Inteligência Artificial

Busca com Adversários
(Jogos)

Sumário

- Jogos
  - Jogos vs. Problemas de Busca
  - Tipos de Jogos
- Decisões Ótimas em Jogos
  - Estratégias ótimas
  - Algoritmo Minimax
  - Alpha-Beta Pruning
- Decisões Imperfeitas em Tempo Real
  - Funções de Avaliação
  - Busca Interrompida
- Jogos que Envolvem Sorte

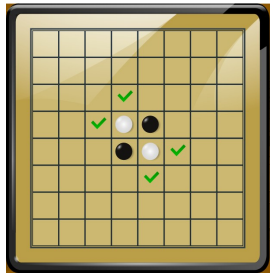
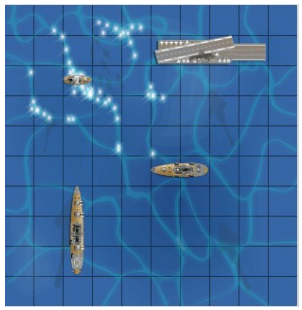
Busca com Adversários

Jogos

Jogos vs. Problemas de Busca

- Jogos, geralmente, são mais difíceis de resolver
  - Deve-se considerar as contingências para as ações do oponente → Estratégia!
  - Muitas vezes a busca de todo o espaço de estados é inviável (tempo restrito)
    - É necessário aproximar
    - Algoritmos pouco eficientes perdem!
    - Xadrez
      - Espaço de estados: 10^{40}
      - Fator de ramificação médio: 35
      - Número médio de movimentos por jogador: 50
      - Número médio de nós na árvore de busca: $\mathbf{35^{100}}$
        - O número estimado de átomos no universo visível é $\mathbf{10^{80}}$

Tipos de Jogos

	Determinísticos	Estocásticos (Sorte)
Totalmente Observáveis (Informação Perfeita)	 	
Parcialmente Observáveis (Informação Imperfeita)		

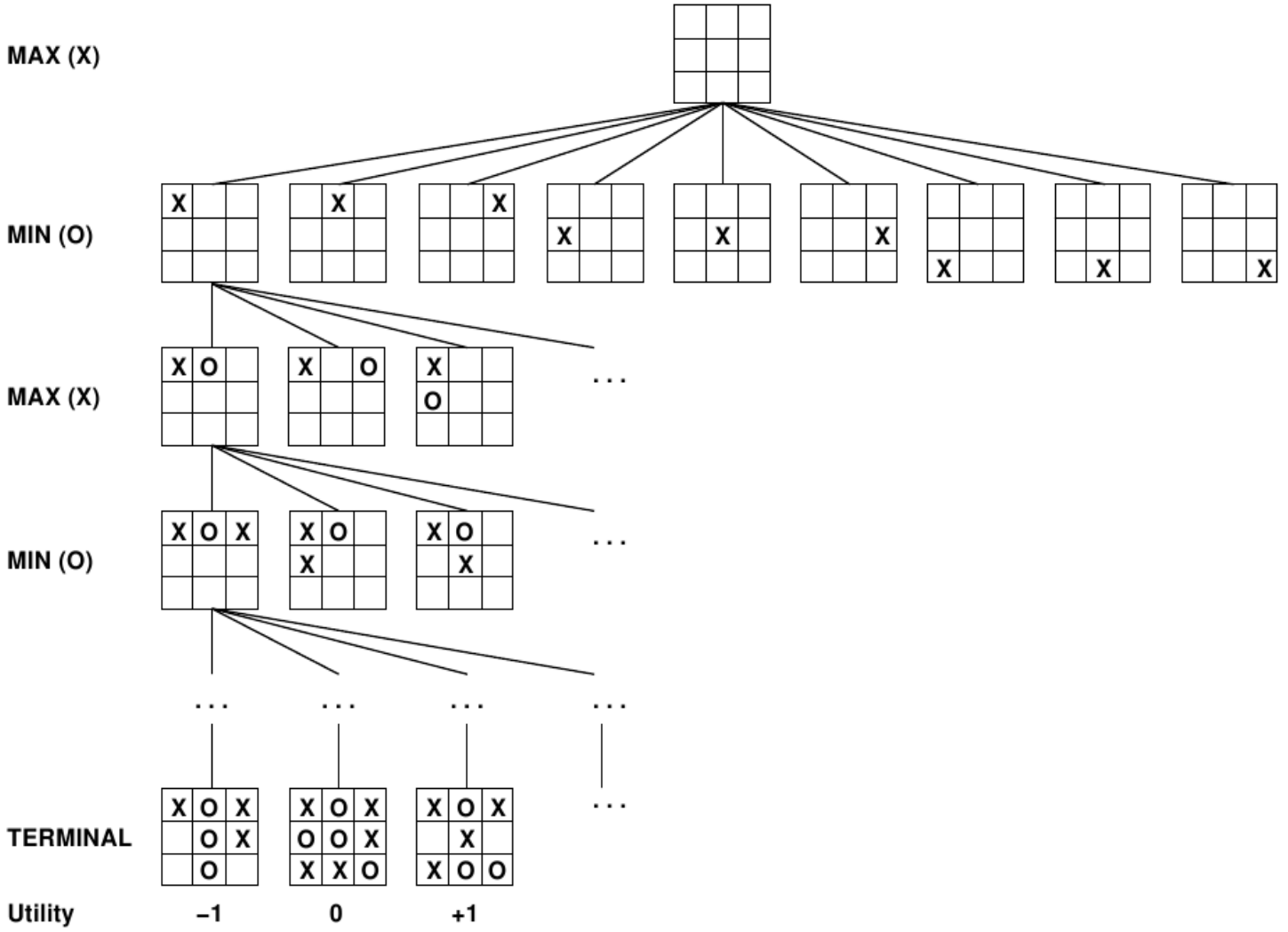
Busca com Adversários

Decisões Ótimas em Jogos

Formulação do Problema

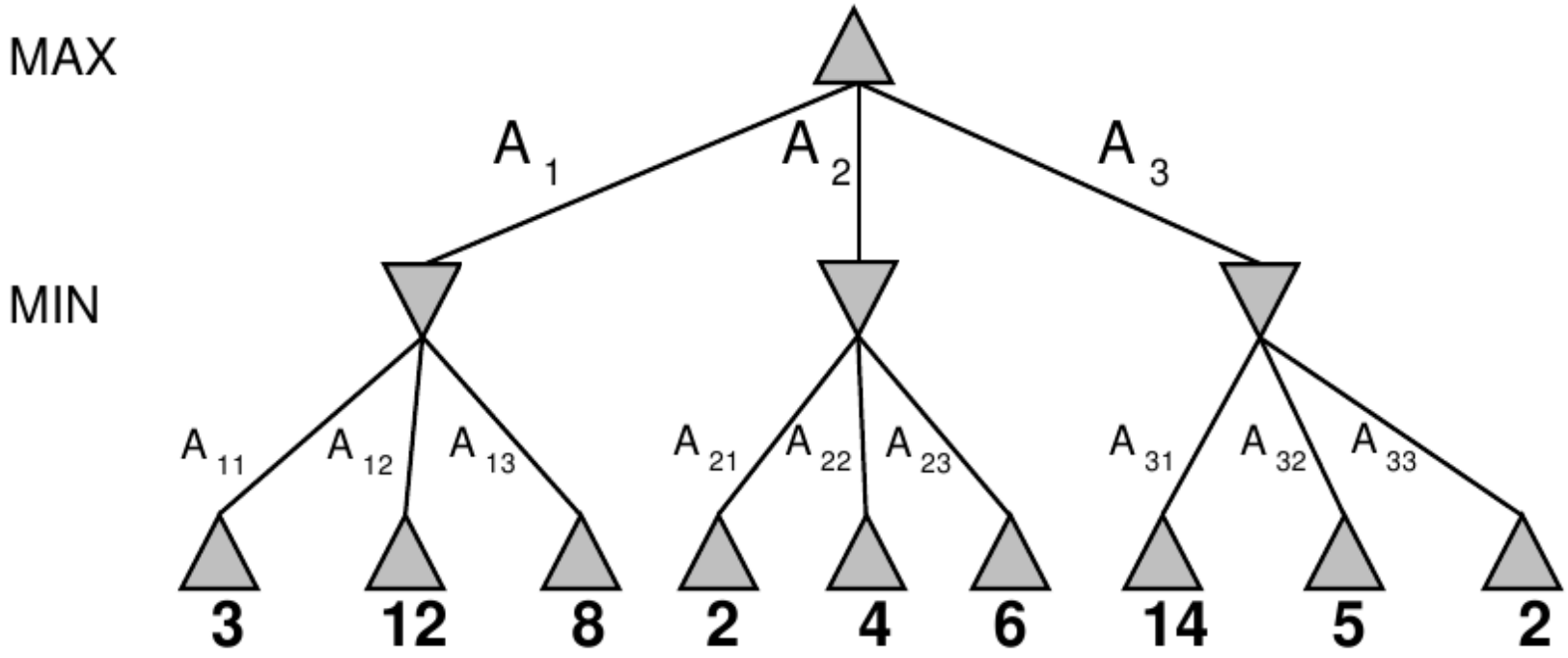
- Estado inicial
  - Posição inicial das peças
  - Definição do jogador a começar
- Função sucessor
  - Retorna uma lista de pares [movimento, *estado*], cada um indicando um movimento válido e o estado resultante
- Teste de término
  - Determina quando o jogo termina. Estados onde o jogo termina são chamados *Estados Terminais*
- Função de Utilidade (ou *Função objetivo*)
  - Associa um valor numérico a cada estado
    - Xadrez: Jogo de soma zero: (vence, perde, empata)

