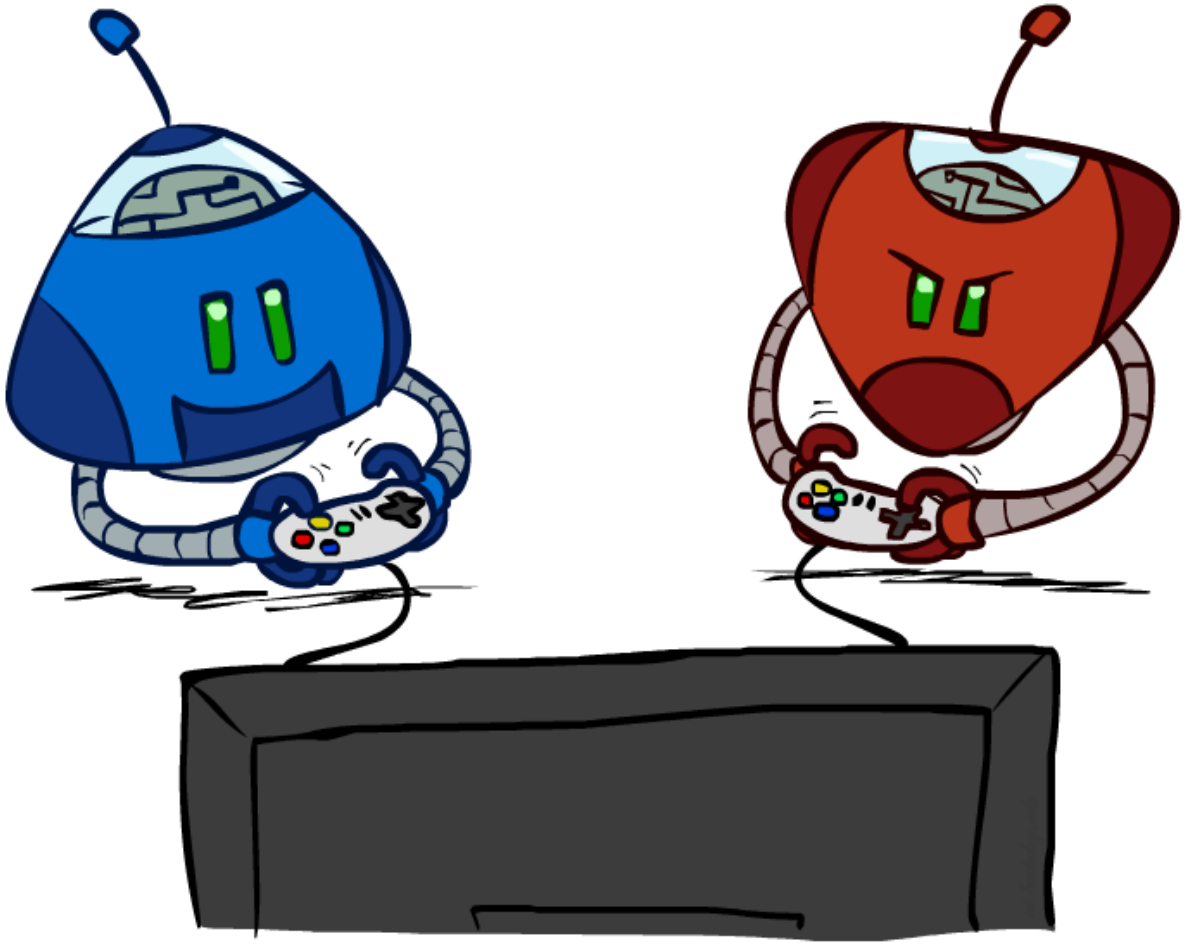
Game Playing

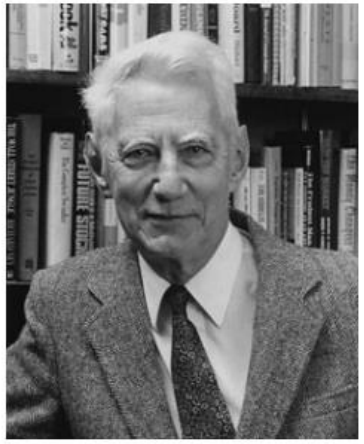
Artificial Intelligence



Contributors

John Von Neumann
Min-max theorem

John McCarthy
Alpha-Beta Pruning



Claude Shannon
Min-max depth cutoff

D. G. Prinz, Alan Turing
Chess Program

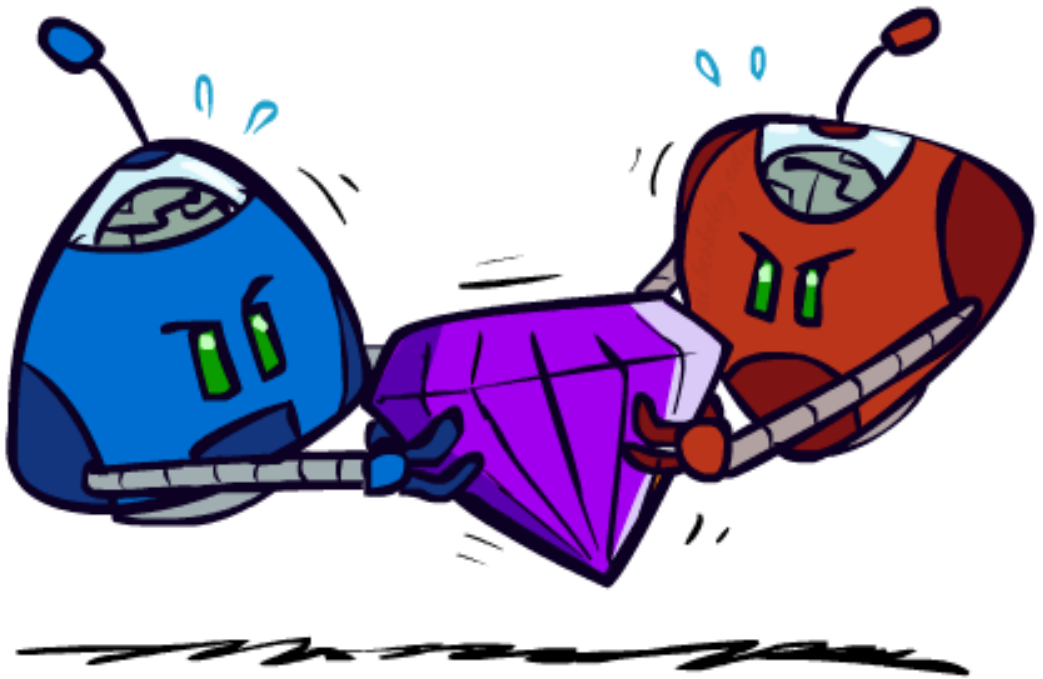
Donald Knuth
Alpha-Beta Analysis

IBM Deep Blue

Why Game

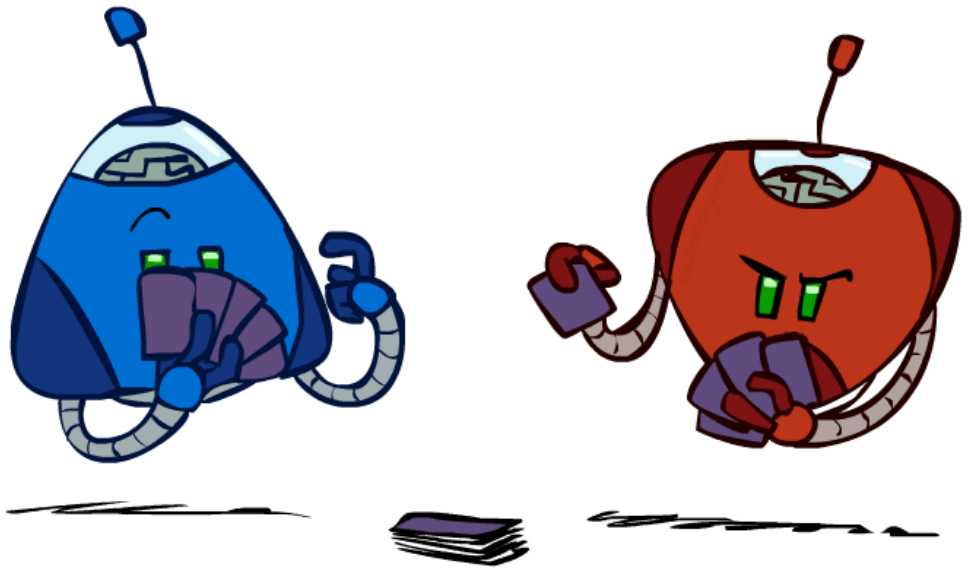
- Why games?
  - Games like Chess or Go are compact settings that mimic the uncertainty of interacting with the natural world involving other rational agents
  - For centuries humans have used them to exert their intelligence
  - Recently, there has been great success in building game programs that challenge human supremacy
- Games vs. Search Problems
  - "Unpredictable" opponent
    - Solution is a strategy
    - Specifying a move for every possible opponent reply
  - Time limits
    - Generally intractable search space, so agents cannot enumerate all possible winning moves
    - Unlikely to find goal, must approximate

Adversarial Games



Types of Games

- Many different kinds of games!



- Axes:
  - Deterministic or stochastic?
  - One, two, or more players?
  - Zero sum?
  - Perfect information (can you see the state)?

- Want algorithms for calculating a strategy (policy) which recommends a move from each state

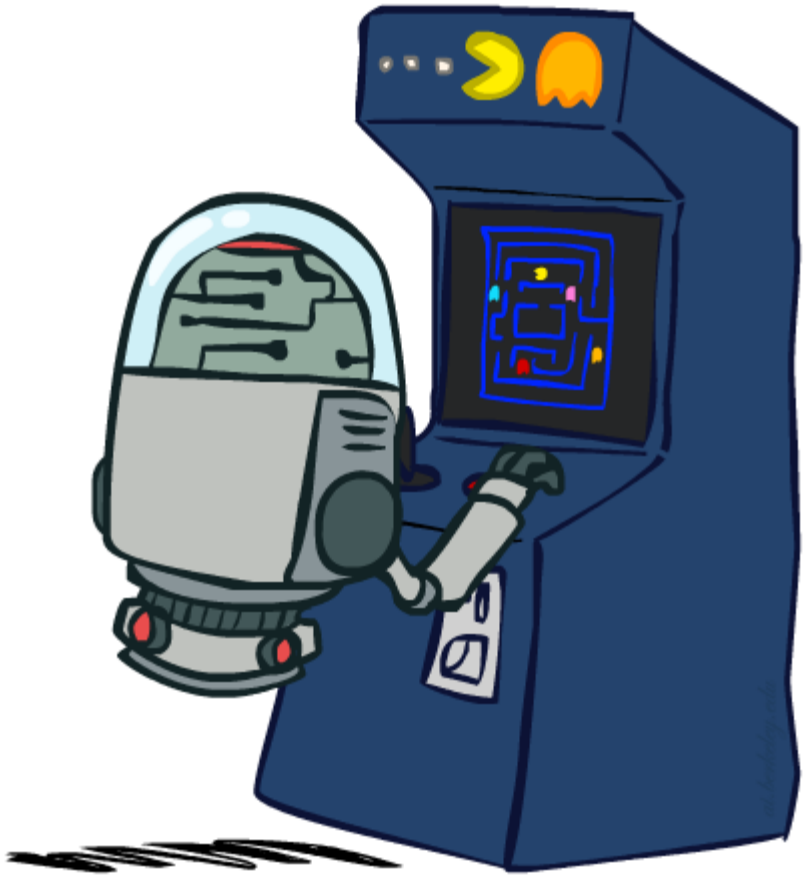
Types of Game

	Deterministic	Stochastic
Perfect Information	chess, checkers, go	backgammon, monopoly
Imperfect Information	blind tictactoe, battleships	bridge, poker, scrabble, nuclear war

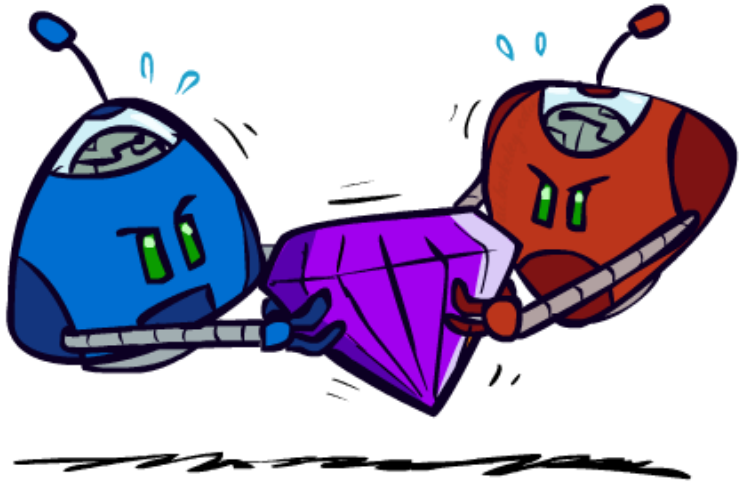
Deterministic Games

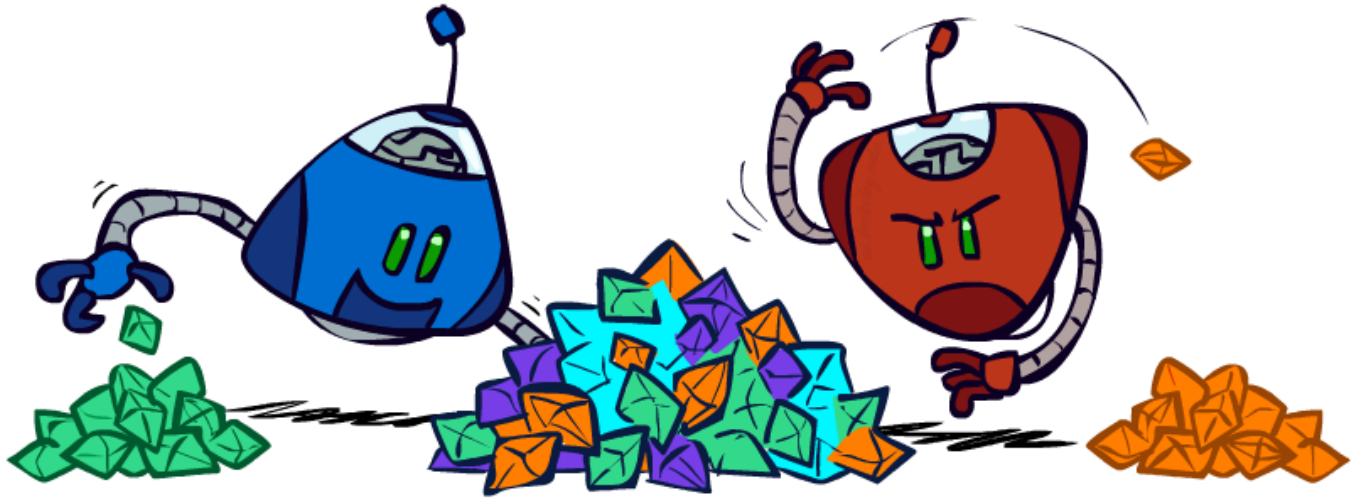
- Many possible formalizations, one is:
  - States: S (start at s_0)
  - Players: P={1...N} (usually take turns)
  - Actions: A (may depend on player / state)
  - Transition Function: SxA $\rightarrow$ S
  - Terminal Test: S $\rightarrow$ {t,f}
  - Terminal Utilities: SxP $\rightarrow$ R

