

# Artificial Intelligence

Assignment 2(Maze)

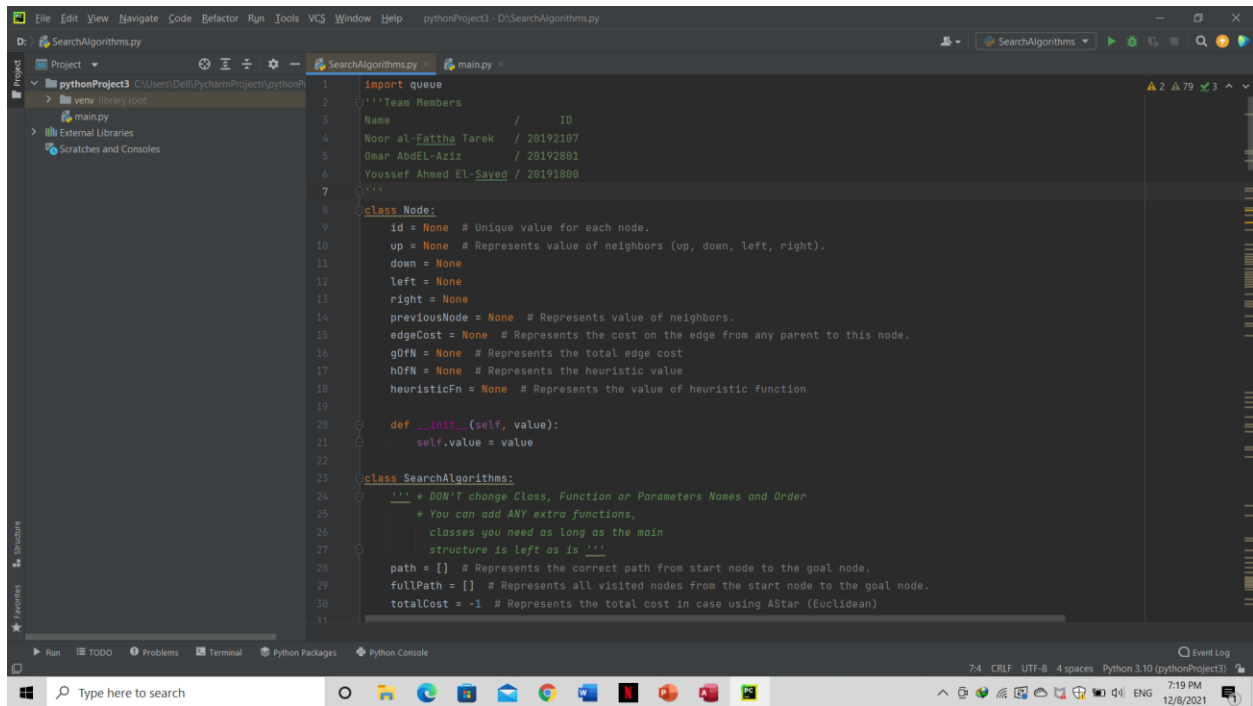
Name	ID
Noor Al-Fattha Tarek Wahdan	20192107
Youssef Ahmed El-Sayed	20191800
Omar AbdEl-Aziz	20192801