Árvore do Jogo da Velha (Tic-tac-toe)



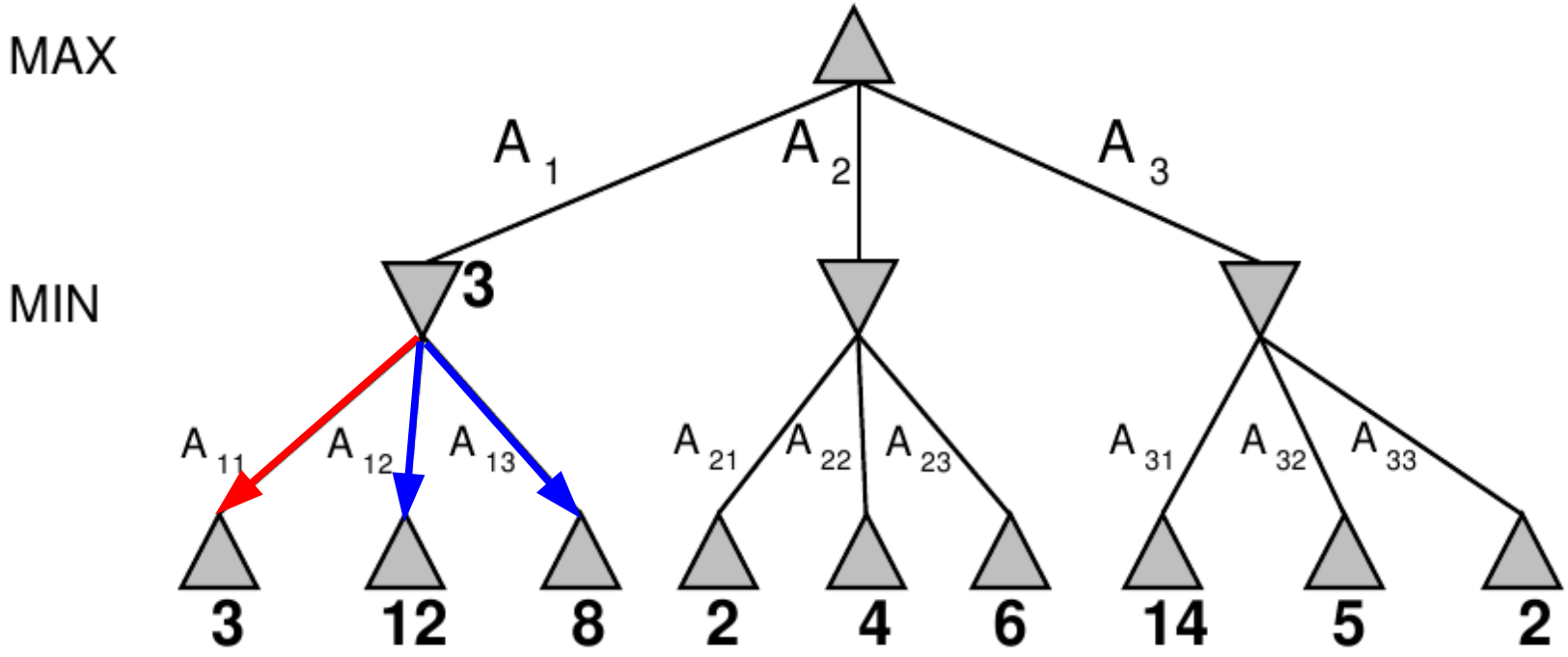
Jogo mais Simples

- Jogo com um movimento de profundidade
  - Dois meio movimentos (ou *plies*)
    - *Two-ply game*



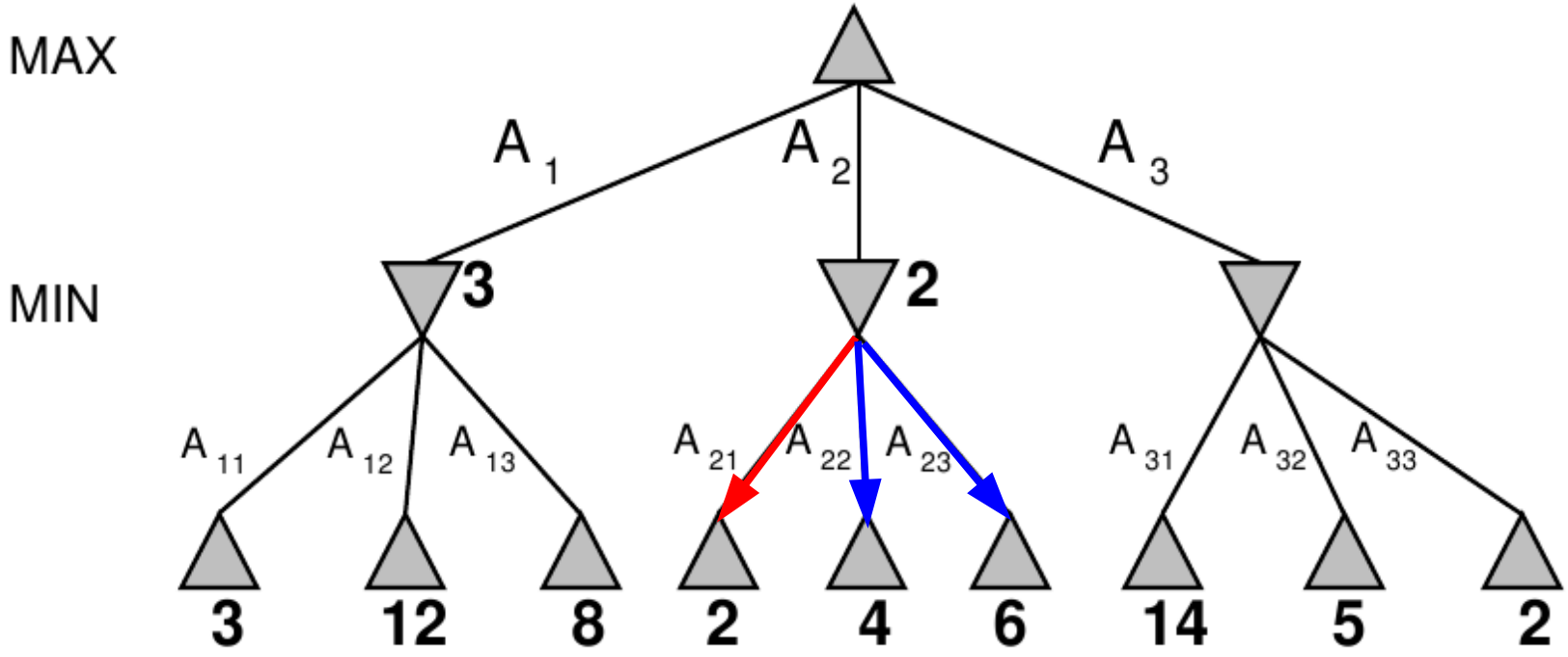
Jogo mais Simples

- Jogo com um movimento de profundidade
  - Dois meio movimentos (ou *plies*)
    - *Two-ply game*



