Maze solving using A* algorithm

Project 2

Maze solve using A* algorithm
CSE 404 (Lab)

Submitted by:

Khandakar Mahedi Hasan
21101031

Md. Bijoy
21101033

Niamur Rashid Niloy
21101013

Submitted to:

Dr. Nasima Begum

Assistant Prof.

Department of Cse

Project Overview

This project is a Python-based implementation of the A* pathfinding algorithm on a grid, equipped with visualization capabilities. It allows users to input grid details, blocked cells, start and goal nodes, and calculates the optimal path using either Manhattan or Diagonal heuristics.

Features

- File and Manual Input: Supports two input methods—file-based for predefined grids and manual input for interactive grid creation.
- Heuristic Options: Manhattan and Diagonal heuristics.
- Visualization: Plots the grid, obstacles, path, and important steps using matplotlib.
- Environment Flexibility: Designed for compatibility with local environments and Google Colab.

Project Structure

The project code consists of several functions for input handling, heuristic calculation, pathfinding, and visualization.

1. file_input()

Handles file-based input for grid dimensions, obstacles, start, and goal positions.

- Arguments: File input (global: prompts user for filename) or (colab: request for file input).
- Returns: (grid_size_x, grid_size_y, blocked_obstacles, start, goal) if input is valid, otherwise an error message.
- Environment Support: Designed for both Google Colab and other environments.

2. manual_input()

Allows users to manually enter grid details, obstacles, and start/goal positions.

- Arguments: Terminal input (grid_size_x, grid_size_y, blocked_obstacles, start, goal).
- Returns: (grid_size_x, grid_size_y, blocked_obstacles, start, goal) if input is valid, otherwise an error message.

3. calculate_open_grid(grid_size_x, grid_size_y, blocked_obstacles()

Generates a list of open cells (non-blocked) in the grid.

- Arguments:
    - grid_size_x: Number of horizontal cells.
    - grid_size_y: Number of vertical cells.
    - blocked_obstacles: List of blocked cells.
- Returns: List of open cell coordinates.

4. calculate_heuristic_value_Manhattan(open_grid, goal_node()

Calculates the Manhattan distance heuristic for each open cell relative to the goal.

- Arguments:
    - open_grid: List of open cells.
    - goal_node: Goal node coordinates.
- Returns: (heuristic_dict, "Manhattan", time_taken)

5. calculate_heuristic_value_Diagonal(open_grid, goal_node()

Calculates the diagonal distance heuristic.

- Arguments:
    - open_grid: List of open cells.
    - goal_node: Goal node coordinates.
- Returns: (heuristic_dict, "Diagonal", time_taken)

6. calculate_adjacent_node(node, grid_size_x, grid_size_y, blocked_obstacles()

Determines valid adjacent nodes for a given cell, avoiding blocked cells.

- Arguments:
    - node: Current cell coordinates.
    - grid_size_x, grid_size_y: Grid dimensions.
    - blocked_obstacles: List of blocked cells.
- Returns: List of valid adjacent nodes.

7. get_adjacent_nodes(open_grid, grid_size_x, grid_size_y, blocked_obstacles()

Generates a dictionary of all open cells and their valid adjacent nodes.

- Arguments: Similar to calculate_adjacent_node.
- Returns: Dictionary of each cell with its adjacent nodes.

8. calculate_path_cost(start_node, values()

Computes path cost for each adjacent node using Euclidean distance.

- Arguments:
    - start_node: Current cell.
    - values: List of adjacent cells.
- Returns: List of tuples containing each adjacent node and its path cost.

9. get_path_cost(adjacent_nodes)

Creates a dictionary of path costs between all adjacent cells.

- Arguments: adjacent_nodes (dictionary of cells and their adjacent nodes().
- Returns: Dictionary of path costs for each node pair.

10. node_iteration(start, goal, heuristic_value_dict, path_cost_dict()

Runs the A* algorithm to find the optimal path from start to goal node.

- Arguments:
    - start, goal: Start and goal cell coordinates.
    - heuristic_value_dict: Dictionary of heuristic values.
    - path_cost_dict: Dictionary of path costs.
- Returns: (path, total_time, final_cost) if path is found.

11. plot_path_grid(grid_size_x, grid_size_y, blocked_obstacles, start, goal, path, name, time_took, final_path_cost()

Plots the grid with obstacles, start, goal, and optimal path.

