Experiment 10: Solving a Markov Decision Process (MDP) in GridWorld

Anushka, 23/CS/071

November 2025

1 Introduction

In this experiment, we solve a stochastic GridWorld Markov Decision Process (MDP) using the Value Iteration algorithm. The GridWorld is a 3x4 grid where the agent must navigate to reach a goal state while avoiding a pit. Actions are stochastic: the agent may slip to the left or right of the intended direction. We introduce a living penalty to encourage shorter paths and analyze how this penalty affects the agent's optimal policy. The experiment aims to compute the optimal value function and policy, visualize them, and observe the effect of different living penalties on agent behavior.

2 Task 1: Defining the GridWorld MDP

2.1 States, Actions, and Rewards

- **States (S):** All cells except the wall at (1,1). Terminal states: Goal (0,3), Pit (1,3)
- **Actions (A):** ['up', 'down', 'left', 'right']
- **Rewards (R):**
  - Goal: +1
  - Pit: -1
  - All other states: living penalty (default -0.04)
- **Discount Factor:** $\gamma = 0.99$

2.2 Python Implementation — GridWorld Setup

Listing 1: GridWorld setup

```python
import numpy as np

ROWS, COLS = 3, 4
WALL = (1, 1)
GOAL = (0, 3)
PIT  = (1, 3)
TERMINALS = {GOAL, PIT}
```

```python

ACTIONS = ['up', 'down', 'left', 'right']
ACT_TO_DELTA = {
    'up': (-1, 0),
    'down': (1, 0),
    'left': (0, -1),
    'right': (0, 1),
}

SLIP_LEFT = {'up':'left','down':'right','left':'down','right':'up'}
SLIP_RIGHT= {'up':'right','down':'left','left':'up','right':'down'}

def in_bounds(s):
    r, c = s
    return 0 <= r < ROWS and 0 <= c < COLS

def step(s, a):
    if s in TERMINALS:
        return s
    dr, dc = ACT_TO_DELTA[a]
    ns = (s[0] + dr, s[1] + dc)
    if not in_bounds(ns) or ns == WALL:
        return s
    return ns

def get_all_states():
    return [(r,c) for r in range(ROWS) for c in range(COLS) if (r,c) !=
        WALL]

def make_rewards(living_penalty=-0.04):
    R = {}
    for s in get_all_states():
        if s == GOAL:
            R[s] = 1.0
        elif s == PIT:
            R[s] = -1.0
        else:
            R[s] = living_penalty
    return R

def get_next_states(s, a):
    if s in TERMINALS:
        return [(1.0, s)]
    intended = step(s, a)
    left = step(s, SLIP_LEFT[a])
    right = step(s, SLIP_RIGHT[a])
    return [(0.8, intended), (0.1, left), (0.1, right)]

get_all_states()
```

3 Task 2: Value Iteration Algorithm

Listing 2: Value Iteration Algorithm

```python
def value_iteration(gamma=0.99, theta=1e-4, living_penalty=-0.04):
    R = make_rewards(living_penalty)
    states = get_all_states()
    V = {s: 0.0 for s in states}
    V[GOAL] = 0.0
    V[PIT]  = 0.0
    def q_value(s, a):
        return sum(p * (R[ns] + gamma * V[ns]) for p, ns in
            get_next_states(s, a))
    while True:
        delta = 0.0
        for s in states:
            if s in TERMINALS:
                continue
            v = V[s]
            V[s] = max(q_value(s, a) for a in ACTIONS)
            delta = max(delta, abs(v - V[s]))
        if delta < theta:
            break
    return V
```

4 Task 3: Policy Extraction

Listing 3: Extract Optimal Policy

```python
ARROWS = {'up':'^', 'down':'v', 'left':'<', 'right':'>'}

def extract_policy(V, gamma=0.99, living_penalty=-0.04):
    R = make_rewards(living_penalty)
    policy = {}
    def q_value(s, a):
        return sum(p * (R[ns] + gamma * V[ns]) for p, ns in
            get_next_states(s, a))
    for s in get_all_states():
        if s in TERMINALS:
            policy[s] = None
        else:
            qs = [(a, q_value(s, a)) for a in ACTIONS]
            policy[s] = max(qs, key=lambda x: x[1])[0]
    return policy
```

