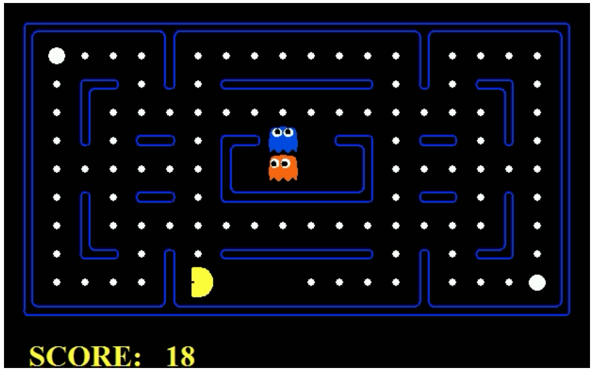
Outline

- Games
- Optimal Decisions in Games
- α-β Pruning
- Imperfect, Real-time Decisions
- Stochastic Games
- Partially Observable Games
- State-of-the-Art Game Programs
- Alternative Approaches

Games

Search in multiagent environments

- Each agent needs to consider the actions of other agents and how they affect its own welfare.
- The unpredictability of other agents introduce contingencies into the agent's problem-solving process



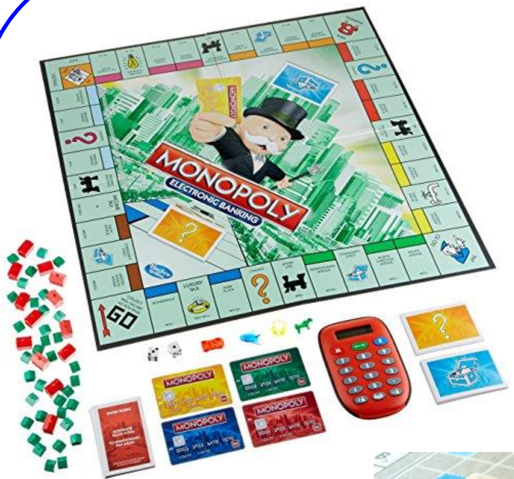
Game theory

- **Game theory** views any multiagent environment as a game.
  - The impact of each agent on the others is “significant,” regardless of whether the agents are cooperative or competitive.
- Types of games

	Deterministic	Chance
Perfect information	Chess, Checkers, Go, Othello	Backgammon, Monopoly
Imperfect information		Bridge, poker, scrabble nuclear war

Types of Games

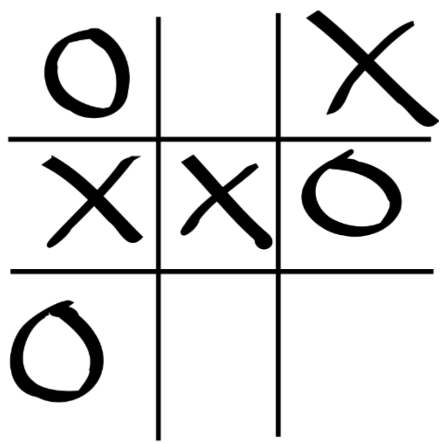




Adversarial search

- **Adversarial search** (known as **games**) covers competitive environments in which the agents' goals are in conflict.
- **Zero-sum games** of perfect information
  - Deterministic, fully observable environments, turn-taking, two-player
  - The utility values at the end are always equal and opposite.



Games vs. Search problems

- Complexity: games are too hard to be solved
  - Chess: $b \approx 35$, $d \approx 100$ (50 moves/player) $\rightarrow$ graph of 10^{40} nodes, search tree of 35^{100} or 10^{154} nodes
  - Go: $b \approx 1000$ (!)
- Time limits: make some decision even when calculating the optimal decision is infeasible
- Efficiency: penalize inefficiency severely
  - Several interesting ideas on how to make the best possible use of time are spawn.

