Busca Competitiva

Aydano Pamponet Machado
aydano.machado@ic.ufal.br

Busca competitiva ℗ Jogos

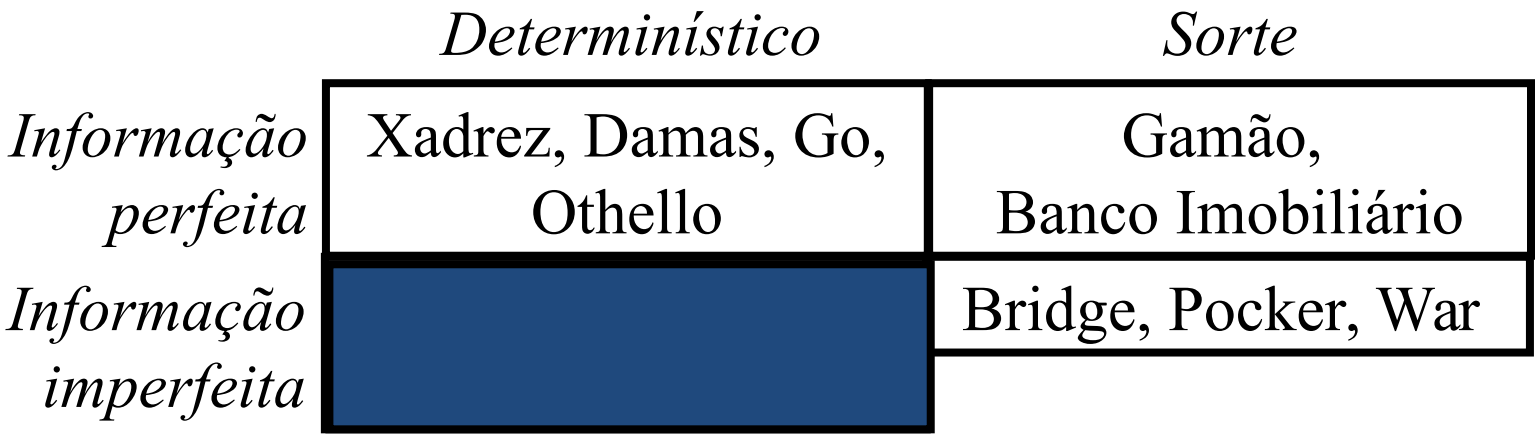
- **Por que estudar jogos?**

  Jogos oferecem ...
  - **Engajamento intelectual**
  - **Abstração**
  - **Representabilidade**
  - **Medida de desempenho**

  ⇒ Ideal para validar métodos de resolução de problemas da IA

**"Quero dizer o que disse a Herbert Simon ao telefone, há alguns minutos atrás: *estou profundamente sensibilizado pela notícia que um programa de computador tenha finalmente ultrapassado o campeão do mundo de xadrez num jogo completo*. Para todos aqueles que como nós assistiram ao nascimento do campo da IA, este era sem dúvida o problema do grande desafio. O xadrez computacional não é, certamente, toda a IA, mas como o primeiro amor, fica conosco para sempre (parece super sentimental, mas é sincero). Herbert Simon disse-me: *Bem, talvez eu não tenha sido demasiado preciso ao prever o futuro em dez anos, mas fiz tudo*

Tipos de Jogos



Considerações preliminares

- **Aplicações atrativas para métodos IA desde o início**
  - Formulação simples do problema (ações bem definidas)
  - Ambiente acessível;
  - Abstração (representação simplificada de problemas reais)
  - Sinônimo de inteligência
  - 1º algoritmo para Xadrez: Claude Shannon na década de 50
- **Problemas desafiadores**
  - Tamanho + limitação de tempo (35^{100} nós para Xadrez)
  - Incerteza devido à imprevisibilidade do oponente
  - Problema “contingencial”: agente deve agir antes de completar a busca

Jogos como problemas a serem resolvidos (1/2)

- **Principal diferença: incertezas devido a ...**
  - **Presença de um adversário**
    - Não se sabe o que o adversário fará até que ele o faça
  - **Complexidade**
    - Os jogos mais interessantes são simplesmente complexos demais para serem resolvidos por meios exaustivos

