In [ ]:
```
# Problem Description:

# We'll consider a 4x4 grid similar to the FrozenLake.
# The agent will start at position (0, 0) and the goal is at (3, 3).
# There will be holes at positions (1, 1), (1, 3), and (2, 2).
# The agent can move in four directions: up, down, left, right.
# If the agent moves into a hole or outside the grid, the episode ends with a reward of -1.
# Reaching the goal gives a reward of +1.
# Any other move gives a reward of 0.
```

In [8]:
```
import numpy as np

# Define the environment
grid_size = 4
holes = [(1, 1), (1, 3), (2, 2)]
start = (0, 0)
goal = (3, 3)
actions = ["up", "down", "left", "right"]
action_indices = {a: i for i, a in enumerate(actions)}

# Initialize the Q-table
q_table = np.zeros((grid_size, grid_size, len(actions)))

# Hyperparameters
alpha = 0.1
gamma = 0.9
epsilon = 0.2

# Functions for our environment

def is_valid_state(state):
    x, y = state
    return x >= 0 and x < grid_size and y >= 0 and y < grid_size

def get_next_state(current_state, action):
    x, y = current_state
    if action == "up":
        x -= 1
    elif action == "down":
        x += 1
    elif action == "left":
        y -= 1
    elif action == "right":
        y += 1
    return (x, y) if is_valid_state((x, y)) else current_state

def get_reward(state):
    if state in holes:
        return -1
    if state == goal:
        return 1
    return 0

def choose_action(state, epsilon):
    if np.random.uniform(0, 1) < epsilon:
        return np.random.choice(actions)
    return actions[np.argmax(q_table[state[0], state[1], :])]

# Q-learning

num_episodes = 5000

for _ in range(num_episodes):
    state = start
    done = False

    while not done:
        action = choose_action(state, epsilon)
        next_state = get_next_state(state, action)
        reward = get_reward(next_state)

        if next_state in holes or next_state == goal:
            done = True
```

```
        q_table[state[0], state[1], action_indices[action]] = q_table[state[0], state[1], action_indices[
            reward + gamma * np.max(q_table[next_state[0], next_state[1], :]) - q_table[state[0], state[1

        state = next_state

# Test the Q-table

state = start
path = [state]
while state != goal:
    action = choose_action(state, -1)  # Always choose the best action
    state = get_next_state(state, action)
    path.append(state)

print("Path from start to goal:", path)
```

```
Path from start to goal: [(0, 0), (1, 0), (2, 0), (2, 1), (3, 1), (3, 2), (3, 3)]
```

In [ ]: