Reinforcement Learning Assignment 02

Waruni Alagalle Hitgedara (202487136)
Prasanth Senthan (202398190)
Memorial University of Newfoundland, Canada

Spring 2025

Contents

1 Results and analysis **4**
1.1 Part 01: Random policy with equal probability 4
1.1.1 Solving the system of Bellman equations 4
1.1.2 Iterative policy evaluation 5
1.1.3 Discussion 6
1.2 Part 01: Optimal policy 7
1.2.1 Explicitly solving the Bellman optimality equation 7
1.2.2 Policy iteration with iterative policy evaluation 9
1.2.3 Policy improvement with value iteration 11
1.2.4 Final analysis 12

2 Part 02: **12**
2.1 Monte Carlo with exploring starts (ES) 12
2.1.1 Monte Carlo with Exploring Starts Results 13
2.2 Monte Carlo with ϵ-soft policy (no ES) 14
2.2.1 Monte Carlo with ϵ-soft policy (no ES) results 15
2.3 Off-policy Monte Carlo control with importance sampling and equiprobable behavior policy 19
2.3.1 Off-policy Monte Carlo control with importance sampling and equiprobable behavior policy results 19

3 Code Repository **21**

Module information

This document fulfills the requirements for the Reinforcement Learning Module (DSCI-6650-001).

Introduction

Environment setup

A 5×5 Gridworld environment is used, where each cell represents a unique state. The agent can move up, down, left, or right at each step. If an action would take the agent off the grid, the agent remains in the same state, and a penalty of -0.5 is incurred.

Special squares are defined as follows:

- **Blue square:** Any action yields a reward of 5 and sends the agent to the red square.
- **Green square:** Any action yields a reward of 2.5 and sends the agent to either the yellow or red square with equal probability.
- **White / Yellow / Red squares:** Moving from one of these squares gives a reward of 0. An off-grid move from these gives a reward of -0.5.

In Part 2, the environment is modified by introducing terminal black squares. Reaching a black square ends the episode. Additionally, all actions from white, yellow, or red squares yield a reward of -0.2.

Algorithms implemented

The following reinforcement learning algorithms were applied to evaluate and improve policies within the Gridworld environment:

- **Uniform Random Policy Evaluation:** A baseline policy was considered where the agent selects each of the four possible actions (up, down, left, right) with equal probability of 0.25 in every state. The value function for this policy was estimated using two methods:
    1. By explicitly solving the system of Bellman equations as a set of linear equations.
    2. By performing iterative policy evaluation, updating the state values repeatedly until convergence.
- **Optimal Policy Determination:** The optimal policy for the Gridworld environment was derived using the following methods:
    1. By explicitly solving the Bellman optimality equation as a nonlinear system.
    2. By applying policy iteration, which alternates between iterative policy evaluation and policy improvement until convergence.

3. By using value iteration, where the value function is updated based on the Bellman optimality equation and the optimal policy is extracted from the final value function.

Simulation setup

All simulations were conducted using a fixed discount factor of $\gamma = 0.95$. For Part 1, the value functions were estimated using both analytical and iterative techniques. In Part 2, the modified Gridworld introduced terminal black squares and negative rewards for movement.

Monte Carlo simulations were run with the following setups:

- **Exploring Starts:** Episodes were initiated from random state-action pairs to ensure adequate exploration.
- **ϵ-soft Policy:** Policies were initialized with equal action probabilities, and ϵ was set to balance exploration and exploitation.
- **Importance Sampling:** The behavior policy selected all actions uniformly, while the target policy was improved iteratively using weighted returns.

Performance was evaluated by examining convergence of the value function and the learned optimal policies. Visualizations of value grids and directional policies were used to interpret results.

Reproducibility measures

To ensure the reproducibility of results across multiple runs, specific steps were taken to control randomness and maintain a consistent experimental environment:

- **Random seed initialization:** A fixed random seed was set for each simulation iteration using `np.random.seed()`.
- **Library consistency:** The implementation was run in a controlled Python environment with consistent versions of NumPy and Matplotlib to avoid discrepancies due to version differences.

1 Results and analysis

1.1 Part 01: Random policy with equal probability

1.1.1 Solving the system of Bellman equations

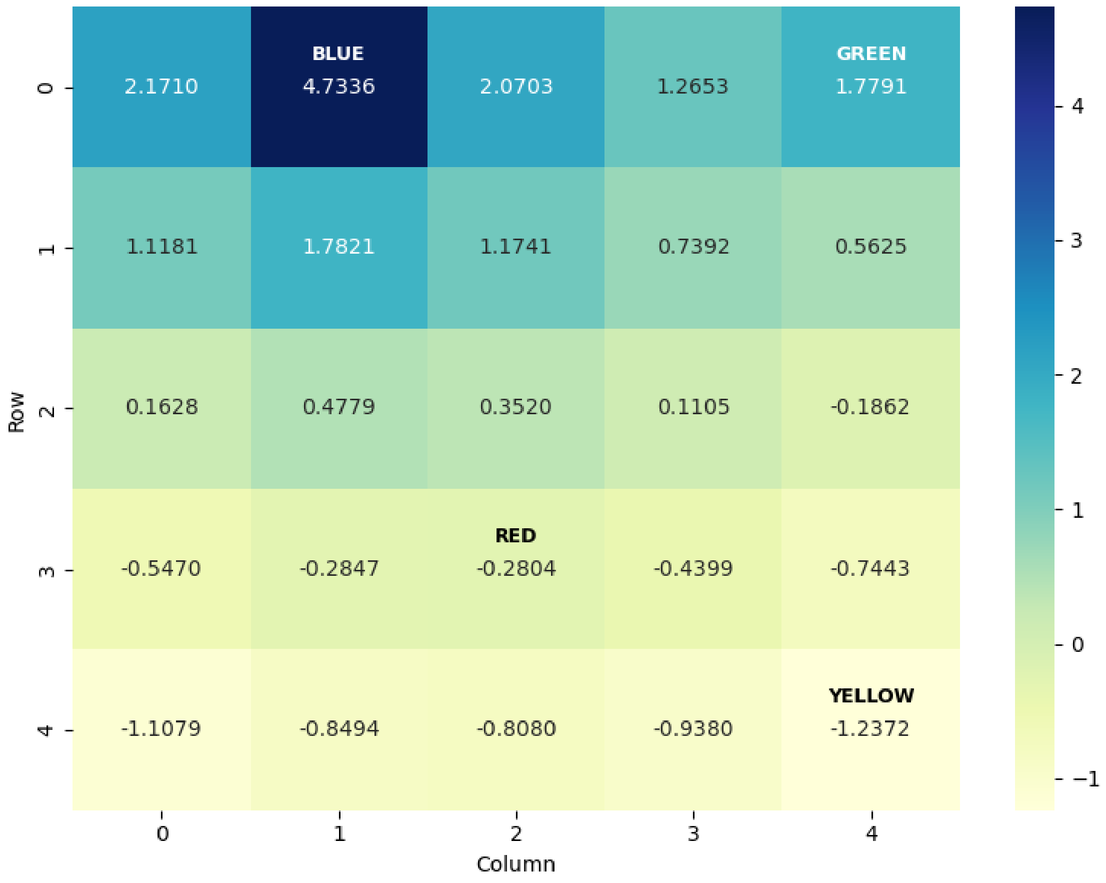

Figure 1: Value function (Bellman equations) - random policy, $\gamma = 0.95$

The heatmap in figure 1 reveals the expected cumulative discounted rewards for each state in the 5×5 gridworld. The highest value of approximately **4.7336** is observed at **state (0,1)**, corresponding to the blue terminal state. The agent gets an immediate reward of +5 when it reaches this blue square. Being in this position also helps the agent collect more rewards in the future.

The green terminal state at (0,4) shows a value of 1.7791, reflecting its positive reward. The red terminal state at (3,2) has a negative value of -0.2804, indicating the penalty for reaching this state. The yellow terminal state at (4,4) shows the lowest value of -1.2372.

