INTRODUCTION TO ARTIFICIAL INTELLIGENCE
PROJECT 2

MIHIR KULKARNI MAK575
SHAIVI BANSAL SB2756

1) EXPLAIN HOW THE SPACE RAT KNOWLEDGE BASE SHOULD BE UPDATED, BASED ON WHETHER THE BOT RECEIVES A PING OR DOES NOT RECEIVE A PING.

In order to track the space rat's location, the bot maintains a knowledge base which assigns a probability value to all the cells in the grid. This probability value initially depicts the likelihood of the rat being in a particular cell.
At alternating timesteps, the bot receives an input from the rat sensor (in the form of a "ping" or "no ping"). The input from the sensor is used to adjust the probabilities of the cells in order to refine and improve the estimation of the rat's position over time. Initially each open cell has a similar probability of containing the rat and so the bot has no concrete information about the rat's location. The feedback from the sensor is used to help narrow down these possibilities.

When the bot receives positive feedback from the sensor, i.e. a ping has been heard, it ideally implies that the rat is close to the bot. This is because the ping probability function is inversely proportional to the Manhattan distance between the position of the bot and the rat. With increasing distance, the probability of hearing a ping decreases. Due to this property, we applied Bayes' Theorem to update the bot's knowledge base. The theorem increases the probability of the nearby cells containing the rat while decreasing the probability for farther distant cells. Using the ping probability function, this approach helps the bot focus on the most probable areas (in this case, the cells closer to it).

Whereas if the bot receives negative feedback, i.e. does not hear a ping, it implies that the rat is farther away from the bot. This information is used by Bayes' Theorem to decrease the probability of closer cells and increase the probability of distant cells. It uses the absence of the ping to rule out closer regions and focus on the most probable areas (in this case, cells that are out of a particular range).

This update process is applied iteratively every alternative timestep. As more sensor input is gathered, the probability distribution (knowledge base of the bot) becomes more and more accurate and eventually converges on the rat's true position. Positive and negative feedback are both handled by the bot dynamically to refine its search and narrow down the rat's location effectively.

Update Formulas:
Given Information:

- bot_pos: Current position of the bot in the grid.
- rat_pos: Position of the space rat in the grid.
- alpha: Sensitivity parameter that influences the probability of receiving a ping.
- prob_grid: Probability grid where each cell (x,y) holds the probability that the rat is located there.
- hear_prob: Boolean indicating whether a ping was detected (True) or not (False).

Ping Probability: Given the bot position (bot_pos) and a grid cell (j), the probability of hearing a ping when space rat is in cell j:

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

Where, d(bot_pos, j) is the Manhattan Distance between cells bot_pos and j
And α (alpha) depicts the sensitivity of the sensor.

Updating Probability Grid when a Ping is Detected: If hear_prob = True (ping was heard), according to Bayes' Rule, the probabilities of the closer cells increases and probability of the farther cells decreases.

$$P(j \mid \text{ping}) = \frac{P\big(\text{ping} \mid d(i,j)\big) \cdot P(j)}{\sum_k P\big(\text{ping} \mid d(i,k)\big) \cdot P(k)}$$

Where,

- P(j) is the prior probability that the rat is in cell j.
- P(ping | d(i, j)) is the probability of detecting a ping if the rat is in cell j.
- The denominator normalizes the probabilities across all cells to ensure they sum to 1.

Updating the Probability Grid when No Ping is Detected: If hear_prob = False (ping was not heard), according to Bayes' Rule, the probabilities of the distant cells will increase and probability of the closer cells will decrease.

$$P(j \mid \text{noping}) = \frac{\Big(1 - P\big(\text{ping} \mid d(i,j)\big)\Big) \cdot P(j)}{\sum_k \Big(1 - P\big(\text{ping} \mid d(i,k)\big)\Big) \cdot P(k)}$$

Where,

- 1 − P(ping | d(i, j)) represents the probability of not hearing a ping when the rat is in cell j.
- The denominator normalizes the probabilities across all cells to ensure they sum to 1.

