CS 520

INTRO TO ARTIFICIAL INTELLIGENCE

PROJECT 2 - SPACE RATS

Team Members

1. Tejaswini Abburi (ta633)
2. Prajwal Srinivas (ps1458)

Contents

Problem Statement

A blast of cosmic radiation has wiped some of the bot's memory, and it no longer knows where it is in the ship. It has access to a map, but it does not know its current location in the ship. Additionally, a space rat has gotten into the ship and needs to be caught.

The bot must identify its location, navigate through the ship, locate the space rat, and catch it.

Project Overview

The project design for Bots consists of two main phases:

Phase 1: Locating the bot's position within the grid.

Phase 2: Tracking and capturing the space rat using probabilistic models.

At every timestep, the bot can perform one of three actions:

1. Blocked Cell Sensing: Determines the number of blocked neighboring cells.
2. Space Rat Detector: A sensor that pings based on the bot's distance from the rat, following an exponential probability function.
3. Movement Mechanism: Moves in cardinal directions (Up, Down, Left, Right), updating its position knowledge.

The ultimate goal is to design and test the bot, then evaluate its performance compared to a baseline bot using various values of the detector sensitivity parameter α .

Approach

Spaceship Layout

The spaceship is represented as a 30x30 grid, by keeping all outer edges blocked. The grid and layout environment is initialized using the *SpaceshipEnvironment* class, which creates a grid with randomly placed blocked cells. The bot and the rat are placed in random open cells. The position of bot, rat are unknown yet.

class `SpaceshipEnvironment` encapsulates the entire grid generation, placing agents, Track path, knowledge_base_history

1. *initialize_grid()* method creates a randomized spaceship layout.

   This method ensures:
   - Outer walls are always blocked (set to 0).
   - The inner open cells are dervied from project 1 with randomly opening inner cells and following up with opening its neighboring cells.

2. *place_agents()* method places Bot, Rat at random positions.

   This method ensures:
   - If Specified, we can place Bot and Rat at designated cell, Otherwise, both bot and rat are placed randomly in open cells.
   - Additionally, There's a check condition added to not place bot and rat at same cell.

Identification of Bot Position

In this phase, the bot identifies its location within the grid using:

1. Blocked Neighbor Sensing: The bot can look at the eight cells surrounding it and count how many are blocked.
2. Movement Feedback: As the bot tries to move around, it keeps track of whether those moves are successful or not. If a move fails, that also provides information about which positions are unlikely to be its current location.

Algorithm logic

1. Knowledge Base Initialization: The bot starts by assuming all open cells are possible locations for it.
2. Sense Blocked Neighbors: It then detects how many of the surrounding cells are blocked and uses that information to eliminate impossible positions from its knowledge base.
3. Attempt Movement: the bot tries moving in the direction with the most open cells. If the move succeeds, it can rule out positions that are inconsistent with that movement. If the move fails, it can also use that feedback to narrow down its location.

Mathematics and Bayesian Update:

$$K = \{k \in K : N_{\text{blocked}}(k) = \text{observed value}\}$$

For example: If the bot senses 3 blocked neighbors, all cells inconsistent with this observation are removed from the knowledge base.

Navigating to Space Rat

The probabilistic approach is used to track the rat's location within the grid.
(self.rat_probablities) function is used to maintain a grid for the likelihood of the rat being in each cell. The bot moves towards cells with higher probabilities of containing the rat, while also considering exploration and avoiding recently visited cells

Normalization:
After each movement update, the probability distribution must be normalized to ensure that the sum of probabilities over the entire grid equals 1. The normalization formula is:

$$P_{\text{new}}(x, y) = \frac{P_{\text{new}}(x, y)}{\sum_{i=1}^{N} \sum_{j=1}^{N} P_{\text{new}}(i, j)}$$

In this equation:

- The numerator $P_{\text{new}}(x, y)$ represents the updated probability for cell (x, y).
- The denominator $\sum_{i=1}^{N} \sum_{j=1}^{N} P_{\text{new}}(i, j)$ is the total sum of probabilities across all cells in the grid.

This ensures that the sum of all updated probabilities across the entire grid equals 1, thereby maintaining a valid probability distribution.

Sensor Ping Detection Update:

When the bot detects the rat nearby through a sensor "ping," it increases the probability in the cells near the detection point. Let's say the sensor detects a ping at cell (x_d, y_d), and the probability update due to this ping is governed by the distance d from (x_d, y_d):

$$P_{\text{new}}(x, y) = P_{\text{current}}(x, y) \cdot e^{-\beta \cdot d(x,y)}$$

where:

- $d(x, y)$ is the distance from the detection cell (x_d, y_d), often calculated using the Euclidean distance: $d = \sqrt{(x - x_d)^2 + (y - y_d)^2}$.
- β is a decay constant that controls how rapidly the probability decreases with distance from (x_d, y_d).

The probability distribution is again normalized after this update.

Heatmap Visualization:

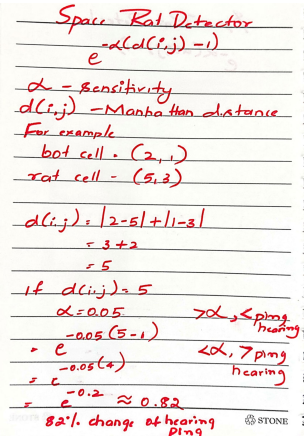
The final distribution can be visualized as a heatmap. Cells with higher values of $P(x, y)$ appear darker, indicating higher confidence in the rat's location. The updated distribution after each step informs the bot's next move.

Space rat detector (Ping/No Ping)

The Space rat detector returns a ping or no ping depending on the closeness of the space rat with the bot. More specifically, it provides information in regard to the identity of the same cell occupied by the bot and the space rat. A probabilistic approach is used to realize this. The probability of receiving a ping is determined by the Manhattan distance between the bot and the rat.

$$e^{-\alpha(d(i,j)-1)}$$

α is the detector's sensitivity constant for all $\alpha > 0$
$d(i, j)$ is the Mahattan distance between bot cell(i) and rat cell(j)



For example:

$$\text{bot cell} = (2, 1)$$

$$\text{rat cell} = (5, 3)$$

Calculating the Manhattan distance:

$$d(i, j) = |2 - 5| + |1 - 3|$$
$$= 3 + 2 = 5$$

If $d(i, j) = 5$ and $\alpha = 0.05$:

$$e^{-\alpha(d(i,j)-1)} = e^{-0.05(5-1)}$$
$$= e^{-0.05\times4} = e^{-0.2} \approx 0.82$$

This indicates an 82% chance of hearing a ping.