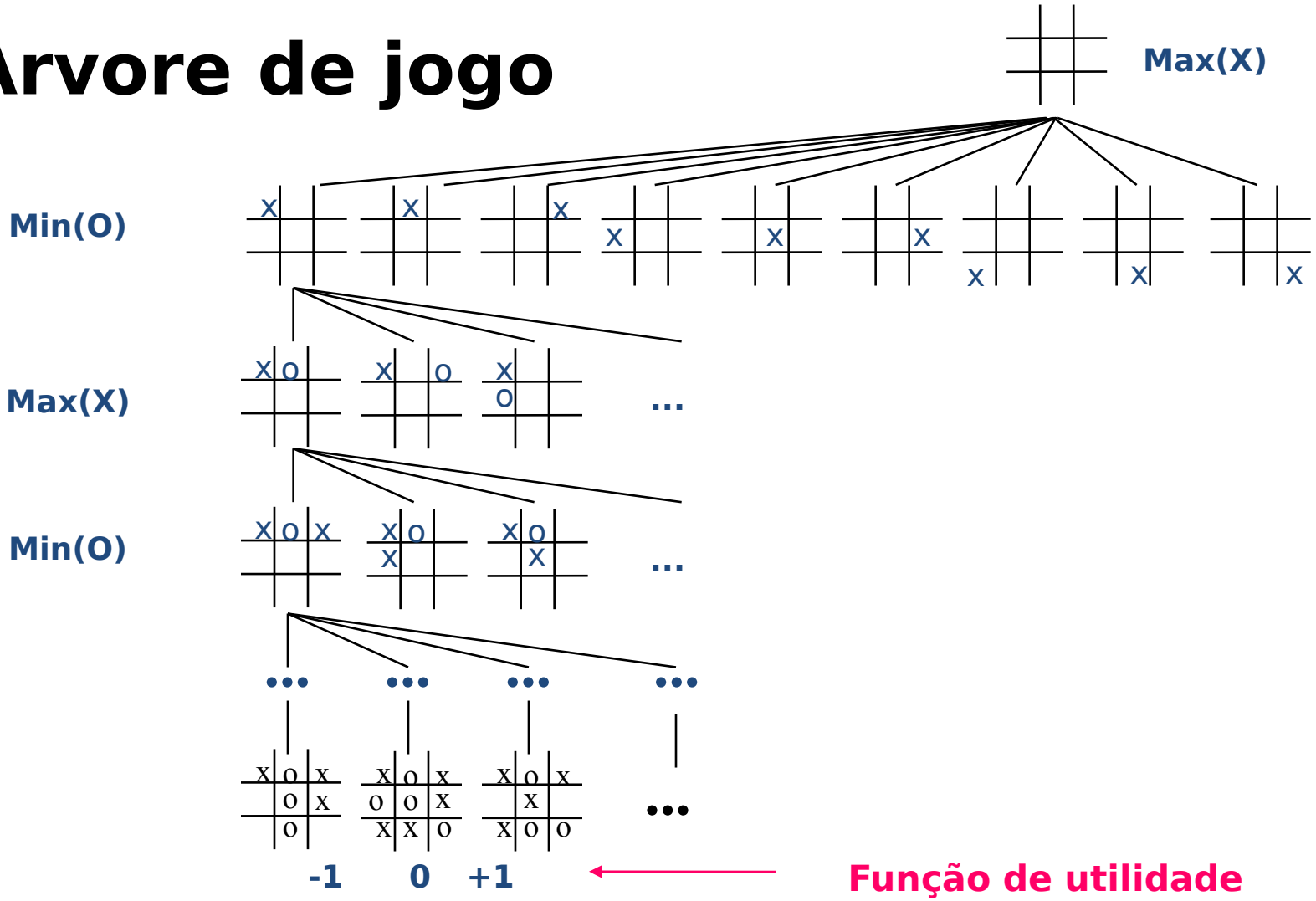
    Exemplo: Xadrez tem um grau de expansão de ordem *35*
    - Há também incerteza por não se ter todos os recursos computacionais para garantir a escolha da melhor jogada
  - **Estratégias de busca**
    - Em problemas de busca normais, busca-se uma sequência de movimentos que maximizem a qualidade da solução
    - para jogos isso não é factível, pois o adversário sempre buscará uma situação que minimize as chances de se chegar à vitória

Jogos como problemas a serem resolvidos (1/2)