5 Task 4: Visualization

Listing 4: Visualize Value Function and Policy

```python

import matplotlib.pyplot as plt
import numpy as np

def values_to_grid(V):
    grid = np.full((ROWS, COLS), np.nan)
    for r in range(ROWS):
        for c in range(COLS):
            if (r,c) == WALL:
                continue
            grid[r,c] = V.get((r,c), np.nan)
    return grid

def plot_value_heatmap(V, title):
    grid = values_to_grid(V)
    plt.figure(figsize=(5,3))
    plt.imshow(grid, interpolation='nearest')
    plt.title(title)
    plt.colorbar()
    for (i, j), val in np.ndenumerate(grid):
        if not np.isnan(val):
            plt.text(j, i, f"{val:.2f}", ha='center', va='center')
        else:
            plt.text(j, i, '   ', ha='center', va='center')
    plt.xticks(range(COLS))
    plt.yticks(range(ROWS))
    plt.show()

def policy_to_grid(policy):
    grid = np.full((ROWS, COLS), '', dtype=object)
    for r in range(ROWS):
        for c in range(COLS):
            s = (r,c)
            if s == WALL: grid[r,c] = '   '
            elif s == GOAL: grid[r,c] = 'G'
            elif s == PIT:  grid[r,c] = 'P'
            else:
                a = policy.get(s, None)
                grid[r,c] = ARROWS[a] if a else ''
    return grid

def plot_policy_grid(policy, title):
    grid = policy_to_grid(policy)
    plt.figure(figsize=(5,3))
    plt.imshow(np.zeros((ROWS, COLS)), vmin=0, vmax=1)
    for (i, j), val in np.ndenumerate(grid):
        plt.text(j, i, val, ha='center', va='center', fontsize=14)
    plt.title(title)
    plt.xticks(range(COLS))
    plt.yticks(range(ROWS))
    plt.show()

# Default run (R=-0.04)
V_default = value_iteration(gamma=0.99, theta=1e-4, living_penalty=-0.04)
pi_default = extract_policy(V_default, gamma=0.99, living_penalty=-0.04)
plot_value_heatmap(V_default, "Value Heatmap    Default (R=-0.04)")
plot_policy_grid(pi_default, "Policy    Default (R=-0.04)")

# Q1/Q2/Q3: change penalties
V_zero = value_iteration(gamma=0.99, theta=1e-4, living_penalty=0.0)
pi_zero = extract_policy(V_zero, gamma=0.99, living_penalty=0.0)
plot_value_heatmap(V_zero, "Value Heatmap    No Penalty (R=0.0)")
plot_policy_grid(pi_zero, "Policy    No Penalty (R=0.0)")

V_high = value_iteration(gamma=0.99, theta=1e-4, living_penalty=-0.5)
pi_high = extract_policy(V_high, gamma=0.99, living_penalty=-0.5)
plot_value_heatmap(V_high, "Value Heatmap    High Penalty (R=-0.5)")
```

```
plot_policy_grid(pi_high, "Policy      High Penalty (R=-0.5)")
```

6 Plots

6.1 Default Living Penalty (R=-0.04)

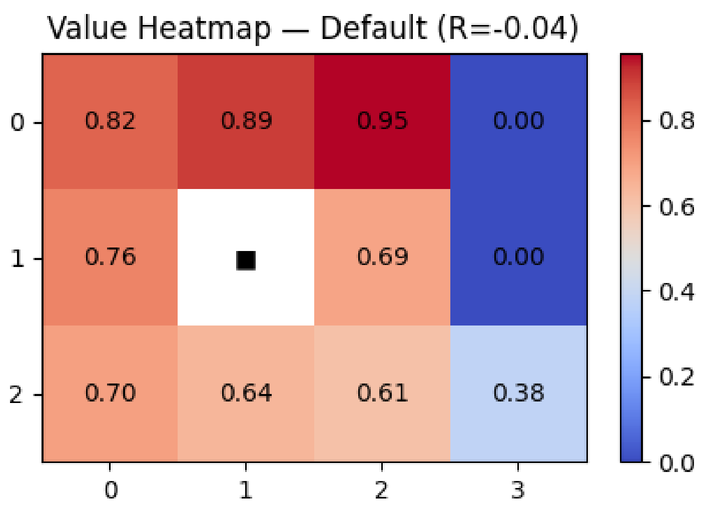

Figure 1: Value Heatmap — Default Living Penalty (R=-0.04)

