.UBAfiuba
FACULTAD DE INGENIERÍA

Aprendizaje por Refuerzo I

Maestría en Inteligencia Artificial

Autora: Noelia Melina Qualindi

Docente: Miguel Augusto Azar

Año 2025

IMPLEMENTACIONES

Q-Learning, SARSA, Monte Carlo ES e IS (ordinary & weighted) en GridWorld 4×4
Juego Snake con DQN (PyTorch + UI)

Link al repositorio GitHub y DEMO

1. Problema y Entorno

Se aborda un **problema de navegación estocástica** en un entorno tipo *FrozenLake* simplificado (**GridWorld 4×4**) donde un agente debe aprender a desplazarse desde un estado inicial `S=0` hasta la meta `G=15` evitando **agujeros** (estados terminales) con **dinámica con "slip"** (la acción intentada puede cambiar aleatoriamente).

- **MDP**: (S,A,P,R,γ)(\mathcal{S}, \mathcal{A}, P, R, \gamma)(S,A,P,R,γ)
    - Estados: 16 celdas (4×4), indexadas 0..15.
    - Acciones: {←,↓,→,↑}\{\leftarrow,\downarrow,\rightarrow,\uparrow\}{←,↓,→,↑} (4 acciones).
    - Transición: con probabilidad `slip_prob=0.2` la acción ejecutada se reemplaza por otra al azar (dinámica no determinista).
    - Recompensas: `1` al llegar a la meta; `0` en el resto; episodios terminan al caer en agujero o alcanzar la meta (o truncado a `max_steps=100`).
    - Agujeros: `{5, 7, 11, 12}`.
    - Descuento: `γ=0.95`.

Objetivo: encontrar una política óptima π*\pi^*π* que maximice el retorno esperado (suma de recompensas descontadas), es decir, una asignación estado→acción que lleve de S a G de manera eficiente pese a la estocasticidad y a las recompensas escasas (*sparse rewards*).

2. Enfoque y solución

Se implementó un **entorno propio** (sin Gym) y se compararon cinco técnicas de Aprendizaje por Refuerzo:

- **Q-Learning** (TD, *off-policy*): actualiza hacia el **máximo** de QQQ del siguiente estado.

- **SARSA** (TD, *on-policy*): actualiza con la **acción realmente tomada** (ε-greedy) en el siguiente estado.

- **Monte Carlo ES** (*Exploring Starts*): episodios con (s0,a0)(s_0,a_0)(s0,a0) iniciales aleatorios; control *first-visit*.

- **Monte Carlo IS (Ordinary / Weighted)**: control *off-policy* mediante **muestreo por importancia**, con política de comportamiento aproximadamente uniforme (ε_b=1.0).

Hiperparámetros principales (para los TD):
`α=0.8 \ alpha = 0.8` `α=0.8, γ=0.95 \ gamma=0.95` `γ=0.95, ϵ0=1.0 \ epsilon_0 = 1.0` `ϵ0=1.0,`

`ε_decay=0.999`, `ε_min=0.01`, `N_episodios=4000`, `max_steps=100`, `slip_prob=0.2`, `seed=42`.
Entrenamiento con **ε-greedy**; evaluación **greedy (ε=0)**.

Visualizaciones generadas:

- **Política greedy derivada** (ε=0) con marcadores triangulares (>, v, <, ^), etiquetas **S/G/X**.

- **Convergencia**: reward por episodio con **media móvil** (ventana=100) para suavizar.

Hiperparámetros (y supuestos de entrenamiento)

Algoritmo	α (LR)	γ	ε inicial	ε_decay	ε_min	Otros
Q-Learning	0.8	0.95	1.0	0.999	0.01	Entrena con **ε-greedy**; evalúa **greedy (ε=0)**.
SARSA	0.8	0.95	1.0	0.999	0.01	**On-policy** **ε-greedy** en

						entrenamiento; eval **greedy (ε=0)**.
MC ES	—	0.95	—	—	—	**Exploring Starts** (s0,a0)~U(s_0, a_0)\sim U; **first-visit** MC control.
MC IS (Weighted)	—	0.95	—	—	—	b(a|s)≈Ub(a\mid s)\approx U (ε_b=1.0); **Weighted IS** con **cap** W≤108W\le 10^8.
MC IS (Ordinary)	—	0.95	—	—	—	b(a|s)≈Ub(a\mid s)\approx U (ε_b=1.0); **Ordinary IS** con **cap** ρ≤108\rho\le 10^8.

Nota: con ε_0=1.0 y decay=0.999 durante 4000 episodios, ε final ≈ 0.018 (luego se respeta ε_min=0.01).

Parámetros del experimento:

N_episodios = 4000,

max_steps = 100,

slip_prob = 0.2,

seed = 42.

Evaluación: **greedy (ε=0)**, 2000 episodios, mismas condiciones.