- **Formulação**
  - **Estado inicial**: posições do tabuleiro + de quem é a vez
  - **Estado final**: posições em que o jogo acaba
  - **Operadores**: jogadas legais
  - **Função de utilidade**: valor numérico do resultado (pontuação)
- **Busca: algoritmo *minimax***
  - **Idéia**: maximizar a utilidade (ganho) supondo que o adversário vai tentar minimizá-la
  - Minimax faz busca cega em profundidade
  - O agente é MAX e o adversário é MIN

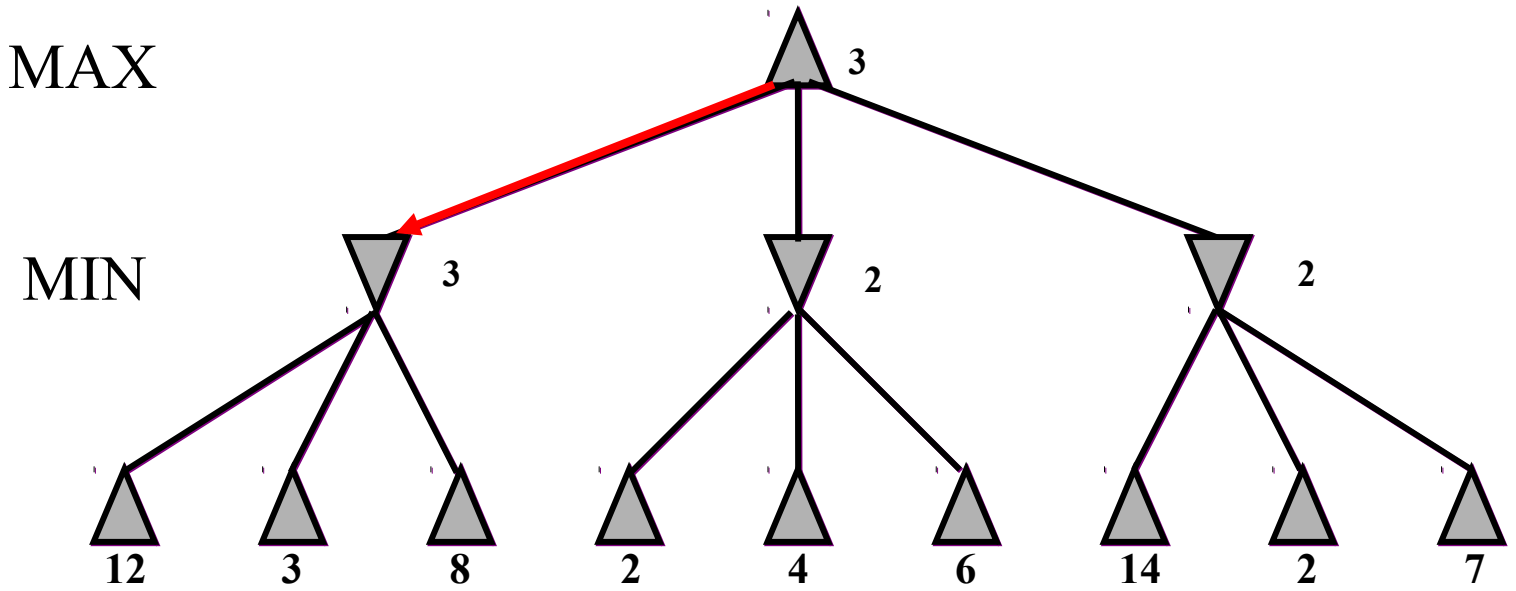
Exemplo: jogo da velha

- **Arvore de jogo**



Decisão Minimax

- **Minimax**
  - **Passos:**
    - Gera a árvore inteira até os estados terminais
    - Aplica a função de utilidade nas folhas
    - Propaga os valores subindo a árvore através do minimax
    - Determinar qual o valor que será escolhido por MAX



Algoritmo Minimax

- **Características:**
  - Ideal para jogos determinísticos e com informação perfeita
  - *Completo*, se a árvore é finita
- **Problemas**:
  - Tempo gasto é totalmente impraticável, porém o algoritmo serve como base para outros métodos mais realísticos
  - **Complexidade de tempo**: $O(b^m)$
  - **Complexidade de espaço**: $O(bm)$ (exploração em profundidade)
- **Para melhorar**:
  - Substituir a profundidade n de minimax(n) pela estimativa de minimax(n): **função de avaliação**
  - Podar a árvore onde a busca seria irrelevante: **poda alfa-beta**