- Solution for a player is a policy: S $\rightarrow$ A



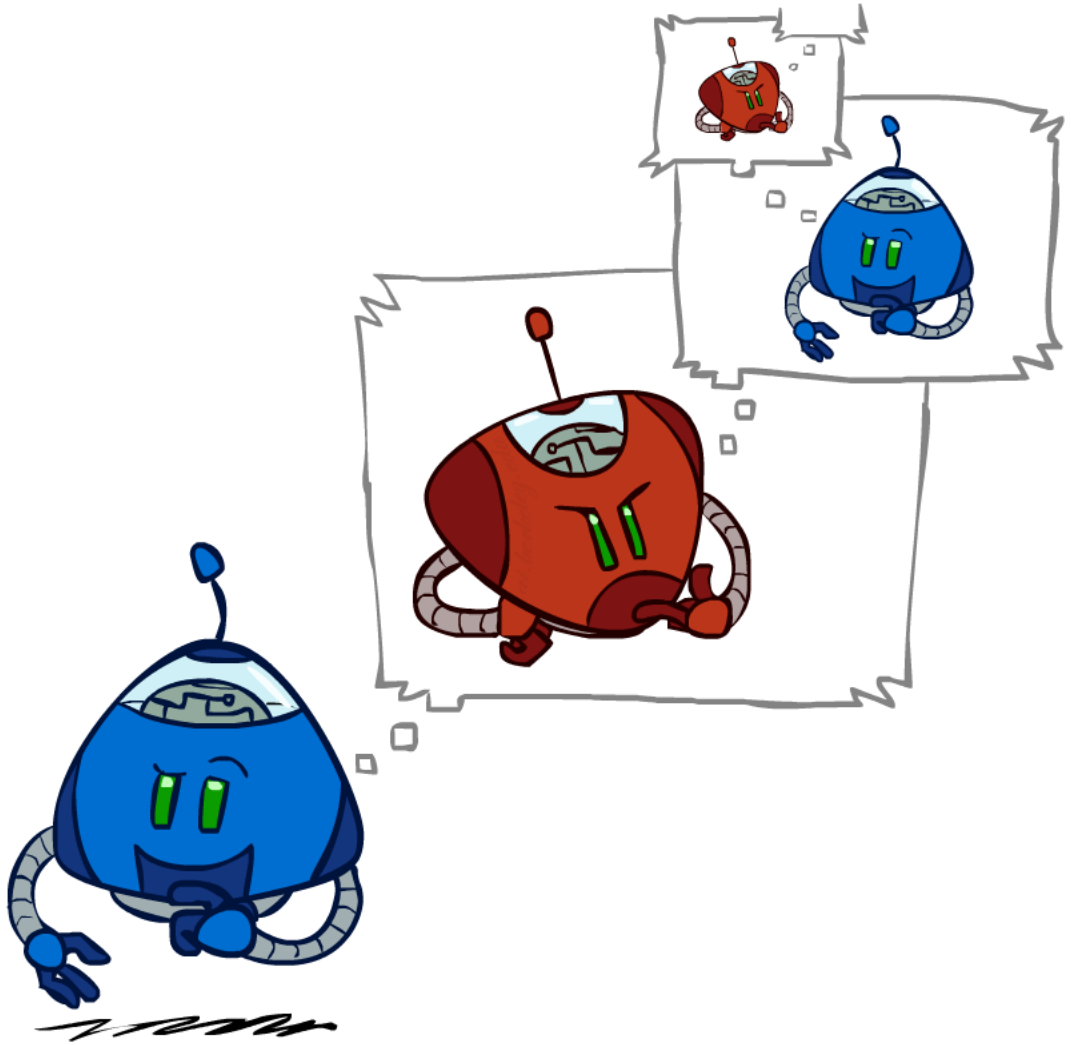
Zero-Sum Games



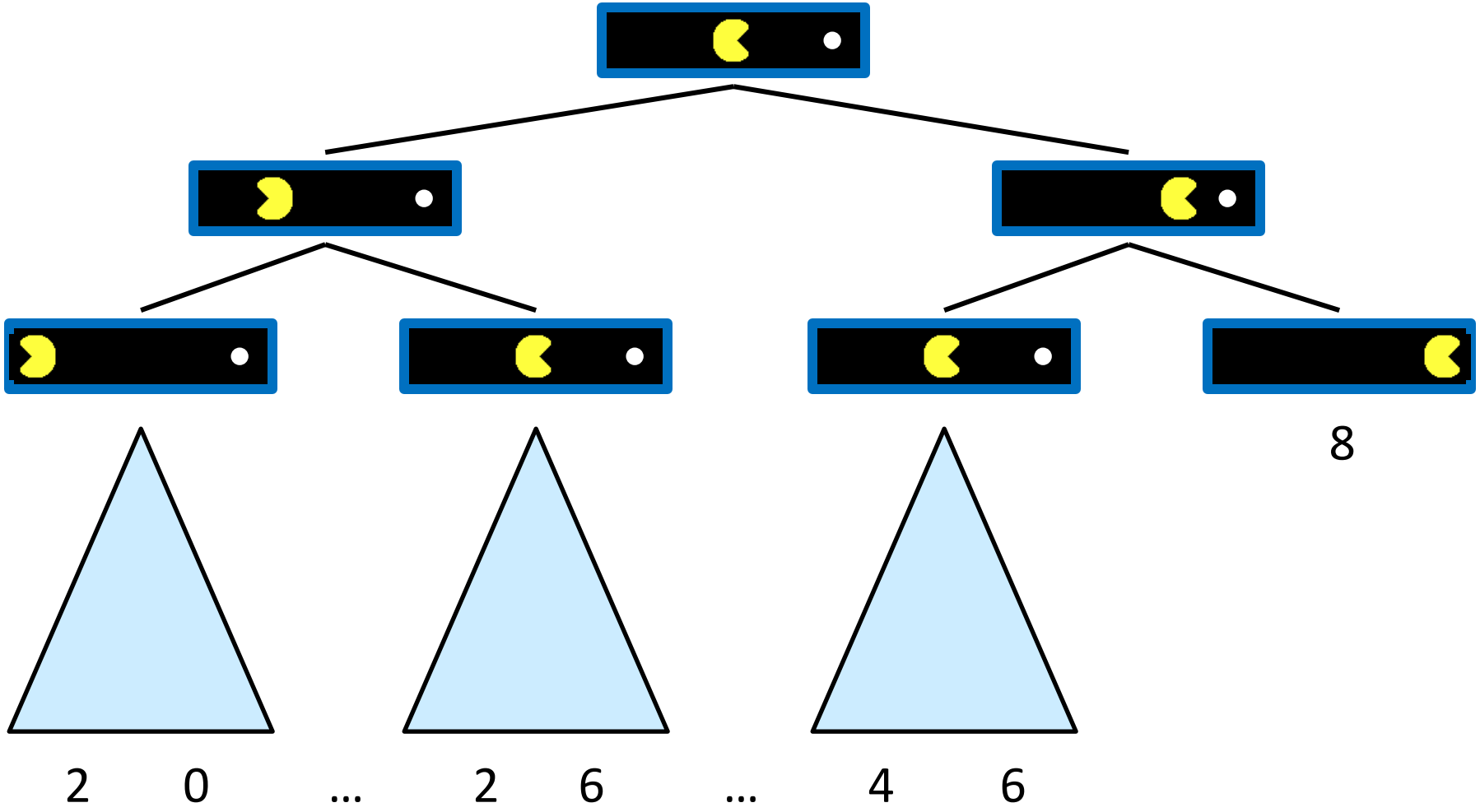


- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Lets us think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition
- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, and more are all possible