Under Supervision TA/ Ayman Adel Iskandar

Group: A1.1 Monday from 1:00 To 3:00



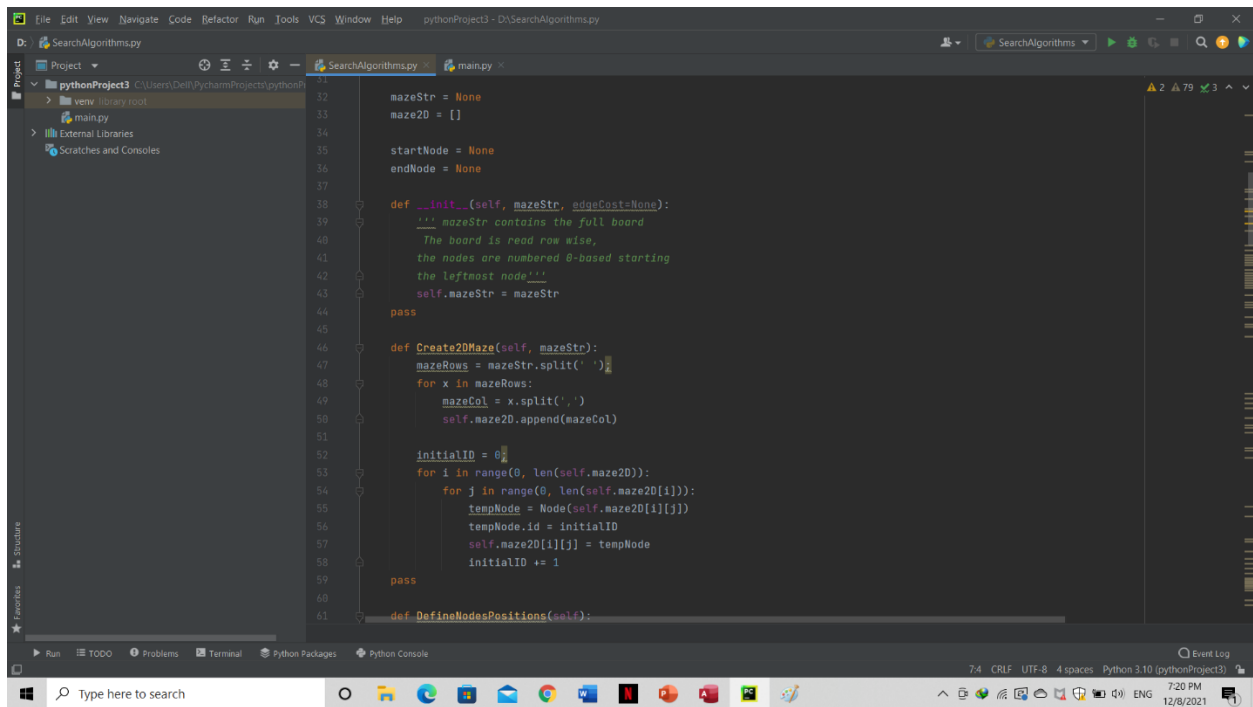
## Screenshot1(Code):



This screenshot shows the first part of the `SearchAlgorithms.py` file in the PyCharm IDE. The code includes imports, team member information, and the definition of the `Node` class and the `SearchAlgorithms` class.

```
1 import queue
2 '''Team Members
3 Name / ID
4 Noor al-Fattha Tarek / 20192107
5 Omar Abdel-Aziz / 20192801
6 Youssef Ahmed El-Sayed / 20191800
7 '''
8 class Node:
9     id = None # Unique value for each node.
10    up = None # Represents value of neighbors (up, down, left, right).
11    down = None
12    left = None
13    right = None
14    previousNode = None # Represents value of neighbors.
15    edgeCost = None # Represents the cost on the edge from any parent to this node.
16    gOfN = None # Represents the total edge cost
17    hOfN = None # Represents the heuristic value
18    heuristicFn = None # Represents the value of heuristic function
19
20    def __init__(self, value):
21        self.value = value
22
23    class SearchAlgorithms:
24        ''' * DON'T change Class, Function or Parameters Names and Order
25         * You can add ANY extra functions,
26         classes you need as long as the main
27         structure is left as is '''
28
29        path = [] # Represents the correct path from start node to the goal node.
30        fullPath = [] # Represents all visited nodes from the start node to the goal node.
31        totalCost = -1 # Represents the total cost in case using AStar (Euclidean)
```

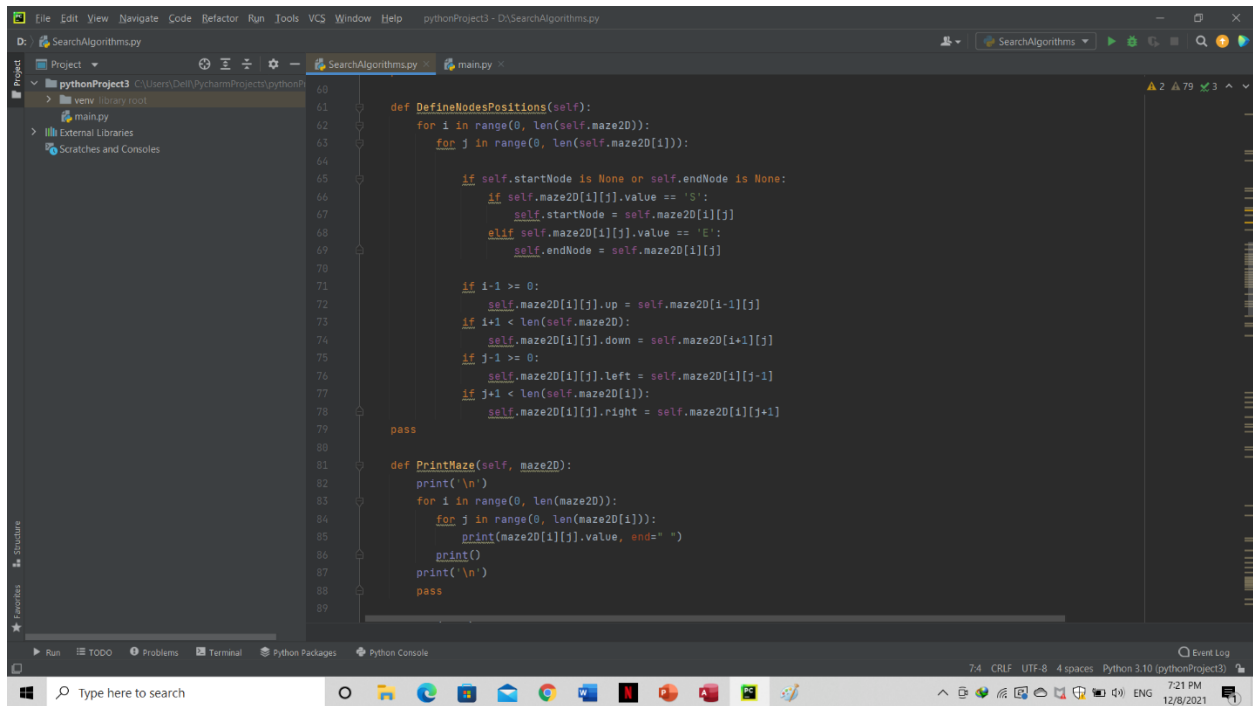
## Screenshot2(Code):



This screenshot shows the second part of the `SearchAlgorithms.py` file in the PyCharm IDE. The code continues with the initialization of variables and the implementation of methods for the `SearchAlgorithms` class.

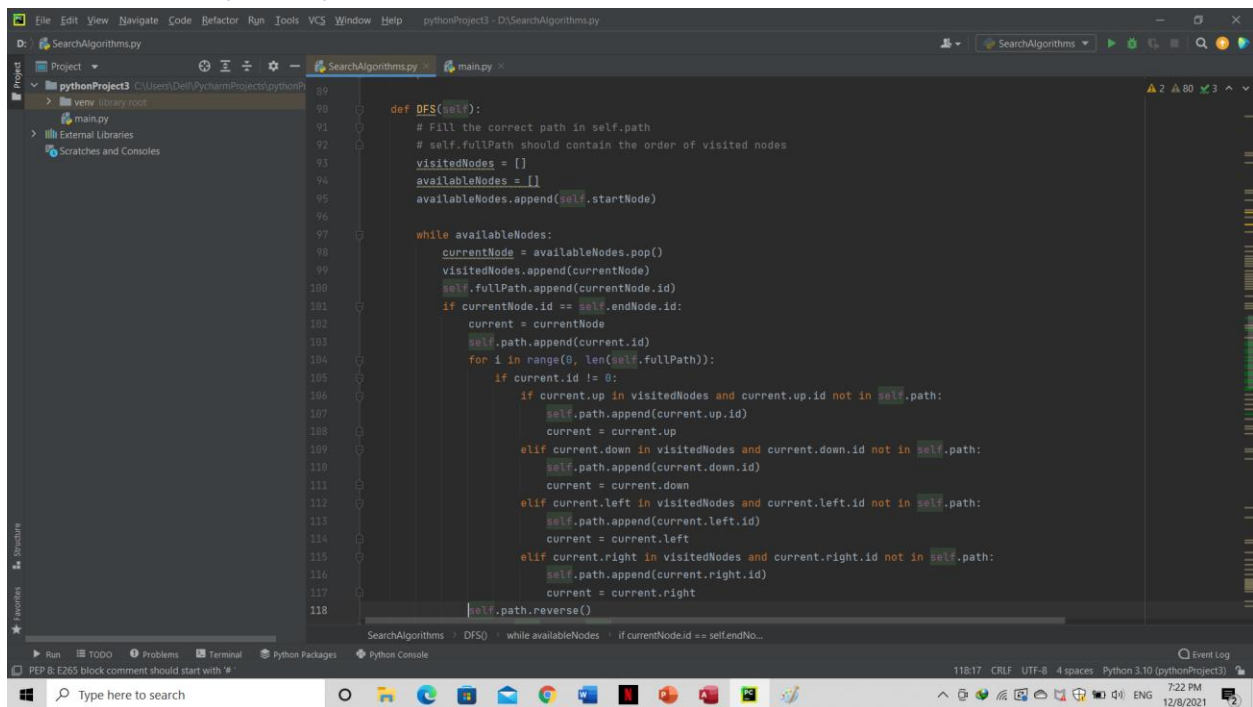
```
32 mazeStr = None
33 maze2D = []
34
35 startNode = None
36 endNode = None
37
38 def __init__(self, mazeStr, edgeCost=None):
39     ''' mazeStr contains the full board
40     The board is read row wise,
41     the nodes are numbered 0-based starting
42     the leftmost node'''
43     self.mazeStr = mazeStr
44     pass
45
46 def Create2DMaze(self, mazeStr):
47     mazeRows = mazeStr.split('\n')
48     for x in mazeRows:
49         mazeCol = x.split(',')
50         self.maze2D.append(mazeCol)
51
52     initialID = 0
53     for i in range(0, len(self.maze2D)):
54         for j in range(0, len(self.maze2D[i])):
55             tempNode = Node(self.maze2D[i][j])
56             tempNode.id = initialID
57             self.maze2D[i][j] = tempNode
58             initialID += 1
59     pass
60
61 def DefineNodesPositions(self):
```