Below is a **solved example** that we computed for our understanding of the project and how the probabilities could be updated. We have attached it here to help convey the concept of our implementation with a simplified simulation.

```
# Example 5x5 grid where 1 represents blocked cells, 0 represents open
cells
# for the sake of simplicity, we have assumed that all inner cells are
open in this example

grid = [
    [1, 1, 1, 1, 1],
    [1, 0, 0, 0, 1],
    [1, 0, 0, 0, 1],
    [1, 0, 0, 0, 1],
    [1, 1, 1, 1, 1]
]

# Step 1: Initializing the probability map - Initially, there would be
an equal chance of all open cells containing the rat
# 9 open cells --> each has 1/9 probability = 0.111

initial_probs = [
    [None,   None,   None,   None,   None],
    [None,   0.111,  0.111,  0.111,  None],
    [None,   0.111,  0.111,  0.111,  None],
    [None,   0.111,  0.111,  0.111,  None],
    [None,   None,   None,   None,   None]
]

# Assuming bot is at cell (1,1)

# Assuming α = 0.1

# In order to use the ping probability function, we first calculate the
manhattan distance between the bot and each cell (represented as d(i,
j))
# Manhattan distances from (1,1):
distances = [
    [None,   None,   None,   None,   None],
    [None,   0,      1,      2,      None],
    [None,   1,      2,      3,      None],
    [None,   2,      3,      4,      None],
    [None,   None,   None,   None,   None]
]

# Probability of ping at each distance with α = 0.1 (using prob
formula, ping_probs = e^(-α(d(i, j) - 1)))
ping_probs = [
```

```
    [None,   None,   None,    None,   None],
    [None,   1.000,  0.905,   0.819,  None],
    [None,   0.905,  0.819,   0.741,  None],
    [None,   0.819,  0.741,   0.670,  None],
    [None,   None,   None,    None,   None]
]

# After this, there are 2 cases. Case 1 is when a ping is heard and
Case 2 when no ping is heard

# CASE 1
'''
If we get a ping:
For each cell, multiply current probability by P(ping_probs)
Example: Cell at (1,2) had 0.111 initially
P(ping_probs) at distance 1 is 0.905
New probability = 0.111 * 0.905 = 0.100)

After a ping: Probabilities become higher for cells closer to the bot
'''
unnorm_probs_after_ping = [
    [None,   None,   None,    None,   None],
    [None,   0.111,  0.100,   0.091,  None],
    [None,   0.100,  0.091,   0.082,  None],
    [None,   0.091,  0.082,   0.074,  None],
    [None,   None,   None,    None,   None]
]

# Normalized probabilities after ping:
norm_probs_after_ping = [
    [None,   None,   None,    None,   None],
    [None,   0.152,  0.137,   0.124,  None],
    [None,   0.137,  0.124,   0.112,  None],
    [None,   0.124,  0.112,   0.101,  None],
    [None,   None,   None,    None,   None]
]

# CASE 2
'''
If we DON'T get a ping:
For each cell, multiply by (1 - P(ping_probs))
Example: Cell at (1,2) had 0.111 initially
P(no ping) at distance 1 is (1 - 0.905) = 0.095
New probability = 0.111 * 0.095 = 0.011

After no ping: Probabilities become higher for cells farther from the
bot
'''
```

```
unnorm_probs_after_no_ping = [
    [None,   None,   None,    None,   None],
    [None,   0.000,  0.011,   0.020,  None],
    [None,   0.011,  0.020,   0.029,  None],
    [None,   0.020,  0.029,   0.037,  None],
    [None,   None,   None,    None,   None]
]

# Normalized probabilities after no ping:
norm_probs_after_no_ping = [
    [None,   None,   None,    None,   None],
    [None,   0.000,  0.062,   0.113,  None],
    [None,   0.062,  0.113,   0.164,  None],
    [None,   0.113,  0.164,   0.209,  None],
    [None,   None,   None,    None,   None]
]
```

2) EXPLAIN THE DESIGN AND ALGORITHM FOR THE BOT THAT YOU DESIGN, BEING AS SPECIFIC AS POSSIBLE AS TO WHAT YOUR BOT IS ACTUALLY DOING. HOW DOES YOUR BOT MAKE USE OF THE INFORMATION AVAILABLE TO MAKE INFORMED DECISIONS ABOUT WHAT TO DO NEXT?

