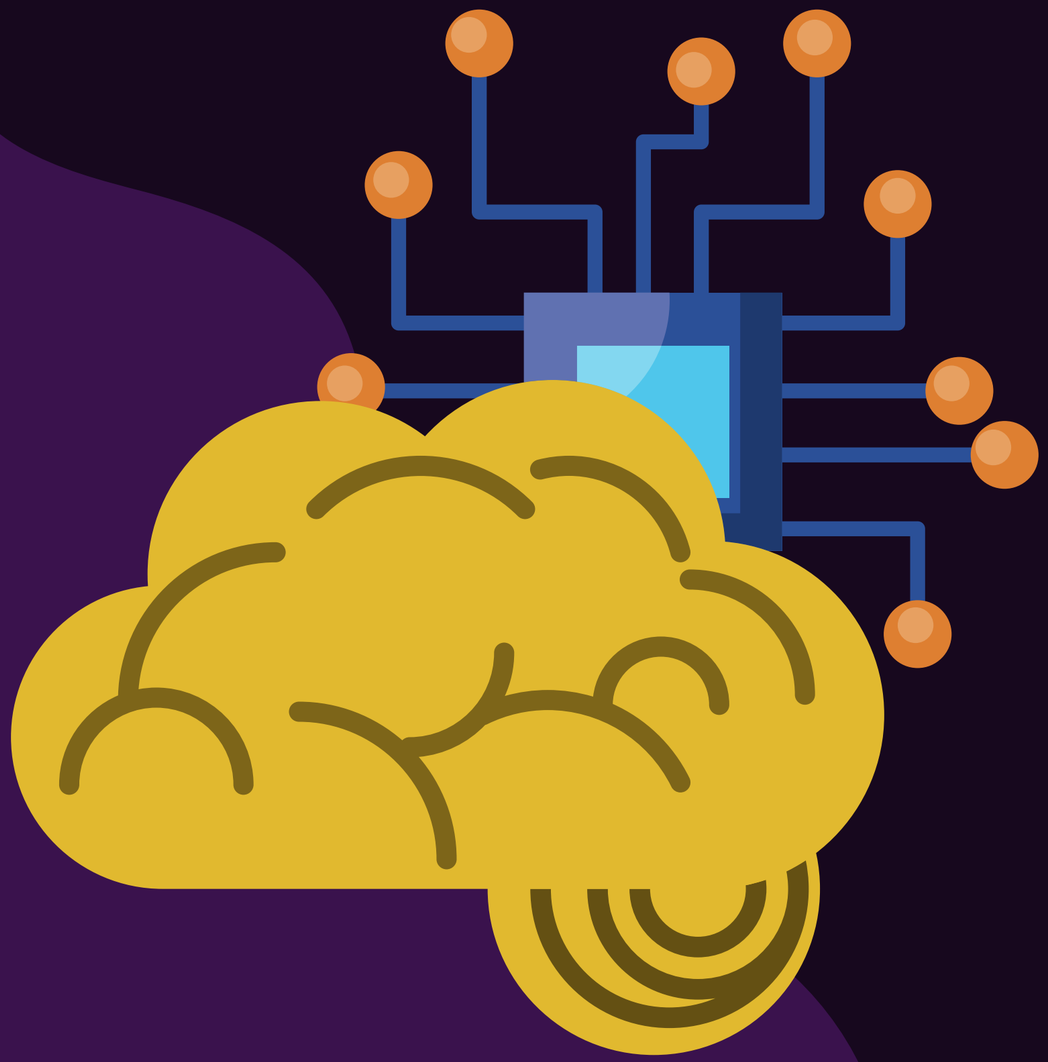


AGENTE CONNECT-4: IMPLEMENTACIÓN, VALIDACIÓN Y OPTIMIZACIÓN

Darek Aljuri y Valentina Ruiz

Objetivos

Diseñar un agente inteligente que juegue Connect-4 de manera autónoma.

→

Aplicar conceptos del curso para ambientes con incertidumbre

→

Mostrar implementación, validación, optimización y capacidad de análisis crítico.

Arquitectura del agente inicial

Tablero

- Matriz 6×7 de NumPy.
- Codificación:
- 1 → jugador amarillo
- -1 → jugador rojo
- 0 → celda vacía

MCTS (UCB) + Tácticas

- **Versión 1:**

MCTS puro con 800 iteraciones → demasiado lento → Gradescope daba timeout.

- **Versión 2:**

MCTS optimizado con 50 iteraciones y sin tácticas → rápido, pero fallaba tácticamente.

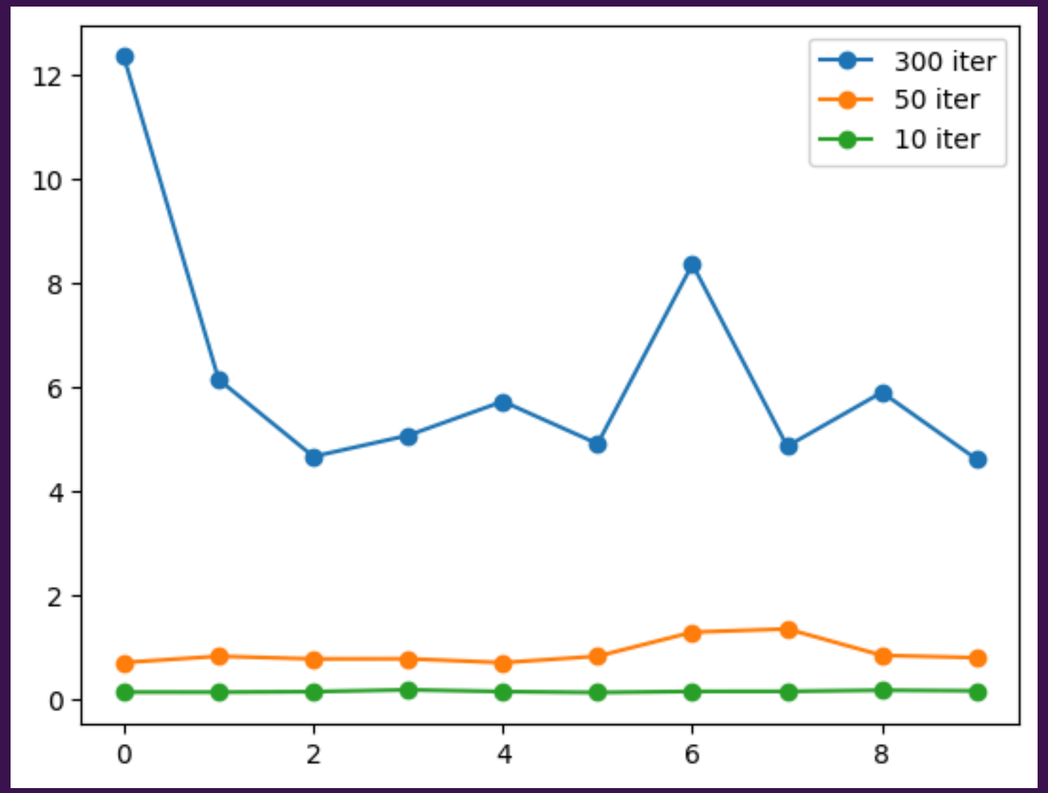
- **Versión 3 (final):**

MCTS (50 iteraciones) + immediate_tactics → rápido, no bloquea jugadas, no pierde por errores básicos.

