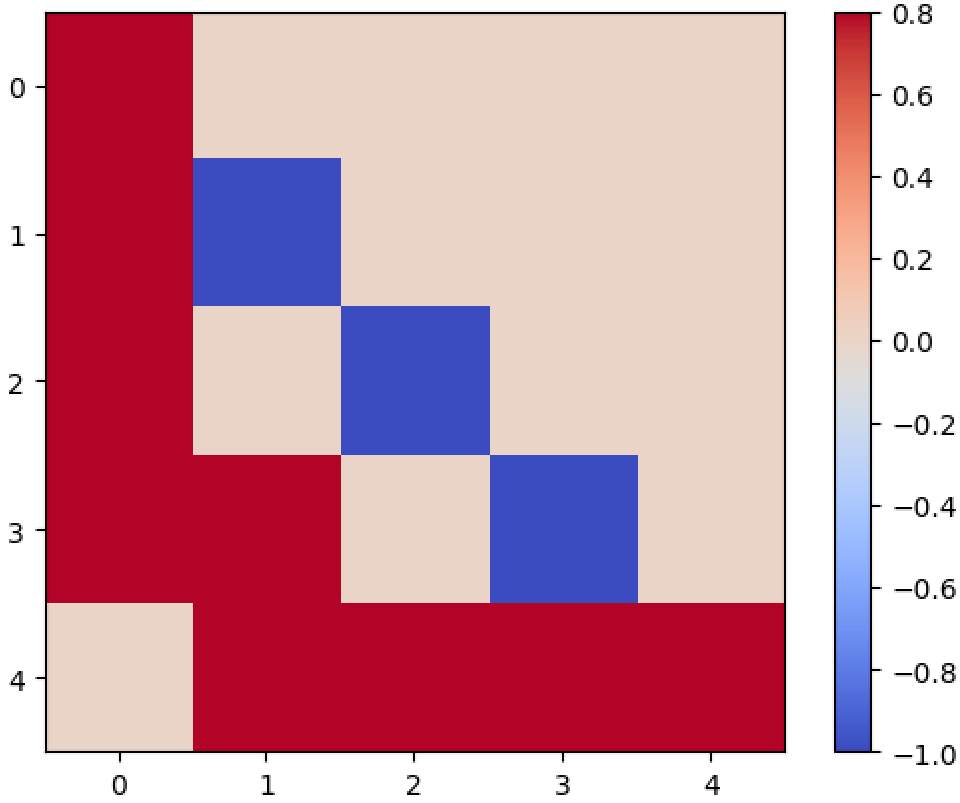
In [1]:
```
import numpy as np
import random
import matplotlib.pyplot as plt

# Define maze dimensions and parameters
maze_size = (5, 5)
start = (0, 0)  # Starting position
goal = (4, 4)   # Goal position
obstacles = [(1, 1), (2, 2), (3, 3)]  # Obstacles in the maze

# Parameters for Q-learning
learning_rate = 0.1
discount_factor = 0.9
exploration_rate = 1.0
exploration_decay = 0.995
min_exploration_rate = 0.01
episodes = 1000

# Initialize Q-table with zeros
q_table = np.zeros(maze_size + (4,))  # 4 actions: up, down, left, right

# Actions
actions = {
    0: (-1, 0),  # Up
    1: (1, 0),   # Down
    2: (0, -1),  # Left
    3: (0, 1)    # Right
}

# Function to check if position is within bounds and not an obstacle
def is_valid_position(position):
    return (0 <= position[0] < maze_size[0] and
            0 <= position[1] < maze_size[1] and
            position not in obstacles)

# Function to take an action
def take_action(state, action):
    next_state = (state[0] + actions[action][0], state[1] + actions[actio
    return next_state if is_valid_position(next_state) else state

# Function to calculate reward
def get_reward(state):
    if state == goal:
        return 100  # Reward for reaching the goal
    elif state in obstacles:
        return -10  # Penalty for hitting an obstacle
    else:
        return -1  # Penalty for each move to encourage efficiency

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

    while not done:
        # Choose action using epsilon-greedy policy
        if random.uniform(0, 1) < exploration_rate:
            action = random.choice(range(4))  # Explore
```

```
        else:
            action = np.argmax(q_table[state])  # Exploit

        # Take action and observe the next state and reward
        next_state = take_action(state, action)
        reward = get_reward(next_state)

        # Update Q-value
        best_next_action = np.argmax(q_table[next_state])
        q_table[state][action] = (q_table[state][action] +
                                  learning_rate * (reward + discount_fact
                                                   q_table[state][action]

        # Move to next state
        state = next_state

        # Check if reached the goal
        if state == goal:
            done = True

    # Decay exploration rate
    exploration_rate = max(min_exploration_rate, exploration_rate * explo

# Test the agent
state = start
path = [state]
done = False

while not done:
    action = np.argmax(q_table[state])
    state = take_action(state, action)
    path.append(state)
    if state == goal:
        done = True

print("Path taken by the agent to reach the goal:")
print(path)

# Visualization of the maze and the path
maze = np.zeros(maze_size)
for obs in obstacles:
    maze[obs] = -1
maze[start] = 0.5  # Start position
maze[goal] = 1.5   # Goal position

for step in path:
    maze[step] = 0.8  # Mark the path

plt.imshow(maze, cmap='coolwarm', interpolation='nearest')
plt.colorbar()
plt.show()
```

```
Path taken by the agent to reach the goal:
[(0, 0), (1, 0), (2, 0), (3, 0), (3, 1), (4, 1), (4, 2), (4, 3), (4, 4)]
```



In [ ]: