Project : 01

Course Title : Artificial Intelligence & Expert System

Course Code :ETE 475

Section : 01

Semester : Fall 2022

Submitted To **:**

Zahidur Rahman

Lecturer

Department of ECE

Submitted By **:**

Student's Name **:** Md Nafis Rahman

Student's ID **:** 2018-3-55-006

Project 1: Search

Introduction: Pacman is one of the most popular game in the world. The goal of this game is to accumulate points by eating all the Pac-Dots in the maze, completing that ’stage’ of the game and starting the next stage and maze of Pac-dots. There are several ghosts roaming the maze, trying to kill Pac-Man. If any of the ghosts hit Pac-Man, he loses a life; when all lives have been lost, the game is over. With the help of problem-solving agent, we can automatically find optimal paths through its maze world considering both reaching locations (e.g., finding all the corners) and eating all the dots in as few steps as possible. To design this intelligent agent implemented several uninformed search algorithms-algorithms that are given no information about the problem other than its definition (e.g., DFS, BFS, UCS).

Here I edited **search.py** for answering following questions.

Q1: Finding a Fixed Food Dot using Depth First Search:

```
startingNode = problem.getStartState()
    if problem.isGoalState(startingNode):
        return []

    myQueue = util.Stack()
    visitedNodes = []
    # (node,actions)
    myQueue.push((startingNode, []))

    while not myQueue.isEmpty():
        currentNode, actions = myQueue.pop()
        if currentNode not in visitedNodes:
            visitedNodes.append(currentNode)

            if problem.isGoalState(currentNode):
                return actions

            for nextNode, action, cost in problem.getSuccessors(currentNode):
                newAction = actions + [action]
                myQueue.push((nextNode, newAction))
```

```
PS C:\Users\ASUS\Downloads\search (1)\nafis> python autograder.py -q q1
C:\Users\ASUS\Downloads\search (1)\nafis\autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of importlib and slated for removal in Python
 module's documentation for alternative uses
  import imp
Starting on 11-25 at 16:32:49

Question q1
===========
*** PASS: test_cases\q1\graph_backtrack.test
***     solution:               ['1:A->C', '0:C->G']
***     expanded_states:        ['A', 'D', 'C']
*** PASS: test_cases\q1\graph_bfs_vs_dfs.test
***     solution:               ['2:A->D', '0:D->G']
***     expanded_states:        ['A', 'D']
*** PASS: test_cases\q1\graph_infinite.test
***     solution:               ['0:A->B', '1:B->C', '1:C->G']
***     expanded_states:        ['A', 'B', 'C']
*** PASS: test_cases\q1\graph_manypaths.test
***     solution:               ['2:A->B2', '0:B2->C', '0:C->D', '2:D->E2', '0:E2->F', '0:F->G']
***     expanded_states:        ['A', 'B2', 'C', 'D', 'E2', 'F']
*** PASS: test_cases\q1\pacman_1.test
***     pacman layout:          mediumMaze
***     solution length: 130
***     nodes expanded:         146

### Question q1: 3/3 ###


Finished at 16:32:49
```

Q2 : Breadth First Search:

```
startingNode = problem.getStartState()
    if problem.isGoalState(startingNode):
        return []

    myQueue = util.Queue()
    visitedNodes = []
    # (node,actions)
    myQueue.push((startingNode, []))

    while not myQueue.isEmpty():
        currentNode, actions = myQueue.pop()
        if currentNode not in visitedNodes:
            visitedNodes.append(currentNode)

            if problem.isGoalState(currentNode):
                return actions

            for nextNode, action, cost in problem.getSuccessors(currentNode):
                newAction = actions + [action]
                myQueue.push((nextNode, newAction))

    util.raiseNotDefined()
```

```
PS C:\Users\ASUS\Downloads\search (1)\nafis> python autograder.py -q q2
C:\Users\ASUS\Downloads\search (1)\nafis\autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of importlib and slated for removal in P
 module's documentation for alternative uses
  import imp
Starting on 11-25 at 16:34:09

Question q2
===========
*** PASS: test_cases\q2\graph_backtrack.test
***     solution:               ['1:A->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C', 'D']
*** PASS: test_cases\q2\graph_bfs_vs_dfs.test
***     solution:               ['1:A->G']
***     expanded_states:        ['A', 'B']
*** PASS: test_cases\q2\graph_infinite.test
***     solution:               ['0:A->B', '1:B->C', '1:C->G']
***     expanded_states:        ['A', 'B', 'C']
*** PASS: test_cases\q2\graph_manypaths.test
***     solution:               ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***     expanded_states:        ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
*** PASS: test_cases\q2\pacman_1.test
***     pacman layout:          mediumMaze
***     solution length: 68
***     nodes expanded:         269

### Question q2: 3/3 ###


Finished at 16:34:09

Provisional grades
==================
```