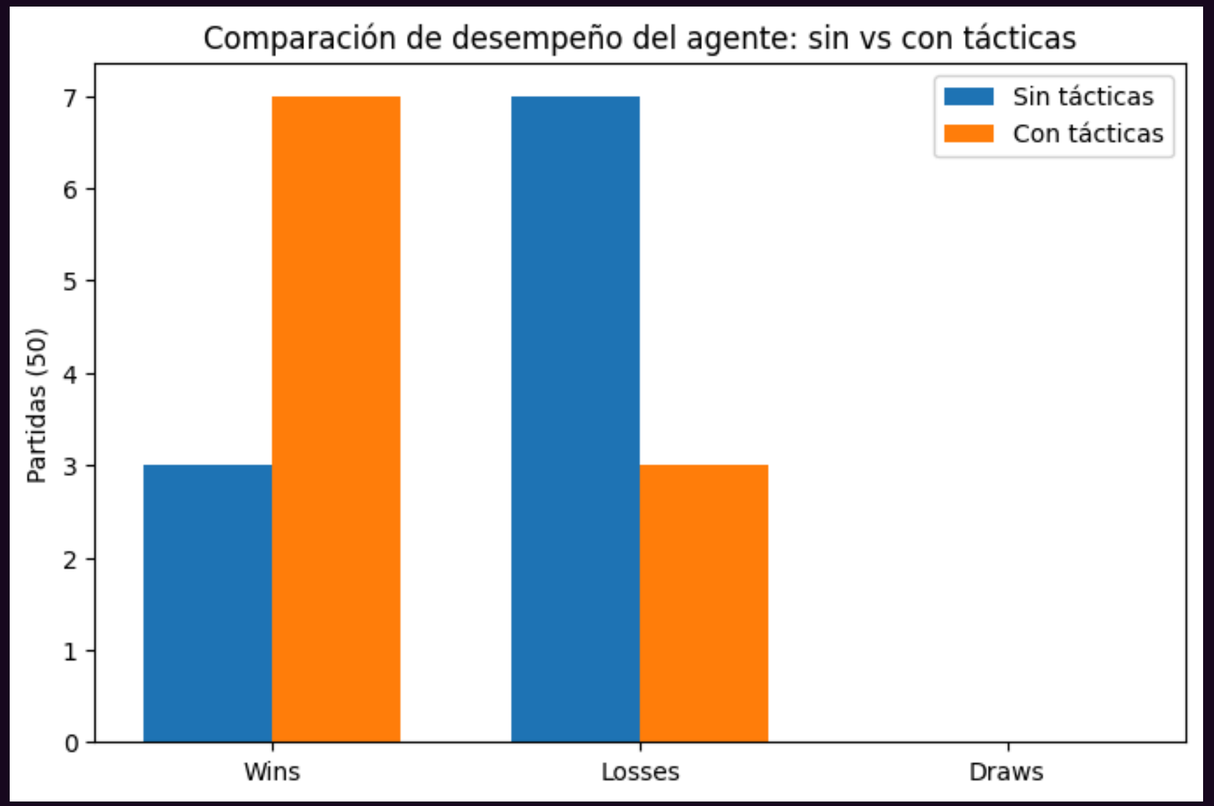
¿Por qué es optimo?

- Funciona bien contra rivales simples, pero no es óptimo porque solo razona a muy corto plazo y no aprende de experiencias pasadas.
- MCTS con tan pocas simulaciones no capta planes profundos ni patrones estratégicos, y los rollouts aleatorios generan decisiones inconsistentes.
- Además, el agente no tiene memoria: cada partida empieza "desde cero", repitiendo errores que ya cometió antes.

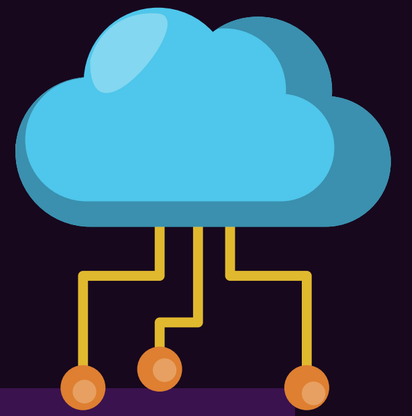
Rendimiento



Desempeño



Decisiones estratégicas + tácticas + aprendizaje real basado en experiencia



1 Política base (tácticas + MCTS)

Antes de usar aprendizaje, el agente usa esta política fuerte:

A. Tácticas inmediatas

B. Si no hay táctica → usa MCTS

2 Representación del estado usando hashing

Cada tablero se convierte en un hash SHA-1

Esto permite representar cualquier tablero como un ID único, y guardar conocimiento sin ocupar mucha memoria.

3 Memoria del agente (Q-table + visitas)

Guardamos aprendizaje en una memoria a largo plazo del agente.

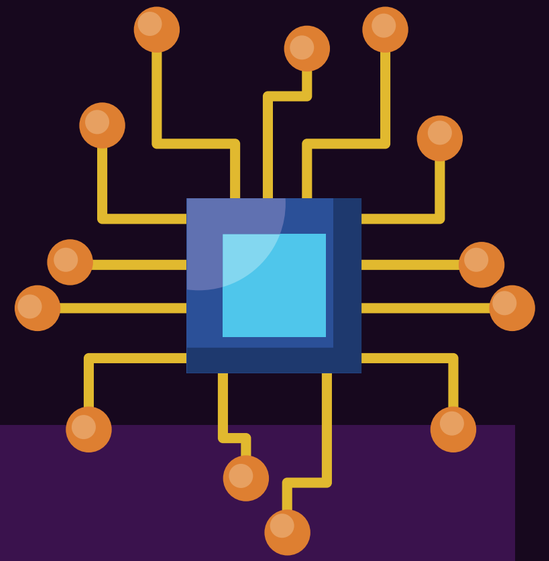
Q((state_hash, action)) = valor aprendido

N((state_hash, action)) = cuántas veces se ha visto

- ***El MCTS se usa cuando no es posible evaluar todos los estados (expansión de árbol inteligente).***
- ***UCB permite seleccionar acciones balanceando exploración y explotación.***

Generalized Policy Iteration:

- Evalúa política con Monte Carlo
- Mejora política usando Q-greedy



4 Aprendizaje Monte Carlo (FVMC)

Cuando termina la partida:

- Ganó → recompensa = +1
- Perdió → recompensa = –1
- Empató → recompensa = 0

Luego recorre el episodio al revés actualizando solo la primera visita

5 Policy Improvement automático (si hay Q la usa)

Si el estado YA existe en la Q-table juega la acción con mayor valor Q (es decir, la acción aprendida)
Elegimos argmax(Q(s,a)),

Si NO existe vuelve a la estrategia base (tácticas + MCTS con selección + expansión + simulación + backpropagation)

Entrenamiento LOCAL

Creamos varios agentes:

8 agentes inteligentes

8 agentes aleatorios

→

Y TODOS comparten la misma Q-table

→

- Los random crean estados raros → el agente aprende a evitarlos

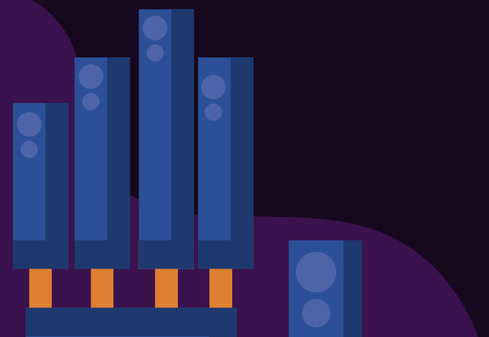
Los inteligentes crean estados profundos → el agente aprende jugadas buenas

- La Q-table crece muchísimo en poco tiempo

Entrenamiento

Q-Table

- **Q[(state_hash, action)]** → almacena el valor aprendido para una acción en un estado específico.
  - El state_hash (SHA-1) representa de forma compacta el tablero completo.
  - El valor Q está en el rango (-1, +1),
- **N[(state_hash, action)]** → almacena cuántas veces se ha visitado ese par (s,a)



FASE 1 — Las Policies (ALjuriRuiz y HelloPolicy) "juegan"

Eligen acciones. El agente inteligente aprende.
El random solo genera estados.

