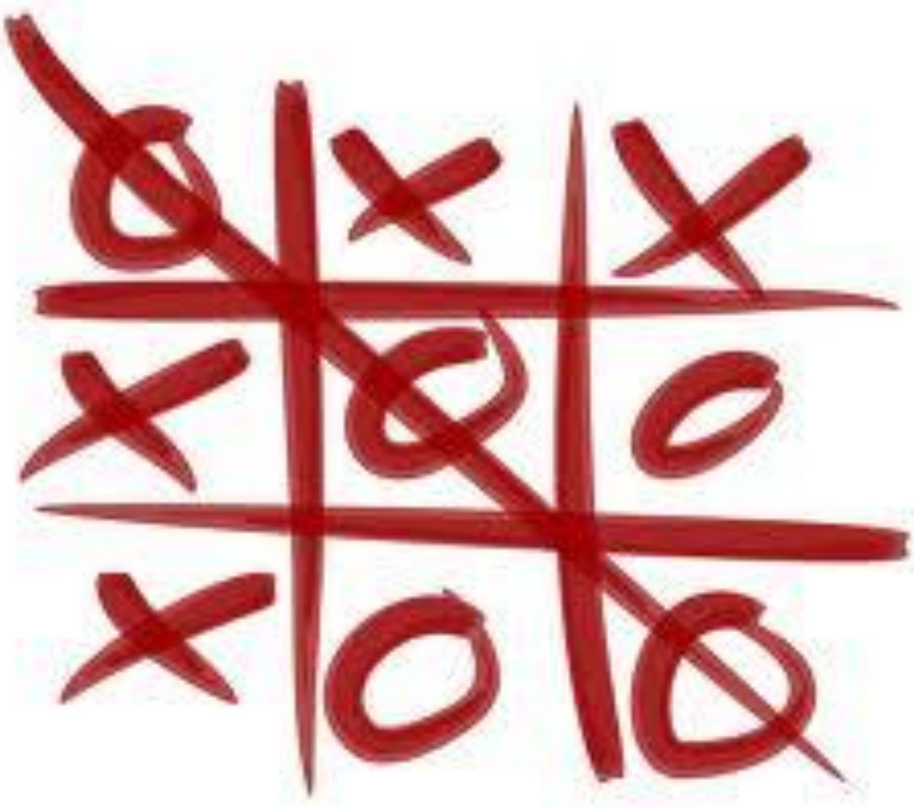
Adversarial Search

Game Playing

An AI Favorite

- structured task, often a symbol of “intelligence”
- clear definition of success and failure
- does not require large amounts of knowledge (at first glance)
- focus on games of perfect information
- multiplayer, chance



Game Playing

Initial State is the initial board/position

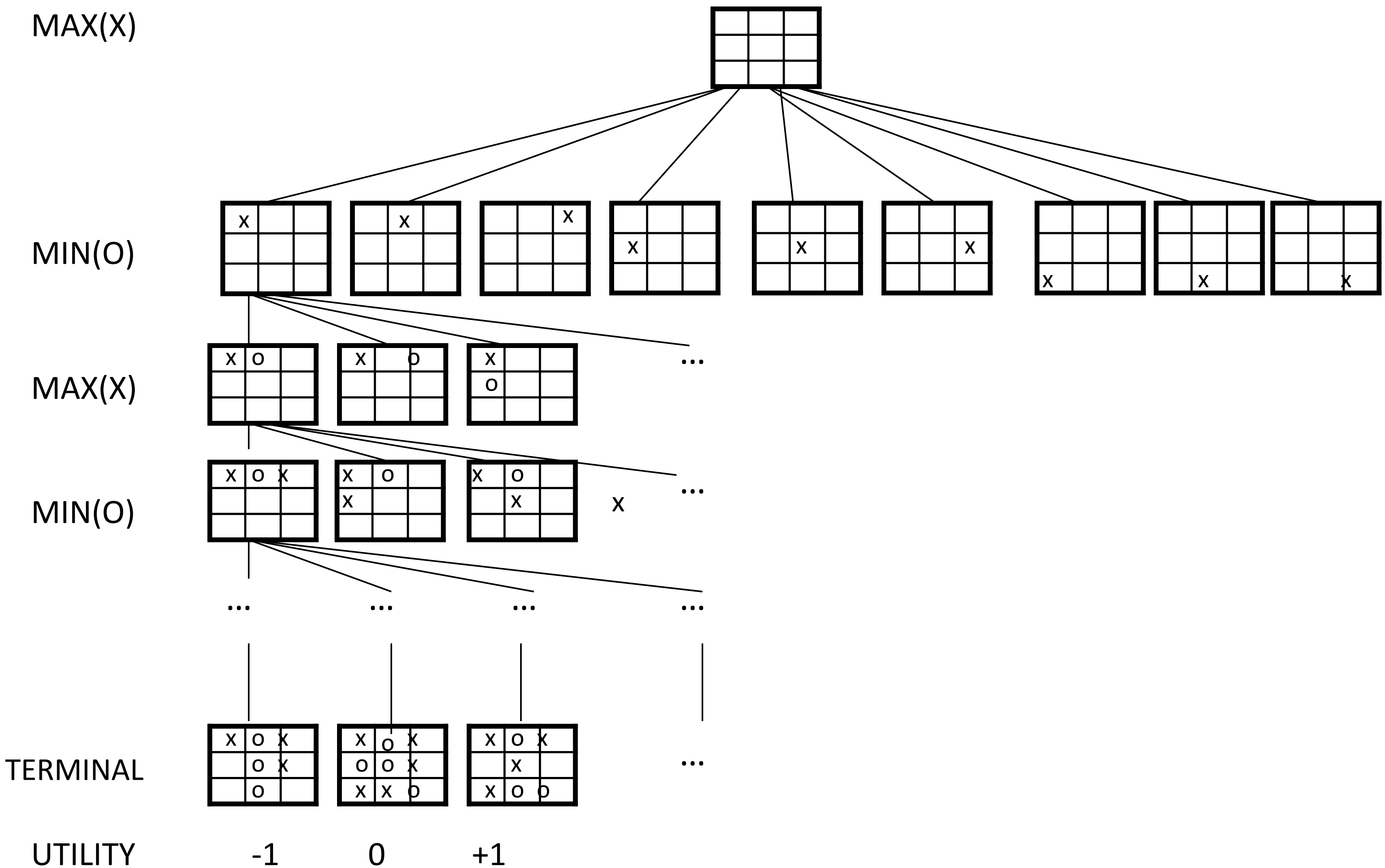
Successor Function defines the set of legal moves from any position

Terminal Test determines when the game is over

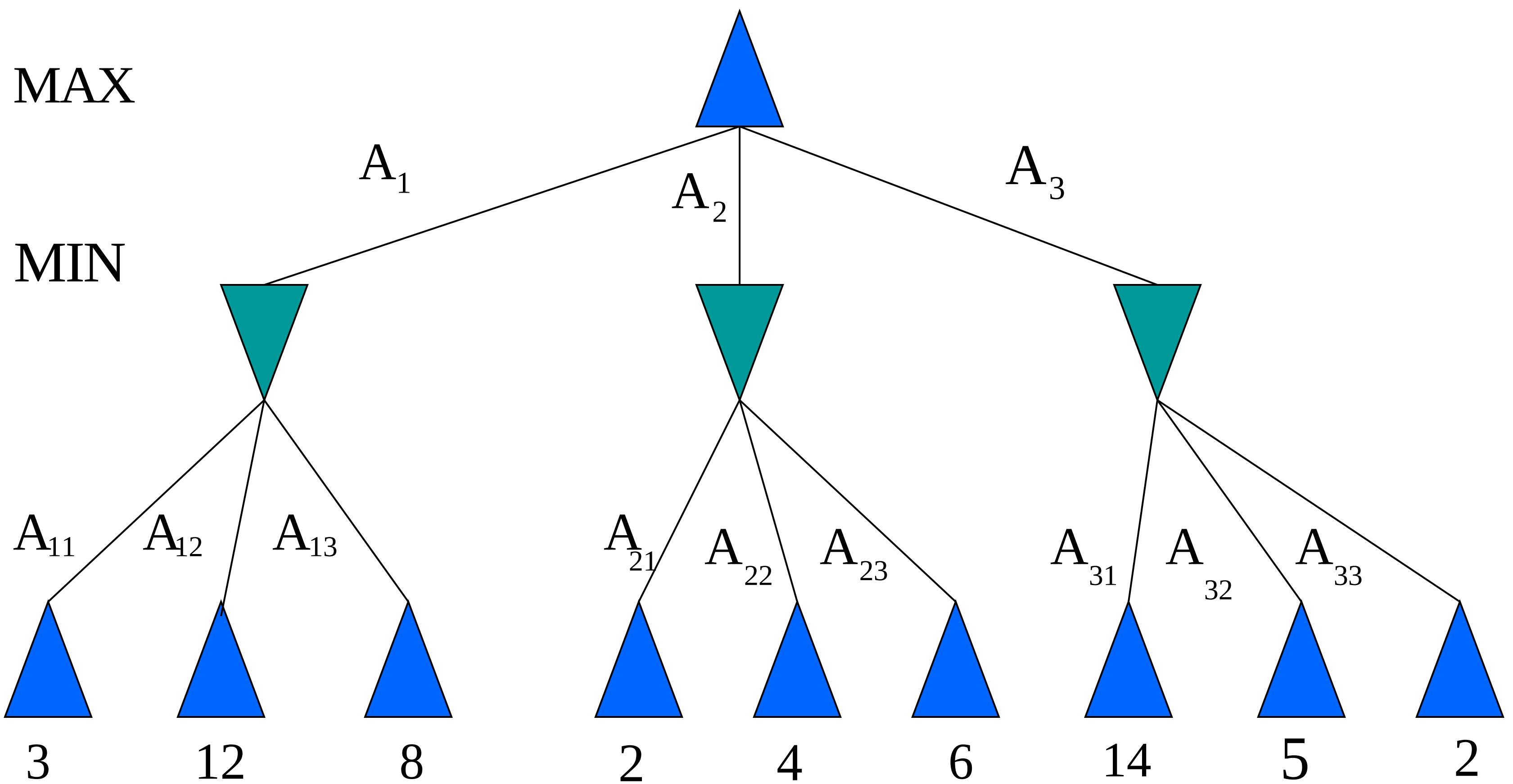
Utility Function gives a numeric outcome for the game

For chess, only win, lose, draw. Backgammon: +192 to -192.

Partial Search Tree for Tic-Tac-Toe



Game Playing as Search



Two Ply

Simplified Minimax Algorithm

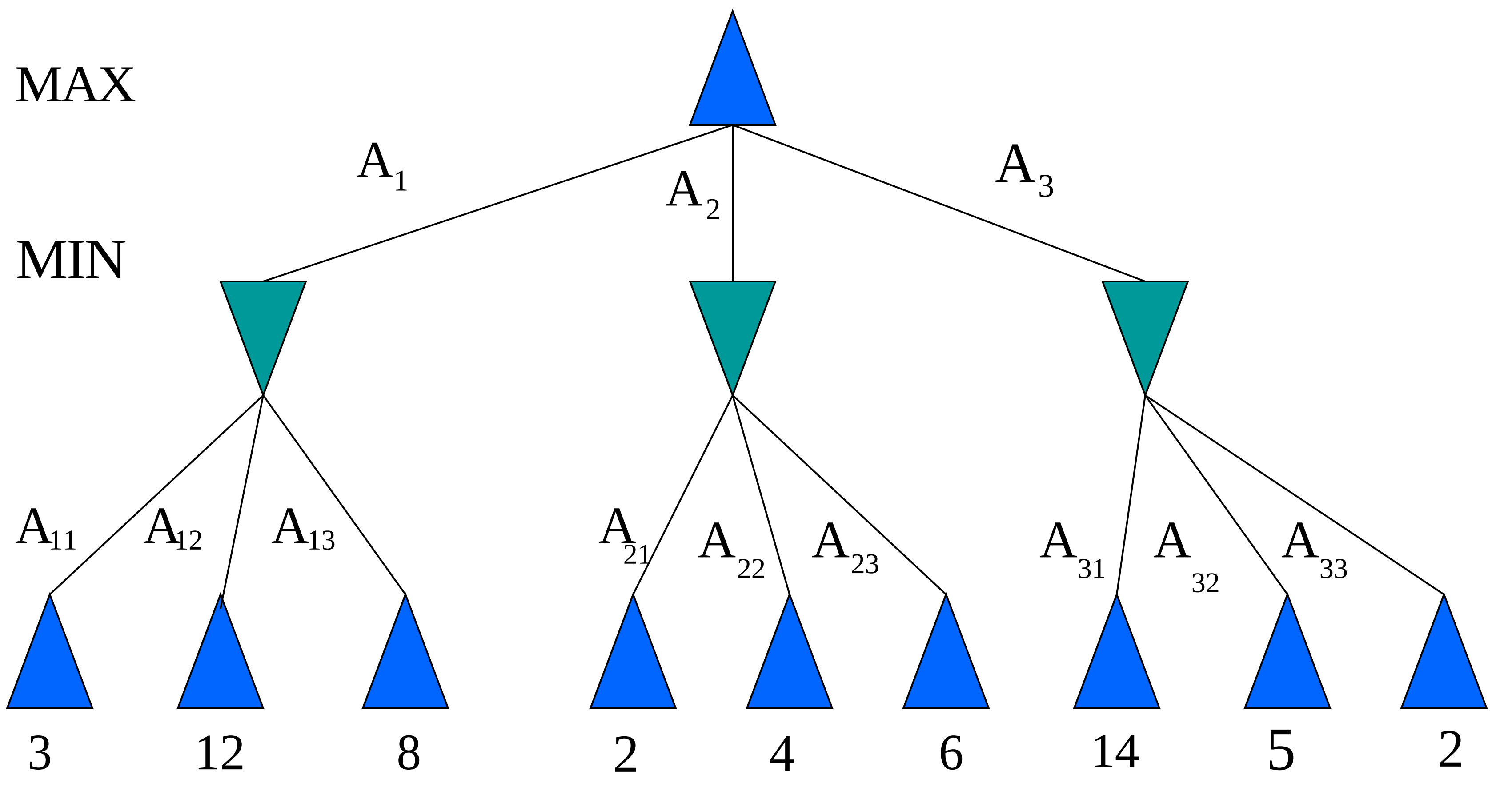
1. Expand the entire tree below the root.

2. Evaluate the terminal nodes as wins for the minimizer or maximizer (i.e. utility).

3. Select an unlabeled node, *n*, all of whose children have been assigned values. If there is no such node, we're done --- return the value assigned to the root.

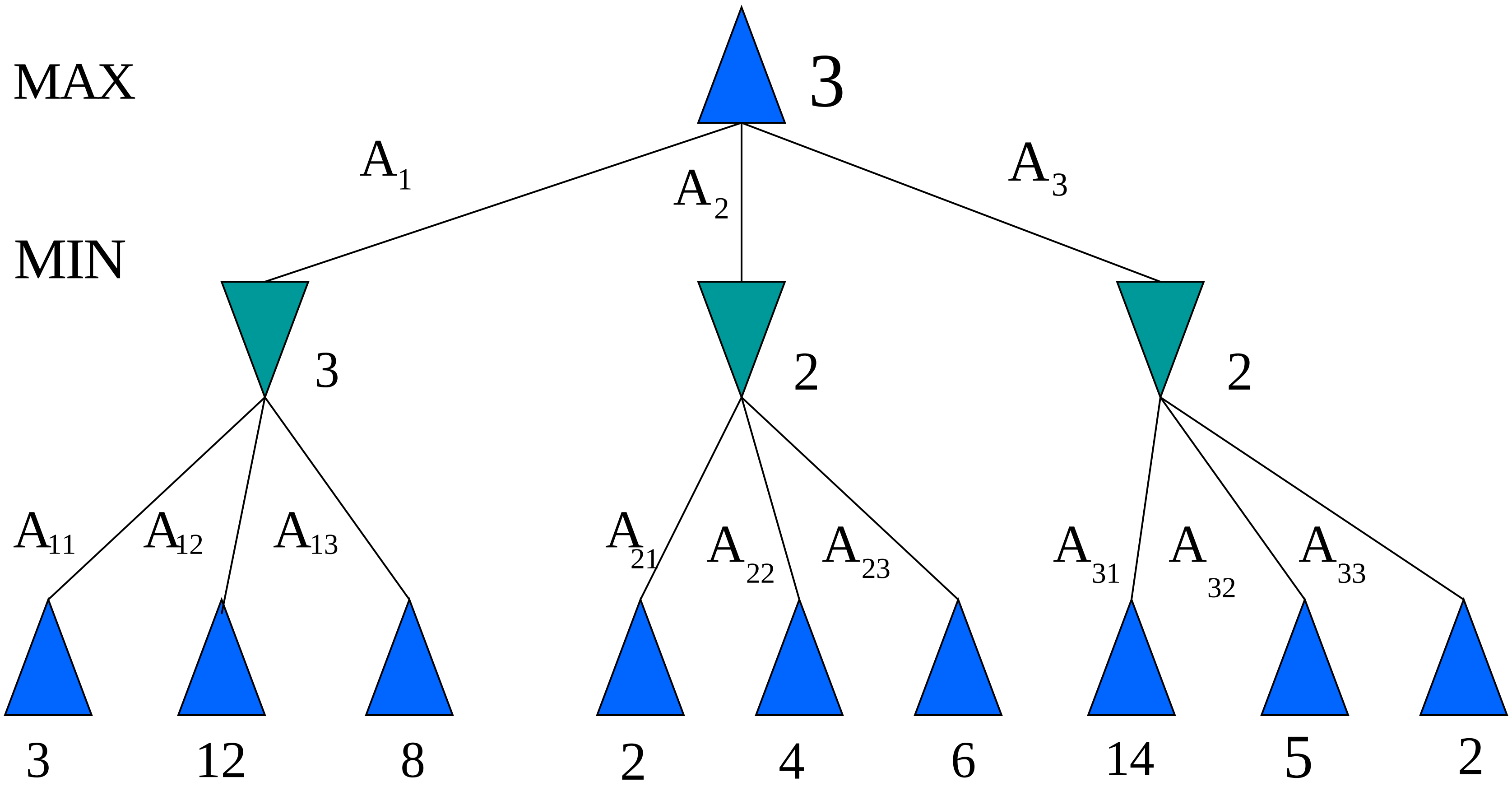
4. If *n* is a minimizer move, assign it a value that is the minimum of the values of its children. If *n* is a maximizer move, assign it a value that is the maximum of the values of its children. Return to Step 3.

Another Example



According to minimax, which action to take? A=A1 B=A2 C=A3

Another Example



Minimax

function MINIMAX-DECISION(*game*) **returns** *an operator*
    for each *op* **in** OPERATORS[game]**do**
        VALUE[*op*]←MINIMAX-VALUE(APPLY(*op,game*),*game*)
    end
    return the *op* with the highest VALUE[*op*]

---

function MINIMAX-VALUE(*state,game*) **returns** *a utility value*
    if TERMINAL-TEST[*game*](*state*) **then**
        return UTILITY[*game*](*state*)
    else if MAX is to move in *state* **then**
        return the highest MINIMAX-VALUE of SUCCESSORS(*state*)
    else
        return the lowest MINIMAX-VALUE of SUCCESSORS(*state*)

Improving Minimax: $\alpha - \beta$ Pruning

Idea: Avoid generating the whole search tree

Approach: Analyze which subtrees have no influence on the solution

$\alpha - \beta$ Search

α = best choice (highest) found so far for max, initially $-\infty$
β = best choice (lowest) found so far for min , initially $+\infty$

We'll call $\alpha - \beta$ procedure recursively with a narrowing range between α and β.

Maximizing levels may reset α to a higher value;
Minimizing levels may reset β to a lower value.

Features of Evolution

Player

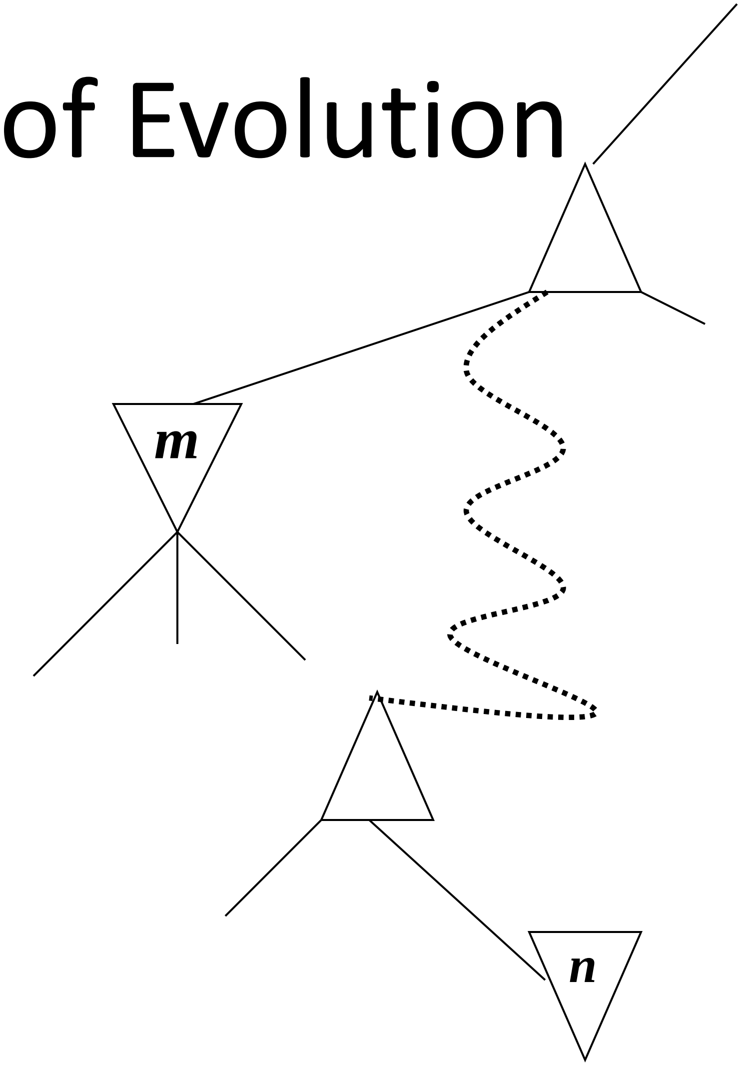
Opponent

..

..

Player

Opponent



If *m* is better than *n* for Player, never get to *n* in play.

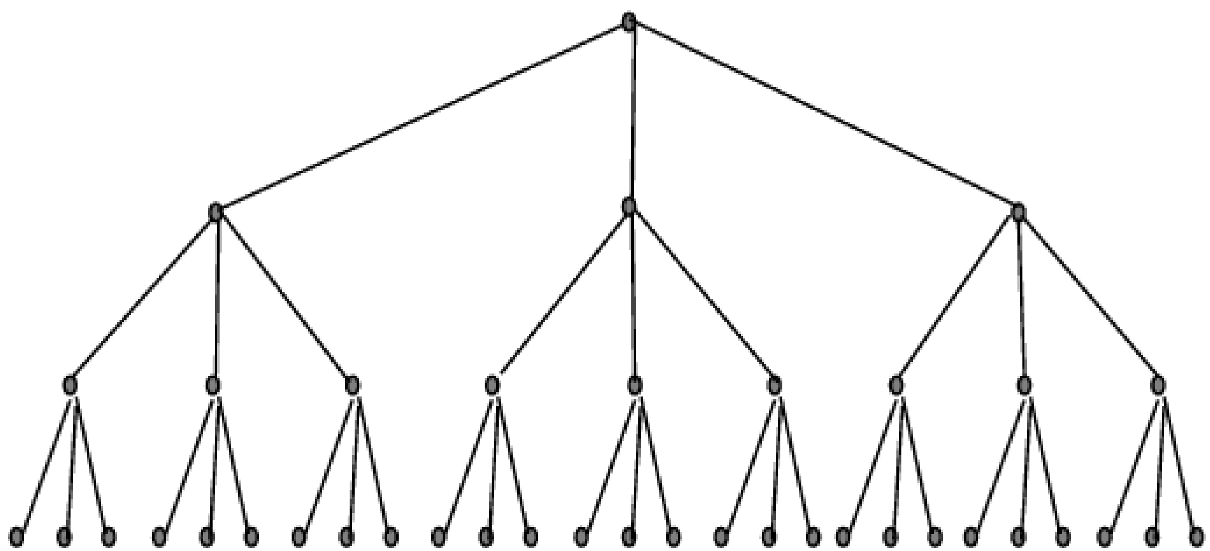
$\alpha - \beta$ Search Algorithm

1. If terminal state, compute e(n) and return the result.
2. Otherwise, if the level is a **minimizing** level,
    - Until no more children or $\beta \leq \alpha$
        - $\upsilon_i \leftarrow \alpha - \beta$ search on a child
        - If $\upsilon_i < \beta, \beta \leftarrow \upsilon_i$.
    - Return $min(\upsilon_i)$
3. Otherwise, the level is a **maximizing** level:
    - Until no more children or $\alpha \geq \beta$,
        - $\upsilon_i \leftarrow \alpha - \beta$ search on a child.
        - If $\upsilon_i > \alpha$, set $\alpha \leftarrow \upsilon_i$
    - Return $max(\upsilon_i)$

Search Space Size Reductions

Worst Case: In an ordering where worst options evaluated first, all nodes must be examined.

Best Case: If nodes ordered so that the best options are evaluated first, then what?



The Need for Imperfect Decisions

Problem: Minimax assumes the program has time to search to the terminal nodes.

Solution: Cut off search earlier and apply a heuristic evaluation function to the leaves.

Static Evaluation Functions

Minimax depends on the translation of board quality into single, summarizing number. Difficult. Expensive.

- Add up values of pieces each player has (weighted by importance of piece).
- Isolated pawns are bad.
- How well protected is your king?
- How much maneuverability to you have?
- Do you control the center of the board?
- Strategies change as the game proceeds.

Design Issues for Heuristic Minimax

Evaluation Function:

Need to be carefully crafted and depends on game! What criteria should an evaluation function fulfill?

Linear Evaluation Functions

- $\boldsymbol{w}_1 \boldsymbol{f}_1 + \boldsymbol{w}_2 \boldsymbol{f}_2 + \ldots + \boldsymbol{w}_n \boldsymbol{f}_n$
- This is what most game playing programs use

- Steps in designing an evaluation function:

    1. Pick informative features.

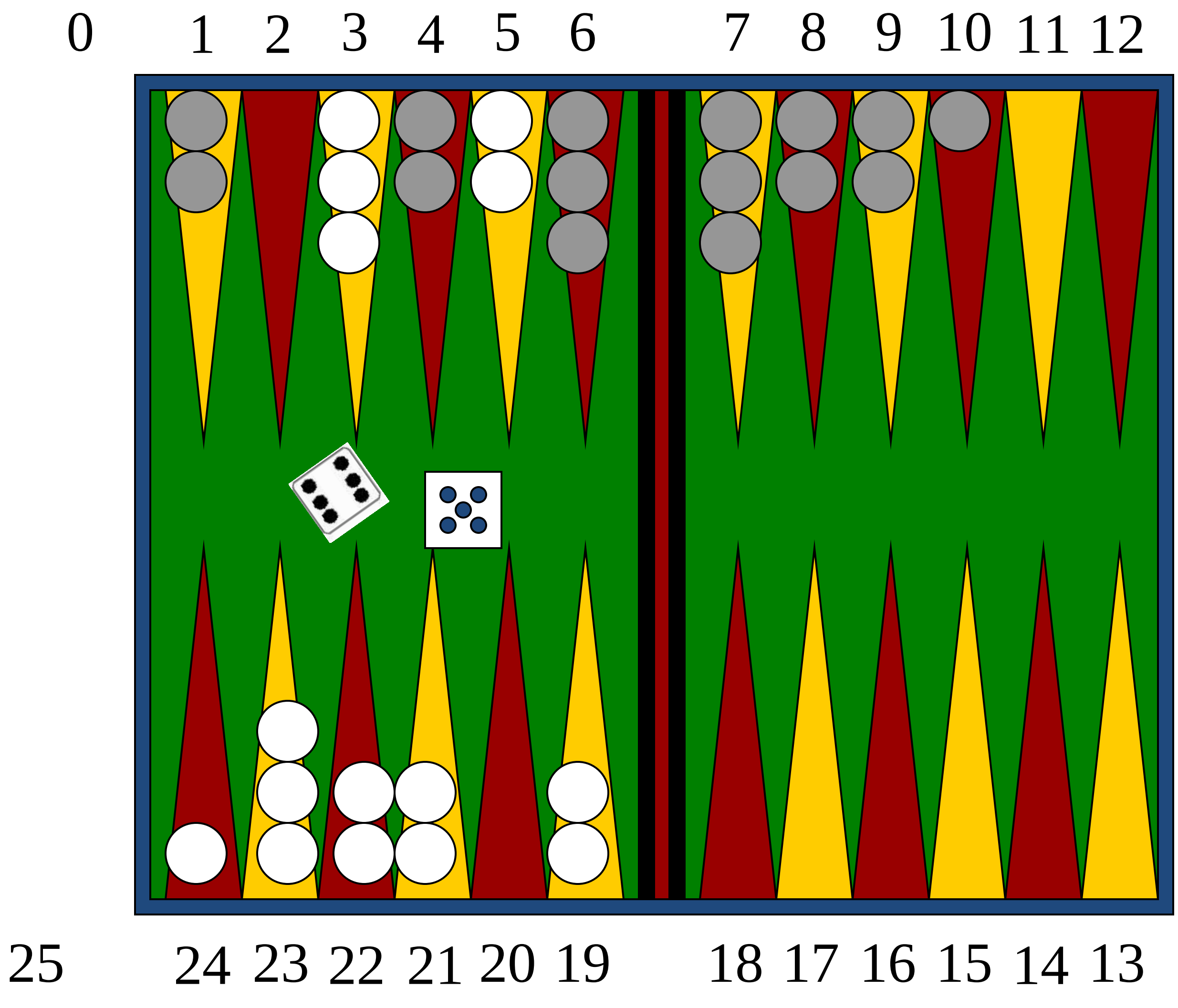
    2. Find the weights that make the program play well

Design Issues for Heuristics Minimax

Search: search to a constant depth

What are problems with constant search depth?

Backgammon Board

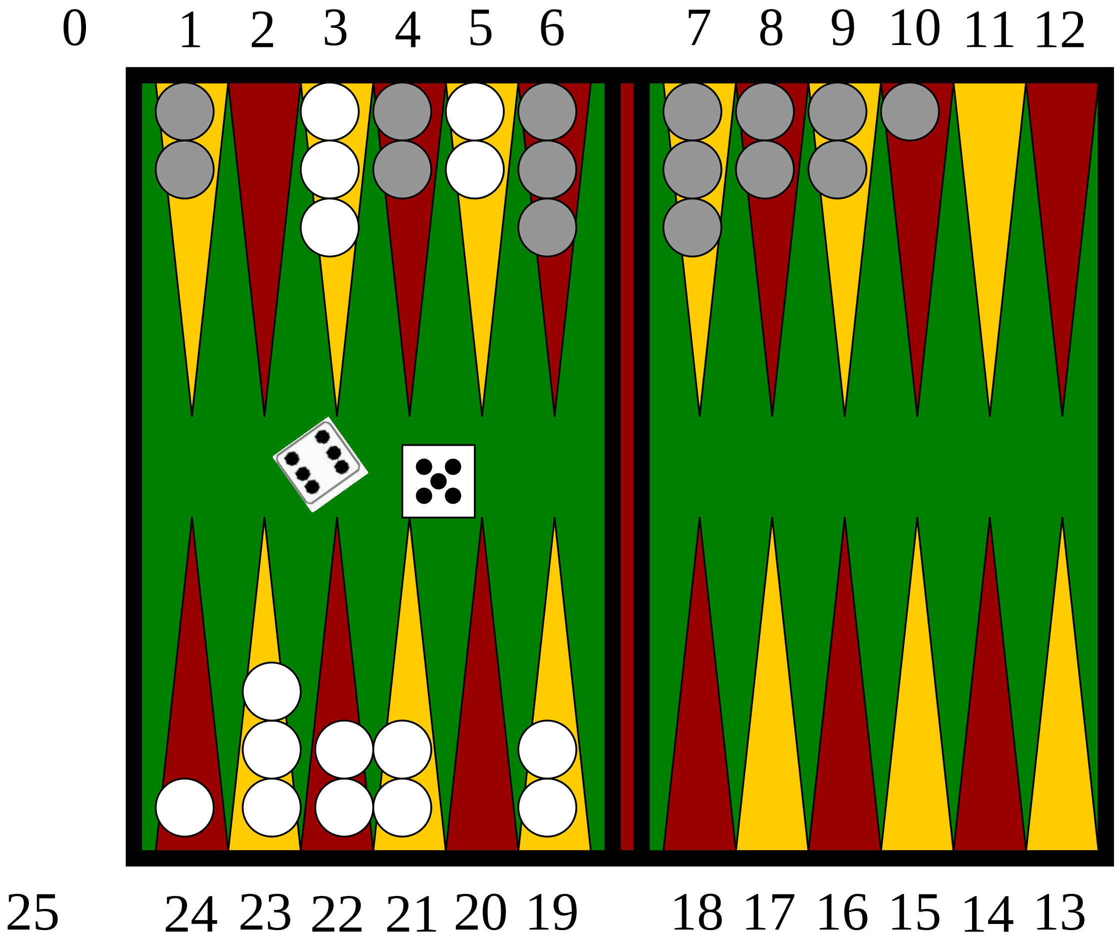


Backgammon - Rules

- Goal: move all of your pieces off the board before your opponent does.
- Black moves counterclockwise toward 0.
- White moves clockwise toward 25.
- A piece can move to any position except one where there are two or more of the opponent's pieces.
- If it moves to a position with one opponent piece, that piece is captured and has to start it's journey from the beginning.

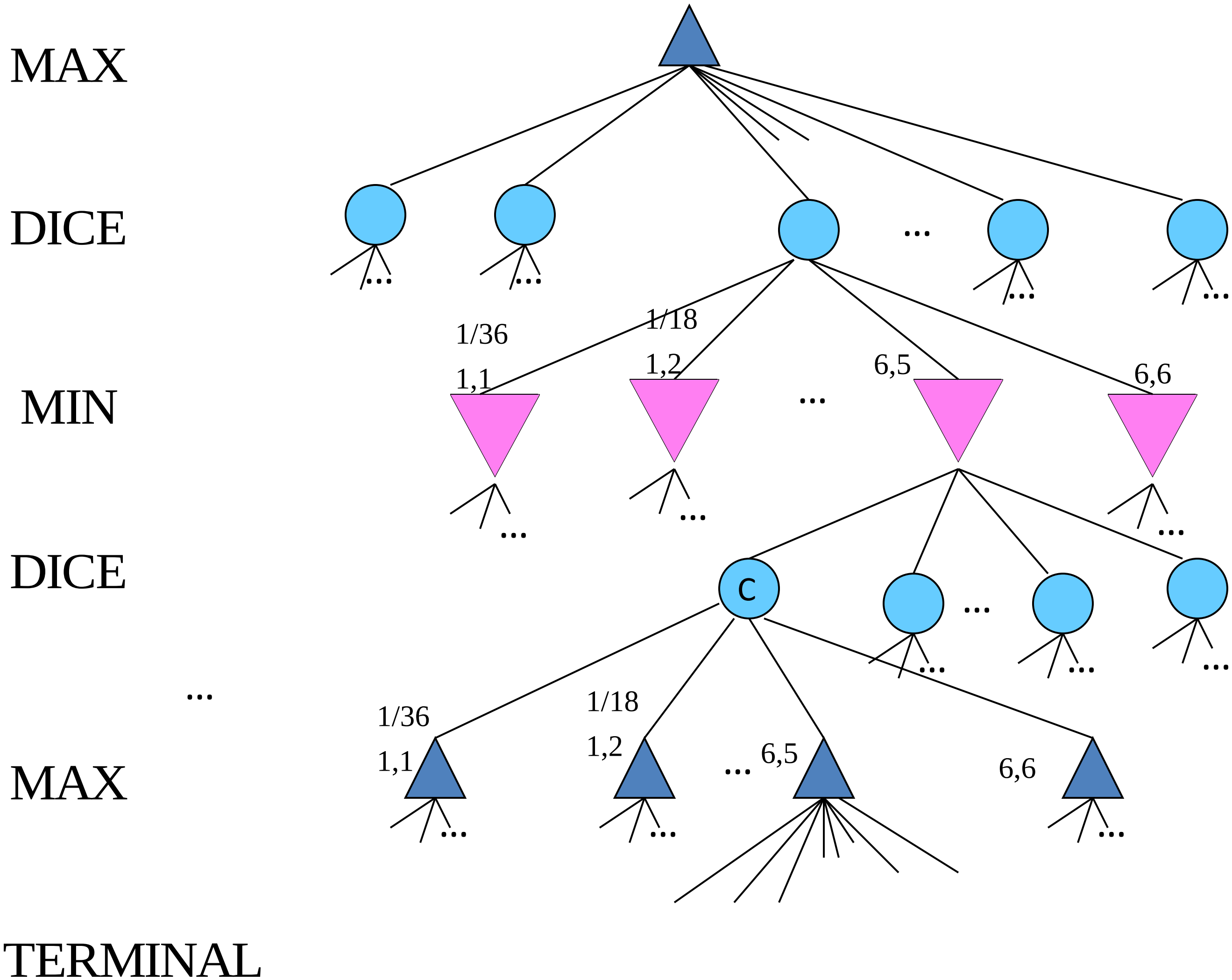
Backgammon - Rules

- If you roll doubles you take 4 moves (example: roll 5,5, make moves 5,5,5,5).
- Moves can be made by one or two pieces (in the case of doubles by 1, 2, 3 or 4 pieces)
- And a few other rules that concern *bearing off* and *forced moves*.



White has rolled 6-5 and has 4 legal moves: (5-10,5-11),
(5-11,19-24), (5-10,10-16) and (5-11,11-16).

Game Tree for Backgammon

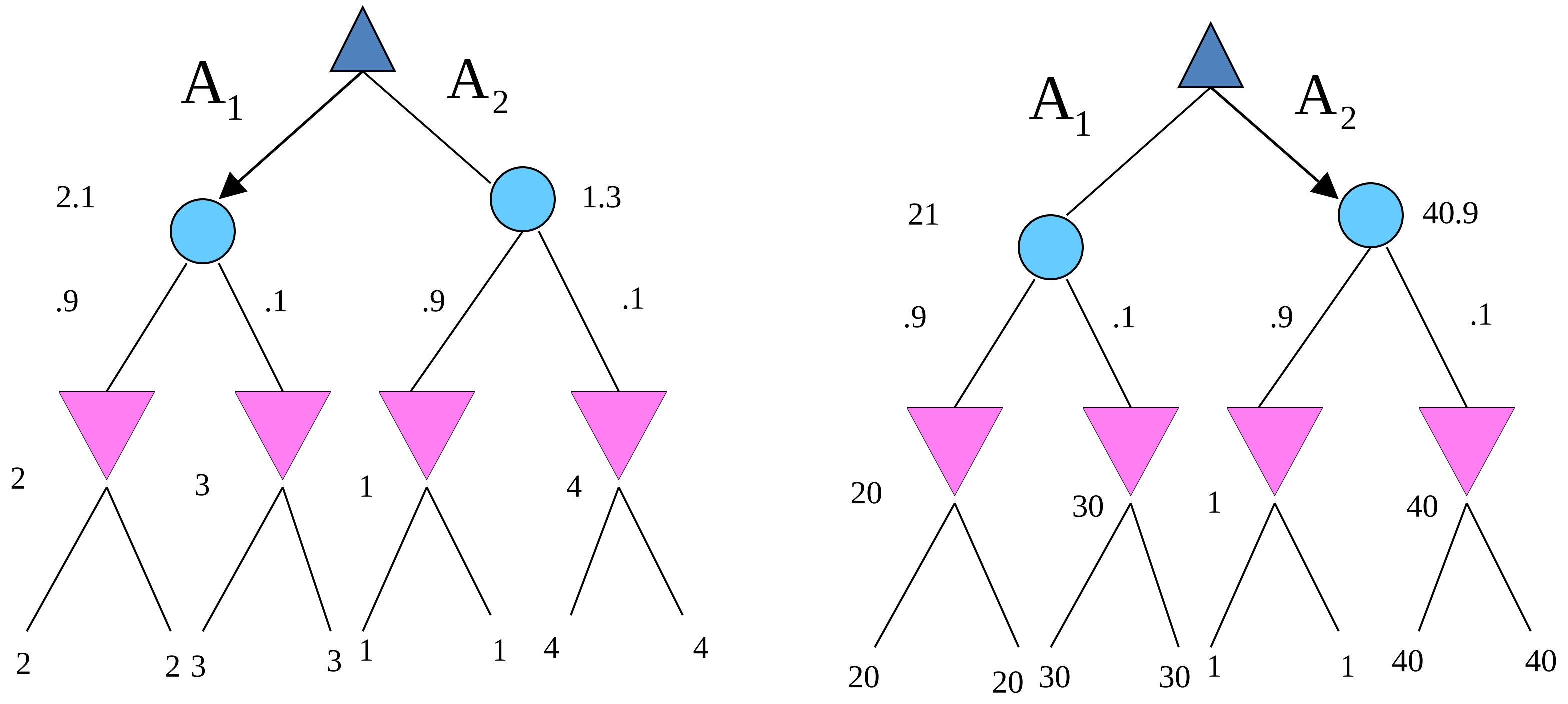


Expectiminimax

Expectiminimax(n) =

Utility(n)	**for n, a terminal state**
$max_{s \in Succ(n)}$ expectiminimax($\boldsymbol{s}$)	**for n, a Max node**
$min_{s \in Succ(n)}$ expectiminimax($\boldsymbol{s}$)	**for n, a Min node**
$\Sigma_{\boldsymbol{s} \in \boldsymbol{Succ}(\boldsymbol{n})} \boldsymbol{P}(\boldsymbol{s}) * \text{expectiminimax}(\boldsymbol{s})$	**for n, a chance node**

Evaluation function



State of the Art in Backgammon

- 1980: *BKG* using two-ply (depth 2) search and lots of luck defeated the human world champion.
- 1992: Tesauro combines Samuel's learning method with neural networks to develop a new evaluation function (search depth 2-3), resulting in a program ranked among the top 3 players in the world.

State of the Art in Checkers

- **1952:** Samuel developed a checkers program that learned its own evaluation function through self play.
- **1990:** *Chinook* (J. Schaeffer) wins the U.S. Open. At the world championship, Marion Tinsley beat *Chinook*.
- **2005:** Schaeffer et al. solved checkers for “White Doctor” opening (draw) (about 50 other openings).

State of the Art in Go

Large branching factor makes regular search methods inappropriate.

Best computer Go programs ranked only “weak amateur”.

Employ pattern recognition techniques and limited search.

$2,000,000 prize available for first computer program to defeat a top level player.

History of Chess in AI

500	**Legal chess**
1200	**Occasional player**
2000	**World-ranked**
2900	**Gary Kasparov**

Early 1950's Shannon and Turing both had programs that (barely) played legal chess (500 rank).

1950's Alex Bernstein's system, (500 + ε)

1957 Herb Simon claims that a computer chess program would be world chess champion in 10 years...yeah, right.

1966 McCarthy arranges computer chess match, Stanford vs. Russia. Long, drawn-out match. Russia wins.

1967 Richard Greenblatt, MIT. First of the modern chess programs, *MacHack* (1100 rating).

1968 McCarthy, Michie, Papert bet Levy (rated 2325) that a computer program would beat him within 10 years.

1970 ACM started running chess tournaments. Chess 3.0-6 (rated 1400).

1973 By 1973...Slate: "It had become too painful even to look at Chess 3.6 any more, let alone work on it."

1973 Chess 4.0: smart plausible-move generator rather than speeding up the search. Improved rapidly when put on faster machines.

1976 Chess 4.5: ranking of 2070.

1977 Chess 4.5 vs.~Levy. Levy wins.

1980's Programs depend on search speed rather than knowledge (2300 range).

1993 DEEP THOUGHT: Sophisticated special-purpose computer; $\alpha - \beta$ search; searches 10-ply; singular extensions; rated about 2600.

1995 DEEP BLUE: searches 14-ply; iterative deepening $\alpha - \beta$ search; considers 100-200 billion positions per move; regularly reaches depth 14; evaluation function has 8000+ features; singular extensions to 40-ply; opening book of 4000 positions; end-game database for 5-6 pieces.

1997 DEEP BLUE: first match won against world-champion (Kasparov). 2002 IBM declines re-match. FRITZ played world champion Vladimir Kramnik. 8 games. Ended in a draw.

Concludes “Search”

- **Uninformed search:** DFS / BFS / Uniform cost search
  - time / space complexity
  - size search space: up to approx. 10^{11} nodes
  - special case: **Constraint Satisfaction / CSPs**
    - generic framework: variables & constraints
    - backtrack search (DFS); propagation
    - (forward-checking / arc-consistency,
    - variable / value ordering

Informed Search: use heuristic function guide to goal

Greedy best-first search

A search / provably optimal*

Search space up to approximately 10^{25}

Local search

Greedy / Hillclimbing

Simulated annealing

Tabu search

Genetic Algorithms / Genetic Programming

search space 10^{100} to 10^{1000}

Aversarial Search / Game Playing

minimax Up to ~10^{10} nodes, 6–7 ply in chess.

alpha-beta pruning Up to ~10^{20} nodes, 14 ply in chess. *provably optimal*

Search and AI

Why such a central role?

- Basically, because lots of tasks in AI are **intractable**. Search is "only" way to handle them.
- Many applications of search, in e.g., Learning / Reasoning / Planning / NLU / Vision
- Good thing: much recent progress (10^{30} quite feasible; sometimes up to 10^{1000}).

Qualitative difference from only a few years ago!