Jogos de 2 jogadores

Capítulo 6

Resumo

- ♢ Jogos
- ♢ Jogada perfeita
  - Decisões minimax
  - Corte α–β (pruning)
- ♢ Limites dos recursos e avaliação aproximada
- ♢ Jogos de Sorte
- ♢ Jogos de informação imperfeita

Jogos vs. problemas de pesquisa

Com um oponente "imprevisível" $\Rightarrow$ a solução é uma estratégia
que especifica a jogada para todas as respostas possíveis do oponente

Limites temporais $\Rightarrow$ em vez de encontrar o objectivo procura aproximações

Plano de ataque:

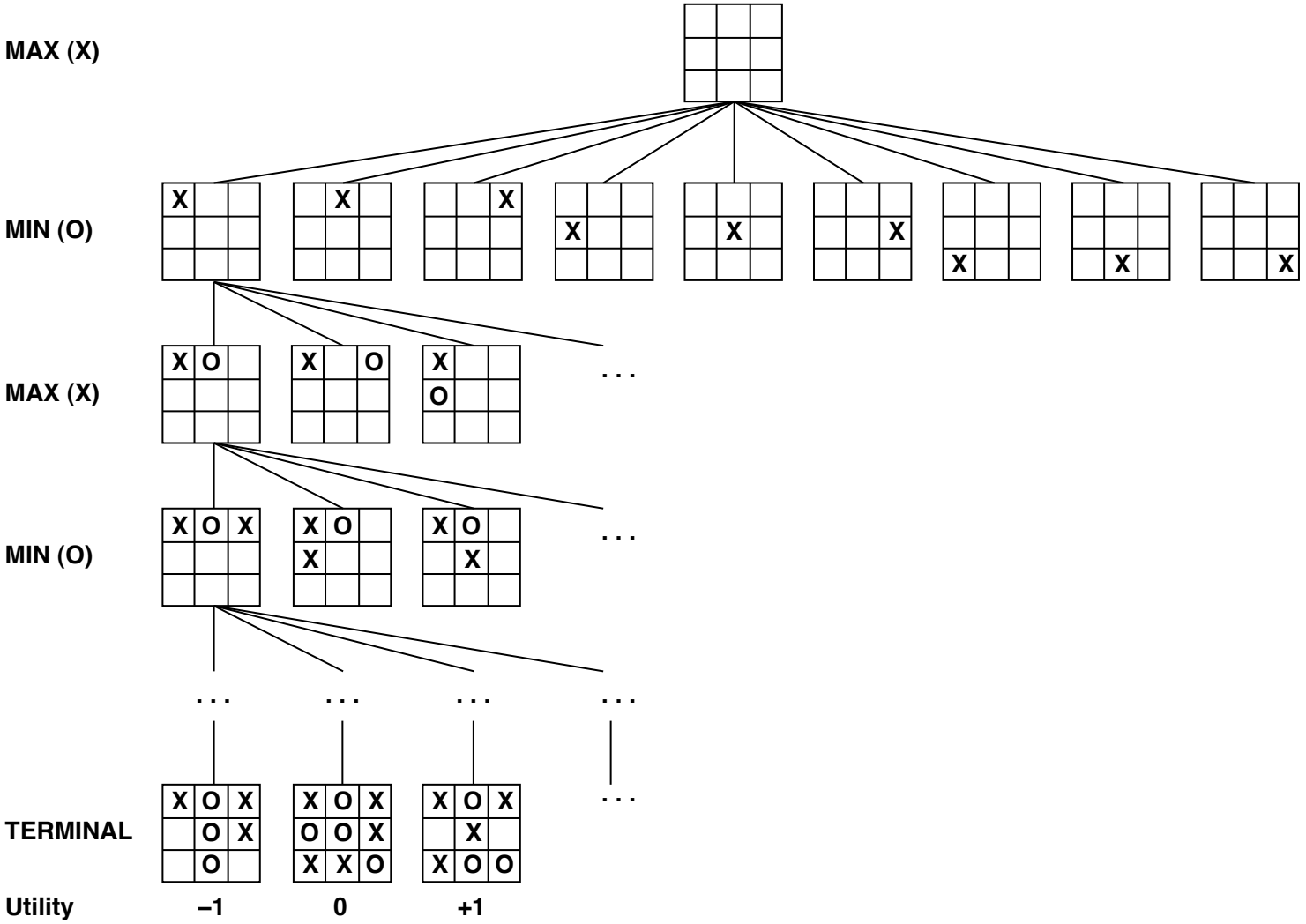
- O computador considera diferentes linhas de jogo (Babbage, 1846)
- Algoritmo para jogar de forma perfeita (Zermelo, 1912; Von Neumann, 1944)
- Horizonte finito, avaliação aproximativa (Zuse, 1945; Wiener, 1948; Shannon, 1950)
- Primeiro programa que joga Xadrez (Turing, 1951)
- Técnicas de aprendizagem automática para melhorar a correcção da avaliação (Samuel, 1952–57)
- Cortes para permitir pesquisas mais profundas (McCarthy, 1956)

Tipos de Jogos

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

Árvore do Jogo

(2-jogadores, jogo deterministico, jogam alternados)

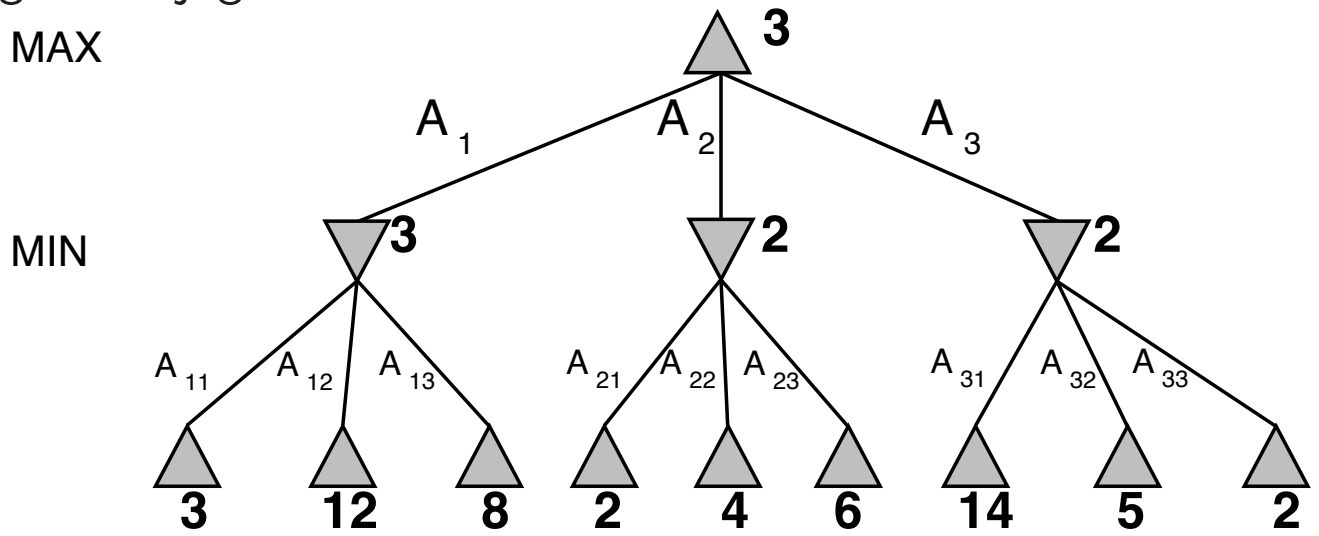

MAX (X)
MIN (O)
MAX (X)
MIN (O)
TERMINAL
Utility
–1
0
+1

Minimax

Jogada perfeita para jogos determiniscos, com informação perfeita

Ideia: escolher a jogada com maior valor minimax

E.g., jogo de 2 jogadores:



Algoritmo Minimax

function MINIMAX-DECISION(*state*) **returns** *an action*
  inputs: *state*, current state in game

  return the *a* in ACTIONS(*state*) maximizing MIN-VALUE(RESULT(*a*, *state*))

---

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*

Propriedades do minimax

Completo??

Propriedades do minimax

Completo?? Só se a árvore é finita (o xadrez tem regras especificas para ser finito).

Pode existir uma estratégia finita mesmo que a arvore seja infinita!

Óptimo??

Propriedades do minimax

Completo?? Sim, se a árvore é finita (o xadrez tem regras especificas para ser finito).

Óptimo?? Sim, contra um oponente óptimo, Senão??

Complexidade Temporal??

Propriedades do minimax

Completo?? Sim, se a árvore é finita (o xadrez tem regras especificas para ser finito).

Óptimo?? Sim, contra um oponente óptimo, Senão??

Complexidade Temporal?? $O(b^m)$

Complexidade Espacial??

Propriedades do minimax

Completo?? Sim, se a árvore é finita (o xadrez tem regras especificas para ser finito).

Óptimo?? Sim, contra um oponente óptimo, Senão??

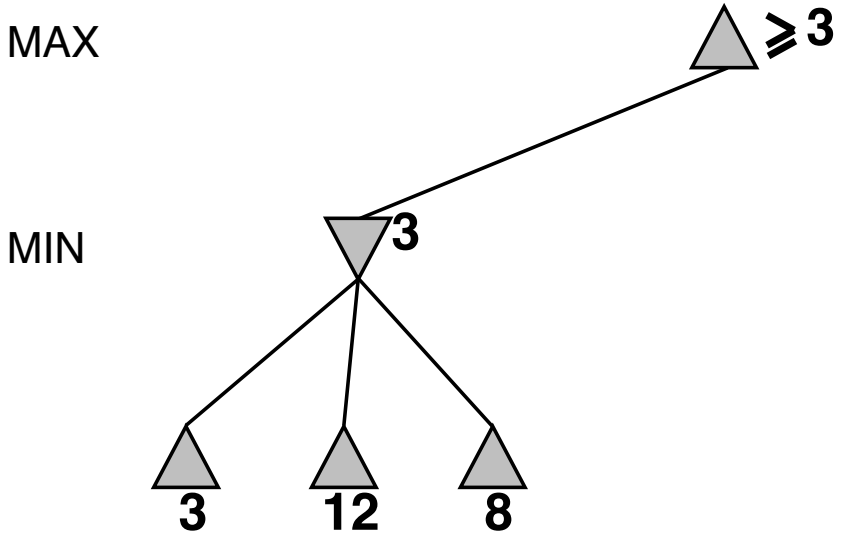
Complexidade Temporal?? $O(b^m)$

Complexidade Espacial?? $O(bm)$ (exploração com a pesquisa em profundidade)

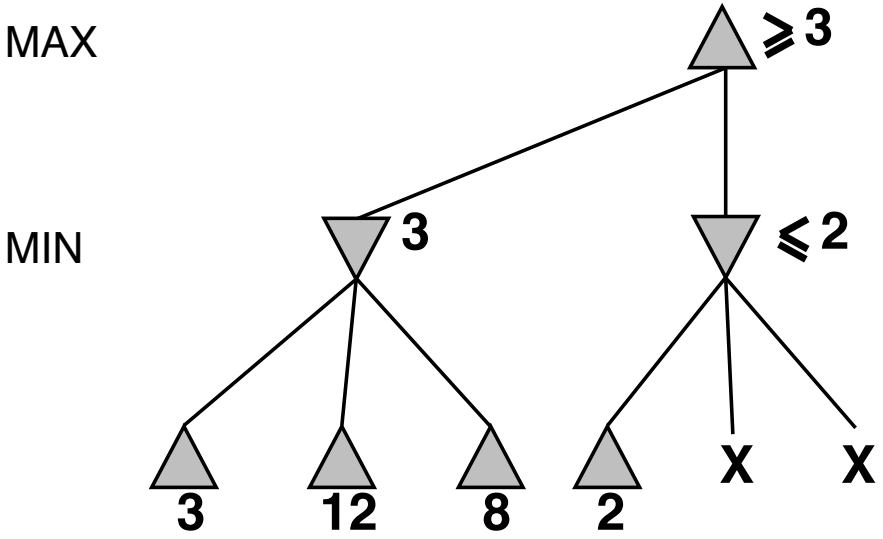
Para o xadrez, $b \approx 35$, $m \approx 100$ para jogos "razoáveis"
$\Rightarrow$ não é possível encontrar a solução exacta

Mas é necessário explorar todos os caminhos?

