Lab Exercise 5: “Treasure Hunt MDP — Calculating Returns”

Scenario:
You are navigating a small grid-world island in search of treasure. Each cell is a state.
At each step, you can move **North**, **South**, **East**, or **West** (if not blocked by the island's edge).
Some cells contain rewards (treasure), others contain penalties (quicksand).
Your task: **simulate episodes, collect rewards, and compute the total discounted return for each episode**.

Environment Setup

- Grid: 3×4
- Start: top-left cell (0, 0)
- Terminal states:
  - Treasure at (0, 3) → reward **+10**
  - Quicksand at (1, 3) → reward **-10**
- Other cells → step penalty **-1** (to encourage faster treasure finding)

Tasks

1. **Define the MDP**
   - States as (row, col) pairs
   - Actions: ['N', 'S', 'E', 'W']
   - Rewards and terminal states as above
   - Transition probabilities
2. **Simulate Episodes**
   - Random policy: choose each action with equal probability
   - End episode if terminal state is reached or step limit (max 20) is hit
3. **Implement Return Function**
   - For each episode, compute G(t)
4. **Experiment**

- Run 10 episodes with $\gamma = 0.9$ and print G_0 for each episode
- Change γ to 0.5 and compare results — observe the impact of discounting

---

Sample Output

Episode 1: States visited: [(0,0), (0,1), (1,1), (1,2), (1,3)]

Rewards: [-1, -1, -1, -10]

G_0: -12.71

Episode 2: ...

...

Please find below a sample python skeleton for your understanding:

```
import random

ROWS = 3
COLS = 4
ACTIONS = ['N', 'S', 'E', 'W']
REWARDS = { (0, 3): 10,
            (1, 3): -10  }
STEP_REWARD = -1

# Terminal states
TERMINAL_STATES = [(0, 3), (1, 3)]

ACTION_EFFECTS = {
  'N': [(-1, 0), (0, -1), (0, 1)],
  'S': [(1, 0), (0, -1), (0, 1)],
  'E': [(0, 1), (-1, 0), (1, 0)],
  'W': [(0, -1), (-1, 0), (1, 0)]
```

```
}
PROBS = [0.8, 0.1, 0.1]
def in_bounds(state):
    """Check if the state is inside the grid."""
    r, c = state
    return 0 <= r < ROWS and 0 <= c < COLS


def move(state, action):
    """Move from current state according to stochastic transitions."""
    effects = ACTION_EFFECTS[action]
    chosen_effect = random.choices(effects, PROBS)[0]
    new_r, new_c = state[0] + chosen_effect[0], state[1] + chosen_effect[1]
    if in_bounds((new_r, new_c)):
        return (new_r, new_c)
    return state  # If out of bounds, stay in place


def get_reward(state):
    """Return the reward for a state."""
    return REWARDS.get(state, STEP_REWARD)
```

complete the code and submit the python notebook as a PDF in the lms page before the deadline.