**NAME:** RUPESH DHIRWANI

**CLASS:** D16AD

**ROLL NO:** 10

```
import numpy as np

class GridWorld:
    def __init__(self, shape):
        self.shape = shape
        self.num_states = np.prod(shape)
        self.num_actions = 4  # 4 possible actions: up, down, left, right
        self.transitions = self._initialize_transitions()
        self.rewards = np.zeros(self.num_states)
        self.rewards[-1] = 1  # Reward of 1 at the terminal state

    def _initialize_transitions(self):
        transitions = np.zeros((self.num_states, self.num_actions, self.num_states))
        for s in range(self.num_states):
            i, j = np.unravel_index(s, self.shape)
            if i > 0:
                transitions[s, 0, np.ravel_multi_index((i-1, j), self.shape)] = 1  # Up
            if i < self.shape[0] - 1:
                transitions[s, 1, np.ravel_multi_index((i+1, j), self.shape)] = 1  # Down
            if j > 0:
                transitions[s, 2, np.ravel_multi_index((i, j-1), self.shape)] = 1  # Left
            if j < self.shape[1] - 1:
                transitions[s, 3, np.ravel_multi_index((i, j+1), self.shape)] = 1  # Right
        return transitions

def value_iteration(gridworld, gamma=0.9, theta=1e-6):
    num_states = gridworld.num_states
    num_actions = gridworld.num_actions
    transitions = gridworld.transitions
    rewards = gridworld.rewards

    V = np.zeros(num_states)

    while True:
        delta = 0
        for s in range(num_states):
            v = V[s]
            V[s] = max(np.sum(transitions[s, :, :] * (rewards + gamma * V), axis=1))
            delta = max(delta, abs(v - V[s]))
        if delta < theta:
            break

    policy = np.argmax(np.sum(transitions * (rewards + gamma * V), axis=2), axis=1)

    return policy, V

# Create a 3x4 grid world
gridworld = GridWorld(shape=(3, 4))

# Run value iteration
policy, V = value_iteration(gridworld)

print("Optimal Policy:")
print(policy.reshape(gridworld.shape))
print("\nOptimal Value Function:")
print(V.reshape(gridworld.shape))
```

```
Optimal Policy:
[[1 1 1 1]
 [1 1 1 1]
 [3 3 3 0]]

Optimal Value Function:
[[3.45315401 3.83683861 4.26315475 4.73683927]
 [3.83683861 4.26315475 4.73683927 5.26315534]
 [4.26315475 4.73683927 5.26315534 4.73683981]]
```