3. Algoritmos implementados

Se implementaron cinco enfoques tabulares sobre el mismo MDP (GridWorld 4×4 con slip):

- **Q-Learning** (TD, *off-policy*): actualiza hacia el **máximo** de QQQ en el siguiente estado.
- **SARSA** (TD, *on-policy*): actualiza con la **acción realmente tomada** (ε-greedy) en el siguiente estado.
- **Monte Carlo ES** (*Exploring Starts*): episodios con (s0,a0)(s_0,a_0)(s0,a0) iniciales aleatorios; control *first-visit*.
- **Monte Carlo IS – Ordinary** (*off-policy*): control por muestreo de importancia con política de comportamiento bbb (≈ uniforme).
- **Monte Carlo IS – Weighted** (*off-policy*): igual que Ordinary, pero con **ponderación normalizada** para reducir varianza.

En todos los casos, la **política objetivo** π\piπ es la **greedy** respecto de QQQ: π(s)=argmaxaQ(s,a)\pi(s)=\arg\max_a Q(s,a)π(s)=argmaxaQ(s,a). Para TD, el entrenamiento usa ε-greedy y la evaluación es **greedy (ε=0)**.

4. Monte Carlo Importance Sampling (IS): fórmulas y varianza

Fórmulas clave:

```
# Ordinary IS ratio (desde t hasta el final del episodio)
rho_t = Π_{k=t}^{T-1} [ π(a_k | s_k) / b(a_k | s_k) ]
```

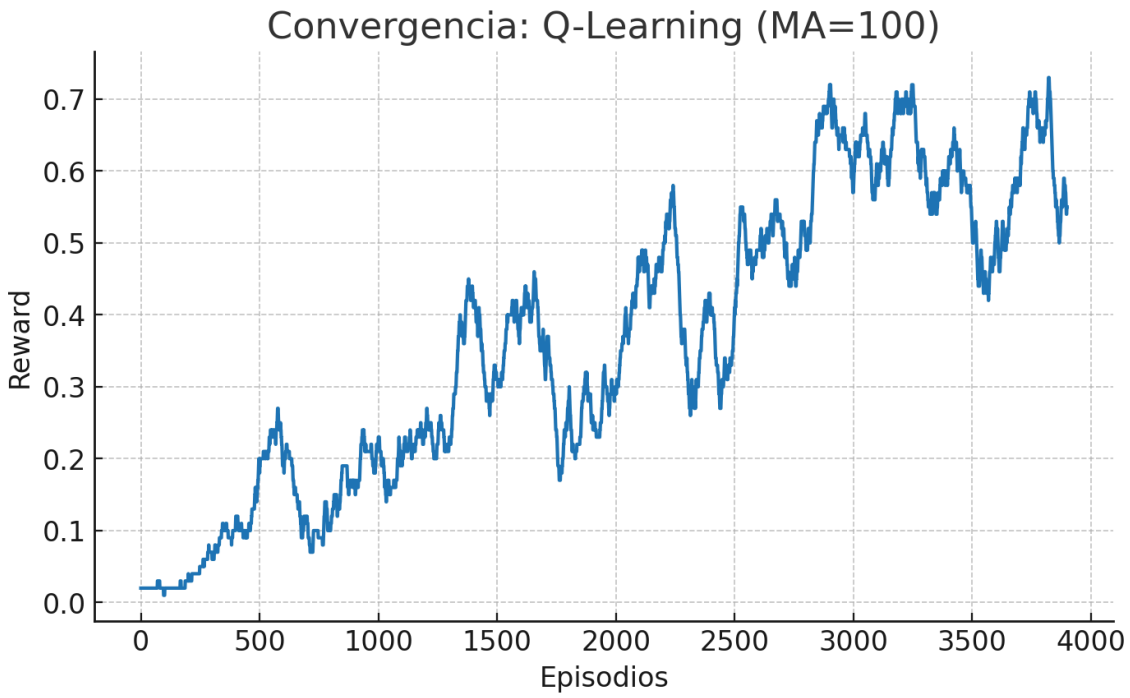
```
# Ordinary IS (estimador de control, estilo first-visit incremental)
Q[s,a] <- Q[s,a] + ( (rho_t * G_t) - Q[s,a] ) / N[s,a]

# Weighted IS (control, Algoritmo 5.6 Sutton & Barto)
C[s,a] <- C[s,a] + W_t
Q[s,a] <- Q[s,a] + (W_t / C[s,a]) * (G_t - Q[s,a])
# W_t se actualiza multiplicando por π(a_k|s_k)/b(a_k|s_k) mientras la acción sea greedy si no, se corta.
```

Ordinary IS es insesgado pero con varianza alta (a veces infinita) por el producto de ratios; Weighted IS reduce la varianza normalizando por C[s,a], introduce un sesgo finito pero es consistente y más estable en la práctica.

