* Note: For more clarity of the code, please refer to the IPNB file first
* ***Note: In the latest version, the outputs are returned in the main function***
* Introduction:

Our project is in the field of making a game(pack\_man) and finding a path to food by means of the function of the agent and the environment:

Now first we have input file like demofile.txt:

The project has five parts. The first part is writing the agent function and the environment

The agent function is:

def agent(percept):

  randomAction=0

  if percept[2]=='a':

     percept[2]='-'

  if percept[2]=='f':

    return 20  #---------------------------->this number is my random selected numer that i selecte so we can set each all number or string or character choosed for this

  elif percept[2]=='\*':

    randomAction=setblockstatus(actions)

  else:

    randomAction=random.randint(1,4)

  actions.append(randomAction)

  return randomAction

the environment is:

def enviroment(action):

  setStatusPosition(action)

  current\_situation=original\_list[statusPos[0]][statusPos[1]]

  current\_percept=[statusPos[0],statusPos[1],current\_situation]

  historyOfPercepts.append(current\_percept)

  return current\_percept

As you can see, a fraction of other functions have been called inside these functions, and for the purpose of brevity of the code, I have included only the main function to clear up the ambiguities

* In the second part of the project, there is a random map creation, the code of which is given as a part of the project code:

# ----------------------------------------------------------->create RANDOM\_MAP

def generate\_map(m, n):

    # ----------------------Define characters for terrain and objects

    terrain\_chars = ['-', '\*']

    object\_chars = ['a', 'f']

    # ----------------------Generate empty map

    map\_data = [['-' for \_ in range(n)] for \_ in range(m)]

    # ----------------------Place walls around the perimeter

    for i in range(m):

        for j in range(n):

            if i == 0 or i == m - 1 or j == 0 or j == n - 1:

                map\_data[i][j] = '\*'

    # ----------------------Place random walls within the houses

    for i in range(1, m - 1):

        for j in range(1, n - 1):

            if map\_data[i][j] == '-':

                if random.random() < 0.2:  # Adjust the probability as needed

                    map\_data[i][j] = '\*'

    # -----------------------Place objects on the map

    for char in object\_chars:

        while True:

            row = random.randint(1, m - 2)

            col = random.randint(1, n - 2)

            if map\_data[row][col] == '-':

                map\_data[row][col] = char

                break

    return map\_data

    # -----------------find positions of current\_agent\_position'

def find\_character\_position(map\_data, character):

    for i, row in enumerate(map\_data):

        for j, char in enumerate(row):

            if char == character:

                return i, j

    return None, None

f=open("demofile.txt","a")

f.truncate(0)

f.close()

def write\_map\_to\_file(map\_data, filename):

    with open(filename, "r+") as f:

        f.write(f'{len(map\_data)},{len(map\_data[0])}\n')

        a\_row,a\_col = find\_character\_position(map\_data,'a')

        f.write(f'{a\_row},{a\_col}\n')

        f.write('\n')

        count=0

        for row in map\_data:

            count+=1

            if count!=len(map\_data):

              f.write(''.join(row) + '\n')

            else:

              f.write(''.join(row))

            # f.write(''.join(row) + '\n')

# Example usage:

print("now you can create random map by m and n that you enter them as size of map(m\*n):\n")

print("--> enter the m(max\_num of rows):")

m = input()

print("-->enter the n(max\_num of cols):")

n = input()

m=int(m)

n=int(n)

map\_data = generate\_map(m, n)

write\_map\_to\_file(map\_data, "demofile.txt")

f.close()

All these two pages of the code that you have seen, the generatemap function is related to the creation of the map randomly, and in the next functions, the map is written to file using the default characters of the project;like(‘\*’or ’a’ or ’f’….).