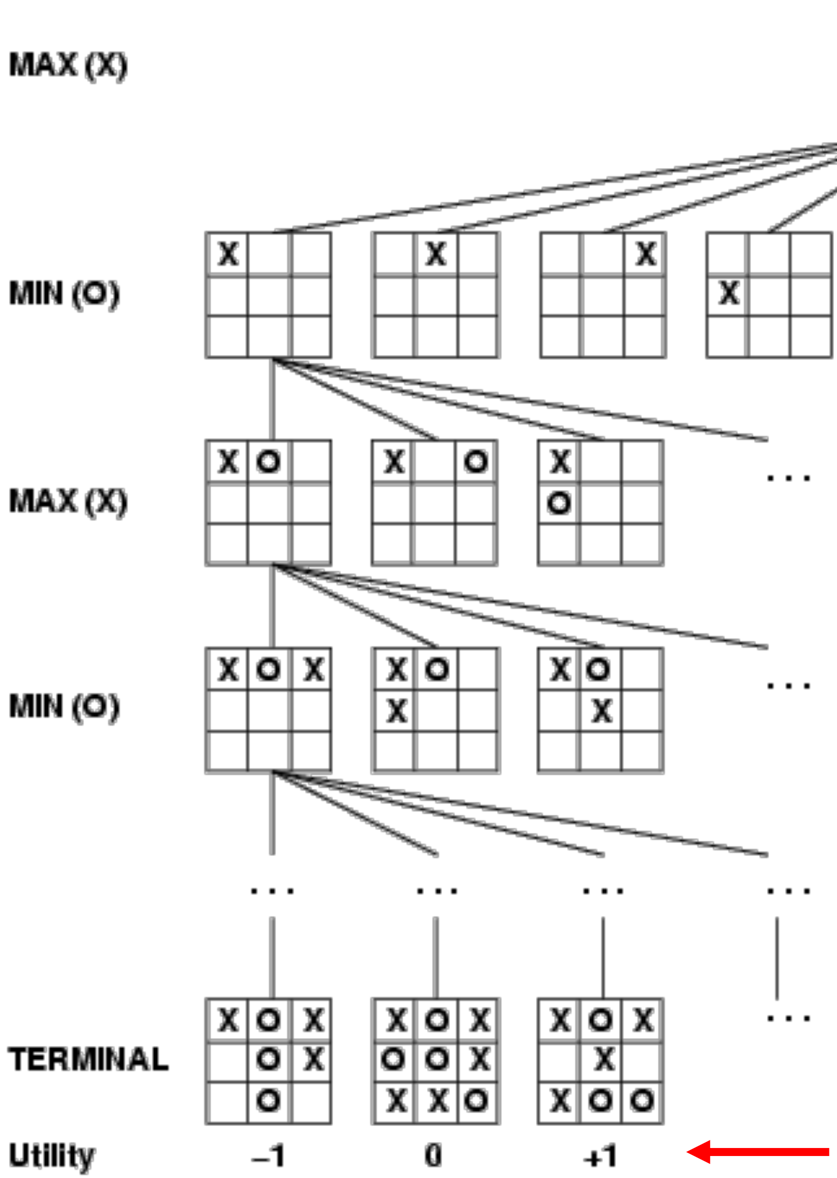
Primary assumptions

- Two players only, called MAX and MIN.
  - MAX moves first, and then they take turns moving until the game ends
  - Winner gets reward, loser gets penalty.
- Both players have complete knowledge of the game's state
  - E.g., chess, checkers and Go, etc. Counter examples: poker
- No element of chance
  - No dice thrown, no cards drawn, etc.
- Zero-sum games
  - The total payoff to all players is the same for every game instance.
- Rational players
  - Each player always tries to maximize his/her utility

Games as search

- S_0 – Initial state: How the game is set up at the start
  - E.g. board configuration of chess
- $\textit{PLAYER}(s)$: Which player has the move in a state, MAX/MIN?
- $\textit{ACTIONS}(s)$ – Successor function: A list of (move, state) pairs specifying legal moves.
- $\textit{RESULT}(s, a)$ – Transition model: Result of move a on state s
- $\textit{TERMINAL} - \textit{TEST}(s)$: Is the game finished?
  - States where the game has ended are called **terminal states**
- $\textit{UTILITY}(s, p)$ – Utility function: A numerical value of a terminal state s for a player p
  - E.g. chess: win (**+1**), lose (**0**) and draw (**1/2**), backgammon: [0, 192]

The game tree of Tic-Tac-Toe



MAX uses search tree to determine next move.

from the point of view of MAX

Examples of game: Checkers

- Complexity
  - ~ 10^{18} nodes, which may require 100k years with 106 positions/sec
- **Chinook** (1989-2007)
  - The first computer program that won the world champion title in a competition against humans
  - 1990: won 2 games in competition with world champion Tinsley (final score: 2-4, 33 draws)
  - 1994: 6 draws
- Chinook's search
  - Ran on regular PCs, played perfectly by using alpha-beta search combining with a database of 39 trillion endgame positions

Examples of game: Chess

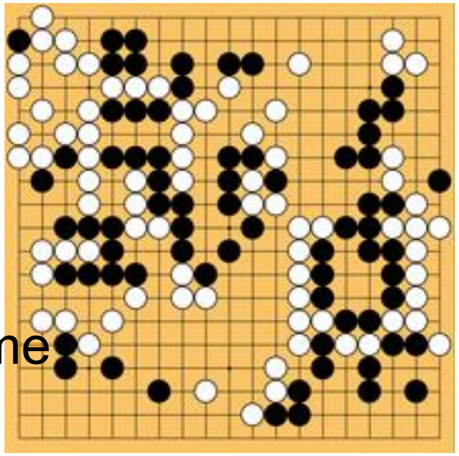
- Complexity
  - b ≈ 35, d ≈ 100, 10^{154} nodes (!!)
  - Completely impractical to search this
- **Deep Blue** (May 11, 1997)
  - Kasparov lost a 6-game match against IBM's Deep Blue (1 win Kasp – 2 wins DB) and 3 ties.
- In the future, focus will be to allow computers to **LEARN** to play chess rather than being **TOLD** how it should play

Deep Blue

- Ran on a parallel computer with 30 IBM RS/6000 processors doing alpha–beta search
- Searched up to 30 billion positions/move, average depth 14 (be able to reach to 40 plies)
- Evaluation function: 8000 features
  - highly specific patterns of pieces (~4000 positions)
  - 700,000 grandmaster games in database
- Working at 200 million positions/sec, even Deep Blue would require **10^{100}** years to evaluate all possible games.
  - (The universe is only 10^{10} years old.)
- Now: algorithmic improvements have allowed programs running on standard PCs to win World Computer Chess Championships.
  - Pruning heuristics reduce the effective branching factor to less than 3