Algoritmo Minimax

função DECISÃO-MINIMAX(*estado*) retorna *uma ação*

$v \leftarrow$ VALOR-MAX(*estado*)
Retornar a ação em SUCESSOR(estado) com

função VALOR-MAX(*estado*) retorna *um valor de utilidade*

Se TESTE-TERMINAL(*estado*) então retornar UTILIDADE(*estado*)
$v \leftarrow -\infty$
Para cada *s* em SUCESSORES(*estado*) faça
$v \leftarrow$ MAX(v,VALOR-MIN(*s*))

função VALOR-MIN(*estado*) retorna *um valor de utilidade*

Se TESTE-TERMINAL(*estado*) então retornar UTILIDADE(*estado*)
$v \leftarrow +\infty$
Para cada *s* em SUCESSORES(*estado*) faça
$v \leftarrow$ MIN(v,VALOR-MAX(*s*))
Retornar *v*

Função de Avaliação

- Reflete as chances de ganhar**:** baseada no valor material

Exemplo: valor de uma peça independentemente da posição das outras

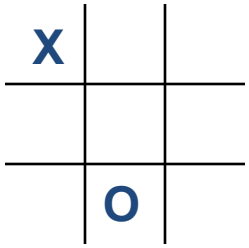
- Função Linear de Peso de propriedade do nó:

$$w_1 f_1 + w_2 f_2 + \ldots + w_n f_n$$

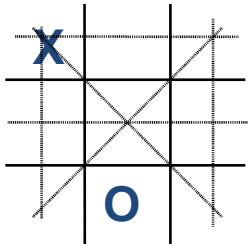
Exemplo: No Xadrez, os pesos (w) poderiam ser o tipo de pedra (Peão - 1, ...,
Rainha - 9) e os valores da função f poderiam ser o número de cada peça no tabuleiro

- Escolha das propriedades relevantes ainda não pode ser realizada
- **Escolha crucial**: compromisso entre precisão e eficiência

