ADVERSARIAL SEARCH

Today

- Reading
  - AIMA Chapter 5.1-5.5, 5.7,5.8
- Goals
  - Introduce adversarial games
  - Minimax as an optimal strategy
  - Alpha-beta pruning
  - (Real-time decisions)

Adversarial Games

- People like games!
- Games are fun, engaging, and hard-to-solve
- Games are amenable to study: precise, easy-to-represent state space

Game pieces found in a burial site in Southeast Turkey. Dated about 3000 BC

"Game of Twenty squares" discovered in a burial site in Ur. Dated about 2550-2400 BC

Backgammon is also among one of the oldest games still played today

Adversarial Games

- Two-player games have been a focus of AI as long as computers have been around

Checkers

Solved: state space was completely mapped out!

Backgammon and Chess

Computers can compete at a championship level

Go

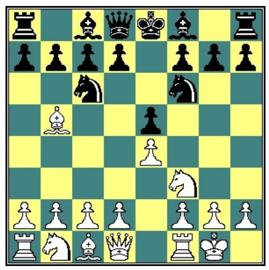
Computers are still at an amateur club-level

Adversarial Games

- Humans and computers have different relative strengths in game play

humans

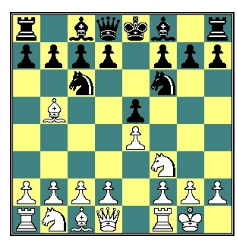
good at evaluating the strength of a board for a player



computers

good at looking ahead in the game to find winning combinations of moves

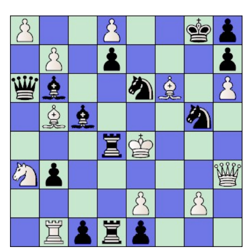
How humans play games



An experiment (by deGroot) was performed in which chess positions were shown to novice and expert players.

experts could reconstruct these perfectly

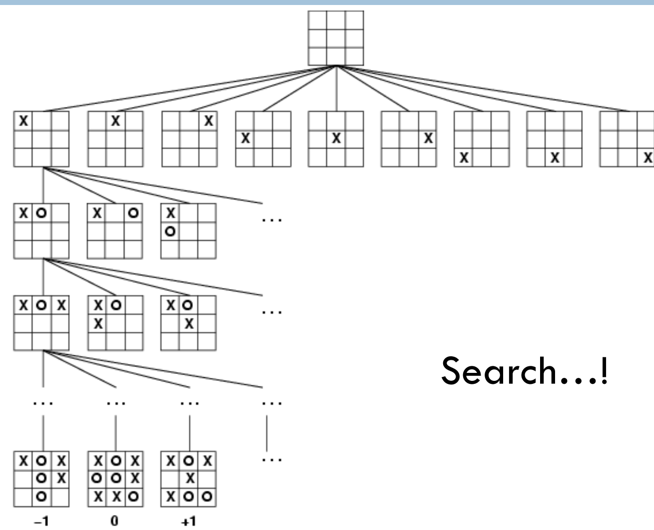
novice players did far worse…



Random chess positions (not legal ones) were then shown to the two groups

experts and novices did just as badly at reconstructing them!

How computers play games



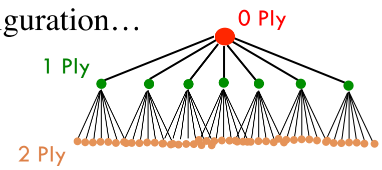
Search…!

Terminology

- deterministic vs. stochastic games
- initial state, successor function, goal test,…
- utility function: defines the final numeric value for a game that ends in terminal state s for player p
  - Chess: +1, 0, ½ for a win, loss, or draw
- zero-sum game: equal and opposite utilities
  - If I win, you lose.
  - Chess: 0 +1 , 1 + 0, ½ + ½
- policy: a function that maps from the set of states to the set of possible actions

Branching factor and depth

On average, there are fewer than 40 possible moves that a chess player can make from any board configuration…