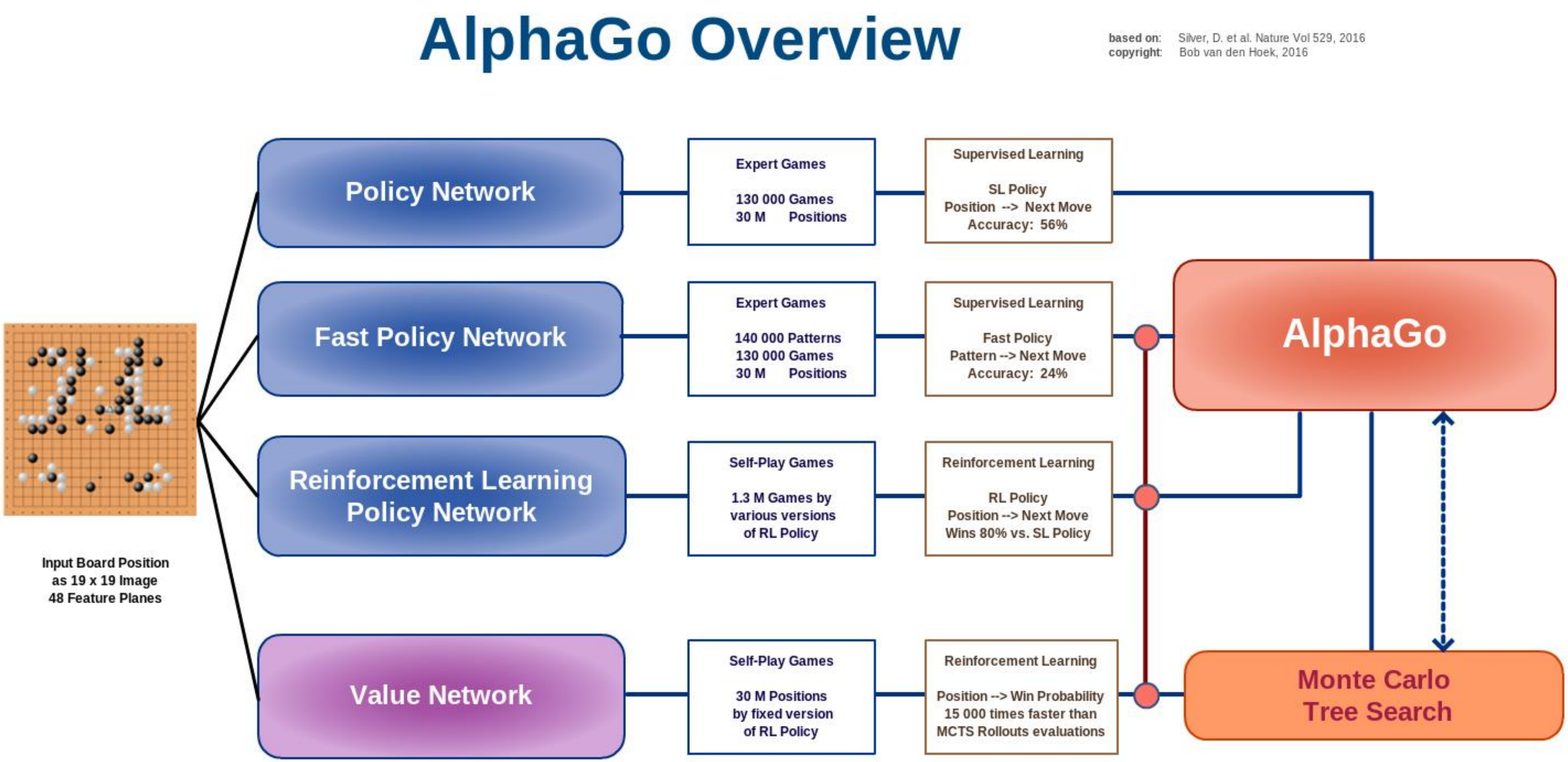
GO

1 million trillion trillion trillion trillion more configurations than chess!



- Complexity
  - Board of 19x19, $b \approx 361$, average depth ≈ 200
  - 10^{174} possible board configuration.
  - Control of territory is unpredictable until the endgame.
- **AlphaGo** (2016) by Google
  - Beat 9-dan professional Lee Sedol (4-1)
  - Machine learning + Monte Carlo search guided by a "value network" and a "policy network" (implemented using *deep neural network* technology)
  - Learn from human + Learn by itself (self-play games)

AlphaGo overview



Optimal Decisions in Games

- *The Minimax Algorithm*
- *Optimal Decisions in Multiplayer Games*

Optimal decision in games

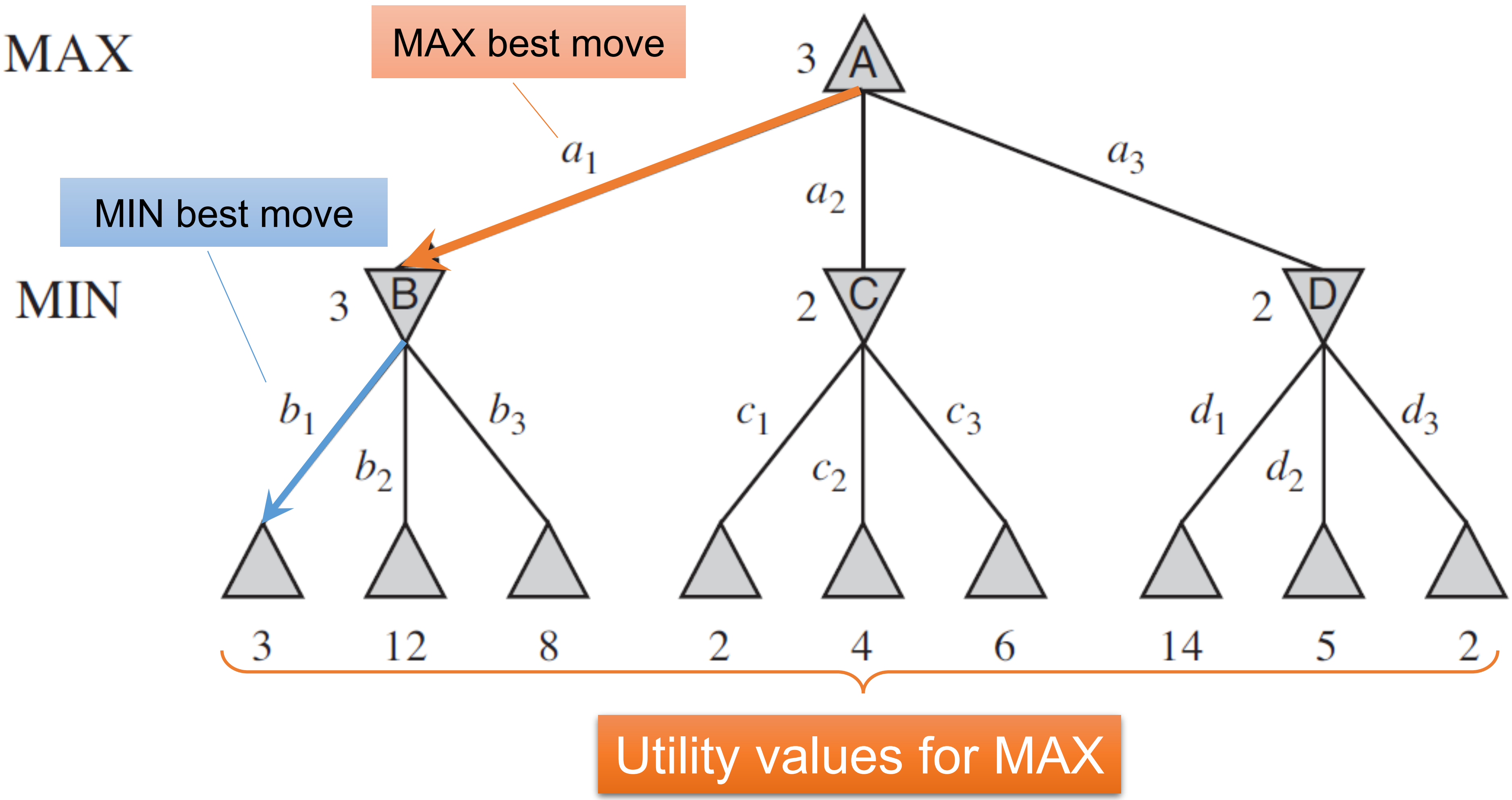
- Normal search problem
  - Optimal solution is a sequence of action leading to a goal state.
- Games
  - A search path that guarantee win for a player
  - The optimal strategy can be determined from the **minimax value** of each node

For MAX

$$\text{MINIMAX}(s) = \begin{cases} \boxed{\text{UTILITY}(s)} & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

Assuming that both players play optimally from there to the end of the game

An example of two-ply game tree



The minimax algorithm

- Compute the **minimax decision** from the current state
- Use a simple recursive computation of the minimax values of each successor state
  - The recursion proceeds all the way down to the leaves of the tree, and then the minimax values are **backed up** through the tree as the recursion unwinds.

