

COMP0271: Inteligência Artificial

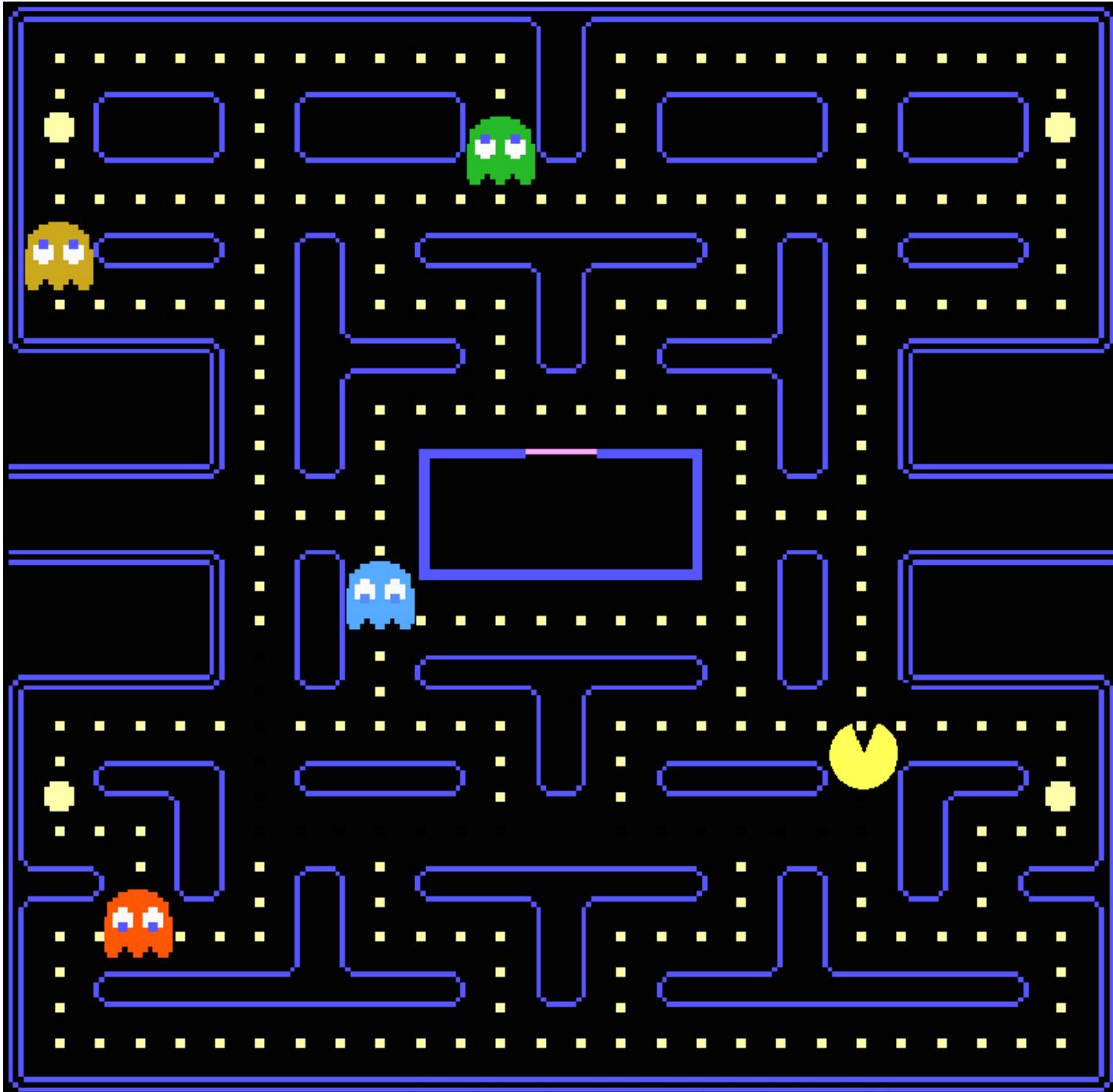
Busca em ambiente com adversários



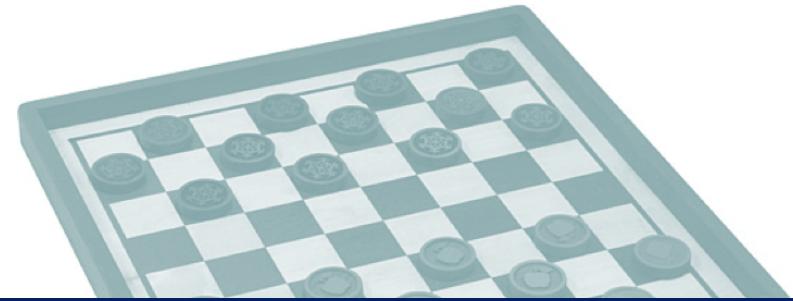
Professor: Hendrik Macedo

Universidade Federal de Sergipe, Brasil

E quando o espaço de busca envolve elementos competitivos com objetivos conflitantes?

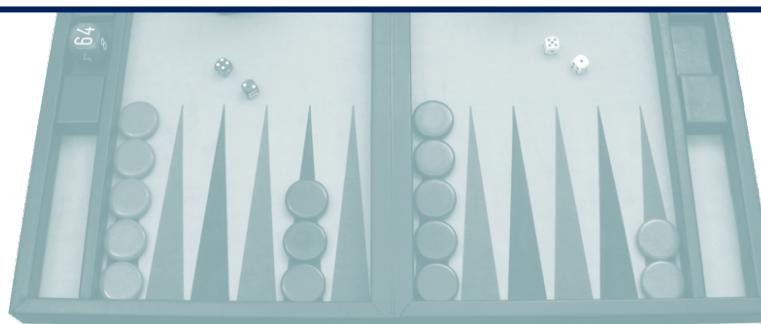




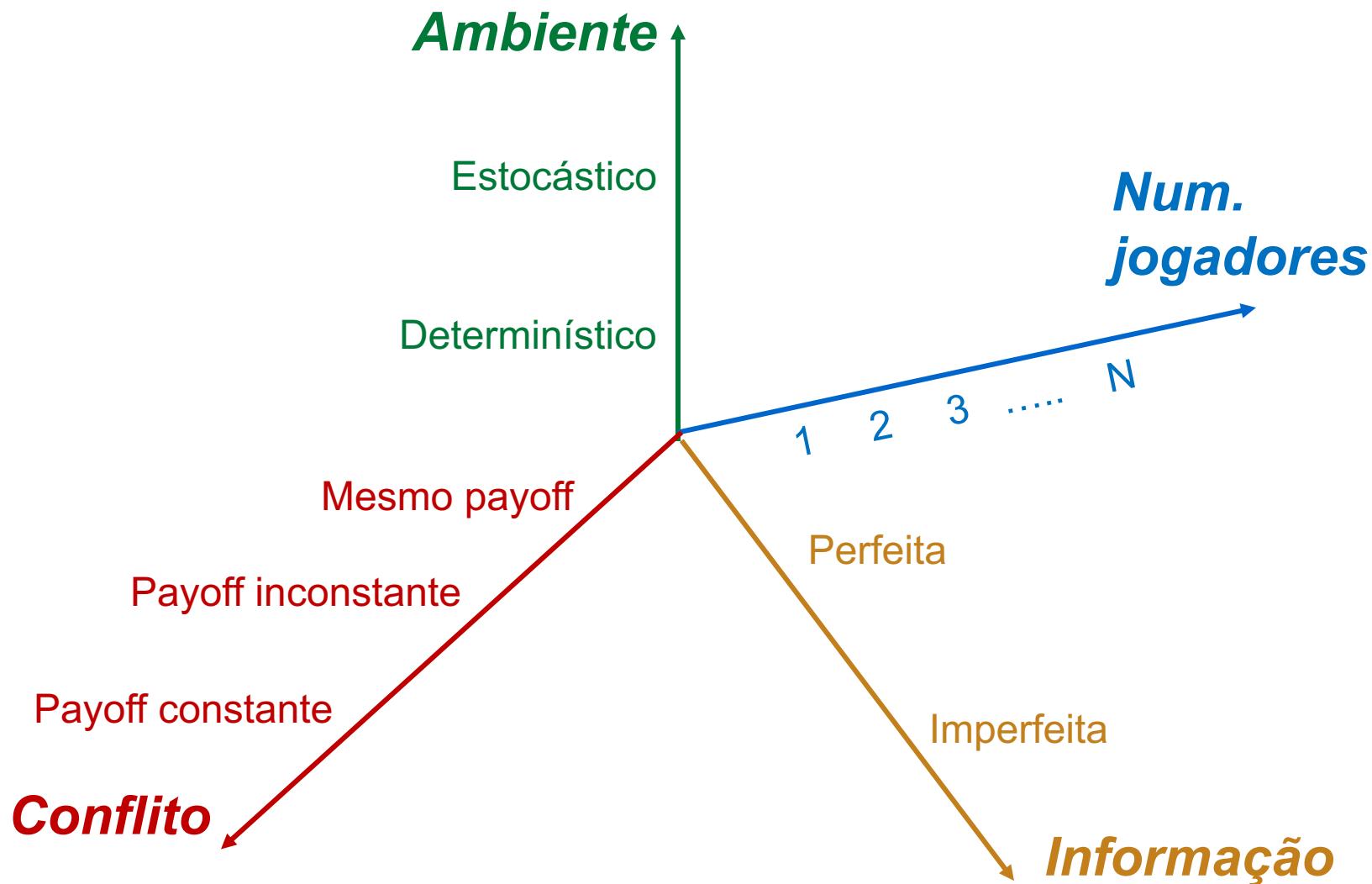


Em todos esses casos, o que é uma **Solução**?

Estratégia (Política) que recomenda um movimento para cada estado do ambiente



Categorização de jogos



Ambiente .: determinístico

- Formalização
 - Estados: S (inicial: s_0)
 - Jogadores: $P=\{1\dots N\}$ (usualmente em turnos)
 - Ações: A (pode depender do jogador / estado)
 - Função de transição: $S \times A \rightarrow S$
 - Teste objetivo: $S \rightarrow \{t,f\}$
 - Utilidades: $S \times P \rightarrow R$
- Solução é uma **política**: $S \rightarrow A$

Conflito

Jogos de soma constante
(competição pura)

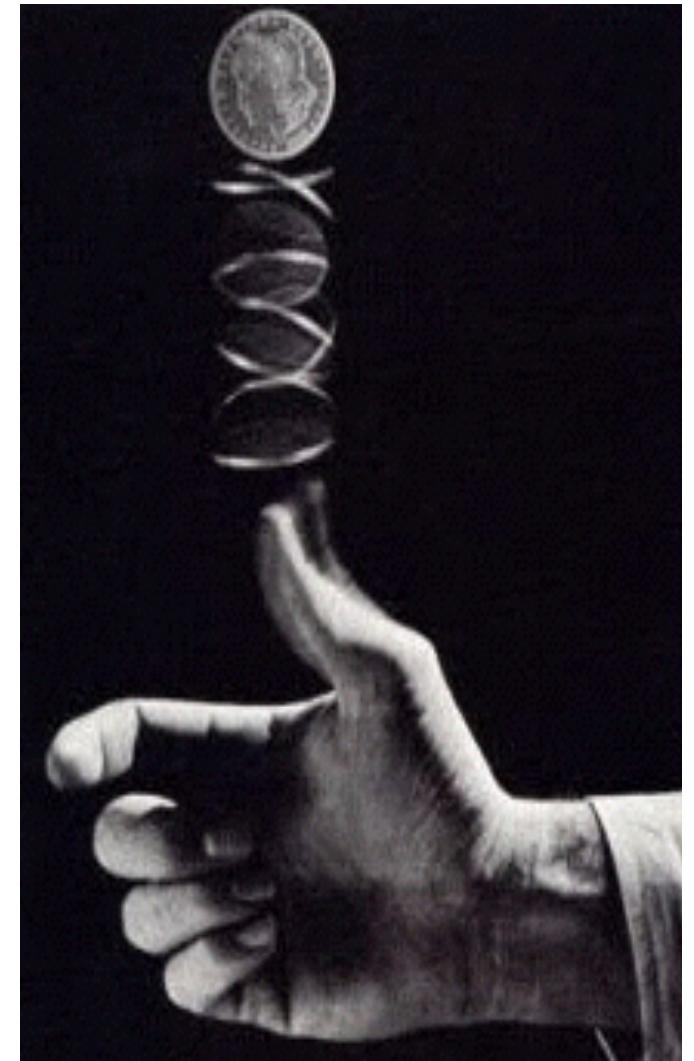
$c = 0 \rightarrow$ Jogos de Soma Zero

Exatos 2 jogadores
(interesses opostos)

Para todo $a \in A_1 \times A_2$

$$u_1(a) + u_2(a) = c$$

	cara	coroa
cara	1, -1	-1, 1
coroa	-1, 1	1, -1
	$u_1(\text{cara}, \text{cara}) = +1$	
	<u>$u_2(\text{cara}, \text{cara}) = -1$</u>	
	0 (zero)	...



