DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Árboles de juegos
Análisis y Diseño de Algoritmos

Tipos de juegos

	Juegos deterministas	Juegos de azar
Con información perfecta	Ajedrez, damas, Go, Othello	Backgammon, Monopoly
Con información imperfecta	barquitos	Bridge, poker, scrabble

Árboles de juegos

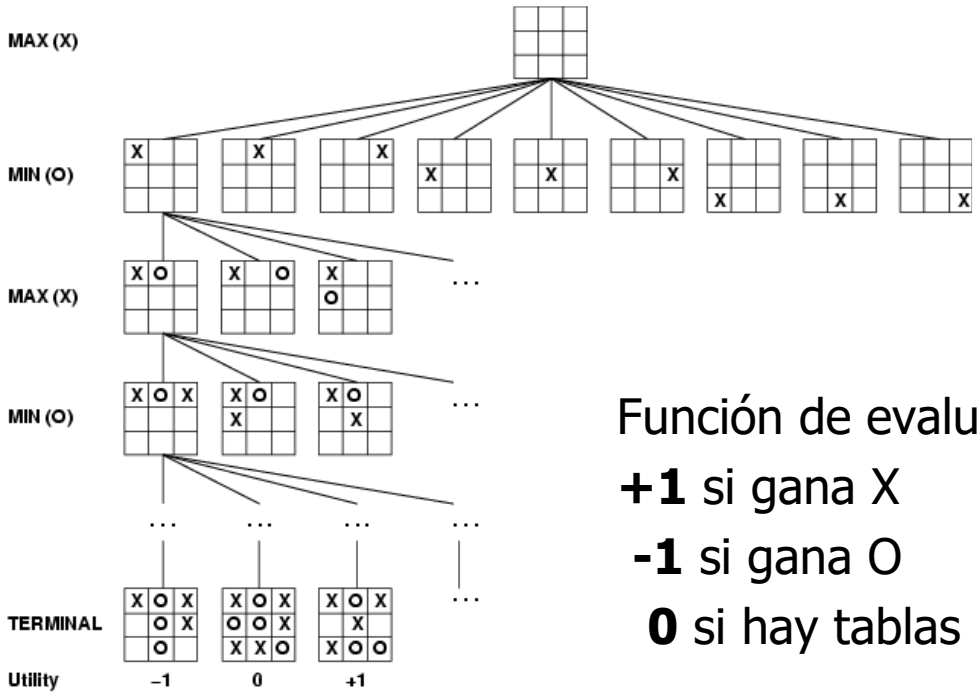
Juego perfecto

- Dos jugadores
- Movimientos intercalados
- Suma cero (la ganancia de uno es la pérdida del otro).
- Información perfecta (ambos jugadores tienen acceso a toda la información sobre el estado del juego: no se ocultan información el uno al otro).
- No interviene el azar (p.ej. dados).

Ejemplos:

Nim, Grundy, 3 en raya, conecta-4, damas, ajedrez…

Árboles de juegos



Función de evaluación:

+1 si gana X

-1 si gana O

0 si hay tablas

Árboles de juegos

Complejidad de algunos juegos

Juego	Estados
3 en raya	9! = 362280
Conecta-4	10^{13}
Damas	10^{18}
Ajedrez	10^{50}
Go	10^{170}

Minimax

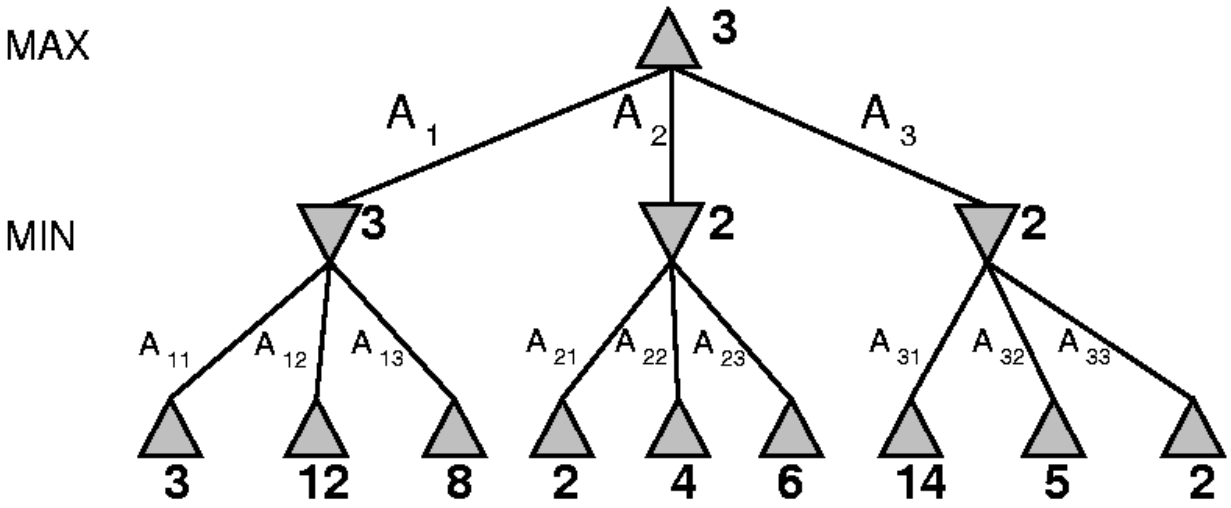
Estrategia perfecta para juegos deterministas.

IDEA: Elegir el movimiento que nos lleva a la posición que nos asegura una recompensa máxima en el peor caso (valor minimax).

- MAX: Cuando movemos nosotros, elegimos el nodo de máximo valor.
- MIN: Cuando mueve nuestro oponente, elige el nodo de menor valor (para nosotros).

Minimax

Árbol con 2 niveles (2-ply):



Minimax

function MINIMAX-DECISION(*state*) **returns** *an action*

$v \leftarrow$ MAX-VALUE(*state*)
return the *action* in SUCCESSORS(*state*) with value v

---

function MAX-VALUE(*state*) **returns** *a utility value*

if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
$v \leftarrow -\infty$
for a, s in SUCCESSORS(*state*) **do**
$v \leftarrow$ MAX(v, MIN-VALUE(s))
return v

