

MINIMAX

BÚSQUEDA ADVERSARIA
Agentes que maximizan

Donde examinaremos los problemas que surgen cuando tratamos de planear en un mundo donde otros agentes planean contra nosotros.



Agenda

- Fundamentos
- Teoría de Juegos
 - Dilema del prisionero
- Juegos de Suma Cero
- Problemas de Búsqueda Adversaria
- Algoritmo Minimax
- Optimizaciones
- Aplicaciones
- Ventajas y Desventajas

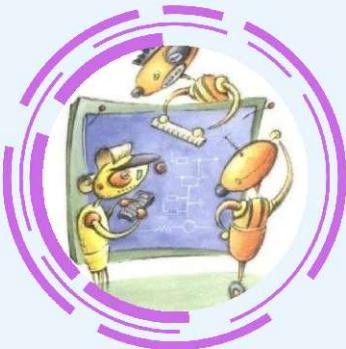


Recordemos...



Agentes Racionales

Cuando los agentes toman decisiones basadas en la maximización de la utilidad esperada.



Entorno Multiagente

En dónde cualquier agente tendrá que considerar las acciones de otros agentes y cómo afectan a su propio bienestar.



Entorno Competitivo

Es un caso especial de entorno multiagente en el que los agentes tienen objetivos opuestos.



Entorno Cooperativo

Es aquel en el que múltiples agentes trabajan juntos para alcanzar un objetivo común.

Recordemos...



Entorno Determinista

Es aquel en el que el resultado de cada acción siempre es predecible y no hay elementos de azar o incertidumbre.



Entorno Estocástico

Es aquel en el que el resultado de una acción no es completamente predecible porque hay un elemento de azar o incertidumbre involucrado.



Fundamentos



Teoria de Juegos

Es un campo de estudio matemático que analiza modelos estratégicos donde múltiples agentes toman decisiones interdependientes.

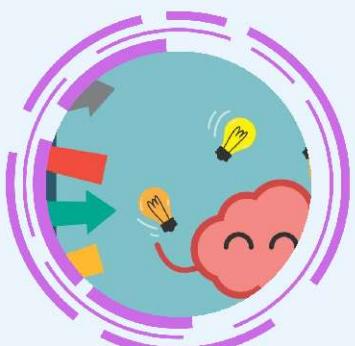
Se utiliza en economía, inteligencia artificial y teoría de decisiones, modela situaciones competitivas y cooperativas.



Busqueda Adversaria

Son juegos en los que los jugadores tienen intereses opuestos y buscan maximizar sus propios resultados mientras minimizan los del oponente.

Ejemplos incluyen el ajedrez, el go y el tres en raya.



Toma de Decisiones

En los juegos adversarios, la toma de decisiones se basa en estrategias que consideran tanto las propias acciones como las del oponente



Juegos de Suma Cero

Juegos donde la ganancia de un jugador es exactamente la pérdida del otro.

En este tipo de juegos, el objetivo es maximizar la propia ganancia mientras se minimiza la del oponente, como en el ajedrez o el póker.



Teoría de Juegos



Desarrollada en sus comienzos como una herramienta para entender el comportamiento de la economía, la teoría de juegos se ha ido extendiendo a muchos otros campos, como la biología, las ciencias de la computación, la sociología, la politología, la psicología y la filosofía.

A raíz de juegos como el dilema del prisionero, en los que el egoísmo generalizado perjudica a los jugadores, la teoría de juegos ha atraído también la atención de los investigadores en informática, usándose en inteligencia artificial y cibernética



Dilema del prisionero

La policía arresta a dos sospechosos.

No hay pruebas suficientes para condenarlos y, tras haberlos separado, los visita a cada uno y les ofrece el mismo trato.

Si uno confiesa y su cómplice no, el cómplice será condenado a la pena total, tres años, y el primero será liberado.

Si uno calla y el cómplice confiesa, el primero recibirá esa pena y será el cómplice quien salga libre.

Si ambos confiesan, ambos serán condenados a dos años.

Si ambos lo niegan, todo lo que podrán hacer será encerrarlos durante un año por un cargo menor.

	B A	 B stays silent	 B testifies
 A stays silent	  R, -1 R, -1	  S, -3 T, 0	
 A testifies	  T, 0 S, -3	  P, -2 P, -2	



Juegos de Suma Cero



Describe una situación en la que la ganancia o pérdida de un participante se equilibra con exactitud con las pérdidas o ganancias de los otros participantes.

Falacia de Suma Cero

Es el error de suponer que en cualquier situación de intercambio o competencia, una persona solo puede ganar si otra pierde

- No todos los sistemas son cerrados y de recursos fijos.
- Muchas situaciones permiten la cooperación y el crecimiento mutuo.

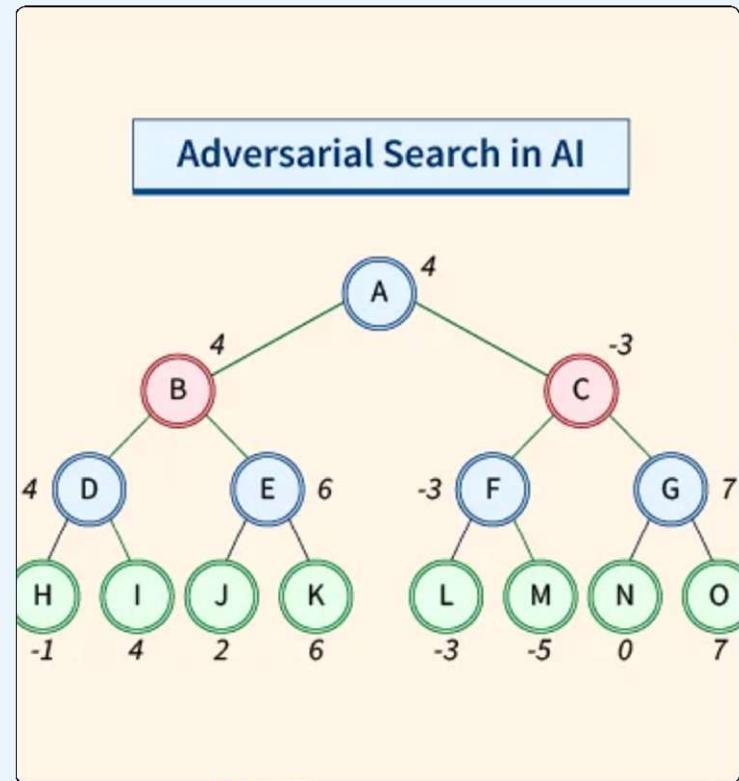
Problemas de Busqueda Adversaria

Características

- Determinísticos o estocásticos
- Uno, dos, n jugadores
- Visibilidad total o parcial
- Juegos que suman 0

Objetivo

Una estrategia (política) que recomiende un movimiento desde un estado



Problemas de Búsqueda Adversaria

Juegos Determinísticos

- N Estados
- 1 a N Jugadores (P)
- Acciones que dependen de P (jugador)
- Meta (Terminal/Goal)
- Utility Value (Valor de Utilidad)
- Terminal Utility (Utilidad/Valor final)

Juegos
Determinísticos



Problemas de Busqueda Adversaria

Suma Cero

- Utilidades opuestas
- Uno maximiza y el otro minimiza
- Adversario, competición pura



General Games

- Utilidades independientes
- Todos maximizan con su utilidad
- Cooperación, indiferencia, competición



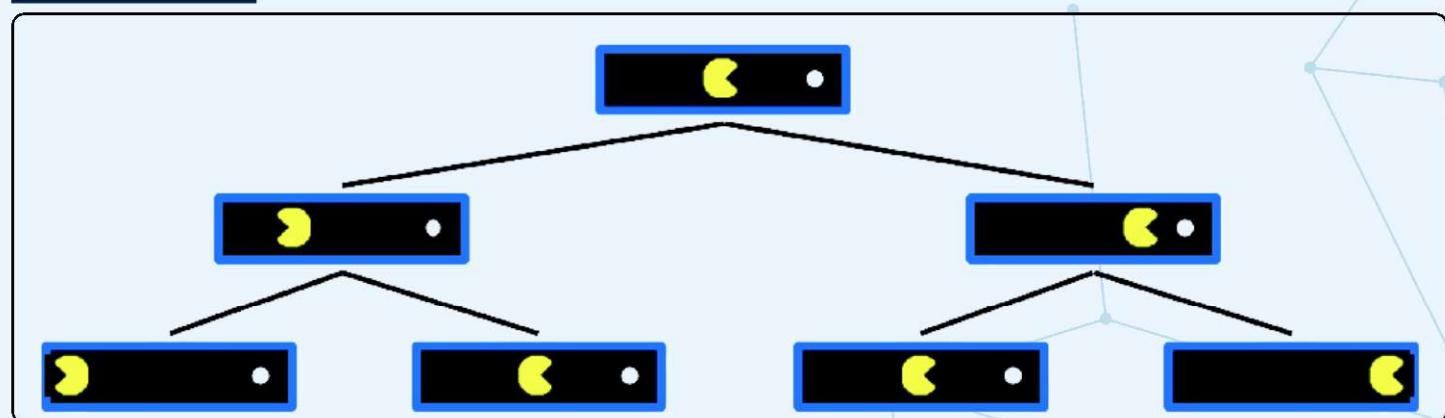
1 AGENTE (Agente que maximiza)