Conflito

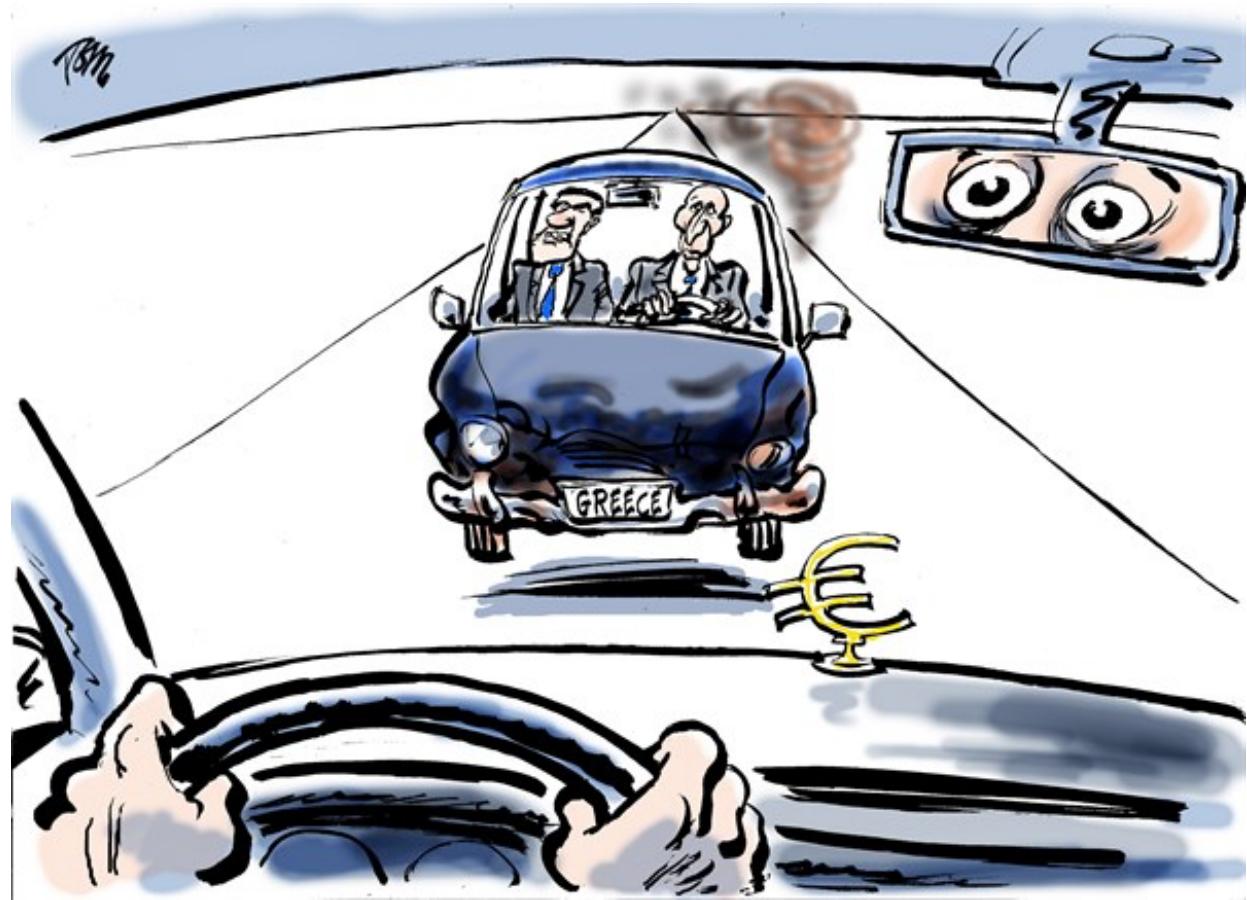
Jogos de payoff comum
(cooperação pura)

Não há conflito de interesses
Desafio é coordenação

Para toda $a \in A_1 \times \dots \times A_n$

$$u_i(a) = u_j(a)$$

		esquerda	direita
esquerda	1	0	$u_1(\text{esq},\text{esq}) = u_2(\text{esq},\text{esq}) = 1$
direita	0	1	...



Conflito

Jogos com elementos de competição e cooperação

Há conflito de interesses, mas é necessário saber cooperar

Não há restrições sobre $u_i(a)$

	romance	ação
romance	2,1	0,0
ação	0,0	1,2

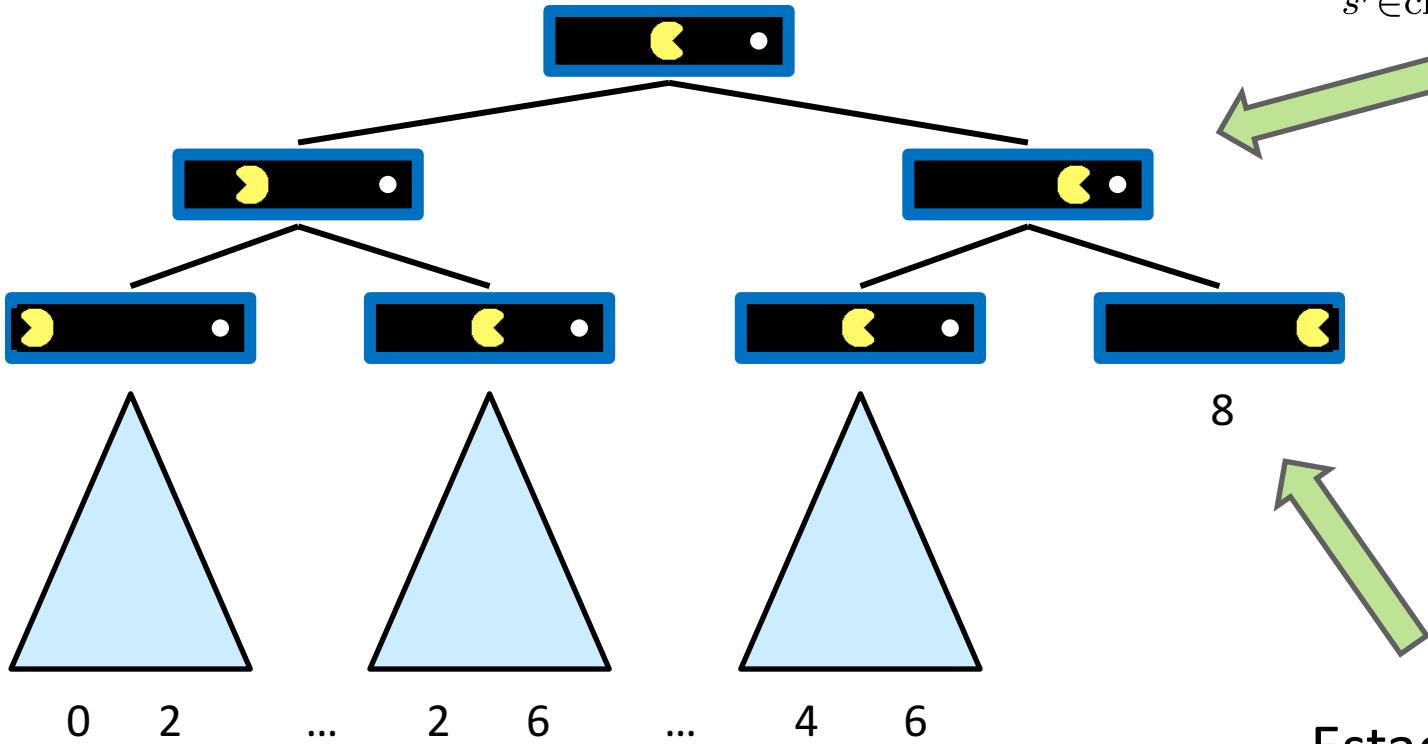
$$u_1(\text{romance}, \text{romance}) = 2$$
$$u_2(\text{romance}, \text{romance}) = 1$$

$$u_1(\text{romance}, \text{ação}) = 0$$
$$u_2(\text{romance}, \text{ação}) = 0$$



Número de jogadores = 1

O melhor resultado possível
(Utilidade) a partir daquele
estado!



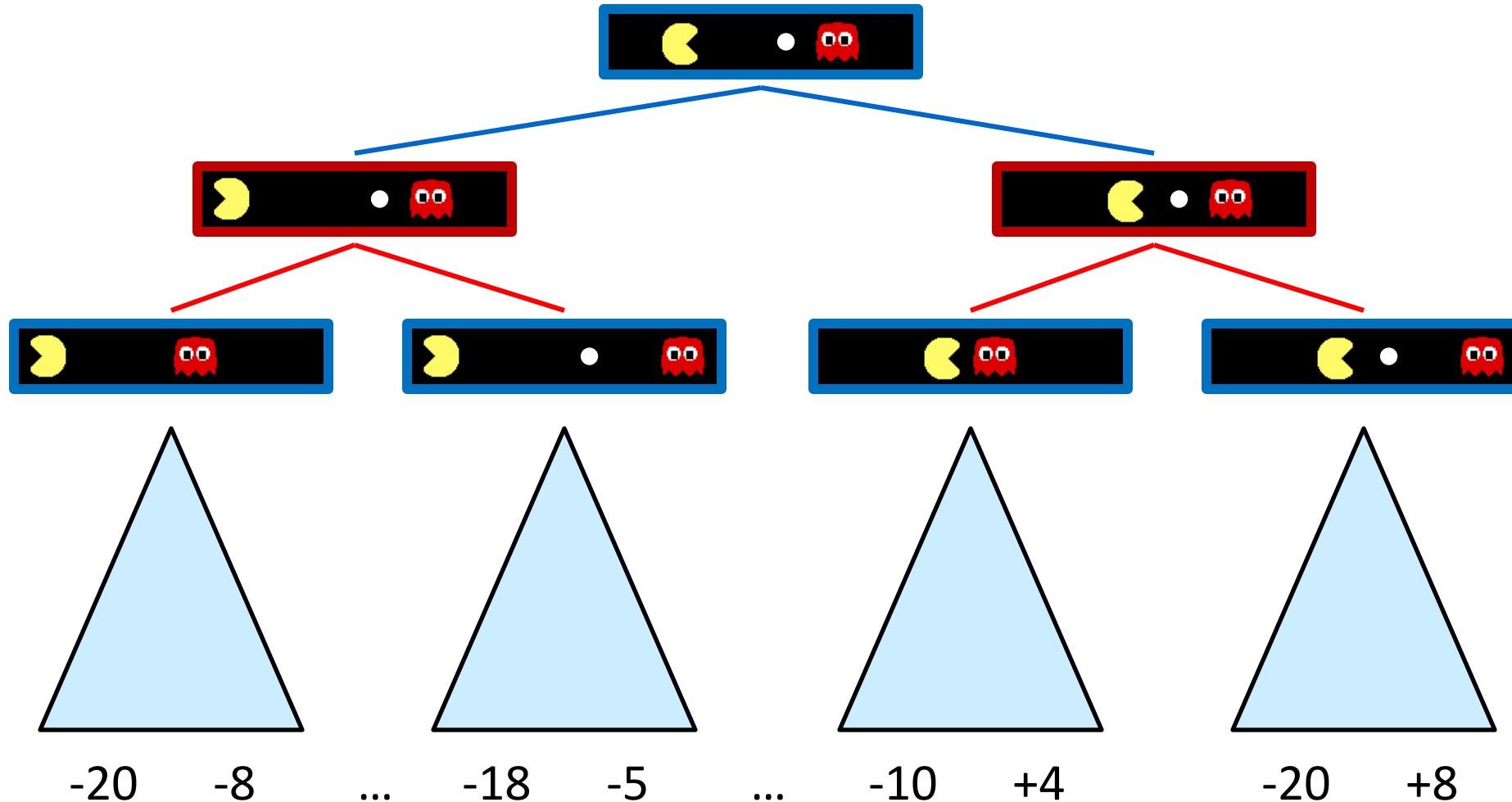
Estados não-terminais:

