Report

Project 3 - Probability in Grid world

Parvathi Mahesh Hedathri: pm850
Kavya Kavuri: kk1069
Harish Udhayakumar: hu33
November 14 2021

Base for all agents:

The grid world is represented as a numpy matrix. The values are assigned to the terrain in the following way.
Probability with which Blocked cells are initialized = p = 0.3
Probability with which any terrain type is assigned to unblocked cell after the previous operation = ⅓ for each terrain type

There are 5 types of matrices that represent the probabilistic gridworld. They are:

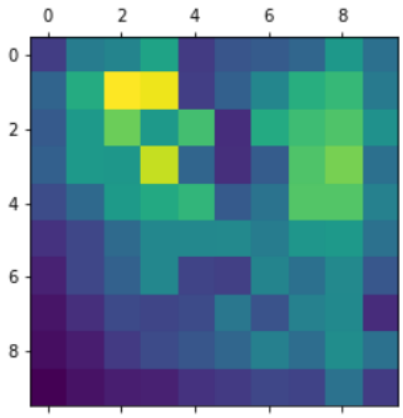


Fig. 1: Certainty matrix

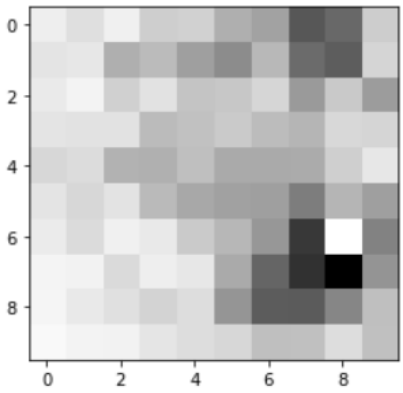


Fig. 2: Belief matrix

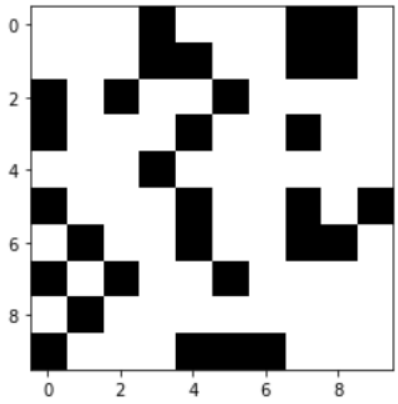


Fig. 3: Full Grid world

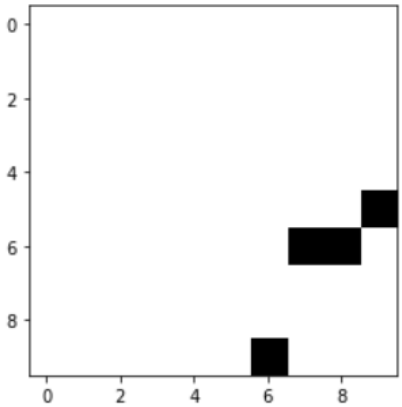


Fig. 4: Agent Grid world

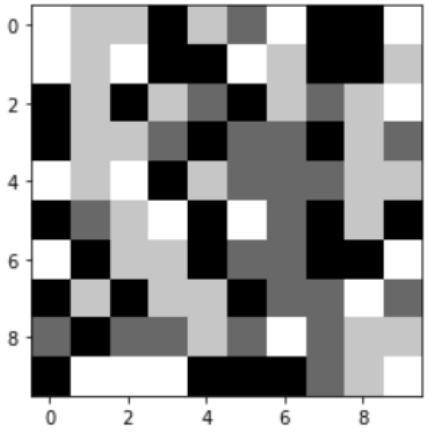


Fig. 5: Terrain matrix

Belief Matrix: The Belief matrix shows the probability of the cell containing the target. The darkest shade in Belief Matrix is where the probability of containing the target is the highest.
Value of a cell in Belief Matrix -> $P_{a,b}(t)$ = P(containing the target in (a,b) at time t)

Certainty Matrix: The Certainty matrix shows the probability of the target being found in a particular cell. The shade of Certainty Matrix determines the probability of successfully finding the target in that cell.
Value of a cell in Certainty Matrix -> $P_{a,b}(t)$ = P(successfully finding the target in (a,b) at time t)

Full gridworld: The Full gridworld matrix is the actual gridworld, showing blocked and free cells for the agent to traverse through the maze.

Terrain matrix: The Terrain matrix holds the information about the terrain type of individual cells in the maze. The color range from lightest to darkest corresponds to terrain type from flat to hilly to forest.
Values stored in Terrain Matrix:
FLAT -> 0
HILLY -> 1
FOREST -> 2
BLOCKED -> 3

Agent gridworld: The Agent gridworld is the subset of the terrain matrix. This stores what the agent finds in the process of searching for the target. Values in this matrix follow the same convention as the Terrain Matrix.

Question 1: Prior to any interaction with the environment, what is the probability of the target being in a given cell?

The target is equally probable to be in any of the cells in the grid world initially. Hence, it is given by:

$$P_{i,j}(t) = \frac{1}{Dimension * Dimension}$$

Question 2: Let $P_{i,j}(t)$ be the probability that cell (i,j) contains the target, given the observations collected up to time t. At time t+1, suppose you learn new information about cell (x, y). Depending on what information you learn, the probability for each cell needs to be updated. What should the new $P_{(i, j)}(t + 1)$ be for each cell (I, j) under the following circumstances:

- At time t + 1 you attempt to enter (x, y) and find it is blocked?

$P_{x,y}(t)$ is the initial probability of (x,y) containing the target..
Since (x, y) is blocked, $P_{x,y}(t+1) = 0$

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

- At time t + 1 you attempt to enter (x; y), find it unblocked, and learn its terrain type?

By Bayes' theorem, $P(A|B) = \frac{P(A)*P(B|A)}{P(B)}$

Initial assumption in the agent's grid world is that all the cells are unblocked. So if we enter a cell (x,y) and find it unblocked, it will not affect other cells because this doesn't give any information on the position of the target. So

$$P_{i,j}(t+1) = P_{i,j}(t) \text{ for all } (i,j)$$

- At time t + 1 you examine a cell (x; y) of terrain type flat, and fail to find the target?

Computing the probability when (i,j) ≠ (x,y) :

A: Target in (i,j)
B: Failed to find target at (x,y) , here (x,y) ≠ (i,j)

We want to find P(A|B)
P(A) = Pi,j(t) -> We already know the initial probabilities - (1)
P(B|A) = P(Failed to find target at (x ,y) | (i, j) contains target) = 1 - (2)
P(B) = P(Failed to find target at (x, y))

$P(B) = \Sigma_{\text{All (i,j) , (x,y)}}$ P(Failed to find target at (x,y) and target is in (i,j))
$= [\Sigma_{\text{All (i,j)}}$ P(Target in (i,j))*P(Failed to find target in (x,y)| Target in (i,j))]
+ P(Failed to find target in (x,y)| Target in (x,y))

$P(B) = [\Sigma_{\text{All (i,j)}} P_{i,j}(t)] + P_{x,y}(t)*FNR(x,y)$ - (3)

Using (1), (2), (3) we can calculate P(A|B) for all (i,j) where (i,j) ≠ (x,y)

$$P_{i,j}(t+1) = \frac{P_{i,j}(t)}{[\Sigma_{All\ i,j} P_{i,j}(t)] + P_{x,y}(t)*FNR(x,y)}$$

Computing the probability when (i,j) = (x,y) :

A: Target in (x,y)

B: Failed to find target at (x,y)

