Adversarial Search

Inteligencia Artificial

Adversarial Search

Adversarial Search aborda entornos competitivos en los que dos o más agentes tienen objetivos en conflicto, lo que da lugar a problemas de búsqueda adversaria.

Estos algoritmos se suelen centrar en juegos como el ajedrez, el Go y el póker.

Para los investigadores de IA, la naturaleza simplificada de estos juegos es una ventaja: el estado de un juego es fácil de representar y los agentes suelen estar restringidos a un número reducido de acciones, cuyos efectos están definidos por reglas precisas.

GAME THEORY

Adversarial Search

Existen al menos tres enfoques que podemos adoptar en entornos multiagente:

Economía:

Trata a los agentes como una gran colectividad en lugar de modelar sus acciones individuales. Se analizan patrones generales de comportamiento en lugar de intentar predecir las acciones de cada agente de manera independiente.

EJEMPLOS

- Modelado del tráfico vehicular: En lugar de predecir la ruta exacta de cada conductor, los modelos pueden estimar cómo los patrones de tráfico se ven afectados por cambios en la infraestructura o en la cantidad de vehículos.
- Simulaciones económicas: Los gobiernos utilizan modelos económicos para prever cómo afectarán las políticas fiscales y monetarias a la inflación y el desempleo.

Adversarial Search

Existen al menos tres enfoques que podemos adoptar en entornos multiagente:

Modelado como un entorno no determinista:

Los agentes adversarios se tratan como parte del entorno, similar a factores aleatorios. Se modela la incertidumbre de las acciones de estos agentes, pero sin considerar que tienen intenciones específicas.

EJEMPLO: Juego de póker con modelado probabilístico

Supongamos que queremos desarrollar un sistema de inteligencia artificial para jugar al póker. En lugar de considerar que los oponentes están tomando decisiones estratégicas, podríamos tratarlos como eventos aleatorios. Por ejemplo, podríamos modelar que cada jugador elige su acción (apostar, retirarse, subir la apuesta) con una probabilidad específica basada en el historial de jugadas observadas.

Adversarial Search

Existen al menos tres enfoques que podemos adoptar en entornos multiagente:

Búsqueda adversaria en árboles de juego:

modelamos explícitamente a los agentes adversarios y asumimos que están tratando activamente de minimizar nuestro éxito mientras maximizan el suyo. Para resolver estos problemas, se utilizan técnicas de búsqueda en árboles de juego, como ***el algoritmo Minimax****, que calcula las mejores jugadas bajo la suposición de que el oponente jugará de la mejor manera posible.*

EJEMPLO: Ajedrez y el algoritmo Minimax

En el ajedrez, cada jugador busca maximizar su probabilidad de ganar mientras minimiza las oportunidades del oponente. Supongamos que nuestra IA debe decidir su próximo movimiento. Con el algoritmo Minimax, genera un árbol de posibles movimientos futuros y asume que el oponente elegirá siempre la mejor jugada para sí mismo.

Enfoque	Cómo modela a los agentes	Ejemplo
Economía (modelado agregado)	Se analizan patrones globales de comportamiento sin modelar agentes individuales.	Predicción de precios en la bolsa de valores.
Modelado no determinista	Se trata a los agentes como factores aleatorios en el entorno, sin intención estratégica.	Modelar jugadores de póker como eventos probabilísticos.
Búsqueda adversaria	Se modela explícitamente a los agentes como oponentes con objetivos opuestos.	IA jugando ajedrez con algoritmo Minimax.

GAME THEORY-Two-player zero-sum games

Game theory-Two-player zero-sum games

Los juegos más comúnmente estudiados en el ámbito de la inteligencia artificial (como el ajedrez y el Go) son lo que los teóricos de juegos llaman juegos deterministas, de dos jugadores, por turnos, de **información perfecta** y **suma cero**.

- ***"Información perfecta"*** *es un sinónimo de "totalmente observable".*
- ***"Suma cero"*** *significa que lo que es bueno para un jugador es igual de malo para el otro: no hay un resultado de "ganar-ganar".*

En los juegos, a menudo usamos el término **movimiento** como sinónimo de **acción** y **posición** como sinónimo de **estado**.

