Adversarial Search

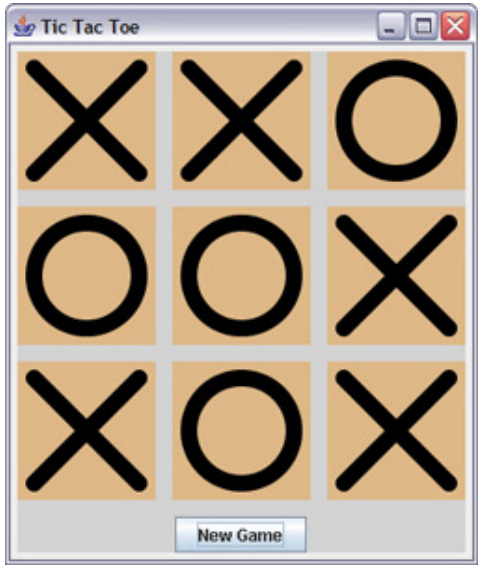
Larry Holder
School of EECS
Washington State University

Games

- Classic AI challenge
  - Easy to represent
  - Difficult to solve
- Perfect information (e.g., Chess, Checkers)
  - Fully observable and deterministic
- Imperfect information (e.g., Poker)
- Chance (e.g., Backgammon)

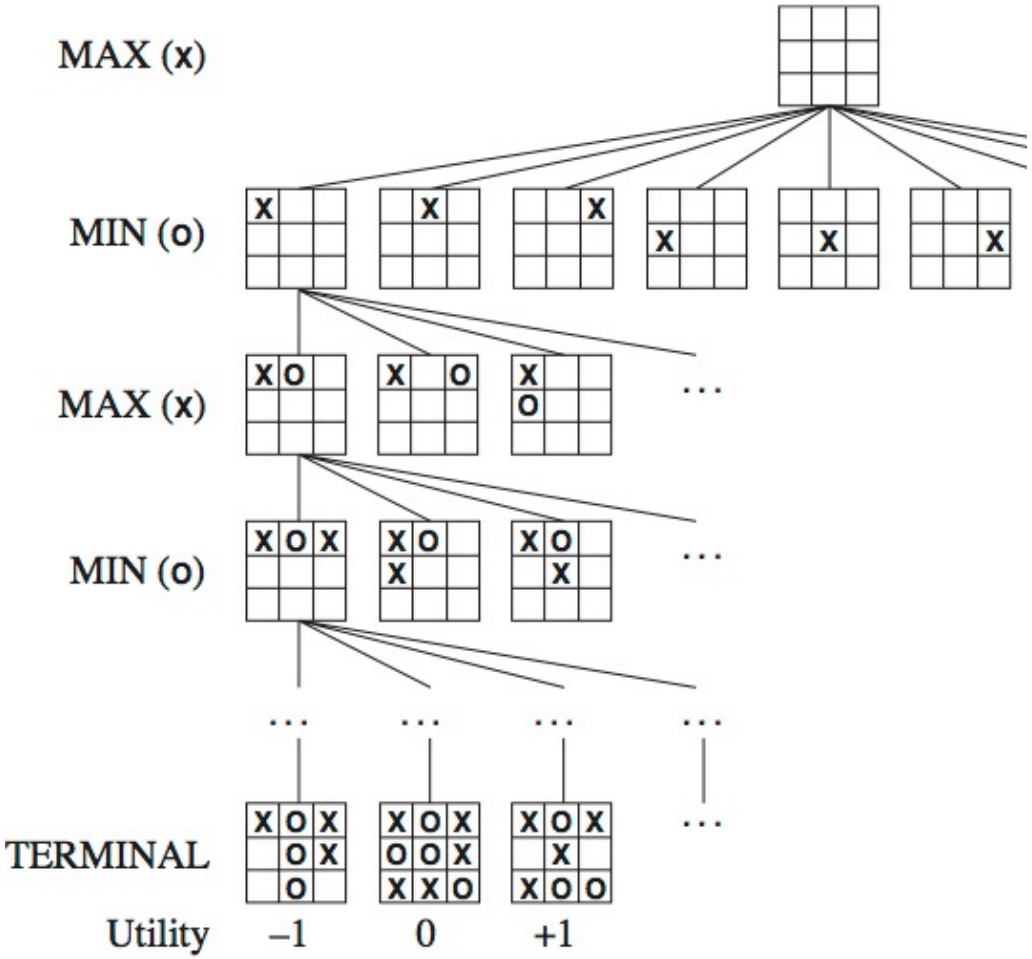
Tic-Tac-Toe

- State space has about $3^9 = 19{,}683$ nodes
- Average branching factor about 2
- Average game length about 8
- Search tree has about $2^8 = 256$ nodes