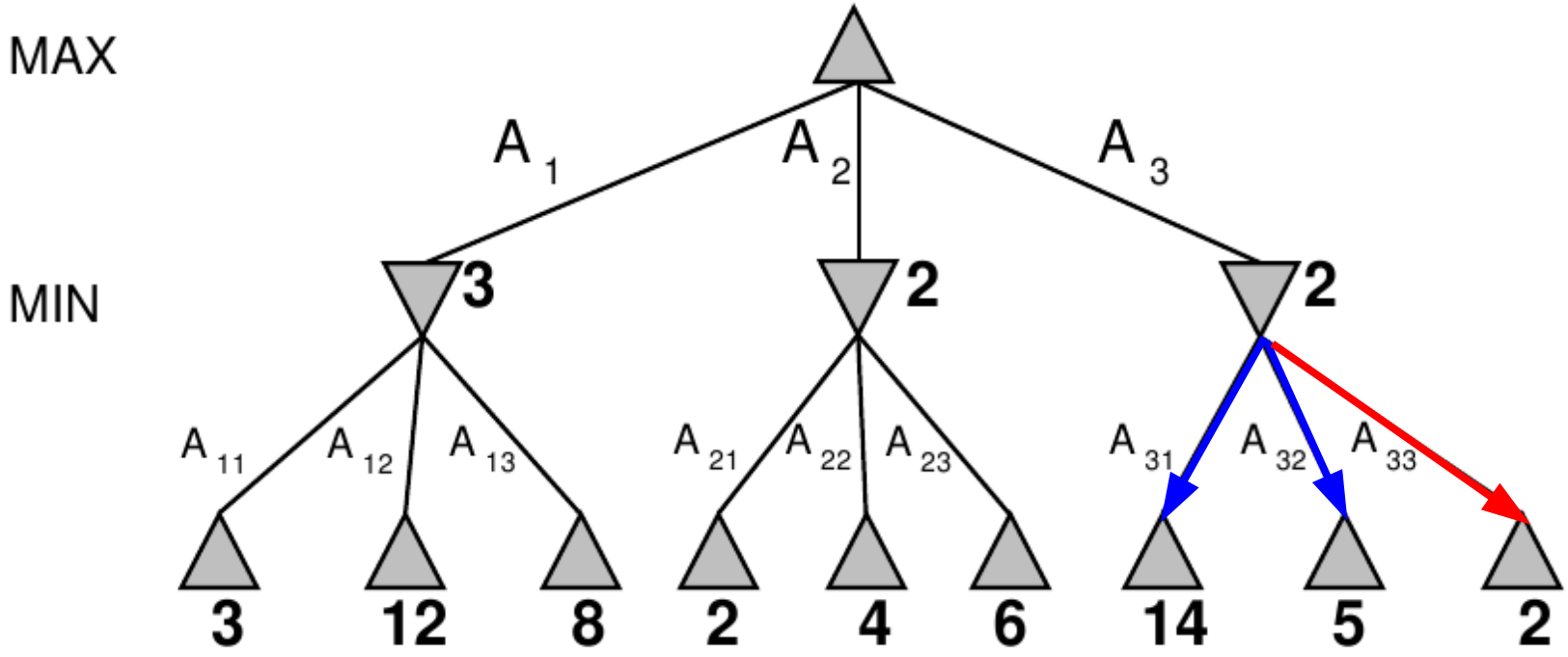
Jogo mais Simples

- Jogo com um movimento de profundidade
  - Dois meio movimentos (ou *plies*)
    - *Two-ply game*



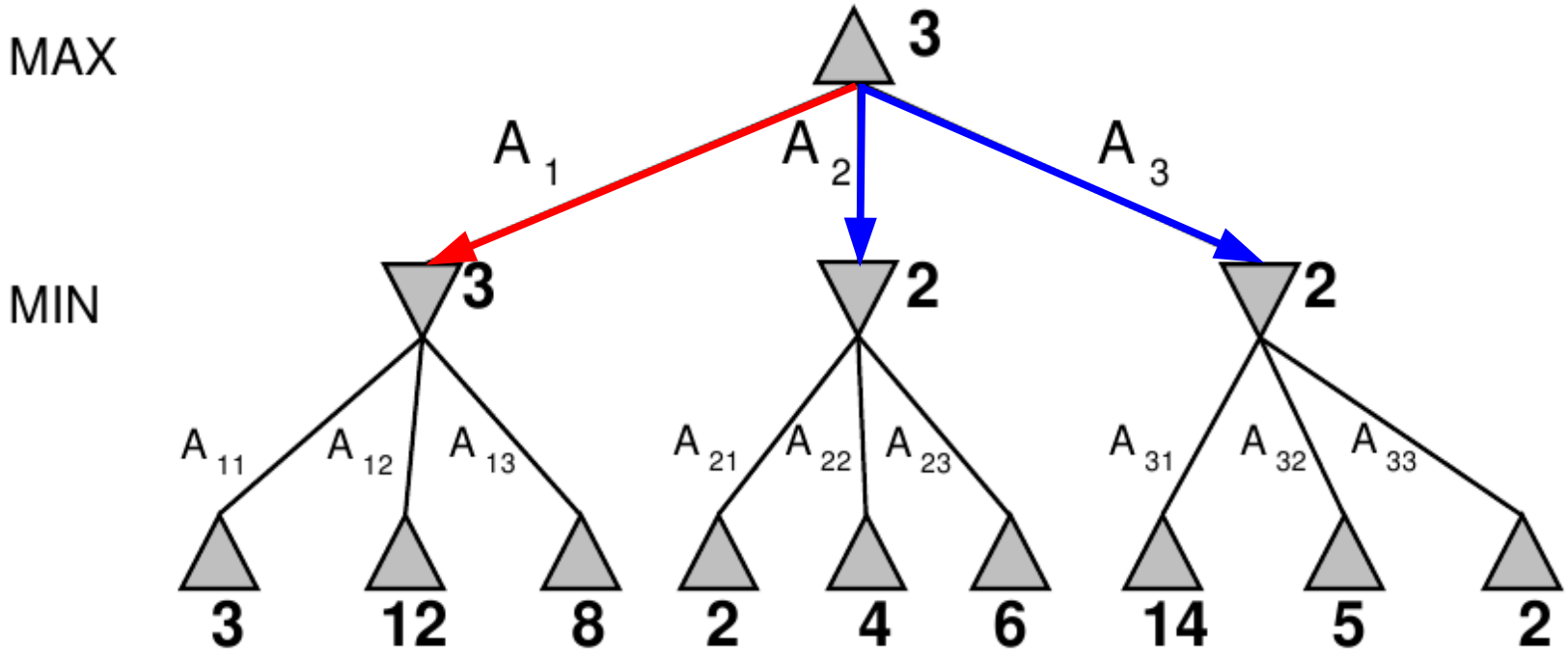
Jogo mais Simples

- Jogo com um movimento de profundidade
  - Dois meio movimentos (ou *plies*)
    - *Two-ply game*



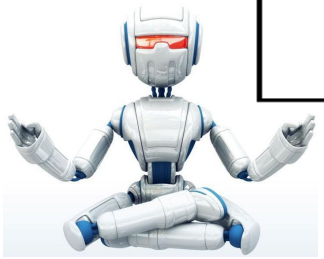
Jogo mais Simples

- Jogo com um movimento de profundidade
  - Dois meio movimentos (ou *plies*)
    - *Two-ply game*



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 ← −∞
    for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
    return v