- Arguments:
    - grid_size_x, grid_size_y: Grid dimensions.
    - blocked_obstacles: List of blocked cells.
    - start, goal: Start and goal cell coordinates.
    - path: Optimal path from start to goal.
    - name: Heuristic name.
    - time_took, final_path_cost: Pathfinding time and cost.
- Returns: None (displays plot).

12. plot_heuristic_grid(grid_size_x, grid_size_y, blocked_obstacles, start, goal, heuristic, name, time_took_to_calculate()

Visualizes the heuristic values across the grid.

- Arguments: Similar to plot_path_grid, plus heuristic.
- Returns: None (displays plot).

Usage Instructions

- Choose either manual input or file input.
- File input has colab environment exclusive only or global environment option.
- The code automatically calculates open path, heuristic values(Manhattan, Diagonal distance), optimal shortest path is calculated using A* algorithm.
- For visulization the maze is plotted with key features.

Dependencies

- numpy
- matplotlib

For Google Colab users, ensure file upload permissions are granted when using file input.

Inputs

Formate for input

- grid_size_x
- grid_size_y
- number of blocked_obstacles
    - coordination of blocked_obstacles for n numbers
- start coordination
- goal coordination

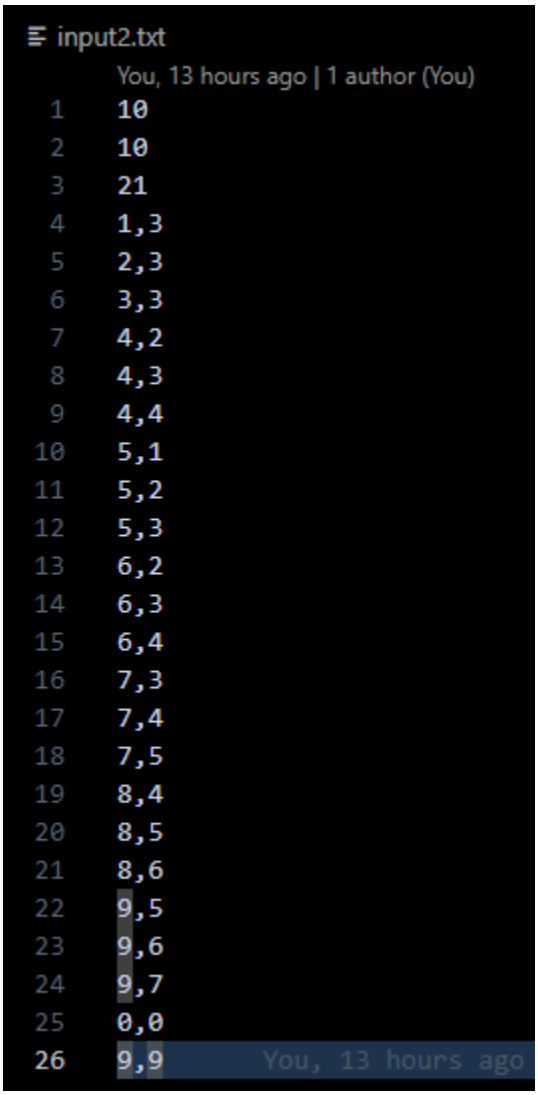
Manual input sample:
First enter 2 for entering manual mode.

```
"1" for using file input mode.
"2" for manual input using terminal mode.
        Select:
```

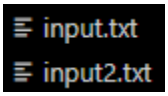
Then enter appropriate values shown in “Formate for input” section.

```
"1" for using file input mode.
"2" for manual input using terminal mode.
        Select: 2
Enter the size of the grid in horizontal(x) direction: 10
Enter the size of the grid in vertical(y) direction: 10
Enter the number of blocked obstacles: 20
        Enter obstacle 1 coordination: 2,0
        Enter obstacle 2 coordination: 4,0
        Enter obstacle 3 coordination: 9,0
        Enter obstacle 4 coordination: 6,1
        Enter obstacle 5 coordination: 8,1
        Enter obstacle 6 coordination: 2,2
        Enter obstacle 7 coordination: 4,3
        Enter obstacle 8 coordination: 8,3
        Enter obstacle 9 coordination: 1,4
        Enter obstacle 10 coordination: 4,4
        Enter obstacle 11 coordination: 6,4
        Enter obstacle 12 coordination: 7,5
        Enter obstacle 13 coordination: 0,6
        Enter obstacle 14 coordination: 3,6
        Enter obstacle 15 coordination: 9,6
        Enter obstacle 16 coordination: 0,7
        Enter obstacle 17 coordination: 7,7
        Enter obstacle 18 coordination: 3,8
        Enter obstacle 19 coordination: 1,9
        Enter obstacle 20 coordination: 5,9
**Enter start node coordination: 0,0
##Enter goal node coordination: 9,9
```

