Artificial Intelligence
Lab 6: Maze Solving with BFS and DFS (Guidelines)

1. Recall the algorithm of BFS and DFS (from the last theory class)
2. Define a complex maze as a 2D list.
   a. ‘S’ should denote start, ‘G’ should denote the goal
   b. ‘0’ denotes free cells, ‘1’ denotes walls
   c. Example:

```
maze = [
    ['S', 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0],
    [1,   0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0],
    [0,   0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0],
    [0,   1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0],
    [0,   1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
    [1,   1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0],
    [0,   0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
    [0,   1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0],
    [0,   1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0],
    [0,   1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0],
    [0,   0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0],
    [1,   1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0],
    [0,   1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0],
    [0,   1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0],
    [0,   0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0],
     [1,   1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1],
    [0,   0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0],
    [0,   1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [0,   0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1,   1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 'G'],
]
```

3. We need to convert the maze from mixed data structure(strings and integers) to uniform numerical format. So, Create a function “**to_numeric_grid(maze)**” that converts the maze to numpy array with the following values:
   a. ‘S’ (start) = 2, ‘G’(goal) = 3, ‘0’(free cell) = 0, ‘1’(wall) = 1
4. Create a function **“find_pos(value)”** to find the positions of ‘S’ (start) and ‘G’ (goal) in the maze
5. Visualize the Maze (use matplotlib) before we start solving.

6. We need to find all legal moves in the cell-skipping walla and boundaries. So, create a function “**get_neighbors(r,c)”.** Use the following hints:
   a. There are 4 cardinal directions: right, left, down, up. So we attempt to move one step in each of the 4 directions

   b. Compute the neighbor’s position. [Hint: r,c = r+dr, c+dc]
   c. Check if the neighbor is within bounds. [Hint: 0 < r < ROWS, same for the columns]
   d. Yield those moments that satisfy the above check.

7. Create a single function to implement **DFS or BFS with visualization(animation).** Use the following hints:
   a. Use “cmap = matplotlib.colors.ListedColormap()” for picking colors.
      i. White = free, Black = Wall
      ii. Orange = Start, Red = Goal
      iii. Green color = cells visited during search
      iv. Blue color = cells that are part of the final path
   b. Initialize the frontier:
      i. BFS (Data Structure used = Queue): Use “deque” python object for queue behavior
      ii. DFS (Data structure used = Stack): Use “list” python object for stack behavior [Hint: A list can be used as a stack, if you use .pop() to remove the recently added (or “rightmost”) element in the list.
   c. Implement the search loop. If BFS gets the first element(FIFO. If DFS, gets the last element (LIFO).
      i. Hint: *“while frontier: (r, c), path = frontier.popleft() if algorithm == 'bfs' else frontier.pop()”*
   d. Check the visited state and color the path as green color
   e. Check the goal state and color the path as blue color
8. Call the above function.
   a. The user should be prompted to choose 2 options: “bfs” or “dfs”.
   b. If the user inserts uppercase letters, automatically convert them to lowercase.
   c. Above function should show the animation. The figure (animation) should close when the user presses a button (e.g. pressing a “Q” button on the keyboard).

9. Bonus Points:
   a. Log the path length and steps taken
   b. Add maze randomization or multiple maze options
   c. Show total visited nodes and execution time