El escenario con un solo jugador (pacman), debe alcanzar la bolita (pellet)



Supongamos que Pacman comienza con 10 puntos y pierde 1 punto por movimiento hasta que se come la bolita

Game Tree

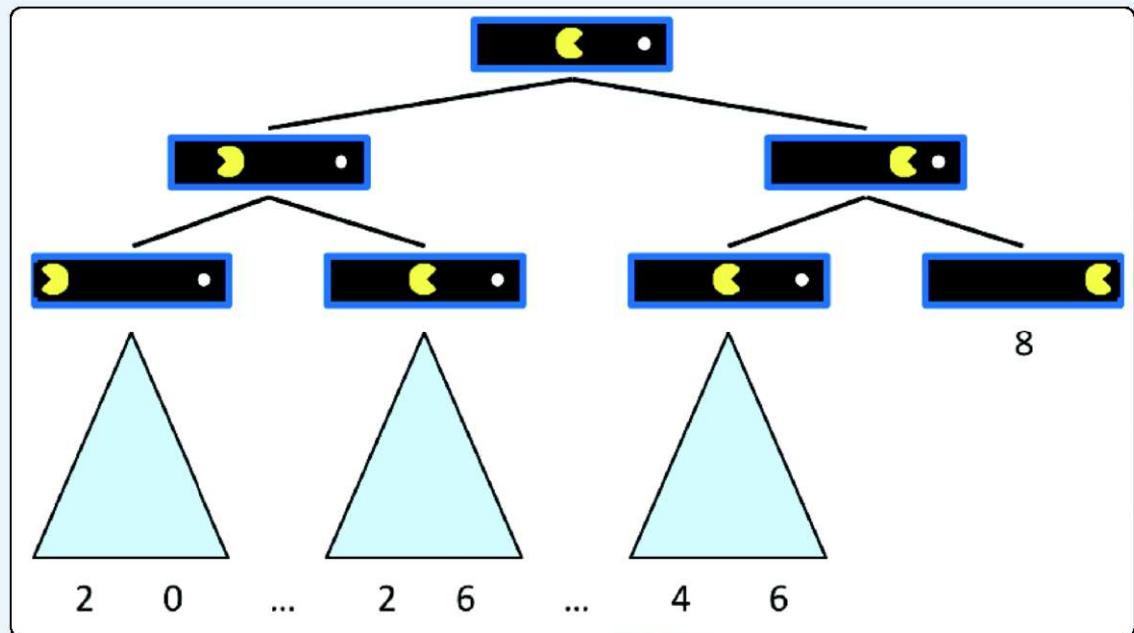


1 AGENTE

El escenario con un solo jugador (pacman), debe alcanzar la bolita (pellet)

Game Tree

Si Pacman va directo a la bolita, termina el juego con una puntuación de 8 puntos, mientras que si retrocede en cualquier punto, termina con una puntuación de menor valor



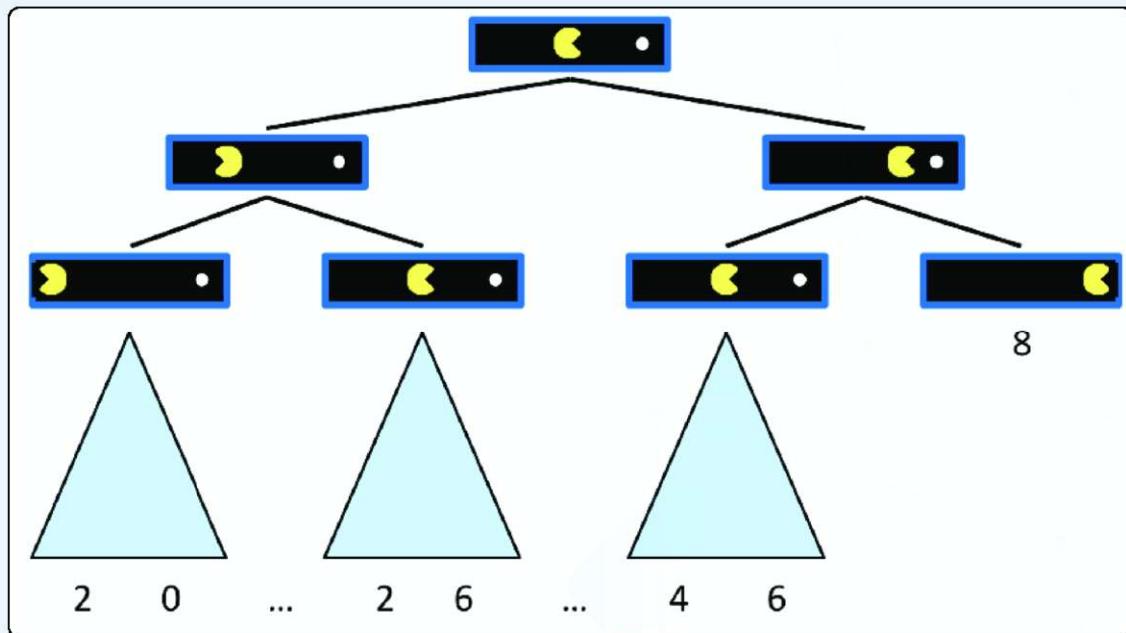
1 AGENTE

El escenario con un solo jugador (pacman), debe alcanzar la bolita (pellet)

Game Tree

El valor (V) de un estado (s) se define como el mejor resultado posible (utilidad) que un agente puede lograr a partir de ese estado.

Simplemente, piense en la utilidad de un agente como su puntuación o la cantidad de puntos que obtiene.



1 AGENTE

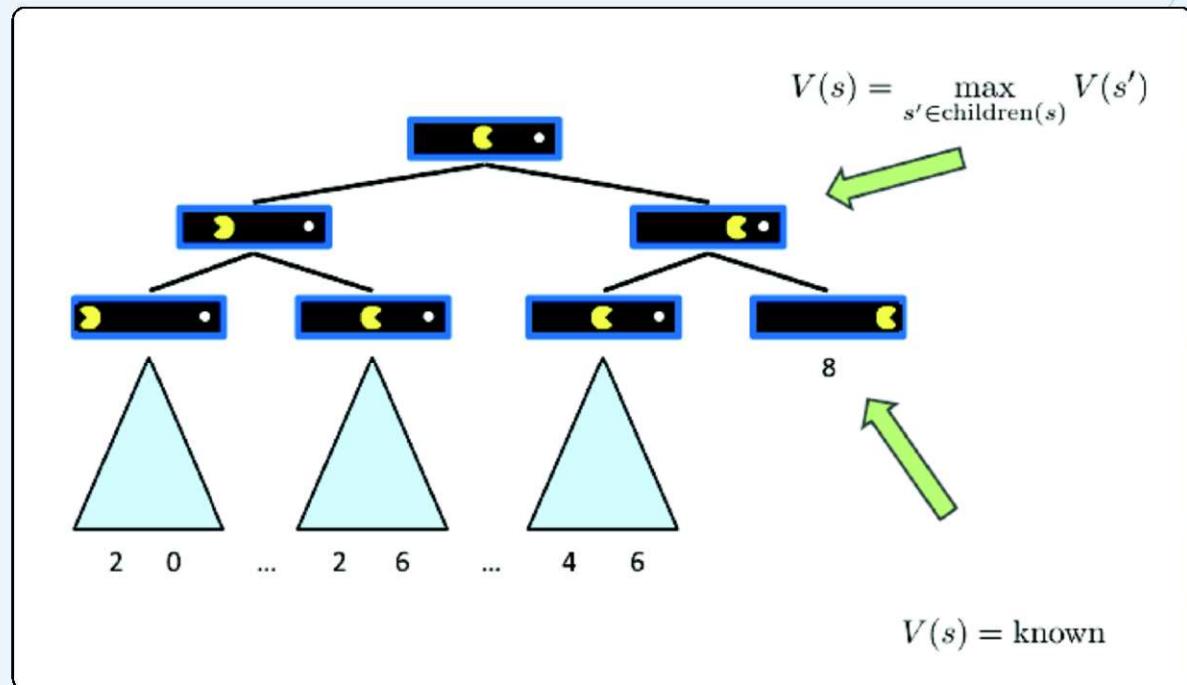
El escenario con un solo jugador (pacman), debe alcanzar la bolita (pellet)

Iremos buscando el que mayor utilidad nos representa entre los estados intermedios (s'). ($\max V(s')$)

El valor de un estado terminal $V(s)$, llamado utilidad terminal, es siempre un valor conocido determinista y una propiedad inherente del juego.

En nuestro ejemplo de Pacman, el valor del estado terminal más a la derecha es simplemente 8, la puntuación que Pacman obtiene al ir directo a la bolita.

