COMPSCI 761: ADVANCED TOPICS IN ARTIFICIAL INTELLIGENCE

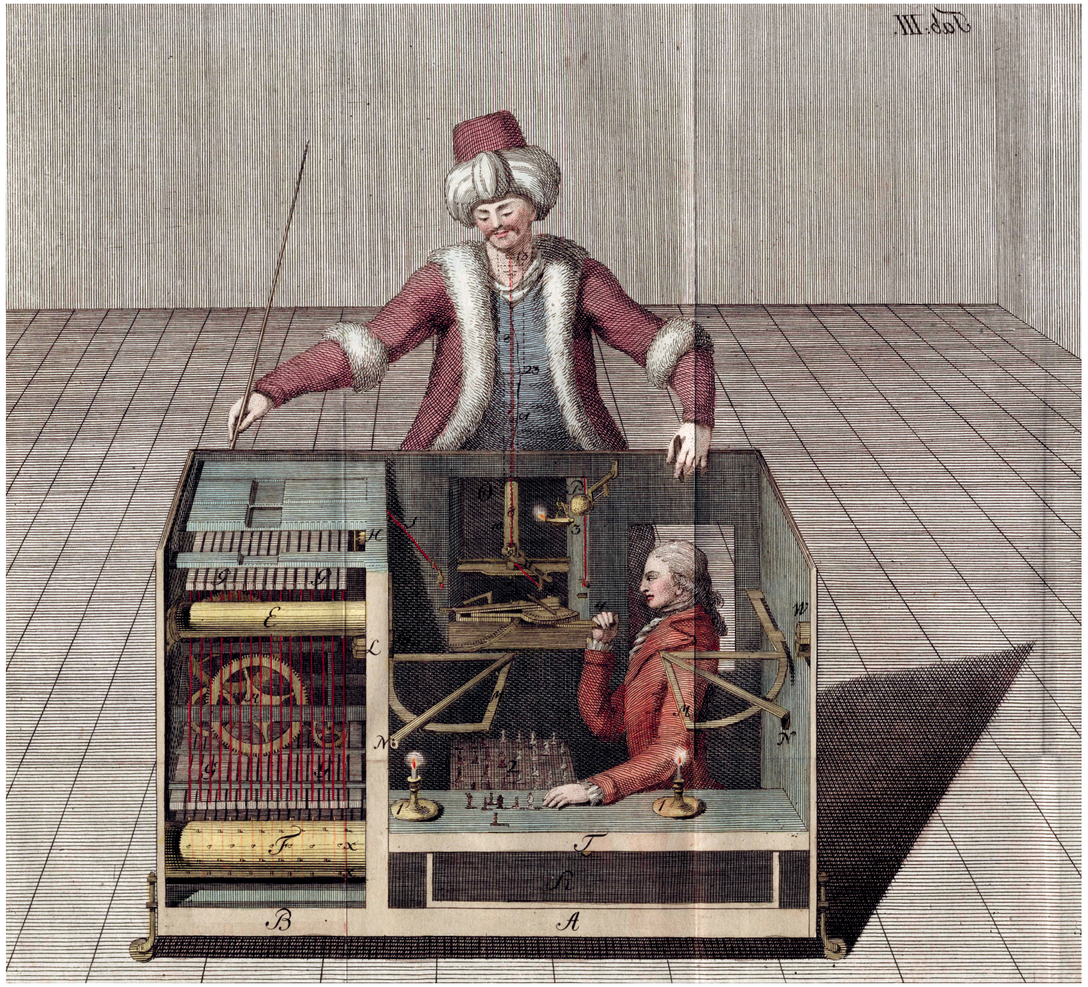
GAMES

Anna Trofimova, August 2022

TODAY

- Games history and motivation
- Minimax
- α-β pruning

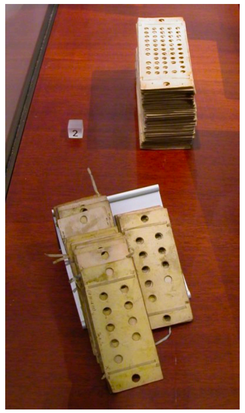
MECHANICAL TURK

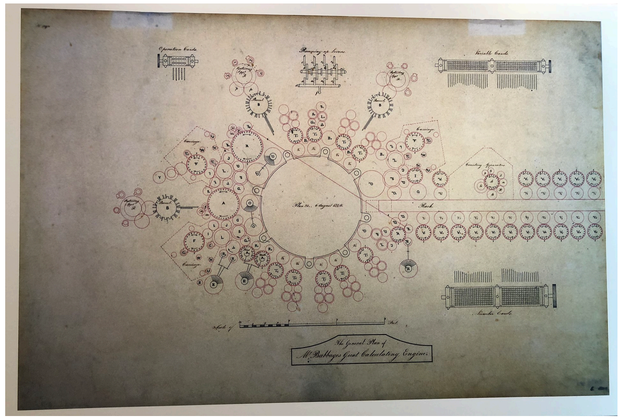


- Constructed and unveiled in 1770 by Wolfgang von Kempelen for Maria Theresa of Austria
- A mechanical illusion - a machine with a human player inside
- Was destroyed by fire in 1854 in Chinese Museum of Charles Willson Peale, US.
- How did the human operator knew the move of the opponent?

THE ANALYTICAL ENGINE

- Babbage (1791 - 1871) is a computer pioneer, mathematician, engineer, inventor, philosopher
  - was working on The Analytical Engine, a mechanical general-purpose computer with arithmetic logic units and control flow
  - documented the tic-tac-toe automaton in his 1864 autobiography, Passages from the Life of a Philosopher
    - But he never build the machine as it wasn't profitable