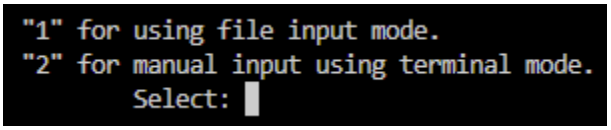
File input:
File input accepts text format file having appropriate values shown in “Formate for input” section.



The extension for the file must be .txt for the text file.



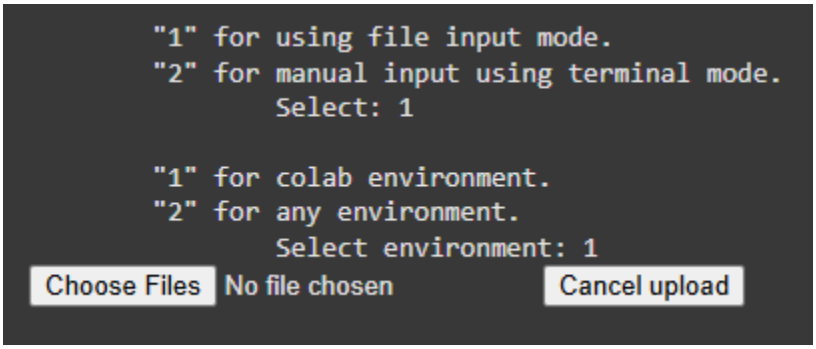
To enter file input mode first enter 1 after running the code in the terminal.



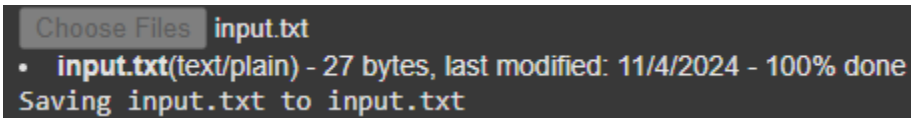
For google colab exclusive enter 1 again in the terminal.

Upload file in colab:

When colab environment to request file can Choose File or Cancel the request. When choosing the file from your local device it requires the text file having values shown in “Formate for input” section.



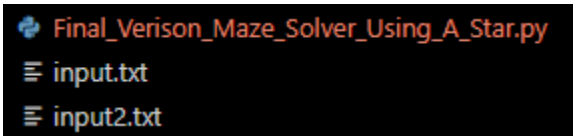
After choosing file the code will be executed



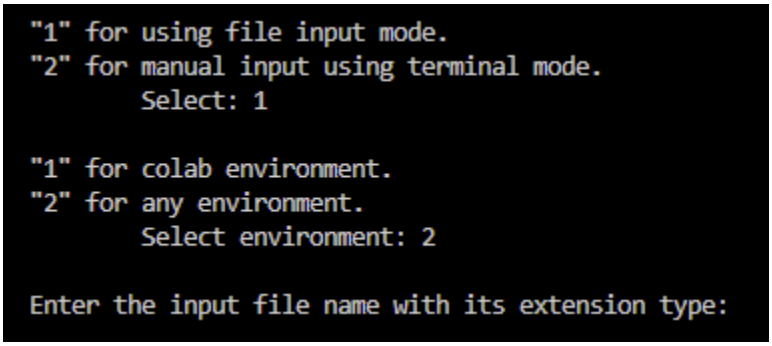
File input environment have another option for global environment. It works on colab, or cloud or local os.

Environment setup for local VS code:

The text file needs to be located in the same folder that this code is located.



When selecting option 2 in the environment it requests input file name with extension type.



