RUTGERS
UNIVERSITY | NEW BRUNSWICK

Utsav Patel (RUID - 211009880, NetID - upp10)

Manan Vakta (RUID - 206003851, NetID - mv651)

Sneh Desai (RUID - 211003857, NetID - sd1324)

Jal Shah (RUID - 203003781, NetID - js2985)

Master of Science in Computer Science

Probability in GridWorld

November 2021

The School of Graduate Studies

Contents

List of Figures

Chapter 1

Question 1, 2 and 3

1.1 Question 1

Prior to any interaction with the environment, we have no information about any of the cells in our grid world. By the given rules of creating the grid world, our target can be in any of these cells since the agent assumes all of them are unblocked. Thus -

$$
\begin{aligned}
&P(\textit{Target in any cell (x,y) prior to interaction with environment}) \\
&= P(\textit{Target in (x,y)} \cap \textit{(x,y) is blocked}) + P(\textit{Target in (x,y)} \cap \textit{(x,y) is unblocked}) \\
&\textit{Since target cannot be in a blocked cell; and by applying Bayes Rule, we get -} \\
&= 0 + P(\textit{Target in (x,y)} \mid \textit{(x,y) is unblocked}) * P(\textit{(x,y) is unblocked}) \\
&= \frac{1}{0.7 * \textit{dimension*dimension}} * 0.7 \\
&= \frac{1}{\textit{dimension*dimension}}
\end{aligned}
$$

Where dimension = The number of rows or columns in our square grid.

Note - We have used Total Cost in our report to denote the same thing as Total Actions which is the sum of Total Movements and Total Examinations

1.2 Question 2

Let us go through the circumstances in order -

- Probability cell (i,j) contains the target at time t + 1, as we attempt to enter cell (x,y) and find it is blocked -

$$P_{i,j}(t+1) = P((i,j) \text{ contains target at } t \mid (x,y) \text{ is blocked at } t)$$

This can be re-written as :

$$P_{i,j}(t+1) = P((i,j) \text{ contains target at } t \mid (x,y) \text{ does not contain target at } t)$$

By Bayes Rule we get,

$$P_{i,j}(t+1) = \frac{P_{i,j}(t) * P((x,y) \text{ does not contain target at } t \mid (i,j) \text{ contains target at } t)}{P((x,y) \text{ does not contain the target at } t)}$$

Now here we know that :

$$P((x,y) \text{ does not contain target at } t \mid (i,j) \text{ contains the target at } t) = 1$$

Therefore we get that:

$$P_{i,j}(t+1) = \frac{P_{i,j}(t) * 1}{1 - P_{x,y}(t))}$$

If (x,y) = (i,j), we know a blocked cell cannot contain the target, thus :

$$P_{i,j}(t+1) = 0$$

- Probability cell (i,j) contains the target at time t + 1, as we enter cell (x,y) and find it is unblocked and we learn its terrain type -

We get the following equation:

$$P_{i,j}(t+1) = P((i,j) \text{ contains the target at time } t \mid (x,y) \text{ is unblocked})$$

As discussed by the professor, for the sake of simplicity we can assume that the probability of containing target at time t is independent of (x,y) being unblocked.

Since we are not examining any new cells, the probability of cell (i,j) at time t+1 remains the same as we only learn the terrain type of the cell but we do not examine the probability of target being found in that cell.

Therefore we get that:

$$P_{i,j}(t+1) = P_{i,j}(t)$$

- Probability cell (i,j) contains the target at time t + 1 as we enter cell (x,y) and find it has flat terrain type and we fail to find the target -

  We can represent this situation as -

  $P_{i,j}(t+1) = P(\textit{(i,j) contains the target at time t} \mid \textit{fail to find target in flat (x,y) at time t})$

  By Bayes Rule,

  $$P_{i,j}(t+1) = \frac{P_{i,j}(t) * P(\textit{fail to find target in (x,y) at time t} \mid \textit{(i,j) contains target at t})}{P(\textit{fail to find target in (x,y) at time t})}$$

  There are 2 cases we have to consider,

  Case 1: When (i,j) $\neq$ (x,y)

  $$= P(\textit{fails to find target in (x,y) at time t} \mid \textit{(i,j) containing target at t}) = 1$$

  Thus,

  $$P_{i,j}(t+1) = \frac{P_{i,j}(t) * 1}{P(\textit{fails to find target in (x,y) at time t})}$$

Here, let us expand the denominator

$$= P(\textit{Fails to find Target in cell (x,y) at time t})$$

$$= \sum_{(i,j)} P(\textit{Fails to find Target in cell (x,y) at time t} \cap \textit{(i,j) contains target at time t})$$

$$= \sum_{(i,j)} P(\textit{Fails to find Target in cell (x,y) at time t} \mid \textit{(i,j) contains target at time t}) * P_{i,j}(t)$$

$$= \sum_{(i,j)\neq(x,y)} P_{i,j}(t) + P_{x,y}(t) * k$$

$$= 1 - P_{x,y}(t) + P_{x,y}(t) * k$$

Here we use k to represent the False-Negative rate of the terrain of (i,j). Thus for any terrain when (i,j) $\neq$ (x,y) the formula is -

$$P_{i,j}(t+1) = \frac{P_{i,j}(t)}{1 - P_{x,y}(t) * (1-k)}$$

Case 2: When (i,j) = (x,y), the numerator simplifies like this -

$$= P(\textit{fails to find target in (x,y) at time t} \mid \textit{(i,j) containing target at t}) = k$$

And the denominator simplifies as shown in Case 1. Thus for any terrain when (i,j) = (x,y) the formula is -

$$P_{i,j}(t+1) = \frac{P_{i,j}(t) * k}{1 - P_{x,y}(t) * (1-k)}$$

So when the terrain is Flat ($k = 0.2$), the final formulae are as follows : When (i,j) $\neq$ (x,y):

$$P_{i,j}(t+1) = \frac{P_{i,j}(t)}{1 - P_{x,y}(t) * (0.8)}$$

And When (i,j) = (x,y):

$$P_{i,j}(t+1) = \frac{P_{i,j}(t) * 0.2}{1 - P_{x,y}(t) * (0.8)}$$

- Using the above generalized formulae, we can say that when the terrain is Hilly ($k = 0.5$) :

When (i,j) $\neq$ (x,y):

$$P_{i,j}(t+1) = \frac{P_{i,j}(t)}{1 - P_{x,y}(t) * (0.5)}$$

And When (i,j) = (x,y):

$$P_{i,j}(t+1) = \frac{P_{i,j}(t) * 0.5}{1 - P_{x,y}(t) * (0.5)}$$

- Using the above generalized formulae, we can say that when the terrain is Forest ($k = 0.8$):

  When (i,j) $\neq$ (x,y):

  $$P_{i,j}(t+1) = \frac{P_{i,j}(t)}{1 - P_{x,y}(t) * (0.2)}$$

  And When (i,j) = (x,y):

  $$P_{i,j}(t+1) = \frac{P_{i,j}(t) * 0.8}{1 - P_{x,y}(t) * (0.2)}$$

- Probability cell (i,j) contains the target at time t + 1, as we find the target in cell (x,y) - Since we have already found the target at cell (x,y), the probability of any other cell containing the target immediately goes to 0. Thus,

  $$P_{i,j}(t+1) = 0$$

  And,

  $$P_{x,y}(t+1) = 1$$

1.3 Question 3

Given that we visit a cell (x,y) and learn its terrain type at time t and we also know the probability of cell (x,y) containing the target, we can calculate the probability of *finding* the target in cell (x,y) as follows -

$= P($***Finding*** *the target in cell (x,y) at t* $\mid$ *Terrain type of (x,y))*

$= P($*(x,y) contains the target at time t)* $*$ *(1 - False Negative Rate of the Terrain Type)*

$= P_{x,y}(t) * (1 - k)$

Here k = False negative rate of our Terrain Type

- If x,y is Hilly we get:

$$P(\textit{Finding the target in (x,y) at time t }) = P_{x,y}(t) * 0.5$$

- If x,y is Flat we get:

$$P(\textit{Finding the target in (x,y) at time t }) = P_{x,y}(t) * 0.8$$

- If x,y is Forest we get:

$$P(\textit{Finding the target in (x,y) at time t }) = P_{x,y}(t) * 0.2$$

- If x,y has never been visited:

  In this case (x,y) has an equal chance to be of any terrain type

$$= P(\textbf{\textit{Finding}} \textit{ the target in cell (x,y) at t} \mid \textit{(x,y) is any Terrain Type})$$
$$= P(\textit{(x,y) contains the target at time t}) * (\frac{(1 - \textit{FN Flat})}{3} + \frac{(1 - \textit{FN Hilly})}{3} + \frac{(1 - \textit{FN Forest})}{3})$$
$$= P_{x,y}(t) * (0.5)$$

Chapter 2

Question 4

2.1 Implementation for Agent 6 and 7

2.1.1 src/Agent6.py and src/Agent.py

- Agent 6 inherits the Agent class.
- Agent.py contains the following functions :
  - **init**: In this function we initialize a variety of the Agents properties like the agents copy of the maze, the current estimated goal of the agent, the path taken by agent, current position of the agent, number of examinations performed by the agent. We store the probability values of each cell of the maze in a numpy array so we can iterate and change these values as efficiently as possible.
  - **pre_planning**: This function calls another function from helper.py that sets the current estimated goal of the agent.
  - **planning:** This function calls another function in helper.py that plans a path to the current estimated goal.
  - **reset**: This function resets all the properties of the agent to their default values.
- Agent6.py contains the following functions and inherits all the properties from Agent.py :
  - **execution**: This function calls another function in helper.py where the agent moves through the path returned by the planning function.

  - **examine**: This function calls another function from helper.py that lets us examine the cell we are currently in to search for the target.