---

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
    return v
```



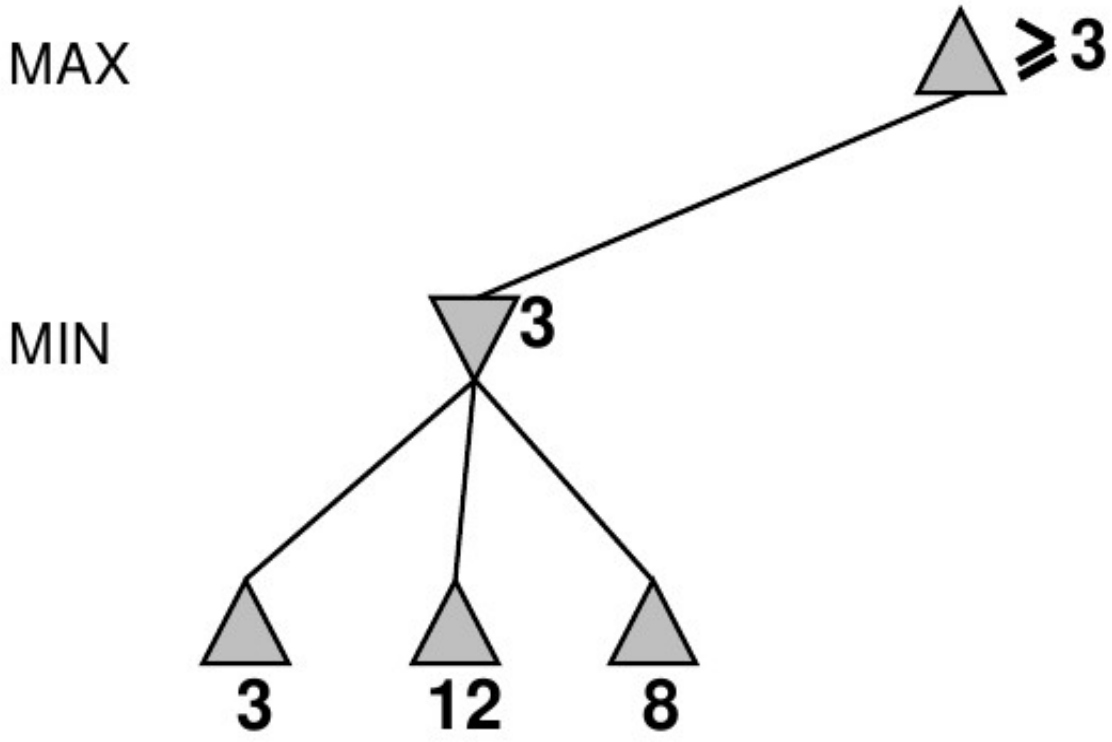
Análise do Algoritmo

- Busca em Profundidade
  - Busca completa (se a árvore for finita)
  - Optimalidade
    - Ótimo contra oponentes ótimos
      - E contra oponentes não ótimos?
  - Complexidade no Tempo para pior caso $O(b^m)$
    - $b \rightarrow$ fator de ramificação (movimentos legais)
    - $m \rightarrow$ profundidade da árvore
    - Impraticável para a maior parte dos jogos reais!
  - Como reduzir o espaço de busca?

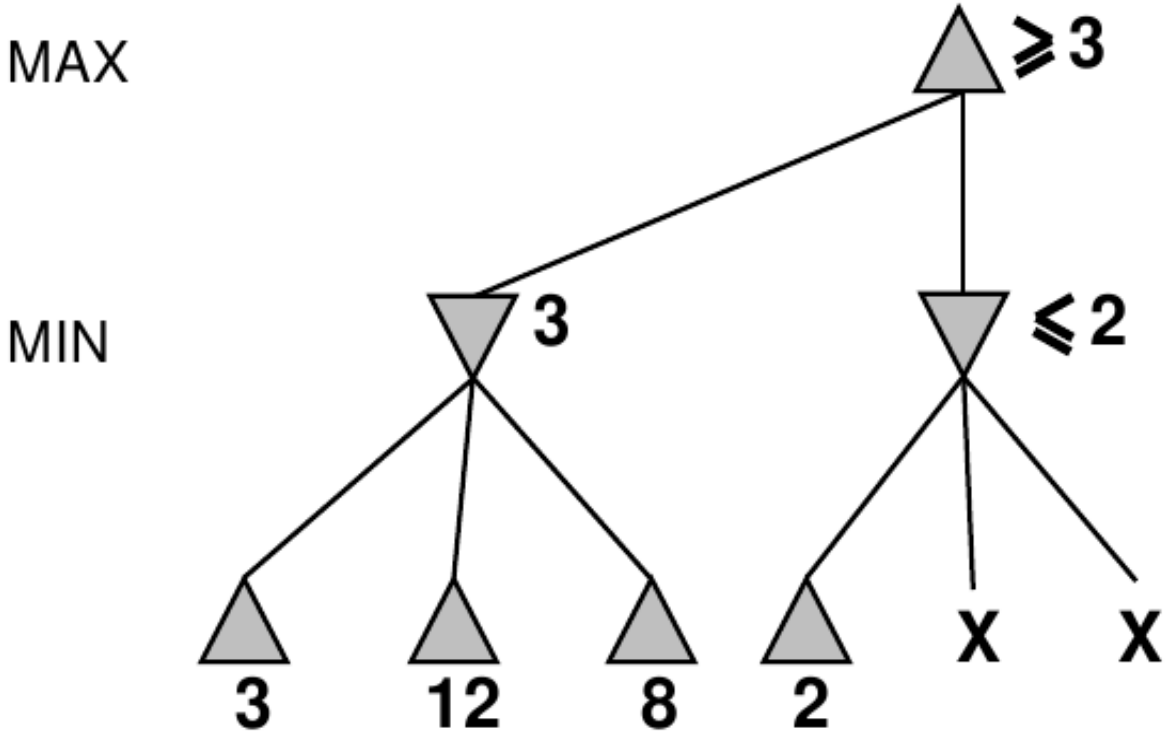
Alpha-Beta Pruning

- O algoritmo Minimax examina um número de estados que varia *exponencialmente* com o número de movimentos
  - Não é possível retirar o termo exponencial e manter a decisão ótima
  - Mas pode-se reduzir em muito o espaço de busca
    - Como? Podando a árvore!

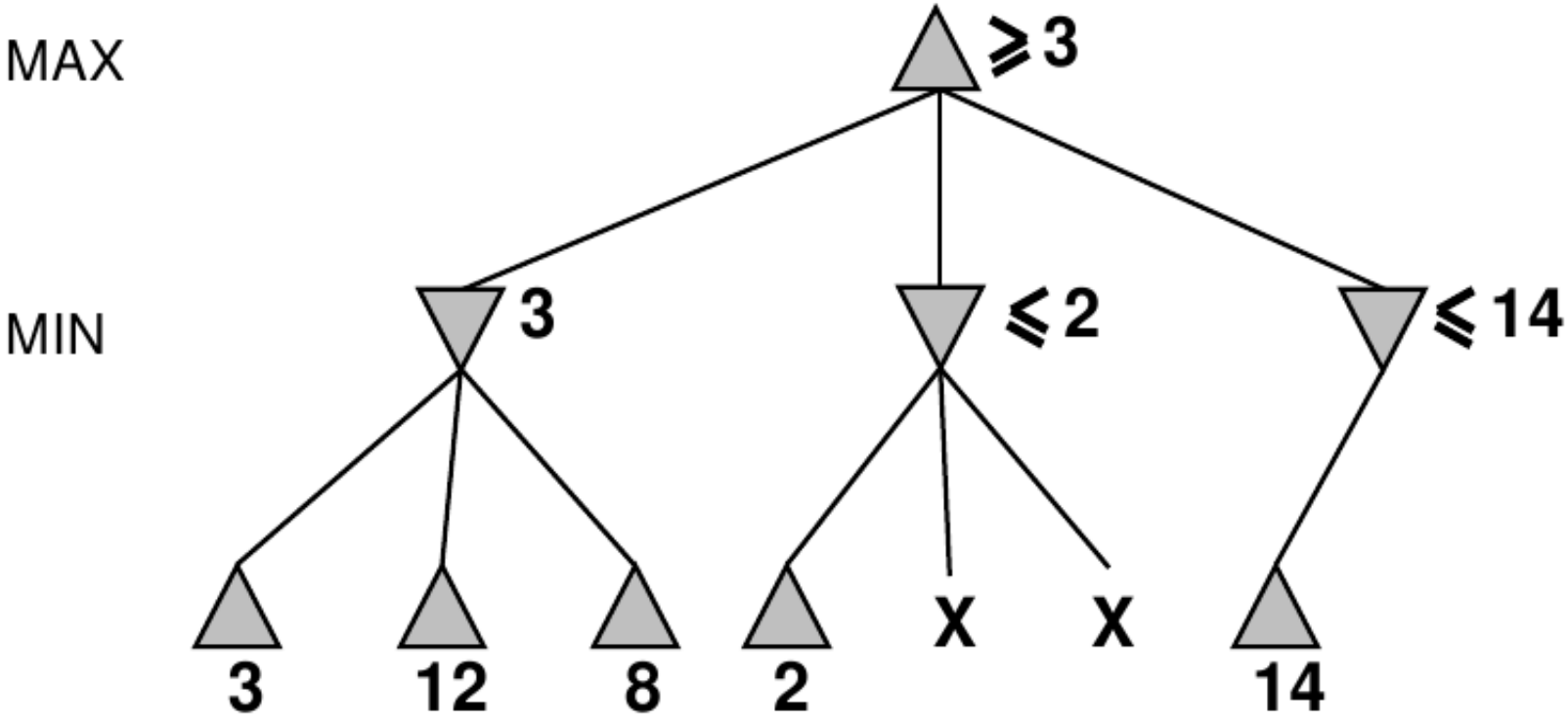
Alpha-Beta Pruning



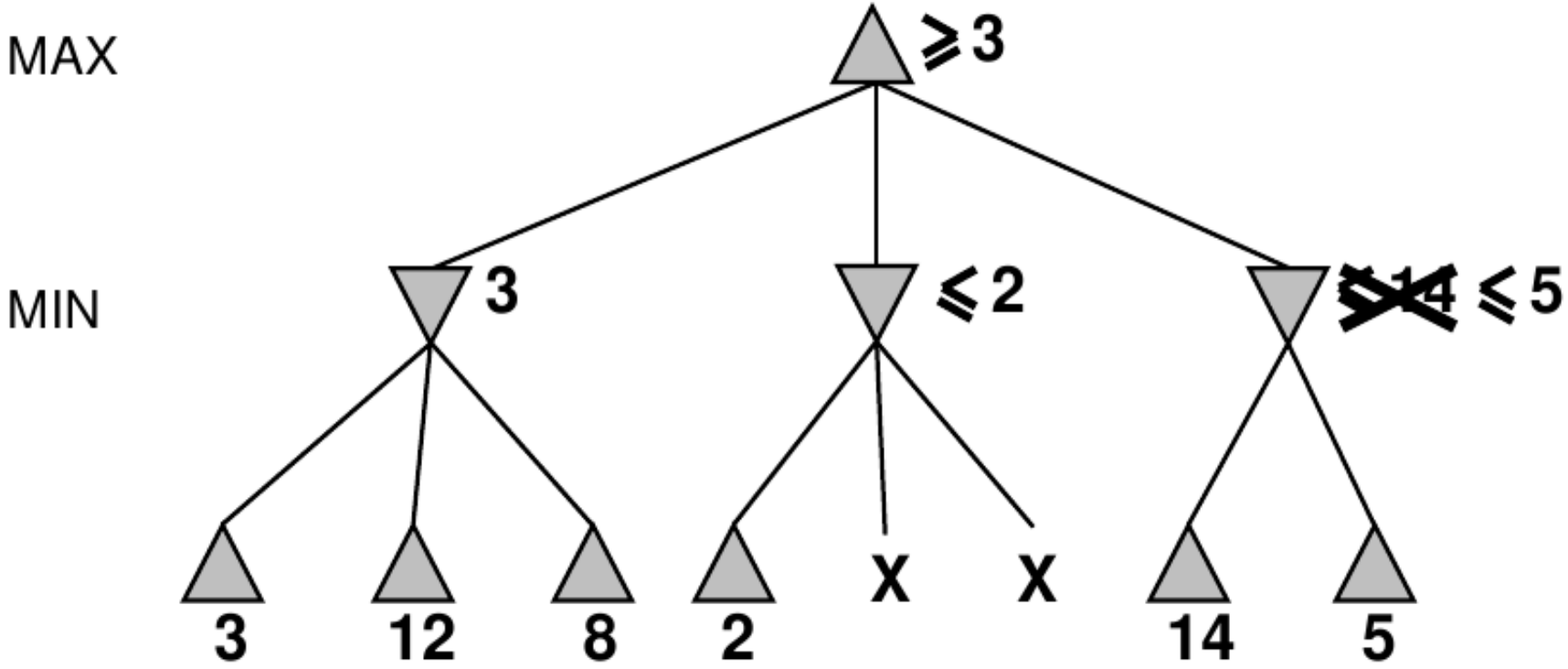
Alpha-Beta Pruning



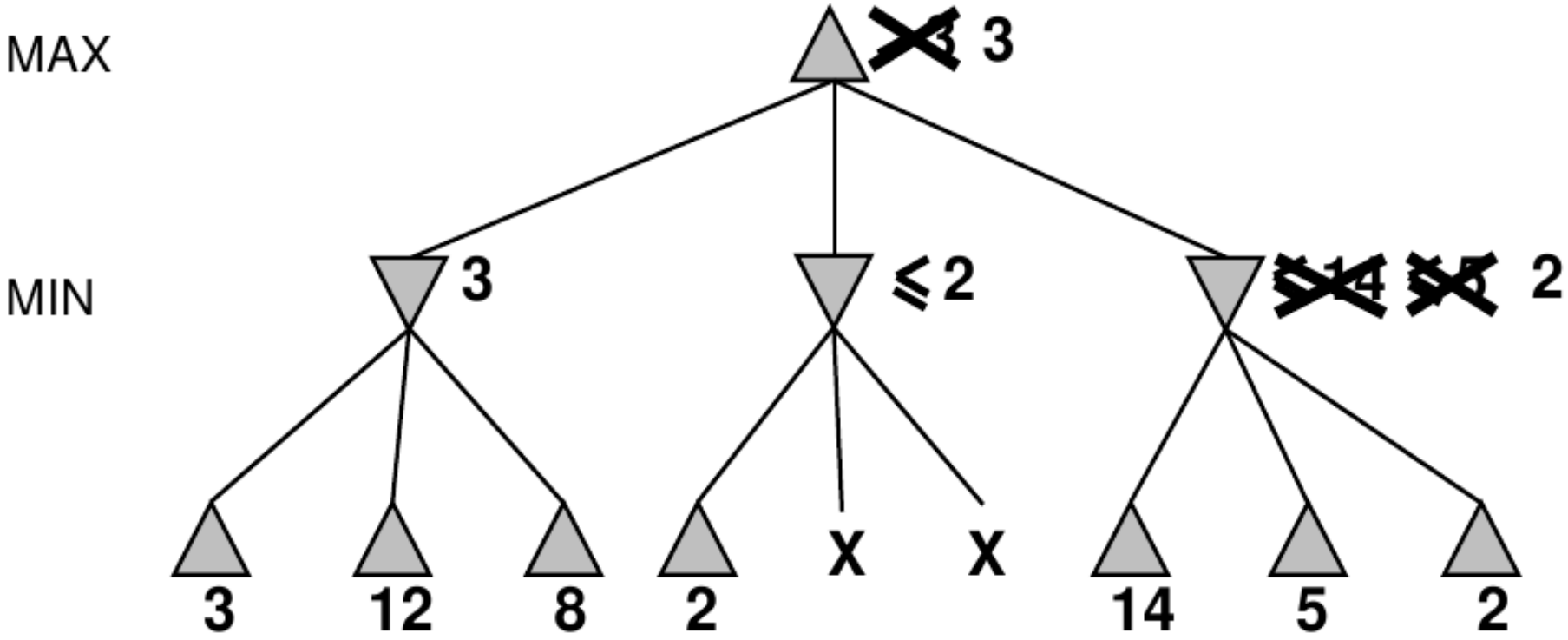
Alpha-Beta Pruning



Alpha-Beta Pruning



Alpha-Beta Pruning



Algoritmo Minimax com Alpha-Beta Pruning

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

---

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 ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v

---

function MIN-VALUE(state, α, β) returns a utility value
    same as MAX-VALUE but with roles of α, β reversed
```