$P(A) = P_{x,y}(t)$

$P(B|A) = FNR(x,y)$

$P(B) = [\Sigma_{All\ (i,j)}\ P_{i,j}(t)] + P_{x,y}(t)*FNR(x,y)$ -- (from eq (3))

So, we can calculate P(A|B) for (x, y) using above equations.

$$P_{i,j}(t+1) = \frac{P_{x,y}(t)*FNR(x,y)}{[\Sigma_{All\ i,j} P_{i,j}(t)] + P_{x,y}(t)*FNR(x,y)}$$

Therefore, for a cell of terrain = flat,

$$P_{i,j}(t+1) = \frac{P_{x,y}(t)*0.2}{[\Sigma_{All\ i,j} P_{i,j}(t)] + P_{x,y}(t)*0.2}$$

- At time t + 1 you examine cell (x; y) of terrain type hilly, and fail to find the target?

$$P_{i,j}(t+1) = \frac{P_{x,y}(t)*0.5}{[\Sigma_{All\ i,j} P_{i,j}(t)] + P_{x,y}(t)*0.5}$$

- At time t + 1 you examine cell (x; y) of terrain type forest, and fail to find the target?

$$P_{i,j}(t+1) = \frac{P_{x,y}(t)*0.8}{[\Sigma_{All\ i,j} P_{i,j}(t)] + P_{x,y}(t)*0.8}$$

- At time t + 1 you examine cell (x; y) and find the target?

If we find target at (x,y):

$P_{x,y}(t+1) = 1$ all other $P_{i,j}(t+1) = 0$

Question 3: At time t, with probability Pi, j(t) of cell (I, j) containing the target, what is the probability of finding the target in cell (x; y):

- If (x; y) is hilly?

Px,y(t)*(1-0.5) = Px,y(t)*(0.5)

- If (x, y) is flat?

$P_{x,y}(t)*(1-0.2) = P_{x,y}(t)*(0.8)$

- If (x, y) is forest?

$P_{x,y}(t)*(1-0.8) = P_{x,y(}t)*(0.2)$

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

P(finding target in cell (x,y)) = Σ for all t P(finding target in cell (x,y) and terrain = t)
= Σ all t P(terrain = t)P(finding target in cell (x,y) | terrain = t)

= P(terrain = flat)P(finding target in (x,y) | (x,y) is flat) + P(terrain = hilly)P(finding target in (x,y) | (x,y) is hilly) + P(terrain = forest)P(finding target in (x,y) | (x,y) is forest)

= $0.7*1/3*P_{x,y}(t)*(1-0.2) + 0.7*1/3*P_{x,y}(t)*(1-0.5) + 0.7*1/3*P_{x,y}(t)*(1-0.8)$
= $0.35*P_{x,y}(t)$

Question 4: Implement Agent 6 and 7. For both agents, repeatedly run each agent on a variety of randomly generated boards (at constant dimension) to estimate the number of actions (movement + examinations) each agent needs on average to find the target. You will need to collect enough data to determine which of these agents is superior. Do you notice anything about the movement/examinations distribution for each agent? Note, boards where the target is unreachable from the initial agent position should be discarded.

In case of agent 6, the decision to examine the cell with the highest probability containing the target is made based on belief states after every iteration. Whereas, in the case of agent 7, the decision to examine the cell with highest probability is based on the certainty states at every iteration.

Note: The certainty state is updated only after the belief state is updated after every iteration.

On implementation and data collection, we found that agent8 > agent7 > agent6 in terms of superiority. In question 6 we have plotted graphs and analysed the collected data to show how agent8 is superior compared to other two agents.

On plotting the movement/examinations distribution, we noticed that agent8 has a lesser movement/examination ratio as compared to other agents. Also, Agent7 has a lesser movement/examination ratio compared to agent6.

Implementation of Agent6:
https://github.com/harish-udhayakumar/multi-terrain-probabilistic-sensing/tree/main/agent_6
Implementation of Agent7:
https://github.com/harish-udhayakumar/multi-terrain-probabilistic-sensing/tree/main/agent_7

Question 5: (20 points) Describe your algorithm, be explicit as to what decisions it is making, how, and why. How does the belief state ($P_{i,j}(t)$) enter into the decision making? Do you need to calculate anything new that you didn't already have available?

Algorithm Flowchart Agent6:

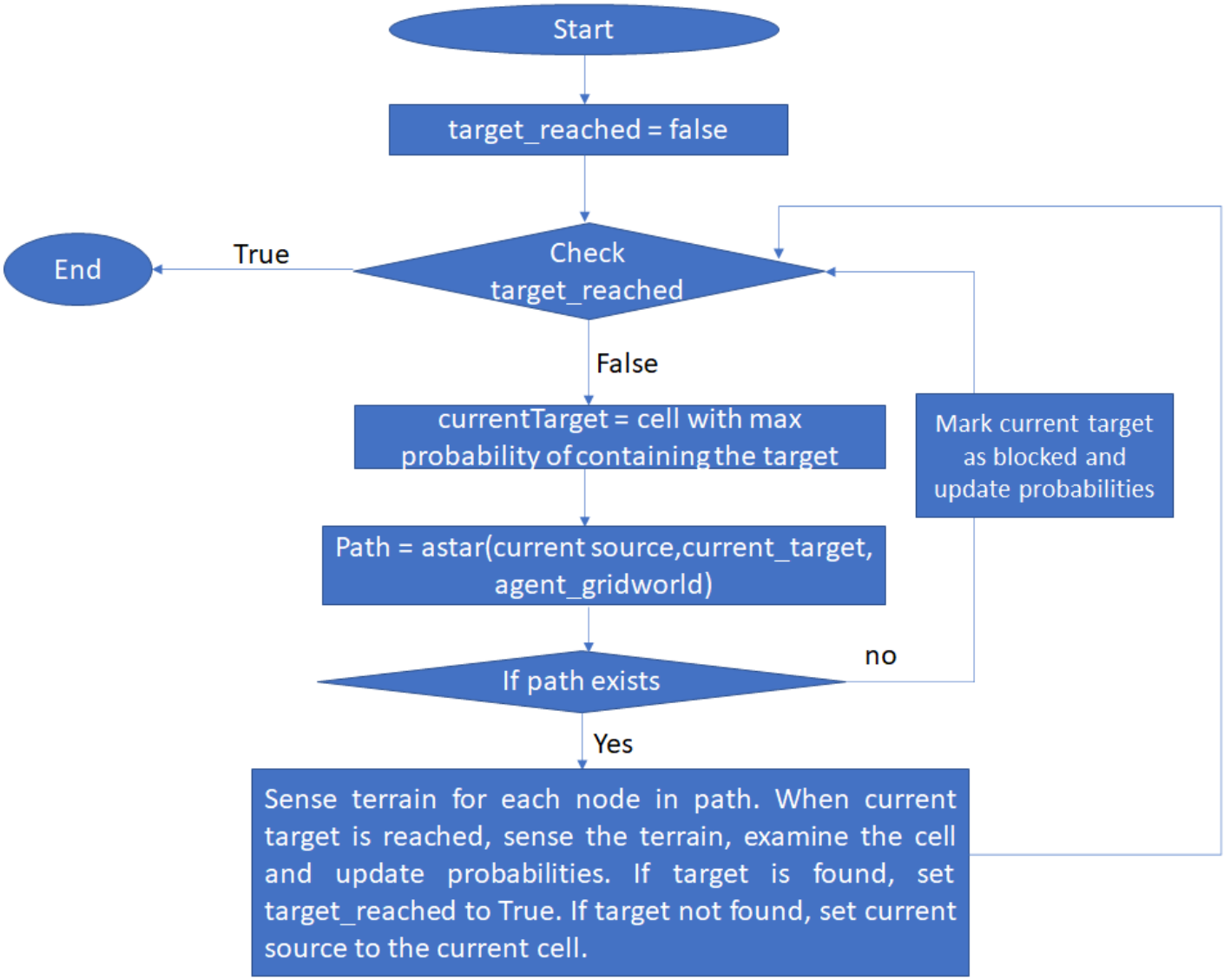


Fig. 6: Flowchart for Agent 6

Description:

Agent 6:

Agent 6 directly goes to the highest probable cell first. After a few iterations, the probabilities don't remain the same anymore. We observed that, if the forest cell is examined by the agent and it fails to find the target, it is decreased only a little bit as compared to others. So agent 6 visits all forest cells until they all become less than or equal to hilly cells. And it goes on in this way. Here we have observed that forest cells always have higher probabilities and next comes hilly cells and next is flat cells. This is the reason why Agent 6's movements are quite high. (Check Fig. 10 explanation for actual data).

Algorithm Flowchart Agent7:

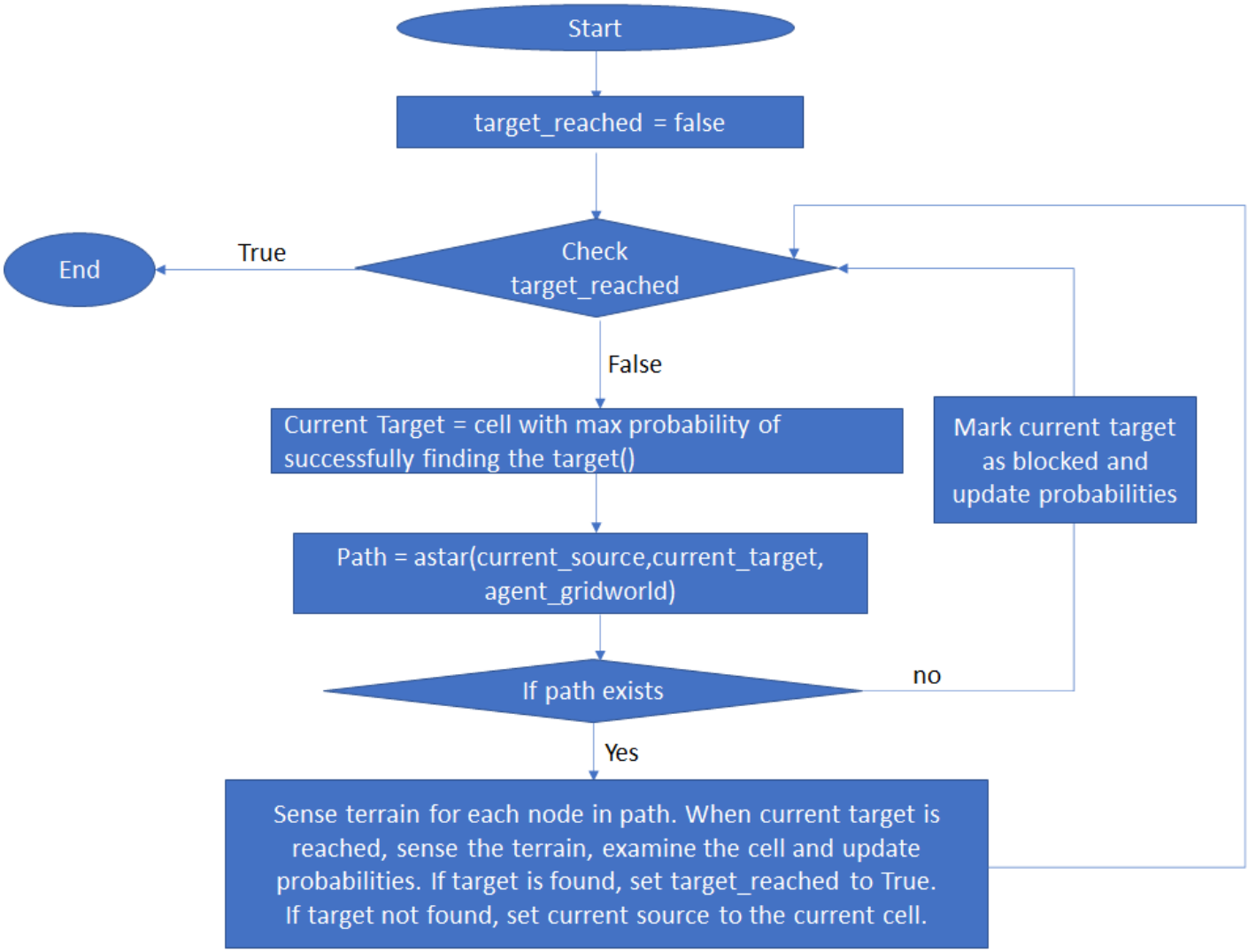


Fig. 7: Flowchart for Agent 7

Description:

Agent 7:

The Agent 7 works in a similar way as to Agent 6, except that it takes the max. probability cell from the certainty matrix. Here the main difference is that Agent 7 multiplies the probability of containing target with 0.2 for forest, 0.5 for hilly, 0.8 for flat and 0.35 for unvisited. So this means the beliefs are *punished* more for forest cells and they are *punished* less for flat cells. The hilly and unvisited cells lie in between. But we also need to note that the beliefs of forest cells are highest and the beliefs of flat cells are the lowest. Analyzing the values in the certainty matrix, we have concluded that the values in certainty values are kind of normalized so that all kinds of cells get a fair chance.

Question 6: (25 points) Implement Agent 8, run it sufficiently many times to give a valid comparison to Agents 6 and 7, and verify that Agent 8 is superior.

Algorithm Flowchart Agent 8:

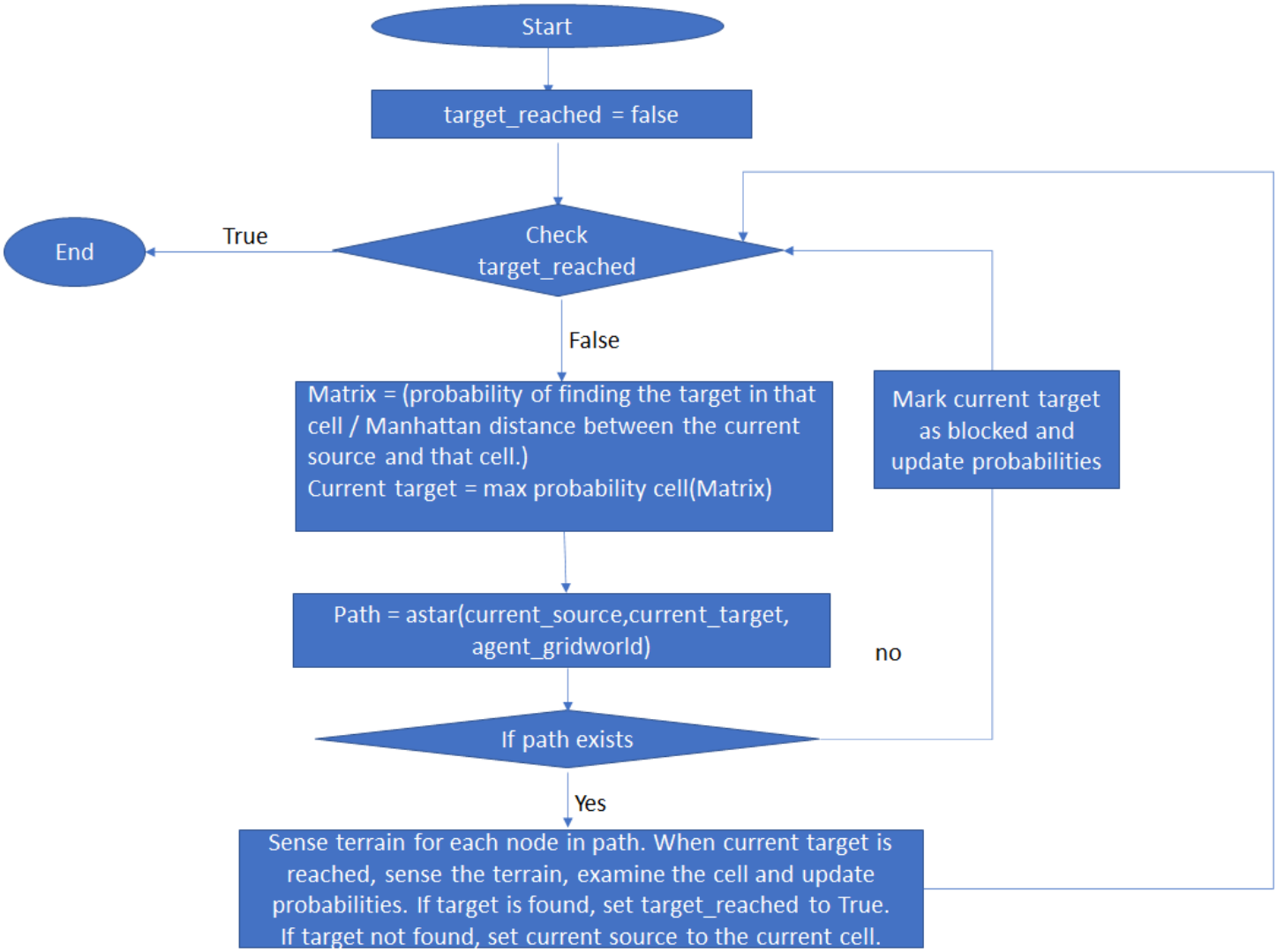


Fig. 8: Flowchart for Agent 8

Description:

Let cell(a,b) have maximum probability of finding the target but it is far away from the source, then agent8 will pick another cell(c,d) whose probability is not as high as cell(a,b) but is closer to the source. This way agent8 will perform lesser actions compared to other agents. To achieve this, we generate the new probabilities for each cell by dividing the probability of finding the target in that cell with the Manhattan distance between the current source cell and that cell. So, we are reducing the probability of finding the target in that cell if it is far away from source. The cell which is closer to source will have higher probability.

Implementation of Agent8:
https://github.com/harish-udhayakumar/multi-terrain-probabilistic-sensing/tree/main/agent_8

Below is the data obtained from executing all the agents with dimension of grid to be 50x50.

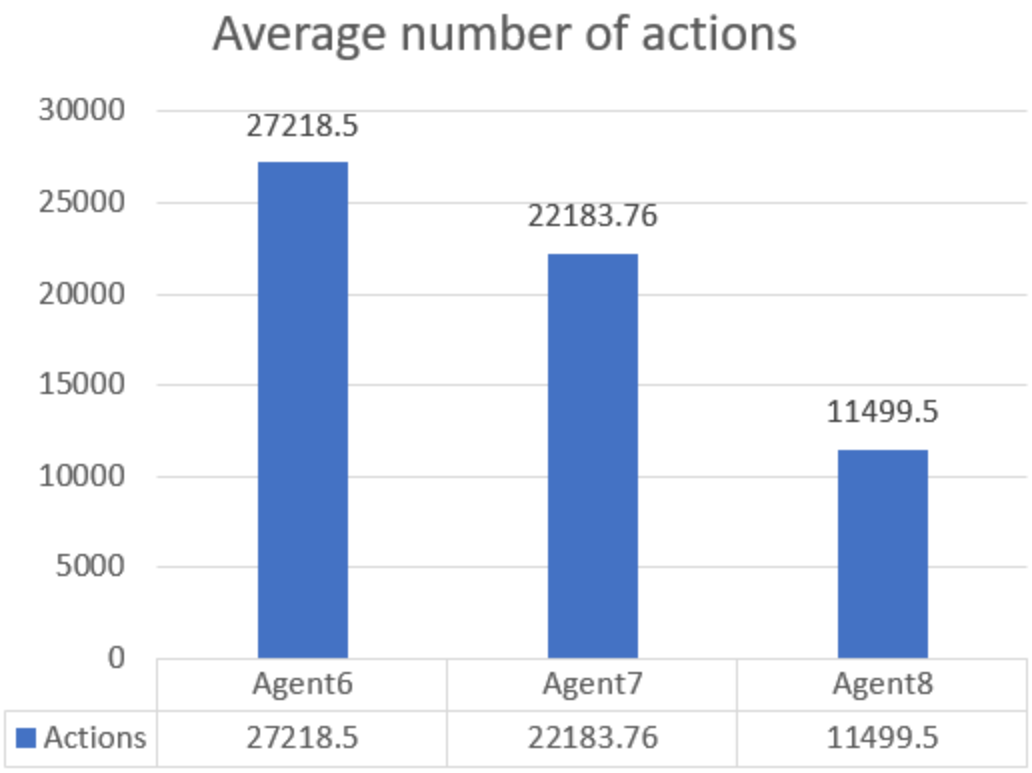


Fig. 9: Avg no. of actions by agents VS Agents

In Fig. 9, we can clearly see that Agent 7 is better than Agent 6 and Agent8 is performing much better than Agent 7 and Agent6. So Agent 8 > Agent 7 > Agent 6 in terms of performance.