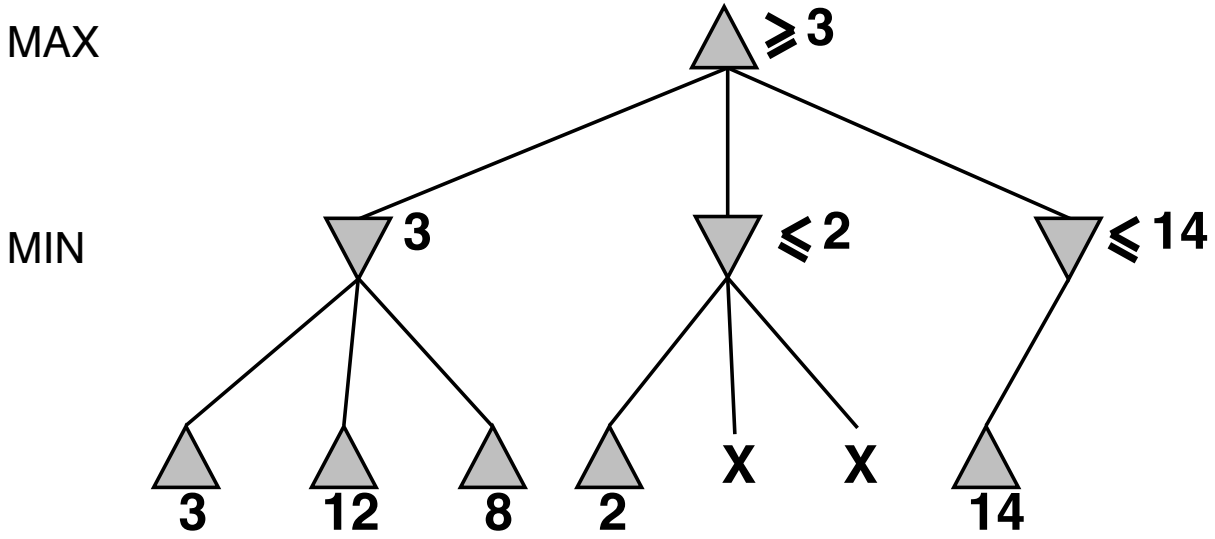
α–β exemplo de corte



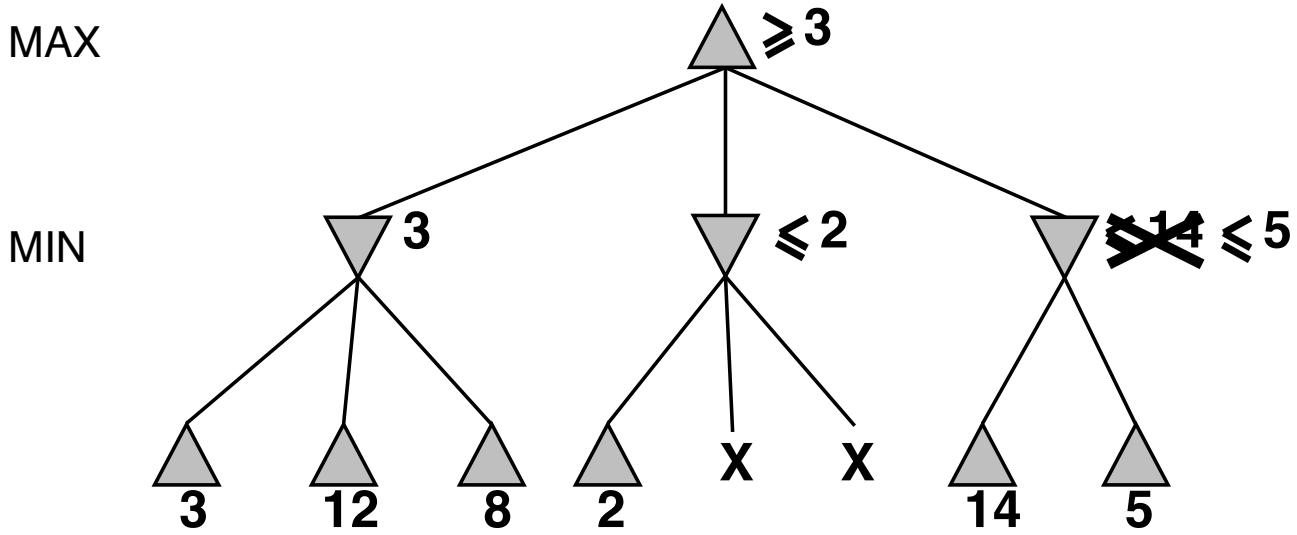
α–β exemplo de corte



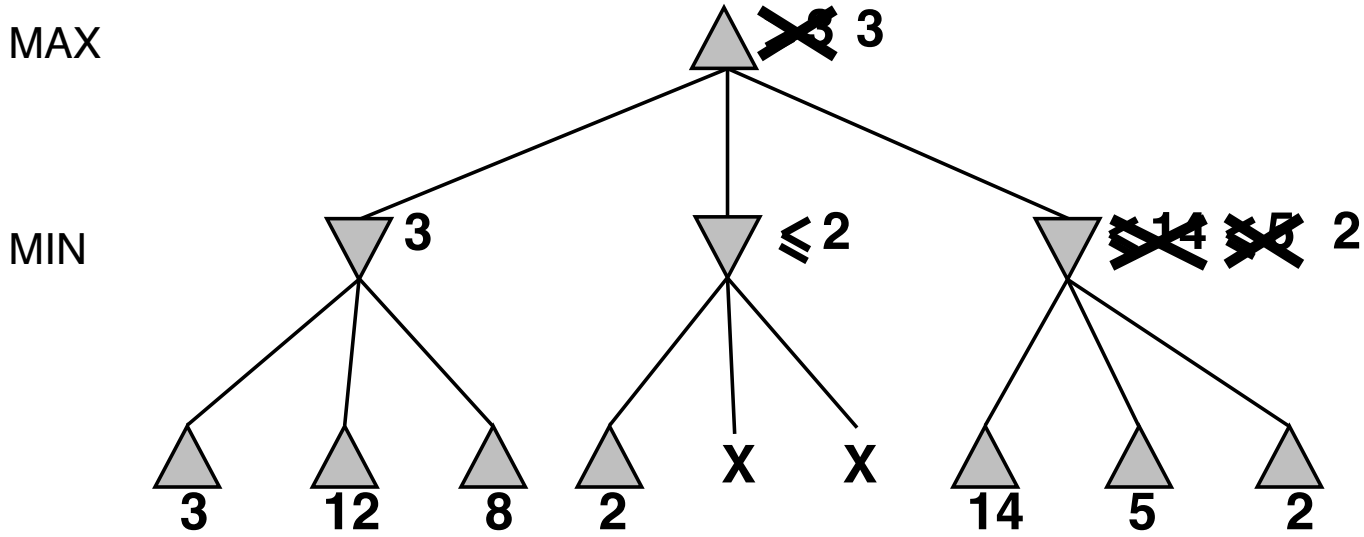
α–β exemplo de corte



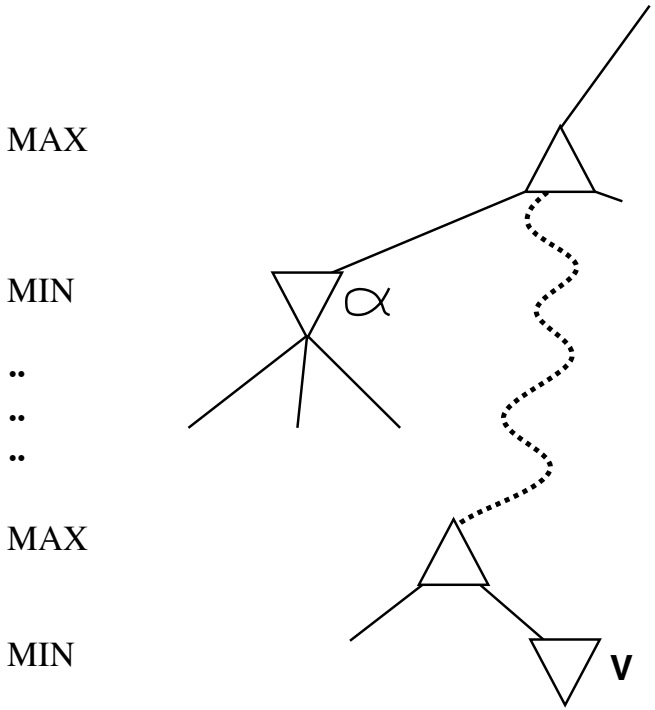
α–β exemplo de corte



α–β exemplo de corte



Porque é que se chama α–β?



α é o melhor valor (de MAX) encontrado até ao caminho corrente

Se V é pior que α, MAX vai evitá-lo $\Rightarrow$ corta o ramo

β define-se de forma semelhante para MIN

O Algoritmo α–β

function ALPHA-BETA-DECISION($state$) **returns** an action
  return the a in ACTIONS($state$) maximizing MIN-VALUE(RESULT(a, $state$))

---

function MAX-VALUE($state, \alpha, \beta$) **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

---