FASE 2 — El módulo training_env (SELF-PLAY + APRENDIZAJE)

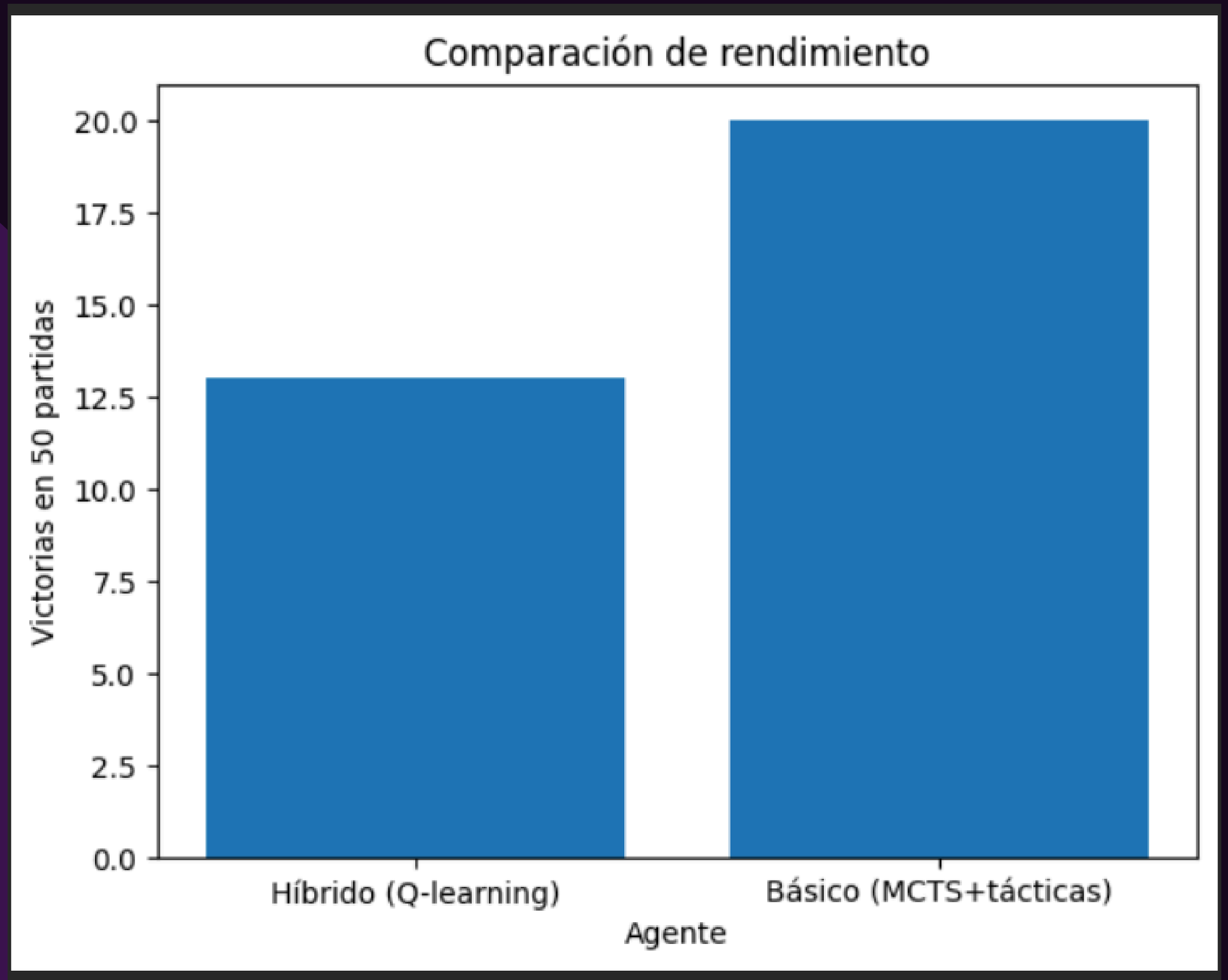
Organiza partidas entre agentes, calcula recompensas.
Actualiza las Q-tables usando Monte Carlo.

FASE 3 — Q-table final

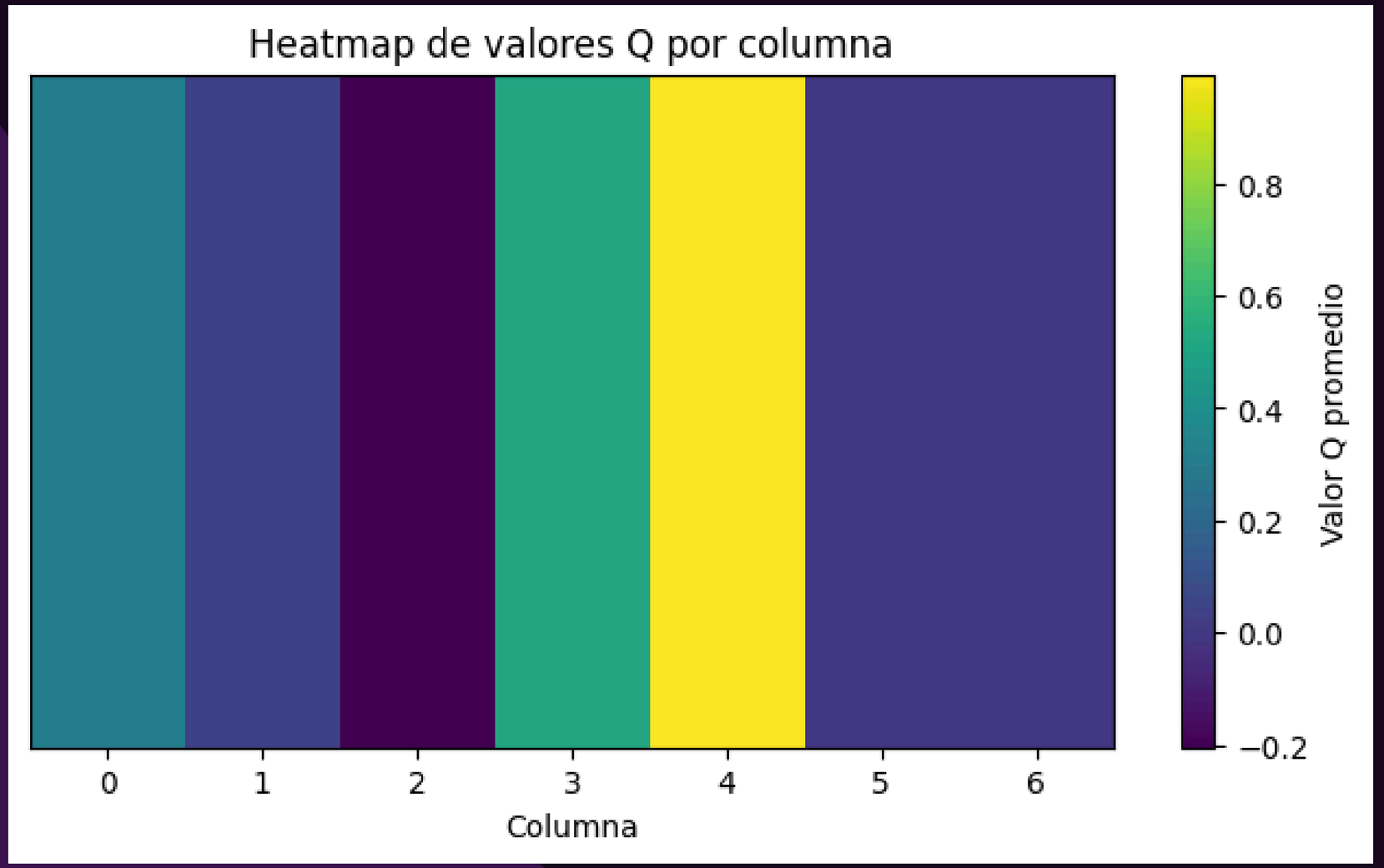
Se guarda el conocimiento aprendido en Q_table.json.

FASE 4 — Evaluation_utils (evalúa)

Carga la Q-table. Simula juegos sin aprender.
Genera métricas y gráficas.



El Q-learning usó FVMC para actualizar la tabla Q con episodios completos, pero como la Q-table se generó a partir de partidas entre un agente aleatorio y un MCTS limitado, parte de los **valores aprendidos resultaron subóptimos.**



Prefirió la columna 4 por ser más ventajosa, mientras que la columna 2 obtuvo valores Q negativos al asociarse con resultados desfavorables.

Limitaciones

Dominancia de estados basura

La Q-table aprendió demasiado de los agentes aleatorios, lo que genera un modelo sesgado.

