.UBAfiuba
FACULTAD DE INGENIERÍA

Trabajo práctico I

Aprendizaje por refuerzos II

Maestría en Inteligencia Artificial

Alumnos:

Ing. Fabricio Denardi (FIUBA)
Ing. Bruno Martin Masoller Gancedo (FIUBA)

Ciudad de Buenos Aires, Junio de 2025

Resumen

El objetivo del trabajo práctico fue implementar una competencia de algoritmos de aprendizaje por refuerzos para resolver el problema de *Mountain Car*, utilizando el entorno de Gymnasium.

La competencia creada incluye varias versiones de DQL Learning, PPO (vistos en Aprendizaje por refuerzos II) y se sumó a este desafío, un algoritmo de Aprendizaje por refuerzos I, QLearning.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este capítulo se detalla la resolución del trabajo práctico que consistió en la competencia entre varios algoritmos de aprendizaje por refuerzos para resolver un problema en particular de gymnasium.

1.1. *Gymnasium*

Gymnasium [1] es una biblioteca de Python para la creación, uso y evaluación de entornos de simulación en el desarrollo de algoritmos de *Reinforcement Learning* (Aprendizaje por Refuerzo). Es el sucesor de la biblioteca *Gym* desarrollada por OpenAI, y forma parte del ecosistema de *Farama Foundation*.

Gymnasium proporciona una interfaz estandarizada para una amplia variedad de entornos, lo que facilita la comparación de algoritmos y la reproducción de experimentos. Entre sus entornos más conocidos se encuentran los de control clásico como *CartPole*, *MountainCar* y *Acrobot*, además de entornos más complejos en 2D y 3D.

1.2. *MountainCar*

La competencia realizada en el trabajo práctico consistió en resolver el problema de *MountainCar* [2].

El problema de *Mountain Car* es un MDP determinista que consiste en un automóvil ubicado estocásticamente en el fondo de un valle sinusoidal. Las únicas acciones posibles son las aceleraciones que pueden aplicarse al automóvil en cualquiera de las dos direcciones.

El objetivo del MDP es acelerar estratégicamente el automóvil para que alcance el estado objetivo en la cima de la colina derecha. Existen dos versiones del entorno *Mountain Car* en **Gymnasium**: una con acciones discretas y otra con acciones continuas. Esta versión corresponde a la de acciones discretas.

1.2.1. Espacio de observación

La observación es un *array* de tipo `ndarray` con forma `(2,)` donde cada elemento representa lo siguiente:

TABLA 1.1. Espacio de observación del entorno *Mountain Car* en Gymnasium. La observación es un arreglo `ndarray` de forma `(2,)` donde cada entrada representa una magnitud física del sistema.

Observación	Mínimo	Máximo	Unidad
Posición del automóvil (eje x)	-1.2	0.6	posición (m)
Velocidad del automóvil	-0.07	0.07	velocidad (v)

1.2.2. Espacio de acción

El entorno *Mountain Car* utiliza un espacio de acción discreto con tres posibles acciones deterministas. Estas acciones corresponden a diferentes formas de aplicar fuerza al automóvil para intentar que alcance la cima de la colina derecha.

TABLA 1.2. Espacio de acción del entorno *Mountain Car* en Gymnasium.

Acción	Descripción
0	Acelerar hacia la izquierda
1	No acelerar
2	Acelerar hacia la derecha

1.2.3. Recompensas

El objetivo del agente en el entorno *Mountain Car* es alcanzar lo más rápido posible la bandera ubicada en la cima de la colina derecha. Para fomentar la eficiencia temporal, el entorno penaliza al agente con una recompensa de **-1** por cada paso de tiempo que transcurre sin alcanzar el objetivo.

1.2.4. Estado inicial

La posición del automóvil se asigna aleatoriamente con distribución uniforme en el intervalo $[-0{,}6, -0{,}4]$. La velocidad inicial del automóvil siempre se asigna como 0.

1.2.5. Fin del episodio

El episodio termina si se cumple alguna de las siguientes condiciones:

- **Terminación:** La posición del automóvil es mayor o igual a 0.5 (la posición objetivo en la cima de la colina derecha).
- **Truncamiento:** La duración del episodio alcanza los 200 pasos.

1.2.6. ¿Por qué es interesante el entorno *MountainCar*?