Game Tree



Adversario (Agente que minimiza)

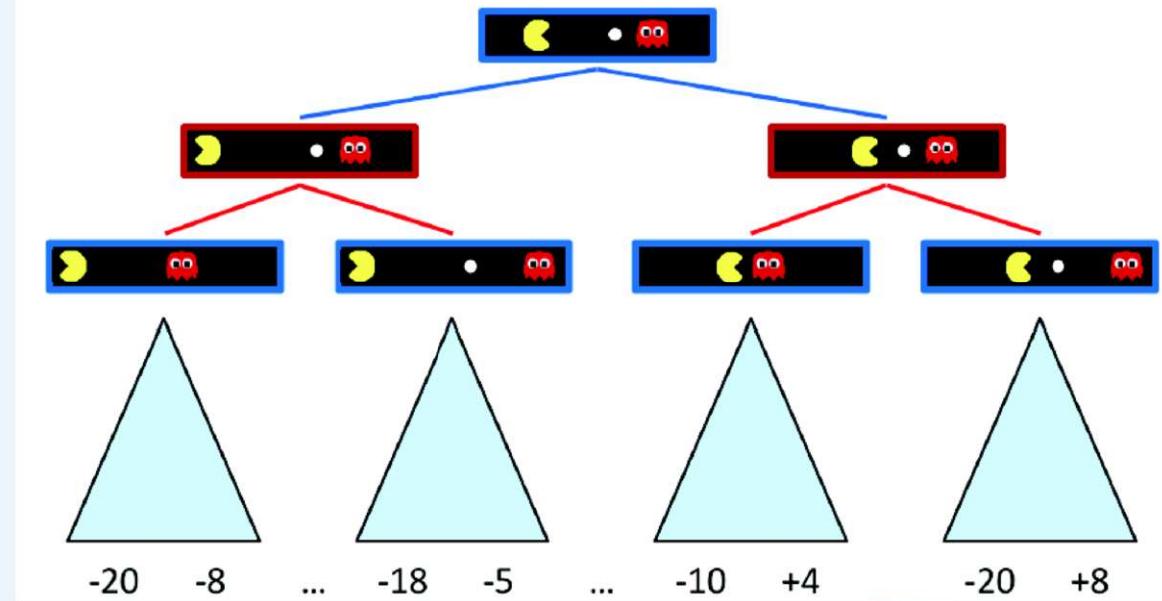
Agregamos un adversario



Introduzcamos un nuevo tablero de juego con un fantasma adversario que quiere evitar que Pacman se coma la bolita

Los dos agentes se turnan para realizar movimientos, lo que lleva a un árbol de juego donde los dos agentes activan y desactivan las capas del árbol que "controlan".

Game Tree



Estados

Pacman

Ghost

Adversario

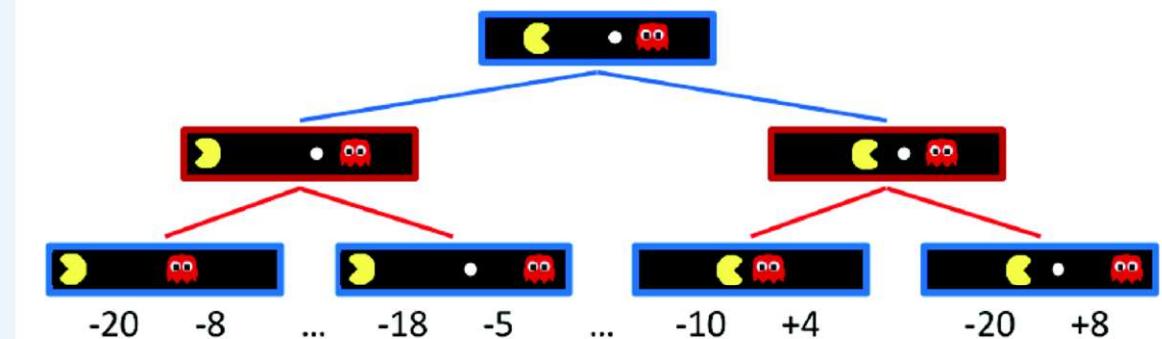
Agregamos un adversario



Para simplificar, truncaremos este árbol de juego a un árbol de profundidad 2 y asignaremos valores inventados a los estados terminales

Agregar nodos controlados por fantasmas cambia el movimiento que Pacman cree que es óptimo, y el nuevo movimiento óptimo se determina con el algoritmo minimax.

Game Tree



Estados

Pacman

Ghost

Adversario

Agregamos un adversario



En lugar de maximizar la utilidad sobre los hijos en cada nivel del árbol, el algoritmo minimax solo maximiza sobre los hijos de los nodos controlados por Pacman, mientras que minimiza sobre los hijos de los nodos controlados por fantasmas.

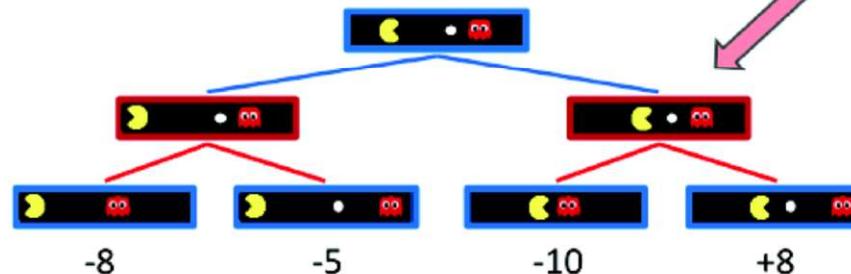
Game Tree

Agente

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

Oponente

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal

$$V(s) = \text{known}$$

Estados

Pacman

Ghost



Decisiones óptimas en juegos: Minimax

Consideraremos juegos con dos jugadores, que llamaremos MAX y MIN.

- MAX mueve primero, y luego mueven por turno hasta que el juego se termina.
- Al final de juego, se conceden puntos al jugador ganador y penalizaciones al perdedor.

$$\text{VALOR-MINIMAX}(n) = \begin{cases} \text{UTILIDAD}(n) & \text{si } n \text{ es un estado terminal} \\ \max_{s \in \text{Sucesores}(n)} \text{VALOR-MINIMAX}(s) & \text{si } n \text{ es un estado MAX} \\ \min_{s \in \text{Sucesores}(n)} \text{VALOR-MINIMAX}(s) & \text{si } n \text{ es un estado MIN} \end{cases}$$

Decisiones óptimas en juegos: Minimax

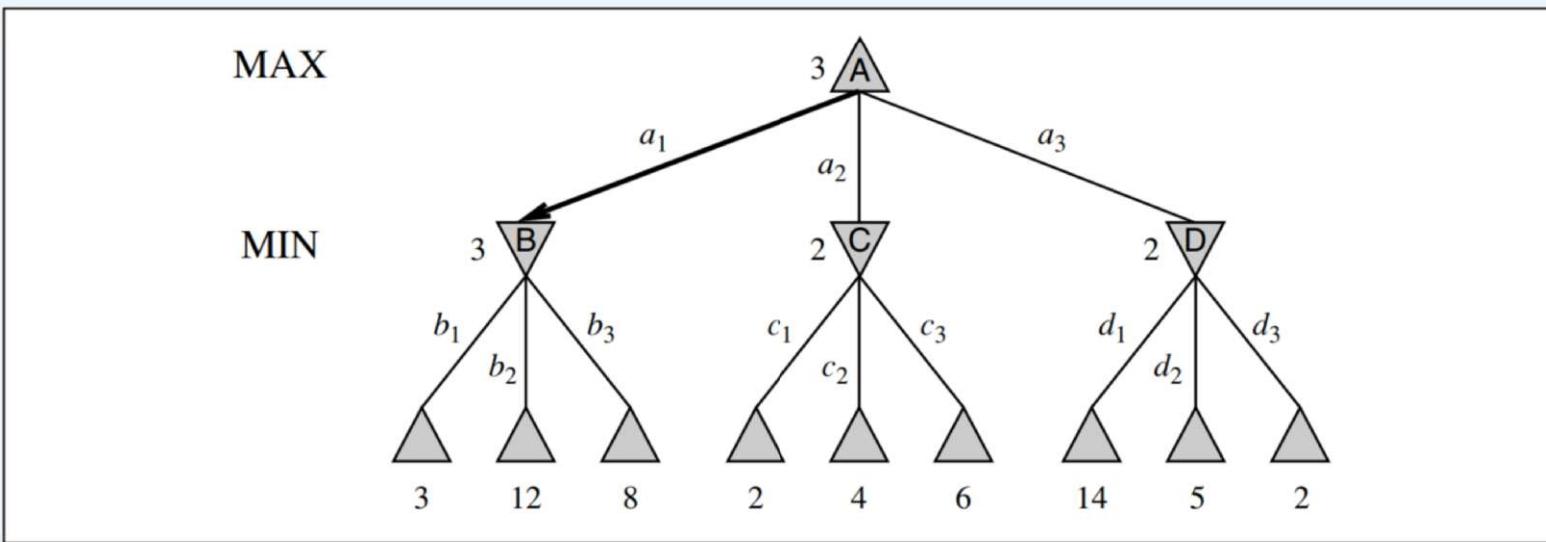


Figura 6.2 Un árbol de juegos de dos capas. Los nodos Δ son «nodos MAX», en los que le toca mover a MAX, y los nodos ∇ son «nodos MIN». Los nodos terminales muestran los valores de utilidad para MAX; los otros nodos son etiquetados por sus valores minimax. El mejor movimiento de MAX en la raíz es a_1 , porque conduce al sucesor con el valor minimax más alto, y la mejor respuesta de MIN es b_1 , porque conduce al sucesor con el valor minimax más bajo.