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

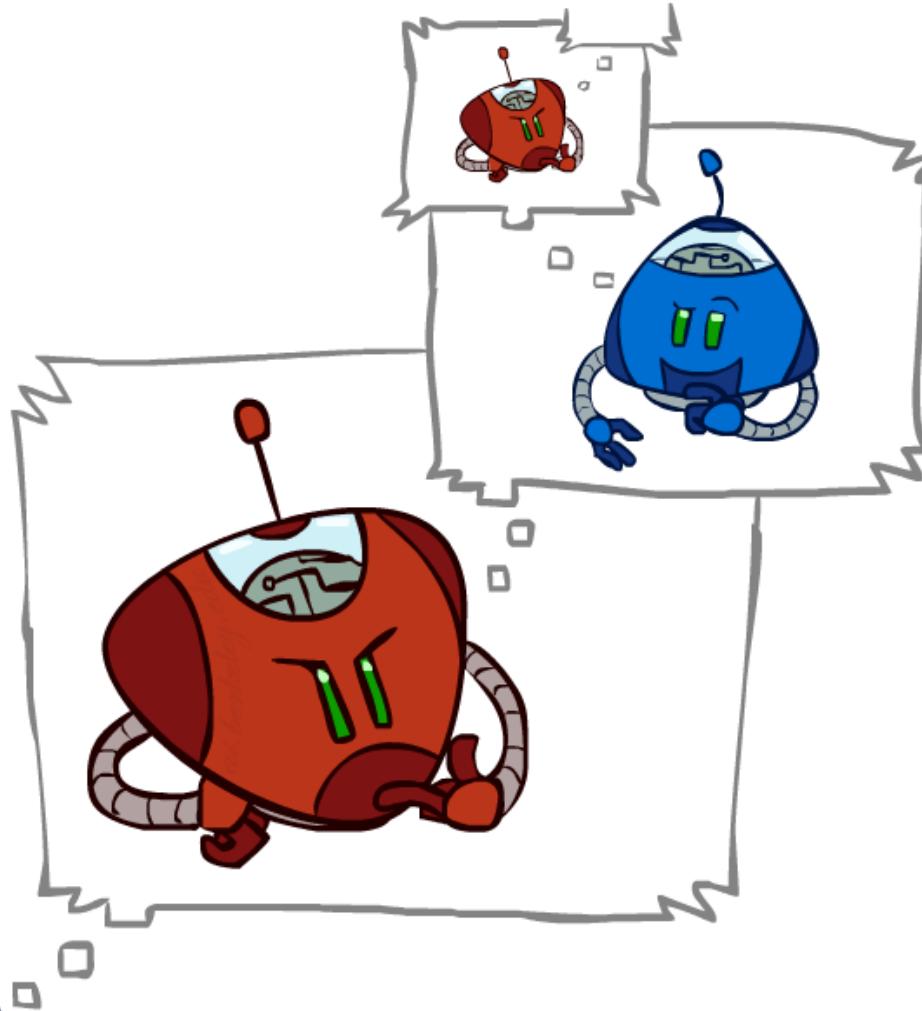
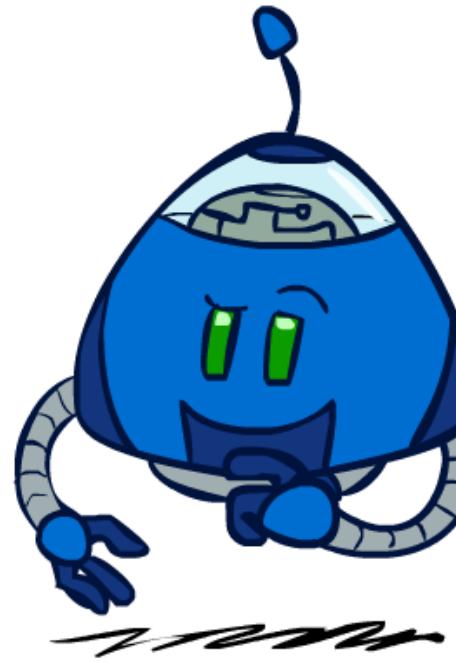
Estados terminais:

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

Número de jogadores > 1



Rationale!



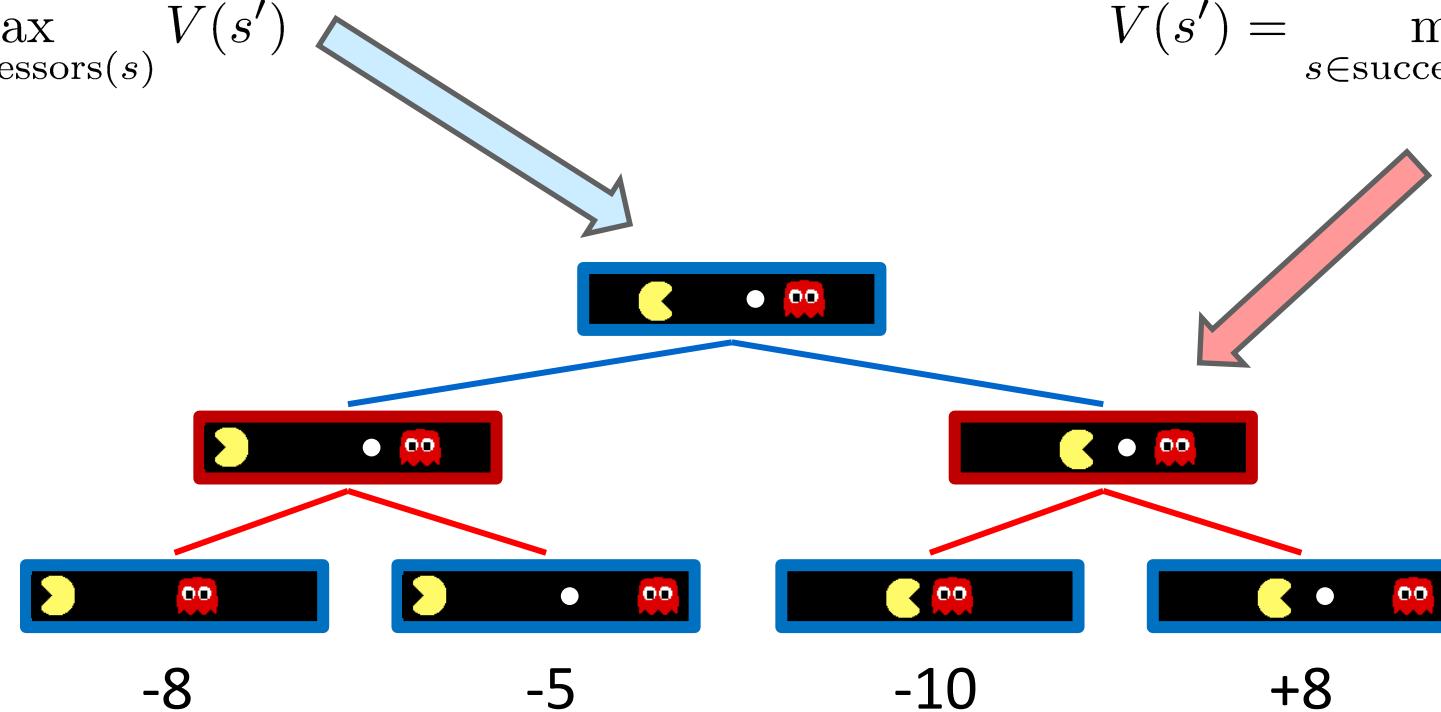
Árvore Minimax

Estados sobre controle do agente:

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

Estados sobre controle do oponente:

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



Estados terminais:

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

Teorema Minimax (von Neumann, 1928)

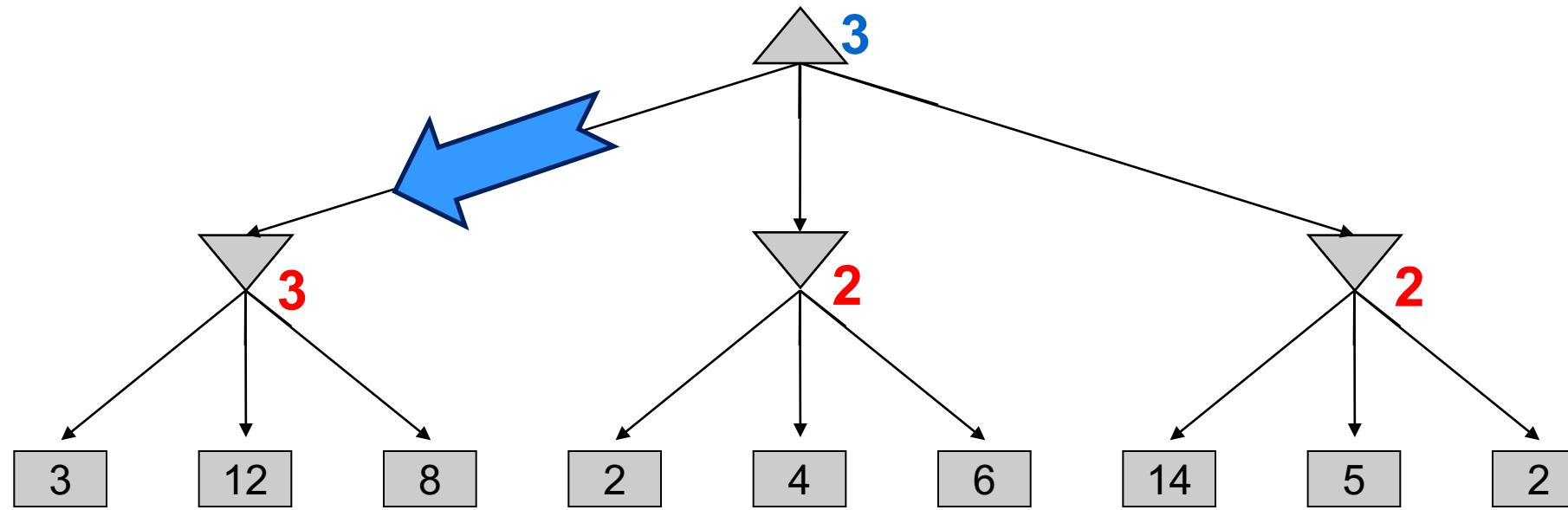
Em qualquer jogo finito de soma zero de 2 jogadores, [em qualquer *Equilíbrio Nash*], cada jogador recebe um payoff que é igual a ambos valores maxmin e minmax.