Q3 : Varying the Cost Function:

```
def uniformCostSearch(problem):
    "Search the node of least total cost first. "
    "*** YOUR CODE HERE ***"

    startingNode = problem.getStartState()
    if problem.isGoalState(startingNode):
        return []

    visitedNodes = []

    pQueue = util.PriorityQueue()
    #((coordinate/node , action to current node , cost to current node),priority)
    pQueue.push((startingNode, [], 0), 0)

    while not pQueue.isEmpty():

        currentNode, actions, prevCost = pQueue.pop()
        if currentNode not in visitedNodes:
            visitedNodes.append(currentNode)

            if problem.isGoalState(currentNode):
                return actions

            for nextNode, action, cost in problem.getSuccessors(currentNode):
                newAction = actions + [action]
                priority = prevCost + cost
                pQueue.push((nextNode, newAction, priority),priority)
    util.raiseNotDefined()
```

```
*** PASS: test_cases\q3\graph_backtrack.test
***     solution:               ['1:A->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C', 'D']
*** PASS: test_cases\q3\graph_bfs_vs_dfs.test
***     solution:               ['1:A->G']
***     expanded_states:        ['A', 'B']
*** PASS: test_cases\q3\graph_infinite.test
***     solution:               ['0:A->B', '1:B->C', '1:C->G']
***     expanded_states:        ['A', 'B', 'C']
*** PASS: test_cases\q3\graph_manypaths.test
***     solution:               ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***     expanded_states:        ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
*** PASS: test_cases\q3\ucs_0_graph.test
***     solution:               ['Right', 'Down', 'Down']
***     expanded_states:        ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases\q3\ucs_1_problemC.test
***     pacman layout:          mediumMaze
***     solution length: 68
***     nodes expanded:         269
*** PASS: test_cases\q3\ucs_2_problemE.test
***     pacman layout:          mediumMaze
***     solution length: 74
***     nodes expanded:         260
*** PASS: test_cases\q3\ucs_3_problemW.test
***     pacman layout:          mediumMaze
***     solution length: 152
***     nodes expanded:         173
*** PASS: test_cases\q3\ucs_4_testSearch.test
***     pacman layout:          testSearch
***     solution length: 7
***     nodes expanded:         14
*** PASS: test_cases\q3\ucs_5_goalAtDequeue.test
***     solution:               ['1:A->B', '0:B->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C']

### Question q3: 3/3 ###


Finished at 16:35:05

Provisional grades
==================
Question q3: 3/3
------------------
Total: 3/3
```

Q4 : A* search:

```
ef aStarSearch(problem, heuristic=nullHeuristic):
    """Search the node that has the lowest combined cost and heuristic first."""
    "*** YOUR CODE HERE ***"

    startingNode = problem.getStartState()
    if problem.isGoalState(startingNode):
        return []

    visitedNodes = []

    pQueue = util.PriorityQueue()
    #((coordinate/node , action to current node , cost to current node),priority)
    pQueue.push((startingNode, [], 0), 0)

    while not pQueue.isEmpty():

        currentNode, actions, prevCost = pQueue.pop()

        if currentNode not in visitedNodes:
            visitedNodes.append(currentNode)

            if problem.isGoalState(currentNode):
                return actions

            for nextNode, action, cost in problem.getSuccessors(currentNode):
                newAction = actions + [action]
                newCostToNode = prevCost + cost
                heuristicCost = newCostToNode + heuristic(nextNode,problem)
                pQueue.push((nextNode, newAction, newCostToNode),heuristicCost)

    util.raiseNotDefined()
```

```
PS C:\Users\ASUS\Downloads\search (1)\nafis> python autograder.py -q q4
C:\Users\ASUS\Downloads\search (1)\nafis\autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of importlib and slated for removal in Python 3.12;
 module's documentation for alternative uses
  import imp
Starting on 11-25 at 16:36:18

Question q4
===========
*** PASS: test_cases\q4\astar_0.test
***     solution:               ['Right', 'Down', 'Down']
***     expanded_states:        ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases\q4\astar_1_graph_heuristic.test
***     solution:               ['0', '0', '2']
***     expanded_states:        ['S', 'A', 'D', 'C']
*** PASS: test_cases\q4\astar_2_manhattan.test
***     pacman layout:          mediumMaze
***     solution length: 68
***     nodes expanded:         221
*** PASS: test_cases\q4\astar_3_goalAtDequeue.test
***     solution:               ['1:A->B', '0:B->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C']
*** PASS: test_cases\q4\graph_backtrack.test
***     solution:               ['1:A->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C', 'D']
*** PASS: test_cases\q4\graph_manypaths.test
***     solution:               ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***     expanded_states:        ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']

### Question q4: 3/3 ###


Finished at 16:36:18

Provisional grades
==================
Question q4: 3/3
------------------
Total: 3/3

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

Q7 Eating All The Dots:

```
class FoodSearchProblem:
    """
    A search problem associated with finding the a path that collects all of the
    food (dots) in a Pacman game.