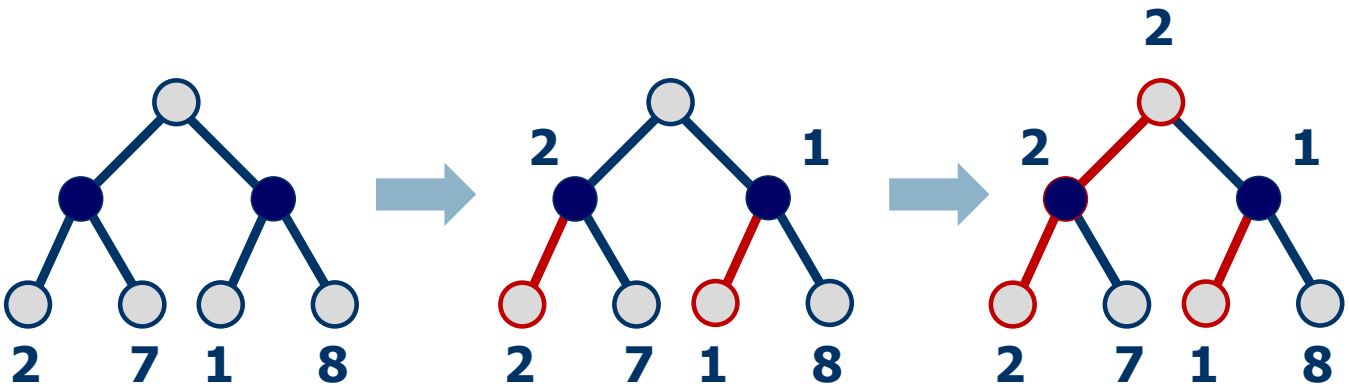
---

function MIN-VALUE(*state*) **returns** *a utility value*

if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
$v \leftarrow \infty$
for a, s in SUCCESSORS(*state*) **do**
$v \leftarrow$ MIN(v, MAX-VALUE(s))
return v

Minimax

Búsqueda minimax (primero en profundidad):



MAX
MIN

Minimax

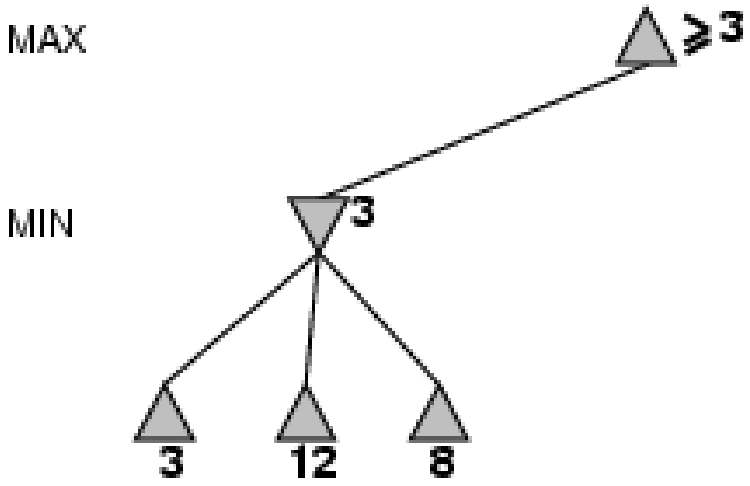
Complejidad

b = Factor de ramificación del árbol
d = Profundidad del árbol de juego

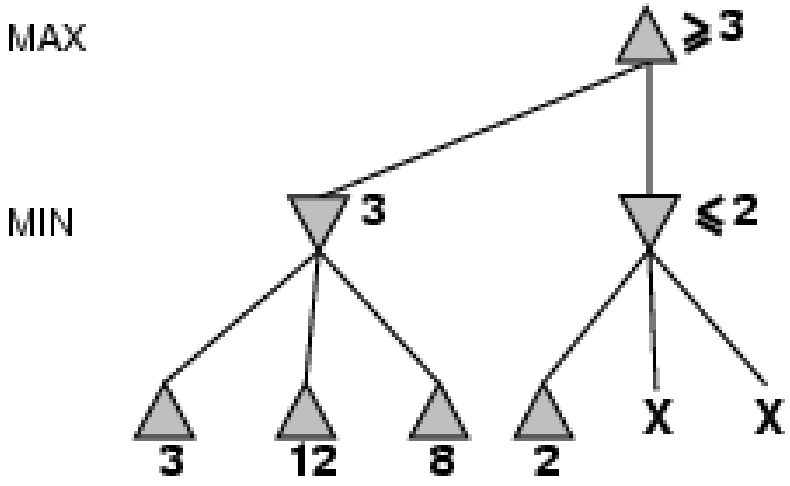
- Tiempo: **O(b^d)**.
- Espacio: **O(bd)**.

Ejemplo
En el ajedrez, b≈35 y d≈100, por lo que **no** podemos explorar el árbol completo del juego.