- The above given **pre_planning**, **planning**, **execution** and **examine** are the 4 main phases that our agents **6, 7 and 8** follow.

2.1.2 helper/helpers.py

- helper.py contains all the helper functions for our agents. Let us go through the important ones -

- **astar_search -**

  - This function takes as input the agent's copy of the maze, the current position of the agent and the current estimated goal of the agent.

  - It is called by the agent during the **planning** phase.

  - It returns a path from the current position to the current estimated goal based on the agents copy of the maze.

- **forward_execution -**

  - This function takes as input the agents copy of the maze, the actual maze, the current position of agent, a numpy array that contains the false negative rates of all cells in the maze and the goal position of the agent.

  - It is called by the agent during the **execution** phase.

  - This function attempts to move the agent through the path returned by the astar_search function.

  - We break out of this function if the agent successfully reaches the estimated goal position or the agent encounters a block in the path returned by astar_search.

  - This function returns the path followed by the agent and position of the agent after execution. We can use the last index of the path followed by the agent to get the current position of the agent.

- **compute_current_estimated_goal -**

  - This is one of the main functions that we use in our code. It takes as input the agents copy of the maze, the current position of the agent, a numpy array that contains the probabilities that a cell contains the target for each cell of the maze, a numpy array with the false negative rates of all cells in the maze and the agent's number.
  - It is called by the agent during the **pre-planning** phase.
  - As the name suggest, this function attempts to estimate the current best goal based on the conditions we have been given for each agent.
  - For agent 6, our goal is the cell with the highest probability of *containing* the target. We use *least distance* from current position to break ties between multiple cells with the same probability. And then pick a cell randomly to break ties between equidistant cells.
  - This function returns the current estimated goal to our agent that the agent will then use to plan and execute a path.

- **examine_and_propagate_probability -**
  - This function is called if we discover a block in our path during execution or if our current estimated goal cannot be reached because it is in a part of the maze that is unreachable. In the case of Agent 9 it can also be called whenever we want to examine a cell in our path.
  - It is called by the agent after the **planning** phase, after the **execution** phase and during the **examination** phase.
  - Since this function is called when we encounter a block in our path or we discover that the current estimated goal is in an unreachable part of the grid and essentially "blocked"; we first set the probability that the blocked cell contains the target to 0. And then increase the probabilities of all other cells using the formulae discussed in section 1.2
  - This function returns "False" to our main agent6 function that tells the agent that a target has not yet been found.

- **check_and_propagate_probability -**

  - This function is called if the agent reaches the current estimated goal and wants to examine the position it is currently in.
  - It is called by the agents during the **examination** phase. It is also called by Agent 9 during the **execution** phase.
  - We first check if the cell we are in contains the target. If true then we have a chance to find the target based on the terrain of the examined cell.
  - If we find the target we are done and the Agent stops searching for the target in the current maze.
  - If we do not find the target then we change the probability of the current cell containing the target and then propagate that change to the rest of the grid based on the equations given in Section 1.2
  - This function returns True or False based on if the Target was found or not.

2.1.3 test/agent6.py

- This is the main file that generates the data for our comparisons. Let us go through the only function in this file -
- **find_the_target -**
  - This function takes as input the number of the agent that we want to generate results for.
  - The function first creates a maze in accordance with the rules set by the assignment. We then place a target randomly in an unblocked cell and make sure that the target is reachable from the start position. If it is not we discard the maze and repeat this process.
  - Now that we have a solvable maze to put our agent in, we reset all attributes of the agent and run the agents **pre_planning** function.
  - Next we pass the current estimated goal calculated by the **pre_planning** function to the **planning** function so that we can plan a path to the current estimated goal.
  - We then pass this planned path to the **execution** function where we move the agent

along the path and discover blocks and terrain types in its way.

- If the agent gets blocked or reaches the current estimated goal, we call the **examine** function to compute and propagate the probability as required.
- The function will keep running the **pre_planning**, **planning**, **execution**, **examine** functions till we find the target.
- Once the target is found we return the number of examinations and movements made by the agent to calculate the average cost of finding the target for our agent.

2.1.4 Agent 7

As we can see from the above description of Agent 6's implementation, most of the functions that are used by agent 6 are generic and can be used by the other agents as well. In fact, Agent 7 follows the exact same four phase process of **pre_planning**, **planning**, **execution**, and **examination**. The only function that changes is -

- **compute_current_estimated_goal -**
  - As we can see from our implementation of this function; it takes as input the agents copy of the maze, the current position of the agent and the **agent's number**.
  - If we pass '7' as the agent's number to this function, the current best estimated goal will now be the cell with the highest probability of *finding* the target. The calculations for this can be seen in section 1.3.
  - Thus this function will prioritize all cells with the highest probability of *finding* the target. Then from all cells with equal highest probabilities, it will select the cells that are closest to current position of the agent. To break ties between equidistant cells we select a cell randomly.
  - The function then returns this cell as the current_estimated_goal for the agent.

2.2 Comparison of Agent 6 and 7

Please refer to Chapter 5 for an in-depth comparison of agents 6,7 and 8.

Chapter 3

Question 5, 6 and 7

3.1 Question 5

As stated in the requirements for Agent 8, we wanted to make an agent that has the exact same information as Agents 6, 7 and can move and examine in the exact same way. The only difference in our approach between agent 6,7 and 8 is where Agent 8 decides to go using the information that is available to all our agents.

We noticed that for Agent 6 and 7, a big part of the total average cost to find the target was the amount of movement the Agent had to do in a 100x100 grid. The movements would be on average around **5 times** more than the average number of examinations performed by the agent.

Thus we decided to make Agent 8 have a better average cost by heavily de-incentivizing movement at the cost of a few more examinations on average.

We achieved this reduction in total movements by tweaking the formula that our agent was using to compute the current estimated goal. For agent 7 (which performs better than agent 6 as shown in our comparison) we were using the following utility function-

$$U_{i,j}(t) = \text{Probability of } \textit{finding} \text{ target at (i,j)}$$

For agent 8 we tweaked our formula like this -

$$U_{i,j}(t) = \frac{\text{Probability of } \textbf{finding} \text{ target at (i,j)}}{\textbf{Distance} \text{ from Current Position}}$$

This simple tweak made it so that our agent heavily incentivized a low distance from current position to examine next.

To implement this, we required distance matrix in the utility function which we were not using in agents 6 and 7.

3.2 Question 6

3.2.1 Implementation of Agent 8

As stated in the implementation for Agent 7, the main 4 phases of our Agent 8 remain the same - **pre_planning**, **planning**, **execution**, and **examination**.

The only change we make for Agent 8 is during the **pre_planning** phase. Instead of choosing the current estimated goal solely based on probability, we choose current estimated goal in a way that heavily incentivizes a low distance from current position.

3.2.2 Comparison of Agent 8 with Agents 6 and 7

Please refer to chapter 5 for an in-depth comparison of agents 6,7 and 8.

3.3 Question 7

Agent 8 can definitely be improved further. Instead of prioritizing the cell with the highest probability, we could target a region of the grid that has the highest probability of containing the target and then examine cells in that region. Currently, agent 8 is being greedy and choosing a cell with the highest probability for the current timestamp. Instead of that, agent 8 can choose a cell which would give it more rewards (less actions overall) in future.

Chapter 4

Bonus Question

4.1 Implementation of Agent 9

As seen in the implementation of Agent 6, 7 and 8, Agent9 inherits the Agent.py with the structure defined in section 2.1

Agent 9 modifies the **init** method slightly; it initializes one more property of the Agent - **is_target_in_neighbors** to False. This added property is used by Agent 9 during the target finding process when it senses for the presence of the target in its valid neighboring cells. In addition to the initialization phase, Agent 9 passes through 4 more phases in an attempt to find the target. These 4 phases are:

- **Sense -** In this phase, the Agent checks if one of the **valid neighbors** (neighboring unblocked cells) contains the target or not and updates probabilities accordingly.
- **Pre-planning -** This is slightly different from the Pre-planning followed by Agents 6, 7, 8. The current estimated goal is calculated only when the property **is_target_in_neighbors** is returned as False by the sensing phase. When this property is false, Agent 9's current estimated goal calculation is the same as Agent 8's. When **is_target_in_neighbors** is True, no goal cell has to be estimated as the target is already in the valid neighboring cells. Subsequently, the Agent moves to the Execution phase.
- **Planning -** In this phase, we use A* search to find a path from the current position of the agent to the current estimated goal which was calculated in the Pre-Planning phase. This phase is only called when **is_target_in_neighbors** is set to True during the **Sense**

phase.

- **Execution -** This phase is now a combination of the execution and examination phases of previous agents. Here in this phase, the Agent uses repeated forward A* to traverse the path determined during the planning phase(if necessary) and examines a cell whenever certain conditions are met.

The above four phases are implemented using the below described functions:

- **Sense():** This function is used to execute the **Sense** phase described as above. Initially, the property of Agent **is_target_in_neighbors** is set to False. We then check if the target resides within the valid neighboring cells of the Agent. This is done with the help of the co-ordinates of the actual target position passed to this function. If the target is in one of the valid neighboring cells, then **is_target_in_neighbors** is set to True. Now since **is_target_in_neighbors** is True, the probability is updated for all the cells of the maze. There are 2 possible cases -

  – **is_target_in_neighbors** is set to True - The probability of all the cells except the Agent's valid neighbors, is set to 0 and the probability of the valid neighbors, is increased maintaining the scaling factor using the formula below:

  For sake of simplicity we define set of neighboring cells = N

  When we sense the target is in neighbors:

  We know P((i,j) contains target |target $\in$ N) = 0 where (i,j) $\notin$ N.

