

---

Inteligencia Artificial

# Juegos

---

Jorge Luis Guevara Diaz

[www.jorge.sistemasyservidores.com](http://www.jorge.sistemasyservidores.com)

---

# Que veremos hoy?

- Juegos
  - Juegos Perfectos
    - Desiciones minimax
    - Poda  $\alpha$ - $\beta$
  - Recursos Limitados y evaluación aproximada
-

---

# Juegos

- Son un ejemplo de entornos multiagente
    - ¿Qué es lo que hacen otros agentes y como afecta a nuestro éxito?
    - Entornos multiagente cooperativos vs competitivos
    - Entornos multiagente competitivos da lugar a una búsqueda entre adversarios ejem juegos
  - Porque estudiar juegos?
    - Es divertido
    - Interesante materia de estudio, son difíciles
    - Fácil de representar y los agentes están restringidos a un número pequeño de acciones
-

---

# Juegos vs Búsqueda

## ■ Búsqueda – no hay adversarios

- ❑ Solución es un método para encontrar una meta
- ❑ Heurísticas y técnicas PSR pueden encontrar soluciones óptimas
- ❑ Función de evaluación estima el costo del nodo inicial a la meta dado un nodo
- ❑ Ejemplos: planeamiento de rutas, planificación de actividades

## ■ Juegos – existen adversarios

- ❑ La solución es una estrategia (la estrategia especifica el movimiento para cada posible réplica del oponente)
  - ❑ Los límites del tiempo fuerzan a una solución aproximada
  - ❑ Función de evaluación: evalúa la bondad de la posición del juego
  - ❑ Ejemplo : ajedrez, tres en raya, etc
-

---

# Tipos de Juegos

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

---

---

# Juegos vs problemas de Búsqueda

- Planes de ataque:

- La computadora considera las posibles líneas del juego (Babbage, 1846)
  - Algoritmo para juego perfecto (Zermelo, 1912; Von Neumann, 1944)
  - Horizonte finito, evaluación aproximada (Zuse, 1945; Wiener, 1948; Shannon, 1950)
  - Primer programa de ajedrez (Turing, 1951)
  - Machine learning para mejorar la precisión de la evaluación (Samuel, 1952{57)
  - Poda para permitir búsqueda más profunda (McCarthy, 1956)
-

---

# Juegos como problemas de búsqueda

- Dos jugadores : MAX y MIN  
Max mueve primero y luego se toman turnos hasta que el juego acabe
  - **Juegos como Búsqueda**
    - **Estado inicial** : ejemplo configuración inicial del tablero de ajedrez
    - **Función sucesor** : lista de pares (movimientos, estados) especificando movimientos legales
    - **Test terminal**: el juego esta terminado?
    - **Función de utilidad**: dar un valor numérico al estado terminal, ejemplo, ganador +1, perdedor -1 empates 0
  - Max usa un **árbol de búsqueda** para determinar el siguiente movimiento
-



# Estrategias Óptimas

Buscar una estrategia contingente para Max asumiendo un oponente infalible MIN

Asumpcion: ambos oponentes juegan de manera óptima

Dado un árbol de juego, la estrategia óptima puede ser determinada usando el valor MINIMAX de cada nodo

Idea: Escoger el movimiento con mas alto valor minimax

VALOR-MINIMAX( $n$ ) =

UTILILIDAD( $n$ )

Si  $n$  es un

nodo terminal

$\max_{s \in \text{sucesores}(n)} \text{VALOR-MINIMAX}(s)$

Si  $n$  es un nodo max

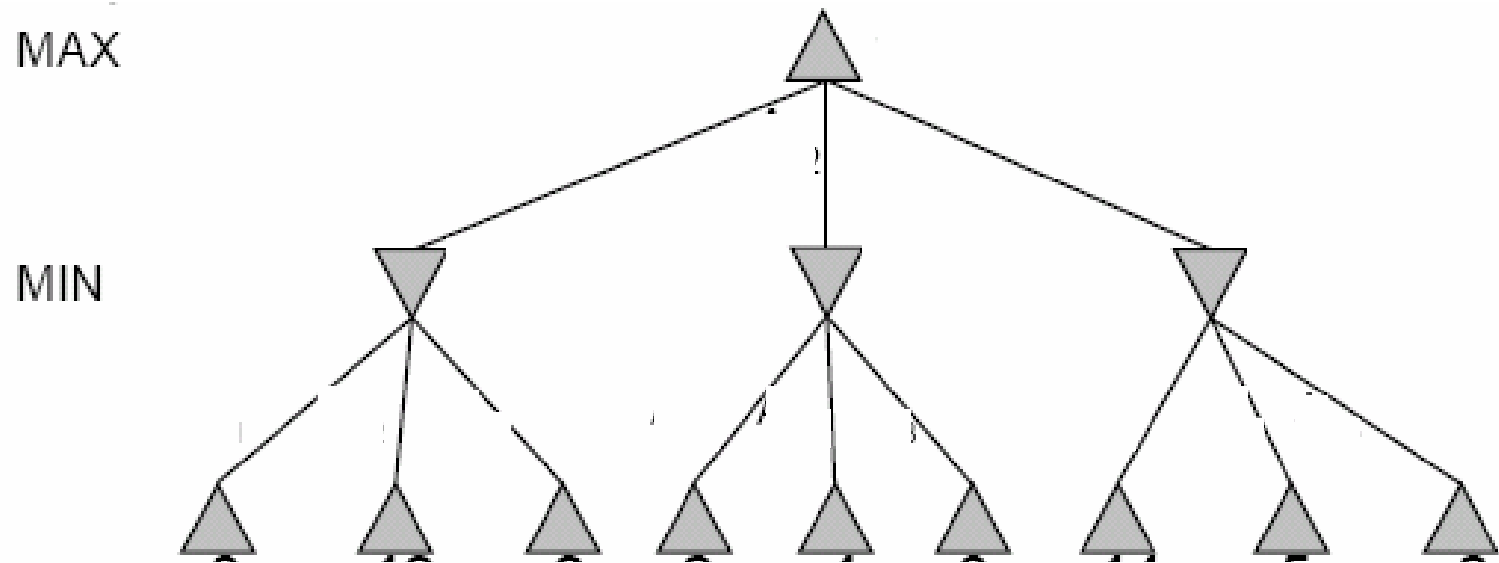
$\min_{s \in \text{sucesores}(n)} \text{VALOR-MINIMAX}(s)$

Si  $n$  es un nodo min

---

# Decisión Minimax

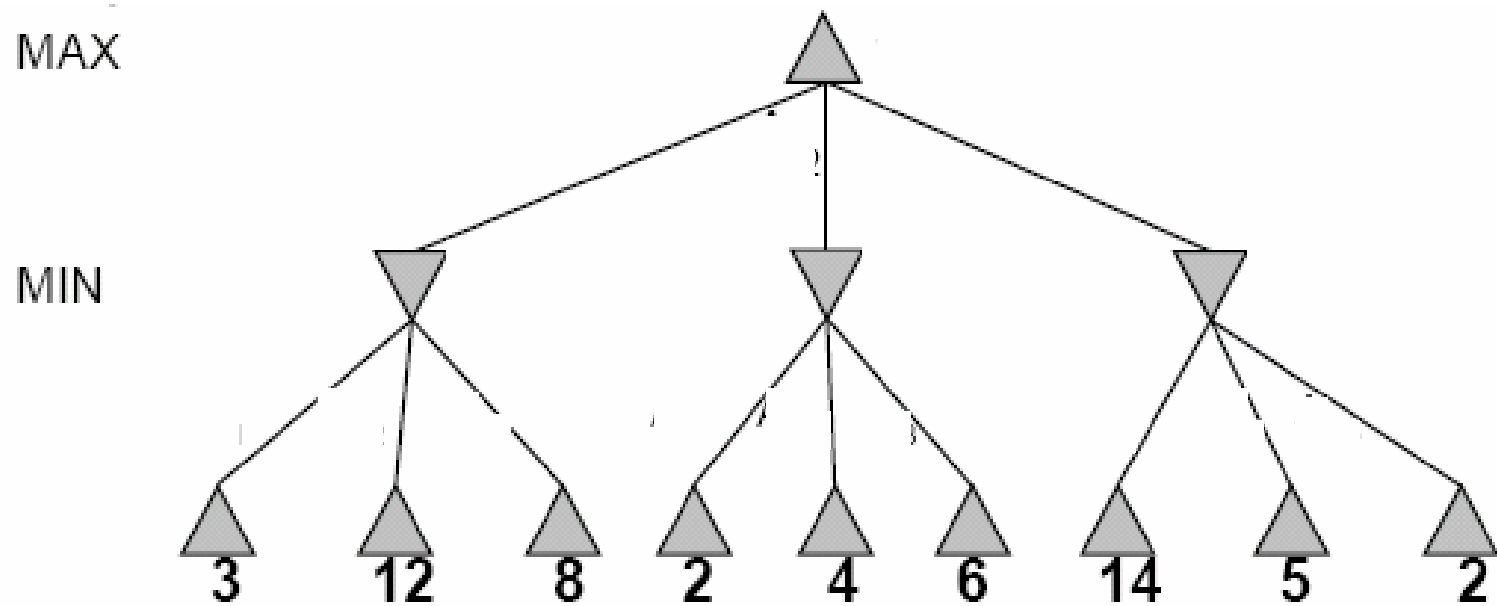
## Ejemplo 2-ply



---

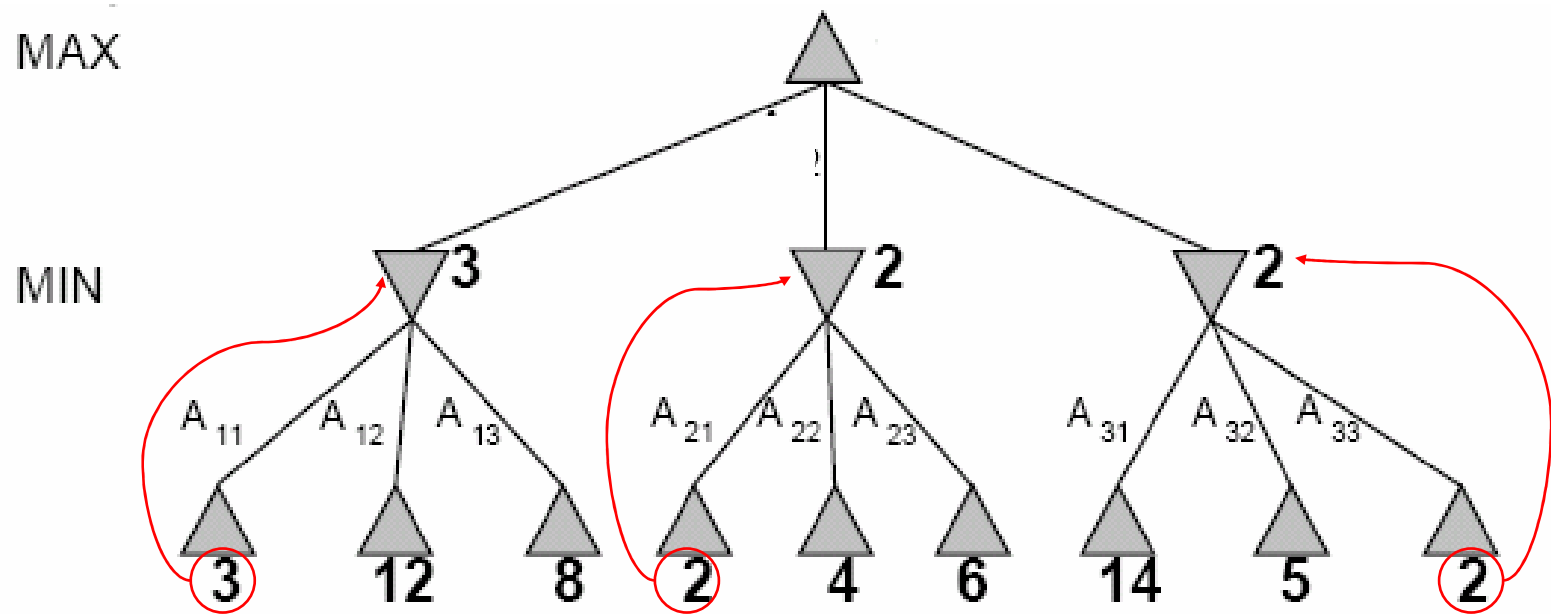
# Decisión Minimax

## Ejemplo 2-ply



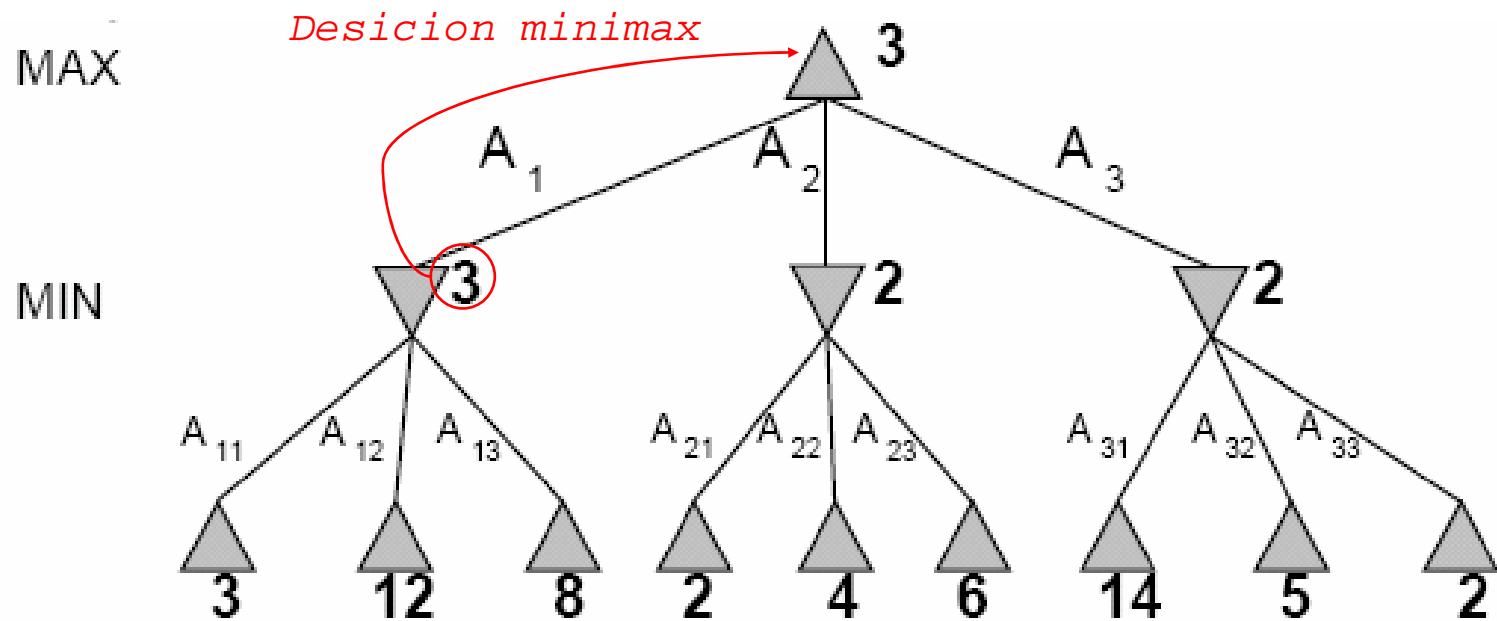
# Decisión Minimax

## Ejemplo 2-ply



# Decisión Minimax

## Ejemplo 2-ply



*Minimax maximiza el peor caso del resultado para max.*

---

# Que pasa si MIN no juega de manera óptima?

Definición de juego óptimo para MAX asume que MIN juega óptimamente: maximiza el peor caso de resultado para MAX. Pero si MIN no juega óptimamente, MAX podrá hacerlo mucho mejor. [probar]

## Función de evaluación para el tres en raya

Se puede considerar lo siguiente:

$$100A+10B+C-(100D+10E+F)$$

**A** = número de líneas con 3 X's

**B** = número de líneas no bloqueadas con un par de X's

**C** = número de líneas no bloqueadas con una sola X

Similarmente para D, E y F pero con las configuraciones de las O's

---

# Algoritmo Minimax

**funcion** DESICION-MINIMAX(*estado*) **retornar** *una accion*

**entrada:** *estado*, actual estado en el juego

$v \leftarrow \text{VALOR-MAX}(\text{estado})$

**retornar** la *accion* en SUCESORES(*estado*) con valor  $v$

---

**funcion** VALOR-MAX(*estado*) **retornar** *un valor de utilidad*

**si** TEST-TERMINAL(*estado*) **entonces** **retornar** UTILIDAD(*estado*)

$v \leftarrow \infty$

**for**  $a, s$  en SUCESORES(*estado*) **do**

$v \leftarrow \text{MAX}(v, \text{VALOR-MIN}(s))$

**retornar**  $v$

---

**function** VALOR-MIN(*estado*) **retornar** *un valor de utilidad*

**si** TEST-TERMINAL(*estado*) **entonces** **retornar** UTILIDAD(*estado*)

$v \leftarrow \infty$

**for**  $a, s$  en SUCESORES(*estado*) **do**

$v \leftarrow \text{MIN}(v, \text{VALOR-MAX}(s))$

**retornar**  $v$

---

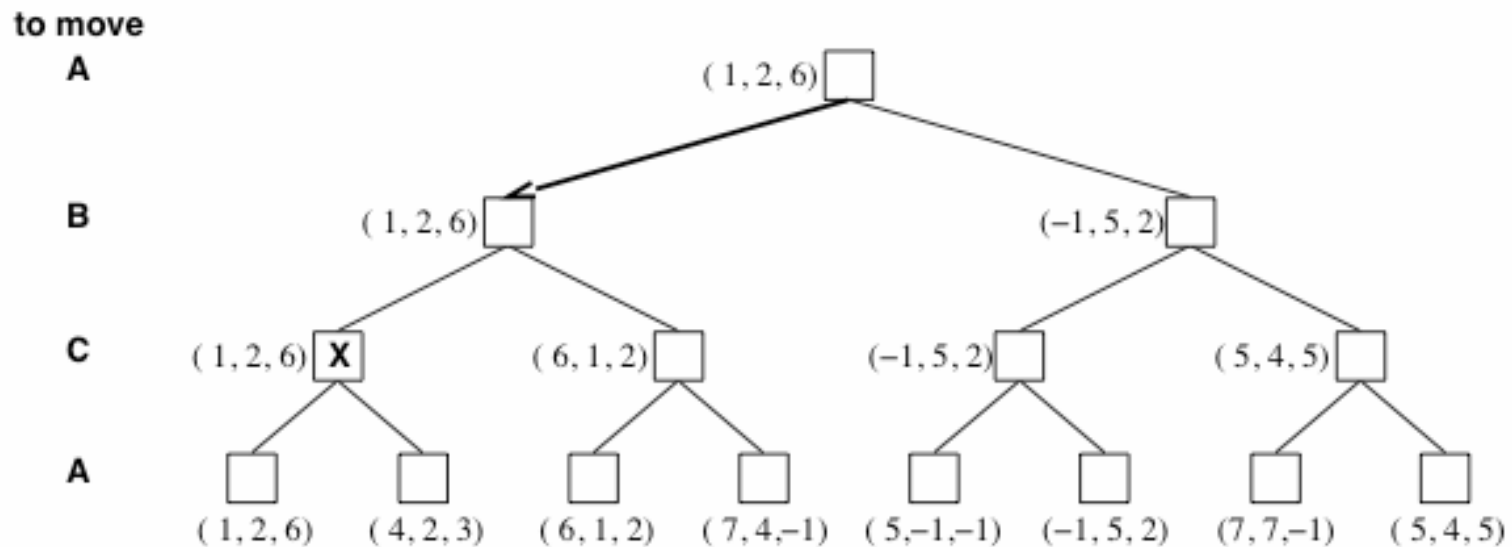
---

# Propiedades del minimax

- Completo? Si (si el árbol es finito)
  - Óptimo? Si (contra un oponente óptimo)
  - Complejidad de tiempo?  $O(b^m)$  😞
  - Complejidad de espacio?  $O(bm)$  (exploración primero en profundidad) 😊
  - 
  - Para ajedrez,  $b \approx 35$ ,  $m \approx 100$  para juegos “razonables” → solución exacta completamente inviable.
-

# Juegos Multijugador

- Juegos permiten mas de dos jugadores
- Los valores minimax ahora son vectores



---

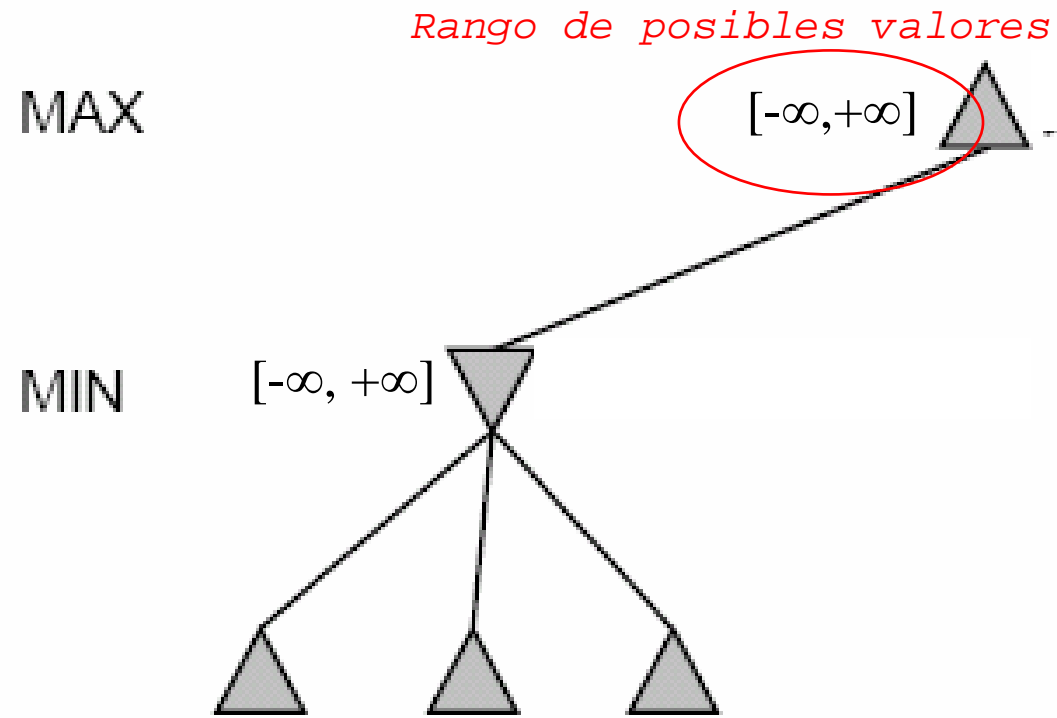
# Problema de la Búsqueda Minimax

- Número de estados de juegos es exponencial al número de movimientos
    - Solución: No examinar cada nodo
    - ==> poda Alfa-beta
      - Eliminar ramas que no tengan influencia final en la decisión
  - ejemplo
-

