Project Report: Visualization of Pathfinding Algorithms

1. Introduction

This project involves the development of a graphical application using Tkinter in Python. Its purpose is to implement and visualize, in real time, two pathfinding algorithms: **A*** and **Dijkstra**. The application allows users to configure various parameters—such as the algorithm selection, heuristic function, and whether diagonal movement is permitted—to observe how the search process evolves in a grid environment.

2. Objectives

- **Comparison and Analysis of Algorithms:**
  Enable users to compare A* and Dijkstra by highlighting the differences between using a heuristic (A*) and relying solely on accumulated cost (Dijkstra).
- **Interactive Visualization:**
  Provide a real-time graphical visualization of the search process. This helps users understand which nodes are expanded and how the frontier evolves as the algorithm progresses.
- **Flexible Configuration:**
  Allow customization of the heuristic function (choosing between Manhattan and Euclidean) and control over movement directions via a checkbox that enables or disables diagonal movements. This design ensures that the heuristic function remains independent of the actual movement directions.
- **User Interactivity and Customization:**
  Let users select start and goal points, as well as modify terrain types (e.g., water, wall, grass), so that the impact of different configurations on the search process can be clearly observed.

3. General Description and Design

3.1. Key Features

- **Algorithm Selection:**
  Users can choose between A* and Dijkstra from a drop-down menu. While A* leverages a heuristic to direct the search toward the goal, Dijkstra's algorithm ignores the heuristic by effectively using a value of zero.
- **Heuristic Configuration:**
  The application supports two heuristic functions—Manhattan and Euclidean. The selected heuristic influences the cost estimation to the goal but does not affect the set of possible movement directions.

- **Movement Directions:**
  A checkbox controls whether diagonal movements are allowed. This option is entirely independent of the heuristic function, ensuring that the choice of heuristic does not inadvertently enable diagonal moves.
- **Real-Time Visualization:**
  During the search, the grid is updated to indicate:
  - **Expanded Nodes:** Displayed in light gray to show nodes that have been visited.
  - **Frontier Nodes:** Shown in cyan to highlight nodes that have been added to the priority queue.
  - **Final Path:** Once the goal is reached, the computed path is displayed in orange (excluding the goal cell to preserve its identification).

3.2. Code Structure

- **Imports and Configuration:**
  The project uses standard libraries such as Tkinter for the GUI, `heapq` for managing the priority queue, and NumPy for Euclidean distance calculations. The terrain configuration—specifying colors and weights—is defined in a dictionary, allowing for easy customization.
- **PathfindingApp Class:**
  This class encapsulates the entire application functionality. Key methods include:
  - `draw_grid`: Renders the grid based on the current state.
  - `on_click`: Handles user interactions for setting start/goal positions and modifying terrain.
  - `find_path` and `calculate_path`: Contain the logic for executing the search algorithms (A* or Dijkstra) and managing the real-time visualization.
  - Additional methods handle updates for algorithm selection, heuristic choice, diagonal movement toggling, and terrain setting.
- **Separation of Heuristic and Movement:**
  The design ensures that the heuristic function does not dictate the movement directions. The checkbox exclusively controls whether diagonal moves are considered, thereby maintaining a clear separation between cost estimation and movement configuration.

4. Development and Implementation

4.1. Search Process and Visualization

- **A* vs. Dijkstra:**
  - **A* Algorithm:** Prioritizes nodes based on the sum of the accumulated cost and the heuristic estimate to the goal. This targeted approach reduces unnecessary node expansions.
  - **Dijkstra's Algorithm:** By setting the heuristic to zero, the algorithm relies solely on the accumulated cost, resulting in a more uniform, radial exploration of the grid.
- **Real-Time Animation:**
  The application uses `time.sleep(DELAY)` and `self.canvas.update()` to

introduce short pauses and update the GUI at each iteration. This provides users with a clear, step-by-step visualization of:

- Nodes being expanded (colored light gray).
- Neighbors being added to the frontier (colored cyan).

4.2. User Interface

The interface is designed for clarity and ease of use:

- **Dropdown Menus:** For selecting the algorithm (A* or Dijkstra) and the heuristic function (Manhattan or Euclidean).
- **Checkbox for Diagonals:** Controls whether diagonal movements are allowed.
- **Interactive Grid:** Users can click on the grid to set start and goal points and to modify the terrain.
- **Dynamic Visualization:** As the algorithm runs, the grid updates in real time to reflect the search process.

5. Results and Analysis

The implemented application clearly demonstrates:

- **Efficiency Differences:**
  A* tends to focus the search in the direction of the goal when using an appropriate heuristic, thus reducing the number of nodes evaluated compared to Dijkstra’s algorithm, which explores more broadly.

- **Impact of Diagonal Movements:**
  Allowing diagonal movement can result in shorter paths. However, it may also yield routes that are less natural in some scenarios. The clear separation between the heuristic and movement options ensures that users can control these aspects independently.

6. Conclusions

Educational Value

The application is a useful tool for understanding the fundamental differences between A* and Dijkstra. The real-time visualization aids in comprehending how each algorithm expands nodes and searches for a path.

Clear Separation of Concerns:

By decoupling the heuristic function from the movement directions, the project ensures a more accurate and flexible implementation of the algorithms.