Design and Algorithm for improved bot:

The implementation of the bot is done in 2 phases:

Phase 1: Localization – The bot tries to determine its own position in the grid using elimination and movement strategies based on data that it receives from the sensor.
Phase 2: Rat Catching – After the localization is done, the bot tracks and catches the rat. This is done by using probability-based knowledge base.

Phase 1: Localization

In order to find its own location, the bot iteratively refines and narrows down its knowledge base (bot_kb) which contains the possible locations in the grid. It alternates between sensing its environment (blocked neighbors) and moving in the most open direction to eliminate possible locations.

- The bot knowledge base (bot_kb) starts with all open cells in the grid at timestep 0.
- Blocked Neighbor Sensing: The bot counts how many of its 8 neighboring cells around its current position are blocked (sensing_neighbours_blocked). The bot then uses its updated knowledge base to eliminate all cells whose blocked neighbor count does not match the sensed value.
- Movement: In alternating time steps, the bot finds the most commonly open direction (check_common_direction) across all possible options in the grid. It then attempts to move in that direction (attempt_movement).
- Knowledge Base Update: If the attempted movement was successful, the bot then eliminates all positions from the knowledge base where that direction was blocked. And on the other hand, if the movement fails, the bot removes all positions from the knowledge base where that direction was open.
- The alternating sensing and moving steps continue until there is only one cell remaining in the bot knowledge base. This final cell is identified as the bot's actual position.

Phase 2: Rat Catching

The goal in phase 2 is to track and catch the space rat. Since the location of the rat is unknown to the bot, we use a probability based knowledge base to estimate and find the rat's location. This includes a probability grid which stores the likelihood of each cell containing the rat. We use the input from the sensor (hearing a ping or not) to update the probabilities.

These updated probabilities are used to narrow down the rat's position in the grid.

- Initially the grid has a uniform probability distribution over all open cells. If there are N open cells in the grid, each cell is initialized with a probability of 1/N.
- After this, the entire grid is divided into quadrants in order to focus on particular regions that seem most promising.
- Probability Updates: At each timestep, the bot needs to refine its knowledge base based on the feedback received from the sensor. The sensory input gives the bot an idea about the rat's proximity.

    - A ping indicates that the rat is close by. In this case, the probabilities of the cells closer to the bot's current position are increased. This is done using Bayes' Theorem:

$$P(x', y' \mid \text{ping}) = \frac{P\big(\text{ping} \mid d(\text{bot\_pos}, x', y')\big) \cdot P(x', y')}{\sum_{(i,j)} P\big(\text{ping} \mid d(\text{bot\_pos}, i, j)\big) \cdot P(i, j)}$$

Where, P(x', y' | ping): Prior probability of the rat being in cell (x', y')
P (ping | d(bot_pos, x', y')): Likelihood of detecting a ping

    - No ping indicates that the rat is far away. In this case, the probabilities of cells farther from the bot are increased. This is done using Bayes' Theorem:

$$P(x', y' \mid \text{noping}) = \frac{\Big(1 - P\big(\text{ping} \mid d(\text{bot\_pos}, x', y')\big)\Big) \cdot P(x', y')}{\sum_{(i,j)} \Big(1 - P\big(\text{ping} \mid d(\text{bot\_pos}, i, j)\big)\Big) \cdot P(i, j)}$$

    - Updating the probabilities using Bayes' Theorem ensures that the knowledge base (probability grid) reflects the most likely positions of the rat.

Updating the probabilities using Bayes' Theorem ensures that the knowledge base (probability grid) reflects the most likely positions of the rat.