Game Tree

- MAX wants to maximize its outcome
- MIN wants to minimize its outcome
- Search tree refers to the search for a player's next move
- Terminal node
- Utility

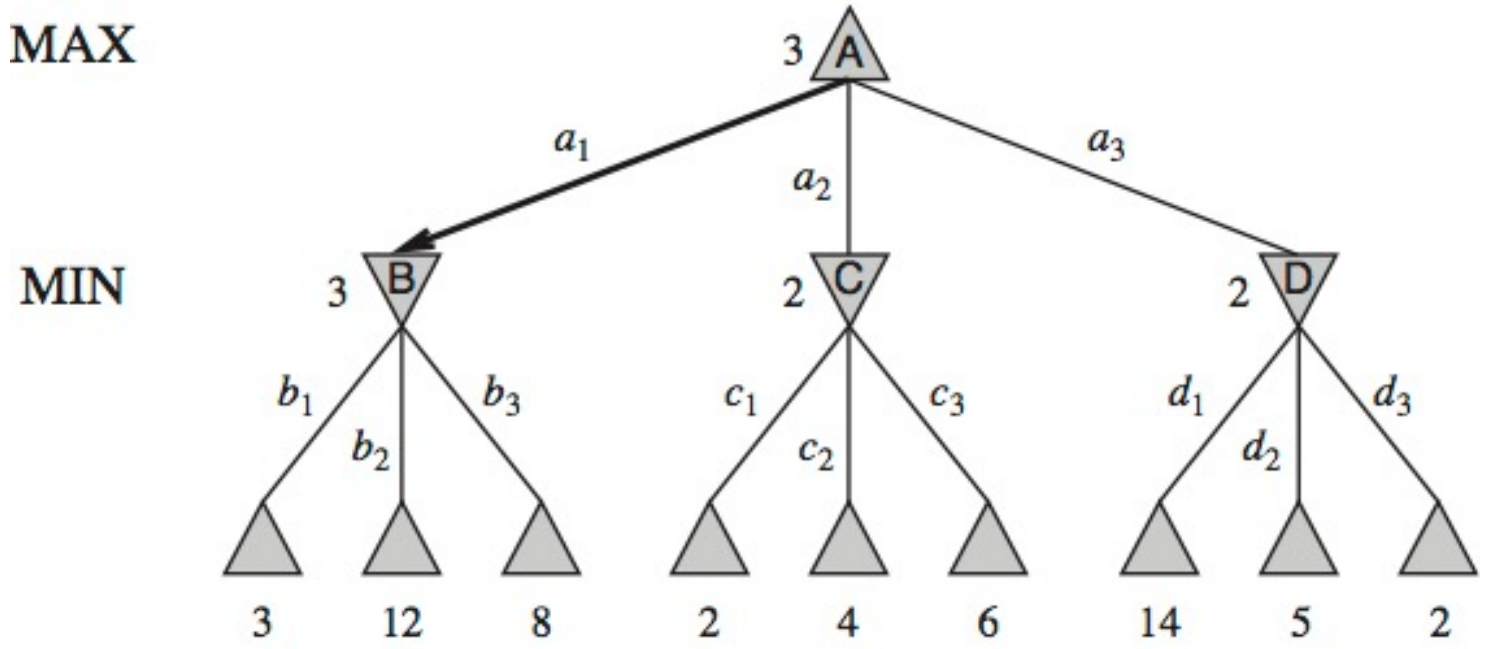


Chess

- State space about 10^{40} nodes
- Average branching factor about 35
- Average game length about 100 (50 moves per player)
- Search tree has about $35^{100} = 10^{154}$ nodes

Garry Kasparov vs. IBM's Deep Blue (1997)

Optimal Play



Optimal Play

- Minimax value
  - Best player can achieve assuming all players play optimally

$$\text{Minimax(s)} = \begin{cases} \text{Utility}(s) & \text{if} \quad \text{TerminalTest}(s) \\ \max_{a \in \text{Actions(s)}} \text{Minimax}(\text{Result}(s,a)) & \text{if} \quad \text{Player(s)} = \text{MAX} \\ \min_{a \in \text{Actions(s)}} \text{Minimax}(\text{Result}(s,a)) & \text{if} \quad \text{Player(s)} = \text{MIN} \end{cases}$$

