PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

MDPs and Monte Carlo

IIC3675 - Reinforcement Learning

Group 40

Group members:
William Aarland, Pascal Lopez Wilkendorf

Delivery date: 13th of April 2025

1 Introduction

This report corresponds to the second assignment of the course IIC3675: Reinforcement Learning, which focuses on solving Markov Decision Processes (MDPs) using two key techniques: **Dynamic Programming** and **Monte Carlo methods**.

The assignment is based on implementing and evaluating various algorithms presented in Chapters 3, 4, and 5 of [1]. Through the resolution of different problems in the task we aim to explore value estimation, policy evaluation, policy improvement, and control via reinforcement learning.

We implement classic algorithms like iterative policy evaluation, value iteration, and on-policy Monte Carlo control, while analyzing their performance.

2 Tasks

In this section, we present the solution to each of the tasks described in the assignment. For each part, we include explanations, implementation details, results (with plots and tables where appropriate), and our analysis or justification as requested.

a) Demonstrating properties of the value function v_π

We want to show that:

$$v_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a \mid s)\, q_\pi(s, a)$$

Proof. By definition, the state-value function under a policy π is:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] \tag{1}$$

Applying the Law of Total Expectation (Equation (4)), over the first action taken:

$$v_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a \mid s)\, \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \tag{2}$$

Then, using the definition of the action-value function:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] \tag{3}$$

we substitute into Equation 2:

$$\begin{aligned} v_\pi(s) &= \sum_{a \in \mathcal{A}(s)} \pi(a \mid s)\, \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \\ &= \sum_{a \in \mathcal{A}(s)} \pi(a \mid s)\, q_\pi(s, a) \end{aligned}$$

Thus, the property is shown. □

Properties used

- Law of Total Expectation

$$\mathbb{E}\left[X\right] = \sum_i P(B_i) \cdot \mathbb{E}\left[X \mid B_i\right] \tag{4}$$

b) Demonstrating properties of the action-value function q_π

We want to show that:

$$q_\pi(s,a) = \sum_{r\in\mathcal{R}} \sum_{s'\in\mathcal{S}} p\left(r, s' \mid s, a\right)\left(r + \gamma v_\pi(s')\right) \tag{5}$$

Proof. By definition, the action-value function under a policy π is:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] \tag{6}$$

Using the recursive definition of return $G_t = R_{t+1} + \gamma G_{t+1}$ (Equation (3.9) in [1]), we expand:

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} \mid s, a\right] + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid s, a\right] \end{aligned}$$

To evaluate these expectations, we consider all possible pairs (r, s') of immediate rewards and next states, governed by the environment's transition dynamics $p(r, s' \mid s, a)$. Since action a is fixed, we do not sum over actions.

By the definition of expected value for discrete random variables:

$$\begin{aligned} q_\pi(s,a) &= \sum_{r\in\mathcal{R}} \sum_{s'\in\mathcal{S}} p(r, s' \mid s, a)\left[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']\right] \\ &= \sum_{r\in\mathcal{R}} \sum_{s'\in\mathcal{S}} p(r, s' \mid s, a)\left(r + \gamma v_\pi(s')\right) \end{aligned}$$

Thus, we have shown:

$$q_\pi(s,a) = \sum_{r\in\mathcal{R}} \sum_{s'\in\mathcal{S}} p(r, s' \mid s, a)\left(r + \gamma v_\pi(s')\right)$$

□

c)

This task is given with a deterministic Markov Decision Problem (MDP), where there are two policies π_l or π_r. The task is to find the optimal policy given different reward discount values γ. For this task, the following information is given:

