EXPERIMENT 10

Solving a Markov Decision Process(MDP)

Ankit Kandulna

23/CS/060

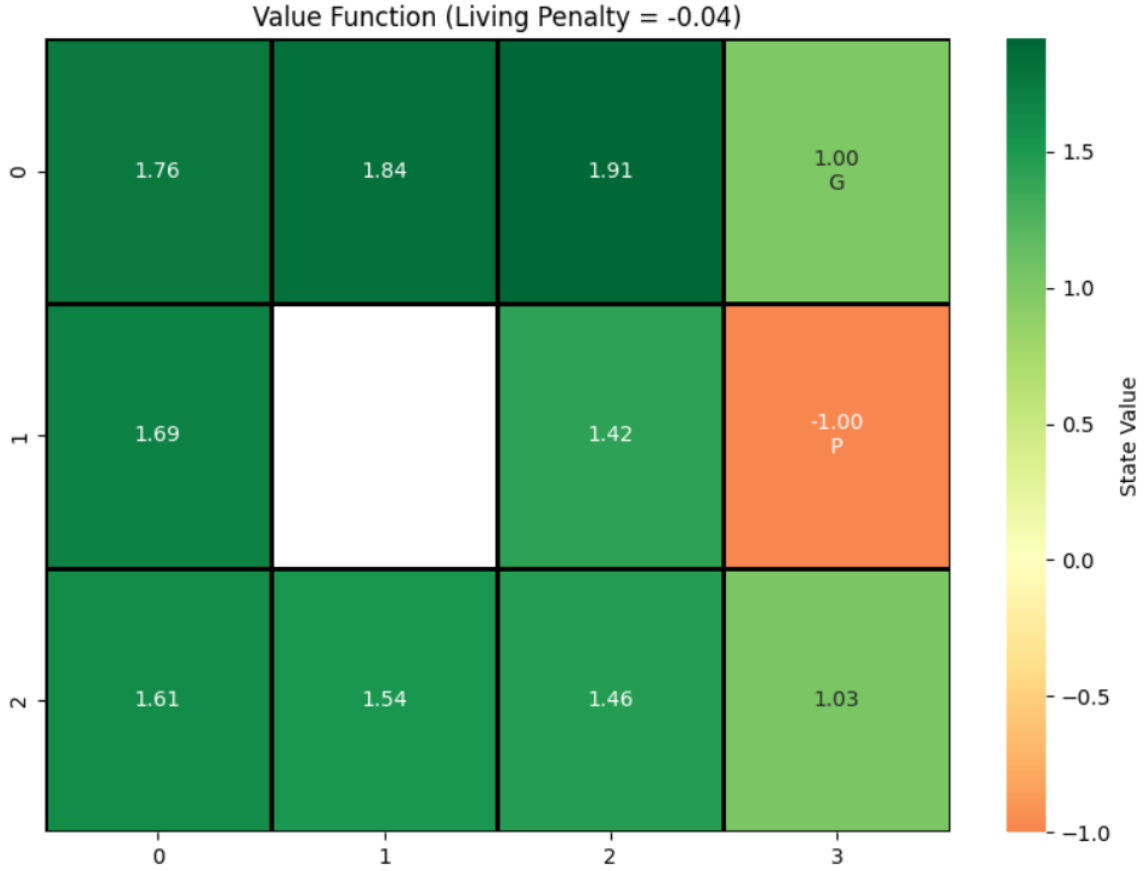
1. Source Code:

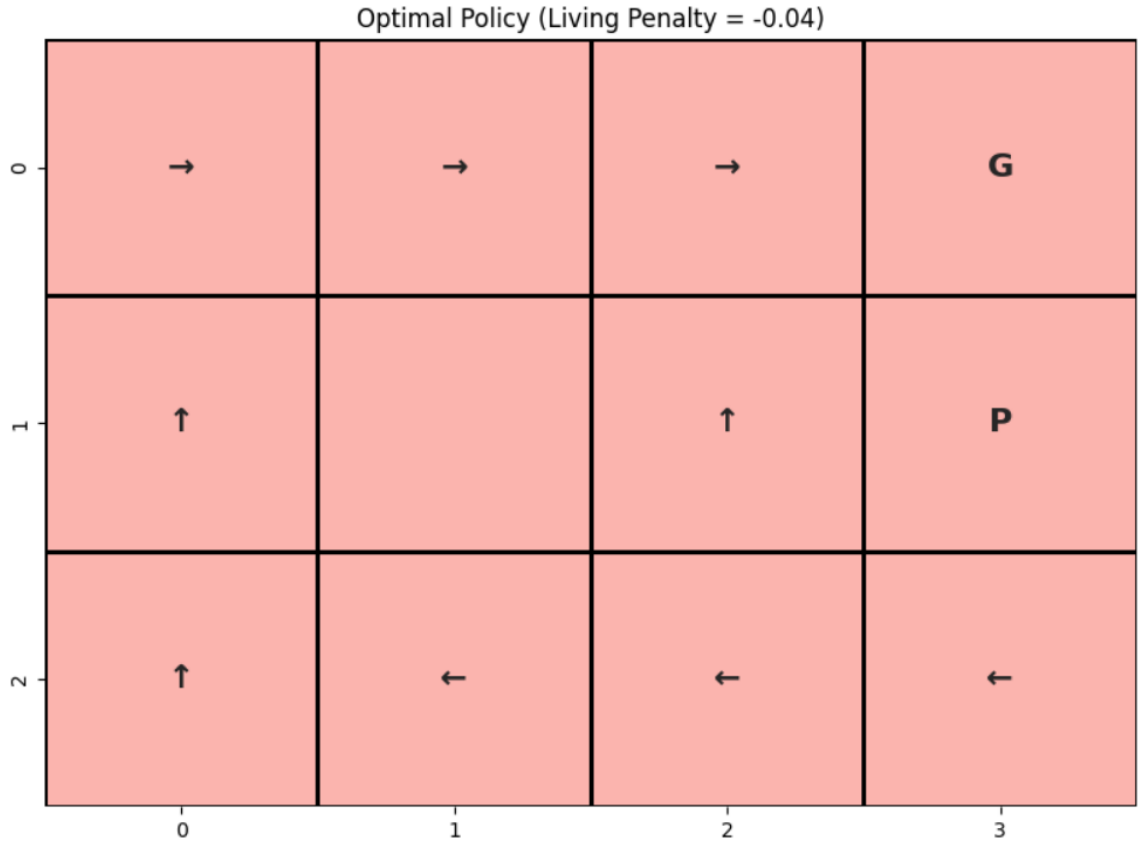
```python
def value_iteration(env: GridWorld, theta: float = 1e-4, max_iterations: int = 1000) -> Dict[Tuple[int, int], float]
    """
    Value Iteration algorithm

    Parameters:
    - env: GridWorld environment
    - theta: convergence threshold
    - max_iterations: maximum number of iterations

    Returns:
    - V: converged value function
    """
    # Initialize value function
    V = {state: 0.0 for state in env.states}

    print("Running Value Iteration...")
    for iteration in range(max_iterations):
        delta = 0
        V_new = V.copy()

        for state in env.states:
            if env.is_terminal(state):
                V_new[state] = env.get_reward(state)
                continue

            # Calculate value for each action
            action_values = []
            for action in env.get_possible_actions(state):
                action_value = 0
                transitions = env.get_transition_probabilities(state, action)
```

```
                for prob, next_state in transitions:
                    reward = env.get_reward(next_state)
                    action_value += prob * (reward + env.gamma * V[next_state])

                action_values.append(action_value)

            # Update value with maximum action value
            if action_values:
                V_new[state] = max(action_values)

            # Track maximum change
            delta = max(delta, abs(V_new[state] - V[state]))

        V = V_new

        # Check convergence
        if delta < theta:
            print(f"Converged after {iteration + 1} iterations (delta = {delta:.6f})")
            break

        if iteration % 100 == 0:
            print(f"Iteration {iteration}: delta = {delta:.6f}")

    return V

# Run value iteration
V = value_iteration(env)
print()
```

```
def extract_policy(env: GridWorld, V: Dict[Tuple[int, int], float]) -> Dict[Tuple[int, int], str]:
    """
    Extract optimal policy from value function

    Parameters:
    - env: GridWorld environment
    - V: value function

    Returns:
    - policy: optimal policy mapping states to actions
    """
    policy = {}

    for state in env.states:
        if env.is_terminal(state):
            policy[state] = 'T'  # Terminal
            continue

        best_action = None
        best_value = -float('inf')

        for action in env.get_possible_actions(state):
            action_value = 0
            transitions = env.get_transition_probabilities(state, action)

            for prob, next_state in transitions:
                reward = env.get_reward(next_state)
                action_value += prob * (reward + env.gamma * V[next_state])

            if action_value > best_value:
                best_value = action_value
                best_action = action

        policy[state] = best_action

    return policy
```

2. Final Results:





3. Analysis:

1. Default Living Penalty (-0.04):

The agent receives a small negative reward per step.

The policy successfully avoids the pit at (1,3) and finds the goal at (0,3).

2. Living Penalty = 0.0:

Without a penalty per step, the agent is less incentivized to take the shortest path.

The policy still reaches the goal while avoiding the pit, but may not prioritize the fastest path.

3. High Living Penalty (-0.5):

With a high step penalty, the agent strongly prefers the shortest path to the goal.

The policy avoids all unnecessary moves and reaches the goal in minimum steps.