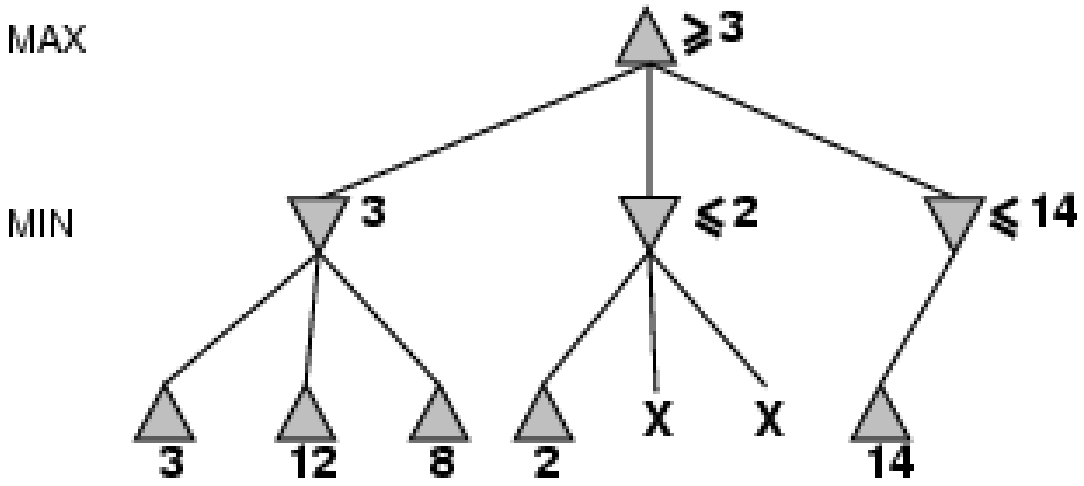
Poda α-β



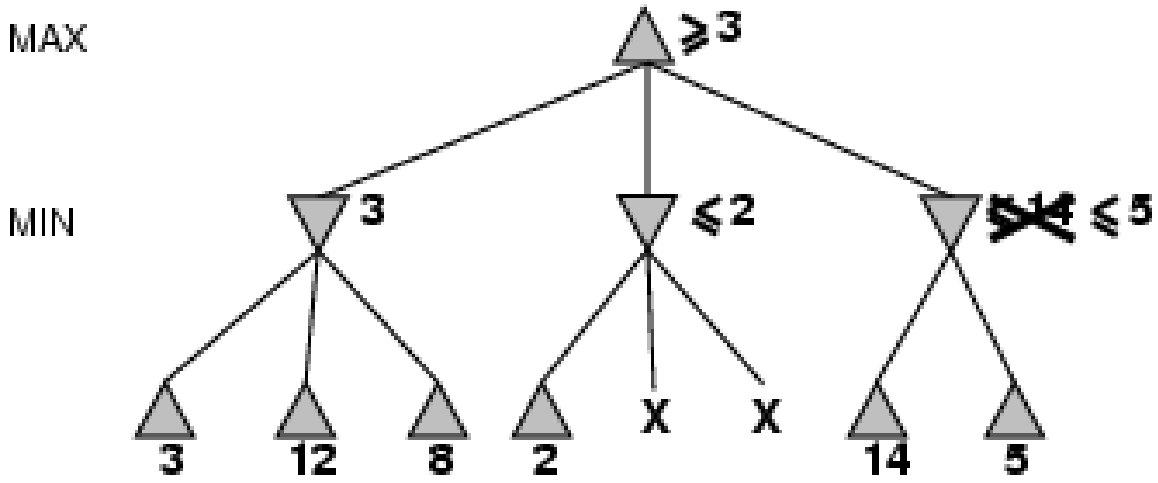
Poda α-β



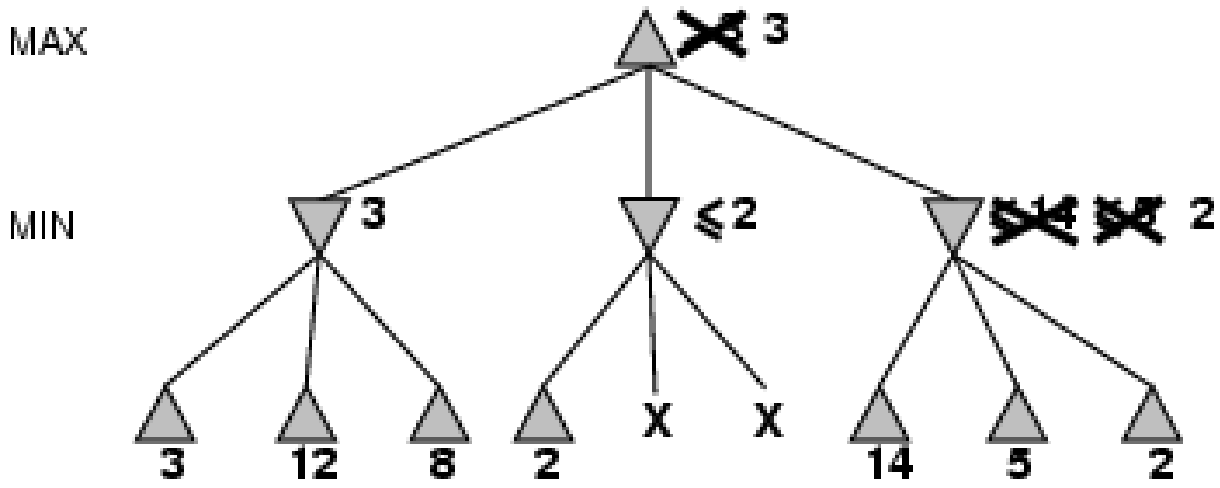
Poda α-β



Poda α-β



Poda α-β



Poda α-β

function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty, +\infty$)
  return the *action* in SUCCESSORS(*state*) with value v