Game theory-Two-player zero-sum games

Llamaremos a nuestros dos jugadores **MAX** y **MIN**

MAX mueve primero y luego los jugadores se turnan para moverse hasta que el juego termina. Al final del juego, se otorgan puntos al jugador ganador y se aplican penalizaciones al perdedor.

Un juego se puede definir formalmente con los siguientes elementos:

- S_0: The **initial state**, which specifies how the game is set up at the start.
- TO-MOVE(s): The player whose turn it is to move in state s.
- ACTIONS(s): The set of legal moves in state s.
- RESULT(s, a): The **transition model**, which defines the state resulting from taking action a in state s.
- IS-TERMINAL(s): A **terminal test**, which is true when the game is over and false otherwise. States where the game has ended are called **terminal states**.
- UTILITY(s, p): A **utility function** (also called an objective function or payoff function), which defines the final numeric value to player p when the game ends in terminal state s

Game theory-Two-player zero-sum games

Al igual que en los algoritmos de búsqueda el estado inicial, la función **ACTIONS** y la función **RESULT** definen el **grafo del espacio de estados**, un grafo donde los vértices son estados, los bordes son movimientos y un estado puede ser alcanzado por múltiples caminos.

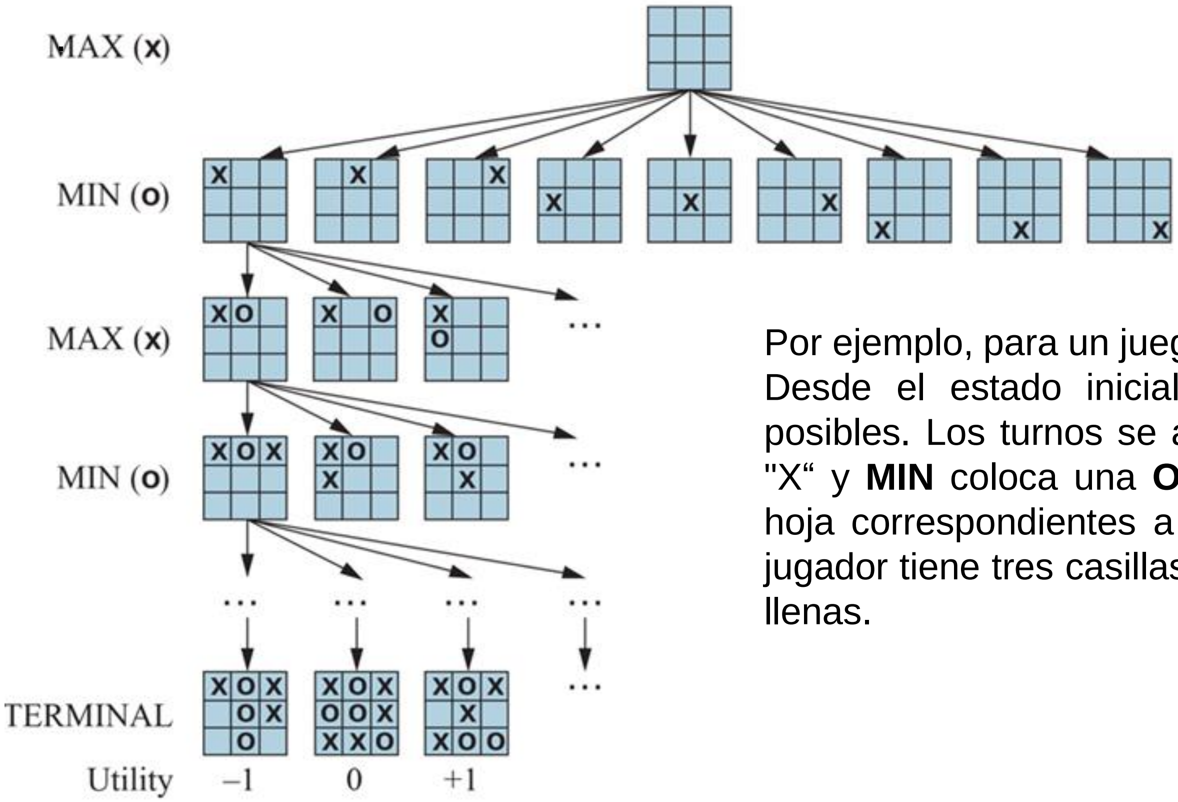
Podemos superponer un **árbol de búsqueda** sobre una parte de ese grafo para determinar qué movimiento realizar.