  Let's calculate for the case where (i,j) $\in$ N

  $P_{i,j}$(t+1) = P((i,j) contains target at t|target $\in$ N)

  $$= \frac{\text{P(target} \in \text{N)|(i,j) contains target)*}P_{i,j}(\text{t})}{\text{P(target} \in \text{N)}}$$

  $$= \frac{1*P_{i,j}(\text{t})}{\text{P(target} \in \text{N)}} \qquad \text{-(i)}$$

  As we know that: P(target $\in$ N)+P(target $\notin$ N)=1

  P(target $\in$ N)

  $= 1 - \text{P(target} \notin \text{N)}$

  $= 1 - \text{P(target is somewhere except N)}$

  $= 1 - \sum_{(x,y)\notin N} P_{x,y}(t)$

  Putting the above equation in (i) we get:

$$P_{i,j}(t+1) = \frac{P_{i,j}(t)}{1 - \sum_{(x,y)\notin N} P_{x,y}(\text{t})}$$

$$= \frac{P_{i,j}(t)}{\sum_{(x,y)\in N} P_{x,y}(\text{t})}$$

– **is_target_in_neighbors** is False - The probability for all the Agent's valid neighboring cells, is set to 0 and the probability of all other cells is increased maintaining the scaling factor using the formula below- When we sense that the target is not in valid neighbors:

We know P(i,j contains target |target $\notin$ N) = 0, where (i,j) $\in$ N.

Let's calculate for the case where (i,j) $\notin$ N

$P_{i,j}$(t+1) = P((i,j) contains target |target $\notin$ N)

$$= \frac{\text{P(target} \notin \text{N |(i,j) contains target)*}P_{i,j}(\text{t})}{\text{P(target} \notin \text{N)}}$$

$$= \frac{1*P_{i,j}(\text{t})}{\text{P(target} \notin \text{N)}} \qquad \text{-(i)}$$

P(target $\notin$ N)

$= \sum_{(x,y)}$ (P(target $\notin$ N) $\cap$ ((x,y) contains target))

$= \sum_{(x,y)}$ P(target $\notin$ N |x,y contains target)*$P_{x,y}$(t)

$= \sum_{(x,y)\notin N} P_{x,y}(\text{t})$

$= 1 - \sum_{(x,y)\in N} P_{x,y}(\text{t})$

Putting the above equation in (i) we get:

$$\text{P}_{i,j}(\text{t+1}) = \frac{P_{i,j}(\text{t})}{1 - [\sum_{(x,y)\in N} P_{x,y}(\text{t})\ ]}$$

Thus, after the above-mentioned process, we have successfully formed the belief state at time 't' for the Agent. At time 't+1', the target can move to any of its 4 cardinal neighbors. Now the **prediction state** for time 't+1' is also calculated at time 't' itself. This prediction state is calculated by distributing the probability of all the cells containing the target at time 't' to their 4 neighbors equally. This is done because the target is equally likely to move to any of the 4 cells neighboring the current cell at time 't+1'. Thus, the Agent has successfully formulated a prediction state for time 't+1' at time 't' which it will use for the calculation of the current estimated goal. An interesting thing to notice is that the **prediction state** the Agent has calculated at time 't' for time 't+1', actually

becomes the belief state for time 't+1' as we have covered every possibility of the target cell's movement at time 't+1'.

- **Pre_Planning():** If **is_target_in_neighbors** is False, then the Agent needs to move to another cell which has a high likelihood of containing the target. This estimation of the goal cell is done in the same way as Agent8 (explained in section 3.2.1). If **is_target_in_neighbors** is True at time 't', then the agent doesn't need to move as the target cell is already in one of its valid neighboring cells. Thus the agent directly moves ahead to execution().

- **Planning():** If **is_target_in_neighbors** is False, a path is planned to the current estimated goal using A*. If **is_target_in_neighbors** is True, the Agent does not need to plan a path as the target is already in one of the valid neighboring cells of Agent 9.

  Note : Before the execution phase of Agent 9, the target is allowed to move randomly to one of its neighboring cells in the 4 cardinal directions. This is implemented using the move_target() function which is one of the helper functions for Agent 9.

- **Execution():** Initially in the execution phase, we set **target_found** to false. Then we determine the terrain type of the cell the Agent is currently in and set its FALSE NEGATIVE rate accordingly by calling a function **update_status()** in the helpers/helper.py as used in Agent 6, 7 and 8. Now comes the main part of Agent 9's implementation. If **is_target_in_neighbors** is set to True at time 't' then as stated before, the Agent doesn't need to estimate a goal cell to reach to at time 't+1'. This is because the target was allowed to move before this phase and it is more likely that the target has now relocated itself to the cell the Agent is currently in. Because of this, the Agent examines the cell it is currently in, and the probability of the maze is updated accordingly using the function **examine_and_propagate_probability()** described in section 2.1.2. If during this examination, we find that the target is found successfully, then we set target_found to True and return this to the main calling function **find_moving_target()** which drives this whole start to end process of finding the target. If **is_target_in_neighbors** was set to false at time 't' then the agent moves 1 step along the planned path. This happens at time 't+1' and so both the target and the Agent move at the same time. When we reach the goal cell, If it is blocked then we update is.blocked property to true and also change the probability of all the cells containing the target using the function **exam-**

ine_and_propagate_probability(). If it is not blocked, then the Agent has successfully moved to the current estimated goal cell and we repeat the 4 phase process of the agent again.

Note - We use the update_status() function during the execution phase to update the terrain type and set the FALSE NEGATIVE RATE of each cell that we have visited.

4.2 Comparison of Agent 9 with Agent 6, 7 and 8

Please refer to section 5.2 for an in-depth comparison.

Chapter 5

Comparison

5.1 Comparison of Agents 6, 7 and 8

Let us take a look at the data that we obtained for running 80 iterations for target in **each** terrain type for Agent 6, 7 and 8 in a 100x100 grid world -

Note - All values given below have been averaged out to show movements, examinations and actions per iteration.

Target in Terrain	*Movements*	*Examinations*	*Actions*
Flat	70154	8734	78888
Hilly	125486	14548	140034
Forest	142269	14606	156875
Any	112636.33	12629.33	125265.667

Table 5.1: Agent 6

Target in Terrain	*Movements*	*Examinations*	*Actions*
Flat	37441	7707	45148
Hilly	52166	9959	62125
Forest	137734	18381	156115
Any	75780.33	12015.66	87796

Table 5.2: Agent 7

Target in Terrain	*Movements*	*Examinations*	*Actions*
Flat	9166	7695	16861
Hilly	15925	13035	28960
Forest	31699	25509	57208
Any	18930	15413	34343

Table 5.3: Agent 8

We also have a table containing the ratios of movements and examinations for each agent -

Target in Terrain	*Agent*6	*Agent*7	*Agent*8
Flat	2.9996	2.43519	1.16607
Hilly	4.6848	2.87546	1.18357
Forest	6.48652	5.35368	1.20895

Table 5.4: Ratios of movements/examinations All Agents

We plotted box-plots and histograms for each agent to help us visually see the results we have obtained. We decided to split up the graphs according to the terrain of the target since the graphs are more readable this way -