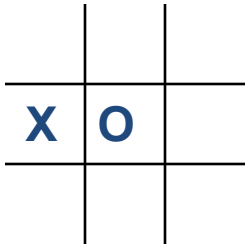
Função de avaliação: ex. jogo da velha



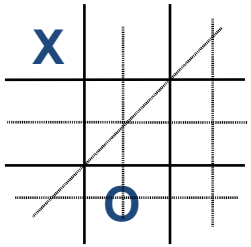
$h = 6 - 5 = 1$



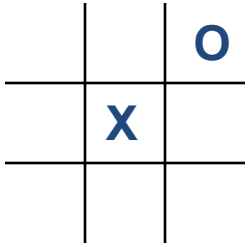
X tem 6 possibilidades



$h = 4 - 6 = -2$



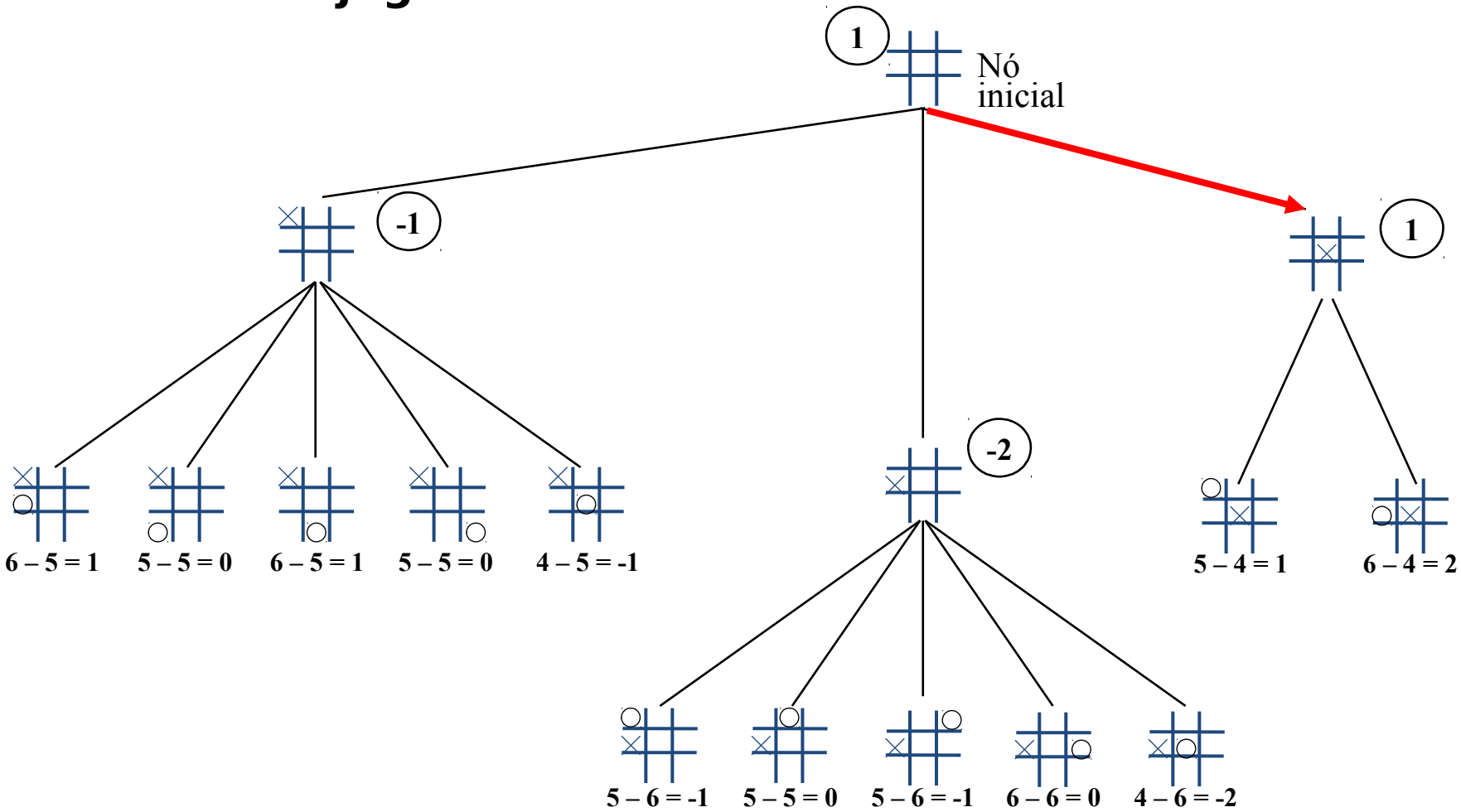
O tem 5 possibilidades



$h = 5 - 4 = 1$

Função de avaliação: ex. jogo da velha

- **Árvore de jogo**



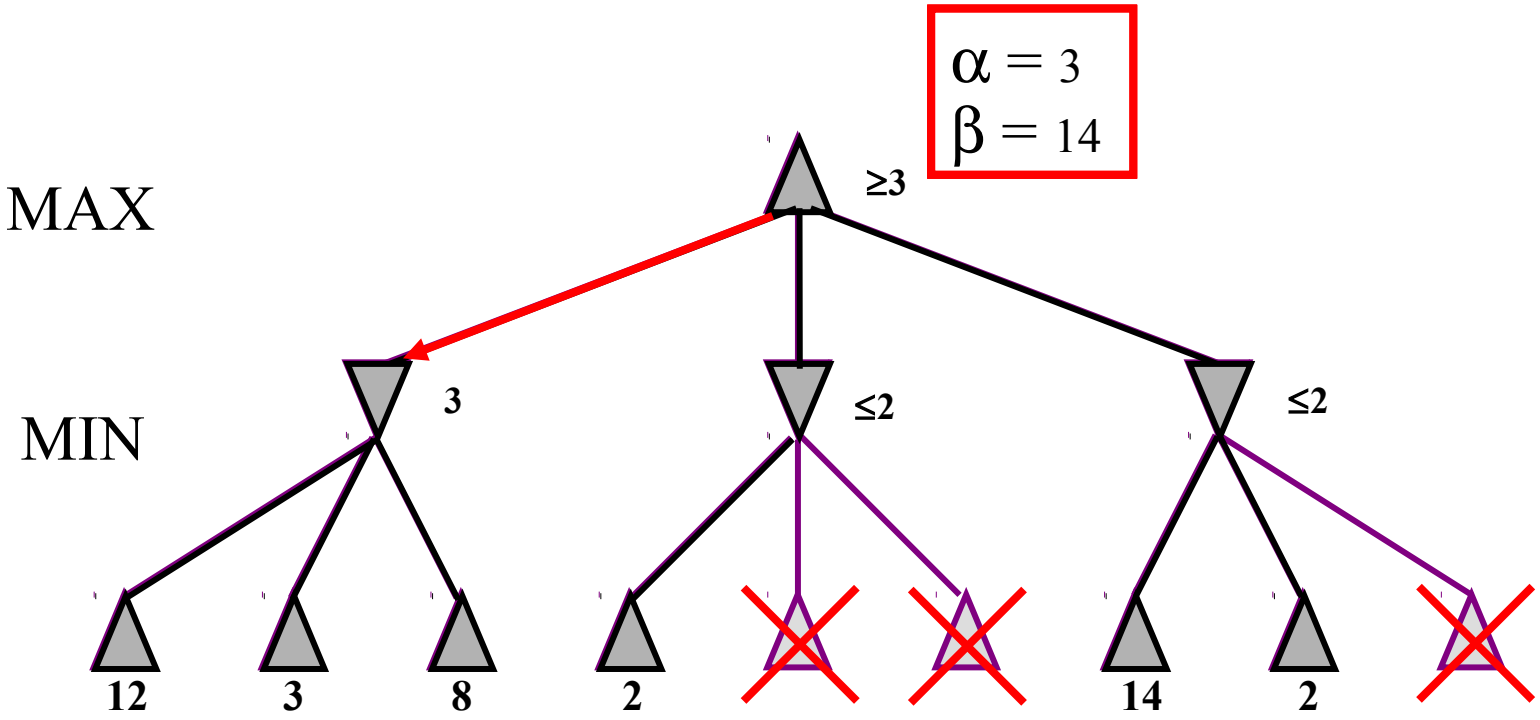
Função de avaliação

- **Quando aplicar a função de avaliação?**
  - Definir uma profundidade máxima ou iterativa não funciona devido à incerteza inerente ao problema
- **Solução**: **Procura Tranquila** (*Quiescence search)*
  - **Idéia**: evitar avaliação em situações a partir das quais pode haver mudanças bruscas
  - No caso do jogo da velha, toda posição é tranquila mas no xadrez não.... (ex. um peça de xadrez a ser comida)
  - **Algoritmo**: Se a situação (nó) é “tranquila”, então aplica a função de avaliação, senão busca até encontrar uma situação “tranqüila”

Poda alfa-beta