* In the third part: My project does not have a graphic environment😊
* In the fourth and fifth sections: we have a series of inputs and outputs, which I have brought to you with examples of given inputs and output of the code

Input 1:

10,10

2,2

\*\*\*\*\*\*\*\*\*\*

\*\*----\*\*-\*

\*-a\*---\*-\*

\*\*\*\*\*-\*\*\*\*

\*\*\*\*-----\*

\*\*\*-\*\*---\*

\*-\*\*\*\*-\*\*\*

\*--\*---f-\*

\*--\*--\*--\*

Output1:

Coordinates of the starting point :

(2, 2)

The coordinates of the place of food :

(7, 7)

the number of step to find path using 'bruth\_forts' mehthod :

289

The list consists of the selected coordinate and motion pairs:

[[(2, 2), 4, (2, 1), 3, (3, 1), 1, (2, 1), 3, (3, 1), 1, (2, 1), 2, (2, 2), 3, (3, 2), 1, (2, 2), 3, (3, 2), 1, (2, 2), 3, (3, 2), 1, (2, 2), 2, (2, 3), 4, (2, 2), 3, (3, 2), 1, (2, 2), 4, (2, 1), 3, (3, 1), 1, (2, 1), 4, (2, 0), 2, (2, 1), 2, (2, 2), 1, (1, 2), 3, (2, 2), 4, (2, 1), 3, (3, 1), 1, (2, 1), 4, (2, 0), 2, (2, 1), 3, (3, 1), 1, (2, 1), 4, (2, 0), 2, (2, 1), 1, (1, 1), 3, (2, 1), 2, (2, 2), 1, (1, 2), 3, (2, 2), 3, (3, 2), 1, (2, 2), 1, (1, 2), 4, (1, 1), 2, (1, 2), 1, (0, 2), 3, (1, 2), 4, (1, 1), 2, (1, 2), 2, (1, 3), 4, (1, 2), 2, (1, 3), 2, (1, 4), 4, (1, 3), 2, (1, 4), 4, (1, 3), 3, (2, 3), 1, (1, 3), 1, (0, 3), 3, (1, 3), 1, (0, 3), 3, (1, 3), 4, (1, 2), 1, (0, 2), 3, (1, 2), 3, (2, 2), 3, (3, 2), 1, (2, 2), 2, (2, 3), 4, (2, 2), 1, (1, 2), 4, (1, 1), 2, (1, 2), 4, (1, 1), 2, (1, 2), 1, (0, 2), 3, (1, 2), 4, (1, 1), 2, (1, 2), 2, (1, 3), 1, (0, 3), 3, (1, 3), 2, (1, 4), 4, (1, 3), 4, (1, 2), 4, (1, 1), 2, (1, 2), 4, (1, 1), 2, (1, 2), 3, (2, 2), 3, (3, 2), 1, (2, 2), 1, (1, 2), 2, (1, 3), 4, (1, 2), 2, (1, 3), 3, (2, 3), 1, (1, 3), 3, (2, 3), 1, (1, 3), 2, (1, 4), 2, (1, 5), 2, (1, 6), 4, (1, 5), 3, (2, 5), 1, (1, 5), 3, (2, 5), 3, (3, 5), 1, (2, 5), 3, (3, 5), 2, (3, 6), 4, (3, 5), 2, (3, 6), 4, (3, 5), 4, (3, 4), 2, (3, 5), 3, (4, 5), 