maximizar o ganho mínimo

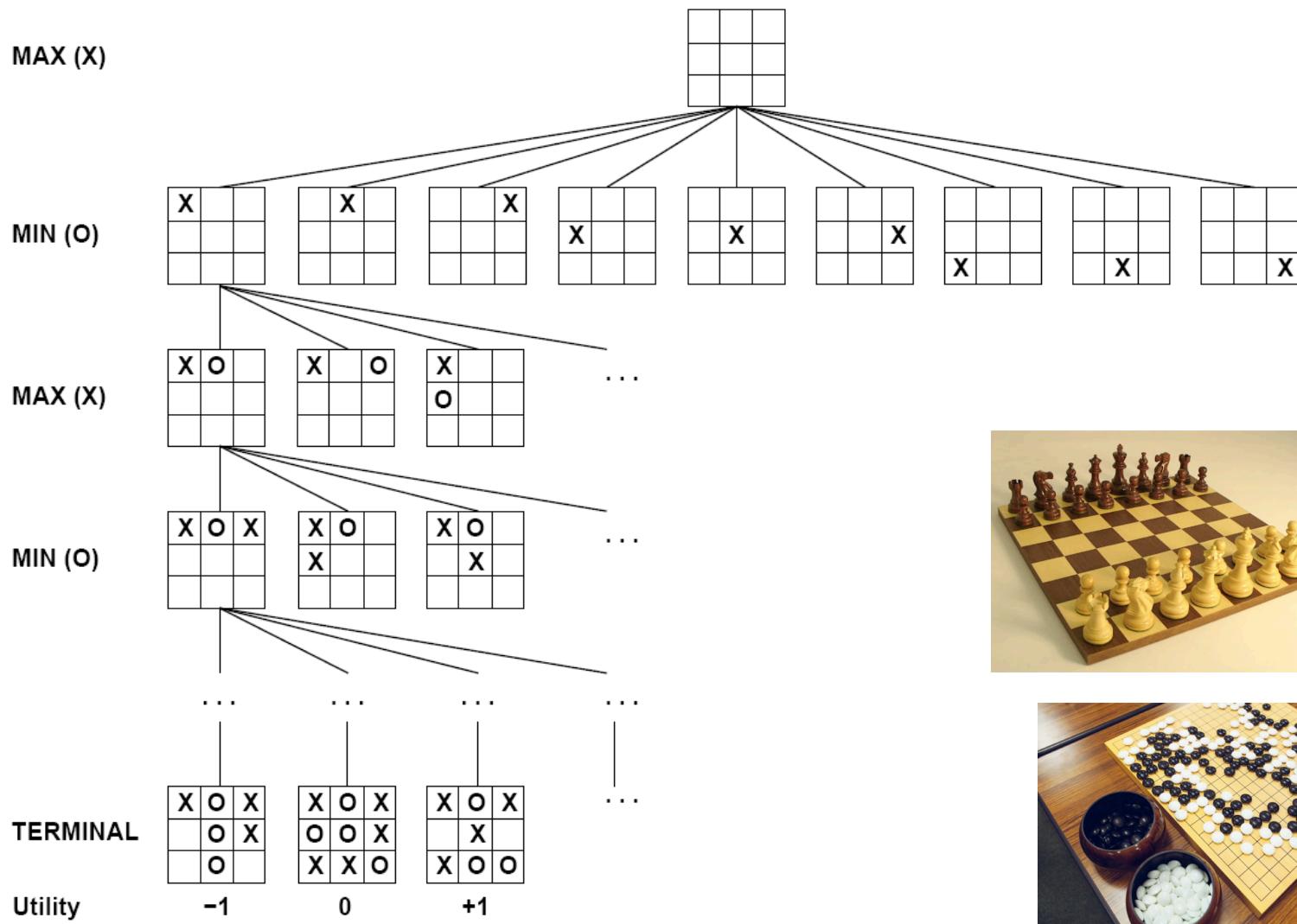
minimizar a perda máxima



Minimax: acompanhamento do raciocínio



Jogo da velha



Minimax: Implementação

```
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)

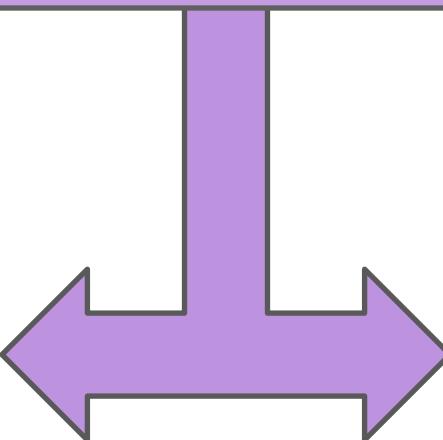
```
def max-value(state):
```

 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

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

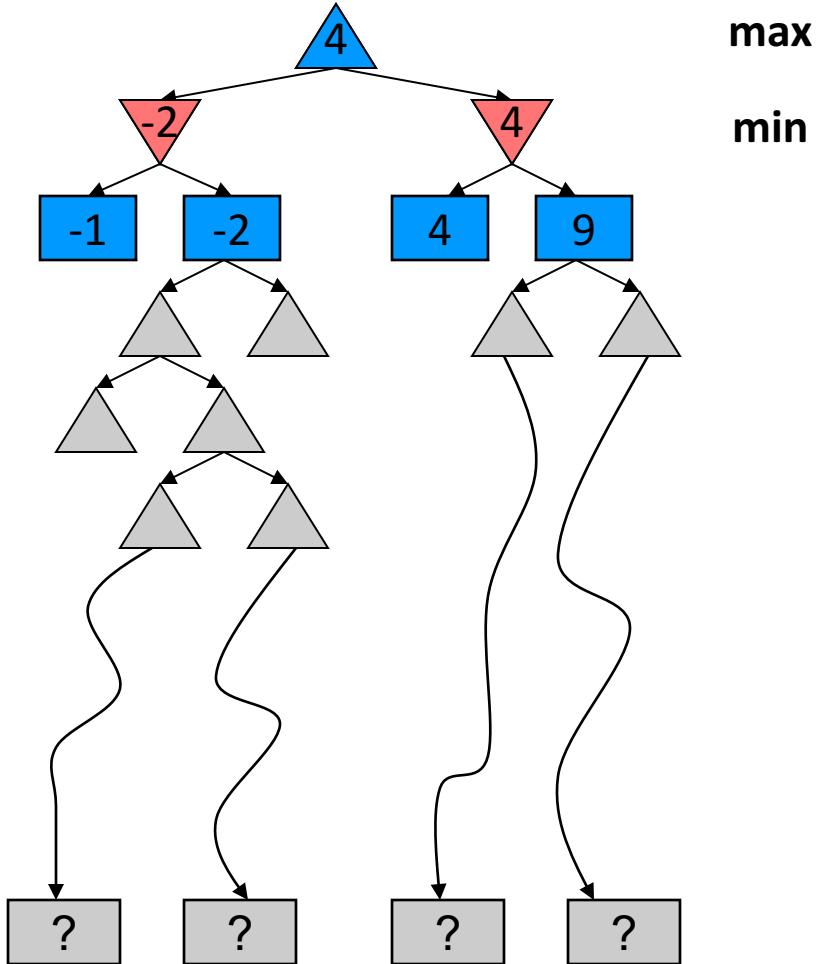
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

Minimax: Eficiência

- Comportamento equivalente ao DFS
 - Tempo: $O(b^m)$
 - Espaço: $O(bm)$

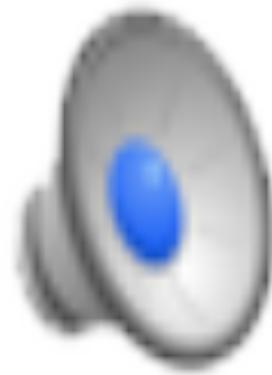
Problema

- Em jogos reais, não dá pra buscar até as folhas porque a árvore de jogo é muito profunda!!
- Solução: Iterative Deepening
- Substitui valor de Utilidade (de nós terminais) por **Função de Avaliação (Eval)** de nós não terminais

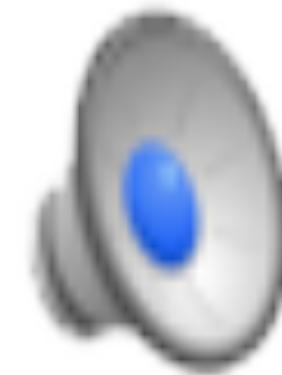


Profundidade importa bastante!

Função de Avaliação aplicada no nível 2



Função de Avaliação aplicada no nível 10

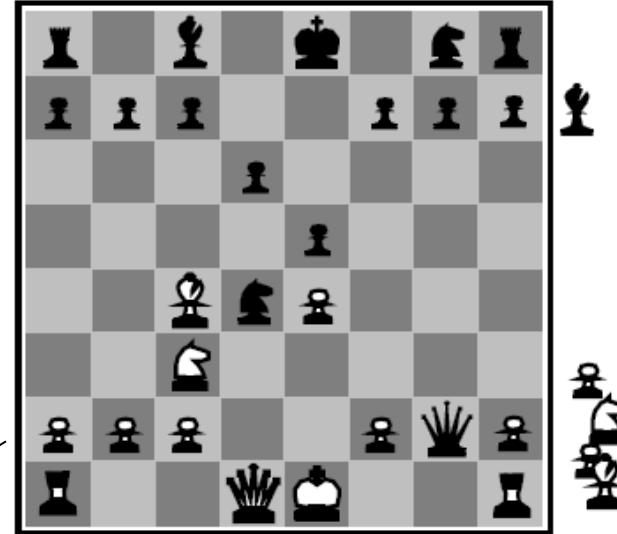
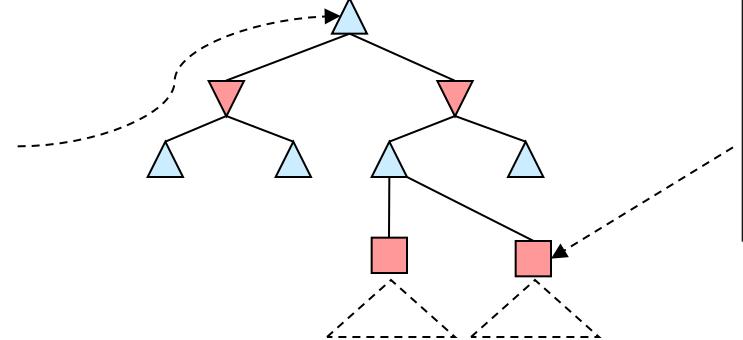


Função de Avaliação (Eval)



Black to move

White slightly better



White to move

Black winning

- Função ideal: retorna o valor minimax real da posição
- Na prática: soma ponderada linear de características

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

- Ex: $f_1(s) = (\text{num rainhas brancas} - \text{num rainhas pretas})$, etc.

Fantamas inteligentes com Minimax



Coordenação tática parece emergir
da aplicação do algoritmo Minimax

Mais problemas

- Imagine um programa para jogar xadrez...
 - Tempo limite = 100 seg/jogada
 - Programa jogador = 10000 nós(seg)
 - Programa jogador = 10^6 nós/jogada
 - $b^m = 10^6$
- ...só que no xadrez: $b \approx 35$