5.1.1 Graphs for Target in Flat Terrain

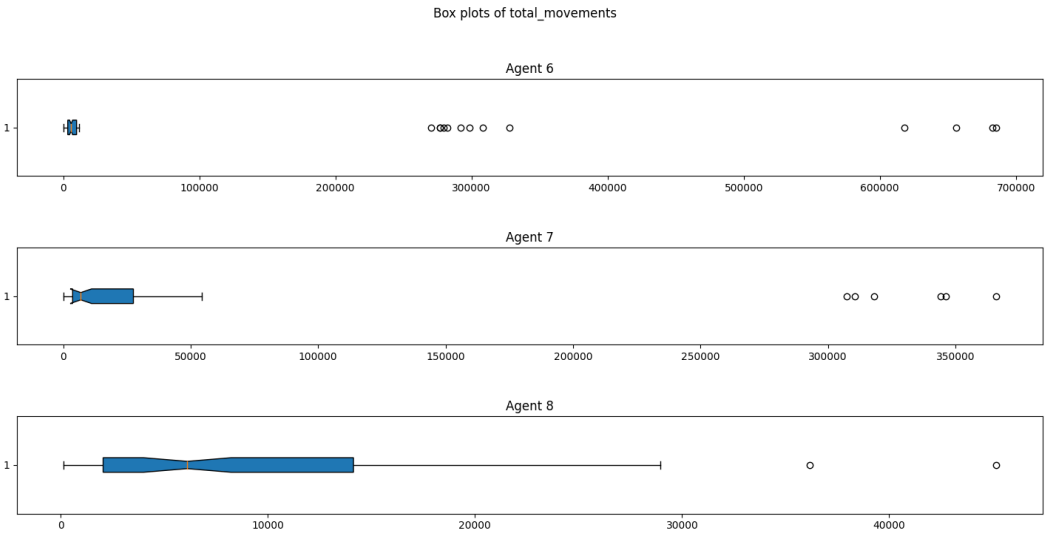

Figure 5.1: Agent 6,7 and 8 Total Movements Box-plot

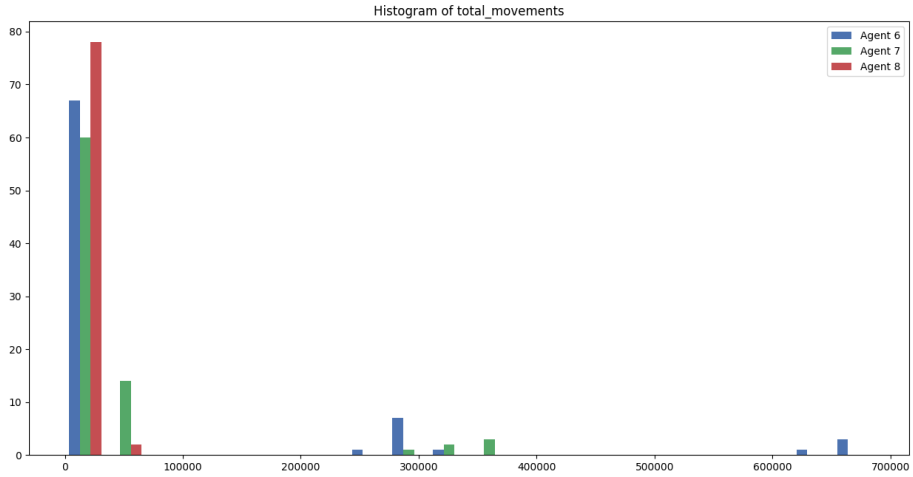

Figure 5.2: Agent 6,7 and 8 Total Movements Histogram

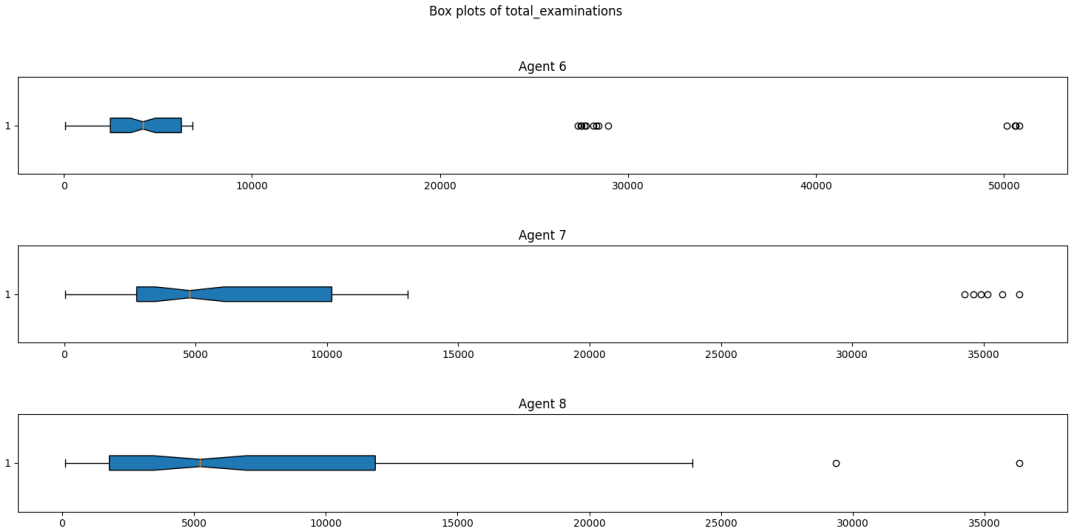

Figure 5.3: Agent 6,7 and 8 Total Examinations Box-plot

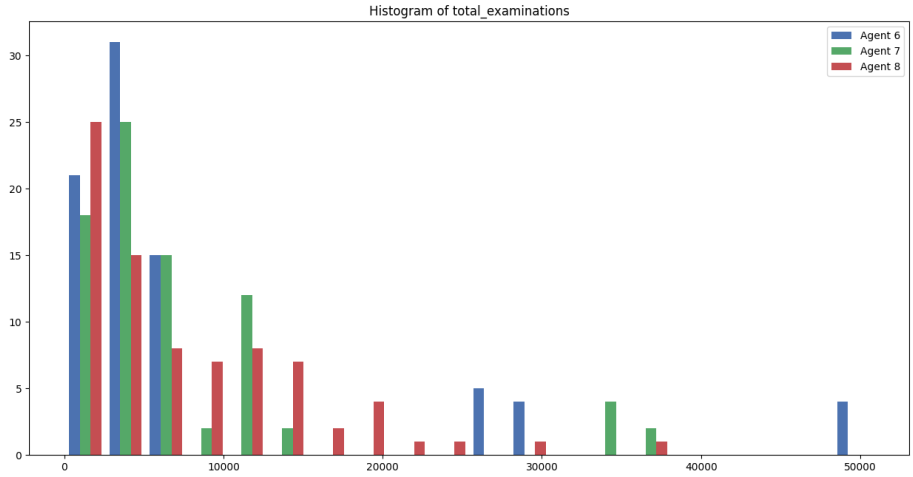

Figure 5.4: Agent 6,7 and 8 Total Examinations Histogram

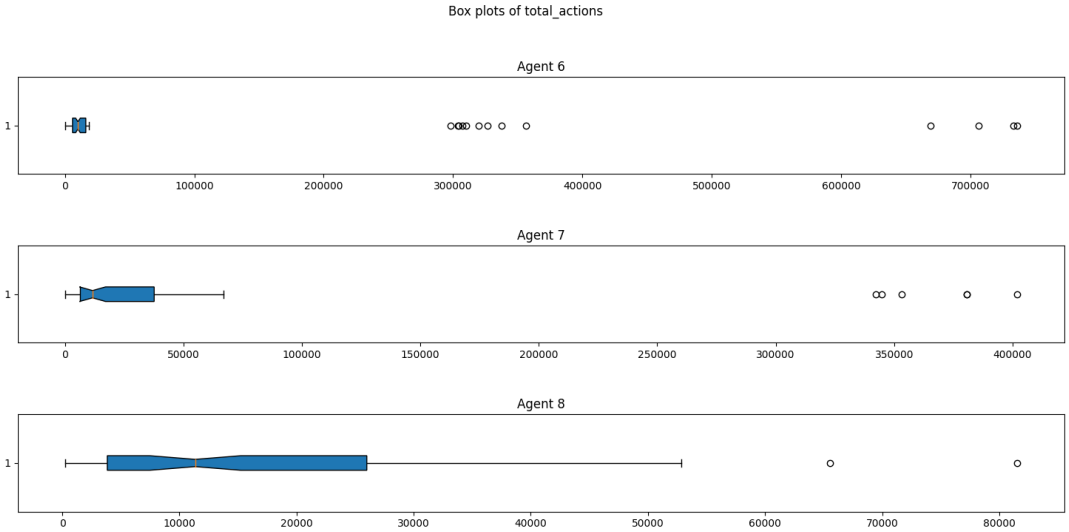

Figure 5.5: Agent 6,7 and 8 Total Cost Box-plot

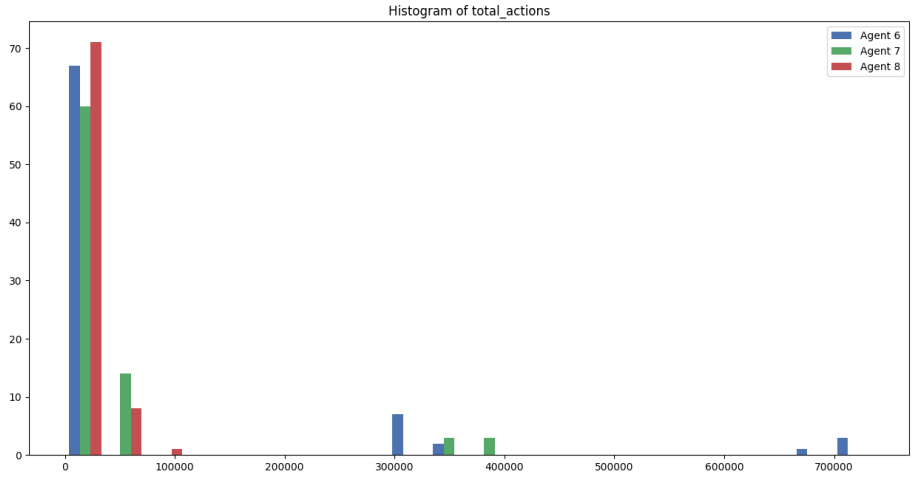

Figure 5.6: Agent 6,7 and 8 Total Cost Histogram

5.1.2 Graphs for Target in Hilly Terrain

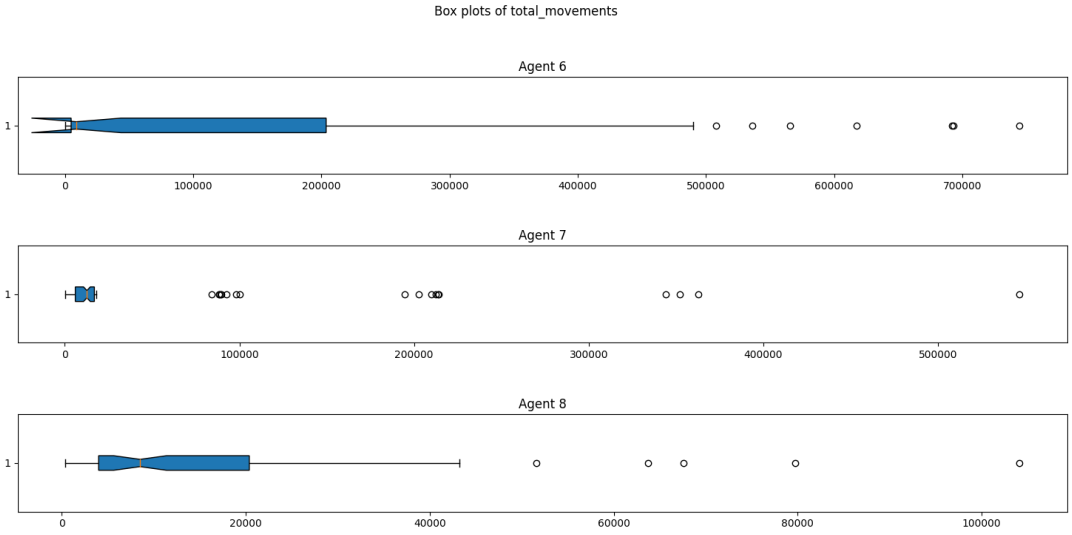

Figure 5.7: Agent 6,7 and 8 Total Movements Box-plot

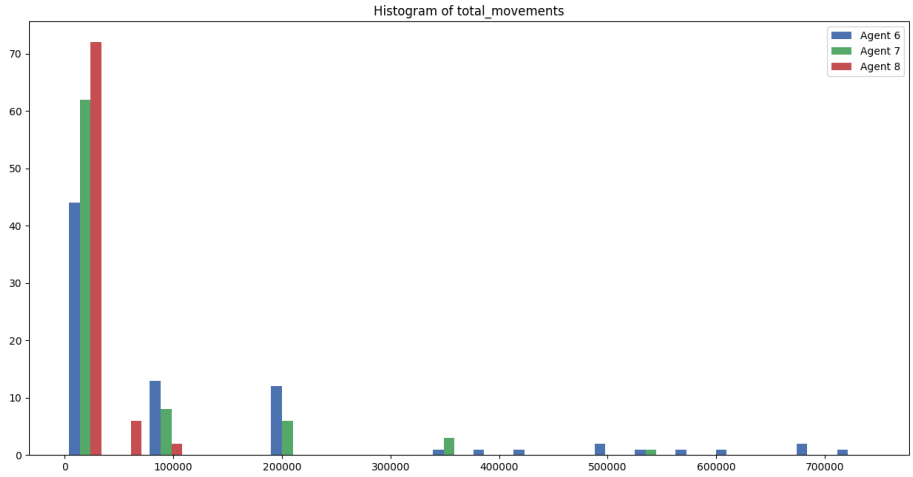

Figure 5.8: Agent 6,7 and 8 Total Movements Histogram

