CS520: INTRODUCTION TO AI

Project 2 Report

Space Rats

By,

Chiranjeevi Bhuvaneshwari Pamarthy [cp1270]

Shweatha Bathina Mallikarjuna [sb2808]

Table of Contents

1.Abstract

In this project, we simulate a bot navigating a grid-based spaceship to locate and catch a moving target space rat. The ship is represented as a **30×30** grid composed of open and closed cells, where the bot can move only through open paths. The bot starts with no knowledge of the rat's location and relies on limited sensor data and probabilistic reasoning to make informed decisions. We implemented two bot strategies: a baseline bot using *belief-based pathfinding* and an improvised version that builds on it by incorporating smarter, utility-driven movement. One of the key techniques involved maintaining a belief matrix to estimate the rat's possible locations and updating it based on ping feedback. The aim was to explore how improved decision-making logic can enhance performance in a dynamic and uncertain environment. This report details the bot designs and their behaviors across different scenarios within the simulated ship.

2. Introduction

2.1 Ship Generation – The Environment

To set up the spaceship environment for the Space Rats project, we generate a 30×30 grid with a mix of walls and open paths. Unlike the previous fire-based simulation, there is no fire and no suppression button. Instead, the focus is on bot localization and target tracking. All outer edge cells are blocked to ensure the bot, and the rat remain within the ship.

How the Ship is Created

1. **Creating Paths and Walls**
    a. The grid is initialized with random placement of walls and open spaces inside the 30×30 area.
    b. The outer boundary (first and last row/column) is completely blocked to prevent out-of-bounds movement.
2. **Fixing Dead Ends**
    a. We ensure the grid has no isolated regions by checking that all open cells are reachable.
    b. If there are any completely enclosed or unreachable areas, some walls are removed to maintain solvability.

3. **Placing the Bot and the Space Rat**
    a. The bot's actual starting location is randomly chosen among open cells, but the bot itself does not know its position initially.
    b. The space rat is also placed randomly in another open cell, separate from the bot's starting location.
    c. This randomized placement makes each simulation unique while preserving fair conditions for testing.

4. **Saving and Using the Layouts**

    a. Each generated layout is checked to ensure it is valid and that there is enough space for the bot to move and explore.
    b. These layouts are saved and reused across multiple simulation runs for consistency in performance evaluation.
    c. The grid uses the following encoding: walls (0), open paths (1); the bot and rat positions are tracked separately in the simulation logic and not marked directly on the grid.

2.2 The Rat

After setting up the environment, the space rat is placed randomly in one of the open cells, making sure it doesn't start too close to the bot. In the stationary version of the simulation, the rat stays in the same spot the whole time, which makes it easier for the bot to eventually find and catch it. But in the moving version, things get trickier—the rat moves to a nearby open cell after each move the bot makes. This means the bot must keep updating its guess about where the rat might be, all while relying on limited sensor data. This change makes the problem more challenging and gives us a better way to test how smart and adaptable the bot really is.

2.3 Bots

In this project, the bot goes through two main phases. In **Phase 1**, it must figure out where it is on the ship since it starts off completely disoriented. Once it knows its exact location, it moves on to **Phase 2**, where it starts searching for the space rat using its sensors. All four of our bot versions follow this same structure, but the way they think and respond in each phase differs depending on how smart they are and whether the rat is stationary or moving.

Phase 1 – Figuring Out Where the Bot Is

At the beginning of each simulation, the bot doesn't know where it has been placed on the ship. All it knows is what the ship looks like and how many walls are next to it; that's what the sensor tells it. It starts by assuming it could be in *any* open cell. From there, it begins to eliminate possibilities.

The bot alternates between sensing and trying to move. First, it checks how many walls are next to it and rules out any locations that don't match that pattern. Then it tries to move in a direction — if the move is successful, it eliminates all locations where that move would have been blocked. If the move fails, it eliminates places where it should've been able to move. It keeps doing this until it's sure where it is. That's when it moves on to Phase 2.

Phase 2 – Finding and Catching the Space Rat

Once the bot knows where it is, it starts using its space rat sensor. This sensor gives a ping when the bot is near the rat — the closer the bot is, the stronger the signal (depending on a parameter called alpha, α). The tricky part is that the ping doesn't always give a clear answer, especially if the rat is far away or α is very small or very large.