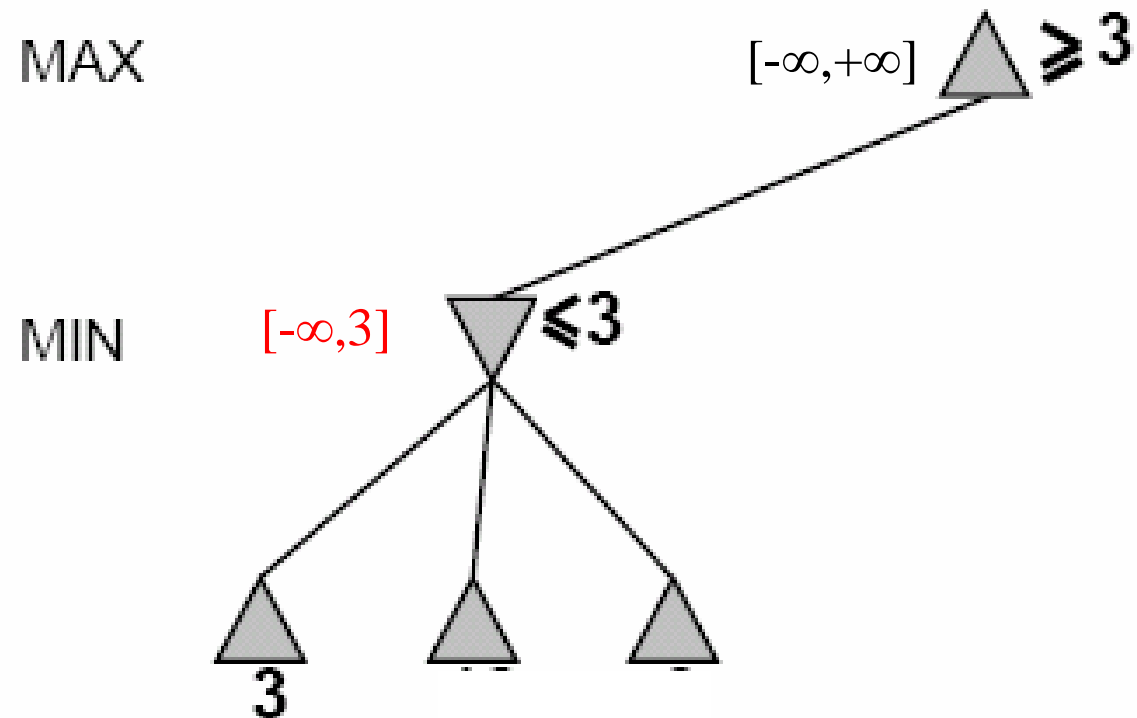
---

# Alfa-Beta Ejemplo

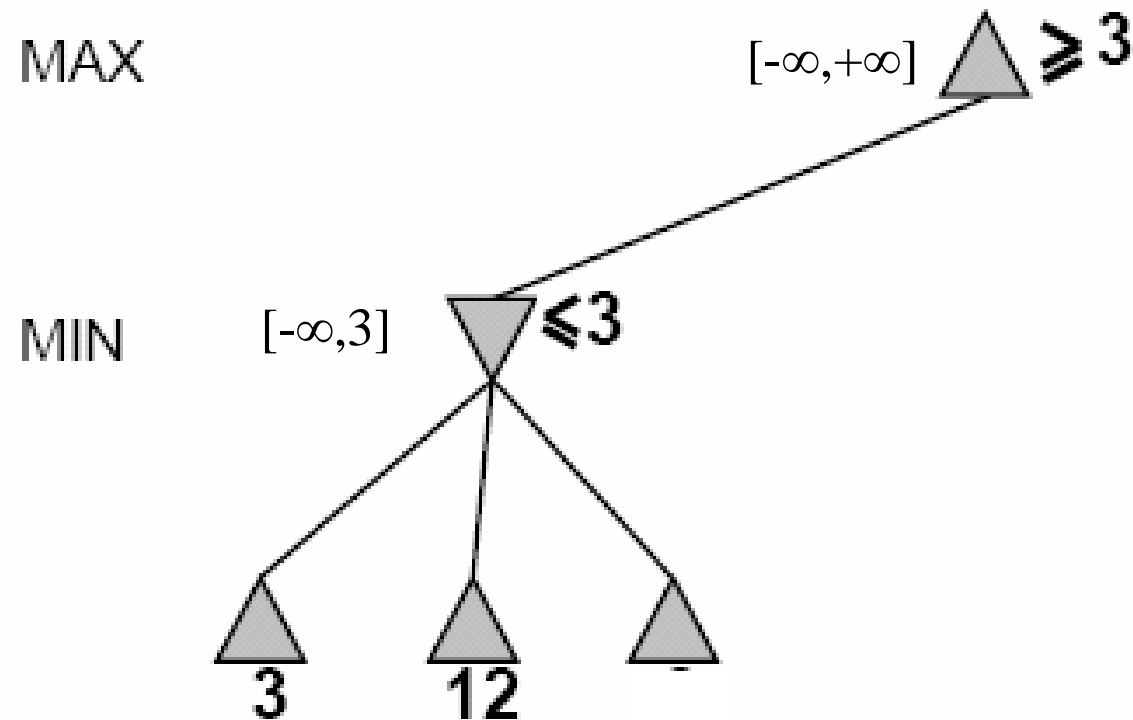
**Hacer búsqueda primero en profundidad hasta la primera hoja**



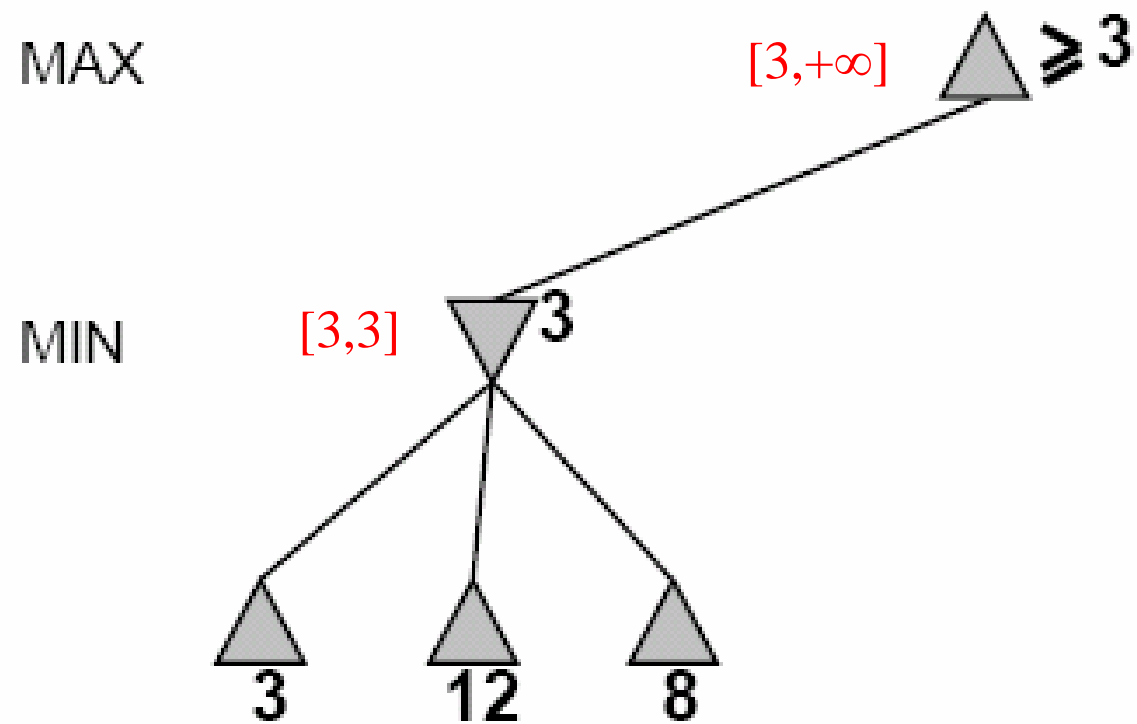
# Alfa-Beta Ejemplo (continuación)



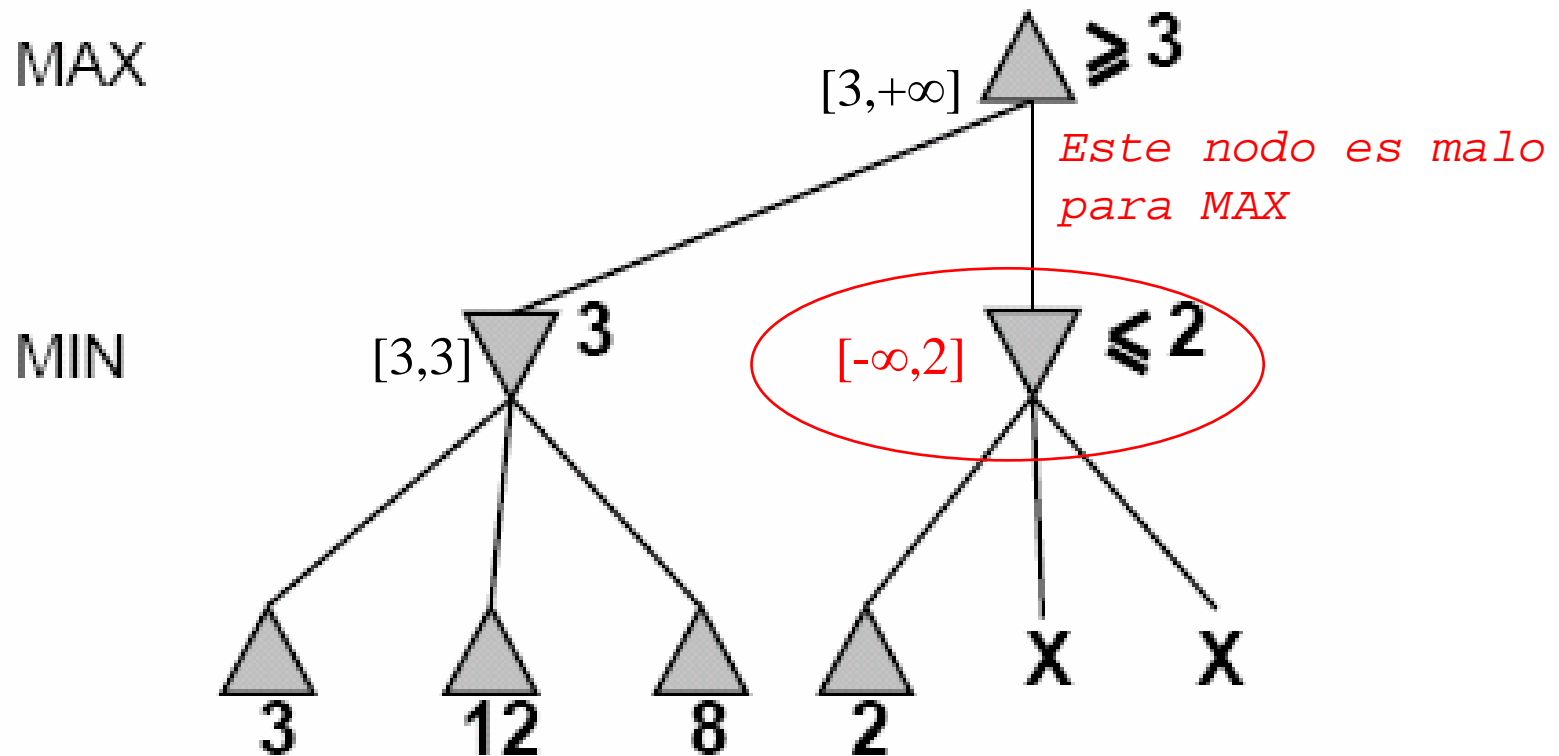
# Alfa-Beta Ejemplo (continuación)