Análise do Algoritmo

- O algoritmo reduz o espaço de busca sem alterar o resultado do minimax!
  - Se houver uma ordenação boa das jogadas (melhores testadas primeiro), a redução ainda é mais efetiva
    - Considerando ordenação perfeita, o espaço de busca passa a ser $O(b^{m/2})$
      - Dobra a profundidade solucionável!
      - Ordenação nunca é perfeita (se fosse, poderia ser utilizado um algoritmo de ordenação para buscar o resultado)
      - Infelizmente 35^{50} ainda é impossível de se solucionar em tempo hábil

Melhorias possíveis

- Adicionar aprendizado à busca
  - O agente pode aprender quais ações geralmente levam a resultados melhores e testá-las primeiro
    - Exemplo: Colocar uma cruz (X) no meio do tabuleiro geralmente leva a resultados melhores
- Manter uma tabela de Hash com estados já testados
  - Técnica chamada de Tabela de Transposição
    - Programação dinâmica
    - Se muitos nós forem gerados por segundo, torna-se inviável (qual nó armazenar?)

Busca com Adversários

Decisões Imperfeitas em Tempo Real

Decisões Imperfeitas

- Mesmo utilizando *alpha-beta pruning* o espaço de busca pode continuar imprati-cável
  - Cada movimento pode gastar muito tempo para ser resolvido
- Alternativa
  - Usar uma função heurística
    - A *função de utilidade* é trocada por uma *função de avaliação* (heurística)
    - O *teste de término* é substituído por um *teste de interrupção*

Funções de Avaliação

- Estima o valor de utilidade de um jogo a partir de um determinado estado
- Requisitos
  - Deve ordenar os estados terminais na mesma ordem da função de utilidade
    - Senão, um agente chegará a resultados sub-ótimos mesmo se a busca alcançar estados terminais
  - Seu cálculo não deve ser demorado
  - O seu valor para um determinado estado deve ser fortemente ligado às chances de vitória
    - Apesar de xadrez não ser um jogo de sorte, a informação incompleta requer que o agente “advinhe” o melhor caminho

Funções de Avaliação Xadrez

- Funções de avaliação calculam a partir de características do estado
  - e.g. Número de peões de cada jogador
- A função divide os estados em classes de equivalência
  - Estados nestas classes possuem as mesmas características
  - A função não sabe qual estado, mas sabe qual classe de equivalência
  - A partir da classe de equivalência, pode-se determinar a chance de sucesso
    - 72% dos estados da categoria → vitória (+1)
    - 20% → derrota (-1)
    - 8% → impate (0)
    - Valor esperado ponderado: (0.72) + (-0.20) + (0) = 0.52
  - Ainda assim, muito complexo (precisa de muita experiência sobre as classes)