El hashing no distingue simetrías

Estados equivalentes no se reconocen como iguales, desperdicio de memoria.

1

2

3

4

Aprendizaje lento y poco profundo

Monte Carlo FVMC aprende solo al final del episodio, aprendizaje lento.

La política base sigue siendo mucho más fuerte

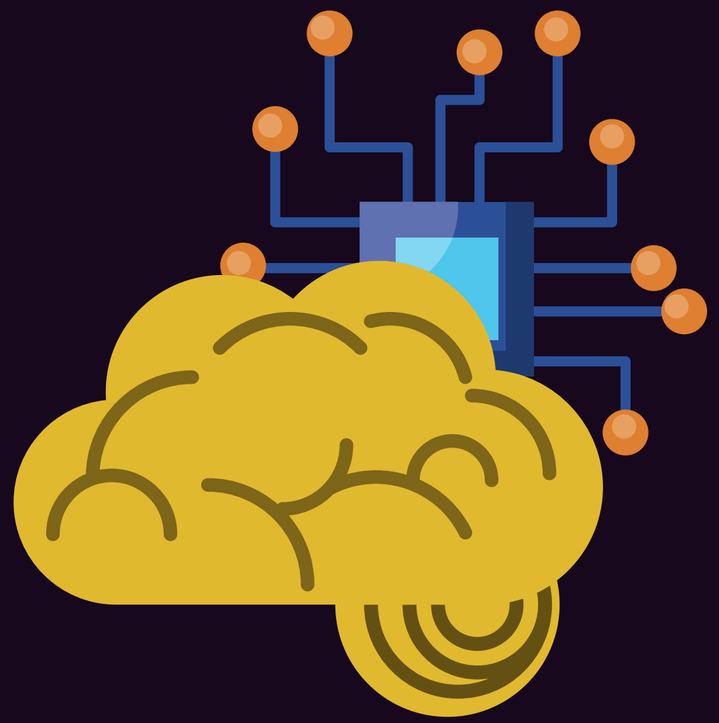
El agente híbrido no logra superar al MCTS (150 iteraciones) + tácticas.

HelloPolicyDefensivo

La versión original era completamente aleatoria

Producía partidas muy cortas, **estados basura** y sin calidad estratégica, la Q-table aprendida eradébil.

"la probabilidad de golpear un estado deseado con trials aleatorios es casi 0"



Trial-Based Online Policy Improvement

Para aprender Q-values útiles la política que genera los trials debe ser “mejor que random“

Implementa dos tácticas simples:

1. Si el oponente puede ganar → bloquear
2. Si él mismo puede ganar → ganar
3. Si no hay peligro → juega random

Produce partidas mucho más largas = más estados en la Q-table.

Genera estados críticos y decisiones reales = Recibe experiencias que valen la pena almacenar

Aumenta el número de estados únicos en la Q-table

Mejora la estabilidad del aprendizaje Monte Carlo

Agente Local con UCB

El agente exploraba MAL (sin UCB)

- NO tenía un sistema de selección equilibrado
- Elegía movimientos aleatorios casi siempre
- Exploraba sin control, lo cual en RL es peligroso porque genera:

Millones de estados inútiles, ruido en la Q-table, aprendizaje lento o inconsistente, no distinguía buenas jugadas nuevas de malas jugadas nuevas

“Exploration without guidance leads to a combinatorial explosion and noisy value estimates.” *(slide Atomic Decisions)*

Aprender el valor Q(s,a) midiendo la recompensa total esperada después de tomar la acción a en el estado s, sin necesidad de conocer el modelo del entorno.

Antes todo estado nuevo recibía un valor completamente aleatorio.
Ahora cada estimación se promedia con muchas partidas anteriores.

método learn() REAL (FVMC)

Implementación entrenamiento local en la policy final

La Policy Final (ALjuriRuiz) integra tres componentes:

1. Usa Memoria aprendida, .json (Q-learning)
2. Tácticas inmediatas (juego reactivo determinista)
3. MCTS fuerte para explorar en estados desconocidos

"**Search should complement learned value estimates, not replace them.**"
Slides Monte-Carlo Tree Search

Uso de Q-learning → Política basada en memoria

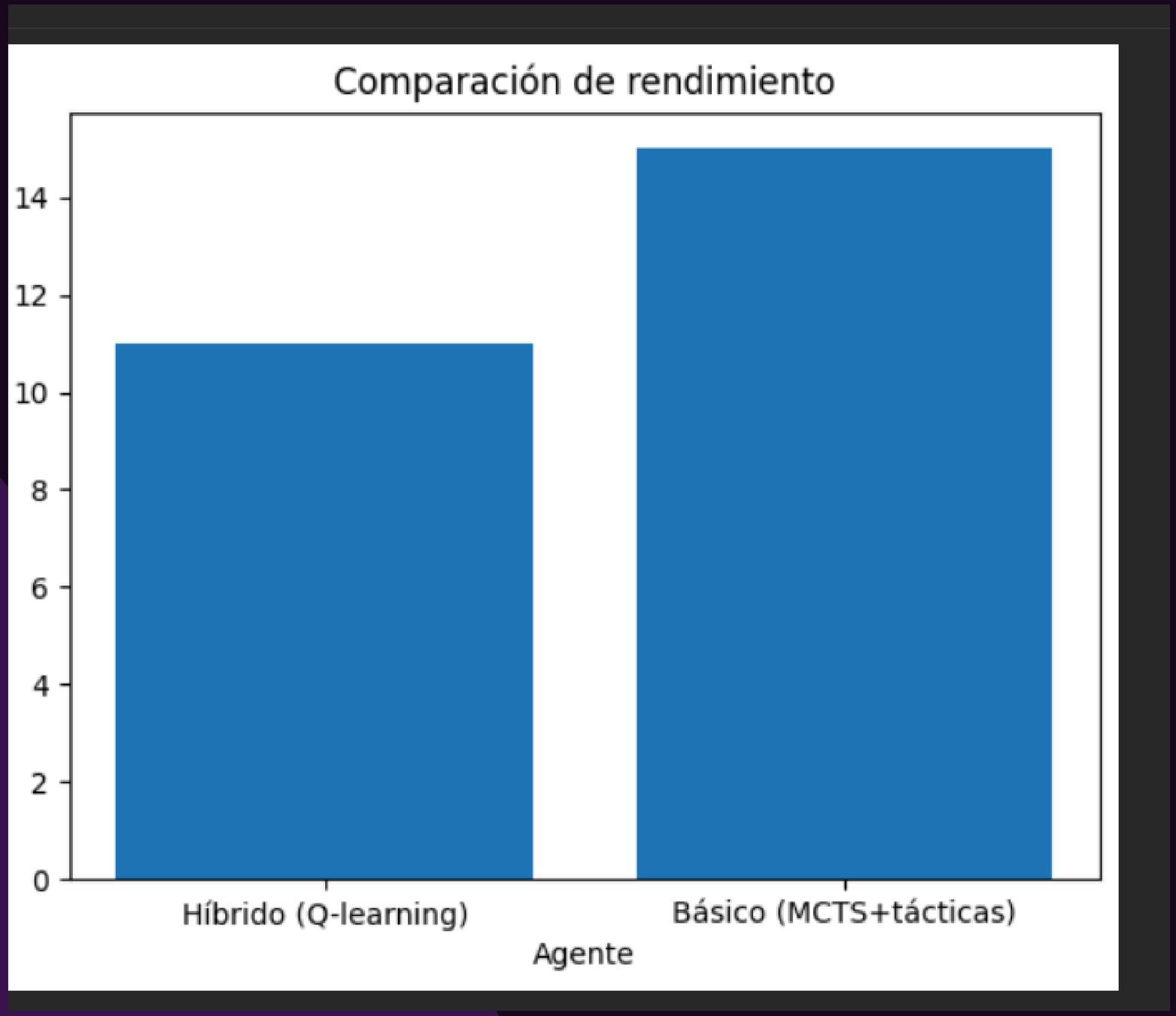
"El objetivo del RL es aprender una función Q(s,a) que guíe decisiones futuras sin necesidad de explorar completamente el árbol."