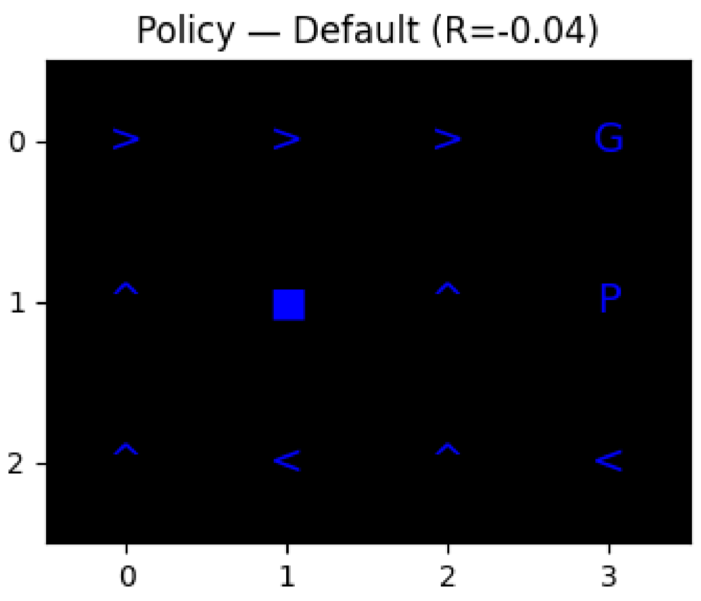

Figure 2: Optimal Policy — Default Living Penalty (R=-0.04)

6.2 No Living Penalty (R=0.0)

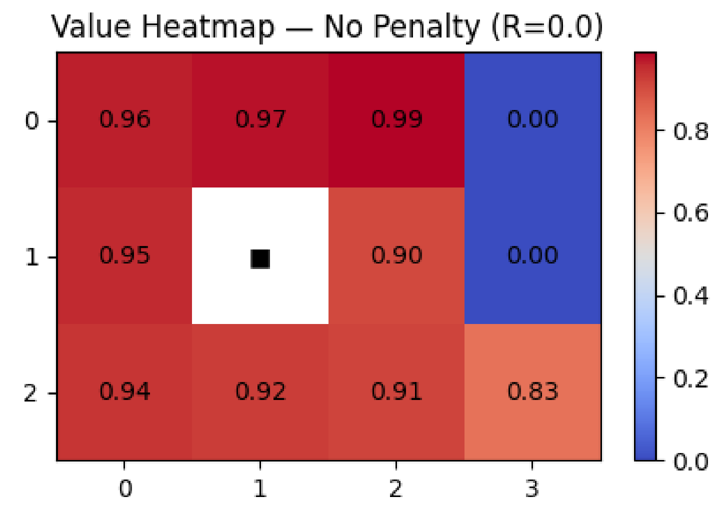

Figure 3: Value Heatmap — No Living Penalty (R=0.0)