function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  same as MAX-VALUE but with roles of α, β reversed

Propriedades do α–β

O corte **não** afecta o resultado final

A ordem das jogadas pode aumentar os cortes

Com uma "ordem perfeita," Complexidade temporal $= O(b^{m/2})$
$\Rightarrow$ **dobra** a profundidade da solução

Um exemplo simples do valor de raciocinar sobre a relevância dos cálculos que vão ser feitos (uma forma de metaraciocínio)

Infelizmente, 35^{50} ainda é impossível!

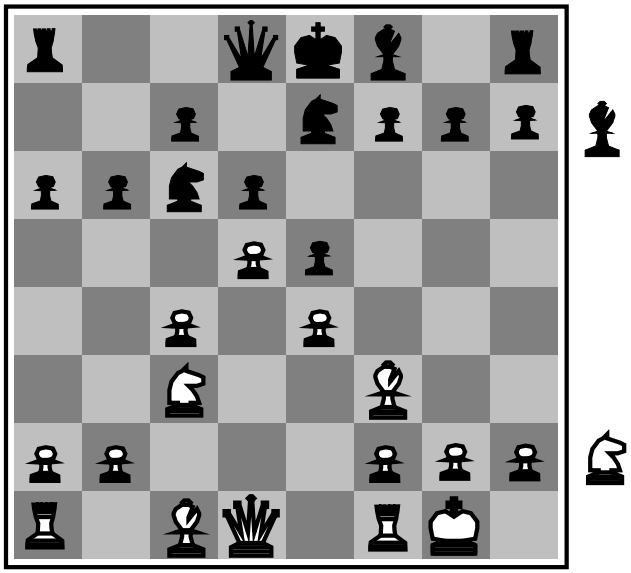
Limites dos recursos

Aproximação standart:

- Usar CUTOFF-TEST em vez de TERMINAL-TEST
  e.g., limitar a profundidade (e talvez adicionar quiescence search-pesquisa dos nós mais interessantes)
- Usar EVAL em vez de UTILITY
  i.e., função de avaliação estima a adequação de uma posição