- Minimax decision
  - Action that leads to minimax value

Minimax Algorithm

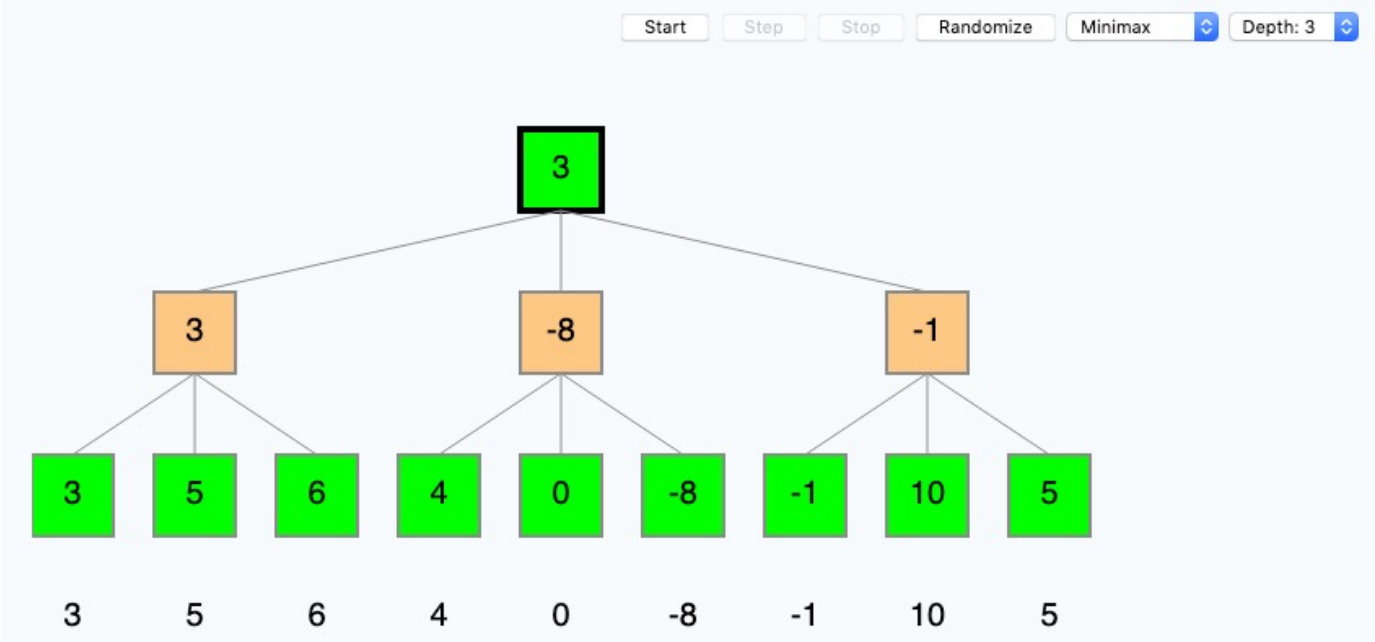
```
function MINIMAX-DECISION (state) returns an action
  return arg max_{a ∈ ACTIONS(state)} MIN-VALUE(RESULT(state,a))

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

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

Minimax Demo

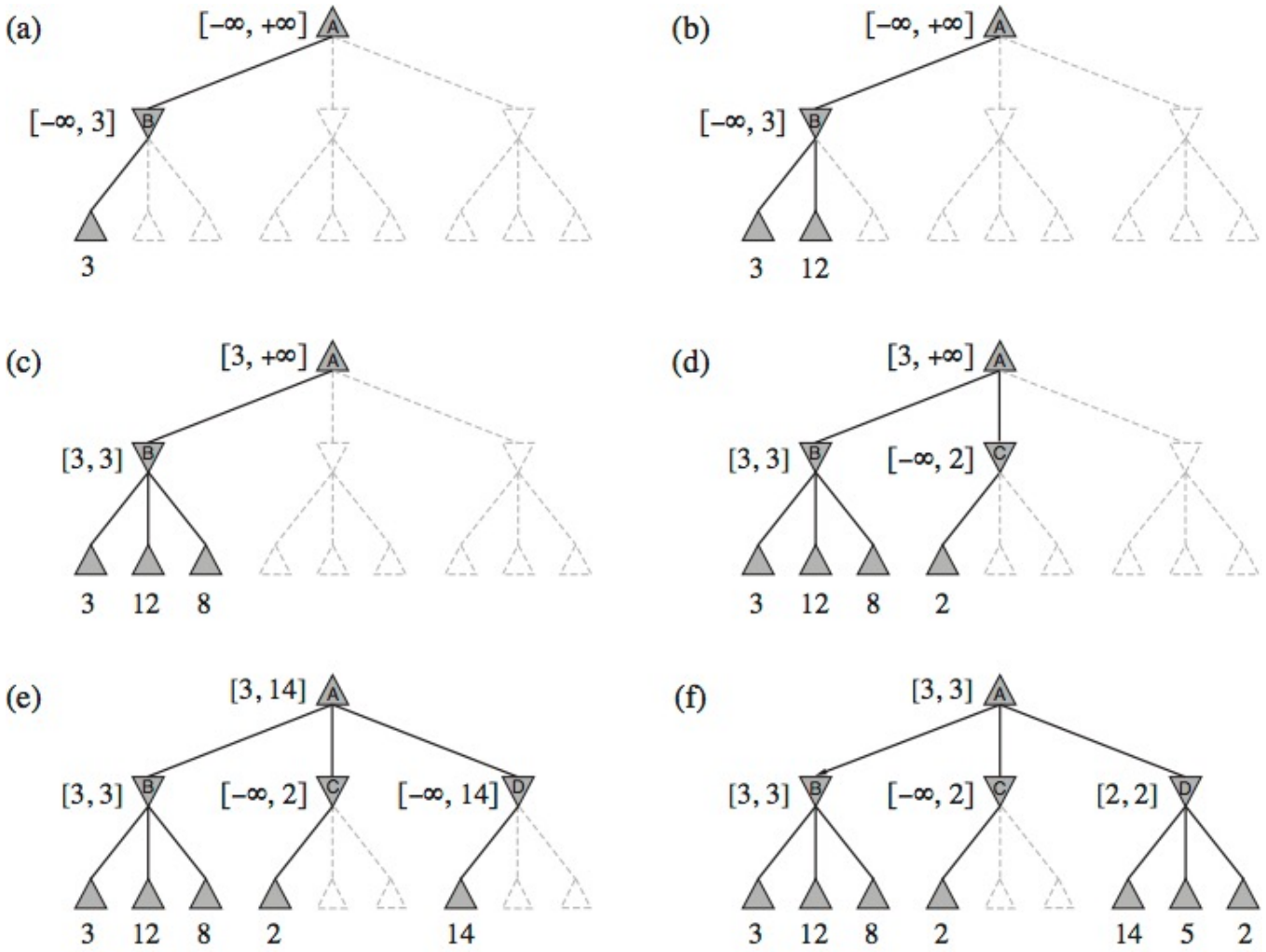
- www.yosenspace.com/posts/computer-science-game-trees.html



Minimax Algorithm

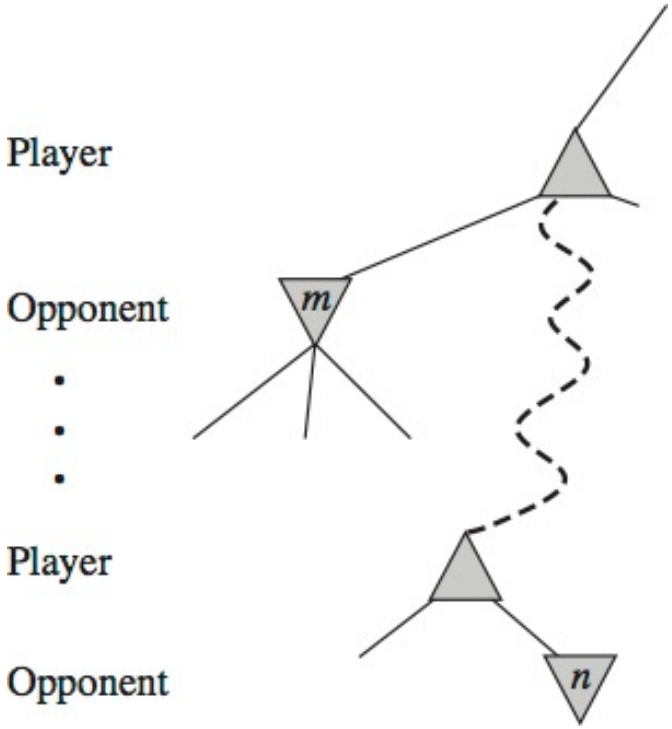
- Essentially depth-first search of game tree
- Time complexity: $O(b^m)$
  - m = maximum tree depth
  - b = legal moves at each state
- Space complexity
  - Generates all actions: O(bm)