- States: $S = \{s_0, s_r, s_l\}$
- Actions: $A = \{a, a_r, a_l\}$, where
  - $A(s_0) = \{a_r, a_l\}$
  - $A(s_l) = A(s_r) = \{a\}$
- Rewards: $R = \{0, 1, 2\}$
- Transitions:
  - $p(s_l, 1 \mid s_0, a_l) = 1$
  - $p(s_r, 0 \mid s_0, a_r) = 1$
  - $p(s_0, 0 \mid s_l, a) = 1$
  - $p(s_0, 2 \mid s_r, a) = 1$
- Deterministic policies:

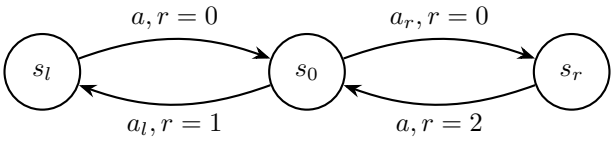

Figure 1: Visualization of the MDP in task c)

- $\pi_l(s_0) = a_l$
- $\pi_r(s_0) = a_r$

This information is visualized in Figure 1.

Since the policies are deterministic, the policies and state transitions are given as follows:

- Policy π_l: $s_0 \xrightarrow{a_l,\ r=1} s_l \xrightarrow{a,\ r=0} s_0 \xrightarrow{a_l,\ r=1} \dots$
- Policy π_r: $s_0 \xrightarrow{a_r,\ r=0} s_r \xrightarrow{a,\ r=2} s_0 \xrightarrow{a_r,\ r=0} \dots$

Under the assumptions that this policy continues to infinity, $|\gamma| < 1$, and once a policy is chosen it does not change during the course of time, we can calculate the returns of the different policies.

$$V_{\pi_l}(s_0) = 1\gamma^0 + 0\gamma^1 + 1\gamma^2 + 0\gamma^3 + \dots \tag{7}$$
$$= \sum_{k=0}^{\infty} \gamma^{2k} \tag{8}$$
$$= \frac{1}{1-\gamma^2} \tag{9}$$

$$V_{\pi_r}(s_0) = 0\gamma^0 + 2\gamma^1 + 0\gamma^2 + 2\gamma^3 + \dots \tag{10}$$
$$= 2\gamma \sum_{k=0}^{\infty} \gamma^{2k} \tag{11}$$
$$= \frac{2\gamma}{1-\gamma^2} \tag{12}$$

Then the value-functions for the policies π_l and π_r will be Equations (9) and (12) respectively. For different discount factors, we can calculate and find the optimal policy. The results are presented in Table 1.

Discount factor γ	$V_{\pi_l}(s_0)$	$V_{\pi_r}(s_0)$	**Optimal policy**
0	1.00	0.00	π_l
0.5	1.33	1.33	$\pi_l \vee \pi_r$
0.9	5.26	9.47	π_r

Table 1: Value functions for different policies under varying discount factors

d) Iterative Policy Evaluation Results

Below are the results of running the *Iterative Policy Evaluation* on different MDP with different initial conditions. The results of the simulations to the **GridProblem**, **CookieProblem**, and **GamblerProblem** are presented in Tables 2 to 4 respectively.

Grid Size	$V(s_0)$	**Time** (s)
3x3	−8.000	0.009
4x4	−18.000	0.054
5x5	−37.333	0.145
6x6	−60.231	0.289
7x7	−93.496	1.028
8x8	−131.193	1.178
9x9	−180.001	1.910
10x10	−233.879	3.080

Table 2: Iterative policy evaluation results for **GridProblem**

Grid Size	$V(s_0)$	**Time** (s)
3x3	5.380	1.350
4x4	2.477	4.045
5x5	1.291	9.454
6x6	0.729	18.770
7x7	0.437	34.389
8x8	0.273	56.512
9x9	0.177	89.563
10x10	0.118	141.377

Table 3: Iterative policy evaluation results for **CookieProblem**

P**(heads)**	$V(s_0)$	**Time** (s)
0.25	0.067	0.574
0.40	0.284	0.678
0.55	0.612	0.876