To handle this uncertainty, the bot keeps a "belief" map — a grid where it tracks how likely it is that the rat is in each cell. Every time it uses the sensor and either hears a ping or not, it updates that belief. Then it moves toward the cell that currently seems the most likely to contain the rat. If it ends up in the same cell, it catches the rat.

When the rat is moving, things get more complicated. Now the bot must predict where the rat might go next and adjust its belief map accordingly. This makes Phase 2 much more challenging and tests how well the bot can handle uncertainty and change.

The Four Bot Versions

Bot 1 – Baseline Bot, Stationary Rat

This version of the bot follows a straightforward two-step loop: sense and move. After receiving a sensor reading, it updates its belief matrix to reflect the likelihood of the rat's location and moves toward the cell with the highest belief using Breadth-First Search (BFS). Since the rat doesn't move in this scenario, the bot eventually catches it, especially if the sensor provides reliable feedback. However, the bot doesn't consider movement cost or uncertainty, it purely reacts to the belief values, which can lead to inefficient wandering if the sensor is noisy.

Bot 1 – Baseline Bot, Moving Rat

In this setup, the same simple logic is applied, but the rat now moves randomly after every bot action. The bot continues to rely solely on the latest sensor feedback without accounting for the fact that the rat may have already moved. As a result, it often chases outdated information and wastes time moving toward previously high-belief cells that are no longer relevant. Without any belief spreading or transition modeling, the bot's efficiency drops noticeably, highlighting the limitations of a purely reactive approach in a dynamic environment.

Bot 2 – Smarter Bot, Stationary Rat

The smarter bot enhances the original strategy by incorporating a utility-based decision-making model. It still updates its belief matrix using the sensor and distance-based probability, but it also evaluates the cost of moving to each potential target. Instead of blindly selecting the highest belief cell, the bot considers the expected utility of each move, balancing belief confidence with movement effort. In a stationary setup, this thoughtful planning makes the bot more efficient and accurate, especially when the sensor is unreliable or misleading.

Bot 2 – Smarter Bot, Moving Rat

This version combines belief-based reasoning with a model of the rat's motion to handle the most complex scenario. After sensing, the bot not only updates its belief based on sensor data but also performs a belief propagation step to simulate where the rat could have moved. It spreads belief to neighboring open cells, capturing the uncertainty caused by the rat's mobility. The bot then applies its utility function to choose the best next step, accounting for both probability and movement cost. As a result, it adapts well to change, avoids chasing stale information, and performs more consistently than any of the other versions.

3. Data Analysis:

3.1 Updating the Rat's Belief Matrix (Knowledge Base)

The bot maintains a **belief matrix**, which represents the probability of the space rat being in each cell of the 30×30 grid. At every time step, this matrix is updated based on whether the bot receives a **ping** or not from its sensor. The ping indicates whether the bot is currently in the same cell as the rat.

Case 1: Ping Received

If the bot receives a ping, it confirms that the rat is in the same cell as the bot. In this case, the belief matrix is updated with complete certainty:

- Set the probability of the current cell (bot's location) to **1**.
- Set the probability of all other cells to **0**.

Formula:

If the bot is at position *(x, y)*:

$$\text{belief[i][j]} = \begin{cases} 1 & if\ (i,j) = (x,y) \\ 0 & otherwise \end{cases}$$

Case 2: No Ping Received

If the bot does **not** receive a ping, it knows the rat is **not** in the same cell. Therefore:

1. Set the probability of the current cell to **0**.
2. Renormalize the rest of the matrix so that the total probability sums to **1**.

Steps:

- Let the bot be at position *(x, y)*.
- First, update the current cell:

$$\text{belief}[x][y] = 0$$

- Then, calculate the sum of the remaining probabilities:

$$\text{total} = \sum_{(i,j)\neq(x,y)} \text{belief}[i][j]$$

- Finally, normalize all other cells:

$$\text{belief}[i][j] = \frac{\text{belief}[i][j]}{\text{total}}, \quad \text{for all } (i,j) \neq (x,y)$$

This update process allows the bot to maintain a realistic estimate of the rat's possible location, adjusting its strategy based on sensor feedback at every step.