1.1.2 Iterative policy evaluation

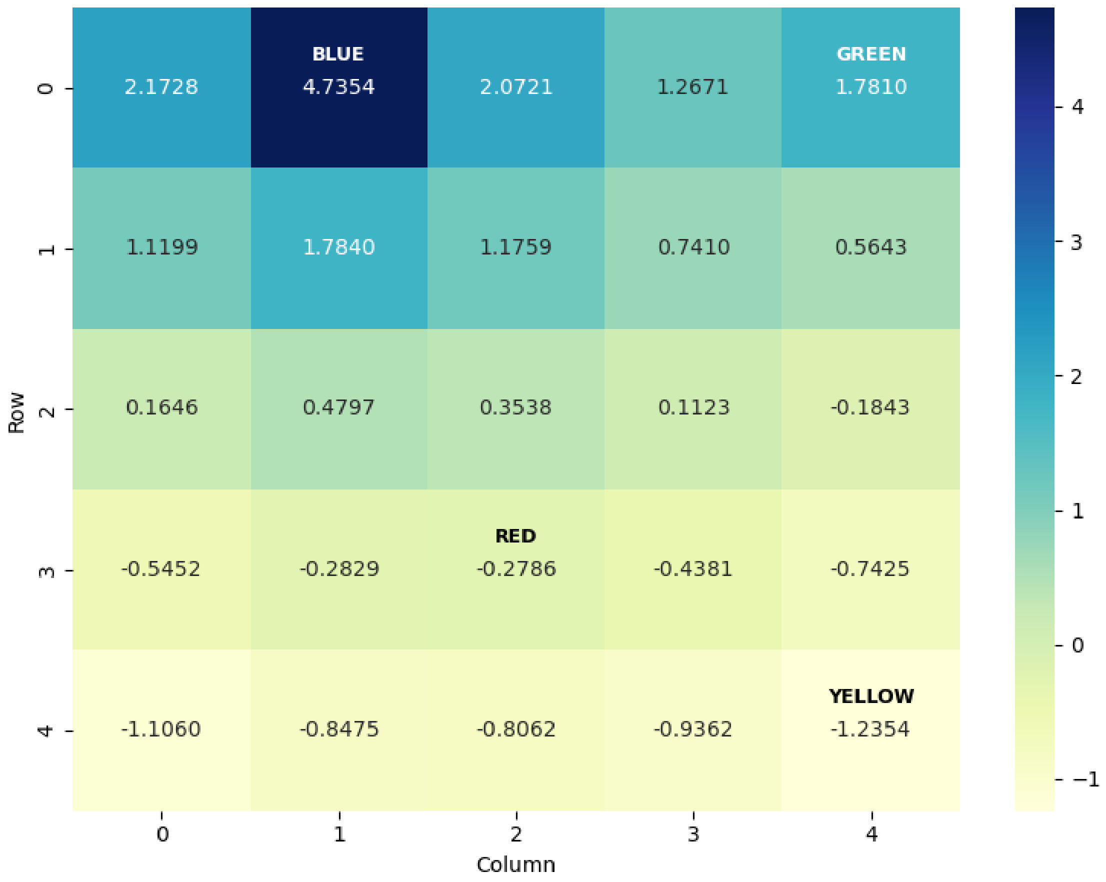

Figure 2: Value function (iterative policy evaluation), $\gamma = 0.95$

Figure 2 heatmap shows the value function computed using iterative policy evaluation for the 5×5 gridworld environment. This method repeatedly updates the value of each state until the **values converge to their true expected returns** under the random policy.

The highest value of approximately **4.7354** is found at **state (0,1)**, which is the blue terminal state. The agent gets an immediate reward of +5 when it reaches this blue square. Being in this position also helps the agent collect more rewards in the future.

The green terminal state at (0,4) shows a value of 1.7810, reflecting its positive reward. The red terminal state at (3,2) has a negative value of -0.2786, indicating the penalty for reaching this state. The yellow terminal state at (4,4) shows the lowest value of -1.2354.

1.1.3 Discussion

Which states have the highest value?

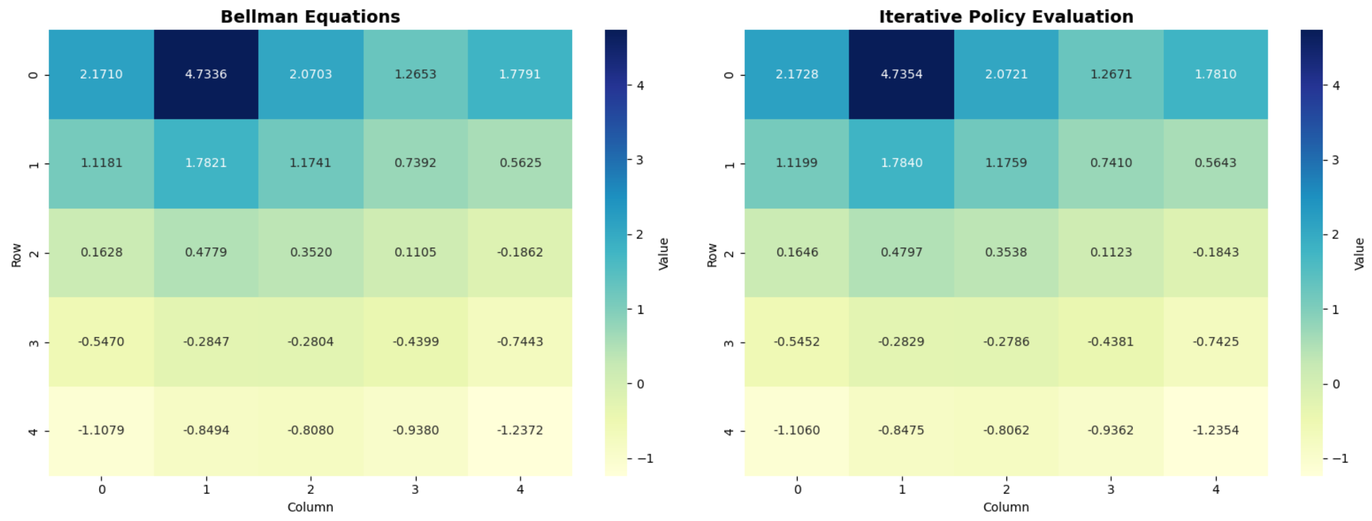

Figure 3: Value function comparison: both methods (bellman equations & iterative policy evaluation)

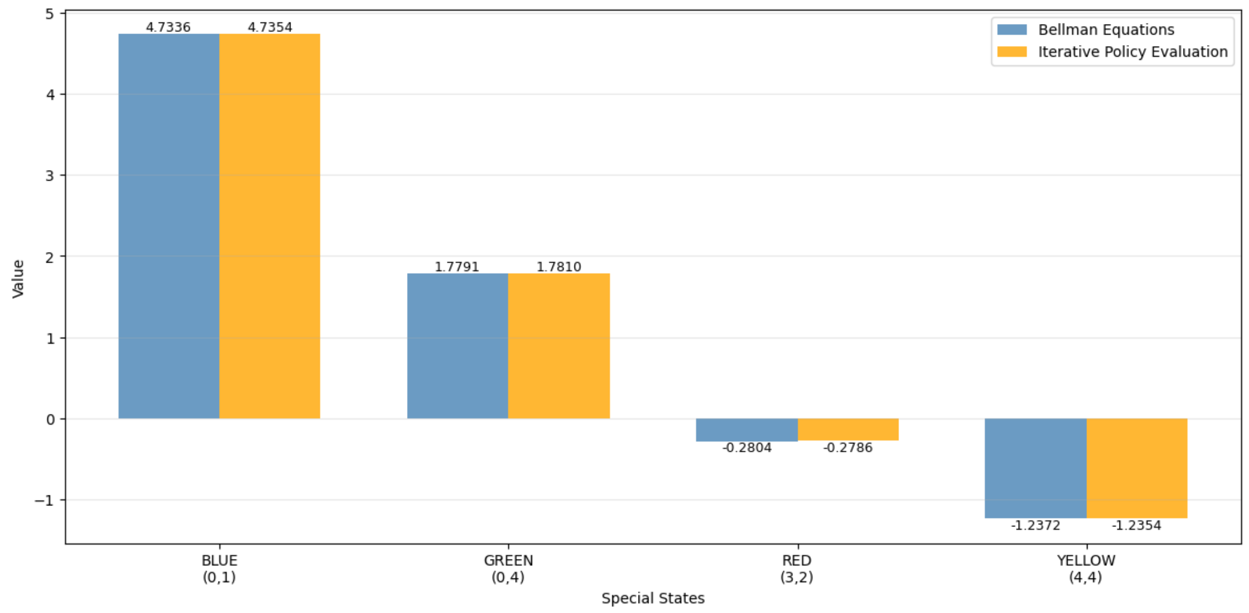

Figure 4: Special state values: method comparison

Considering the figure 3 and figure 4:

Highest Value States:

1. **Blue** terminal state at (0,1) - approximately 4.73 in both methods
2. **Green** terminal state at (0,4) - approximately 1.78 in both methods

Does this surprise you?