Adversarial Search

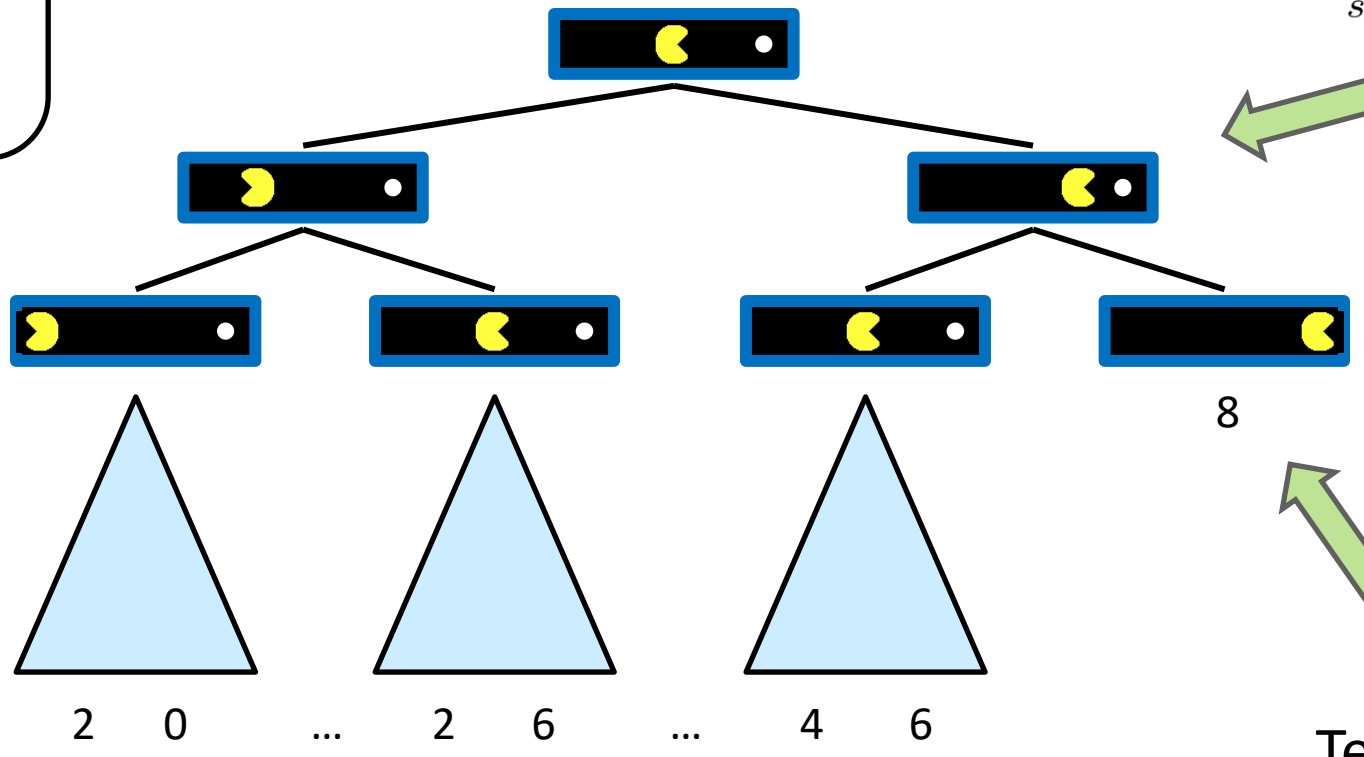


Single-Agent Trees



Value of a State

Value of a state:
The best achievable outcome (utility) from that state



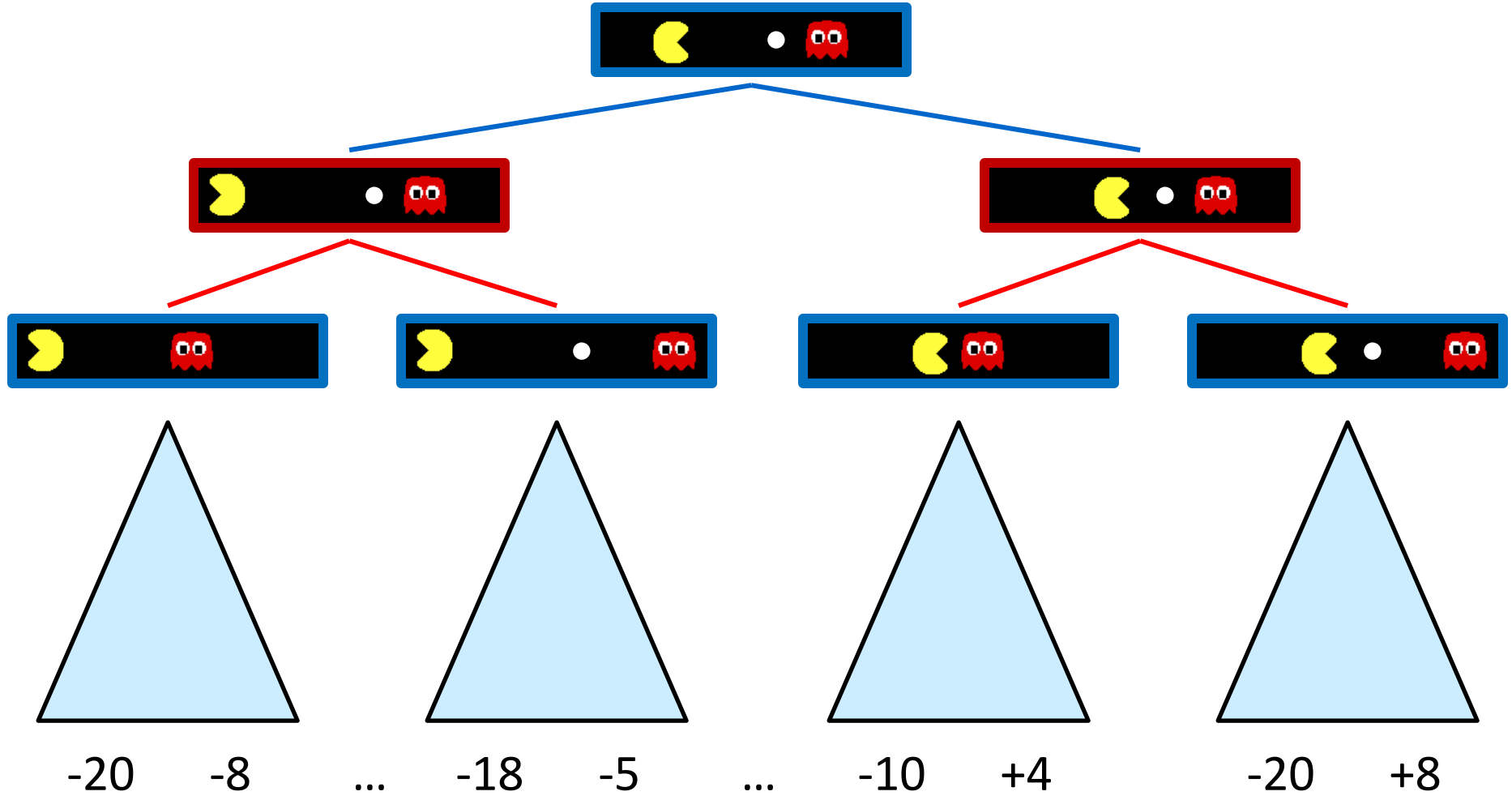
Non-Terminal States:

$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$

Terminal States:

$$V(s) = \text{known}$$

Adversarial Game Trees



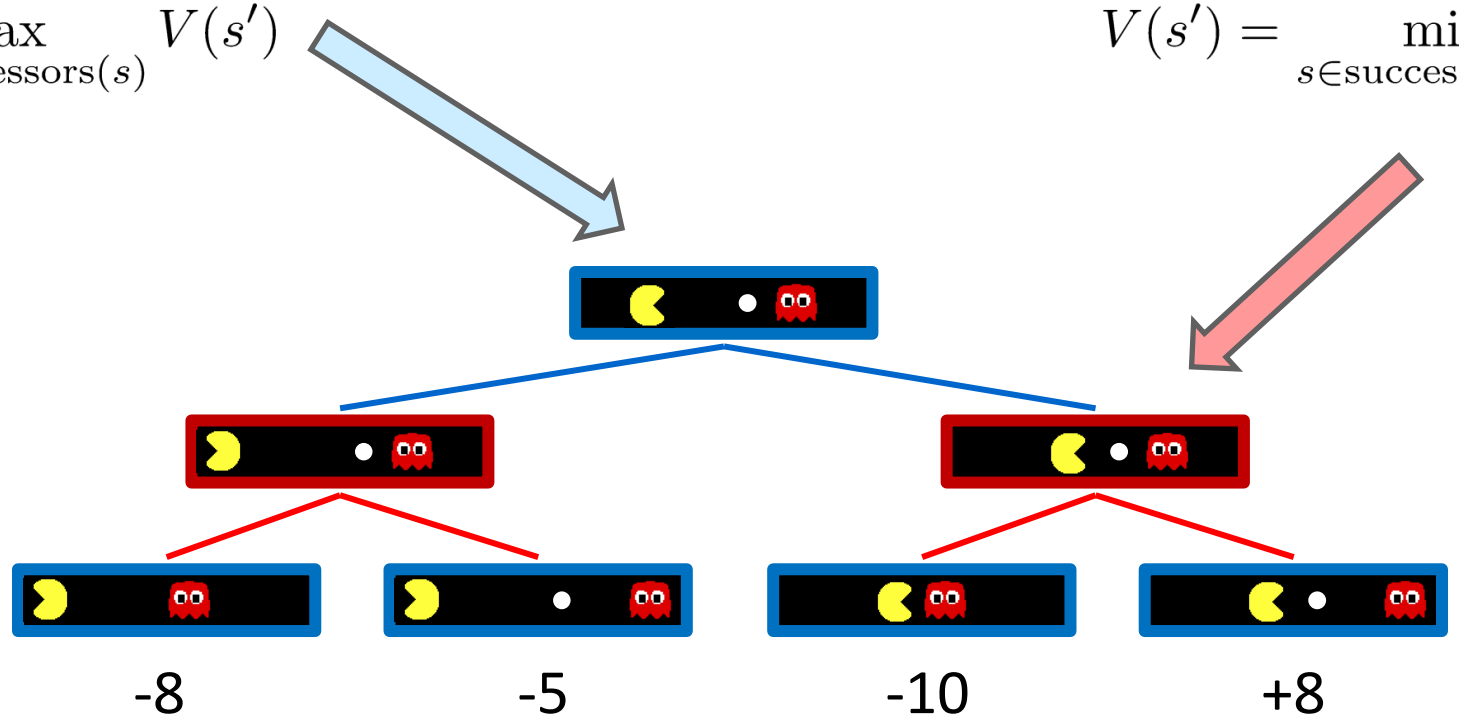
Minimax Values

States Under Agent's Control:

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

States Under Opponent's Control:

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

$$V(s) = \text{known}$$

Tic-Tac-Toe Game Tree

MAX (X)

MIN (O)

MAX (X)

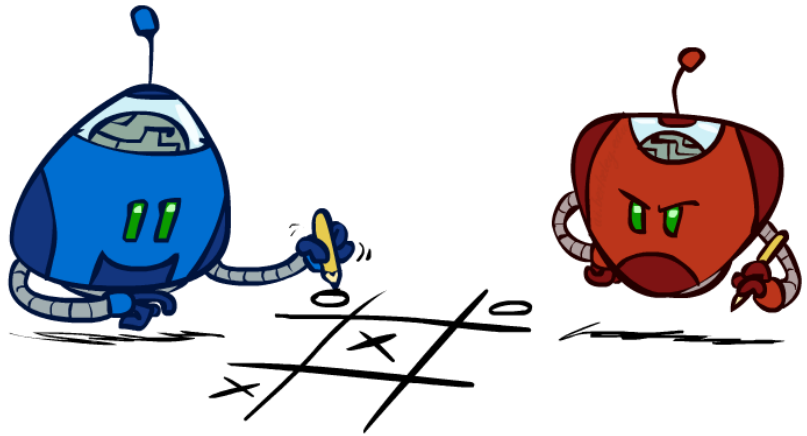
MIN (O)

TERMINAL

Utility

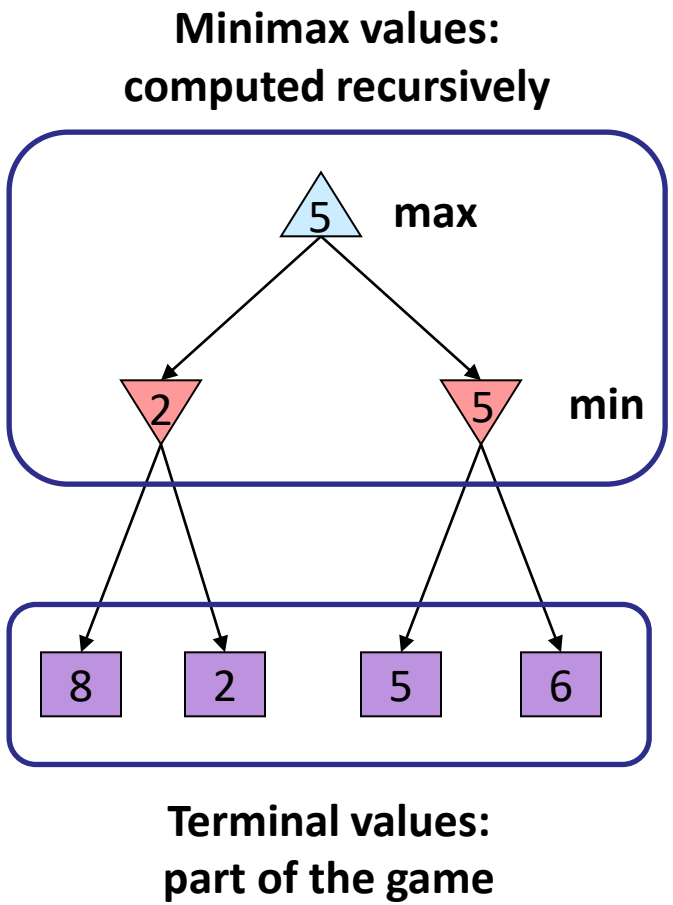
−1 0 +1

tictactoe – 9!=362880 terminal nodes



Adversarial Search (Minimax)

- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes result
- Minimax search:
  - A state-space search tree
  - Players alternate turns
  - Compute each node's minimax value: the best achievable utility against a rational (optimal) adversary



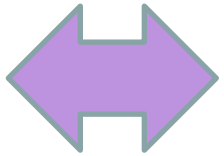
Minimax algorithm

- Minimax algorithm
  - Perfect play for deterministic, 2-player game
  - Max tries to maximize its score
  - Min tries to minimize Max's score (Min)
  - Goal: Max to move to position with highest minimax value

    → Identify best achievable payoff against best play

Minimax Implementation

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, min-value(successor))
    return v
```



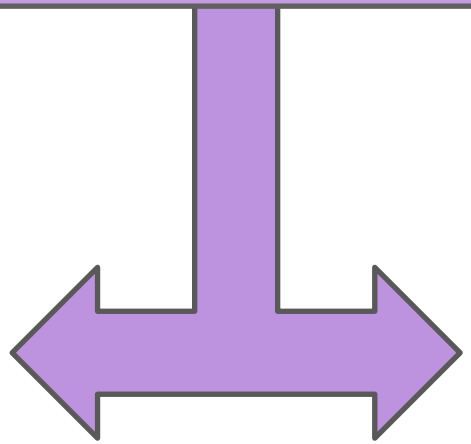
```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, max-value(successor))
    return v
```

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

Minimax Implementation (Dispatch)

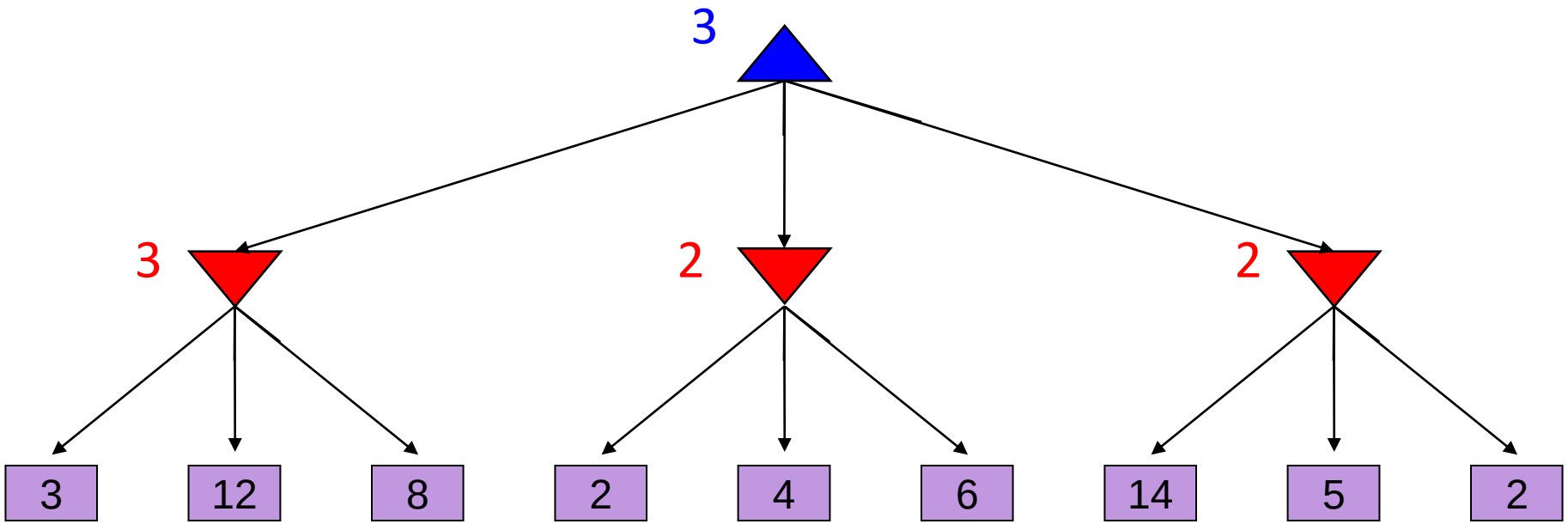
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```