2, (4, 6), 4, (4, 5), 2, (4, 6), 4, (4, 5), 3, (5, 5), 1, (4, 5), 2, (4, 6), 2, (4, 7), 2, (4, 8), 2, (4, 9), 4, (4, 8), 3, (5, 8), 1, (4, 8), 3, (5, 8), 2, (5, 9), 4, (5, 8), 3, (6, 8), 1, (5, 8), 2, (5, 9), 4, (5, 8), 1, (4, 8), 1, (3, 8), 3, (4, 8), 4, (4, 7), 3, (5, 7), 1, (4, 7), 4, (4, 6), 4, (4, 5), 4, (4, 4), 4, (4, 3), 2, (4, 4), 2, (4, 5), 4, (4, 4), 1, (3, 4), 3, (4, 4), 3, (5, 4), 1, (4, 4), 2, (4, 5), 2, (4, 6), 1, (3, 6), 3, (4, 6), 2, (4, 7), 4, (4, 6), 3, (5, 6), 1, (4, 6), 3, (5, 6), 1, (4, 6), 2, (4, 7), 2, (4, 8), 1, (3, 8), 3, (4, 8), 2, (4, 9), 4, (4, 8), 1, (3, 8), 3, (4, 8), 3, (5, 8), 1, (4, 8), 3, (5, 8), 1, (4, 8), 3, (5, 8), 1, (4, 8), 1, (3, 8), 3, (4, 8), 3, (5, 8), 1, (4, 8), 3, (5, 8), 1, (4, 8), 2, (4, 9), 4, (4, 8), 3, (5, 8), 2, (5, 9), 4, (5, 8), 1, (4, 8), 2, (4, 9), 4, (4, 8), 1, (3, 8), 3, (4, 8), 2, (4, 9), 4, (4, 8), 4, (4, 7), 1, (3, 7), 3, (4, 7), 1, (3, 7), 3, (4, 7), 3, (5, 7), 4, (5, 6), 2, (5, 7), 1, (4, 7), 3, (5, 7), 2, (5, 8), 2, (5, 9), 4, (5, 8), 2, (5, 9), 4, (5, 8), 4, (5, 7), 4, (5, 6), 4, (5, 5), 2, (5, 6), 1, (4, 6), 1, (3, 6), 3, (4, 6), 3, (5, 6), 2, (5, 7), 4, (5, 6), 2, (5, 7), 2, (5, 8), 2, (5, 9), 4, (5, 8), 4, (5, 7), 2, (5, 8), 1, (4, 8), 2, (4, 9), 4, (4, 8), 1, (3, 8), 3, (4, 8), 3, (5, 8), 3, (6, 8), 1, (5, 8), 2, (5, 9), 4, (5, 8), 1, (4, 8), 3, (5, 8), 1, (4, 8), 4, (4, 7), 3, (5, 7), 4, (5, 6), 1, (4, 6), 1, (3, 6), 3, (4, 6), 2, (4, 7), 1, (3, 7), 3, (4, 7), 3, (5, 7), 1, (4, 7), 2, (4, 8), 2, (4, 9), 4, (4, 8), 2, (4, 9), 4, (4, 8), 4, (4, 7), 1, (3, 7), 3, (4, 7), 2, (4, 8), 3, (5, 8), 2, (5, 9), 4, (5, 8), 1, (4, 8), 4, (4, 7), 2, (4, 8), 3, (5, 8), 1, (4, 8), 1, (3, 8), 3, (4, 8), 1, (3, 8), 3, (4, 8), 2, (4, 9), 4, (4, 8), 3, (5, 8), 4, (5, 7), 4, (5, 6), 3, (6, 6), 3, (7, 6), 1, (6, 6), 2, (6, 7), 4, (6, 6), 2, (6, 7), 4, (6, 6), 2, (6, 7), 4, (6, 6), 4, (6, 5), 2, (6, 6), 3, (7, 6), 4, (7, 5), 2, (7, 6), 2, (7, 7)]] :