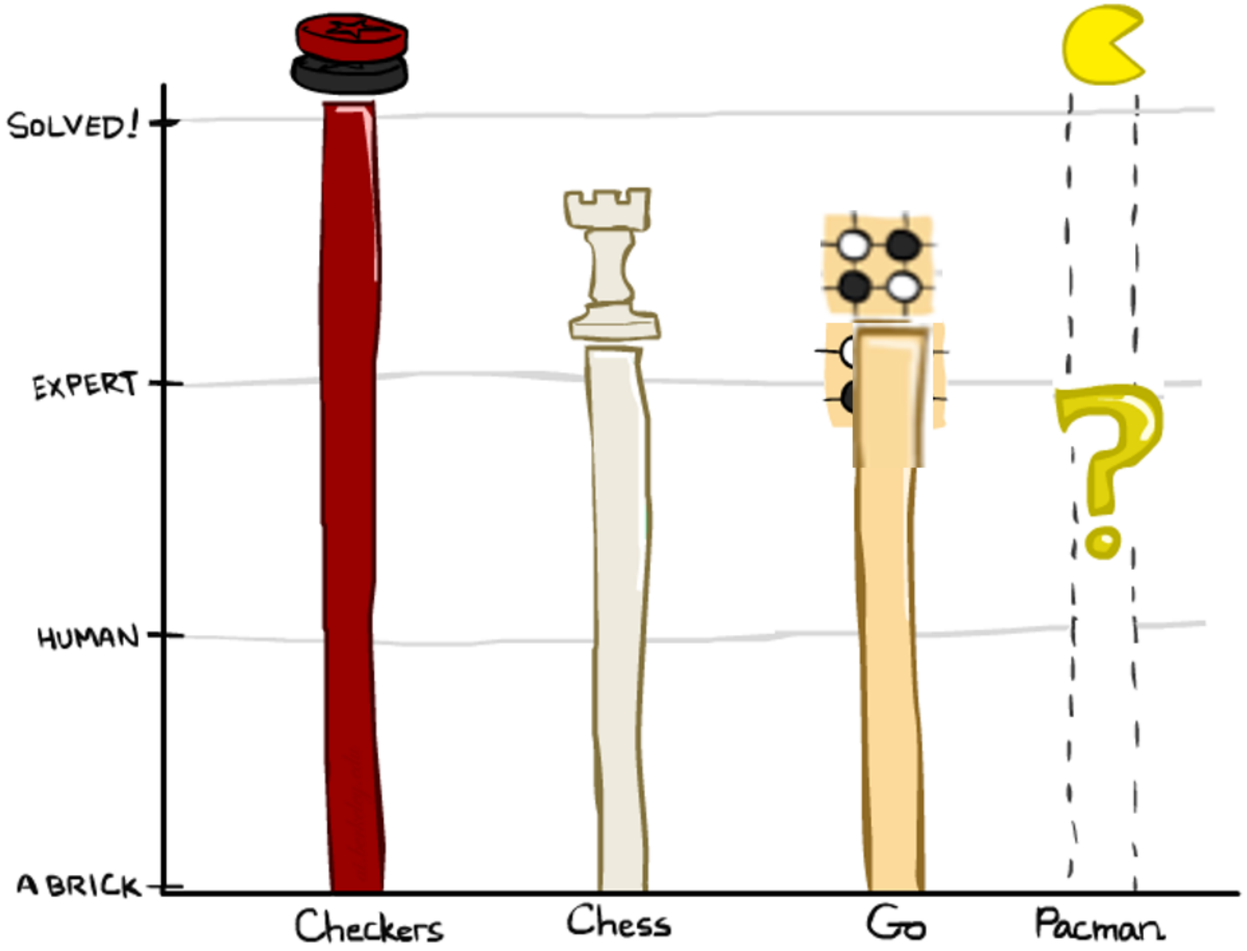


A BRIEF HISTORY

- Checkers:
  - 1950: First computer player.
  - 1959: Samuel's self-taught program.
  - 1994: First computer world champion: Chinook defeats Tinsley
  - 2007: Checkers solved! Endgame database of 39 trillion states
- Chess:
  - 1945-1960: Zuse, Wiener, Shannon, Turing, Newell & Simon, McCart
  - 1960s onward: gradual improvement under "standard model"
  - 1997: Deep Blue defeats human champion Gary Kasparov
  - 2021: Stockfish rating 3551 (vs 2870 for Magnus Carlsen).
- Go:
  - 1968: Zobrist's program plays legal Go, barely (b>300!)
  - 1968-2005: various ad hoc approaches tried, novice level
  - 2005-2014: Monte Carlo tree search -> strong amateur
  - 2016-2017: AlphaGo defeats human world champions

A BRIEF HISTORY

- Checkers:
  - 1950: First computer player.
  - 1959: Samuel's self-taught program.
  - 1994: First computer world champion: Chinook defeats Tinsley
  - 2007: Checkers solved! Endgame database of 39 trillion states
- Chess:
  - 1945-1960: Zuse, Wiener, Shannon, Turing, Newell & Simon, McCart
  - 1960s onward: gradual improvement under "standard model"
  - 1997: Deep Blue defeats human champion Gary Kasparov
  - 2021: Stockfish rating 3551 (vs 2870 for Magnus Carlsen).
- Go:
  - 1968: Zobrist's program plays legal Go, barely (b>300!)
  - 1968-2005: various ad hoc approaches tried, novice level
  - 2005-2014: Monte Carlo tree search -> strong amateur
  - 2016-2017: AlphaGo defeats human world champions