The minimax algorithm

function MINIMAX-DECISION(*state*) **returns** an action
  return arg $\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(*state*, *a*))

---

function MAX-VALUE(*state*) **returns** a utility value
  if TERMINAL-TEST(*state*) **then return** UTILITY(state)
  $v \leftarrow -\infty$
  for each *a* **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

---

function MIN-VALUE(*state*) **returns** a utility value
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for each *a* **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s, a)))
  return v

The minimax algorithm

- A complete depth-first exploration of the game tree
- Completeness
  - Yes (if tree is finite)
- Optimality
  - Yes (against an optimal opponent)
- Time complexity
  - $O(b^m)$
- Space complexity
  - $O(bm)$ (depth-first exploration)

Note:

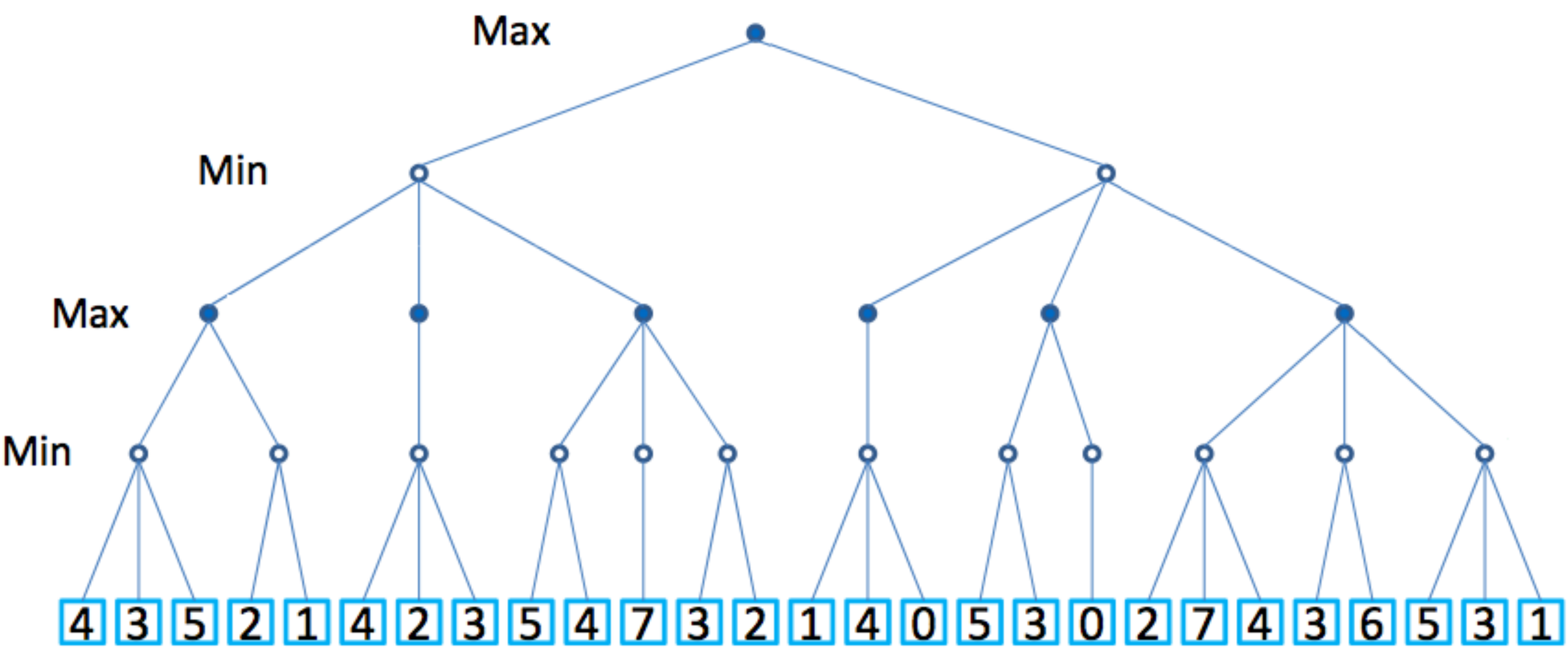
m: the maximum depth of the tree

b: the legal moves at each point

For chess, $b \approx 35, m \approx 100$ for "reasonable" games
$\rightarrow$ exact solution completely infeasible