Decisiones óptimas en juegos: Minimax

Si estás implementando un agente de IA para un juego:

- *max-value* representaría al jugador que quiere ganar (MAX).
- *min-value* representaría al oponente que intenta bloquearlo (MIN).

Se usa este algoritmo para recorrer el árbol de posibles movimientos y decidir la mejor jugada.

```
def max-value(state):  
    initialize v = -∞  
    for each successor of state:  
        v = max(v, min-value(successor))  
    return v
```

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

```
def min-value(state):  
    initialize v = +∞  
    for each successor of state:  
        v = min(v, max-value(successor))  
    return v
```

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Minimax

```
def value(state):
```

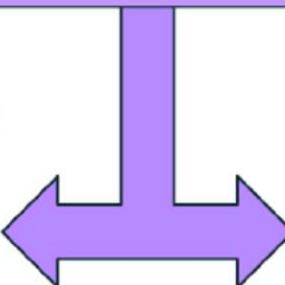
 if the state is a terminal state: return the state's utility
 if the next agent is MAX: return max-value(state)
 if the next agent is MIN: return min-value(state)

```
def max-value(state):
```

 initialize v = -∞
 for each successor of state:
 v = max(v, value(successor))
 return v

```
def min-value(state):
```

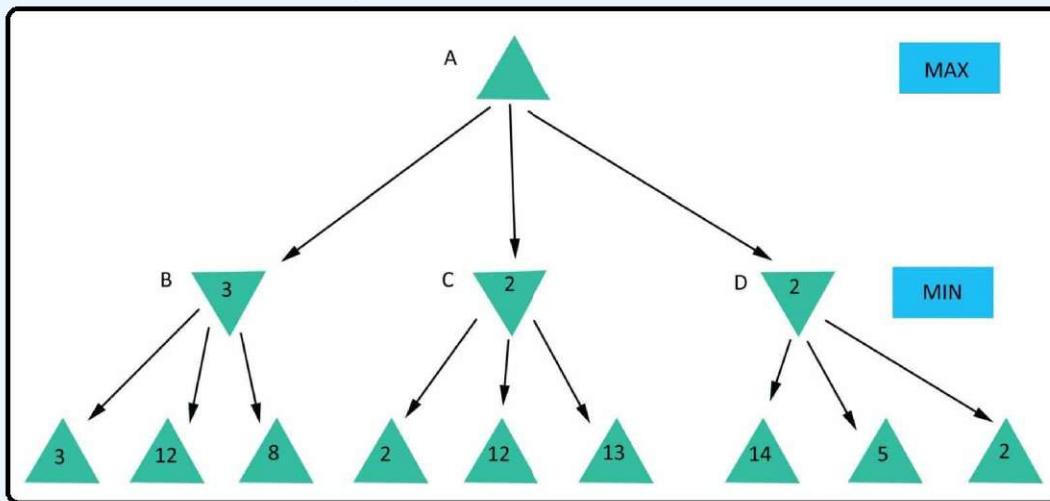
 initialize v = +∞
 for each successor of state:
 v = min(v, value(successor))
 return v



Minimax

El algoritmo Minimax es una técnica utilizada en juegos de dos jugadores con suma cero y turnos alternados, como el ajedrez o Pac-Man contra fantasmas.

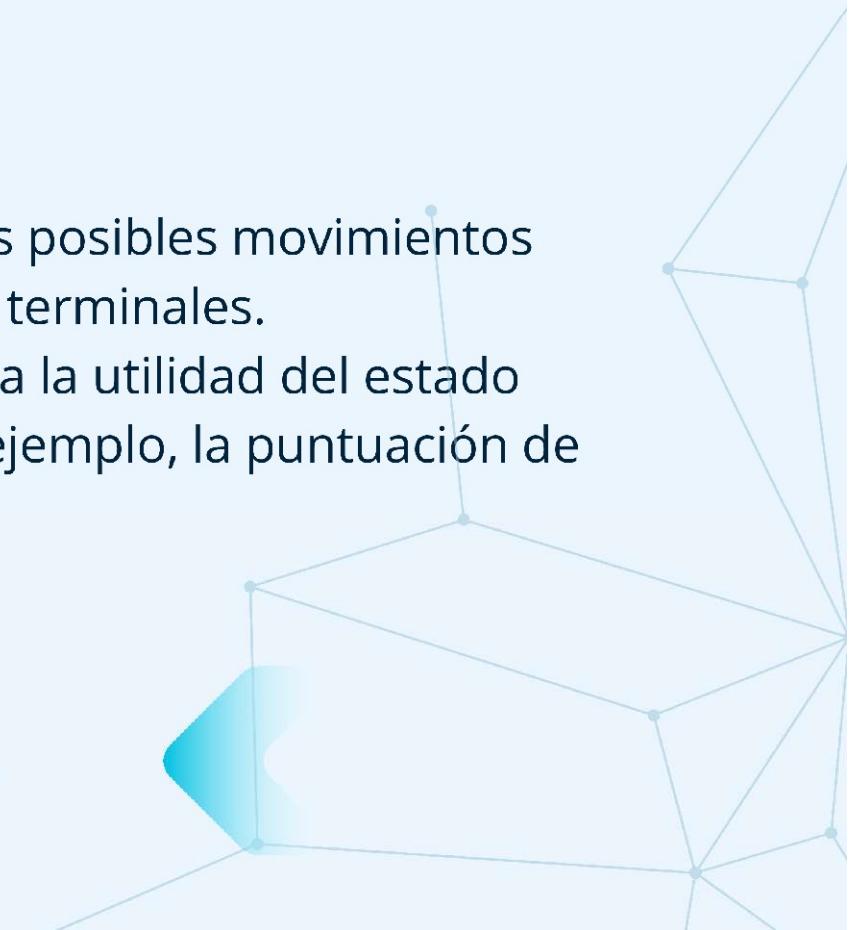
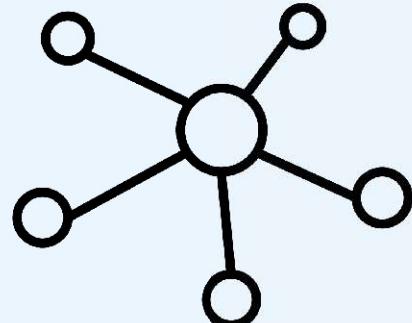
Su propósito es encontrar la mejor jugada para el jugador MAX (quien busca maximizar su puntuación) asumiendo que el oponente (MIN) jugará de la mejor forma posible para minimizar la puntuación de MAX.



Minimax

1. Exploración del árbol de juego:

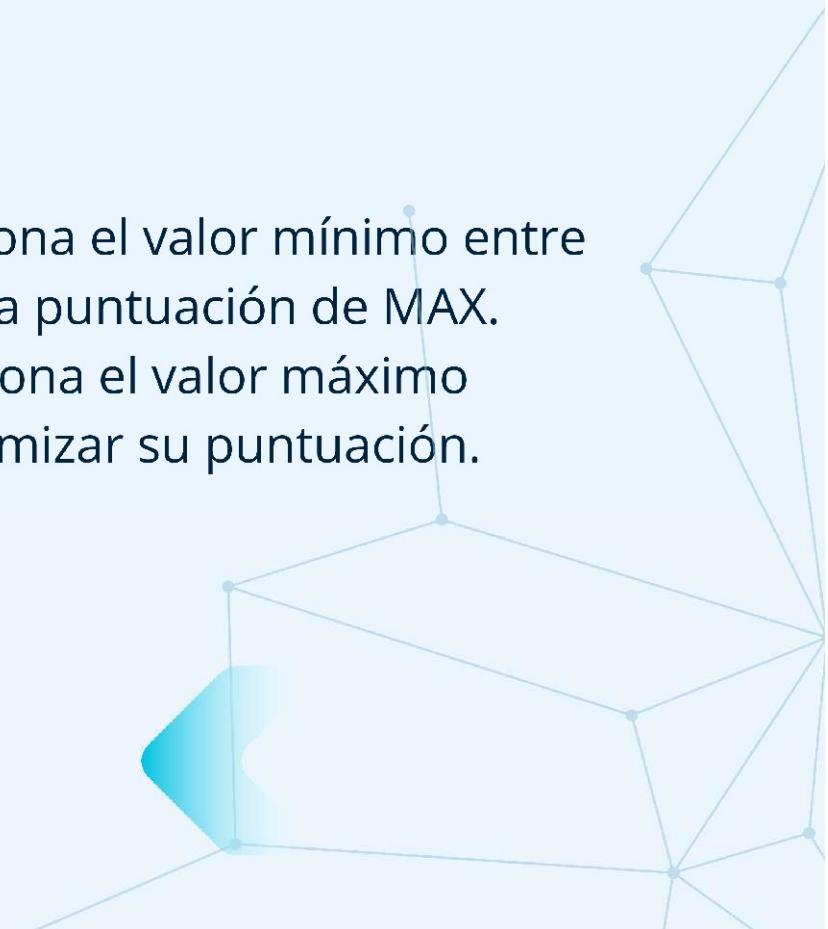
- Se genera el árbol de juego con todos los posibles movimientos desde el estado actual hasta los estados terminales.
- En los nodos terminales (hojas), se evalúa la utilidad del estado usando una función de evaluación (por ejemplo, la puntuación de Pac-Man).