3.2 Bot – 2 [Improvised based on baseline]

Sensor Model, Belief Update, and Movement Decision Logic

In this simulation, the bot is designed to find and catch a space rat hidden somewhere on a 30×30 grid. The bot doesn't know where the rat is, so it relies on a combination of sensor feedback, belief tracking, and smart decision-making to guide its actions.

Sensor Model and Ping Probability

To estimate how close the rat is, the bot uses a simulated sensor that returns a "ping" based on the distance to the rat. If the rat is in the same cell as the bot, the ping is guaranteed. As the distance increases, the chance of getting a ping decreases.

This behavior is modeled using an exponential decay function:

$$P(\text{ping} \mid d(i,j)) = e^{-\alpha(d(i,j)-1)}$$

Where:

- d (i, j) is the Manhattan distance between the bot and the rat.
- α is a sensitivity constant that controls how quickly the ping probability drops off with distance.

If the bot is in the same cell as the rat (d=0d = 0d=0), it always gets a ping. For all other distances, the bot calculates the ping probability using the formula and compares it with a random value to decide whether a ping is received.

Belief Update Based on Sensor Feedback

The bot maintains a **belief matrix**, which stores the probability that the rat is in each open cell on the grid. After each sensor reading, this matrix is updated using Bayesian inference.

For each cell (i, j), the update works as follows:

- **If a ping is received:**

$$\text{belief}[i][j] = P(\text{ping} \mid d(i,j)) \times \text{belief}[i][j]$$

- **If no ping is received:**

$$\text{belief}[i][j] = (1 - P(\text{ping} \mid d(i,j))) \times \text{belief}[i][j]$$

Once all cells are updated, the belief matrix is normalized so that the total probability sums to 1. This ensures that the matrix remains a valid probability distribution. If the total probability becomes zero (due to rounding or extreme values), the bot resets the matrix with equal values across all open cells.

This process helps the bot refine its understanding of where the rat might be, even when the sensor data is uncertain or noisy.

Movement Decision Logic

After updating the belief matrix, the bot needs to decide where to move next. It doesn't just go to the highest-probability cell—it also considers how far that cell is and how risky the move might be.

To make this decision, the bot calculates an **expected cost** for every possible destination using:

$$\text{Expected Cost} = \text{Distance}+1+(1-\text{Probability}) \times C$$

Where:

- **Distance** is the Manhattan distance to the cell.
- The extra **1** account for the cost of performing a sensor check.
- **Probability** is the belief value for that cell.
- **C** is a fixed cost if the rat isn't there (set to 10 in this simulation).

This formula helps the bot choose a location that has a high chance of success but doesn't require too much effort to reach. After identifying the cell with the lowest expected cost, the bot uses Breadth-First Search (BFS) to find a path to that cell and moves one step in that direction.

This process repeats after every move, allowing the bot to continuously adapt based on new information and changing conditions (like if the rat moves).

This combination of probabilistic sensing, belief updates, and utility-based movement helps the bot perform efficiently even when dealing with uncertainty, limited information, and a dynamic environment.