Requiere planificación y aprendizaje a largo plazo
El agente no puede alcanzar el objetivo directamente con una sola acción. Como el motor del auto es débil, debe oscilar hacia atrás primero (subiendo la colina izquierda) para acumular suficiente impulso y así subir la colina derecha.

Esto obliga al agente a aprender una política basada en secuencias de acciones que no parecen útiles de inmediato, lo que representa un desafío común en muchos problemas del mundo real.

Fomenta el aprendizaje por prueba y error
El entorno proporciona una recompensa escasa y negativa:

- El agente recibe una recompensa de **-1** por cada paso hasta alcanzar el objetivo.
- No existen recompensas positivas intermedias, solo al final del episodio.

Esto convierte al entorno en un buen caso de estudio para evaluar cómo los algoritmos manejan la exploración eficiente y el aprendizaje en presencia de recompensas diferidas.

1.3. Conclusiones

Capítulo 2

Competencia

En este capítulo se describe la competencia, resultados obtenidos, comparaciones entre modelos. En definitiva es la resolución del trabajo práctico.

2.1. Organización del código

En el repositorio del trabajo práctico [3] se puede acceder al código completo.

Es de acceso público y listo para ser ejecutado.

2.2. Modelos participantes

En esta sección se detallan todos los modelos intervinientes con sus hiperparámetros.

Varios de esos corresponden a *Double Deep Q-Learning* aunque con alguna variante o cambio de hiperparámetro.

Cada modelo tiene la misma estructura de código, según se ve en la figura 2.1

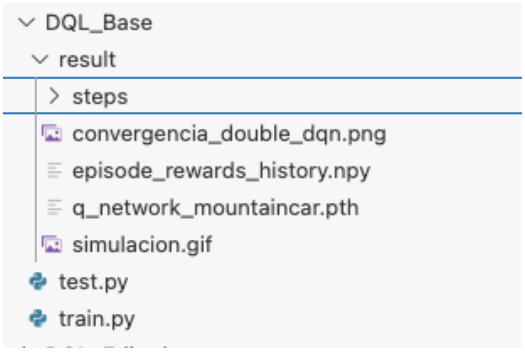

FIGURA 2.1. Estructura del código.

- train.py: código para el entrenamiento. Todo el código está autocontenido. Puede pensarse como que no está modularizado aunque la idea fue que no compartan código ni funciones para poder *customizar* el modelo participante lo más posible.
- test.py: ejecuta una prueba del código. result: carpeta que contiene los resultados del modelo.

2.2.1. Double Deep Q-Learning I

En la tabla 2.1 se pueden ver los hiperparámetros de este primer modelo de la competencia.

TABLA 2.1. Hiperparámetros utilizados en el entrenamiento del modelo

Hiperparámetro	Valor
Maximum episodes	5000
Maximum steps per episode	200
Learning rate	0.001
Discount factor (gamma)	0.99
Starting epsilon	1.0
Final epsilon	0.05
Epsilon decay episodes	3000
Replay buffer size	100,000
Batch size	64
Target network update frequency	1000
Print frequency	100
Smoothing window size	100

2.2.2. Double Deep Q-Learning Base

En la tabla 2.2 se pueden ver los hiperparámetros de esta versión modelo de la competencia.

TABLA 2.2. Hiperparámetros utilizados en el entrenamiento del modelo

Hiperparámetro	Valor
Maximum episodes	1250
Maximum steps per episode	1000
Learning rate	0.0005
Discount factor (gamma)	0.99
Starting epsilon	0.2
Final epsilon	0.01
Epsilon decay episodes	1000
Replay buffer size	50,000
Batch size	128
Target network update frequency	500
Print frequency	50
Smoothing window size	100

2.2.3. Double Deep Q-Learning Editado

En la tabla 2.3 se pueden ver los hiperparámetros de esta variante del modelo de la competencia.

La particularidad de este modelo es que se ha modificado la recompensa. A la recompensa por llegar a la meta, se le agrega una más inmediata de ir a la derecha:

TABLA 2.3. Hiperparámetros utilizados en el entrenamiento del modelo

Hiperparámetro	Valor
Maximum episodes	1250
Maximum steps per episode	1000
Learning rate	0.0005
Discount factor (gamma)	0.99
Starting epsilon	0.2
Final epsilon	0.01
Epsilon decay episodes	1000
Replay buffer size	50,000
Batch size	128
Target network update frequency	500
Print frequency	50
Smoothing window size	100

```
position = next_obs[0]
reward += (position + 0.5)
```

Esto llevó a resultados interesantes que más adelante se comentarán.

2.2.4. Proximal Policy Optimization (PPO)

En la tabla 2.4 se pueden ver los hiperparámetros de este modelo de la competencia.

TABLA 2.4. Hiperparámetros utilizados en el entrenamiento del modelo

Hiperparámetro	Valor
Policy	MlpPolicy
Environment	train_env
Learning rate	0.0005
Number of steps (n_steps)	12000
Batch size	128
Number of epochs (n_epochs)	200
Discount factor (gamma)	0.99
GAE lambda	0.95
Clip range	0.2
Entropy coefficient (ent_coef)	0.0
Value function coefficient (vf_coef)	0.5
Maximum gradient norm (max_grad_norm)	0.5
Verbosity	1
Seed	42

2.2.5. Q-Learning

Si bien es un algoritmo visto en la asignatura previa, Aprendizaje por Refuerzos I, se optó por incluirlo en esta competencia para poder tener una comparación con un *outsider*.

En la tabla 2.5 se pueden ver los hiperparámetros de esta variante del modelo de la competencia.

TABLA 2.5. Hiperparámetros utilizados en el entrenamiento del modelo

Hiperparámetros	**Valor**
Learning rate	0.1
Discount factor (gamma)	0.99
Initial exploration rate	0.2
Minimum exploration rate	0.01
Exploration decay rate	0.995
Number of episodes	1250
Maximum steps per episode	1000

2.3. Resultados obtenidos

La figura 2.2 muestra un gráfico de convergencia de todos los modelos que participaron en la competencia.

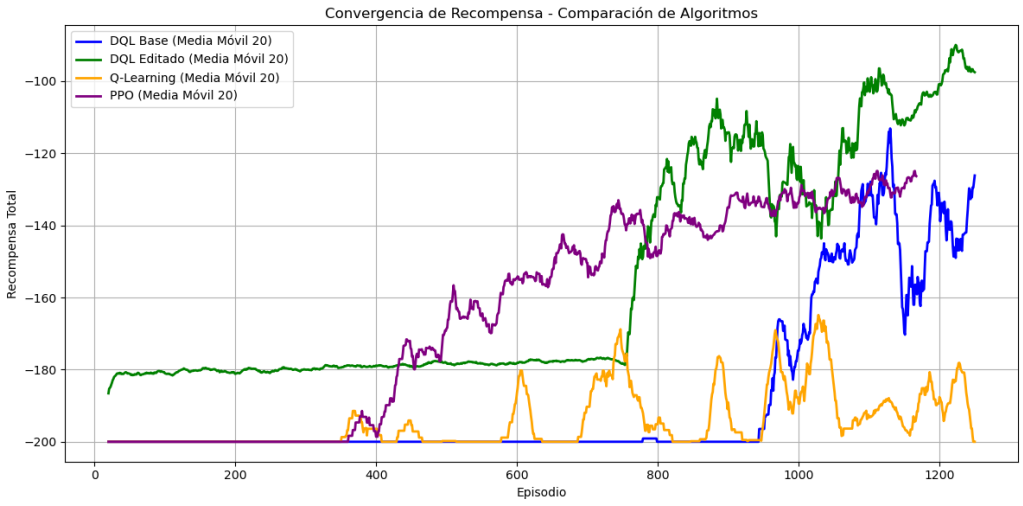

FIGURA 2.2. Convergencia de todos los modelos.

2.3.1. Double Deep Q-Learning I

La figura 2.3 muestra el gráfico de convergencia del modelo.

