RUTGERS

THE STATE UNIVERSITY OF NEW JERSEY

PROBABILISTIC SENSING

INTRO TO AI – CS520

PROJECT MEMBERS:

PRASHANT P KANTH (NETID: PPK31)

PARTH GOEL (NETID: PG514)

UTKARSH JHA (NETID: UJ22)

UNDER GUIDANCE OF: -

PROF. CHARLES COWAN

AND

ARAVIND SIVA (TA)

ASSIGNMENT - 3

Probability in the Gridworld

A robot is located in the gridworld, and wants to find a hidden target. The robot can move between unblocked cells (up/down/left/right), and can examine the cell it is currently in to try to determine if the target is there. However, examination can potentially yield **false negatives**. If the target is in the cell being examined, the examination fails with different probabilities based on the terrain.

Terrain Type of Cells	**False Negative Rate**
Flat (Relatively easy to search)	0.2
Hilly (Potentially easy to hide)	0.5
Thick Forest (Very easy to miss target)	0.8

Table 1.1 False Negative Rates for each terrain type

Hence if the agent/robot examines a cell and fails to find the target there, it does not mean that the target isn't there - but it does reduce the likelihood that the target is there. If however the agent examines the cell and finds the target, the grid is solved - there are no false positives.

At every point in time, the agents will need to track a belief state, describing the probability of the target being in each cell - this will need to be updated as it learns more about the environment.

The robot initially does not know what cells are blocked, or the terrain type of each cell. Hence initially the probability of the target being in a given cell will be same for each cell in the gridworld, which can be represented as:

1) Prior to any interaction with the environment, the probability of the target being in a given cell:

$$P i, j = 1/n, \text{ where } n = \text{number of cells in gridworld, or}$$
$$n = \text{rows} * \text{cols}$$

The robot is 'blindfolded', so it can only tell a cell is blocked if it attempts to move into that cell and fails. When it successfully enters (or starts in) a cell, however, the robot can sense the terrain type of that cell. We can write below probabilities from the information that we have:

$$P x, y(cell\ containing\ target \mid x, y\ is\ blocked) = 0$$
$$P x, y(examine\ fails \mid x, y\ is\ flat) = 0.2$$
$$P x, y(examine\ fails \mid x, y\ is\ hilly) = 0.5$$
$$P x, y(examine\ fails \mid x, y\ is\ forest) = 0.8$$

The movement of agent in this unknown gridworld of different terrain types is a random process which is based on probability calculations of belief state and observations.

One property that makes the study of a random process much easier is the **"Markov property"**. In a very informal way, the Markov property says, for a random process, that if we know the value taken by the process at a given time, we won't get any additional information about the future behaviour of the process by gathering more knowledge about the past. Stated in slightly more mathematical terms, for any given time, the conditional distribution of future states of the

process given present and past states depends only on the present state and not at all on the past states (**memoryless property**).[1]

$$\mathrm{P}(future \mid present, past) = \mathrm{P}(future \mid present)$$

Let $\mathrm{P}i,j(t)$ be the probability that cell (i,j) contains the target, given the observations collected up to time t. At time $t+1$, suppose the agent learns new information about cell (x,y). Depending on what information it learns, the probability for each cell needs to be updated. The new $\mathrm{P}i,j(t+1)$ for each cell (i,j) under the following circumstances:

Note: 1) All observations till time t will be absorbed in $\mathrm{P}i,j(t)$
2) At any given time t, $\sum \mathrm{P}i,j(t) = 1$

2.1) At time t + 1 agent attempts to enter (x, y) and find it is blocked?

Agent will have an observation only for cell (x, y) – *cell (x, y) is blocked*

$$\mathrm{P}x,y(t+1) = \frac{\mathrm{P}x,y(t) * Observation(t+1)}{\sum \mathrm{P}i,j(t)}$$

$$\mathrm{P}x,y(t+1) = \mathrm{P}x,y(t) * \mathrm{Px,y}(cell\ containing\ target \mid x,y\ is\ blocked)$$

$$\mathrm{P}x,y(t+1) = 0$$

Also, for all other cells probabilities gets updated as:

$$\mathrm{P}i,j(t+1) = \frac{\mathrm{P}i,j(t)}{\mathrm{P}x,y(t+1) + \sum_{-(x,y)} \mathrm{P}i,j(t)}$$

$$\mathrm{P}i,j(t+1) = \frac{\mathrm{P}i,j(t)}{0 + \sum_{-(x,y)} \mathrm{P}i,j(t)}$$

Or,

$$\mathrm{P}i,j(t+1) = \frac{\mathrm{P}i,j(t)}{\sum Pi,j}$$

2.2) At time t + 1 agent attempts to enter (x, y), finds it unblocked, and also learns its terrain type?

Observation about terrain type will be updated in discovered grid.

There will be no change in probability value of any cell, since agent only examines a cell once it reaches an identified target.

[1] – explanation of Markov property taken from: towardsdatascience.com

2.3) At time t + 1 agent examines cell (x, y) of terrain type flat, and fails to find the target?

Agent will have an observation only for cell (x, y) – *cell (x, y) is flat terrain and examine failed*

$$\mathrm{P}x,y(t+1) = \frac{\mathrm{P}x,y(t) * Observation(t+1)}{\sum \mathrm{P}i,j(t)}$$

$$\mathrm{P}x,y(t+1) = \mathrm{P}x,y(t) * \mathrm{P}x,y(examine\ fails \mid x,y\ is\ flat)$$

$$\mathrm{P}x,y(t+1) = \mathrm{Px,y}(t) * 0.2$$

Also, for all other cells probabilities gets updated as:

$$\mathrm{P}i,j(t+1) = \frac{\mathrm{P}i,j(t)}{\mathrm{P}x,y(t+1) + \sum_{-(x,y)} \mathrm{P}i,j(t)}$$

Or,

$$\mathrm{P}i,j(t+1) = \frac{\mathrm{P}i,j(t)}{\sum Pi,j}$$

2.4) At time t + 1 agent examines cell (x, y) of terrain type hilly, and fails to find the target?

Agent will have an observation only for cell (x, y) – *cell (x, y) is hilly terrain and examine failed*

$$\mathrm{P}x,y(t+1) = \frac{\mathrm{P}x,y(t) * Observation(t+1)}{\sum \mathrm{P}i,j(t)}$$

$$\mathrm{P}x,y(t+1) = \mathrm{P}x,y(t) * \mathrm{P}x,y(examine\ fails \mid x,y\ is\ hilly)$$

$$\mathrm{P}x,y(t+1) = \mathrm{Px,y}(t) * 0.5$$

Also, for all other cells probabilities gets updated as:

$$\mathrm{P}i,j(t+1) = \frac{\mathrm{P}i,j(t)}{\mathrm{P}x,y(t+1) + \sum_{-(x,y)} \mathrm{P}i,j(t)}$$

Or,

$$\mathrm{P}i,j(t+1) = \frac{\mathrm{P}i,j(t)}{\sum Pi,j}$$

2.5) At time t + 1 agent examines cell (x, y) of terrain type forest, and fails to find the target?

Agent will have an observation only for cell (x, y) – *cell (x, y) is forest terrain and examine failed*

$$\mathrm{P}x,y(t+1) = \frac{\mathrm{P}x,y(t) * Observation(t+1)}{\sum \mathrm{P}i,j(t)}$$

$$\mathrm{P}x,y(t+1) = \mathrm{P}x,y(t) * \mathrm{P}x,y(examine\ fails \mid x,y\ is\ forest)$$

$$\mathrm{P}x,y(t+1) = \mathrm{Px,y}(t) * 0.8$$

Also, for all other cells probabilities gets updated as:

$$\mathrm{P}i,j(t+1) = \frac{\mathrm{P}i,j(t)}{\mathrm{P}x,y(t+1) + \sum_{-(x,y)} \mathrm{P}i,j(t)}$$

Or,

$$\mathrm{P}i,j(t+1) = \frac{\mathrm{P}i,j(t)}{\sum Pi,j}$$

2.6) At time t + 1 agent examines cell (x, y), and finds the target?