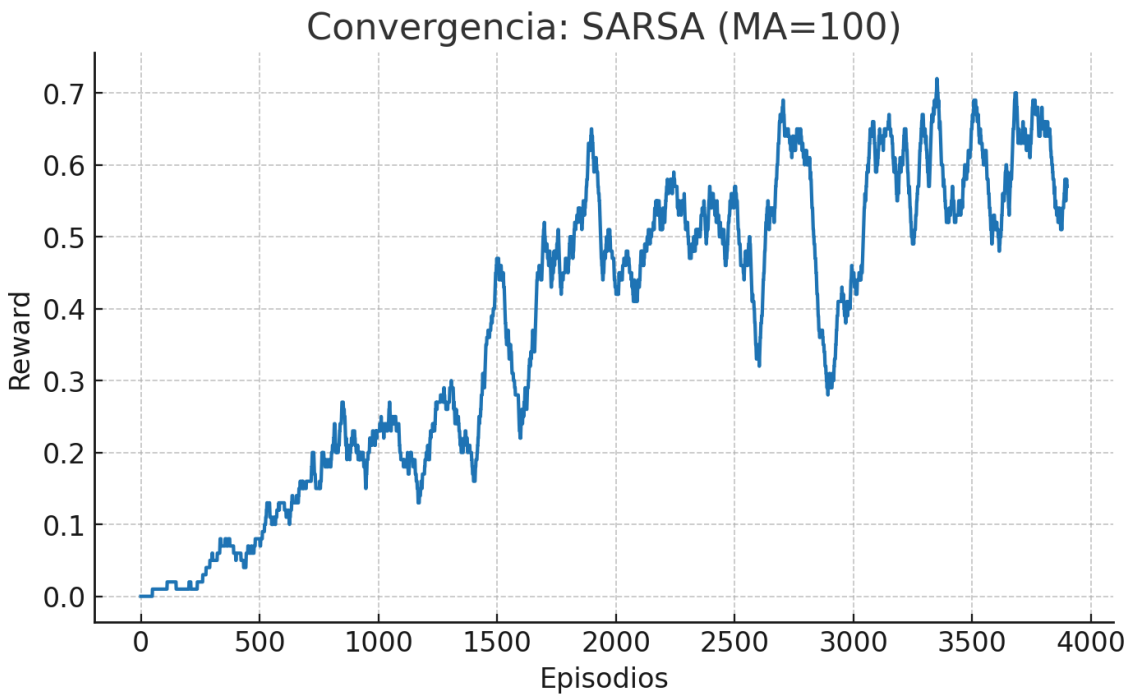
5. Resultados

Gráficos de convergencia (media móvil=100)

