In [1]:
```python
import numpy as np
import random
```

In [3]:
```python
# Maze environment
# 0 = free space, -1 = obstacle, 1 = goal
maze = np.array([
    [0, 0, 0, 0],
    [0, -1, 0, 0],
    [0, 0, 0, -1],
    [0, 0, 0, 1]
])

# Display the maze
print("Maze layout (0=free, -1=obstacle, 1=goal):")
print(maze)
```

```
Maze layout (0=free, -1=obstacle, 1=goal):
[[ 0  0  0  0]
 [ 0 -1  0  0]
 [ 0  0  0 -1]
 [ 0  0  0  1]]
```

In [5]:
```python
# Q-learning parameters
alpha = 0.7       # Learning rate
gamma = 0.9       # Discount factor
epsilon = 0.3     # Exploration rate
episodes = 500    # Training episodes

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

# Initialize Q-table
q_table = np.zeros((4, 4, len(actions)))
```

In [7]:
```python
def is_valid(state):
    x, y = state
    return 0 <= x < 4 and 0 <= y < 4 and maze[x, y] != -1

def get_next_state(state, action):
    x, y = state
    if action == 'up':    x -= 1
    if action == 'down':  x += 1
    if action == 'left':  y -= 1
    if action == 'right': y += 1
    if not is_valid((x, y)):
        return state, -5  # invalid move penalty
    if maze[x, y] == 1:
        return (x, y), 10  # goal reward
    return (x, y), -1      # normal step cost
```

In [9]:
```python
for episode in range(episodes):
    state = (0, 0)
    done = False
```

```python
    while not done:
        # ε-greedy action selection
        if random.uniform(0, 1) < epsilon:
            action = random.choice(actions)
        else:
            action = actions[np.argmax(q_table[state[0], state[1]])]

        next_state, reward = get_next_state(state, action)

        # Q-value update
        old_value = q_table[state[0], state[1], actions.index(action)]
        next_max = np.max(q_table[next_state[0], next_state[1]])

        new_value = old_value + alpha * (reward + gamma * next_max - old_value)
        q_table[state[0], state[1], actions.index(action)] = new_value

        state = next_state

        # Stop if goal is reached
        if maze[state[0], state[1]] == 1:
            done = True

print("✅ Training completed!")
```

```
✅ Training completed!
```

In [11]:

```python
state = (0, 0)
path = [state]
while maze[state[0], state[1]] != 1:
    action = actions[np.argmax(q_table[state[0], state[1]])]
    state, _ = get_next_state(state, action)
    if state in path:
        break  # prevent infinite loop
    path.append(state)

print("\nLearned path to goal:")
print(path)
```

```
Learned path to goal:
[(0, 0), (1, 0), (2, 0), (3, 0), (3, 1), (3, 2), (3, 3)]
```

In [ ]:

In [ ]:

In [ ]: