NAME: RUPESH DHIRWANI

ROLL NO: 10

CLASS: D16AD

```
import numpy as np

class GridWorldMDP:
    def __init__(self, num_rows, num_cols, terminal_states, rewards):
        self.num_rows = num_rows
        self.num_cols = num_cols
        self.terminal_states = terminal_states
        self.rewards = rewards
        self.num_states = num_rows * num_cols
        self.actions = [(0, 1), (0, -1), (1, 0), (-1, 0)]  # right, left, down, up
        self.transition_prob = self._initialize_transition_prob()

    def _initialize_transition_prob(self):
        transition_prob = np.zeros((self.num_states, len(self.actions), self.num_states))
        for state in range(self.num_states):
            for action_index, action in enumerate(self.actions):
                next_state = self._get_next_state(state, action)
                transition_prob[state, action_index, next_state] = 1
        return transition_prob

    def _get_next_state(self, state, action):
        row, col = divmod(state, self.num_cols)
        next_row = max(0, min(row + action[0], self.num_rows - 1))
        next_col = max(0, min(col + action[1], self.num_cols - 1))
        next_state = next_row * self.num_cols + next_col
        if (next_row, next_col) in self.terminal_states:
            return next_state, self.rewards[(next_row, next_col)]
        return next_state, 0

    def policy_evaluation(self, policy, discount_factor=0.9, theta=0.0001):
        V = np.zeros(self.num_states)
        while True:
            delta = 0
            for state in range(self.num_states):
                v = V[state]
                new_v = 0
                for action_index, action in enumerate(self.actions):
                    next_state, reward = self._get_next_state(state, action)
                    new_v += policy[state, action_index] * (reward + discount_factor * V[next_state])
                V[state] = new_v
                delta = max(delta, abs(v - V[state]))
            if delta < theta:
                break
        return V

    def policy_iteration(self, discount_factor=0.9):
        policy = np.ones((self.num_states, len(self.actions))) / len(self.actions)
        while True:
            V = self.policy_evaluation(policy, discount_factor)
            policy_stable = True
            for state in range(self.num_states):
                old_action = np.argmax(policy[state])
                action_values = np.zeros(len(self.actions))
                for action_index, action in enumerate(self.actions):
                    next_state, reward = self._get_next_state(state, action)
                    action_values[action_index] = reward + discount_factor * V[next_state]
                best_action = np.argmax(action_values)
                if old_action != best_action:
                    policy_stable = False
                policy[state] = np.eye(len(self.actions))[best_action]
            if policy_stable:
                break
        return policy, V

def main():
    num_rows = 5
    num_cols = 3
    terminal_states = {(0, 0): 1, (4, 2): 1}
    rewards = {(0, 0): 1, (4, 2): 1}
    grid_world = GridWorldMDP(num_rows, num_cols, terminal_states, rewards)

    # Policy iteration
    optimal_policy, optimal_values = grid_world.policy_iteration()

    # Displaying the optimal policy
```

```python
    actions_str = ['R', 'L', 'D', 'U']
    for state in range(grid_world.num_states):
        row = state // num_cols
        col = state % num_cols
        print(f"State ({row}, {col}): Action {actions_str[np.argmax(optimal_policy[state])]} Optimal Value: {optimal_values[state]}")

if __name__ == "__main__":
    main()
```

```
State (0, 0): Action L Optimal Value: 9.999153585021714
State (0, 1): Action L Optimal Value: 9.999238226519543
State (0, 2): Action L Optimal Value: 8.999314403867588
State (1, 0): Action U Optimal Value: 9.999238226519543
State (1, 1): Action L Optimal Value: 8.999314403867588
State (1, 2): Action L Optimal Value: 8.09938296348083
State (2, 0): Action U Optimal Value: 8.999314403867588
State (2, 1): Action L Optimal Value: 8.09938296348083
State (2, 2): Action D Optimal Value: 8.999153585021714
State (3, 0): Action U Optimal Value: 8.09938296348083
State (3, 1): Action R Optimal Value: 8.999153585021714
State (3, 2): Action D Optimal Value: 9.999153585021714
State (4, 0): Action R Optimal Value: 8.999153585021714
State (4, 1): Action R Optimal Value: 9.999153585021714
State (4, 2): Action R Optimal Value: 9.999153585021714
```