 - $35^m = 10^6 \rightarrow m = 4$ (profundidade)
 - Estudos de xadrez mostram que:
 - ver 4 jogadas à frente \approx iniciante
 - ver 8 jogadas \approx bom jogador
 - ver 12 jogadas \approx Kasparov



- ...e no GO: $b \geq 300$



Solução

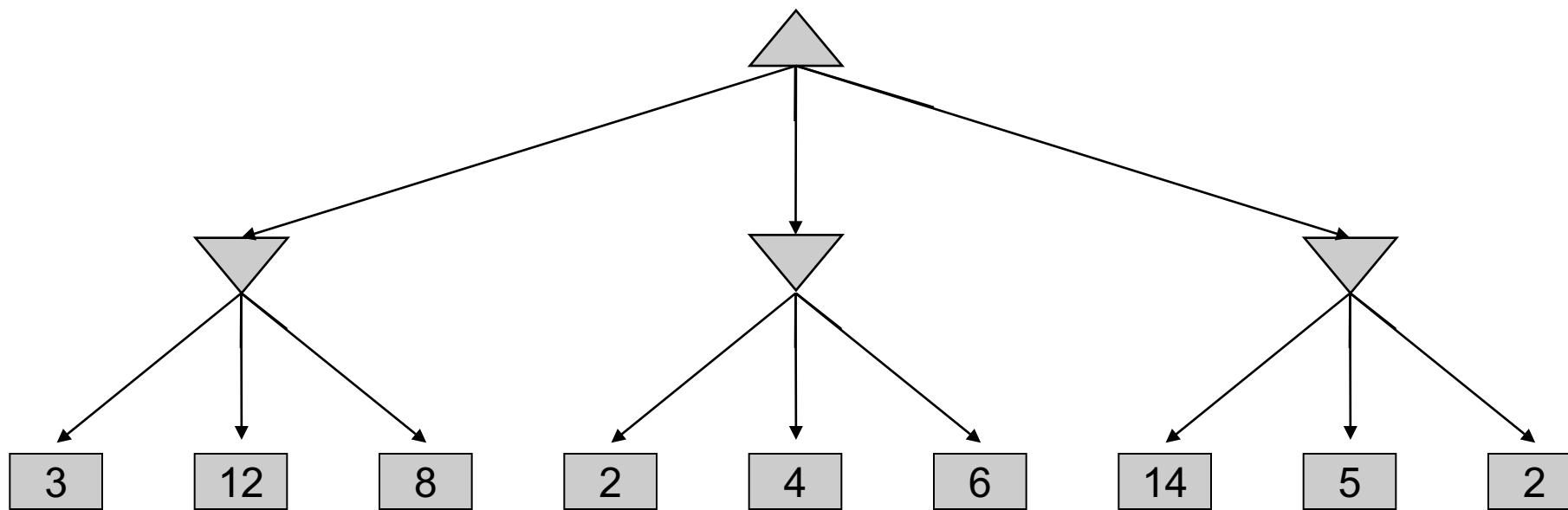
Poda inteligente (e responsável) da árvore de jogo!



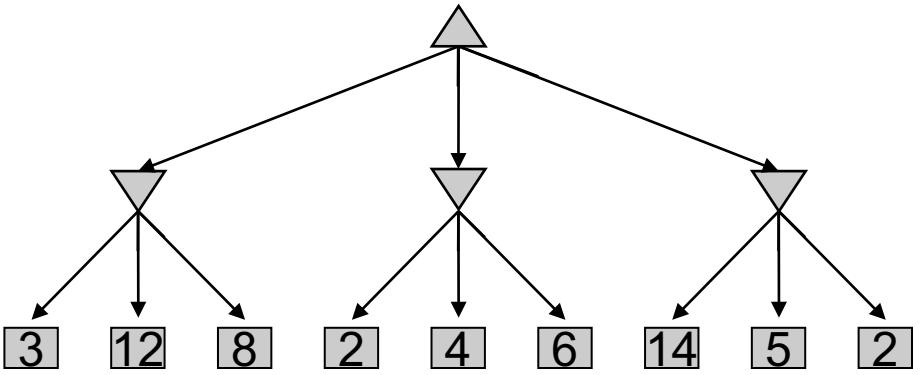
Minimax com Poda α - β

α : melhor opção de MAX no caminho corrente até a raiz
 β : melhor opção de MIN no caminho corrente até a raiz

Poda α - β



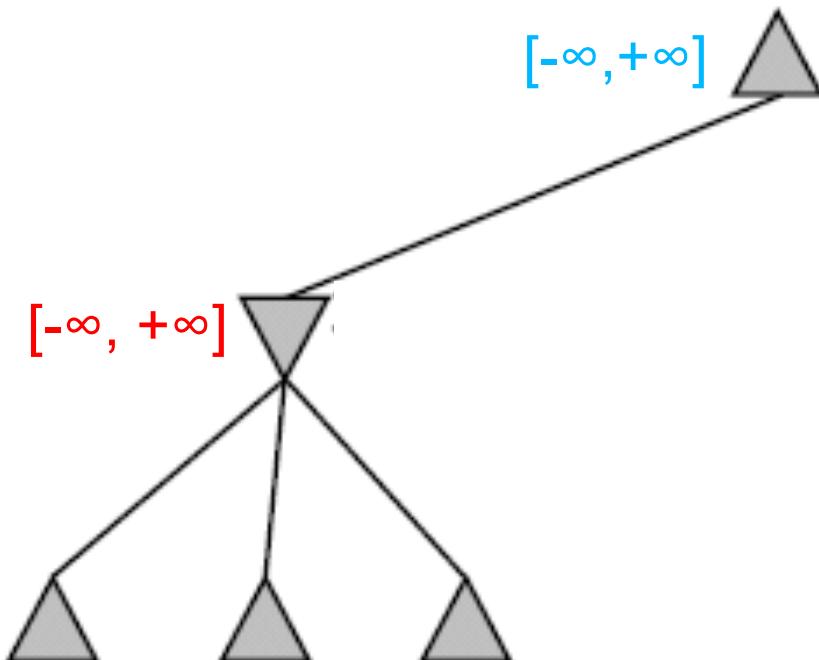
Poda α - β



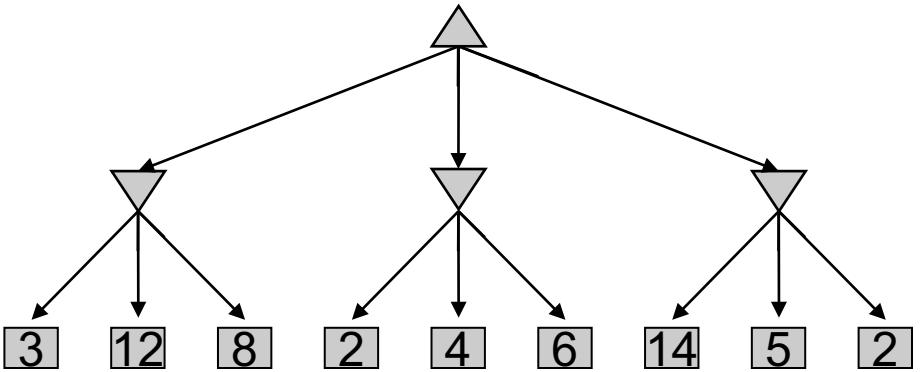
MAX

MIN

$[-\infty, +\infty]$



Poda α - β

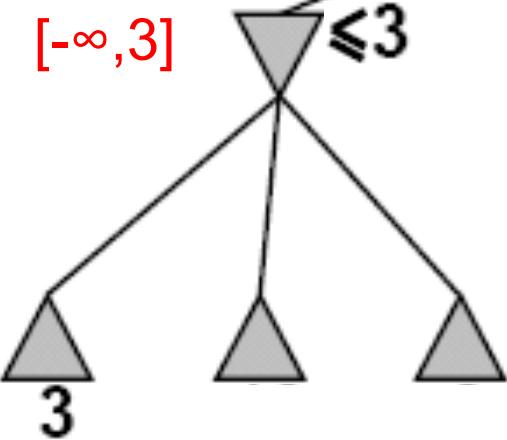


MAX

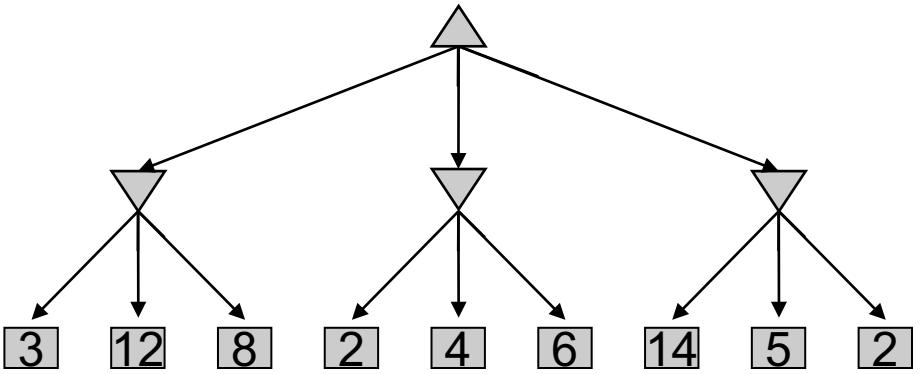
MIN

$[3, +\infty]$

≥ 3



Poda α - β

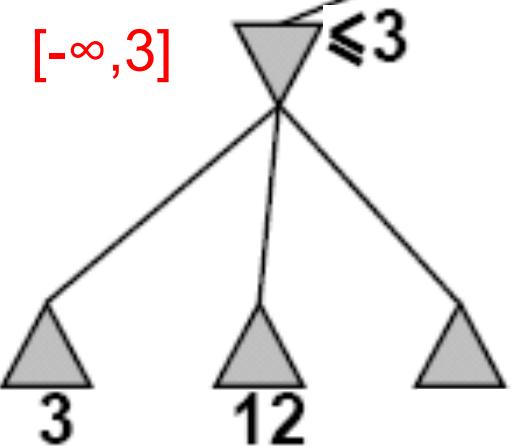


MAX

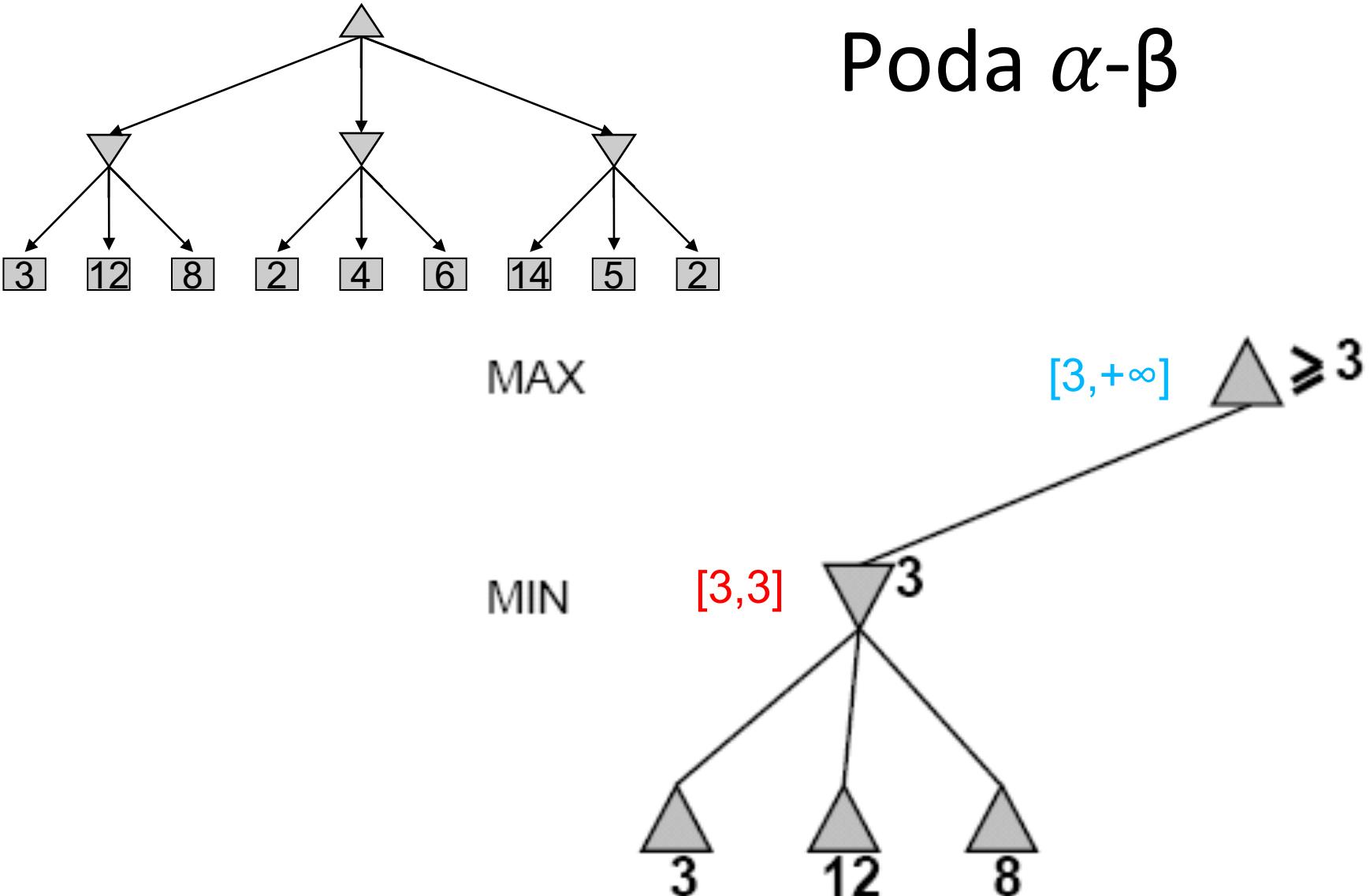
MIN

$[3, +\infty]$

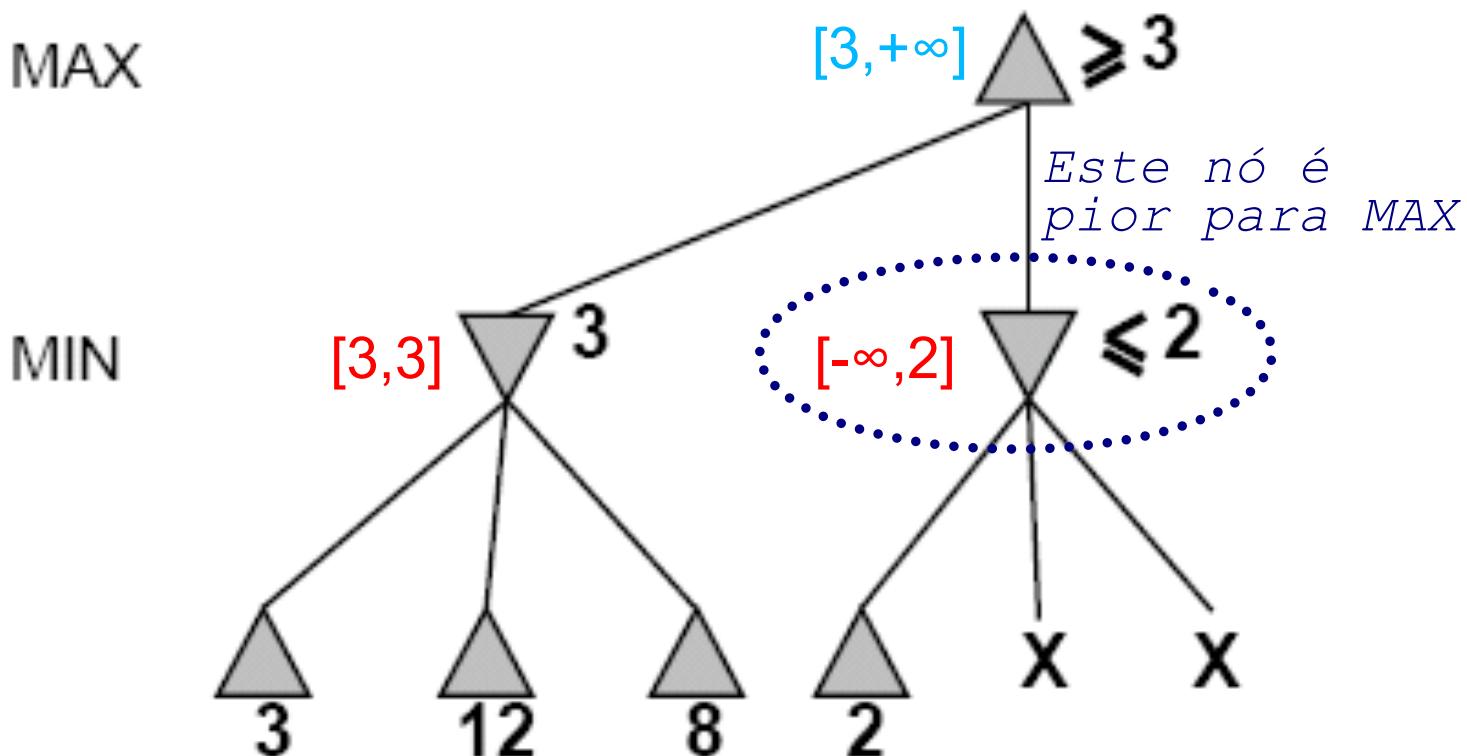
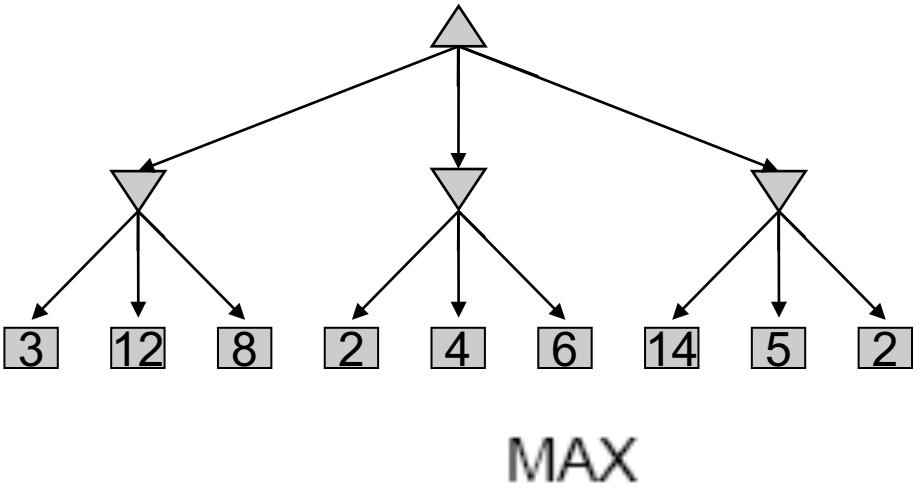
≥ 3



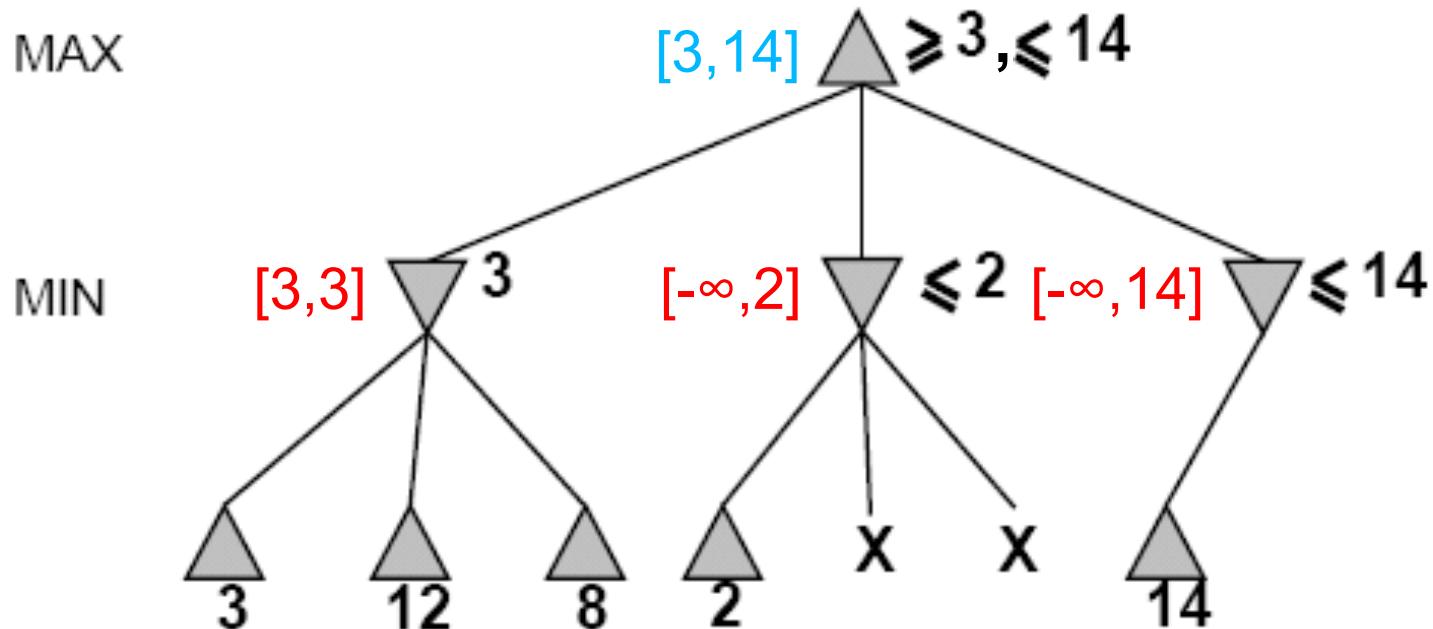
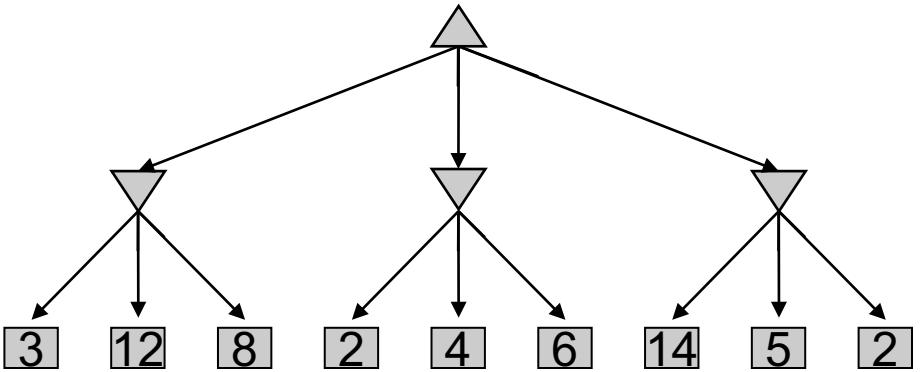
Poda α - β



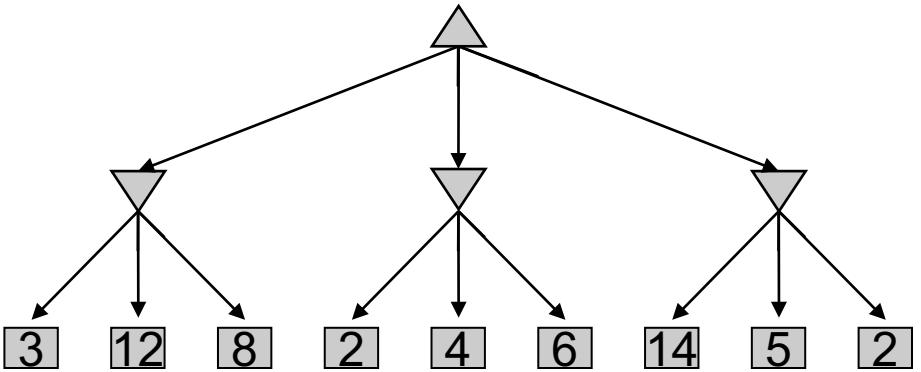
Poda α - β



Poda α - β

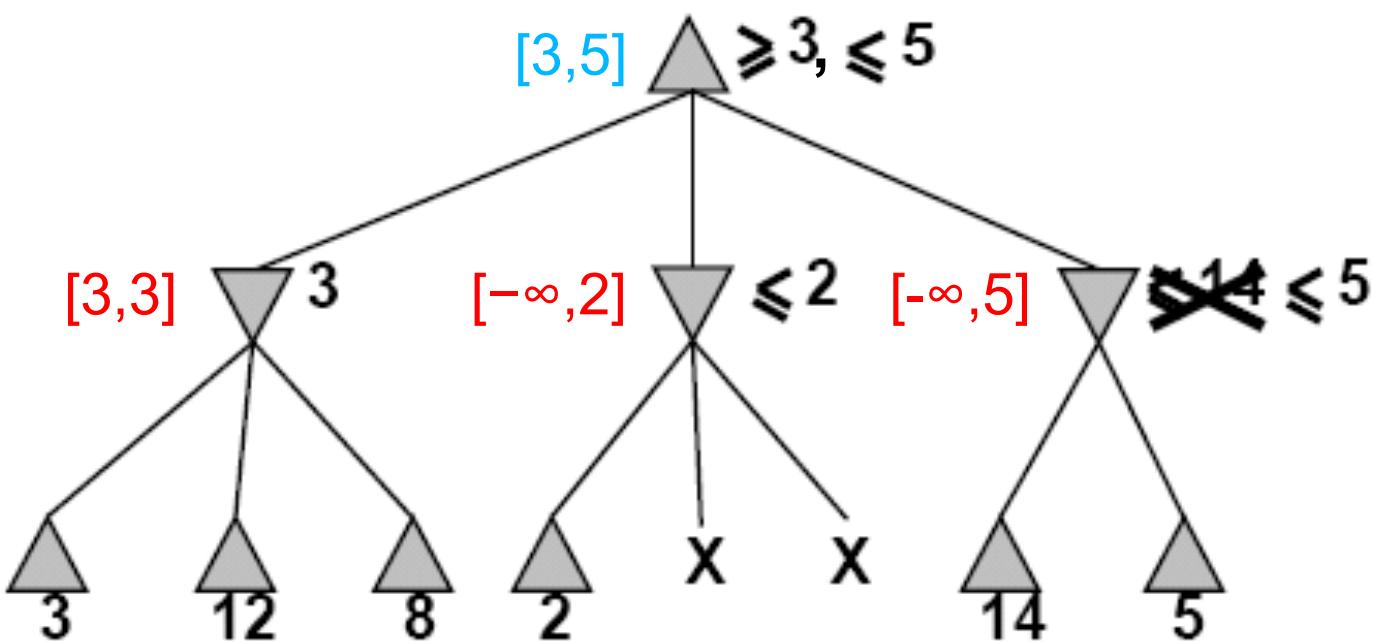


Poda α - β

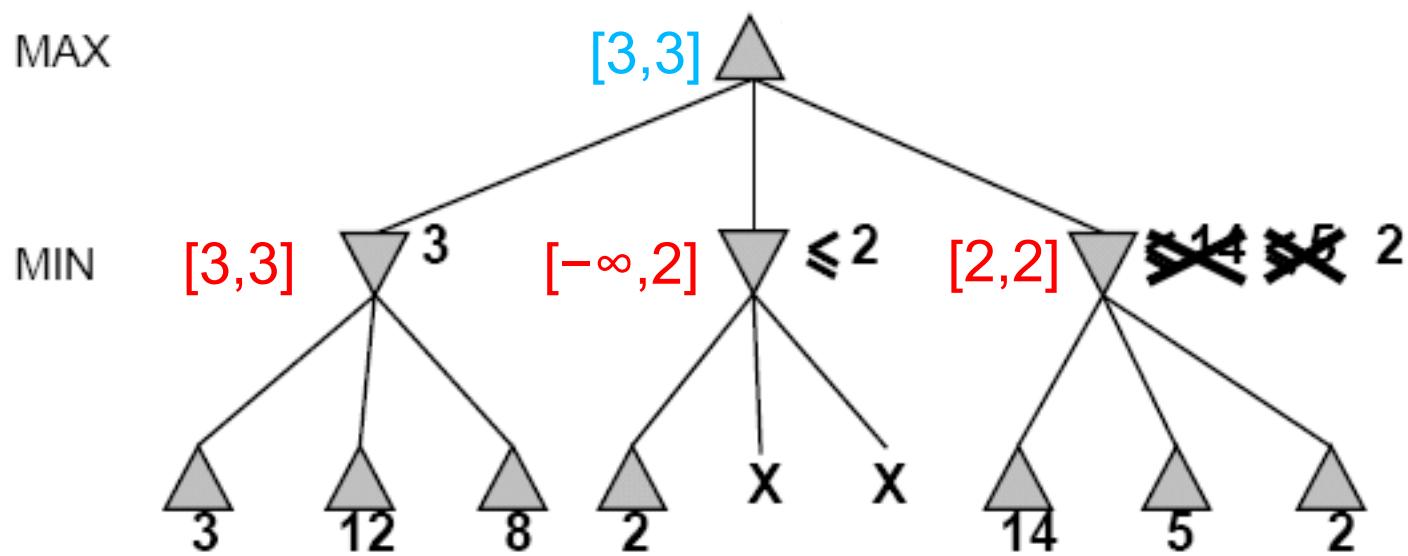
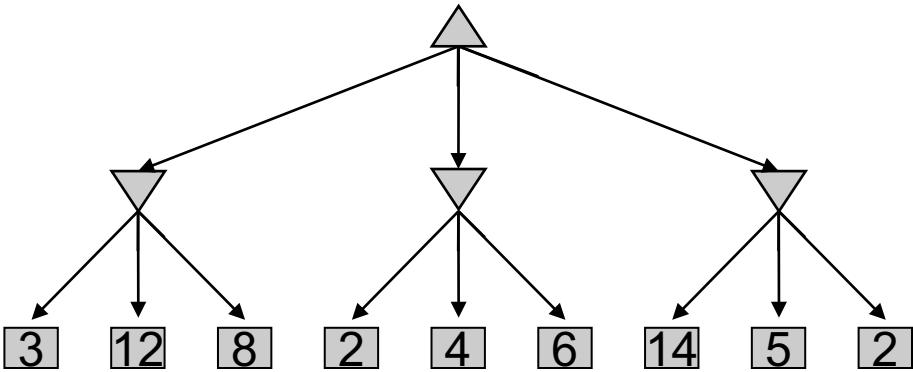


MAX

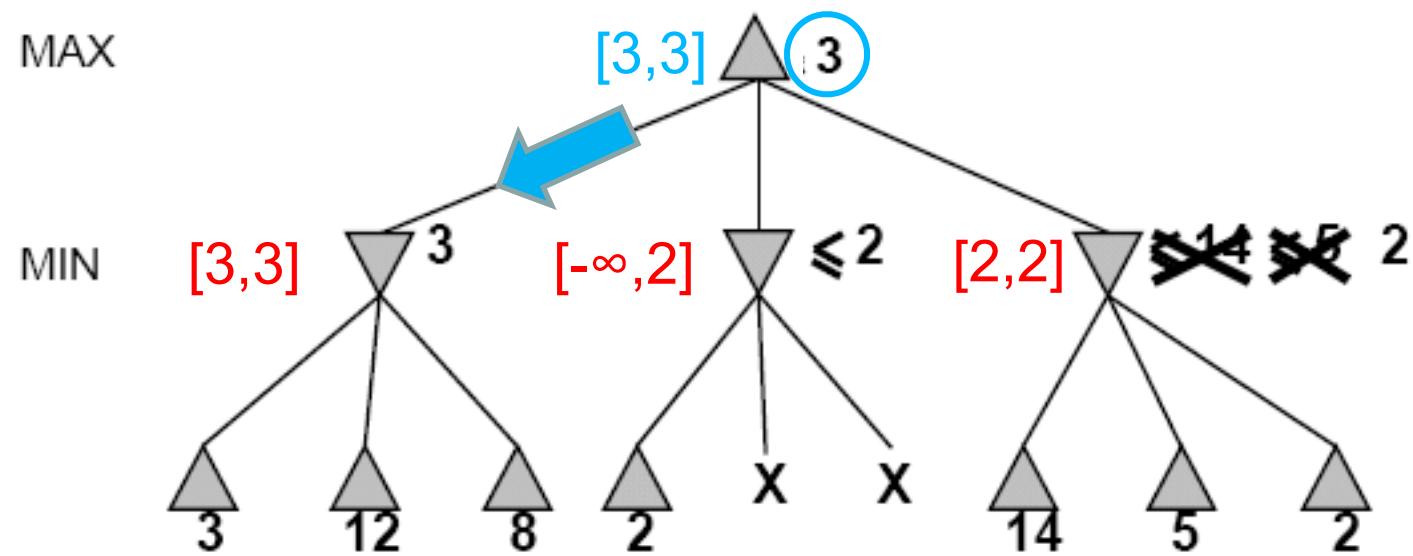
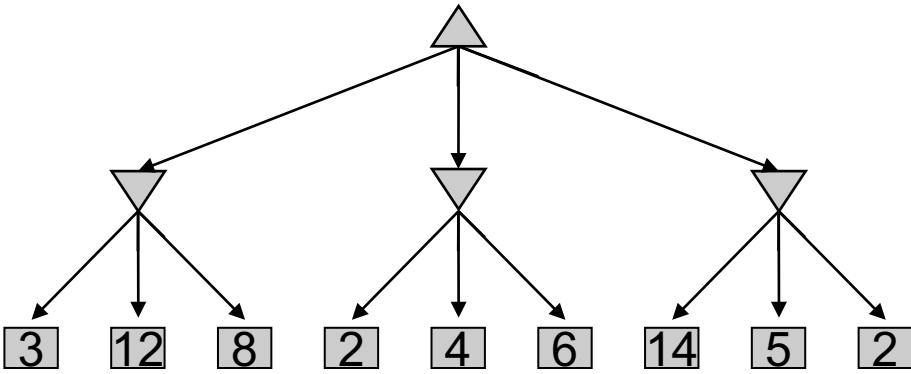
MIN



Poda α - β



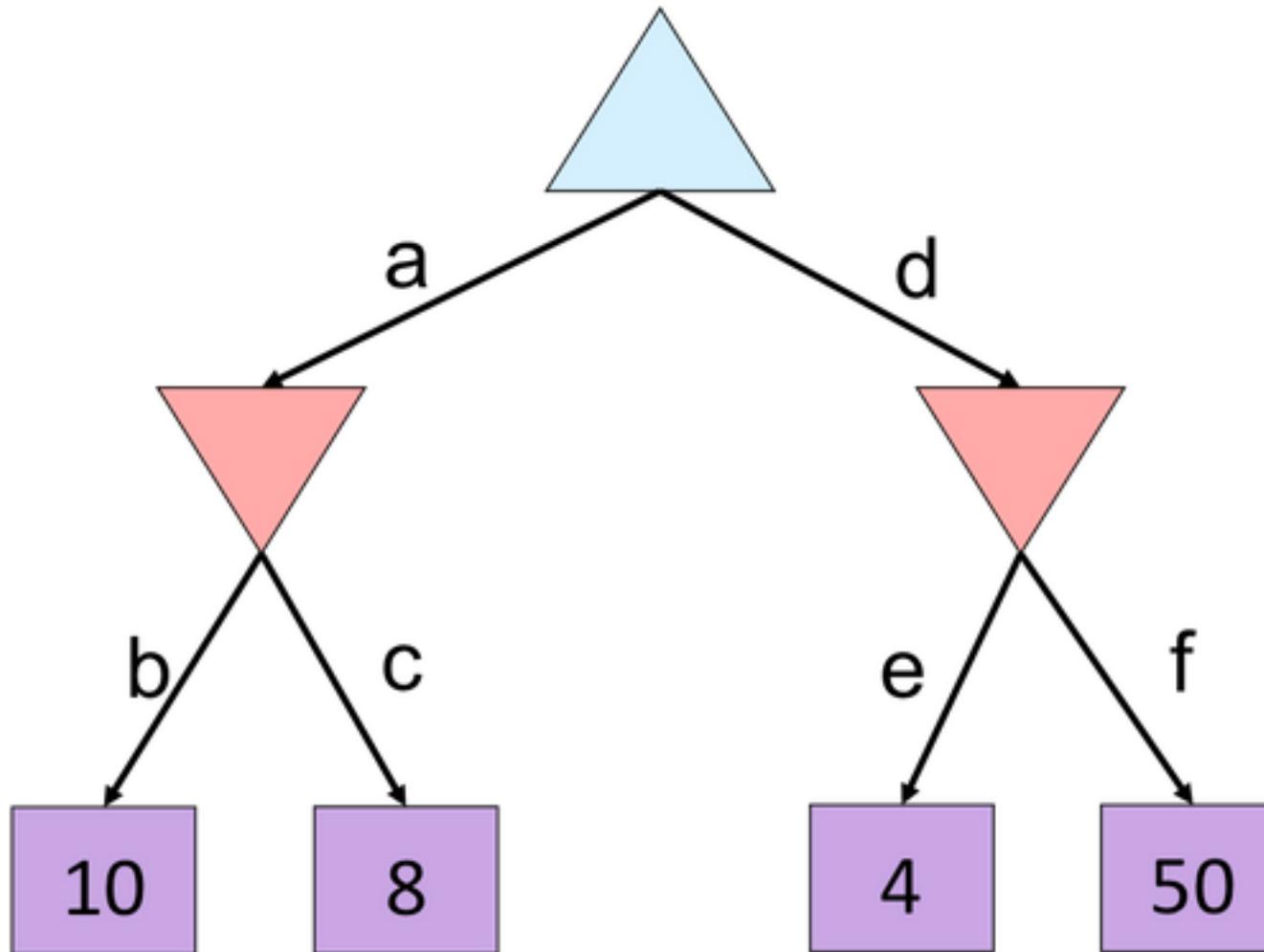
Poda α - β



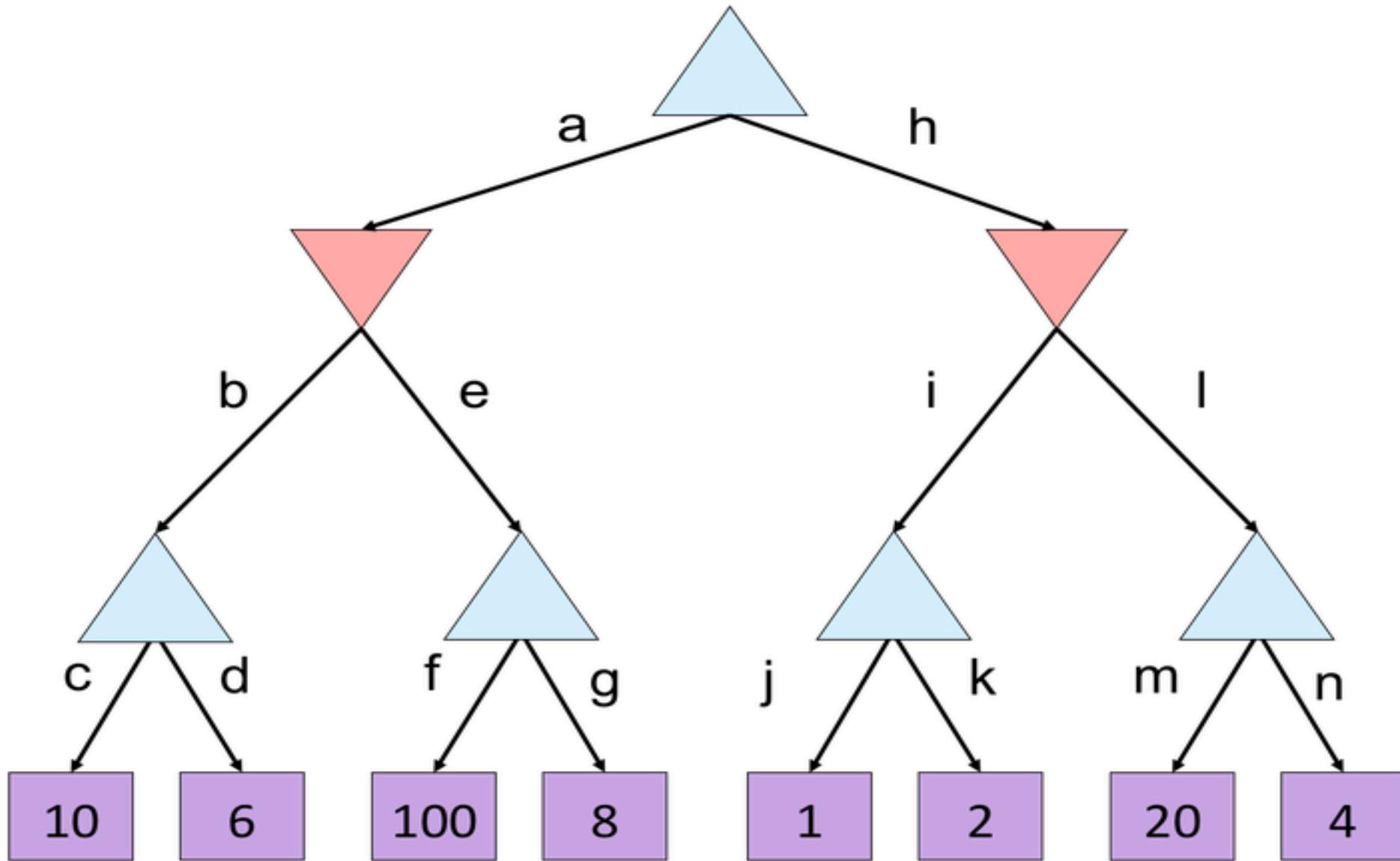
Poda α - β : propriedades

- Poda não influencia no valor minimax computado!
- Boa ordenação dos filhos aumenta a efetividade da poda
- Com ordenação perfeita:
 - Complexidade de tempo $\rightarrow O(b^{m/2})$
 - Duplica profundidade alcançada pela busca!

Q: Resolva minimax com α - β



Q: Resolva minimax com α - β