Minimax

2. Propagación de valores hacia arriba:

- En los nodos donde juega MIN, se selecciona el valor mínimo entre sus hijos, porque MIN intenta minimizar la puntuación de MAX.
- En los nodos donde juega MAX, se selecciona el valor máximo entre sus hijos, porque MAX intenta maximizar su puntuación.



Minimax

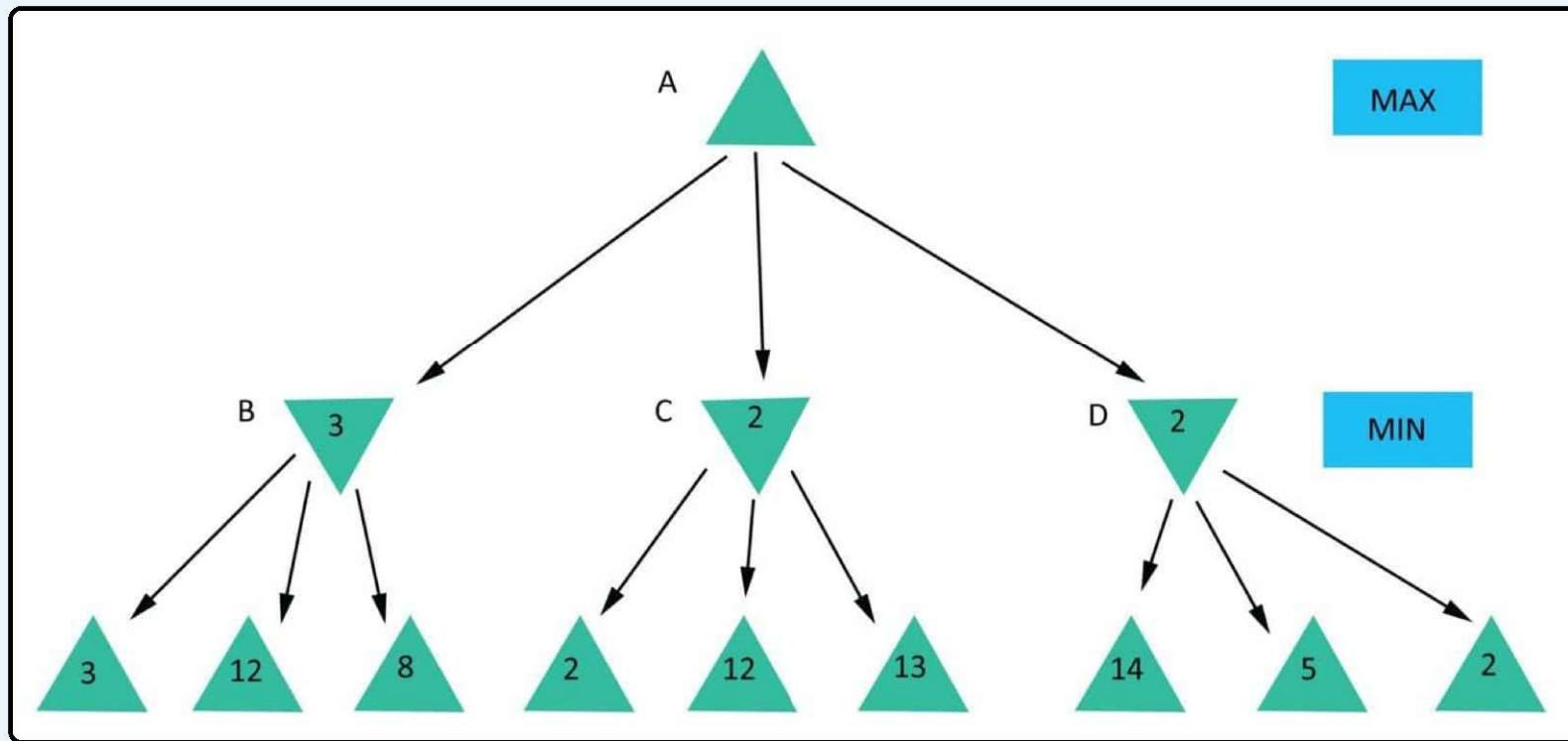
Elección de la mejor jugada:

- El jugador MAX elige la acción que lleva al estado con el mayor valor minimax.



Ejemplo del Funcionamiento

Si tenemos tres estados terminales con valores de utilidad 3, 12 y 8, MIN elegiría el mínimo (3) como la mejor opción en su turno. Luego, MAX evaluaría entre diferentes opciones y seleccionaría la de mayor valor como su mejor jugada.

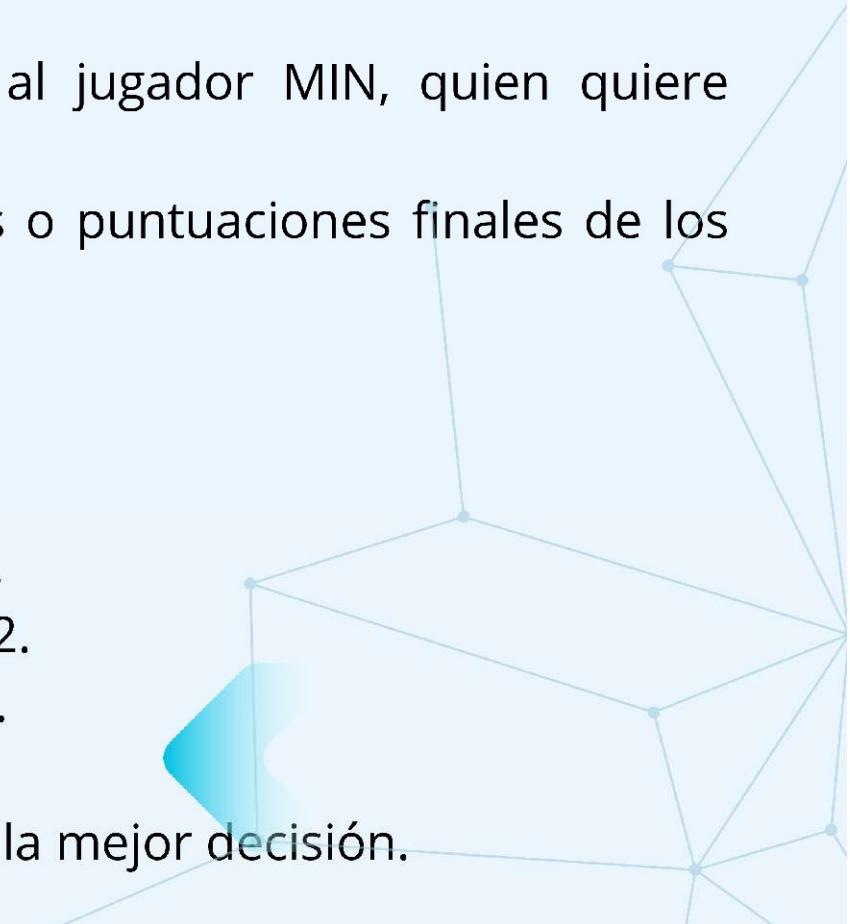


Niveles del árbol:

- Nivel 0 (Raíz - Nodo A) Representa al jugador MAX, quien quiere maximizar la utilidad.
- Nivel 1 (Nodos B, C y D) Representan al jugador MIN, quien quiere minimizar la utilidad.
- Nivel 2 (Hojas) Representan las utilidades o puntuaciones finales de los estados del juego.

Cálculo Minimax:

- MIN elige el valor mínimo de sus hijos:
 - Nodo B tiene hijos 3, 12 y 8 MIN elige 3.
 - Nodo C tiene hijos 2, 12 y 13 MIN elige 2.
 - Nodo D tiene hijos 14, 5 y 2 MIN elige 2.
- MAX elige el valor máximo de sus hijos:
 - Entre B (3), C (2) y D (2), MAX elige 3 como la mejor decisión.