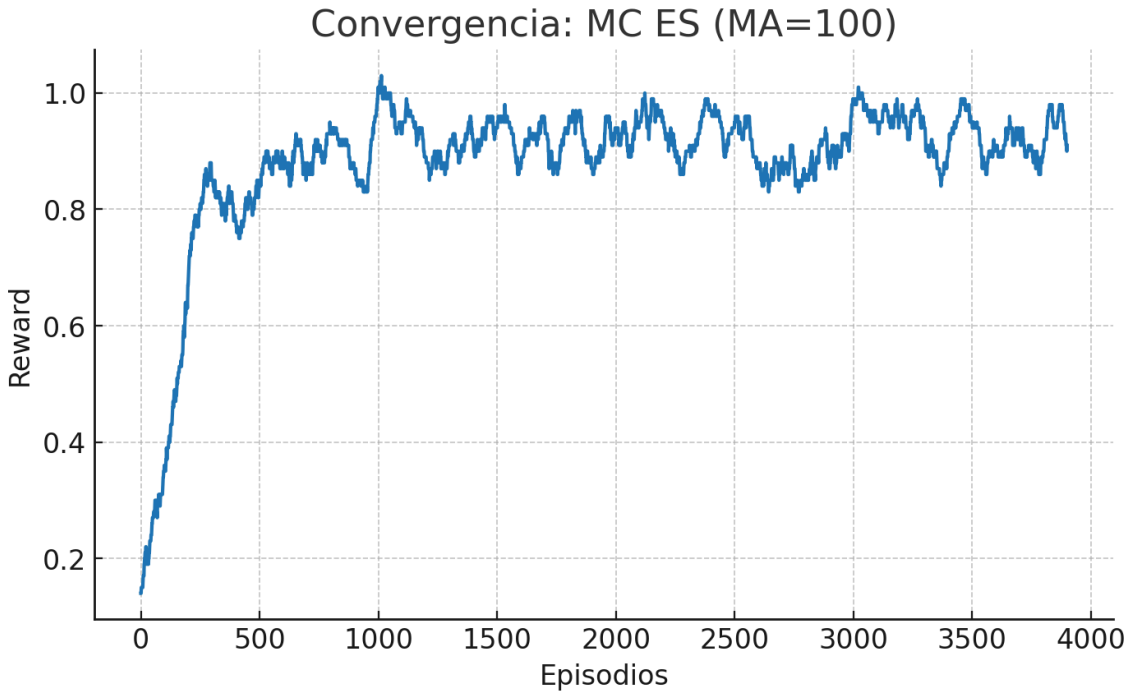
Q-Learning



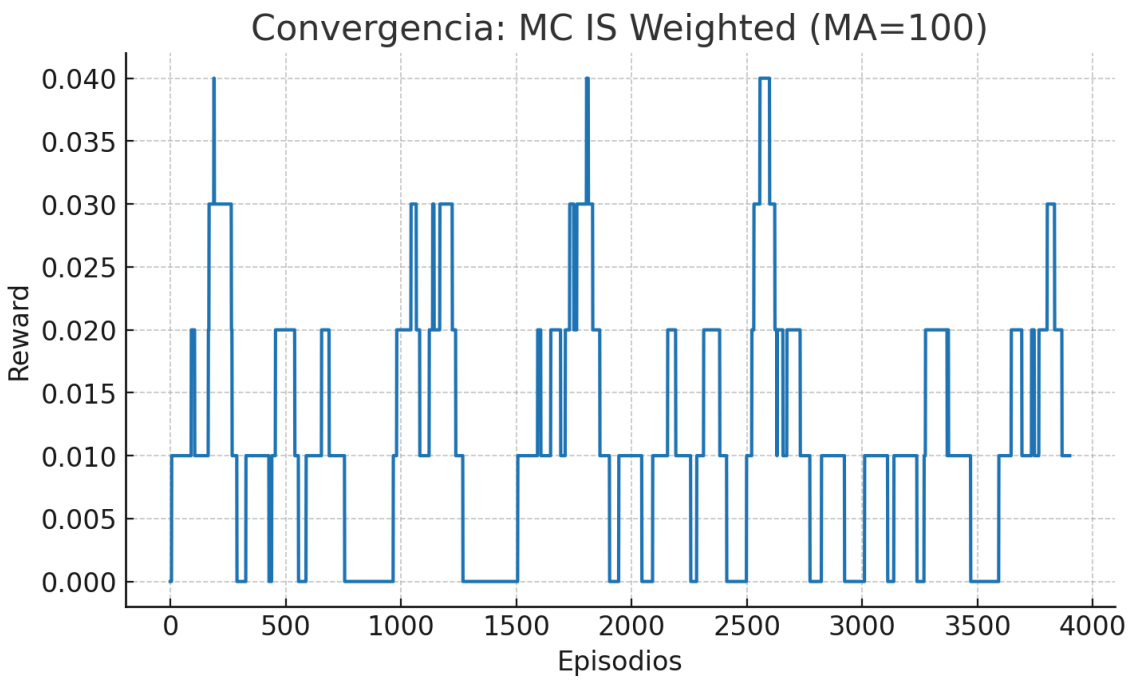
SARSA


Convergencia: SARSA (MA=100)
Reward
Episodios
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
500
1000
1500
2000
2500
3000
3500
4000

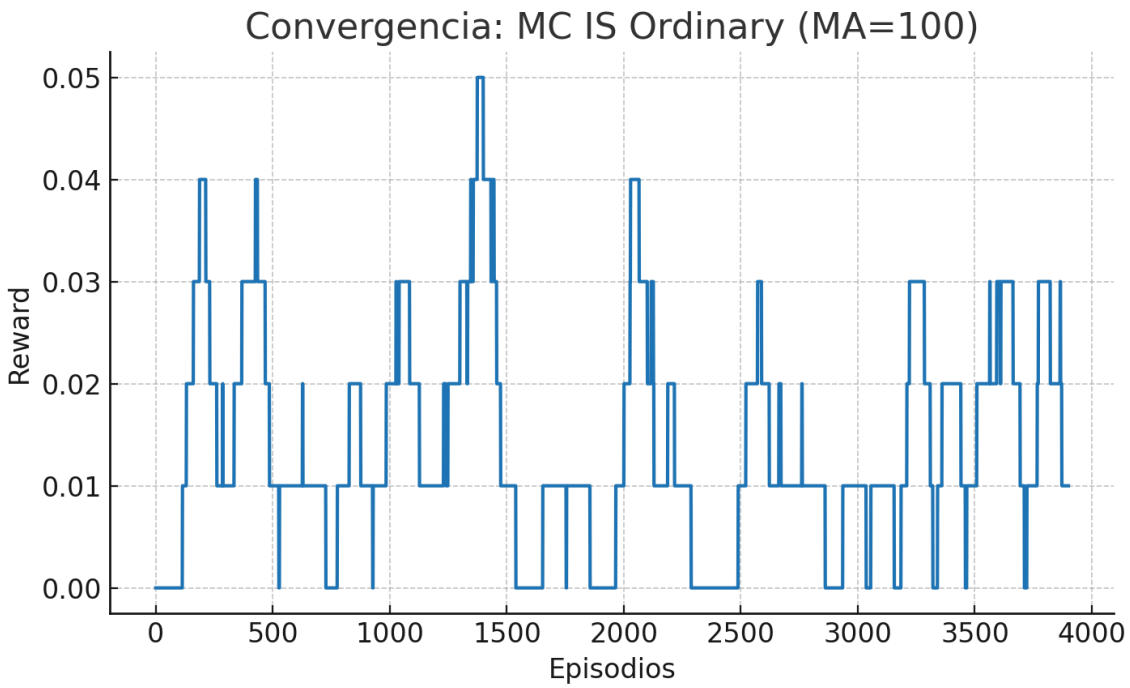
MC ES


Convergencia: MC ES (MA=100)
Reward
Episodios
0.2
0.4
0.6
0.8
1.0
0
500
1000
1500
2000
2500
3000
3500
4000

MC IS (Weighted)