This result is **not surprising** for **several reasons**:

- Blue state has the highest value because it gives the agent the biggest immediate reward of +5 points. When the agent reaches this square, it gets these points right away, making it the most valuable position.
- Green state comes second because it also provides a positive reward, though smaller than the blue state. This makes it the second most attractive destination for the agent.
- Under a **uniform random policy** where the agent moves in any direction with equal probability, **states that give immediate positive rewards** will naturally have the highest values. The blue state offers the largest reward, so it's logical that it would be the most valuable state to reach.
- Both methods (Bellman equations and iterative policy evaluation) give nearly identical results, confirming that these are indeed the optimal value estimates for the random policy.

1.2 Part 01: Optimal policy

1.2.1 Explicitly solving the Bellman optimality equation

For each state s, the optimal value function $V^*(s)$ satisfies the Bellman optimality equation:

$$V^*(s) = \max_a \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma \cdot V^*(s')\right]$$

This equation is **non-linear** due to the presence of the max operator, so we cannot solve it using a simple linear system such as:

$$A \cdot V = b$$

Instead, we use **value iteration**, an iterative algorithm that repeatedly applies the Bellman optimality update and converges to the optimal value function $V^*(s)$.

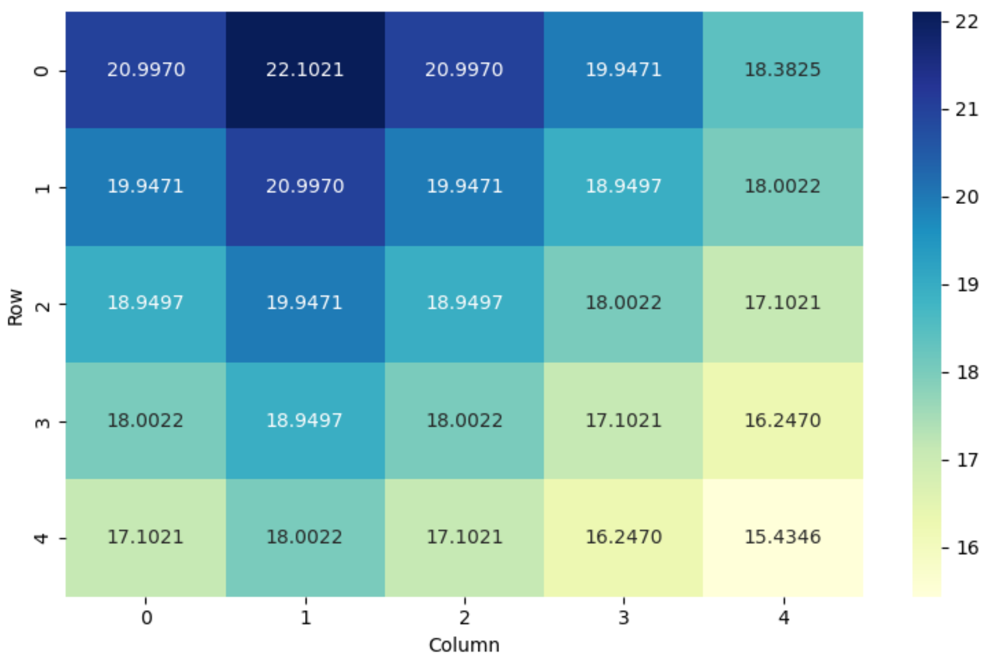

Figure 5: State values via Bellman optimality equation

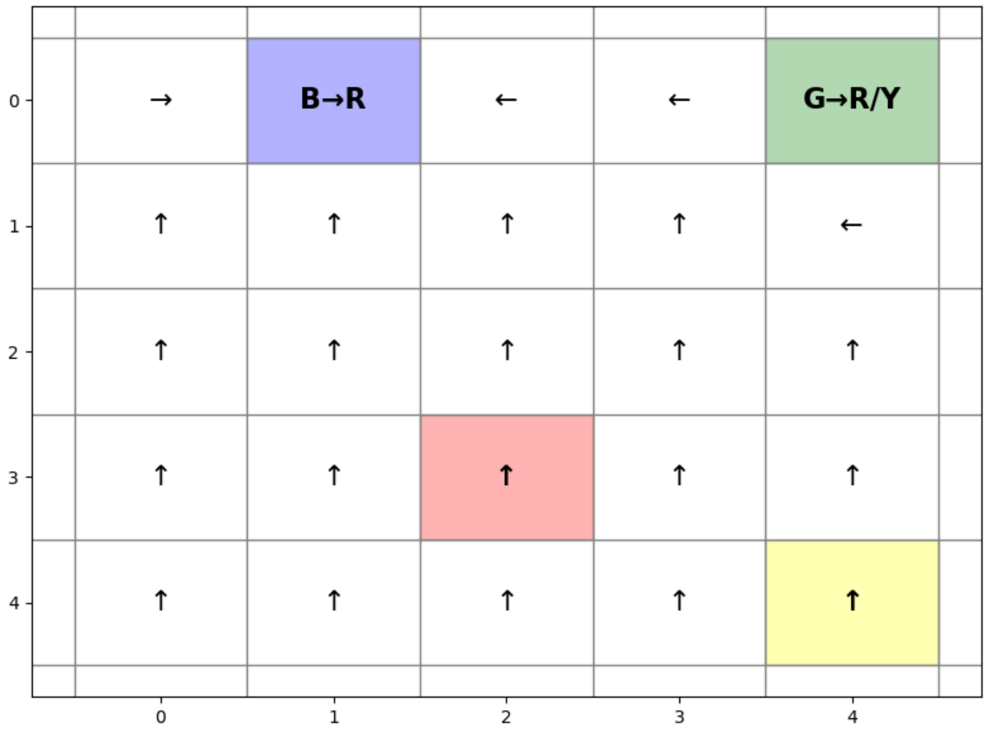

Figure 6: Optimal policy learned via Bellman optimality equation

Based on figure 5, figure 6 and the results obtained:

- The value iteration algorithm for Bellman Optimality Equation converged after 212 iterations with a final delta of 0.0001.
- The highest value of **22.1021** is achieved at the blue terminal state at position (0,1), confirming that the algorithm correctly identifies the most rewarding state due to its immediate reward of +5.
- **Policy Optimality:** The optimal policy demonstrates most states show upward arrows (↑) directing the agent toward the high-value blue.