Tácticas Inmediatas → Decisiones atómicas críticas

Algunas decisiones son tan críticas que no deben ser delegadas a heurísticas o modelos probabilísticos.

MCTS cuando no hay memoria → Exploración guiada por UCB

UCB crea un balance entre exploración y explotación que permite navegar grandes espacios sin expandirlos completamente.



Agente híbrido (Q-learning):
(11, 9, 0)

Agente básico (MCTS+tácticas):
(15, 4, 1)

Mejora Q values

Sistema original

- 8 agentes entrenables usando nuestra política híbrida
- 8 agentes random débiles
- MCTS con solo 10 iteraciones

Esto generaba dos problemas fundamentales:

- Se estaba entrenando sobre un MCTS subóptimo (pocas iteraciones)
- Los Q-values provenían de estados casi nunca visitados

Nuevo sistema

- MCTS "bueno" con 50 iteraciones

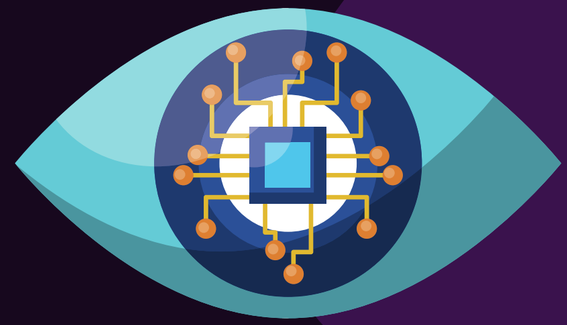
"More iterations reduce bias and improve value estimation"

- Cambio el grupo de entrenamiento

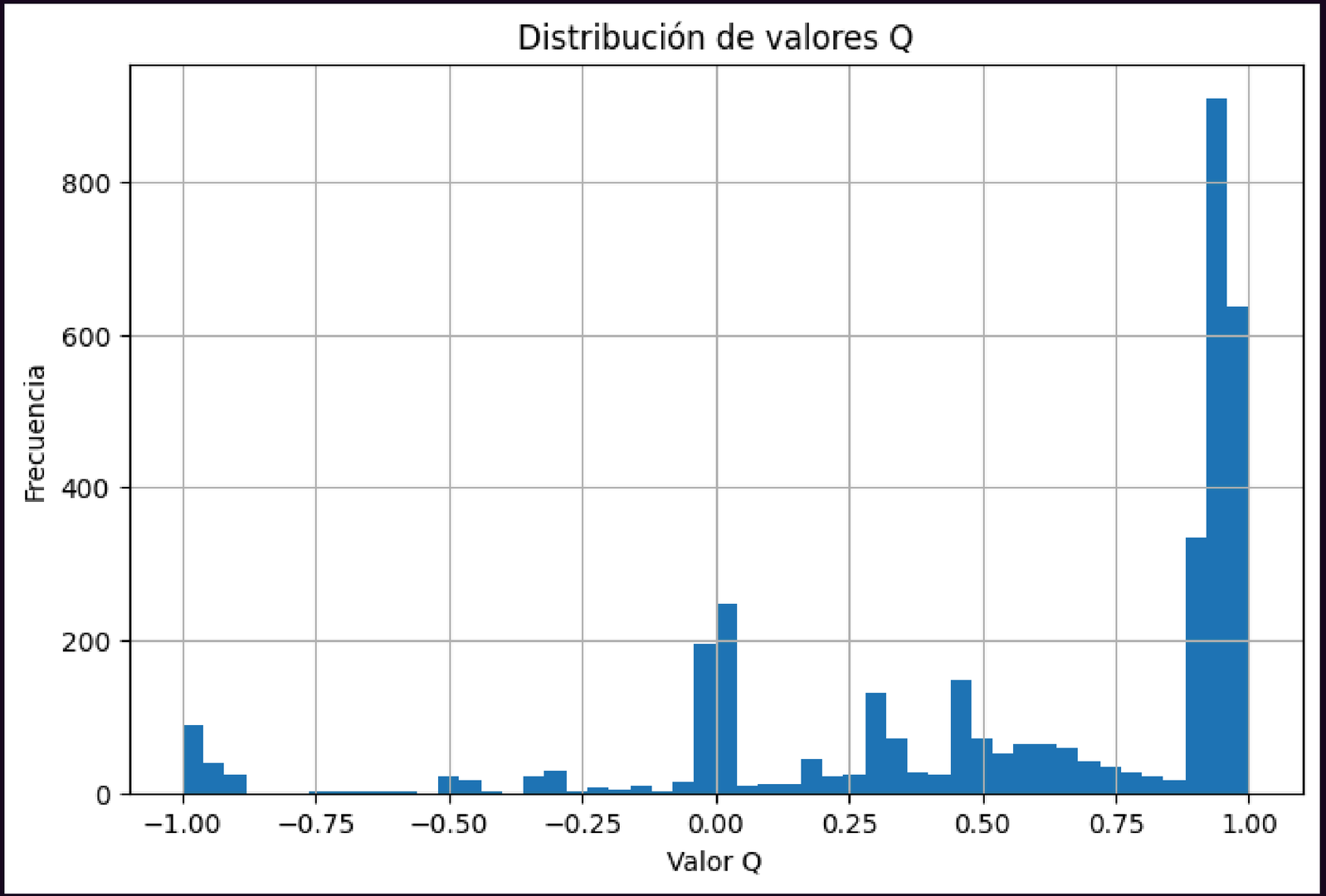
8 entrenables
1 random fuerte defensivo (HelloPolicyDefensivo)
la Q-table deja de llenarse de basura producida por agentes 100% aleatorios
"Competitive environments require meaningful opponents to generate useful trajectories."

- Limpiar la Q-table eliminando ruido (estados con N≤2)

Si un (s,a) apareció ≤ 2 veces → eliminarlo



GRAFICAS Q_TABLE



- Hay picos en Q = 1 y Q = -1, lo cual indica que el agente identificó claramente:

- Estados donde una acción lleva casi siempre a ganar (Q≈1)

- Estados donde una acción lleva casi siempre a perder (Q=−1)

Los valores intermedios (entre −0.5 y +0.5) representan posiciones ambiguas donde la acción no fue determinante o se visitó pocas veces.

- El agente sí logró aprendizaje estable, diferenciando buenas acciones de malas acciones.
- Muestra que el proceso de FVMC + limpieza dejó una tabla Q consistente.

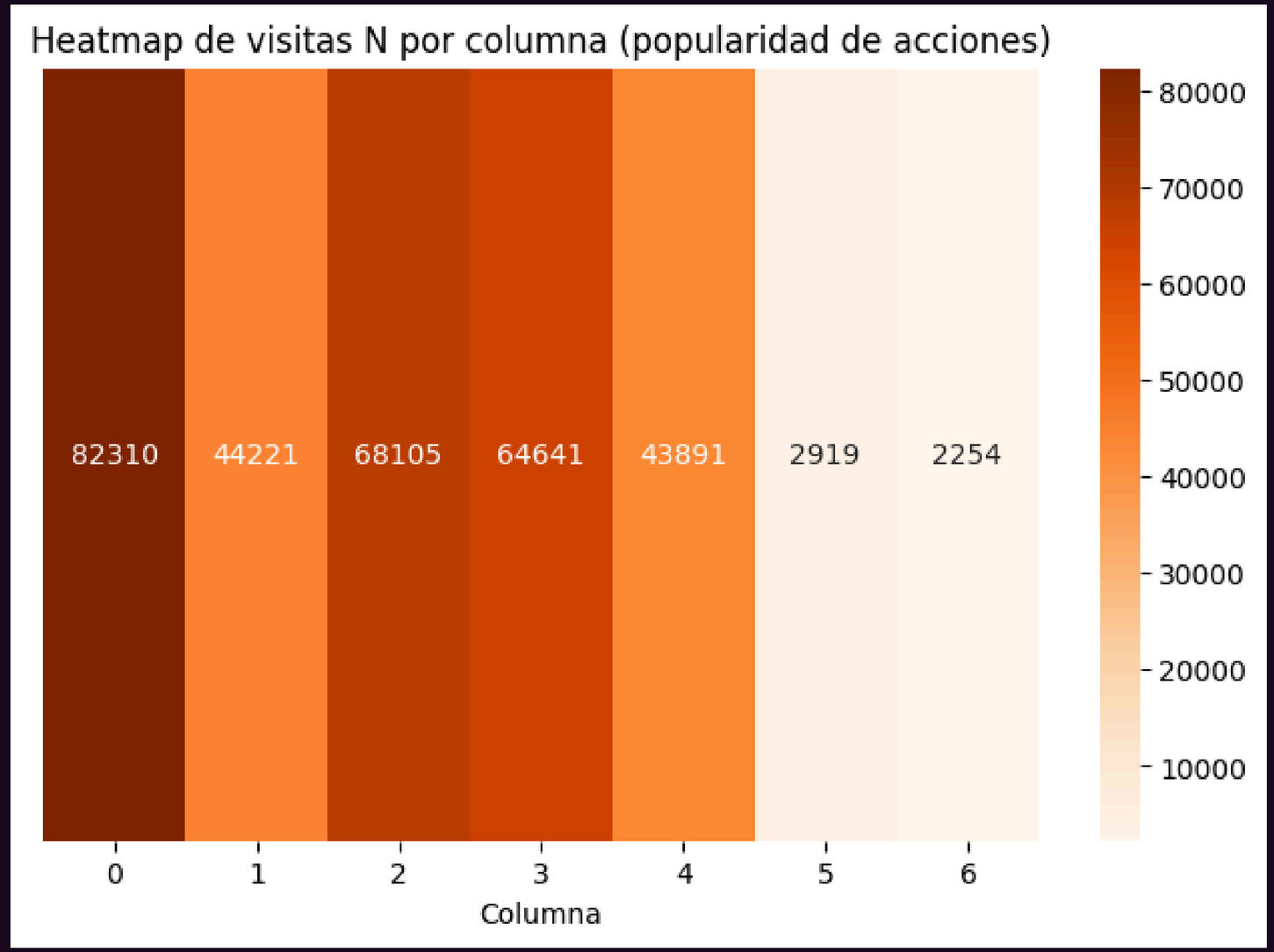
Cuántas veces el agente escogió cada columna

- Las columnas 0, 2, 3 y 4 tienen la mayor cantidad de visitas, lo que coincide con estrategias óptimas en Connect-4:

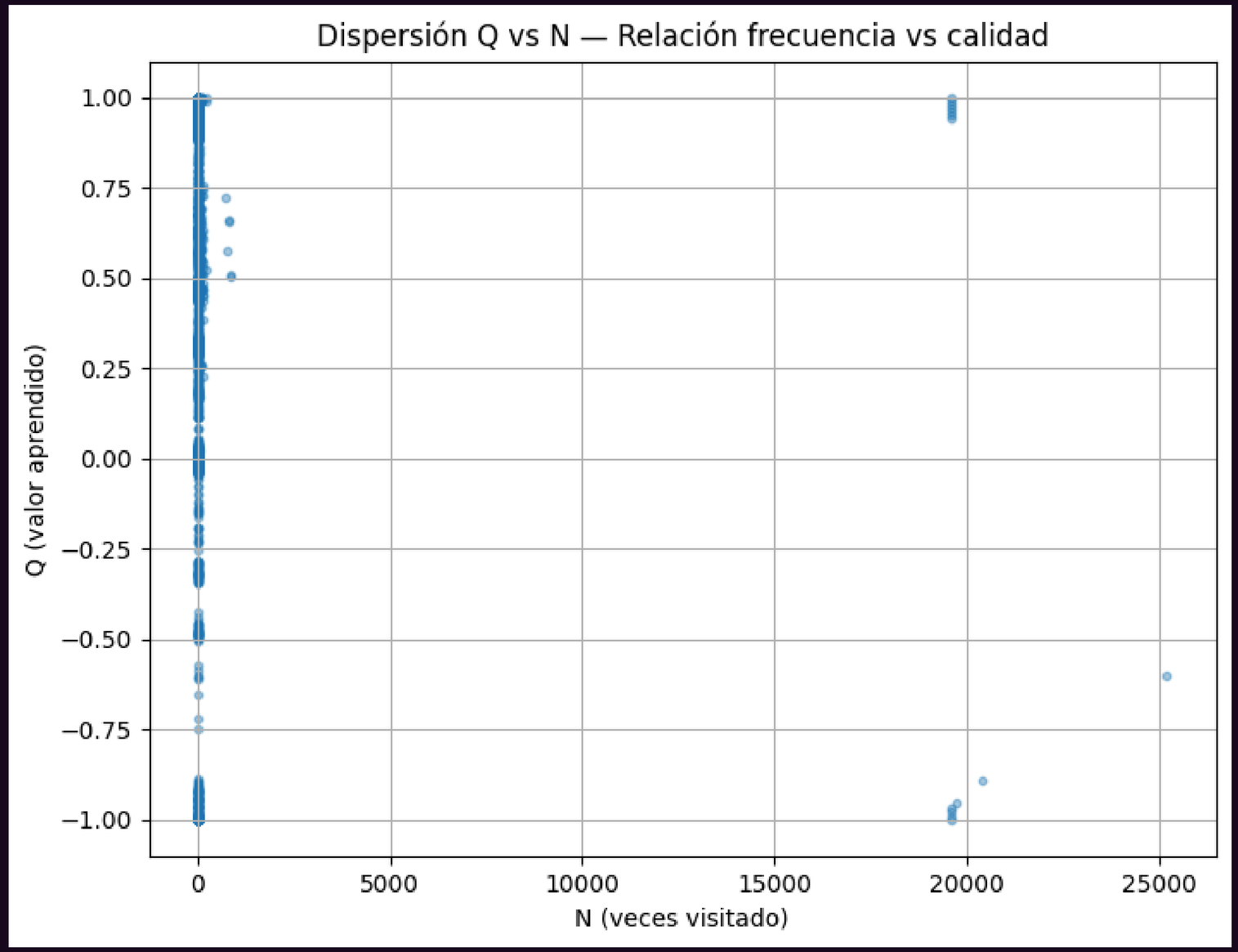
- Las columnas centrales dan más combinaciones ganadoras.

- Las esquinas tienden a generar menos opciones de victoria.

- Las columnas 5 y 6 casi no se visitan



El agente aprendió a evitar acciones malas por experiencia real.
- N confirma que el entrenamiento no fue aleatorio sino guiado por patrones fuertes.



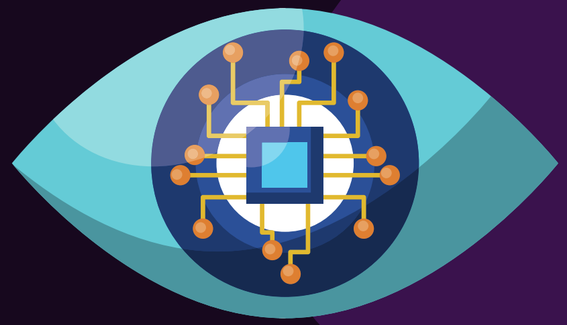
Puntos con N muy alto representan acciones repetidamente evaluadas:

- Si su Q es cercano a +1 = acción altamente confiable.

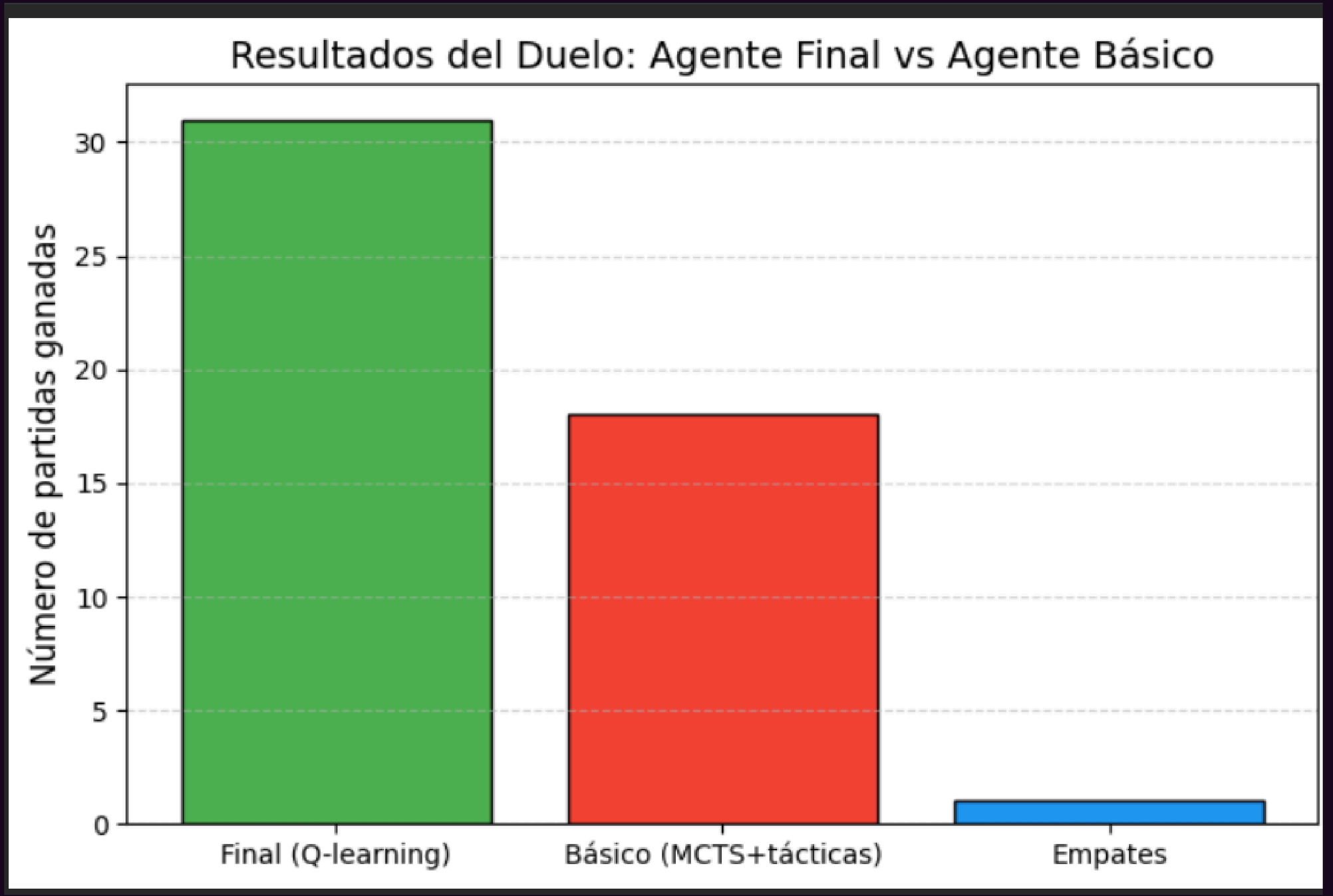
- Si su Q es cercano a –1 = acción que el agente aprendió a evitar.

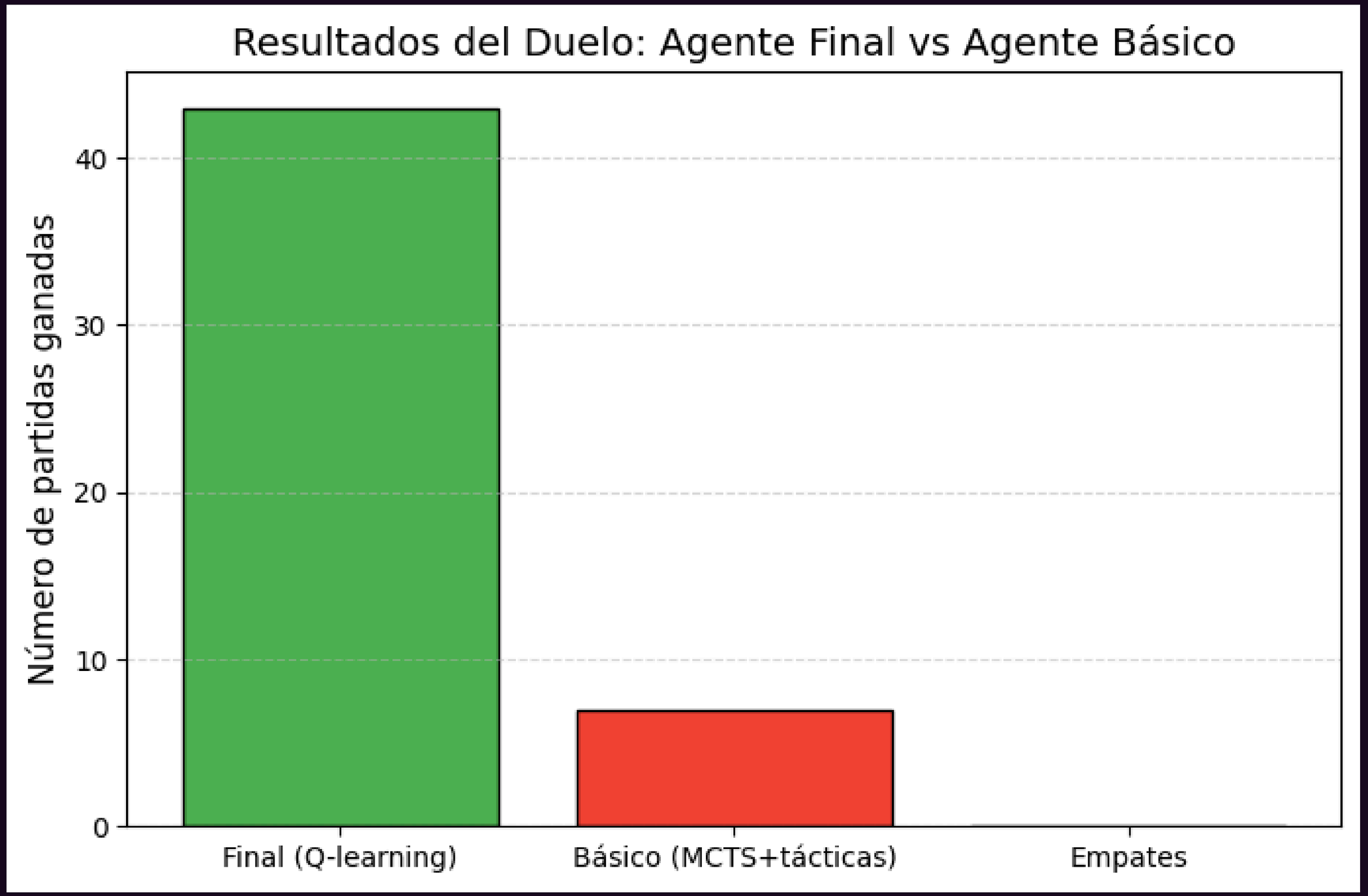
Confirma la propiedad central de Monte Carlo en:
Mientras más se visita un (state, action), más estable es su valor Q

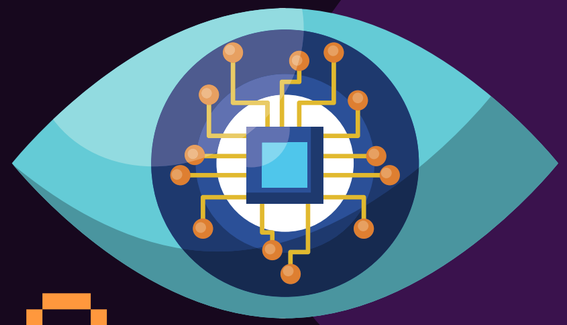
Exploró mucho al inicio (muchos estados con N bajo).
Explotó acciones buenas repetidas veces (N alto con Q cercano a 1).



IMPEMENTACION DE NUEVO Q_TABLES EN POLICY

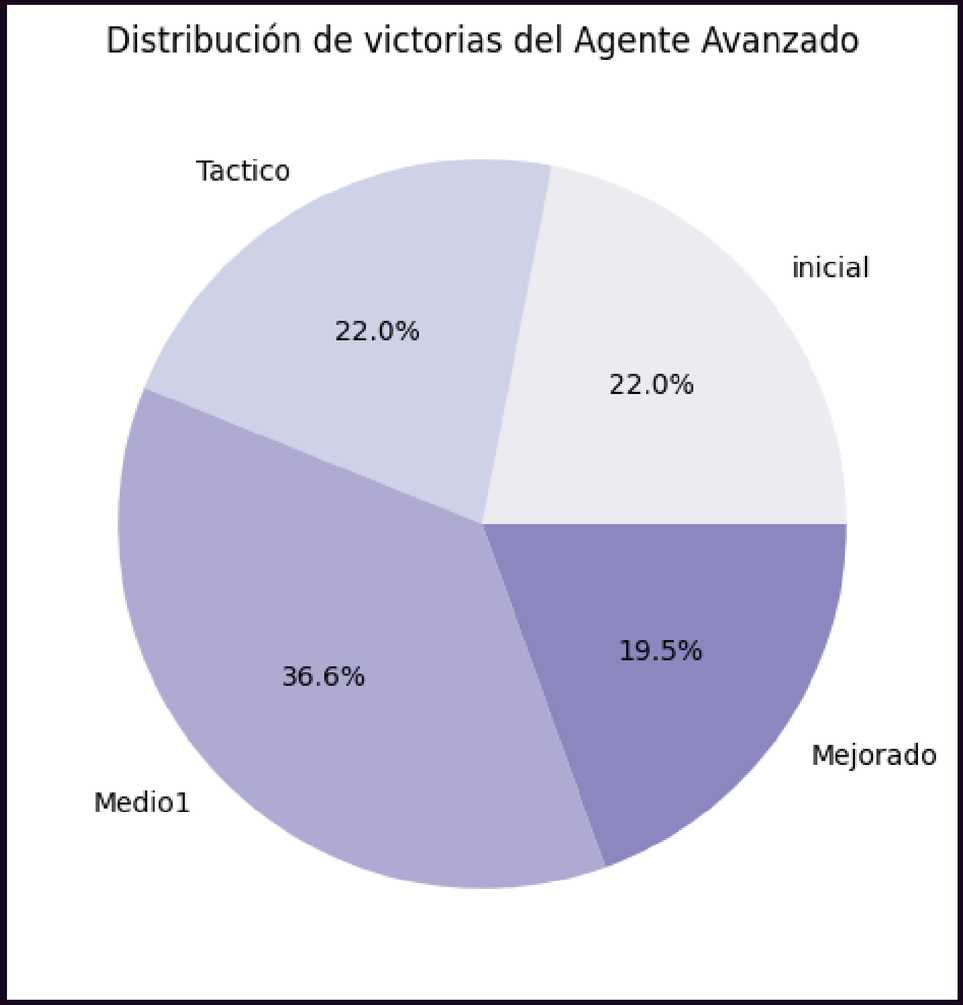

Resultados del Duelo: Agente Final vs Agente Básico
Número de partidas ganadas
0
5
10
15
20
25
30
Final (Q-learning)
Básico (MCTS+tácticas)
Empates

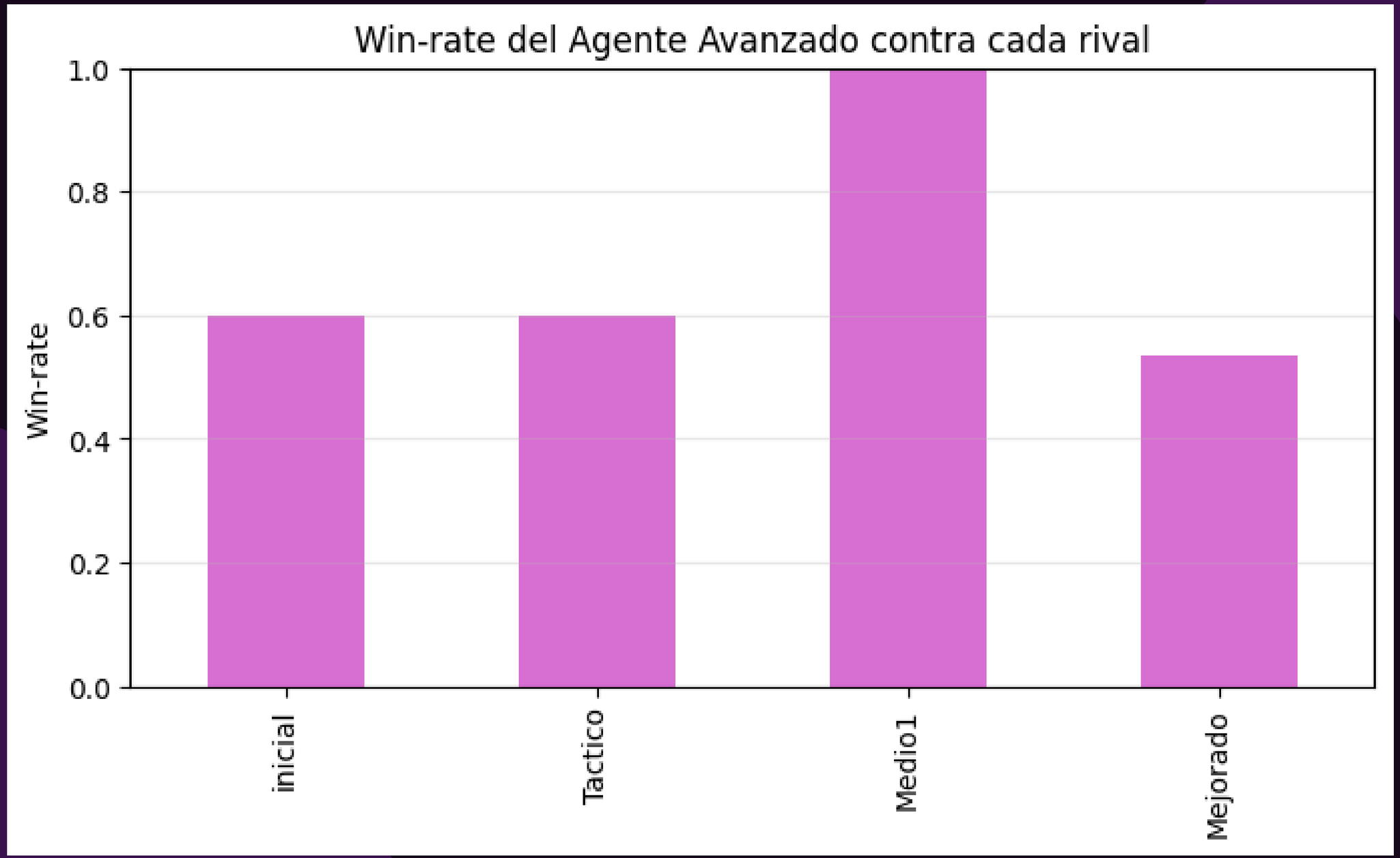

Resultados del Duelo: Agente Final vs Agente Básico
Número de partidas ganadas
0
10
20
30
40
Final (Q-learning)
Básico (MCTS+tácticas)
Empates



AL PONER A COMEPTIR TODOS LOS AGENTES CONTRA EL FINAL

Avanzado vs inicial → 9 - 6 (empates 0)
Avanzado vs Tactico → 9 - 6 (empates 0)
Avanzado vs Medio1 → 15 - 0 (empates 0)
Avanzado vs Mejorado → 8 - 5 (empates 2)
=== TOTAL DE VICTORIAS DEL AVANZADO === 41




Win-rate del Agente Avanzado contra cada rival
Win-rate
1.0
0.8
0.6
0.4
0.2
0.0
inicial
Tactico
Medio1
Mejorado

Como se observa en las gráfica, la policy final es considerablemente superior a todas las demás políticas, superando tanto a las usadas durante el entrenamiento como a la versión inicial.