



UNIVERSITAT POLITÈCNICA DE CATALUNYA  
BARCELONATECH

Facultat d'Informàtica de Barcelona



Sistemas Inteligentes Distribuidos

# Agente de aprendizaje por refuerzo para el entorno Cliff Walking

Lluc Martínez Busquets

Eric Medina León

Àlex Rodríguez Rodríguez

2024/2025 Q2

## Resumen

Este trabajo presenta un estudio detallado sobre la implementación de cuatro algoritmos de aprendizaje por refuerzo en el entorno *CliffWalking-v0* de la librería de Python *Gymnasium*: *Value Iteration*, *Direct Estimation*, *Q-Learning* y *REINFORCE*. El entorno se configura con el modo *is\_slippery* activado, lo cual introduce estocasticidad en las transiciones de estado. El objetivo es evaluar el rendimiento de cada algoritmo, así como qué parámetros e hiperparámetros son los óptimos para su funcionamiento.

# Índice

<b>1</b>	<b>Introducción</b>	<b>4</b>
1.1	Caracterización del entorno	4
1.1.1	Espacio de acciones	5
1.1.2	Estados inicial y terminal	5
1.1.3	Función de recompensa	5
1.2	Entorno experimental	6
<b>2</b>	<b>Value Iteration</b>	<b>7</b>
2.1	Descripción del algoritmo	7
2.2	Experimentación	9
2.2.1	Experimento factor de descuento & umbral de convergencia	9
2.2.1.1	Diseño experimental	9
2.2.1.2	Resultados	9
<b>3</b>	<b>Direct Estimation</b>	<b>10</b>
3.1	Descripción del algoritmo	10
3.2	Experimentación	11
3.2.1	Experimento factor de descuento & número de trayectorias	12
3.2.1.1	Diseño experimental	12
3.2.1.2	Resultados	12
3.2.2	Experimento número de episodios de entrenamiento	12
3.2.2.1	Diseño experimental	12
3.2.2.2	Resultados	13
3.2.3	Experimento Patience	13
3.2.3.1	Diseño experimental	13
3.2.3.2	Resultados	14
<b>4</b>	<b>Q-learning</b>	<b>15</b>
4.1	Descripción del algoritmo	15
4.2	Experimentación	17
4.2.1	Experimento factor de descuento & tasa de aprendizaje	17
4.2.1.1	Diseño experimental	17
4.2.1.2	Resultados	18
4.2.2	Experimento tasa de exploración ( $\epsilon$ ) & decaimiento de la tasa de exploración ( $\epsilon$ )	22
4.2.2.1	Diseño experimental	22
4.2.2.2	Resultados	23
4.2.3	Experimento tasa de aprendizaje & decaimiento de la tasa de aprendizaje	26
4.2.3.1	Diseño experimental	26
4.2.3.2	Resultados	27
4.2.4	Experimento número de episodios	27
4.2.4.1	Diseño experimental	27
4.2.4.2	Resultados	28
4.2.5	Experimento penalización de la acción izquierda	28

---

4.2.5.1	Diseño experimental . . . . .	28
4.2.5.2	Resultados . . . . .	29
<b>5</b>	<b>Reinforce . . . . .</b>	<b>30</b>
5.1	Descripción del algoritmo . . . . .	30
5.2	Experimentación . . . . .	30
<b>6</b>	<b>Conclusiones . . . . .</b>	<b>31</b>
<b>7</b>	<b>Bibliografía . . . . .</b>	<b>32</b>
<b>8</b>	<b>Apéndices . . . . .</b>	<b>33</b>

---

# 1. Introducción

El aprendizaje por refuerzo es un paradigma de aprendizaje automático en el que un agente aprende a tomar decisiones interactuando con un entorno y recibiendo recompensas o penalizaciones.

Este trabajo se centra en la implementación y evaluación experimental de cuatro algoritmos de aprendizaje por refuerzo en el entorno *CliffWalking-v0*, de la librería de Python *Gymnasium*, con el objetivo de comparar su rendimiento y eficiencia en dicho entorno. Los algoritmos implementados son: *Value Iteration*, *Direct Estimation*, *Q-Learning* y *REINFORCE*.

## 1.1. Caracterización del entorno

El entorno de *Cliff Walking*, propuesto originalmente por *Sutton & Barto*, es un entorno clásico para evaluar algoritmos de aprendizaje por refuerzo. En este entorno, el agente debe navegar por una cuadrícula de dimensiones 4x12 evitando caer en un acantilado, lo que representa una penalización significativa. El objetivo del agente es llegar a la esquina inferior derecha de la cuadrícula de la forma más eficiente posible, maximizando la recompensa acumulada a lo largo del tiempo y minimizando el número de pasos necesarios para alcanzar la meta.

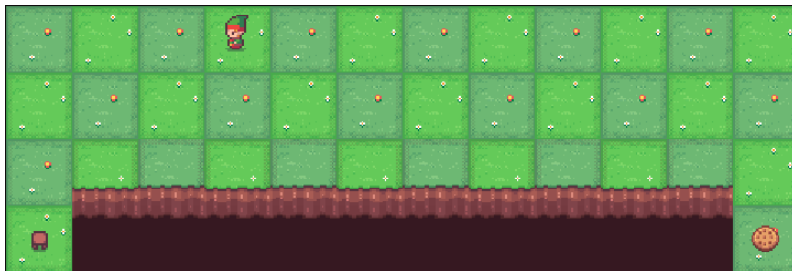


Figura 1: Captura de pantalla del entorno *Cliff Walking*. El agente comienza en la esquina inferior izquierda (s0) y debe llegar a la esquina inferior derecha (sT), evitando caer en el acantilado (cliff) representado por las celdas marrones.

A continuación se caracteriza de forma detallada el entorno:

- **Observabilidad:** Totalmente observable. El agente recibe en cada paso su posición exacta en la cuadrícula, sin ruido ni información oculta, por lo que tiene acceso total al estado relevante.
- **Número de agentes:** Un único agente.
- **Determinismo:** Estocástico, ya que está activado el modo `is_slippery=True`. En este caso, por cada acción que el agente toma, hay una probabilidad de aproximadamente el 66.7% de que el agente se resvale hacia una dirección perpendicular a la acción deseada.
- **Atomicidad:** Secuencial. Las decisiones del agente tienen consecuencias que dependen de toda la historia de acciones y percepciones, y los efectos futuros de las acciones importan para maximizar la recompensa acumulada.

- 
- **Dinamicidad:** Estático. El estado del entorno sólo cambia cuando el agente toma una acción; no hay cambios “por sí mismos” mientras el agente razona.
  - **Continuidad:** Discreto. Tanto el espacio de estados (posiciones en la cuadrícula) como el de acciones (arriba, abajo, izquierda, derecha) y el tiempo de decisión son discretos.
  - **Conocimiento:** Conocido. Las reglas de transición (aunque estocásticas) y la función de recompensa están definidas de antemano y son accesibles al agente.

#### 1.1.1. Espacio de acciones

El agente dispone de un conjunto finito de acciones

$$\mathcal{A} = \{\text{Arriba, Derecha, Abajo, Izquierda}\},$$

cada una de las cuales intenta desplazar al agente una celda en la dirección indicada.