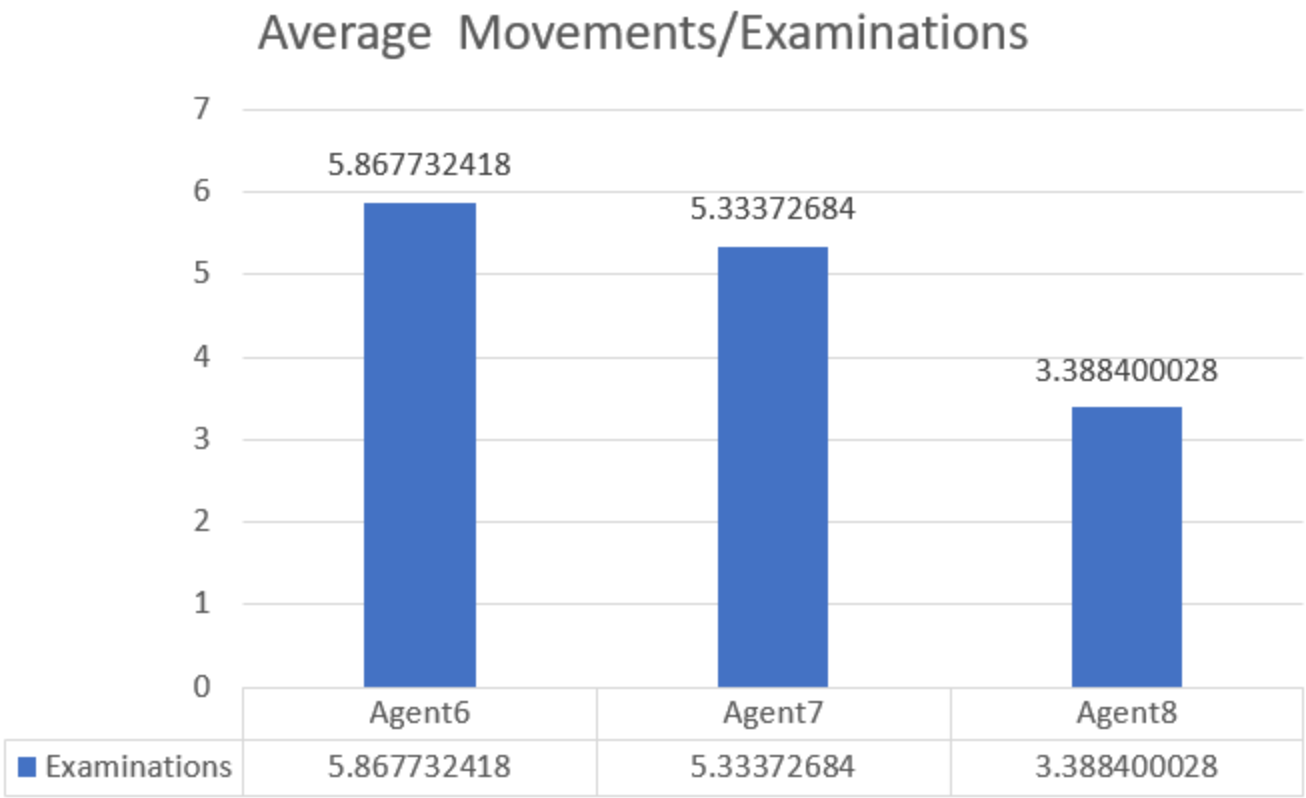


Fig. 10: Avg. of Movements/Examinations for Agents VS Agents

In Fig. 10, the number of movements/examinations of Agent8 < Agent 7 < Agent 6. Again the data is suggesting that Agent 8 is doing better than the other agents. This can be better understood with raw data of Agent 6,7,8. It is as follows:

Avg. of total no of movements by Agent 6: 24116.51
Avg. of total no of movements by Agent 7: 19327.08
Avg. of total no of movements by Agent 8: 8618.96

Avg. of Total no of examinations for Agent 6: 3102.75
Avg. of Total no of examinations for Agent 7: 2856.68
Avg. of Total no of examinations for Agent 8: 2880.54

So from the above data we can see that there is not much difference in the no. of examinations but there is a drastic difference in the total number of movements amongst the agents. Agent 8 is moving way less than the others, on an average.

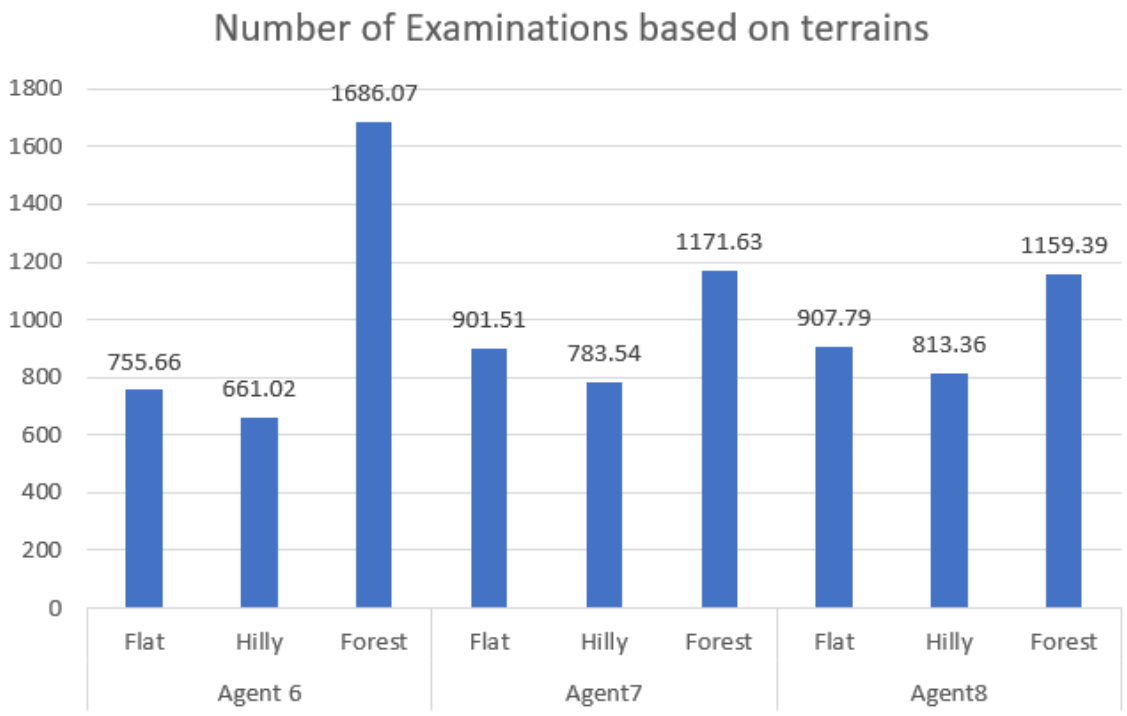


Fig. 11: Avg. no. of Examinations for all terrains VS Terrains of all Agents

Here we can see that the forest is examined the highest number of times for all agents because, when the target is not found in forest, the probability for forest cells are reduced by a small factor, so initially the agents visit the forest cells more. But based on the algorithm, agent 8 chooses better than agent 7, so the flat and hilly examinations of Agent 8 are more than Agent 7 and the same values for Agent 7 are in turn higher than Agent 6. Clearly the formulae used in Agents are better than the previous agents.

Question 7: (5 points) How could you improve Agent 8 even further? Be explicit as to what you could do, how, and what you would need.

We can consider more cells while taking the decision to plan the path. For example, consider two cells1:

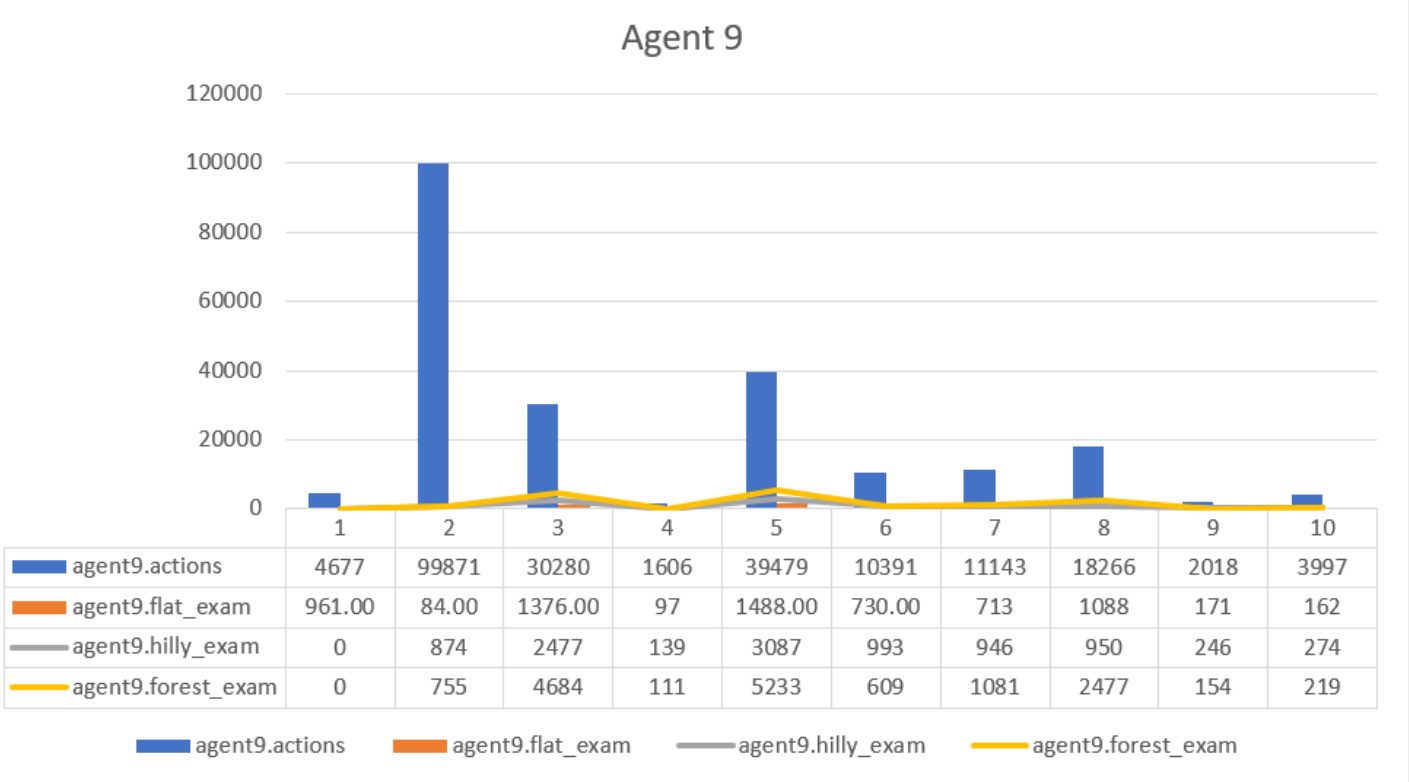
$cell_1$ = max{P(successfully finding the target)/(distance $_{\text{from current cell to current source}}$)}
$cell_2$ = max{P(successfully finding the target)}
Now the agent will go from the current source -> $cell_1$, examine($cell_1$) -> $cell_2$, examine($cell_2$) -> repeat the process. The agent stops whenever the target is found.

We are using $cell_2$ formula because it is clear that agent 7 performs better than agent6, we can certainly say it is good to use $cell_1$ formula because Agent 8 performs better than Agent 7. Here we are combining the best candidate from Agent 7 and best candidate from current Agent 8 for further improvement on Agent 8.

Bonus: (30 points) For Agent 9, we let the target move each time step, but only to one of its immediate neighbors (uniformly at random among unblocked neighbors). The Agent can additionally sense in each cell whether the target is in one of its 8 immediate neighbors, but not which one. Build an Agent that adapts to this moving target and extra partial sensing. How does it decide where to go next, and what to do? How is the belief state updated? Implement, and generate enough data for a good comparison.

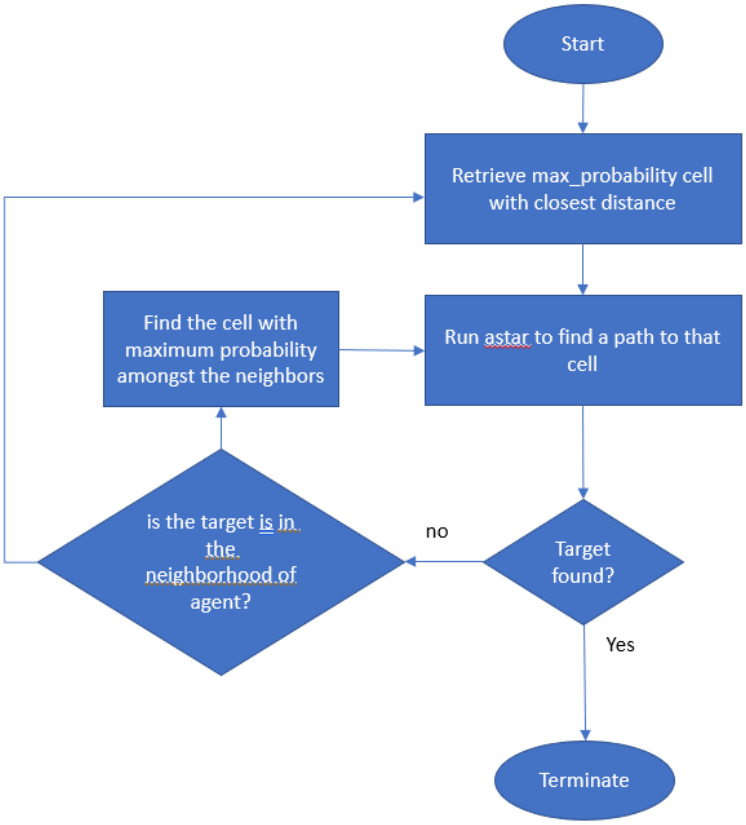
In this case, the target moves one step at a time to one of it's random unblocked neighboring cells.
Initially belief is divided equally among all the cells in the gridworld.

Updating Probabilities:
The probabilities are updated until the agent is able to find the target.
The agent examines if the current max probability cell has the target. If the target is present then the agent stops. If the target is not present, then depending on the sensing information, the agent takes the next decision.
If the target is sensed among neighbors, then it examines the cell with max probability among the neighbors.
If the target is not sensed, then the agent goes back to the first step, where it finds the max probability cell.
If the target is found, then terminate.
The certainty state of the cell for moving the target is calculated the same way as for the stationary target case.



	1	2	3	4	5	6	7	8	9	10
agent9.actions	4677	99871	30280	1606	39479	10391	11143	18266	2018	3997
agent9.flat_exam	961.00	84.00	1376.00	97	1488.00	730.00	713	1088	171	162
agent9.hilly_exam	0	874	2477	139	3087	993	946	950	246	274
agent9.forest_exam	0	755	4684	111	5233	609	1081	2477	154	219

Flowchart describing the function of Agent9



Implementation of Agent9:
https://github.com/harish-udhayakumar/multi-terrain-probabilistic-sensing/tree/main/agent_9

