Q-Learning Autonomous Agent

Model Explanation

This project implements a **Q-learning agent** in a 2D grid world. The agent learns to reach a goal while avoiding obstacles using a reinforcement learning approach based on trial-and-error and reward maximization.

Environment Setup

- **Grid size**: 10x10 cells
- **States**: Each cell in the grid represents a unique state `(x, y)`
- **Actions**: Up, Down, Left, Right (represented as 0, 1, 2, 3)
- **Obstacles**: Placed randomly or statically within the grid
- **Goal**: Fixed cell at the bottom-right corner

Q-Learning Logic

- **Q-table**: A dictionary with keys as states and values as arrays of 4 action-values
- **Update Rule**:
  Q(s, a) = Q(s, a) + alpha * (reward + gamma * max(Q(s')) - Q(s, a))
  Where:
  - `alpha` = 0.1 (learning rate)
  - `gamma` = 0.9 (discount factor)
  - `epsilon` starts at 1.0 and decays per episode for exploration
- **Rewards**:
  - `-1`: Normal move
  - `-5`: Collision with obstacle
  - `+10`: Reaching the goal

Training Procedure

- 500 training episodes
- Agent selects actions via ε-greedy policy
- Updates Q-table based on feedback
- After training, the agent follows the learned Q-policy

Challenges Faced

- **Exploration vs. Exploitation**: Too much random movement in early episodes; ε decay was critical.
- **Q-table Size**: Sparse exploration led to many unvisited state-action pairs initially.
- **Obstacle Handling**: Required tuning the penalty to ensure the agent learned to avoid them reliably.
- **Grid Boundaries**: Preventing the agent from going out of bounds was tricky during action calculations.
- **Training Time**: Higher episodes helped, but tuning learning rate and epsilon decay took experimentation.

Ideas for Improvement

- **Deep Q-Learning (DQN)**: Replace Q-table with a neural network for scalability.
- **Dynamic Obstacles**: Make the environment more challenging by adding moving blocks.
- **Multiple Agents**: Introduce multi-agent coordination and competition.
- **Sensor Simulation**: Use simulated LIDAR to make the agent perceive surroundings realistically.
- **Real-time Dashboard**: Add performance graphs (collisions, steps, rewards) in real-time.
- **Generalization**: Train on multiple maps and test on unseen environments.