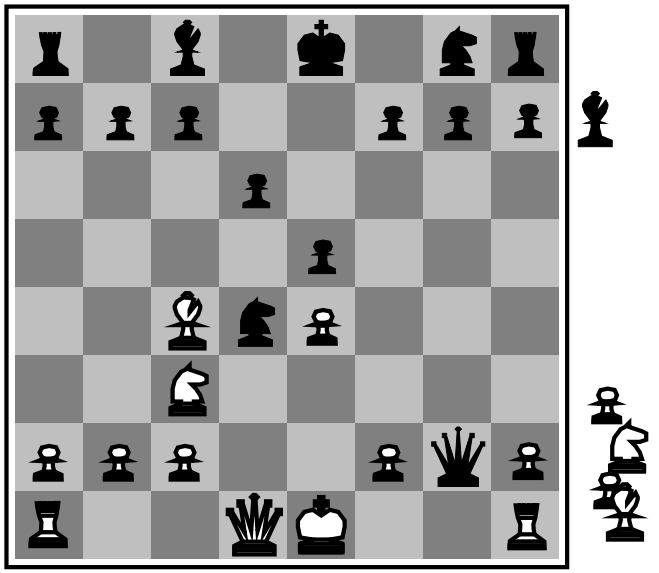
Suponha que temos 100 segundos, explorar 10^4 nós/segundo
$\Rightarrow 10^6$ nós por jogada $\approx 35^{8/2}$
$\Rightarrow \alpha$–β atinge a profundidade 8 $\Rightarrow$ um bom programa para jogar xadrez

Funções de avaliação



Black to move

White slightly better



White to move

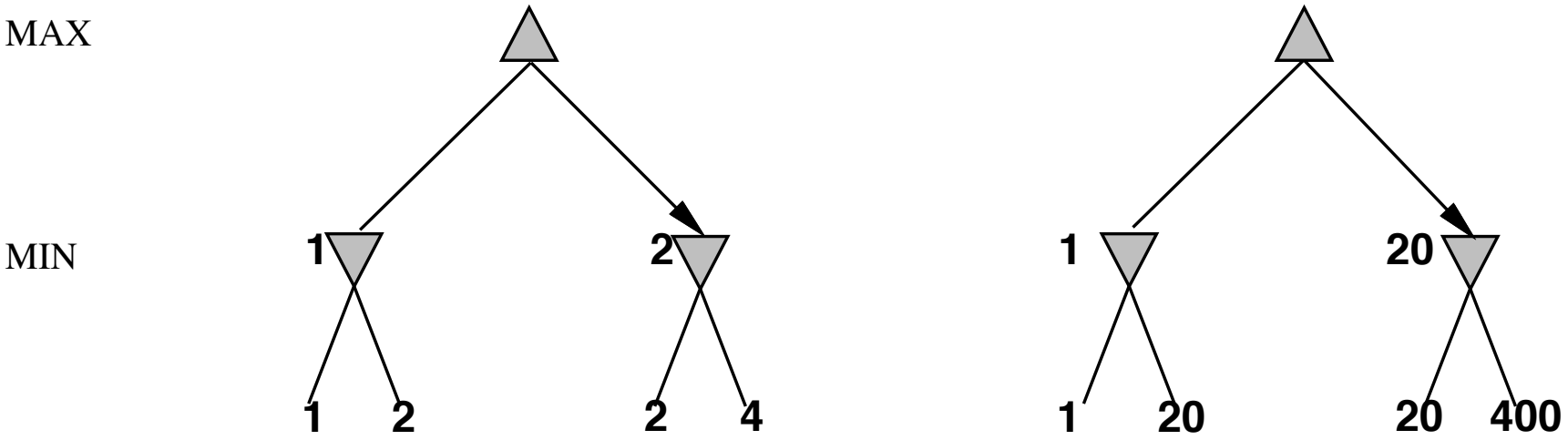
Black winning

Para o xadrez, a soma linear pesada dos atributos

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

e.g., $w_1 = 9$ com $f_1(s) =$ (numero de rainhas brancas) – (numero de rainhas pretas), etc.

Desvio: Os valores exactos não interessam



O comportamento é preservado numa transformação **monotonica** de EVAL

Só a ordem interessa:

A recompensa em jogos deterministicos funciona como uma função de utilidade ordinal

Jogos deterministicos na prática

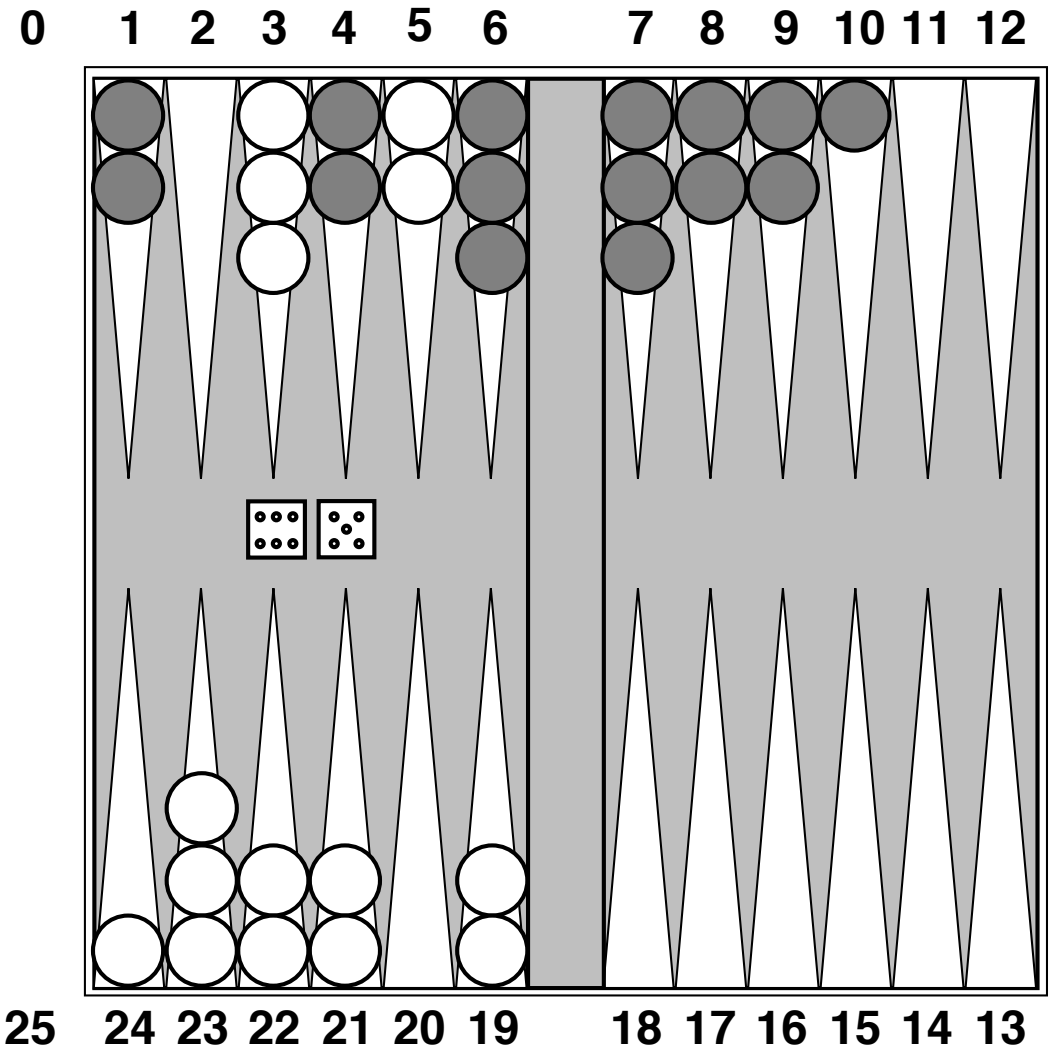
Damas: Chinook terminou o reinado de 40 anos como campeão mundial de Marion Tinsley em 1994. Usou uma base de dados com as posições finais que definiam uma jogada perfeita para todas as posições possíveis com 8 ou menos peças num tabuleiro, um total de 443,748,401,247 posições.