Quiz 01: Minimax algorithm

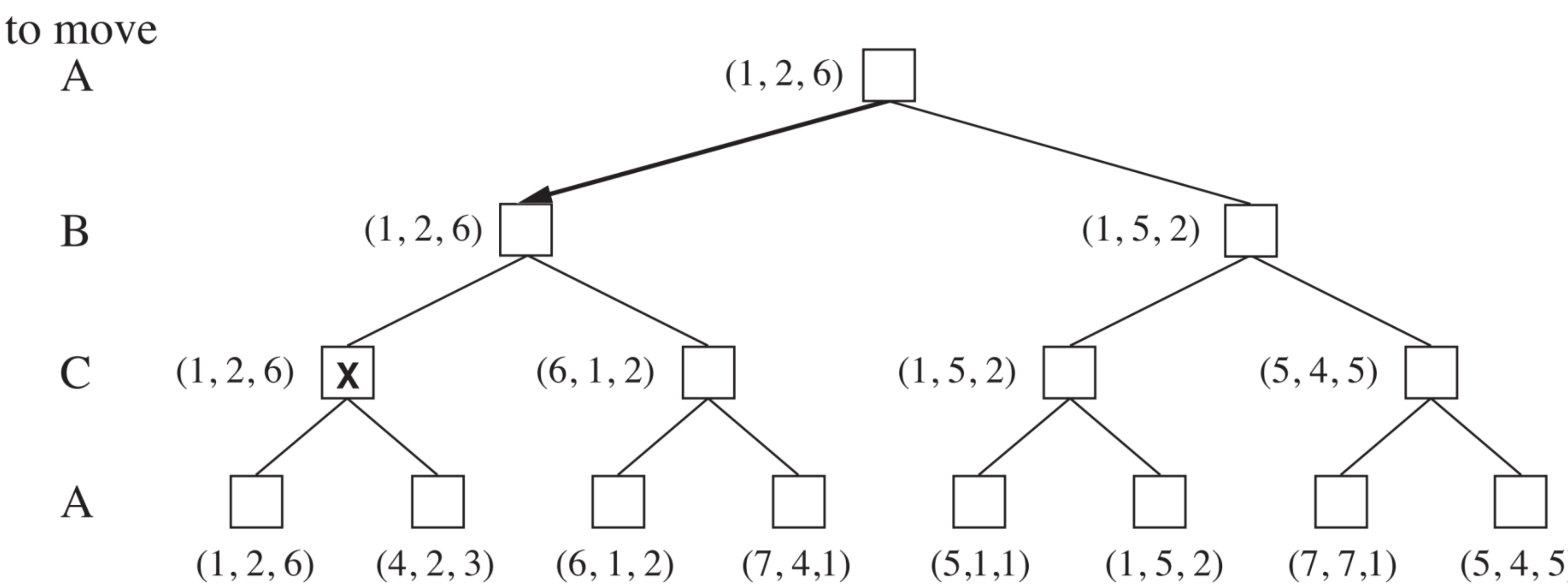
- Calculate the utility value for the remaining nodes
- Which node should MAX and MIN choose?



Optimality in multiplayer games

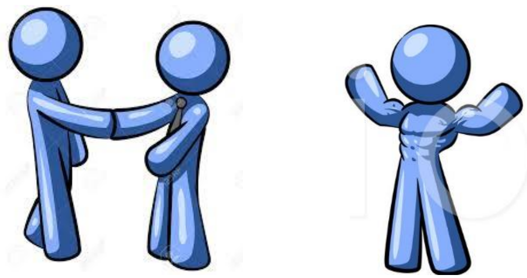
- A single value is replaced with a vector of values.

  → the UTILITY function return a vector of utilities
- For terminal states, this vector gives the utility of the state from each player's viewpoint.

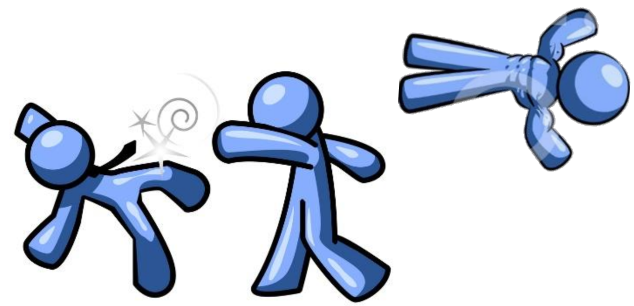


Optimality in multiplayer games

- Multiplayer games usually involve alliances, which are made and broken as the game proceeds.



A and B are weak while C is strong.
A forms an alliance with B.



C becomes weak.
A or B could violate the agreement

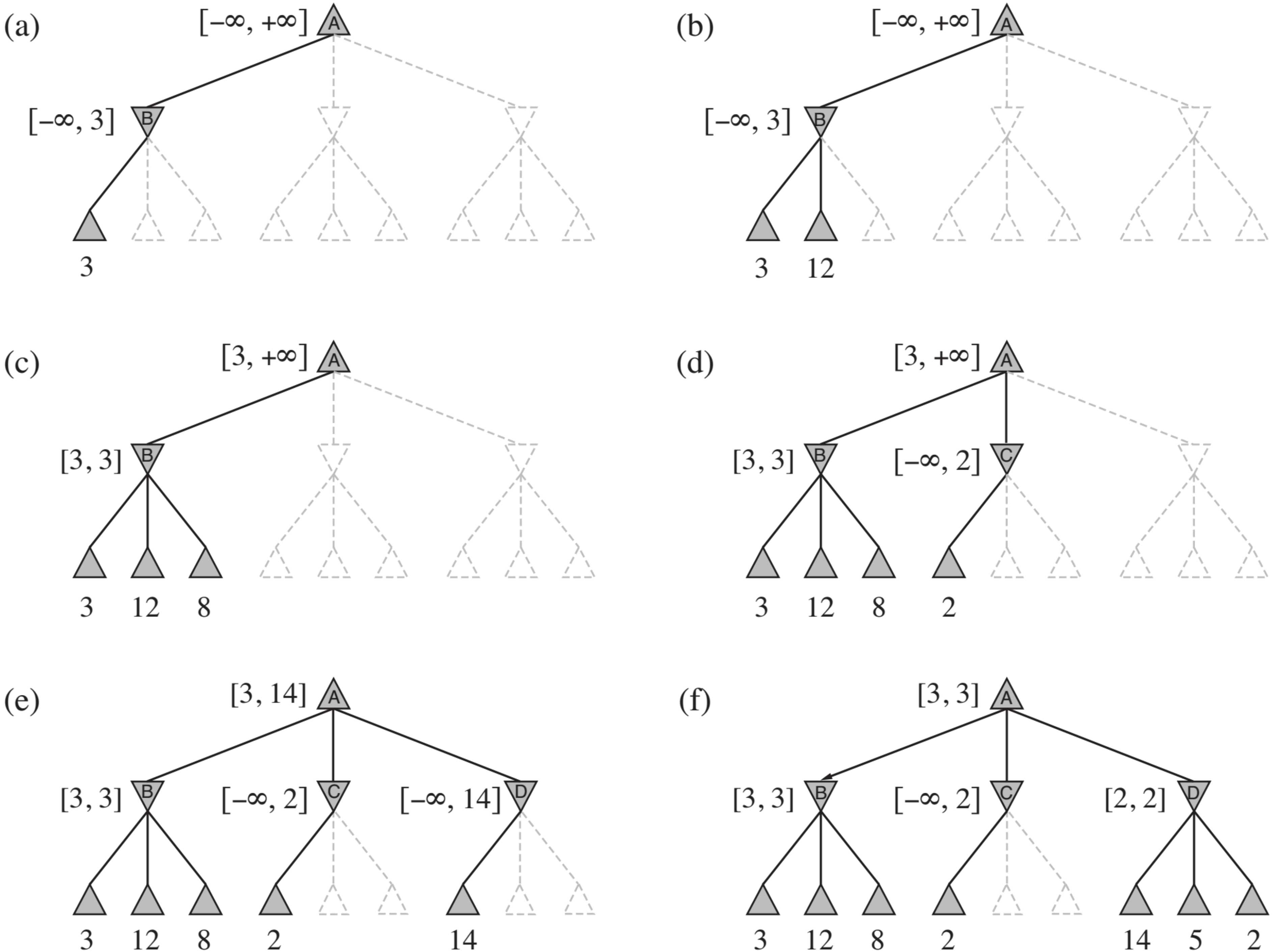
- If the game is not zero-sum, then collaboration can also occur with just two players.

Alpha-Beta Pruning

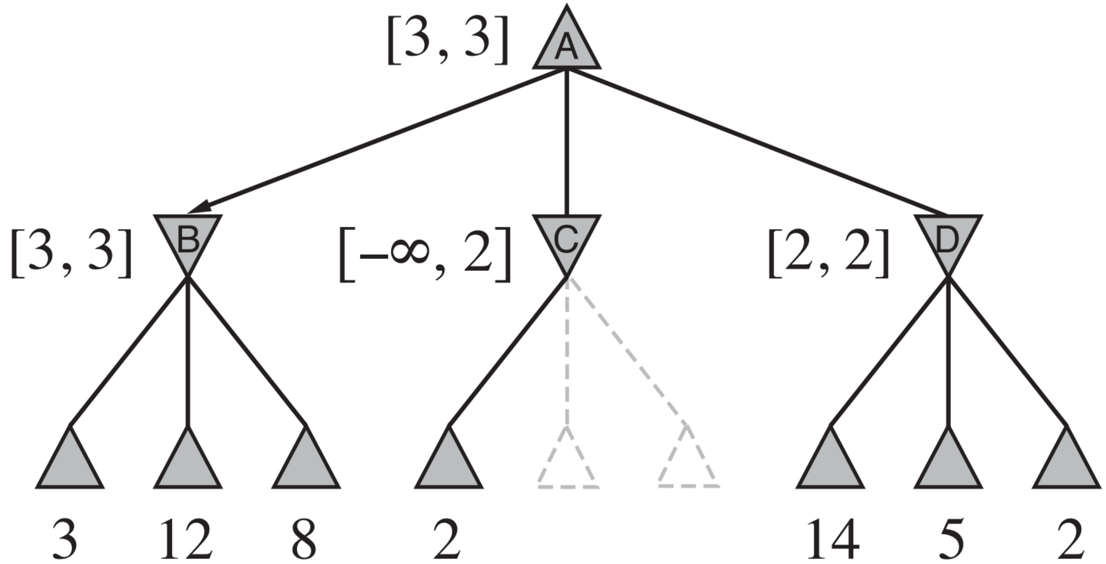
- *Move Ordering*

Problem with minimax search

- The number of game states is exponential in the tree's depth
  $\rightarrow$ Do not examine every node
- **Alpha-beta pruning:** Prune away branches that cannot possibly influence the final decision
- **Bounded lookahead**
  - Limit depth for each search
  - This is what chess players do: look ahead for a few moves and see what looks best



(a)
[−∞, +∞]
A
[−∞, 3]
B
3
(b)
[−∞, +∞]
A
[−∞, 3]
B
3
12
(c)
[3, +∞]
A
[3, 3]
B
3
12
8
(d)
[3, +∞]
A
[3, 3]
B
[−∞, 2]
C
3
12
8
2
(e)
[3, 14]
A
[3, 3]
B
[−∞, 2]
C
[−∞, 14]
D
3
12
8
2
14
(f)
[3, 3]
A
[3, 3]
B
[−∞, 2]
C
[2, 2]
D
3
12
8
2
14
5
2



Another way to look at this is as a simplification of the formula for MINIMAX.

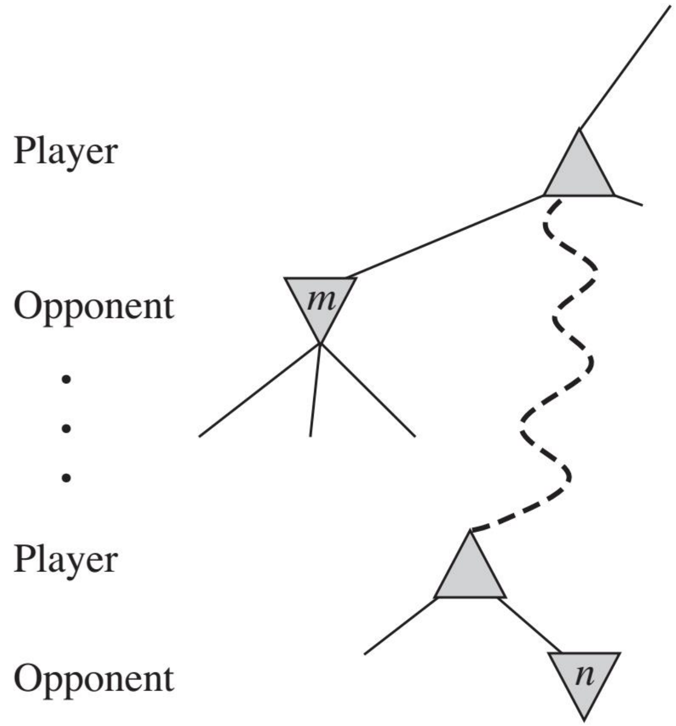
Let the two unevaluated successors of node C have values x and y.

Then the value of the root node is given by

$$
\begin{aligned}
\textsc{Minimax}(root) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\
&= \max(3, \min(2,x,y), 2) \\
&= \max(3, z, 2) \qquad \text{where } z = \min(2,x,y) \leq 2 \\
&= 3.
\end{aligned}
$$

Alpha-beta pruning

- If a move n is determined to be worse than move m that has already been examined and discarded, then examining move n once again is pointless.



α = the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.

β = the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.