Once the agent finds the target, it means that the gridworld has been solved, as there are no false negatives.

$$\mathrm{P}x,y(t+1) = 1$$

For all other cells, probability will drop to 0.

$$\mathrm{P}i,j(t+1) = 0$$

From the information that we have for different terrain types:

$$\mathrm{P}x,y(examine\ fails \mid x,y\ is\ flat) = 0.2$$
$$\mathrm{P}x,y(examine\ fails \mid x,y\ is\ hilly) = 0.5$$
$$\mathrm{P}x,y(examine\ fails \mid x,y\ is\ forest) = 0.8$$

We can compute below probabilities,
Flat Terrain:

$$\mathrm{P}x,y(examine\ succeeds \mid x,y\ is\ flat) = 1 - \mathrm{P}x,y(examine\ fails \mid x,y\ is\ flat)$$
$$\mathrm{P}x,y(examine\ succeeds \mid x,y\ is\ flat) = 0.8$$

Hilly Terrain:

$$\mathrm{P}x,y(examine\ succeeds \mid x,y\ is\ hilly) = 1 - \mathrm{P}x,y(examine\ fails \mid x,y\ is\ hilly)$$
$$\mathrm{P}x,y(examine\ succeeds \mid x,y\ is\ hilly) = 0.5$$

Forest Terrain:

$$\mathrm{P}x,y(examine\ succeeds \mid x,y\ is\ forest) = 1 - \mathrm{P}x,y(examine\ fails \mid x,y\ is\ forest)$$
$$\mathrm{P}x,y(examine\ succeeds \mid x,y\ is\ forest) = 0.2$$

At time t, with probability $\mathrm{P}i,j(t)$ of cell (i,j) containing the target, the probability of finding the target in cell (x,y) will be:

Let $\mathrm{P}x,y(t)$ be the probability of finding a target in a cell at time t given its probability of containing the target at time t, $\mathrm{P}i,j(t)$.

3.1) If (x,y) is hilly?

$$\mathrm{P}x,y(t) = \mathrm{P}i,j(t) * \mathrm{P}x,y(examine\ succeeds \mid x,y\ is\ hilly)$$
$$\mathrm{P}x,y(t) = \mathrm{P}i,j(t) * 0.5$$

3.2) If (x,y) is flat?

$$\mathrm{P}x,y(t) = \mathrm{P}i,j(t) * \mathrm{P}x,y(examine\ succeeds \mid x,y\ is\ flat)$$
$$\mathrm{P}x,y(t) = \mathrm{P}i,j(t) * 0.8$$

3.3) If (x,y) is forest?

$$\mathrm{P}x,y(t) = \mathrm{P}i,j(t) * \mathrm{P}x,y(examine\ succeeds \mid x,y\ is\ forest)$$
$$\mathrm{P}x,y(t) = \mathrm{P}i,j(t) * 0.2$$

3.4) If (x,y) has never been visited?

Since, the agent has not visited (x,y) yet, it can't say anything about the probability of finding the target in unvisited cells.
However, if we apply marginalisation here, the agent will then be able to find a probability for cells that have not been visited.
There is equal probability of these cells being forest, hilly or flat terrain. Therefore, $\mathrm{P}x,y(t)$ can be written as:

$$\mathrm{P}x,y(t) = \mathrm{P}i,j(t) * [\mathrm{Px,y}(flat) * \mathrm{Px,y}(examine\ succeeds \mid x,y\ is\ flat) + \mathrm{P}x,y(hilly) * \mathrm{Px,y}(examine\ succeeds \mid x,y\ is\ hilly) + \mathrm{Px,y}(forest) * \mathrm{P}x,y(examine\ succeeds \mid x,y\ is\ forest)]$$

Now, $\mathrm{P}x,y(block) = 0.3$ and hence, $\mathrm{P}x,y(unblocked) = 0.7$
These unblocked cells have equal probability of being flat, hilly or forest, which is $(0.7/3)$
The above equation can be rewritten as:

$$\mathrm{P}x,y(t) = \mathrm{P}i,j(t) * [0.7/3 * (\mathrm{P}x,y(examine\ suceeds \mid x,y\ is\ flat) + \mathrm{P}x,y(examine\ succeeds \mid x,y\ is\ hilly) + \mathrm{P}x,y(examine\ succeeds \mid x,y\ is\ forest))]$$

Or,

$$\mathrm{P}x,y(t) = \mathrm{P}i,j(t) * [0.7/3 * (0.8 + 0.5 + 0.2)]$$
$$\mathrm{P}x,y(t) = \mathrm{P}i,j(t) * [(0.7/3) * 1.5]$$

The agents starts knowing nothing about the cell contents or terrains, then learns about its environment by traveling through it and taking measurements (moving and sensing / examining), updating its probabilities as it goes.

Let us look at Agent 6 and Agent 7:

Agent 6:

i) At any given time, the agent identifies the cell with the highest probability of containing the target, given what the agent has observed.
ii) If there are multiple cells that have equal (maximum) probability, ties should be broken by distance; if there are equidistant cells with equal (maximum) probability, ties should be broken uniformly at random.
iii) The agent then plans a route to that cell using Repeated A*.
iv) As the agent moves along the planned route, it is both sensing the terrain of the cells it moves into and whether or not the cell it is trying to move into is blocked.
    - If a blocked cell is encountered, or the cell with the highest probability of containing the target changes based on information collected while traveling, the agent must re-plan a new route.
v) Once it arrives at the intended cell, it will examine the cell, and then repeat the process as necessary.

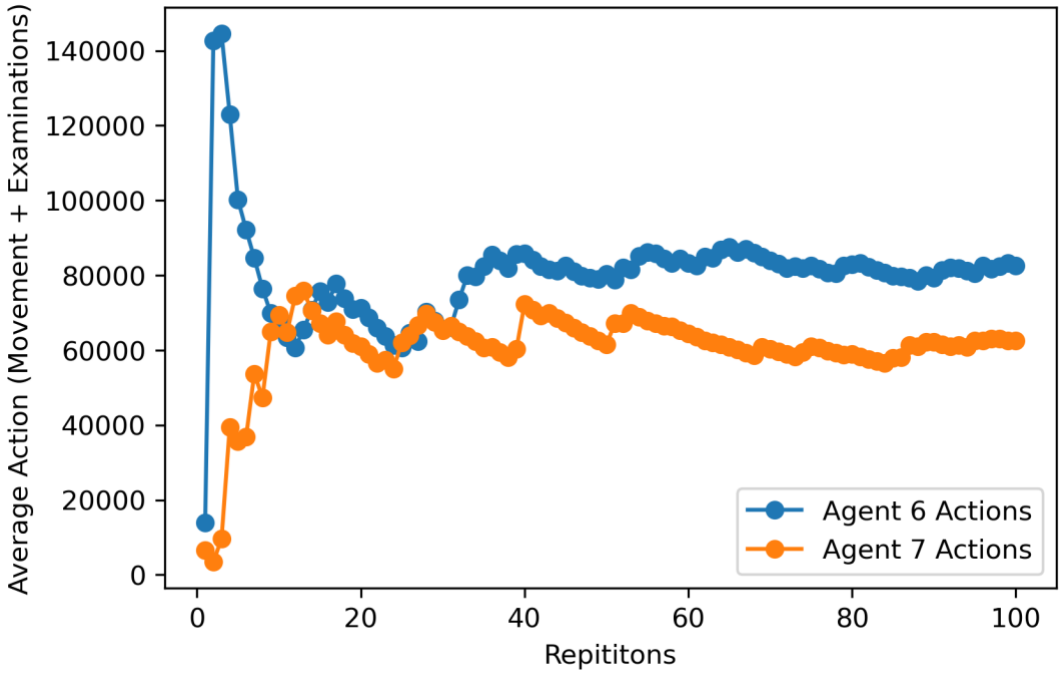
Agent 7:

i) At any given time, the agent identifies the cell with the highest probability of successfully finding the target, given what the agent has observed.
ii) If there are multiple cells that have equal (maximum) probability, ties should be broken by distance; if there are equidistant cells with equal (maximum) probability, ties should be broken uniformly at random.
iii) The agent then plans a route to that cell using Repeated A*.
iv) As the agent moves along the planned route, it is both sensing the terrain of the cells it moves into and whether or not the cell it is trying to move into is blocked.
    - If a blocked cell is encountered, or the cell with the highest probability of finding the target changes based on information collected while traveling, the agent must re-plan a new route.
v) Once it arrives at the intended cell, it will examine the cell, and then repeat the process as necessary.

We will generate gridworld with different terrains (flat, forest, hilly) and blocks (p=0.3). Randomly spawn agent and target at unblocked cells and run Agent 6 and Agent 7 on a variety of randomly generated gridworlds.
For both agents, we will estimate the number of actions (movement + examinations) each agent needs on average to find the target.

