### **Robot Center Navigation**

#### **Introduction**

This project demonstrates a robot navigation system using the A\* search algorithm implemented in Pygame. The robot navigates from a randomly chosen start point to the center of a user-defined rectangular pillar on a grid, avoiding the pillar. The application takes user input for pillar vertices using a Tkinter GUI and visualizes the robot's pathfinding process.

#### **Technologies Used**

1. **Pygame**: A cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries.
2. **Tkinter**: The standard GUI library for Python, used here to create a simple interface for user input.
3. **Python**: The primary programming language used for implementing the logic and algorithms.

#### **Project Structure**

1. **Initialization**:
   * initialize\_pygame: Sets up the Pygame window with specified dimensions and initializes the display.
   * load\_robot\_image: Loads and scales the robot image to fit within the grid cells.
   * load\_grass\_image: Loads and scales the grass image to fit within the grid cells.
2. **A\* Search Algorithm**:
   * heuristic: A function to estimate the cost from the current node to the end node, using the Manhattan distance.
   * reconstruct\_path: Reconstructs the path from the start node to the end node once the end node is reached.
   * a\_star\_search: Implements the A\* search algorithm to find the shortest path from the start to the end node, avoiding obstacles.
3. **Grid Management**:
   * draw\_grid: Draws the grid and cells on the Pygame screen, displaying grass for free cells and yellow for obstacles.
   * validate\_path: Ensures that the generated path is within the bounds and does not collide with obstacles.
4. **Main Logic**:
   * main: The main function that initializes the game, sets up obstacles, finds start and end points, and handles the game loop.
   * get\_input: A Tkinter function to get user input for the obstacle vertices.

#### **Explanation of the A\* Search Algorithm**

The A\* search algorithm is a popular pathfinding and graph traversal algorithm. It is widely used because of its performance and accuracy. A\* efficiently finds the shortest path from a start node to an end node by combining the features of Dijkstra's Algorithm and a heuristic approach.

* **Heuristic**: A\* uses a heuristic to guide its search. The heuristic used here is the Manhattan distance, which is the sum of the absolute differences between the current node's coordinates and the goal node's coordinates. This heuristic helps the algorithm prioritize nodes that appear to be closer to the goal.
* **G Score**: The cost of the path from the start node to the current node.
* **F Score**: The estimated total cost of the path through the current node to the end node (G Score + Heuristic).

A\* maintains two sets:

* **Open Set**: Nodes that need to be evaluated.
* **Closed Set**: Nodes that have already been evaluated.

The algorithm follows these steps:

1. **Initialization**: Start by adding the initial node to the open set.
2. **Evaluation**: Remove the node with the lowest F Score from the open set and evaluate it.
3. **Neighbor Examination**: For each neighbor of the current node, calculate the tentative G Score. If this G Score is lower than any previously recorded G Score, record it as the best path to the neighbor and update the F Score.
4. **Goal Check**: If the current node is the goal node, reconstruct the path using the recorded parent nodes.
5. **Repetition**: Repeat steps 2-4 until the goal is reached or the open set is empty.

#### **Logic for Finding the Center of the Obstacle**

#### The center of the obstacle is calculated based on the vertices provided by the user. The logic is as follows:

#### **Extract Coordinates**:

#### Extract the x and y coordinates from the vertices provided by the user.

#### **Calculate Min and Max Values**:

#### Determine the minimum and maximum values for both x and y coordinates from the list of vertices.

#### **Compute Center**:

#### Calculate the center coordinates by taking the average of the minimum and maximum values for both x and y coordinates.

#### This approach ensures that the center of the rectangular obstacle is accurately determined, allowing the robot to navigate to this point.

#### **How the Code Works**

1. **User Input**:
   * The get\_input function uses Tkinter to collect obstacle vertices from the user.
2. **Grid and Obstacles**:
   * The main function initializes the grid and sets up the rectangular obstacle based on user input.
3. **Pathfinding**:
   * The a\_star\_search function computes the shortest path from the start to the end point, avoiding obstacles. If no valid path is found, the user is prompted to enter different coordinates.
4. **Visualization**:
   * The grid and robot movement are visualized using Pygame. The robot's movement is slowed down to demonstrate the pathfinding process clearly.

#### **Running the Project**

1. **Dependencies**:
   * Ensure you have Python installed along with Pygame and Tkinter libraries.
2. **Execution**:
   * Run the script to start the Tkinter GUI. Enter the vertices for the rectangular obstacle within the specified range (0-39 for both x and y coordinates).
   * The Pygame window will display the grid, obstacle, and the robot's pathfinding process.