Names: Fionna Zhang, Katherine Lee, YuJu Ku
NetIds: fwz2, ksl103, yk544

CS 520 : Project 3 - Probability in Gridworld

Question 1

Prior to any interaction with the environment, we do not know where any of the blocked or unblocked cells are, so the probability of the target being in any of the cells is equal. The probability is $1/(\dim^2)$.

Question 2

Let Pi,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 Pi,j (t + 1) be for each cell (i, j) under the following circumstances:

1. At time t + 1 you attempt to enter (x, y) and find it is blocked?
   - $Px, y\ (t\ +\ 1)\ =\ 0$
     $Pi, j\ (t\ +\ 1)$
     $=\ P(cell\ (i,\ j)\ contains\ target\ |\ cell\ (x,\ y)\ are\ blocked)$
     $=\ Pi, j\ (t)\ +\ [\ (Px, y\ (t)\ -\ Px, y\ (t\ +\ 1))\ \times \frac{Pi,j\ (t)}{1 - Px,y(t)}\ ]$
   - Eg. In a 2x2 gridworld, suppose that we find cell (0, 1) is blocked, then the probability of cell (0, 0), (1, 0), (1, 1) becomes 0.14, 0.29, 0.57

(0, 0) 0.1	(0, 1) 0.3	→	(0, 0) 0.14	(0, 1) 0
(1, 0) 0.2	(1, 1) 0.4		(1, 0) 0.29	(1, 1) 0.57

2. At time t + 1 you attempt to enter (x, y), find it unblocked, and also learn its terrain type?
   - Until we examine the cell to check if the target is in the cell, we cannot change the probability of any cell (i, j) containing the target
3. At time t + 1 you examine cell (x, y) of terrain type flat, and fail to find the target?
   - If we fail to find the target in a terrain type flat cell, it is still possible that the cell contains the target, Px,y (t+1) reduces to Px,y (t) * 0.2, and all Pi,j (t+1) increases
     $Px, y\ (t\ +\ 1)\ =\ Px, y(t)\ \times\ 0.2$
     $Pi, j\ (t\ +\ 1)$
     $=\ P(cell\ (i,\ j)\ contains\ target\ |\ fail\ to\ find\ target\ in\ cell\ (x,\ y)\ with\ terrain\ type\ flat)$
     $=\ Pi, j\ (t)\ +\ [\ (Px, y\ (t)\ -\ Px, y\ (t\ +\ 1))\ \times \frac{Pi,j\ (t)}{1 - Px,y(t)}\ ]$
     $=\ Pi, j\ (t)\ +\ [\ 0.8\ \times\ Px, y\ (t)\ \times \frac{Pi,j\ (t)}{1 - Px,y(t)}\ ]$

- Eg. In a 2x2 gridworld, suppose that we examine and find cell (0, 1) is flat, then the probability of cell (0, 1) becomes 0.3 * 0.2 = 0.06, cell (0, 0), (1, 0), (1, 1) becomes 0.13, 0.27, 0.54

(0, 0) 0.1	(0, 1) 0.3	→	(0, 0) 0.13	(0, 1) 0.06
(1, 0) 0.2	(1, 1) 0.4		(1, 0) 0.27	(1, 1) 0.54

4. At time t + 1 you examine cell (x, y) of terrain type hilly, and fail to find the target?
   - If we fail to find the target in a terrain type flat cell, it is still possible that the cell contains the target, Px,y (t+1) reduces to Px,y (t) * 0.5, and all Pi,j (t+1) increases

     $Px, y\ (t\ +\ 1)\ =\ Px, y(t)$ × 0.5

     $Pi, j\ (t\ +\ 1)$

     $=\ P(cell\ (i,\ j)\ contains\ target\ |\ fail\ to\ find\ target\ in\ cell\ (x,\ y)\ with\ terrain\ type\ flat)$

     $=\ Pi, j\ (t)\ +\ [\ (Px, y\ (t)\ -\ Px, y\ (t\ +\ 1))$ × $\frac{Pi,j\ (t)}{1\ -\ Px,y(t)}\ ]$

     $=\ Pi, j\ (t)\ +\ [\ 0.5$ × $Px, y\ (t)$ × $\frac{Pi,j\ (t)}{1\ -\ Px,y(t)}\ ]$
   - Eg. In a 2x2 gridworld, suppose that we examine and find cell (0, 1) is hilly, then the probability of cell (0, 1) becomes 0.3 * 0.5 = 0.15, cell (0, 0), (1, 0), (1, 1) becomes 0.13, 0.27, 0.54

(0, 0) 0.1	(0, 1) 0.3	→	(0, 0) 0.12	(0, 1) 0.15
(1, 0) 0.2	(1, 1) 0.4		(1, 0) 0.24	(1, 1) 0.49

5. At time t + 1 you examine cell (x, y) of terrain type forest, and fail to find the target?
   - If we fail to find the target in a terrain type flat cell, it is still possible that the cell contains the target, Px,y (t+1) reduces to Px,y (t) * 0.8, and all Pi,j (t+1) increases

     $Pi, j\ (t\ +\ 1)$

     $=\ P(cell\ (i,\ j)\ contains\ target\ |\ fail\ to\ find\ target\ in\ cell\ (x,\ y)\ with\ terrain\ type\ flat)$