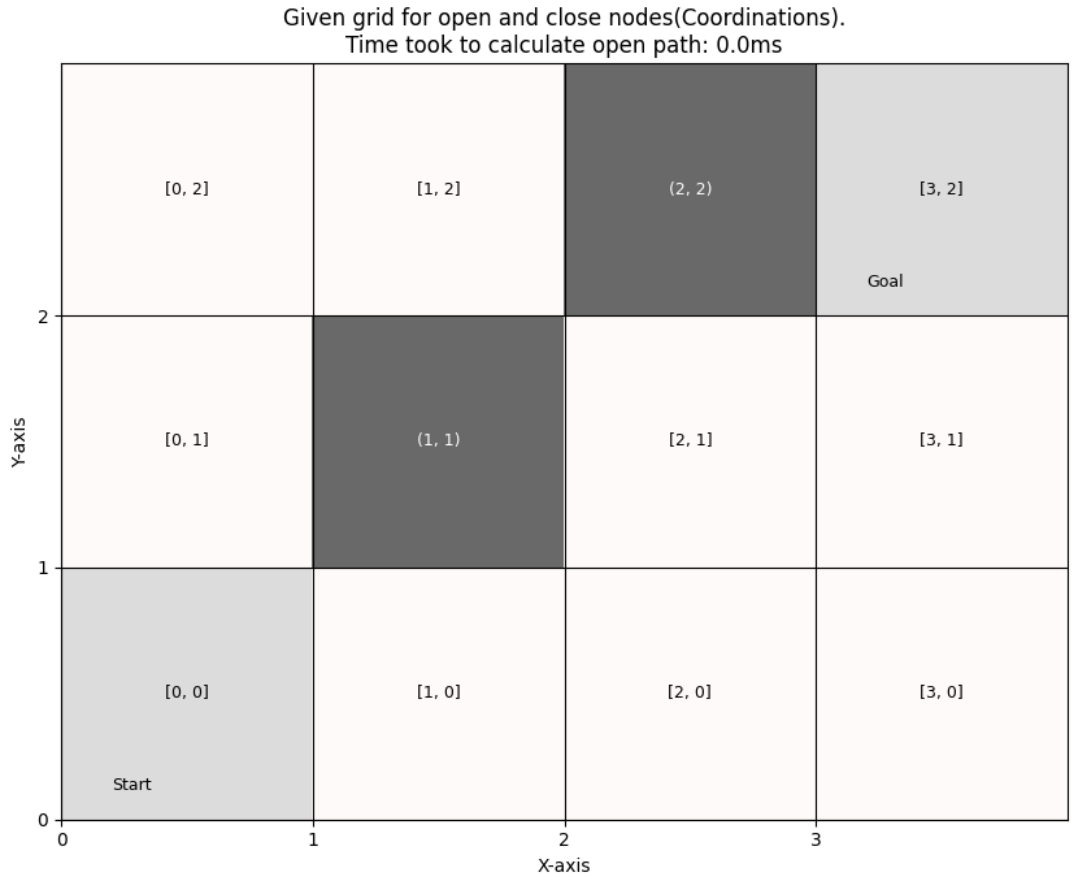
Entering the input file name with its extension type executes the program.

```
"1" for using file input mode.
"2" for manual input using terminal mode.
        Select: 1

"1" for colab environment.
"2" for any environment.
        Select environment: 2

Enter the input file name with its extension type: input.txt
```