Enhanced Bot Implementation

Enhanced bot uses most sophisticated strategy by utilizing probabilistic reasoning and adaptive movement.For every potential move, it calculates a utility score, weighing multiple factors to make the optimal choice

Movement Decision

There are few actions that Enhances bot takes into consideration.

- Rat Probability: The bot considers the likelihood of the rat being in each nearby cell.
- Visit Frequency: The bot wouldn't waste time repeatedly searching the same location, our bot keeps track of where it's been. It assigns lower utility to frequently visited cells, encouraging exploration of new areas.

- Track Recent Positions: To avoid getting stuck in loops, the bot maintains a short-term memory of its recent positions. It's like leaving breadcrumbs – the bot knows where it's just been and avoids backtracking unnecessarily.

The utility is calculated as:

$$utility = (rat\_probability * 0.7) + (visit\_penalty * 0.3)$$

Enhanced bot uses a utility-based approach to decide its next move. This decision-making process balances exploration and exploitation, considering the probability of the rat's presence and the frequency of previous visits to each cell. This avoid repetitive paths very often.

```
def calculate_move_utility(self, move):
    dx, dy = move
    x, y = self.env.bot_true_position
    new_pos = (x + dx, y + dy)
    if new_pos in self.last_positions:
        return -3
    visit_count = self.visit_counts[new_pos[0], new_pos[1]]
    visit_penalty = math.exp(-0.5 * visit_count)
    rat_prob = self.rat_probabilities[new_pos[0], new_pos[1]]
    if self.env.moving_rat:
        return rat_prob * 0.4 + visit_penalty * 0.6
    else:
        return rat_prob * 0.7 + visit_penalty * 0.3
```

Initial Probability Distribution:
If the rat's initial position is unknown, the probability $P(x, y)$ for each cell in an $N \times N$ grid is:

$$P(x, y) = \frac{1}{N^2}$$

where N is the grid size. This assumes a uniform distribution across all cells.
If the initial position (x_0, y_0) is known:

$$P(x, y) = \begin{cases} 1, & \text{if } (x, y) = (x_0, y_0) \\ 0, & \text{otherwise} \end{cases}$$

Movement Update Formula:
As the rat moves, the probability of its location spreads from each cell (x, y) to its neighboring cells. The movement update formula is:

$$P_{\text{new}}(x, y) = (1 - \alpha) \cdot P_{\text{current}}(x, y) + \frac{\alpha}{N_{\text{neighbors}}} \sum_{(x', y')} P_{\text{current}}(x', y')$$

Where:

- α is a decay factor (a value between 0 and 1) that controls how much probability spreads out to neighboring cells.
- $N_{\text{neighbors}}$ is the number of possible neighboring cells (e.g., 4 if the rat can only move up, down, left, or right).
- $P_{\text{current}}(x, y)$ is the current probability at cell (x, y).
- $P_{\text{current}}(x', y')$ represents the probability of neighboring cells (x', y') that the rat could move to from (x, y).

This formula redistributes part of the probability from each cell to its neighbors, simulating the rat's movement.

Probabilistic Knowledge Base

After each detection attempt, Enhanced Bot updates its knowledge base and it maintains a grid of probabilities for rat location, Updates probabilities based on ping detector readings and accounts for sensor uncertainty using exponential decay model

Initial Distribution: When the search begins, the bot assumes the rat could be anywhere, spreading probability evenly across all open cells.

Update Mechanism: After each action and sensor reading, the bot updates its knowledge. It's like a constantly evolving heat map of rat likelihood.

Sensor Integration: When the bot's sensor pings, it doesn't just narrow down to one location. Instead, it increases probabilities in a radius around the ping location, with closer cells getting a higher boost. The mathematical model behind this is an exponential decay function:

$$P(x, y) = P(x, y) \times e^{-\beta d}$$

Where:

- d is the distance from the ping location.
- β is a decay constant.

Adaptive Behavior

Adjusts strategy based on ping detection history, modifies exploration rate based on search progress, Implements memory of recent positions to avoid loops

Dynamic Exploration Rate: As the search progresses, the bot adjusts how much it values exploration versus exploitation. Early on, it might favor covering new ground, but as it

gathers more information, it becomes more focused on high-probability areas.

Pattern Recognition: The bot learns to recognize patterns in the rat's movement. If it notices the rat tends to move in a certain direction, it adjusts its predictions accordingly.

Efficiency Improvements: Over time, the bot learns to make more efficient movements, reducing unnecessary steps and optimizing its path through the spaceship.

This answer the question, design and algorithm for the Enhanced Bot, being as specific as How does your bot make use of the information available to make informed decisions about what to do next?

Baseline Bot Implementation

As per the project description, Baseline bot operates in two phases

Phase 1: Self- Localization

In this Phase, the bot will begin to identify its location on the ship and it alterates between sensing blocked neighbors and moves and it keep updating its knowledge base of locations in each action.

1. Maintain a knowledge base of possible bot locations.
2. Alternate between sensing blocked neighbors and moving:
    - Sensing: Eliminate locations that don't match the sensed data.
    - Moving: Choose the most commonly open direction and attempt to move.
    - Update knowledge base based on successful or unsuccessful moves.
3. Repeat until the bot's location is uniquely identified.

Phase 2: Rat Tracking

Once the bot knows its location, it focuses on finding the rat. The knowledge base for the rat's location is represented as a probability distribution over the grid.

Bayes theorem is applied to adjust the probabilities.

1. Maintain a probability distribution for the rat's location.

    Probability Updates for Rat Location

    For a cell (x, y), let:

    $$P(R_{x,y}) \quad \text{be the prior probability of the rat being in cell } (x, y)$$

    $$P(\text{ping} \mid R_{x,y}) \quad \text{be the probability of a ping given the rat is in } (x, y)$$

Using Bayes' theorem, we can update our assumption after a ping:

$$P(R_{x,y} \mid \text{ping}) = \frac{P(\text{ping} \mid R_{x,y}) \cdot P(R_{x,y})}{P(\text{ping})}$$

And after no ping:

$$P(R_{x,y} \mid \text{no\_ping}) = \frac{P(\text{no\_ping} \mid R_{x,y}) \cdot P(R_{x,y})}{P(\text{no\_ping})}$$

Where:

$$P(\text{ping}) = \sum_{x,y} P(\text{ping} \mid R_{x,y}) \cdot P(R_{x,y})$$

$$P(\text{no\_ping}) = 1 - P(\text{ping})$$

Code Implementation :

```
def update_rat_knowledge(self, got_ping):

    new_probs = np.zeros_like(self.rat_probabilities)
    bot_x, bot_y = self.env.bot_true_position

    for i in range(self.env.size):
        for j in range(self.env.size):
            if self.env.grid[i, j] == 1:
                distance = abs(i - bot_x) + abs(j - bot_y)
                if distance == 0:
                    ping_prob = 1.0
                else:
                    ping_prob = math.exp(-self.alpha * (distance - 1))

                if got_ping:
                    new_probs[i, j] = ping_prob * self.
                    rat_probabilities[i, j]
                else:
                    new_probs[i, j] = (1 - ping_prob) * self.
                    rat_probabilities[i, j]

    if new_probs.sum() > 0:
        new_probs /= new_probs.sum()

    self.rat_probabilities = new_probs
```

This answer the question, How the space rat knowledge base should be updated, based on whether the bot receives a ping or does not receive a ping.

