Space Rat & Brave Bot: Project 2 in Introduction to AI (CS-520)

Harsha Rajendra (hr458), Santhosh Janakiraman (sj1230)
Computer Science Department, Rutgers University
hr458@scarletmail.rutgers.edu, sj1230@scarletmail.rutgers.edu

1.0 Abstract

This project involves designing a customized autonomous bot to navigate a structured, grid-based environment representing the layout of a "ship" and locate a target, the "space rat." The environment is modeled as a 30x30 grid, where approximately 60% of the cells are open, and the outer edge cells are permanently blocked. The bot faces two main challenges: it has no knowledge of its initial location due to memory loss from cosmic interference, and it must track down the space rat within the grid.

The bot operates in two key phases: Localization and Target Pursuit. In the Localization phase, the bot iteratively determines its exact position by sensing the number of adjacent blocked cells and attempting directional movements. By combining sensory data and movement outcomes, the bot systematically narrows down potential locations until it identifies its precise position. In the Target Pursuit phase, the bot uses a probabilistic detector to track the space rat. This detector "pings" based on proximity, with higher probability as the bot moves closer. If the bot and space rat occupy the same cell, the detector confirms their alignment.

Our bot's performance is compared against a baseline bot through various tests, focusing on metrics such as the number of moves taken, sensing actions, and detector activations. To optimize tracking e iciency, we analyze the sensitivity parameter- α, which affects the detector's probability model. As α increases, the likelihood of distant pings decreases, challenging tracking but preserving precision when close; conversely, low α values lead to frequent pings, which may saturate the sensor. The analysis is plotted as a function of α, providing insights into optimal settings.

In advanced tests, the space rat moves randomly after each bot action, introducing dynamic motion. This requires an update to the bot's knowledge base using adaptive probability calculations, enabling it to anticipate and track the moving target. The comparative performance of the customized bot versus the baseline in this dynamic environment highlights the effectiveness of adaptive tracking strategies.

2.0 The Task

In this project, the goal is to design and implement a bot that navigates a grid-based layout representing a ship, where some cells are blocked (walls), and a "space rat" has infiltrated. Due to cosmic radiation, the bot has lost its memory of its current position, so it must first determine where it is on the ship. Once localized, it must then track down and capture the space rat.
The bot has three main actions:

- **Movement:** The bot can move in the four cardinal directions (up, down, left, right).
- **Blocked Cell Sensing:** The bot can sense the number of neighboring cells that are blocked to help it identify its location on the ship.
- **Rat Detection:** The bot can use a rat detector, which "pings" with a probability that depends on the distance to the rat and a sensitivity parameter, α.

The project is divided into two phases:

- **Phase 1 (Self-Localization):** The bot uses blocked cell sensing to identify its position by eliminating possibilities over multiple movements.

- **Phase 2 (Rat Tracking and Capture):** Once localized, the bot uses rat detection pings to track down and capture the rat.

Additional Requirement: Dynamic Rat Movement
In the initial setup, the space rat was assumed to be stationary. However, as an additional requirement, the space rat will move dynamically in this project. After each action by the bot, the rat moves randomly within open cells. This requires the bot to continuously update its probability estimates to adapt to the rat's changing position.
The bot's performance is evaluated based on the number of each type of action taken (movements, blocked cell sensing actions, and rat detector actions) across different values of α, which determines the rat detector's sensitivity. This evaluation helps identify the most effective α range for tracking a moving target efficiently.

3.0 The Ship

The ship's layout is designed on a **30×30 square grid**, with all cells initially set as 'blocked.' A random interior cell is marked 'open,' initiating an iterative process that opens additional blocked cells with exactly one open neighbor, continuing until no further cells meet this criterion. To ensure approximately **60%** of the grid is accessible, dead-end cells—defined as open cells with only one open neighbor - are strategically opened. Roughly **half** of these dead ends are then randomly chosen to open one of their blocked neighbors, enhancing connectivity and optimizing the layout for efficient navigation.

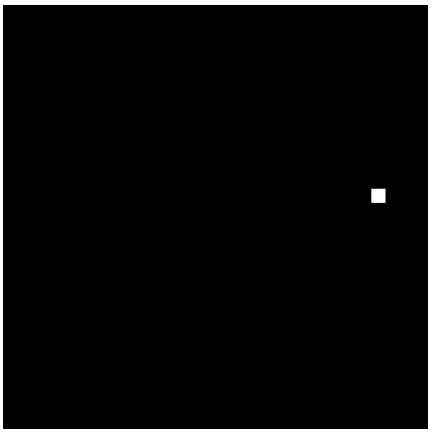

Figure 1: The Initial Grid with Starting Cell, Here (13,26)

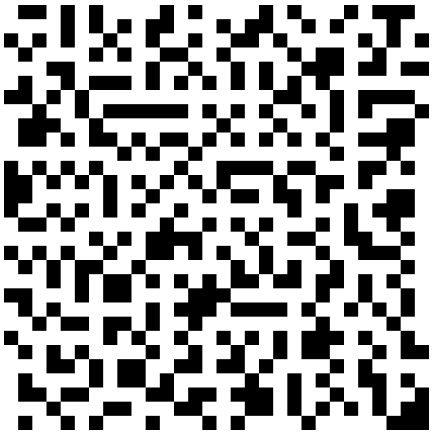

Figure 2: The Grid after Opening Cells and Finding Dead Ends

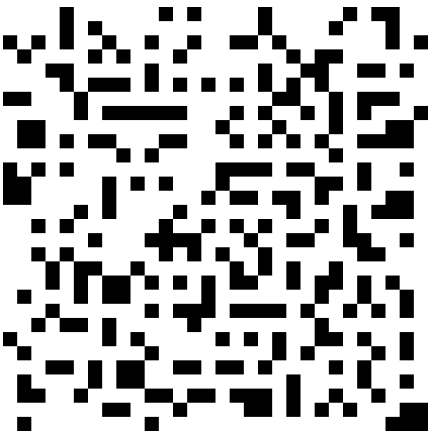

Figure 3: The Grid after Expanding Dead Ends

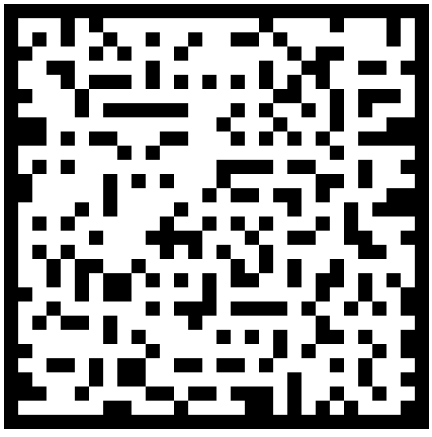

Figure 4: The Final Grid after blocking all Edge Cells

Grid Layout Formation:

1. **Grid Initialization and Initial Cell Selection:** A 30×30 square grid is established with all cells initially blocked. An interior cell, e.g., (13,26), is randomly selected and marked as open, serving as the starting point for subsequent modifications.

2. **Accessibility Iterative Process and Dead End Identification:** Iteratively, all currently blocked cells that have exactly one open neighbor are identified, and one of these cells is opened randomly. This process continues until no further cells meet this criterion.

3. **Final Layout and Connectivity Enhancement:** The algorithm randomly selects half of the identified dead ends to mark one of their neighboring blocked cells as open, enhancing connectivity and layout. After completing all iterations and adjustments, the grid has more open areas, allowing for easier movement and navigation within the layout.

4. **Boundary Setting for Outer Edge Cells:** Once the final layout is generated, all outer edge cells are permanently set as blocked to form a secure boundary around the grid, ensuring that the layout has well-defined borders.

4.0 Placing the Bot and the Rat

The bot is placed randomly in an open cell within the ship's grid, allowing it to move one cell at a time in any of the four cardinal directions: up, down, left, or right. The space rat is also positioned randomly in an open cell, setting the stage for the bot's mission to locate and capture it.

To identify its location within the ship, the bot begins by using *blocked cell sensing*. By counting the number of blocked neighboring cells, the bot gradually narrows down its possible positions, moving in a way that maximizes information gained from each sensing action. Once it has accurately localized itself, the bot employs a *rat detector* with probabilistic pings that indicate the likelihood of the rat's proximity. The probability of detecting the rat's presence is influenced by a sensitivity parameter α and the distance between the bot and the rat, following an exponential decay model.

This approach allows the bot to make informed movements, prioritizing areas with higher probabilities of rat presence, effectively guiding it toward the rat for capture.

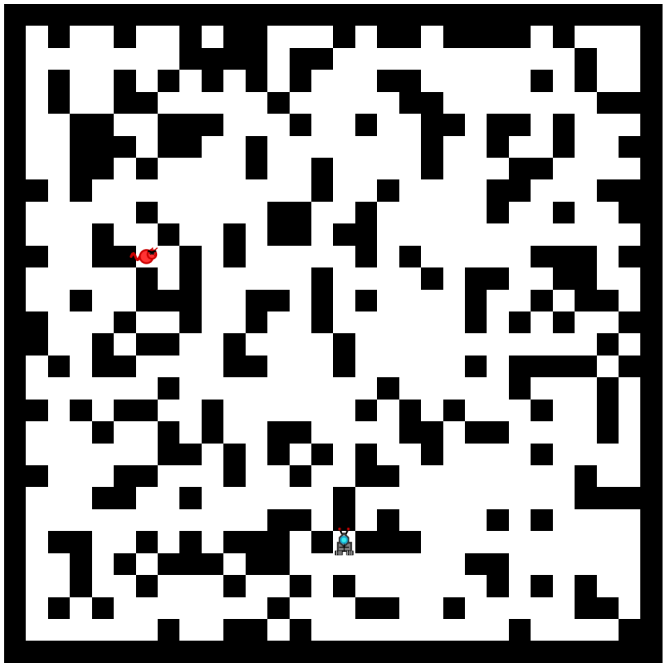

Figure 5: The Grid Layout After Deployment: Bot (Blue) and Space Rat (Red)

5.0 Probability and Detection Systems

5.1 The Space Rat Detector

We utilize a **detection system** following an exponential decay model to locate the space rat. The core of our detection system is based on the probability formula:

Detection Probability Formula:

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

where $d(i,j)$ is Manhattan distance and $\alpha \in [0, 1.0]$

where $d(i,j)$ represents the **Manhattan distance** between the bot's position (cell i) and the rat's position (cell j), and α is the **sensitivity parameter** between 0 and 1.0. This formula captures how detection probability decreases with distance.