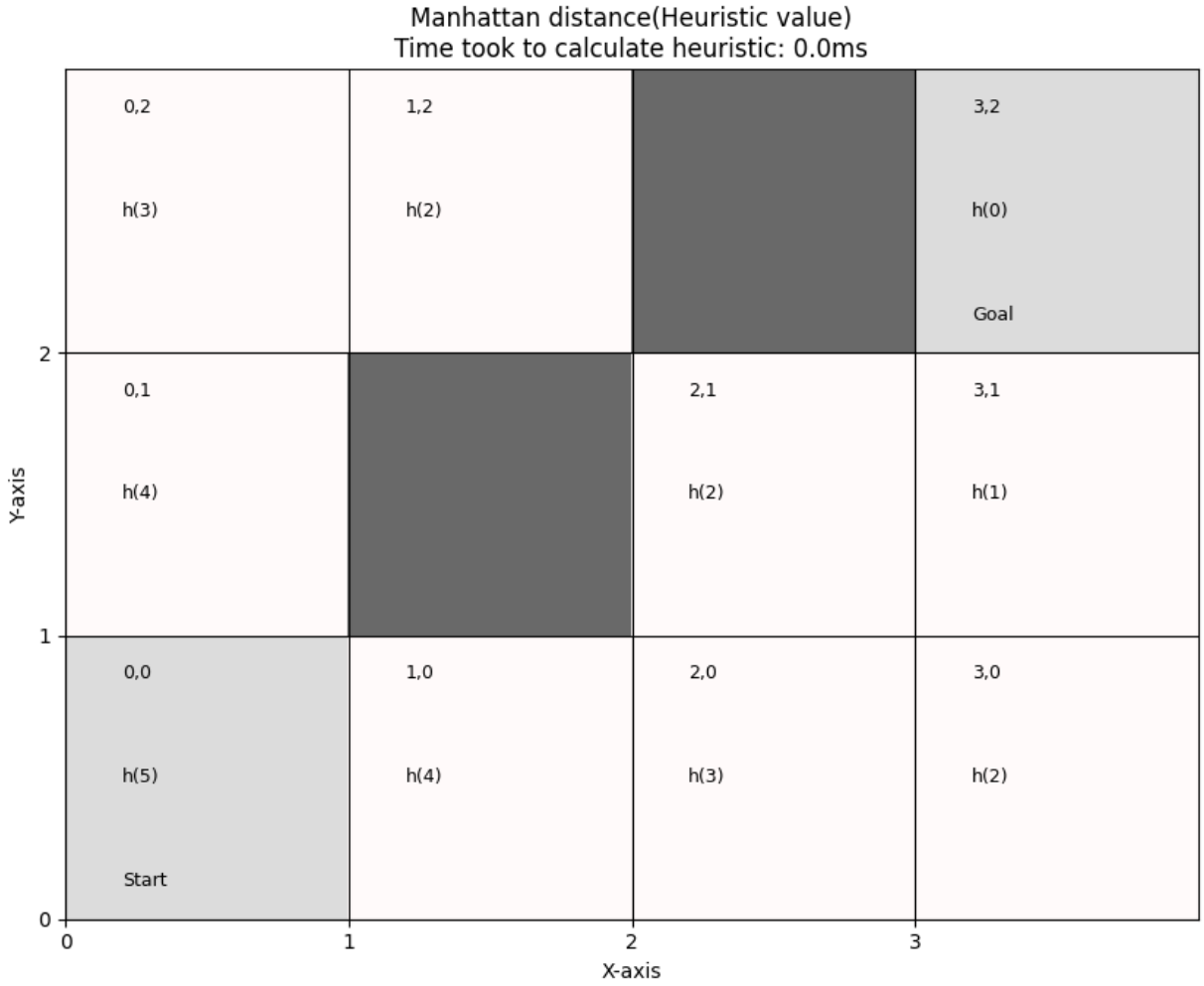
Output

There are 5 graphical outputs:

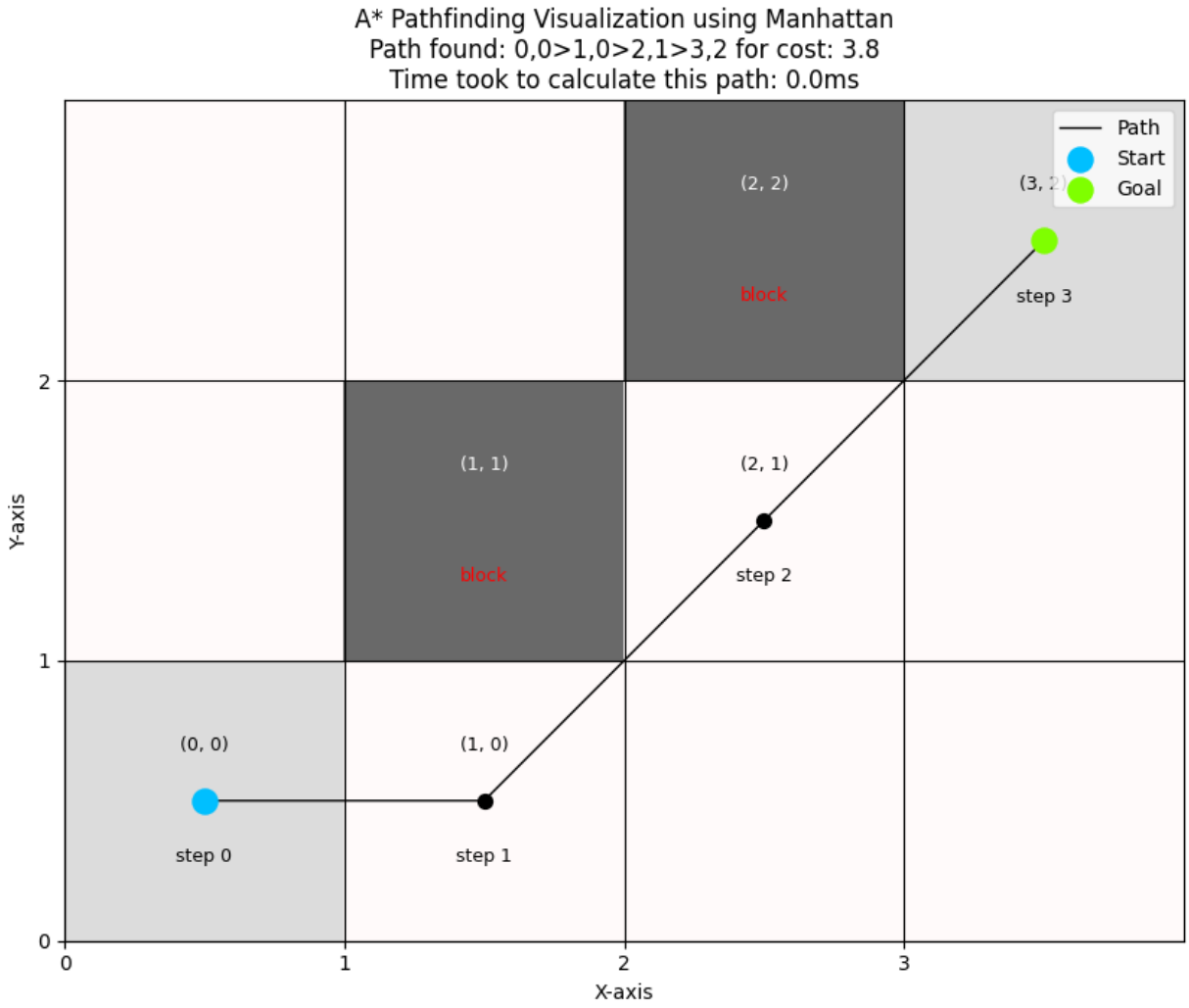
Output no 1: First output plots maze with its actual grid size having blocked and open cells/tiles using values from input sample. The time took to plot this grid is also shown in the top.