```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

Minimax Example

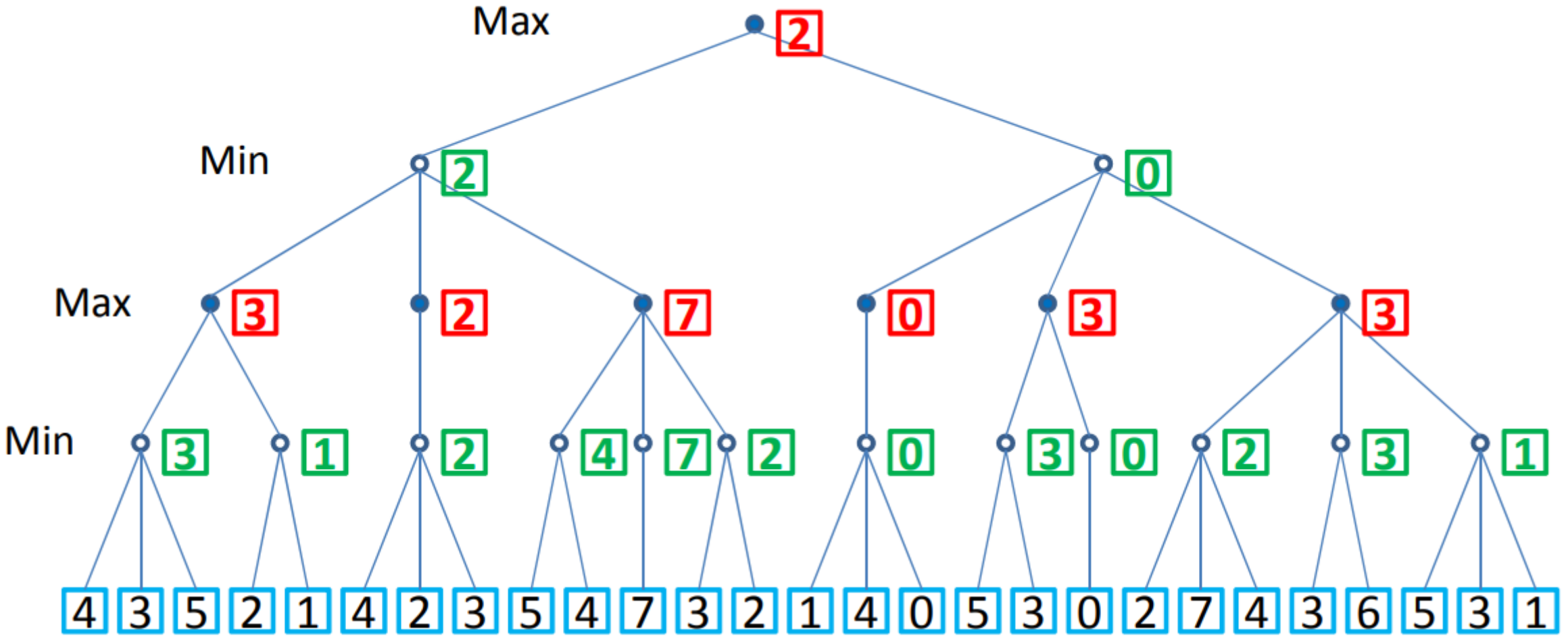


Poll

What kind of search is Minimax Search?

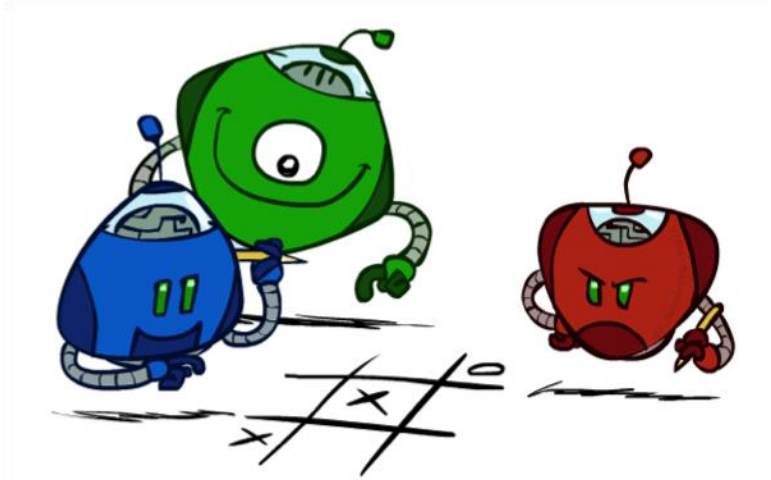
A) BFS
B) DFS
C) UCS
D) A*

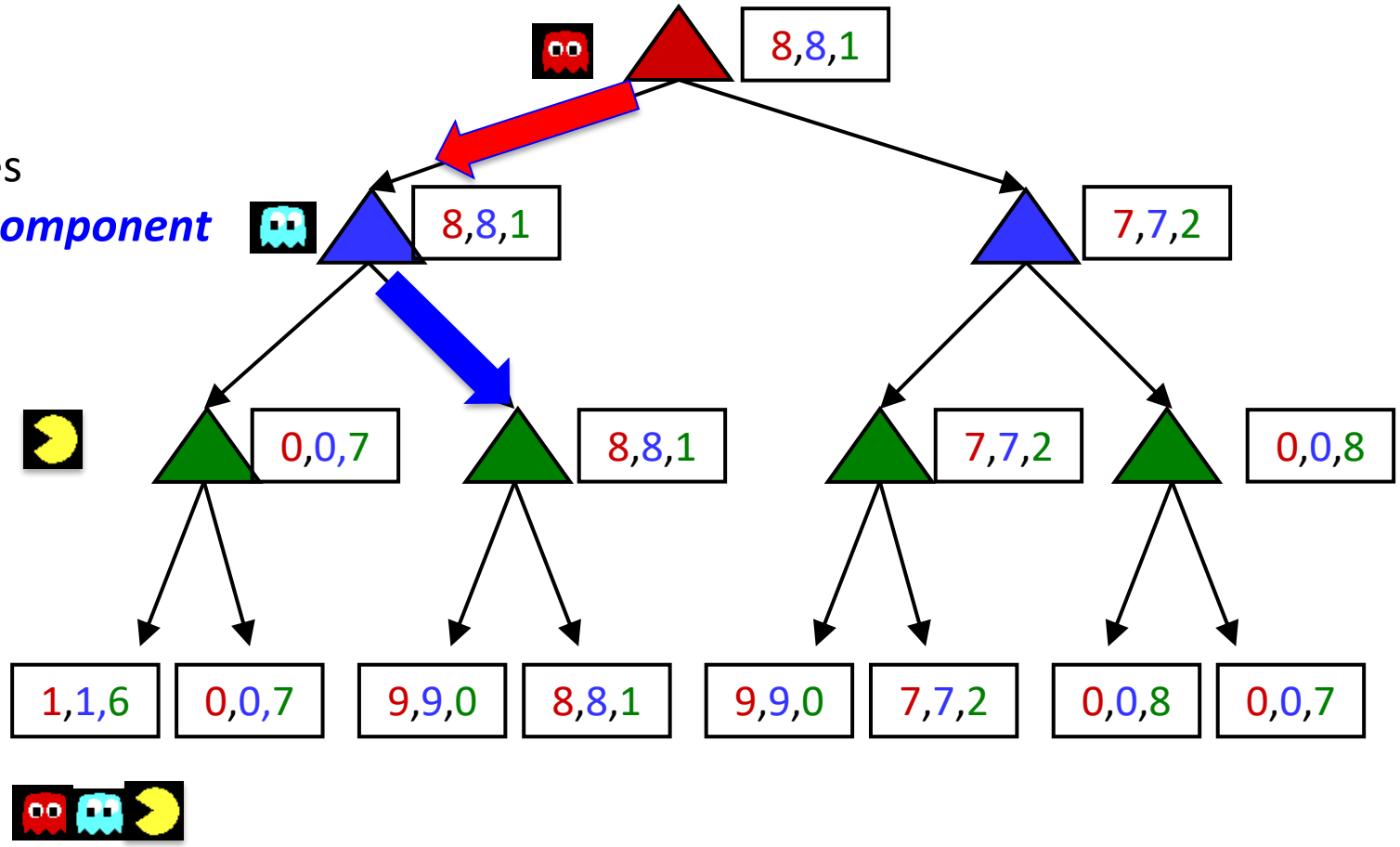
Minimax Example



Generalized minimax

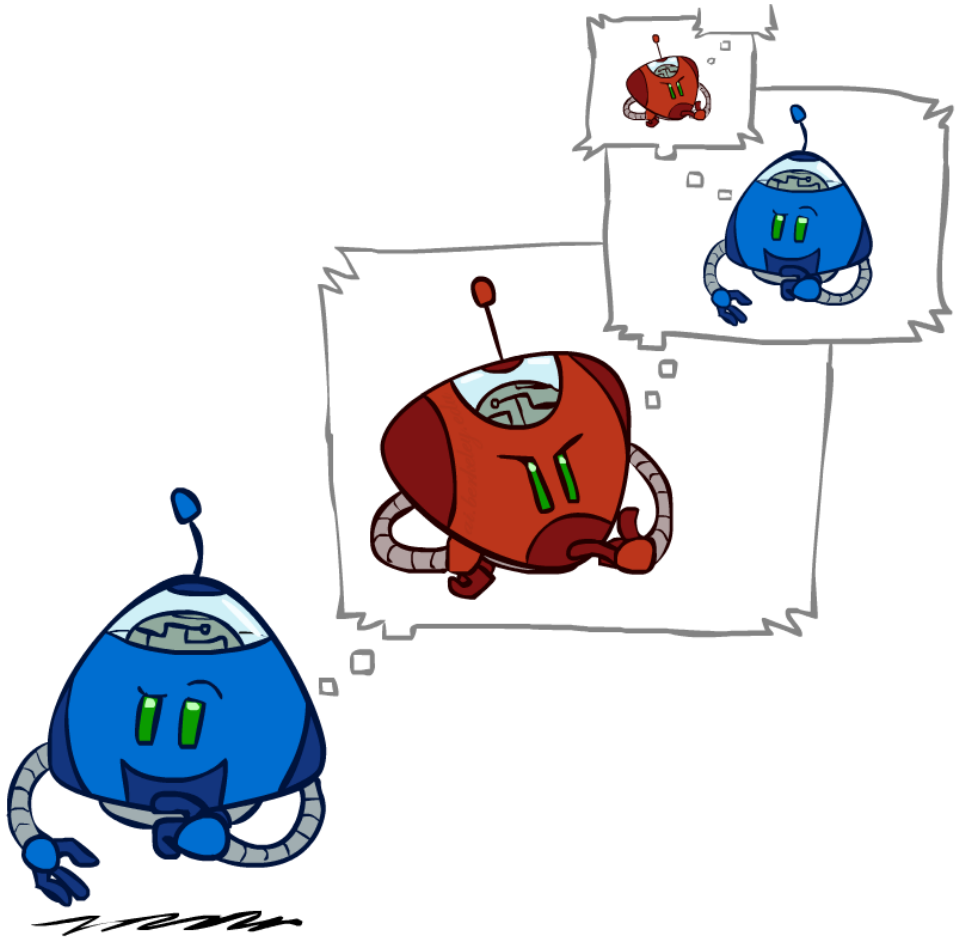
- What if the game is not zero-sum, or has multiple players?
- Generalization of minimax:
  - Terminals have ***utility tuples***
  - Node values are also utility tuples
  - ***Each player maximizes its own component***
  - Can give rise to cooperation and competition dynamically…





Minimax Efficiency

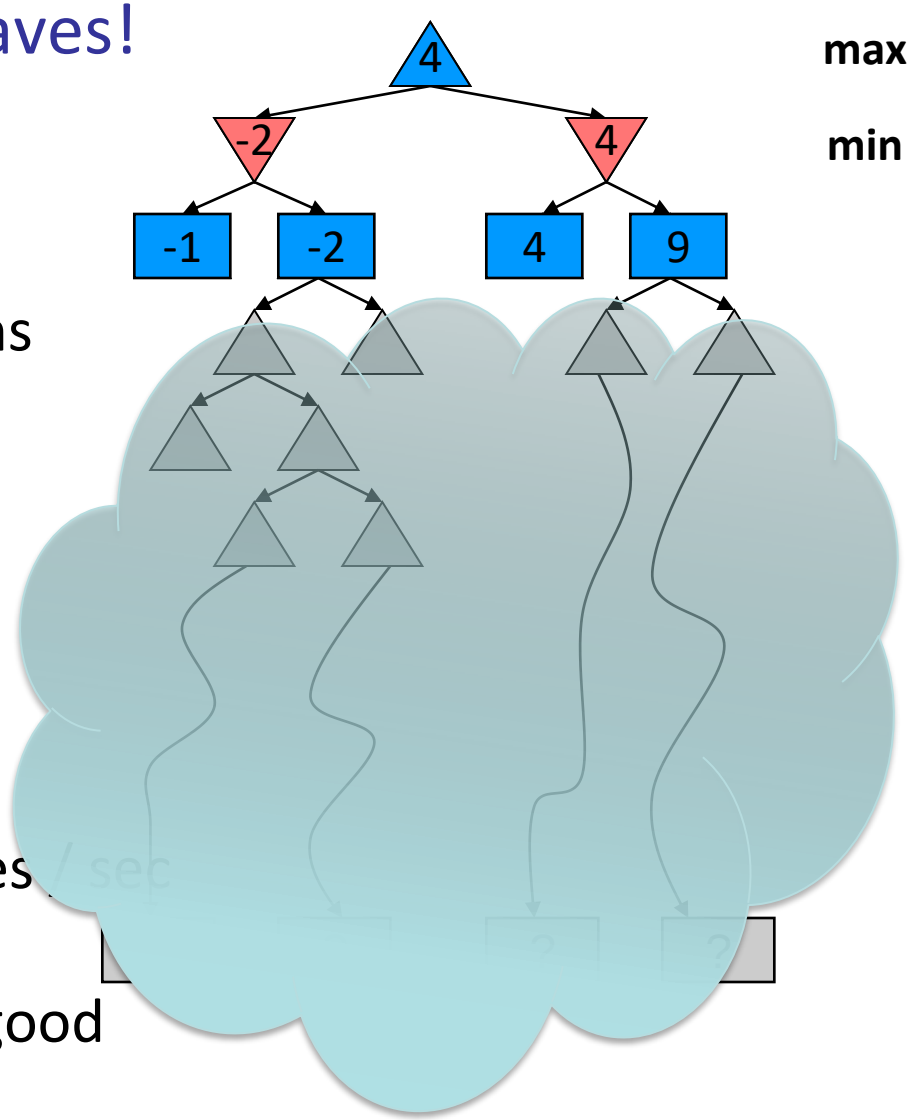
- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: O(bm)
- Minimax Solution
  - Using the current state as the initial state, build the game tree uniformly to the maximal depth ***m*** (called horizon) feasible within the time limit
  - Evaluate the states of the leaf nodes
  - ***Back up*** the results from the leaves to the root and pick the best action assuming the best play by MIN (worst for MAX)
- Example: For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?