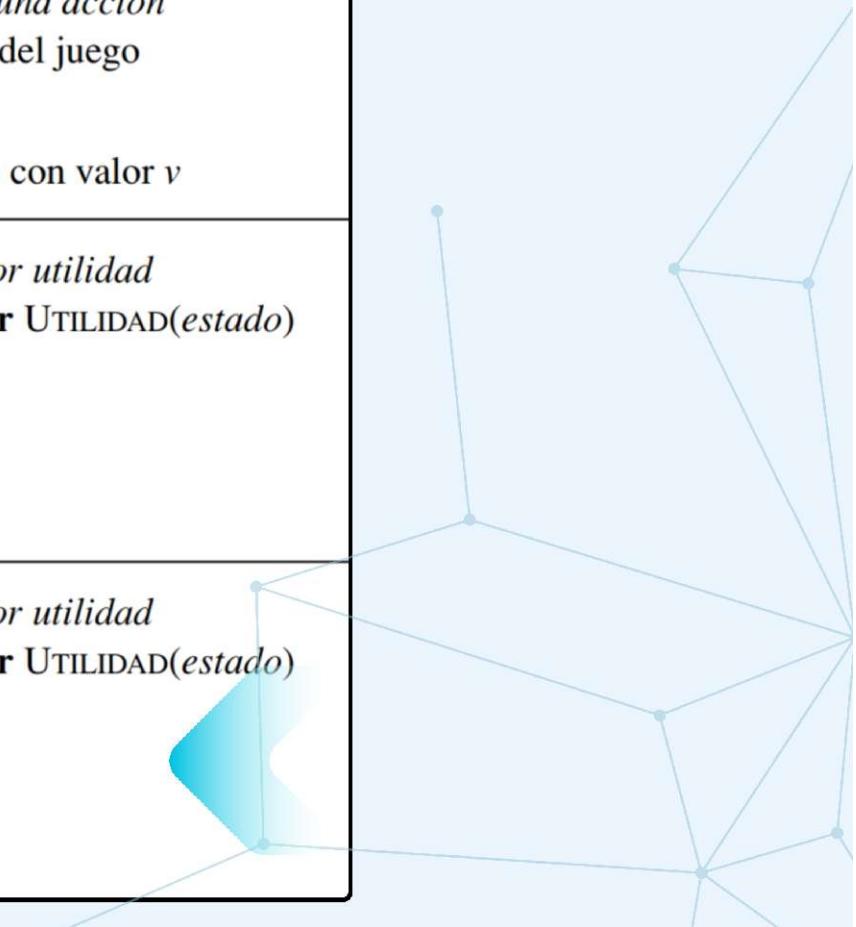
Minimax

función DECISIÓN-MINIMAX(*estado*) devuelve una acción
variables de entrada: *estado*, estado actual del juego

$v \leftarrow \text{MAX-VALOR}(\text{estado})$
devolver la acción de SUCESORES(*estado*) con valor v

función MAX-VALOR(*estado*) devuelve un valor utilidad
si TEST-TERMINAL(*estado*) **entonces devolver** UTILIDAD(*estado*)
 $v \leftarrow -\infty$
para un *s* en SUCESORES(*estado*) **hacer**
 $v \leftarrow \text{MAX}(v, \text{MIN-VALOR}(s))$
devolver v

función MIN-VALOR(*estado*) devuelve un valor utilidad
si TEST-TERMINAL(*estado*) **entonces devolver** UTILIDAD(*estado*)
 $v \leftarrow \infty$
para un *s* en SUCESORES(*estado*) **hacer**
 $v \leftarrow \text{MIN}(v, \text{MAX-VALOR}(s))$
devolver v



Complejidad Computacional

Tiempo:

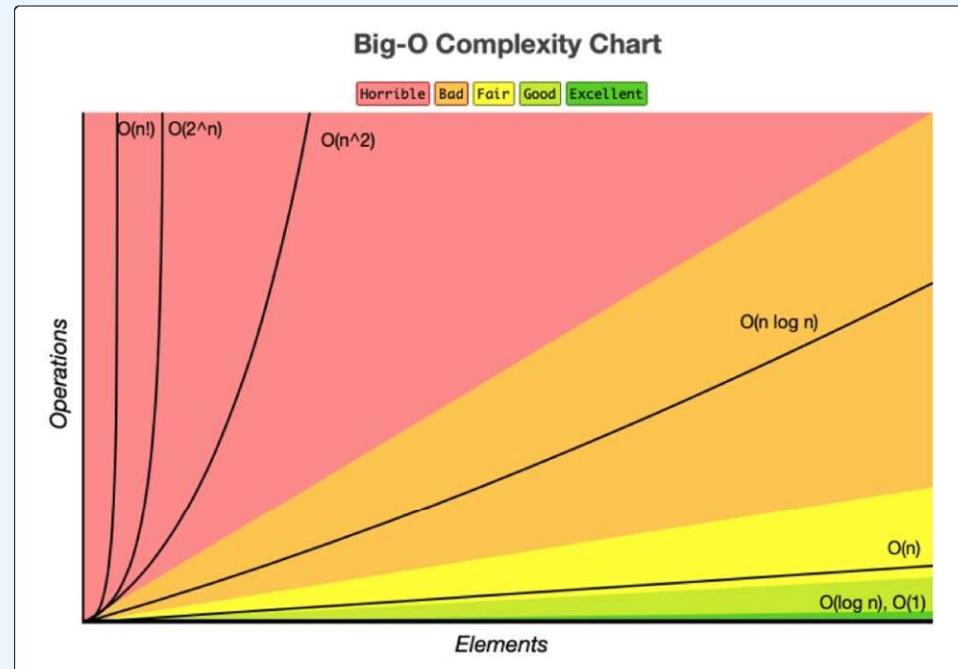
$O(b^m)$ donde:

- b es el número de movimientos posibles en cada turno (factor de ramificación).
- m es la profundidad máxima del árbol.

Es un crecimiento exponencial, lo que lo hace impráctico en juegos complejos como el ajedrez.

Espacio:

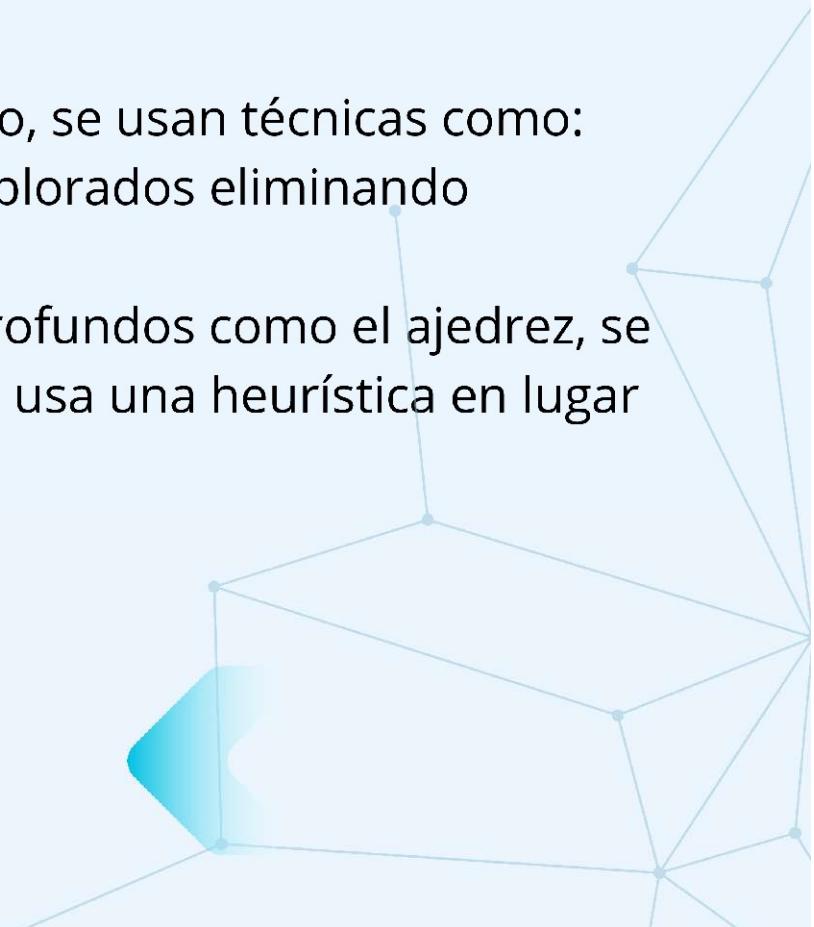
- $O(bm)$ si se generan todos los nodos a la vez.
- $O(m)$ si se usa una implementación recursiva con generación en tiempo real.



Optimización

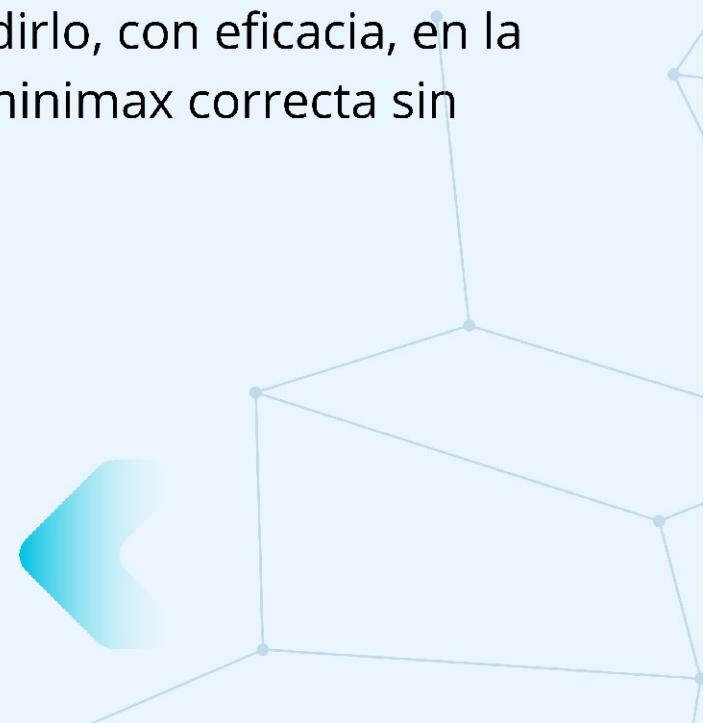
Dado que Minimax es computacionalmente costoso, se usan técnicas como:

- *Poda alfa-beta*: Reduce la cantidad de nodos explorados eliminando movimientos que no afectan la decisión final.
- *Funciones de evaluación heurísticas*: En juegos profundos como el ajedrez, se detiene la búsqueda en cierta profundidad y se usa una heurística en lugar de calcular hasta los estados finales.