2. Alternate between using the rat detector and moving:

   - Detector: Update probabilities based on ping/no ping.
   - Moving: Move towards the most likely rat location.

3. Repeat until the rat is found (same cell as the bot).

Moving Rat Scenario

Probability Calculation factors

Breakdown of how probabilities are calculated and updated for a moving rat scenario are as follows:

Initial Probability Distribution:
Uniform Probability: Initially, the probability of the rat being in any cell is uniformly distributed across the grid, assuming we don't know its exact position. For example, in a 30x30 grid, the initial probability for each cell might be set to 9001, assuming equal likelihood.

Known Position:
If the initial position of the rat is known, the probability distribution starts with a high probability at that position and very low probabilities elsewhere. Movement Update

Movement Patterns:
Since the rat is moving randomly, the probability distribution spreads out in each direction where the rat could move. In a grid, the rat can typically move up, down, left, or right. This randomness is modeled by spreading probability from the previous cell into its neighboring cells.

Decay Factor:
With each step, the probability that the rat remains in a particular cell decays slightly. For example, if the rat has a probability of being in cell $(5, 5)$, and it can move to adjacent cells, that probability is divided and distributed across adjacent cells (like $(4, 5), (6, 5), (5, 4), and (5, 6)$) with a decay factor α.

Probability Propagation with Alpha α:
Alpha Influence: The user-selected alpha (α) controls how quickly probability spreads from one cell to adjacent cells. A lower α (like 0.05) means the probability spread is smaller, making it likely that the rat is still near its last known position. A higher α (like 0.2) spreads the probability faster, accounting for the rat moving farther.
Neighboring Probability Distribution: For each step, probability is redistributed:

$$P_{\text{new}}(x, y) = (1 - \alpha) \cdot P_{\text{current}}(x, y) + \frac{\alpha}{N} \sum_{(x', y')} P_{\text{current}}(x', y')$$

Sensor Feedback and Ping Detection:
Ping Probability Boost: When a sensor detects the rat nearby, a "ping" detection signal boosts the probability in that region of the grid. If the bot detects a ping in a specific cell, it updates the probability for nearby cells based on the signal strength, usually assigning a higher probability to the cells closer to the bot.

Probability Recalculation:

When a ping is detected, probabilities are adjusted to reflect this new data, increasing the likelihood in nearby cells and decreasing it elsewhere.

Heatmap Visualization:
The probability matrix is visualized as a heatmap to help track the rat's probable location. Darker regions in the heatmap indicate higher probabilities, guiding the bot's search efforts.

In summary, in the moving rat scenario:
Initial distribution is set across the grid or around the known starting position. Probabilities spread across adjacent cells with each movement based on α. Sensor pings refine the distribution, concentrating probability near detected locations. This constantly updated distribution helps the bot predict the rat's likely location as it moves.

1. Rat Movement: The move_rat method in SpaceshipEnvironment implements random rat movement with a 30% chance of moving each step.

2. Knowledge Base Update: The update_rat_knowledge method is modified to spread probabilities to neighboring cells after each update.

   Formula used :

$$P(\text{rat}_{t+1} = i) = \sum_j P(\text{rat}_{t+1} = i | \text{rat}_t = j) \cdot P(\text{rat}_t = j)$$

   Where $P(\text{rat}_{t+1} = i | \text{rat}_t = j)$ is the transition probability from cell j to i.

```
if self.env.moving_rat:
  spread_probs = new_probs.copy()
  for i in range(1, self.env.size-1):
      for j in range(1, self.env.size-1):
          if self.env.grid[i, j] == 1:
              for di, dj in [(0,1), (1,0), (0,-1), (-1,0)]:
                  ni, nj = i + di, j + dj
                  if self.env.grid[ni, nj] == 1:
                      spread_probs[ni, nj] += new_probs[i, j] * 0.1
  new_probs = spread_probs
```

Data and Analysis

Evaluation of Enhanced Bot vs BaseLine Bot

The performance analysis for Enhanced Bot and Baseline bot are significantly influenced by the parameter α as it determines the sensitivity of the space rat detector. Hence in order to potentially challenge the bots, α factor is set between 0.05 to 0.2.