Definimos el **árbol de juego** completo como un árbol de búsqueda que sigue cada secuencia de movimientos hasta un estado terminal.

$$\text{ACTIONS}(Arad) = \{ToSibiu, ToTimisoara, ToZerind\}.$$

$$\text{RESULT}(Arad, ToZerind) = Zerind.$$

Game theory-Two-player zero-sum games



Por ejemplo, para un juego de ‘Triqui’
Desde el estado inicial, **MAX** tiene nueve movimientos posibles. Los turnos se alternan entre **MAX** colocando una "X“ y **MIN** coloca una **O** hasta que alcanzamos los nodos hoja correspondientes a los estados terminales, donde un jugador tiene tres casillas en línea o todas las casillas están llenas.

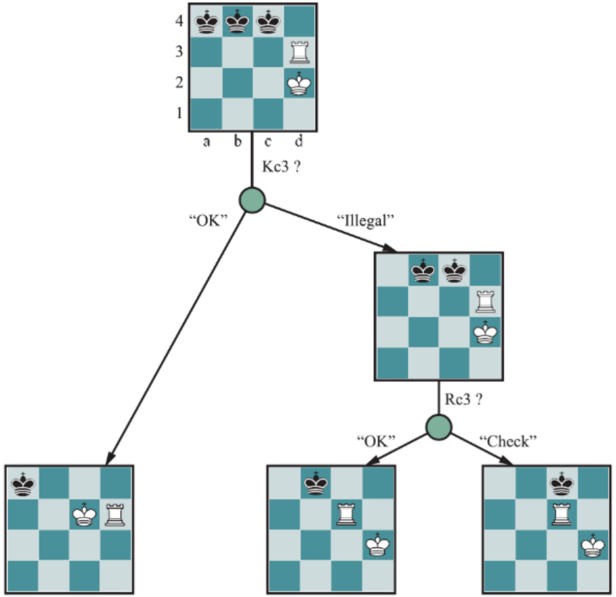
GAME THEORY-Decisiones Optimas en Juegos

Game theory- Decisiones Optimas en Juegos

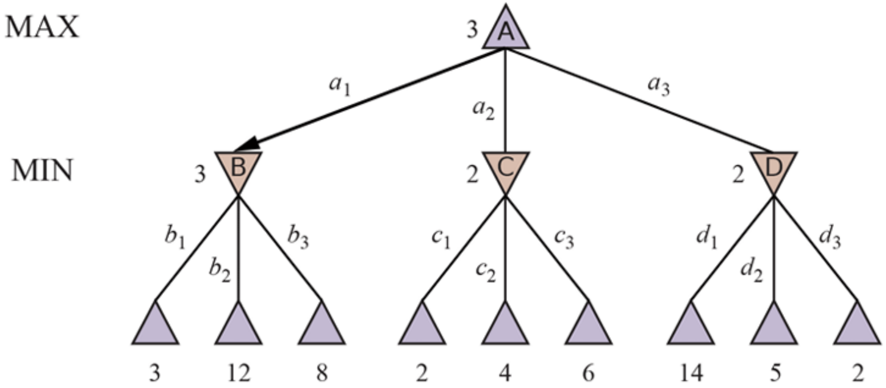
MAX quiere encontrar una secuencia de acciones que lo lleven a ganar, pero **MIN** tiene algo que decir al respecto.

Esto significa que la estrategia de **MAX** debe ser un plan condicional: una estrategia contingente que especifique una respuesta a cada uno de los movimientos posibles de **MIN**.

Para juegos con múltiples puntuaciones de resultado, necesitamos un algoritmo llamado **búsqueda minimax**.



Game theory- Decisiones Optimas en Juegos



Los movimientos posibles para **MAX** en el nodo raíz están etiquetados como $\mathbf{a_1}$**,** $\mathbf{a_2}$ y $\mathbf{a_3}$.
Las posibles respuestas a $\mathbf{a_1}$ por parte de **MIN** son $\mathbf{b_1}$**,** $\mathbf{b_2}$**,** $\mathbf{b_3}$, y así sucesivamente. Este juego en particular termina después de que **MAX** y **MIN** realicen un movimiento cada uno.

