TCSS 435
Programming Assignment 1

Objective: Learn how to set-up and initialize our Maze GUI in Python.

Instructions: In this assignment you will learn how to setup the Maze GUI in Python required for our AI assignments. Follow the instructions step-by-step:

Step 1: Installing Maze Generator package
Download the file `pymaze.py` provided with this assignment in your working directory. The python-based Maze Generator is developed & created by Muhammad Ahsan Naeem.

Step 2: Creating a basic Maze
Create a source code file called `MazeRunner.py` and add the following code:

```
import pymaze as maze
m = maze.maze()   # creates a default maze of size 10x10
m.CreateMaze()    # default goal is set at (1,1) and pattern = horizontal
m.run()
```

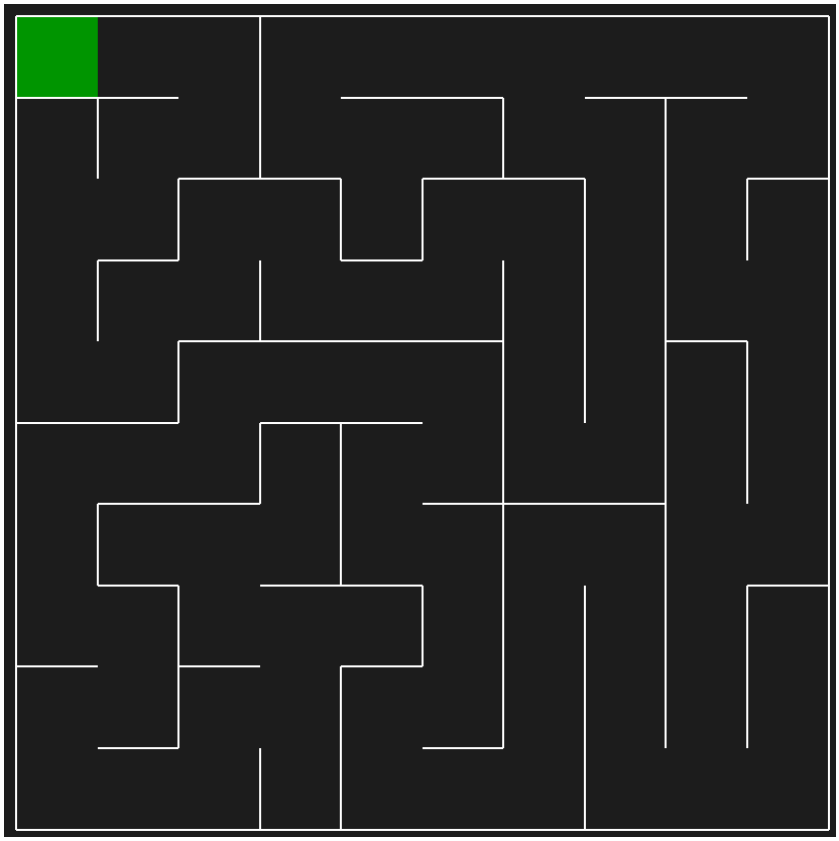

Figure 1: Resulting maze of size 10x10 with goal at cell (1,1)

Optional Parameters:

- Maze size can be changed by passing parameters (row, column) during the initialization of the maze object:

```
m = maze.maze(5,5)
```

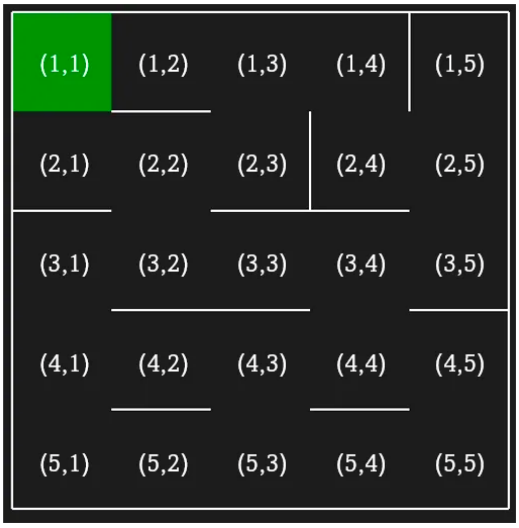

Figure 2: Indices of the Maze cells

- `CreateMaze()` parameters:
  - Goal: update the default goal to desired cell (row, column)

    ```
    m.CreateMaze(5,5)
    ```
  - Pattern: change the maze from horizontal (default) or vertical patterns (maze lines)

    ```
    m.CreateMaze(5,5, pattern= 'v')
    ```
  - Multiple paths: update the maze to hold multiple paths from any cell to goal (default: one path from any cell to goal)

    ```
    m.CreateMaze(loopPercent=100)
    ```
  - Theme: change the theme from dark (default) to light

    ```
    m.CreateMaze(theme=maze.COLOR.light)
    ```

To Do:
Update the source code to create a light themed maze of size 20x30 with goal cell randomly generated.