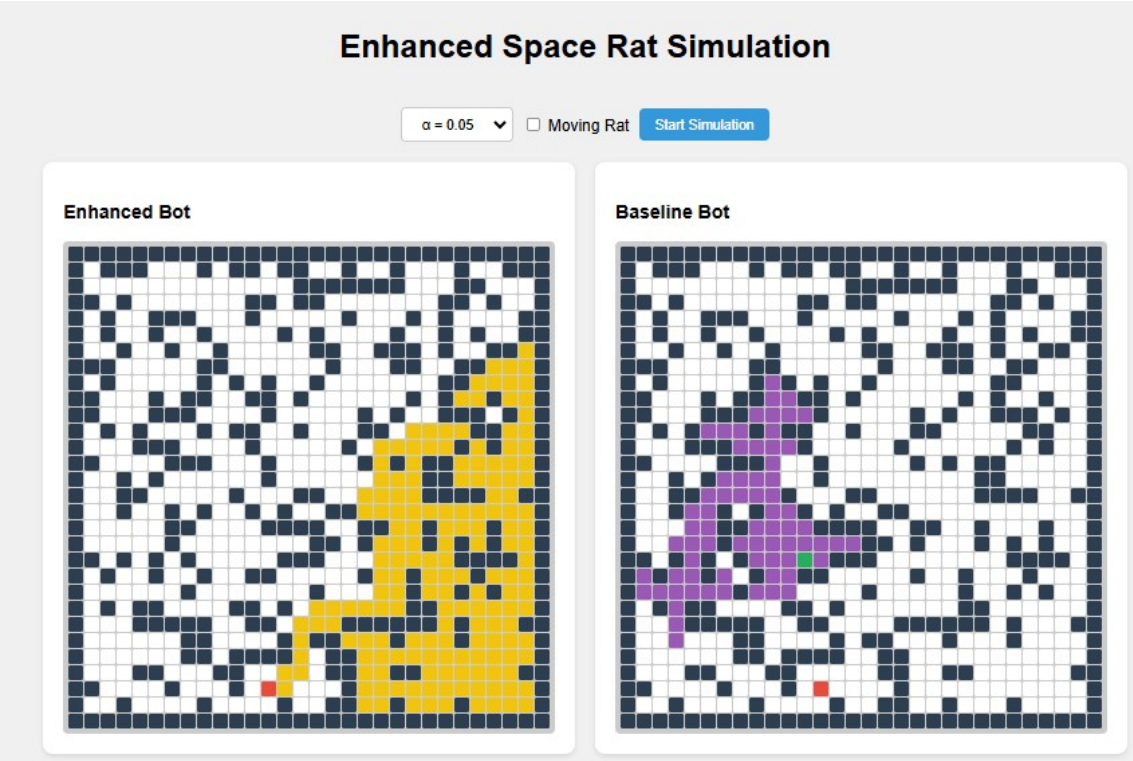


Figure 1: Ship Layout

Figure 1 is the Ship layout that includes unknown positions for the Bot and Rat. There are two comparison grid layouts referring to the Enhanced Bot grid and the Baseline Bot grid. The α factor is set to 0.05 as default and then for the simulation we take 20 interval points for the value of α.

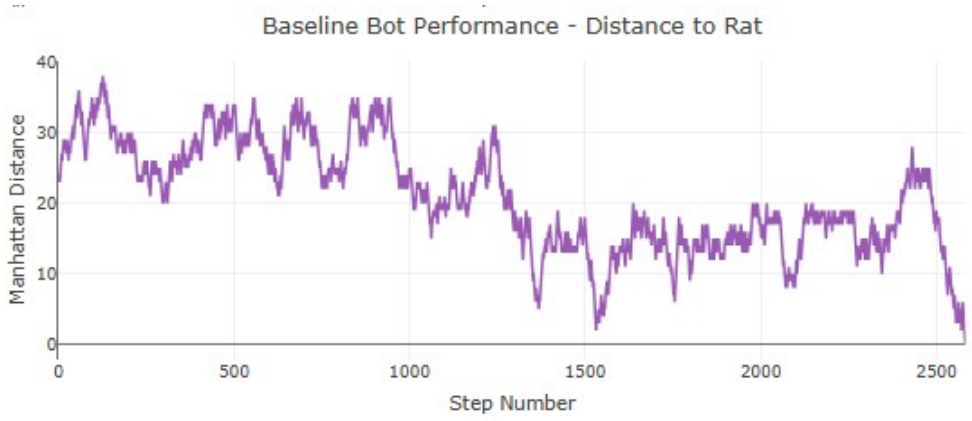


Figure 2: Graph with uncertain interpretation.

Figure 2 shows the Manhattan distance between the Baseline Bot and the target (rat) over time. The distance fluctuates, indicating inconsistent progress towards the target. Although

there are periods where the bot gets closer, the overall pattern shows limited success in consistently reducing the distance.

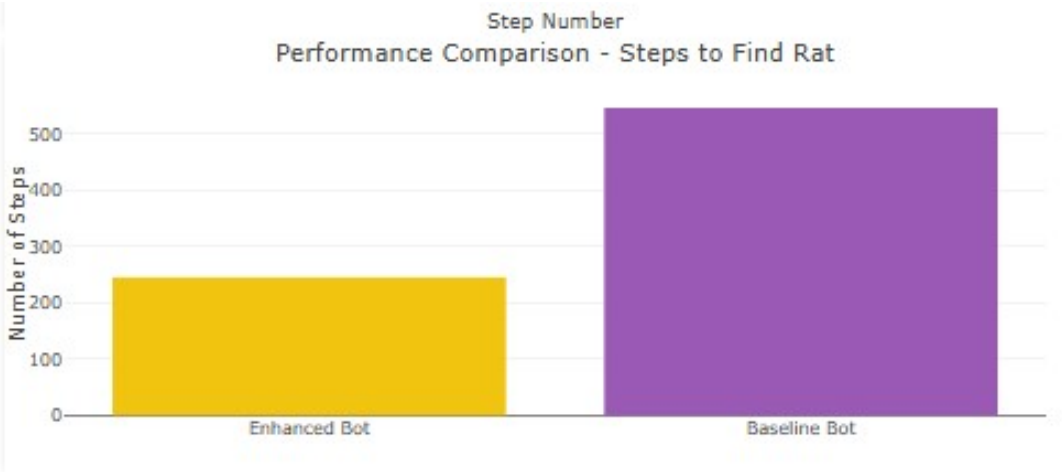


Figure 3: Performance comparison bar graph.

This bar graph is generated for $\alpha = 0.05$. There is a significant difference between the number of steps taken by the Enhanced Bot compared to the Baseline Bot to locate the Rat. This highlights that the Enhanced Bot's improved movement and decision-making perform much better than the Baseline Bot.

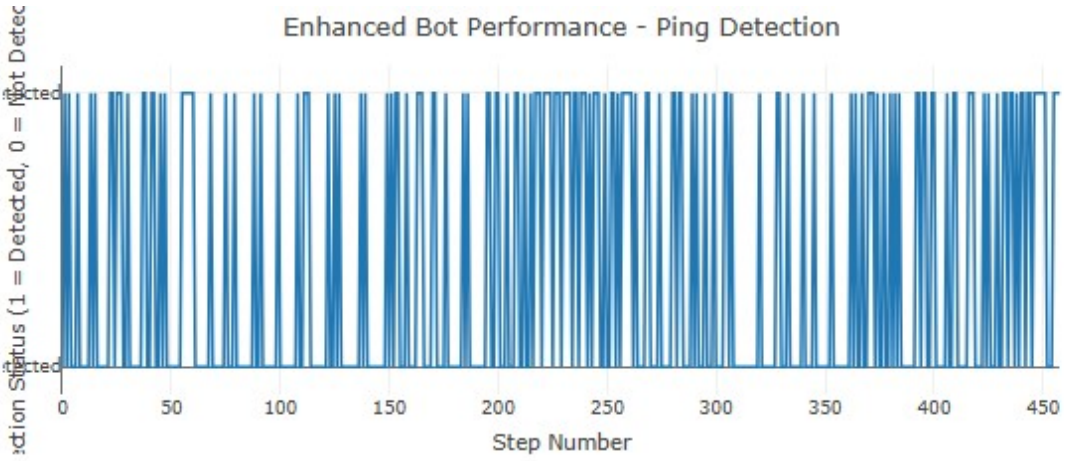


Figure 4: Ping detection status.

A value of 1 indicates that a ping was detected, while 0 indicates that no ping was detected. At lower α values, the ping detection frequency is very high, increasing the chances for the Bots to move toward the Rat's location.

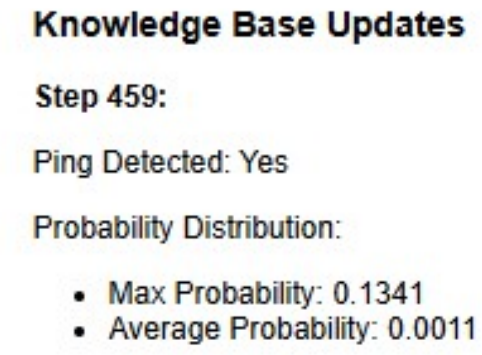


Figure 5: Evaluation details.

Performance evaluation page showing the number of steps taken at the latest time step, along with updates on ping detection and probability distribution.

Figure 6: Grid for alpha factor 0.2.

Figure 6 compares the "Enhanced Bot" (left) and "Baseline Bot" (right) navigation, showing wider, more efficient exploration (yellow) by the Enhanced Bot at an alpha factor of 0.2, while the Baseline Bot (purple) covers a narrower path.

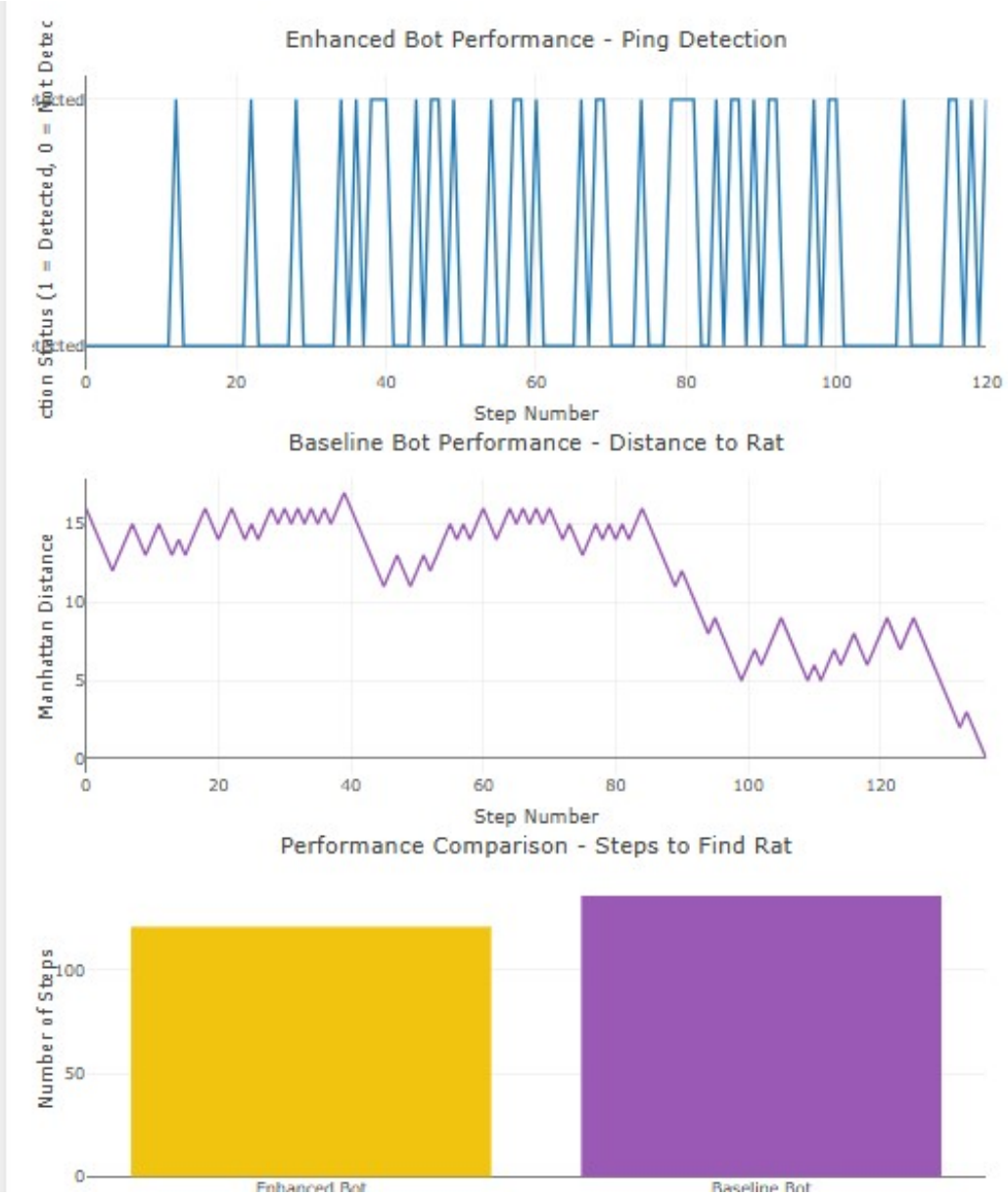


Figure 7: Complete Performance comparison focuses on ping detection interval.

Figure 7 compares Enhanced and Baseline Bot performance. The Enhanced Bot shows frequent target pings, while the Baseline Bot gradually reduces its distance. The Enhanced Bot requires slightly fewer steps at alpha factor 1.0.

Figure 8: Performance comparison focuses on ping detection interval.

Figure 8 shows the performance of enhanced and baseline bots across 100 simulations, with alpha values from 0.02 to 0.2 (0.02 intervals). When the α is higher, Bot receive no ping for series number of actions that is captured in the first image. The enhanced bot detects the rat consistently and reaches it faster, while the baseline bot shows fluctuating distance and slower detection.