Poda Alfa-Beta

El problema de la búsqueda minimax es que el número de estados que tiene que examinar es exponencial en el número de movimientos. Lamentablemente no podemos eliminar el exponente, pero podemos dividirlo, con eficacia, en la mitad. La jugada es que es posible calcular la decisión minimax correcta sin mirar todos los nodos en el árbol de juegos.



Poda Alfa-Beta

La poda alfa-beta puede aplicarse a árboles de cualquier profundidad, y, a menudo, es posible podar subárboles enteros.

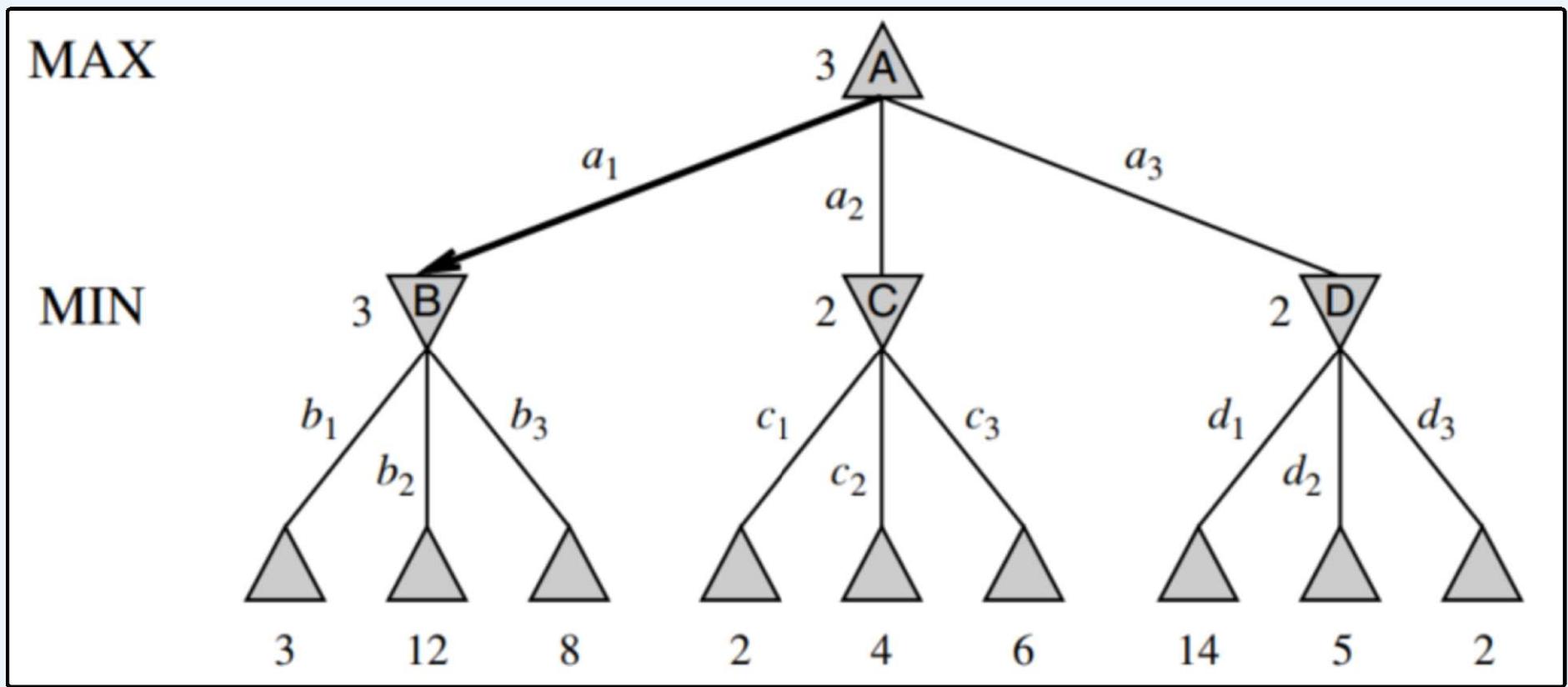


Valores Alfa y Beta

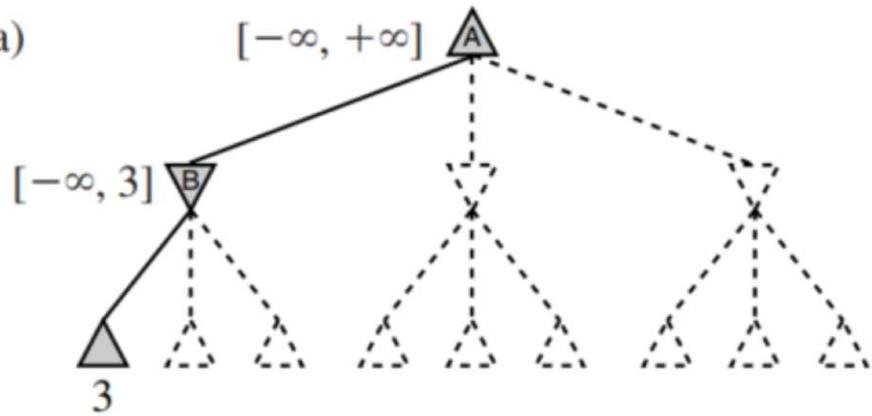
- α (*alfa*): Mejor valor garantizado para el jugador MAX en el camino actual.
- β (*beta*): Mejor valor garantizado para el jugador MIN en el camino actual.

Ejemplo de Poda Alfa-Beta

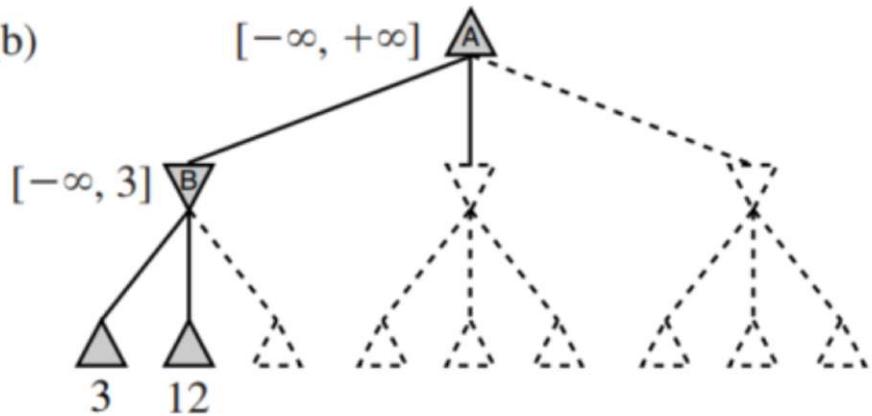
Los nodos terminales muestran los valores de utilidad para MAX; los otros nodos son etiquetados por sus valores minimax.



(a)



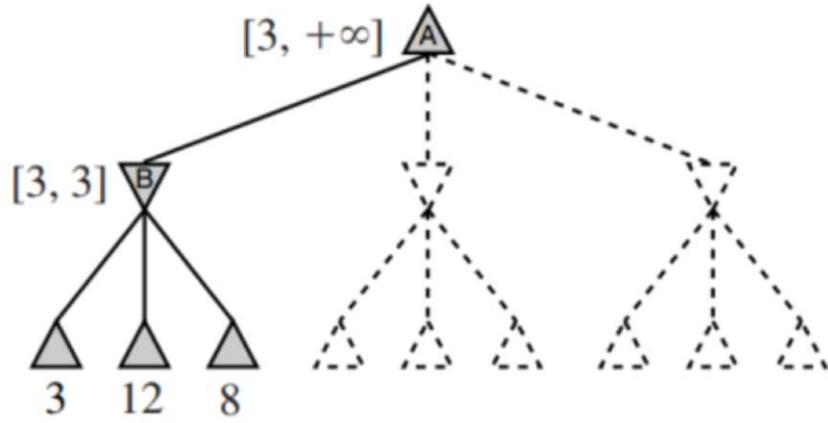
(b)



(a) La primera hoja debajo de B tiene el valor 3. De ahí B, que es un nodo MIN, tiene un valor de como máximo 3.

