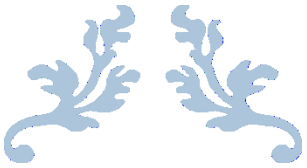


[AI-PACKAGE]

Milestone 2

Milestone: 2
Due date: **6th April 2019**
Artificial Intelligence
Instructor: Dr. Marco Alfonse

Ain Shams University
Faculty of Computer & Information Sciences Course:
Third Grade

General instructions:

Regarding your AI-Package:

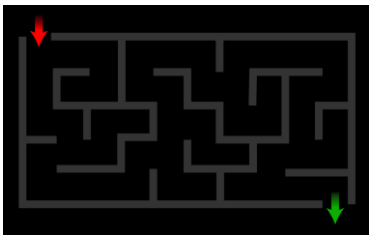
1. Each **team member** should participate of **each** milestone, as he/she will be asked for a **modification** for **any** milestone.
2. Add your new milestone folder to your 'AI-Package'.
3. Only **One AI-Package** folder should be available on shared folder either by **updating** old package **or remove** the old and upload the **whole** package again (including previous milestones).
4. After you finish writing your code:
   a. Open the folder shared with your team on GoogleDrive (the one with your team number)
   b. Upload all the project files with the same hierarchy.
4. Submit **only running** code that you have tested before.
5. **Compressed** files (.zip/.rar) are **not allowed.**
6. The Submission of package is **only** through **your shared folder on google drive.**

Regarding the search algorithms submission [Milestone 2]:

1. Add a new folder named 'SearchAlgorithms' in the 'AI-Package' project.
2. Add the shared template file named 'SearchAlgorithms.py' to 'SearchAlgorithms' folder.
3. Your code should be written **only** in "SearchAlgorithms.py'" file under 'SearchAlgorithms' folder.
4. Please, read code documentation carefully.
5. Your code should be generic for any dimension of a given maze.
6. This milestone will be **autograded**.

This package is intended for team work contribution. Sharing ideas or part of the answers is considered plagiarism and will not be tolerated.
All submissions will be checked for plagiarism automatically.

In this milestone, you are expected to solve a 2-D maze using DFS, BFS, UCS and A* (using 2 different heuristics Euclidian and Manhattan). A maze is path typically from start node 'S' to Goal node 'E'.



Input: 2D maze represented as a string and Edge Cost (in case of UCS and A*)

Output:

i. Path: the path to go from Start to End
ii. Full Path: the path of all visited nodes.
iii. Total Cost (only in UCS and A*)

The input and output are explained below.

```
Maze: 'S,.,.,#,.,.,. .,#,.,.,.,#,. .,#,.,.,.,.,. .,.,#,#,.,.,.
#,.,#,E,.,#,.'
```

- Maze is a string, rows are separated by **space** and columns are separated by **comma** ','.
- The board is read **row wise**, the nodes are numbered **0-based** starting
  the leftmost node.
- You have to create your own board as a 2D array of **Nodes**.

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


```
EdgeCost: [0, 15, 2, 100, 60, 35,
           30, 3, 100, 2, 15, 60,
           100, 30, 2, 100, 2, 2,
           2, 40, 30, 2, 2, 100,
           100, 3, 15, 30, 100, 2,
           100, 0, 2, 100, 30]
```

- Edge cost is a list, will be passed for UCS, AStar (Euclidian, Manhattan).
- Each Node has an edge value that represents the cost from any parent node to this node.

For Calculating the heuristic value apply the two methods:

1. Euclidean: Take the square root of the sum of the squares of the differences of the coordinates.
   - For example, if x = (a, b) and y=(c, d) , the Euclidean distance between x and y is

$$\sqrt{(a-c)^2+(b-d)^2}.$$

2. Manhattan: Take the sum of the absolute values of the differences of the coordinates.
   - For example, if x = (a, b) and y=(c, d) , the Euclidean distance between x and y is

$$|a-c|+|b-d|$$

Sample Run:

For BFS:

Input:

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

Output:

```
BFS Path: [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]
```

For DFS:

Input:

```
searchAlgo = SearchAlgorithms('S,.,.,#,.,.,. .,#,.,.,.,#,. .,#,.,.,.,.,. .,.,#,#,.,.,. #,.,#,E,.,#,.')
path, fullPath = searchAlgo.DFS()
print('**DFS**\nPath is: ' + str(path) + '\nFull Path is: ' + str(fullPath) + '\n\n')
```

Output:

```
DFS Path: [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]
```

For UCS:

Input:

```
searchAlgo = SearchAlgorithms('S,.,.,#,.,.,. .,#,.,.,.,#,. .,#,.,.,.,.,. .,.,#,#,.,.,. #,.,#,E,.,#,.', [0, 15, 2, 100, 60, 35, 30, 3
                                                                                                          , 100, 2, 15, 60, 100, 30, 2
                                                                                                          , 100, 2, 2, 2, 40, 30, 2, 2
                                                                                                          , 100, 100, 3, 15, 30, 100, 2
                                                                                                          , 100, 0, 2, 100, 30])
path, fullPath, TotalCost = searchAlgo.UCS()
print('** UCS **\nPath is: ' + str(path) + '\nFull Path is: ' + str(fullPath) + '\nTotal Cost: ' + str(
    TotalCost) + '\n\n')
```

Output:

```
UCS Path: [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]
Total Cost: 30
```

For A* using Euclidian Heuristic:

Input:

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

Output:

```
AstarEcludianHeuristic Path: [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]
Total Cost: 31.0
```

For A* using Manhattan Heuristic::

Input:

```
searchAlgo = SearchAlgorithms('S,.,.,#,.,.,. .,#,.,.,.,#,. .,#,.,.,.,.,. .,.,#,#,.,.,. #,.,#,E,.,#,.')
path, fullPath, TotalCost = searchAlgo.AStarManhattanHeuristic()
print('**ASTAR with Manhattan Heuristic **\nPath is: ' + str(path) + '\nFull Path is: ' + str(
    fullPath) + '\nTotal Cost: ' + str(TotalCost) + '\n\n')
```

Output:

```
AstarManhattanHeuristic Path: [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, 25, 19, 32, 32, 32, 31]
Total Cost: 10.0
```

The template code is explained below.

SearchAlgorithms.py file:

It contains two classes:

b. Class Node represents a cell in the board of game.

```
class Node:
    id = None  # Unique value for each node.
    up = None  # Represents value of neighbors (up, down, left, right).
    down = None
    left = None
    right = None
    previousNode = None  # Represents value of neighbors.
    edgeCost = None  # Represents the cost on the edge from any parent to this node.
    gOfN = None  # Represents the total edge cost
    hOfN = None  # Represents the heuristic value
    heuristicFn = None  # Represents the value of heuristic function

    def __init__(self, value):
        self.value = value
```

2) Class SearchAlgorithms:

I. Do not change class functions, parameters, or order
II. You can add any extra attributes, functions or classes you need as long as the main structure is left as it is.
III. Implement the given functions.

```
class SearchAlgorithms:
    """..."""
    path = []  # Represents the correct path from start node to the goal node.
    fullPath = []  # Represents all visited nodes from the start node to the goal node.
    totalCost = -1  # Represents the total cost in case using UCS, AStar (Euclidean or Manhattan)

    def __init__(self, maseStr, edgeCost=None):
        ''' maseStr contains the full board
         The board is read row wise,
        the nodes are numbered 0-based starting
        the leftmost node'''
        pass

    def DFS(self):
        # Fill the correct path in self.path
        # self.fullPath should contain the order of visited nodes
        return self.path, self.fullPath

    def BFS(self):
        ...
        return self.path, self.fullPath

    def UCS(self):
        # Fill the correct path in self.path
        # self.fullPath should contain the order of visited nodes
        return self.path, self.fullPath, self.totalCost

    def AStarEuclideanHeuristic(self):
        # Cost for a step is calculated based on edge cost of node
        # and use Euclidean Heuristic for evaluating the heuristic value
        # Fill the correct path in self.path
        # self.fullPath should contain the order of visited nodes
        return self.path, self.fullPath, self.totalCost

    def AStarManhattanHeuristic(self):
        ...
        return self.path, self.fullPath, self.totalCost
```