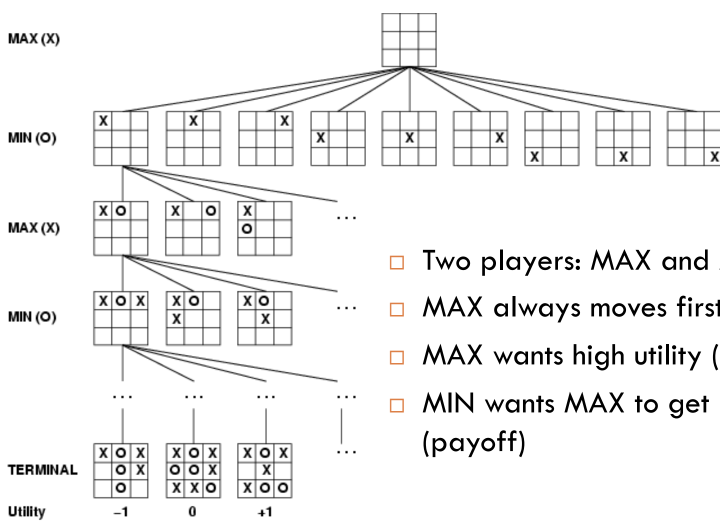
18 Ply!!

Hydra at home in the United Arab Emirates…

Branching Factor Estimates
for different two-player games

Tic-tac-toe	4
Connect Four	7
Checkers	10
Othello	30
Chess	40
Go	300

Simplified representation for two-player games



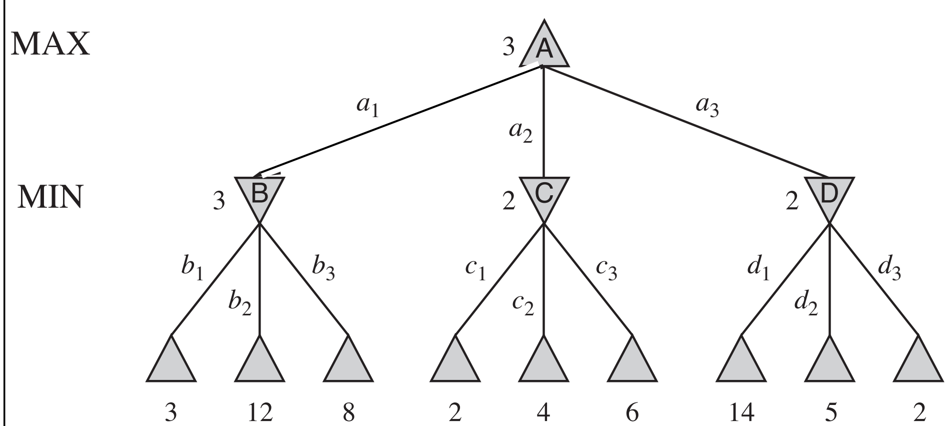
- Two players: MAX and MIN
- MAX always moves first
- MAX wants high utility (payoff)
- MIN wants MAX to get low utility (payoff)

Minimax: an optimal strategy

- An optimal strategy is one that is at least as good as any other, no matter what the opponent does
  - If there's a way to force the win, it will
  - Will only lose if there's no other option

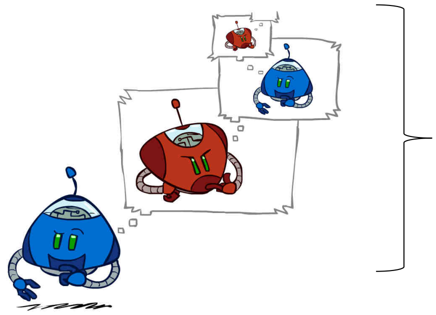
- Minimax is an optimal strategy assuming both players play optimally

Minimax: an optimal strategy



What action should MAX take?

Minimax: an optimal strategy



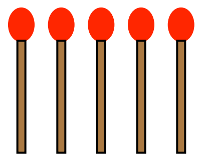
If I did this, then he would do that, but then I would do that, and then he would do this…

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_a \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_a \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

Minimax: An Optimal Strategy

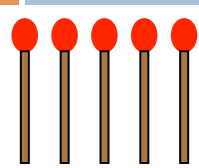
```
function MINIMAX-DECISION(state) returns an action
   v ← MAX-VALUE(state)
   return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   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
```