Xadrez: o Deep Blue derrotou o campeão mundial Gary Kasparov num torneio de 6 jogos em 1997. O Deep Blue pesquisa 200 milhões de posições por segundo, usa uma função de avaliação muito sofisticada, e métodos não divulgados para estender alguns níveis no limite da pesquisa até à jogada 40.

Otello: O campeão humano recusou competir contra computadores, que são muito bons neste jogo.

Go: O campeão humano recusou competir contra computadores, que que são muito maus neste jogo. No go, $b > 300$, a maioria dos programas usa bases de conhecimento com padrões para sugerir a melhor jogada.

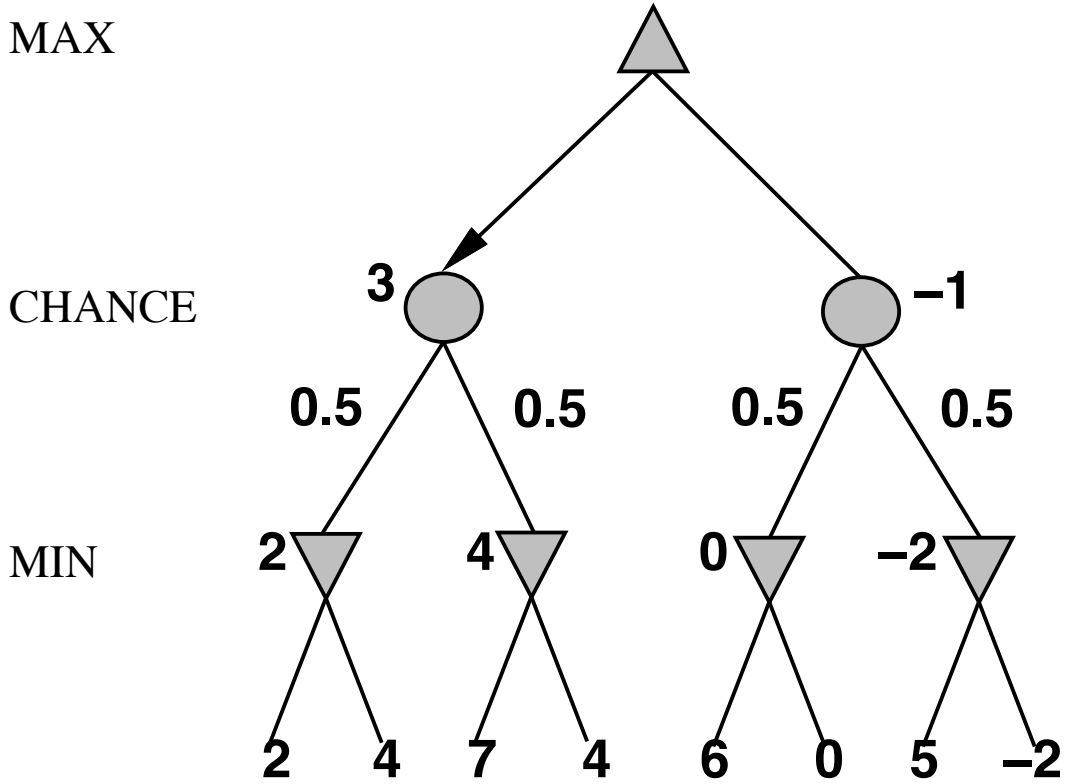
Jogos não deterministicos: backgammon -gamão



Jogos não deterministicos em geral

Nos jogos não determisticos a sorte é dada pelos dados ou pelo baralhar das cartas

Um exemplo simples é atirar uma moeda ar:



Algoritmo para jogos não deterministicos

EXPECTIMINIMAX dá a jogada perfeita

É como o MINIMAX, excepto que tem que lidar com nós de sorte:

...
Se *estado* é um nó MAX **então**
    retorna o maior EXPECTIMINIMAX-VALUE de SUCCESSORS(*estado*)
Se *estado* é um nó MIN **então**
    retorna o menor EXPECTIMINIMAX-VALUE de SUCCESSORS(*estado*)
Se *estado* é um nó de sorte **então**
    retorna a média de EXPECTIMINIMAX-VALUE de SUCCESSORS(*estado*)
...