\*\*\*\*\*\*\*\*\*\*

\*\*----\*??\*

\*--\*?-???\*

\*\*\*?\*-\*\*\*\*

\*??\*-----\*

\*???\*\*---\*

\*????\*-\*\*\*

\*????--f?\*

\*????????\*

\*\*\*\*\*\*\*\*\*\*

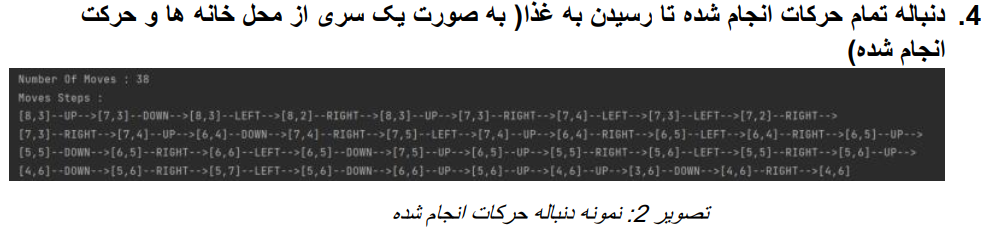
the number of step to find shortest path :

12

the short path is :

[(2, 2), 1, (1, 2), 2, (1, 3), 2, (1, 4), 2, (1, 5), 3, (2, 5), 3, (3, 5), 3, (4, 5), 2, (4, 6), 3, (5, 6), 3, (6, 6), 3, (7, 6), 2, (7, 7)]

* First point: According to the description of the project section of the project: (Write programs that play the game can find the food), so according to my understanding of it, in the first part of the project, the goal is to write a program using the agent function and the environment that can only find a path,actually that is, the goal is not to find the optimal path in the first part.
* Second point: As you can see in the first part, all the paths traveled by the agent function are printed. The reason is that the number of steps is high and the path is long. Due to the irrationality of the agent and the lack of recognition of the traveled path in each step, my algorithm It chooses randomly and finally acts in the same way until it reaches the food, so our method is based on brute-force.
* The third point, considering that the agent is irrational and moves randomly and irrationally every time, so it is possible to take very high steps so that the computer is running for several minutes, because our algorithm is not optimal and in If this problem occurs, run the code again.
* 4th point: according to the request of the project, according to the table below, the project asked us all the routes traveled from start to reaching the food, which is in the form of a list, that is, the same as the first list.



* The fifth point: In part number five, it is considered that it has a memory input, so in this part it is asked that it asks us for the shortest path considering the memory, so we can choose one of the tree or graph search methods considering the optimality and According to the analysis search, I have come to this result to use the IDS search that I I have given its code below:
* def get\_neighbors(node, map):
* neighbors = []
* row, col = node
* for dr, dc in [(0, 1), (1, 0), (0, -1), (-1, 0)]:
* new\_row, new\_col = row + dr, col + dc
* if 0 <= new\_row < len(map) and 0 <= new\_col < len(map[0]) and map[new\_row][new\_col] != '\*':
* neighbors.append((new\_row, new\_col))
* return neighbors
* def dls\_shortest\_path(map, start, goal, depth):
* if start == goal:
* return [start]
* if depth == 0:
* return None
* for neighbor in get\_neighbors(start, map):
* result = dls\_shortest\_path(map, neighbor, goal, depth - 1)
* if result is not None:
* return [start] + result
* return None
* def ids\_shortest\_path(map, start, goal):
* depth = 0
* while True:
* result = dls\_shortest\_path(map, start, goal, depth)
* if result is not None:
* return result
* depth += 1
* this code have tree function that using into each.
* Sixth point:As you can see in the output of the first example, I have found the optimal route using IDS and the number of steps of the optimal route along with the list of the traveled route, similar to the first list, and printed it.
* Seventh point: As you can see, I printed my outputs as above to avoid some ambiguities.
* Input2:

10,10

8,8

\*\*\*\*\*\*\*\*\*\*

\*\*----\*\*-\*

\*--\*---\*-\*

\*-\*\*\*-\*\*\*\*

\*-f\*-----\*

\*\*\*-\*\*---\*

\*-\*----\*\*\*

\*--\*-----\*

\*--\*--\*-a

\*\*\*\*\*\*\*\*\*\*\*

10,10

9,9

Coordinates of the starting point :

(8, 8)

The coordinates of the place of food :

(4, 2)

the number of step to find path using 'bruth\_forts' mehthod :

549

The list consists of the selected coordinate and motion pairs:

[[(8, 8), 4, (8, 7), 1, (7, 7), 2, (7, 8), 1, (6, 8), 3, (7, 8), 4, (7, 7), 2, (7, 8), 3, (8, 8), 4, (8, 7), 4, (8, 6), 2, (8, 7), 2, (8, 8), 1, (7, 8), 3, (8, 8), 3, (9, 8), 1, (8, 8), 2, (8, 9), 4, (8, 8), 1, (7, 8), 2, (7, 9), 4, (7, 8), 3, (8, 8), 4, (8, 7), 4, (8, 6), 2, (8, 7), 4, (8, 6), 2, (8, 7), 4, (8, 6), 2, (8, 7), 4, (8, 6), 2, (8, 7), 3, (9, 7), 1, (8, 7), 2, (8, 8), 3, (9, 8), 1, (8, 8), 1, (7, 8), 3, (8, 8), 3, (9, 8), 1, (8, 8), 3, (9, 8), 1, (8, 8), 1, (7, 8), 2, (7, 9), 4, (7, 8), 3, (8, 8), 2, (8, 9), 4, (8, 8), 2, (8, 9), 4, (8, 8), 3, (9, 8), 1, (8, 8), 1, (7, 8), 4, (7, 7), 2, (7, 8), 2, (7, 9), 4, (7, 8), 3, (8, 8), 2, (8, 9), 4, (8, 8), 3, (9, 8), 1, (8, 8), 1, (7, 8), 4, (7, 7), 2, (7, 8), 2, (7, 9), 4, (7, 8), 1, (6, 8), 3, (7, 8), 1, (6, 8), 3, (7, 8), 1, (6, 8), 3, (7, 8), 3, (8, 8), 3, (9, 8), 1, (8, 8), 1, (7, 8), 3, (8, 8), 2, (8, 9), 4, (8, 8), 2, (8, 9), 4, (8, 8), 2, (8, 9), 4, (8, 8), 4, (8, 7), 1, (7, 7), 3, (8, 7), 1, (7, 7), 1, (6, 7), 3, (7, 7), 2, (7, 8), 3, (8, 8), 4, (8, 7), 1, (7, 7), 1, (6, 7), 3, (7, 7), 3, (8, 7), 2, (8, 8), 2, (8, 9), 4, (8, 8), 2, (8, 9), 4, (8, 8), 2, (8, 9), 4, (8, 8), 4, (8, 7), 3, (9, 7), 1, (8, 7), 4, (8, 6), 2, (8, 7), 1, (7, 7), 4, (7, 6), 3, (8, 6), 1, (7, 6), 4, (7, 5), 1, (6, 5), 3, (7, 5), 2, (7, 6), 2, (7, 7), 4, (7, 6), 3, (8, 6), 1, (7, 6), 2, (7, 7), 2, (7, 8), 2, (7, 9), 4, (7, 8), 3, (8, 8), 1, (7, 8), 1, (6, 8), 3, (7, 8), 2, (7, 9), 4, (7, 8), 1, (6, 8), 3, (7, 8), 3, (8, 8), 4, (8, 7), 3, (9, 7), 1, (8, 7), 4, (8, 6), 2, (8, 7), 2, (8, 8), 2, (8, 9), 4, (8, 8), 4, (8, 7), 3, (9, 7), 1, (8, 7), 1, (7, 7), 3, (8, 7), 4, (8, 6), 2, (8, 7), 4, (8, 6), 2, (8, 7), 2, (8, 8), 3, (9, 8), 1, (8, 8), 2, (8, 9), 4, (8, 8), 1, (7, 8), 2, (7, 9), 4, (7, 8), 2, (7, 9), 4, (7, 8), 3, (8, 8), 4, (8, 7), 3, (9, 7), 1, (8, 7), 3, (9, 7), 1, (8, 7), 3, (9, 7), 1, (8, 7), 1, (7, 7), 2, (7, 8), 4, (7, 7), 1, (6, 7), 3, (7, 7), 2, (7, 8), 1, (6, 8), 3, (7, 8), 4, (7, 7), 1, (6, 7), 3, (7, 7), 3, (8, 7), 1, (7, 7), 4, (7, 6), 