```
import numpy as np
import matplotlib.pyplot as plt

# Function to generate a random maze with a guaranteed path
def generate_maze(size=(4, 4), obstacle_prob=0.3):
    maze = np.zeros(size, dtype=int)
    num_rows, num_cols = size

    # Randomly place obstacles
    for i in range(num_rows):
        for j in range(num_cols):
            if (i, j) not in [(0, 0), (num_rows - 1, num_cols - 1)]:
                if np.random.rand() < obstacle_prob:
                    maze[i, j] = 1

    # Ensure a path exists (naive approach)
    maze[0, 0] = 0
    maze[num_rows - 1, num_cols - 1] = 0
    return maze

# Generate the maze
maze = generate_maze(size=(6, 6), obstacle_prob=0.3)

# Define rewards
reward = np.full(maze.shape, -1)
reward[maze.shape[0] - 1, maze.shape[1] - 1] = 100  # Goal reward

# Define actions
actions = ['up', 'down', 'left', 'right']

# Define Q-table
q_table = np.zeros((*maze.shape, len(actions)))

# Hyperparameters
alpha = 0.1  # Learning rate
gamma = 0.9  # Discount factor
epsilon = 1.0  # Exploration rate
epsilon_decay = 0.995
min_epsilon = 0.01
episodes = 1000

# Helper functions
def is_valid_move(state, action):
    x, y = state
    if action == 'up' and x > 0:
        return maze[x - 1, y] == 0
    if action == 'down' and x < maze.shape[0] - 1:
        return maze[x + 1, y] == 0
    if action == 'left' and y > 0:
        return maze[x, y - 1] == 0
    if action == 'right' and y < maze.shape[1] - 1:
        return maze[x, y + 1] == 0
    return False

def move(state, action):
    x, y = state
    if action == 'up':
        return (x - 1, y)
    if action == 'down':
        return (x + 1, y)
    if action == 'left':
        return (x, y - 1)
    if action == 'right':
        return (x, y + 1)

# Q-learning algorithm
for episode in range(episodes):
    state = (0, 0)
    done = False
    total_reward = 0

    while not done:
        # Choose action
        if np.random.rand() < epsilon:
            action = np.random.choice(actions)
        else:
            action = actions[np.argmax(q_table[state])]

        if is_valid_move(state, action):
            new_state = move(state, action)
            reward_value = reward[new_state]

            # Update Q-table
            q_table[state][actions.index(action)] = q_table[state][actions.index(action)] + alpha * (
                reward_value + gamma * np.max(q_table[new_state]) - q_table[state][actions.index(action)]
            )

            state = new_state
            total_reward += reward_value

            if state == (maze.shape[0] - 1, maze.shape[1] - 1):  # Goal
                done = True

    # Decay epsilon
    epsilon = max(min_epsilon, epsilon * epsilon_decay)
```

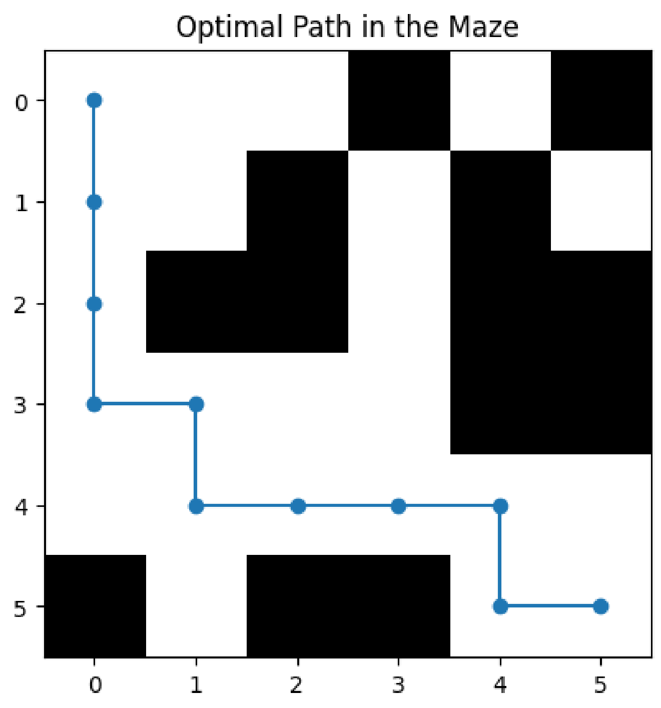
```
print("Training complete!")

# Visualize the optimal path
state = (0, 0)
path = [state]
while state != (maze.shape[0] - 1, maze.shape[1] - 1):
    action = actions[np.argmax(q_table[state])]
    state = move(state, action)
    path.append(state)

print("Optimal Path:", path)

# Plot the maze and the path
plt.imshow(maze, cmap='gray_r')
path_x, path_y = zip(*path)
plt.plot(path_y, path_x, marker='o')
plt.title("Optimal Path in the Maze")
plt.show()
```

```
Training complete!
Optimal Path: [(0, 0), (1, 0), (2, 0), (3, 0), (3, 1), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5)]
```


Optimal Path in the Maze