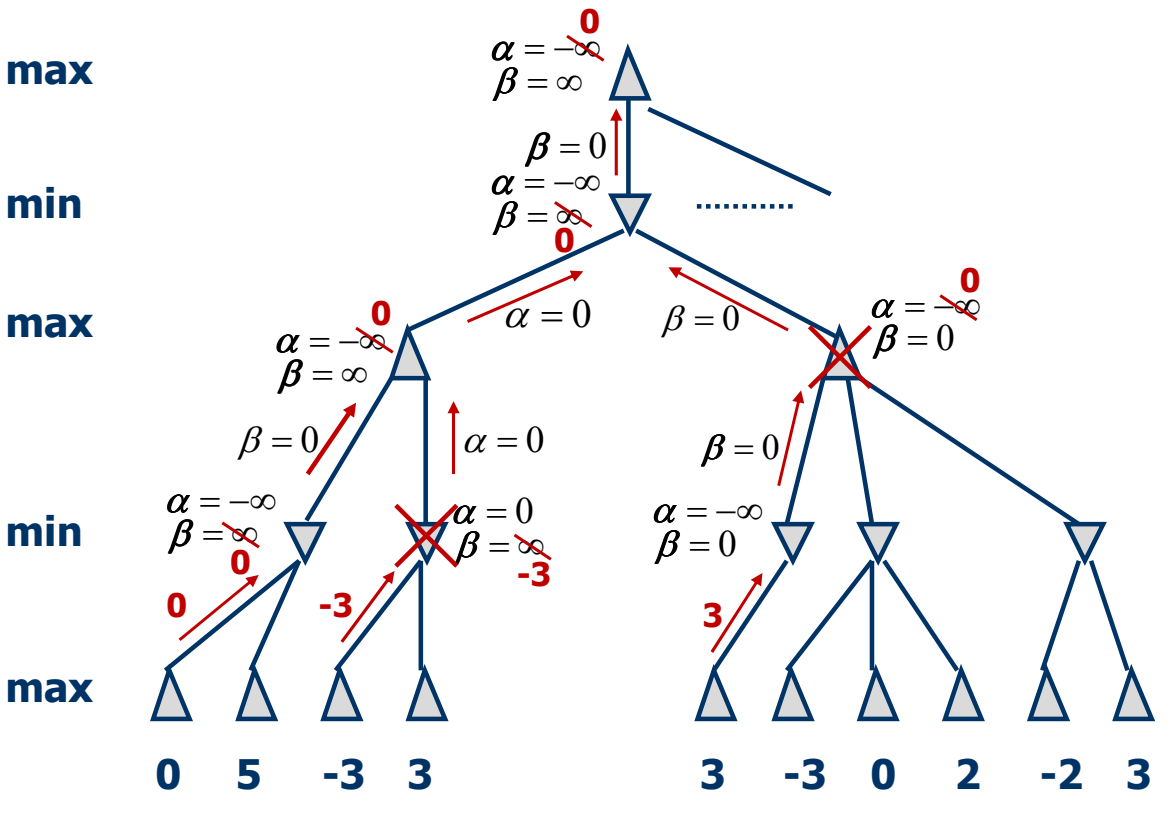
---

function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

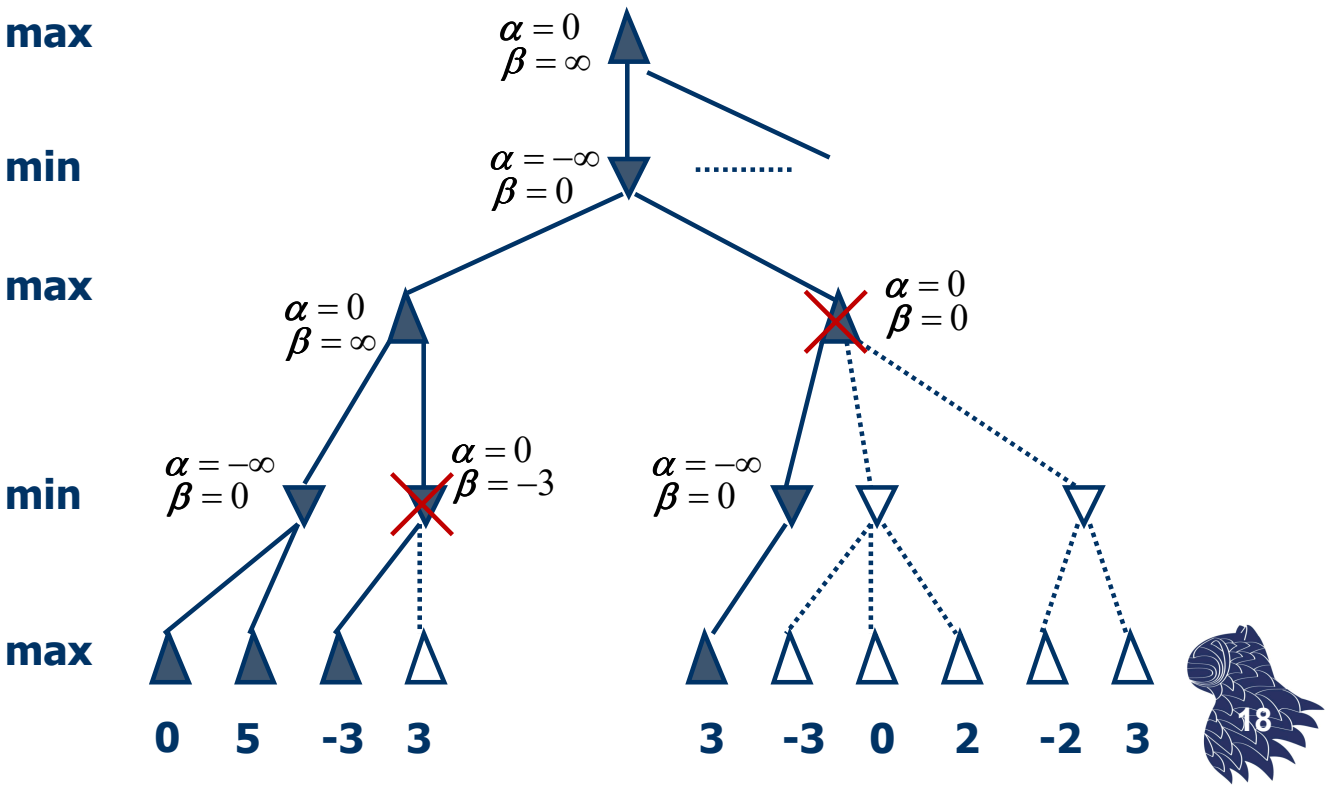
Poda α-β

```
function MIN-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(s, α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```


Poda α-β



Poda α-β



Poda α-β

- La poda α-β no afecta al resultado del juego.
- Cuanto mejor ordenemos los movimientos, más efectiva será la poda.
- Con una ordenación "perfecta", la complejidad del algoritmo es **$O(b^{d/2})$**.

  En otras palabras, con el mismo esfuerzo podremos explorar un árbol del doble de profundidad.

Poda α-β

¿Por qué se llama poda α-β?

- α es el valor de la mejor opción encontrada para el jugador MAX:

  MAX evitará cualquier movimiento que tenga un valor v peor que α (poda si $v<\alpha$).

- β es el valor de la mejor opción encontrada para MIN (mínimo encontrado hasta ahora):

  MIN evitará cualquier movimiento que tenga, para él, un valor v peor que β (poda si $v>\beta$)

En la práctica…

Si disponemos de 100 segundos por movimiento y podemos explorar 10^4 nodos por segundo,
sólo podremos analizar 10^6 nodos por movimiento:

Solución habitual:

- Cota de profundidad
- Función de evaluación (heurística): Solución aproximada

  p.ej.
  $Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$

En la práctica…