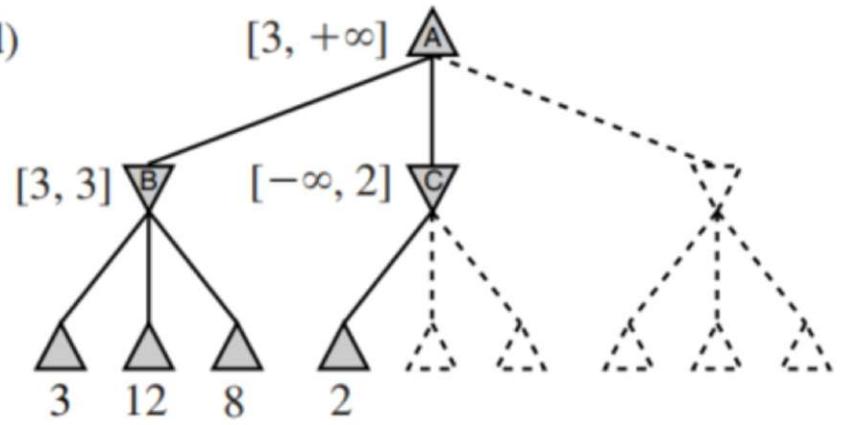
(b) La segunda hoja debajo de B tiene un valor de 12; MIN evitaría este movimiento, entonces el valor de B es todavía como máximo 3.

(c)



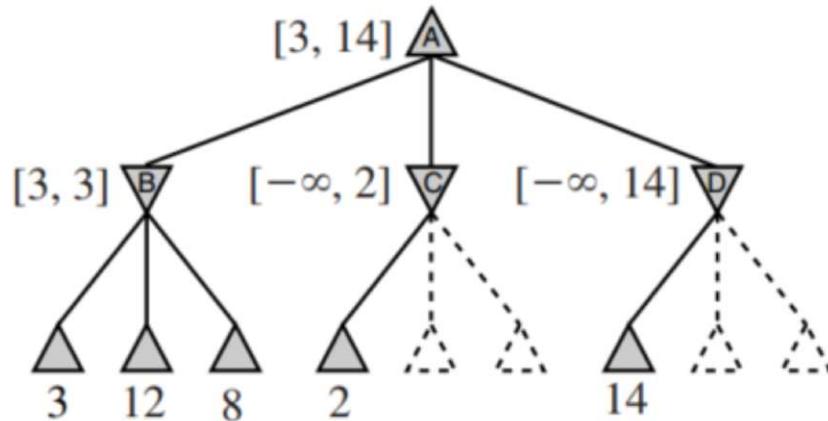
(c) La tercera hoja debajo de B tiene un valor de 8; hemos visto a todos los sucesores de B, entonces el valor de B es exactamente 3. Ahora, podemos deducir que el valor de la raíz es al menos 3, porque MAX tiene una opción digna de 3 en la raíz.

(d)



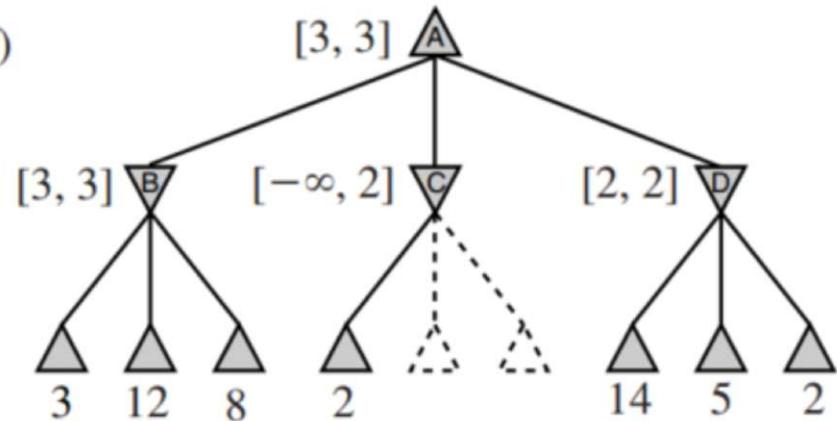
(d) La primera hoja debajo de C tiene el valor 2. De ahí, C, que es un nodo MIN, tiene un valor de como máximo 2. Pero sabemos que B vale 3, entonces MAX nunca elegiría C. Por lo tanto, no hay ninguna razón en mirar a los otros sucesores de C. Este es un ejemplo de la poda de alfa-beta.

(e)



(e) La primera hoja debajo de D tiene el valor 14, entonces D vale como máximo 14. Este es todavía más alto que la mejor alternativa de MAX (es decir, 3), entonces tenemos que seguir explorando a los sucesores de D. Note también que ahora tenemos límites sobre todos los sucesores de la raíz, entonces el valor de la raíz es también como máximo 14.

(f)



(f) El segundo sucesor de D vale 5, así que otra vez tenemos que seguir explorando. El tercer sucesor vale 2, así que ahora D vale exactamente 2. La decisión de MAX en la raíz es moverse a B, dando un valor de 3.

Simulador

- <https://pascsha.ch/info2/abTreePractice/>
- <http://homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html>



Categoría	Aplicación	Descripción
Juegos de Mesa	Damas	Evaluá las posibles secuencias de movimientos para maximizar las ganancias y minimizar las pérdidas.
Videojuegos	Juegos de laberintos y rompecabezas	Encuentra rutas óptimas considerando posibles movimientos de oponentes.
Medicina	Diagnóstico basado en decisiones	Puede ayudar en la selección óptima de tratamientos considerando riesgos.

Ventajas

1. Encuentra la mejor jugada en un entorno determinista

- Minimax garantiza encontrar la estrategia óptima si el árbol de juego se puede evaluar completamente.

2. No necesita conocimientos probabilísticos

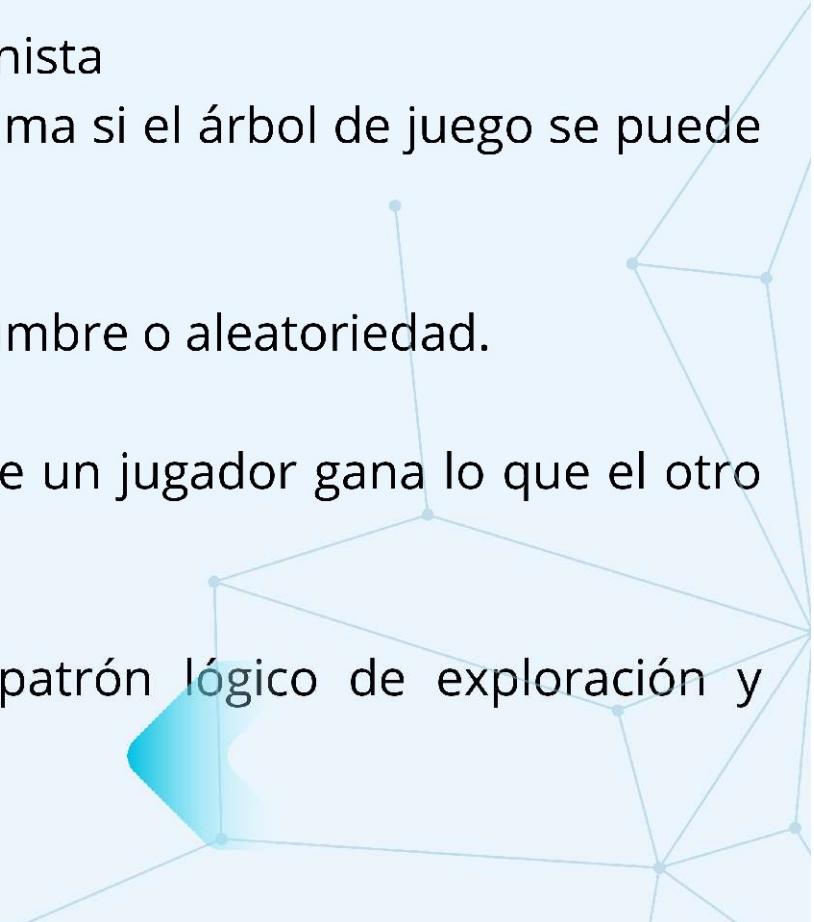
- Se basa en decisiones lógicas y no en incertidumbre o aleatoriedad.

3. Aplicable a juegos de suma cero

- Funciona bien en juegos de adversarios donde un jugador gana lo que el otro pierde (ajedrez, tic-tac-toe, damas, etc.).

4. Estructura clara y bien definida

- Su implementación es sencilla y sigue un patrón lógico de exploración y evaluación de estados.



Desventajas

1. Explosión combinatoria

- Para juegos con muchos movimientos posibles, el árbol crece exponencialmente, haciendo que el algoritmo sea ineficiente sin mejoras como poda alfa-beta.

2. Asume que el oponente juega de manera óptima

- No considera que el oponente pueda cometer errores o jugar de forma irracional.

3. No maneja incertidumbre o información oculta

- En juegos con elementos aleatorios (dados, cartas ocultas) o información incompleta, Minimax no es suficiente.

4. No se adapta a juegos muy dinámicos

- En juegos con tiempos de respuesta cortos, puede ser poco práctico por el tiempo de cálculo que requiere.