  - Generates one action: O(m)
- Practical?

Pruning Search Tree



Alpha-Beta Pruning

- Prune parts of the search tree that MAX and MIN would never choose
- α = value of best choice for MAX so far (highest value)
- β = value of best choice for MIN so far (lowest value)
- Keep track of alpha α and beta β during search



If m > n, Player will never move to n.

Alpha-Beta Pruning

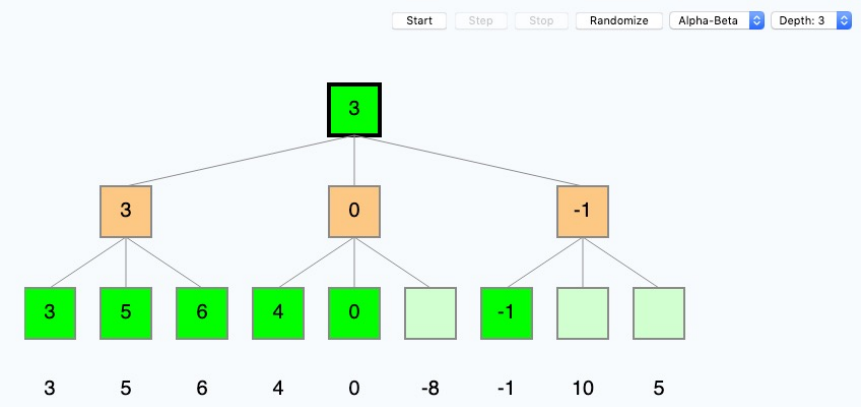
```
function ALPHA-BETA-SEARCH (state) returns an action
  v ← MAX-VALUE(state, -∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE (state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← -∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(state,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v
```