Aplicación: Ajedrez

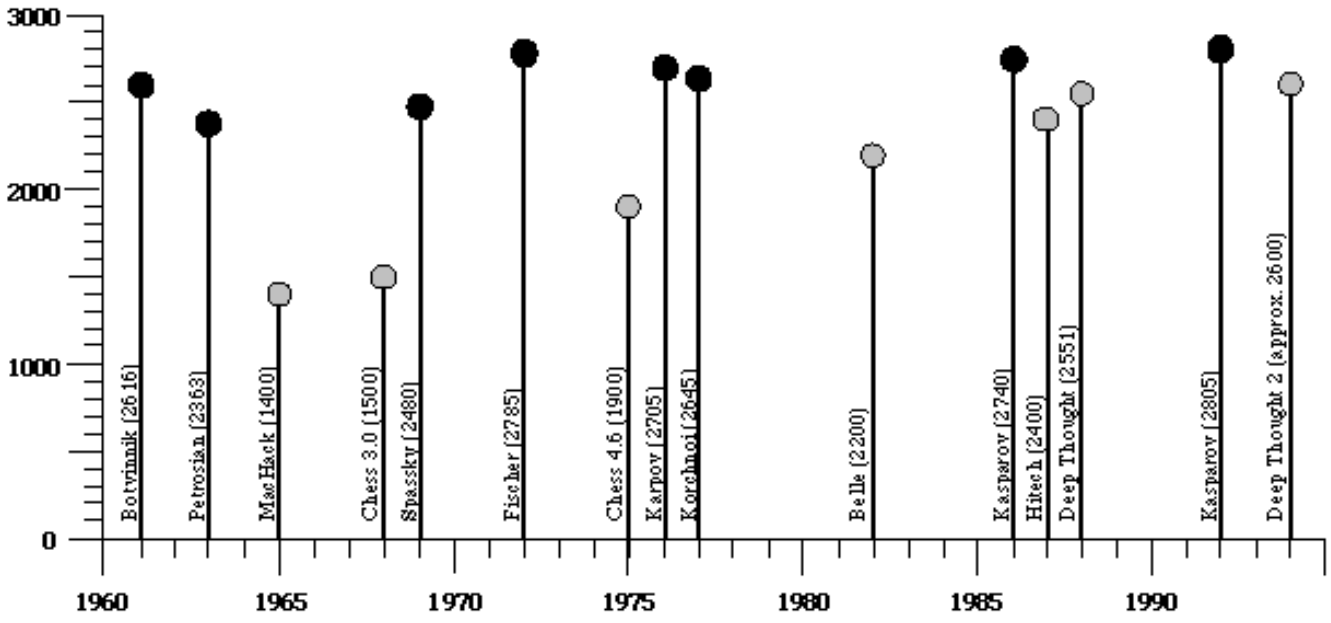
b=35

- 4-ply (novato)
  $d = 4 \rightarrow b^d = 1.5 \times 10^6$
- 8-ply (maestro): Programa típico para PC
  $d = 8 \rightarrow b^d = 2.25 \times 10^{12}$
- 12-ply (¿Kasparov?):
  $d = 12 \rightarrow b^d = 3.4 \times 10^{18}$

  En Deep Blue, el valor medio de b
  se reducía de 35 a 6 utilizando la poda α-β.

En la práctica…

Aplicación: Ajedrez



En la práctica…

Aplicación: Ajedrez

Deep Blue, IBM, 1997

https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)

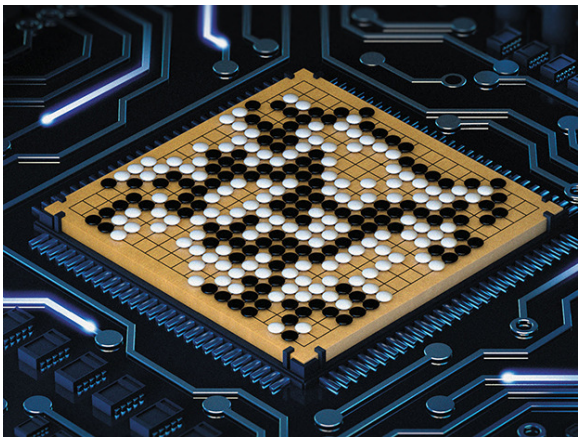
En la práctica…

- **Damas**: Chinook venció al campeón del mundo, Marion Tinsley, en 1994 usando una base de datos que definía el juego perfecto para todas los finales de partida con 8 o menos piezas (443748 millones de posiciones).
- **Ajedrez**: Deep Blue venció a Gary Kasparov en 1997 analizando 200 millones de posiciones por segundo y usando 8000 características y heurísticas que le permitían analizar algunas secuencias de hasta 40 ply.
- **Othello**: Los campeones humanos se niegan a jugar contra ordenadores porque son demasiado buenos.
- **Go**: Los campeones humanos se negaban a jugar contra ordenadores porque eran demasiado malos (b>300)…

En la práctica…

… hasta que llegó AlphaGo (2015):

Octubre 2015, Londres:
AlphaGo (Google DeepMind) vence al campeón europeo Fan Hui [2-dan], 5-0.



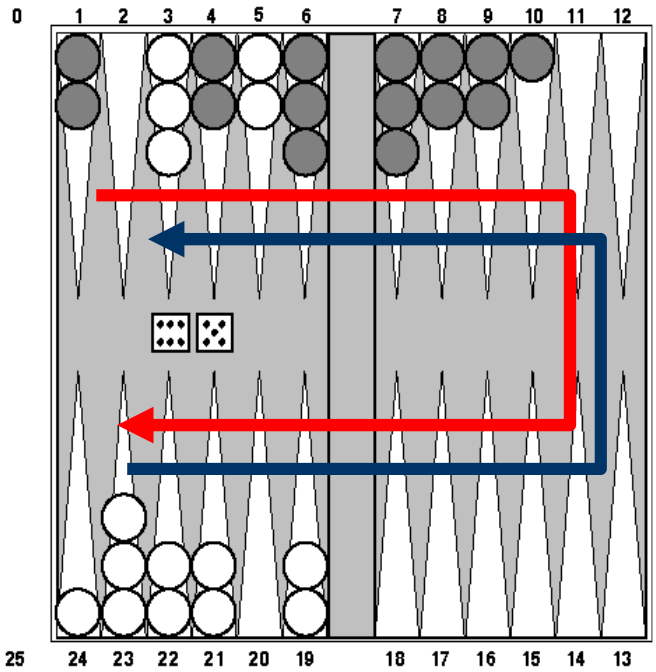
Marzo de 2016, Seúl: $1M
AlphaGo (Google DeepMind) vence a Lee Sedol [9-dan], 4-1.

AlphaGo

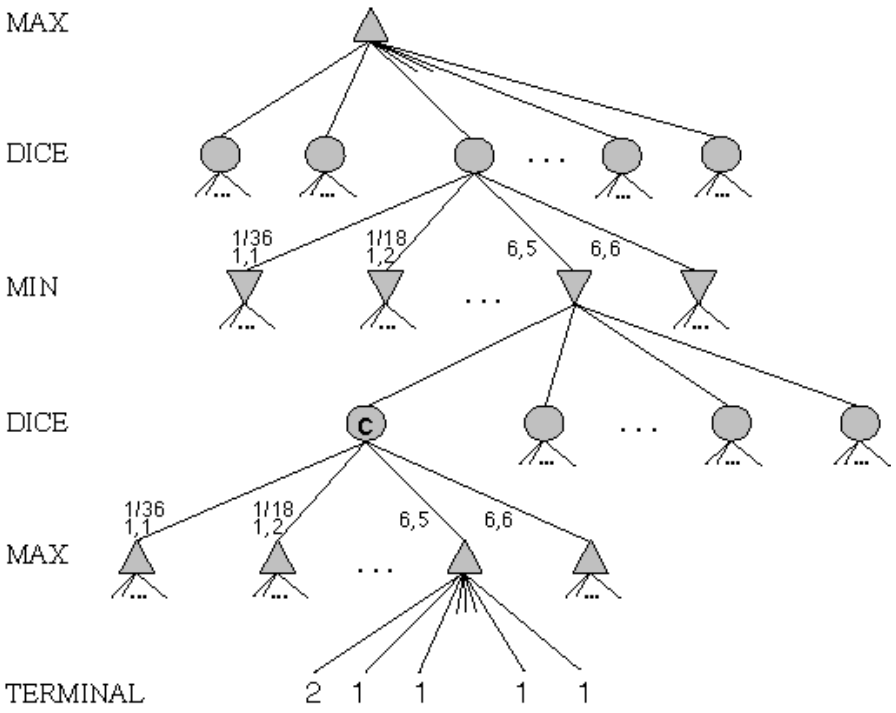
https://en.wikipedia.org/wiki/AlphaGo

Juegos de azar

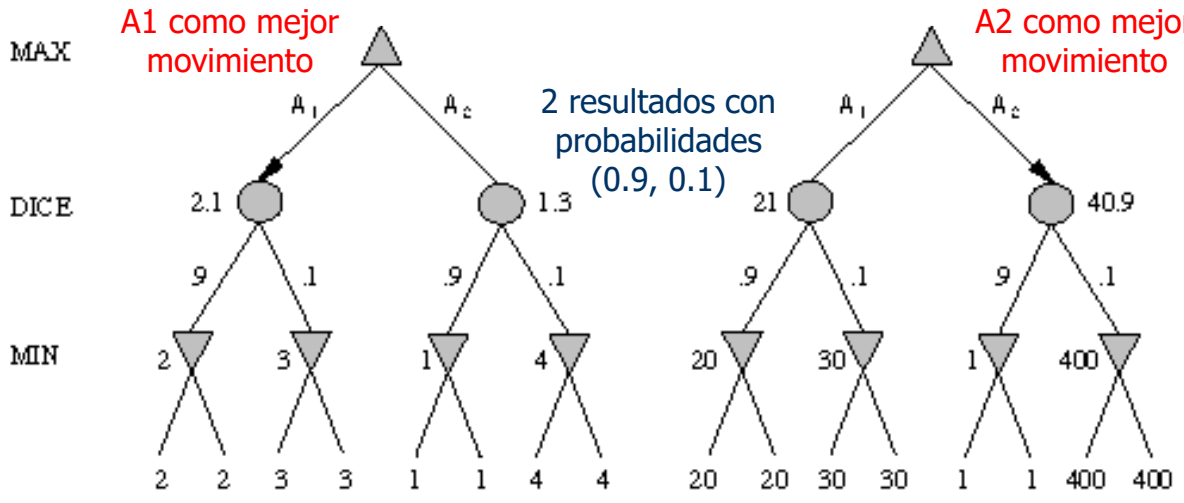
Hay juegos, como el Backgammon, en los que interviene el azar (en forma de dados):



Juegos de azar



Juegos de azar



Deep Learning

Resolución de juegos combinando aprendizaje automático [ML: Machine Learning] con búsqueda en árboles de juegos…

p.ej.

Búsqueda de Monte Carlo

[Monte Carlo tree/game search]

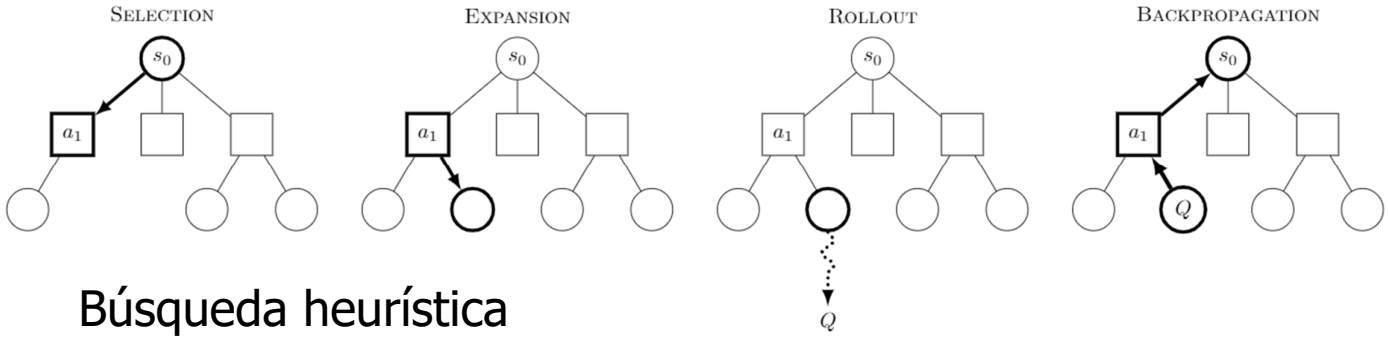
+

Aprendizaje por refuerzo [reinforcement learning]

Deep Learning

Búsqueda de Monte Carlo

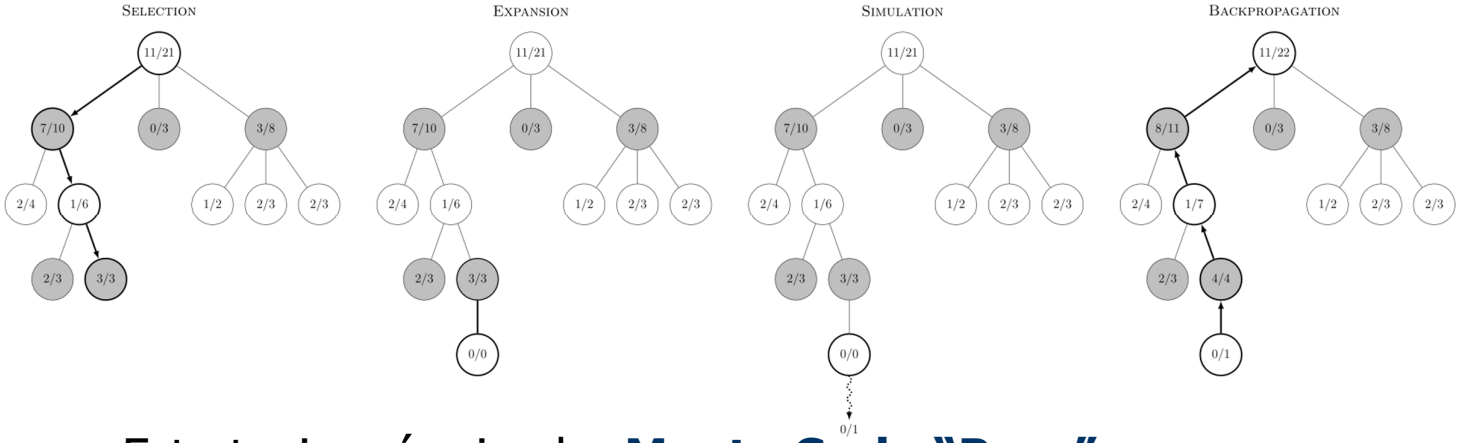
[MCTS: Monte Carlo Tree Search]



Búsqueda heurística

- Se analizan los movimientos más prometedores, expandiendo el árbol de búsqueda de forma aleatoria.
- Se repite el proceso en multitud de ocasiones [playouts o roll-outs], jugando en cada una de ellas hasta el final del juego (con movimientos aleatorios).

Búsqueda de Monte Carlo
[MCTS: Monte Carlo Tree Search]

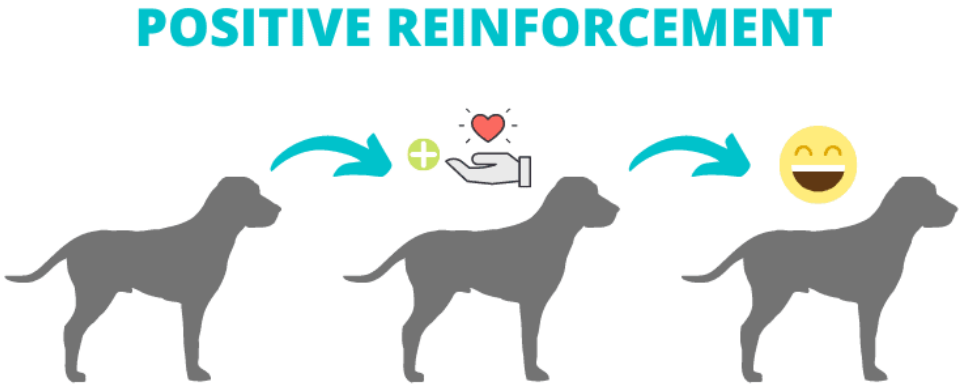


Estrategia más simple: **Monte Carlo "Puro"**