- Quadrant-Based Target Selection:
    - After calculating and updating the probabilities, the bot calculates the total probability in each quadrant and identifies the quadrant with the highest total probability. This reduces the search space and ensures that the bot focuses on the most likely region to avoid unnecessary exploration. This move prioritizes exploitation over exploration.
    - After finding the target quadrant, the bot calculates a weighted center based on the probabilities of all individual cells in the quadrant. This was done by finding the cell with the highest probability density within the quadrant. This

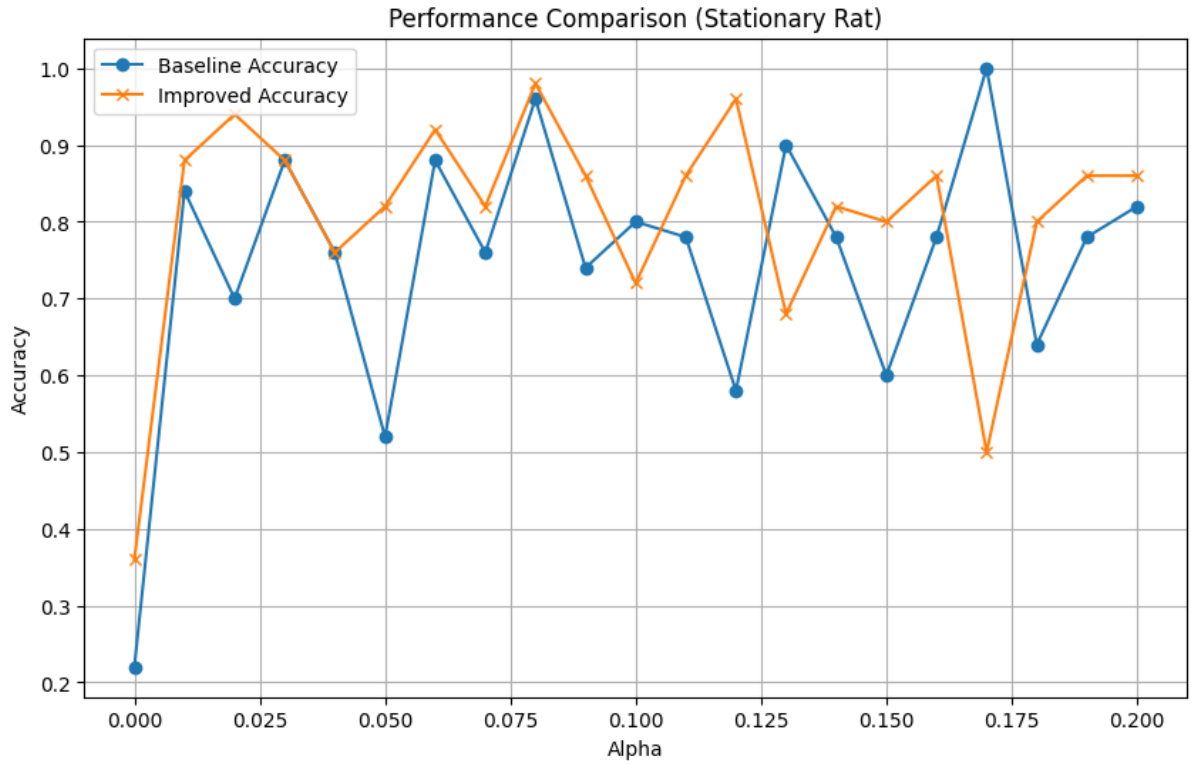
allows the bot to narrow down the possibilities and use this as its next target.

- Movement Toward the Target:
    - After finding the target quadrant, the bot calculates the shortest path to the weighted center using grid navigation logic which avoids blocked cells and prioritizes open paths.
    - We do not want to assume completely that the rat is in the selected quadrant. It is possible that the rat is in a neighbouring quadrant of the selected one, just at the edges. Therefore, in order to minimize the time the bot will take to go the selected weighted center, we move the bot halfway towards the target, ping and sense for the rat again, and if the quadrant is the same as before, commit to going to the target quadrant.

- Refinement After Reaching the Target: If the bot fails to find the rat at the target location, the probabilities of the target quadrant are normalized and redistributed across sub-quadrants based on the sensor output.
    - If the bot reaches the weighted center and does not find the rat there, the bot refines its search by further subdividing the quadrant into smaller sub-quadrants.
    - Each sub-quadrant is used to compute probabilities, and the bot repeats the target selection and movement process for the most likely sub-quadrant.