Resource Limits

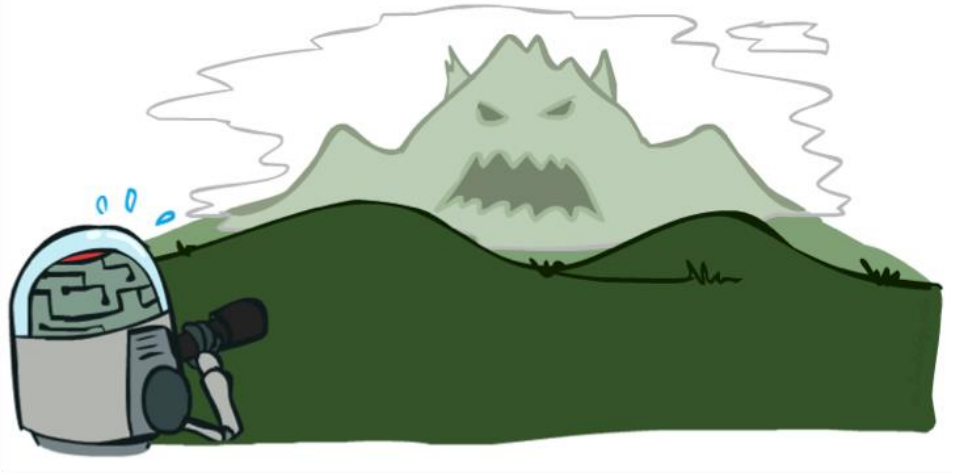
Resource Limits

- Problem: In realistic games, cannot search to leaves!
- Solution 1: Bounded lookahead
  - Search only to a preset ***depth limit*** or ***horizon***
  - Use an ***evaluation function*** for non-terminal positions
- Guarantee of optimal play is gone
- More plies make a BIG difference
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - For chess, b=~35 so reaches about depth 4 – not so good



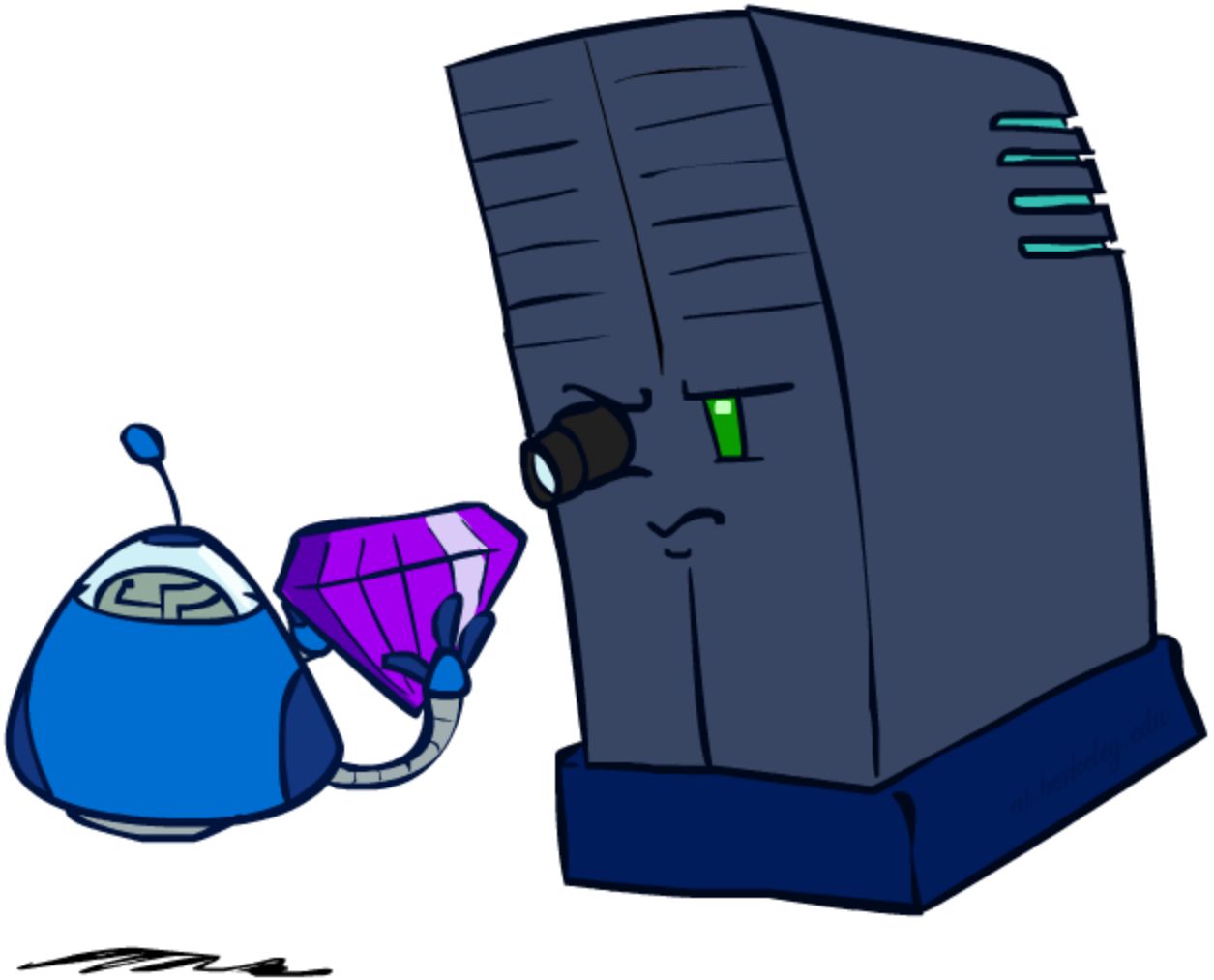
Depth Matters

- Evaluation functions are always imperfect
- Deeper search => better play (usually)
- Or, deeper search gives same quality of play with a less accurate evaluation function
- An important example of the tradeoff between complexity of features and complexity of computation



[Demo: depth limited (L6D4, L6D5)]

Evaluation Functions

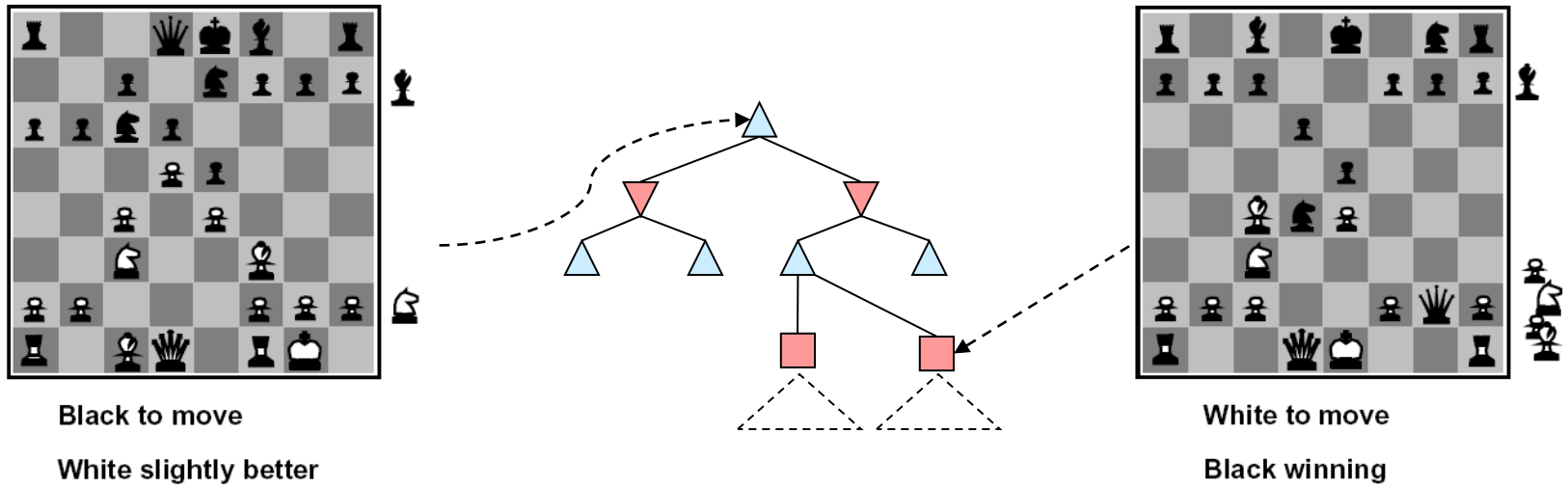


Evaluation Function

- Function E: state s $\rightarrow$ number E(s)
  - E(s) is a heuristics: estimates how favorable s is for MAX
  - E(s) > 0 → s is favorable to MAX (the larger the better)
  - E(s) < 0 → s is favorable to MIN
  - E(s) = 0 → s is neutral
- Features may include
  - Number of pieces of each type
  - Number of possible moves
  - Number of squares controlled
- Why using backed-up values?
  - At each non-leaf node ***n***, the backed-up value is the value of the best state that MAX can reach at depth ***m*** if MIN plays well (by the same criterion as MAX applies to itself)
  - If **E** is to be trusted in the first place, then the backed-up value is a better estimate of how favorable STATE(***n***) is than E(STATE(***n***))

Evaluation Functions

- Evaluation functions score non-terminals in depth-limited search



- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

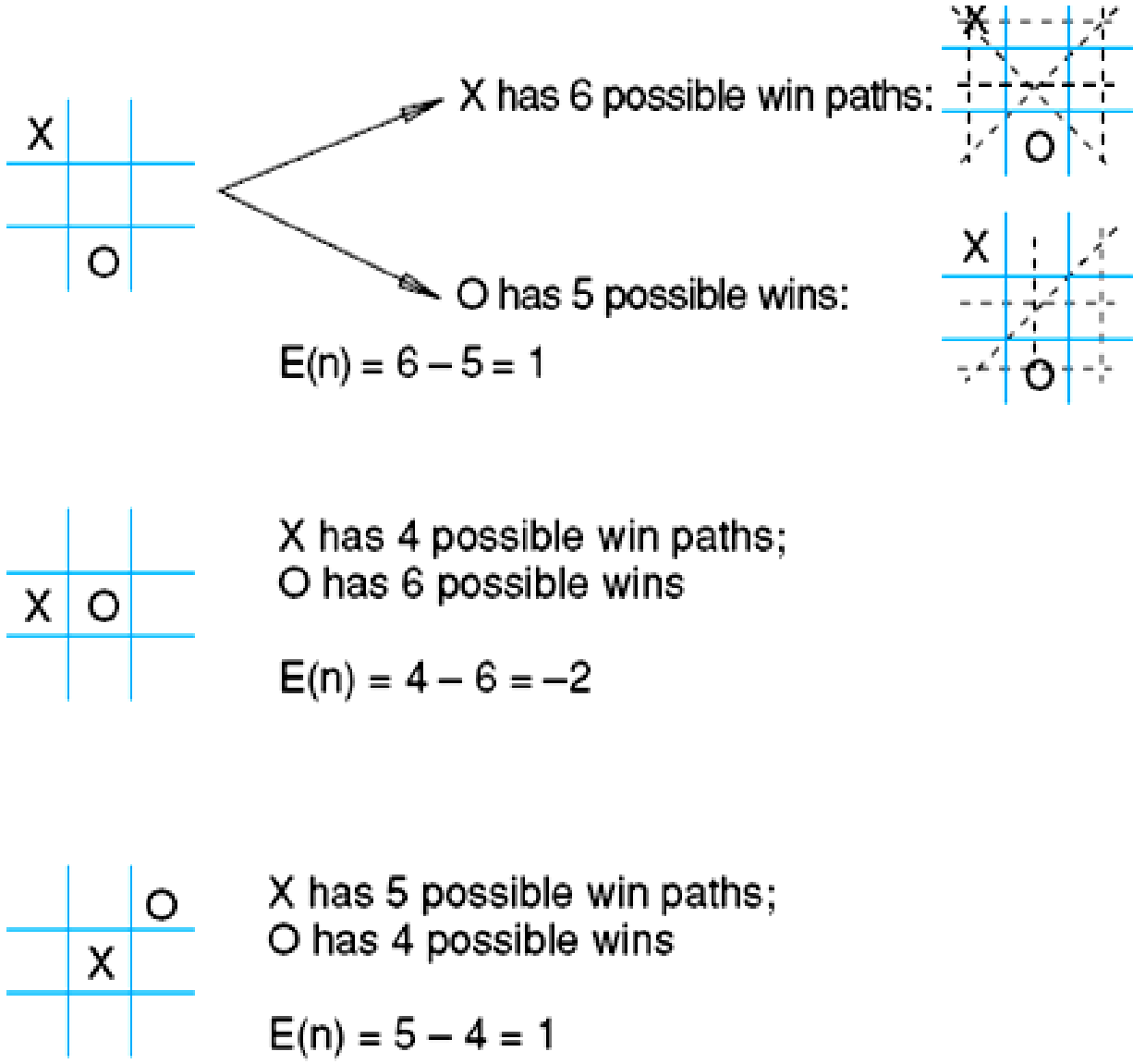
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

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

- e.g. $f_1(s)$ = (num white queens – num black queens), etc.