Dado un árbol de juego, la estrategia óptima puede determinarse calculando el **valor minimax** de cada estado en el árbol, el cual escribimos como

MINIMAX(s).

Game theory- Decisiones Optimas en Juegos

El valor minimax es la utilidad (para **MAX**) de estar en ese estado, **suponiendo que ambos jugadores juegan de manera óptima** desde allí hasta el final del juego.
El valor minimax de un estado terminal es simplemente su utilidad.
En un estado no terminal:

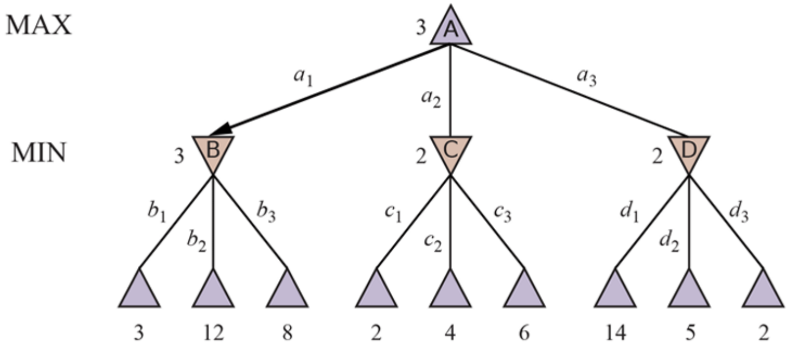
- **MAX** prefiere moverse al estado con el valor máximo cuando es su turno.
- **MIN** prefiere moverse al estado con el valor mínimo (es decir, el valor mínimo para **MAX**, que a su vez es el máximo para **MIN**).

$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s, \textsc{max}) & \text{if } \textsc{Is-Terminal}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s, a)) & \text{if } \textsc{To-Move}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s, a)) & \text{if } \textsc{To-Move}(s) = \textsc{min} \end{cases}$$

Game theory- Decisiones Optimas en Juegos

Apliquemos estas definiciones al árbol de juego
Los nodos terminales en el nivel inferior obtienen sus valores de utilidad desde la función **UTILITY** del juego.

- El primer nodo **MIN**, etiquetado como **B**, tiene tres estados sucesores con valores **3, 12 y 8**, por lo que su valor **minimax** es **3**.
- De manera similar, los otros dos nodos **MIN** tienen un valor minimax de **2**.
- La raíz es un nodo **MAX**; sus estados sucesores tienen valores minimax de **3, 2 y 2**, por lo que su valor minimax es **3**.
- También podemos identificar la **decisión minimax** en la raíz: la acción $\mathbf{a_1}$ es la mejor opción para **MAX**, ya que conduce al estado con el valor minimax más alto.



Game theory- Decisiones Optimas en Juegos

La definición de juego óptimo para **MAX** asume que **MIN** también juega de manera óptima.

¿Qué pasa si **MIN** no juega de manera óptima?

- En ese caso, **MAX** hará al menos lo mismo que si enfrentara a un jugador óptimo, o incluso mejor.
- Sin embargo, esto no significa que siempre sea lo mejor seguir el movimiento óptimo minimax contra un oponente subóptimo.

Game theory- Decisiones Optimas en Juegos

Ahora que podemos calcular **MINIMAX(s)**, podemos convertirlo en un **algoritmo de búsqueda** que encuentre la mejor jugada para **MAX**:

- **Prueba todas las acciones posibles** y elige la que tenga el estado resultante con el valor **minimax** más alto.

- Es un algoritmo recursivo que:
  - **Explora hasta los nodos terminales** del árbol.
  - **Regresa los valores minimax** a medida que la recursión se deshace.

```
function MINIMAX-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state)
  return move

function MAX-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
    if v2 > v then
      v, move ← v2, a
  return v, move

function MIN-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
    if v2 < v then
      v, move ← v2, a
  return v, move
```

Game theory- Decisiones Optimas en Juegos

El algoritmo minimax realiza una exploración completa en profundidad del árbol de juego. Si la profundidad máxima del árbol es *m* y hay *b* movimientos legales en cada punto, entonces la complejidad temporal del algoritmo minimax es $O(b^m)$.