E quando não for possível definir uma função de avaliação explícita?
Por exemplo, o caso de jogos onde as regras são desconhecidas

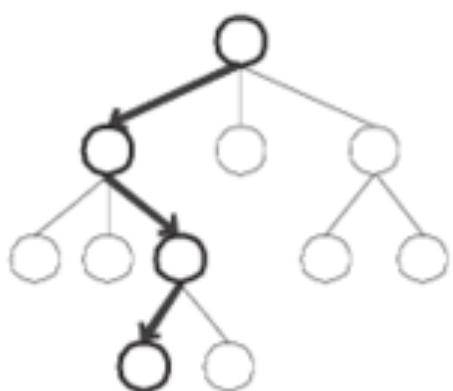
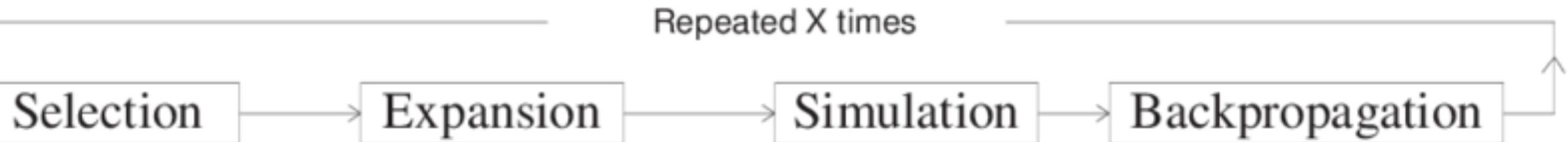
Monte Carlo Tree Search (MCTS)

- Aplicação em jogos é baseada nos *playouts*.
- Resumo
 - Em cada *playout*, o jogo vai até o fim **selecionando-se movimentos aleatoriamente**.
 - O **resultado** do jogo no *playout* é **usado para ponderar os nós na árvore de jogo** de modo que nós melhores sejam mais prováveis de serem escolhidos em *playouts* futuros

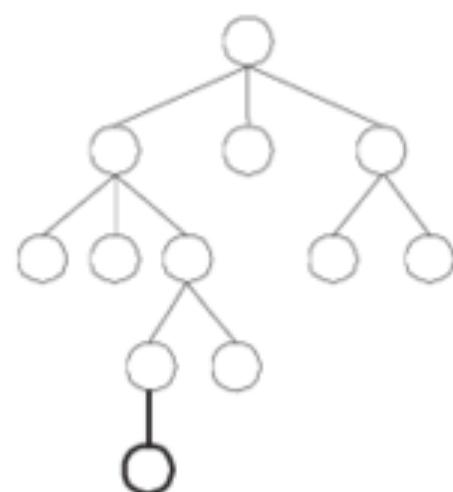
Monte Carlo Tree Search (MCTS)

- Cada round do MCTS possui 4 etapas:

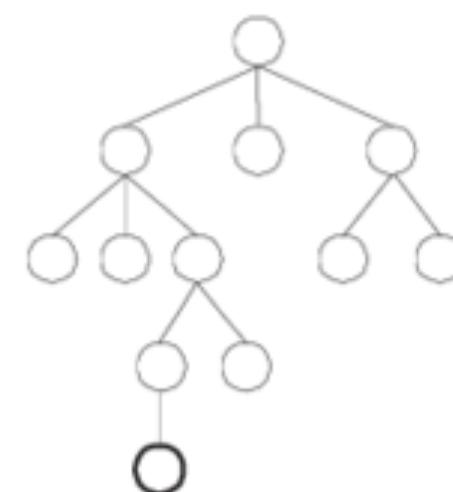
- Seleção: usa uma função de seleção para escolher o nó a expandir até atingir um que ainda não possua estatísticas gravadas. Ex: relação entre vitórias obtidas por partidas jogadas
- Expansão: cria a estatística para o nó que se chegou na fase de seleção. Expande um próximo nó.
- Simulação: para cada nó expandido, simule uma partida até o fim escolhendo ações de forma aleatória (*rollout*)
- Backpropagation: usa o resultado da simulação para atualizar a relação vitória/jogos nos nós no caminho até a raiz.



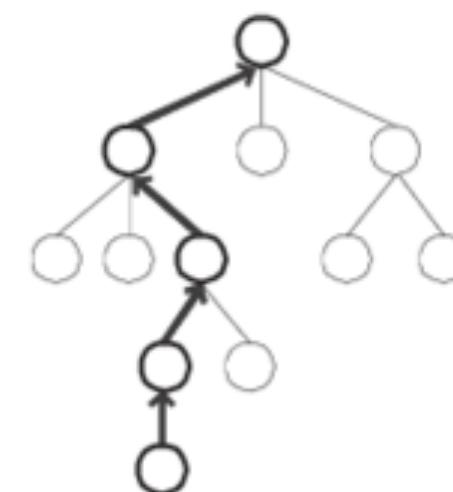
The selection function is applied recursively until a leaf node is reached



One or more nodes are created

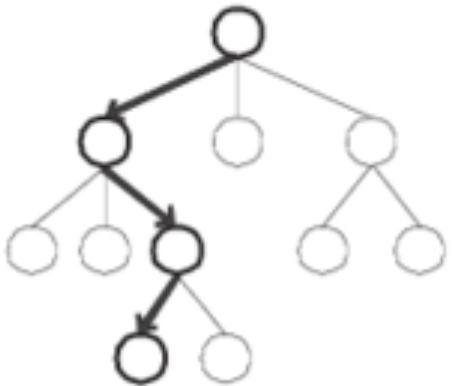


One simulated game is played



The result of this game is backpropagated in the tree

Selection



DIFICULDADE:

manter bom balanço entre
EXPLOITATION de variantes com altas taxas de
vitória vs. **EXPLORATION** de movimentos com
poucas simulações.

Fórmula famosa: UCT (*Upper Confidence Bound 1 applied to trees*) by *Kocsis and Szepesvári*

número de vitórias para o nó
considerado depois do movimento i

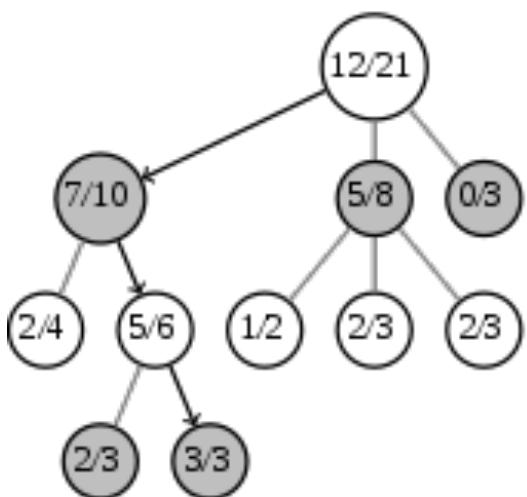