Step 3: Adding agents to the Maze
We can place agent (one or more) inside the Maze. An agent can be thought of as a physical agent like a robot or it can simply be used to highlight or point a cell in the maze. We add an agent by creating an agent object:

```
import pymaze as maze
m = maze.maze()    # creates a default maze of size 10x10
m.CreateMaze()     # default goal is set at (1,1) and pattern = horizontal
a = maze.agent(m) # default position of the agent is last cell (row,col)
m.run()
```

Optional Parameters for Agent class:

- Location of the agent: You can update the default position (bottom-right cell) of the agent by passing the row and column of the cell.

```
a = maze.agent(m,8,9)
```

- Goal of the agent: The default goal for the agent is the goal of the Maze. You can change the goal of the agent by setting `goal = (row, column)`.
- Size of the Agents: The default size of the agent is smaller than the cell dimensions. You can change the setting by setting `filled = True` and the agent will fill the whole cell.
- Shape of the agent: The default shape of the agent is square. You can change the shape by setting `shape = 'arrow'` and the agent will be arrowhead shaped.
- See the footprints: When you implement some search algorithms you can visualize the agent path in the maze by setting the `footprints = True`. Footprints is just the shape of the agent but with a different color shade.
- Color of the Agent: You can change the color of the agent by setting `color= maze.COLOR.red`

```
...
a = maze.agent(m,8,9,shape = 'arrow',footprints = True )
a.position = (7,9)
a.position = (6,9)
a.position = (5,9)
m.run()
```

Step 4: Moving Agent in the Maze

After creating a Maze and agent (one or more) inside the Maze, we can make the agent move on a specific path. We will use method `tracePath` from the maze class that takes one dictionary as the input argument. The key of the dictionary is the agent and the value is the path we want that agent to follow. The `tracePath` method will simulate the agent moving on the path.

There are three ways we can specify a path for the agent to follow:

- Path as a Dictionary: As shown above the path can be a dictionary with key-value pairs representing the movement from key-cell to value-cell.

```
...
a = maze.agent(m,8,9,shape = 'arrow',footprints = True )
path = {(8,9):(7,9), (7,9):(6,9)}
m.tracePath({a:path})
m.run()
```

- Path as List: There is also a possibility to provide the path as a List of cells. Then the agent will follow the path starting from the first cell inside the list to the last cell.

```
m.tracePath({a:[(8,9),(7,9),(6,9)]})
```

- Path as String: We can also provide the path as a string of movement directions (EWNS), e.g. 'EENWWSES' is a string of 8 steps for the agent to follow.

```
m.tracePath({a:'EENWWSES'})
```

To Do:
Create another agent in default shape (square) with footprints and set the color of the agent to be red. Position the agent at a random cell. Make sure the agent does not occupy the same position as the goal and the first agent. Move the agent 2-3 steps in each direction. You can trace agent path using dictionary/list/string.

Optional Parameters for `tracePath` method:

- Kill the agent: It is possible to kill the agent after it completes the path. By setting the paramenter `kill=True` and the agent will be killed after 300 milliseconds after completing the path.
- Movement Speed: We can control the movement speed of the agent using the parameter `delay` having the default value of 300 milliseconds. It is a time delay between the movement steps of the agent.
- Mark some cells: For different demonstrations, it might be needed to mark a few cells. For that, there is an option of `showMarked=True` and any cell present inside the list of maze `markCell` will be marked if the agent passes through that cell.

A few attributes of the Maze class that you should know to implement some search algorithms are:

- Rows: `m.rows` gives the number of rows of the maze `m`.
- Columns: `m.cols` gives the number of columns of the maze `m`.
- Grid: `m.grid` is a list with all cells from (1,1) to last.
- Map of the Maze: A Maze is generated randomly. It is important to know the information of different opened and closed walls of the Maze. That information is available in the attribute `maze_map`. It is a dictionary with the keys as the cells of the Maze and value as another dictionary with the information of the four walls of that cell in four directions: East, West, North and South.

```
print(m.maze_map)
{(1, 1): {'E': 1, 'W': 0, 'N': 0, 'S': 1}, (2, 1): {'E': 0, 'W': 0, 'N': 1, 'S': 1},
(3, 1): {'E': 0, 'W': 0, 'N': 1, 'S': 1}, (4, 1): {'E': 1, 'W': 0, 'N': 1, 'S': 0},
(5, 1): {'E': 1, 'W': 0, 'N': 0, 'S': 0}, (1, 2): {'E': 1, 'W': 1, 'N': 0, 'S': 1},
(2, 2): {'E': 1, 'W': 0, 'N': 1, 'S': 0}, (3, 2): {'E': 0, 'W': 0, 'N': 0, 'S': 1},
(4, 2): {'E': 0, 'W': 1, 'N': 1, 'S': 1}, (5, 2): {'E': 1, 'W': 1, 'N': 1, 'S': 0},
(1, 3): {'E': 0, 'W': 1, 'N': 0, 'S': 0}, (2, 3): {'E': 0, 'W': 1, 'N': 0, 'S': 1},
(3, 3): {'E': 0, 'W': 0, 'N': 1, 'S': 1}, (4, 3): {'E': 1, 'W': 0, 'N': 1, 'S': 0},
(5, 3): {'E': 1, 'W': 1, 'N': 0, 'S': 0}, (1, 4): {'E': 1, 'W': 0, 'N': 0, 'S': 1},
(2, 4): {'E': 0, 'W': 0, 'N': 1, 'S': 1}, (3, 4): {'E': 0, 'W': 0, 'N': 1, 'S': 1},
(4, 4): {'E': 0, 'W': 1, 'N': 1, 'S': 0}, (5, 4): {'E': 0, 'W': 1, 'N': 0, 'S': 0},
(1, 5): {'E': 0, 'W': 1, 'N': 0, 'S': 1}, (2, 5): {'E': 0, 'W': 0, 'N': 1, 'S': 1},
(3, 5): {'E': 0, 'W': 0, 'N': 1, 'S': 1}, (4, 5): {'E': 0, 'W': 0, 'N': 1, 'S': 1},
(5, 5): {'E': 0, 'W': 0, 'N': 1, 'S': 0}}
```

Submission Guideline: Upload your updated source code file `MazeRunner.py` to the assignment submission link.