Minimax: Baby Nim

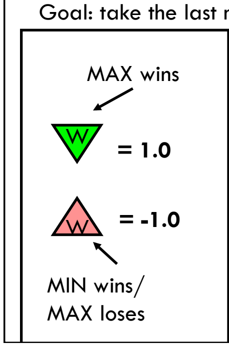


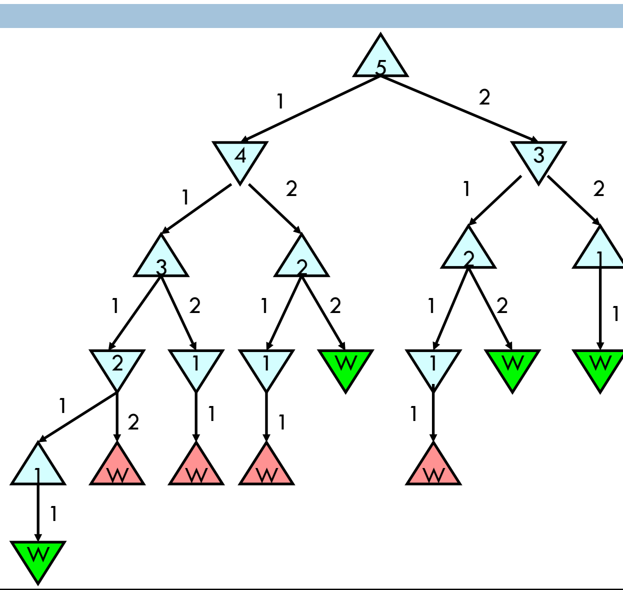
Take 1 or 2 at each turn
Goal: take the last match