$$\frac{w_i}{n_i}$$

número de simulações para
o nó considerado depois do movimento i

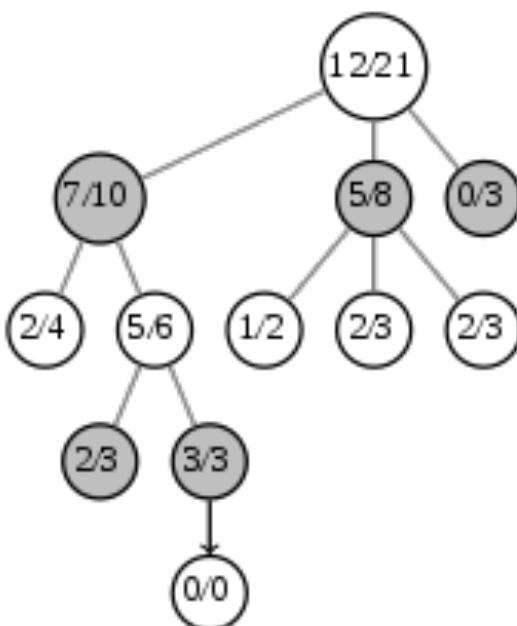
número total de simulações
depois do movimento i

C é o parâmetro de exploração = $\sqrt{2}$.
Na prática, escolhido empiricamente.

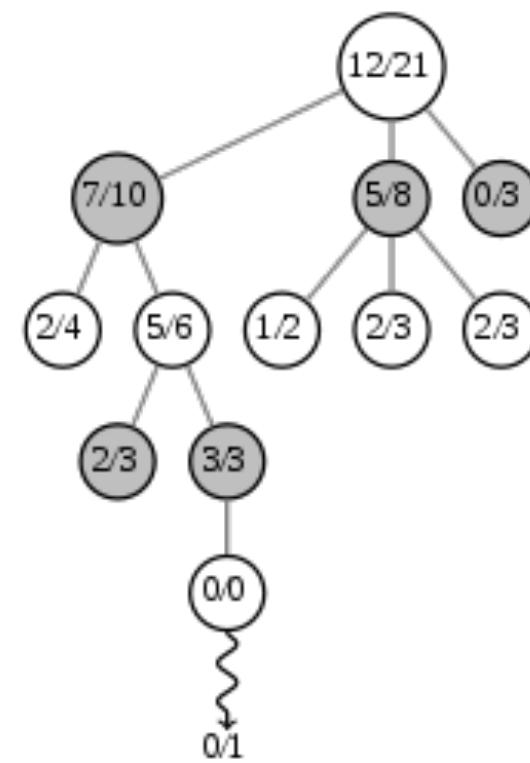
Selection



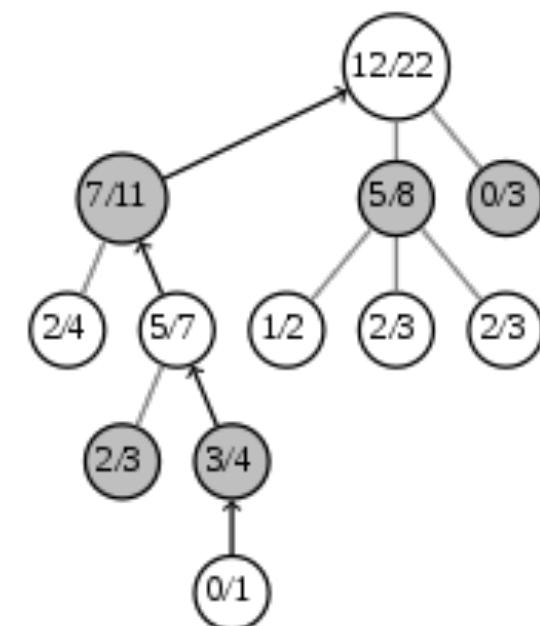
Expansion



Simulation



Backpropagation



Monte Carlo Tree Search (MCTS)

- Não precisa de uma função de avaliação explícita
- A árvore de jogo cresce de forma assimétrica uma vez que o método se concentra nas subárvores mais promissoras → resultados melhores para jogos com alto fator de expansão.
- Pode ser interrompido a qualquer momento.
- Pode não “enxergar” um caminho específico que leve a uma derrota
- Foi provado que a avaliação dos movimentos no MCTS **converge para minimax** ... só que lentamente!

Para saber mais....

- Campbell, M. S., & Marsland, T. A. (1983). A Comparison of Minimax Tree Search Algorithms*. *Artificial Intelligence*, 20(4), 347-367.
 - Seção 2 traz algoritmos alternativos
 - Seção 3.1 traz critérios de comparação de performance

DeepBlue - 1997



IBM RS/6000

- IBM RS/6000
 - 480 VLSI [espec xadrez]
 - busca de hdw, 200M pos/sec
 - min-max, podas alfa-beta
 - profundidade até 40 [análise de movimentos forçados]
 - BD de finais de jogos
 - 40k aberturas
 - fun. aval. com 8k características

Confronto '97: 2 x 3 x 1

Para saber mais....

- Campbell, M., Hoane, A. J., & Hsu, F. H. (2002). Deep blue. *Artificial intelligence*, 134(1), 57-83.
 - Seção 4 descreve o mecanismo de busca baseado em software: "dual credit with delayed extensions"
 - algoritmo está descrito (até linha 26 é bastante similar a alpha-beta)
 - Seção 7 fala da função de avaliação