    A search state in this problem is a tuple ( pacmanPosition, foodGrid ) where
      pacmanPosition: a tuple (x,y) of integers specifying Pacman's position
      foodGrid:       a Grid (see game.py) of either True or False, specifying
remaining food
    """
    def __init__(self, startingGameState: pacman.GameState):
        self.start = (startingGameState.getPacmanPosition(),
startingGameState.getFood())
        self.walls = startingGameState.getWalls()
        self.startingGameState = startingGameState
        self._expanded = 0 # DO NOT CHANGE
        self.heuristicInfo = {} # A dictionary for the heuristic to store
information

    def getStartState(self):
        return self.start

    def isGoalState(self, state):
        return state[1].count() == 0

    def getSuccessors(self, state):
        "Returns successor states, the actions they require, and a cost of 1."
        successors = []
        self._expanded += 1 # DO NOT CHANGE
        for direction in [Directions.NORTH, Directions.SOUTH, Directions.EAST,
Directions.WEST]:
            x,y = state[0]
            dx, dy = Actions.directionToVector(direction)
            nextx, nexty = int(x + dx), int(y + dy)
            if not self.walls[nextx][nexty]:
                nextFood = state[1].copy()
                nextFood[nextx][nexty] = False
                successors.append( ( ((nextx, nexty), nextFood), direction, 1) )
        return successors

    def getCostOfActions(self, actions):
```

```
        """Returns the cost of a particular sequence of actions.  If those
actions
        include an illegal move, return 999999"""
        x,y= self.getStartState()[0]
        cost = 0
        for action in actions:
            # figure out the next state and see whether it's legal
            dx, dy = Actions.directionToVector(action)
            x, y = int(x + dx), int(y + dy)
            if self.walls[x][y]:
                return 999999
            cost += 1
        return cost

class AStarFoodSearchAgent(SearchAgent):
    "A SearchAgent for FoodSearchProblem using A* and your foodHeuristic"
    def __init__(self):
        self.searchFunction = lambda prob: search.aStarSearch(prob,
foodHeuristic)
        self.searchType = FoodSearchProblem
```

```
Question q7
===========
*** PASS: test_cases\q7\food_heuristic_1.test
*** PASS: test_cases\q7\food_heuristic_10.test
*** PASS: test_cases\q7\food_heuristic_11.test
*** PASS: test_cases\q7\food_heuristic_12.test
*** PASS: test_cases\q7\food_heuristic_13.test
*** PASS: test_cases\q7\food_heuristic_14.test
*** PASS: test_cases\q7\food_heuristic_15.test
*** PASS: test_cases\q7\food_heuristic_16.test
*** PASS: test_cases\q7\food_heuristic_17.test
*** PASS: test_cases\q7\food_heuristic_2.test
*** PASS: test_cases\q7\food_heuristic_3.test
*** PASS: test_cases\q7\food_heuristic_4.test
*** PASS: test_cases\q7\food_heuristic_5.test
*** PASS: test_cases\q7\food_heuristic_6.test
*** PASS: test_cases\q7\food_heuristic_7.test
*** PASS: test_cases\q7\food_heuristic_8.test
*** PASS: test_cases\q7\food_heuristic_9.test
*** FAIL: test_cases\q7\food_heuristic_grade_tricky.test
***     expanded nodes: 12517
***     thresholds: [15000, 12000, 9000, 7000]

### Question q7: 2/4 ###


Finished at 17:18:56

Provisional grades
==================
Question q4: 3/3
Question q7: 2/4
------------------
Total: 5/7
```

<u>Final Result:</u>

```
### Question q8: 0/3 ###


Finished at 17:20:37

Provisional grades
==================
Question q1: 3/3
Question q2: 3/3
Question q3: 3/3
Question q4: 3/3
Question q5: 0/3
Question q6: 0/3
Question q7: 2/4
Question q8: 0/3
------------------
Total: 14/25

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

PS C:\Users\ASUS\Downloads\search (1)\nafis>
```

Discussion: In this whole project I have learned the implementation of AI algorithms such as DFS, BFS, A* etc. A folder named search was given after that I have edited search.py & searchAgent.py file for answering following questions.

The following project is available here: https://github.com/nafisrahman006/AI.git