1.2.2 Policy iteration with iterative policy evaluation

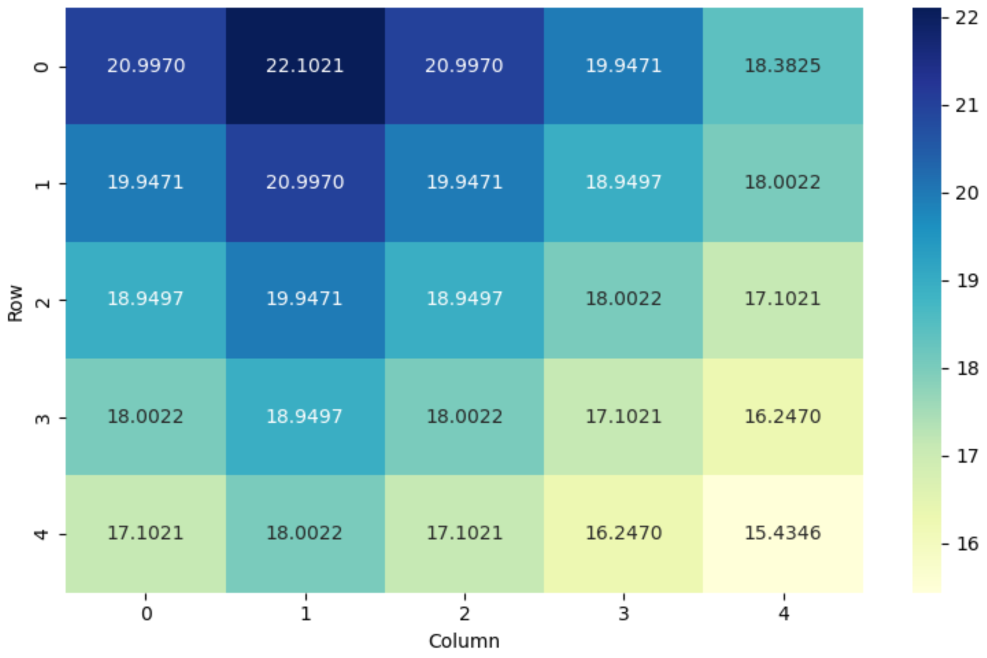

Figure 7: State values via policy iteration

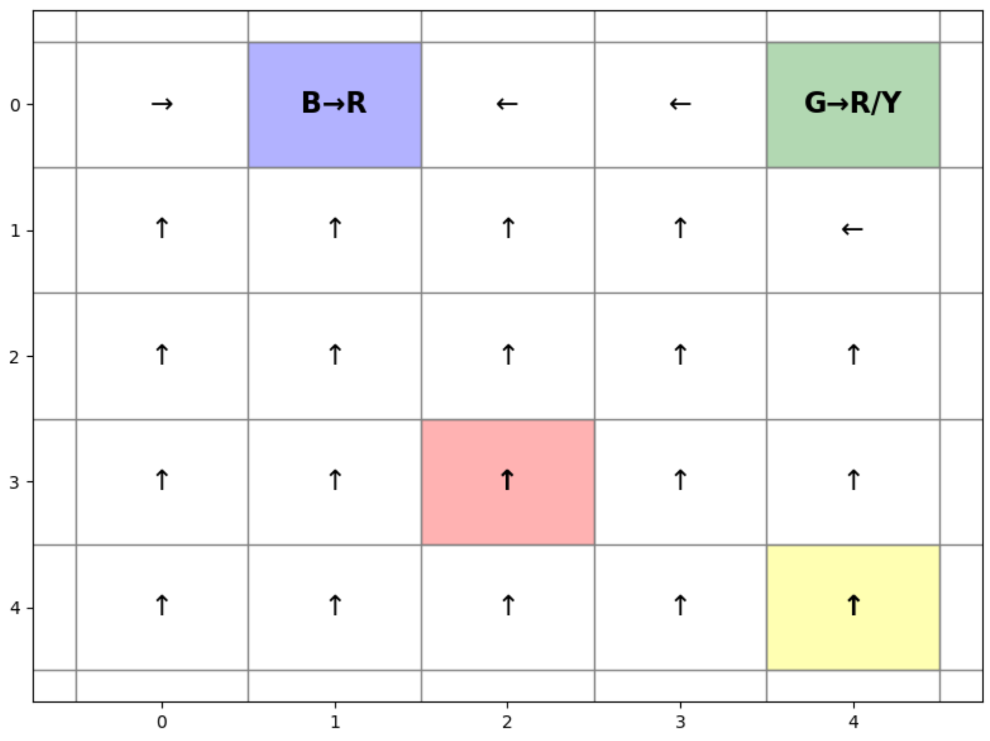

Figure 8: Optimal policy learned via policy iteration

Based on figure 7, figure 8 and the results obtained:

- Policy iteration with iterative policy evaluation converged in only **5 iterations**, demonstrating significantly faster convergence compared to value iteration algorithm for Bellman optimality equation (212 iterations).
- The algorithm showed systematic improvement with policy changes decreasing from 19 states in iteration 1 to 0 changes in iteration 5.
- The final value function matches the value iteration results exactly, with the highest value of **22.1021** at the blue terminal state (0,1), confirming both methods find the same optimal policy.

1.2.3 Policy improvement with value iteration

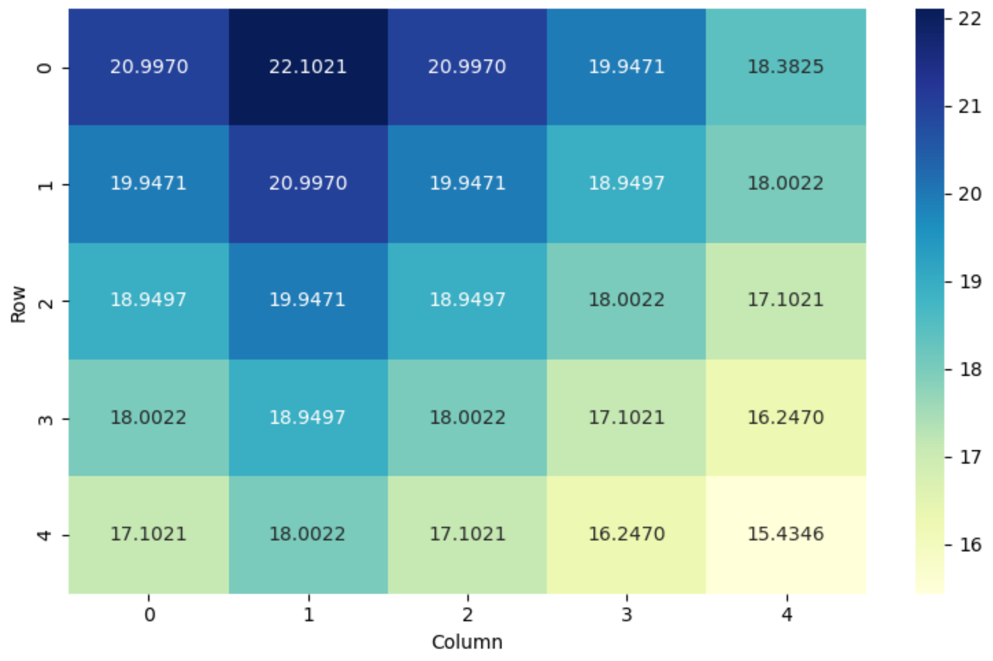

Figure 9: State values via policy improvement with value iteration

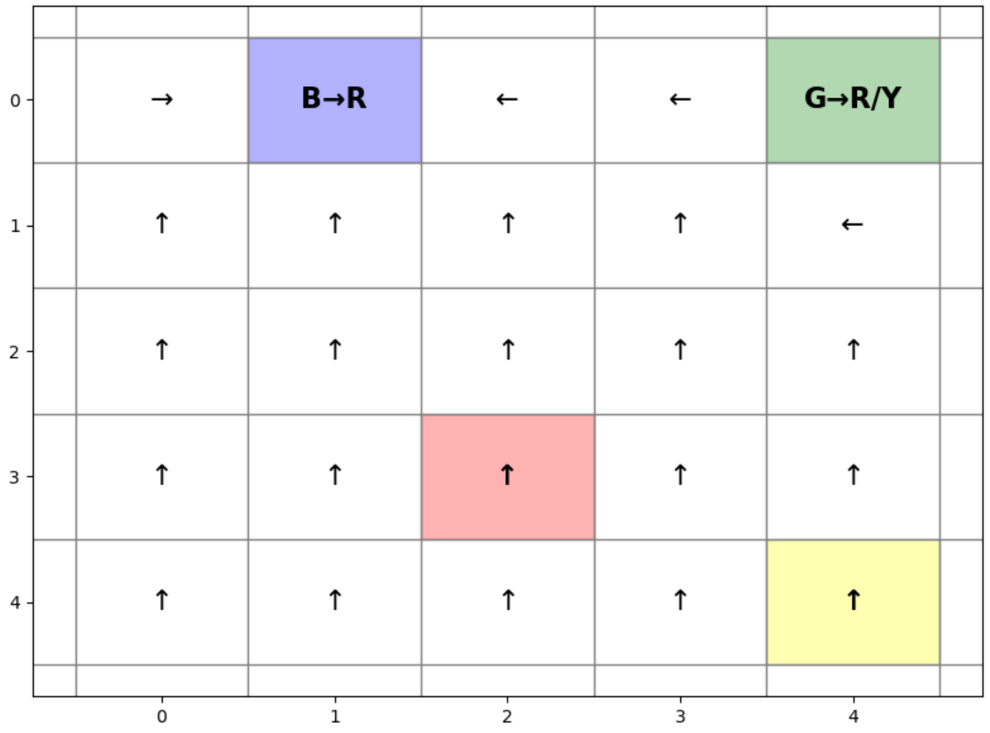

Figure 10: Optimal policy learned via policy improvement with value iteration

Based on figure 9, figure 10 and the results obtained:

- This approach combines initial policy evaluation (168 iterations) with full value iteration (212 iterations) to explore the policy improvement process from a starting policy.
- The policy improvement achieved a certain increase in value function performance, from an initial maximum value of 5.0 to the optimal value of **22.1021**.
- Policy improvement modified **18 out of 23** normal states, indicating that the initial random policy was significantly suboptimal.

1.2.4 Final analysis

Table 11 presents a comprehensive comparison of the three dynamic programming methods implemented for solving the grid world problem. The results demonstrate that while all methods converge to essentially the same **optimal value (approximately 22.10)**. Policy iteration achieves convergence in only 5 iterations, making it the **most efficient approach**. Comparatively, both Bellman optimality equation and policy improvement with value iteration require 212 iterations, with the final method requiring an additional 168 iterations for initial policy evaluation.

Table 1: Performance Metrics Comparison (All three methods)

Method	**Iterations**	**Max value**
Bellman optimality equation	212	22.102111
Policy iteration	5	22.102192
Policy improvement with value iteration	168 (initial policy evaluation) + 212 (value iteration)	22.102111

2 Part 02:

2.1 Monte Carlo with exploring starts (ES)

The Monte Carlo method with exploring starts was implemented using an ϵ-soft policy with $\epsilon = 0.1$. The algorithm was trained for 20,000 episodes with a discount factor of $\gamma = 0.95$.