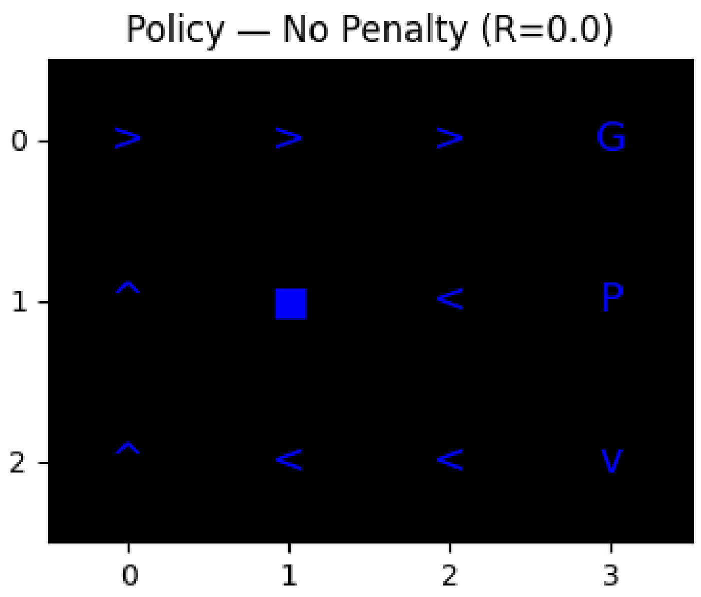

Figure 4: Optimal Policy — No Living Penalty (R=0.0)

6.3 High Living Penalty (R=-0.5)

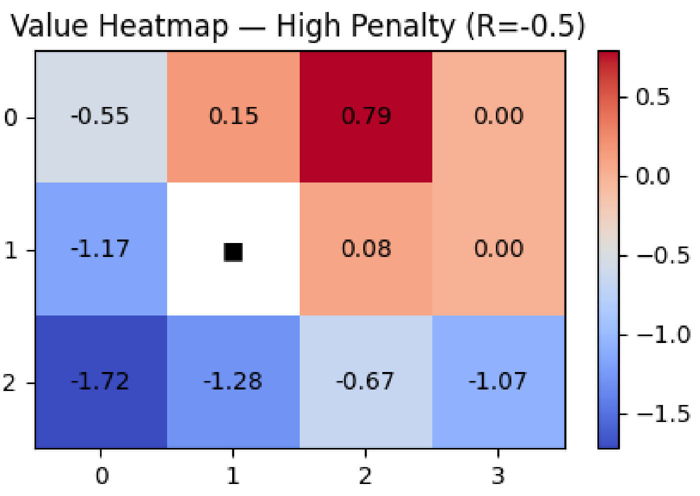

Figure 5: Value Heatmap — High Living Penalty (R=-0.5)

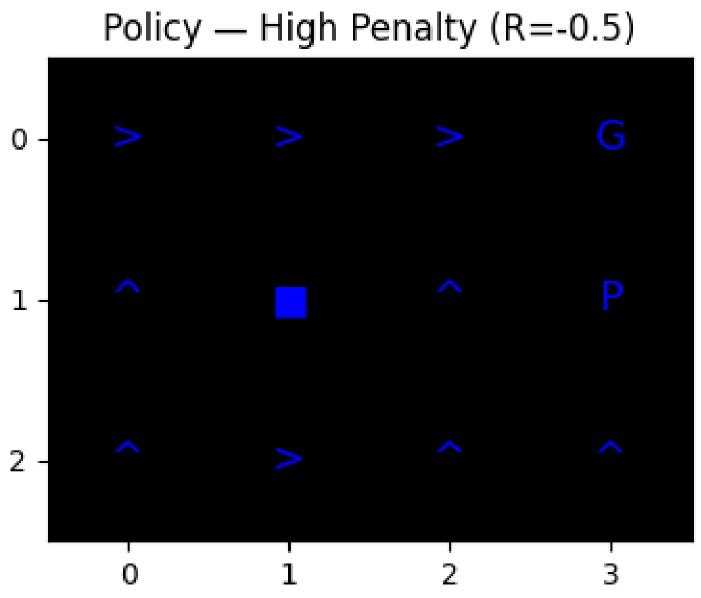

Figure 6: Optimal Policy — High Living Penalty (R=-0.5)

7 Observations and Analysis

- **Default living penalty (-0.04):** Agent avoids the pit and finds the shortest path to the goal.
- **No living penalty (0.0):** Agent still reaches the goal but is less incentivized to take the shortest path; may wander more.
- **High living penalty (-0.5):** Agent aggressively chooses the shortest path to minimize cumulative penalty, even risking nearby pit.

- **Effect of stochasticity:** Slipping probabilities cause the agent to choose safer paths to avoid falling into the pit.
- **Value function interpretation:** Higher values correspond to states closer to the goal with lower risk of entering the pit.