The alpha-beta search algorithm

function ALPHA-BETA-SEARCH(*state*) **returns** an action
  $v \leftarrow$ 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*)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a),α,β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
return v

The alpha-beta search algorithm

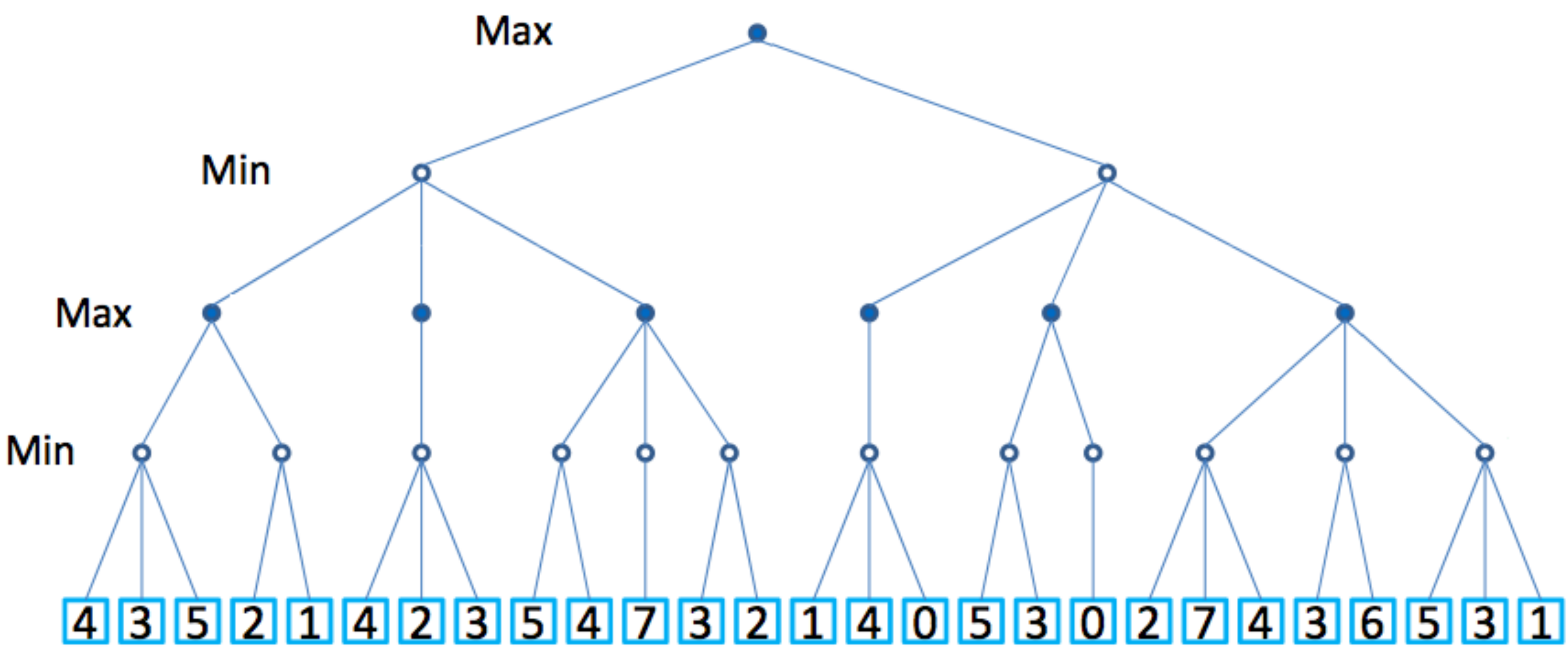
function MIN-VALUE(*state*,α,β) **returns** a utility value
  if TERMINAL-TEST(*state*) **then return** UTILITY(state)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS(state) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) ,α,β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

Properties of alpha-beta pruning

- Pruning does not affect the result
  - Its worst case is as good as the minimax algorithm
- Good move ordering improves effectiveness of pruning
  - With "perfect ordering,“: time complexity $O(b^{m/2}) \to$ x2 search depth
  - In chess, Deep Blue achieved depth reduction from 38 to 6
- Killer move heuristic
  - First, IDS search with 1 ply deep and record the best path.
  - Then search 1 ply deeper with the recorded path to inform move ordering
- Transposition table avoids re-evaluation a state

Quiz 02: Alpha-beta pruning

- Calculate the utility value for the remaining nodes.
- Which nodes should be pruned?



Imperfect Real-Time Decisions

- *Evaluation Functions*
- *Cutting off Search*
- *Forward Pruning*
- *Search versus Lookup*

Heuristic minimax

- Both minimax and alpha-beta pruning search all the way to terminal states.
  - This depth is usually **impractical** because moves must be made in a reasonable amount of time (~ minutes).
- Cut off the search earlier with some depth limit
- Use an evaluation function
  - An estimation for the desirability of position (win, lose, tie?)

$$
\text{H-MINIMAX}(s,d) = \begin{cases} \boxed{\text{EVAL}(s) \qquad \text{if CUTOFF-TEST}(s,d)} \\ \max_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MIN}. \end{cases}
$$

Evaluation functions

- The evaluation function should order the terminal states in the same way as the true utility function does
  - States that are wins must evaluate better than draws, which in turn must be better than losses.
- The computation must not take too long!
- For nonterminal states, their orders should be strongly correlated with the actual chances of winning.

Evaluation functions

- For chess, typically linear weighted sum of features

$$Eval(s) = w_1 f_1\,(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

  - where f_i could be the numbers of each kind of piece on the board, and w_i could be the values of the pieces
  - E.g., $Eval(s) = 9q + 5r + 3b + 3n + p$
- Implicit strong assumption: the contribution of each feature is independent of the values of the other features.
  - E.g., assign the value 3 to a bishop ignores the fact that bishops are more powerful in the endgame → **Nonlinear combination**

Cutting off search

- *Minimax Cutoff* is identical to *Minimax Value* except
  1. $Terminal?$ is replaced by $Cutoff?$
  2. $Utility$ is replaced by $Eval$