     $=\ Pi, j\ (t)\ +\ [\ (Px, y\ (t)\ -\ Px, y\ (t\ +\ 1))$ × $\frac{Pi,j\ (t)}{1\ -\ Px,y(t)}\ ]$

     $=\ Pi, j\ (t)\ +\ [\ 0.2$ × $Px, y\ (t)$ × $\frac{Pi,j\ (t)}{1\ -\ Px,y(t)}\ ]$
   - Eg. In a 2x2 gridworld, suppose that we examine and find cell (0, 1) is hilly, then the probability of cell (0, 1) becomes 0.3 * 0.8 = 0.24, cell (0, 0), (1, 0), (1, 1) becomes 0.13, 0.27, 0.54

(0, 0) 0.1	(0, 1) 0.3	→	(0, 0) 0.11	(0, 1) 0.24

(1, 0) 0.2	(1, 1) 0.4		(1, 0) 0.22	(1, 1) 0.43

6. At time t + 1 you examine cell (x, y) and find the target?
    - $Px, y\ (t\ +\ 1)\ =\ 1$
    - $Pi, j\ (t\ +\ 1)\ =\ 0$

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):

A = cell (x,y) containing the target
B = finding the target in cell (x,y)
P(B) = P(B | A)*P(A)/P(A | B)
= P(finding target in cell (x,y))
= P(finding the target in cell (x,y) | (x,y) contains target) * P((x,y) contains target) / P (cell (x,y) contains target | found target in cell (x,y))
= success rate * Pi,j(t) / 1
= (1 - false negative rate) * Pi,j(t)

1. If (x, y) is hilly?
    - $Px, y\ (t)$
      $=\ Pi, j(t)\ (1\ -\ false\ negative\ rate\ of\ hilly)$
      $=\ Pi, j(t)\ (1\ -\ 0.5)$
2. If (x, y) is flat?
    - $Px, y\ (t)$
      $=\ Pi, j(t)\ (1\ -\ false\ negative\ rate\ of\ flat)$
      $=\ Pi, j(t)\ (1\ -\ 0.2)$
3. If (x, y) is forest?
    - $Px, y\ (t)$
      $=\ Pi, j(t)\ (1\ -\ false\ negative\ rate\ of\ forest)$
      $=\ Pi, j(t)\ (1\ -\ 0.8)$
4. If (x, y) has never been visited?
    - $Px, y\ (t)$
      $=\ Pi, j(t)\ (\frac{(1-false\ negative\ rate\ of\ flat)+(1-false\ negative\ rate\ of\ hilly)+(1-false\ negative\ rate\ of\ forest)}{3})$
      $=\ Pi, j(t)\ (\frac{(1-0.2)+(1-0.5)+(1-0.8)}{3})$

Question 4

	Average Moves	**Average Examinati**	**Average Time**	**Worst Case**	**Worst Case**	**Worst Case Time**

	Made	ons	Taken	Moves Made	Examinations	Taken
Agent 6	2816.29	626.59	22.13491035	29171	5212	336.82652282714844
Agent 7	2609.04	588.03	19.47781681	17168	3332	206.10266137123108

On average, Agent 7 provided better results in all three performance areas (average moves made, average examinations, and average time taken). In contrast to Agent 6, who determined visited but unexamined cells have the same constant probability, Agent 7 determined each cell's probability with a more restricted rule according to the cells' terrain type. That is to say, based on the probability updated by Agent 6, Agent 7 multiplies each cell's probability of containing the target with their terrain type's success rate of finding the target as long as they were visited and their terrain types were learned. As for the unvisited cells, Agent 7 multiplies their probability of containing the target with an average success rate of the three terrain types. We can conclude that with identifying the probability of "successfully finding the target", Agent 7 had more limited choice regarding the goal node, which, on average, made it find the target more efficiently.

Question 5

Whilst working on Agents 6 and 7, we noticed that both agents would often return to the actual goal several times before being able to confirm it as the goal because it was getting false negatives. This was adding a significant amount of time and work to our agent.

In order to address this problem, each time Agent 8 reaches the cell where there is the highest probability of *containing* the target, it checks for the goal several times, in order to minimize the chance that it is being fooled by the false negatives as well as to avoid making extra moves and examinations into cells that do not contain the target and doing extra work.

We picked a 95% threshold for success in order to determine how many extra times we should check for the target. In order to calculate the number of times the agent needs to examine any given cell in order to have at least a 95% chance of finding the target, we model the situation as follows:

Let r be the false negative rate of the terrain of the given cell. We want to examine the cell N times to have a 95% chance of finding the target if it is in the given cell. The probability of finding the target on the first examination is (1-r). The probability of finding the target on the second examination is r(1-r), since we must first fail to find the target. The probability of finding the target on the third examination is $r^2(1-r)$, since we must first fail to find the target two examinations in a row. In general, the probability of finding the target on the i^{th} examination is $r^{i-1}(1-r)$.