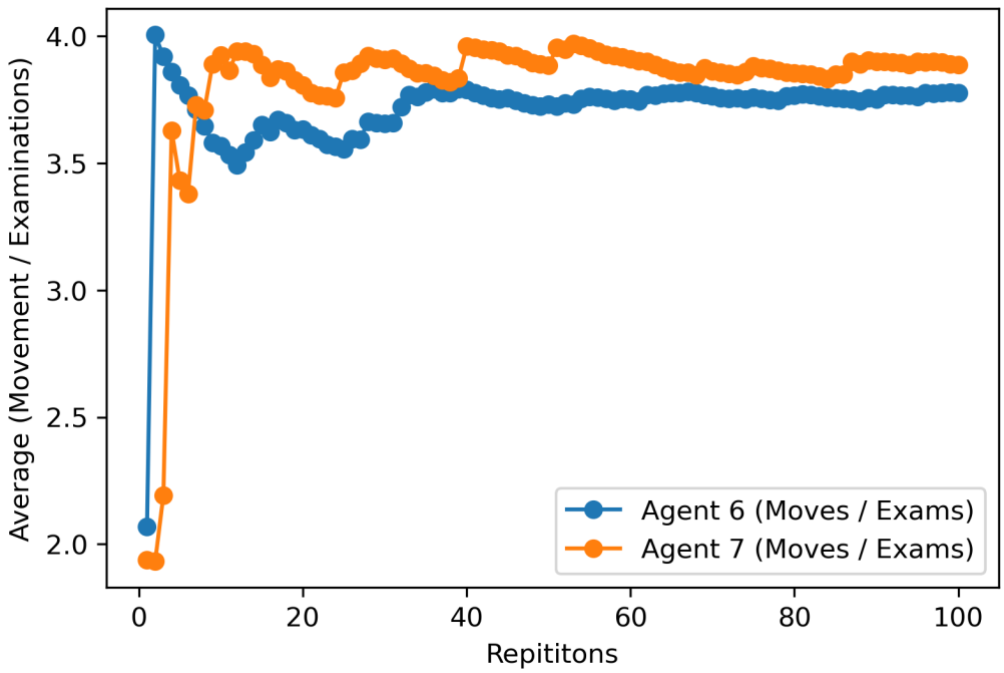
For the purpose of comparison we will consider only those gridworlds that are solvable (there exists a valid path from agent's location to target's location).

4) Comparison plots between Agent 6 and Agent 7



Plot 1.1 Average Actions (Movement + Examinations) for 100 repetitions on 101x101 grid

As we can read from the graph above (Plot 1.1), Agent 7 takes less number of actions compared to Agent 6 on an average for 100 runs on randomly generated gridworlds of 101x101 dimension.



Plot 1.2 Average (Movements / Examinations) for 100 repetitions on 101x101 grid

Plot 1.2 tells us that on an average Agent 7 does less examinations on the gridworld compared to its movement when a target is identified. This distribution of movements / examinations is

expected for Agent 7, as whenever the agent senses a terrain type, the probability of finding the target for all cells in the gridworld changes and thus there will be cases when the target (cell with $max(\mathrm{Px,y}(finding\ target))$) is shifted mid-way, before even reaching the previously identified target, which results in less number of examinations.

From results of Plot 1.1 and Plot 1.2, we can say that (on an average) Agent 7 outperforms Agent 6. But it also lingers a question, can our agents do better ? What else can we do to make our agents take less number of actions and find the target, given the information that we already have ? On this note, we will start to build our Agent 8, with a gleamy hope of outperforming Agent 6 and Agent 7.

5) Implementation Algorithm for Agent 8

All the steps essentially remains same, the difference would be how we select our target, and how we reject certain cells.
As we observed earlier that on an average Agent 7 emerged as a winner over Agent 6, so, we would like to build the target selection mechanism over Agent 7's mechanism. Agent 7 used to select target based on the highest probability of successfully finding the target, given what the agent has observed till time t.

Target Selection

Remember from Section 3.1, 3.2, 3.3 and 3.4, the formula to calculate the probability of finding a target at time t, $\mathrm{P}x,y(t)$, given its probability of containing the target at time t, $\mathrm{P}i,j(t)$:

If (x,y) is hilly:

$$\mathrm{P}x,y(t) = \mathrm{P}i,j(t) * 0.5$$

If (x,y) is flat:

$$\mathrm{P}x,y(t) = \mathrm{P}i,j(t) * 0.8$$

If (x,y) is forest:

$$\mathrm{P}x,y(t) = \mathrm{P}i,j(t) * 0.2$$

If (x,y) has never been visited:

$$\mathrm{P}x,y(t) = \mathrm{P}i,j(t) * [(0.7/3) * 1.5]$$

As our goal is to reduce the number of actions, we will introduce a **utility** calculation that gives equal importance to **probability and distance**. We will calculate this utility in such a way that we are able to select an optimal target. This target might be a cell with slightly less $\max(\mathrm{P}x,y(t))$ but more closer to agent and hence adding more value for examination.

To obtain a utility value, $\sqcup\boldsymbol{x,y(t)}$, we first normalize the distances and then divide each cell's $\mathrm{P}x,y(t)$ by its normalized distance from the agent's current position.
To normalize the distances, we take the distance of the max distant cell, $dmax$, from the agent's current position, which would be at any point in time distance of one of the four corners of the gridworld.

$$dmax(t) = max(d(0,0), d(0,\mathrm{dim\_}y-1), d(\mathrm{dim\_}x-1,0), d(\mathrm{dim\_}x-1,\mathrm{dim\_}y-1))$$

(Note:- Here, we are taking dim_x-1 and dim_y-1, as array indexes start from 0.)

Once we have $dmax(t)$, we divide each cell's distance from agent's current position, $dx, y(t)$, by $dmax(t)$ and add 1 to it. We add 1 so that the values come between $[1,2]$ and by doing this we are easily able to consider the cell on which agent stands (as the distance of that cell would be 0 and when we try to find it's utility value, it would throw error for divideByZero). Let's denote normalized distance of cell (x, y) at time t by $\aleph x, y(t)$.

$$\aleph x, y(t) = 1 + \frac{dx, y(t)}{dmax(t)}$$

(Note:- Distances calculated here are manhattan distance.)

Now we have all the required values to calculate utility of a cell (x, y) at time t.

$$\sqcup x, y(t) = \frac{\mathrm{P}x, y(t)}{\aleph x, y(t)}$$

If there are multiple cells with equal $max(\sqcup x, y(t))$, consider the immediate neighbours of these cells as separate clusters. Calculate the mean total utility value, $\mu_{x,y}(t)$, of each cluster. The cells cluster that have the maximum mean total utility value, $\mu(t)$ will be taken forward.

If there are multiple cells with equal $\mu(t)$, ties are broken by selecting the cells that are at the closest distance from agent's position at time t. If we still have multiple cells, target is chosen at random from among these cells.

Let's try to visualise this to have a better understanding.
Consider a grid of 7x7 (dim_x = 7, dim_y = 7):
Now, we have $\mathrm{P}x, y(t)$ for each cell which is probability of finding a target, given its probability containing the target at time t. Assume that agent is at cell A.

	0	1	2	3	4	5	6
0							
1							
2							
3					**A**		
4							
5							
6							

First, we need to calculate $dmax(t)$ to normalize all distances.
d(0,0) = 7; d(0,6) = 5; d(6,0) = 7; d(6,6) = 5
$dmax(t) = max(7,5,7,5) = 7$

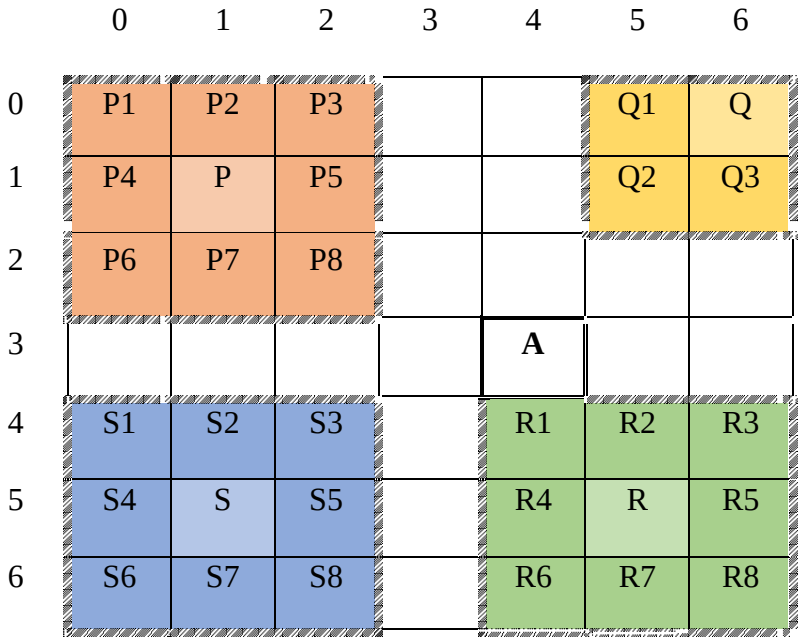
Now, we need to normalize individual cell's distance from agent's position. We will find each cell's manhattan distance $dx, y(t)$, divide it by $dmax(t)$ and add 1.

$$\aleph x, y(t) = 1 + \frac{dx, y(t)}{dmax(t)} = 1 + \frac{dx, y(t)}{7}$$

Next step is to find utility value for each cell:

$$\sqcup x, y(t) = \frac{\mathrm{P}x, y(t)}{\aleph x, y(t)}$$

Let' say that we have 4 cells (P, Q, R and S) with equal $max(\sqcup x, y(t))$ value. We need to break tie by taking individual clusters of these cells, finding mean of each cluster's total utility $\mu_{x,y}(t)$ and then taking cells that have maximum mean cluster utility $\mu(t)$.