Jogos não deterministicos na prática

Os dados aumentam b: 21 resultados possíveis com 2 dados
Backgammon (Gamão) $\approx$ 20 jogadas (podem ser 6,000 com 1-1 nos dados)

$$\text{profundidade } 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$$

À medida que a profundidade aumenta. a probabilidade de atingir um dado nó diminui

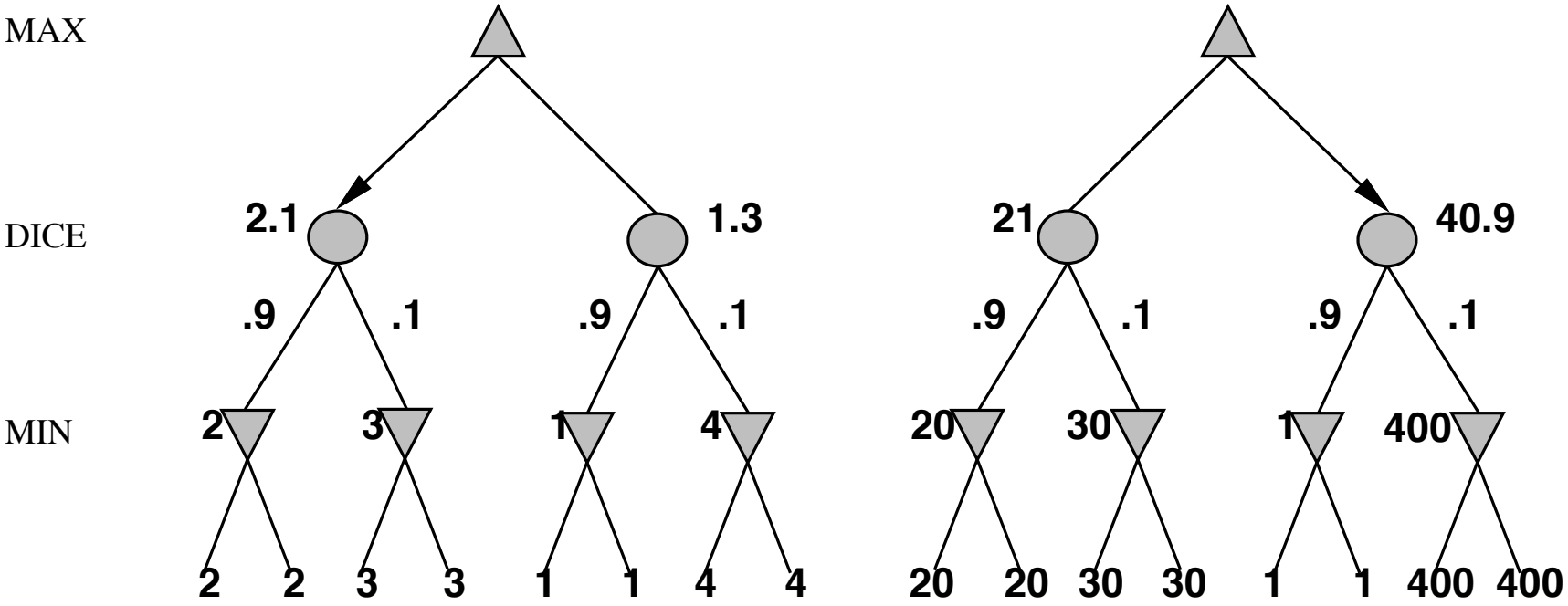
$\Rightarrow$ a importância de olhar para a frente é reduzida

O corte α–β tem menos efeito

TDGammon usa pesquisa em profundidade limitada a 2 + uma função de avaliação Eval muito boa

$\approx$ nível do campeão do mundo

Desvio: valores exactos são importantes



O comportamento é preservado só com transformações lineares positivas de Eval

Assim Eval deve ser proporcional à recompensa esperada

Jogos com informação imperfeita

E.g., jogos de carta, onde as cartas iniciais do oponente não são conhecidas

Podemos calcular a probabilidade de cada jogada possível

É como se tivéssemos um dado muito grande para atirar no inicio do jogo*

Ideia: calcular o valor minimax de cada acção em cada jogada (conjunto de cartas),

escolher a acção com o maior valor esperado de todas as jogadas (conjuntos de cartas)*

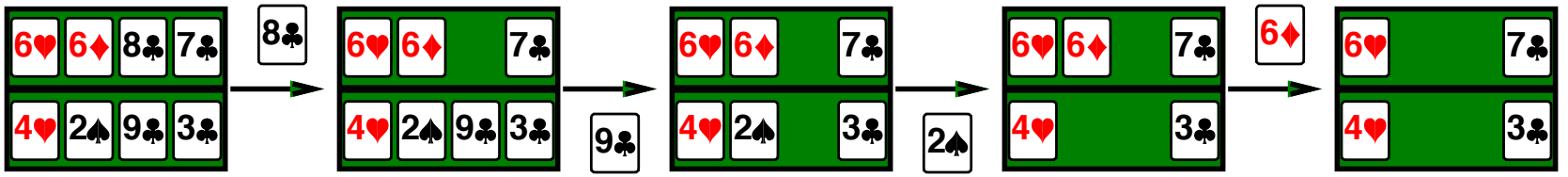
Caso especial: se uma acção é óptima para todos os conjuntos de cartas então é óptima.*

GIB, o melhor programa de bridge actual

1) gera 100 mãos consistentes com a informação conhecida
2) escolhe a acção que ganha mais mãos em média