Since our agent is done with the gridworld once we find the target, finding the target on examinations 1 through N are mutually exclusive events. We want the combined probability of finding the target on examinations 1 through N to be 0.95. We then get the equation:

$$(1 - r) + r(1 - r) + r^2(1 - r) + \ldots + r^{N-2}(1 - r) + r^{N-1}(1 - r) = 0.95$$

This is equivalent to:

$$(1 - r)(\sum_{i=1}^{N-1} r^i) = (1 - r)(\frac{1-r^N}{1-r}) = 1 - r^N = 0.95 \rightarrow 0.05 = r^N$$

Or

$$\frac{log(0.05)}{log(r)} = N$$

For each of the terrains:

- Flat (20% false negative)
    - N = log(0.05)/log(0.20) = 1.86 = 2 times
- Hilly (50% false negative)
    - N = log(0.05)/log(0.50) = 4.32 = 5 times
- Forest(80% false negative)
    - N = log(0.05)/log(0.80) = 13.43 = 14 times

If after checking the presumed goal cell several times, we still do not find the target, we adjust our weights accordingly.

Question 6

In order to test Agent 8's superiority against the other agents, we ran Agents 6, 7, and 8 on the same 100 20x20 grid worlds in order to compare their average number of moves made, average number of examinations and average amount of time taken to find the target (in seconds).

Without Incrementing Cells Examined:

100 Tests on 20x20 Dimension Grids	**Average Moves Made**	**Average Examinations**	**Average Time Taken (seconds)**	**Worst Case Moves Made**	**Worst Case Examinations**	**Worst Case Time Taken**
Agent 6	2969.02	654.34	71.66607825	29711	5409	1054.839644
Agent 7	2668.09	603.29	64.06245567	15382	2865	503.6894701
Agent 8	743.08	226.46	5.454888813	15202	2918	530.837497

We can see from the above table that Agent 8 performed significantly better than Agent 6 and Agent 7 in all three performance areas: average moves made, average examinations, and average amount of time taken to find the goal. Compared with Agent 6, Agent 8, on average, made about 0.25 the amount of moves, and 0.35 the number of examinations. Compared with Agent 7, Agent 8, on average made about .28 the number of moves and .37 the number of examinations.

Agent 8, once it reaches a cell where it thinks the target is contained, checks for the goal several times before moving on, relative to the type of terrain of the cell. Because of this, we were mixed on whether we thought that constituted extra examinations. On one hand, the agent, if it finds the target, simply returns the goal and if it does not, updates the probability the corresponding number of times to compensate, which could be seen as a single examination, with several checks. On the other hand, if we think of each examination as each time that the agent checks for the goal, then we should be incrementing each time we run the check, even if we have not moved at all. Because we felt conflicted as to whether this constituted extra examinations, we ran our tests again, this time incrementing each time the agent checked for the target cell, even if it hadn't moved.

Incrementing Cells Examined:

100 Tests on 20x20 Dimension Grids	**Average Moves Made**	**Average Examinations**	**Average Time Taken**	**Worst Case Moves Made**	**Worst Case Examinations**	**Worst Case Time Taken**
Agent 6	2792.96	604.26	70.4993589	19984	3960	763.448976
Agent 7	2150.27	494.8	46.4264193 2	13839	2537	454.2657349
Agent 8	1010.18	272.61	13.2192365 5	2648	15655	470.136807

The results show largely the same results as when we had not been incrementing each time the target was checked. Overall, Agent 8 performed significantly better in average moves made, average examinations, and average amount of time taken. Even with the extra examinations, average examinations for Agent 8 came in significantly lower than for Agent 6 and Agent 7. Compared with Agent 8, when examinations were not being counted several times, Agent 8 on average only did about 50 more examinations per grid.

In its worst case, in both trials, Agent 8 performed significantly better than Agent 6 and about as poorly as Agent 7. However, given that Agent 8 in both trials of 100 tests performed, on average, significantly better than Agent 6 and Agent 7 in all three areas (average moves made, average examinations, and average time taken), we feel confident in Agent 8's superior performance over the other agents.

Question 7

Agent 8 is based entirely off of Agent 6 and is looking specifically at the probability of finding the target *inside* the goal cell. Since at its worst case, Agent 8 currently performs about as well as Agent 7, it would be interesting to see whether Agent 8 looking at *successfully finding* the goal in the target cell as well as incrementally checking for the goal in every target cell would improve average performance of Agent 8 and make it better than Agent 7 in the worst case scenario.

Additionally, we think it would be interesting to combine the two probabilities of Agent 6 and 7: striking a balance in picking a new goal node between, which cells the agent thinks will *contain* the target, and which cells increase the probability of *successfully finding* the target. We do not think that this balance would be a 50/50 split, given that Agent 7 performs significantly better than Agent 6 both in worst case and in average case. However, we do still think that there is value in trying to determine which cell we think *contains* the target, especially as more and more cells are traversed, so we would need to either determine a threshold of cells traversed at which we would stop looking at which cells have a probability of successfully finding the target and switch over to looking for which cells contain the target or, we would need to find a way to determine which would be more valuable to us at a given time.

Bonus

Agent 9 implementation:

For this part, we did not take false negative rates of the three terrain types into consideration. Since terrain types are ignored, the cells that potentially contain the target have the same probability. Initially, we assume that every cell in the gridworld potentially contains the target, and has an identical probability. Since probabilities are all the same, Agent 9 picks the nearest cell as its goal and moves toward it. Every time Agent 9 moves a step, it senses whether the target is in one of its 8 immediate neighbors. If the agent senses the target in one of its 8 neighbors, the cells that potentially contain the target become only the immediate 8 neighbors. As a result, Agent 9 replans a path to one of the neighbors as its goal. However, since the target moves each time the agent moves, if Agent 9 senses nothing in its neighborhood once it has moved a step, the cells that potentially contain the target become the four cardinal directions of the cell that previously contained the target. Again, Agent 9 replans its path to the nearest cell that may contain the target. This process continues until Agent 9 either successfully finds the target.