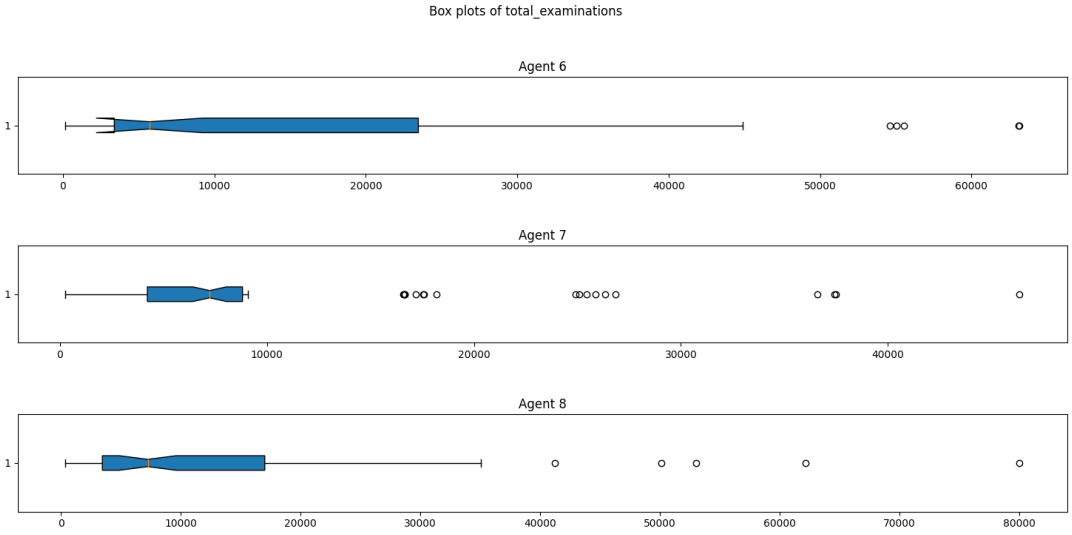

Figure 5.9: Agent 6,7 and 8 Total Examinations Box-plot

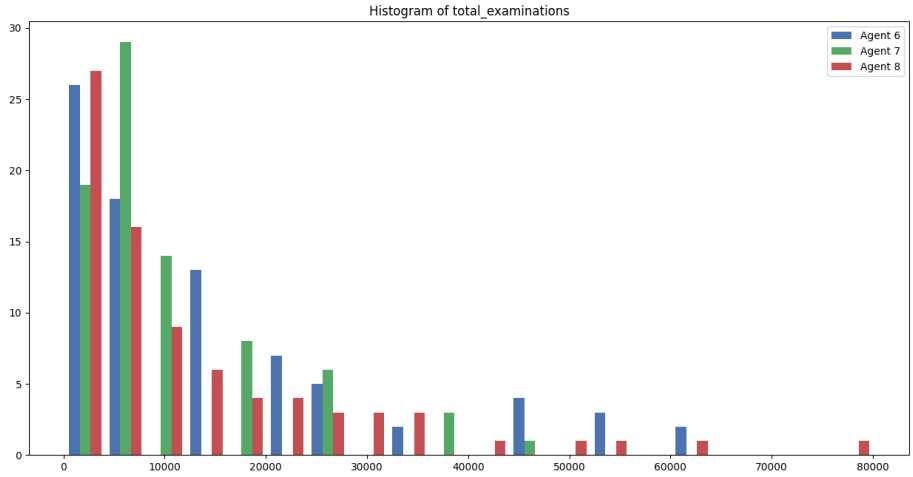

Figure 5.10: Agent 6,7 and 8 Total Examinations Histogram

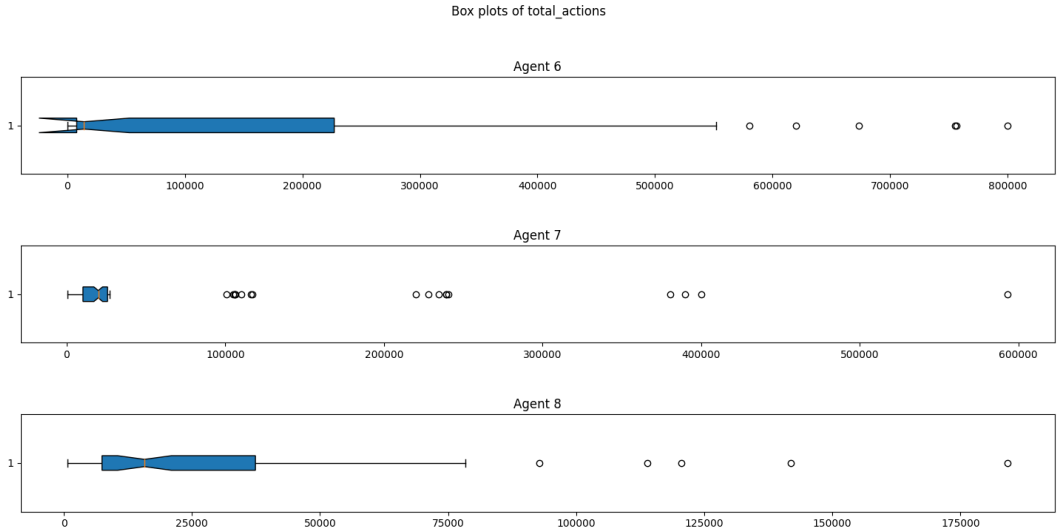

Figure 5.11: Agent 6,7 and 8 Total Cost Box-plot

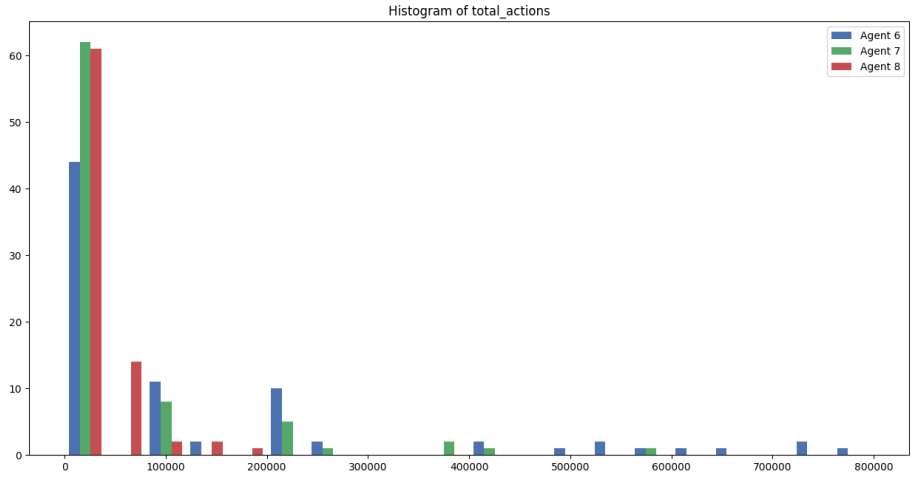

Figure 5.12: Agent 6,7 and 8 Total Cost Histogram

5.1.3 Graphs for Target in Forest Terrain

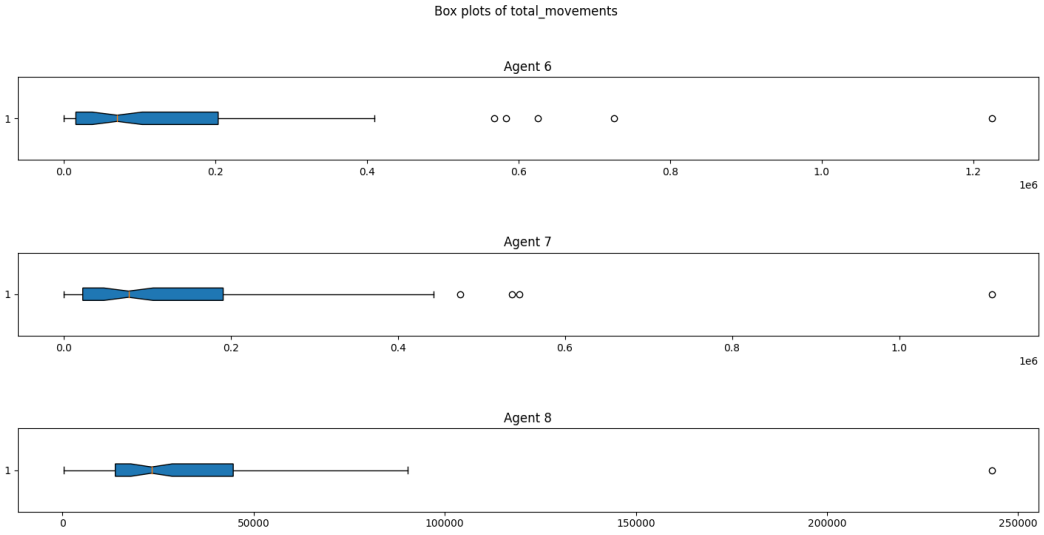

Figure 5.13: Agent 6,7 and 8 Total Movements Box-plot

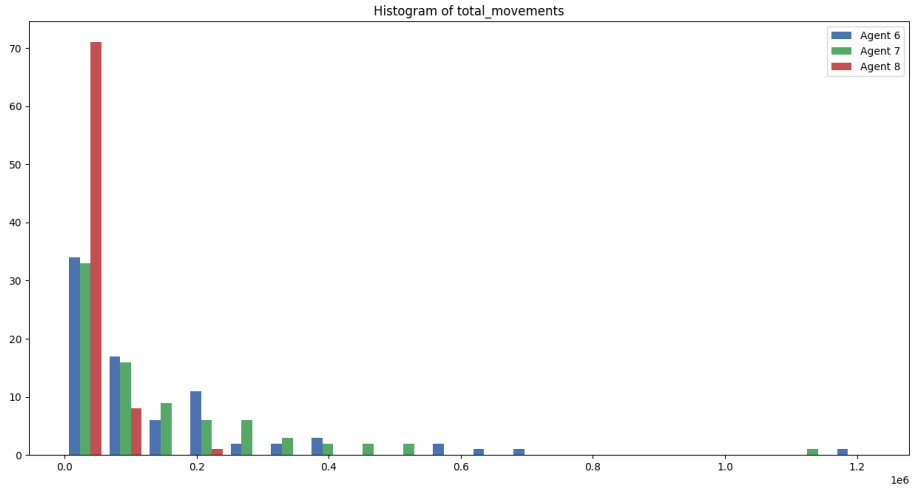

Figure 5.14: Agent 6,7 and 8 Total Movements Histogram

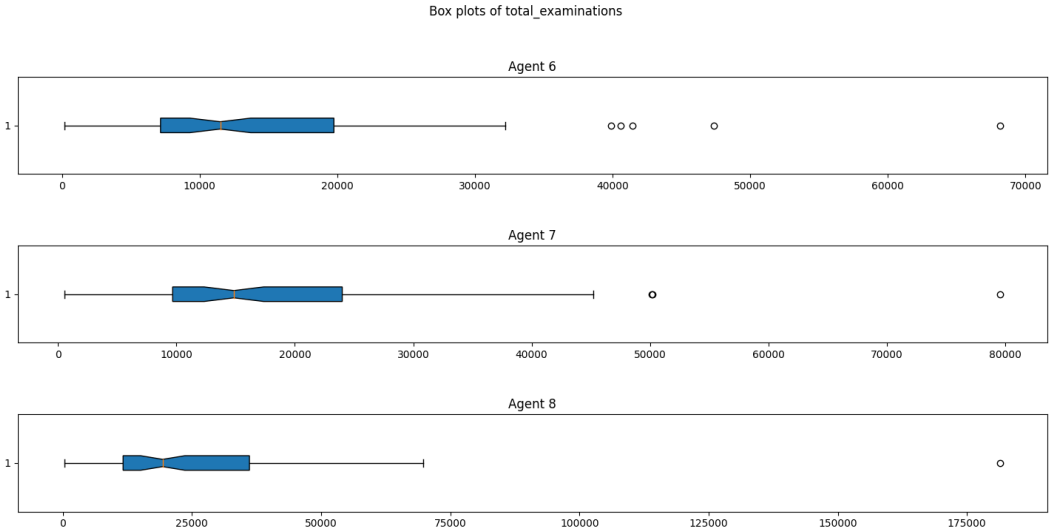

Figure 5.15: Agent 6,7 and 8 Total Examinations Box-plot

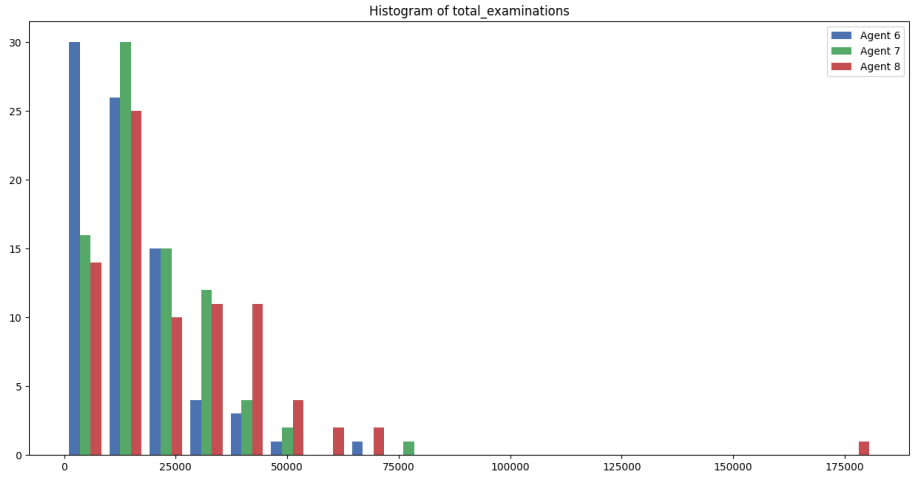

Figure 5.16: Agent 6,7 and 8 Total Examinations Histogram