#### 1.1.2. Estados inicial y terminal

- **Estado inicial**  $s_0 = (3, 0)$ , correspondiente a la esquina inferior izquierda de la cuadrícula.
- **Estado terminal**  $s_T = (3, 11)$ , la meta en la esquina inferior derecha; al llegar aquí, el episodio termina.

Si durante el episodio el agente cae en el acantilado, este regresa al estado inicial  $s_0$  y continua con el episodio.

#### 1.1.3. Función de recompensa

La señal de recompensa  $R(s, a, s')$  se define como:

$$R(s, a, s') = \begin{cases} -100, & \text{si } s' \text{ es una celda de acantilado (cliff),} \\ -1, & \text{en cada transición válida que no alcance la meta ni el cliff,} \\ 0, & \text{al alcanzar el estado terminal } s_T. \end{cases}$$

De este modo, el agente está incentivado a llegar cuanto antes a la meta evitando caer en el precipicio.

---

## 1.2. Entorno experimental

Componente	Descripción
Sistema operativo	Ubuntu 24.04.2 LTS
Kernel Linux	6.10.3-061003-generic
CPU	AMD Ryzen 7 5825U (8 núcleos, 16 hilos, hasta 4.5 GHz)
GPU integrada	AMD Radeon Graphics (Barcelo)
Memoria RAM	16 GB DDR4
Intérprete de Python	Python 3.9.18

Cuadro 1: Entorno de hardware y software utilizado en los experimentos

---

## 2. Value Iteration

### 2.1. Descripción del algoritmo

La iteración por valor es un método de programación dinámica para resolver un Proceso de Decisión de Markov (MDP) y encontrar simultáneamente la función valor óptima  $V^*$  y la política óptima  $\pi^*$ . Se basa en la relación de Bellman óptima:

$$V^*(s) = \max_{a \in A} \sum_{s'} P(s' | s, a) [R(s, a, s') + \gamma V^*(s')],$$

donde:

- $S$  es el conjunto de estados.
- $A$  es el conjunto de acciones.
- $P(s' | s, a)$  es la probabilidad de transición de  $s$  a  $s'$  dado  $a$ .
- $R(s, a, s')$  es la recompensa recibida al transitar.
- $\gamma \in [0, 1)$  es el factor de descuento.

A continuación se presenta el pseudocódigo genérico de Value Iteration que se ha implementado en este proyecto, seguido de las decisiones de diseño adoptadas en la implementación de Python.

---

**Algorithm 1** Value Iteration

---

**Require:** Conjunto de estados  $S$ , conjunto de acciones  $A$ ,  $P(s' | s, a)$  y  $R(s, a, s')$ , factor de descuento  $\gamma \in [0, 1)$ , umbral de convergencia  $\varepsilon > 0$

**Ensure:** Función valor  $V$  y política óptima  $\pi$

```
1: Inicializar  $V(s) \leftarrow 0, \forall s \in S$ 
2: repeat
3:    $\Delta \leftarrow 0$ 
4:   for all  $s \in S$  do
5:      $V_{\text{old}} \leftarrow V(s)$ 
6:      $V(s) \leftarrow \max_{a \in A} \sum_{s'} P(s' | s, a) [R(s, a, s') + \gamma V(s')]$ 
7:      $\Delta \leftarrow \max(\Delta, |V(s) - V_{\text{old}}|)$ 
8:   end for
9: until  $\Delta \leq \varepsilon$ 
10: for all  $s \in S$  do
11:    $\pi(s) \leftarrow \arg \max_{a \in A} \sum_{s'} P(s' | s, a) [R(s, a, s') + \gamma V(s')]$ 
12: end for
13: return  $V, \pi$ 
```

---



---

## Decisiones de diseño en la implementación Python

- **Cálculo de  $Q(s, a)$  con manejo del estado terminal:** para calcular el valor de cada acción consideramos que si se ha llegado a un estado terminal, el término de arranque posterior (*bootstrap*) se anula:

$$Q(s, a) = \sum_{s'} p [r + \gamma V(s')] \quad \longrightarrow \quad \text{bootstrap} = 0 \text{ si } s' \text{ es terminal.}$$

De esta forma, se garantiza que al terminar el episodio, no se incorporen erróneamente estimaciones de valor posteriores a la terminación.

- **Evaluación periódica de la política:** tras cada iteración de valor calculamos la recompensa media de la política actual en  $N = 100$  episodios de longitud máxima  $T_{\text{máx}} = 200$  (método `check_improvements`), tanto para monitorizar progresos como para registrar la mejor recompensa y la iteración en que ocurre. Se fijan los valores de  $N$  y  $T_{\text{máx}}$  para evitar que el algoritmo se detenga por un número excesivo de episodios, lo que podría ocurrir si la política converge a una política subóptima. En este caso, el algoritmo se detendría sin haber explorado adecuadamente el espacio de estados.

---

## 2.2. Experimentación

En el caso de iteración de valor, se han decidido estudiar el efecto de diferentes valores del factor de descuento  $\gamma$  y el parámetro de convergencia  $\epsilon$  en el rendimiento del algoritmo.

### 2.2.1. Experimento factor de descuento & umbral de convergencia

#### 2.2.1.1 Diseño experimental

El objetivo de este experimento es analizar cómo los parámetros factor de descuento ( $\gamma$ ) y umbral de convergencia ( $\epsilon$ ) afectan el rendimiento del algoritmo de iteración de valor.

<b>Observación</b>	El rendimiento y optimalidad de la política encontrada por <i>Value Iteration</i> se ven afectados por los valores de $\gamma$ y $\epsilon$ .
<b>Planteamiento</b>	Para cada pareja de valores de $\gamma$ y $\epsilon$ , se compara la tasa de acierto (llegar al estado final), la recompensa media, número de pasos y tiempo de entrenamiento del algoritmo.
<b>Hipótesis</b>	Se espera que un mayor valor de $\gamma$ conduzca a una política más óptima, mientras que un menor valor de $\epsilon$ permita una convergencia más rápida con una menor precisión.
<b>Método</b>	<ul style="list-style-type: none"><li>■ Se elige un conjunto de valores para <math>\gamma</math> y <math>\epsilon</math>: <math>\gamma \in \{0.5, 0.7, 0.9, 0.95, 0.99\}</math> y <math>\epsilon \in \{1 \times 10^{-1}, 1 \times 10^{-2}, 1 \times 10^{-4}, 1 \times 10^{-8}\}</math>.</li><li>■ Para cada combinación de <math>\gamma</math> y <math>\epsilon</math>, se ejecuta el algoritmo <i>Value Iteration</i> en el entorno.</li><li>■ Se evalúa la política obtenida probándola con 500 episodios.</li></ul>

Cuadro 2: Value Iteration - Experimento 1 - Factor de descuento & umbral de convergencia

#### 2.2.1.2 Resultados

---

## 3. Direct Estimation

### 3.1. Descripción del algoritmo

La versión de Estimación Directa que se ha implementado corresponde a un *Método Monte Carlo basado en modelo*, en el cual:

1. Se recolectan muestras de transición  $(s, a, s', r)$  jugando acciones aleatorias.
2. Se estiman empíricamente

$$\begin{aligned}\hat{T}(s, a, s') &= \frac{N(s, a, s')}{\sum_u N(s, a, u)}, \\ \hat{R}(s, a, s') &= \frac{\sum_{i=1}^{N(s, a, s')} r_i(s, a, s')}{N(s, a, s')}.\end{aligned}$$

donde  $N(s, a, s')$  representa el número de veces que se ha observado la transición del estado  $s$  al estado  $s'$  tomando la acción  $a$ , y  $r_i(s, a, s')$  es la recompensa obtenida en la  $i$ -ésima transición de  $s$  a  $s'$  mediante la acción  $a$ .