Funções de Avaliação Xadrez

- Geralmente utiliza-se uma combinação de valores das características em uma função linear
  - Peão=1;Bispo,Cavalo=3;torre=5;rainha=9.
  - Boa estrutura de peões = ½; Rei a salvo = ½
  - Soma-se todas as características ponderadas na função de avaliação

$$EVAL(s) = \sum_{i=0}^{n} w_i f_i(s)$$

- Em programas atuais, a função *não é linear*

Busca Interrompida

- Em vez de testar se o estado é terminal para retornar, verifica se a profundidade limite foi atingida
  - Se o limite foi atingido, retorna o valor da função de avaliação do nó atual
    - Melhor ainda, pode-se utilizar busca com profundidade iterativa (busca às cegas) e, quando o tempo acabar, retornar o melhor resultado
  - Supondo que se tenha 100 segundos
    - Exploração de 10^4 nós/segundo
      - 10^6 nós por movimentação (aprox 35^4)
        - *Alpha-Beta Pruning* atingindo profundidade 8!
          - Para "bater" um campeão mundial geralmente requer-se profundidade 14

Melhorias Busca Interrompida

- Evitar cálculos da função de avaliação em posições não-quiescentes
  - Posições onde o valor da função de avaliação muda drasticamente
    - Exemplo: Estados no xadrez onde uma captura é iminente
- Horizonte de Efeitos
  - Tentar evitar um movimento do oponente que causa danos sérios mas é inevitável
    - Saber que é inevitável está fora da profundidade máxima calculável
  - Solução: Extensão Singular
    - Se uma alternativa for “claramente” melhor que todas as outras, explorar somente ela
      - Fator de ramificação = 1 no nó que o caso ocorrer

Combinando as Técnicas

- Xadrez
  - Implementação da função de Avaliação
  - Teste de Interrupção com busca quiescente
  - Tabela de Transposição grande
  - Programa que avalia 1 milhão de nós/segundo
    - 3 minutos por movimento = 200 milhões de nós
- Minimax = 5 *plies*
- *Alpha-Beta pruning* = 10 *plies*
- Humano
  - Médio: 6-8 *plies*
  - Experiente: 10 *plies*
  - Campeão: 14 *plies*

Busca com Adversários

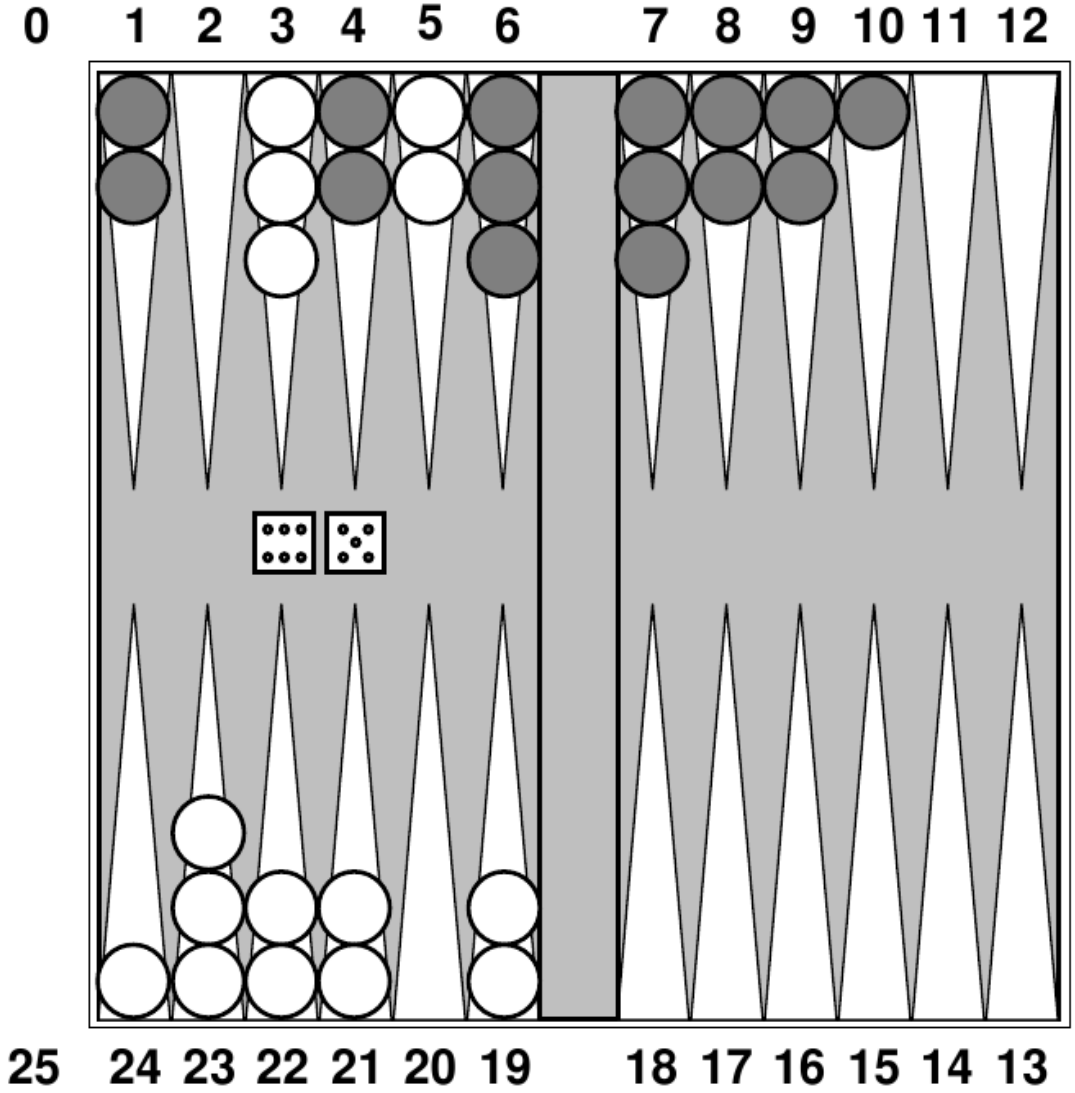
Jogos que Envolvem Sorte

Jogos que Envolvem Sorte

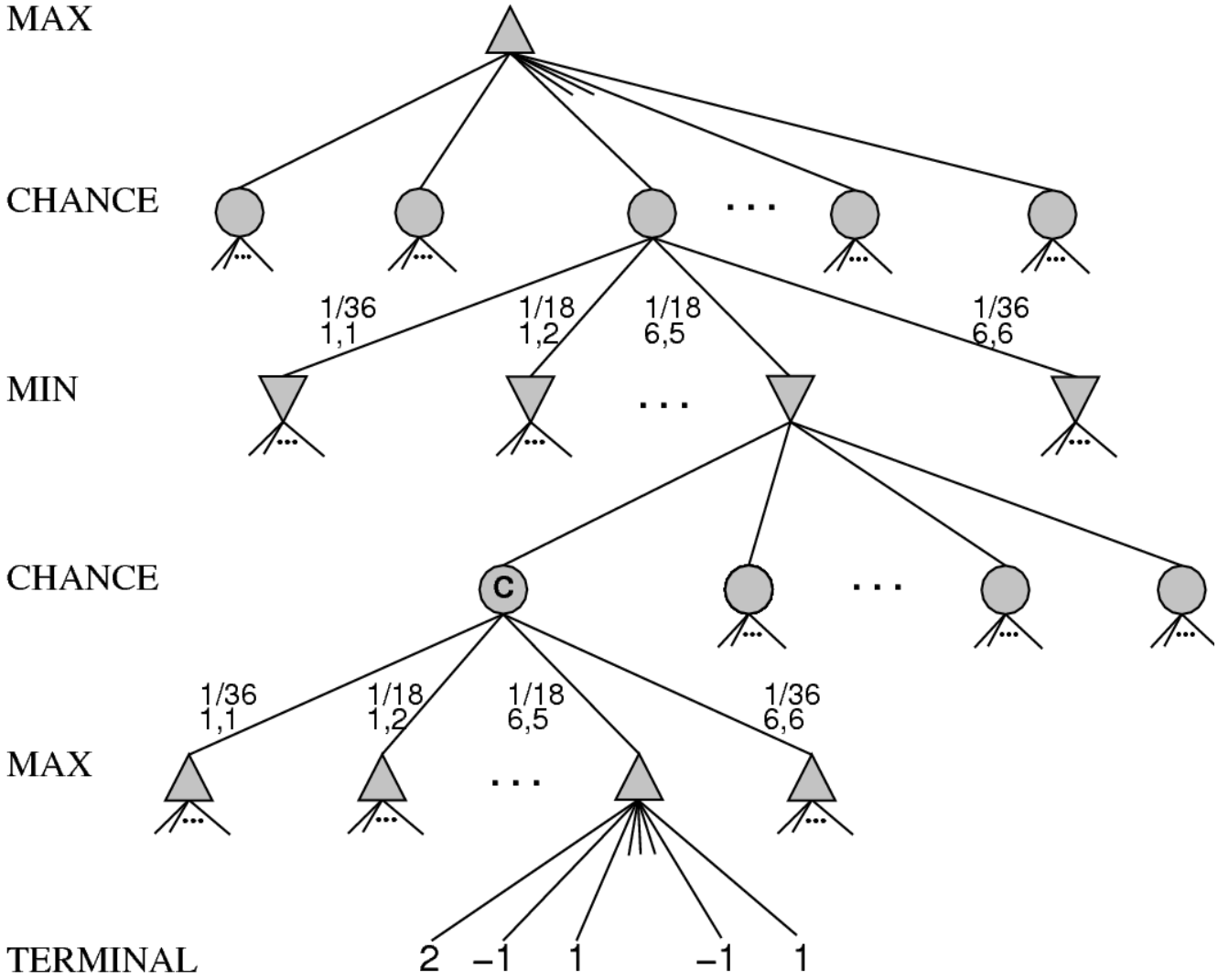
- Muitos jogos envolvem sorte
  - Buscam imitar o comportamento da vida real
    - Simulam a sorte com dados, distribuição de cartas, etc.