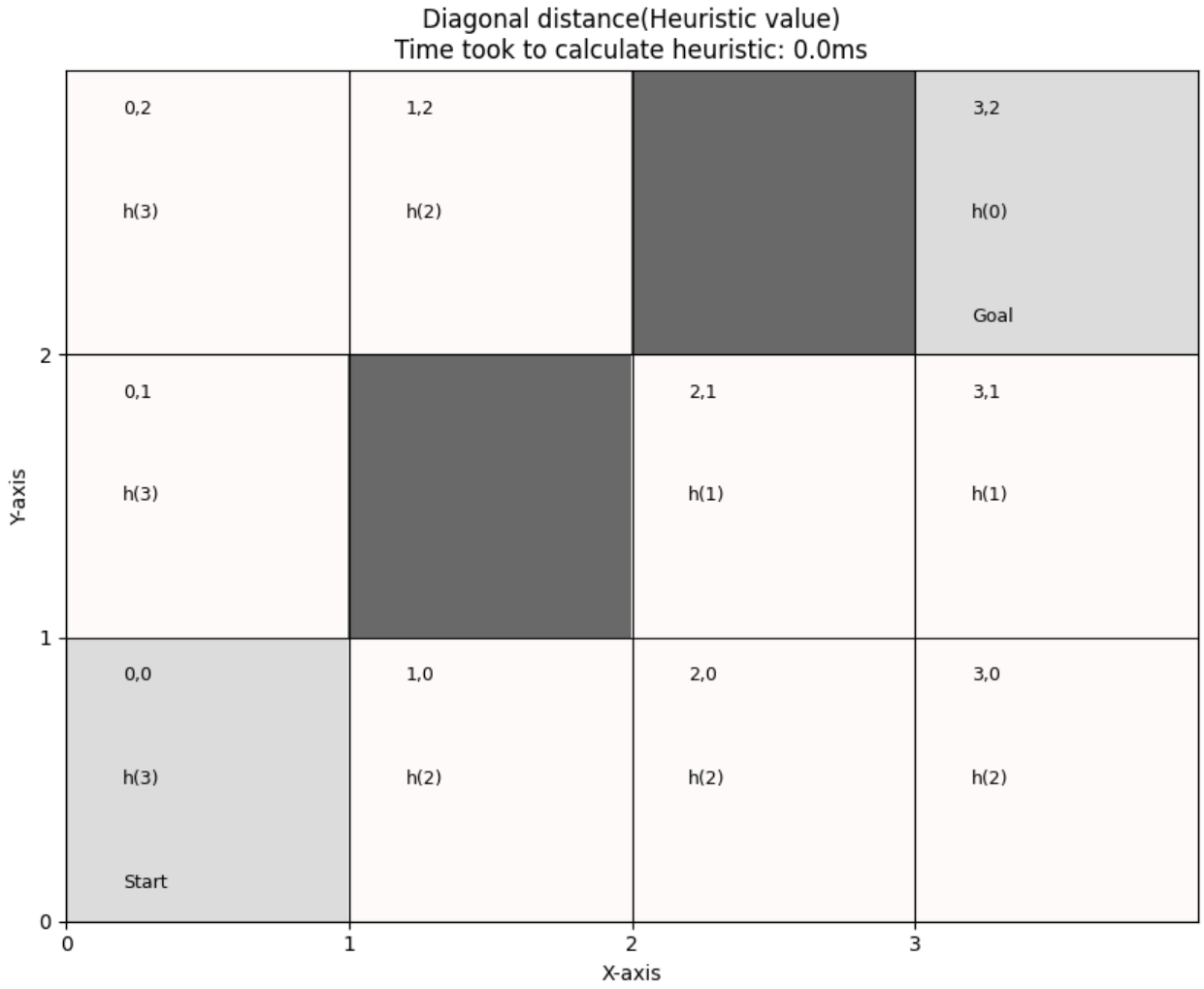
Output 2: This output plots the grid with heuristic value using Manhattan distance. Time took to calculate heuristic is also shown in milliseconds.