APPENDIX

Agent6:

```
def getMaxProbabilityCell(self, source):
        '''
       This function returns a cell with maximum probability of containing the target.
        '''
        mat = self.belief_matrix

       maxProb = mat.max()
       occurences = list(np.where(mat == maxProb))
       number_of_occurrences = len(occurences[0])
       node_to_return = source
       if number_of_occurrences > 1:
          dist = []
          previous_manhattan_distance = 9999
          for index in range(number_of_occurrences):

             _row = occurences[0][index]
             _col = occurences[1][index]

             target_node = (_row, _col)
             # find manhattan distance
             current_manhattan_distance = cityblock([source], target_node)
             dist.append([target_node,current_manhattan_distance])

             # code for finding manhattan distance between current node and the node in loop
             if current_manhattan_distance < previous_manhattan_distance:
                previous_manhattan_distance = current_manhattan_distance
          dist = [x for x in dist if x[1]==previous_manhattan_distance]
          node_to_return = dist[np.random.randint(0,len(dist))][0]

       else:
          row = occurences[0][0]
          col = occurences[1][0]
          node_to_return = (row, col)


       return node_to_return



def run_agent_6(agent6, maze):
    '''
    This function runs agent6.
    agent6:  object of type agent6 which contains utility functions to run agent6
    maze: object which contains utility matrices-terrain,full_grid_world,agent_gridworld
    '''
    global maze_size
    source = maze.source
    terrain_matrix = maze.terrain_matrix
    full_grid_world_matrix = maze.full_grid_world_matrix
    agent_gridworld = agent6.agent_gridworld

    target_reached = False
    while not target_reached:
       current_target = agent6.getMaxProbabilityCell(source)
```

```
initialize(maze_size,0.3)
path = astar(source,current_target, agent6.agent_gridworld_0_1)
if(len(path)==0):
    agent6.agent_gridworld_0_1[current_target[0],current_target[1]] = 1
    agent6.update_cells_when_xy_blocked(current_target[0], current_target[1], maze_size)
    agent6.agent_gridworld.itemset(current_target,3)
    maze.full_grid_world_matrix[current_target[0],current_target[1]] =1
    maze.terrain_matrix[current_target[0],current_target[1]] = 3
    continue

for i in range(len(path)):
    node = path[i]
    agent6.actions += 1
    if full_grid_world_matrix.item(node) == 0:
        terrain_type = terrain_matrix.item(node)
        agent6.agent_gridworld.itemset(node, terrain_type)
        if i == len(path)-1:
            agent6.actions += 1
            if(terrain_type == 0):
                agent6.flat_exam += 1
            elif(terrain_type == 1):
                agent6.hilly_exam += 1
            else:
                agent6.forest_exam += 1
            if(agent6.examine(node, terrain_type, maze)):
                print("found goal")
                target_reached = True
                break
            else:
                agent6.update_cells_when_xy_failed_with_some_terrainType(node[0], node[1], terrain_type, maze_size)
                source = path[i]
                break
    elif full_grid_world_matrix.item(node) == 1:
        agent6.actions += 2
        agent6.agent_gridworld_0_1[node[0],node[1]] = 1
        agent6.update_cells_when_xy_blocked(node[0], node[1], maze_size)
        agent6.agent_gridworld.itemset(node,3)
        source = path[i-1]
        break
```

Agent7:

```
def getMaxCertaintyCell(self, source):
    '''
    This function returns a cell with maximum probability of findingthe target.
    '''
    mat = self.certainty_matrix

    maxProb = mat.max()
    occurences = list(np.where(mat == maxProb))
    number_of_occurrences = len(occurences[0])
    node_to_return = source

    if number_of_occurrences > 1:
        dist = []
        previous_manhattan_distance = 9999
        for index in range(number_of_occurrences):
```

```
            _row = occurences[0][index]
            _col = occurences[1][index]

            target_node = (_row, _col)
            # find manhattan distance
            current_manhattan_distance = cityblock([source], target_node)
            dist.append([target_node,current_manhattan_distance])

            # code for finding manhattan distance between current node and the node in loop
            if current_manhattan_distance < previous_manhattan_distance:
                previous_manhattan_distance = current_manhattan_distance
        dist = [x for x in dist if x[1]==previous_manhattan_distance]
        node_to_return = dist[np.random.randint(0,len(dist))][0]
    else:
        row = occurences[0][0]
        col = occurences[1][0]
        node_to_return = (row, col)

    return node_to_return



def run_agent_7(agent7, maze):
    '''
    This function runs agent7.
    agent7:  object of type agent7 which contains utility functions to run agent7
    maze: object which contains utility matrices-terrain,full_grid_world,agent_gridworld
    '''

    global maze_size

    source = maze.source
    terrain_matrix = maze.terrain_matrix
    full_grid_world_matrix = maze.full_grid_world_matrix
    agent_gridworld = agent7.agent_gridworld

    target_reached = False
    while not target_reached:
        current_target = agent7.getMaxCertaintyCell(source)
        initialize(maze_size,0.3)
        path = astar(source,current_target, agent7.agent_gridworld_0_1)
        if(len(path)==0):
            agent7.agent_gridworld_0_1[current_target[0],current_target[1]] = 1
            agent7.update_beliefs_when_xy_blocked(current_target[0], current_target[1], maze_size)
            agent7.update_certainties(maze)

            agent7.agent_gridworld.itemset(current_target,3)
            maze.full_grid_world_matrix[current_target[0],current_target[1]] =1
            maze.terrain_matrix[current_target[0],current_target[1]] = 3

            continue

        for i in range(len(path)):
            node = path[i]
            agent7.actions += 1
            if full_grid_world_matrix.item(node) == 0:
                agent7.visited_matrix.itemset(node, True)
                terrain_type = terrain_matrix.item(node)
```

```
            agent7.agent_gridworld.itemset(node, terrain_type)
            if i == len(path)-1:
               agent7.actions += 1
               if(terrain_type == 0):
                  agent7.flat_exam += 1
               elif(terrain_type == 1):
                  agent7.hilly_exam += 1
               else:
                  agent7.forest_exam += 1
               if(agent7.examine(node, terrain_type, maze)):
                  print("found goal")
                  target_reached = True
                  break
               else:
                  #update probabiities when failed to be found and terrain type=terrain_matrix.item(node)
                  agent7.update_beliefs_when_xy_failed_with_some_terrainType(node[0], node[1], terrain_type, maze_size)
                  agent7.update_certainties(maze)
                  source = path[i]
                  break
         elif full_grid_world_matrix.item(node) == 1:
            agent7.actions += 2
            agent7.agent_gridworld_0_1[node[0],node[1]] = 1
            agent7.agent_gridworld.itemset(node,3)
            agent7.update_beliefs_when_xy_blocked(node[0], node[1], maze_size)
            agent7.update_certainties(maze)

            source = path[i-1]
            break
```

Agent8:

```
def getMaxProbabilityCell(self, source, maze_size):
      '''
      This function returns a cell with maximum probability of finding the target which is at a close distance to source.
       '''
      mat = np.full((maze_size, maze_size), 1/(maze_size*maze_size))
      for i in range(maze_size):
         for j in range(maze_size):
            manhattan_dist = cityblock([source], (i,j))
            if manhattan_dist!=0:
               mat[i][j] = self.certainty_matrix[i][j]/manhattan_dist
            else:
               mat[i][j] = self.certainty_matrix[i][j]

      maxProb = mat.max()
      occurences = list(np.where(mat == maxProb))
      number_of_occurrences = len(occurences[0])
      node_to_return = source
      if number_of_occurrences > 1:
         dist = []
         previous_manhattan_distance = 9999
         for index in range(number_of_occurrences):

            _row = occurences[0][index]
            _col = occurences[1][index]
```

```
            target_node = (_row, _col)
            # find manhattan distance
            current_manhattan_distance = cityblock([source], target_node)
            dist.append([target_node,current_manhattan_distance])

            # code for finding manhattan distance between current node and the node in loop
            if current_manhattan_distance < previous_manhattan_distance:
                previous_manhattan_distance = current_manhattan_distance
        dist = [x for x in dist if x[1]==previous_manhattan_distance]
        node_to_return = dist[np.random.randint(0,len(dist))][0]

    else:
        row = occurences[0][0]
        col = occurences[1][0]
        node_to_return = (row, col)


    return node_to_return



def run_agent_8(agent8, maze):
    '''
    This function runs agent8.
    agent8:  object of type agent8 which contains utility functions to run agent8
    maze: object which contains utility matrices-terrain,full_grid_world,agent_gridworld
    '''
    global maze_size
    source = maze.source
    terrain_matrix = maze.terrain_matrix
    full_grid_world_matrix = maze.full_grid_world_matrix
    agent_gridworld = agent8.agent_gridworld

    target_reached = False
    while not target_reached:
        current_target = agent8.getMaxProbabilityCell(source, maze_size)
        initialize(maze_size,0.3)
        path = astar(source,current_target, agent8.agent_gridworld_0_1)
        if(len(path)==0):
            agent8.agent_gridworld_0_1[current_target[0],current_target[1]] = 1
            agent8.update_cells_when_xy_blocked(current_target[0], current_target[1], maze_size)
            agent8.update_certainties(maze)

            agent8.agent_gridworld.itemset(current_target,3)
            maze.full_grid_world_matrix[current_target[0],current_target[1]] =1
            maze.terrain_matrix[current_target[0],current_target[1]] = 3
            continue

        for i in range(len(path)):
            node = path[i]
            agent8.actions += 1
            if full_grid_world_matrix.item(node) == 0:
                terrain_type = terrain_matrix.item(node)
                agent8.visited_matrix.itemset(node, True)
                agent8.agent_gridworld.itemset(node, terrain_type)
                if i == len(path)-1:
                    agent8.actions += 1
```

```
            if(terrain_type == 0):
               agent8.flat_exam += 1
            elif(terrain_type == 1):
               agent8.hilly_exam += 1
            else:
               agent8.forest_exam += 1
            if(agent8.examine(node, terrain_type, maze)):
               print("found goal")
               target_reached = True
               break
            else:
               #update probabiities when failed to be found and terrain type=terrain_matrix.item(node)
               agent8.update_cells_when_xy_failed_with_some_terrainType(node[0], node[1], terrain_type, maze_size)
               agent8.update_certainties(maze)
               source = path[i]
               break
         elif full_grid_world_matrix.item(node) == 1:
            agent8.actions += 2
            agent8.agent_gridworld_0_1[node[0],node[1]] = 1
            agent8.update_cells_when_xy_blocked(node[0], node[1], maze_size)
            agent8.update_certainties(maze)
            agent8.agent_gridworld.itemset(node,3)
            source = path[i-1]
            break
```

Agent9:

```
def getMaxProbabilityCell(self, source, maze):
      '''
      This function returns a cell with maximum probability of containing the target for agent9
       '''

      neighbor_list = self.getNeighbors(maze.maze_size, source[0], source[1], maze)

      maximum_probabilistic_neighbor = -1
      node_to_return = (0,0)
      if maze.target in neighbor_list:
         for i in neighbor_list:
            if maximum_probabilistic_neighbor < self.certainty_matrix.item(i):
               maximum_probabilistic_neighbor = self.certainty_matrix.item(i)
               node_to_return = i
         return node_to_return
      else:

         array_sum = np.sum(self.certainty_matrix)
         array_has_nan = np.isnan(array_sum)
         if array_has_nan:
            self.certainty_matrix = np.full((maze_size, maze_size), 1/(maze_size*maze_size))

         mat = self.certainty_matrix
         maxProb = mat.max()

         occurences = list(np.where(mat == maxProb))
         number_of_occurrences = len(occurences[0])
         node_to_return = source
         if number_of_occurrences > 1:
```

```
            dist = []
            previous_manhattan_distance = 9999
            for index in range(number_of_occurrences):

                _row = occurences[0][index]
                _col = occurences[1][index]

                target_node = (_row, _col)
                # find manhattan distance
                if target_node != source:
                    current_manhattan_distance = cityblock([source], target_node)
                    dist.append([target_node, current_manhattan_distance])

                    # code for finding manhattan distance between current node and the node in loop
                    if current_manhattan_distance < previous_manhattan_distance:
                        previous_manhattan_distance = current_manhattan_distance
            dist = [x for x in dist if x[1]==previous_manhattan_distance]
            node_to_return = dist[np.random.randint(0,len(dist))][0]

        else:
            row = occurences[0][0]
            col = occurences[1][0]
            node_to_return = (row, col)
        print("The next target to be visited is", node_to_return)
        return node_to_return


def run_agent_9(agent9, maze):
    '''
    This function runs agent9.
    agent9:  object of type agent9 which contains utility functions to run agent9
    maze: object which contains utility matrices-terrain,full_grid_world,agent_gridworld
    '''
    source = maze.source
    target = maze.target
    terrain_matrix = maze.terrain_matrix
    full_grid_world_matrix = maze.full_grid_world_matrix
    agent_gridworld = agent9.agent_gridworld
    valid = []

    # counters
    examining_count = 0
    number_of_agent_moves = 0
    number_of_target_moves = 0
    message = ""
    hilly_exam = 0
    flat_exam = 0
    forest_exam = 0

    if source == target:
        target_reached = True
        message = 'found goal'
        print(message)
        return number_of_target_moves, number_of_agent_moves, examining_count, flat_exam, hilly_exam, forest_exam, message
    else:
        target_reached = False

    while not target_reached:
```

```
initialize(maze_size,0.3)
current_target = agent9.getMaxProbabilityCell(source, maze)
path = astar(source,current_target, agent9.agent_gridworld_0_1)[0]
if(len(path)==0):
   agent9.agent_gridworld_0_1[current_target[0],current_target[1]] = 1
   agent9.update_beliefs_when_xy_blocked(current_target[0], current_target[1], maze_size)
   agent9.agent_gridworld.itemset(current_target,3)
   maze.full_grid_world_matrix[current_target[0],current_target[1]] =1
   maze.terrain_matrix[current_target[0],current_target[1]] = 3
   message = "There is no path to the target, try another grid"
   print(message)
   break

for i in range(len(path)):

   node = path[i]
   number_of_agent_moves += 1
   agent9.visited_matrix.itemset(node,1)

   #remove
   if number_of_agent_moves > 500:
      break

   if full_grid_world_matrix.item(node) == 0:
      terrain_type = terrain_matrix.item(node)
      agent9.agent_gridworld.itemset(node, terrain_type)
      if i == len(path)-1:
         examining_count += 1
         if terrain_type == 0:
            flat_exam += 1
         elif terrain_type == 1:
            hilly_exam += 1
         elif terrain_type == 2:
            forest_exam += 1
         if(agent9.examine(node, terrain_type, maze)):
            message = 'found goal'
            print(message)
            target_reached = True
            break
         else:
            #update probabiities when failed to be found and terrain type=terrain_matrix.item(node)
            current_target, type1, type2 = maze.target_moves(maze_size, node)
            number_of_target_moves += 1
            agent9.update_beliefs_when_xy_failed_with_some_terrainType(node[0], node[1], terrain_type, maze_size)
            agent9.update_certainties(maze)
            source = path[i]
            break
   elif full_grid_world_matrix.item(node) == 1:
      agent9.agent_gridworld_0_1[node[0],node[1]] = 1
      current_target, type1, type2 = maze.target_moves(maze_size, node)
      number_of_target_moves += 1
      print('node', node, ' is blocked and current_target is ', current_target)
      print("\n")
      agent9.update_beliefs_when_xy_blocked(node[0], node[1], maze_size)
      agent9.update_beliefs_when_xy_failed_with_some_terrainType(node[0], node[1], terrain_type, maze_size)
      agent9.update_certainties(maze)
      agent9.agent_gridworld.itemset(node,3)
      source = path[i-1]
```

```
            break
        source = current_target
    return number_of_target_moves, number_of_agent_moves, examining_count, flat_exam, hilly_exam, forest_exam, message
```