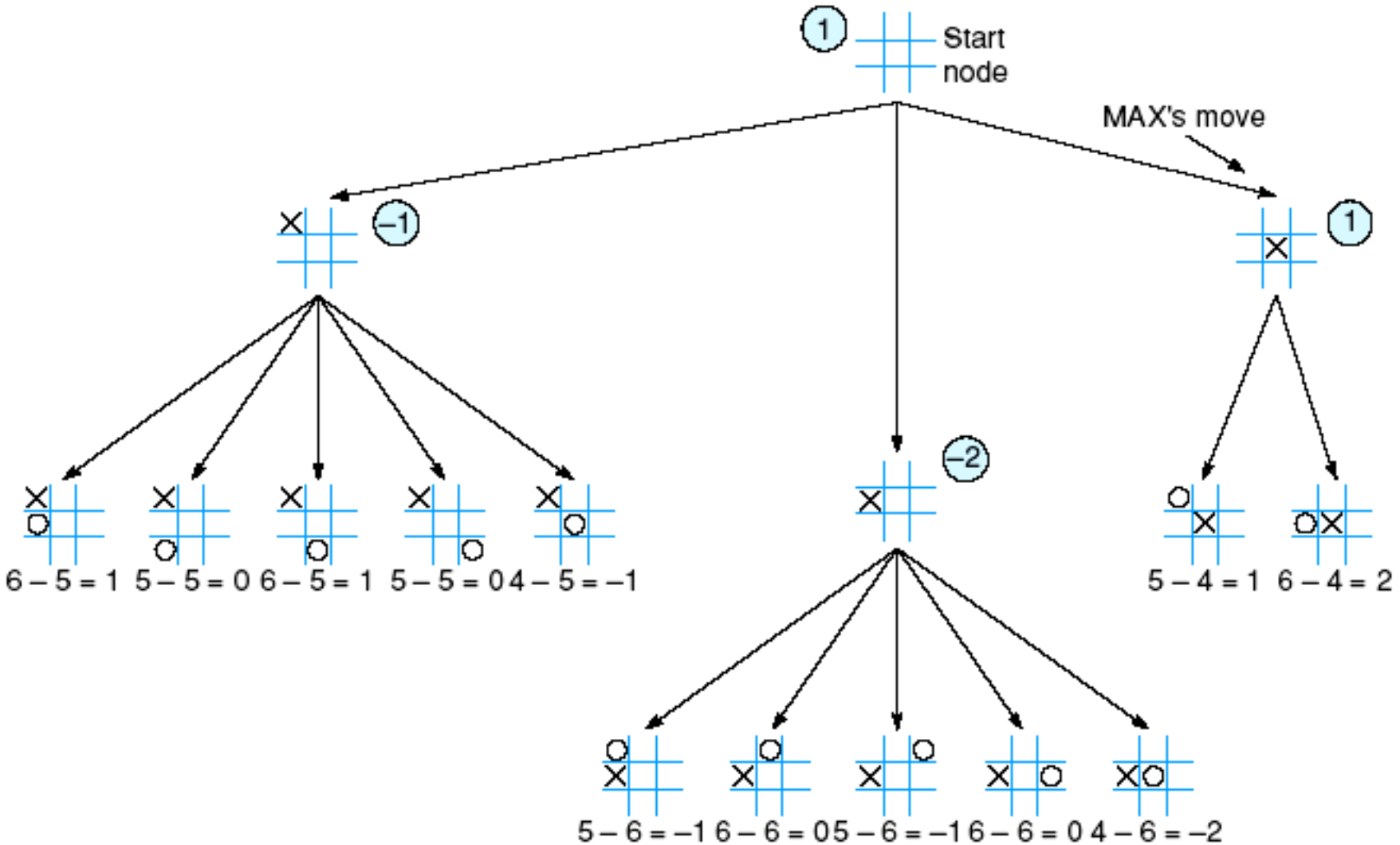
Example: A Heuristic for Tic-Tac-Toe

- Heuristic: E(*n*) = M(*n*) – O(*n*)
  - M(*n*) = Total possible wining lines for MAX
  - O(*n*) = Total possible wining lines for the opponent



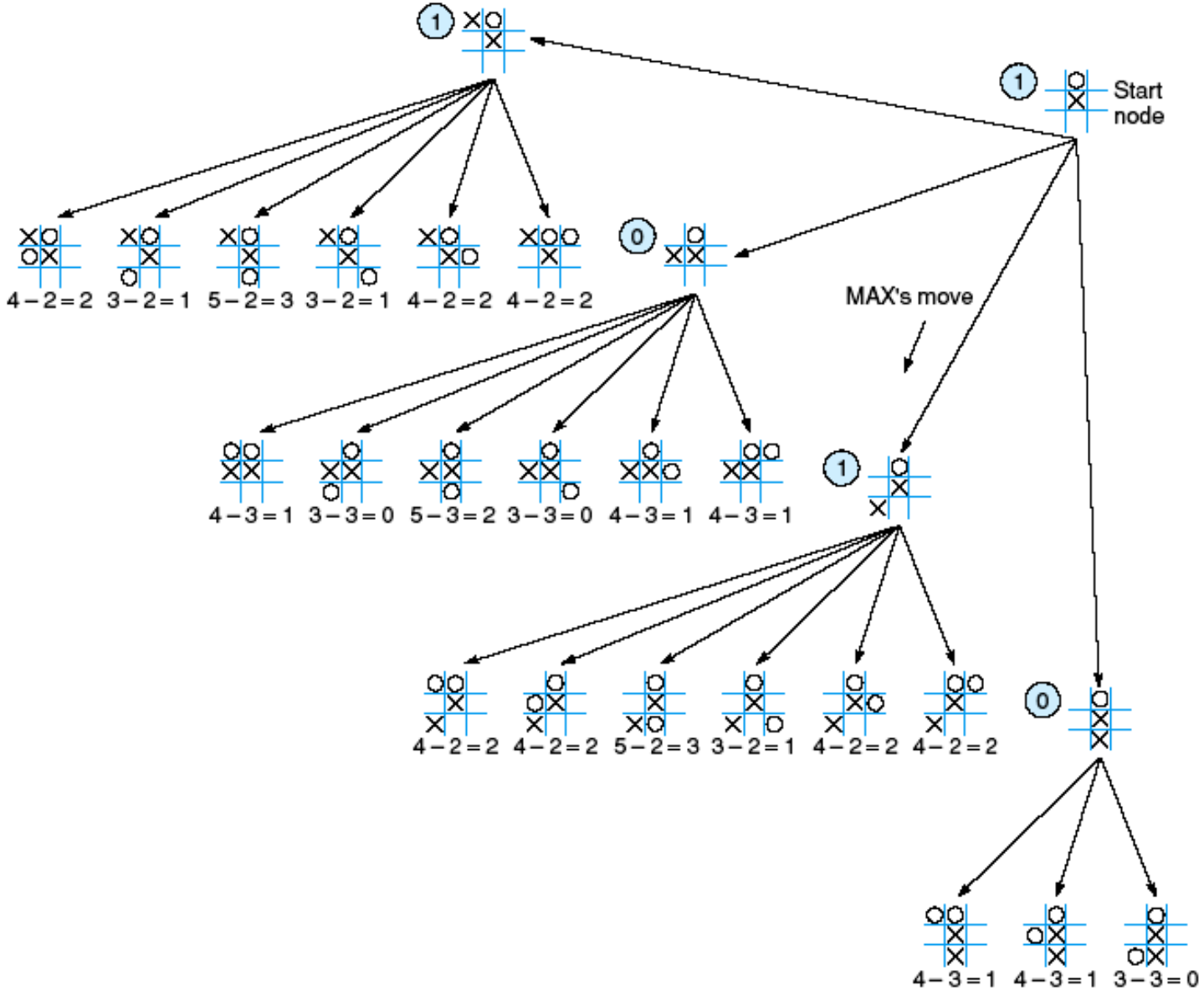
Example: 2-ply Minimax on Tic-Tac-Toe

Tic-Tac-Toe Tree at horizon 2 shows the backed up Minimax values



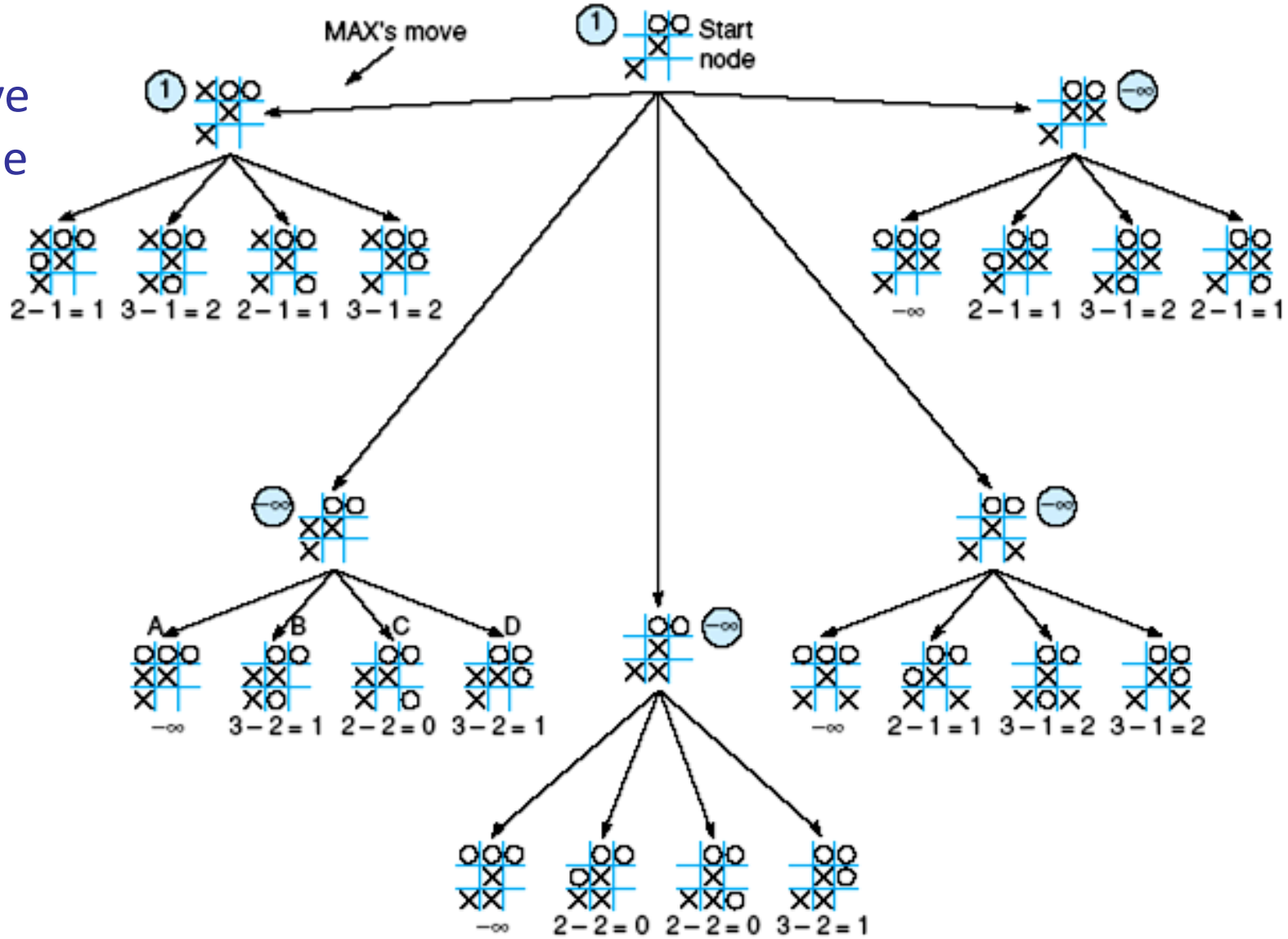
Example: Tic-Tac-Toe (cont.)

2-ply Minimax applied to one of two possible MAX second moves



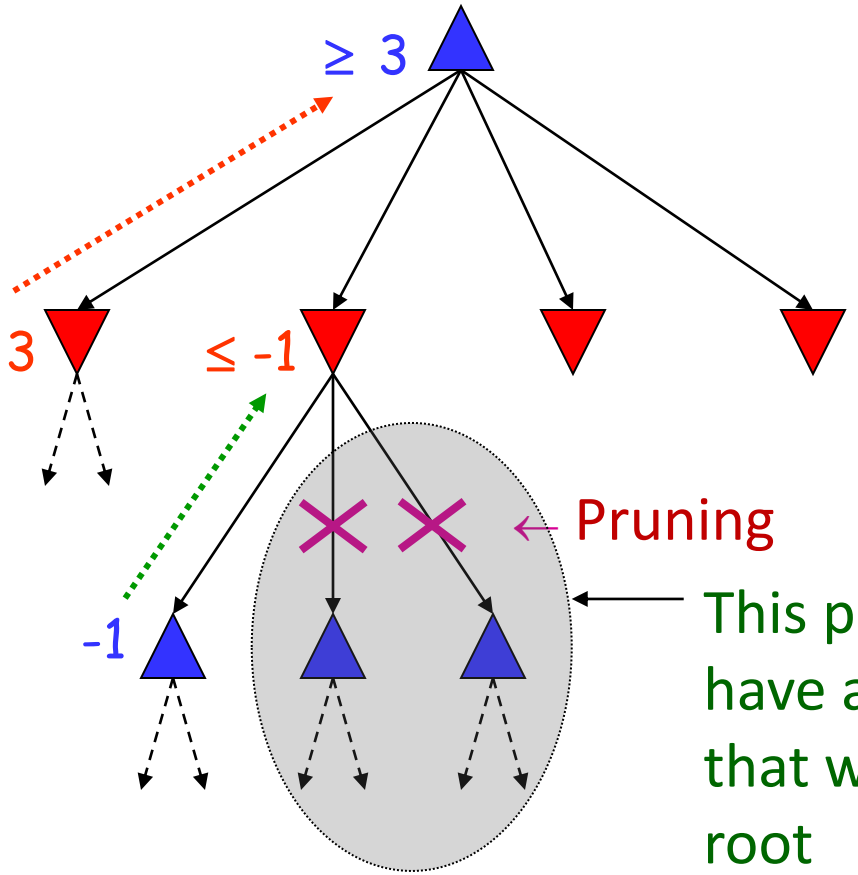
Example: Tic-Tac-Toe (cont.)