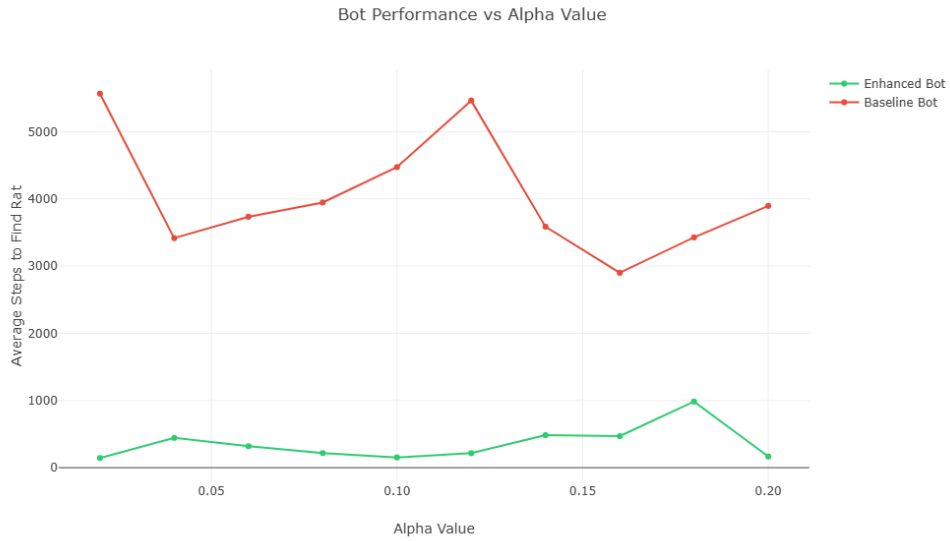


Figure 9: Performance comparison across 100 simulation .

Figure 9 shows the average steps needed for enhanced and baseline bots to find a moving rat across 100 simulations, with alpha values from 0.02 to 0.2. The enhanced bot consistently requires fewer steps than the baseline bot, demonstrating improved efficiency.

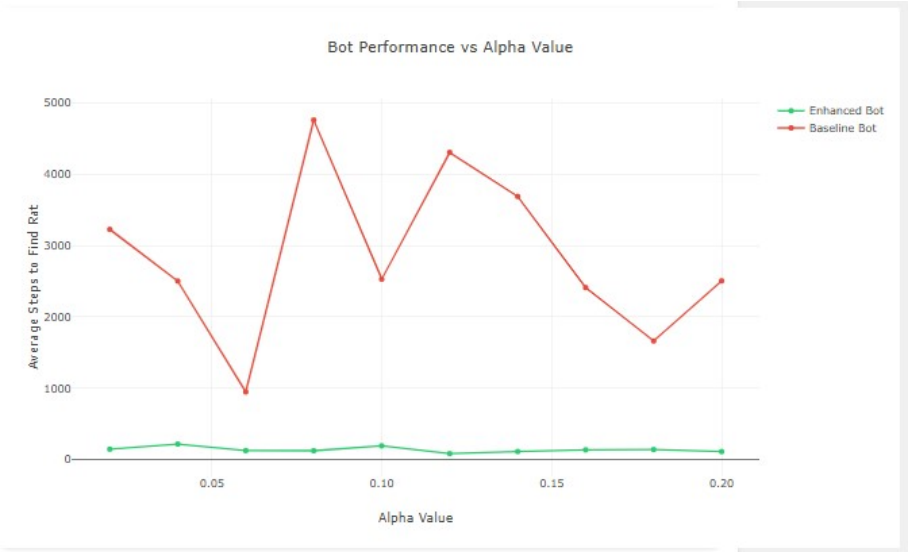


Figure 10 shows the average steps taken by enhanced and baseline bots to find a stationary rat across 100 simulations, with alpha values from 0.02 to 0.2 (0.01 intervals). The enhanced bot remains highly efficient, requiring far fewer steps than the baseline bot, which shows significant fluctuations in performance.

Comparison of Enhanced Bot vs Baseline Bot on Rat Movements

Figure 11: Comparison grid based on Rat Movements .

Figure 11 shows the performance of an Enhanced Bot versus a Baseline Bot in finding a moving target (rat). The light red colored cells represents the previous rat path. While baseline bot runs over the same cells quite often, Enhanced bot eliminates running over same path multiple times and catches the rat more efficient way.

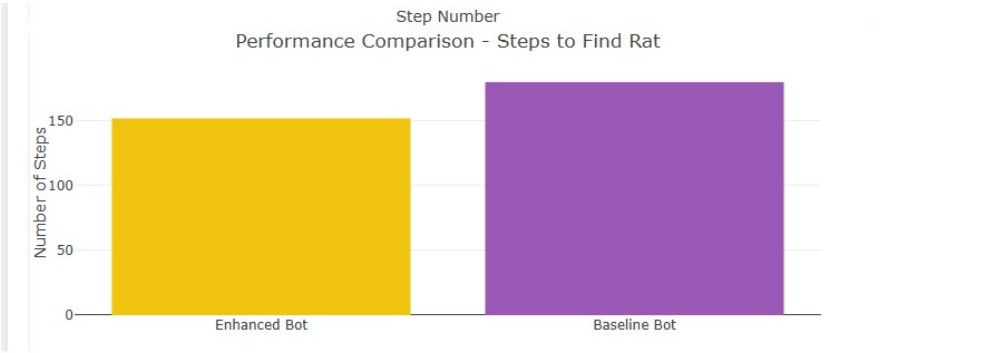


Figure 12: Bar graph for above grid simulation.

Figure 12 is the final performance comparison result based on the Figure 11 Grid, Enhanced bot is efficient in catching the rat faster than the Baseline bot even when the target is moving and position of the target is unknown.

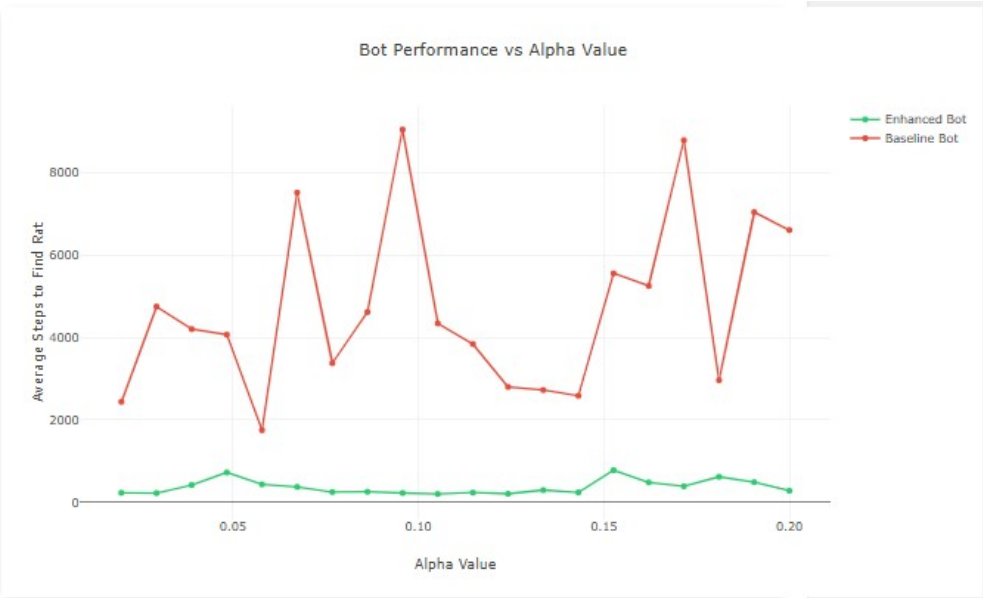


Figure 13: Performance comparision across 100 simulation .