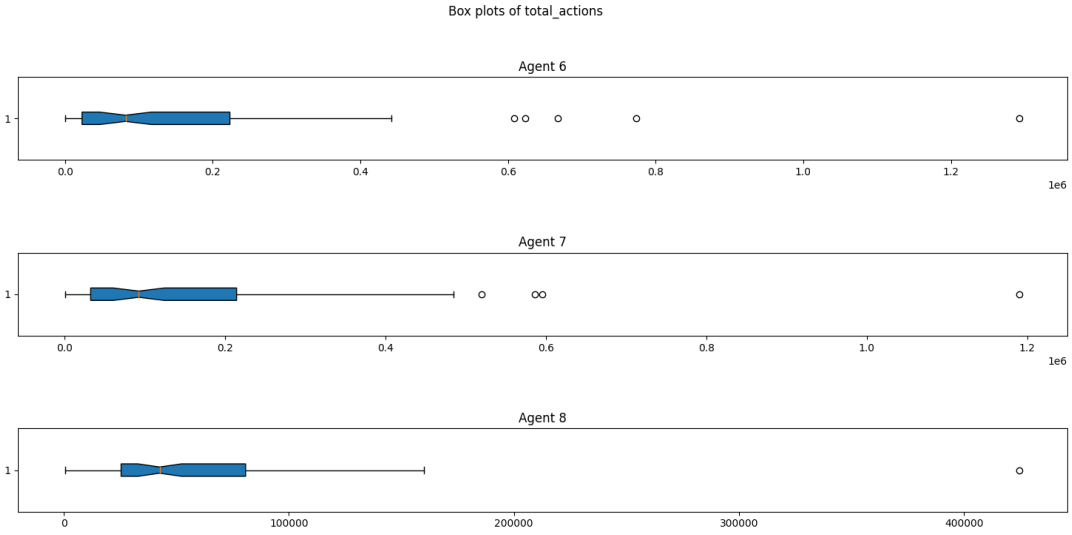

Figure 5.17: Agent 6,7 and 8 Total Cost Box-plot

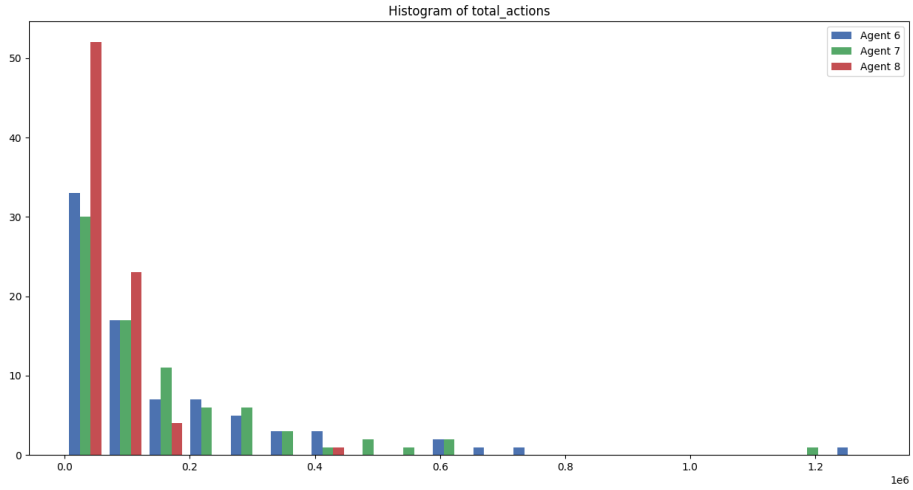

Figure 5.18: Agent 6,7 and 8 Total Cost Histogram

5.1.4 Comparing Agent 6 and 7

We can see from the above data that Agent 6 is outperformed by Agent 7 in terms of movements, examinations and the total cost. This is a consequence of the way Agent 6 decides its current estimated goal. Agent 6 does not take into consideration the terrain type of the target while deciding its current estimated goal. Agent 7 on the other hand incentivizes examining flat cells rather than hilly or forest cells. This is why Agent 7 performs much better than Agent 6 when the target is in a Flat or Hilly cell but performs about equally to Agent 6 when the target is in a Forest.

