**Q-Learning Maze Solver**

This repository contains an implementation of **Q-Learning**, a reinforcement learning algorithm, to train an agent to navigate through a maze and reach a goal.

**Problem Statement**

The goal is to train an agent to find the shortest path from a starting position (origin) to a goal in a **grid-based maze**. The maze consists of:

* **Walls (1)**: Blocks that the agent cannot pass through.
* **Roads (0)**: Open paths where the agent can move.
* **Origin (-1)**: The starting position of the agent.
* **Goal (2)**: The target positions the agent needs to reach.

The agent learns by taking actions (moving up, down, left, or right) and updating its knowledge using a **Q-table**.

**Implementation Details**

* **Reinforcement Learning**: Q-learning is used to train the agent by updating a Q-table that stores the expected rewards for different actions at each state.
* **Exploration vs. Exploitation**: The agent chooses actions based on an **ε-greedy policy**, balancing random exploration with exploiting the best-known moves.
* **Rewards System**:
  + **Reaching the goal**: +100 points
  + **Hitting a wall or boundary**: -10 points
  + **Moving normally**: -1 point (to encourage efficiency)

**How It Works**

1. **Initialize the Q-table** with zeros.
2. **Iterate through training episodes**, where the agent:
   * Selects an action based on the Q-table and exploration strategy.
   * Moves to the next state and receives a reward.
   * Updates the Q-table based on the Q-learning formula.
   * Repeats until it reaches the goal.
3. **Visualize the agent's movement** in the maze.
4. **Track learning progress** by plotting the number of steps taken to reach the goal in each iteration.

**Learning Progress**

As training progresses, the number of steps taken to reach the goal decreases, indicating that the agent is learning an optimal path.

**Example Learning Curve:**

A plot is generated showing how many steps the agent takes per iteration, demonstrating improvement over time.

**Visualization**

The program visualizes the maze with different colors:

* **Blue**: Origin (-1)
* **White**: Roads (0)
* **Black**: Walls (1)
* **Green**: Goal (2)
* **Red (or other color)**: Agent's position

**Future Improvements**

* Implement **dynamic learning rate decay** for better convergence.
* Add **different maze structures** for increased complexity.
* Improve **path visualization** to display the learned optimal route

A black square with a blue and white square in the middle

Description automatically generated

A graph with blue lines

Description automatically generated