```
function MIN-VALUE (state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(state,a), α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

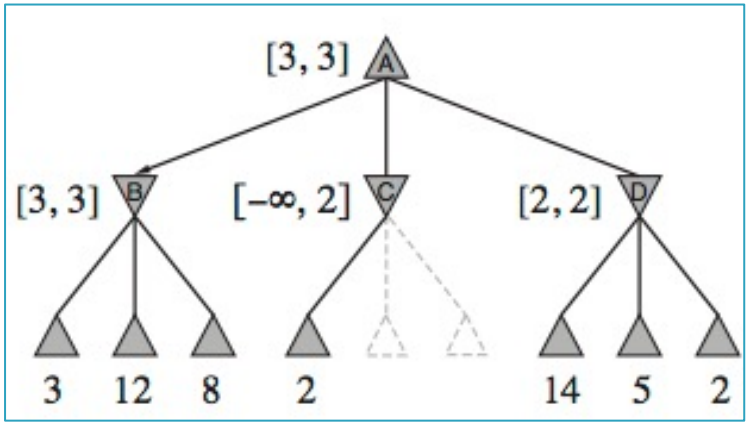
Alpha-Beta Pruning Demo

- www.yosenspace.com/posts/computer-science-game-trees.html



Move Ordering

- ALPHA-BETA-SEARCH still $O(b^m)$ worst case
- If order moves by value, then could prune maximally (always choose best move next)
  - Achieve $O(b^{m/2})$ time
  - Branching factor $b^{1/2}$
  - Chess: 35 → 6
  - But not practical
- Choosing moves randomly
  - Achieve $O(b^{3m/4})$ average case
- Choosing moves based on impact
  - E.g., chess: captures, threats, forward, backward
  - Closer to $O(b^{m/2})$



Real-Time Game Play

- Minimax and Alpha-Beta search to terminal nodes
- Impractical for most games due to time limits
- Employ cutoff test to treat nodes as terminal nodes
- Heuristic evaluation function at these nodes to estimate utility
- d = depth

$$\text{H-Minimax}(s,d) =$$

$$\begin{cases} \text{Eval}(s) & \text{if} \quad \text{CutoffTest}(s,d) \\ \max_{a \in \text{Actions(s)}} \text{H-Minimax}(\text{Result}(s,a), d+1) & \text{if} \quad \text{Player(s)} = \text{MAX} \\ \min_{a \in \text{Actions(s)}} \text{H-Minimax}(\text{Result}(s,a), d+1) & \text{if} \quad \text{Player(s)} = \text{MIN} \end{cases}$$

Real-Time Game Play

- Cutoff test
  - Depth-limit, iterative deepening until time's up
- Heuristic evaluation function EVAL(s)
  - Weighted combination of features

$$Eval(s) = \sum_{i=1}^{n} w_i f_i(s)$$

    - E.g., chess
      - $f_1(s)$ = #pawns, $w_1 = 1$
      - $f_4(s)$ = #bishops, $w_4 = 3$
  - Learn weights
  - Learn features