3. Se aplica iteración de valor sobre el MDP estimado  $(\hat{T}, \hat{R}, \gamma)$  para obtener

$$V^*(s) = \max_a \sum_{s'} \hat{T}(s, a, s') \left[ \hat{R}(s, a, s') + \gamma V^*(s') \right],$$

y de ahí la política óptima

$$\pi^*(s) = \arg \max_a \sum_{s'} \hat{T}(s, a, s') \left[ \hat{R}(s, a, s') + \gamma V^*(s') \right].$$

A continuación se presenta el pseudocódigo genérico de Direct Estimation que se ha implementado en este proyecto, seguido de las decisiones de diseño adoptadas en la implementación de Python.

---

**Algorithm 2** Estimación Directa (Model-based Monte Carlo)

---

**Require:** Factor de descuento  $\gamma$ , número de trayectorias  $N$ , tolerancia  $\varepsilon$ , máximo de iteraciones  $K$

```
1: Inicializar contadores de transición y recompensa vacíos
2: Inicializar  $V(s) \leftarrow 0$  para todo estado  $s$ 
3: for  $t = 1, \dots, K$  do
4:   Recolectar datos:
5:   for  $i = 1, \dots, N$  do
6:     Jugar un paso aleatorio, obtener  $(s, a, s', r)$ 
7:     Incrementar  $N(s, a, s')$  y acumular recompensa  $r$  para el par  $(s, a, s')$ 
8:   end for
9:   Ajustar modelo:
10:  for all  $(s, a)$  do
11:     $\hat{T}(s, a, s') \leftarrow \frac{N(s, a, s')}{\sum_u N(s, a, u)}$ 
12:     $\hat{R}(s, a, s') \leftarrow \frac{\sum_{i=1}^{N(s, a, s')} r_i(s, a, s')}{N(s, a, s')}$ 
13:  end for
14:  Iteración de valor:
15:   $\Delta \leftarrow 0$ 
16:  for all  $s$  do
17:    for all  $a$  do
18:       $Q(s, a) \leftarrow \sum_{s'} \hat{T}(s, a, s') [\hat{R}(s, a, s') + \gamma V(s')]$ 
19:    end for
20:     $V_{\text{nuevo}}(s) \leftarrow \max_a Q(s, a)$ 
21:     $\Delta \leftarrow \max\{\Delta, |V_{\text{nuevo}}(s) - V(s)|\}$ 
22:     $V(s) \leftarrow V_{\text{nuevo}}(s)$ 
23:  end for
24:  if  $\Delta < \varepsilon$  then break
25:  end if
26: end for
27: return  $V$ , y derivar  $\pi^*(s) = \arg \max_a Q(s, a)$ 
```

---

## Decisiones de diseño en la implementación Python

- **Criterio de parada por paciencia.** Además de la tolerancia en la iteración de valor, detenemos el entrenamiento si no hay mejora en la recompensa media durante `PATIENCE` iteraciones, midiendo esto con la función `check_improvements()`.

## 3.2. Experimentación

En el caso de estimación directa se analizan diferentes parámetros como el factor de descuento, el número de trayectorias, episodios de entrenamiento y el número máximo de iteraciones sin mejora para detener el entrenamiento (*patience*).

---

### 3.2.1. Experimento factor de descuento & número de trayectorias

#### 3.2.1.1 Diseño experimental

El objetivo de este experimento es analizar el rendimiento del algoritmo *Direct Estimation* en función del factor de descuento y el número de trayectorias.

<b>Observación</b>	El rendimiento del algoritmo varía con el factor de descuento y el número de trayectorias.
<b>Planteamiento</b>	Para cada combinación de $\gamma$ y <i>número de trayectorias</i> , se compara la tasa de acierto (llegar al estado final), la recompensa media, número de pasos y tiempo de entrenamiento del algoritmo.
<b>Hipótesis</b>	Un mayor factor de descuento y un mayor número de trayectorias mejorarán el rendimiento del algoritmo.
<b>Método</b>	<ul style="list-style-type: none"><li>■ Se fijan 500 episodios de entrenamiento y un número máximo de iteraciones sin mejora para detener el entrenamiento (<i>patience</i>) de 100.</li><li>■ Se eligen los siguientes valores para <math>\gamma</math> y <i>número de trayectorias</i>: <math>\gamma \in \{0.5, 0.7, 0.9, 0.95, 0.99\}</math> y <i>número de trayectorias</i> <math>\in \{10, 100, 500, 1000\}</math>.</li><li>■ Para cada combinación de <math>\gamma</math> y <i>número de trayectorias</i>, se ejecuta el algoritmo <i>Direct Estimation</i> en el entorno.</li><li>■ Se evalúa la política obtenida probándola con 500 episodios.</li><li>■ Se repite el proceso para cada combinación de <math>\gamma</math> y <i>número de trayectorias</i> 10 veces.</li></ul>

Cuadro 3: Direct Estimation - Experimento 1 - Factor de descuento & número de trayectorias

#### 3.2.1.2 Resultados

### 3.2.2. Experimento número de episodios de entrenamiento

#### 3.2.2.1 Diseño experimental

El objetivo de este experimento es analizar el rendimiento del algoritmo *Direct Estimation* en función del número de episodios de entrenamiento.

<b>Observación</b>	El rendimiento del algoritmo varía con el número de episodios de entrenamiento.
<b>Planteamiento</b>	Se compara la tasa de acierto (llegar al estado final), la recompensa media, número de pasos y tiempo de entrenamiento del algoritmo para diferentes números de episodios de entrenamiento.
<b>Hipótesis</b>	Un mayor número de episodios de entrenamiento mejorará el rendimiento del algoritmo.
<b>Método</b>	<ul style="list-style-type: none"> <li>■ Se fijan los mejores valores para <math>\gamma</math> y <i>número de trayectorias</i> del experimento anterior.</li> <li>■ Se eligen los siguientes valores para <i>número de episodios de entrenamiento</i>: <i>número de episodios de entrenamiento</i> <math>\in \{100, 500, 1000, 5000\}</math>.</li> <li>■ Para cada <i>número de episodios</i>, se ejecuta el algoritmo <i>Direct Estimation</i> en el entorno.</li> <li>■ Se evalúa la política obtenida probándola con 500 episodios.</li> <li>■ Se repite el proceso para cada <i>número de episodios</i> 10 veces.</li> </ul>

Cuadro 4: Direct Estimation - Experimento 2 - Número de episodios de entrenamiento

### 3.2.2.2 Resultados

### 3.2.3. Experimento Patience

#### 3.2.3.1 Diseño experimental

El objetivo de este experimento es ver si el algoritmo *Direct Estimation* puede detenerse antes de que se alcance el número máximo de episodios de entrenamiento, lo que podría reducir el tiempo de entrenamiento sin afectar el rendimiento.