Table 2 presents the key configuration parameters that define the experimental setup for our Monte Carlo implementation.

Table 2: Initial configuration for Monte Carlo with exploring starts

Parameter	**Value**
Grid size	5×5
Discount factor (γ)	0.95
Exploration parameter (ϵ)	0.1
Maximum steps per episode	100
Number of episodes	20,000
Policy initialization	equiprobable

2.1.1 Monte Carlo with Exploring Starts Results

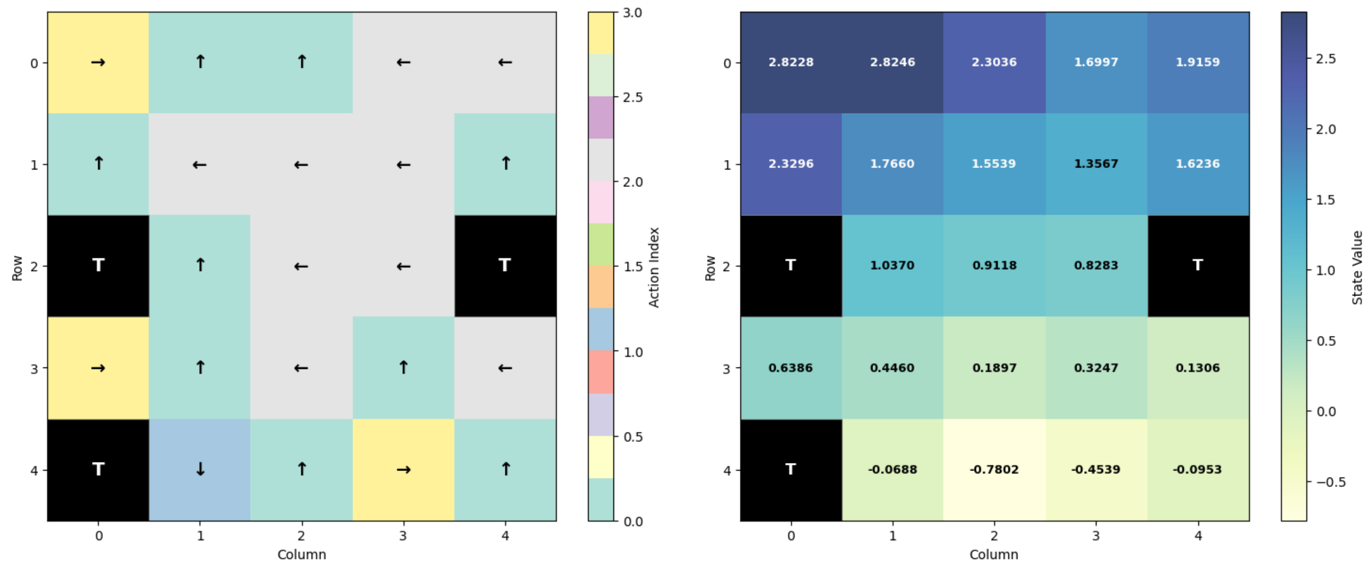

Figure 11: Monte Carlo results : optimal policy with exploring starts (ϵ-soft) (left) and state values (exploring starts) (right)

Figure 11 illustrates the results obtained from the Monte Carlo with exploring starts implementation. The left panel shows the optimal policy learned by the algorithm. In addition where arrows indicate the best action for each state and black squares marked with 'T' represent terminal states. The right panel displays the corresponding state values.

The learned optimal policy is shown in table 3, where arrows indicate the best action for each state and 'T' represents terminal states.

Table 3: Optimal Policy with Exploring Starts

→	↑	↑	←	←
↑	←	←	←	↑
T	↑	←	←	T
→	↑	←	↑	←
T	↓	↑	→	↑

Table 4 presents the final Q-values for the special squares in the gridworld. The state values derived from the Q-values provide insight into the desirability of each state:

- **Maximum State Value:** 2.8246 (Blue square region)
- **Minimum State Value:** -0.7802 (Red square region)

Table 4: Final Q-values for special squares (Monte Carlo with exploring starts)

Square	**Up**	**Down**	**Left**	**Right**
Blue (0,1)	2.8246	2.8240	2.8238	2.8223
Green (0,4)	1.8498	1.4655	1.9159	1.7557
Red (4,2)	-0.7802	-0.7803	-0.7816	-0.8376
Yellow (4,4)	-0.0953	-0.3606	-0.5951	-0.7190

The results demonstrate that the algorithm successfully learned to navigate toward the high-reward teleportation squares (Blue and Green) while avoiding areas that lead to negative rewards. The Blue square shows the highest state values because stepping onto it provides an immediate reward of +5.0 through teleportation, making it highly valuable despite the Red square destination having negative continuation values.

2.2 Monte Carlo with ϵ-soft policy (no ES)

To evaluate the impact of exploration on learning performance, the Monte Carlo method without exploring starts was implemented using varying ϵ values while maintaining consistent experimental parameters. Table 5 presents the configuration parameters used across all experiments.

Table 5: Initial Configuration for Monte Carlo without Exploring Starts

Parameter	**Value**
Grid size	5×5
Discount factor (γ)	0.95
Exploration parameter (ϵ)	0.1, 0.3, 0.5, 0.7, 0.8, 0.9
Maximum steps per episode	100
Number of episodes	20,000
Policy initialization	equiprobable
Starting state	fixed at (0,0)
Episode initialization	policy-based action selection

2.2.1 Monte Carlo with ϵ-soft policy (no ES) results

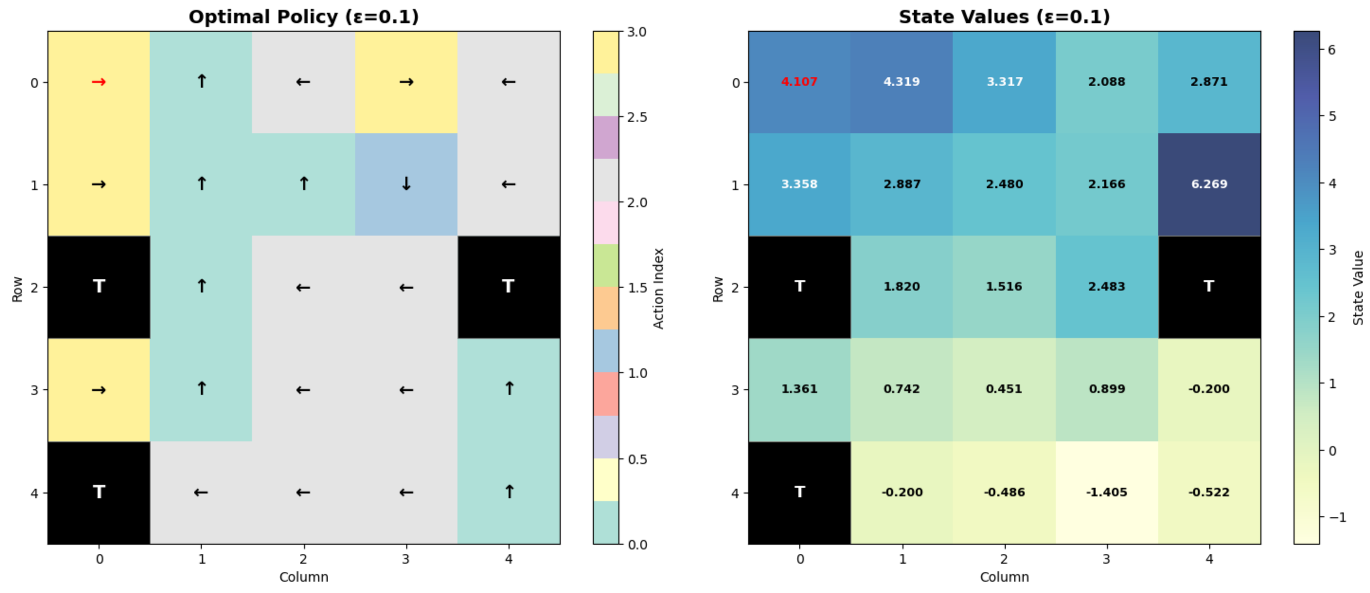

Figure 12: Monte Carlo results : optimal policy without exploring starts with (ϵ-soft = 0.1) (left) and state values (without exploring exploring starts) (right)