- **Poda alfa-beta (*alpha-beta pruning*)**
  - **Função**: Não expandir desnecessariamente nós durante o minimax
  - **Idéia**: não vale a pena piorar, se já achou algo melhor
  - Mantém 2 parâmetros:
    - α - melhor escolha (maior valor) para MAX
    - β - melhor escolha (menor valor) para MIN
  - **Teste de expansão**:
    - α não pode diminuir (não pode ser menor que um ancestral)
    - β não pode aumentar (não pode ser maior que um ancestral)

Poda alfa-beta



Algoritmo - poda alfa-beta

função BUSCA-ALFA-BETA(*estado*) retorna *uma ação*

$v \leftarrow$ VALOR-MAX(*estado*,-∞,+∞)
Retornar a ação em SUCESSOR(*estado*) com valor

função VALOR-MAX(*estado*,α,β) retorna *um valor de utilidade*

Se TESTE-TERMINAL(*estado*) então retornar UTILIDADE(*estado*)
$v \leftarrow$ -∞
Para cada *s* em SUCESSORES(*estado*) faça
$v \leftarrow$ MAX(v,VALOR-MIN(*s*,α,β))
Se v ≥ β então retornar *v*

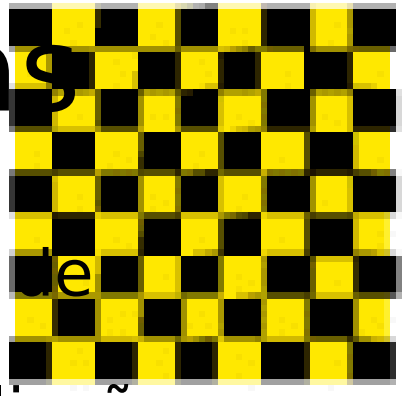
função VALOR-MIN(*estado*,α,β) retorna *um valor de utilidade*

