Facsímil 02

Inteligencia clásica

Capítulo 11

Juegos: decidir con otros actores

686f6c61 · 2026

Capítulo 11: Juegos: decidir con otros actores

Entrando en el tema

Hasta ahora hemos buscado caminos, satisfecho restricciones y construido planes. En todos esos casos el mundo podía ser difícil, pero no necesariamente estaba intentando ganarnos.

Un juego cambia la pregunta. Ya no basta con “¿qué acción me acerca al objetivo?”. Ahora hay que preguntar: “¿qué acción sigue siendo buena cuando otra persona, sistema, regla o instrucción externa reacciona?”.

Esto aparece en sitios muy cotidianos. Un sistema de validación mueve un umbral y los casos límite cambian de forma. Un moderador bloquea una formulación y aparecen rodeos lingüísticos. Un agente con herramientas lee una página web y esa página contiene otra orden que compite con la del usuario. Un competidor responde a tu precio. La distribución no está quieta: aprende de ti.

La teoría de juegos moderna nace con la idea de modelar decisiones interdependientes, popularizada por von Neumann y Morgenstern.[1] En IA, los juegos fueron uno de los laboratorios clásicos para estudiar búsqueda con otros actores que responden. Shannon ya formulaba en 1950 cómo programar una computadora para jugar al ajedrez mediante búsqueda, evaluación y elección de jugadas.[2]

Cuando otro actor también elige

No es una notación que nos inventemos: es la forma estándar de describir un juego secuencial en IA, heredera de la teoría de juegos de von Neumann y Morgenstern y del tratamiento clásico de los juegos como búsqueda. La describimos de forma mínima como una tupla:[3]

$$\mathcal{J} = (S, A, T, u, \tau)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
S	Estados posibles del juego.	Posiciones de un tablero o estados de un flujo.
$A(s)$	Acciones legales en el estado s.	Mover pieza, aprobar, bloquear, escalar.
$T(s, a)$	Transición tras aplicar una acción.	Nuevo tablero o nuevo estado del sistema.
$u(s)$	Utilidad de un resultado.	Ganar $+1$, perder -1, coste alto -5.
$\tau(s)$	Jugador que decide en ese estado.	MAX, MIN, usuario, sistema.

La diferencia con la planificación del capítulo anterior es sutil pero enorme: la transición ya no depende solo de mis acciones. También depende de respuestas.

PROBLEMA	PREGUNTA CENTRAL	RIESGO SI LO OLVIDAS
Búsqueda	¿Cómo llego al objetivo?	Explorar demasiado.
Planificación	¿Qué secuencia es ejecutable?	Saltarte precondiciones.
Juego	¿Qué pasa si alguien responde con un objetivo distinto?	Diseñar solo para el caso feliz.

En un juego de suma cero, lo que MAX gana lo pierde MIN. Muchos productos reales no son suma cero pura, pero el modelo sigue siendo útil como gimnasia mental: obliga a imaginar al actor que persigue un objetivo propio.

Antes de seguir con fórmulas, bajémoslo a escenas reconocibles:

ESCENA	QUÉ ELIGES TÚ	QUÉ ELIGE LA OTRA PARTE	QUÉ ENSEÑA
Trabajo de clase	Escribir una respuesta rápida.	La rúbrica exige justificar pasos.	No optimices solo velocidad.
Agente con correo	Resumir un email.	El email contiene otra orden.	Datos e instrucciones no son lo mismo.
Soporte	Cerrar un ticket.	El usuario vuelve si no resolviste la causa.	El resultado real llega después.
Planificación de turnos	Dar el turno preferido a alguien.	Otra regla exige cubrir una franja crítica.	Hay objetivos que compiten.
Producto con precios	Bajar el precio.	Clientes y competidores reajustan conducta.	Una acción cambia el entorno.

La palabra “juego” no significa pelea. Significa interdependencia: mi decisión modifica tus opciones, y tu respuesta modifica el valor de mi decisión.

Minimax: elegir contra una respuesta buena

Minimax es la versión más limpia de esta idea, y tampoco es nuestra: la garantía de que existe un valor óptimo bajo juego adversario es el **teorema minimax** que von Neumann demostró en 1928, y Shannon la llevó al ajedrez por computadora en 1950. MAX quiere maximizar la utilidad; MIN quiere minimizarla. La función de valor se define recursivamente:[4]

$$V(s) = \begin{cases} u(s) & \text{si } s \text{ es terminal} \\ \max_{a \in A(s)} V(T(s,a)) & \text{si } \tau(s) = MAX \\ \min_{a \in A(s)} V(T(s,a)) & \text{si } \tau(s) = MIN \end{cases}$$

PIEZA	LECTURA SENCILLA	EJEMPLO
Hoja terminal	Resultado ya evaluado.	Victoria, derrota, caso resuelto.
Turno de MIN	La otra parte elige tu peor continuación.	Una instrucción externa compite con la orden principal.
Turno de MAX	Tú eliges la mejor garantía.	Control que sigue bien ante respuestas distintas.
Valor de raíz	Decisión recomendada.	Acción robusta, no acción optimista.

La enseñanza importante no es que todos los otros actores sean perfectos. Es que una decisión no se evalúa sola. Se evalúa por el árbol de respuestas que permite.

Si una acción parece brillante solo cuando nadie reacciona, no era una buena acción: era un deseo.

Ejemplo cercano: un agente puede hacer tres cosas con una herramienta delicada.

ACCIÓN DE MAX	SI TODO VA FÁCIL	SI APARECE UNA INSTRUCCIÓN CONFLICTIVA	VALOR MINIMAX	LECTURA HUMANA
`seguir_automatico`	$+9$	-8	-8	Brilla en el caso cómodo, pero se cae cuando hay tensión.
`pedir_revision`	$+7$	$+3$	$+3$	Más lento, pero razonable.
`limitar_tool`	$+5$	$+4$	$+4$	No es espectacular, pero aguanta mejor.

Minimax elegiría `limitar_tool`: no porque sea la opción más vistosa, sino porque su resultado garantizado es mejor. Esta es la idea que quiero que te lleves: a veces la decisión madura no maximiza el mejor caso, sino que cuida el caso difícil.

Poda alfa-beta: no mirar lo que ya no decide

Minimax exacto puede crecer como una bestia:

$$O(b^d)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
b	Factor de ramificación: acciones por estado.	30 jugadas posibles.
d	Profundidad explorada.	6 turnos.
b^d	Nodos aproximados a explorar.	30^6, demasiado grande.

La poda alfa-beta llega con una idea preciosa: no cambia la respuesta de minimax, cambia cuánto trabajo hace para encontrarla. Knuth y Moore analizaron formalmente esta técnica y su dependencia del orden de exploración.[5]

Mantenemos dos límites:

$$\alpha = \text{mejor valor garantizado para MAX}, \quad \beta = \text{mejor valor garantizado para MIN}$$

Una rama se poda cuando:

$$\alpha \geq \beta$$

CONCEPTO	QUÉ SIGNIFICA	INTUICIÓN
α	MAX ya tiene una alternativa al menos así de buena.	“No acepto menos que esto”.
β	MIN ya puede forzar una alternativa al menos así de mala para MAX.	“La otra parte no me dejará mejorar por aquí”.
$\alpha \geq \beta$	La rama no puede cambiar la decisión.	Cortamos sin perder exactitud.
Orden de acciones	Qué rama miramos primero.	Buen orden produce más poda.

En agentes modernos, esta idea se parece a no gastar herramientas caras, llamadas a modelos o exploraciones que ya no pueden cambiar la decisión. Para podar necesitas límites parciales: coste, riesgo, probabilidad, permisos o valor esperado.

Siguiendo el ejemplo anterior, imagina que ya evaluamos `limitar_tool` y sabemos que garantiza 4. Ese 4 se convierte en α: MAX ya tiene una alternativa aceptable.

Ahora exploramos `seguir_automatico`. La primera respuesta posible nos da -8. Como estamos en turno de MIN, la otra parte puede quedarse con ese -8. Da igual que todavía exista una rama cómoda con $+9$: MIN no tiene por qué escogerla. Esa rama ya no puede superar el 4 garantizado de `limitar_tool`, así que podemos cortarla.

MOMENTO	QUÉ SABEMOS	QUÉ HACEMOS
Ya vimos `limitar_tool`	MAX puede garantizar 4.	$\alpha = 4$.
Entramos en `seguir_automatico`	MIN encuentra una continuación con -8.	$\beta = -8$.
Comparamos	$\alpha = 4 \geq \beta = -8$.	Podamos lo que queda de esa rama.
Resultado	La decisión no cambia.	Ahorramos exploración.

En clase suele costar porque parece contraintuitivo: “¿cómo voy a ignorar una rama que podría tener $+9$?”. La respuesta es que esa rama depende de que MIN quiera ayudarte. Minimax no asume eso.

Modo ingeniero: minimax con poda en código

Toda la sección cabe en una función recursiva. Es uno de los algoritmos más bonitos de la IA clásica: la recursión de minimax con los dos límites α y β que hacen el corte.

```
def minimax(estado, alfa, beta):
    if es_terminal(estado):
        return utilidad(estado)

    if turno(estado) == "MAX":
        valor = float("-inf")
        for a in acciones(estado):
            valor = max(valor, minimax(transicion(estado, a), alfa, beta))
            alfa = max(alfa, valor)
            if alfa >= beta:
                break                  # poda: MIN no dejara llegar aqui
        return valor
    else:                              # turno de MIN
        valor = float("inf")
        for a in acciones(estado):
            valor = min(valor, minimax(transicion(estado, a), alfa, beta))
            beta = min(beta, valor)
            if alfa >= beta:
                break                  # poda: MAX ya tiene algo mejor
        return valor

# se arranca con los limites mas amplios posibles
mejor_valor = minimax(estado_inicial, float("-inf"), float("inf"))
```

Lee el `break`: en cuanto $\alpha \geq \beta$, se dejan de explorar los hermanos restantes, porque la otra parte nunca dejará que esa rama mejore lo ya garantizado. La respuesta es **idéntica** a la de minimax sin poda; solo cambia cuánto trabajo cuesta. Y como anticipamos, el orden importa: si pruebas primero las acciones buenas, α y β se estrechan antes y podas más. Por eso los motores reales ordenan las jugadas antes de explorarlas.

Cuando hay azar: expectimax

Minimax asume que la otra parte siempre elige su peor jugada para ti. A veces eso es demasiado pesimista, porque la otra «parte» no es un adversario, sino el azar: un dado, un usuario impredecible, una API que falla el 5 % de las veces. Para esos casos se usa **expectimax**: donde minimax pondría un `min`, expectimax pone un **valor esperado**, una media ponderada por la probabilidad de cada resultado.[6]

$$V(s) = \sum_{a} P(a)\, V(T(s,a)) \quad \text{en un nodo de azar}$$

En código es el mismo árbol con un tercer tipo de nodo:

```
if turno(estado) == "AZAR":
    return sum(prob(a) * minimax(transicion(estado, a), alfa, beta)
               for a in acciones(estado))
```

La diferencia de mentalidad es grande. Minimax protege contra el peor caso, ideal cuando hay un adversario real (un atacante, un usuario que busca el fallo). Expectimax optimiza el caso promedio, ideal cuando la incertidumbre es estadística y no maliciosa. Confundirlos cuesta caro: tratar el azar como un adversario te vuelve excesivamente conservador, y tratar a un adversario como azar te deja expuesto. La pregunta de diseño es siempre la misma: lo que responde, ¿busca hacerte daño o solo es incierto?

Funciones de evaluación

No siempre podemos llegar al final del árbol. En ajedrez, en Go, en un flujo de revisión o en un agente con herramientas, la profundidad útil se acaba antes que el mundo. Entonces cortamos a una profundidad dada y puntuamos el estado intermedio con una **función de evaluación**. La forma más clásica, la que ya proponía Shannon para programar el ajedrez, es una combinación lineal de señales del estado, aunque en un sistema real esas señales, pesos y escala se aprenden, se ajustan o se validan contra partidas, simulaciones o decisiones históricas:[7]

$$\text{Eval}(s) = \sum_{i} w_i \phi_i(s)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$\phi_i(s)$	Señal observable del estado.	Riesgo, coste, evidencia, satisfacción.
w_i	Peso de esa señal.	Seguridad pesa más que velocidad.
$\text{Eval}(s)$	Puntuación aproximada del estado.	Estado prometedor o peligroso.

La función de evaluación es conocimiento del dominio convertido en número. Ahí está su poder y su peligro: si eliges señales pobres, el algoritmo optimiza una caricatura del problema. Si eliges señales medibles y las validas contra casos reales, la evaluación se vuelve una pieza de ingeniería, no una opinión.

DOMINIO	SEÑALES ÚTILES	ERROR TÍPICO
Ajedrez	Material, movilidad, seguridad del rey.	Capturar material y dejar mate.
Soporte	Resolución, satisfacción, coste, riesgo.	Cerrar tickets rápido sin resolver.
RAG	Relevancia, evidencia, cita, actualidad.	Respuesta fluida sin soporte.
Seguridad	Impacto, probabilidad, detectabilidad.	Optimizar falsos positivos y dejar casos importantes fuera.

Si premias mal, el sistema buscará bien lo equivocado. Esa frase conviene tenerla muy subrayada.

Una forma de empatizar con esto es pensar en una rúbrica. Si en un examen solo puntúas que el resultado final sea correcto, alguien puede acertar por casualidad y parecer competente. Si también puntúas pasos, unidades, justificación y límites, la evaluación se parece más a lo que realmente querías medir.

SISTEMA	SEÑAL QUE PARECE BUENA	LO QUE FALTABA MEDIR
Agente de soporte	Ticket cerrado rápido.	Reapertura, satisfacción y evidencia.
Respuesta RAG	Texto convincente.	Citas, actualidad y trazabilidad.
Moderación	Pocas revisiones manuales.	Casos dudosos que el sistema dejó pasar.
Agente con tools	Tarea completada.	Coste, permisos y reversibilidad.

La función de evaluación siempre educa al sistema. Si educas con una señal pobre, no te sorprendas de recibir un comportamiento pobre con buena presentación.

Monte Carlo: decidir simulando

Monte Carlo acepta una concesión: quizá no puedo calcular el árbol completo, pero puedo simular muchas trayectorias y estimar el valor medio de una acción. El método de Monte Carlo, que Metropolis y Ulam formalizaron en 1949, hace justo esto: estima un valor difícil de calcular mediante el promedio de muchas muestras aleatorias.[8] El valor estimado de una acción es la media de los retornos observados:

$$\hat{V}(a) = \frac{1}{n}\sum_{i=1}^{n} R_i$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
a	Acción que estamos estimando.	Limitar una herramienta.
R_i	Retorno observado en la simulación i.	Resultado de un caso simulado.
n	Número de simulaciones.	100, 1 000, 10 000.
$\hat{V}(a)$	Valor medio estimado.	Calidad esperada de la acción.

La incertidumbre de esa media baja despacio. Es el **error estándar** de la media muestral, una ley básica de la estadística: la precisión crece con la raíz del número de muestras, no con el número de muestras.

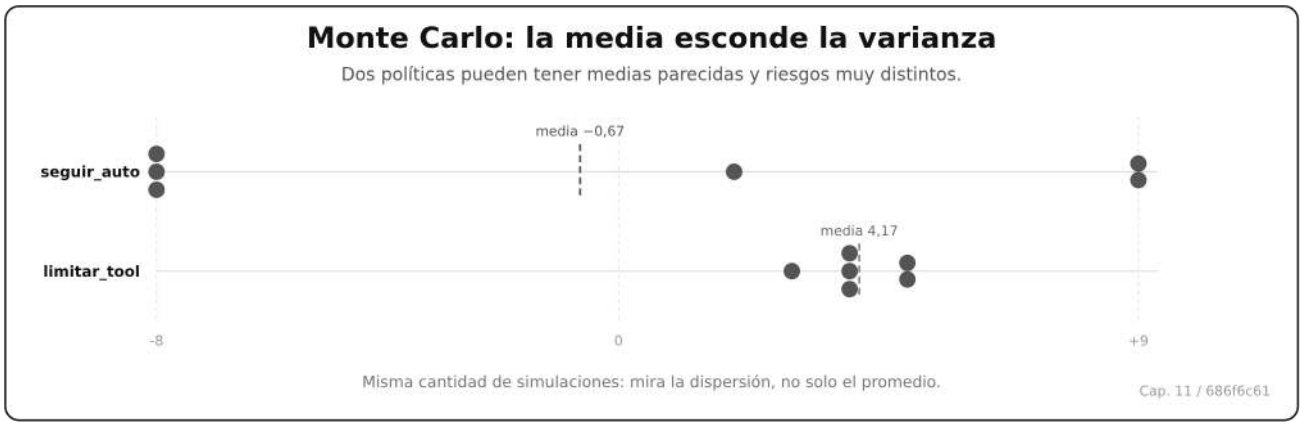
$$\text{error} \approx \frac{\sigma}{n}$$

Cuadruplicar simulaciones no divide el error entre cuatro; lo divide aproximadamente entre dos. Esta humildad estadística es importante. Monte Carlo no convierte un simulador malo en verdad: solo estima bien lo que el simulador representa.

Ejemplo: antes de permitir que un agente ejecute una herramienta, simulas conversaciones con tres políticas.

POLÍTICA	SIMULACIONES OBSERVADAS	MEDIA	LECTURA
`limitar_tool`	4, 5, 4, 3, 4, 5	4,17	Bastante estable.
`seguir_automatico`	9, -8, -8, 2, -8, 9	-0,67	Puede ir muy bien o muy mal.
`pedir_revision`	3, 7, 3, 6, 3, 7	4,83	Buena media, con coste operativo.

Monte Carlo te ayuda a ver distribución, no solo una historia. Si solo miras el primer 9, `seguir_automatico` parece brillante. Si miras varias trayectorias, aparece la fragilidad.



MCTS: explorar y explotar

Monte Carlo Tree Search añade una pregunta: si tengo presupuesto limitado, ¿dónde simulo?

La respuesta clásica es equilibrar explotación y exploración. Kocsis y Szepesvári propusieron UCT, que aplica ideas de bandidos multi-brazo al árbol de búsqueda.[9] Browne y colaboradores ofrecen una revisión amplia de MCTS y sus variantes.[10]

Una forma habitual de seleccionar acción es:

$$UCT(s,a) = Q(s,a) + c \quad \frac{\ln N(s)}{N(s,a)}$$

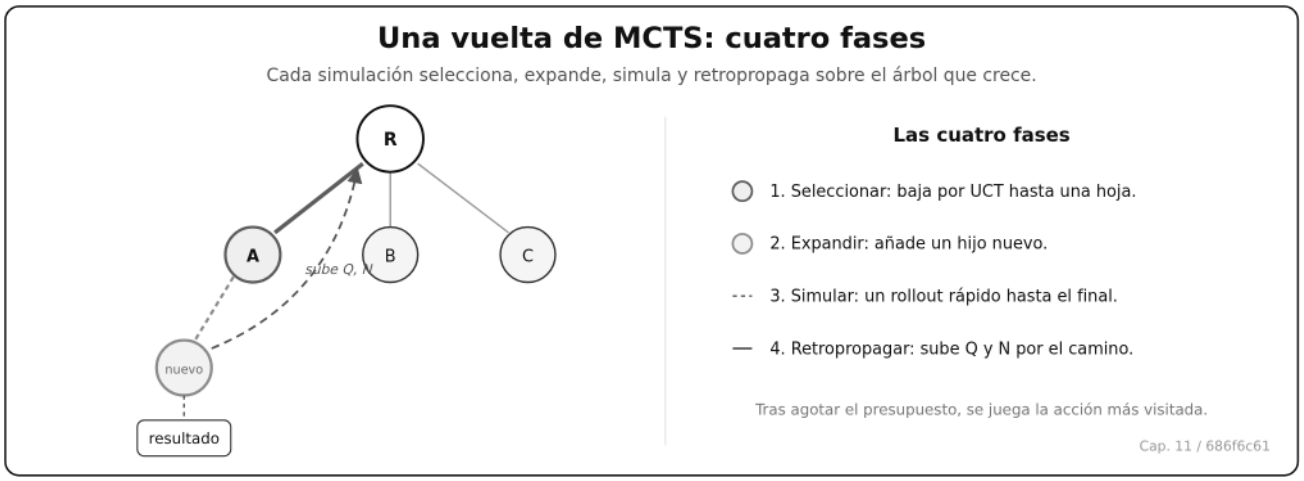
SÍMBOLO	SIGNIFICADO	INTUICIÓN
$Q(s,a)$	Valor medio observado de la acción.	Explotar lo que va bien.
$N(s)$	Visitas al estado.	Cuánto sabemos del nodo padre.
$N(s,a)$	Veces que probamos esa acción.	Cuánto sabemos de esa rama.
c	Peso de exploración.	Más alto: más curiosidad.

El primer término explota: elige acciones con buena media. El segundo explora: empuja ramas poco visitadas para no casarnos demasiado pronto con la primera señal buena.

Las cuatro fases de MCTS

UCT decide a dónde mirar, pero ¿cómo se construye el árbol? MCTS repite, vuelta a vuelta, cuatro fases sobre un árbol que va creciendo:

FASE	QUÉ HACE
Seleccionar	Desde la raíz, baja por el árbol eligiendo en cada nodo la acción de mayor UCT, hasta llegar a un nodo que aún no está expandido del todo.
Expandir	Añade un hijo nuevo: una acción que todavía no estaba en el árbol.
Simular	Desde ahí juega un *rollout* rápido hasta un final, a menudo con jugadas al azar, y obtiene un resultado.
Retropropagar	Sube ese resultado por el camino recorrido, actualizando $Q(s,a)$ y $N(s,a)$ de cada nodo visitado.



Cada vuelta gasta una simulación y afina un poco las estimaciones; cuanto más presupuesto, más fiable es el árbol. La gracia es que el propio UCT concentra las vueltas donde hay señal sin abandonar del todo lo poco explorado. Cuando se agota el presupuesto, se juega la acción de la raíz más visitada, que suele coincidir con la de mejor valor. Así jugó AlphaGo: MCTS guiado por redes neuronales, no fuerza bruta.

En una clase, esto se parece a preparar un examen. Si siempre estudias el tema que ya te sale bien, explotas. Si dedicas algo de tiempo al tema que apenas has mirado, exploras. MCTS formaliza ese equilibrio: profundiza donde hay señal, pero reserva presupuesto para descubrir sorpresas.

SITUACIÓN	EXPLOTAR	EXPLORAR
Agente con tools	Probar más la política que ya funciona.	Revisar una tool poco usada pero crítica.
Producto	Optimizar el flujo con mejor conversión.	Probar una variante nueva con pocos datos.
Evaluación	Añadir casos parecidos a los conocidos.	Buscar casos límite que no aparecen en la muestra.
Estudio	Repasar lo que dominas.	Entrenar el punto que todavía te incomoda.

Por eso c importa tanto en UCT. Si c es pequeño, el sistema se queda cerca de lo que ya parece bueno. Si c es grande, se permite investigar más. No hay valor universal: depende del coste de equivocarte y de cuánto te duela no descubrir una opción mejor.

Simular con LLMs no es lo mismo

Un LLM puede generar escenarios, usuarios sintéticos, instrucciones alternativas o diálogos de prueba. Eso es útil. Pero no debemos confundir plausibilidad textual con muestra estadística de un proceso real.

USO DE LLM	SÍ APORTA	NO DEMUESTRA
Pruebas de tensión	Ideas de casos límite y variaciones de instrucción.	Frecuencia real de esos casos.
Producto	Objeciones y casos límite.	Conversión esperada.
Soporte	Tickets sintéticos para ampliar criterios.	Distribución real de incidencias.
Evals	Casos iniciales para cubrir huecos.	Calidad final sin datos reales.

La regla práctica: usa el LLM para descubrir hipótesis; usa datos, experimentos, revisión humana o simuladores formales para justificar decisiones.

Decisión con instrucciones en tensión

La conexión con agentes modernos es directa. Un sistema que llama herramientas tiene superficie de interacción. Documentos recuperados, páginas web, tickets, correos o entradas de usuario pueden contener instrucciones que compiten con la tarea principal.

Fecha de corte: 10 de junio de 2026. En esta sección uso OWASP 2025 como marco de riesgos vigente para aplicaciones con LLM, pero lo importante para este capítulo no es memorizar una lista concreta: es aprender a modelar instrucciones externas, permisos, presupuesto y acciones excesivas como respuestas posibles dentro del árbol de decisión.

OWASP incluye riesgos específicos de aplicaciones con LLM, como instrucciones no confiables, exposición de datos, uso inseguro de salidas y acciones excesivas de agentes.[11] Visto desde juegos, eso significa que una instrucción externa no es ruido: es otra fuerza dentro del sistema.

CONTROL	PREGUNTA DE TENSIÓN
Permisos mínimos	¿Qué pasa si el modelo intenta una herramienta que no debería?
Separar datos e instrucciones	¿Un documento recuperado puede dar órdenes al agente?
Límites de presupuesto	¿Pueden forzar llamadas infinitas o caras?
Trazas	¿Verás una desviación antes de que tenga efecto real?
Pruebas de tensión continuas	¿Tu eval incluye casos nuevos o solo los de lanzamiento?