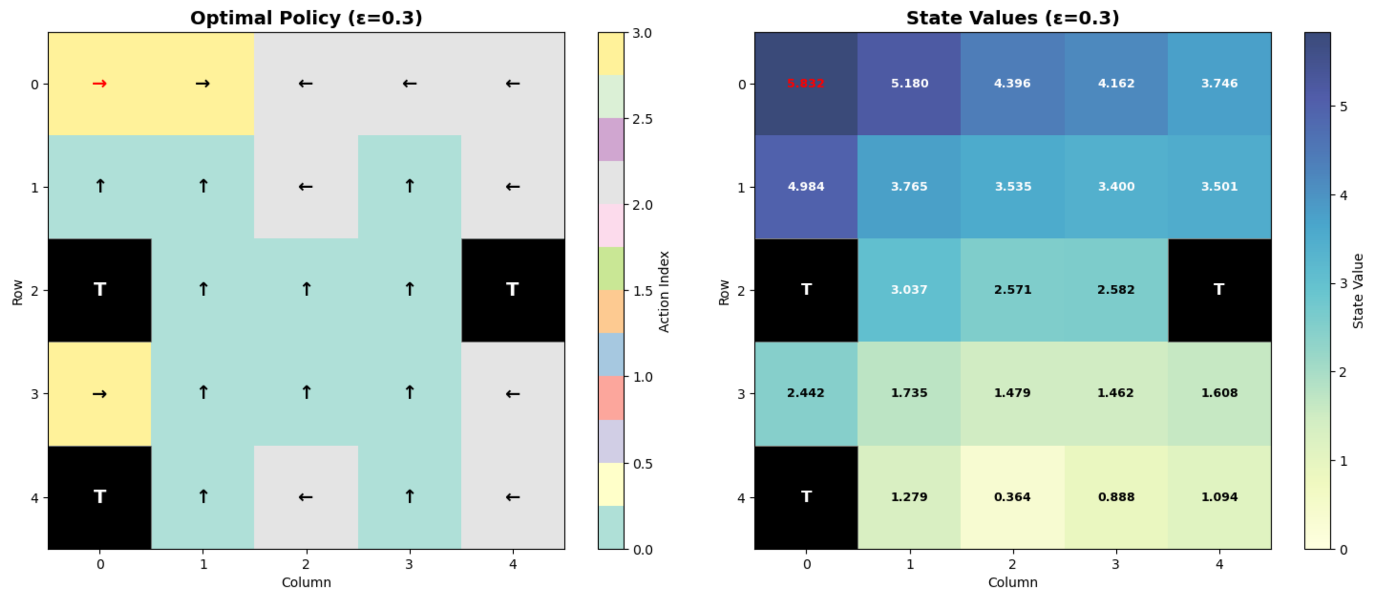

Figure 13: Monte Carlo results : optimal policy without exploring starts with (ϵ-soft = 0.3) (left) and state values (without exploring exploring starts) (right)

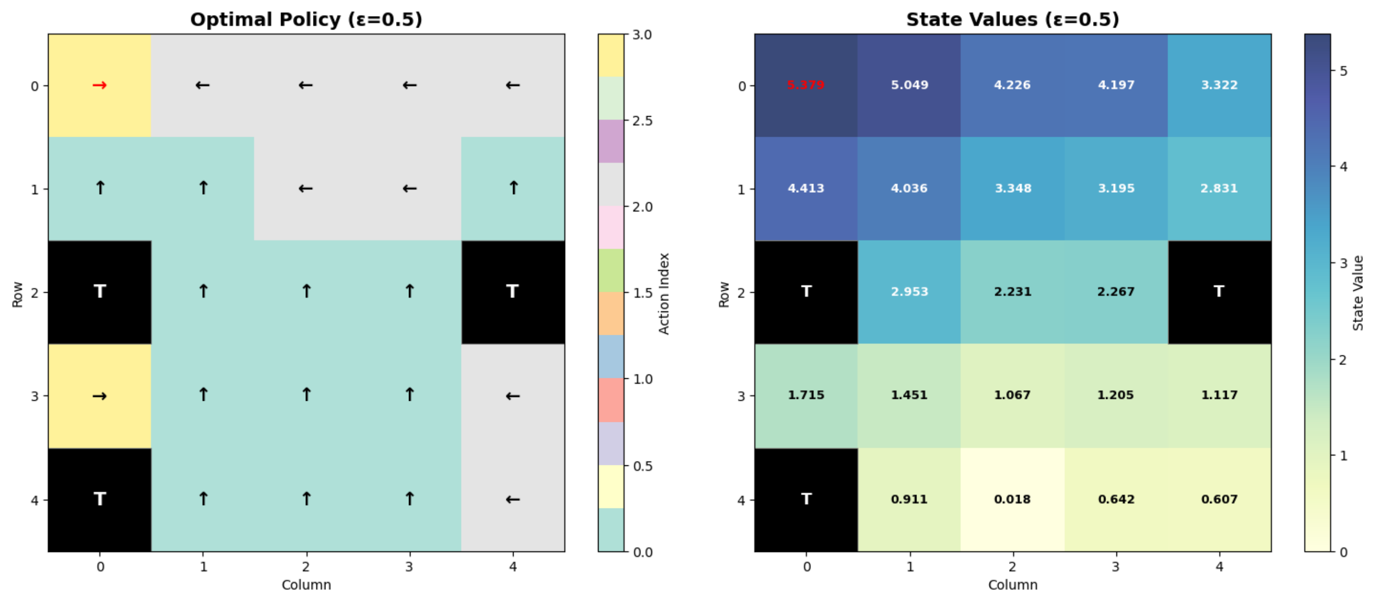

Figure 14: Monte Carlo results : optimal policy without exploring starts with (ϵ-soft = 0.5) (left) and state values (without exploring exploring starts) (right)

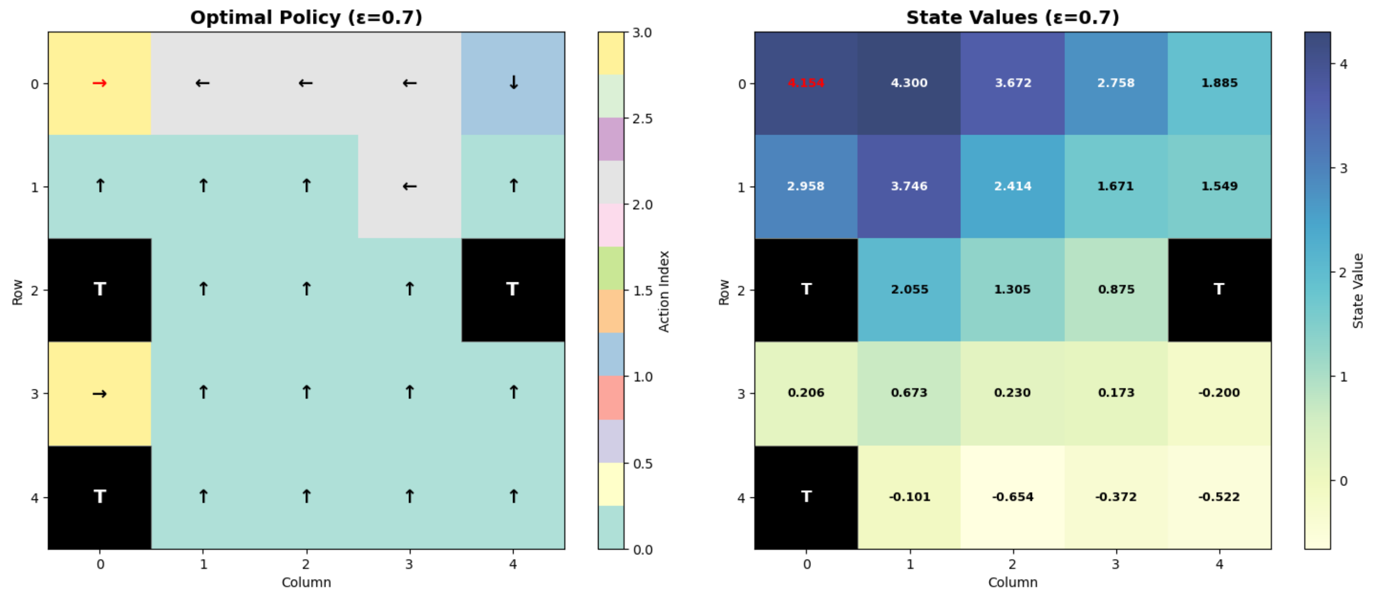

Figure 15: Monte Carlo results : optimal policy without exploring starts with (ϵ-soft = 0.7) (left) and state values (without exploring exploring starts) (right)