Se TESTE-TERMINAL(*estado*) então retornar UTILIDADE(*estado*)
$v \leftarrow$ +∞
Para cada *s* em SUCESSORES(*estado*) faça
$v \leftarrow$ MIN(v,VALOR-MAX(*s*,α,β))
Se v ≤ α então retornar *v*
β ← MIN(β,v)
Retornar *v*

Alguns jogos: Xadre

- Apesar de receber durante muito tempo bastante atenção, o progresso rumo a um nível medíocre foi inicialmente muito lento
- **1970**: Primeiro programa a ganhar o ACM North American Computer Chess Championship, utilizando **busca alfa-beta** incrementada de um livro de **aberturas** e de algoritmos infalíveis de **finalizações de jogos**
- **1982**: Belle, primeiro **computador concebido exclusivamente para xadrez**, pode analisar *alguns milhões* de posições a cada jogada
- **1985**: Hitech, colocado entre os 800 melhores jogadores do mundo, pode analisar *10 milhões* de posições a cada jogada.
- **1993**: Deep Thought 2, um dos 100 melhores jogadores do mundo, pode analisar *500 milhões* de posições a cada jogada, chegando a uma profundidade de 11.
- **1997**: Deep Blue (IBM) ganha do campeão mundial Gary Kasparov, podendo analisar 1 bilhão de posições a cada jogada, numa velocidade de 200 milhões de posições por segundo, chegando a uma profundidade 14

Alguns jogos: Damas

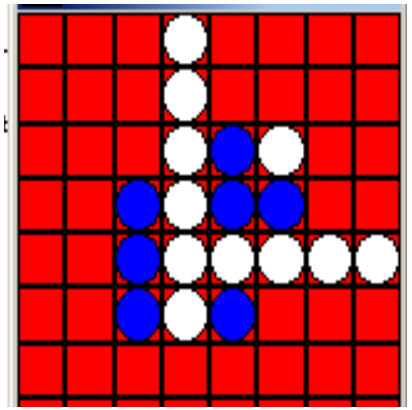


- **1952**: Samuel desenvolve um programa de damas
  que **aprende** sua própria função de avaliação jogando consigo mesmo
- **1992**: Chinook (J. Schaeffer) ganha o U.S. Open

  No campeonato mundial, Chinook ganha de Marion Tinsley, campeão mundial por 40 anos
- Foi utilizada:
  - Busca alfa-beta
  - Um banco de dados de finalizações de jogos (com 8 ou menos peças no tabuleiro) pré-computadas, num total de 444 bilhões de posições.

Alguns jogos: Othelo

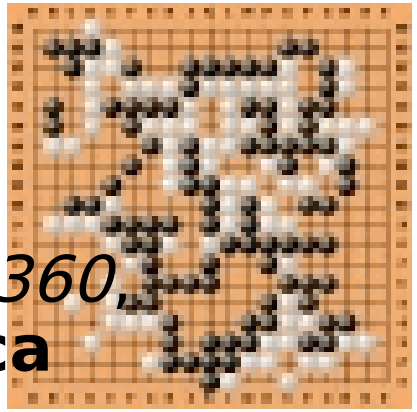
- Campeões humanos se recusam a competir com computadores, por terem resultados muito bons



Alguns jogos: Gamão

- A inclusão da incerteza dos dados torna a busca um luxo caro
- **1980**: BKG, usando bastante sorte ganha do campeão mundial humano
- **1992**: Tesauro combina o método de **aprendizagem por reforço** de Samuel **com redes neurais** para desenvoilver uma nova função de avaliação, resultando num programa colocado entre os 3 maiores jogadores do mundo.

Alguns jogos: Go



- O fator de ramificação se aproxima de *360*, **impraticável para métodos de busca** regulares
- Prêmio de $2,000,000 para o primeiro programa de computador que ganhar de um jogador humano do topo do ranking mundial
- Deve se beneficiar de busca intensiva utilizando métodos de raciocínio sofisticado, como bases de conhecimento sobre padrões para sugerir movimentos plausíveis
- Campeões humanos se recusam a competir com computadores, por terem resultados muito ruins

Considerações finais

- Jogos ilustram vários pontos importantes da IA:
  - A perfeição é inatingível; é preciso aproximá-la
  - É uma boa idéia para pensar a respeito de como pensar a respeito de algum problema
  - Incerteza restringe a obtenção de valores para estados

Segundo S. Russell, os jogos estão para a IA assim como as corridas estão para os projetos de automóveis.