2-ply Minimax applied to X's move near the end of the game

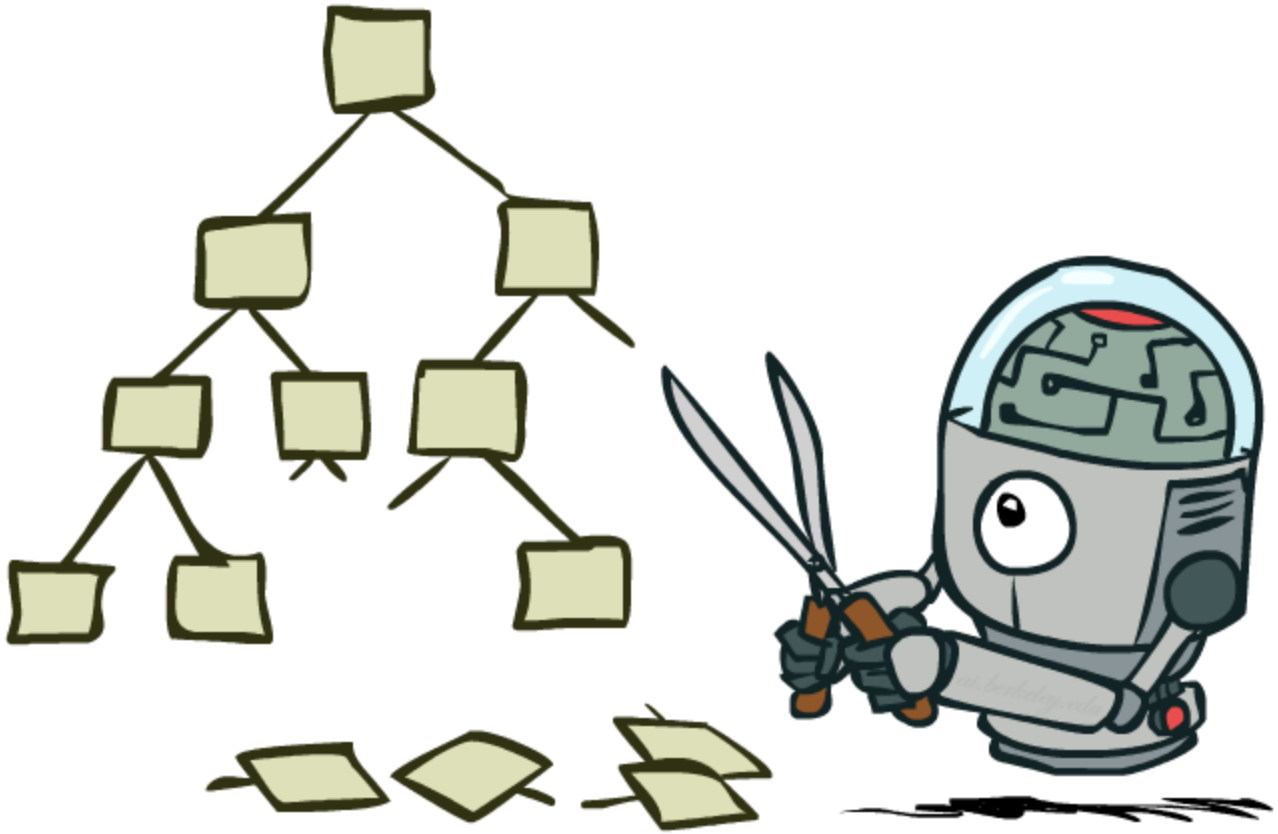


Can we do better?

- Yes ! Much better !



Game Tree Pruning



Alpha-Beta Implementation

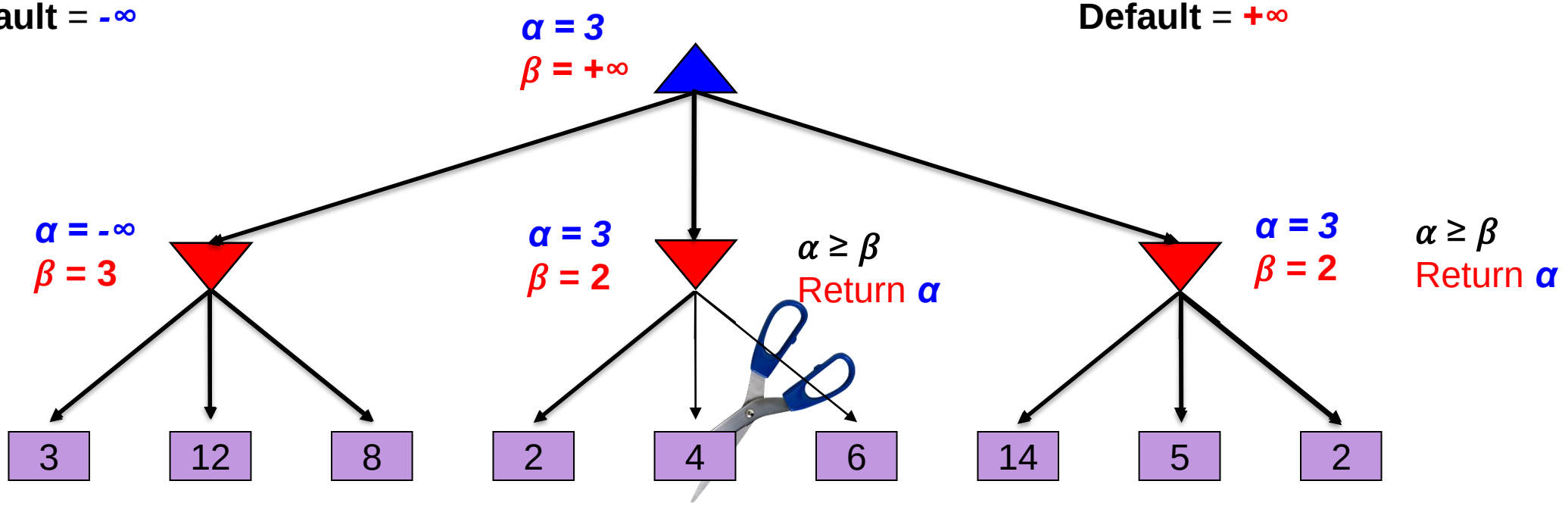
```
1:  procedure MinimaxAlphaBeta(N, α, β)
2:      Inputs
3:          N a node in a game tree
4:          α, β real numbers
5:      Output
6:          The value for node N
7:      if N is a leaf node then
8:          return value of N
9:      else if N is a MAX node then
10:         for each child C of N do
11:             Set α ← max(α, MinimaxAlphaBeta(C, α, β))
12:             if α ≥ β then
13:                 return β
14:         return α
15:     else
16:         for each child C of N do
17:             Set β ← min(β, MinimaxAlphaBeta(C, α, β))
18:             if α ≥ β then
19:                 return α
20:         return β
```

Node Type	LEAF	MAX	MIN
Update		α	β
Return	Utility Value	α	β
Condition		$\alpha \geq \beta$	
if Condition True Then Return		β	α

Alpha-Beta Pruning Example

α = best option so far from any MAX node on this path
Default = $-\infty$

β = best option so far from any MIN node on this path
Default = $+\infty$

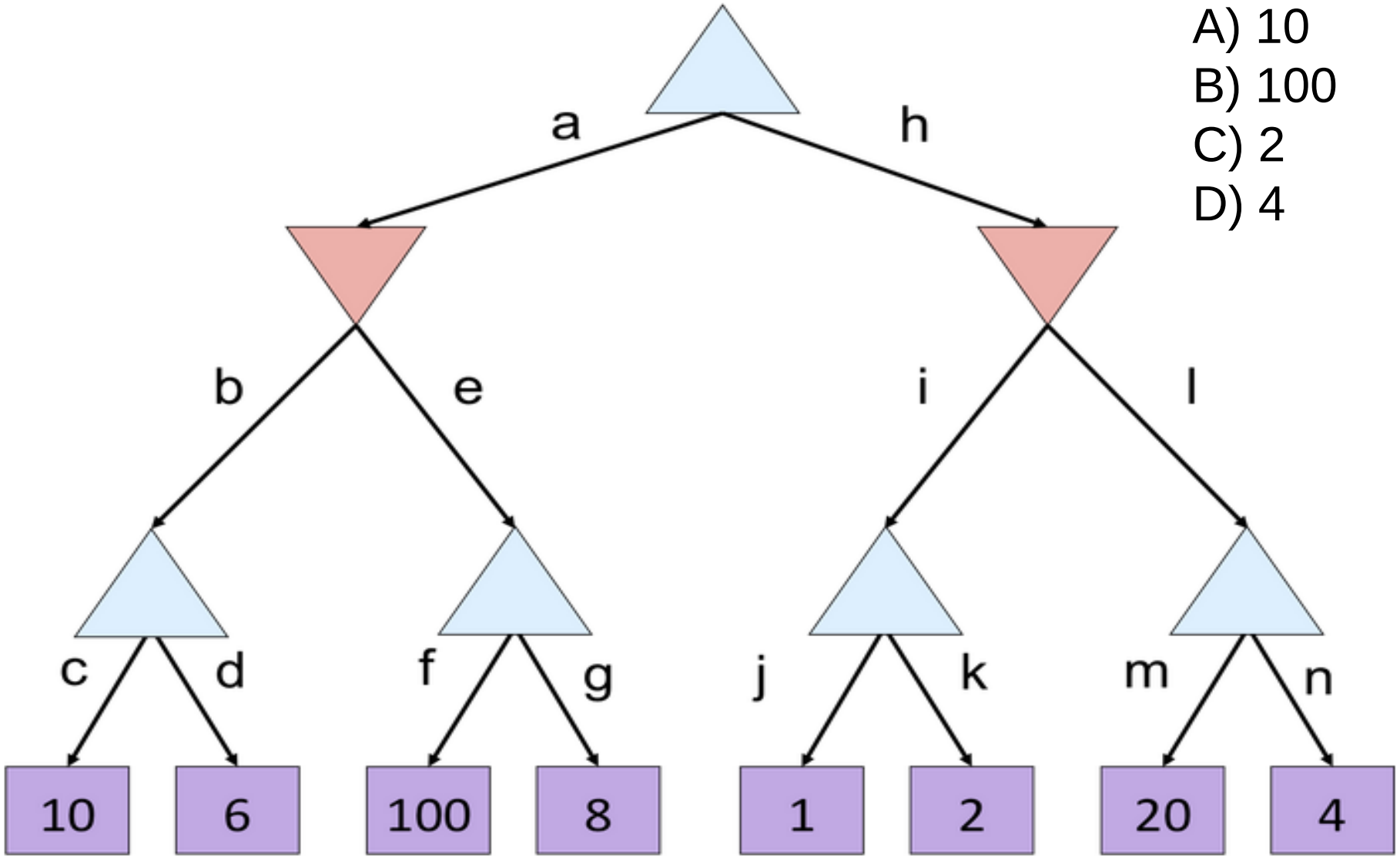


Node Type	LEAF	MAX	MIN
Update		α	β
Return	Utility Value	α	β
Condition		$\alpha \geq \beta$	
if Condition True Then Return		β	α

Minimax Quiz

What is the value of the top node?

A) 10
B) 100
C) 2
D) 4



Alpha Beta Quiz

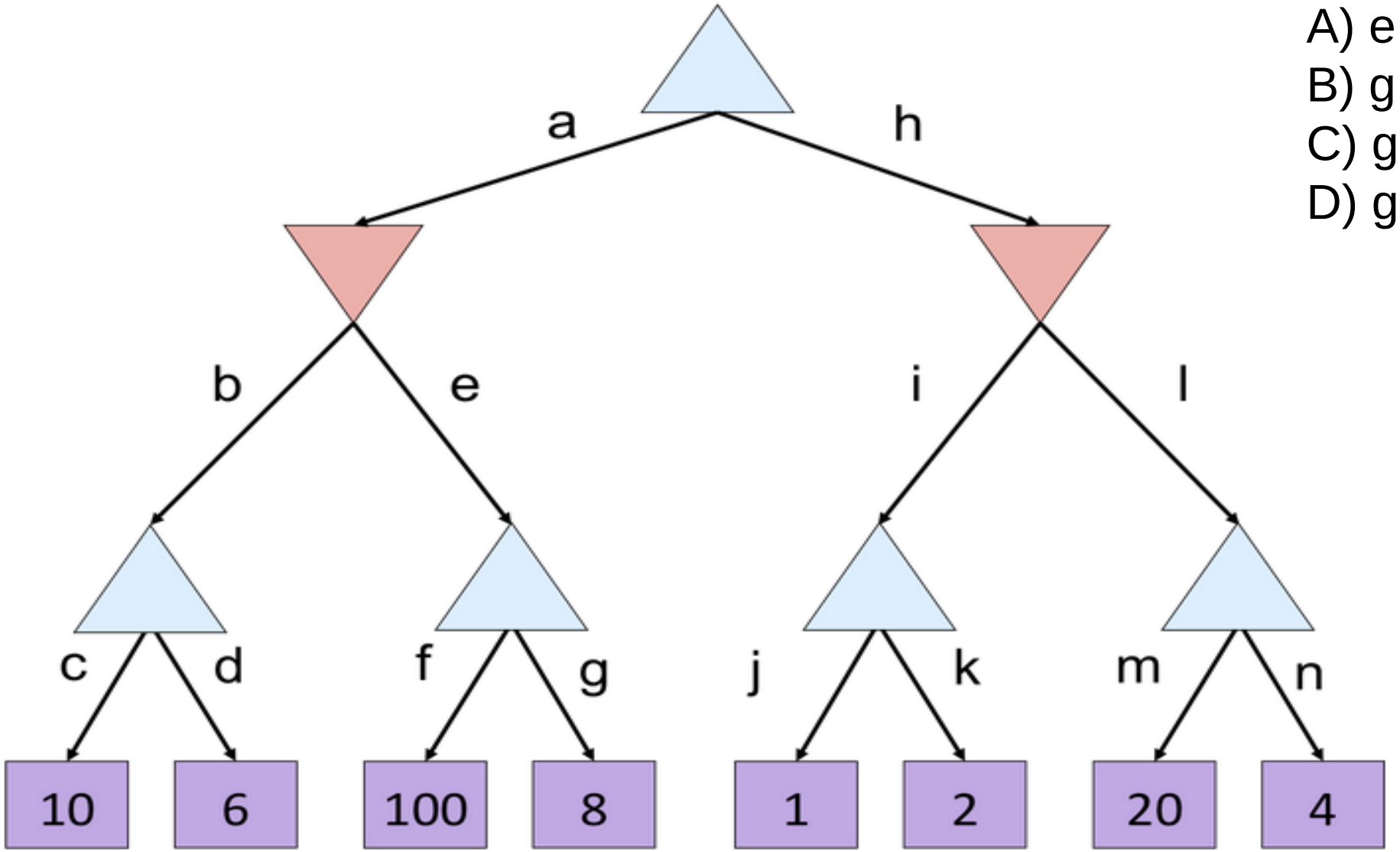
Which branches are pruned?