Table 4: Iterative policy evaluation results for **GamblerProblem**

e) Iterative Policy Evaluation: Analysis of Results

The convergence time varies primarily due to the size of the state space, the amount of randomness in the transitions, and the value of the discount factor. For example, the problem that takes the longest to solve is CookieProblem 10x10 (141.377 seconds), significantly more than GridProblem 10x10 (3.080 seconds), even though both use grids of the same size. This is most likely due to the more complex structure of the CookieProblem. This problem includes both the states of agent, and the cookie. Furthermore, the cookie respawns in a random location after being eaten, and a discount factor of $\gamma = 0.99$ leads to a slower convergence.

f) Affects of lowering discount factor

Lowering the discount factor γ generally leads to faster convergence in iterative policy evaluation. This is because a smaller γ causes the algorithm to prioritize immediate rewards, which means the value function stabilizes more quickly. When γ is high, the agent considers rewards that are far in the future, and value updates must propagate through many more states, which increases the number of iterations required to meet the convergence threshold.

g) Evaluation of the Optimality of Greedy Policies

The greedy policies derived from the value estimates in Section 2d were evaluated to assess their optimality by applying the greedy operator once again. In other words, we computed the greedy policy of the already obtained greedy policy. If the policy does not change after this operation, it indicates that it is already optimal with respect to its value function.

In the **GridProblem**, this test confirmed optimality: the greedy policy remained unchanged, and the value function consists of exact multiples of -2, which aligns with the expected number of steps to reach the goal under an optimal shortest-path strategy.

In the **CookieProblem**, the policy found from the beginning was already optimal. Applying the greedy operator again resulted in the same policy, confirming that the initial value estimates led to an optimal strategy, even under the increasing complexity and stochastic nature of the environment as grid size grows.

In the **GamblerProblem**, optimal policies were found for all values of p_h except when $p_h = 0.55$. In this case, the greedy policy changed slightly upon reapplying the greedy operator, suggesting that the original estimates were not fully converged. This is expected, as higher values of p_h increase the number of viable actions and make the optimal strategy more nuanced.

All of these results are presented in Table 5.

Problem	$V_{\mathbf{greedy}}(s_0)$	**Optimal**
GridProblem 3x3	-2.000	True
GridProblem 4x4	-2.000	True
GridProblem 5x5	-4.000	True
GridProblem 6x6	-4.000	True
GridProblem 7x7	-6.000	True
GridProblem 8x8	-6.000	True
GridProblem 9x9	-8.000	True
GridProblem 10x10	-8.000	True
CookieProblem 3x3	48.760	True
CookieProblem 4x4	35.907	True
CookieProblem 5x5	28.197	True
CookieProblem 6x6	23.063	True
CookieProblem 7x7	19.402	True
CookieProblem 8x8	16.661	True
CookieProblem 9x9	14.535	True
CookieProblem 10x10	12.838	True
GamblerProblem $p_h = 0.25$	0.250	True
GamblerProblem $p_h = 0.4$	0.400	True
GamblerProblem $p_h = 0.55$	0.730	False

Table 5: Greedy Policy Evaluation and Optimality Check Across Different Environments

h) Value Iteration

In this task, we aim to compute the optimal value function by applying the *Bellman Optimality Equation* iteratively. This approach, known as *Value Iteration*, updates the value estimates by repeatedly selecting the action that maximizes the expected return. As the value function converges, we obtain both the optimal value function and the corresponding optimal policy.

The results for each domain are presented in Table 6, which includes the estimated optimal value of the initial state $V(s_0)$ and the total execution time for convergence, using a convergence threshold of $\theta = 10^{-10}$.