Figure 13 shows the performance of an Enhanced Bot versus a Baseline Bot in finding a moving target (rat) over different alpha values (0.02 to 0.2) in 400 simulations. The Enhanced Bot consistently requires fewer steps, displaying stable performance, while the Baseline Bot has high variability with generally more steps needed.

Figure 14: Performance comparision across 100 simulation .

Figure 14 shows the performance comparison between an Enhanced Bot and a Baseline Bot in locating a moving target at various alpha values. The Enhanced Bot consistently maintains a low average step count, while the Baseline Bot's performance fluctuates significantly with higher step counts across the alpha range.

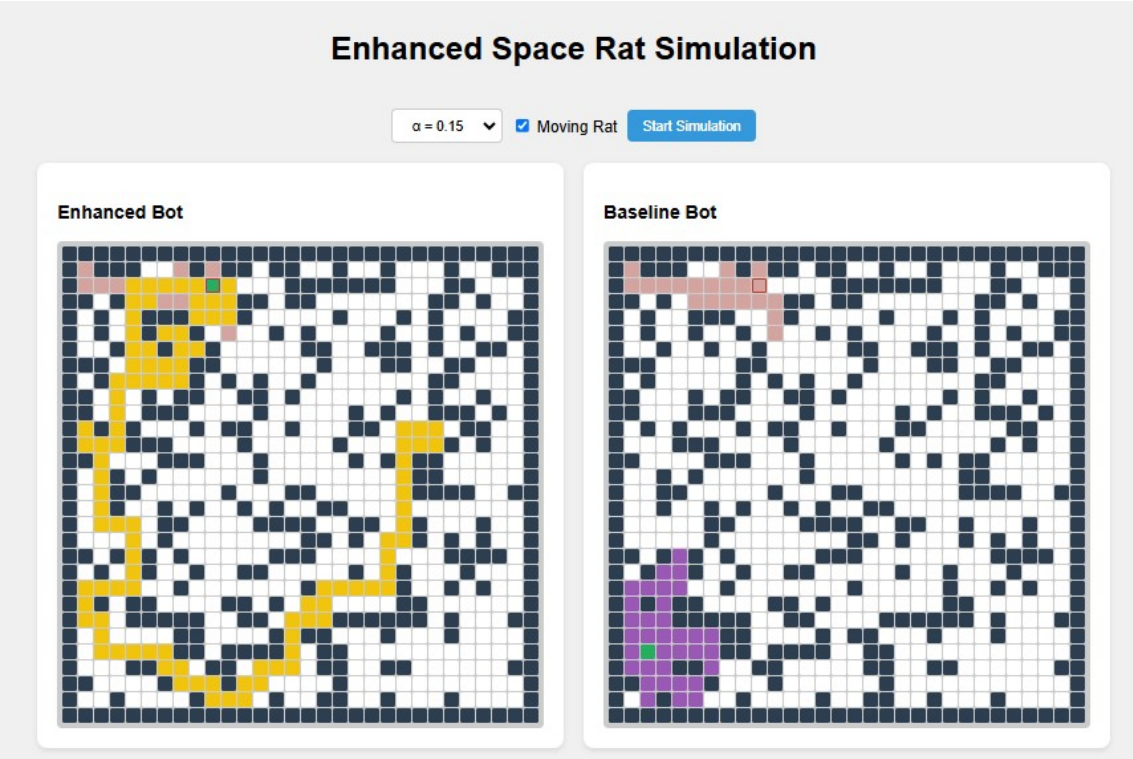


Figure 15: Performance comparision in moving rat scenario at α 0.15 .

Figure 15 shows the performance of an Enhanced Bot versus a Baseline Bot in finding a moving target (rat) at $\alpha = 0.15$. The Enhanced Bot consistently requires fewer steps, displaying stable performance, while the Baseline Bot has high variability with generally more steps needed.