Each colour represents one cluster. Cluster P , Cluster Q , Cluster R and Cluster S
Now, we calculate mean utility of each individual cluster:

$$\mu_P(t) = \frac{\sqcup p1(t) + \sqcup p2(t) + \sqcup p3(t) + \sqcup p4(t) + \sqcup p5(t) + \sqcup p6(t) + \sqcup p7(t) + \sqcup p8(t)}{8}$$

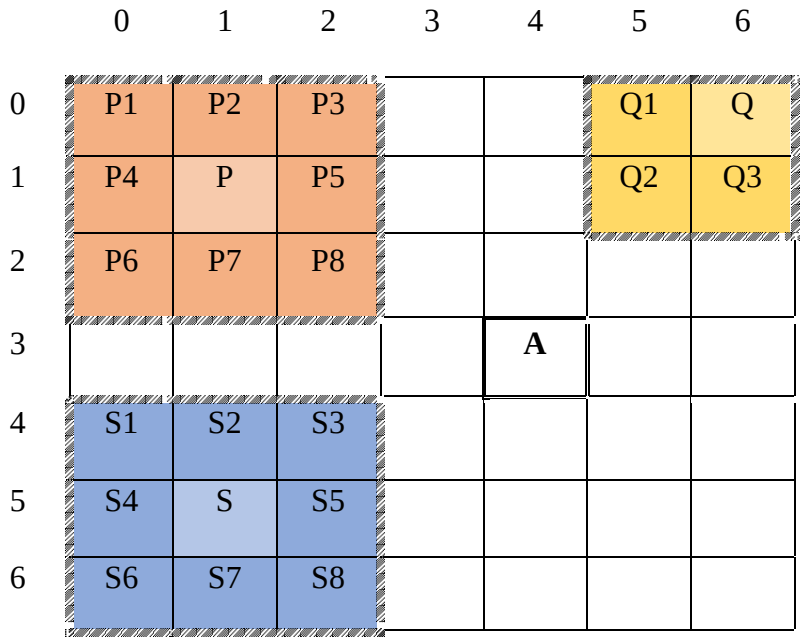
$$\mu_Q(t) = \frac{\sqcup q1(t) + \sqcup q2(t) + \sqcup q3(t)}{3}$$

$$\mu_R(t) = \frac{\sqcup r1(t) + \sqcup r2(t) + \sqcup r3(t) + \sqcup r4(t) + \sqcup r5(t) + \sqcup r6(t) + \sqcup r7(t) + \sqcup r8(t)}{8}$$

$$\mu_S(t) = \frac{\sqcup s1(t) + \sqcup s2(t) + \sqcup s3(t) + \sqcup s4(t) + \sqcup s5(t) + \sqcup s6(t) + \sqcup s7(t) + \sqcup s8(t)}{8}$$

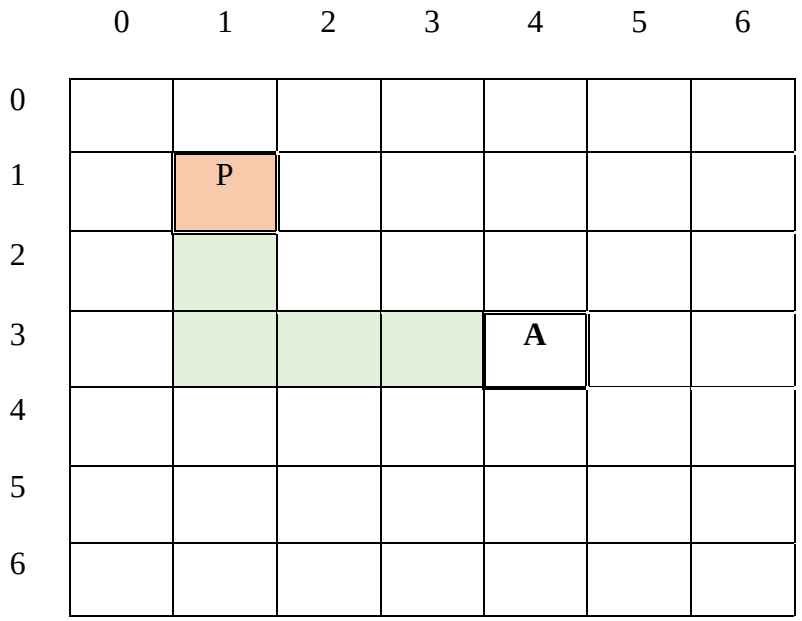
Calculate maximum mean utility, $\mu(t) = max\left(\mu_P(t), \mu_Q(t), \mu_R(t), \mu_S(t)\right)$

Let's say that 3 clusters have same value as $\mu(t)$, Cluster P, Cluster Q and Cluster S. That means, we still have 3 cells with tie: P, Q and S.



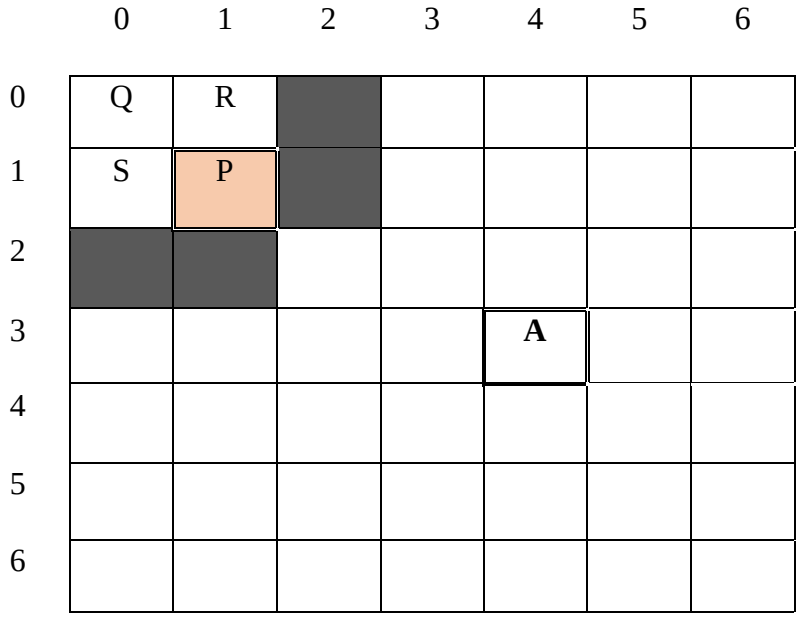
Now, we need to break tie based on minimum $dx, y(t)$ from agent's position.
If we calculate manhattan distance of cell P, Q and S from A, they all come to 5, and hence calculating $min(dp(t), dq(t), ds(t)) = 5$ still leaves all three cells: P, Q and S.

Finally we break the tie and select any of these cells at random (P was selected randomly) as identified target.



Rejecting unreachable cells

Let us consider the same grid as above but with blocks around cell P. Let us call neighbouring cells of P as Q, R and S.



In this scenario, agent will not be able to find a path to target P. Now, we can essentially programme our agent to discard cells that were not reachable. Agent will look for all possible paths and would have covered essentially all reachable cells of the grid in this case, and thus can discard cells that are not present in the knowledge base. At this point, agent will make probability of containing target, $P_{i,j}(t) = 0$, for all identified blocked cells as well as for unreachable cells, P, Q, R and S, and increase $P_{i,j}(t)$ for all other cells.