<b>Observación</b>	El algoritmo <i>Direct Estimation</i> tiene fases largas de entrenamiento sin mejora.
<b>Planteamiento</b>	Se compara la tasa de acierto (llegar al estado final), la recompensa media, número de pasos y tiempo de entrenamiento del algoritmo para diferentes valores de iteraciones sin mejora para detener el entrenamiento ( <i>patience</i> ).
<b>Hipótesis</b>	Un valor de <i>patience</i> más bajo permitirá al algoritmo detenerse antes y reducir el tiempo de entrenamiento sin afectar significativamente el rendimiento.
<b>Método</b>	<ul style="list-style-type: none"> <li>■ Se fijan los mejores valores para <math>\gamma</math>, <i>número de trayectorias</i> y <i>número de episodios de entrenamiento</i> de los experimentos anteriores.</li> <li>■ Se eligen los siguientes valores para <i>patience</i>: <math>patience \in \{10, 100, 1000\}</math>.</li> <li>■ Para cada <i>patience</i>, se ejecuta el algoritmo <i>Direct Estimation</i> en el entorno.</li> <li>■ Se evalúa la política obtenida probándola con 500 episodios.</li> <li>■ Se repite el proceso para cada <i>patience</i> 10 veces.</li> </ul>

Cuadro 5: Direct Estimation - Experimento 3 - Patience

### 3.2.3.2 Resultados

---

## 4. Q-learning

### 4.1. Descripción del algoritmo

Q-learning es un algoritmo de aprendizaje por refuerzo. La característica fundamental de Q-learning es su capacidad para aprender de forma off-policy, es decir, puede aprender la política óptima mientras sigue una política de exploración diferente (como  $\varepsilon$ -greedy). El algoritmo actualiza iterativamente sus estimaciones  $Q(s, a)$  utilizando la ecuación de Bellman. A medida que el aprendizaje progresa, las estimaciones de  $Q$  convergen hacia los valores óptimos, permitiendo derivar la política óptima como  $\pi^*(s) = \arg \max_a Q(s, a)$ .

A continuación se presenta el pseudocódigo genérico de Direct Estimation que se ha implementado en este proyecto, seguido de las decisiones de diseño adoptadas en la implementación de Python.

---

**Algorithm 3** Q-Learning

---

```
1: Inicializar  $Q(s, a) \leftarrow 0$  para todo  $s \in S, a \in A$ 
2: for episodio  $\leftarrow 1$  to  $N_{\text{episodios}}$  do
3:    $\varepsilon \leftarrow \max(\varepsilon_{\text{mín}}, \varepsilon_0 \cdot \text{decay}^{\text{episodio}})$ 
4:   Inicializar  $s \leftarrow s_0$ 
5:   for  $t \leftarrow 1$  to  $T_{\text{máx}}$  do
6:     if  $\text{rand}() \leq \varepsilon$  then
7:        $a \leftarrow$  acción aleatoria
8:     else
9:        $a \leftarrow \arg \max_{a'} Q(s, a')$ 
10:    end if
11:    Ejecutar  $a$ , observar  $r, s'$ 
12:     $\text{td\_target} \leftarrow r + \gamma \max_{a''} Q(s', a'')$ 
13:     $\text{td\_error} \leftarrow \text{td\_target} - Q(s, a)$ 
14:     $Q(s, a) \leftarrow Q(s, a) + \alpha \text{td\_error}$ 
15:    if  $s'$  es terminal then
16:      break
17:    end if
18:     $s \leftarrow s'$ 
19:  end for
20: end for
```

---

### Decisiones de diseño en la implementación Python

■ Política  $\varepsilon$ -greedy con decaimiento:

$$\pi(a \mid s) = \begin{cases} \frac{1}{|A|}, & \text{con probabilidad } \varepsilon, \\ 1, & \text{si } a = \arg \max_{a'} Q(s, a'), \\ 0, & \text{en otro caso.} \end{cases}$$



---

Al inicio de cada episodio:

$$\varepsilon \leftarrow \text{máx}(\varepsilon_{\text{mín}}, \varepsilon \cdot (\text{decay})^{\text{episodio}}).$$

- **Wrapper de recompensas customizado:** Se penaliza la acción Izquierda añadiendo una recompensa peor que la original, ya que en ningún caso interesa que el agente se desplace hacia la izquierda.

---

## 4.2. Experimentación

En esta sección se presentan los experimentos realizados para evaluar el rendimiento del algoritmo Q-Learning en el entorno. Se analiza cómo diferentes parámetros del algoritmo afectan su capacidad para encontrar políticas óptimas, su convergencia y su eficiencia.

### 4.2.1. Experimento factor de descuento & tasa de aprendizaje

#### 4.2.1.1 Diseño experimental

El objetivo de este experimento es analizar cómo el factor de descuento y la tasa de aprendizaje afectan el rendimiento del algoritmo Q-Learning.

<b>Observación</b>	El factor de descuento ( $\gamma$ ) y la tasa de aprendizaje ( $\alpha$ ) de exploración son parámetros críticos en el algoritmo Q-Learning.
<b>Planteamiento</b>	Para cada pareja de valores de $\gamma$ y $\alpha$ , se compara la tasa de acierto (llegar al estado final), la recompensa media, número de pasos y tiempo de entrenamiento del algoritmo.
<b>Hipótesis</b>	Un mayor factor de descuento y una tasa de aprendizaje más lenta mejorarán el rendimiento del algoritmo.
<b>Método</b>	<ul style="list-style-type: none"><li>■ Se fijan 1000 episodios de entrenamiento, <math>\epsilon</math> inicial de 0.9, coeficiente de decaimiento de <math>\epsilon</math> de 0.95, un coeficiente de decaimiento de la tasa de aprendizaje de 0.99 y una penalización de la acción “moverse a la izquierda” de <math>-1</math>.</li><li>■ Se eligen los siguientes valores para <math>\gamma</math> y <math>\alpha</math>: <math>\gamma \in \{0.5, 0.7, 0.9, 0.95, 0.99\}</math> y <math>\alpha \in \{0.1, 0.2, 0.5, 0.8\}</math>.</li><li>■ Para cada combinación de <math>\gamma</math> y <math>\alpha</math>, se ejecuta el algoritmo Q-Learning en el entorno.</li><li>■ Se evalúa la política obtenida probándola con 500 episodios.</li><li>■ Se repite el proceso para cada combinación de <math>\gamma</math> y <math>\alpha</math> 20 veces para obtener una muestra representativa (debido a la estocasticidad del entorno).</li></ul>

Cuadro 6: Q-Learning - Experimento 1 - Factor de descuento & tasa de aprendizaje

#### 4.2.1.2 Resultados

Los resultados del experimento se han representado mediante heatmaps para facilitar la visualización de los datos. En cada gráfico, el eje X representa el factor de descuento ( $\gamma$ ) y el eje Y representa la tasa de aprendizaje ( $\alpha$ ). Los colores indican el valor de la métrica correspondiente. Cada valor representa la media de las 20 ejecuciones del algoritmo (cada ejecución está representada por la media de 500 episodios) de la correspondiente combinación de parámetros.