3.3 Evaluation and Comparison of Bots

Bot 1 Performance – Stationary Rat

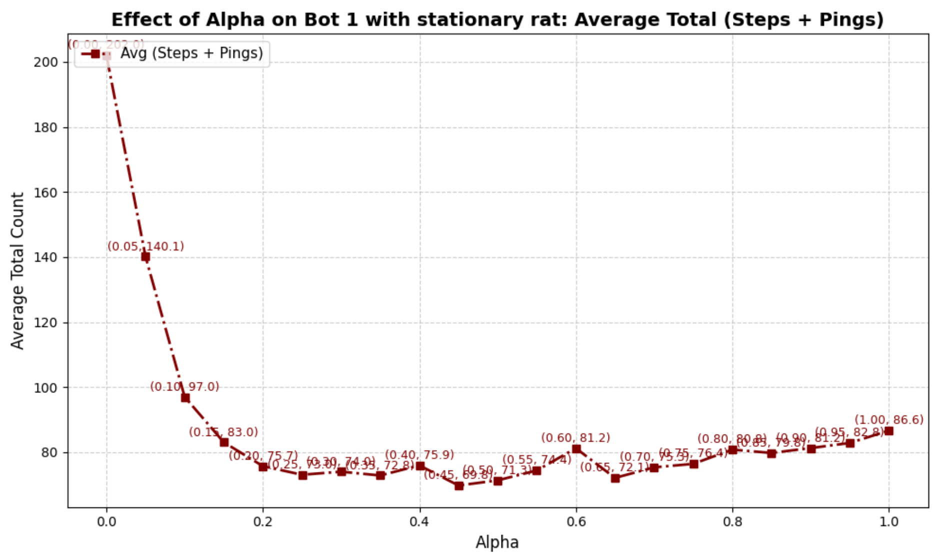

Fig. 3.3.1 Bot 1 – Stationary Rat: Path and Belief Visualization [Combined Performance: Total Effort (Steps + Sensors)]

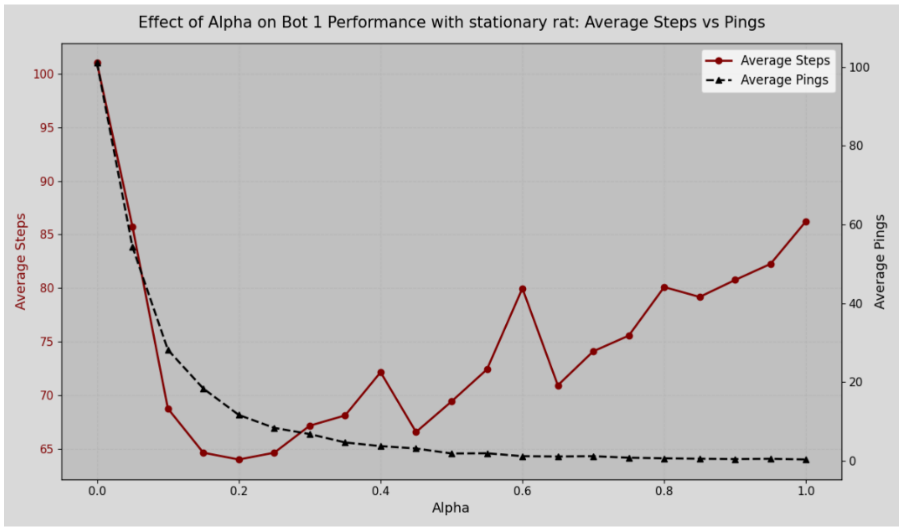

Fig. 3.3.2 Bot 1 – Stationary Rat: Path and Belief Visualization

- **Low α (0.00–0.10):**
  Too many pings → belief matrix stays uniform → bot moves randomly → high effort (~200 steps).
- **Moderate α (0.30–0.55):**
  Sensor becomes informative → belief updates improve → performance peaks at α = 0.45 (lowest total cost).
- **High α (> 0.60):**
  Few or no pings → feedback too sparse → bot guesses → effort increases again.
- **Overall Insight:**
  Bot 1 is most effective when α is in the moderate range, where sensor feedback is balanced — not too noisy or too quiet.

Bot 2 Performance – Stationary Rat

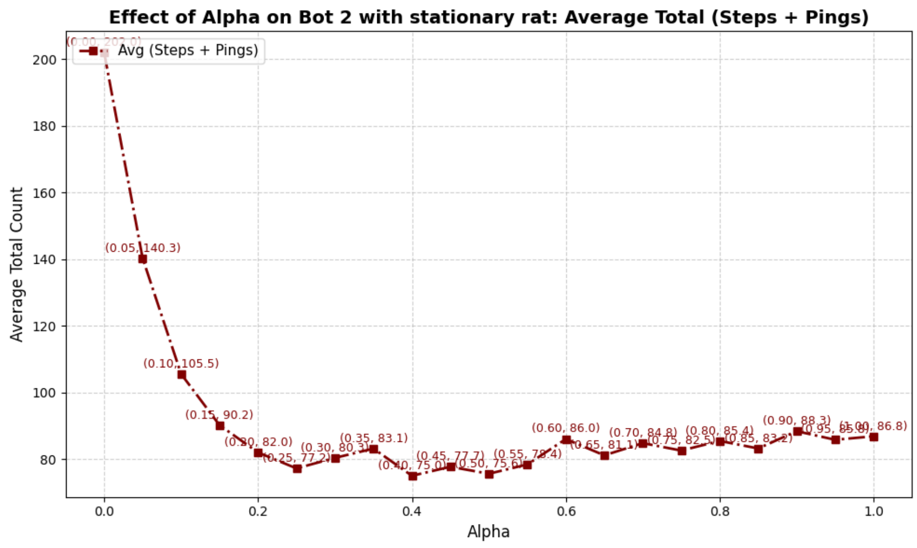

Fig. 3.3.3 Bot 2 – Stationary Rat: Path and Belief Visualization [Combined Performance: Total Effort (Steps + Sensors)]