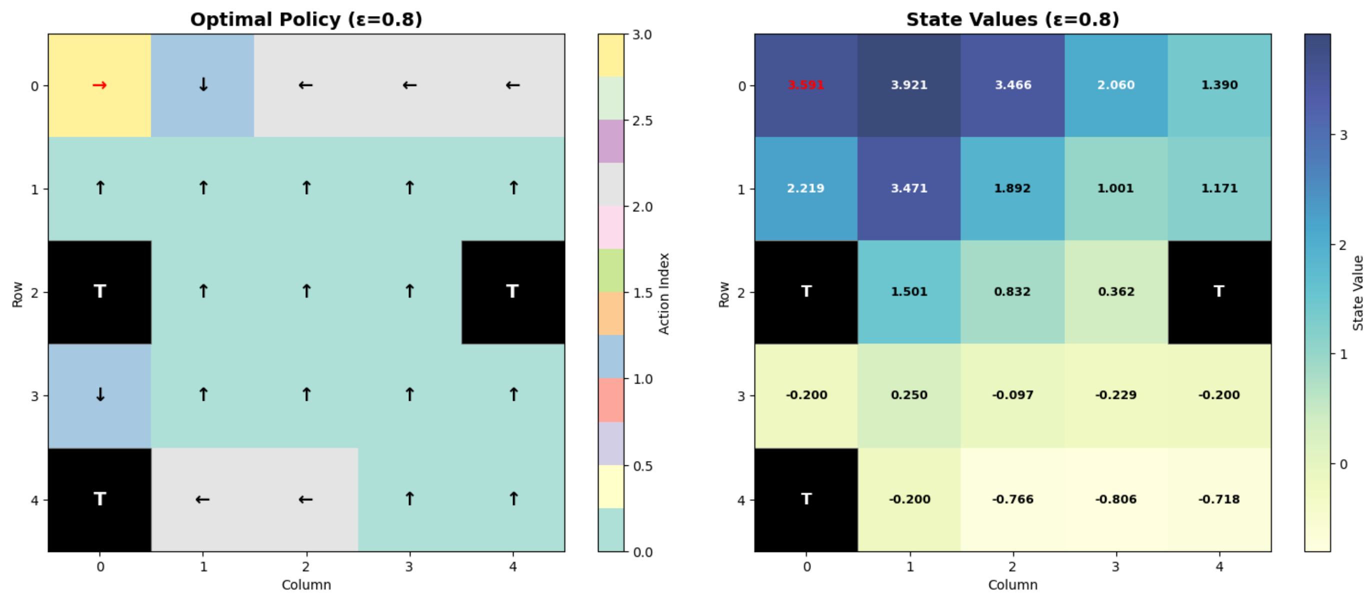

Figure 16: Monte Carlo results : optimal policy without exploring starts with (ϵ-soft = 0.8) (left) and state values (without exploring exploring starts) (right)

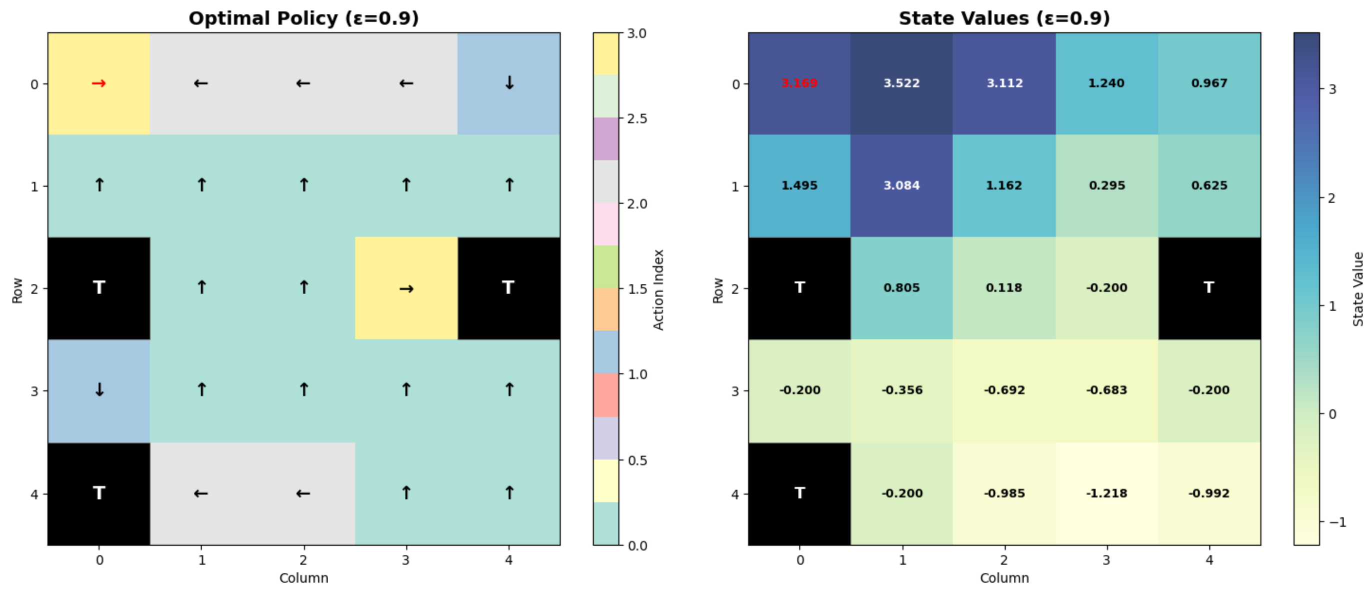

Figure 17: Monte Carlo results : optimal policy without exploring starts with (ϵ-soft = 0.9) (left) and state values (without exploring exploring starts) (right)

The experimental results from Figures 12, 13, 14, 15, 16, 17, reveal a complex relationship between exploration intensity and learning performance:

$\epsilon = 0.1$ **(Poor exploration):** (Figure 12)

- **Fixed starting bias**: Always starts at (0,0), limited exploration

- **Blue square underexplored**: Only 10% random actions insufficient
- **Local optimum found**: Agent discovers alternative high-value path at (1,4)
- **Value 6.2695 at (1,4) > 4.3194 at Blue square**

$\epsilon = 0.3 - 0.5$ **(Optimal range):** (Figures 13, 14))

- **Better exploration**: 30-50% exploration allows broader search
- **Blue square properly valued**: $\sim$5.0 + additional values indicate good discovery
- **Balanced exploration-exploitation**: Finds near-optimal policies
- **Highest Value at (0,0) ($\epsilon = 0.3$ - 5.8325 and $\epsilon = 0.5$ - 5.3785 > Blue square $\epsilon = 0.3$ - 5.1801 and $\epsilon = 0.5$ - 5.0492)**

$\epsilon = 0.7 - 0.9$ **shows blue maximum but lower values:** (Figures 15, 16, 17)
High Exploration Problem:

- **Too much randomness**: 70-90% random actions
- **Policy degradation**: Excessive exploration prevents convergence
- **Value decrease trend**: $4.30 \rightarrow 3.92 \rightarrow 3.52$ as ϵ increases
- **Blue becomes maximum by elimination**: Other areas explored poorly

As the ϵ value increases in the ϵ-soft policy, the agent explores more but performance doesn't always improve. Very low ϵ values ($\epsilon = 0.1$) suffer from **insufficient exploration**, leading to **exploration bias** where the agent gets trapped in local optima near the fixed starting state (0,0). ϵ values ($\epsilon = 0.3 - 0.5$) achieve optimal performance by **balancing exploration and exploitation**. These configurations properly discover and value the blue square teleportation mechanism, resulting in the **highest Blue square approximate values (5.18 and 5.05 respectively)**.High ϵ values ($\epsilon = 0.7 - 0.9$) demonstrate the adverse effects of excessive exploration. While the Blue square becomes the maximum-valued state in these configurations, the **overall performance degrades significantly**, with Blue square values declining from approximately 4.30 to 3.52 as exploration increases.

To be concluded, Monte Carlo method without exploring starts demonstrates optimal performance at moderate exploration levels ($\epsilon = 0.3 - 0.5$), with insufficient exploration causing local optima and excessive exploration degrading policy convergence.