Go

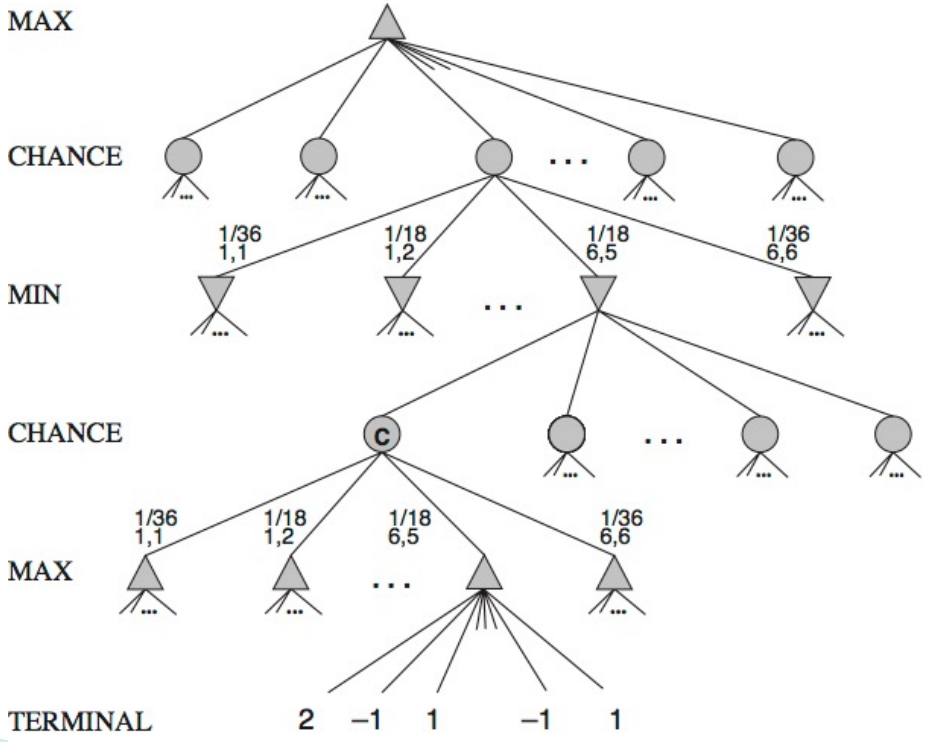
- State space about 10^{170} nodes
- Average branching factor about 250
- Average game length about 200 (100 moves per player)
- Search tree has about $250^{200} = 10^{480}$ nodes

Lee Sodol vs. Google DeepMind's AlphaGo (2016)

deepmind.com/research/alphago

Stochastic Games

- Element of chance (e.g., dice roll)
- Include chance nodes in game tree
  - Branch to possible outcomes with their probabilities



Stochastic Games

- Can't compute minimax values
- Can compute expected minimax values

$$\text{ExpectiMinimax}(s) =$$

$$\begin{cases} \text{Utility}(s) & \text{if} \quad \text{TerminalTest}(s) \\ \max_{a \in \text{Actions(s)}} \text{ExpectiMinimax}(\text{Result}(s,a)) & \text{if} \quad \text{Player(s)} = \text{MAX} \\ \min_{a \in \text{Actions(s)}} \text{ExpectiMinimax}(\text{Result}(s,a)) & \text{if} \quad \text{Player(s)} = \text{MIN} \\ \sum_r P(r)\,\text{ExpectiMinimax}(\text{Result}(s,r)) & \text{if} \quad \text{Player(s)} = \text{CHANCE} \end{cases}$$

  - r represents possible chance event (e.g., dice roll)
  - Result(s,r) = state s with a particular outcome r

Stochastic Games

- Chance nodes increase branching factor
- Search time complexity $O(b^m n^m)$
  - Where n is the number of chance outcomes
  - E.g., backgammon: $n = 21$, $b \approx 20$ (can be large)
  - Can only search a few moves ahead
- Estimate ExpectiMinimax values

Partially Observable Games

- Can reason about all possible states of unknown information
- If P(s) represents probability of each unknown state s, then best move is:

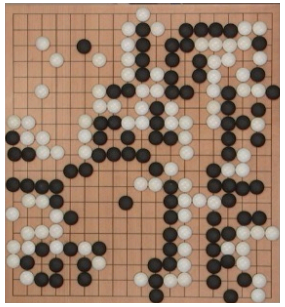
$$\arg\max_{a} \sum_{s} P(s)\text{Minimax}(\text{Result}(s,a))$$