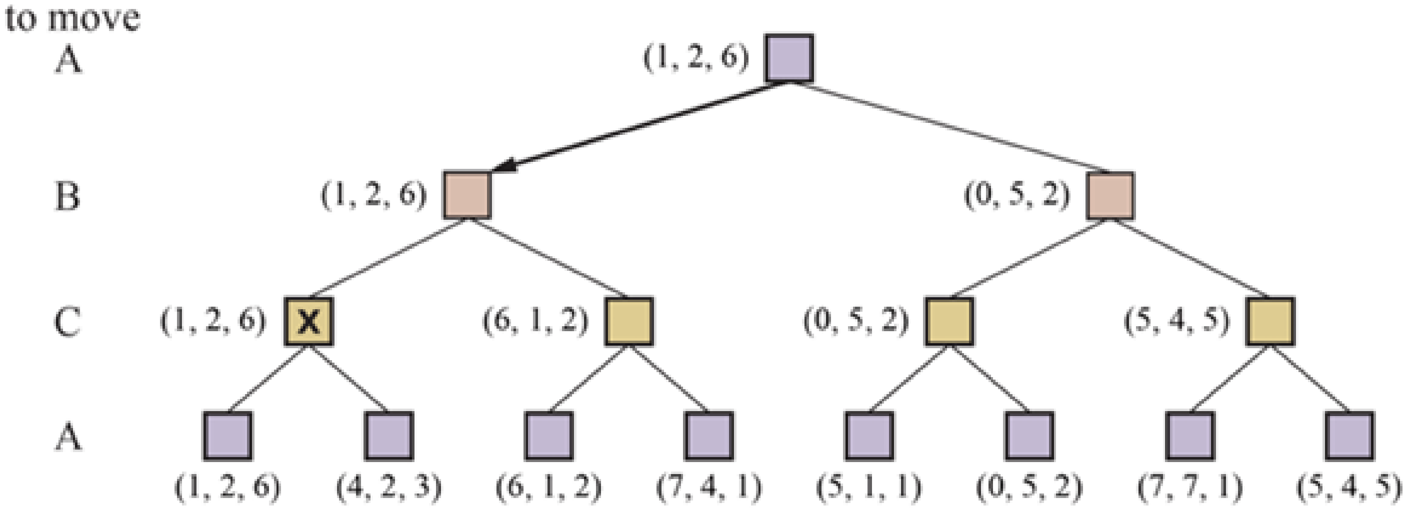
La complejidad espacial es O(bm) para un algoritmo que genera todas las acciones a la vez, o O(m) para un algoritmo que genera acciones una a la vez.

La complejidad exponencial hace que MINIMAX sea impráctico para juegos complejos; por ejemplo, el ajedrez tiene un factor de ramificación de aproximadamente 35 y el juego promedio tiene una profundidad de aproximadamente 80 jugadas, y no es factible buscar $35^{80} \approx 10^{123}$ estados.

Sin embargo, MINIMAX sirve como base para el análisis matemático de juegos. Al aproximar el análisis minimax de varias maneras, podemos derivar algoritmos más prácticos.