## Screenshot3(Code):



```
60
61 def DefineNodesPositions(self):
62     for i in range(0, len(self.maze2D)):
63         for j in range(0, len(self.maze2D[i])):
64
65             if self.startNode is None or self.endNode is None:
66                 if self.maze2D[i][j].value == 'S':
67                     self.startNode = self.maze2D[i][j]
68                 elif self.maze2D[i][j].value == 'E':
69                     self.endNode = self.maze2D[i][j]
70
71             if i-1 >= 0:
72                 self.maze2D[i][j].up = self.maze2D[i-1][j]
73             if i+1 < len(self.maze2D):
74                 self.maze2D[i][j].down = self.maze2D[i+1][j]
75             if j-1 >= 0:
76                 self.maze2D[i][j].left = self.maze2D[i][j-1]
77             if j+1 < len(self.maze2D[i]):
78                 self.maze2D[i][j].right = self.maze2D[i][j+1]
79
80         pass
81
82 def PrintMaze(self, maze2D):
83     print('\n')
84     for i in range(0, len(maze2D)):
85         for j in range(0, len(maze2D[i])):
86             print(maze2D[i][j].value, end=" ")
87         print()
88     print('\n')
89     pass
```

## Screenshot4(Code):



```
89
90 def DFS(self):
91     # Fill the correct path in self.path
92     # self.fullPath should contain the order of visited nodes
93     visitedNodes = []
94     availableNodes = []
95     availableNodes.append(self.startNode)
96
97     while availableNodes:
98         currentNode = availableNodes.pop()
99         visitedNodes.append(currentNode)
100         self.fullPath.append(currentNode.id)
101         if currentNode.id == self.endNode.id:
102             current = currentNode
103             self.path.append(current.id)
104             for i in range(0, len(self.fullPath)):
105                 if current.id != 0:
106                     if current.up in visitedNodes and current.up.id not in self.path:
107                         self.path.append(current.up.id)
108                         current = current.up
109                     elif current.down in visitedNodes and current.down.id not in self.path:
110                         self.path.append(current.down.id)
111                         current = current.down
112                     elif current.left in visitedNodes and current.left.id not in self.path:
113                         self.path.append(current.left.id)
114                         current = current.left
115                     elif current.right in visitedNodes and current.right.id not in self.path:
116                         self.path.append(current.right.id)
117                         current = current.right
118             self.path.reverse()
```

The screenshot displays the PyCharm IDE interface. The main editor window shows a Python file named `SearchAlgorithms.py` with the following code:

```

149 self.fullPath.append(currentNode.id)
150 if currentNode.id == self.endNode.id:
151     current = currentNode
152     self.path.append(current.id)
153     visitedNodes.reverse()
154     for visited in visitedNodes:
155         if isValidPath(current, visited):
156             self.path.append(visited.id)
157             current = visited
158             self.path.reverse()
159     return self.path, self.fullPath
160
161 if currentNode.up is not None and currentNode.up not in visitedNodes and currentNode.up.value != '#':
162     availableNodes.put(currentNode.up)
163 if currentNode.down is not None and currentNode.down not in visitedNodes and currentNode.down.value != '#':
164     availableNodes.put(currentNode.down)
165 if currentNode.left is not None and currentNode.left not in visitedNodes and currentNode.left.value != '#':
166     availableNodes.put(currentNode.left)
167 if currentNode.right is not None and currentNode.right not in visitedNodes and currentNode.right.value != '#':
168     availableNodes.put(currentNode.right)
169 return self.path, self.fullPath
170
171 def AStarEuclideanHeuristic(self):
172     # Cost for a step is calculated based on edge cost of node
173     # and use Euclidean Heuristic for evaluating the heuristic value
174     # Fill the correct path in self.path
175     # self.fullPath should contain the order of visited nodes
176     return self.path, self.fullPath, self.totalCost
177
178
179