- If |s| too large, take a random sample
  - Monte Carlo method

State of the Art

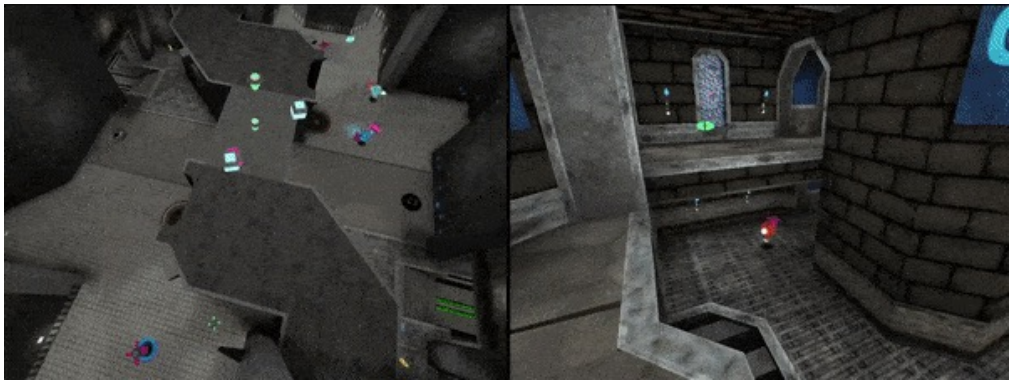
- Checkers (solved, perfect play)
  - Chinook (webdocs.cs.ualberta.ca/~chinook)
  - Open/close database plus brute-force search
- Chess
  - Komodo (komodochess.com) – proprietary
  - Stockfish (stockfishchess.org) – open source
- Go
  - AlphaGo (deepmind.com/research/alphago)
  - Zen (senseis.xmp.net/?ZenGoProgram)
- Backgammon
  - Extreme Gammon (www.extremegammon.com)
  - GNU Backgammon (www.gnu.org/software/gnubg)
  - Neural network based evaluation function
- Poker
  - DeepStack (www.deepstack.ai)
  - Pluribus (ai.facebook.com/blog/pluribus-first-ai-to-beat-pros-in-6-player-poker)

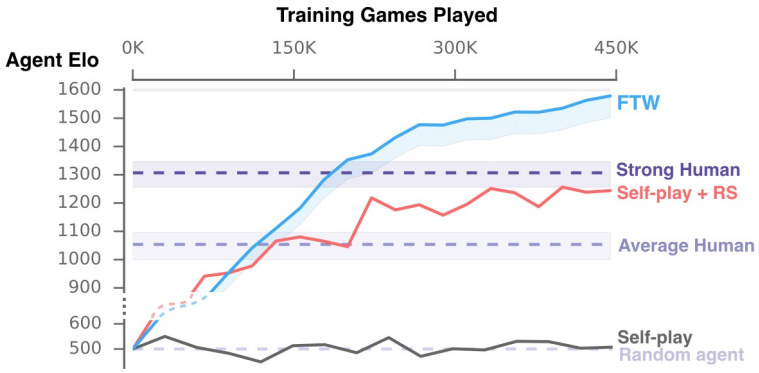




State of the Art

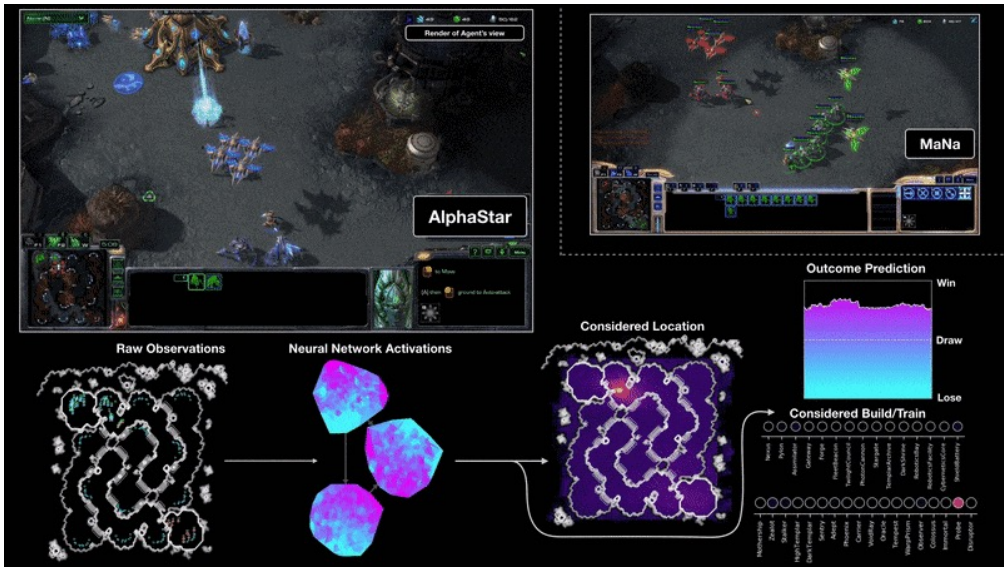
- First-person shooter (FPS) games
  - DeepMind's "For-The-Win" (FTW) Quake III agent
  - deepmind.com/blog/article/capture-the-flag-science