- **Low α (0.00–0.10):**
  Sensor gives pings too often → belief updates aren't helpful → bot struggles like Bot 1 → total cost near max.
- **Moderate α (0.30–0.55):**

  Sensor feedback is reliable → bot uses both belief and movement cost → best performance around α = 0.45–0.55.

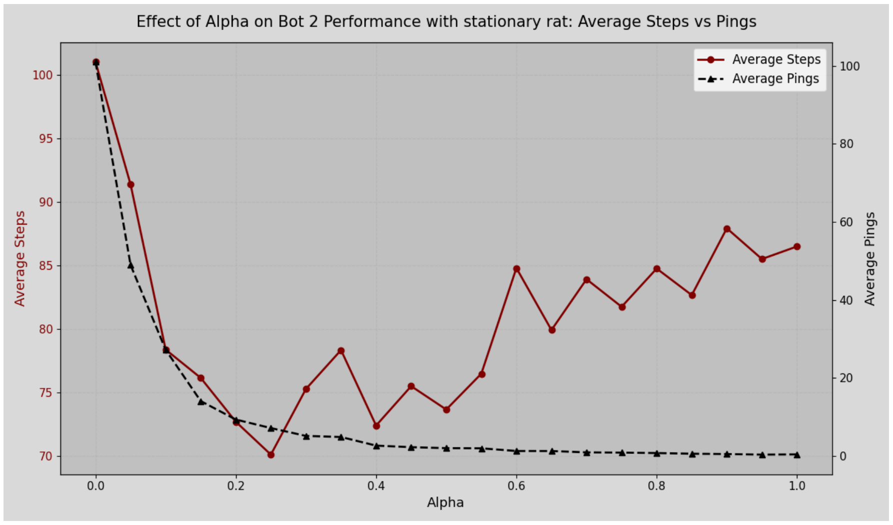

Fig. 3.3.4 Bot 2 – Stationary Rat: Path and Belief Visualization

- **High α (> 0.60):**

  Sensors become less informative → fewer pings → belief updates slow down, but bot still avoid wasteful moves.
- **Overall Insight:**

  Bot 2 performs consistently well across all α values and handles uncertainty better by balancing probability with cost.

3.4 Tracking a Moving Rat – Belief Updates, Bot Performance & Improvements

(i) Updating the Belief When the Rat Moves

In the earlier version of the simulation, the rat remained stationary, so updating the belief matrix based only on sensor feedback was sufficient. However, when the rat starts moving randomly to one of its neighboring open cells after every bot action, the belief matrix needs an additional update to reflect this uncertainty.

To address this, I introduced a **motion update** that spreads the belief from each cell to its adjacent open neighbors. This captures the idea that the rat could have moved into any of those neighboring cells.

The belief for each cell (i, j) is updated using the following formula:

$$\text{new\_belief}[i][j] = \sum_{(x,y)\in \text{Neighbors}(i,j)} \frac{\text{belief}[x][y]}{N(x,y)}$$

Here:

- (x, y) are the neighboring cells of (i, j),
- N (x, y) is the number of open neighbors that cell (x, y) can move to.

This formula helps the bot maintain a realistic and distributed belief, even when the rat is constantly on the move.

(ii) Simulation and Comparison of Bot Performance

I tested both Bot 1 (baseline) and Bot 2 (utility-based) in this updated scenario where the rat moves randomly after each bot action. I also evaluated performance across a range of α values greater than zero, which controls how reliable the sensor feedback is.