##### Tasa de Éxito

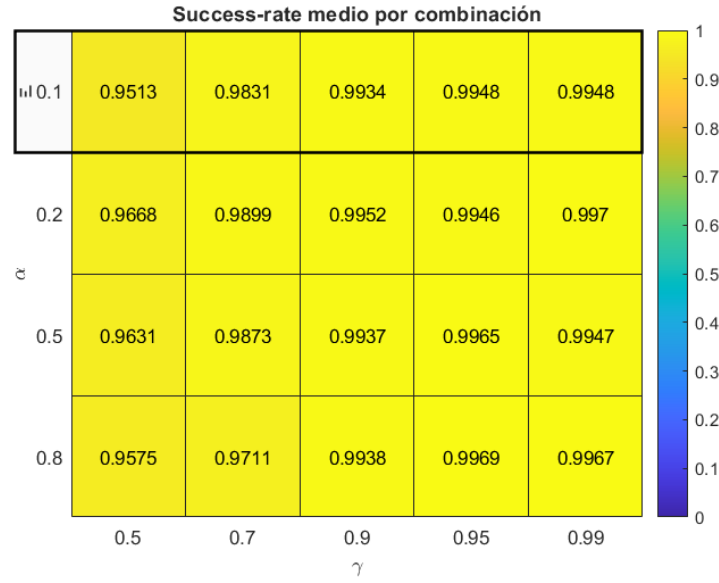


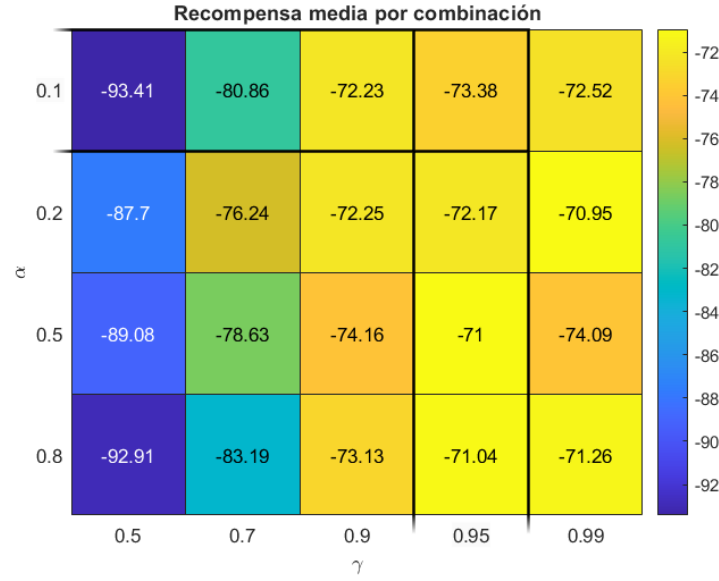
Figura 2: Tasa de éxito para diferentes combinaciones de  $\alpha$  y  $\gamma$

En general, la tasa de éxito en este caso es bastante alta para todas las combinaciones de valores. Sin embargo, puede observarse una pequeña tendencia. Conforme aumenta el factor de descuento ( $\gamma$ ), el porcentaje de éxito se aproxima a 1.

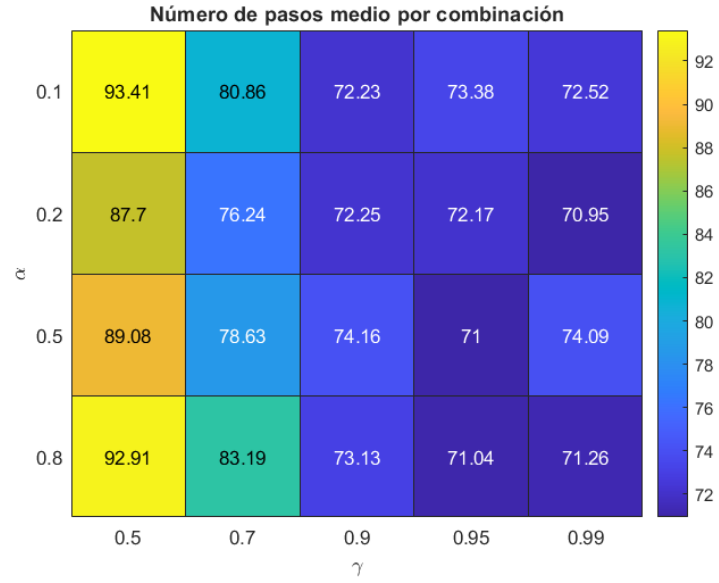
- Con  $\gamma = 0.5$ , la tasa de éxito se mantiene alrededor del 95-96 %
- Para  $\gamma \geq 0.9$ , la tasa de éxito supera consistentemente el 99 %

La mejor combinación se alcanza con  $\gamma = 0.99$  y  $\alpha = 0.2$ , logrando una tasa de éxito del 99.7 %

## Recompensa media y número de pasos medios



(a) Recompensa media para diferentes combinaciones de  $\alpha$  y  $\gamma$



(b) Número de pasos para diferentes combinaciones de  $\alpha$  y  $\gamma$

Figura 3: Análisis de la recompensa media y número de pasos

El análisis de la recompensa media y el número medio de pasos revela que:

- Las recompensas mejoran significativamente (son menos negativas) al aumentar  $\gamma$  y, por lo tanto, el número de pasos también se reduce.
- La tasa de aprendizaje no tiene mucha influencia en esta combinación de parámetros.
- Con  $\gamma = 0.5$ , las recompensas medias oscilan entre -93 y -87, representando trayectorias más largas.

- Las mejores recompensas se obtienen con  $\gamma \geq 0.9$ , alcanzando valores cercanos a -70.

La combinación óptima es  $\gamma = 0.99$ ,  $\alpha = 0.2$  y logra una recompensa media de -70.95. Corresponde con la combinación que obtiene mejor tasa de éxito.

Se puede observar que el número medio de pasos corresponde exactamente con la recompensa media en cada configuración. Esto nos indica que el agente no cae por el barranco en ningún momento, ya que sino la recompensa media sería más negativa (-100 de recompensa por caer en 1 paso, perdería la correspondencia de -1 de recompensa por cada paso).

### Tiempo de entrenamiento

Para el tiempo de entrenamiento, se han elegido las tres combinaciones que mejor rendimiento han dado.