- Neste caso, além da decisão do agente e do oponente é necessário considerar a sorte
  - Jogos que somente dependem da sorte não são interessantes para a IA
    - Nenhum algoritmo ficou rico por ganhar na loteria

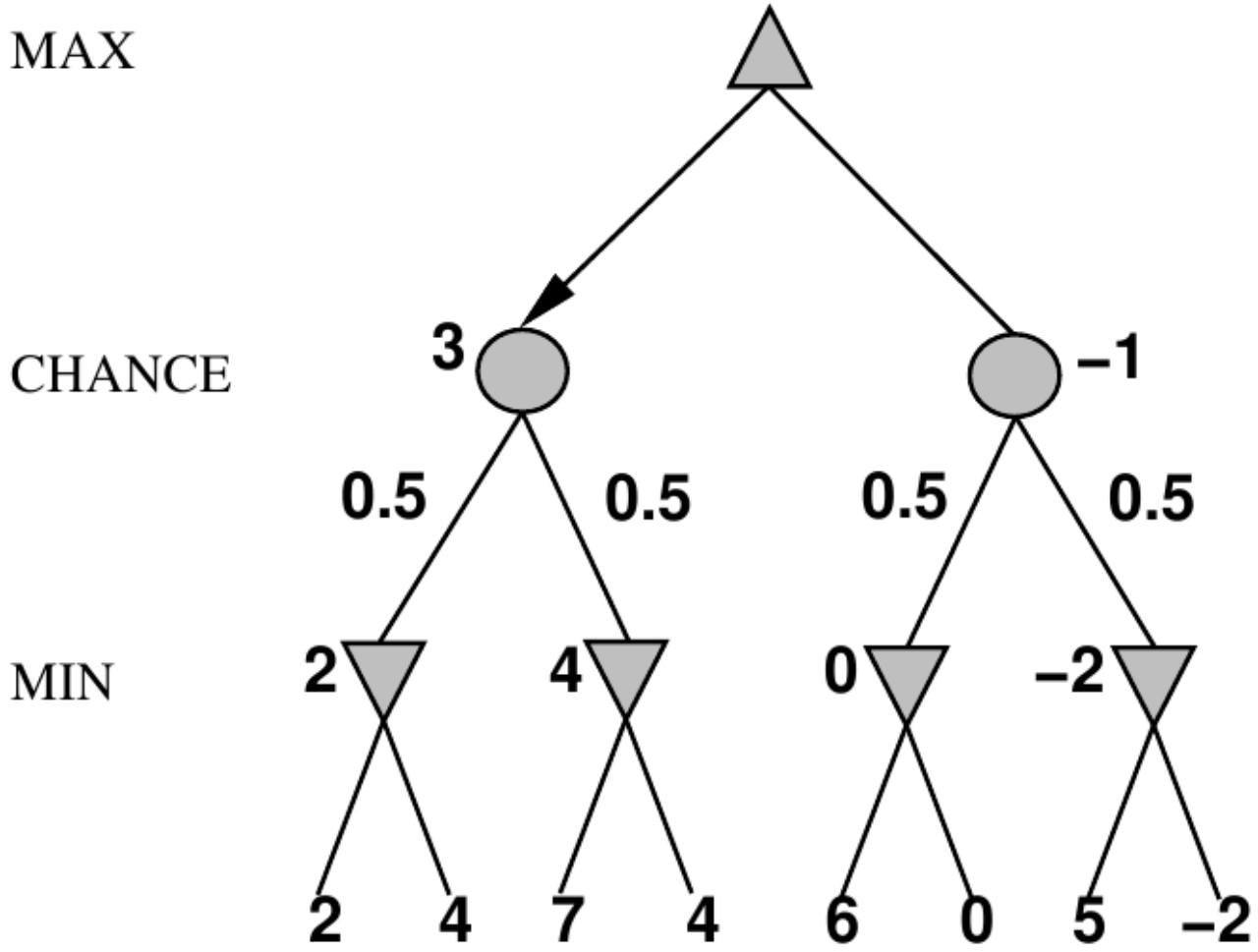
Gamão



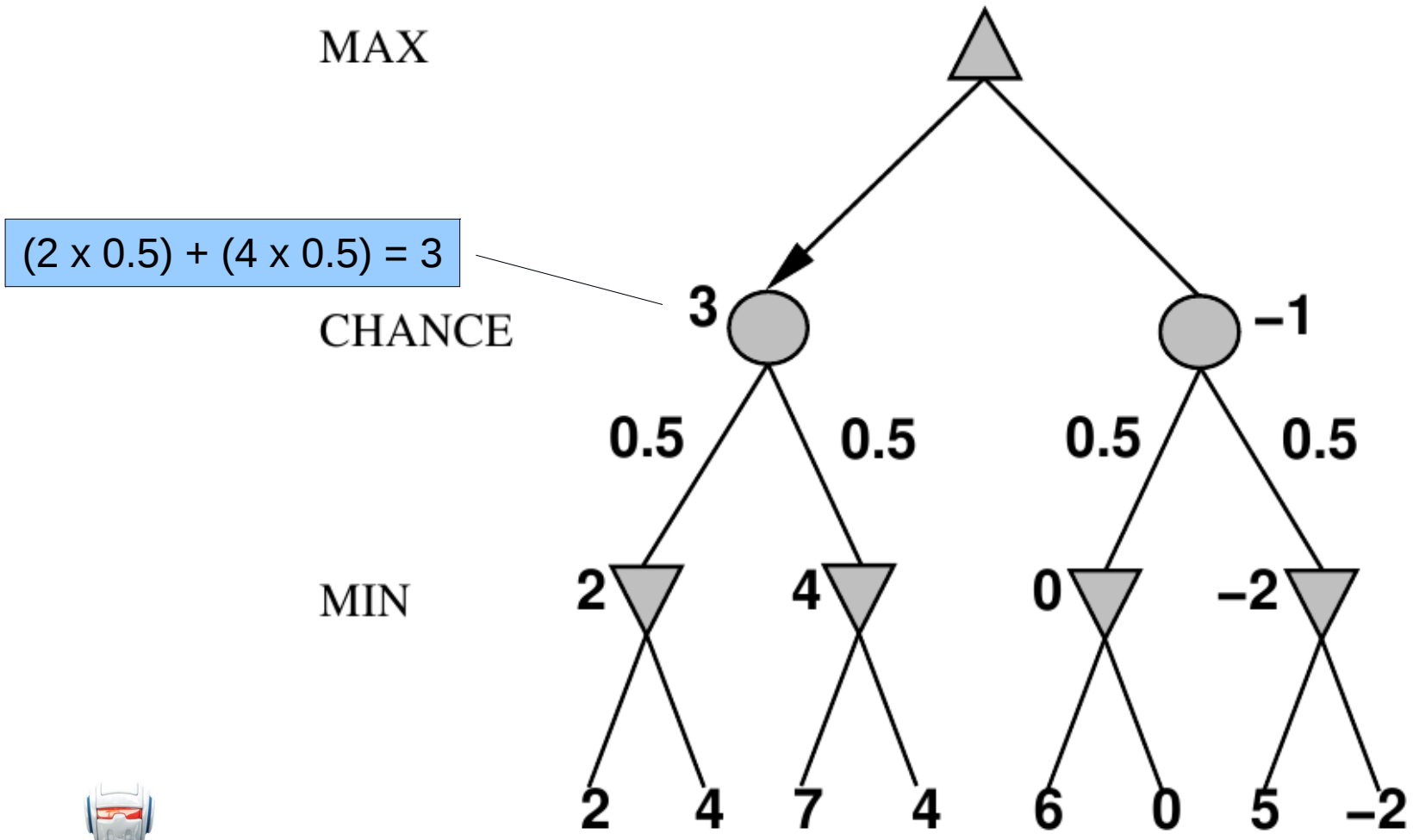
Árvore de Busca do Gamão (Simplificada)



