Phase 4

```
Import numpy as np

Import random

# Environment: 2-phase traffic light (e.g., North-South and East-West)

# States: [Low, Medium, High] traffic for each direction

States = [(I, j) for I in range(3) for j in range(3)]

Actions = [0, 1]  # 0: green for NS, 1: green for EW

# Initialize Q-table

Q_table = np.zeros((len(states), len(actions)))

# Hyperparameters

Alpha = 0.1    # Learning rate

Gamma = 0.8    # Discount factor

Epsilon = 0.1  # Exploration rate

# Reward logic: lower queue length = better

Def get_reward(state, action):

    Ns, ew = state

    If action == 0:

        Return -ew  # Penalize EW wait

    Else:

        Return -ns  # Penalize NS wait
```

```
# Simulate next state (random traffic variation)
Def next_state(state):
  Ns, ew = state
  Ns = np.clip(ns + random.choice([-1, 0, 1]), 0, 2)
  Ew = np.clip(ew + random.choice([-1, 0, 1]), 0, 2)
  Return (ns, ew)

# Training loop
For episode in range(1000):
  State = random.choice(states)
  For _ in range(20):  # Run 20 time steps
    State_idx = states.index(state)

    If random.uniform(0, 1) < epsilon:
      Action = random.choice(actions)  # Explore
    Else:
      Action = np.argmax(q_table[state_idx])  # Exploit

    Reward = get_reward(state, action)
    New_state = next_state(state)
    New_state_idx = states.index(new_state)

    # Q-learning update
    Q_table[state_idx][action] += alpha * (
      Reward + gamma * np.max(q_table[new_state_idx]) – q_table[state_idx][action]
    )
```

```
      State = new_state

# Display Q-table
Print("Learned Q-Table:")
For I, state in enumerate(states):
   Print(f"State {state}: NS_green={q_table[i][0]:.2f}, EW_green={q_table[i][1]:.2f}")
```