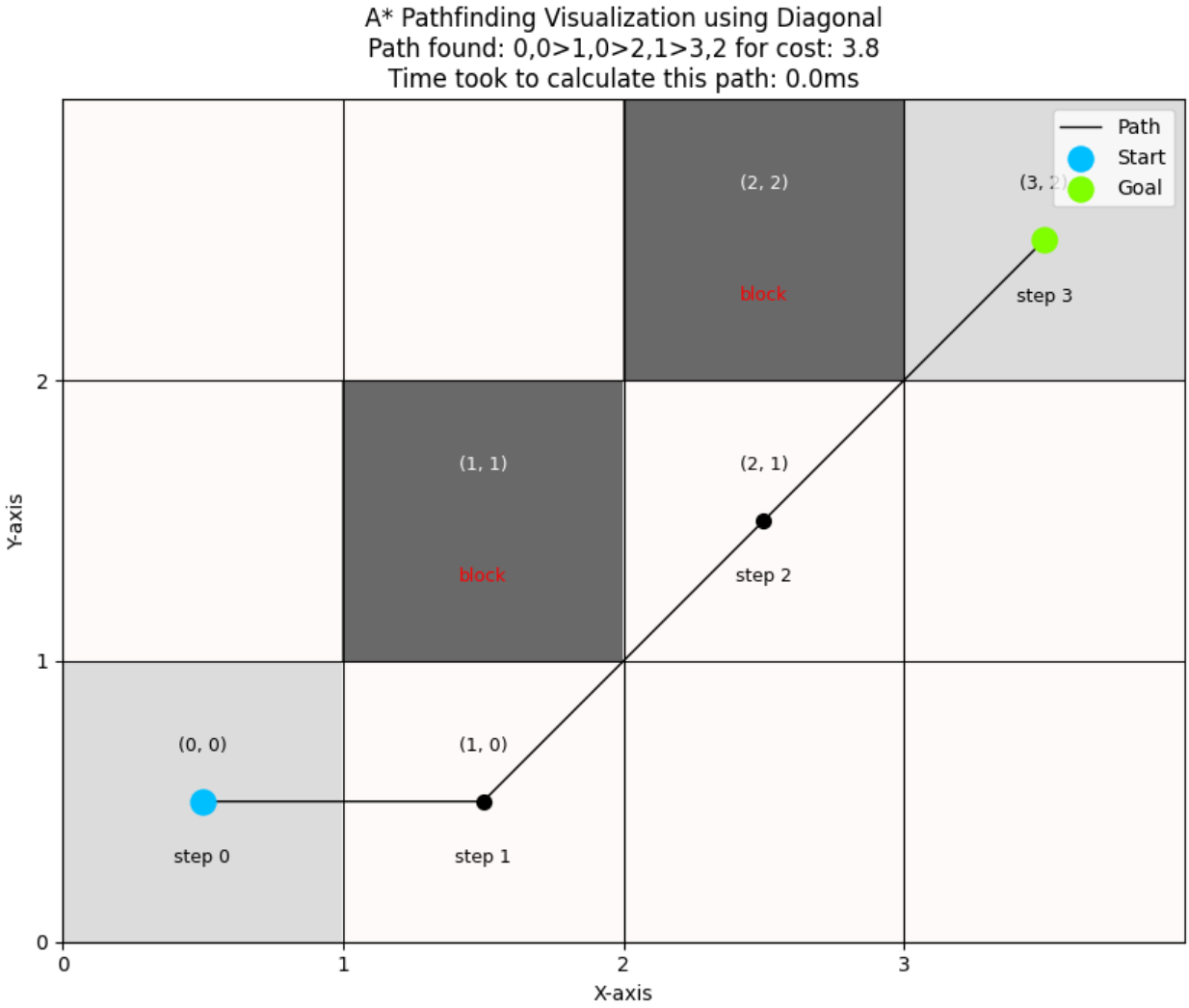
Output 3: This output visualizes shortest path calculated using Manhattan heuristic, path cost with help of A* algorithm. This plot also shows time took to calculate this path along with shortest adjacent node and how many steps took to calculate this path.



Output 4: This output plots the grid with heuristic value using Diagonal distance. Time took to calculate heuristic is also shown in milliseconds.



Output 4: This output visualizes shortest path calculated using Manhattan heuristic, path cost with help of A* algorithm. This plot also shows time took to calculate this path along with shortest adjacent node and how many steps took to calculate this path.



Output on terminal displays adjacent path and adjacent path cost.

```
Path cost:
{'0,0': [['0,1', 1.0], ['1,0', 1.0]], '1,0': [['0,0', 1.0], ['0,1', 1.4], ['2,0', 1.0], ['2,1', 1.4]], '2,0':
[['1,0', 1.0], ['2,1', 1.0], ['3,0', 1.0], ['3,1', 1.4]], '3,0': [['2,0', 1.0], ['2,1', 1.4], ['3,1', 1.0]], '
0,1': [['0,0', 1.0], ['0,2', 1.0], ['1,0', 1.4], ['1,2', 1.4]], '2,1': [['1,0', 1.4], ['1,2', 1.4], ['2,0', 1.
0], ['3,0', 1.4], ['3,1', 1.0], ['3,2', 1.4]], '3,1': [['2,0', 1.4], ['2,1', 1.0], ['3,0', 1.0], ['3,2', 1.0]]
, '0,2': [['0,1', 1.0], ['1,2', 1.0]], '1,2': [['0,1', 1.4], ['0,2', 1.0], ['2,1', 1.4]], '3,2': [['2,1', 1.4]
, ['3,1', 1.0]]}


Path found: 0,0>1,0>2,1>3,2 in 0.0s
Path found: 0,0>1,0>2,1>3,2 in 0.0s
```

Technical difficulties:

- Ensuring correct input format.
- Ensuring correct coordination.
- Calculating adjacent path cost.
- Calculating heuristic.
- Selecting cheapest cost using path cost and heuristic.
- Plotting grid in correct orientation and path.
- Plotting large grid.