Artificial Intelligence
Lab 7: Maze Generation with Prim's Algorithm and Maze Solving with A* Search (Guidelines)

1. Recall the algorithm of Greedy first and A* Search (from the last theory class)
2. Create a maze with Prim's algorithm. Use the following hints:
    a. Use grid of odd size (e.g. 21x21)
    b. Treat the entire grid as walls initially
    c. Use the following Prim's algorithm to carve passages:
        i. Initialize Maze: You can choose a random odd cell as the starting point. Mark this cell as a passage/way (0)
        ii. Add Frontier walls: From the starting cell, add all 2-cell-away neighbors to the wall list. Each wall is stored as (r1, c1, r2, c2) where (r1, c1) is a passage, (r2, c2) is a wall's neighbor cell
        iii. While wall list is not empty:
            1. Randomly pick a wall (r1, c1, r2, c2) from the list
            2. If (r2,c2) is a wall and has only one adjacent passage:
                a. Convert the wall between them (midpoint) to a passage (0) Hint: mid = ((r1 + r2) // 2, (c1 + c2) // 2)
                b. Convert (r2, c2) to a passage
                c. Add its neighbors (2 steps away) to the wall list.
        iv. Repeat until all cells are reachable.
    d. Function to implement: generate_maze(rows, cols). Returns a numpy array maze of 0(paths) and 1(walls)

3. Start and Goal to the maze (recall the last lab)
    a. Example: Place Start S at (1,1), goal G at (ROW-2, COL-2)
    b. Encode: Start=2, Goal=3
    c. Function to implement: place_start_goal(maze). Returns a modified maze
4. Plot the original maze

5. Implement a function to get all legal moves in the cell. Create a function get_neighbors(r.c). [Just like the previous lab]

6. Implement a function to calculate Manhattan distance between a and b
    a. Hints: |x1-x2| + |y1-y2|

7. Implement A* Algorithm in a function. Use the hints:
    a. Use color map: 0=white(free), 1=black(wall), 2=orange(start), 3-red (goal), 4=green(visited), 5=blue(final path)
    b. Use Manhattan distance as heuristic
    c. Track cost and heuristic using a priority queue (min-heap).
    d. Maintain a visited set to avoid cycles!
    e. For Visualization: Green = visited, Blue = Final path

    f. Function to implement: a_star(maze, start, goal)

8. Visualize the search procedure and the solution. Use animation if possible.

9. Bonus task: Log number of visited nodes, path length and execution time