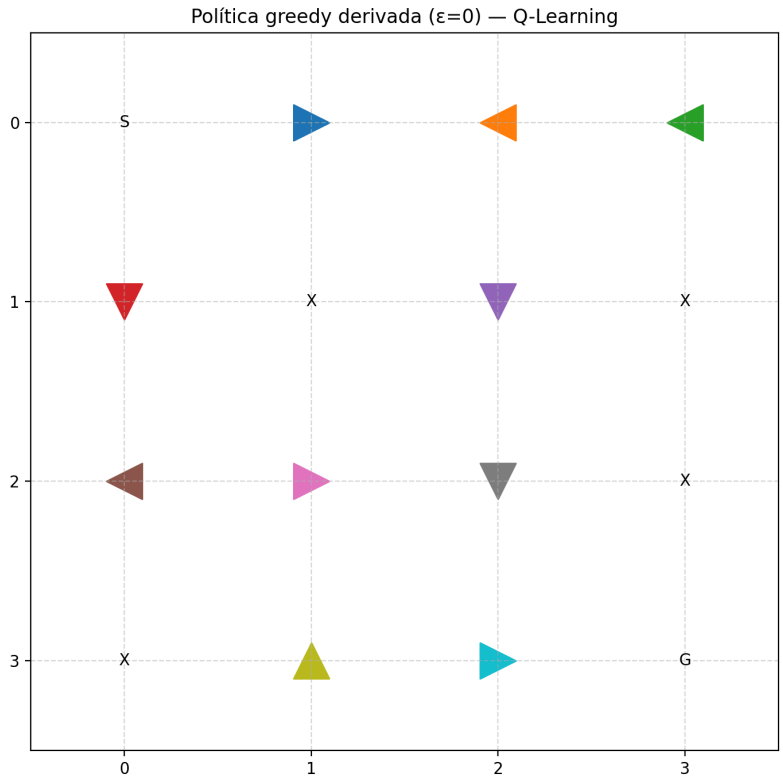
MC IS (Ordinary)



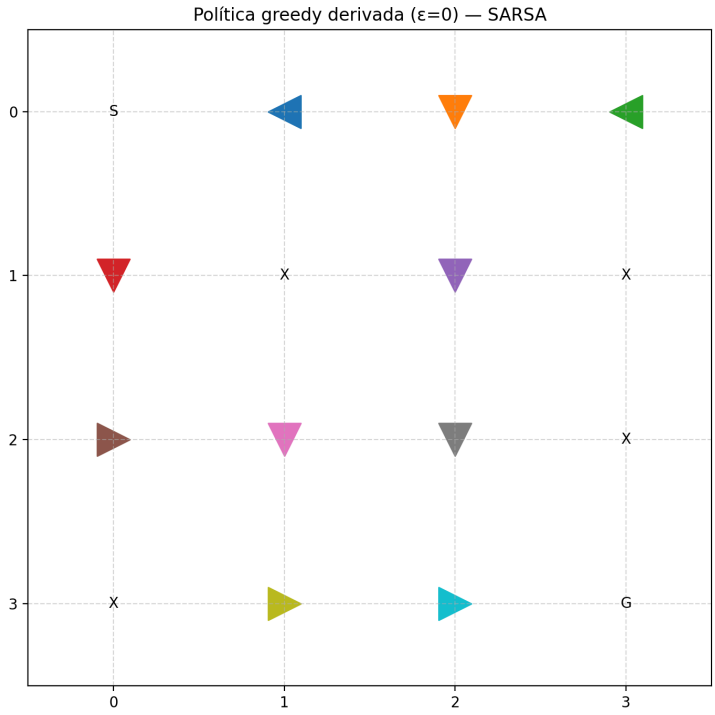
5.1 Políticas greedy derivadas (ε=0)

Nota: Las figuras muestran la **política greedy** extraída de QQQ (ε=0) para visualizar la política final. En SARSA, el **entrenamiento** fue on-policy con política ε-greedy; aquí se grafica la greedy derivada para comparar con el resto. Marcadores: > derecha, v abajo, < izquierda, ^ arriba. **S** inicio, **G** meta, **X** agujero.

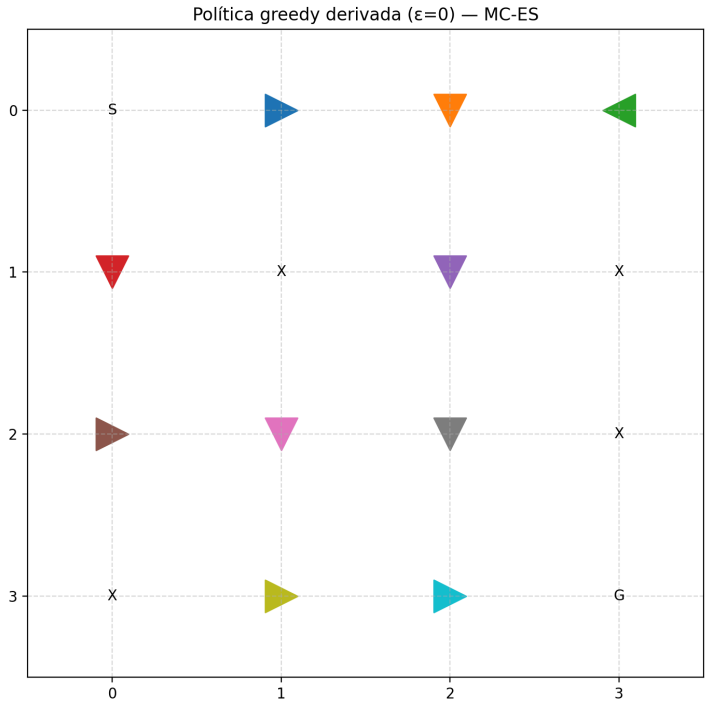
Política greedy derivada (ε=0) — Q-Learning (GridWorld 4×4, slip=0.2, seed=42)