2.3 Off-policy Monte Carlo control with importance sampling and equiprobable behavior policy

The following table 6 represents the initial configuration parameters used for the Monte Carlo with importance sampling algorithm:

Table 6: Initial configuration for Monte Carlo with importance sampling

Parameter	Value
Grid size	5×5
Discount factor (γ)	0.95
Maximum steps per episode	100
Behavior policy	equiprobable (uniform)
Target policy	greedy (deterministic)
Episodes (Experiment 1)	20,000
Episodes (Experiment 2)	200,000

2.3.1 Off-policy Monte Carlo control with importance sampling and equiprobable behavior policy results

Experiment 1: 20,000 episodes

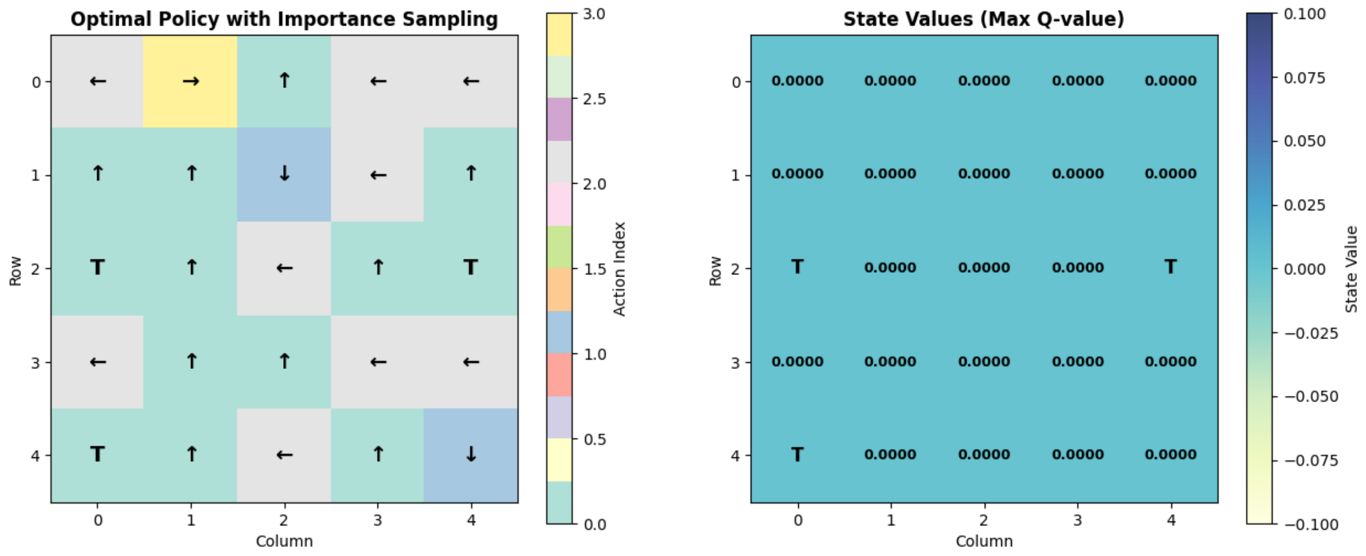

Figure 18: Monte Carlo results : optimal policy with importance sampling (left) and state values with importance sampling (right) - (Episodes 20000)

Based on the table 7, it was noticed that consistently decreasing (more negative) average Q-values indicate the algorithm is primarily learning from low-reward regular movements and boundary violations, with insufficient exposure to the high-reward special transitions.

Table 7: Average Q-Value during training (20000 episodes)

Episode	**Average Q-value**
5,000	-0.2231
10,000	-0.2474
15,000	-0.2600
20,000	-0.2652

Based on figure 18, **20,000 episode** experiment demonstrates a critical failure mode in Monte Carlo with importance sampling, revealing the severe consequences of **insufficient sampling**. The training progression shows **consistently declining Q-values** from -0.2231 to -0.2652, indicating that the algorithm only learned about movement penalties (-0.2) and boundary violations (-0.5) while **completely failing to discover the high-reward special transitions.** Moreover, **all state values remain exactly 0.0000**, with both the **blue square (reward +5) and green square (reward +2.5)** showing zero Q-values [0, 0, 0, 0], proving the algorithm never experienced these crucial transitions. The cumulative importance weights totaling exactly 20,000 (equal to episode count) further confirms that no high-value trajectories were discovered.

The necessity for 200,000 episodes becomes clear when considering the theoretical requirements for off-policy learning success.

Experiment 1: 200,000 episodes

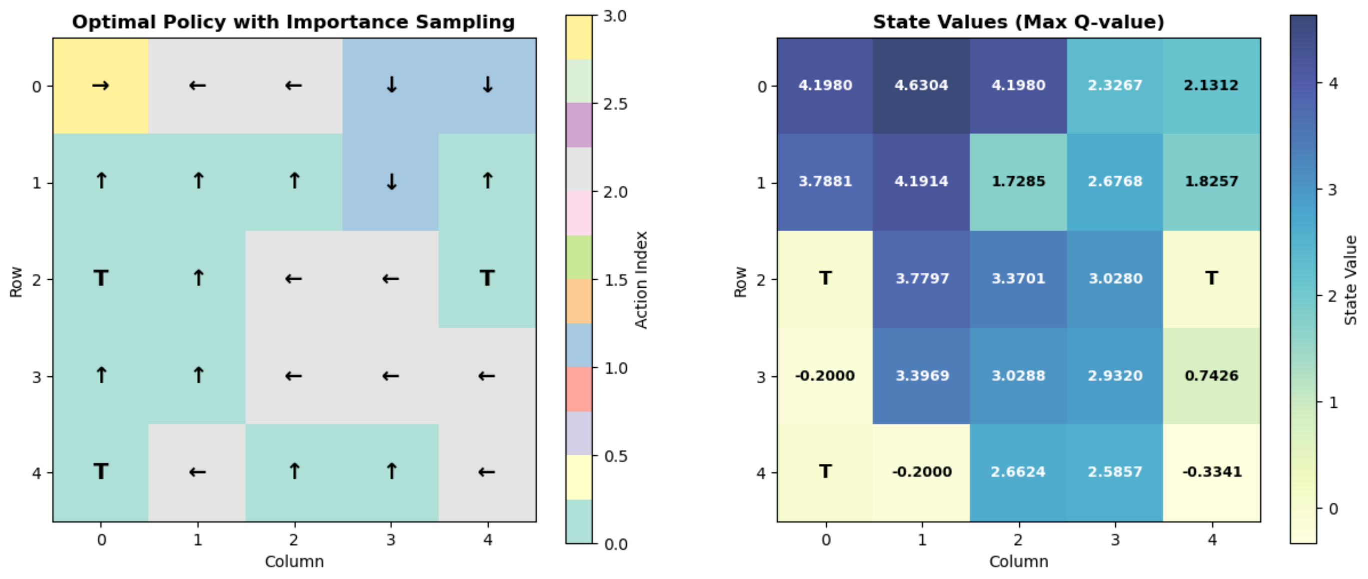

Figure 19: Monte Carlo results : optimal policy with importance sampling (left) and state values with importance sampling (right) - (Episodes 200000)

Table 8: Average Q-Value Progression During Training (200,000 episodes)

Episodes 5,000 - 100,000		Episodes 105,000 - 200,000	
Episode	**Average Q-Value**	**Episode**	**Average Q-Value**
5,000	-0.2231	105,000	0.6403
10,000	-0.2474	110,000	0.6406
15,000	-0.2600	115,000	0.6406
20,000	-0.2652	120,000	0.6064
25,000	-0.2692	125,000	0.5945
30,000	-0.2622	130,000	0.5952
35,000	-0.2819	135,000	0.5955
40,000	-0.2836	140,000	0.5942
45,000	**0.0397**	145,000	0.5942
50,000	0.1256	150,000	0.5837
55,000	0.2106	155,000	1.1526
60,000	0.2041	160,000	1.2026
65,000	0.2440	165,000	1.2028
70,000	0.2431	170,000	1.2031
75,000	0.2771	175,000	1.2033
80,000	0.2771	180,000	1.2037
85,000	0.2659	185,000	1.1872
90,000	0.2572	190,000	1.1872
95,000	0.2572	195,000	1.1869
100,000	0.2441	200,000	1.1871

Based on figure 19 and table 8, 200,000 episode Monte Carlo with importance sampling experiment demonstrates **successful off-policy learning with a dramatic breakthrough at episode 45,000 (highlighted in yellow)**. The algorithm transitions from penalty-only learning (negative Q-values) to discovering high-reward special transitions, ultimately converging to an optimal policy with **maximum state value of 4.6304 at (0,1)**.

Based on the experiments it was confirmed that knowing exact world dynamics **allows precise importance weight computation**, but successful off-policy learning fundamentally depends on **sufficient sample coverage**: the dramatic transition from **zero learning at 20,000 episodes** to **optimal convergence at 200,000 episodes** demonstrates that importance sampling requires critical mass to discover high-reward trajectories.

3 Code Repository

The complete code and `README` for reproducing the results are available at:
`https://github.com/wbandaramun/reinforcement-learning-assignment-2`