Problem	**Optimal Value $V(s_0)$**	**Execution Time (s)**
GridProblem 3x3	-2.000	0.000
GridProblem 4x4	-2.000	0.001
GridProblem 5x5	-4.000	0.002
GridProblem 6x6	-4.000	0.004
GridProblem 7x7	-6.000	0.006
GridProblem 8x8	-6.000	0.014
GridProblem 9x9	-8.000	0.022
GridProblem 10x10	-8.000	0.030
CookieProblem 3x3	48.760	1.277
CookieProblem 4x4	35.907	3.864
CookieProblem 5x5	28.197	10.187
CookieProblem 6x6	23.063	20.074
CookieProblem 7x7	19.402	37.779
CookieProblem 8x8	16.661	63.546
CookieProblem 9x9	14.535	103.288
CookieProblem 10x10	12.838	156.260
GamblerProblem $p_h = 0.25$	0.250	0.138
GamblerProblem $p_h = 0.4$	0.400	0.161
GamblerProblem $p_h = 0.55$	1.000	14.858

Table 6: Results of **Value Iteration** with $\theta = 10^{-10}$: Estimated optimal values $V(s_0)$ and execution time.

h) GamblerProblem optimal policy

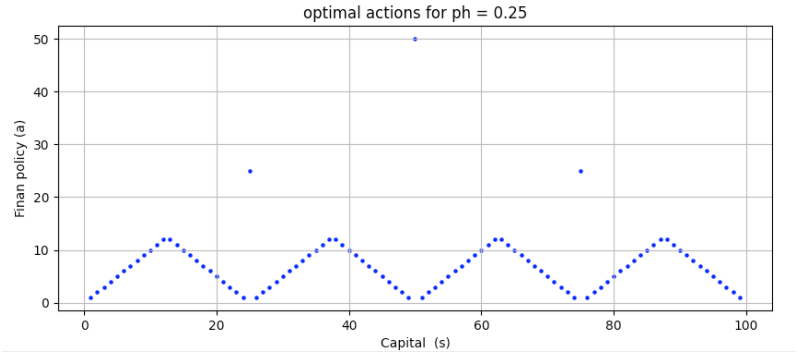

Figure 2: Optimal actions for ph = 0.25

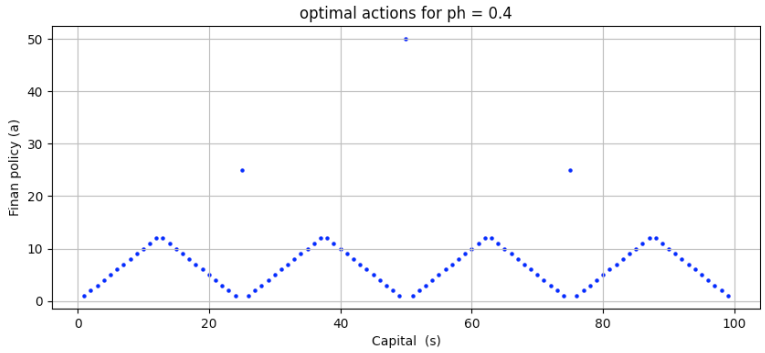

Figure 3: Optimal actions for ph = 0.4

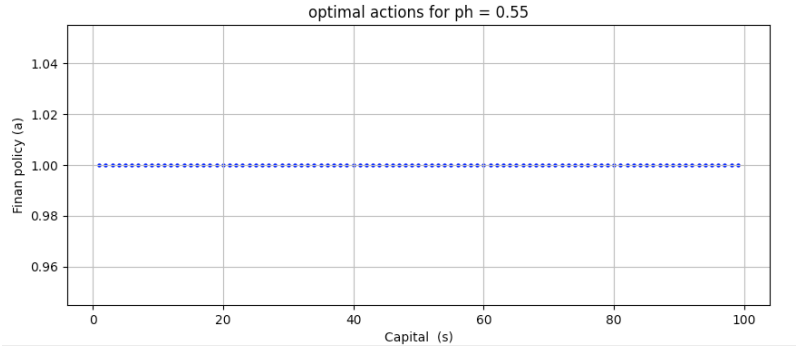

Figure 4: Optimal actions for ph = 0.55

j) On-policy every-visit MC

We implemented a variant of the on-policy every-visit Monte Carlo (MC) control algorithm based on Section 5.4 in [1], incorporating the following adaptations:

- **Incremental Q-value updates**: Instead of storing complete returns, we applied an incremental update rule similar to bandit algorithms (Section 2.4 in [1]), which improves both memory efficiency and computational speed.
- **Every-visit approach**: Q-values were estimated by considering all occurrences of each (state, action) pair within an episode, rather than using the first-visit method.
- **ϵ-greedy policy**: An ϵ-greedy policy was used to balance exploration and exploitation, with $\epsilon = 0.01$ for Blackjack and $\epsilon = 0.1$ for CliffWalking.

Blackjack:

- **Episodes:** 10 million per run, with 5 independent runs.
- **Evaluation:** Every 500,000 episodes, the current greedy policy ($\epsilon = 0$) was evaluated using 100,000 simulations.
- **Hyperparameters:** $\gamma = 1.0$, $\epsilon = 0.01$.

CliffWalking:

- **Episodes:** 200,000 per run, across 5 runs.
- **Evaluation:** The greedy policy was evaluated every 1,000 episodes from the initial state.
- **Hyperparameters:** $\gamma = 1.0$, $\epsilon = 0.1$.

Results from these MC control algorithms are presented in Figures 5 to 8

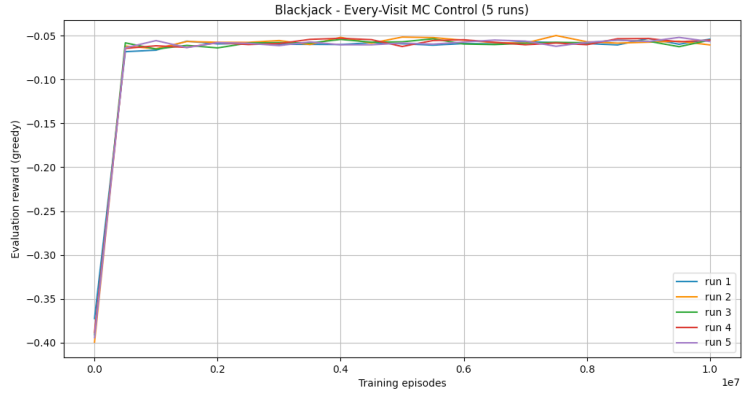

Figure 5: Average Return in Blackjack ($\epsilon = 0.01$)

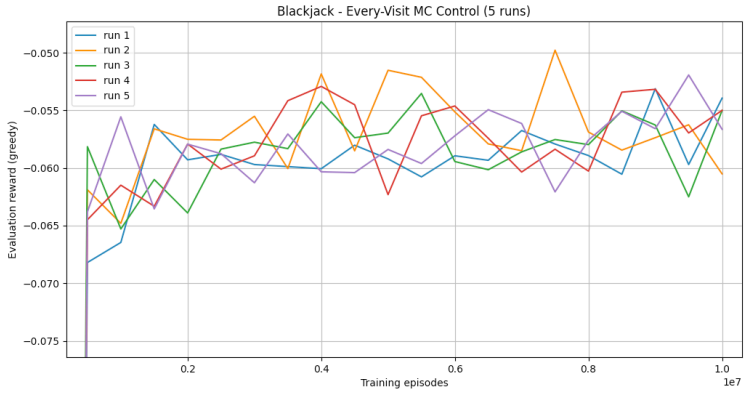

Figure 6: Average Return in Blackjack environment (zoomed, and $\epsilon = 0.01$)