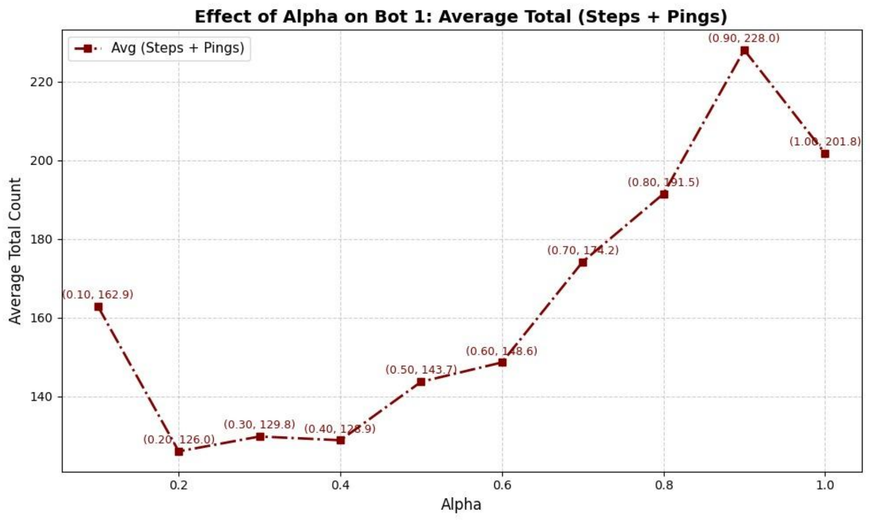

Fig. 3.4.2 Bot 1 – Moving Rat: Path and Belief Visualization [Combined Performance: Total Effort (Steps + Sensors)]