Agent 8:

i) At any given time, the agent identifies the cell with the highest utility value for finding the target, given what the agent has observed.

ii) If there are multiple cells that have equal (maximum) utility, ties should be broken by mean highest utility cluster of each cell; if there are multiple cells with (maximum) mean utility cluster value, ties should be broken by distance; if there are equidistant cells with equal (maximum) mean utility cluster, ties should be broken uniformly at random.

iii) The agent then plans a route to that cell using Repeated A*.

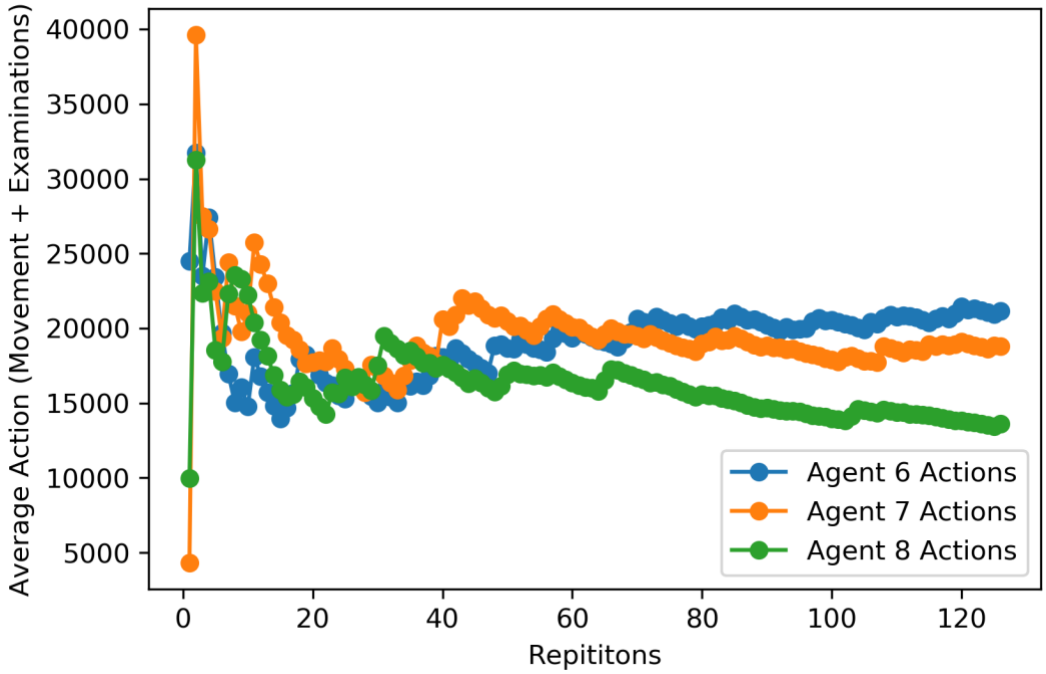
iv) As the agent moves along the planned route, it is both sensing the terrain of the cells it moves into and whether or not the cell it is trying to move into is blocked.
- If a blocked cell is encountered, or the cell with the highest probability of finding the target changes based on information collected while traveling, the agent must re-plan a new route.
- If certain identified target/cells are unreachable, those should be discarded and then agent must re-plan a new route.

v) Once it arrives at the intended cell, it will examine the cell, and then repeat the process as necessary.

We will generate gridworld with different terrains (flat, forest, hilly) and blocks (p=0.3). Randomly spawn agent and target at unblocked cells and run Agent 6, Agent 7 and Agent 8 on a variety of randomly generated gridworlds.
For all three agents, we will estimate the number of actions (movement + examinations) each agent needs on average to find the target.

For the purpose of comparison we will consider only those gridworlds that are solvable (there exists a valid path from agent's location to target's location).

6) Comparison plots between Agent 6, Agent 7 and Agent 8



Plot 1.3 Average Action (Movement + Examinations) for 126 repetitions on 60x60 grid

Plot 1.3 shows that Agent 8 takes less action on an average as compared to Agent 6 and Agent 7. Here, we can see that prioritizing a cell with slightly less $\mathrm{P}x, y(t)$ than $max\big(\mathrm{P}x, y(t)\big)$ but one that is a bit closer to agent than the cell with $max\big(\mathrm{P}x, y(t)\big)$ is proving to be better. We can look at it with a simple example, let's say that we had 7 cells, whose values are like:

Cell	$\mathbf{P}x, y(t)$	$dx, y(t)$	$\aleph x, y(t)$	$\sqcup x, y(t)$
A	0.05	3	$1 + (3/12) = 1.25$	$0.05/1.25 = 0.04$
B	0.06	4	$1 + (4/12) = 1.33$	$0.06/1.33 = 0.0451$
C	0.09	0	$1 + (0/12) = 1$	$0.09/1 = 0.09$
D	0.12	8	$1 + (8/12) = 1.66$	$0.12/1.66 = 0.072$
E	0.20	8	$1 + (8/12) = 1.66$	$0.20/1.66 = 0.12048$
F	0.24	12 $(dmax(t))$	$1 + (12/12) = 2$	$0.24/2 = 0.12$
G	0.22	11	$1 + (11/12) = 1.91$	$0.22/1.91 = 0.1151$

It would be more valuable to visit cell E ($max(\sqcup x, y(t))$) in this case, as $\mathrm{P}x, y(t)$ might be little lower than $max(\mathrm{P}x, y(t))$ but it is not too far behind the max value also, and additionally we would be saving 4 extra steps, hence reducing movements. Thus cell E has more utility $\sqcup x, y(t)$ as compared to others.

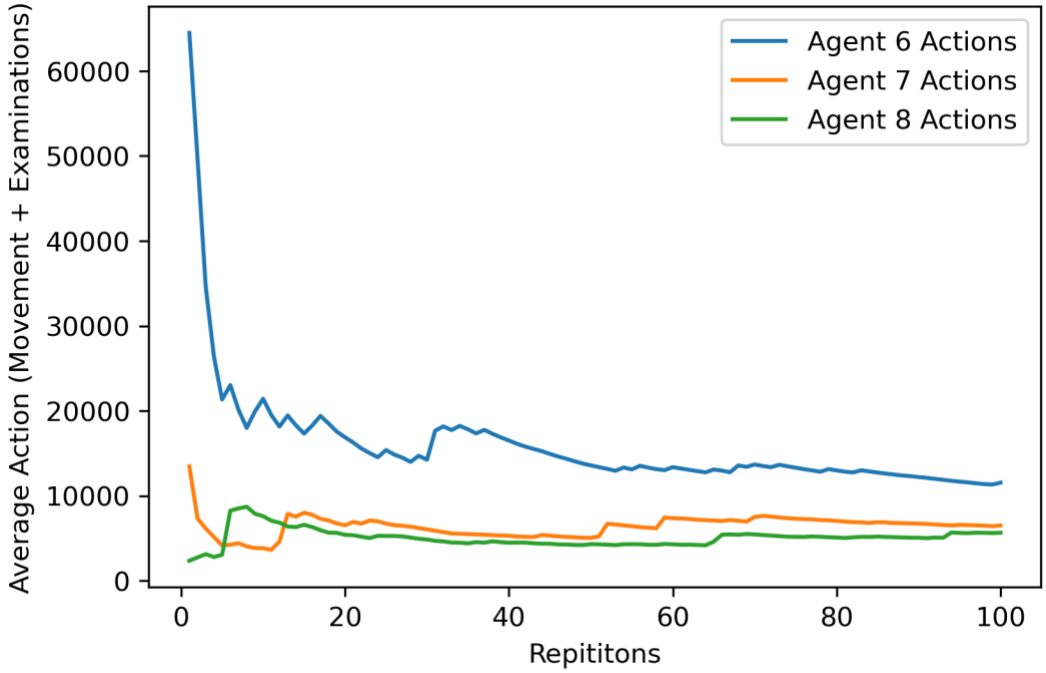
Plot 1.4 Average (Movement / Examinations) for 126 repetitions on 60x60 grid

Plot 1.4 shows that Agent 8 has less (Movement / Examinations) as compared to Agent 6 and Agent 7, which essentially means that Agent 8 does more examinations with respect to its movements. This behaviour is also expected as we are reducing the number of movements, so it will have to examine slightly more with respect to number of movements. However, it does beat Agent 6 and Agent 7 taking less number of total actions (movements + examinations) on an average.