Minimax: Baby Nim

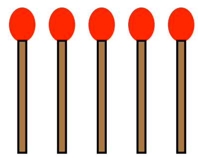


Take 1 or 2 at each turn
Goal: take the last match

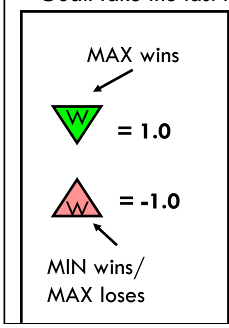


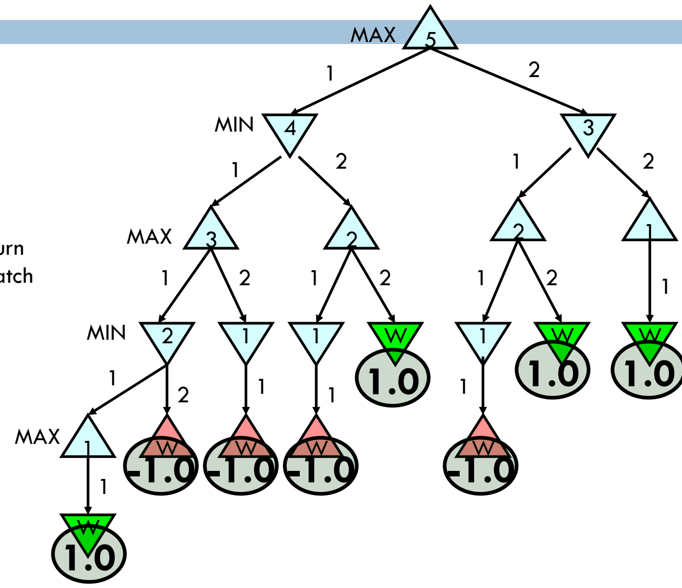


Minimax: Baby Nim

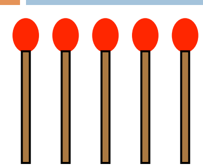


Take 1 or 2 at each turn
Goal: take the last match

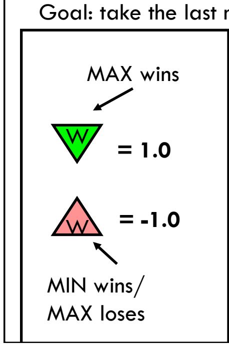


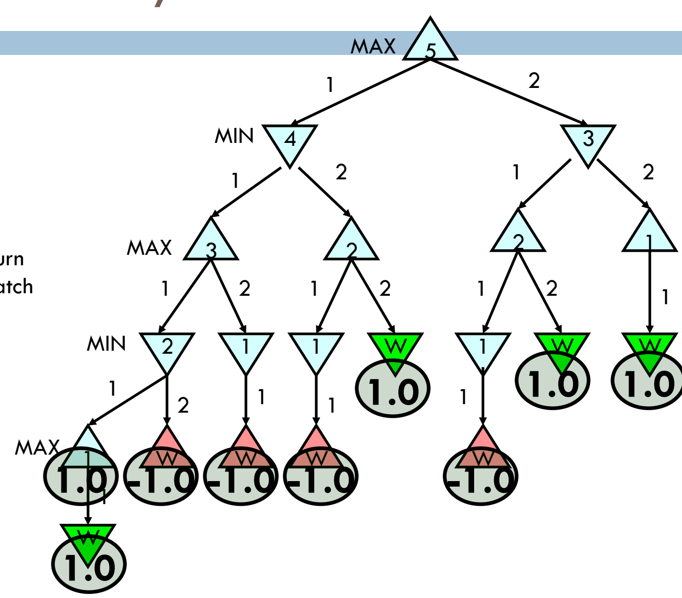


Minimax: Baby Nim

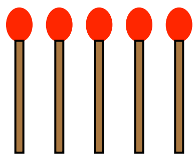


Take 1 or 2 at each turn
Goal: take the last match

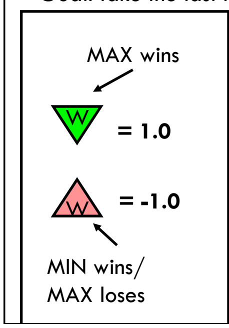


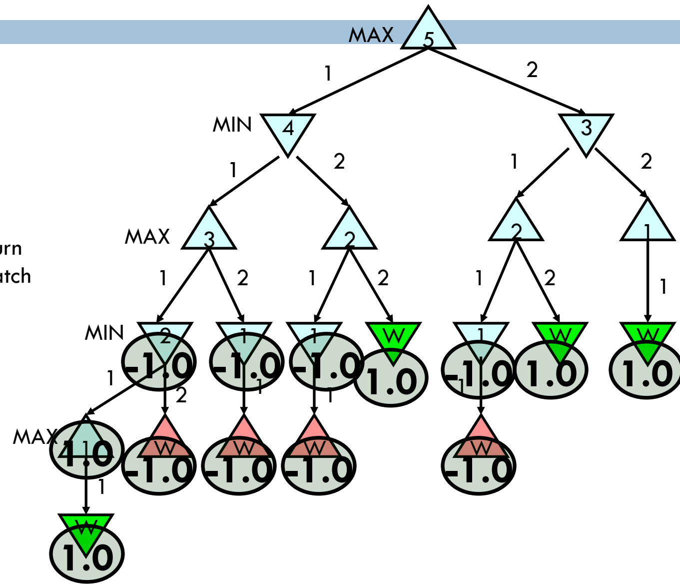


Minimax: Baby Nim

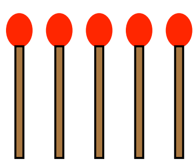


Take 1 or 2 at each turn
Goal: take the last match

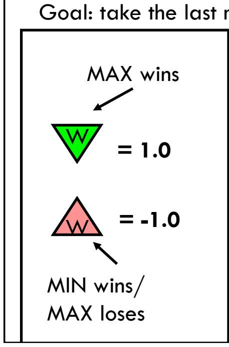