PACMAN - BEHAVIOUR FROM COMPUTATION

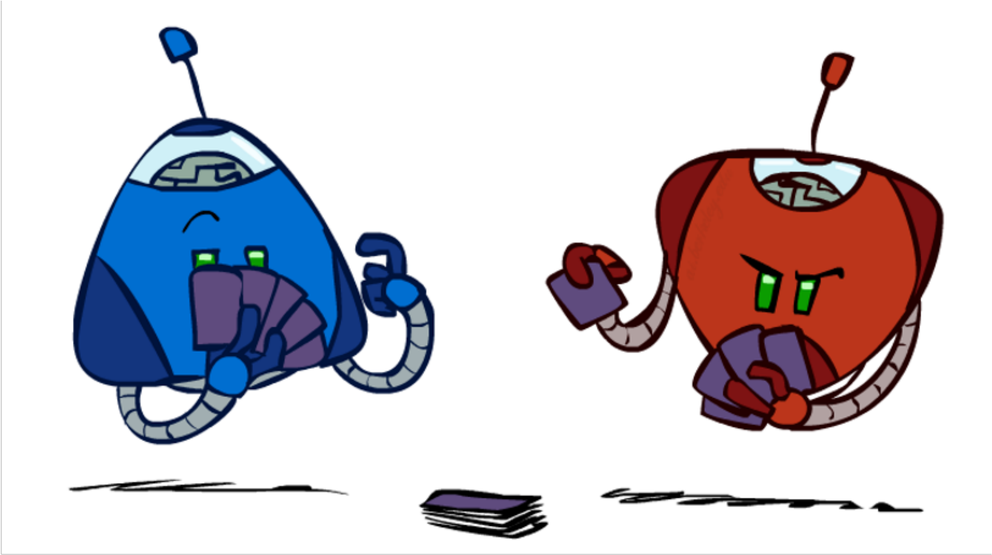
GAMES: MOTIVATION

- Games are a form of multi-agent environment
    - What do other agents do and how do they affect our success?
    - Cooperative vs. competitive multi-agent environments.
    - Competitive multi-agent environments give rise to adversarial search a.k.a. *games*

- Why study games?
    - Games are fun!
    - Historical role in AI
    - Studying games teaches us how to deal with other agents trying to foil our plans
    - *Huge* state spaces – Games are *hard*!
    - Nice, clean environment with clear criteria for success

TYPES OF GAMES

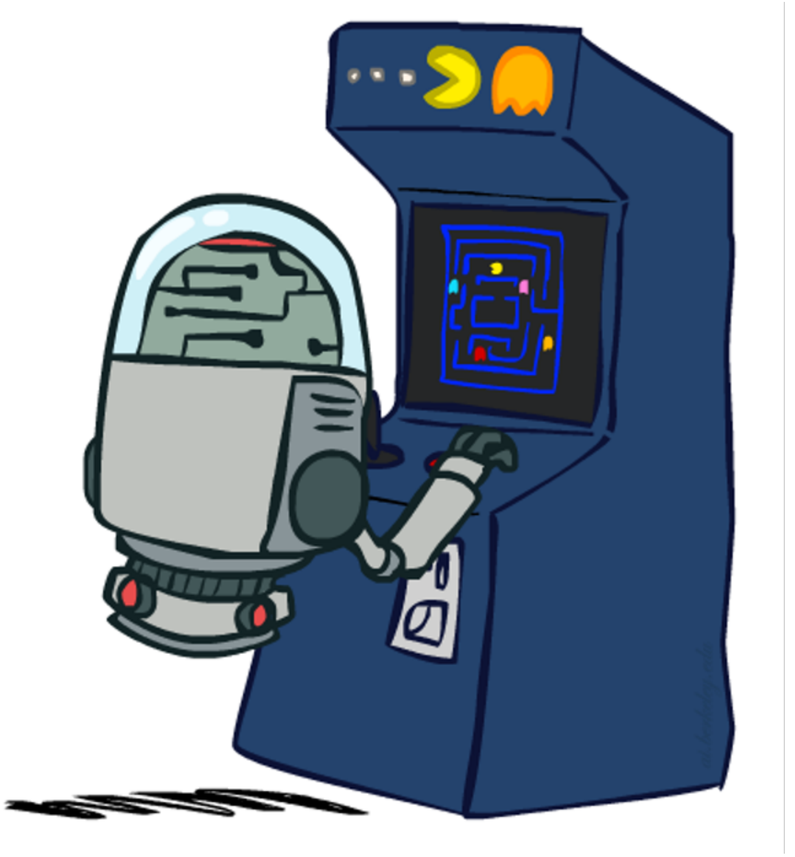
- Game = task environment with > 1 agent
- Axes:
    - Deterministic or stochastic?
    - Perfect information (fully observable)?
    - One, two, or more players?
    - Turn-taking or simultaneous?
    - Zero sum?