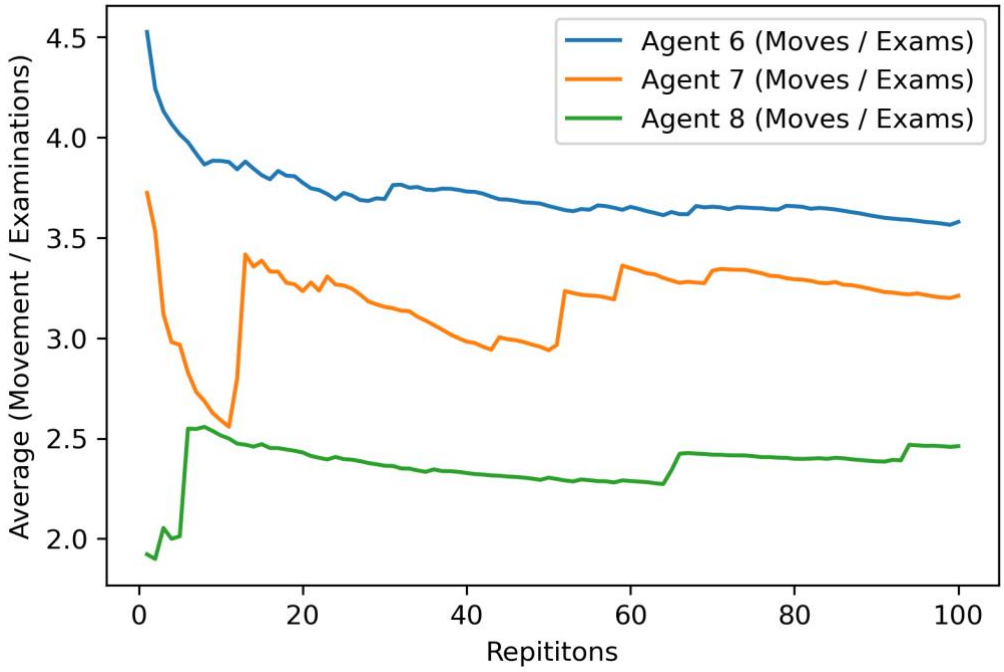
Next, we will look at terrain-wise (Flat, Hilly, Forest) plots between Agent 6, Agent 7 and Agent 8.

Terrain-wise plots between Agent 6, Agent 7 and Agent 8

Flat Terrain

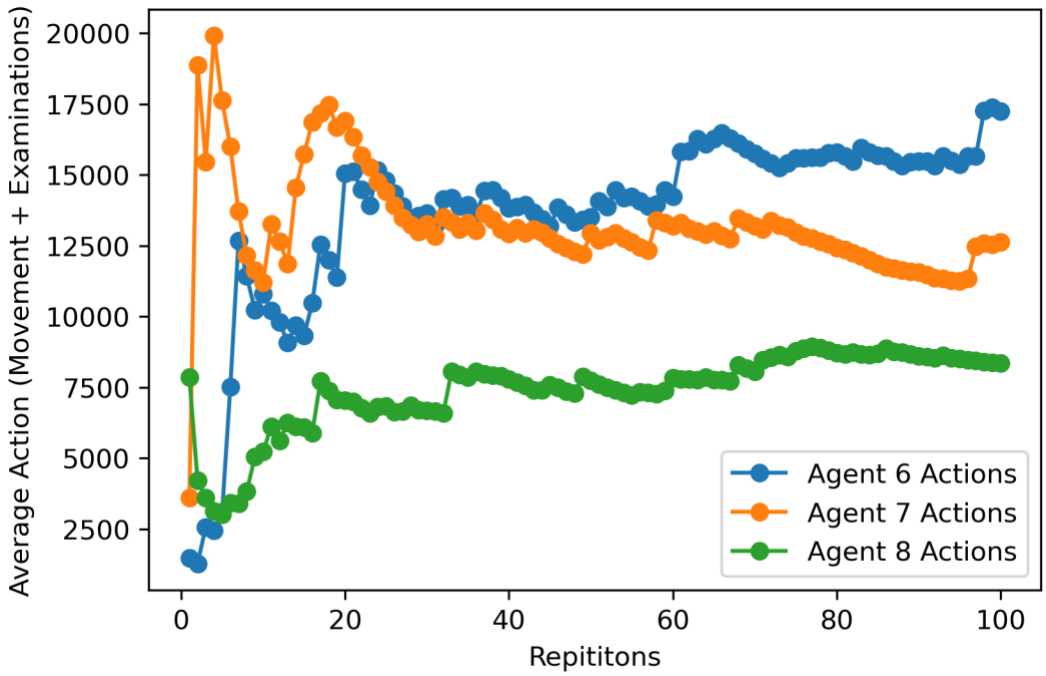


Plot 1.5 Average Action (Moves + Examines) for 100 repetitions on 50x50 grid (Target on Flat Terrain)

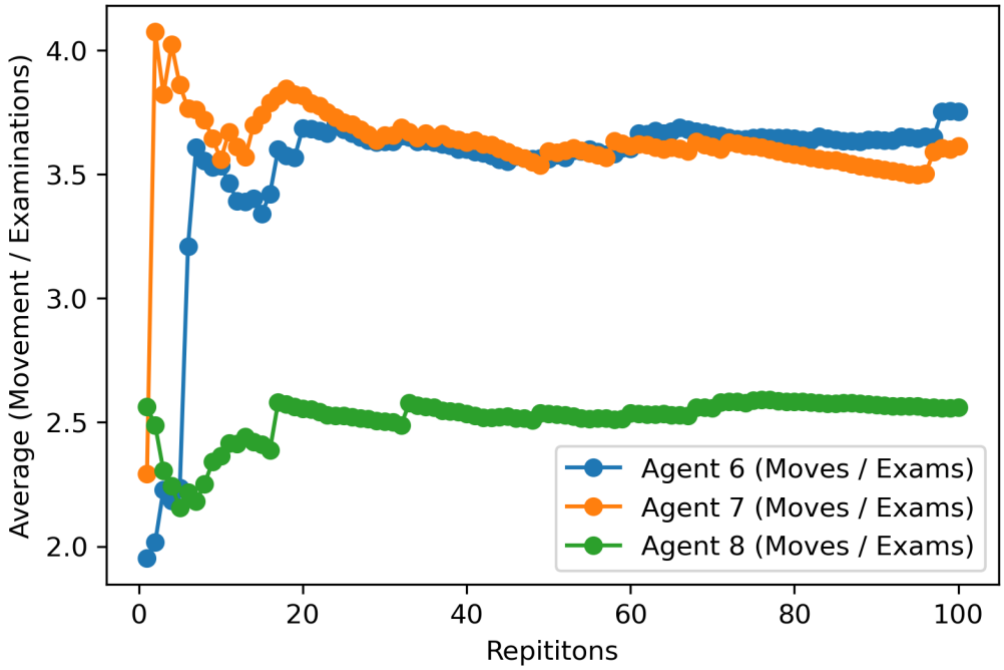


Plot 1.6 Average (Movement / Examinations) for 100 repetitions on 50x50 grid (Target on Flat Terrain)

Hilly Terrain

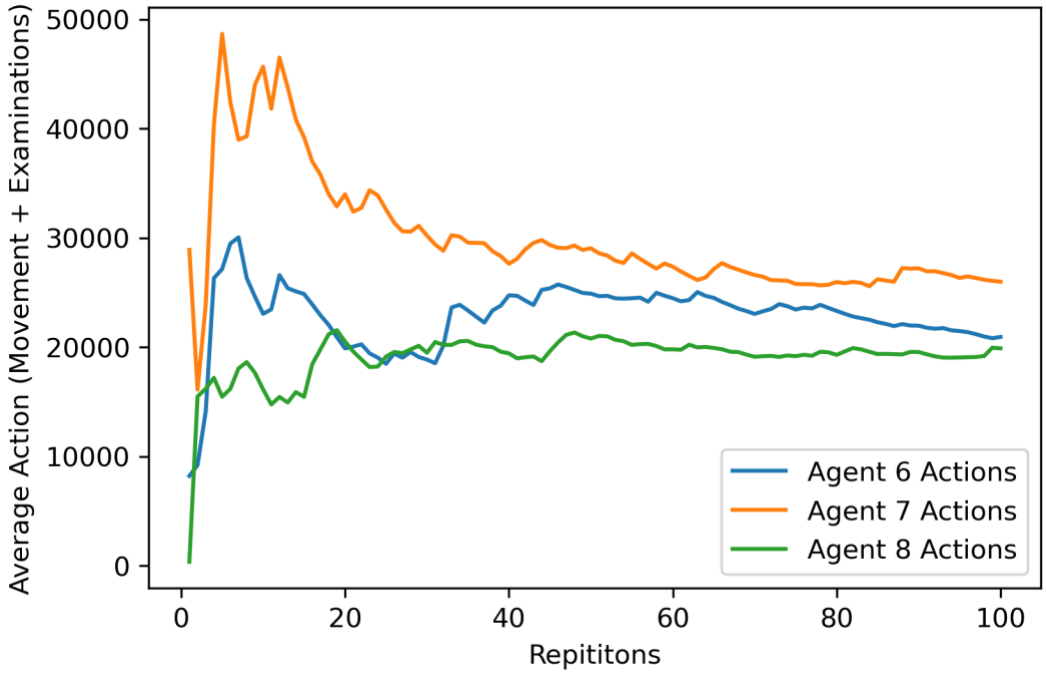


Plot 1.7 Average Action (Moves + Examines) for 100 repetitions on 50x50 grid (Target on Hilly Terrain)