Puente hacia aprendizaje por refuerzo

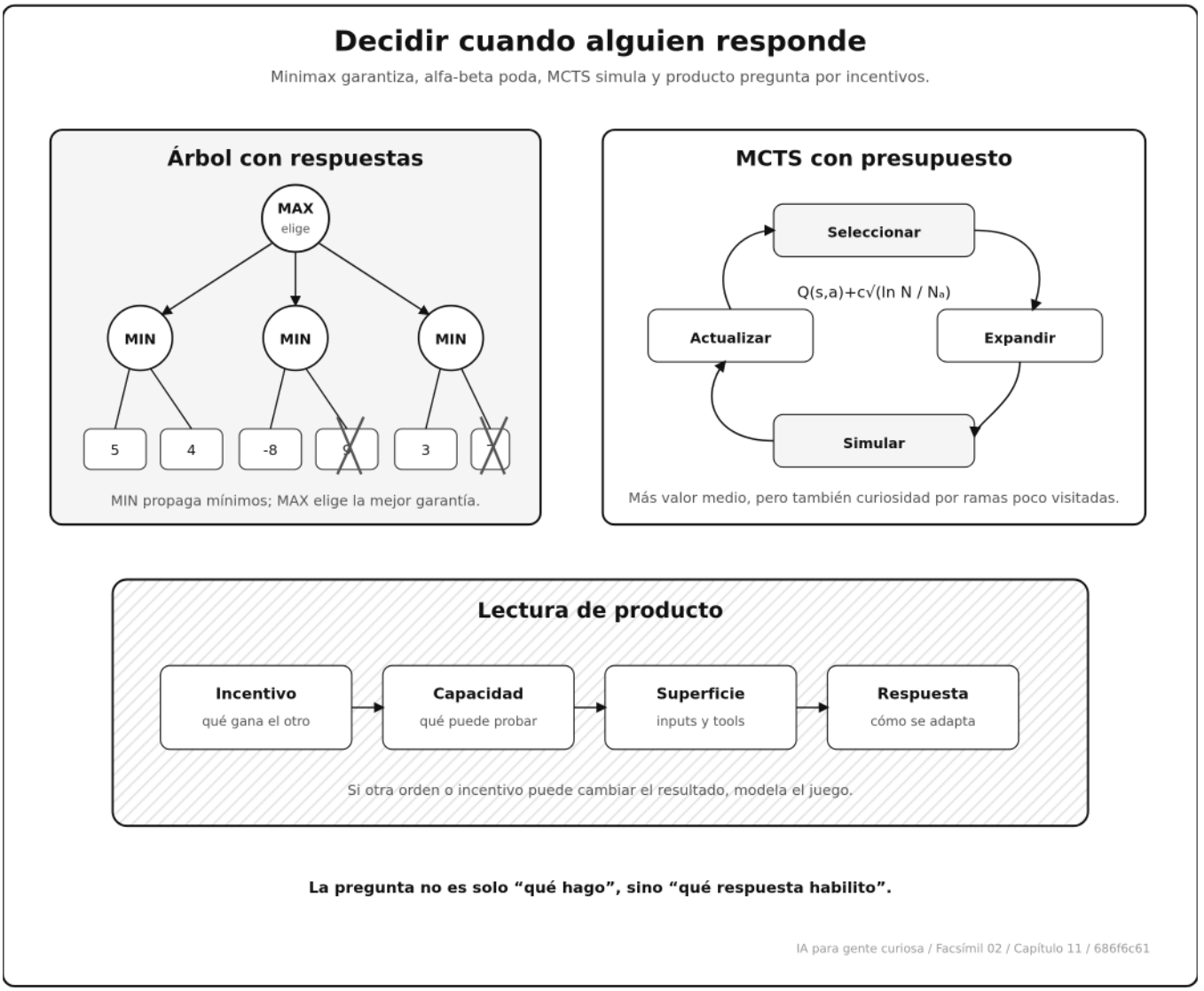
Juegos, MCTS y aprendizaje por refuerzo comparten una pregunta: qué acción conviene ahora para mejorar el valor futuro.

Sutton y Barto formulan el retorno como acumulación de recompensas, normalmente descontadas.[12]

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
G_t	Retorno desde el tiempo t.	Valor futuro de una política.
r_{t+k+1}	Recompensa futura.	Éxito, coste, seguridad, satisfacción.
γ	Factor de descuento.	Cuánto importa el futuro lejano.
Política	Regla para elegir acciones.	Modelo + reglas + routing + permisos.

Esto prepara el terreno para capítulos posteriores: no evaluaremos solo respuestas aisladas, sino comportamiento a lo largo de una trayectoria.



En el día a día

La mentalidad de juegos cambia cómo diseñamos sistemas con IA.

SITUACIÓN	LECTURA DE JUEGO	CONTROL ÚTIL
Usuario bordea una política.	MIN representa la respuesta que presiona tu regla.	Evals de tensión y trazas.
Agente lee contenido externo.	El documento puede traer otra instrucción.	Separar instrucciones de datos.
Tool cara o irreversible.	Una entrada externa puede forzar coste o efecto no deseado.	Presupuesto, permisos y aprobación humana.
Métrica fácil de manipular.	El sistema optimiza el marcador, no el objetivo.	Métricas compuestas y revisión.

Cuando una decisión importante depende de cómo responde otra parte, diseña como si estuvieras jugando una partida. No por paranoia: por respeto a la realidad.

Por qué debería importarte

Porque muchos fallos de IA no ocurren en el primer uso feliz. Ocurren cuando alguien descubre cómo responde el sistema y empieza a optimizar contra esa respuesta.

Si tu agente siempre confía en el documento recuperado, el documento se vuelve un canal de instrucciones. Si tu moderador solo detecta palabras obvias, los rodeos cambian de forma. Si tu eval premia rapidez, el agente aprende atajos. Si tu política no limita presupuesto, una conversación puede acabar en una acción costosa.

Los juegos enseñan a preguntar por el segundo movimiento.