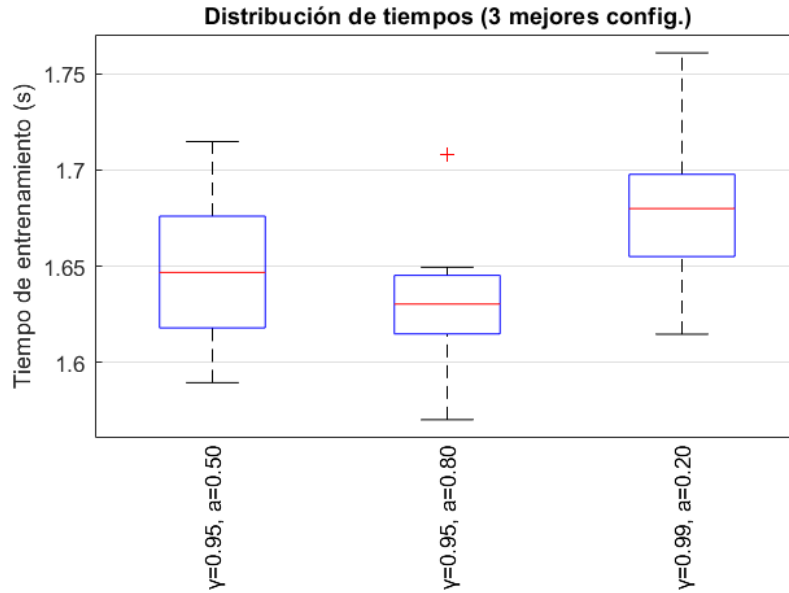


Figura 4: Tiempo de entrenamiento para diferentes combinaciones de  $\alpha$  y  $\gamma$

El análisis del tiempo de ejecución muestra que para las tres combinaciones, el tiempo se mueve entorno a los 1.65 segundos. Hay poca diferencia entre ellos, con una pequeña variabilidad en las colas.

---

## Tabla resumen

$\gamma$	$\alpha$	Métrica	Media	Intervalo de Confianza del 95 %
0.95	0.50	Success-rate	0.997	[0.993, 1.000]
		Rew. media	-71.000	[-74.340, -67.660]
		Steps medios	71.0	[67.7, 74.3]
		Time (s)	1.65	[1.62, 1.67]
0.95	0.80	Success-rate	0.997	[0.994, 1.000]
		Rew. media	-71.039	[-73.977, -68.101]
		Steps medios	71.0	[68.1, 74.0]
		Time (s)	1.63	[1.61, 1.65]
0.99	0.20	Success-rate	0.997	[0.995, 0.999]
		Rew. media	-70.945	[-73.840, -68.051]
		Steps medios	70.9	[68.1, 73.8]
		Time (s)	1.68	[1.66, 1.71]

Cuadro 7: Resultados para combinaciones seleccionadas de  $\gamma$  y  $\alpha$

## Conclusiones

Del análisis experimental se pueden extraer las siguientes conclusiones:

1. El factor de descuento ( $\gamma$ ) tiene un impacto más significativo que la tasa de aprendizaje ( $\alpha$ ) en el rendimiento del algoritmo con la configuración de parámetro definida.
2. Los valores óptimos se encuentran en el rango de  $\gamma \geq 0.9$ .
3. La tasa de aprendizaje óptima parece estar entre 0.2 y 0.5.
4. La mejor combinación general considerando todas las métricas es  $\gamma = \mathbf{0.99}$  y  $\alpha = \mathbf{0.2}$ , que proporciona:
  - Una tasa de éxito del 99.7%.
  - Una recompensa media de -70.95.
  - Un tiempo de entrenamiento de 1.68 segundos.

Aunque los parámetros seleccionados no minimizan el tiempo de entrenamiento, las diferencias temporales observadas son mínimas. En este estudio se ha priorizado la optimización de la tasa de aciertos y la recompensa sobre el tiempo de entrenamiento, ya que estos criterios son más relevantes para evaluar la efectividad del algoritmo.

---

#### 4.2.2. Experimento tasa de exploración ( $\epsilon$ ) & decaimiento de la tasa de exploración ( $\epsilon$ )

##### 4.2.2.1 Diseño experimental

El objetivo de este experimento es analizar cómo la tasa de exploración y su decaimiento afectan el rendimiento del algoritmo Q-Learning.

<b>Observación</b>	El rendimiento y óptimalidad de la política encontrada por Q-Learning se ven afectados por la tasa de exploración y su decaimiento.
<b>Planteamiento</b>	Para cada combinación de $\epsilon$ y decaimiento de $\epsilon$ , se compara la tasa de acierto (llegar al estado final), la recompensa media, número de pasos y tiempo de entrenamiento del algoritmo.
<b>Hipótesis</b>	Un mayor valor de $\epsilon$ y un decaimiento más lento mejorarán el rendimiento del algoritmo.
<b>Método</b>	<ul style="list-style-type: none"><li>■ Se fijan 1000 episodios de entrenamiento, un coeficiente de decaimiento de la tasa de aprendizaje de 0.99, una penalización de la acción “moverse a la izquierda” de <math>-1</math>. y los mejores valores para <math>\gamma</math> y <math>\alpha</math> del experimento anterior.</li><li>■ Se eligen los siguientes valores para <math>\epsilon</math> y decaimiento de <math>\epsilon</math>: <math>\epsilon \in \{0.7, 0.9, 0.95, 0.99\}</math> y decaimiento de <math>\epsilon \in \{0.8, 0.9, 0.95, 0.99\}</math>.</li><li>■ Para cada combinación de <math>\epsilon</math> y decaimiento de <math>\epsilon</math>, se ejecuta el algoritmo Q-Learning en el entorno.</li><li>■ Se evalúa la política obtenida probándola con 500 episodios.</li><li>■ Se repite el proceso para cada combinación de <math>\epsilon</math> y decaimiento de <math>\epsilon</math> 20 veces para obtener una muestra representativa (debido a la estocasticidad del entorno).</li></ul>

Cuadro 8: Q-Learning - Experimento 2 - Tasa de exploracion & decaimiento de la tasa de exploracion

#### 4.2.2.2 Resultados

Los resultados del experimento se han representado mediante heatmaps para facilitar la visualización de los datos. En cada gráfico, el eje X representa la tasa de exploración ( $\epsilon$ ) y el eje Y representa el decaimiento de la tasa de exploración. Los colores indican el valor de la métrica correspondiente. Cada valor representa la media de las 20 ejecuciones del algoritmo (cada ejecución está representada por la media de 500 episodios) de la correspondiente combinación de parámetros.

