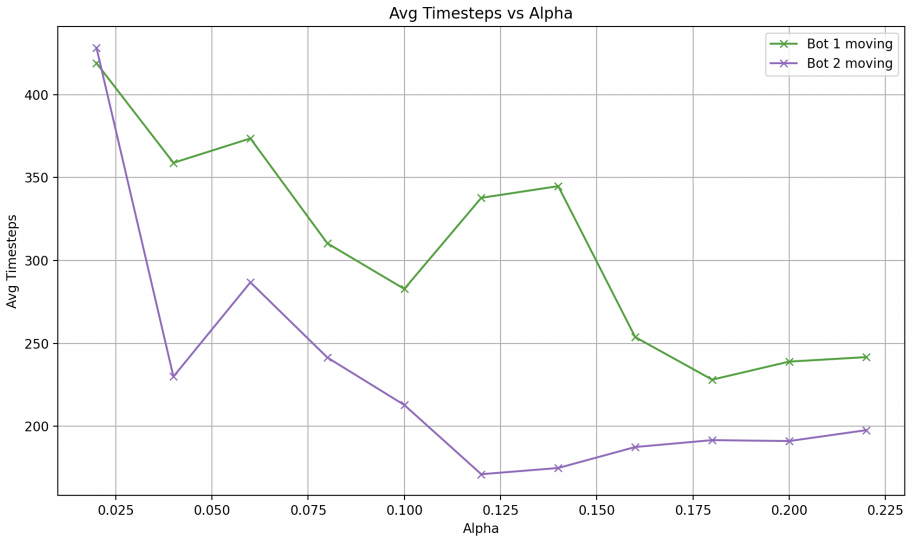
   The probability of the rat being at cell (r, c) at time $t + 1$, denoted as $P_{t+1}(r, c)$, is given by:

$$P_{t+1}(r,c) = \sum_{(r',c') \in \mathcal{N}(r,c)} P_t(r',c') \cdot P\left[(r',c') \to (r,c)\right]$$

   where:

   - $\mathcal{N}(r, c)$ is the set of neighboring cells that could move to (r, c) in one step,
   - $P_t(r', c')$ is the probability the rat is at (r', c') at time t,
   - $P\left[(r', c') \to (r, c)\right]$ is the probability the rat moves from (r', c') to (r, c) at time $t + 1$.

Results:



Observations:

In this graph, we notice that Bot 2 takes on average, fewer time steps than Bot 1, which is reasonable. It makes sense that more heavily weighing for future probabilities during the path planning process helps better track down a moving rat.