→ When bot and rat share the **same cell** ($d(i,j) = 0$), $P(\text{ping}) = 1$

→ As **distance increases**, probability decreases exponentially

- **Small** α (0.1): Detection possible from longer distances
- **Large** α (0.5): Detection mostly limited to nearby cells

When receiving sensor information (ping or no ping), we update rat location beliefs using **Bayes' Theorem**:

$$P(rat_j|ping) = \frac{P(ping|rat_j) \times P(rat_j)}{P(ping)}$$

Where:

- $P(rat_j)$ is our **prior belief** about rat being in cell j
- $P(ping|rat_j)$ is given by our **detection formula**
- $P(ping)$ is the **normalizing factor** calculated as:

$$P(ping) = \sum_k P(ping|rat_k) \times P(rat_k)$$

The sensor provides **binary feedback** (ping or no ping), interpreted probabilistically:

→ **Receiving ping**: Higher probabilities for nearby cells, lower for distant ones

→ **No ping**: Higher probabilities for distant cells, lower for nearby ones

→ **Multiple readings**: Each ping progressively refines location probabilities

5.2 Bot Finding Itself

In a 30x30 grid spaceship environment, our bot employs a sophisticated pattern matching system through wall detection and probability updates. The bot analyzes its surroundings by examining eight neighboring cells, counting blocked cells to create a unique positional signature. This counting mechanism is represented mathematically as:

$$B_{count} = \sum_{n \in \text{neighbors}} b_n, \text{ where } b_n = 1 \text{ if blocked} \tag{1}$$

Each position's probability is continuously refined based on this detection system. Initially distributing equal probability across approximately 540 open cells, the bot updates these probabilities whenever it senses its surroundings:

$$P(bot_i|\text{blocked}) = \begin{cases} \frac{1}{N_{\text{matching}}} & \text{if counted blocks = sensed blocks} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

For example, when sensing reveals 3 blocked neighbors, the initial 540 possible locations typically reduce to around 40 matching cells. The probability for these cells updates to $\frac{1}{40} = 0.025$, while all non-matching cells are eliminated. Movement attempts further refine these probabilities through:

$$P(bot_i|\text{movement}) = \begin{cases} 0 & \text{if movement conflicts with walls} \\ \frac{1}{N_{\text{remaining}}} & \text{for feasible positions} \end{cases} \tag{3}$$

The complete localization process combines these sensing and movement phases sequentially. Each action builds upon previous knowledge:

$$P(bot_i|\text{sequence}) = \prod_{t=1}^{T} P(bot_i|\text{action}_t)$$

. Through this systematic approach, each observation eliminates 60-70% of possibilities. Within 10-15 strategic actions, the bot confidently determines its **starting position** (P = 1.0 for a single cell) using wall pattern validation and movement testing.

6.0 Baseline Bot Design

The baseline bot operates in two critical phases: **localization (Phase 1)** and **tracking the static space rat (Phase 2)**. Both phases involve sensing, probabilistic updates, and movement strategies. To achieve these objectives, the bot employs *Depth-First Search (DFS)* during Phase 1 and *Weighted Breadth-First Search (Weighted BFS)* during Phase 2.

→Phase 1: Localization

In this phase, the bot begins in an unknown 30×30 grid and determines its position through sensing and exploration. It calculates the number of blocked neighbors, as explained in Section 5.0:

$$B_{count} = \sum_{n \in \text{neighbors}} b_n, \quad \text{where } b_n = 1 \text{ if blocked.}$$

Using this count, the bot updates its **knowledge base** by discarding positions that do not match the sensed blocked count. This progressively refines the valid positions. For movement, the bot uses *DFS* (a search algorithm exploring paths deeply before backtracking) to systematically attempt moves in cardinal directions. Each move updates the bot's knowledge:

- **Successful moves:** Reveal new blocked neighbor counts, narrowing possible locations.
- **Blocked moves:** Eliminate inconsistent paths.

This process continues until the bot identifies a **single valid position**, completing localization.

→Phase 2: Space Rat Tracking

Once localization is achieved, the bot transitions to the second phase of tracking the static space rat.In this phase, the bot leverages the **detection system** (Section 5.0) and **Bayesian updates** (updates based on probabilities derived from new evidence) to estimate the rat's location and navigate toward it. The bot uses *Weighted BFS* to prioritize cells based on their probabilities and distances.To optimize tracking, neighbors are dynamically prioritized based on their **position weights** (a measure of how likely a cell contains the rat). Probabilities are calculated using Bayesian inference. The bot sorts these neighbors by their weights, ensuring cells with higher probabilities are explored first. The **movement strategy** evaluates potential moves based on these two factors, balancing the likelihood of the rat's location with efficient navigation.

Throughout this phase, the bot updates its probability distribution using sensor feedback:

- **Ping received:** Probabilities for nearby cells increase (indicating the rat is likely close).
- **No ping:** Higher weights are assigned to more distant cells (suggesting the rat is farther away).

These updates refine the bot's knowledge at every step, ensuring accurate tracking of the static space rat. The phase concludes successfully when the bot navigates to the rat's location and captures it.

Summary

By employing ***DFS*** (for deep path exploration) during Phase 1 and ***Weighted BFS*** (for efficient prioritization of movements) during Phase 2, the baseline bot systematically achieves localization and tracking objectives. The use of dynamic neighbor prioritization and continuous Bayesian updates ensures both **accuracy** and **efficiency** throughout its operations.

7.0 Improved Bot Design

The improved bot operates in two critical phases: **localization (Phase 1)** and **tracking the static space rat (Phase 2)**. Both phases involve sensing, probabilistic updates, and movement strategies, with enhancements over the baseline approach to improve accuracy and efficiency.

→Phase 1: Localization

The improved bot begins in an unknown 30×30 grid, aiming to determine its exact position through sensing and exploration. The localization phase builds on the baseline bot's strategy by incorporating clustering techniques, heuristic-guided movement, and an efficient pathfinding algorithm.

Enhanced Movement and Knowledge Base Refinement

- **Initial Sensing and Knowledge Update:** Similar to the baseline approach, the bot first senses the number of blocked neighbors and uses this information to discard positions inconsistent with the observed blocked count, progressively refining its set of valid positions.
- **Heuristic-Guided Movement Strategy:** The bot prioritizes its movements using a heuristic function defined as:

  $$\text{Heuristic}(p) = \text{Manhattan Distance}(p, \text{target}) - \text{Open Neighbors}(p)$$

  Here, the heuristic considers both the distance to a target position and the number of open neighboring cells, favoring paths with more open neighbors for increased flexibility and reduced dead ends.
- **Clustering and Target Selection:** The bot groups possible valid positions into clusters based on their proximity, focusing on navigating towards the largest cluster to optimize efficiency. This clustering helps in reducing redundant exploration.

- **A* Search for Pathfinding:** The improved bot uses an A* search algorithm to find the most efficient path towards a target position. The cost function combines movement cost and heuristic estimates:

$$f(p) = g(p) + h(p)$$

where $g(p)$ is the cost of reaching position p from the start, and $h(p)$ is the heuristic estimate to the target. The bot maintains a list of visited positions to avoid redundancy and ensure efficient exploration.

- **Iterative Updates:** During each move, the bot updates its knowledge base by sensing blocked neighbors, eliminating inconsistent positions, and refining possible valid locations. This process continues until only one valid position remains, signifying successful localization.

→**Phase 2: Space Rat Tracking**

Once localization is achieved, the bot transitions to tracking the static space rat. This phase emphasizes probabilistic updates and informed movement strategies using advanced filtering techniques.

Probabilistic Updates and Movement Strategy

- **Markov Filtering:** The bot uses a filtering approach to update the probability distribution of the rat's potential positions. The probability for each position (x, y) is updated based on whether a detection ping is received:

$$\text{Probability Update:} \quad P'(x,y) = P(x,y) \cdot \begin{cases} \text{Probability of Ping} & \text{if ping received} \\ 1 - \text{Probability of Ping} & \text{otherwise} \end{cases}$$

This ensures that probabilities are adjusted dynamically based on sensory feedback, with normalization to maintain a valid distribution.

- **A* Search for Movement:** The bot employs A* search to move towards the most probable rat position, prioritizing paths with high probabilities and efficient navigation. The target position is selected based on the highest probability weight, and the cost function balances movement efficiency with the likelihood of encountering the rat.

- **Decision Making:** At each step, the bot decides between sensing and movement using a utility-based approach. This decision is guided by Upper Confidence Bound (UCB) calculations to balance exploration and exploitation:

$$\text{UCB} = \frac{\text{Total Reward}}{\text{Number of Actions}} + c \cdot \sqrt{\frac{\log(\text{Total Actions})}{\text{Number of Actions}}}$$

where c is the exploration parameter. This ensures that the bot explores new possibilities while exploiting known information effectively. The subsequent sections provide a more thorough explanation of these factors.

- **Probabilistic Simulation for Reward Calculation:** The bot simulates potential updates based on receiving or not receiving a ping, calculating the weighted average of probabilities:

$$P_{\text{sim}}(x,y) = \text{Probability of Ping} \times P_{\text{ping}}(x,y) + (1 - \text{Probability of Ping}) \times P_{\text{no ping}}(x,y)$$

These simulated probabilities are used to assess the effectiveness of potential actions, guiding the bot's next steps.

Summary:

The improved bot enhances rat capture efficiency through three integrated strategies. First, it employs A* pathfinding with intelligent cluster-based exploration, avoiding repeated checks of previously visited locations. Second, it maintains accurate probability tracking using exponential decay detection ($P(ping) = e^{-\alpha(d-1)}$), prioritizing cells based on both likelihood of containing the rat and proximity to current position. Finally, it implements efficient decision-making by balancing exploration vs. exploitation and calculating utility scores (movement and sensing utilities)

8.0 Data & Analysis

Q1) ***Space Rat Knowledge Base Updation: for Receiving a Ping and No Ping:***

→Initial Setup and Prior Probabilities:

Assume the bot is at position (b_x, b_y) and the list of possible rat positions is denoted as possible_rat_positions. The prior probability of the rat being in each possible position is calculated as:

$$\text{prior\_prob} = \frac{1}{|\text{possible\_rat\_positions: }|}$$

where |possible_rat_positions| is the total number of possible positions.