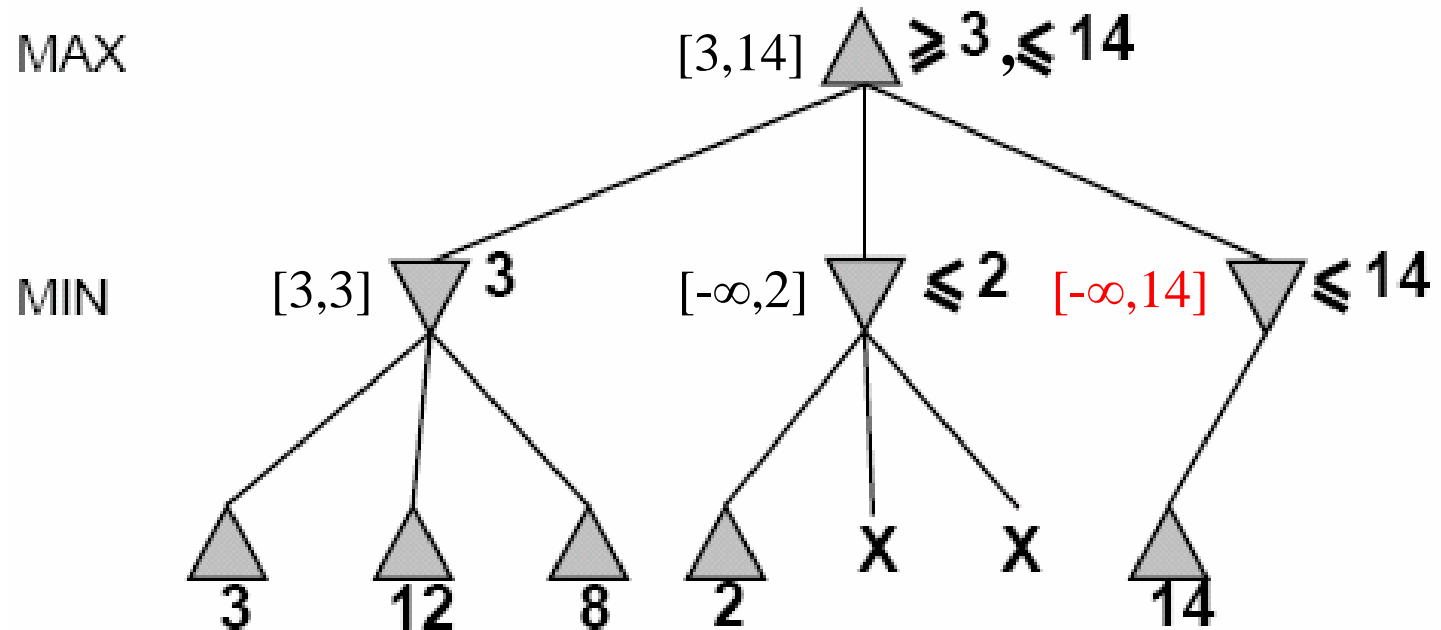
# Alfa-Beta Ejemplo (continuación)



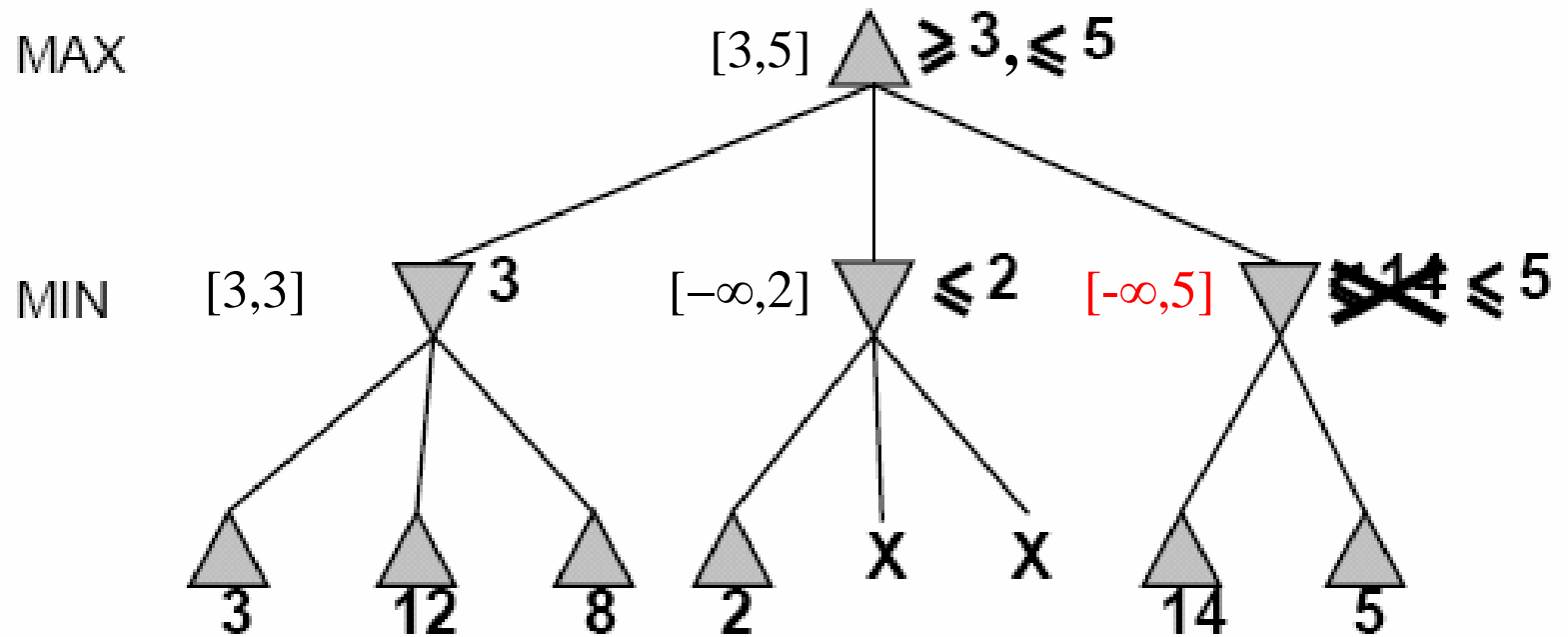
# Alfa-Beta Ejemplo (continuación)



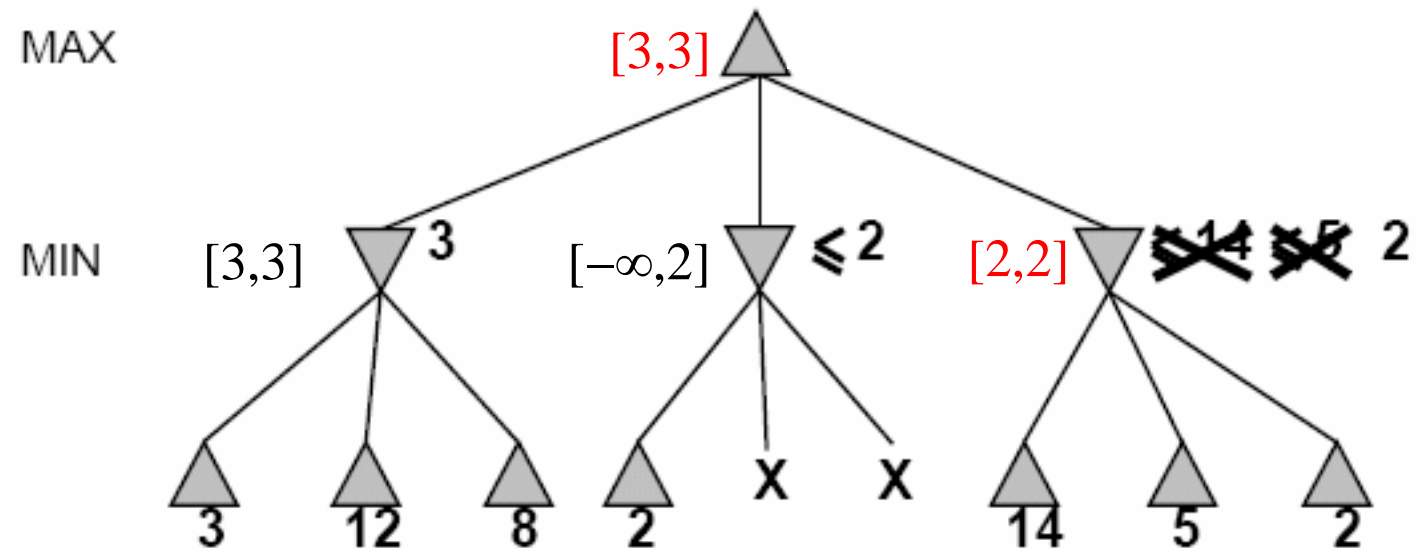
# Alfa-Beta Ejemplo (continuación)