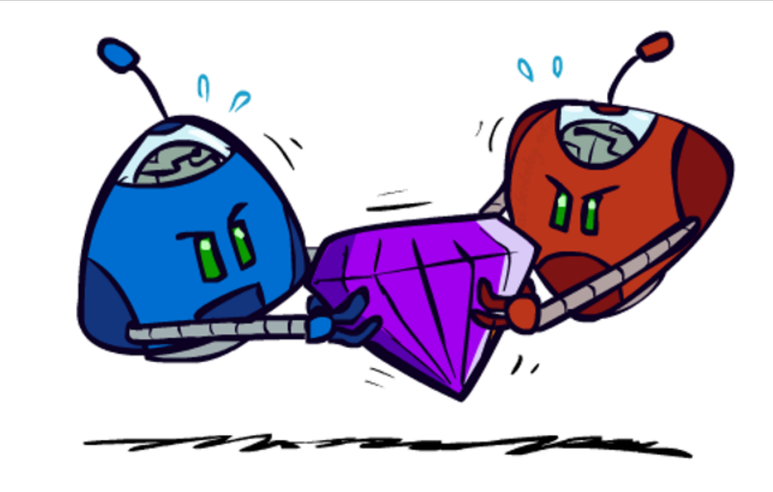
- Want algorithms for calculating a ***contingent plan*** (a.k.a. strategy or policy) which recommends a move for every possible eventuality

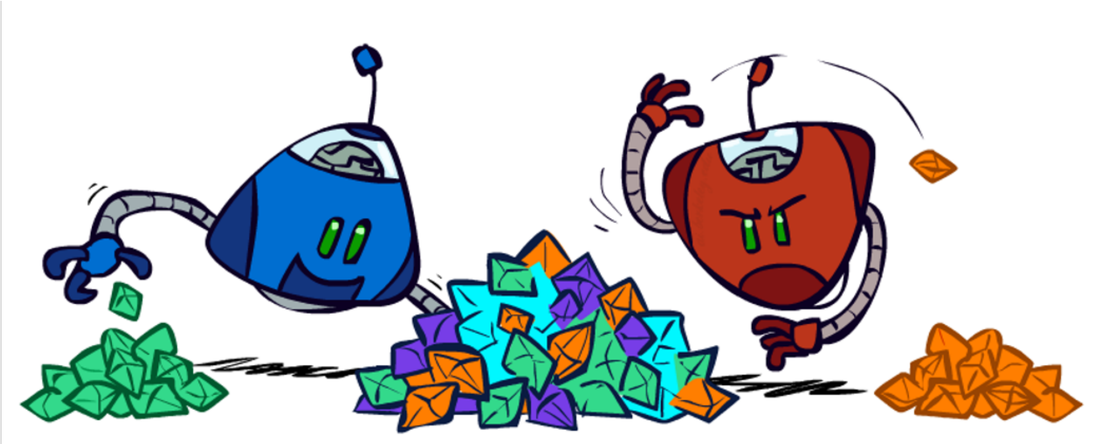
DETERMINISTIC GAMES

- Many possible formalisations, 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



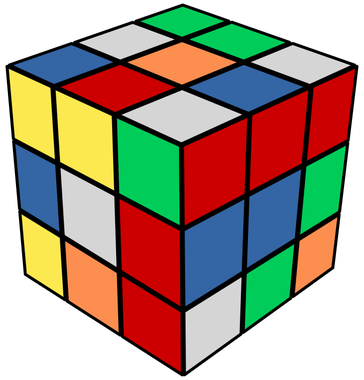


- Zero-Sum Games
  - Agents have ***opposite*** utilities
  - Pure competition:
  - One ***maximizes***, the other ***minimizes***