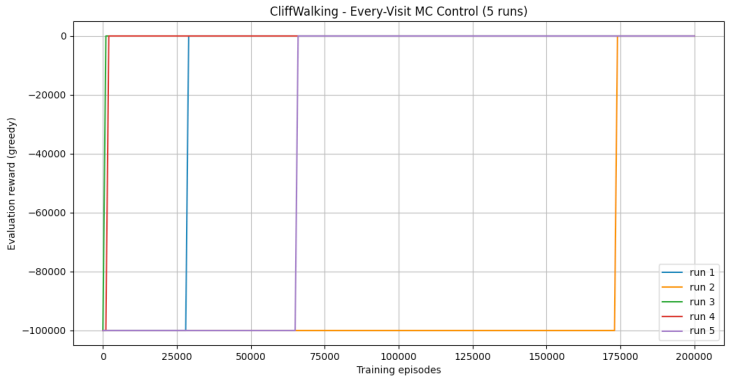

Figure 7: Average Return in Cliff environment ($\epsilon = 0.1$)

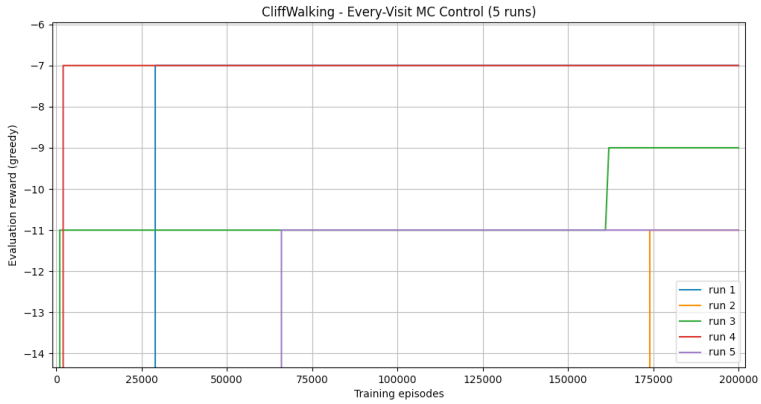

Figure 8: Average Return in Cliff environment (zoomed, and $\epsilon = 0.1$)

k) Blackjack

According to the Monte Carlo algorithm applied to the infinite-deck Blackjack environment, one should stop drawing cards when the learned greedy policy indicates that the optimal action is to "stand." This decision varies depending on the value of the dealer's visible card, the player's current sum, and whether one holds a usable Ace. In general, the algorithm learns to stand when the player's sum is high enough that continuing to draw entails a significant risk of busting. Regarding consistency across the five runs, while there are small variations due to the stochastic nature of learning and differences in the samples collected in each run, the algorithm tends to arrive at similar policies that reflect consistent patterns. This is because with a sufficient number of episodes (in this case, 10 million), the estimates of the action values stabilize and converge toward an optimal policy.

The optimal policies for Blackjack based on the Monte Carlo simulations are shown in Table 7, when having an usable ace, and Table 8 with no usable aces. The table indicates the strategy of whether to *hit*, or *stand*, denoted h and s respectively, based on the sum of the player's hand, and the value of the dealer's visible card.

Player Sum \ Dealer Card	1	2	3	4	5	6	7	8	9	10
12	h	h	h	h	h	h	h	h	h	h
13	h	h	h	h	h	h	h	h	h	h
14	h	h	h	h	h	h	h	h	h	h
15	h	h	h	h	h	h	h	h	h	h
16	h	h	h	h	h	h	h	h	h	h
17	h	h	h	h	h	h	h	h	h	h
18	h	s	s	s	s	s	s	s	h	s
19	s	s	s	s	s	s	s	s	s	s
20	s	s	s	s	s	s	s	s	s	s
21	s	s	s	s	s	s	s	s	s	s

Table 7: Optimal policy with usable aces (Usable Ace = `True`)