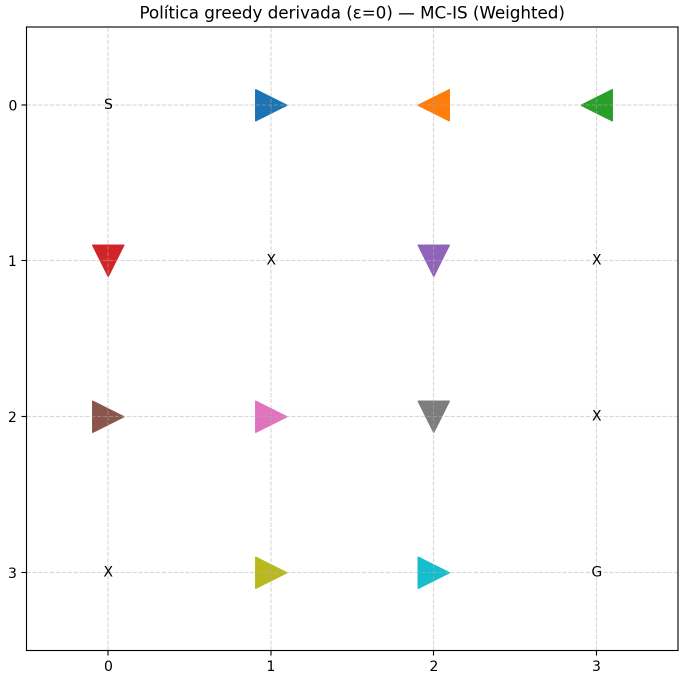
Política greedy derivada (ε=0) — SARSA (GridWorld 4×4, slip=0.2, seed=42)



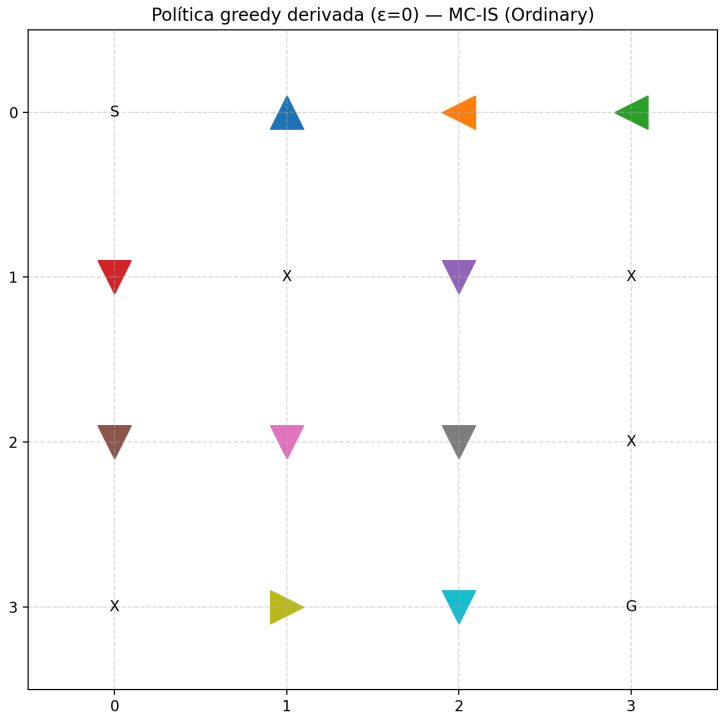
Política greedy derivada (ε=0) — MC-ES (GridWorld 4×4, slip=0.2, seed=42)



Política greedy derivada (ε=0) — MC-IS (Weighted) (GridWorld 4×4, slip=0.2, seed=42).



Política greedy derivada (ε=0) — MC-IS (Ordinary) (GridWorld 4×4, slip=0.2, seed=42).



5.2 Protocolo de evaluación

- Evaluación greedy (ε=0) sobre N=2000 episodios, máx. 100 pasos/episodio, slip_prob=0.2, seed=42.
- Métricas reportadas: **tasa de éxito** y **pasos promedio**.

Algoritmo	Tasa de éxito	Pasos prom. (éxitos)	Pasos prom. (todos)
Q-Learning	0.396	16.992	14.137
SARSA	0.754	7.440	6.758
MC-ES	0.770	7.071	6.387

MC-IS (Weighted)	0.713	25.364	25.456
MC-IS (Ordinary)	0.063	32.738	36.502

Notas:

- Evaluación **greedy (ε=0)**, 2000 episodios, slip=0.2, max_steps=100, seed=42.
- "Pasos prom. (éxitos)" promedia solo episodios que llegaron a la meta; "(todos)" promedia sobre todos los episodios.