We can also see from the above graphs that the vast majority of the mazes for agent 6 and 7 are solved in a certain range of total cost. But Agent 6 tends to have a lot more outliers that take a very high number of movements + examinations to solve.

An interesting thing to note from the graphs is that the histogram of the number of examinations/movements follows approximately the probability density function of an exponential distribution.

As we can see from the ratio table, Agent 7 has lower ratios across the board when compared to Agent 6.

5.1.5 Comparing Agent 8 with Agents 6 and 7

We can see very clearly from the above data that Agent 8 eclipses Agent 6 and 7 in terms of total cost. But we see that the number of examinations performed by Agent 8 is higher than the number of examinations performed by Agent 6 or 7. This is because we designed Agent 8 in a way that slightly increases the number of examinations but significantly reduces the number of movements.

An interesting thing to note is the ratios displayed in the movement/examinations table. We see that Agent 8's ratios are almost constant irrespective of the terrain type of the target. This shows us that Agent 8 is not particularly more efficient at solving mazes when the target is in a specific terrain. Whereas as we can see from the ratios of Agent 7, Agent 7 clearly performs much better in mazes where the target has a flat or hilly terrain.

We can see from the above graphs that show the total movement of agents that the total movements of Agent 8 never go above around 100,000 whereas the movements of Agent 6 and 7 vary widely and tend to get very high in a few cases. Whereas from the graphs that show total examinations of agents we see that Agent 8 has a few cases where it performs a lot more examinations than Agents 6 and 7. The graph for total cost shows us that the total cost of agents is heavily determined by the total movements that the agent makes. We also see in the graph for total cost that Agent 8 has a lot fewer outliers when compared to Agent 6 and 7. This shows us that Agent 8 is much more stable and consistent than the other 2 agents.

5.2 Comparison of Agent 9 with Agent 6, 7 and 8

Let us take a look at the data that we obtained for running 300 iterations of Agent 9 in a 100x100 grid world -

Note - All values given below have been averaged out over 300 iterations.

Target in Terrain	*Movements*	*Examinations*	*Total*
Any	3348	16	3363

Table 5.5: Agent 9

5.2.1 Graphs for Agent 9

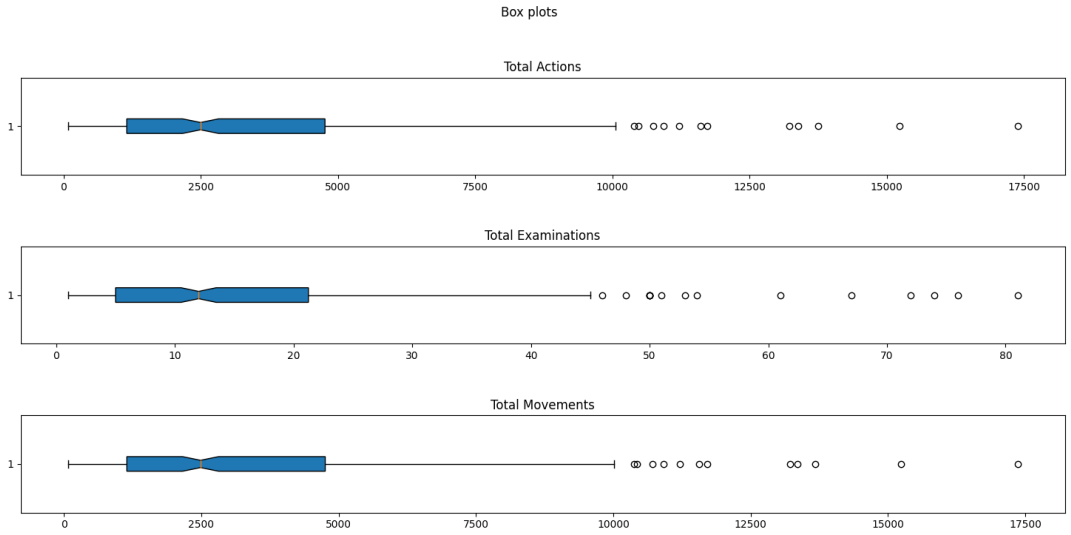

Figure 5.19: Agent 9 Box-plots

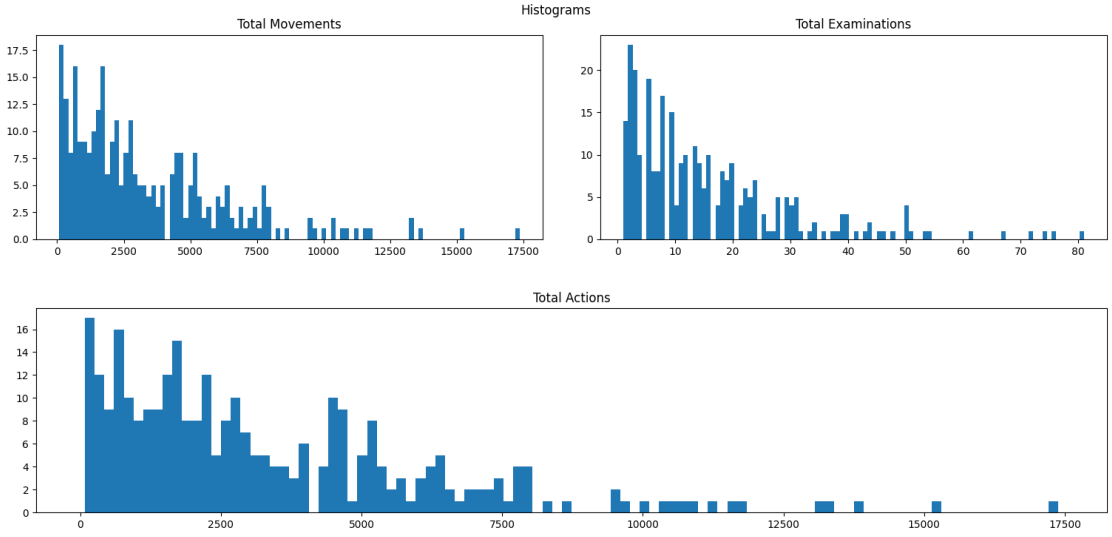

Figure 5.20: Agent 9 Histograms

5.2.2 Comparing Agent 9 with Agents 6,7 and 8

We can see from the graphs and tabular results of Agent 9 that it performs better than Agent 8 which was already better than Agent 6 and 7. Specifically, Agent 9 performs 3 times better than Agent 8 when comparing average overall costs, even when the target is in flat terrain type for Agent 8..

On taking a closer look, we notice that Agent 9's average number of examinations falls roughly between 15 to 20 **total examinations** which is insignificant when compared to other Agents'

average examinations. Agent 9's average number of movements is around 3350. We can see a high discrepancy between movements and examinations for Agent 9 because the Agent will only examine a particular cell when the target is in its valid neighbours. It tactfully decides when to examine a particular cell and removes any chance of a pointless examination. Thus, much of the total final cost is incurred in locating and moving to the target's position.

We can also see a very interesting result from the histograms that 70% percent of total mazes are solved in the range of total cost from 200 to 2750. In contrast to this, even though the target is in flat terrain type terrain type for Agent 8, 90% of the mazes are solved while incurring the total cost in the range of 15000 to 17000.(We consider the case for the target in flat terrain type for Agent 8 because it performs the best in this particular condition.)

This shows the prowess of Agent 9's sensing technique which helps it find the target efficiently even though the target is a moving one.

Appendix A

Code[1]

A.1 Code used by Agents 6,7,8 and 9

```
def compute_current_estimated_goal(maze: list, current_pos: tuple, agent: int,
                                   probability_of_containing_target: np.
                                   ndarray, false_negative_rates: np.
                                   ndarray,
                                   p_of_containing_target_next_step=None):
    """
    Function is used to find a current estimated goal
    :param maze: Maze object
    :param current_pos: current position
    :param agent: agent number
    :param probability_of_containing_target: numpy array to store probabilities
                                        (current belief)
    :param false_negative_rates: numpy array to store false negative rates
    :param probability_of_containing_target_next_step: prediction using current
                                        belief for agent 9
    :return: current estimated goal
    """
    # Prepare a list to store indexes of maximum probabilities
    indexes_of_max_probability = list()

    # distance array to find distance from source to all nodes
    distance_array = length_of_path_from_source_to_all_nodes(maze, current_pos)
```