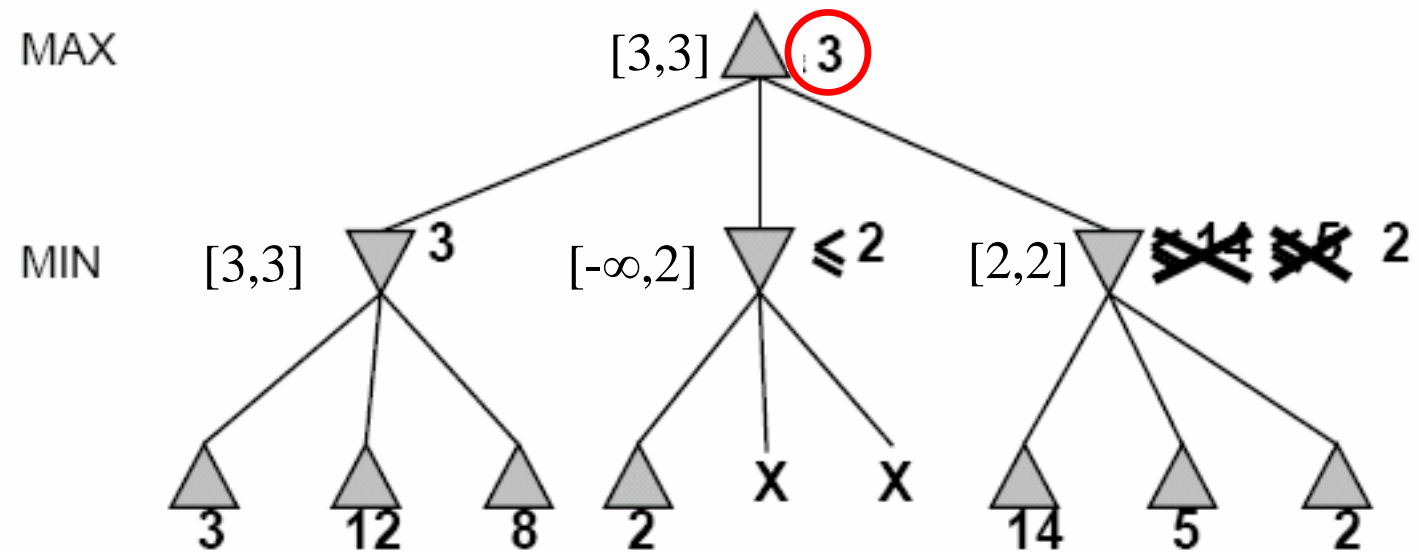
# Alfa-Beta Ejemplo (continuación)



# Alfa-Beta Ejemplo (continuación)



# Alfa-Beta Ejemplo (continuación)



# Algoritmo Poda Alfa-Beta

**function** BUSQUEDA-ALFA-BETA (*estado*) **retornar** *una accion*  
**inputs:** *estado*, estado actual en el juego  
 $v \leftarrow \text{MAX-VALOR}(\text{estado}, -\infty, +\infty)$   
**return** la *accion* en SUCESORES(*estado*) con valor  $v$

---

**function** MAX-VALOR(*estado*,  $\alpha$ ,  $\beta$ ) **retornar** *un valor utilidad*  
**if** TEST-TERMINAL (*estado*) **then** **retornar** UTILIDAD(*estado*)  
 $v \leftarrow -\infty$   
**for**  $a, s$  en SUCESORES(*estado*) **do**  
     $v \leftarrow \text{MAX}(v, \text{MIN-VALOR}(s, \alpha, \beta))$   
    **if**  $v \geq \beta$  **then** **retornar**  $v$   
     $\alpha \leftarrow \text{MAX}(\alpha, v)$   
**retornar**  $v$

---

# Algoritmo Poda Alfa-Beta

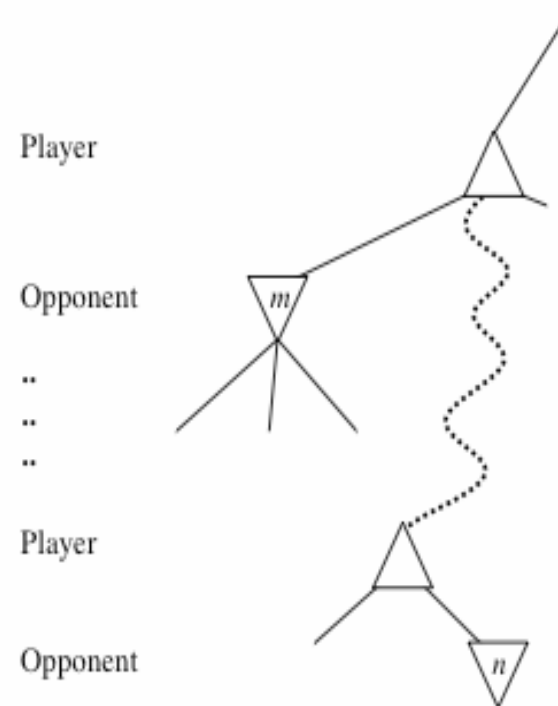
---

```
function MIN-VALOR(estado,  $\alpha$ ,  $\beta$ ) retornar un valor de utilidad
  if TEST-TERMINAL (estado) then retornar UTILIDAD(estado)
   $v \leftarrow +\infty$ 
  for a,s en SUCESORES(estado) do
     $v \leftarrow \text{MIN}(v, \text{MAX-VALOR}(s, \alpha, \beta))$ 
    if  $v \leq \alpha$  then retornar  $v$ 
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  retonar  $v$ 
```

---

# Poda Alfa-Beta General

- Considerar un nodo  $n$  en algún lugar en el árbol
- si el jugador tiene una mejor elección en
  - Nodo padre de  $n$
  - O cualquier elección encima de él
- $n$  nunca podrá ser alcanzado en el juego actual.
- Por lo tanto cuando se sabe lo suficiente acerca de  $n$ , este puede ser podado.



---

## Comentarios Finales acerca de la Poda Alfa-Beta

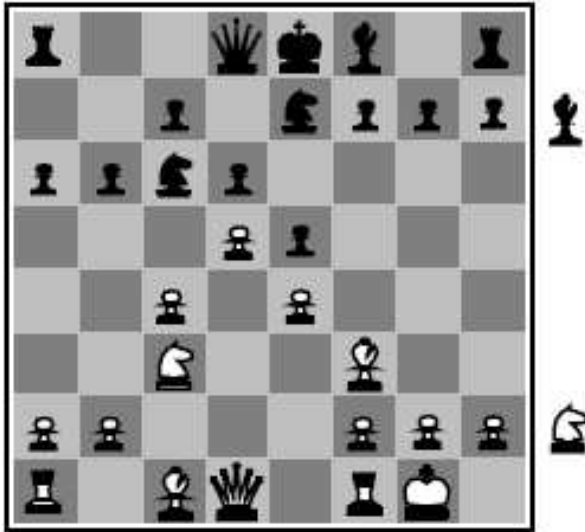
- La poda no afecta los resultados finales
  - Subárboles enteros pueden ser podados.
  - Buenos movimientos mejora la eficacia de la poda
  - Con “movimientos perfectos,” la complejidad del tiempo es  $O(b^{m/2})$ 
    - Factor de ramificación es  $\sqrt{b}$  !!
    - La Poda Alfa-Beta puede buscar dos veces mas que el algoritmo minimax en el mismo monto de tiempo
  - Los estados repetidos son todavía posibles.
    - Guardarlos en memoria = tabla de transposición
-

---

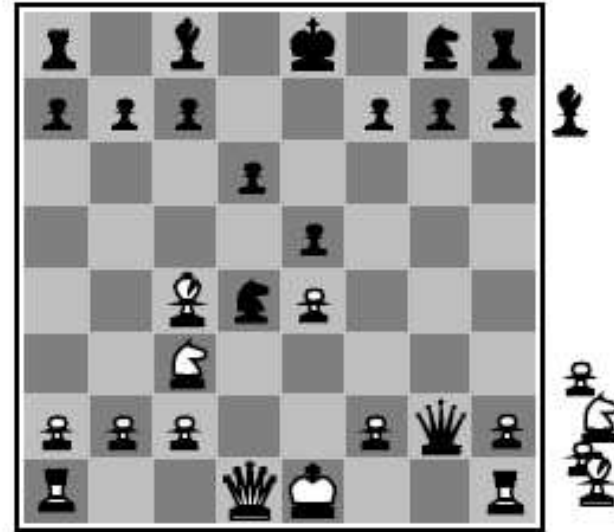
# Recursos Limitados y evaluación aproximada

- Aproximacion estándar:
  - SHANNON (1950):
    - Usar Test-de-Corte en lugar de Test-Terminal
      - Búsqueda en Profundidad limitada (agregar [quiescence search](#))
    - Usar EVAL en lugar de UTILIDAD
      - Función de evaluación que estima la deseabilidad de determinada posición
  
  - cambio:
    - **if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)  
en
    - **if** CUTOFF-TEST(*state,depth*) **then return** EVAL(*state*)
-