2, (7, 7), 2, (7, 8), 3, (8, 8), 3, (9, 8), 1, (8, 8), 3, (9, 8), 1, (8, 8), 1, (7, 8), 1, (6, 8), 3, (7, 8), 2, (7, 9), 4, (7, 8), 1, (6, 8), 3, (7, 8), 3, (8, 8), 2, (8, 9), 4, (8, 8), 1, (7, 8), 3, (8, 8), 4, (8, 7), 1, (7, 7), 4, (7, 6), 3, (8, 6), 1, (7, 6), 2, (7, 7), 4, (7, 6), 4, (7, 5), 4, (7, 4), 4, (7, 3), 2, (7, 4), 2, (7, 5), 1, (6, 5), 2, (6, 6), 1, (5, 6), 2, (5, 7), 3, (6, 7), 1, (5, 7), 1, (4, 7), 4, (4, 6), 2, (4, 7), 3, (5, 7), 3, (6, 7), 1, (5, 7), 3, (6, 7), 1, (5, 7), 2, (5, 8), 3, (6, 8), 1, (5, 8), 1, (4, 8), 3, (5, 8), 3, (6, 8), 1, (5, 8), 2, (5, 9), 4, (5, 8), 3, (6, 8), 1, (5, 8), 1, (4, 8), 4, (4, 7), 1, (3, 7), 3, (4, 7), 4, (4, 6), 4, (4, 5), 4, (4, 4), 2, (4, 5), 4, (4, 4), 4, (4, 3), 2, (4, 4), 1, (3, 4), 3, (4, 4), 2, (4, 5), 1, (3, 5), 3, (4, 5), 3, (5, 5), 1, (4, 5), 1, (3, 5), 3, (4, 5), 1, (3, 5), 4, (3, 4), 2, (3, 5), 3, (4, 5), 2, (4, 6), 1, (3, 6), 3, (4, 6), 1, (3, 6), 3, (4, 6), 1, (3, 6), 3, (4, 6), 3, (5, 6), 3, (6, 6), 1, (5, 6), 4, (5, 5), 2, (5, 6), 1, (4, 6), 3, (5, 6), 3, (6, 6), 1, (5, 6), 3, (6, 6), 2, (6, 7), 4, (6, 6), 1, (5, 6), 2, (5, 7), 4, (5, 6), 1, (4, 6), 4, (4, 5), 4, (4, 4), 1, (3, 4), 3, (4, 4), 4, (4, 3), 2, (4, 4), 2, (4, 5), 2, (4, 6), 4, (4, 5), 4, (4, 4), 3, (5, 4), 1, (4, 4), 4, (4, 3), 2, (4, 4), 3, (5, 4), 1, (4, 4), 4, (4, 3), 2, (4, 4), 2, (4, 5), 3, (5, 5), 1, (4, 5), 1, (3, 5), 4, (3, 4), 2, (3, 5), 3, (4, 5), 3, (5, 5), 1, (4, 5), 1, (3, 5), 2, (3, 6), 4, (3, 5), 4, (3, 4), 2, (3, 5), 2, (3, 6), 4, (3, 5), 4, (3, 4), 2, (3, 5), 2, (3, 6), 4, (3, 5), 3, (4, 5), 4, (4, 4), 3, (5, 4), 1, (4, 4), 4, (4, 3), 2, (4, 4), 4, (4, 3), 2, (4, 4), 2, (4, 5), 4, (4, 4), 4, (4, 3), 2, (4, 4), 4, (4, 3), 2, (4, 4), 2, (4, 5), 1, (3, 5), 2, (3, 6), 4, (3, 5), 1, (2, 5), 1, (1, 5), 2, (1, 6), 4, (1, 5), 2, (1, 6), 4, (1, 5), 4, (1, 4), 2, (1, 5), 3, (2, 5), 3, (3, 5), 2, (3, 6), 4, (3, 5), 3, (4, 5), 3, (5, 5), 1, (4, 5), 2, (4, 6), 1, (3, 6), 3, (4, 6), 1, (3, 6), 3, (4, 6), 2, (4, 7), 3, (5, 7), 2, (5, 8), 4, (5, 7), 2, (5, 