- If the bot's position matches with the rat's position, the output from the sensor confirms that the rat has been caught and the simulation is ended.

3) EVALUATE YOUR BOT VS THE BASELINE BOT, REPORTING A THOROUGH COMPARISON OF PERFORMANCE. PLOT YOUR RESULTS AS A FUNCTION OF ALPHA.



The above graph evaluates the accuracy of the improved bot against the baseline bot as a function of alpha. We have ran 50 simulations for each individual value of alpha. The alpha values range from 0 to 0.2 with a step size of 0.01.

We can derive the following information from it:

- Considering the general trend, the improved bot consistently outperforms the baseline bot across most values of alpha. This indicates that the improvements made over the baseline bot had a positive impact on the performance.
- Both bots show some fluctuations over few values of alpha but the improved bot still maintains a higher accuracy on average and adapts better to the increasing value of alpha.
- The improved bot reaches its peak accuracy of 98% at intermediate values of alpha (0.08).

The tabular representation of the performance comparison between the baseline bot and the improved bot is shown below for different values of alpha:

Alpha	Baseline Accuracy	Improved Accuracy
0	22.00%	36.00%
0.01	84.00%	88.00%
0.02	70.00%	94.00%
0.03	88.00%	88.00%
0.04	76.00%	76.00%
0.05	52.00%	82.00%
0.06	88.00%	92.00%
0.07	76.00%	82.00%
0.08	96.00%	98.00%
0.09	74.00%	86.00%
0.1	80.00%	72.00%
0.11	78.00%	86.00%
0.12	58.00%	96.00%
0.13	90.00%	68.00%
0.14	78.00%	82.00%
0.15	60.00%	80.00%
0.16	78.00%	86.00%
0.17	100.00%	50.00%
0.18	64.00%	80.00%
0.19	78.00%	86.00%
0.2	82.00%	86.00%

4) PREVIOUSLY, THE SPACE RAT WAS ASSUMED TO BE STATIONARY. NOW ASSUME THAT IN EVERY TIMESTEP, AFTER THE BOT TAKES AN ACTION, THE SPACE RAT MOVES IN A RANDOM OPEN DIRECTION.

- **WORK OUT THE SPACE RAT KNOWLEDGE BASE UPDATES BASED ON THIS. WHAT CHANGES, WHAT PROBABILITIES DO YOU NEED TO CALCULATE? AGAIN, THERE IS A CORRECT ANSWER.**

In this case, we are assuming that in every timestep, after the bot takes an action, the space rat moves in a random open direction. To update the knowledge base when the space rat moves randomly after each timestep action, we have considered the new dynamics as a result of this movement.

Previously, the rat was stationary, and so the probability updates at each timestep depended solely on the bot's observations with respect to the sensor ping. Now, we have to consider the probability associated with the rat moving to adjacent open cells along with the bot's sensor input.

There are a few assumptions we have made in order to solve this:

- The space rat can only move to open cells and cannot move to a blocked cell.
- The space rat can move to any of the open cells neighbouring (adjacent to) its current position with equal probability.
- The open neighbouring positions that the rat can move to also include its current position assuming a situation where the rat remains stationary.

Probability Based Knowledge Update:

- Ping-Based Probability Update: Similar to the stationary case, if the bot receives a ping (hear_prob = True), it implies that the rat is nearby.

The ping probability (**P(ping | d(bot_pos, j)**) hence remains the same:

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

Where, d(bot_pos, j) is the Manhattan Distance between cells bot_pos and j
And α (alpha) depicts the sensitivity of the sensor.

- Movement Transition Probability:

$P_t(x, y)$ represents the probability that the rat is in cell (x, y) at time t. After moving randomly, the new probability $P_{t+1}(x', y')$for cell (x', y'):

$$P_{t+1}(x', y') = \sum_{(x,y) \in \mathcal{N}(x', y')} \frac{P_t(x, y)}{\text{degree}(x, y)}$$

Where:
• N (x′, y′): Set of all cells (x, y) that could transition to (x′, y′). These are the neighbouring open cells of (x′, y′).
• degree (x, y): Number of open neighbouring cells of (x, y). This ensures equal distribution of the probability from (x, y) across all possible moves.