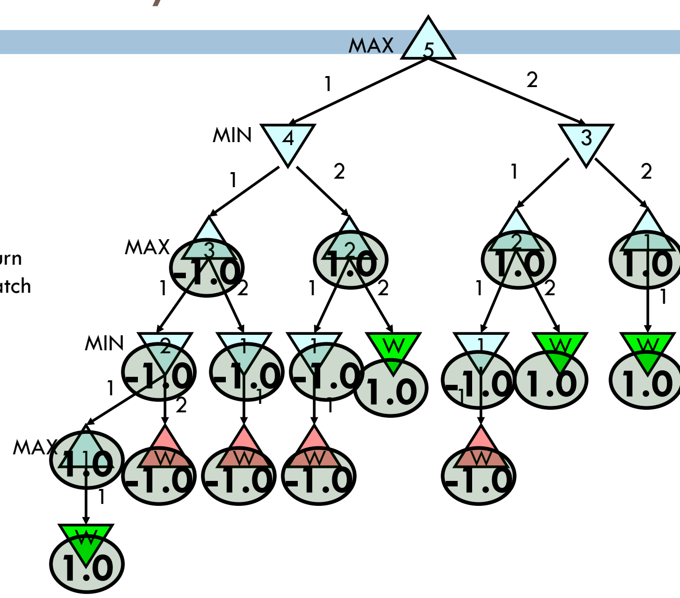


Minimax: Baby Nim

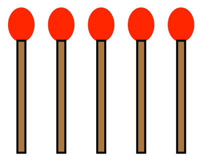


Take 1 or 2 at each turn
Goal: take the last match

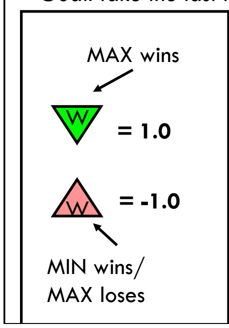


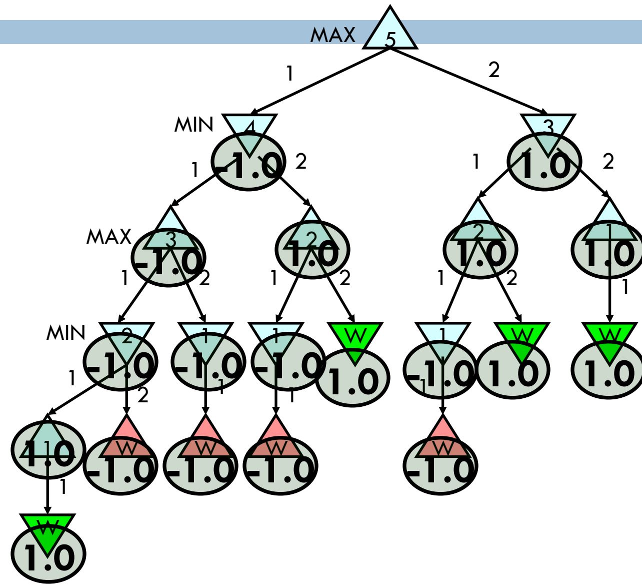


Minimax: Baby Nim

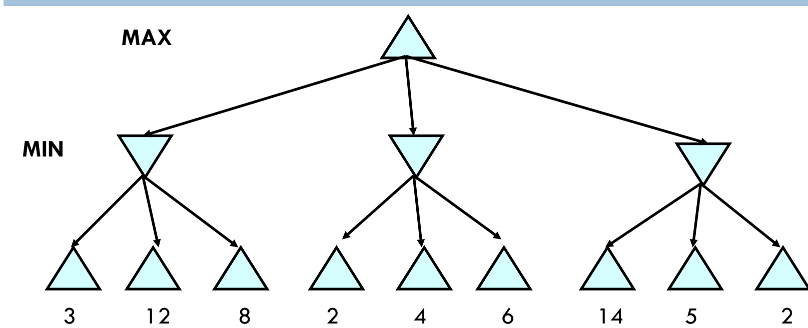


Take 1 or 2 at each turn
Goal: take the last match





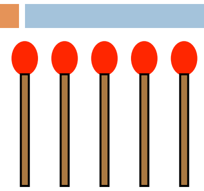
Minimax



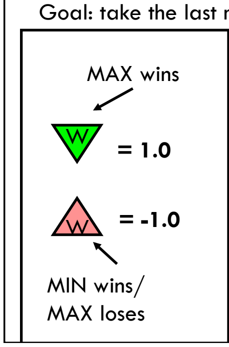
Properties of Minimax

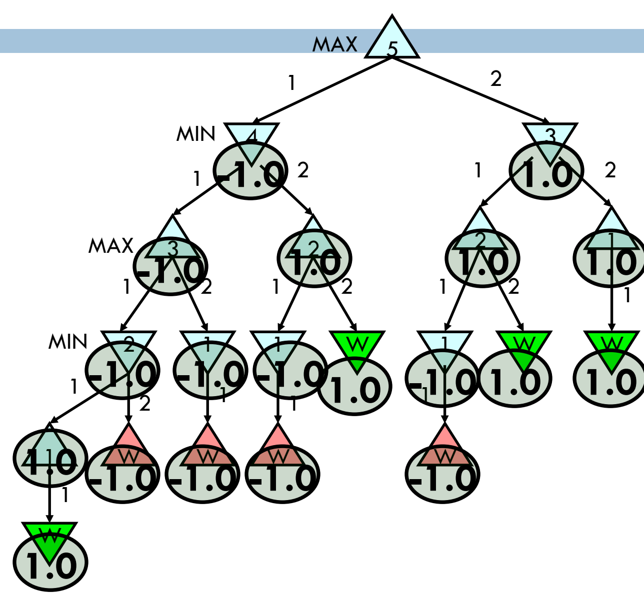
- Minimax performs depth-first exploration of game tree.
  - Recall time complexity for DFS is $O(b^m)$
- For chess, b ≈ 35, d ≈100 for "reasonable" games
  - exact solution completely infeasible
- How can we find the exact solution faster?