Agent 9 Performance & Analysis:

100 Tests on 20x20 Dimension Grids	**Average Moves Made**	**Average Examinations**	**Average Time Taken**

Agent 6	2369.22	539.01	56.69159525
Agent 7	2538.7	575.28	61.24134943
Agent 8	756.32	226.49	5.84196981
Agent 9	2220.06	1163.11	2.090684092

In comparing Agent 9 with the previous agents, we noticed that performance wise, it does not do significantly better than Agents 6 or 7, and does significantly worse than Agent 8 in terms of average moves made and average examinations.

However, surprisingly, even though it moves about the same number of cells as Agents 6 and 7, and makes nearly double the examinations, it also runs significantly faster than any of the other agents. This is likely because Agent 9 can change its goal and replan the path to the goal whenever it senses the target in its neighborhood, meaning the agent can reach and examine goal nodes more easily and efficiently. However, it could also be because Agent 9 does not have the drawbacks of having to navigate false negatives, which hinders the other Agents, particularly Agents 6 and 7. Since it is not restricted by false negative rates, Agent 9 only has to consider the cells around it once it senses the target. In other words, the probabilities of the cells that are neither sensed to contain the target nor expanded by the cells determined to contain the target potentially falls to 0. Thus, this narrows down the path-planning of Agent 9 and helps it find the target faster. Though not superior in other performance areas (moves made and examinations), if you only cared about finding the goal node quickly, Agent 9 could be the superior agent.