# Funciones de evaluación



**Negras mueven**  
**Blancas ligeramente mejor**

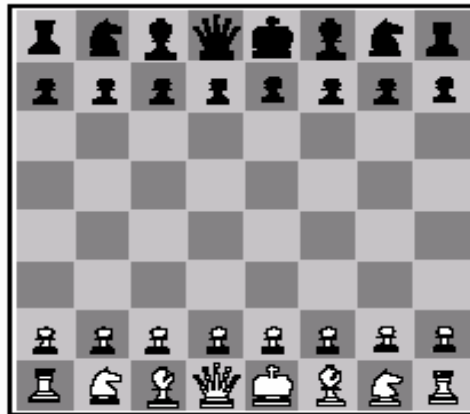


**Blancas Mueven**  
**Negras ganan**

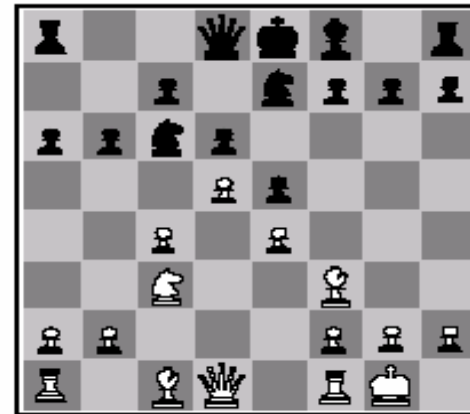
Para ajedrez la función de evaluación es una suma lineal ponderada de características

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

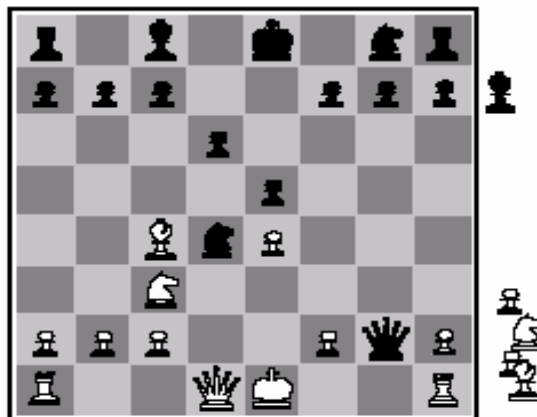
- e.g.,  $w_1 = 9$  con
- $f_1(s) = (\text{número de reinas blancas}) - (\text{número de reinas negras})$ , etc.



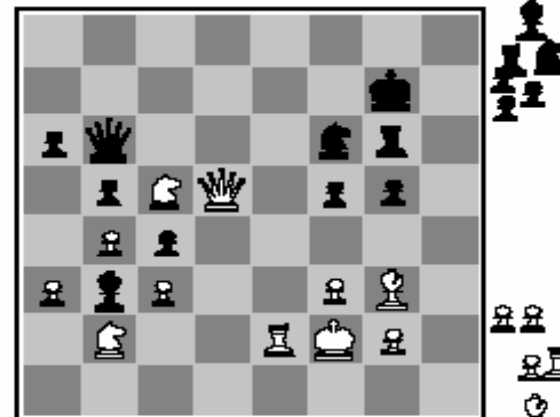
(a) White to move  
Fairly even



(b) Black to move  
White slightly better

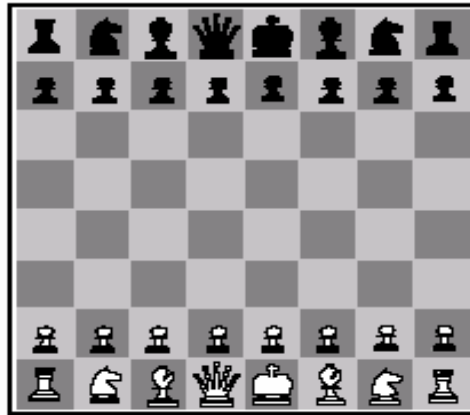


(c) White to move  
Black winning

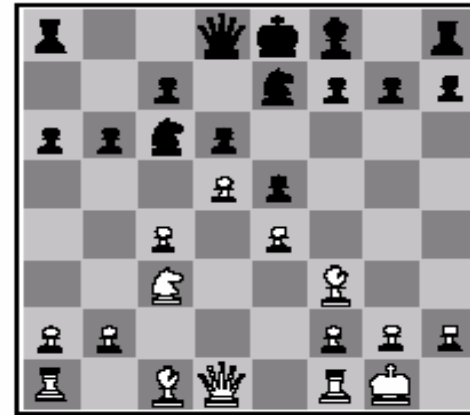


(d) Black to move  
White about to lose

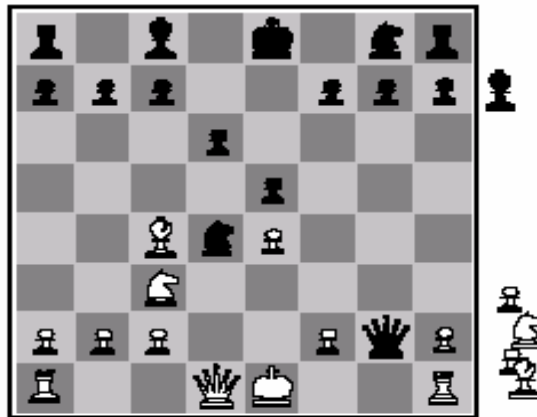
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$