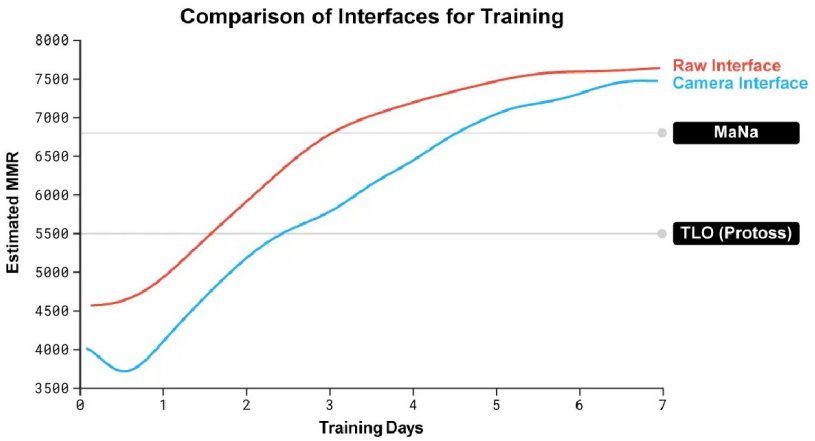




State of the Art

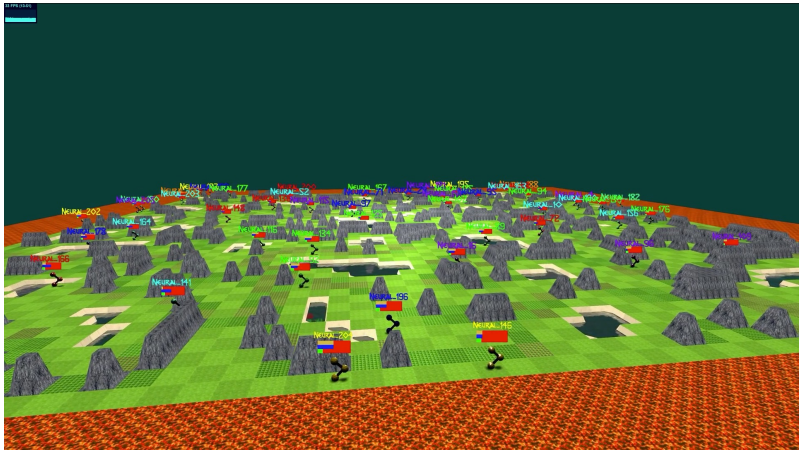
- Real-Time Strategy (RTS) games
  - DeepMind's AlphaStar masters StarCraft

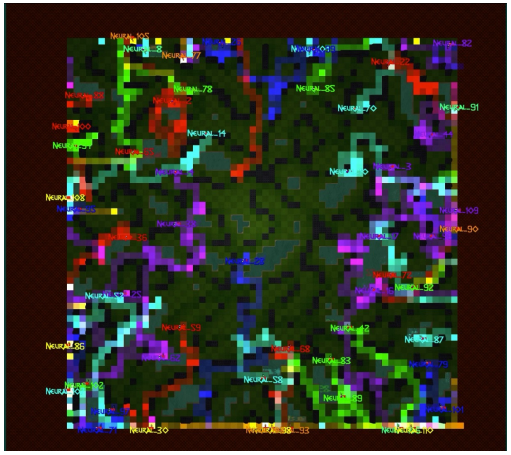




State of the Art

- Role-playing games (RPG/MMORPG)
- Neuro MMO
  - openai.com/blog/neural-mmo





Summary

- Adversarial search and games
- Minimax search
- Alpha-beta pruning
- Real-time issues
- Stochastic and partially observable games
- State of the art …

Are there any games that humans can still beat computers?