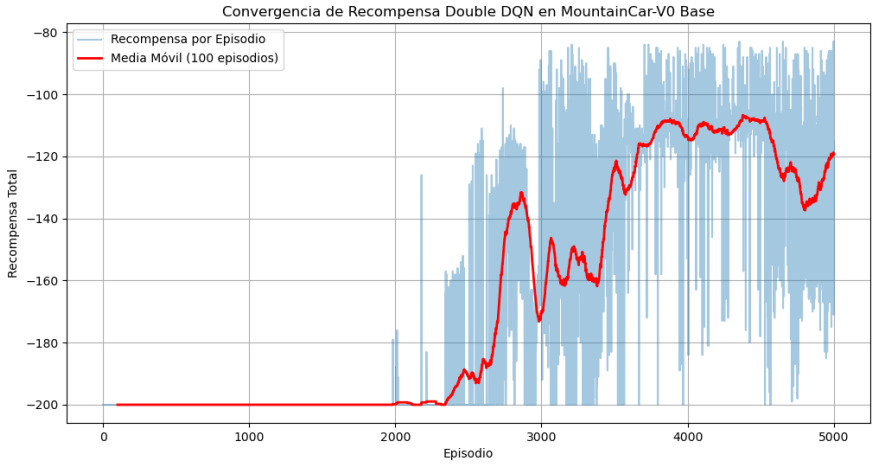

FIGURA 2.3. Convergencia Double Deep Q-Learning I.

2.3.2. Double Deep Q-Learning Base

La figura 2.4 muestra el gráfico de convergencia del modelo.

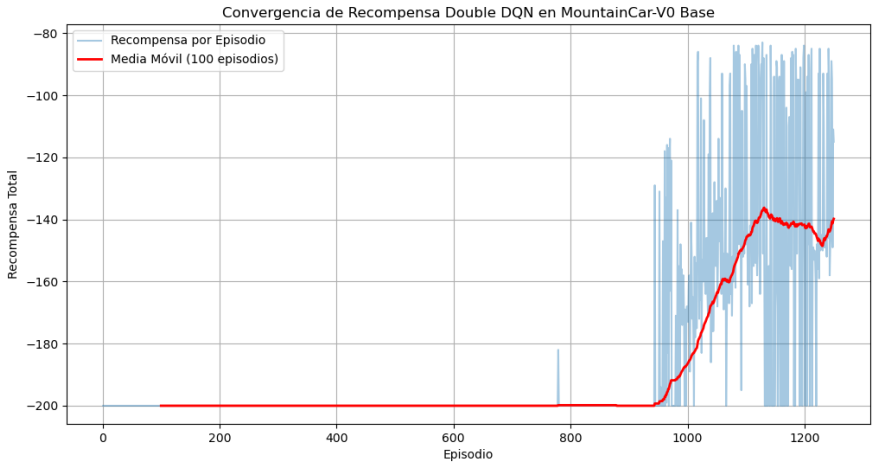

FIGURA 2.4. Convergencia Double Deep Q-Learning Base.

Observaciones: vemos que el algoritmo comienza a aprender en episodios tardíos. Esto particularmente se vio cuando recién en valores bajos de epsilons se lograba obtener una recompensa.

Es por esto que se decidió descartar de la competencia a dicho modelo.

2.3.3. Double Deep Q-Learning Editado

La figura 2.5 muestra el gráfico de convergencia del modelo.

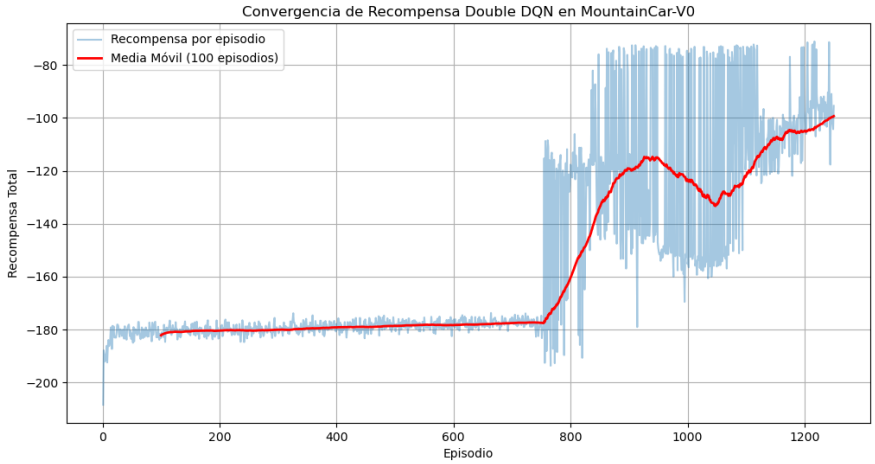

FIGURA 2.5. Convergencia Double Deep Q-Learning Editado.

Observaciones: vemos que con la modificación realizada en la recompenza, el modelo empieza a aprender antes que el anterior que, en escencia, tiene los mismos hiperparámetros.