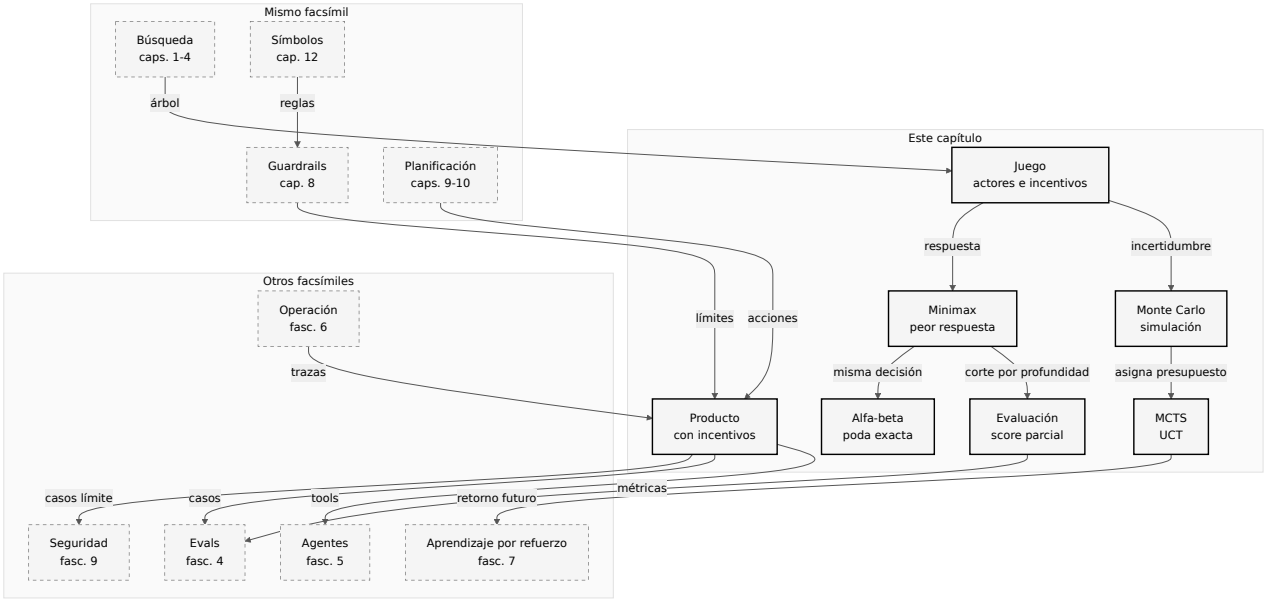
Dónde solía tropezar yo

ERROR	POR QUÉ ES UN ERROR	ANTÍDOTO
Evaluar solo la acción propia	La calidad depende de la respuesta que habilita.	Dibujar al menos un turno del otro actor.
Pensar que minimax describe a toda persona	No todo usuario es racional ni persigue un objetivo opuesto.	Usarlo como caso de tensión, no como psicología humana.
Creer que alfa-beta cambia la decisión	La poda exacta conserva el resultado de minimax.	Separar semántica de eficiencia.
Usar Monte Carlo con simulador débil	Simular mucho no corrige un modelo malo del mundo.	Validar el simulador con datos reales.
Tratar al LLM como simulador estadístico	Genera plausibilidad, no muestras independientes.	Usarlo para hipótesis; contrastar con evidencia.
Diseñar solo para el usuario ideal	Los sistemas reales tienen incentivos, fricción y órdenes en conflicto.	Preguntar qué cambia si el sistema falla.

Cómo encaja todo

Este mapa añade una capa que no estaba en la planificación simple: otras personas, sistemas, reglas o instrucciones también pueden responder. Por eso aparecen minimax, poda alfa-beta, evaluación, Monte Carlo y MCTS.

La decisión aprendida es no evaluar solo el primer movimiento. Hay que mirar qué respuestas habilita una acción, qué peor caso toleras y cuándo merece la pena simular más.


Mismo facsímil
Búsqueda
caps. 1-4
Símbolos
cap. 12
árbol
reglas
Guardrails
cap. 8
Planificación
caps. 9-10
Este capítulo
Juego
actores e incentivos
respuesta
incertidumbre
Minimax
peor respuesta
Monte Carlo
simulación
misma decisión
corte por profundidad
asigna presupuesto
Otros facsímiles
Operación
fasc. 6
límites
acciones
trazas
Producto
con incentivos
Alfa-beta
poda exacta
Evaluación
score parcial
MCTS
UCT
métricas
casos límite
casos
tools
retorno futuro
Seguridad
fasc. 9
Evals
fasc. 4
Agentes
fasc. 5
Aprendizaje por refuerzo
fasc. 7

Vocabulario aprendido

TÉRMINO	DEFINICIÓN
Juego con otros actores	Decisión donde otras partes responden con objetivos propios.
Utilidad	Valor numérico de un resultado para un jugador.
Estrategia	Regla para elegir acciones según estado e información.
Minimax	Elección con mejor valor garantizado ante respuesta óptima de MIN.
Poda alfa-beta	Descarte de ramas que no pueden cambiar la decisión de minimax.
Expectimax	Variante de minimax para el azar: los nodos de azar promedian por probabilidad en vez de minimizar.
Función de evaluación	Heurística que puntúa estados no terminales.
Monte Carlo	Estimación por simulaciones repetidas.
MCTS	Búsqueda en árbol que reparte simulaciones de forma adaptativa.
UCT	Fórmula que mezcla valor observado y exploración de ramas poco visitadas.
Retorno	Recompensa acumulada futura de una trayectoria.