##### Tasa de Éxito

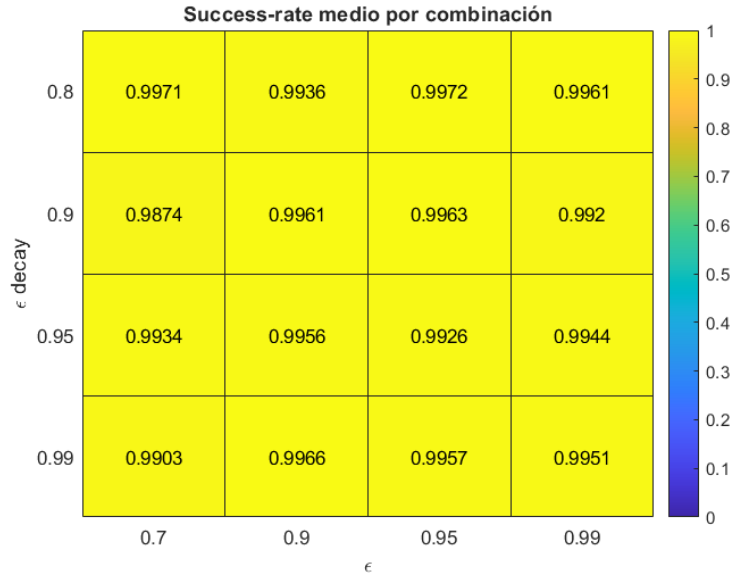
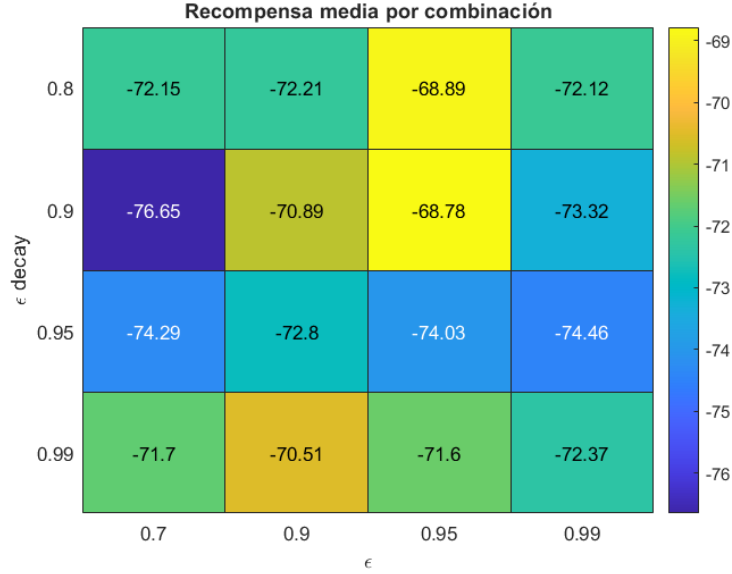


Figura 5: Tasa de éxito para diferentes combinaciones de  $\epsilon$  y su decaimiento

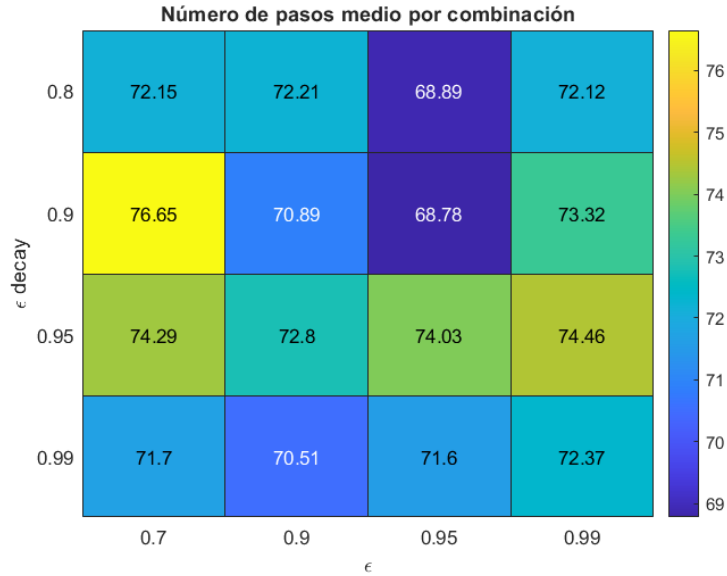
La tasa de éxito también es bastante elevada en este caso y no se observa ninguna tendencia clara. Todos los valores se encuentran entre el 98.7% y el 99.7%. La mejor combinación se alcanza con  $\epsilon = 0.95$  y su decaimiento = 0.8, logrando una tasa de éxito del 99.72%.



## Recompensa media y número de pasos medios



(a) Recompensa media para diferentes combinaciones de  $\epsilon$  y su decaimiento



(b) Número de pasos para diferentes combinaciones de  $\epsilon$  y su decaimiento

Figura 6: Análisis de la recompensa media y número de pasos

El análisis de la recompensa media y el número medio de pasos tampoco revela una tendencia clara. Lo que si que se puede observar es que todas las configuraciones se encuentran entre -76.65 y -68.79 de recompensa, lo que indica que ninguna de ellas se desvía ni obtiene resultados muy negativos.

Se puede observar también que el número medio de pasos corresponde exactamente

---

con la recompensa media en cada configuración. Esto nos indica que el agente no cae por el barranco en ningún momento, ya que sino la recompensa media sería más negativa.

Las combinaciones óptimas son  $\epsilon = 0.95$  y su decaimiento = 0.8, logrando una recompensa media de -68.89, y  $\epsilon = 0.95$  y su decaimiento = 0.9, logrando una recompensa media de -68.78.

### Tiempo de entrenamiento

Para el tiempo de entrenamiento, se han elegido las tres combinaciones que mejor rendimiento han dado.

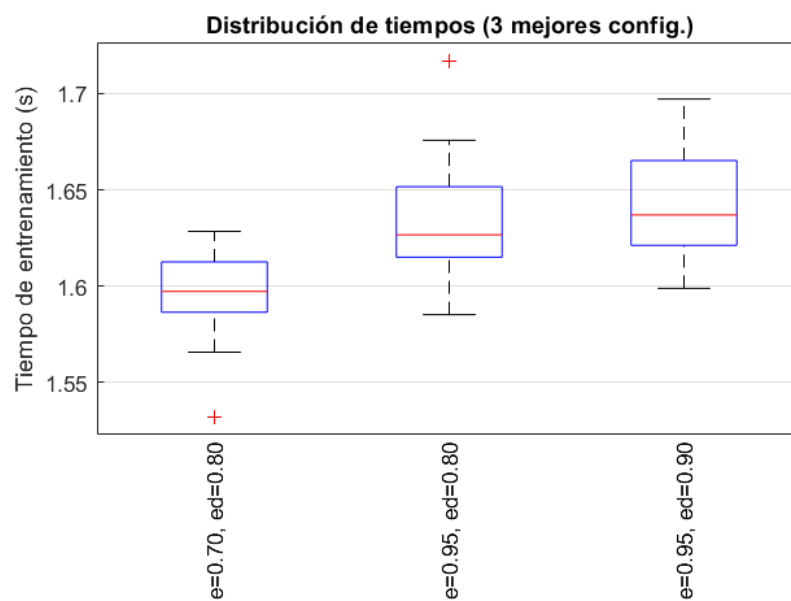


Figura 7: Tiempo de entrenamiento para diferentes combinaciones de  $\epsilon$  y su decaimiento

El análisis del tiempo de ejecución muestra que para las tres combinaciones, el tiempo se mueve entorno a los 1.62 segundos. Hay poca diferencia entre ellos, con una pequeña variabilidad en las colas.