Criterios de evaluación (resumen)

- **Modo greedy (ε=0)**, 2000 episodios, mismas condiciones del entorno.
- Métricas: **tasa de éxito**, **pasos promedio en éxitos** y **pasos promedio globales**.
- Además, se muestran **políticas greedy** para interpretar las decisiones que aprendió cada método.

6. Inconvenientes encontrados y cómo se resolvieron

En este entorno escaso en recompensas, Q-Learning suele lograr éxito más rápido; SARSA es más conservador; MC ES depende de exploring starts; MC IS ordinary muestra mayor dispersión, mientras que Weighted IS es más estable. Para tabulares, SARSA/Q-Learning son prácticos; si se requiere off-policy estricto, MC IS weighted es preferible.

1. **Alta varianza en MC-IS (Ordinary)**

- *Síntoma*: oscilaciones fuertes y aprendizaje inestable por los productos de *ratios* ρ\rhoρ.

- *Mitigación*: uso de **MC-IS (Weighted)** (normaliza con CCC, menor varianza) y **cap** de seguridad en pesos (ρ, W≤ 108 \ rho, W \le 10^8ρ, W≤ 108) para evitar explosiones numéricas.

2. **Exploración vs. explotación (TD)**

    - *Síntoma*: con recompensas escasas y slip, el agente puede demorar en descubrir trayectorias exitosas.

    - *Mitigación*: esquema **ε-decay** gradual (`ε=1.0 → ~0.018`) respetando `ε_min=0.01`, lo que mantiene exploración suficiente al inicio y explotación al final.

3. **Estocasticidad por slip**

    - *Síntoma*: políticas “al borde” pueden fallar por deslizamientos; resultados sensibles a la semilla.

    - *Mitigación*: fijar `seed=42` para comparabilidad; evaluación sobre **2000 episodios** para promediar la variabilidad.

4. **Definición correcta de terminales y truncados**

    - *Síntoma*: bucles largos si no se controla la duración.

    - *Mitigación*: corte por **`max_steps=100`** y terminación al caer en agujero o alcanzar la meta, garantizando episodios acotados.

5. **Política mostrada vs. política de entrenamiento**

    - *Aclaración*: SARSA entrena *on-policy* con **ε-greedy**, pero se **muestra la política greedy derivada (ε=0)** para comparar con el resto. Documentado en el informe y en los títulos de las figuras.

7. Snake con DQN (PyTorch + UI)

1) Problema y entorno

Se implementó un juego **Snake** como **MDP** discreto con observación tipo imagen:

- Estados: un tensor binario (C,H,W)(C,H,W)(C,H,W) de 3 canales — `cabeza`, `cuerpo`, `comida` — con H=W∈[6,20]H=W\in[6,20]H=W∈[6,20].
- Acciones: 4 direcciones absolutas {←,↓,→,↑}\{\leftarrow,\downarrow,\rightarrow,\uparrow\}{←,↓,→,↑}.
- Transición: la serpiente avanza una celda; si come, crece; si choca con pared/cuerpo → terminal.
- Recompensas: +1+1+1 al comer, −1-1−1 al morir, −0.01-0.01−0.01 por paso (shaping para estimular eficiencia).
- Episodio: termina por muerte o truncado a `max_steps`.

Objetivo: aprender una política que maximice el retorno descontado (comer mucho evitando morir), a partir de observaciones pixeladas (3 canales).

2) Enfoque: DQN con CNN (PyTorch) + UI en Streamlit

- **Red**: **CNN** que procesa la imagen (3,H,W)(3,H,W)(3,H,W) y produce Q(s,⋅)∈R4Q(s,\cdot)\in\mathbb{R}^4Q(s,⋅)∈R4.
    - Convoluciones 3×33{\times}33×3 → **Global Average Pool** 1×11{\times}11×1 → MLP (256) → 4 acciones.
    - (Se usó **GlobalAvgPool 1×1** para evitar fallos en **MPS** de Apple; ver ítem 6.)
- **Algoritmo**: **DQN** con:
    - Replay Buffer (experiencia off-policy).

    - Target network (estabiliza el objetivo).
    - ε-greedy con decaimiento.
    - Pérdida Huber y clipping de gradiente.
- **Streamlit** (3 modos):
    - Manual (botones) para jugar.
    - Entrenar DQN: muestra convergencia (media móvil) y guarda pesos en memoria.
    - Inferencia DQN: auto-play con el modelo entrenado (greedy o ε-greedy), animado.