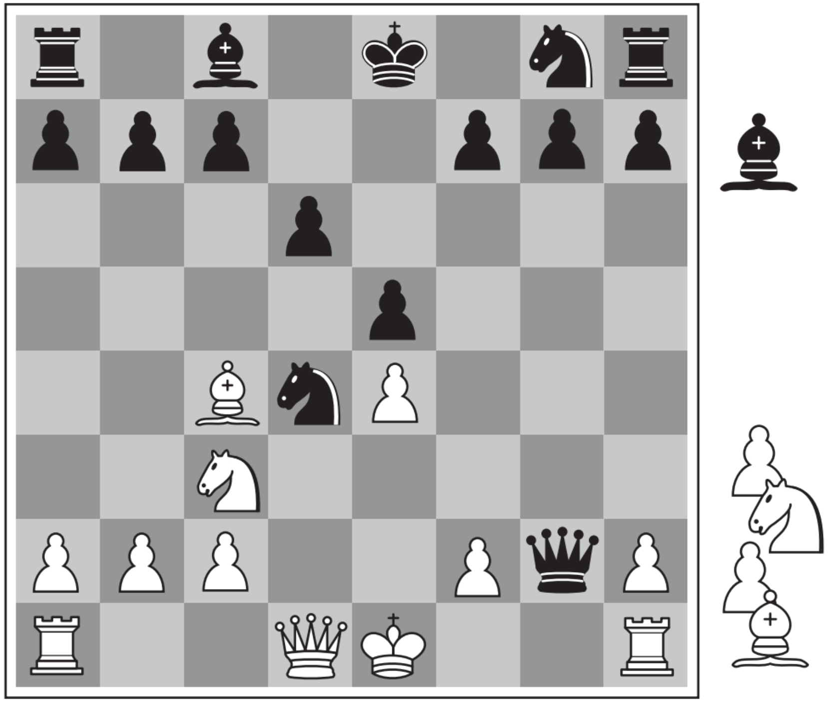
if CUTOFF-TEST(state, depth) **then return** EVAL(state)

- Does it work in practice?
  - $b^m = 10^6, b = 35 \rightarrow m = 4$
  - 4-ply lookahead is a hopeless chess player!
  - 4-ply ≈ human novice, 8-ply ≈ typical PC, human master, 12-ply ≈ Deep Blue, Kasparov

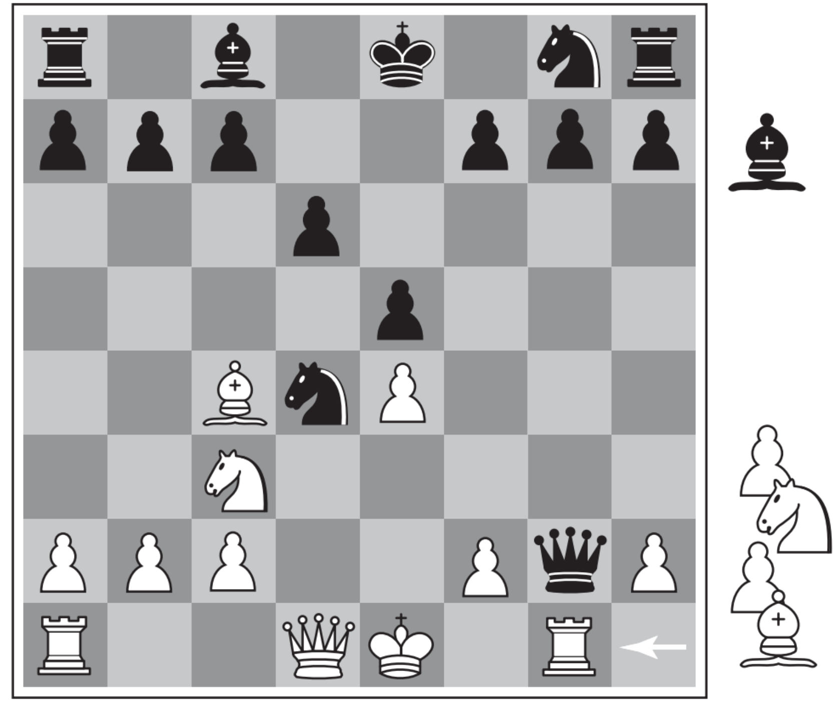
A more sophisticated cutoff test

- Quiescent positions are those unlikely to exhibit wild swings in value in the near future.
  - E.g., in chess, positions in which favorable captures can be made are not quiescent for an evaluation function counting material only
- **Quiescence search:** Expand nonquiescent positions until quiescent positions are reached.

Quiescent positions: An example



(a) White to move

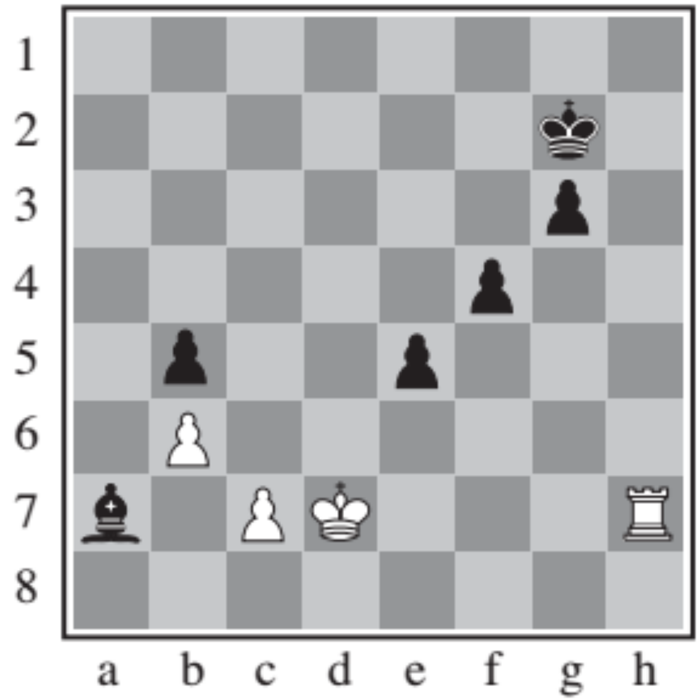


(b) White to move

Two chess positions that differ only in the position of the rook at lower right.
In (a), Black has an advantage of a knight and two pawns, which should be enough to win the game. In (b), White will capture the queen, giving it an advantage that should be strong enough to win.

A more sophisticated cutoff test

- Horizon effect: The program is facing an evitable serious loss and temporarily avoid it by delaying tactics.



With Black to move, the black bishop is surely doomed. But Black can forestall that event by checking the white king with its pawns, forcing the king to capture the pawns.

A more sophisticated cutoff test

- Singular extension: a move that is "clearly better" than all other moves in a given position.
- The algorithm allows for further consideration on a legal singular extension.
  - The tree will be deeper, but there are only a few singular extensions.

Other improvements

- Beam search
  - Forward pruning, consider only a "beam" of the n best moves only
  - Most humans consider only a few moves from each position
  - PROBCUT, or probabilistic cut, algorithm (Buro, 1995)
- Search vs. lookup
  - Use table lookup rather than search for the opening and ending