Exemplo Simples (Moeda)



Exemplo Simples (Moeda)



Algoritmo Minimax-esperado

- Igual ao algoritmo Minimax (ou minimax com *alpha-beta pruning*)
  - Testa também os nós de sorte

```
...
if state is a MAX node then
        return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a MIN node then
        return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a chance node then
        return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
...
```

Jogos de Cartas

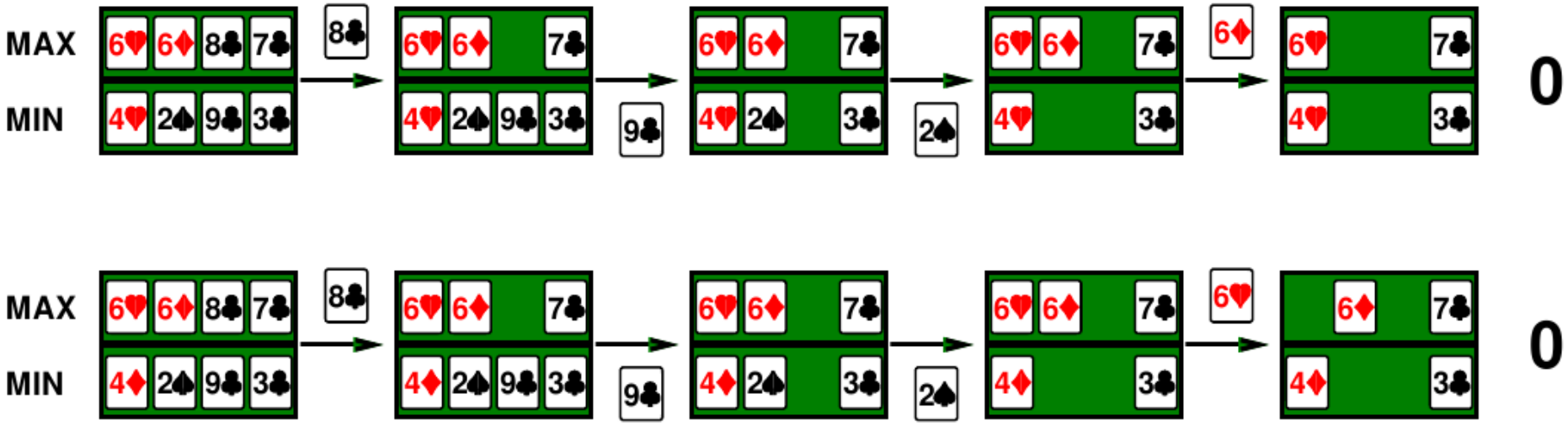
- Como todas as cartas são dadas no início, pode dar a impressão de que um dado gigante é jogado antes da partida
  - Assim, as decisões são como nos outros jogos de sorte, porém podem ser tomadas antes

Jogos de Cartas

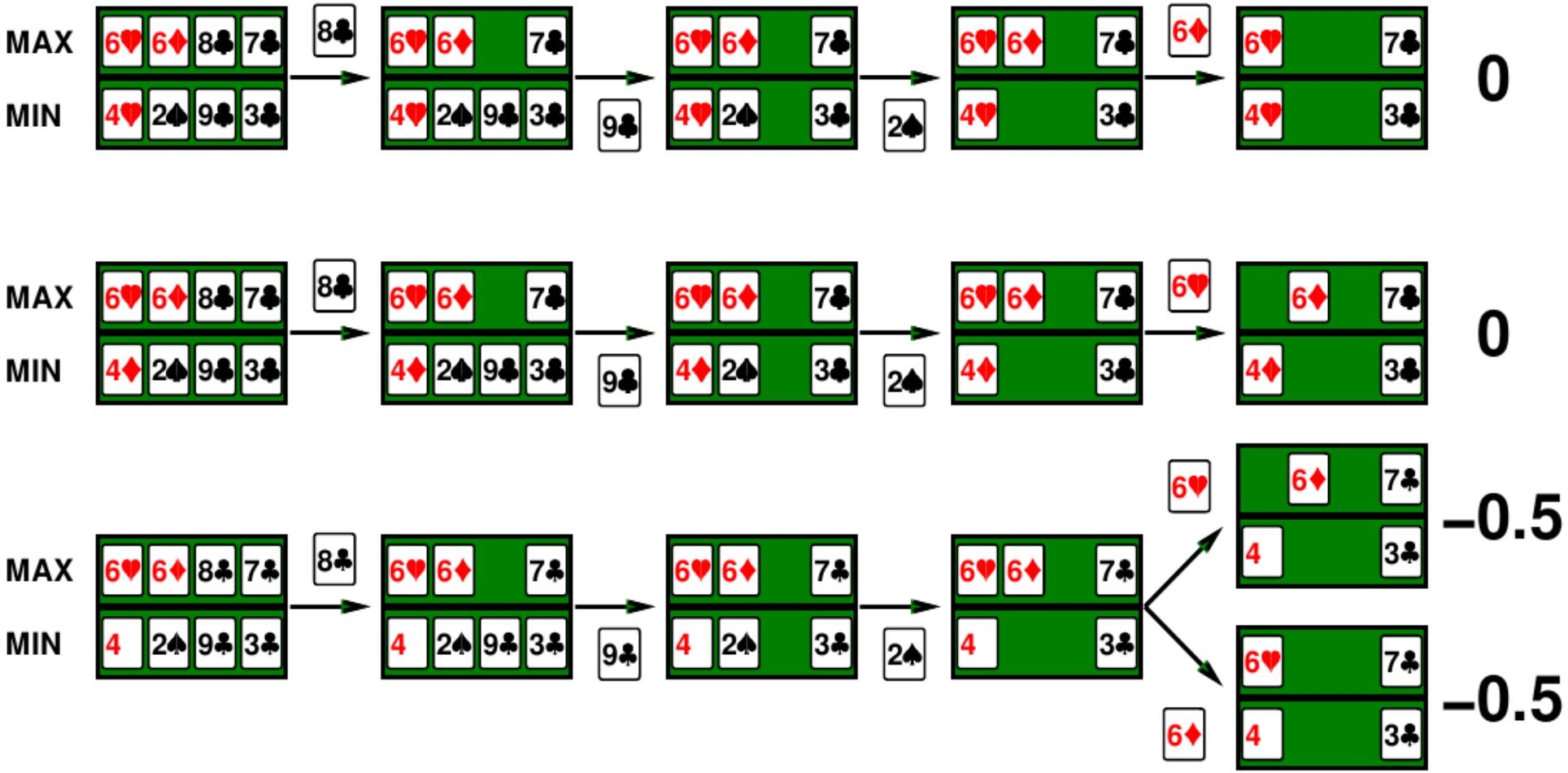
- Como todas as cartas são dadas no início, pode dar a impressão de que um dado gigante é jogado antes da partida
  - Assim, as decisões são como nos outros jogos de sorte, porém podem ser tomadas antes

Jogo de Bridge

Jogo de Bridge



Jogo de Bridge



Exemplo Senso Comum

- Dia 1
  - Caminho A leva para uma pilha de ouro
  - Caminho B leva para uma bifurcação
    - Direita leva para uma pilha de jóias (mais valiosas)
    - Esquerda leva para uma toca de um lobo faminto
- Dia 2
  - Caminho A leva para uma pilha de ouro
  - Caminho B leva para uma bifurcação
    - Direita leva para uma toca de um lobo faminto
    - Esquerda leva para uma pilha de jóias (mais valiosas)
- Dia 3
  - Caminho A leva para uma pilha de ouro
  - Caminho B leva para uma bifurcação
    - Adivinhe corretamente e você ganhará um monte de jóias (mais valiosas). Erre e você irá para uma toca de um lobo faminto

Análise Correta

- A intuição que o valor de uma ação é a média de seus valores em todos possíveis estados está errada!
  - Em ambientes parcialmente observáveis, o valor da ação depende do *estado de crença* no qual o agente está
  - Busca deve ser feita pelos *estados de crença*
  - Deve-se usar ações racionais
    - Agir para obter informação
      - Sinalizar para parceiro (dentro das regras!!!)
      - Agir aleatoriamente para reduzir o estado de crença
      - (Infelizmente, não serão abordados neste curso!)