8), 3, (6, 8), 1, (5, 8), 1, (4, 8), 3, (5, 8), 4, (5, 7), 4, (5, 6), 3, (6, 6), 1, (5, 6), 4, (5, 5), 2, (5, 6), 4, (5, 5), 2, (5, 6), 1, (4, 6), 1, (3, 6), 3, (4, 6), 3, (5, 6), 2, (5, 7), 3, (6, 7), 1, (5, 7), 1, (4, 7), 3, (5, 7), 3, (6, 7), 1, (5, 7), 3, (6, 7), 1, (5, 7), 3, (6, 7), 1, (5, 7), 4, (5, 6), 2, (5, 7), 1, (4, 7), 1, (3, 7), 3, (4, 7), 1, (3, 7), 3, (4, 7), 1, (3, 7), 3, (4, 7), 1, (3, 7), 3, (4, 7), 1, (3, 7), 3, (4, 7), 4, (4, 6), 3, (5, 6), 4, (5, 5), 2, (5, 6), 3, (6, 6), 3, (7, 6), 1, (6, 6), 3, (7, 6), 4, (7, 5), 4, (7, 4), 2, (7, 5), 2, (7, 6), 3, (8, 6), 1, (7, 6), 2, (7, 7), 4, (7, 6), 2, (7, 7), 4, (7, 6), 1, (6, 6), 2, (6, 7), 4, (6, 6), 4, (6, 5), 2, (6, 6), 2, (6, 7), 4, (6, 6), 1, (5, 6), 1, (4, 6), 4, (4, 5), 2, (4, 6), 1, (3, 6), 3, (4, 6), 3, (5, 6), 1, (4, 6), 3, (5, 6), 4, (5, 5), 2, (5, 6), 4, (5, 5), 2, (5, 6), 1, (4, 6), 4, (4, 5), 4, (4, 4), 4, (4, 3), 2, (4, 4), 2, (4, 5), 2, (4, 6), 3, (5, 6), 1, (4, 6), 4, (4, 5), 4, (4, 4), 4, (4, 3), 2, (4, 4), 1, (3, 4), 3, (4, 4), 2, (4, 5), 4, (4, 4), 4, (4, 3), 2, (4, 4), 2, (4, 5), 3, (5, 5), 1, (4, 5), 1, (3, 5), 4, (3, 4), 2, (3, 5), 1, (2, 5), 3, (3, 5), 4, (3, 4), 2, (3, 5), 3, (4, 5), 4, (4, 4), 3, (5, 4), 1, (4, 4), 3, (5, 4), 1, (4, 4), 1, (3, 4), 3, (4, 4), 1, (3, 4), 3, (4, 4), 4, (4, 3), 2, (4, 4), 1, (3, 4), 3, (4, 4), 2, (4, 5), 4, (4, 4), 2, (4, 5), 4, (4, 4), 1, (3, 4), 3, (4, 4), 1, (3, 4), 3, (4, 4), 3, (5, 4), 1, (4, 4), 1, (3, 4), 3, (4, 4), 1, (3, 4), 3, (4, 4), 2, (4, 5), 1, (3, 5), 4, (3, 4), 2, (3, 5), 3, (4, 5), 1, (3, 5), 1, (2, 5), 1, (1, 5), 3, (2, 5), 1, (1, 5), 1, (0, 5), 3, (1, 5), 2, (1, 6), 4, (1, 5), 2, (1, 6), 4, (1, 5), 2, (1, 6), 4, (1, 5), 4, (1, 4), 4, (1, 3), 1, (0, 3), 3, (1, 3), 4, (1, 2), 3, (2, 2), 3, (3, 2), 1, (2, 2), 3, (3, 2), 1, (2, 2), 2, (2, 3), 4, (2, 2), 4, (2, 1), 3, (3, 1), 2, (3, 2), 4, (3, 1), 3, (4, 1), 4, (4, 0), 2, (4, 1), 1, (3, 1), 2, (3, 2), 4, (3, 1), 4, (3, 0), 2, (3, 1), 4, (3, 0), 2, (3, 1), 4, (3, 0), 2, (3, 1), 3, (4, 1), 2, (4, 2)]] :