Se aplica el mismo número de "playouts" para cada movimiento válido del jugador y se escoge el movimiento que condujo a un número mayor de victorias.

Aprendizaje por refuerzo
[RL: Reinforcement Learning]

Método de aprendizaje basado en "prueba y error"...



Deep Learning

Aprendizaje por refuerzo

- Se aprende qué acción maximiza la recompensa que se obtiene mediante una estrategia de "prueba y error" (la recompensa no siempre es inmediata, por lo que hay que explorar secuencias de acciones).
- El aprendizaje nos permitirá ser capaces de evaluar la bondad de cada acción posible en un estado dado y determinar nuestra política de actuación.
- Al final, es como si tuviésemos una tabla que nos indica la acción preferible en cada situación.
- Como el número de situaciones (y acciones posibles) puede ser enorme, esa "tabla" podemos aproximarla con una red neuronal artificial [**Deep RL**]…

Deep Learning

Videojuegos (Atari 2600)

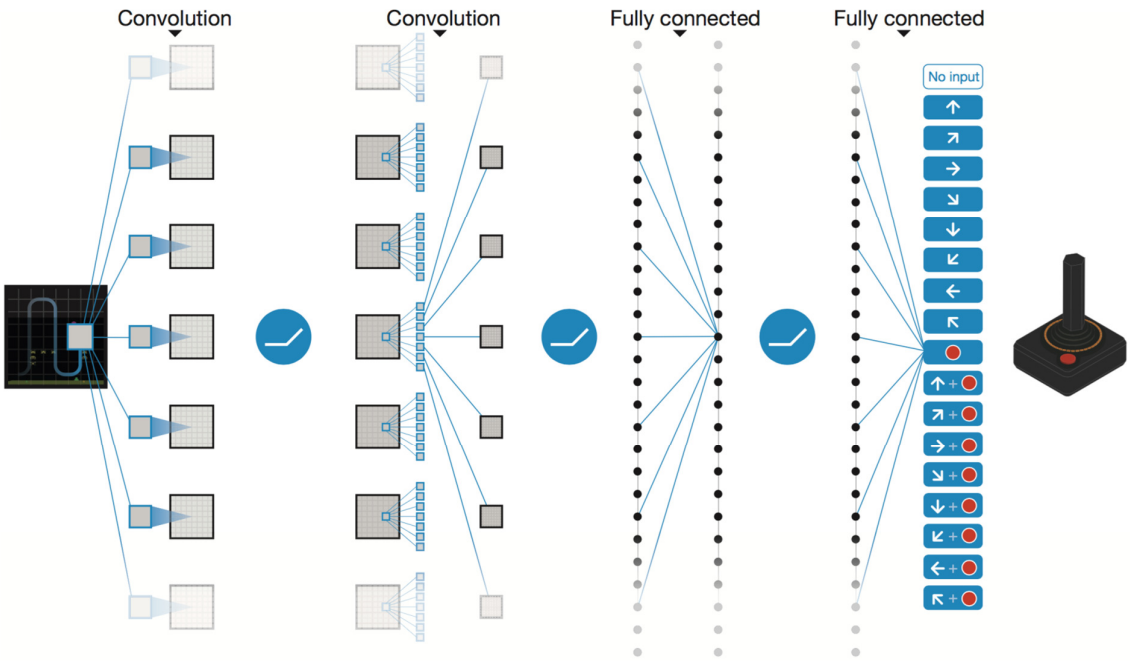
Google DeepMind

"Google AI beats humans at more classic arcade games than ever before"
http://arxiv.org/pdf/1509.06461v1.pdf (September 2015)

Deep Learning

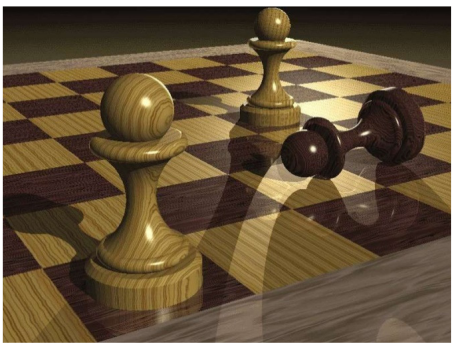
Videojuegos

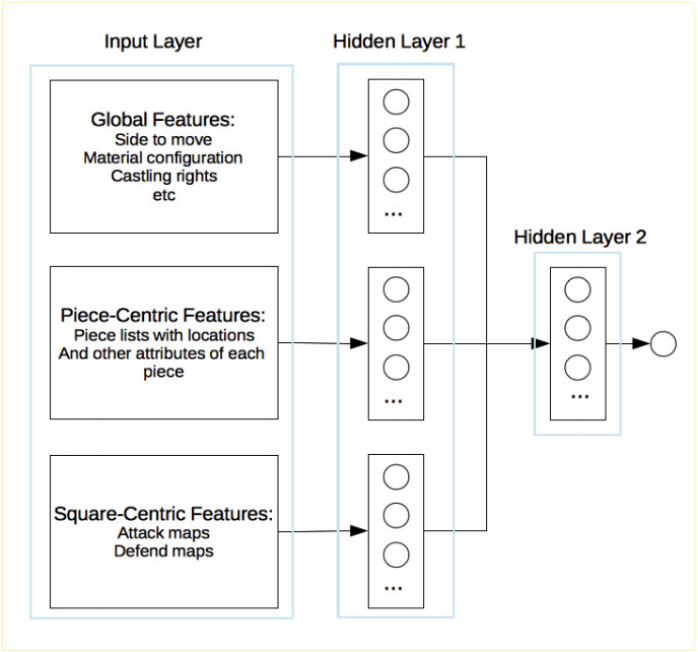
Deep Q Learning (Nature, 2015)



Deep Learning

Ajedrez





Matthew Lai (Imperial College London):

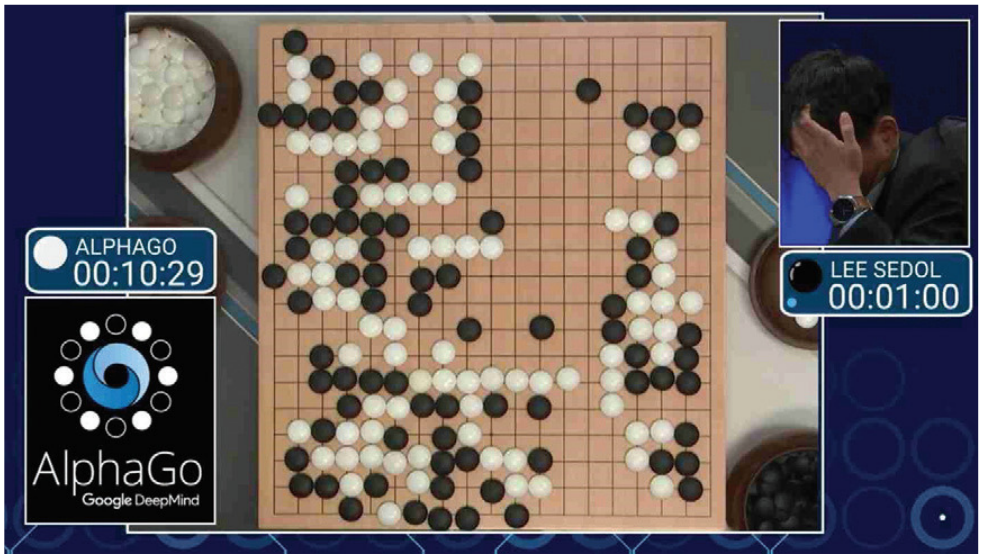
"Giraffe: Using Deep Reinforcement Learning to Play Chess"