Esto lo podemos corroborar en el *tracking* de entrenamiento:

```
--- Entrenamiento (Double DQN) ---
Ep  50: Recompensa -183.48 | Epsilon 0.191
Ep 100: Recompensa -181.17 | Epsilon 0.181
Ep 200: Recompensa -180.43 | Epsilon 0.162
Ep 300: Recompensa -180.02 | Epsilon 0.143
Ep 400: Recompensa -179.10 | Epsilon 0.124
Ep 500: Recompensa -179.06 | Epsilon 0.105
Ep 600: Recompensa -156.53 | Epsilon 0.086
Ep 700: Recompensa -135.39 | Epsilon 0.067
Ep 800: Recompensa -133.22 | Epsilon 0.048
Ep 900: Recompensa -104.07 | Epsilon 0.029
Ep 1000: Recompensa -90.21 | Epsilon 0.010
Ep 1100: Recompensa -94.43 | Epsilon 0.010
Ep 1200: Recompensa -90.26 | Epsilon 0.010
Ep 1250: Recompensa -85.50 | Epsilon 0.010
--- Entrenamiento Finalizado ---
```

2.3.4. Proximal Policy Optimization (PPO)

La figura 2.6 muestra el gráfico de convergencia del modelo.

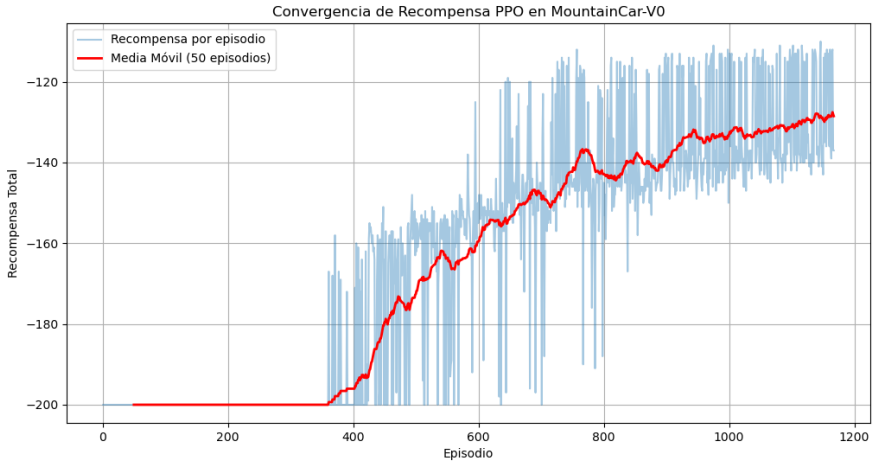

FIGURA 2.6. Convergencia PPO.

2.3.5. Q-Learning

La figura 2.7 muestra el gráfico de convergencia del modelo.

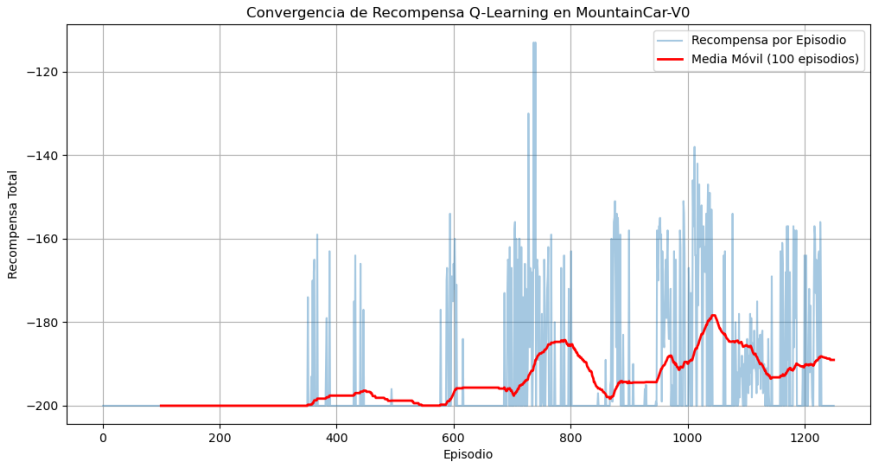

FIGURA 2.7. Convergencia Q-Learning.