Baby Nim



Take 1 or 2 at each turn
Goal: take the last match



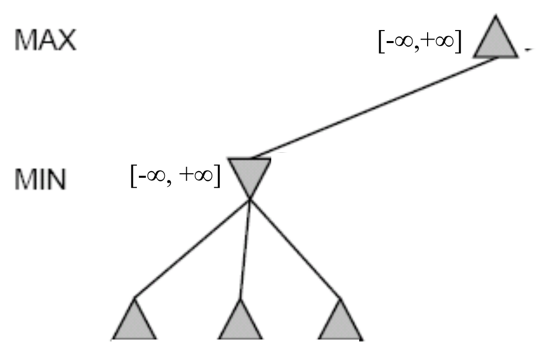


Alpha-Beta Pruning

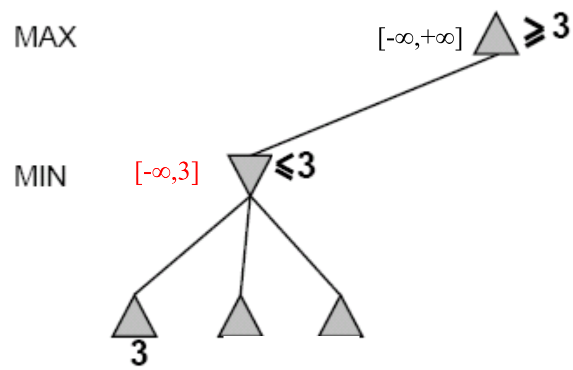
- Alpha-beta pruning: eliminate parts of game tree that don't affect the final result
- Consider a node *n*
  - If a player has a better choice *m* (at a parent or further up), then *n* will never be reached
  - Once we know enough about *n* by looking at some successors we can prune it.