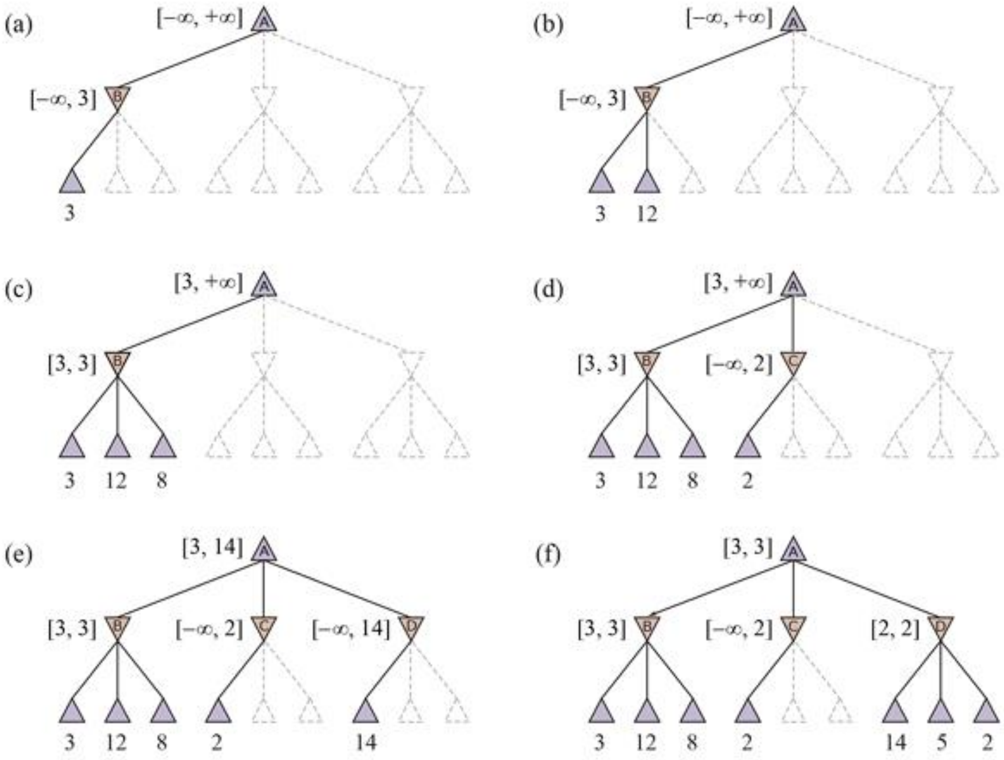
Game theory- Decisiones óptimas en juegos multijugador

Debemos reemplazar el valor único de cada nodo con un **vector** de valores. Por ejemplo, en un juego de tres jugadores con los jugadores **A, B** y **C**, se asocia un vector **(vA, vB, vC)** con cada nodo. Para los estados terminales, este vector proporciona la utilidad del estado desde el punto de vista de cada jugador. (En juegos de dos jugadores y suma cero, el vector de dos elementos se puede reducir a un solo valor, ya que los valores siempre son opuestos). La forma más sencilla de implementar esto es hacer que la función **UTILITY** devuelva un vector de utilidades.



GAME THEORY-Poda Alpha-Beta

Game theory- Poda Alpha Beta



$$
\begin{aligned}
\textsc{Minimax}(root) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\
&= \max(3, \min(2,x,y), 2) \\
&= \max(3, z, 2) \qquad \text{where } z = \min(2,x,y) \le 2 \\
&= 3.
\end{aligned}
$$

Game theory- Poda Alpha Beta

El nombre de la poda *alfa–beta* proviene de los dos parámetros extra en **MAX-VALUE** *(state, α, β)* (ver Figura 5.7) que describen los límites de los valores respaldados que aparecen en cualquier punto del camino:

- **α** = el valor de la mejor elección (es decir, la de **mayor valor**) que hemos encontrado hasta ahora en cualquier punto de elección en el camino para **MAX**. Piensa en **α** como "al menos".
- **β** = el valor de la mejor (es decir, menor valor) elección que hemos encontrado hasta ahora en cualquier punto de elección en el camino para **MIN**. Piensa en **β** como "como mucho".

```
function ALPHA-BETA-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state, −∞, +∞)
  return move

function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
    if v2 > v then
      v, move ← v2, a
      α ← MAX(α, v)
    if v ≥ β then return v, move
  return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
    if v2 < v then
      v, move ← v2, a
      β ← MIN(β, v)
    if v ≤ α then return v, move
  return v, move
```

Inteligencia Artificial

https://stability.ai/blog/stable-diffusion-public-release

https://openai.com/blog/chatgpt/

https://ai.googleblog.com/2022/06/minerva-solving-quantitative-reasoning.html

https://www.youtube.com/watch?v=jMvLCZBXbtc

https://www.deepmind.com/blog/tackling-multiple-tasks-with-a-single-visual-language-model

https://www.deepmind.com/blog/building-safer-dialogue-agents

https://www.deepmind.com/publications/a-generalist-agent

https://www.deepmind.com/research/highlighted-research/alphago

https://ai.facebook.com/research/cicero/

https://openai.com/blog/whisper/

https://valle-demo.github.io/