- Sensor Update: After calculating the transition probabilities, the bot uses the rat detector and refines its probabilities based on the input (hearing or not hearing a ping).

    - If Ping is heard:

$$P_{t+1}(x', y' \mid \text{ping}) = \frac{P\big(\text{ping} \mid d(\text{bot\_pos}, x', y')\big) \cdot P_{t+1}(x', y')}{\sum_{(i,j)} P\big(\text{ping} \mid d(\text{bot\_pos}, i, j)\big) \cdot P_{t+1}(i, j)}$$

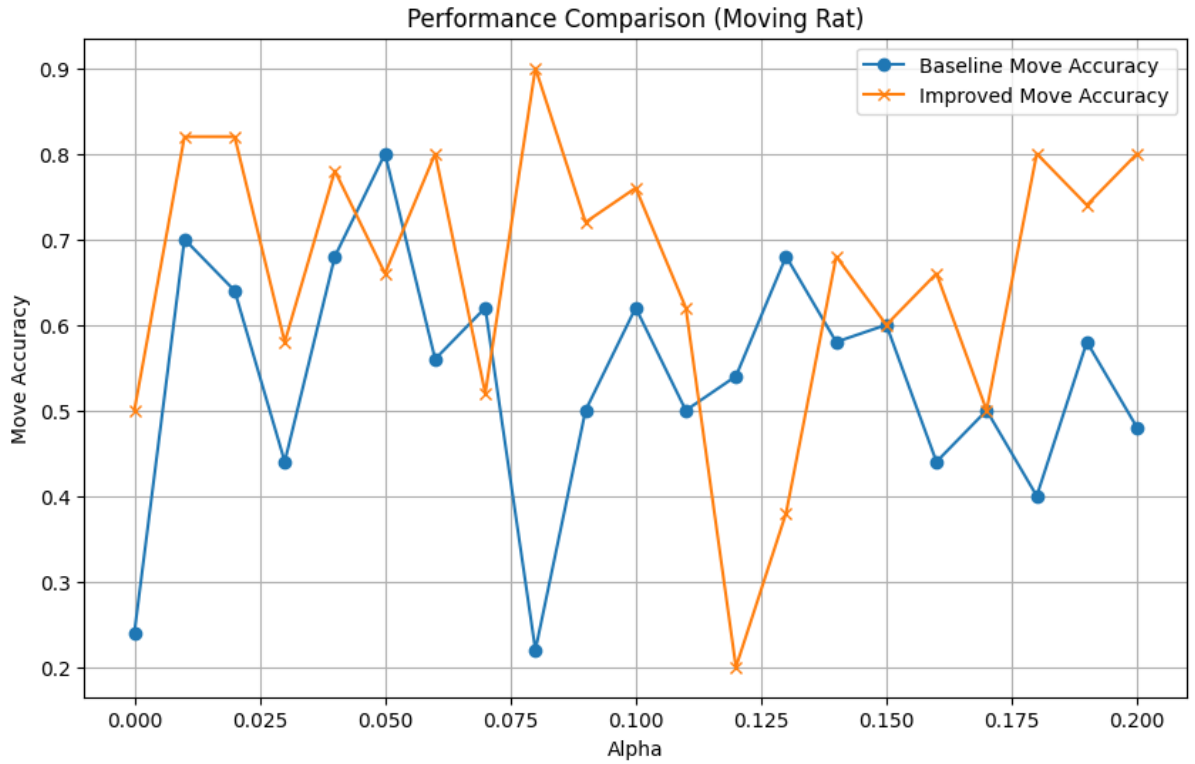
    - If Ping is not heard:

$$P_{t+1}(x', y' \mid \text{no ping}) = \frac{\Big(1 - P\big(\text{ping} \mid d(\text{bot\_pos}, x', y')\big)\Big) \cdot P_{t+1}(x', y')}{\sum_{(i,j)} \Big(1 - P\big(\text{ping} \mid d(\text{bot\_pos}, i, j)\big)\Big) \cdot P_{t+1}(i, j)}$$

- Iterative Updates:

    - We started with the probability distribution $P_t(x,y)$ from the timestep t.
    - We then applied movement transitions to find P(x', y') for time t+1.
    - We refine the probability at time t+1 using Bayes' Theorem based on the sensor's output. This is similar to the probability update that is done when the rat was stationary.