2.4. *Testing*

Todos los modelos lograron el objetivo de llegar a la banderilla de la colina derecha. La tabla 2.6 muestra un resumen del resultado obtenido para cada modelo en la etapa de *testing*.

TABLA 2.6. Pasos numerados para alcanzar el objetivo junto con el enlace al repositorio correspondiente.

Modelo	**Pasos**	**Animación**
Double Deep Q-Learning Edited	90	link
PPO	110	link
Double Deep Q-Learning Base	145	link
Q-Learning	154	link

2.4.1. Modelo ganador

El modelo ganado fue:

Double Deep Q-Learning Edited

2.5. Conclusiones

A partir de los resultados obtenidos, se pueden extraer las siguientes conclusiones:

- El modelo **Double Deep Q-Learning (editado)** fue el más eficiente, alcanzando el objetivo en tan solo **90 pasos**, lo que indica una política efectiva y bien ajustada.
- El algoritmo **Proximal Policy Optimization (PPO)** también mostró un desempeño competitivo, logrando el objetivo en **110 pasos**. Este resultado demuestra su capacidad de generalización y estabilidad en entornos con recompensas escasas.
- La versión base de **Double Deep Q-Learning** requirió **145 pasos**, evidenciando mejoras concretas tras los ajustes implementados en la versión editada.
- El enfoque tradicional de **Q-Learning**, sin técnicas profundas o experiencia replay, fue el menos eficiente, con **154 pasos**, resaltando las limitaciones de los métodos tabulares frente a entornos continuos o de mayor complejidad.
- El entorno *MountainCar* requiere una estrategia de planificación a largo plazo, ya que el agente debe retroceder primero para ganar impulso y luego avanzar para alcanzar la cima.
- En las primeras etapas del entrenamiento, un valor alto de ϵ favorece la exploración de diversas trayectorias, lo cual es fundamental para descubrir la estrategia adecuada.
- A medida que el agente aprende, es necesario reducir progresivamente el valor de ϵ para:
  - Consolidar la política aprendida.
  - Disminuir la cantidad de acciones aleatorias que podrían dificultar el rendimiento.
  - Lograr comportamientos más consistentes y eficientes.
- Un valor de ϵ bajo (por ejemplo, 0.01) en las últimas etapas permite al agente explotar lo aprendido y alcanzar el objetivo en menos pasos, maximizando la recompensa acumulada.

En resumen, los modelos basados en aprendizaje profundo demostraron una ventaja clara en eficiencia y convergencia respecto a métodos más simples, siendo especialmente notoria la mejora introducida por ajustes en el modelo Double DQN.

Reward shaping usado en el modelo Double Deep Q-Learning (editado)

- En el entorno original *MountainCar*, la recompensa es escasa y constante: el agente recibe -1 en cada paso hasta alcanzar el objetivo, sin indicaciones sobre qué tan cerca está.
- El **reward shaping** implementado con

  $$\text{reward} += (\text{position} + 0{,}5)$$

  proporciona una señal de recompensa densa que aumenta conforme el auto se acerca a la cima derecha.
- Dado que la posición varía aproximadamente entre $-1{,}2$ y $0{,}6$, sumar position$+ 0{,}5$ añade una recompensa positiva creciente a medida que el agente se mueve hacia el objetivo.
- Esto incentiva al agente a avanzar hacia la derecha y evita movimientos ineficientes, mejorando la política aprendida.
- Como resultado, el agente recibe información más rica sobre el progreso en cada paso, lo que facilita un aprendizaje más rápido y estable.

El *reward shaping* convierte un entorno con recompensas escasas en uno con señales de progreso más informativas, mejorando significativamente la eficiencia y estabilidad del aprendizaje del agente.

Bibliografía

[1] The Farama Foundation. *Gymnasium: A standard API for reinforcement learning environments*. https://gymnasium.farama.org. Visitado: 2025-05-27. 2023.

[2] The Farama Foundation. *MountainCar Environment – Gymnasium*. https://gymnasium.farama.org/environments/classic_control/mountain_car/. Visitado: 2025-05-27. 2023.

[3] Fabricio Denardi y Bruno Martín Masoller Gancedo. *MIA_01c_AR2: Trabajo práctico final de Aprendizaje por Refuerzo II*. https://github.com/denardifabricio/MIA_01c_AR2. Repositorio del trabajo práctico final del posgrado en Inteligencia Artificial, cohorte MIA 01-2025. 2025.