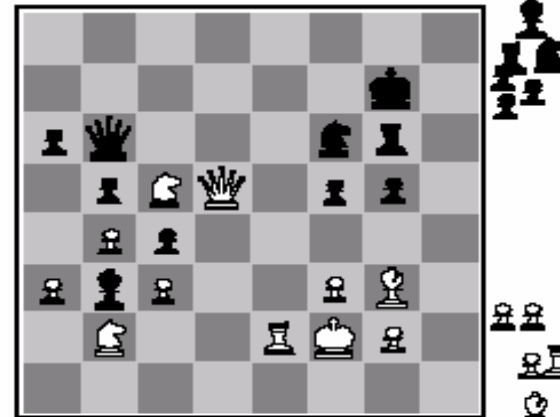
(a) White to move  
Fairly even



(b) Black to move  
White slightly better



(c) White to move  
Black winning

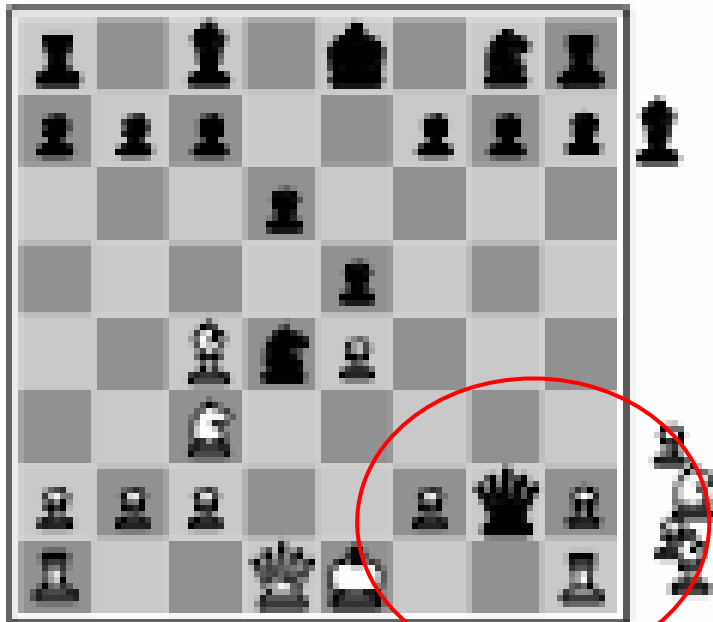


(d) Black to move  
White about to lose

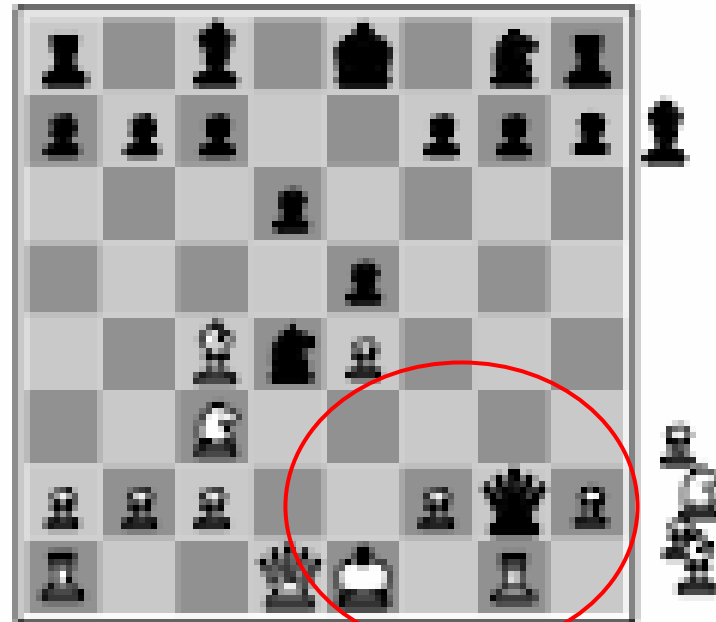
*Adición asume independencia*

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

# Dificultades heurísticas



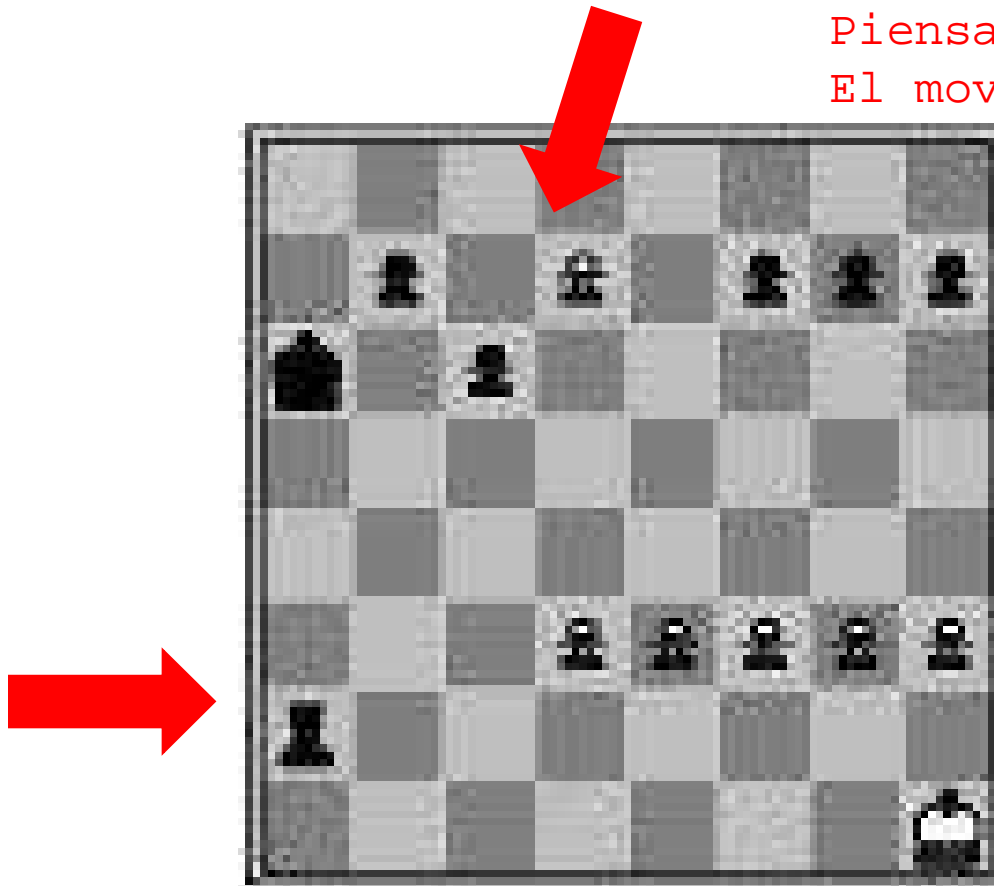
(a) White to move



(b) White to move

# Efecto Horizonte

Búsqueda en profundidad fija  
Piensa que puede evitar  
El movimiento de la reina

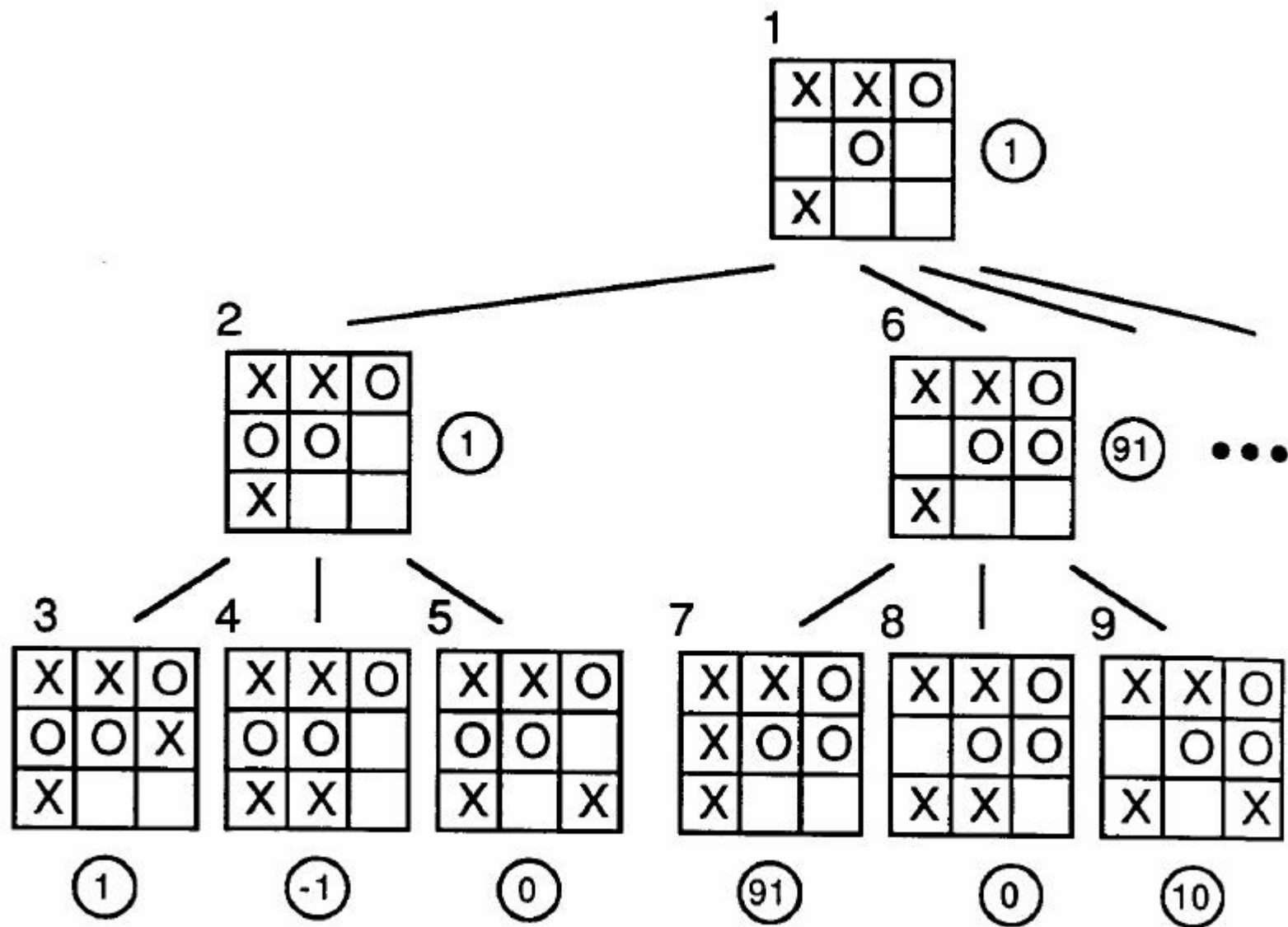


There is more

---

## Ejemplo para el juego 3 en raya

- Cada nivel de nodos en el árbol se conoce como “ply”
  - Ply 0 contiene solo un nodo, correspondiente a la actual posición del tablero del juego
  - Ply 1 contiene los hijos de la raíz del árbol
  - Típicamente un programa de juegos, puede generar todas las posiciones de la pizarra para nodos en un determinado ply como 4
-



---

# Caso : Deep Blue

- Murray Campbell, Feng-Hsiung Hsu, Joseph Hoane IBM
  - Computadora paralela con 30 procesadores IBM RS/6000 corriendo el software de búsqueda
  - 480 procesadores VLSI de ajedrez a “medida” que realizaban la generación de movimientos
  - Deep Blue buscaba 126 000 000 nodos/segundo en promedio, con una velocidad pico de 330 000 000 nodos/segundo
  - Deep Blue generaba 30 000 000 000 de posiciones por movimiento, alcanzando la profundidad 14 rutinariamente
  - Algoritmos base :
    - Búsqueda en profundidad interativa
    - Poda alfabeta con tabla de trasposición
    - Generaba extensiones mas allá del límite de búsqueda (profundidad de 40 plies)
    - Función de evaluación sobre 8000 características, muchas de ellas describían patrones
    - Libro de 4000 posiciones
    - Base de datos de 700 000 jugadas grandmaster de la cual se podian extraer recomendaciones
  - Mejoras en el algoritmo permiten utilizar PC's estandares con un nivel de juego a nivel de campeonato mundial
-

---

# Resumen

- Los juegos son divertidos (y peligrosos)
  - Ilustran importantes puntos en la IA
    - Perfeccion inalcanzable -> aproximación
    - Buena idea para pensar en lo que pensar
    - Incertidumbre restringe la asignación de valores a los estados
-

---

# Referencias Bibliográficas

- Capitulo 6 Artificial Intelligence: A Modern Approach, Russell and Norvig
  - <http://www.aaai.org/AITopics/pmwiki/pmwiki.php/AITopics/Search>
-