Appendix

```
def agent_six(dim, gridworld, start, goal_actual):
    # generate gridworld to keep track of blocks found, terrain, etc.
    gridworld_copy = generate_gridworld(dim, 0)
    weight_dict = {}
    """
    weight_dict: {
        (x,y): {
            examined:
            weight:
        }
    }
    """
    pij_weight = 1
    goal_current = start
    curr = start
    prev = start
    highest = None
    solvable = astarwithstartgoal(dim, gridworld, start, goal_actual)[0]
    blocked_cells = 0
    cells_examined = 0
    moves_made = 0
    unreachable_nodes = []
    if solvable == True:
        path = astarwithstartgoal(dim, gridworld_copy, start, goal_current)[2]
        i = 0
```

```
        while (i <= len(path) - 1):
            # print("goal_current", goal_current)
            moves_made += 1
            prev = curr
            curr = path[i]
            curr_row = curr[0]
            curr_col = curr[1]
            # if curr != goal_current
            if (not curr == goal_current):
                # if curr == blocked
                if (gridworld[curr_row][curr_col] == -1):
                    # print("found block midlist")
                    blocked_cells += 1
                    gridworld_copy[curr_row][curr_col] = -1
                    # regenerate path from prev node
                    cells_examined += 1
                    goal_current = generate_new_goal(dim, curr, weight_dict, pij_weight,
gridworld_copy,
                                                     unreachable_nodes)
                    path = astarwithstartgoal(dim, gridworld_copy, prev, goal_current)[2]
                    goal_prev = goal_current
                    while (len(path) == 0):
                        # regenerate goal node
                        unreachable_nodes.append(goal_prev)
                        goal_current = generate_new_goal(dim, prev, weight_dict, pij_weight,
gridworld_copy,
                                                         unreachable_nodes)
                        # print("goal_current in midlist: ", goal_current)
                        path = astarwithstartgoal(dim, gridworld_copy, prev, goal_current)[2]
                        # print("path after midlist", path)
                        goal_prev = goal_current
                    i = 0
                    # update weight_dict
                    pij_weight, weight_dict = update_pij_weight(dim, Type="block",
pij_weight=pij_weight,
                                                                weight_dict=weight_dict,
currNode=curr)


                # if curr == flat
                elif (gridworld[curr_row][curr_col] == 1):
                    gridworld_copy[curr_row][curr_col] = 1
                    i += 1

                # if curr == hilly
                elif (gridworld[curr_row][curr_col] == 2):
                    gridworld_copy[curr_row][curr_col] = 2
                    i += 1

                # else if curr == forest
                else:
                    gridworld_copy[curr_row][curr_col] = 3
                    i += 1
            # if curr == goal_current
            else:
                # if curr is actually goal node
                cells_examined += 1
                if (curr == goal_actual):
                    if (gridworld[curr_row][curr_col] == 1):
                        gridworld_copy[curr_row][curr_col] = 1
                        if random.random() > 0.2:
                            # did not find false negative
                            # print("found it")
                            return (curr, moves_made, cells_examined)
```

```
                        # if curr == hilly
                    elif (gridworld[curr_row][curr_col] == 2):
                        gridworld_copy[curr_row][curr_col] = 2
                        if random.random() > 0.5:
                            # did not find false negative
                            # print("found it")
                            return (curr, moves_made, cells_examined)
                    # else if curr == forest

                    else:
                        gridworld_copy[curr_row][curr_col] = 3
                        if random.random() > 0.8:
                            # did not get false negative
                            # print("found it")
                            return (curr, moves_made, cells_examined)
                # print("Goal not found.")
                # if cell has been found previously
                if (curr in weight_dict.keys()):
                    # if curr == flat
                    weight_dict[curr]["examined"] += 1
                    if (gridworld[curr_row][curr_col] == 1):
                        pij_weight, weight_dict = update_pij_weight(dim, "flat", pij_weight,
weight_dict, curr)

                    # if curr == hilly
                    elif (gridworld[curr_row][curr_col] == 2):
                        pij_weight, weight_dict = update_pij_weight(dim, "hilly", pij_weight,
weight_dict, curr)

                    # else if curr == forest
                    else:
                        pij_weight, weight_dict = update_pij_weight(dim, "forest", pij_weight,
weight_dict, curr)

                else:
                    goal_is_block = False
                    if (gridworld[curr_row][curr_col] == -1):
                        # print("found block as goal")
                        blocked_cells += 1
                        gridworld_copy[curr_row][curr_col] = -1
                        pij_weight, weight_dict = update_pij_weight(dim, "block", pij_weight,
weight_dict, curr)
                        goal_is_block = True
                    else:

                        # get terrain type of cell
                        # if curr == flat
                        if (gridworld[curr_row][curr_col] == 1):
                            gridworld_copy[curr_row][curr_col] = 1
                            pij_weight, weight_dict = update_pij_weight(dim, "flat", pij_weight,
weight_dict, curr)

                        # if curr == hilly
                        elif (gridworld[curr_row][curr_col] == 2):
                            gridworld_copy[curr_row][curr_col] = 2
                            pij_weight, weight_dict = update_pij_weight(dim, "hilly", pij_weight,
weight_dict, curr)

                        # else if curr == forest
                        else:
                            gridworld_copy[curr_row][curr_col] = 3
                            pij_weight, weight_dict = update_pij_weight(dim, "forest", pij_weight,
weight_dict, curr)

                # generate new goal_current node
```

```
                if (goal_is_block):
                    goal_current = generate_new_goal(dim, prev, weight_dict, pij_weight,
gridworld_copy,
                                                     unreachable_nodes)
                    path = astarwithstartgoal(dim, gridworld_copy, prev, goal_current)[2]
                    goal_prev = goal_current
                    while (len(path) == 0):
                        # regenerate goal node
                        unreachable_nodes.append(goal_prev)
                        goal_current = generate_new_goal(dim, prev, weight_dict, pij_weight,
gridworld_copy,
                                                         unreachable_nodes)