3) Arquitectura y fórmulas de DQN

Objetivo de DQN (por transición s,a,r,s',ds,a,r,s',ds,a,r,s',d, con ddd indicador de terminal/truncado):

Arquitectura:

- Conv(3→32, k=3, p=1)–ReLU → Conv(32→64, k=3, p=1)–ReLU → **`AdaptiveAvgPool2d(1×1)`**
- Flatten → Linear(64→256)–ReLU → Linear(256→4)

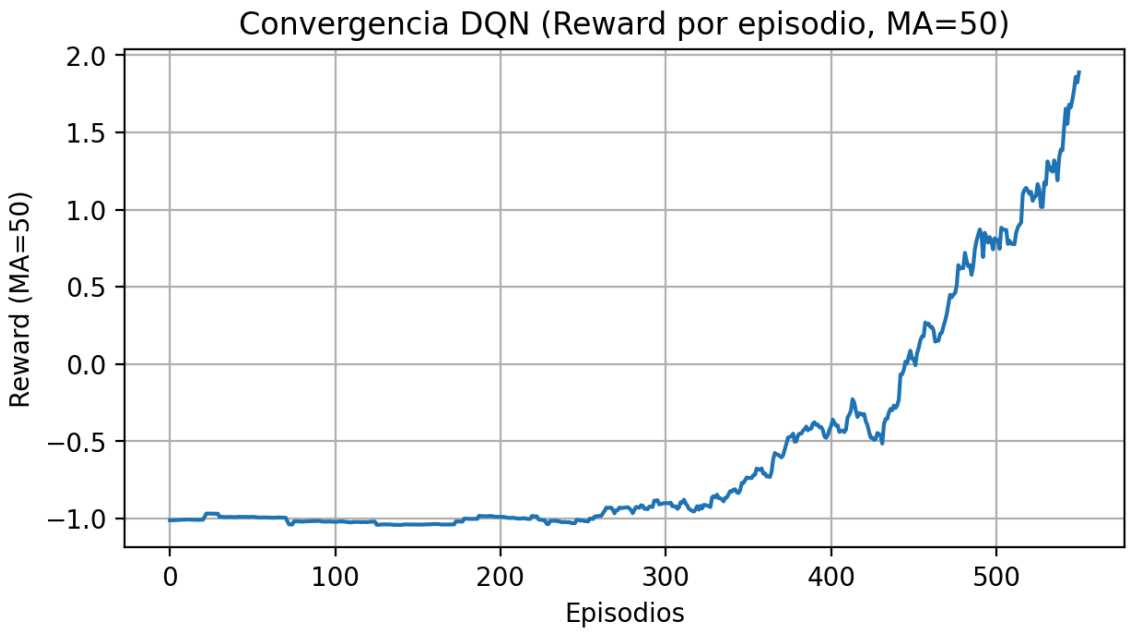
Parámetro	Valor
Grid (H=W)	12 (slider 6–20)
`max_steps`	300
Episodios	600–1000

γ\gamma	0.99
`lr`	1e-3 (Adam)
`batch_size`	64
`buffer_size`	50.000
`start_learn`	1.000 pasos
`target_update_freq`	1.000 pasos
ϵ0\epsilon_0 / ϵmin\epsilon_{\min} / `decay`	1.0 / 0.05 / 0.995
Seed	42

5) Resultados: gráfico de convergencia

Se registra el **reward por episodio** y se grafica su **media móvil** (ventana **50**) para estabilizar la lectura, cumpliendo el requisito de “Reward en función de los episodios”.
En general:

- Al principio predomina el shaping negativo (−0.01 por paso) y muertes (−1-1−1).
- A medida que mejora la política, aparecen **picos positivos** por comidas (+1+1+1) y mayor supervivencia.
- El **MA=50** muestra la tendencia sin ruido.



6) Inconvenientes y cómo se resolvieron

1. **Error MPS (Apple Silicon) con `AdaptiveAvgPool2d((5,5))`**
    - *Síntoma*: `RuntimeError: Adaptive pool MPS: input sizes must be divisible by output sizes`.
    - *Causa*: En MPS, la *adaptive pool* exige que H,WH,WH,W sean múltiplos del tamaño de salida.
    - Solución: cambiar a Global Average Pool 1×1 y ajustar la capa lineal a `Linear(64, 256)`. Alternativamente, un pool dinámico que divida al grid o restringir `grid ∈ {10,15,20}`.
2. **Ejecutar un paso con episodio terminado (modo Manual)**
    - *Síntoma*: `RuntimeError("Episodio terminado; llamá reset().")`.
    - Solución:
        - Deshabilitar los botones de dirección cuando `done=True`.

        - Blindar `do_step`: si `done`, no llamar a `env.step` y mostrar aviso.
        - Al Reset entorno, setear `done=False` para re-habilitar.

3. **Re-render de Streamlit y pérdida de estado**
    - *Síntoma*: variables (modelo/política) “desaparecen” tras interactuar.
    - Solución: usar `st.session_state` para persistir: pesos del DQN, dims de entrada, returns, `done`, `score`, etc.

7) Conclusiones

- El entorno Snake permite validar Deep RL con observaciones tipo imagen y recompensa escasa.
- DQN+CNN aprende políticas razonables con pocos recursos y la convergencia (MA=50) evidencia la mejora.
- Los ajustes de UI/infra (estado persistente, device, pooling) fueron clave para lograr una demo estable e interactiva.

8. Implementación y reproducibilidad

Se proveen scripts, notebook para Colab y videos DEMO en la documentación del repositorio.

Link al colab

Link al colab de snake

Link al repositorio GitHub y DEMO