→Sensing and Determining Ping Status

The bot calculates the Manhattan distance to the actual rat position (r_x, r_y):

$$d = |b_x - r_x| + |b_y - r_y|$$

The probability of receiving a ping given this distance is:

$$\text{likelihood\_ping}(r_x, r_y) = e^{-\alpha \times (d-1)}$$

where α is a decay parameter controlling how quickly the probability decays with distance.
The bot determines if a ping is received using a random draw compared to probability_of_ping.

→ Case 1: Bot Receives a Ping

When a ping is received, the following calculations are made:

Updating Likelihoods:

For each possible rat position (r_x, r_y), the likelihood of receiving a ping given that the rat is at (r_x, r_y) is:

$$\text{likelihood\_ping}(r_x, r_y) = e^{-\alpha \times (d-1)}$$

Here, d is the Manhattan distance between (b_x, b_y) (bot position) and (r_x, r_y).

Calculating Posterior Probabilities:

Using Bayes' theorem, the posterior probability that the rat is at position (r_x, r_y) given that a ping is received is calculated as:

$$\text{posterior\_probability\_ping}(r_x, r_y) = \frac{\text{likelihood\_ping}(r_x, r_y) \times \text{prior\_prob}}{\text{normalization\_constant}}$$

The normalization constant ensures that the sum of all posterior probabilities equals 1:

$$\text{normalization\_constant} = \sum_{(r_x, r_y) \in \text{possible\_rat\_positions}} \text{likelihood\_ping}(r_x, r_y) \times \text{posterior\_prob}$$

Resulting Updated Probabilities:

After normalizing, the updated probabilities reflect an increased likelihood that the rat is closer to the bot due to the ping.

→Case 2: Bot Does Not Receive a Ping

When no ping is received, the following calculations are performed:

Updating Likelihoods:

For each possible rat position (r_x, r_y), the likelihood of not receiving a ping given that the rat is at (r_x, r_y) is:

$$\text{likelihood\_no\_ping}(r_x, r_y) = 1 - e^{-\alpha \times (d-1)}$$

Calculating Posterior Probabilities:

Using Bayes' theorem, the posterior probability that the rat is at position (r_x, r_y) given that no ping is received is:

$$\text{posterior\_probability\_no\_ping}(r_x, r_y) = \frac{\text{likelihood\_no\_ping}(r_x, r_y) \times \text{prior\_prob}}{\text{normalization\_constant}}$$

The normalization constant is calculated as:

$$\text{normalization\_constant} = \sum_{(r_x, r_y) \in \text{ossible\_rat\_positions}} \text{likelihood\_no\_ping}(r_x, r_y) \times \text{posterior\_prob}$$

Resulting Updated Probabilities:

After normalization, the updated probabilities reflect a decreased likelihood that the rat is near the bot due to the absence of a ping.

Summary of Updates:

The knowledge base is updated using the posterior probabilities calculated for either the ping or no ping scenario. The bot then uses these updated probabilities to make subsequent decisions, such as moving toward the most probable rat position.

Note: The improved bot leverages the same update process but uses Markov-filtering and UCB-driven decisions for more efficient tracking and movement.

T.P.O

Q2) ***Improved Bot Algorithm and Design:***

Algorithm 1 Phase 1: Localization

```
Initialize simulation state and grid layout
Set initial bot position and sense blocked neighbors
Calculate possible positions based on initial sensing
Initialize set to track visited positions
while localization is incomplete do
    Call move_bot() to determine next move and update bot's current position
    Track visited positions and check if the move was successful
    if move was successful then
        Update possible positions using sensed blocked neighbors
        Eliminate inconsistent positions
        Call sense_blocked_neighbors() and possible_positions_after_sensing() to refine knowledge
    else
        Adjust possible positions based on failed move
        Eliminate inconsistent paths
        Call sense_blocked_neighbors() and possible_positions_after_sensing() to refine knowledge
    end if
end while
if only one possible position remains then
    Transition to Phase 2 with a randomly selected rat position
    Call phase2() for further operations
else
    Mark localization as failed
end if=0
```

Additional Explanation for `move_bot()`:

- **Initial Setup:** The function uses an A* algorithm to move the bot towards the target cluster of possible positions, aiming to maximize exploration efficiency.
- **Heuristic Calculation:** The heuristic function considers the Manhattan distance to the target position and the number of open neighbors:

  $$\text{Heuristic} = \text{Manhattan Distance} - \text{Open Neighbors}$$

  This encourages movement towards paths with more open spaces.
- **Pathfinding:** The function selects the node with the lowest f-score, updating paths and costs as it explores neighbors. When the target is reached, it moves along the determined path.
- **Fallback:** If no valid moves are found, a fallback mechanism is triggered to avoid the bot getting stuck.

Algorithm 2 Phase 2: Space Rat Tracking

```
Initialize possible rat positions and probability distribution
Set initial bot position and rat position
Initialize variables for UCB-based decision-making
while bot has not caught the rat do
    Calculate distance to the rat and determine if a ping is received
    Calculate UCB values for movement and sensing actions
    if movement has a higher UCB value then
        Determine most probable rat position and call move_bot_towards_rat()
        Update bot state and position
        Calculate rewards based on probability improvement of rat's position estimate
    else
        Call markov_filter_update() to update rat probabilities based on sensing
        Calculate entropy change and update reward for sensing action
    end if
    if bot reaches rat's position then
        End the simulation successfully
    end if
end while=0
```

Additional Explanation for `move_bot_towards_rat()`:

- **Initial Setup:** The function begins by identifying potential **rat positions** and selects the target position with the highest associated **probability**.
- **A* Search with Heuristic:** The bot employs an ***A* search algorithm*** to determine the optimal path toward the selected target position. The heuristic function combines the **Manhattan distance** to the target position and the **position weight**, guiding the bot towards high-probability areas while considering proximity:

$$\text{Heuristic} = \text{Manhattan Distance} - \text{Position Weight}.$$

- **Path Exploration:** The function iteratively selects the position with the lowest **f-score** from the unexplored cells. If this position matches one of the potential rat positions, the bot either moves to the next step or stays in place if already at the target.
- **Neighbor Exploration:** The function evaluates neighboring cells, updating paths and costs dynamically when a better route is found. **Valid moves** are added to the open set for further exploration.
- **Fallback:** In the absence of valid moves, a **fallback mechanism** triggers a random move to ensure the bot avoids stalling, maintaining systematic exploration and tracking.

Improved Bot Overview:

1. **Improved Bot Design Overview**: The improved bot builds upon the baseline by integrating heuristic optimizations, utility-based decision-making, and efficient movement algorithms. It utilizes heuristic-guided A* search with clustering-based prioritization in Phase 1 for precise localization and Markov filtering in Phase 2 for optimized tracking.
2. **Phase 1 Localization**: The bot employs A* search with a heuristic that focuses on probable areas of resolution, leveraging clustering of potential bot positions to reduce unnecessary movements and enhance localization accuracy.
3. **Phase 2 Tracking**: The bot uses a Markov filter to track the static rat, updating belief distributions based on sensor feedback (*ping* or *no ping*) and normalizing the resulting probabilities.
4. **A* Search for Movement**: Movement toward the most probable rat positions is guided by a combination of Manhattan distance and weights derived from probability distributions.

5. **Weighted Heuristic for Movement**: The heuristic for prioritizing moves is defined as:

$$W(i,j) = P(i,j) \cdot \frac{1}{1 + d(i,j)}$$

where $P(i,j)$ is the probability of the rat being in cell (i,j) and $d(i,j)$ is the Manhattan distance from the bot's current position. This heuristic balances proximity and probability to optimize tracking.

6. **Movement Reward Calculation**: The movement reward is determined by the improvement in total probability before and after a movement step:

$$\text{Movement Reward} = \max(0, P_{\text{new}} - P_{\text{old}})$$

where P_{new} and P_{old} are the total probabilities of the rat's location after and before the movement, respectively.

7. **Entropy Calculation for Sensing Reward**: The sensing reward is based on the reduction of entropy in the belief distribution:

$$\text{Sensing Reward} = \max(0, H_{\text{old}} - H_{\text{new}})$$

where H_{old} and H_{new} represent the entropy before and after sensing, quantifying the decrease in uncertainty.

8. **Utility-Based Decision Making**: The bot combines movement and sensing rewards to compute Upper Confidence Bound (UCB) scores, enabling it to balance exploration (trying new actions) and exploitation (choosing actions with high expected rewards) for optimal decision-making.

How Our Bot Makes Use Of The Information Available For Future Actions:

1. Combined Probability Calculation:

- The combined probability for the bot's estimate of the space rat's position takes into account both the likelihood of receiving a ping and not receiving a ping, based on the distance from the rat:

$$\text{Combined Probability} = \text{Prob with Ping} \times \text{Prob of Receiving Ping} + \text{Prob without Ping} \times (1 - \text{Prob of Receiving Ping})$$

- **Probability of Receiving Ping**: This is computed based on the distance between the bot and the position with the highest estimated probability for the rat. It decays exponentially with increasing distance.
- **Probability with Ping**: This probability is calculated for each potential rat position under the assumption that a ping is received.
- **Probability without Ping**: This is computed for each potential position assuming that no ping is received. These probabilities are calculated using our knowledge base updation in **Q1)**.
- **Normalization**: The combined probabilities for all potential rat positions are normalized so that their sum equals 1, ensuring a valid probability distribution:

$$\text{Normalized Probability}(pos) = \frac{\text{Combined Probability}(pos)}{\sum_{p \in \text{possible rat positions}} \text{Combined Probability}(p)}$$

2. Entropy Calculation and Use in Decision-Making:

- Entropy is used to measure the uncertainty in the probability distribution of the rat's location. It is calculated as:

$$\text{Entropy} = -\sum_{p \in \text{possible rat positions}} \text{Probability}(p) \cdot \log(\text{Probability}(p))$$

- A reduction in entropy after sensing indicates that the bot has gained information and reduced uncertainty about the rat's location.

3. Sensing Reward Calculation:

- When the bot performs a sensing action, the change in entropy is calculated:

$$\text{Sensing Reward} = \text{Old Entropy} - \text{New Entropy}$$

- A positive entropy change (reduction in uncertainty) is used to calculate a reward for sensing actions, encouraging actions that lead to better localization of the rat.