                        path = astarwithstartgoal(dim, gridworld_copy, prev, goal_current)[2]
                        goal_prev = goal_current
                else:
                    goal_current = generate_new_goal(dim, curr, weight_dict, pij_weight,
gridworld_copy,
                                                     unreachable_nodes)
                    path = astarwithstartgoal(dim, gridworld_copy, prev, goal_current)[2]
                    goal_prev = goal_current
                    while (len(path) == 0):
                        # regenerate goal node
                        unreachable_nodes.append(goal_current)
                        goal_current = generate_new_goal(dim, curr, weight_dict, pij_weight,
gridworld_copy,
                                                         unreachable_nodes)
                        path = astarwithstartgoal(dim, gridworld_copy, curr, goal_current)[2]
                        goal_prev = goal_current
                i = 0
        # print("Exited loop without finding goal")
    else:
        print("Gridworld is not Solvable")
        return None


def agent_seven(dim, gridworld, start, goal_actual):
    # generate gridworld to keep track of blocks found, terrain, etc.
    gridworld_copy = generate_gridworld(dim, 0)
    weight_dict = {}
    """
    weight_dict: {
        (x,y): {
            examined:
            weight:
        }
    }
    """
    pij_weight = 1
    goal_current = start
    curr = start
    prev = start
    highest = None
    solvable = astarwithstartgoal(dim, gridworld, start, goal_actual)[0]
    blocked_cells = 0
    cells_examined = 0
    moves_made = 0
    unreachable_nodes = []
    if solvable == True:
        path = astarwithstartgoal(dim, gridworld_copy, start, goal_current)[2]
        i = 0
        while (i <= len(path) - 1):
            # print("goal_current", goal_current)
            moves_made += 1
            prev = curr
            curr = path[i]
```

```
            # print(curr)
            curr_row = curr[0]
            curr_col = curr[1]
            # if curr != goal_current
            if (not curr == goal_current):
                # if curr == blocked
                if (gridworld[curr_row][curr_col] == -1):
                    # print("found block midlist")
                    blocked_cells += 1
                    gridworld_copy[curr_row][curr_col] = -1
                    # regenerate path from prev node
                    cells_examined += 1
                    goal_current = generate_new_goal_seven(dim, curr, weight_dict, pij_weight,
gridworld_copy,
                                                           unreachable_nodes)
                    path = astarwithstartgoal(dim, gridworld_copy, prev, goal_current)[2]
                    goal_prev = goal_current
                    while (len(path) == 0):
                        # regenerate goal node
                        unreachable_nodes.append(goal_prev)
                        goal_current = generate_new_goal_seven(dim, prev, weight_dict, pij_weight,
gridworld_copy,
                                                               unreachable_nodes)
                        # print("goal_current in midlist: ", goal_current)
                        path = astarwithstartgoal(dim, gridworld_copy, prev, goal_current)[2]
                        # print("path after midlist", path)
                        goal_prev = goal_current
                    i = 0
                    # update weight_dict
                    pij_weight, weight_dict = update_pij_weight(dim, Type="block",
pij_weight=pij_weight,
                                                                weight_dict=weight_dict,
currNode=curr)


                # if curr == flat
                elif (gridworld[curr_row][curr_col] == 1):
                    gridworld_copy[curr_row][curr_col] = 1
                    i += 1

                # if curr == hilly
                elif (gridworld[curr_row][curr_col] == 2):
                    gridworld_copy[curr_row][curr_col] = 2
                    i += 1

                # else if curr == forest
                else:
                    gridworld_copy[curr_row][curr_col] = 3
                    i += 1
            # if curr == goal_current
            else:
                # if curr is actually goal node
                cells_examined += 1
                if (curr == goal_actual):
                    if (gridworld[curr_row][curr_col] == 1):
                        gridworld_copy[curr_row][curr_col] = 1
                        if random.random() > 0.2:
                            # did not find false negative
                            # print("found it")
                            return (curr, moves_made, cells_examined)

                        # if curr == hilly
                    elif (gridworld[curr_row][curr_col] == 2):
                        gridworld_copy[curr_row][curr_col] = 2
                        if random.random() > 0.5:
```

```
                        # did not find false negative
                        # print("found it")
                        return (curr, moves_made, cells_examined)
                # else if curr == forest

                else:
                    gridworld_copy[curr_row][curr_col] = 3
                    if random.random() > 0.8:
                        # did not get false negative
                        # print("found it")
                        return (curr, moves_made, cells_examined)
            # print("Goal not found.")
            # if cell has been found previously
            if (curr in weight_dict.keys()):
                # if curr == flat
                weight_dict[curr]["examined"] += 1
                if (gridworld[curr_row][curr_col] == 1):
                    pij_weight, weight_dict = update_pij_weight(dim, "flat", pij_weight,
weight_dict, curr)

                # if curr == hilly
                elif (gridworld[curr_row][curr_col] == 2):
                    pij_weight, weight_dict = update_pij_weight(dim, "hilly", pij_weight,
weight_dict, curr)

                # else if curr == forest
                else:
                    pij_weight, weight_dict = update_pij_weight(dim, "forest", pij_weight,
weight_dict, curr)

            else:
                goal_is_block = False
                if (gridworld[curr_row][curr_col] == -1):
                    # print("found block as goal")
                    blocked_cells += 1
                    gridworld_copy[curr_row][curr_col] = -1
                    pij_weight, weight_dict = update_pij_weight(dim, "block", pij_weight,
weight_dict, curr)
                    goal_is_block = True
                else:
                    # get terrain type of cell
                    # if curr == flat
                    if (gridworld[curr_row][curr_col] == 1):
                        gridworld_copy[curr_row][curr_col] = 1
                        pij_weight, weight_dict = update_pij_weight(dim, "flat", pij_weight,
weight_dict, curr)

                    # if curr == hilly
                    elif (gridworld[curr_row][curr_col] == 2):
                        gridworld_copy[curr_row][curr_col] = 2
                        pij_weight, weight_dict = update_pij_weight(dim, "hilly", pij_weight,
weight_dict, curr)

                    # else if curr == forest
                    else:
                        gridworld_copy[curr_row][curr_col] = 3