---

## Tabla resumen

$\epsilon$	$decay$	Métrica	Media	Intervalo de Confianza del 95 %
0.70	0.80	Success-rate	0.997	[0.995, 0.999]
		Rew. media	-72.150	[-75.391, -68.910]
		Steps medios	72.2	[68.9, 75.4]
		Time (s)	1.60	[1.58, 1.61]
0.95	0.80	Success-rate	0.997	[0.995, 0.999]
		Rew. media	-68.889	[-70.869, -66.909]
		Steps medios	68.9	[66.9, 70.9]
		Time (s)	1.63	[1.61, 1.66]
0.95	0.90	Success-rate	0.996	[0.990, 1.003]
		Rew. media	-68.782	[-72.091, -65.473]
		Steps medios	68.8	[65.5, 72.1]
		Time (s)	1.64	[1.62, 1.66]

Cuadro 9: Resultados para combinaciones seleccionadas de  $\epsilon$  y  $decay$

## Conclusiones

Del análisis experimental se pueden extraer las siguientes conclusiones:

1. La tasa de exploración y su decaimiento no parecen tener un impacto significativo en el rendimiento del algoritmo Q-Learning.
2. La mejor combinación general considerando todas las métricas es  $\epsilon = \mathbf{0.95}$  y su decaimiento =  $\mathbf{0.8}$ , que proporciona:
  - Una tasa de éxito del 99.72 %.
  - Una recompensa media de -68.89.
  - Un tiempo de entrenamiento de 1.63 segundos.

Aunque los parámetros seleccionados no minimizan el tiempo de entrenamiento, las diferencias temporales observadas son mínimas. En este estudio se ha priorizado la optimización de la tasa de aciertos y la recompensa sobre el tiempo de entrenamiento, ya que estos criterios son más relevantes para evaluar la efectividad del algoritmo.

### 4.2.3. Experimento tasa de aprendizaje & decaimiento de la tasa de aprendizaje

#### 4.2.3.1 Diseño experimental

El objetivo de este experimento es analizar cómo la tasa de aprendizaje y su decaimiento afectan el rendimiento del algoritmo Q-Learning.

<b>Observación</b>	La tasa de aprendizaje ( $\alpha$ ) y su decaimiento son parámetros que influyen en la convergencia del algoritmo Q-Learning.
<b>Planteamiento</b>	Para cada combinación de $\alpha$ y su decaimiento, se compara la tasa de acierto (llegar al estado final), la recompensa media, número de pasos y tiempo de entrenamiento del algoritmo.
<b>Hipótesis</b>	Una tasa de aprendizaje lenta con un decaimiento gradual mejorará el rendimiento del algoritmo.
<b>Método</b>	<ul style="list-style-type: none"> <li>■ Se fijan 1000 episodios de entrenamiento, una penalización de la acción “moverse a la izquierda” de <math>-1</math>, valores de <math>\gamma</math>, y los mejores valores de <math>\gamma</math>, <math>\epsilon</math> y su decaimiento de los experimentos anteriores.</li> <li>■ Se eligen los siguientes valores para <math>\alpha</math> y su decaimiento: <math>\alpha \in \{0.1, 0.2, 0.5, 0.8\}</math> y decaimiento de <math>\alpha \in \{0.95, 0.99, 0.995, 0.999\}</math>.</li> <li>■ Para cada combinación de <math>\alpha</math> y su decaimiento, se ejecuta el algoritmo Q-Learning en el entorno.</li> <li>■ Se evalúa la política obtenida probándola con 500 episodios.</li> <li>■ Se repite el proceso para cada combinación de <math>\alpha</math> y su decaimiento 20 veces.</li> </ul>

Cuadro 10: Q-Learning - Experimento 3 - Tasa de aprendizaje & decaimiento de la tasa de aprendizaje

#### 4.2.3.2 Resultados

#### 4.2.4. Experimento número de episodios

##### 4.2.4.1 Diseño experimental

El objetivo de este experimento es analizar cómo el número de episodios de entrenamiento afectan el rendimiento del algoritmo Q-Learning.

<b>Observación</b>	El número de episodios de entrenamiento es un parámetro crítico en el algoritmo Q-Learning.
<b>Planteamiento</b>	Se compara la tasa de acierto (llegar al estado final), la recompensa media, número de pasos y tiempo de entrenamiento del algoritmo para diferentes números de episodios de entrenamiento.
<b>Hipótesis</b>	Un mayor número de episodios de entrenamiento mejorará el rendimiento del algoritmo.
<b>Método</b>	<ul style="list-style-type: none"> <li>■ Se fijan los mejores valores para <math>\gamma</math>, <math>\alpha</math>, decaimiento de <math>\alpha</math>, <math>\epsilon</math>, decaimiento de <math>\epsilon</math> de los experimentos anteriores y una penalización de la acción “moverse a la izquierda” de <math>-1</math>.</li> <li>■ Se eligen los siguientes <i>números de episodios de entrenamiento</i>: <math>\{500, 1000, 5000, 10000\}</math>.</li> <li>■ Para cada <i>número de episodios</i>, se ejecuta el algoritmo Q-Learning en el entorno.</li> <li>■ Se evalúa la política obtenida probándola con 500 episodios.</li> <li>■ Se repite el proceso para cada número de episodios 20 veces.</li> </ul>

Cuadro 11: Q-Learning - Experimento 3 - Número de episodios

#### 4.2.4.2 Resultados

#### 4.2.5. Experimento penalización de la acción izquierda

##### 4.2.5.1 Diseño experimental

El objetivo de este experimento es analizar si penalizar acciones poco favorables afectan el rendimiento del algoritmo Q-Learning.

<b>Observación</b>	Moverse a la izquierda no es deseable para el agente en ningún momento, ya que no le acerca al objetivo. Por lo tanto, penalizarla con una recompensa menor que las demás acciones puede alterar el comportamiento del agente.
<b>Planteamiento</b>	Se compara la tasa de acierto (llegar al estado final), la recompensa media, número de pasos y tiempo de entrenamiento del algoritmo para diferentes penalizaciones a la acción “moverse a la izquierda”.
<b>Hipótesis</b>	Penalizar la acción “moverse a la izquierda” mejorará el rendimiento del algoritmo.
<b>Método</b>	<ul style="list-style-type: none"> <li>■ Se fijan los mejores valores para <math>\gamma</math>, <math>\alpha</math>, decaimiento de <math>\alpha</math>, <math>\epsilon</math>, decaimiento de <math>\epsilon</math> de los experimentos anteriores.</li> <li>■ Se eligen los siguientes valores para la penalización de la acción “moverse a la izquierda”: <math>\{-1, -2, -10, -50\}</math>.</li> <li>■ Para cada penalización, se ejecuta el algoritmo Q-Learning en el entorno.</li> <li>■ Se evalúa la política obtenida probándola con 500 episodios.</li> <li>■ Se repite el proceso para cada penalización 20 veces.</li> </ul>

Cuadro 12: Q-Learning - Experimento 4 - Penalización de la acción “moverse a la izquierda”

#### 4.2.5.2 Resultados

---

## 5. Reinforce

### 5.1. Descripción del algoritmo

### 5.2. Experimentación

<b>Observación</b>	
<b>Planteamiento</b>	
<b>Hipótesis</b>	
<b>Método</b>	■

Cuadro 13: Experimento 1

---

## 6. Conclusiones



---

## 7. Bibliografia

### Referencias

- [1] Farama Foundation. Cliff walking environment. [https://gymnasium.farama.org/environments/toy\\_text/cliff\\_walking/](https://gymnasium.farama.org/environments/toy_text/cliff_walking/), 2025. Accedido: 15 de abril de 2025.
- [2] Farama Foundation. Gymnasium: A reinforcement learning library. <https://github.com/Farama-Foundation/Gymnasium>, 2025. Accedido: 15 de abril de 2025.

---

## 8. Apéndice