4. Movement Reward Calculation:

- When the bot moves, the movement reward is based on the improvement in the probability distribution of the rat's estimated position. The total probability for potential rat positions is computed before and after the movement:

$$\text{Total Probability Before} = \sum_{pos \in \text{possible rat positions}} \text{Probability}(pos, \text{before movement})$$

$$\text{Total Probability After} = \sum_{pos \in \text{possible rat positions}} \text{Probability}(pos, \text{after movement})$$

- The movement reward is calculated as:

$$\text{Movement Reward} = \max(0, \text{Total Probability After} - \text{Total Probability Before})$$

- This reward incentivizes movements that lead to a higher probability of accurately estimating the rat's position.

5. Decision-Making with UCB Scores:

- The bot uses UCB-based decision-making to choose between movement and sensing actions. UCB scores are calculated for both actions, balancing exploration (trying new actions) and exploitation (choosing actions with high expected rewards):

$$\text{UCB Score} = \frac{\text{Total Reward for Action}}{\text{Number of Times Action Taken}} + c \times \sqrt{\frac{\log(\text{Total Actions Taken})}{\text{Number of Times Action Taken}}}$$

- Here, c is the exploration parameter, controlling the trade-off between exploration and exploitation (in our calculations, $c = 2$).

T.P.O

Q3) ***Performance Evaluation & Comparative Analysis of Improved Bot vs Baseline Bot:***

1. Comparison of Total Number of Actions:

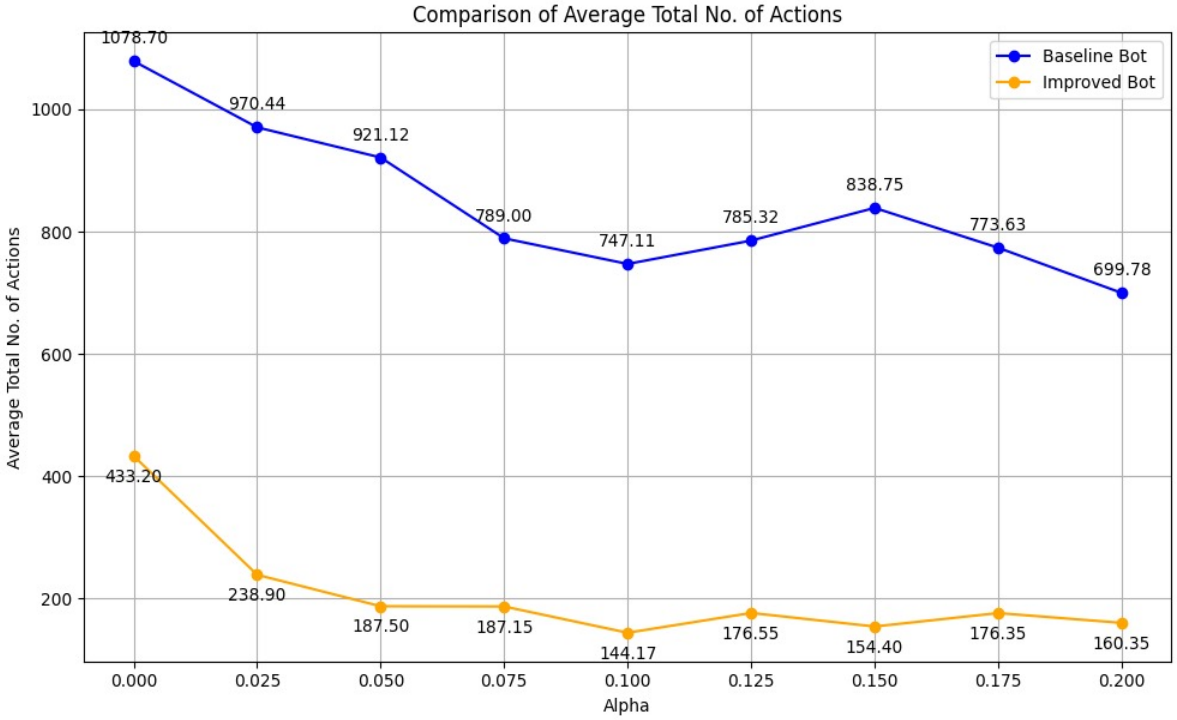

Figure 6: Average Total No. of Actions vs. Alpha (α)

What is plotted:

- **X-axis (Alpha α):** Represents the *sensitivity of the detector*. Smaller α values allow the bot to detect the rat from farther distances, while larger α values cause the detection probability to decay rapidly with distance. The range of α is from **0.0 to 0.2** and **0 to 0.5**, with intervals of **0.025** and **0.1**, respectively.
- **Y-axis (Average Total Number of Actions):** Indicates the total number of actions taken by the bot to locate and capture the rat. The total number of actions is the sum of:
  - **Movement Actions:** Steps taken to navigate the grid (Phase 1 + Phase 2).
  - **Detection Actions:** Use of the detection system to sense the rat's presence (*ping or no ping*).
  - **Blocked Cell Actions:** Actions involving unsuccessful moves due to blocked cells.

  The range is from **0 to 1200**, with intervals of **200**.

Trend:
The baseline bot shows a decreasing trend in total actions as α increases, starting from **1091 actions** at low alpha to **589 actions** around moderate alpha, before increasing again. The improved bot consistently performs fewer total actions, stabilizing between **156 to 220 actions** across lower α values and gradually increasing to **315 actions** at the highest alpha.

Difference:
The improved bot consistently outperforms the baseline bot by requiring significantly fewer total actions,

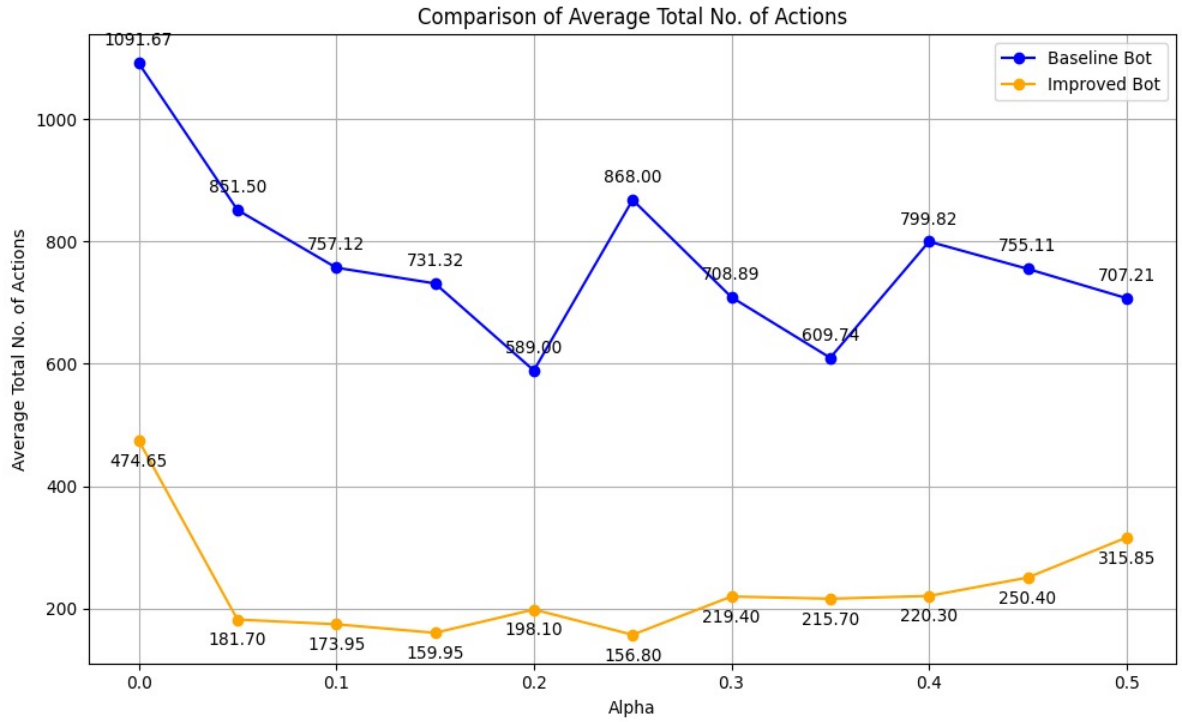

Figure 7: Average Total No. of Actions vs. Alpha (α)

highlighting its *efficiency in combining movement and detection strategies*. This difference is most pronounced at lower alpha values, where the baseline bot's inefficiency is more evident.

2. Comparison of Rat Detector Actions:

What is plotted:

- **X-axis (Alpha α):** Represents the *sensitivity of the detector*. Smaller α values allow the bot to detect the rat from farther distances, while larger α values cause the detection probability to decay rapidly with distance.The range of α is from **0.0 to 0.2** and **0 to 0.5** with intervals of **0.025** and **0.1** respectively.
- **Y-axis (Average Rat Detector Actions):** Shows the average number of times the bot uses its detection system to check for the rat's presence (ping or no ping). The range is from **0 to 450**, with intervals of **50**.

Trend:
The baseline bot reduces its number of detector actions significantly as α increases, dropping from approximately **420 at low alpha** to **196 at high alpha**. Conversely, the improved bot maintains a *stable performance*, requiring only **20-40 detector actions** across all alpha values.

Difference:
The improved bot demonstrates enhanced *detection efficiency*, consistently requiring **far fewer actions** than the baseline bot, regardless of the α value.