                        pij_weight, weight_dict = update_pij_weight(dim, "forest", pij_weight,
weight_dict, curr)

            # generate new goal_current node
            if (goal_is_block):
                goal_current = generate_new_goal_seven(dim, prev, weight_dict, pij_weight,
gridworld_copy,
                                                       unreachable_nodes)
                #print("goal is block goal current:", goal_current)
```

```
                    path = astarwithstartgoal(dim, gridworld_copy, prev, goal_current)[2]
                    goal_prev = goal_current
                    while (len(path) == 0):
                        # regenerate goal node
                        unreachable_nodes.append(goal_prev)
                        goal_current = generate_new_goal_seven(dim, prev, weight_dict, pij_weight,
gridworld_copy,
                                                               unreachable_nodes)
                        path = astarwithstartgoal(dim, gridworld_copy, prev, goal_current)[2]
                        goal_prev = goal_current
                else:
                    goal_current = generate_new_goal_seven(dim, curr, weight_dict, pij_weight,
gridworld_copy,
                                                           unreachable_nodes)
                    #print("goal is not block goal current:", goal_current)
                    path = astarwithstartgoal(dim, gridworld_copy, prev, goal_current)[2]
                    goal_prev = goal_current
                    while (len(path) == 0):
                        # regenerate goal node
                        unreachable_nodes.append(goal_current)
                        goal_current = generate_new_goal_seven(dim, curr, weight_dict, pij_weight,
gridworld_copy,
                                                               unreachable_nodes)
                        path = astarwithstartgoal(dim, gridworld_copy, curr, goal_current)[2]
                        goal_prev = goal_current
                i = 0
        # print("Exited loop without finding goal")
    else:
        print("Gridworld is not Solvable")
        return None

def agent_eight(dim, gridworld, start, goal_actual):
    # generate gridworld to keep track of blocks found, terrain, etc.
    gridworld_copy = generate_gridworld(dim, 0)
    weight_dict = {}
    """
    weight_dict: {
        (x,y): {
            examined:
            weight:
        }
    }
    """
    pij_weight = 1
    goal_current = start
    curr = start
    prev = start
    highest = None
    solvable = astarwithstartgoal(dim, gridworld, start, goal_actual)[0]
    blocked_cells = 0
    cells_examined = 0
    moves_made = 0
    unreachable_nodes = []
    if solvable == True:
        path = astarwithstartgoal(dim, gridworld_copy, start, goal_current)[2]
        i = 0
        while (i <= len(path) - 1):
            # print("goal_current", goal_current)
            moves_made += 1
            prev = curr
            curr = path[i]
            curr_row = curr[0]
            curr_col = curr[1]
            # if curr != goal_current
            if (not curr == goal_current):
```

```
                # if curr == blocked
                if (gridworld[curr_row][curr_col] == -1):
                    # print("found block midlist")
                    blocked_cells += 1
                    gridworld_copy[curr_row][curr_col] = -1
                    # regenerate path from prev node
                    cells_examined += 1
                    goal_current = generate_new_goal(dim, curr, weight_dict, pij_weight,
gridworld_copy,
                                                     unreachable_nodes)
                    path = astarwithstartgoal(dim, gridworld_copy, prev, goal_current)[2]
                    goal_prev = goal_current
                    while (len(path) == 0):
                        # regenerate goal node
                        unreachable_nodes.append(goal_prev)
                        goal_current = generate_new_goal(dim, prev, weight_dict, pij_weight,
gridworld_copy,
                                                         unreachable_nodes)
                        # print("goal_current in midlist: ", goal_current)
                        path = astarwithstartgoal(dim, gridworld_copy, prev, goal_current)[2]
                        # print("path after midlist", path)
                        goal_prev = goal_current
                    i = 0
                    # update weight_dict
                    pij_weight, weight_dict = update_pij_weight(dim, Type="block",
pij_weight=pij_weight,
                                                                weight_dict=weight_dict,
currNode=curr)


                # if curr == flat
                elif (gridworld[curr_row][curr_col] == 1):
                    gridworld_copy[curr_row][curr_col] = 1
                    i += 1

                # if curr == hilly
                elif (gridworld[curr_row][curr_col] == 2):
                    gridworld_copy[curr_row][curr_col] = 2
                    i += 1

                # else if curr == forest
                else:
                    gridworld_copy[curr_row][curr_col] = 3
                    i += 1
            # if curr == goal_current
            else:
                # if curr is actually goal node
                cells_examined += 1
                if (curr == goal_actual):
                    if (gridworld[curr_row][curr_col] == 1):
                        i = 0
                        num_tests = 2
                        while i < num_tests:
                            if i != 0:
                                cells_examined += 1
                            if random.random() > 0.2:
                                # print("found it")
                                return (curr, moves_made, cells_examined)
                            else:
                                pij_weight, weight_dict = update_pij_weight(dim, "flat", pij_weight,
weight_dict, curr)
                            i += 1
                        # if curr == hilly
                    elif (gridworld[curr_row][curr_col] == 2):
                        gridworld_copy[curr_row][curr_col] = 2
```

```
                        i = 0
                        num_tests = 5
                        while i < num_tests:
                            if i != 0:
                                cells_examined += 1
                            if random.random() > 0.5:
                                # print("found it")
                                return (curr, moves_made, cells_examined)
                            else:
                                pij_weight, weight_dict = update_pij_weight(dim, "hilly", pij_weight,
weight_dict, curr)
                            i += 1
                    else:
                        gridworld_copy[curr_row][curr_col] = 3
                        i = 0
                        num_tests = 14
                        while i < num_tests:
                            if i != 0:
                                cells_examined += 1
                            if random.random() > 0.8:
                                # print("found it")
                                return (curr, moves_made, cells_examined)
                            else:
                                pij_weight, weight_dict = update_pij_weight(dim, "forest",