http://arxiv.org/abs/1509.01549 (MSc Thesis, September 2015)

Deep Learning

Go

AlphaGo



https://deepmind.com/research/alphago/

Deep Learning

Go

AlphaGo Zero

https://elpais.com/elpais/2018/12/05/ciencia/1544007034_265553.html

Deep Learning

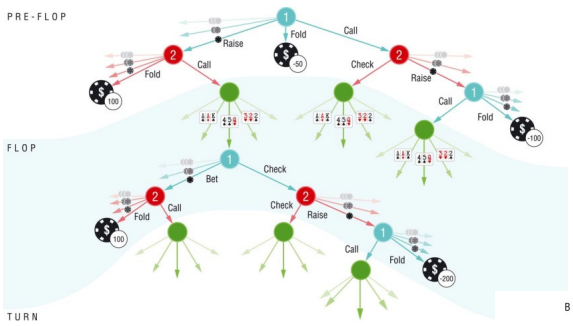
Poker

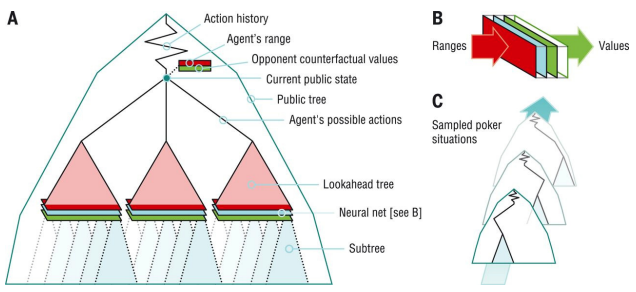
DeepStack

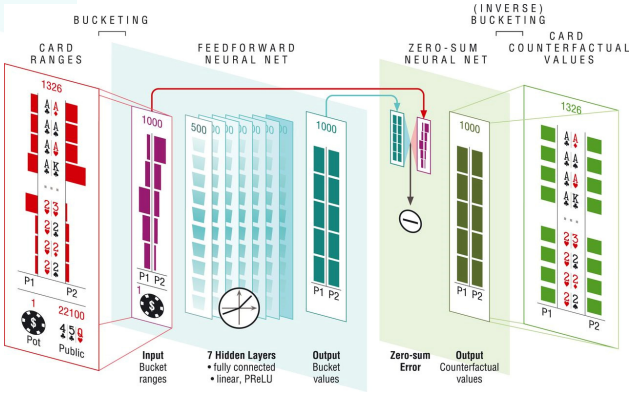
https://www.deepstack.ai/

Deep Learning

Poker







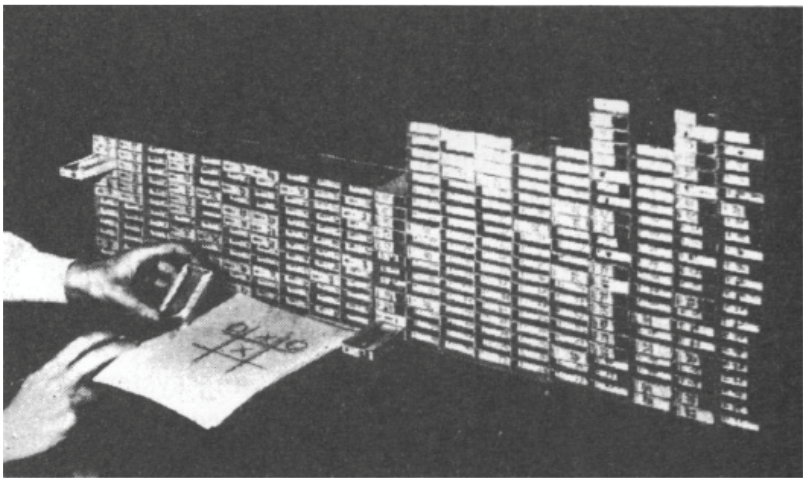
DeepStack: Expert-level artificial intelligence in heads-up no-limit poker
Science, Vol. 356, Issue 6337, pp. 508-513, 5 May 2017
DOI: 10.1126/science.aam6960

Deep Learning

Juegos...

Really???

The Matchbox Machine



1961

MENACE
Matchbox Educable Noughts And Crosses Engine

Donald Michie:
"Experiments on the mechanization of game-learning
Part I. Characterization of the model and its parameters"
The Computer Journal, 6(3):232–236, November 1963,
The British Computer Society, https://doi.org/10.1093/comjnl/6.3.232

Bibliografía

- Stuart Russell & Peter Norvig:
  Artificial Intelligence: A Modern Approach
  [2nd edition] Chapter 6: **Adversarial Search**.
  Prentice Hall, 2002. ISBN 0137903952.
  [4th edition] Chapter 5: **Adversarial Search and Games**.
  Pearson, 2020. ISBN 0134610997
  http://aima.cs.berkeley.edu/