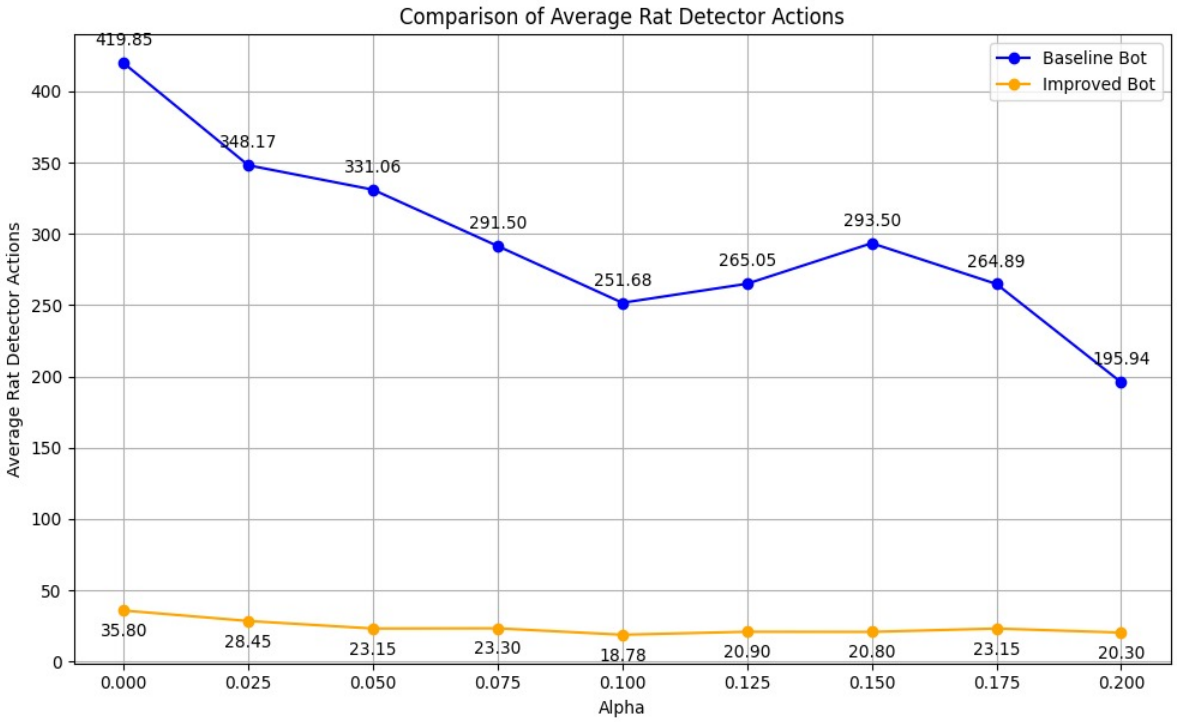

Figure 8: Average No. of Rat Detection Actions vs. Alpha (α)

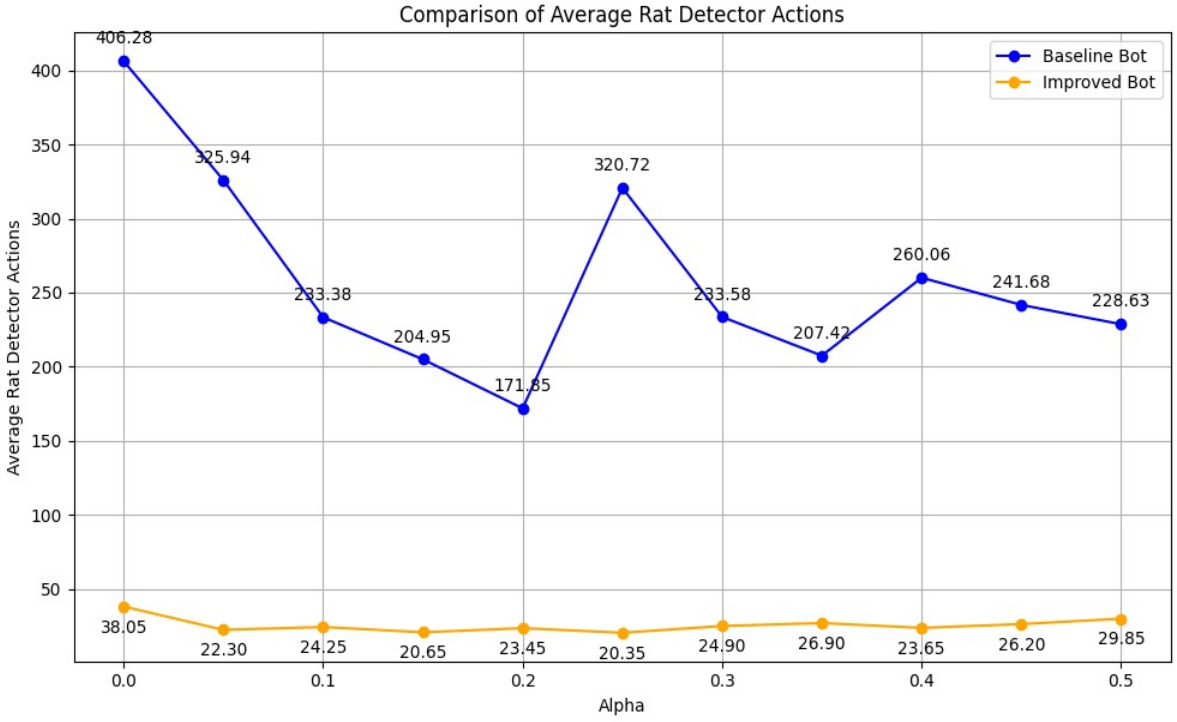

Figure 9: Average No. of Rat Detection Actions vs. Alpha (α)

3. Comparison of Average Movements:

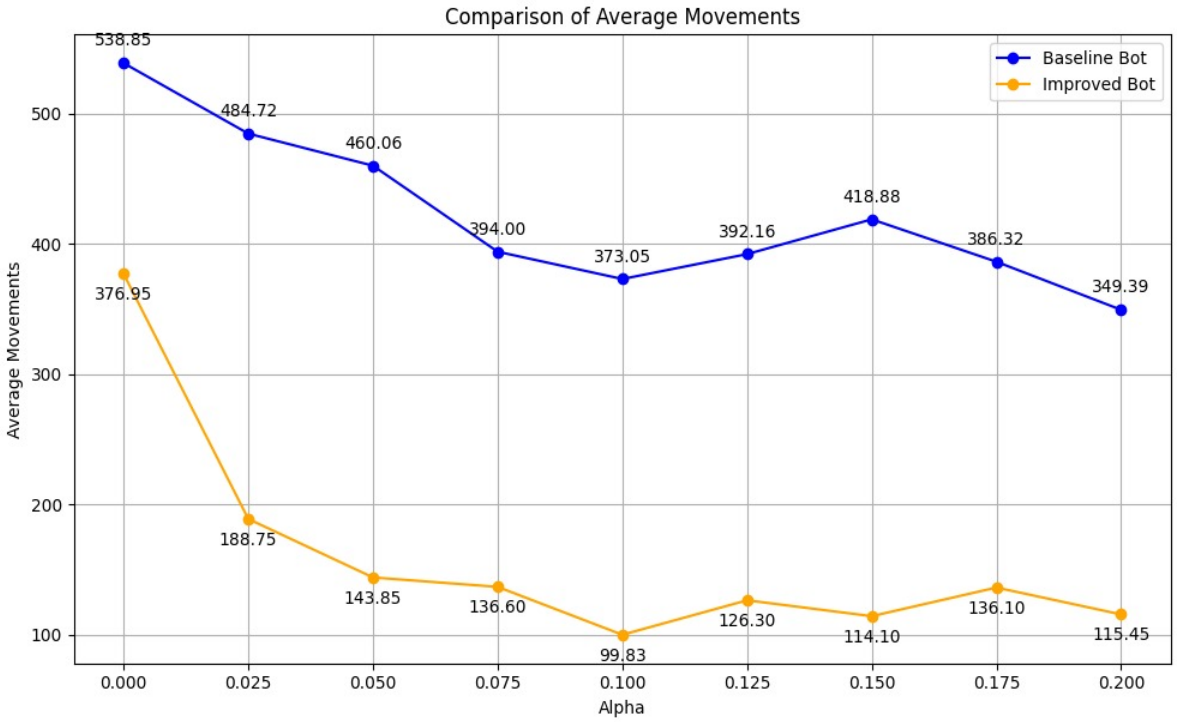

Figure 10: Average No. of Movements vs. Alpha (α)

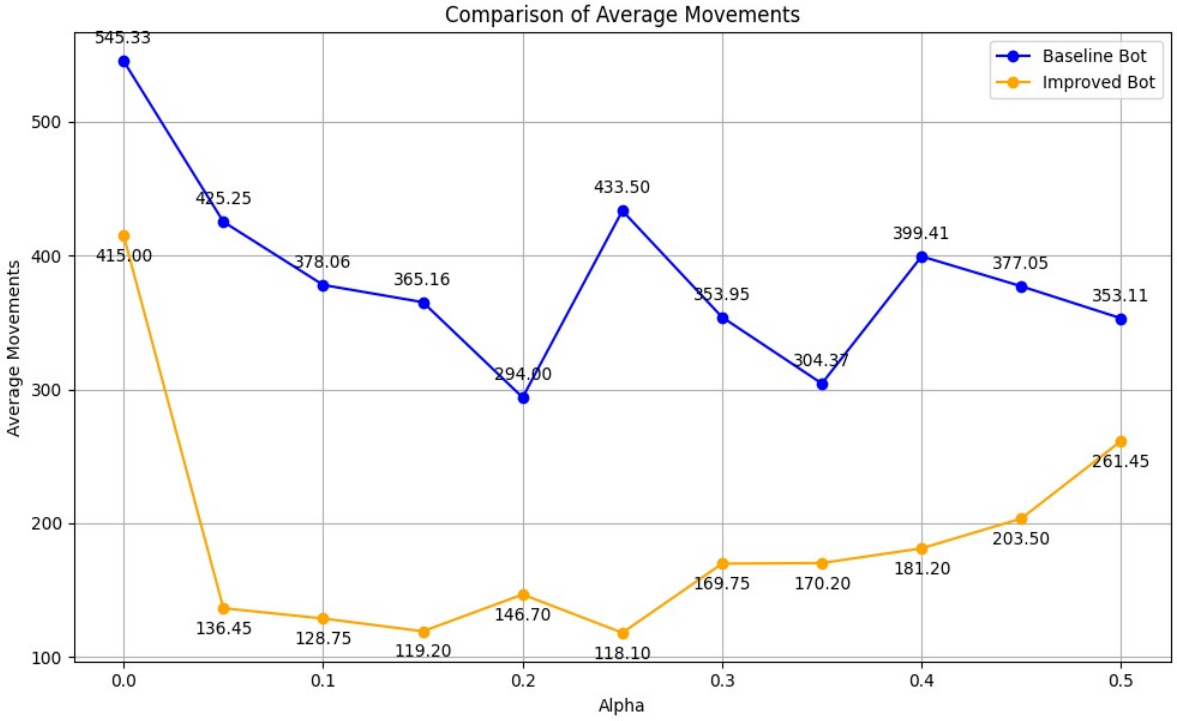

Figure 11: Average No. of Movements vs. Alpha (α)

What is plotted:

- **X-axis (Alpha α):** Represents the *sensitivity of the detector*. Smaller α values allow the bot to detect the rat from farther distances, while larger α values cause the detection probability to decay rapidly with distance. The range of α is from **0.0 to 0.2** and **0 to 0.5** with intervals of **0.025** and **0.1** respectively.
- **Y-axis (Average Movements):** Indicates the average number of movements the bot makes during the simulation to locate and capture the rat. The range is from **0 to 600**, with intervals of **50**.