[1]We have only put the important parts of our code here, please refer to the attached winrar file for any dependant code or for the rest of the code

```
17
18      # Assign current cell's distance to zero to avoid divide by zero error.
19      distance_array[current_pos[0]][current_pos[1]] = INF
20
21      # Normalize the probability
22      probability_of_containing_target /= np.sum(probability_of_containing_target
                                            )
23      if probability_of_containing_target_next_step is not None:
24          p_of_containing_target_next_step /= np.sum(
                                            p_of_containing_target_next_step)
25
26      # Find indexes of maximum probability for agent 6
27      if agent == 6:
28          indexes_of_max_probability = np.where(probability_of_containing_target
                                            == np.amax(
                                            probability_of_containing_target))
29
30      # Find indexes of maximum probability for agent 7
31      elif agent == 7:
32          probability_of_finding_target = np.multiply(
                                            probability_of_containing_target,
                                            ONE_PROBABILITY - false_negative_rates)
                                             indexes_of_max_probability = np.where(
                                            probability_of_finding_target == np.
                                            amax(probability_of_finding_target))
33
34      # Find indexes of maximum probability for agent 8
35      elif agent == 8:
36          probability_of_finding_target = np.multiply(
                                            probability_of_containing_target,
                                            ONE_PROBABILITY - false_negative_rates)
                                             utility_function = np.divide(
                                            probability_of_finding_target,
                                            distance_array)
                                            indexes_of_max_probability = np.where(
                                            utility_function == np.amax(
                                            utility_function))
37
38      # Find indexes of maximum probability for agent 9
```

```
    elif agent == 9:
        probability_of_finding_target = np.multiply(
                                        p_of_containing_target_next_step,
                                        ONE_PROBABILITY - false_negative_rates)
                                         utility_function = np.divide(
                                        probability_of_finding_target,
                                        distance_array)
                                        indexes_of_max_probability = np.where(
                                        utility_function == np.amax(
                                        utility_function))

    # Find indexes from maximum probability with least distance
    indexes_with_max_probability_and_min_distance = np.where(distance_array[
                                        indexes_of_max_probability] == np.amin(
                                        distance_array[
                                        indexes_of_max_probability]))

    # Choose randomly if there's a tie
    random_num = random.randint(0,indexes_with_max_probability_and_min_distance
                                        [0].shape[0] - 1)

    # Return position of current estimated target
    return indexes_of_max_probability[0][indexes_with_max_p_and_min_distance[0]
                                        [random_num]],\
                                        indexes_of_max_probability[1][
                                        indexes_with_max_p_and_min_distance[0][
                                        random_num]]



def compute_probability_when_agent_fails_to_find_target(
                                        probability_of_containing_target: np.
                                        array, false_negative_rates: np.array,
                                        current_pos: tuple):
    """
    Compute probability when agent fails to find target
    :param probability_of_containing_target: update probability directly in
                                        this array
    :param false_negative_rates: numpy array
```

```
    :param current_pos: agent's current position
    :return: None
    """
    p_of_x_y = probability_of_containing_target[current_pos[0]][current_pos[1]]

    reduced_probability = p_of_x_y * false_negative_rates[current_pos[0]][
                                        current_pos[1]]
    probability_denominator = np.sum(probability_of_containing_target) -
                                        p_of_x_y + reduced_probability

    probability_of_containing_target /= probability_denominator
    probability_of_containing_target[current_pos[0]][current_pos[1]] =
                                        reduced_probability/
                                        probability_denominator



def check_and_propagate_probability(probability_of_containing_target: np.array,
                                     false_negative_rates: np.array,
                                    current_pos: tuple, target_pos: tuple):
    """
    Function is used to check and propagate probabilities
    :param probability_of_containing_target: numpy array to store probabilities
    :param false_negative_rates: numpy array to store false negative rates
    :param current_pos: current position
    :param target_pos: target position
    :return: True if agent is able to find out target otherwise False
    """

    # If agent is at target's position, then use false negative rate to find
                                        target in it. If agent has found out,
    # return True otherwise change belief accordingly
    if current_pos == target_pos:

        # If current cell is flat
        if false_negative_rates[current_pos[0]][current_pos[1]] == 0.2:
            x = random.randint(0, 99)
            if x < 20:
```

```
89                  compute_probability_when_agent_fails_to_find_target(
                                        probability_of_containing_target,
                                        false_negative_rates, current_pos)
90              else:
91                  return True
92
93          # If current cell is hilly
94          elif false_negative_rates[current_pos[0]][current_pos[1]] == 0.5:
95              x = random.randint(0, 99)
96              if x < 50:
97                  compute_probability_when_agent_fails_to_find_target(
                                        probability_of_containing_target,
                                        false_negative_rates, current_pos)
98              else:
99                  return True
100
101         # If current cell is forest
102         elif false_negative_rates[current_pos[0]][current_pos[1]] == 0.8:
103             x = random.randint(0, 99)
104             if x < 80:
105                 compute_probability_when_agent_fails_to_find_target(
                                        probability_of_containing_target,
                                        false_negative_rates, current_pos)
106             else:
107                 return True
108     else:
109         compute_probability_when_agent_fails_to_find_target(
                                        probability_of_containing_target,
                                        false_negative_rates, current_pos)
110
111     return False
112
113
114
115 def examine_and_propagate_probability(maze, probability_of_containing_target,
                                        false_negative_rates, current_pos,
                                        target_pos, current_estimated_goal,
                                        node):
116     """
```

```
117     Examine and change probability according to different conditions
118     :param maze: Maze object
119     :param probability_of_containing_target: numpy array of probabilities
120     :param false_negative_rates: numpy array of false negative rates
121     :param current_pos: current position
122     :param target_pos: target position
123     :param current_estimated_goal: current estimated goal
124     :param node: to change examine
125     :return:
126     """
127
128     # check and propagate probability if current position of agent is at
                                        estimated target
129     if current_pos == current_estimated_goal:
130         return check_and_propagate_probability(probability_of_containing_target
                                        , false_negative_rates, current_pos,
                                        target_pos)
131
132     # If maze's node is blocked, change probability accordingly
133     elif maze[node[0]][node[1]].is_blocked:
134         p_of_x_y = probability_of_containing_target[node[0]][node[1]]
135         remaining_probability = np.sum(probability_of_containing_target) -
                                        p_of_x_y
136
137         probability_of_containing_target /= remaining_probability
138         probability_of_containing_target[node[0]][node[1]] = ZERO_PROBABILITY
139
140         return False
141     else:
142         # Else examine node
143
144
145         return check_and_propagate_probability(probability_of_containing_target
                                        , false_negative_rates, node,
                                        target_pos)
```

A.2 Main driver code for Agents 6,7 and 8

```
1
2  def find_the_target(num: int):
3      """
4      Function to run each process for each grid
5      :param num: number of times it's running
6      :return: [total movements, total examinations, total actions]
7      """
8      print('Running for:', num)
9
10     agents = [6, 7, 8]
11
12     # Keep generating grid and target position until we will get valid pair of
                                          it
13     while True:
14         random_maze = generate_grid_with_probability_p(PROBABILITY_OF_GRID)
15         target_pos = generate_target_position(random_maze)
16         if length_of_path_from_source_to_goal(random_maze,
                                          STARTING_POSITION_OF_AGENT, target_pos)
                                           != INF:
17             break
18
19     # Run agent 6,7, and 8 for the above generate grid and target position
20     for agent_num in agents:
21
22         # Print when the agent started it's execution
23         print('Starting agent', agent_num)
24         now = datetime.now()
25         dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
26         print("date and time =", dt_string)
27
28         # Reset agent before using it
29         agent.reset()
30         target_found = False
31
32         # Run the following loop until target is not found
33         while not target_found:
34
35             # First, find the current estimated target
```

```
36              agent.pre_planning(agent_num)
37
38              # Second, prepare a path to reach to this target
39              agent.planning(agent.current_estimated_goal)
40
41              # If the given target is not reachable, set it's probability of
                                    containing target to zero and find
                                    another
42              # target and it's corresponding path
43              while agent.current_estimated_goal not in agent.parents:
44                  agent.maze[agent.current_estimated_goal[0]][agent.
                                    current_estimated_goal[1]].is_blocked =
                                     True
45                  examine_and_propagate_probability(agent.maze, agent.
                                    probability_of_containing_target, agent
                                    .false_negative_rates, agent.
                                    current_position, target_pos, agent.
                                    current_estimated_goal, agent.
                                    current_estimated_goal)
46                  agent.pre_planning(agent_num)
47                  agent.planning(agent.current_estimated_goal)
48
49              # Execute on the generated path
50              agent.execution(random_maze)
51
52              # Examine the current cell
53              target_found = agent.examine(target_pos)
54
55          # Find total number of movements
56          movements = compute_explored_cells_from_path(agent.final_paths)
```

A.3 Main driver code for Agent 9

```
1
2  def find_moving_target(num: int):
3      """
4      This function is used to test agent 9
5      :param num: iteration number
6      :return: (movements, examinations, actions)
7      """
8
9      # Start running agent 9
10     print('Running for', num)
11     agent = Agent9()
12     agent_num = 9
13
14     # Keep generating grid and target position until we will get valid pairs.
15     while True:
16         full_maze = generate_grid_with_probability_p(PROBABILITY_OF_GRID)
17         target_position = generate_target_position(full_maze)
18         if length_of_path_from_source_to_goal(full_maze,
                                                 STARTING_POSITION_OF_AGENT,
                                                 target_position) != INF:
19             break
20
21     target_found = False
22
23     # Run the following while loop until we will get the target
24     while not target_found:
25
26         # First, agent will sense its neighbor and update its belief and
                                                 prediction for the next step
27         agent.sense(target_position)
28
29         # Second, agent will find an estimated goal using the prediction
                                                 probability
30         agent.pre_planning(agent_num)
31
32         # Third, agent will plan a path to reach an estimated goal
33         agent.planning(agent.current_estimated_goal)
34
```

```
35          # If the current estimated goal is not reachable, agent will assign the
                                        probability to zero and redo
36          # pre-planning and planning until it will get a reachable target
37          while (not agent.is_target_in_neighbors) and (agent.
                                        current_estimated_goal not in agent.
                                        parents):
38              agent.maze[agent.current_estimated_goal[0]][agent.
                                        current_estimated_goal[1]].is_blocked =
                                         True
39              examine_and_propagate_probability(agent.maze, agent.
                                        p_of_containing_target_next_step, agent
                                        .false_negative_rates, agent.
                                        current_position, target_position,
                                        agent.current_estimated_goal, agent.
                                        current_estimated_goal)
40              agent.pre_planning(agent_num)
41              agent.planning(agent.current_estimated_goal)
42
43          # Then, next time step will come and target will move one step
44          target_position = move_target(target_position, full_maze)
45
46          # Agent will start its execution and examination
47          target_found = agent.execution(full_maze, target_position)
48
49      # Compute number of movements
50      movements = compute_explored_cells_from_path(agent.final_paths)
51
52      return agent.num_examinations, movements, agent.num_examinations +
                                        movements
```