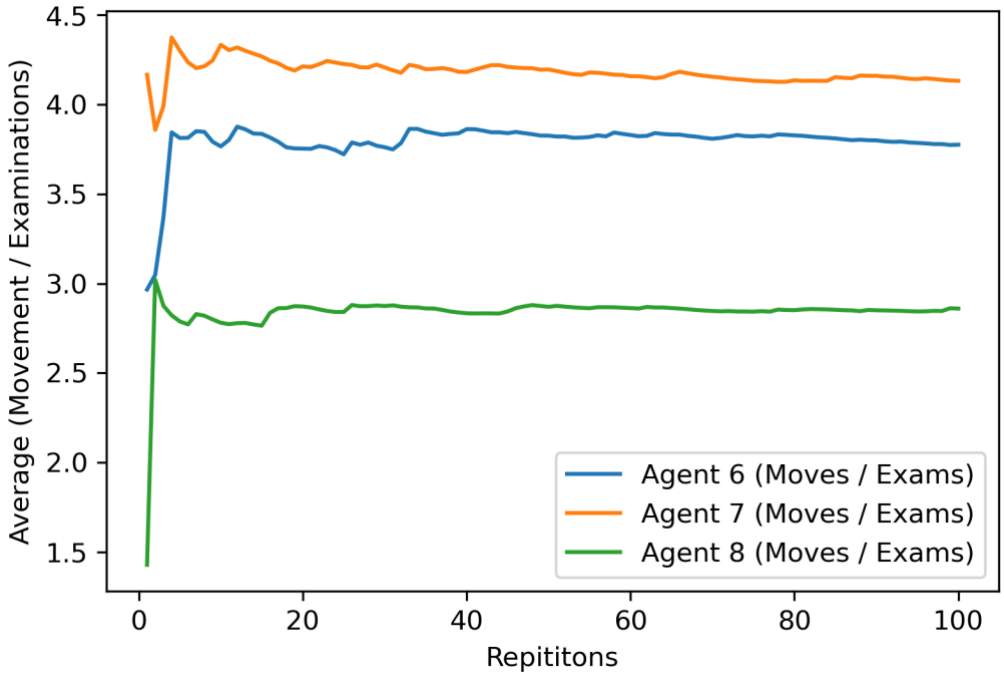


Plot 1.8 Average (Movement / Examinations) for 100 repetitions on 50x50 grid (Target on Hilly Terrain)

Forest Terrain



Plot 1.9 Average Action (Moves + Examines) for 100 repetitions on 50x50 grid (Target on Forest Terrain)



Plot 1.10 Average (Movement / Examinations) for 100 repetitions on 50x50 grid (Target on Forest Terrain)

At this point, on an average Agent 8 performs better as compared to Agent 6 and Agent 7 in terms on total actions performed.

7) Can we make our Agent 8 a bit more efficient in terms of number of actions that it currently takes ? Let us look at few points that might help us achieve that.

We can improve our Agent 8 even further by planning for n cells rather than just one, where n can be any number ranging from 2 to 101*101. Though we can see that taking n as a very high number also won't do us any good as the probabilities of each cell is changing at every time step.
By doing this we're not just worried about our immediate outcome but also about what comes after, which will be advantageous in the long run. Doing this will make sure that chasing a high utility cell we don't go into an area which has a low overall utility. If we'll have to move a lot of steps to get back to a good utility area then that will also waste a lot of moves and therefore increasing total actions.

For example, let's try this for 2 cells.
Let's say that our Agent is at a cell (x, y) at time t. Now we'll check all the cells for finding maximum utility. Here, instead of finding one target with maximum utility and blindly planning a path till there, we can let's say take the top 10.
Now using a single source shortest path algorithm like Dijkstra's (since all the distances are positive) we can make 5 pairs for these chosen 10 cells depending on the distances between them, so the closest cells can be a pair. Then, we iterate through these 5 pairs to find the pair with maximum total utility and plan our path till them. The cell with more utility in the pair can be explored first and the other one can be explored second. This way we're planning for the Agent's future as well in case it isn't able to find the target in the first cell which we feel can improve our algorithm by a lot.

Extra Credit (Agent 9)

As our target is moving every time step, we can't really say anything much about the probabilities of the cells. If we examine a particular cell and don't find the target there, we can't actually reduce the probability of it being there in this question as there is a possibility that the target moves to that cell in the next time step. As we are not being able to reduce any cell's probability that also means that we won't be able to increase the probability of other cells as the sum of the probabilities of all cells must add up to 1. Which implicitly means that our agent will have to move randomly until it's able to sense the target in its vicinity.

One thing we can do through all this is that if we find a blocked cell then we can surely change its probability of containing the target to 0, and therefore we can increase the probabilities of rest of the cells, so they add up to 1. At the same time, at any time stamp, if we can't actually sense the target then we know for sure that the target isn't present in one of the neighbours of our agent's current cell, so we can turn all of the neighbouring cell's probabilities to 0 for that particular time stamp as well.

In case at a particular time stamp, if we are able to sense the target in our vicinity, we'll take a random step to one of our neighbouring cells (as we don't actually know which neighbouring cell contains the target, we just know that one of them contains the target). This step will result in 2 possibilities, first one is that we lose the target and are not able to sense the target anymore as it has moved in the other direction to agent's. In that case we'll have our agent make the reverse move in hope to get back close to our target. Second one is that if we can still sense the target, then our agent will have to again move randomly as many cases can be possible.

Algorithm:

```
Def RunIntoBlockedCell()
        P(Blocked cell) = 0
        Recalculate P(Xj,Yj at Ti), where j = 1-101
Def TargetFinder()
P(Xj,Yj at T0), where j = 1-101 => 1/101*101
If target not sensed at Time stamp Ti:
        P(neighbours at Ti) = 0
        Recalculate P(Xj,Yj at Ti), where j = 1-101
        Move randomly
        If run into a blocked cell:
                RunIntoBlockedCell()
If target sensed at Time stamp Ti:
        Sum(P(neighbours at Ti)) = 1
        Probabilities of rest of the cells = 0
        Move a random step
                If run into a blocked cell:
                        RunIntoBlockedCell()
        If target not sensed at Ti+1:
                Make the reverse move of the move made at Ti
        Else:
                Move randomly
                If run into a blocked cell:
                        RunIntoBlockedCell()
```

Appendix

All 3 agents, Agent 6, Agent 7 and Agent 8 essentially follow same steps.

i) Identify an initial target based on target selection mechanism
ii) Plan a path to identified target
iii) Execute the planned path, while executing, keep sensing terrain types and checking for block cells, keep a check on if the target changed midway based on target selection mechanism for each cell.
   - Replan a path to identified target, if block encountered
   - Plan a path to new target if target changed midway
iv) Once at target cell, examine the cell. Repeat the process as necessary.

The difference for each of these agents is in their target selection algorithm.