Trend:
The baseline bot shows a decrease in movements as α increases, from approximately **539 movements at low alpha** to **349 movements at high alpha**. The improved bot significantly reduces movements across all α values, stabilizing at approximately **115-140 movements**.

Difference:
The improved bot demonstrates superior *efficiency in movement*, achieving its goals with far fewer movements compared to the baseline bot.

→***Movement v/s Sensing:***

Improved Bot: The ratio of movements to sensor actions is notably higher. This indicates that the improved bot uses the information provided by the detector far more efficiently, reducing the reliance on constant sensing and focusing more on calculated movements.

Baseline Bot: The ratio is considerably lower in the baseline case, suggesting that it relies heavily on frequent sensor actions to make incremental progress. This highlights a less efficient use of the detector information, leading to much slower decision-making.

4. Comparison of Blocked Cell Actions:

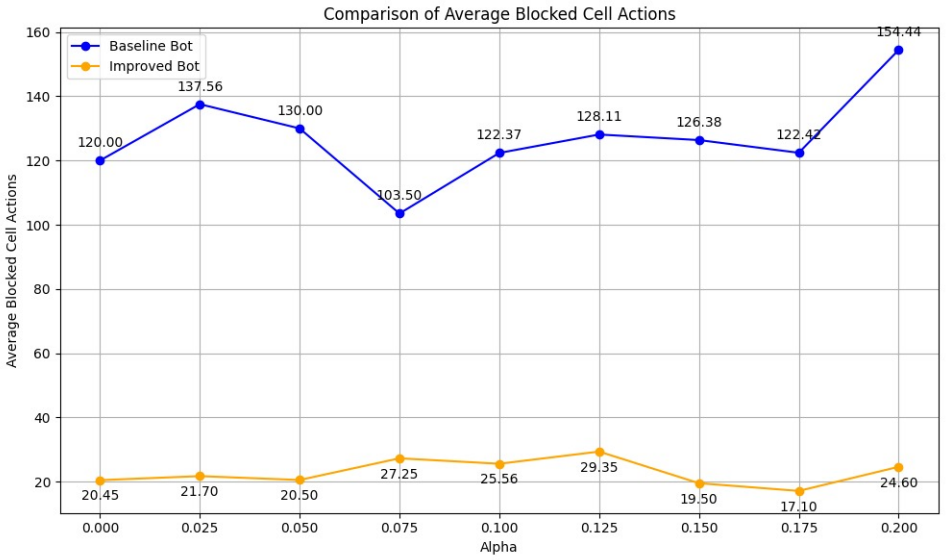

Figure 12: Average Blocked Cell Actions vs. Alpha (α)

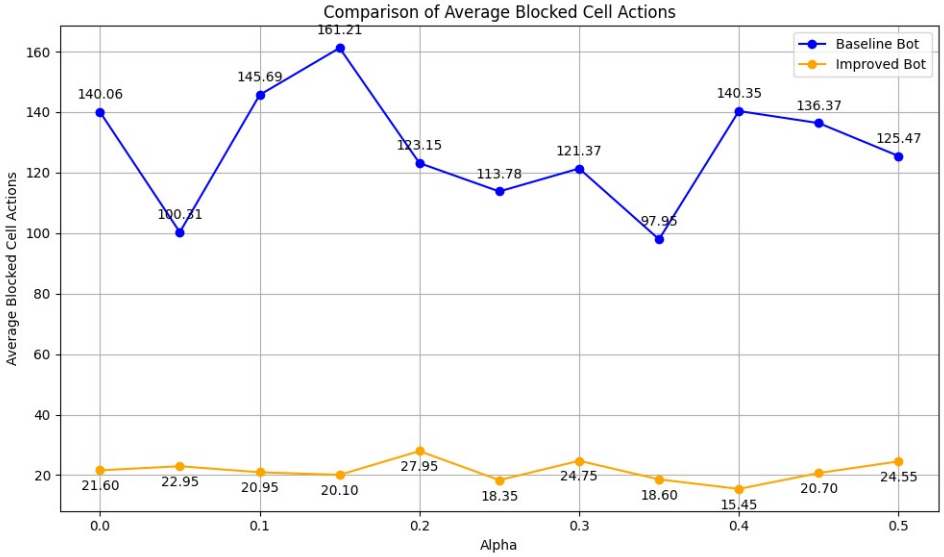

Figure 13: Average Blocked Cell Actions vs. Alpha (α)

What is plotted:

- **X-axis (Alpha α):** Represents the *sensitivity of the detector*. Smaller α values allow the bot to detect the rat from farther distances, while larger α values cause the detection probability to decay rapidly with distance. The range of α is from **0.0 to 0.2** and **0 to 0.5**, with intervals of **0.025** and **0.1**, respectively.
- **Y-axis (Average Blocked Cell Actions):** Shows the average number of blocked cell actions taken by the bot during the simulation. The range is from **0 to 160**, with intervals of **20**.

Trend:
The baseline bot exhibits variability in blocked cell actions, with values ranging from approximately **120 to 154 actions** as α increases. In contrast, the improved bot consistently performs fewer blocked cell actions, stabilizing around **20-30 actions** across all alpha values.

Difference:
The improved bot demonstrates better *pathfinding efficiency*, minimizing unnecessary blocked cell actions compared to the baseline bot.

Q4) ***Dynamic Rat Situation:***

→***Space Rat Knowledge Base Updates:***

1. Transition Probabilities in the Dynamic Rat Scenario:
In the dynamic scenario, a predictive model is introduced to handle the rat's potential movements using transitional probabilities, considering all valid moves from a given rat position (x, y).

Valid Moves Calculation :

For each possible rat position (x, y), the valid moves (open cells within grid boundaries) are denoted as valid_moves. If there are N valid moves from (x, y), the transition probability for moving to a neighboring

cell is given by:

$$\text{transition\_prob}(x, y \to \text{move}) = \frac{1}{N}$$

This indicates that the rat is equally likely to move to any available open direction.

2. Incorporating Transition Probabilities in the Knowledge Base:

The knowledge base is updated using the following steps:

Initial Probability Update (Ping/No Ping):

The probabilities are first updated based on whether the bot receives a ping or not, using an exponential decay function:

$$\text{probability\_of\_ping} = e^{-\alpha \times (d-1)}$$

where d is the Manhattan distance between the bot and the rat's current position. The prior probability of the rat's position is updated by multiplying it by probability_of_ping if a ping is received, or by $1 -$ probability_of_ping otherwise.

Predictive Transition Update:

The next step incorporates the transition probabilities to predict the rat's movements. The new probability for the rat being at position (x, y) at the next time step is given by:

$$P_{\text{new}}(x, y) = \sum_{\text{valid moves } (x', y')} P_{\text{prev}}(x', y') \times \text{transition\_prob}(x', y' \to x, y)$$

Here:

- $P_{\text{prev}}(x', y')$ is the prior probability of the rat being at position (x', y').
- transition_prob$(x', y' \to x, y)$ represents the probability of the rat moving from (x', y') to (x, y).

3. Normalization:

After applying the transition probabilities, the new probabilities are normalized to ensure they sum to 1 across all possible rat positions:

$$P_{\text{normalized}}(x, y) = \frac{P_{\text{new}}(x, y)}{\sum_{\text{all } (x', y')} P_{\text{new}}(x', y')}$$

Summary of the Differences

- **Static Rat:** The probability update is solely based on sensing (ping/no ping).
- **Dynamic Rat:** In addition to sensing updates, a transition probability knowledge base is maintained, which predicts the rat's movements and updates probabilities accordingly.

→Performance Comparison in Dynamic Rat Environment:

1. Comparison of Total Actions

What is plotted:

- **X-axis (Alpha α):** Represents detector sensitivity with ranges: - Figure 14: 0.0 to 0.2 (0.025 intervals) - Figure 15: 0.0 to 0.5 (0.1 intervals)
- **Y-axis (Average Total Actions):** Total actions combining: - Movement Actions (Phase 1 + 2) - Detection Actions (ping/no ping) - Blocked Cell Actions Range: 0 to 1200, intervals of 200

Trend:

The baseline bot shows high variability, requiring 850-1230 actions across values. The improved bot demonstrates significantly lower action counts, averaging 160-450 actions, with optimal performance around = 0.1.

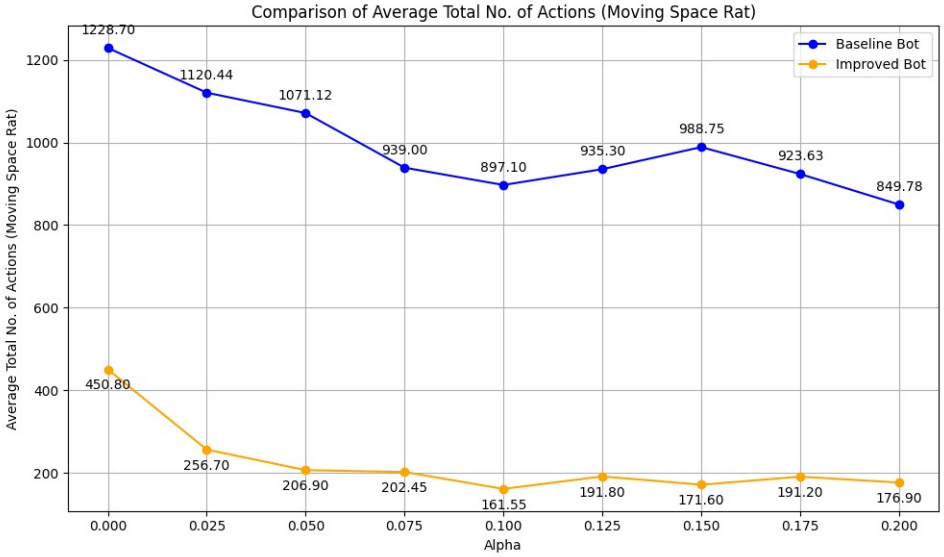

Figure 14: Average Total Actions vs. Alpha (α) [0-0.2]