Alpha-Beta Example

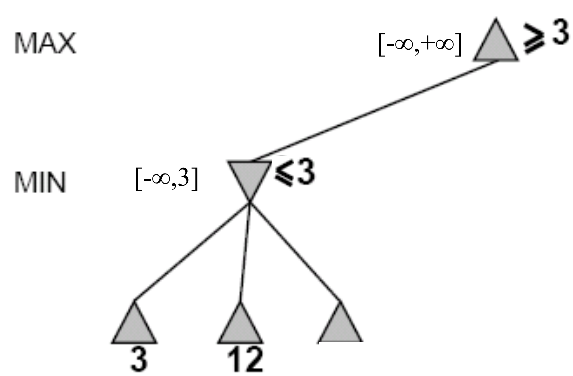
Do depth-first search until first leaf



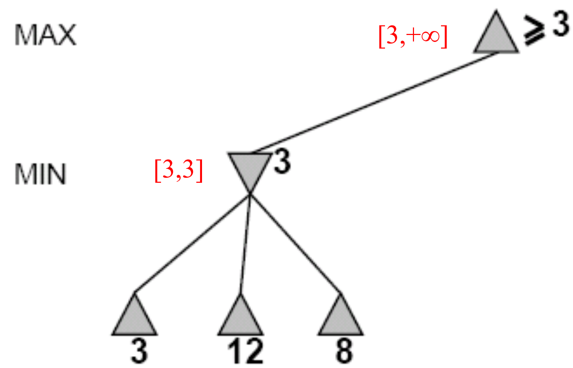
Alpha-Beta Example



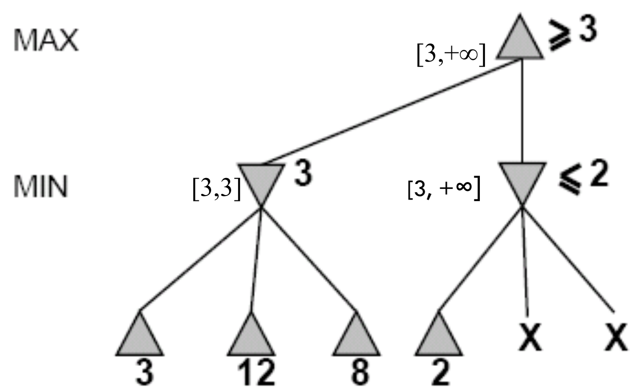
Alpha-Beta Example



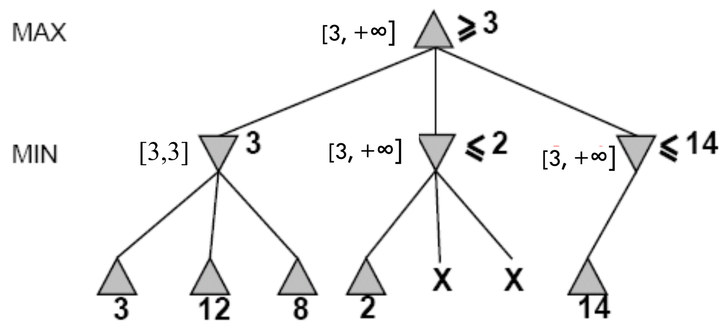
Alpha-Beta Example



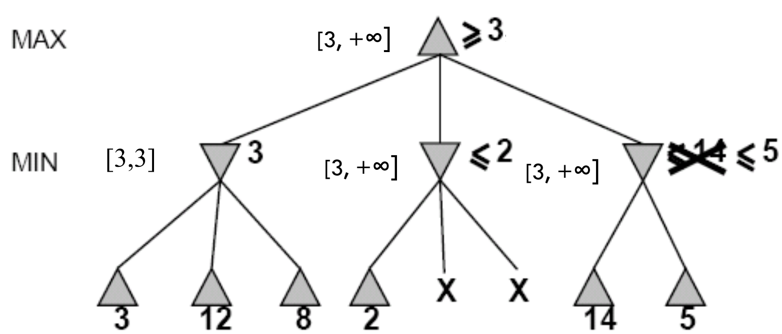
Alpha-Beta Example



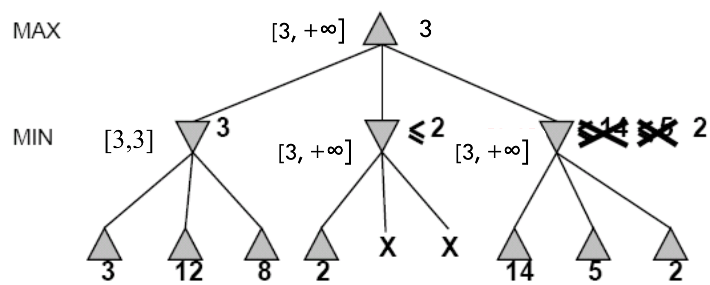
Alpha-Beta Example



Alpha-Beta Example



Alpha-Beta Example

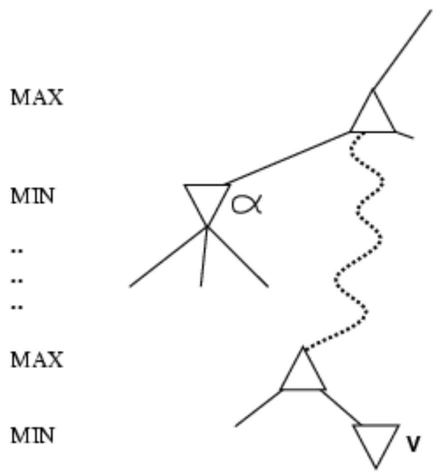


Alpha-Beta pruning

```
function ALPHA-BETA-SEARCH(state) returns an action
   inputs: state, current state in game

   v ← MAX-VALUE(state, −∞, +∞)
   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 ← −∞
   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
   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
```

Why is it called alpha-beta?

- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for MAX
- If v is worse than α, MAX will avoid it
  - prune that branch
- Define β similarly for MIN



Properties of α - β

- Pruning does not affect final result
- However, effectiveness of pruning affected by order in which we examine successors
- What do you do if you don't get to the bottom of the tree on time?