A) e, l

B) g, l

C) g, k, l

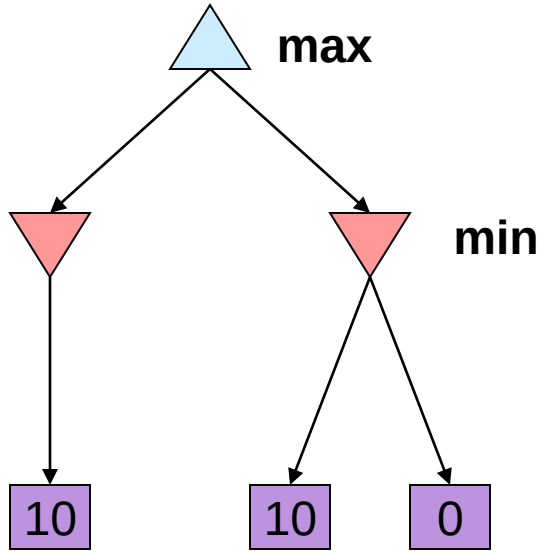
D) g, n



For Practice

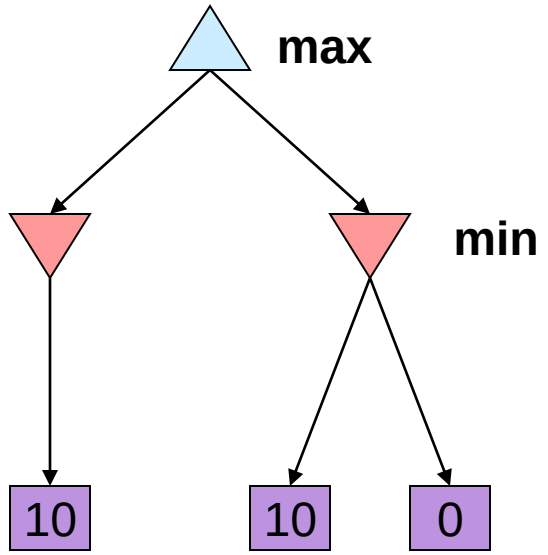
Visit this Website: https://pascscha.ch/info2/abTreePractice/

Alpha-Beta Pruning Properties



Alpha-Beta Pruning Properties

- This pruning has no effect on minimax value computed for the root!
- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value
  - So the most naïve version won't let you do action selection
- Good child ordering improves effectiveness of pruning
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless…
- This is a simple example of metareasoning (computing about what to compute)



Additional Refinements

- Waiting for Quiescence: continue the search until no drastic change occurs from one level to the next.
- Secondary Search: after choosing a move, search a few more levels beneath it to be sure it still looks good.
- Book Moves: for some parts of the game (especially initial and end moves), keep a catalog of best moves to make.

Checkers: Tinsley vs. Chinook

Name:	Marion Tinsley
Profession:	Teach mathematics
Hobby:	Checkers
Record:	Over 42 years loses only 3 games of checkers

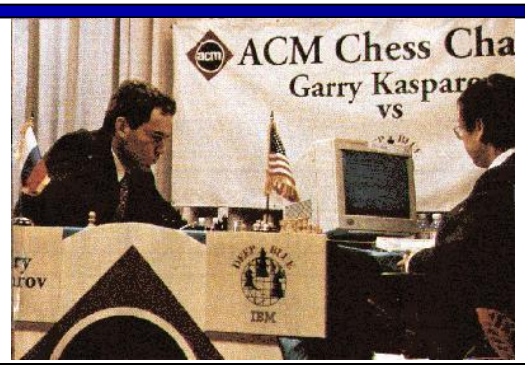
World champion for over 40 years

1994: Mr. Tinsley suffered his 4th and 5th losses against Chinook

Chinook

- First computer to become official world champion of Checkers!

Chess: Kasparov vs. Deep Blue



Kasparov		Deep Blue
5’10”	**Height**	6’ 5”
176 lbs	**Weight**	2,400 lbs
34 years	**Age**	4 years
50 billion neurons	**Computers**	32 RISC processors + 256 VLSI chess engines
2 pos/sec	**Speed**	200,000,000 pos/sec
Extensive	**Knowledge**	Primitive
Electrical/chemical	**Power Source**	Electrical
Enormous	**Ego**	None

1997: Deep Blue wins by 3 wins, 1 loss, and 2 draws

Chess: Kasparov vs. Deep Junior

Deep Junior

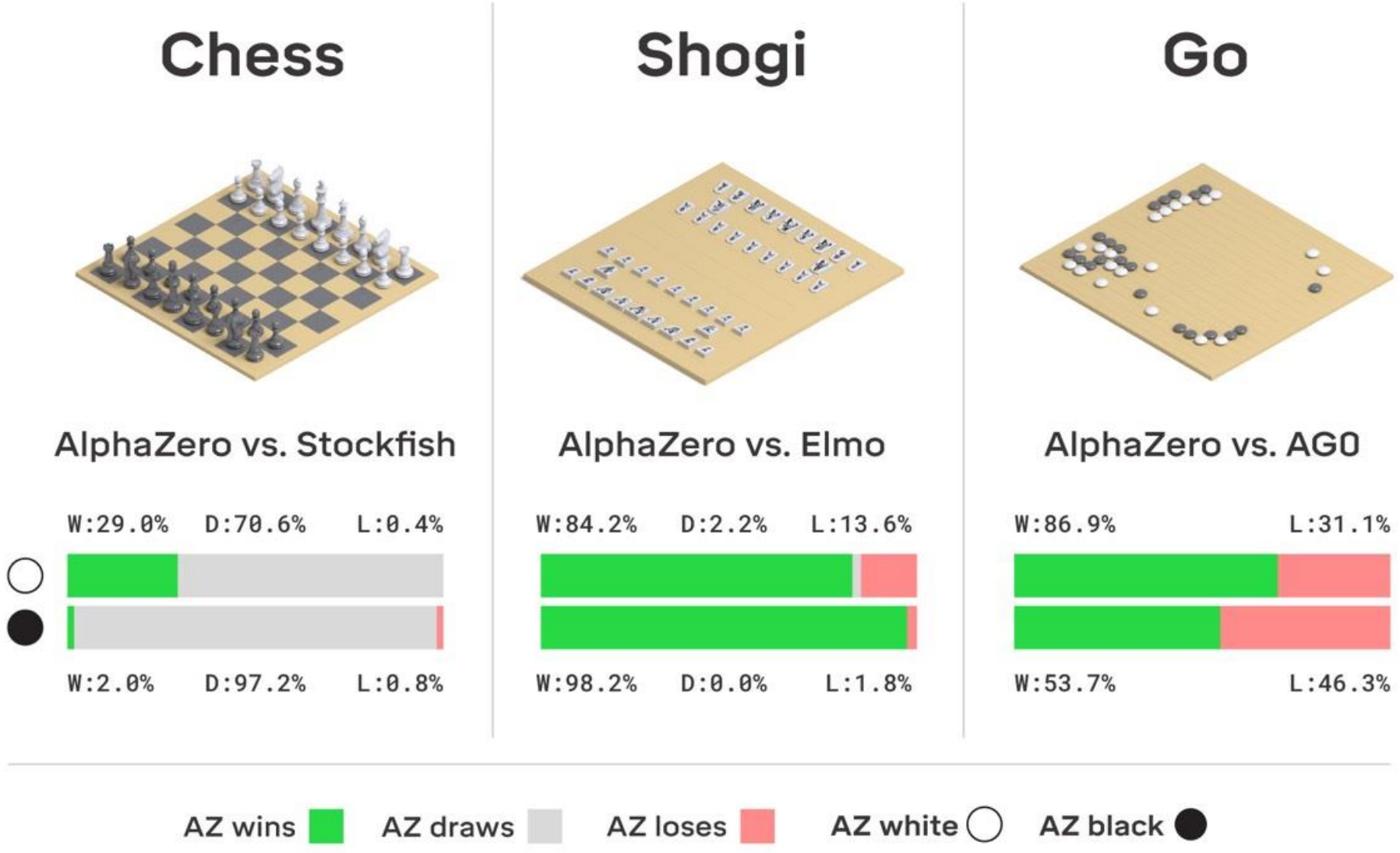
8 CPU, 8 GB RAM, Win 2000

2,000,000 pos/sec

Available at $100

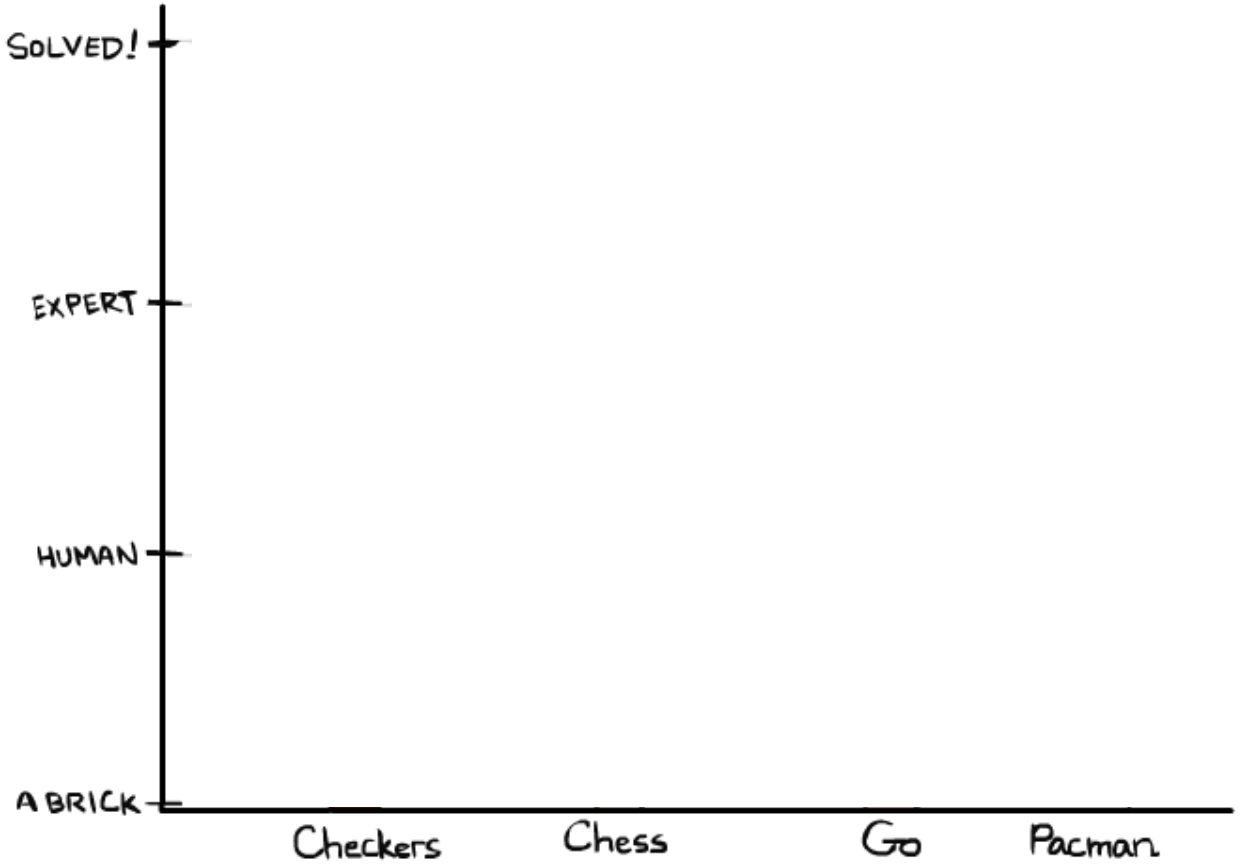
August 2, 2003:Match ends in a 3/3 tie!

Go: Goemate vs. ??



Game Playing State-of-the-Art

- **Checkers: 1950**: First computer player. **1994**: First computer champion: **Chinook** ended 40-year-reign of human champion **Marion Tinsley** using complete 8-piece endgame. 2007: Checkers solved!
- **Chess: 1997**: **Deep Blue** defeats human champion **Gary Kasparov** in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.
- **Go:** Human champions are now starting to be challenged by machines. In go, b > 300! Classic programs use pattern knowledge bases, but big recent advances use Monte Carlo (randomized) expansion methods.



Game Playing State-of-the-Art

- **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!
- **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.
- **Go: 2016: Alpha GO defeats human champion. Uses Monte Carlo Tree Search, learned evaluation function.**
- **Pacman**