- **SIMULATE AND COMPARE BOTH BOTS IN THIS NEW ENVIRONMENT. HOW DOES THE PERFORMANCE CHANGE? AGAIN COMPARE ACROSS ALPHA > 0**.



The above graph evaluates the accuracy of the improved bot against the baseline bot in the moving space rat situation as a function of alpha. We have ran 50 simulations for each individual value of alpha. The alpha values range from 0 to 0.2 with a step size of 0.01.
We can derive the following information from it:

- Considering the general trend, the improved bot consistently outperforms the baseline bot across most values of alpha. This indicates that the improved bot performs better in this new environment and adapts better to a dynamic environment as well.
- Both bots show some fluctuations over few values of alpha but the improved bot still maintains a higher accuracy on average and adapts better to the increasing value of alpha.
- The improved bot reaches its peak accuracy of 90% at intermediate values of alpha (0.08) and the baseline bot reaches its peak accuracy of 80% at a lower alpha value of 0.05.
- Overall the improved bot exhibits more stability across different values of alpha as its accuracy fluctuates a little less than the baseline bot.

The tabular representation of the performance comparison between the baseline bot and the improved bot in the moving space rat situation is shown below for different values of alpha:

Alpha	Baseline Move Accuracy	Improved Move Accuracy
0	24.00%	50.00%
0.01	70.00%	82.00%
0.02	64.00%	82.00%
0.03	44.00%	58.00%
0.04	68.00%	78.00%
0.05	80.00%	66.00%
0.06	56.00%	80.00%
0.07	62.00%	52.00%
0.08	22.00%	90.00%
0.09	50.00%	72.00%
0.1	62.00%	76.00%
0.11	50.00%	62.00%
0.12	54.00%	20.00%
0.13	68.00%	38.00%
0.14	58.00%	68.00%
0.15	60.00%	60.00%
0.16	44.00%	66.00%
0.17	50.00%	50.00%
0.18	40.00%	80.00%
0.19	58.00%	74.00%
0.2	48.00%	80.00%

- **TRY TO IMPROVE ON YOUR BOT DESIGN TO BE EVEN MORE EFFECTIVE IN THIS MOVING SPACE RAT SITUATION. WHAT DID YOU NEED TO CHANGE, IF ANYTHING?**

There a few things we can improve to adapt to the moving space rat situation:

a. Target Quadrant Refinement: Currently, our version of the improved bot focuses on the target quadrant (quadrant with the highest total probability) and moves towards the weighted center of the quadrant. After reaching the weighted center, if it doesn't find the rat it, only then it recalculates and refines the target quadrant.

This approach doesn't account for the movement of the rat. By overly focusing on a single quadrant, the bot might waste steps when the rat is not there. This approach doesn't consider the possibility of stale/outdated information due to the movement of the rat.

Additionally, we only refine the quadrant further if the rat is not found at the weighted center. This doesn't allow the bot to adapt to the rat's movement. If the rat has moved significantly by the time the bot reaches the weighted center, refining the wrong quadrant would be inefficient.

The rat's movement can cause probabilities to become outdated while the bot is moving to the center.

A change we can introduce to improve the bot's performance in the moving rat situation could be to proactively refine the quadrants at multiple time steps. Moreover, instead of

selecting the target quadrant with the highest total probability, we can use a weighted selection based on all quadrant probabilities.

b. Quadrant Confidence: Currently, our version of the improved bot selects the target quadrant with the highest probability but doesn't adjust the probabilities based on confidence levels of quadrants.

A possible improvement could be to assign confidence values to quadrants after some movement steps have been taken. We can reduce confidence in a quadrant if it has been visited a few times without the rat being found there and increase confidence based on the updated probabilities.