General Data Structures used:

- maze – true grid
- d_grid – dictionary to represent discovered grid
- k_base – dictionary, knowledge base
- pb_model – dictionary to store probability of containing a target P$i, j(t)$
- terrain_id = {1:'flat', 2:'hilly', 3:'forest'} *(Blocked cell = -1)*
- pb_find_model - dictionary to store probability of finding a target P$x, y(t)$ for Agent 7
- utility_model – dictionary to store utility values $\sqcup x, y(t)$ for Agent 8
- visited – store visited cells, to help in probability calculation for identified terrain type
- # density of blocks (-1) p0 = 0.3
- false_terrain = { 'flat': 0.2, 'hilly': 0.5, 'forest': 0.8 }
- (tx, ty) – actual target, randomly generated with 1/3 probability on hilly, forest or flat
- (ax, ay) – initial agent's position, randomly generated with 1/3 probability on hilly, forest or flat terrain
- gridSolvable.py – Module to find if random grid is solvable or not

```
# function to randomly generate result for a examined cell, if it contains actual target
def get_result(p_false):
    n = np.random.choice([0, 1], p = [p_false, 1-p_false])
    return n
```

```
# function to examine target
def examine_target(pb_model, false_terrain, target, terrain):
    result = 0

    # check if originally target was here, if it was,
    # use false negative to get result and then update Px,y(containing target)
    if target[0] == tx and target[1] == ty:
        result = get_result(false_terrain[terrain])
        # if target found, set Px,y = 1 and all other Pi,j = 0
        if result == 1:
            pb_model[_format(target[0], target[1])] = 1
            for key in pb_model:
                if key != _format(target[0], target[1]):
                    pb_model[key] = 0
```

```
        return result
      # if target not found, update Px,y(containing target)
      else:
        pb_model[_format(target[0], target[1])] = pb_model[_format(target[0], target[1])] *
false_terrain[terrain]

   # if target didn't exist here in true maze, update Px,y(containing target)
   else:
     pb_model[_format(target[0], target[1])] = pb_model[_format(target[0], target[1])] * false_terrain[terrain]

   # update Pi,j(containing target)
   total = sum(pb_model.values())
   for key in pb_model:
     if key != _format(target[0], target[1]):
       pb_model[key] = pb_model[key]/total

   return result
```

Agent 6:

i) At any given time, the agent identifies the cell with the highest probability of containing the target, given what the agent has observed.

ii) If there are multiple cells that have equal (maximum) probability, ties should be broken by distance; if there are equidistant cells with equal (maximum) probability, ties should be broken uniformly at random.

```
# function to identify target cell for Agent 6
def target_xy(pb_model, agent):
   # get max p value
   _pMax = max(pb_model.values())

   # get all cells with p = _pMax
   pmax_cells = list()
   for key, value in pb_model.items():
     if value == _pMax:
       pmax_cells.append(key)

   # if multiple cells with p = _pMax, find least distant among those
   if len(pmax_cells) == 1:
     return pmax_cells[0], _pMax
   else:
     edist_cells = list()
     min_dist = sys.maxsize
     for cell in pmax_cells:
       x, y = map(int, cell.split(","))
       xy_dist = manhattan_distance((x,y), agent)
       if xy_dist < min_dist:
         edist_cells = [cell]
         min_dist = xy_dist
       elif xy_dist == min_dist:
         edist_cells.append(cell)

   # if multiple equi-distant cells, randomly select one cell as target
   if len(edist_cells) == 1:
     return edist_cells[0], _pMax
   else:
      return np.random.choice(edist_cells), _pMax
```

Agent 7:

i) At any given time, the agent identifies the cell with the highest probability of successfully finding the target, given what the agent has observed.

ii) If there are multiple cells that have equal (maximum) probability, ties should be broken by distance; if there are equidistant cells with equal (maximum) probability, ties should be broken uniformly at random.

```
# function to identify target cell for Agent 7
def ag7_target_xy(pb_model, agent):
    # initialise probability model for finding target
    pb_find_model = {}
    for key in pb_model:
        if key in visited:
            x, y = map(int, key.split(","))
            if d_grid[x, y] != -1:
                terrain = terrain_id[d_grid[x, y]]
                pb_find_model[key] = pb_model[key] * (1 - false_terrain[terrain])
            else:
                pb_find_model[key] = 0
        else:
            pb_find_model[key] = pb_model[key] * ((1-p0)/len(terrain_id)) * sum(false_terrain.values())

    # get max p value
    _pMax = max(pb_find_model.values())

    # get all cells with p = _pMax
    pmax_cells = list()
    for key, value in pb_find_model.items():
        if value == _pMax:
            pmax_cells.append(key)

    # if multiple cells with p = _pMax, find least distant among those
    if len(pmax_cells) == 1:
        return pmax_cells[0], _pMax
    else:
        edist_cells = list()
        min_dist = sys.maxsize
        for cell in pmax_cells:
            x, y = map(int, cell.split(","))
            xy_dist = manhattan_distance((x,y), agent)
            if xy_dist < min_dist:
                edist_cells = [cell]
                min_dist = xy_dist
            elif xy_dist == min_dist:
                edist_cells.append(cell)

    # if multiple equi-distant cells, randomly select one cell as target
    if len(edist_cells) == 1:
        return edist_cells[0], _pMax
    else:
        return np.random.choice(edist_cells), _pMax
```

Agent 8:

i) At any given time, the agent identifies the cell with the highest utility value for finding the target, given what the agent has observed.

ii) If there are multiple cells that have equal (maximum) utility, ties should be broken by mean highest utility cluster of each cell; if there are multiple cells with (maximum) mean utility cluster value, ties should be broken by distance; if there are equidistant cells with equal (maximum) mean utility cluster, ties should be broken uniformly at random.

```
# function to identify target cell for Agent 8
def ag8_target_xy(pb_model, agent):
    # initialise utility model for finding target
    utility_model = {}
    for key in pb_model:
        if key in visited:
            x, y = map(int, key.split(","))
            if d_grid[x, y] != -1:
                terrain = terrain_id[d_grid[x, y]]
                utility_model[key] = pb_model[key] * (1 - false_terrain[terrain])
            else:
                utility_model[key] = 0
        else:
            utility_model[key] = pb_model[key] * ((1-p0)/len(terrain_id)) * sum(false_terrain.values())

    # get maximum distance value from agent's current position
    # at any point one of the four corners of maze will be the farthest from agent
    _dMax = 0
    for corners in [(0,0), (0,dim_y-1), (dim_x-1,0), (dim_x-1,dim_y-1)]:
        temp_d = manhattan_distance(corners, agent)
        if temp_d > _dMax:
            _dMax = temp_d

    # final utility calculation = (prob/dist), to consider max prob and min dist
    for key in utility_model:
        x, y = map(int, key.split(","))
        # add 1 to normalized dist to handle cases where dist from agent = 0, current cell
        dist_xy = 1 + (manhattan_distance((x,y), agent)/_dMax)
        utility_model[key] = utility_model[key]/dist_xy

    # get max utility value
    _uMax = max(utility_model.values())

    # get all cells with _uMax value
    umax_cells = list()
    for key, value in utility_model.items():
        if value == _uMax:
            umax_cells.append(key)

    # if multiple cells with _uMax value, find one with most _uMax of neigbours
    if len(umax_cells) == 1:
        return umax_cells[0], _uMax
    else:
        eTutilMax_cells = list()
        etutilMax = 0
        for cell in umax_cells:
            cell_x, cell_y = map(int, cell.split(","))
```

```
        temp_Tutil = 0
        n_grid = get_neighbours(cell)
        if len(n_grid) == 0:
            discard_cell(pb_model, (cell_x, cell_y))
        else:
            for nb in n_grid:
                temp_Tutil = temp_Tutil + utility_model[nb]
            temp_Tutil = temp_Tutil/len(n_grid)
            if temp_Tutil > etutilMax:
                eTutilMax_cells = [cell]
                etutilMax = temp_Tutil
            elif temp_Tutil == etutilMax:
                eTutilMax_cells.append(cell)

    # if multiple cells with same _uMax of neighbours, find least distant one
    if len(eTutilMax_cells) == 1:
        return eTutilMax_cells[0], _uMax
    else:
        edist_cells = list()
        min_dist = sys.maxsize
        for cell in eTutilMax_cells:
            x, y = map(int, cell.split(","))
            xy_dist = manhattan_distance((x,y), agent)
            if xy_dist < min_dist:
                edist_cells = [cell]
                min_dist = xy_dist
            elif xy_dist == min_dist:
                edist_cells.append(cell)

    # if multiple equi-distant cells, randomly select one cell as target
    if len(edist_cells) == 1:
        return edist_cells[0], _uMax
    else:
        return np.random.choice(edist_cells), _uMax
```

Individual Contributions

Name (Net Id)	**Contributions**
Parth Goel (PG514)	• Several ideas and coordination on Agent 8 implementation • helped in report formation • helped in formation of Agent 7 algorithm • improvements of Agent 8
Prashant Kanth (PPK31)	• Initial probabilities calculation • Agent 6 implementation, • Ideas and coordination on implementation of Agent 8 • Report formation
Utkarsh Jha (UJ22)	• Agent 7 implementation • Ideas on Agent 8 implementation • Report representation ideas • Thoughts for Agent 9 • helped in report formation