Antes de pasar página

- [ ] ¿Puedo explicar por qué un juego no es una búsqueda normal? (Si no, vuelve a «Cuando otro actor también elige».)
- [ ] ¿Entiendo la recursión de minimax para MAX y MIN, y sabría escribirla con poda? (Si no, vuelve a «Minimax» y «Modo ingeniero: minimax con poda en código».)
- [ ] ¿Sé qué representan α y β y por qué el orden de acciones cambia cuánto se poda? (Si no, vuelve a «Poda alfa-beta».)
- [ ] ¿Distingo cuándo usar minimax (adversario) y cuándo expectimax (azar)? (Si no, vuelve a «Cuando hay azar: expectimax».)
- [ ] ¿Distingo función de evaluación de utilidad terminal? (Si no, vuelve a «Funciones de evaluación».)

- [ ] ¿Puedo explicar qué estima Monte Carlo y qué no demuestra? (Si no, vuelve a «Monte Carlo: decidir simulando».)
- [ ] ¿Entiendo las cuatro fases de MCTS y por qué explora y explota? (Si no, vuelve a «Las cuatro fases de MCTS».)
- [ ] ¿Puedo traducir un problema de producto a actores, incentivos y respuestas? (Si no, vuelve a «En el día a día».)

En resumen

IDEA FUERZA	DETALLE
Juegos añaden respuesta.	La acción importa por las opciones que habilita al otro actor.
Minimax busca garantía.	Elige la mejor jugada bajo peor respuesta racional.
Alfa-beta ahorra trabajo.	Poda ramas sin cambiar la decisión exacta de minimax.
Evaluar es diseñar criterio.	Si puntúas mal, el sistema buscará bien lo incorrecto.
Monte Carlo estima.	Simular ayuda con presupuesto finito, pero depende del simulador.
Producto también tiene incentivos.	Si otra orden o interés cambia el resultado, hay un juego que modelar.

Para saber más

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. y Colton, S. (2012). A survey of Monte Carlo Tree Search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), 1-43. https://doi.org/10.1109/TCIAIG.2012.2186810

Knuth, D. E. y Moore, R. W. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4), 293-326. https://doi.org/10.1016/0004-3702(75)90019-3

Kocsis, L. y Szepesvári, C. (2006). Bandit based Monte-Carlo planning. En *Machine Learning: ECML 2006* (LNCS 4212, pp. 282-293). Springer. https://doi.org/10.1007/11871842_29

Metropolis, N. y Ulam, S. (1949). The Monte Carlo Method. *Journal of the American Statistical Association*, 44(247), 335-341. https://doi.org/10.1080/01621459.1949.10483310

OWASP Foundation. (2025). *OWASP Top 10 for LLM and Generative AI Applications 2025*. https://genai.owasp.org/

Russell, S. y Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4.ª ed.). Pearson.

Shannon, C. E. (1950). Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314), 256-275. https://doi.org/10.1080/14786445008521796

Sutton, R. S. y Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2.ª ed.). MIT Press. https://incompleteideas.net/book/the-book-2nd.html

von Neumann, J. (1928). Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100, 295-320. https://doi.org/10.1007/BF01448847

von Neumann, J. y Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press.

Notas

1. von Neumann, J. y Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press. ↩

2. Shannon, C. E. (1950). Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314), 256-275. https://doi.org/10.1080/14786445008521796 ↩

3. von Neumann, J. y Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press. La teoría de juegos formaliza las decisiones interdependientes entre varios actores con utilidades propias. ↩

4. von Neumann, J. (1928). Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100, 295-320. El teorema minimax establece la existencia de una estrategia óptima en juegos de suma cero; Shannon (1950) lo trasladó a la búsqueda en árboles de juego. ↩

5. Knuth, D. E. y Moore, R. W. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4), 293-326. https://doi.org/10.1016/0004-3702(75)90019-3 ↩

6. Russell, S. y Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4.ª ed.). Pearson. El capítulo de juegos formaliza expectimax con nodos de azar que promedian por probabilidad. ↩

7. Shannon, C. E. (1950). Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314), 256-275. https://doi.org/10.1080/14786445008521796. La función de evaluación como suma ponderada de características del tablero aparece ya en este trabajo fundacional. ↩

8. Metropolis, N. y Ulam, S. (1949). The Monte Carlo Method. *Journal of the American Statistical Association*, 44(247), 335-341. https://doi.org/10.1080/01621459.1949.10483310 ↩

9. Kocsis, L. y Szepesvári, C. (2006). Bandit based Monte-Carlo planning. En *Machine Learning: ECML 2006* (LNCS 4212, pp. 282-293). Springer. https://doi.org/10.1007/11871842_29 ↩

10. Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. y Colton, S. (2012). A survey of Monte Carlo Tree Search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), 1-43. https://doi.org/10.1109/TCIAIG.2012.2186810 ↩

11. OWASP Foundation. (2025). *OWASP Top 10 for LLM and Generative AI Applications 2025*. https://genai.owasp.org/ ↩

12. Sutton, R. S. y Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2.ª ed.). MIT Press. https://incompleteideas.net/book/the-book-2nd.html ↩

Cuadernos para practicar

Escanea el código con el móvil para abrir el cuaderno en Google Colab y ejecutarlo sin instalar nada.

Decidir contra alguien: minimax y poda alfa-beta

https://colab.research.google.com/github/686f6c61/iaparagentecuriosa-notebooks/blob/main/Facs%C3%ADmil%2002/04-minimax-alfa-beta.ipynb