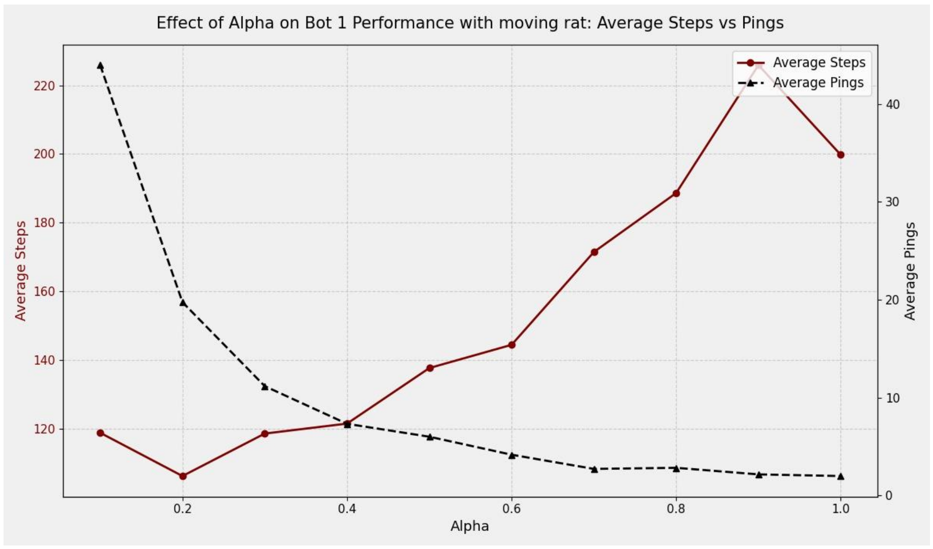

Fig. 3.4.3 Bot 1 – Stationary Rat: Path and Belief Visualization

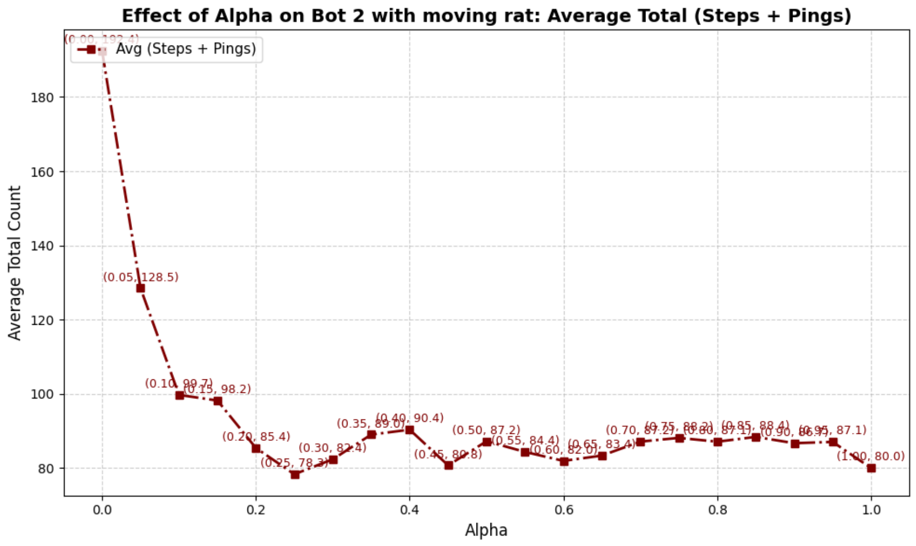

Fig. 3.4.4 Bot 2 – Moving Rat: Path and Belief Visualization [Combined Performance: Total Effort (Steps + Sensors)]

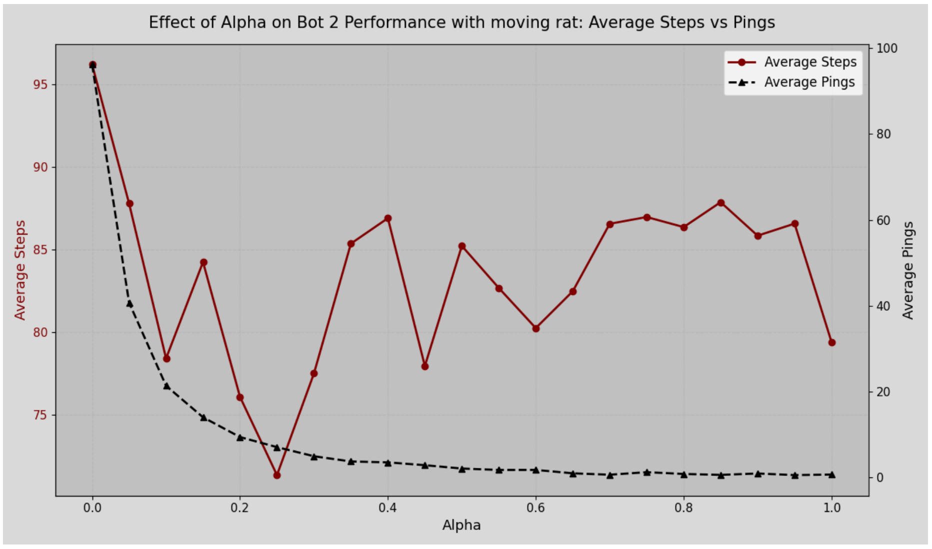

Fig. 3.4.4 Bot 2 – Moving Rat: Path and Belief Visualization

Bot 1 (Baseline):

- Bot 1 struggled in the dynamic setup.
- It continuously chased the highest belief cell but often arrived after the rat had already moved.
- As a result, it wasted many steps and failed to adapt effectively to the rat's movement.

Bot 2 (Utility-Based):

- Bot 2 handled the uncertainty much better.
- It balanced both the belief probability and movement cost in its decision-making.
- This allowed it to avoid distant or low-value targets and adapt more effectively to the rat's behavior.

Influence of α:

- At **low α values** (e.g., 0.00–0.10), the sensor triggered frequently, even when the rat was far. This led to overly uniform belief distributions and poor targeting.
- At **high α values** (above 0.60), the sensor rarely provided feedback, leading to a lack of belief refinement and more guessing.
- The best results were seen in the **moderate α range (0.30–0.55)**, where the sensor gave reliable feedback. In this range, both bots improved, but Bot 2 remained significantly more consistent.

(iii) Improving Bot 2 for the Moving Rat

To make Bot 2 more effective in this new environment, I have implemented the following improvements:

- **Added the motion update step** to spread belief after each rat movement.
- **Increased sensor usage frequency** to keep the belief matrix focused and current.

- **Adjusted the utility function** to give slightly more weight to belief confidence and reduce hesitation in committing to strong candidates.

3.5 Conclusion

These enhancements helped Bot 2 remain accurate and efficient, even in an environment where the target is constantly shifting. Compared to Bot 1, it showed better adaptability, reduced unnecessary movement, and overall stronger performance—especially within the optimal α range.