Output2:

\*\*\*\*\*\*\*\*\*\*

\*?----\*??\*

\*--\*--???\*

\*-\*?\*-\*\*?\*

\*-f\*-----\*

\*???\*\*---\*

\*????--\*\*\*

\*??\*-----\*

\*?????\*--\*

the number of step to find shortest path :

18

the short path is :

[(8, 8), 4, (8, 7), 1, (7, 7), 4, (7, 6), 1, (6, 6), 1, (5, 6), 1, (4, 6), 4, (4, 5), 1, (3, 5), 1, (2, 5), 4, (2, 4), 1, (1, 4), 4, (1, 3), 4, (1, 2), 3, (2, 2), 4, (2, 1), 3, (3, 1), 3, (4, 1), 2, (4, 2)]the number of step to find shortest path :

18

the short path is :

**[(8, 8), 4, (8, 7), 1, (7, 7), 4, (7, 6), 1, (6, 6), 1, (5, 6), 1, (4, 6), 4, (4, 5), 1, (3, 5), 1, (2, 5), 4, (2, 4), 1, (1, 4), 4, (1, 3), 4, (1, 2), 3, (2, 2), 4, (2, 1), 3, (3, 1), 3, (4, 1), 2, (4, 2)]**

\*\*\*\*\*\*\*\*\*\*

\*\*----\*\*-\*

\*--\*---\*-\*

\*-\*\*\*-\*\*\*\*

\*-f\*-----\*

\*\*\*-\*\*---\*

\*-\*----\*\*\*

\*--\*-----\*

\*--\*--\*-a\*\*

\*\*\*\*\*\*\*\*\*

* Multiple return capability **According to the project**:
* def main(): #multi return  
   # return get\_start\_or\_end\_location('a'),get\_start\_or\_end\_location('f'),count\_number\_of\_Step(actions),find\_food,count\_short[0],combine ->this line is completer return  
   return get\_start\_or\_end\_location('a'),get\_start\_or\_end\_location('f'),count\_short[0],combine  
    
  # write into file that we suppose an agent have memory(بخش امتیازی)  
  write\_out\_put\_Map\_two\_list(update\_out\_put\_map[0])  
    
  if \_\_name\_\_ == "\_\_main\_\_":  
   print(main())
* With multi-return capability, the output of Input samples 1 and 2 are :
* Out\_put\_sample\_1:
* ((2, 2), (7, 7), 12, [(2, 2), 1, (1, 2), 2, (1, 3), 2, (1, 4), 2, (1, 5), 3, (2, 5), 3, (3, 5), 3, (4, 5), 2, (4, 6), 3, (5, 6), 3, (6, 6), 3, (7, 6), 2, (7, 7)])
* Out\_put\_sample\_2:
* ((8, 8), (4, 2), 18, [(8, 8), 4, (8, 7), 1, (7, 7), 4, (7, 6), 1, (6, 6), 1, (5, 6), 1, (4, 6), 4, (4, 5), 1, (3, 5), 1, (2, 5), 4, (2, 4), 1, (1, 4), 4, (1, 3), 4, (1, 2), 3, (2, 2), 4, (2, 1), 3, (3, 1), 3, (4, 1), 2, (4, 2)])
* first point: considering that the output test is software, I added the mine function
* second point:
* **The second point is the output in the form of a list including**:
* The first element of the operator coordinate list as a tuple
* The second element of the coordinates of food in the form of a tuple
* The third element is the number of steps as the number
* the fourth element of the traveled path list