Exemplo

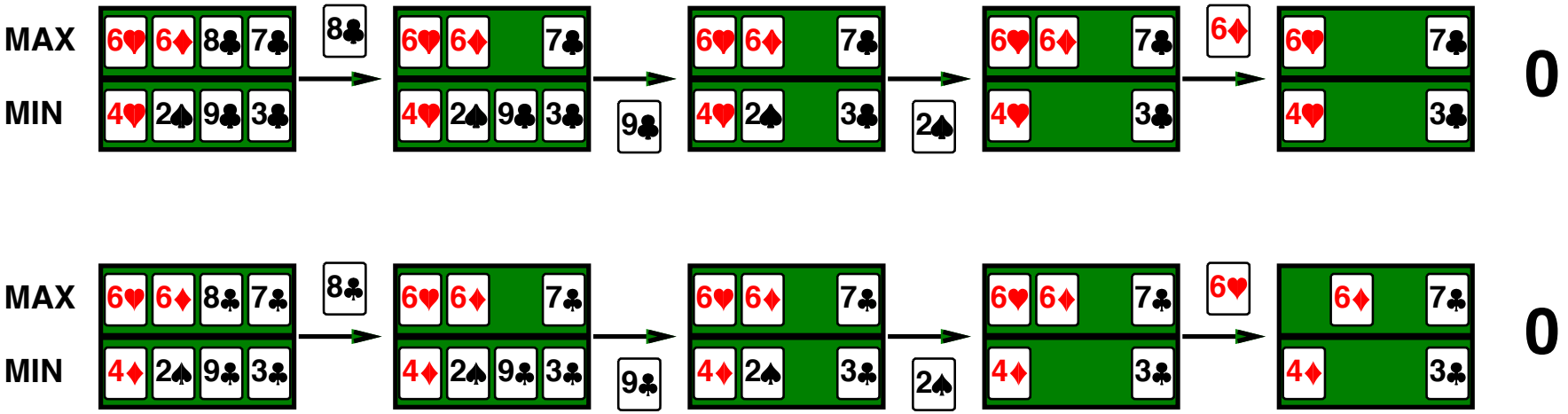
Four-card bridge/whist/hearts hand, MAX joga primeiro



0

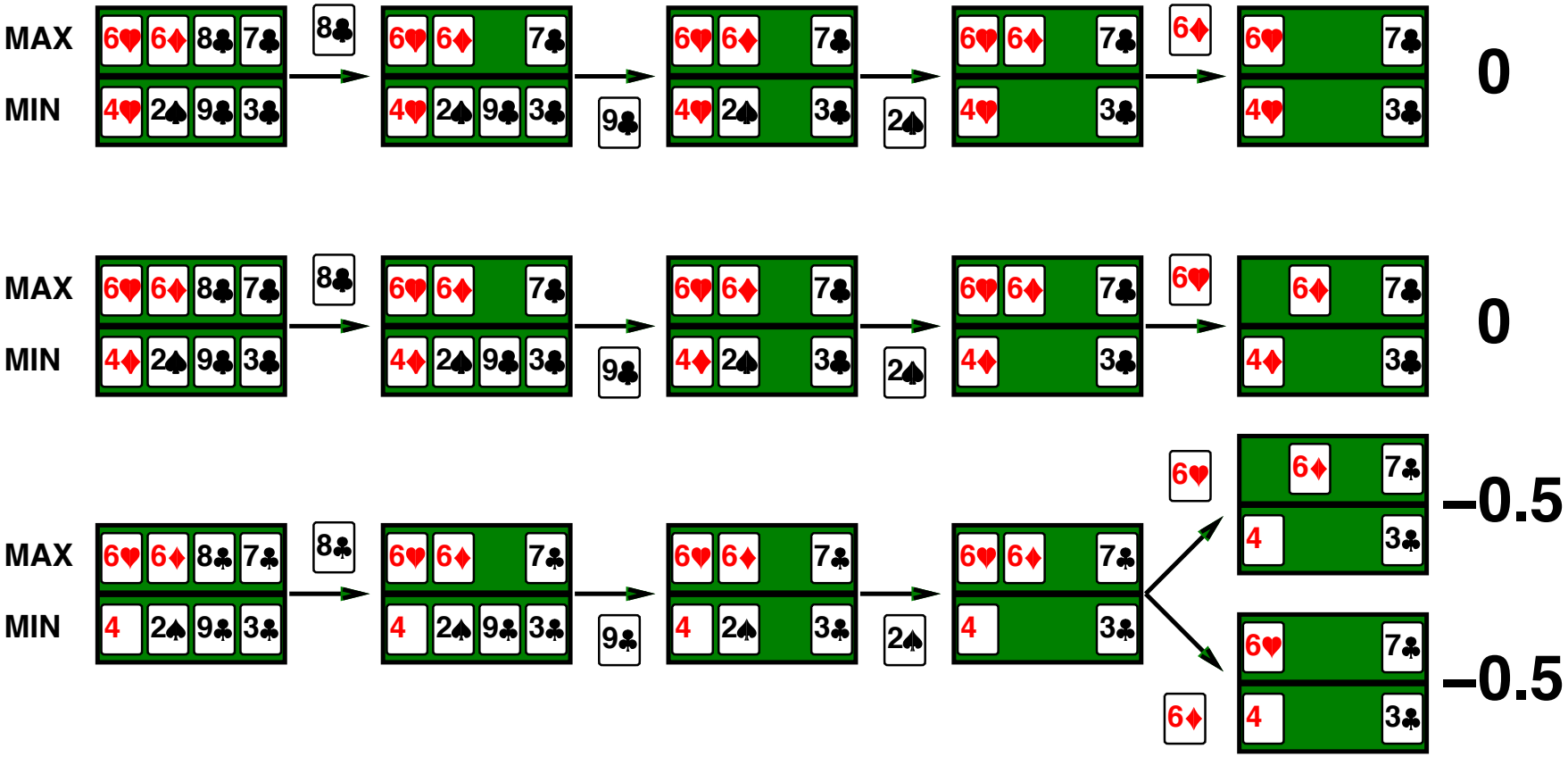
Exemplo

Four-card bridge/whist/hearts hand, MAX joga primeiro



Exemplo

Four-card bridge/whist/hearts hand, MAX joga primeiro



Resumo

Games are fun to work on! (and dangerous)

They illustrate several important points about AI

- ♢ perfection is unattainable ⇒ must approximate
- ♢ good idea to think about what to think about
- ♢ uncertainty constrains the assignment of values to states
- ♢ optimal decisions depend on information state, not real state

Games are to AI as grand prix racing is to automobile design