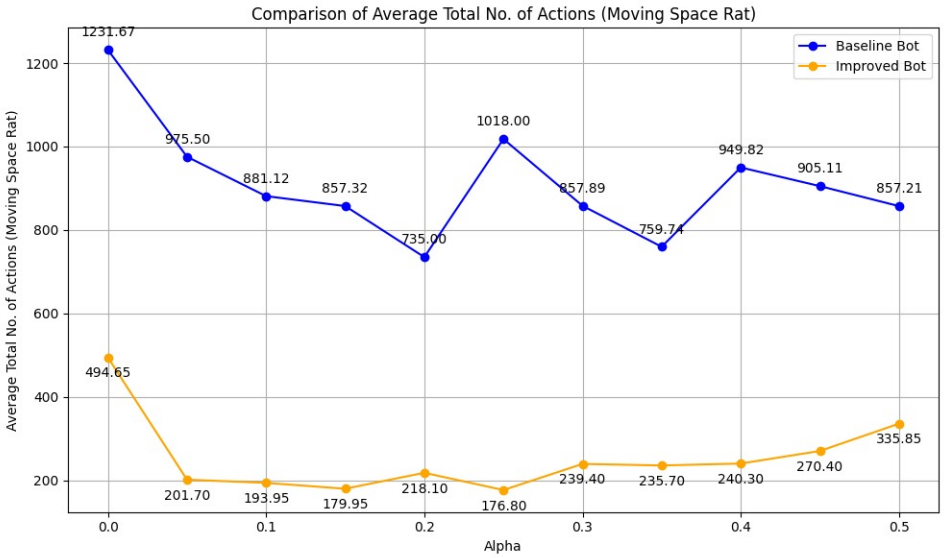

Figure 15: Average Total Actions vs. Alpha (α) [0-0.5]

Difference:
The improved bot achieves approximately 70-75% reduction in total actions needed for successful capture, showing superior efficiency in both movement and sensing strategies.

Efficiency In Dynamic Rat Case:

Improved Bot: The improved bot maintains a consistently low number of actions due to its use of transitional probabilities in the knowledge base updates, accounting for the dynamic movement of the rat. The improved bot demonstrates a higher capability to anticipate and respond to changes, minimizing unnecessary movements.

Baseline Bot: The baseline bot, on the other hand, lacks consideration for the dynamic behavior of the rat. This significantly hampers its performance when faced with unpredictable rat movement.

2. Comparison of Rat Detector Actions:

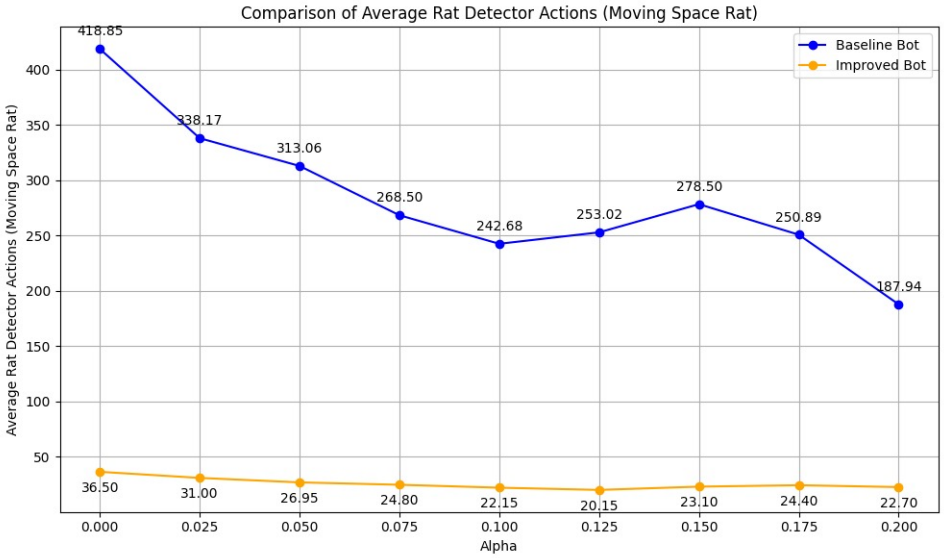

Figure 16: Average Rat Detector Actions vs. Alpha (α) [0-0.2]

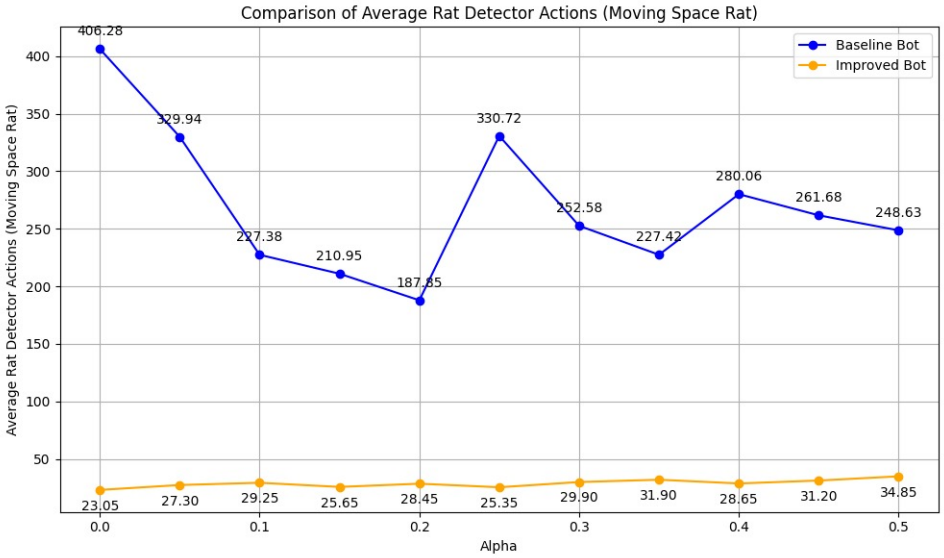

Figure 17: Average Rat Detector Actions vs. Alpha (α) [0-0.5]

What is plotted:

- **X-axis (Alpha α):** Shows detector sensitivity ranges: - Figure 16: 0.0 to 0.2 (0.025 intervals) - Figure 17: 0.0 to 0.5 (0.1 intervals)
- **Y-axis (Rat Detector Actions):** Frequency of ping detection usage Range: 0 to 450 actions

Trend:
The baseline bot shows high detector usage (180-420 actions) with significant variation across values. The improved bot maintains consistently low detector usage (20-35 actions) across all values, showing remarkable stability.

Difference:
The improved bot achieves approximately 90% reduction in detector usage compared to baseline, indicating much more efficient use of sensing actions through strategic decision-making.

3. Comparison of Average Blocked Cell Actions

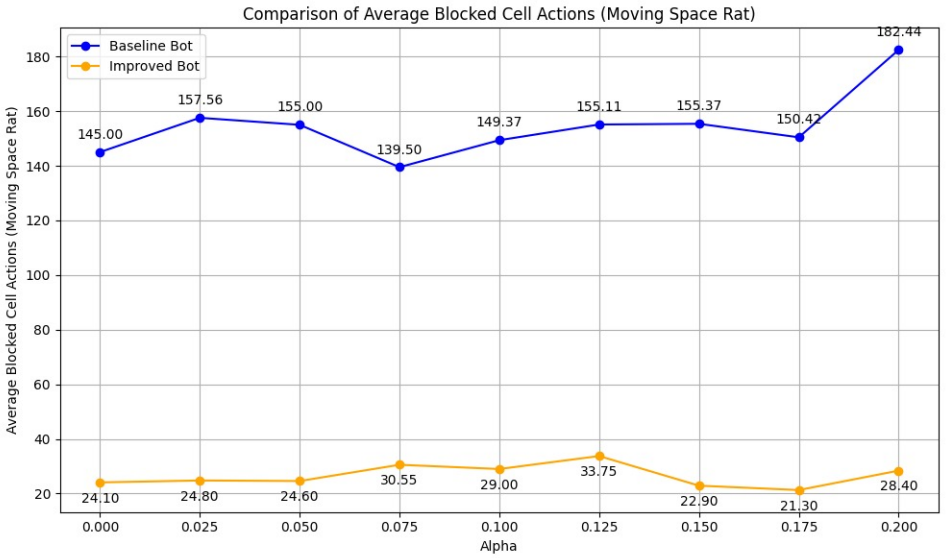

Figure 18: Average Blocked Cell Actions vs. Alpha (α) [0-0.2]

What is plotted:

- **X-axis (Alpha α):** Shows detector sensitivity: - Figure 18: 0.0 to 0.2 (0.025 intervals) - Figure 19: 0.0 to 0.5 (0.1 intervals)
- **Y-axis (Blocked Cell Actions):** Wall detection attempts Range: 0 to 200 actions

Trend:
The baseline bot shows relatively high wall detection activity (140-180 actions) with notable fluctuation. The improved bot maintains consistently low wall detection counts (20-40 actions) across all values.

Difference:
The improved bot achieves approximately 80% reduction in wall detection actions, indicating more efficient path planning and obstacle avoidance.