Player Sum \ Dealer Card	1	2	3	4	5	6	7	8	9	10
12	h	h	h	h	h	h	h	h	h	h
13	h	h	h	h	h	h	h	h	h	h
14	h	h	s	s	s	s	h	h	h	h
15	h	s	s	s	s	s	h	h	h	h
16	h	s	s	s	s	s	h	h	h	h
17	s	s	s	s	s	s	s	s	s	s
18	s	s	s	s	s	s	s	s	s	s
19	s	s	s	s	s	s	s	s	s	s
20	s	s	s	s	s	s	s	s	s	s
21	s	s	s	s	s	s	s	s	s	s

Table 8: Optimal policy without usable aces (Usable Ace = `False`)

l) Best course of action on Cliff

According to the results obtained with Monte Carlo in the CliffWalking setting, the best course of action consists of going up, then moving directly to the right and then descending at the last column, thus avoiding the cliff and obtaining the minimum penalty of -7. This optimal behavior is observed in runs one and four. However, in runs two, three, and five, the agent takes longer and less risky routes, resulting in more negative rewards such as -9 or -11. Although there is a well-defined optimal policy, the outcome is not completely consistent across the five runs. This occurs for several reasons. First, extreme rewards, such as the -100 penalty for falling off the cliff, cause the algorithm to strongly prioritize avoiding that area, even if it means deviating from the shortest route. Second, ϵ-greedy exploration, with ϵ=0.1, introduces random actions during training that can temporarily lead the agent along suboptimal trajectories. And third, since this is a Monte Carlo algorithm, the values are updated only at the end of each episode, which slows down error correction, especially in infrequently repeated trajectories. Together, these factors explain why some runs manage to find optimal policies while others end up with more conservative or less efficient behavior.

Run	`Action Sequence`	Final Reward
1	↑, →, →, →, →, →, ↓	-7
2	↑, →, ↑, →, →, ↑, →, ↓, →, ↓, ↓	-11
3	↑, ↑, →, →, →, →, →, ↓, ↓	-9
4	↑, →, →, →, →, →, ↓	-7
5	↑, ↑, →, ↑, →, →, →, ↓, →, ↓, ↓	-11

Table 9: Action sequences and final rewards for each run (↑: up, ↓: down, →: right, ←: left)

m) Monte Carlos stability

Considering the results obtained in Blackjack and CliffWalking, it can be concluded that Monte Carlo is not completely stable, although it can converge to reasonable long-term policies. In the case of Blackjack, the five runs of 10 million episodes show a progressive and fairly consistent convergence, although with small variations between runs. However, in CliffWalking, the behavior is more variable, and while some learn the optimal route, others follow longer paths. In other words, Monte Carlo can be efficient, but its stability depends strongly on the environment and the number of episodes. Monte Carlo does not always guarantee an optimal policy over a finite number of runs.

n) Cliff problem with size 12

When running Monte Carlo in the Cliff setting with a longer grid, say, size 12, the problem becomes considerably more difficult to solve. As the cliff length increases, so does the number of states and, therefore, the search space. This means the agent must explore many more trajectories to find a safe path that maximizes the reward, which is especially challenging because a single incorrect action can lead to falling off the cliff and receiving a penalty of -100. Furthermore, the setting is constructed such that if the agent does not reach the goal state (G) after 100,000 steps, the episode automatically terminates. This restriction severely limits the number of available steps, making exploration and learning optimal paths even more difficult, especially on large grids. Since Monte Carlo only updates values at the end of the episode, that is, once the goal state is reached (or the step limit is exhausted), learning becomes even slower in these types of domains. Coupled with the random exploration inherent in ϵ-greedy policies, this can cause the agent to continue making suboptimal decisions for a large number of episodes. In short, Monte Carlo can eventually solve the problem, but it requires many more episodes, probably more than 200,000 (the number of episodes in the previous experiment), and its efficiency decreases significantly on longer grids.

References

[1] Richard S. Sutton and Andrew Barto. *Reinforcement learning: An Introduction*. eng. Second edition. Adaptive computation and machine learning. Cambridge, Massachusetts London, England: The MIT Press, 2020. ISBN: 978-0-262-03924-6.