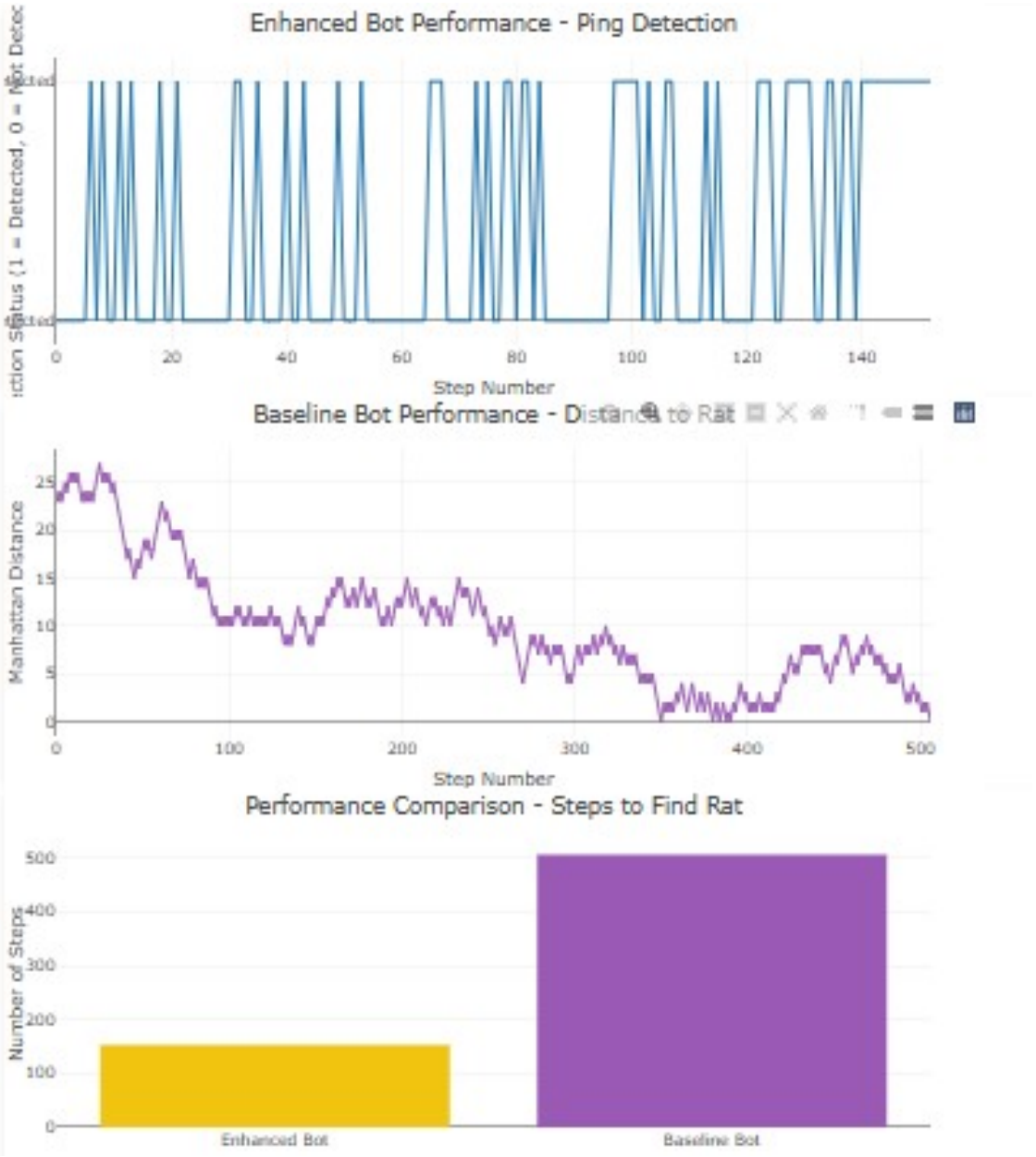


Figure 16: Performance comparison outcome of previous grid.

Figure 16 shows the performance of an Enhanced Bot versus a Baseline Bot in finding a moving target (rat) at $\alpha = 0.15$. this result in larger number of steps and baseline bot showing a vast difference in steps to identify the rat position as it was consistently moving. there are even no ping detected across several cells as well.

Improvements of Enhanced Bot over Baseline Bot

Enhanced Bot imporved strategy

Probabilistic Knowledge Base
- Maintains a grid of probabilities for rat location
- Updates probabilities based on ping detector readings
- Accounts for sensor uncertainty using exponential decay model

Intelligent Movement
- Factors in visit history to avoid repetitive paths
- Uses weighted utility function for move selection
- Balances exploration vs. exploitation

Adaptive Behavior
- Adjusts strategy based on ping detection history
- Modifies exploration rate based on search progress
- Implements memory of recent positions to avoid loops

Changes made to Enhanced Bot based on Rat Movement

Probability Propagation: Predicting Rat Movement:

To account for the rat's movement, we implemented a probability propagation system. After each time step, the bot spreads the probability of the rat's location to neighboring cells. This spreading uses a decay factor α, which controls how quickly probabilities diffuse:

$$P_{\text{new}}(x, y) = (1 - \alpha) \cdot P_{\text{current}}(x, y) + \frac{\alpha}{N} \sum_{(x', y')} P_{\text{current}}(x', y')$$

This formula can be broken down as follows:

- $P_{\text{new}}(x, y)$: The updated probability of the rat being in cell (x, y) at the next time step.
- $P_{\text{current}}(x, y)$: The current probability of the rat being in cell (x, y) before the update.
- α: A decay factor (between 0 and 1) that controls the rate at which probabilities diffuse to neighboring cells.
- N: The number of neighboring cells around the cell (x, y).
- $\sum_{(x', y')}$: The summation over all neighboring cells (x', y') of the current cell (x, y).
- $P_{\text{current}}(x', y')$: The current probability of the rat being in each neighboring cell (x', y').

Adaptive Search Strategy

With a moving target, our bot needed to become more proactive:

- **Predictive Movement**: Instead of just reacting to pings, the bot now tries to anticipate where the rat might move next. It's like a game of chess, always thinking one move ahead.

- **Dynamic Speed Adjustment:** The bot learns to move faster in open areas and more cautiously in cluttered sections of the ship. This mimics how a real searcher might behave – running in open corridors but slowing down to carefully check hiding spots.

- **Ping Interpretation** A ping no longer just means "the rat is nearby." Now, it also provides information about the rat's movement. If the bot gets several pings in a row, it can infer the direction the rat is moving.

Sensor Reinterpretation:
With a moving rat, each ping carries more complex information:

- **Velocity Estimation** By comparing consecutive pings, the bot can estimate not just where the rat is, but how fast it's moving and in what direction.

- **Probability Trails** Instead of just updating probabilities in a circular pattern around a ping, the bot now creates elongated probability "trails" in the direction it thinks the rat is moving.

- **False Negative Handling** The bot now accounts for the possibility that it might miss a ping even when near the rat, due to the rat's movement. This is handled by maintaining a "minimum probability threshold" across all cells.

Conclusion

In conclusion, we have successfully developed and implemented an enhanced bot for space rat tracking, demonstrating significant improvements over the baseline bot. The enhanced bot effectively utilizes a sophisticated utility-based movement decision system, a dynamic probabilistic knowledge base for tracking rat locations, and adaptive behavior that evolves based on search progress and environmental feedback. The comparative analysis indicates that the enhanced bot consistently outperforms the baseline in various scenarios, including both stationary and moving rat situations.

The comparison of the enhanced and baseline bots, led to conclude about substantial improvements in both efficiency and effectiveness The enhanced bot consistently outperformed the baseline in various scenarios, including situations with both stationary and moving rats. The improved exploration strategies reduced unnecessary movements and achieved more accurate rat location predictions using advanced probabilistic modeling, and adaptive behavior that balances exploration and exploitation. Visualizing complex data, like using heatmaps for probability distributions, was incredibly helpful in understanding the bot's decision-making process.

Contributions

Tejaswini Abburi (ta633)

- Designed and implemented the Spaceship Environment grid and agent positioning logic, ensuring blocked and open cell distribution.
- Developed the bot's self-localization algorithm, utilizing blocked neighbor sensing and movement feedback to refine position probabilities.
- Created the enhanced bot's movement decision framework, including utility-based movement balancing exploration and exploitation.
- Contributed to the development of the probabilistic knowledge base, supporting real-time rat location probability updates.
- conducted evaluation tests to compare the Enhanced Bot's performance against the Baseline Bot under varying detector sensitivity levels.

Prajwal Srinivas (ps1458)

- Led the design and implementation of the Space Rat Detector, including probabilistic ping/no ping detection based on distance and decay constants.
- Implemented the rat-tracking and movement algorithms using Bayesian updating for dynamic probability distribution.
- Enhanced the bot's adaptive behavior with memory of recent positions to avoid repetitive paths, improving search efficiency.
- Generated and analyzed heatmap visualizations for the rat's probable location, aiding in visualization and decision-making.
- Ran performance simulations across different α values, documenting improvements in the Enhanced Bot's efficiency over the Baseline Bot.