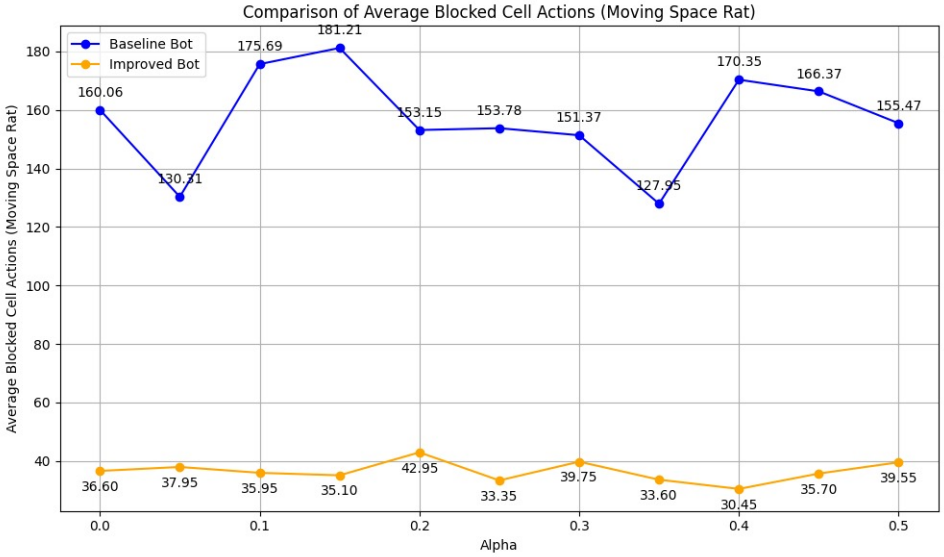

Figure 19: Average Blocked Cell Actions vs. Alpha (α) [0-0.5]

4. Comparison of Average Movements

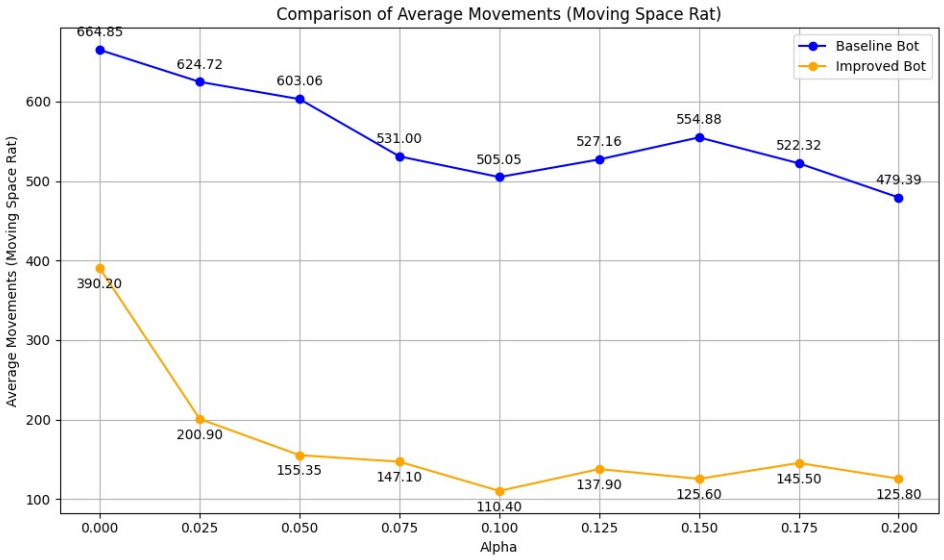

Figure 20: Average Movements vs. Alpha (α) [0-0.2]

What is plotted:

- **X-axis (Alpha α):** Shows detector sensitivity ranges: - Figure 20: 0.0 to 0.2 (0.025 intervals) - Figure 21: 0.0 to 0.5 (0.1 intervals)
- **Y-axis (Average Movements):** Number of movement actions Range: 0 to 700 actions

Trend:

The baseline bot requires substantial movement (450-650 actions) with high variability across values. The improved bot shows significantly reduced movement (100-400 actions), with best performance at 0.1.

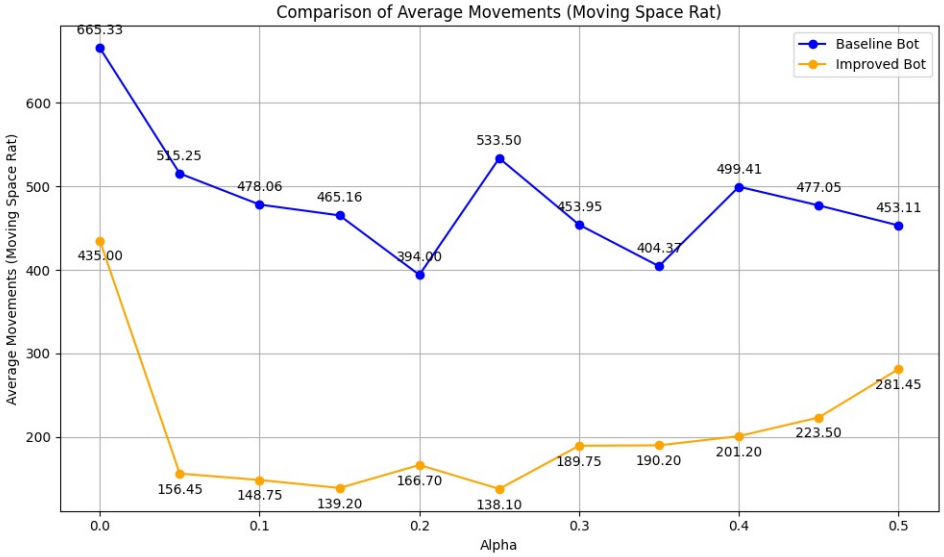

Figure 21: Average Movements vs. Alpha (α) [0-0.5]

Difference:
The improved bot achieves approximately 65-70% reduction in movement actions, demonstrating more efficient pathfinding and target pursuit strategies.

→Bot Design Improvements for Moving Target:

1. Incorporating Entropy Change in UCB Decision-Making :

Entropy Calculation: Given a probability distribution $P(r_x, r_y)$ over possible rat positions, the entropy H is calculated as:

$$H = -\sum_{(r_x, r_y)} P(r_x, r_y) \log(P(r_x, r_y) + \epsilon)$$

where ϵ is a small constant added to handle $\log(0)$ cases gracefully.

Entropy Change: Before taking any action, the bot calculates the entropy H_{before} of its current knowledge about the rat's location. After updating its knowledge (e.g., after sensing or moving), it calculates H_{after}. The change in entropy, which reflects a reduction in uncertainty, is given by:

$$\Delta H = H_{\text{before}} - H_{\text{after}}$$

Incorporation into UCB Framework: The UCB value for choosing an action a is adjusted to include a component that rewards actions reducing entropy, encouraging the bot to prioritize actions that minimize uncertainty:

$$\text{UCB}(a) = \left(\frac{\text{Total Reward}(a)}{\text{Action Count}(a)}\right) + c\sqrt{\frac{\log(\text{Total Actions})}{\text{Action Count}(a)}} + \beta \cdot \Delta H(a)$$

Here, c controls the exploration-exploitation balance, and β determines the emphasis on entropy reduction.

2. Improved A* Movement Logic for Dynamic Rat Case:

Predictive Heuristics for Movement: The bot uses predictive heuristics to account for the rat's potential movements. Instead of targeting only the highest-probability position at the current moment, it considers regions where the rat is more likely to move next based on the transition model. This approach adjusts the bot's movement toward areas that are expected to have a higher cumulative rat presence probability over the next steps.

Dynamic Goal Adjustment: The target position within the A* search is adjusted dynamically based on changes in the probability distribution as new data arrives (e.g., from sensing or movement updates).

Heuristic Score Calculation: The heuristic score $h(n)$ for evaluating a position $n = (x, y)$ during the A* search is modified to consider both the Manhattan distance to the predicted target position and the probability of the rat being there:

$$h(n) = \text{Manhattan Distance}(n, \text{Predicted Target}) - \lambda \cdot P_{\text{predicted}}(n)$$

Here, $P_{\text{predicted}}(n)$ is the predicted probability of the rat being at position n in future steps, and λ is a weight balancing distance minimization with the predicted likelihood of encountering the rat. This approach allows the bot to adapt its path based on both static and dynamic probabilities, improving its chances of intercepting the rat by considering its potential movement trajectory.

T.P.O

References

[1] D. Ashlock, C. Lee, and C. McGuinness, *Search-Based Procedural Generation of Maze-Like Levels*, IEEE Transactions on Computational Intelligence and AI in Games, vol. 3, no. 3, pp. 260–273, Sept. 2011. doi:10.1109/TCIAIG.2011.2138707.

[2] N. K. Kitson, A. C. Constantinou, Z. Guo, et al., *A Survey of Bayesian Network Structure Learning*, Artificial Intelligence Review, vol. 56, pp. 8721–8814, 2023. doi:10.1007/s10462-022-10351-w.

[3] R. Suwanda, Z. Syahputra, and E. Zamzami, *Analysis of Euclidean Distance and Manhattan Distance in the K-Means Algorithm for Variations Number of Centroid K*, Journal of Physics: Conference Series, vol. 1566, pp. 012058, 2020. doi:10.1088/1742-6596/1566/1/012058.

[4] S. Ferson and W. T. Tucker, *Sensitivity Analysis Using Probability Bounding*, Reliability Engineering System Safety, vol. 91, issues 10–11, pp. 1435–1442, 2006. doi:10.1016/j.ress.2005.11.052.

[5] P. K. Panigrahi and S. K. Bisoy, *Localization Strategies for Autonomous Mobile Robots: A Review*, Journal of King Saud University - Computer and Information Sciences, vol. 34, issue 8, part B, pp. 3456–3467, 2022. doi:10.1016/j.jksuci.2021.02.015.

[6] S. Liu, D. Sun, and C. Zhu, *A Dynamic Priority Based Path Planning for Cooperation of Multiple Mobile Robots in Formation Forming*, Robotics and Computer-Integrated Manufacturing, vol. 30, no. 6, pp. 589–595, 2014. doi:10.1016/j.rcim.2014.04.002.

[7] B. Sugandi and A. D. Zain, *Localization of Wheeled Soccer Robots Using Particle Filter Algorithm*, 2019 2nd International Conference on Applied Engineering (ICAE), Batam, Indonesia, pp. 1–4, 2019. doi:10.1109/ICAE47758.2019.9221670.

[8] D. Chen, F. Trevizan, and S. Thiebaux, *Heuristic Search for Multi-Objective Probabilistic Planning*, arXiv preprint, 2023. doi:10.48550/arXiv.2303.14363.

[9] N. Hazim, S. S. M. Al-Dabbagh, and M. A. S. Naser, *Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm*, Journal of Computer and Communications, vol. 4, pp. 15–25, 2016. doi:10.4236/jcc.2016.411002.

[10] D. P. Bertsekas and J. N. Tsitsiklis, *An Analysis of Stochastic Shortest Path Problems*, Mathematics of Operations Research, vol. 16, no. 3, pp. 580–595, 1991.

[11] D. P. Bertsekas and J. N. Tsitsiklis, *An Analysis of Stochastic Shortest Path Problems*, Mathematics of Operations Research, vol. 16, no. 3, pp. 580–595, 1991.

[12] P. Baumgartner, S. Thiébaux, and F. Trevizan, *Heuristic Search Planning With Multi-Objective Probabilistic LTL Constraints*, in: Proceedings of the 16th International Conference on Principles of Knowledge Representation and Reasoning (KR), 2018.

End of Document