```

The left sidebar shows the project structure with folders for `pythonProject3`, `venv`, `library`, `root`, `main.py`, `External Libraries`, and `Scratches and Consoles`. The bottom toolbar includes icons for Run, Problems, Terminal, Python Packages, and Python Console. The status bar at the bottom indicates the current file is `SearchAlgorithms.py`, the search is for `DFSQ`, and the current node is `while availableNodes`.

```

def main():
    searchAlgo = SearchAlgorithms('S...#.....#.....#.#.....#.#.....#.#E..#.#.')
    searchAlgo.Create2DMaze(searchAlgo.mazeStr)
    searchAlgo.DefineNodesPositions()
    searchAlgo.PrintMaze(searchAlgo.maze2D)
    path, fullPath = searchAlgo.DFS()
    print("DFS\nPath is: " + str(path) + "\nFull Path is: " + str(fullPath) + "\n")

    #####

    searchAlgo = SearchAlgorithms('S...#.....#.....#.#.....#.#.....#.#E..#.#.')
    searchAlgo.DefineNodesPositions()
    path, fullPath = searchAlgo.BFS()
    print("BFS\nPath is: " + str(path) + "\nFull Path is: " + str(fullPath) + "\n")

    #####

    searchAlgo = SearchAlgorithms('S...#.....#.....#.#.....#.#.....#.#E..#.#.', [0, 15, 2, 100, 60, 35, 30, 100, 2, 15, 60, 100, 30, 100, 2, 2, 2, 40, 30, 2, 2, 100, 100, 3, 15, 30, 100, 100, 0, 2, 100, 30])

    path, fullPath, TotalCost = searchAlgo.AStarEuclideanHeuristic()
    print("AStar with Euclidean Heuristic\nPath is: " + str(path) + "\nFull Path is: " + str(
        fullPath) + "\nTotal Cost: " + str(TotalCost) + "\n")

    #####

main()

```

Screenshot7(Output):

The screenshot shows the PyCharm IDE interface. The top toolbar includes menus for File, Edit, View, Navigate, Code, Refactor, Run, Tools, Help, and Window. The main editor window displays the code for SearchAlgorithms.py. The Run console at the bottom shows the execution output:

```

C:\Users\De\PycharmProjects\pythonProject3\venv\Scripts\python.exe D:/SearchAlgorithms.py

S . . # . . .
. # . . . . .
. # . . . . .
. . # . . . .
# . # E . . .

**BFS**
Path is: [0, 1, 2, 9, 16, 17, 18, 25, 32, 31]
Full Path is: [0, 7, 14, 21, 22, 29, 1, 2, 9, 16, 17, 18, 25, 32, 31]

**BFS**
Path is: [0, 1, 2, 9, 10, 11, 18, 25, 32, 31]
Full Path is: [0, 7, 1, 14, 2, 21, 9, 22, 16, 10, 29, 17, 17, 11, 18, 18, 4, 18, 25, 19, 25, 19, 5, 25, 19, 32, 26, 26, 20, 32, 26, 26, 20, 6, 32, 26, 26, 20, 31]

**ASTAR with Euclidean Heuristic **
Path is: [0, 1, 2, 9, 10, 11, 18, 25, 32, 31]
Full Path is: [0, 7, 1, 14, 2, 21, 9, 22, 16, 10, 29, 17, 17, 11, 18, 18, 4, 18, 25, 19, 25, 19, 5, 25, 19, 32, 26, 26, 20, 32, 26, 26, 20, 6, 32, 26, 26, 20, 31]
Total Cost: -1

Process finished with exit code 0

```

The bottom status bar indicates the file encoding is UTF-8, the workspace contains 4 spaces, and the Python version is 3.10 (pythonProject3). The system clock shows 7:25 PM on 12/8/2021.