pij_weight, weight_dict, curr)
                            i += 1
                # print("Goal not found.")
                # if cell has been found previously
                if (curr in weight_dict.keys()):
                    # if curr == flat
                    weight_dict[curr]["examined"] += 1
                    if (gridworld[curr_row][curr_col] == 1):
                        pij_weight, weight_dict = update_pij_weight(dim, "flat", pij_weight,
weight_dict, curr)

                    # if curr == hilly
                    elif (gridworld[curr_row][curr_col] == 2):
                        pij_weight, weight_dict = update_pij_weight(dim, "hilly", pij_weight,
weight_dict, curr)

                    # else if curr == forest
                    else:
                        pij_weight, weight_dict = update_pij_weight(dim, "forest", pij_weight,
weight_dict, curr)

                else:
                    goal_is_block = False
                    if (gridworld[curr_row][curr_col] == -1):
                        # print("found block as goal")
                        blocked_cells += 1
                        gridworld_copy[curr_row][curr_col] = -1
                        pij_weight, weight_dict = update_pij_weight(dim, "block", pij_weight,
weight_dict, curr)
                        goal_is_block = True
                    else:

                        # get terrain type of cell
                        # if curr == flat
                        if (gridworld[curr_row][curr_col] == 1):
                            gridworld_copy[curr_row][curr_col] = 1
                            pij_weight, weight_dict = update_pij_weight(dim, "flat", pij_weight,
weight_dict, curr)

                        # if curr == hilly
                        elif (gridworld[curr_row][curr_col] == 2):
```

```
                            gridworld_copy[curr_row][curr_col] = 2
                            pij_weight, weight_dict = update_pij_weight(dim, "hilly", pij_weight,
weight_dict, curr)

                        # else if curr == forest
                        else:
                            gridworld_copy[curr_row][curr_col] = 3
                            pij_weight, weight_dict = update_pij_weight(dim, "forest", pij_weight,
weight_dict, curr)

                # generate new goal_current node
                if (goal_is_block):
                    goal_current = generate_new_goal(dim, prev, weight_dict, pij_weight,
gridworld_copy,
                                                     unreachable_nodes)
                    path = astarwithstartgoal(dim, gridworld_copy, prev, goal_current)[2]
                    goal_prev = goal_current
                    while (len(path) == 0):
                        # regenerate goal node
                        unreachable_nodes.append(goal_prev)
                        goal_current = generate_new_goal(dim, prev, weight_dict, pij_weight,
gridworld_copy,
                                                         unreachable_nodes)
                        path = astarwithstartgoal(dim, gridworld_copy, prev, goal_current)[2]
                        goal_prev = goal_current
                else:
                    goal_current = generate_new_goal(dim, curr, weight_dict, pij_weight,
gridworld_copy,
                                                     unreachable_nodes)
                    path = astarwithstartgoal(dim, gridworld_copy, prev, goal_current)[2]
                    goal_prev = goal_current
                    while (len(path) == 0):
                        # regenerate goal node
                        unreachable_nodes.append(goal_current)
                        goal_current = generate_new_goal(dim, curr, weight_dict, pij_weight,
gridworld_copy,
                                                         unreachable_nodes)
                        path = astarwithstartgoal(dim, gridworld_copy, curr, goal_current)[2]
                        goal_prev = goal_current
                i = 0
        # print("Exited loop without finding goal")
    else:
        print("Gridworld is not Solvable")
        return None

def update_pij_weight(dim, Type, pij_weight, weight_dict, currNode):
        '''
        P = 1 / (dim**2)
        sum of all probabilities = sum of each cell's weight * P =  1
        total weight = dim**2
        curr_weight = current cell's weight
        total weight - weight = sum of all the other cell's weight
        1. If a cell is examined flat but do not find target, it's weight becomes weight*0.2
        2. If a cell is examined hilly but do not find target, it's weight becomes weight*0.5
        3. If a cell is examined forest but do not find target, it's weight becomes weight*0.8
        4. If a cell is blocked, it's weight is 0
        '''
        if not (currNode in weight_dict.keys()):
                weight_dict.update({currNode: {"examined": 1, "weight": pij_weight}})
        else:
                weight_dict[currNode]["examined"] += 1

        total_weight = dim**2
        curr_weight = weight_dict[currNode]["weight"]
        other_weight_sum = total_weight - curr_weight
```

```
        false_negative = 0

        # if terrain type is flat
        if Type == "flat":
                false_negative = 0.2
        # if terrain type is hilly
        elif Type == "hilly":
                false_negative = 0.5
        # if terrain type is forest
        elif Type == "forest":
                false_negative = 0.8
        elif Type == "block":
                false_negative = 0

        diff = curr_weight - (curr_weight*false_negative)

        for cell in weight_dict:
                if cell != currNode:
                        weight_dict[cell]["weight"] = weight_dict[cell]["weight"] +
(weight_dict[cell]["weight"]/other_weight_sum)*diff
                else:
                        weight_dict[cell]["weight"] *=  false_negative

        pij_weight = pij_weight + (pij_weight/other_weight_sum*diff)

        return pij_weight, weight_dict


def success_prob(node, weight_dict, gridworld_copy, pij_weight):
    '''
        P = 1 / (dim**2)
        pij = node's weight * P
        1. If a cell is learned flat, it's success probability is pij*(1-0.2)
        2. If a cell is learned hill, it's success probability is pij*(1-0.5)
        3. If a cell is learned forest, it's success probability is pij*(1-0.8)
        '''
    node_row = node[0]
    node_col = node[1]
    if node in weight_dict:
        node_weight = weight_dict[node]["weight"]
    else:
        node_weight = pij_weight

    terrain_type = gridworld_copy[node_row][node_col]
    # if terrain type is flat
    if terrain_type == 1:
        false_negative = 0.2
    # if terrain type is hilly
    elif terrain_type == 2:
        false_negative = 0.5
    # if terrain type is forest
    elif terrain_type == 3:
        false_negative = 0.8
    else:
        false_negative = 0

    # if cell is blocked
    if terrain_type == -1:
        success_prob = 0
    # if cell is unvisited, times the expected value of 1-FNR of the three terrain type
    elif terrain_type == 0:
        success_prob = node_weight * ((0.2 + 0.5 + 0.8) / 3)
    # if cell is visited and learned terrain type
    else:
        success_prob = node_weight * (1 - false_negative)
```

```
return success_prob
```