 - "The Deep Blue evaluation function is essentially a sum of feature values. The chess chip recognizes roughly 8000 different “patterns”, and each is assigned a value. Features range from very simple, such as a particular piece on a particular square, to very complex, as will be described below in Section 7.2."



Para saber mais...

- Buro, M. (1995, March). Logistello: A strong learning othello program. In *19th Annual Conference Gesellschaft für Klassifikation eV* (pp. 1-3).
 - função de avaliação linear ponderada
 - características mais complexas além de posições do tabuleiro
 - pesos são aprimorados através de “self play”
 - base de conhecimento com 23k jogos



Para saber mais...

- Schaeffer, J., Lake, R., Lu, P., & Bryant, M. (1996). CHINOOK the world man-machine checkers champion. *AI Magazine*, 17(1), 21.
 - In 1992, the seemingly unbeatable World Checker Champion Marion Tinsley defended his title against the computer program CHINOOK.
- J. Schaeffer et al. Checkers is solved. *Science* 317, 1518-1522. DOI: 10.1126/science.1144079 (2007).
 - Heurísticas —> Perfeição

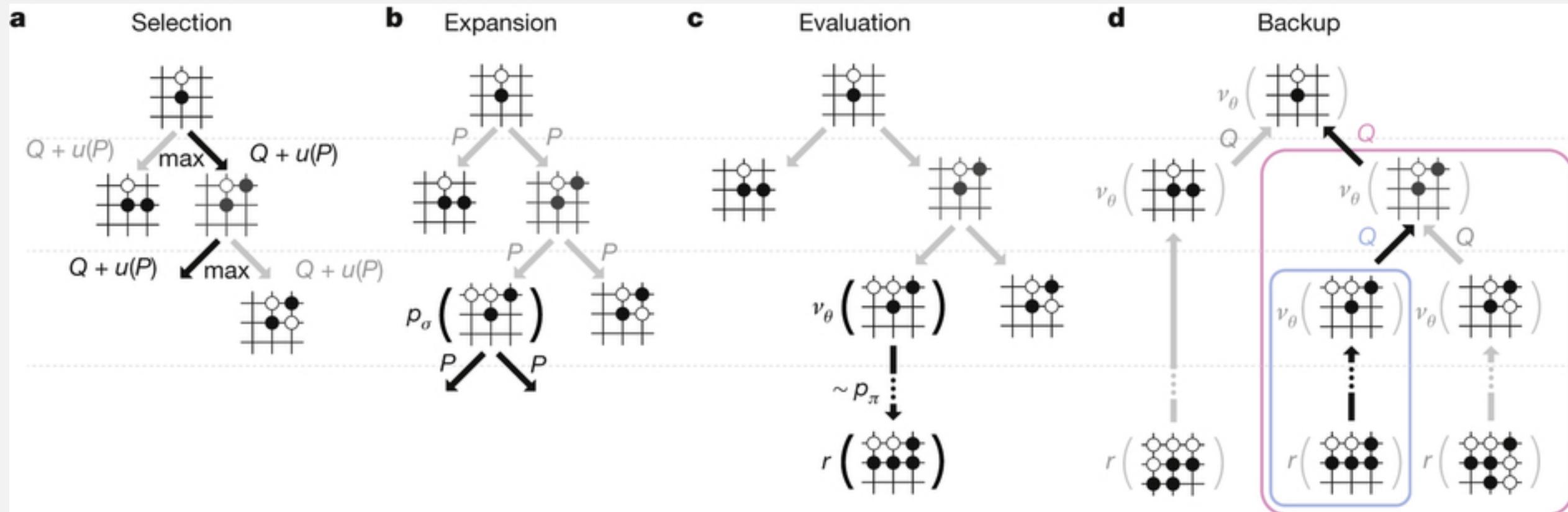


Para saber mais...

- Silver, D. et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), 484-489.
 - from Google DeepMind
 - "We also introduce a new search algorithm that combines Monte Carlo simulation with value and policy networks. Using this search algorithm, our program AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go."
- March 2016, AlphaGo was awarded an honorary 9-dan (master) level in 19×19 go for defeating Lee Sedol by 4 x 1.

Para saber mais...

- Silver, D. et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), 484-489.





<http://www.saense.com.br>

<https://www.facebook.com/saense/>

Hendrik Macedo

Escreve sobre *Inteligência Artificial* no Saense.

<http://www.saense.com.br/autores/artigos-publicados-por-hendrik-macedo/>