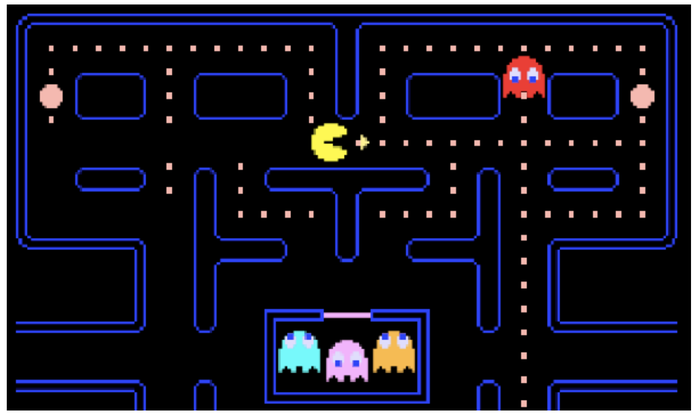
- General Games
  - Agents have ***independent*** utilities
  - Cooperation, indifference, competition, shifting alliances, and more are all possible

.

8-QUEENS WITH RANDOM RESTARTS

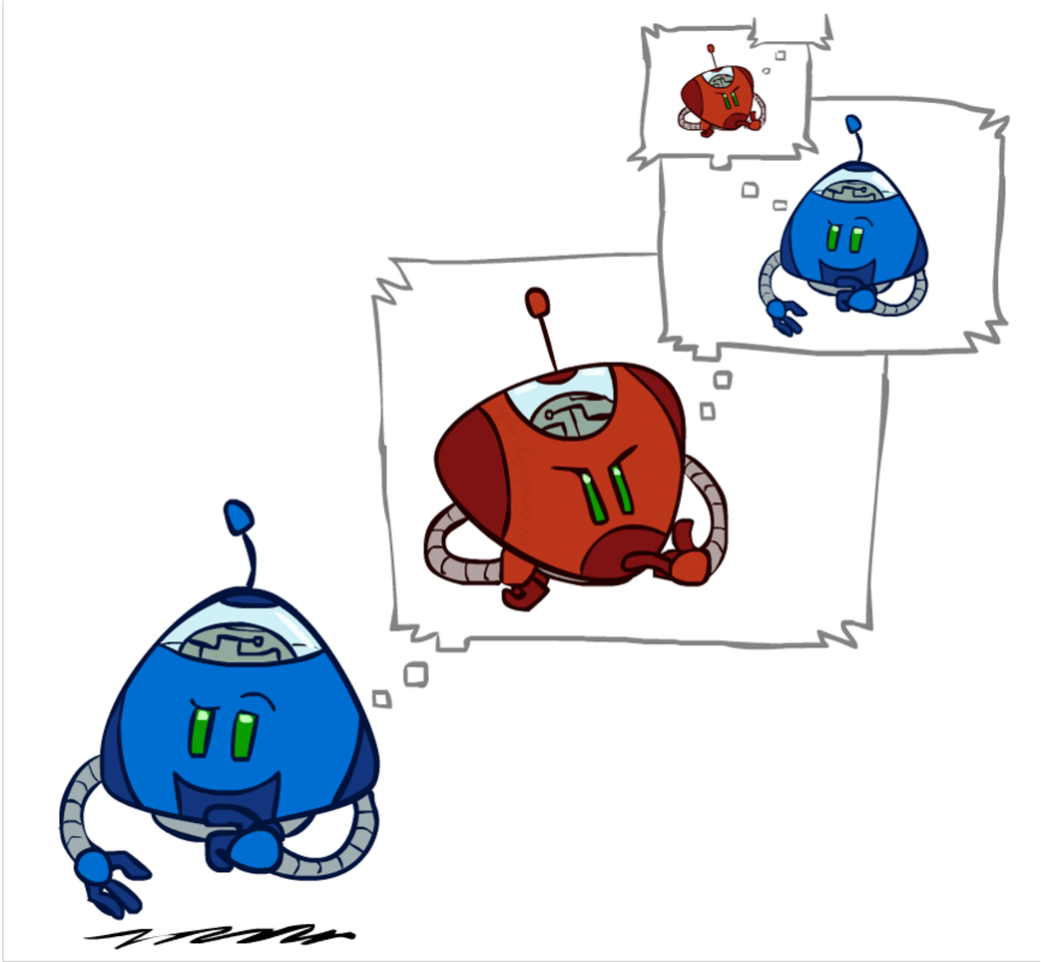


5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9



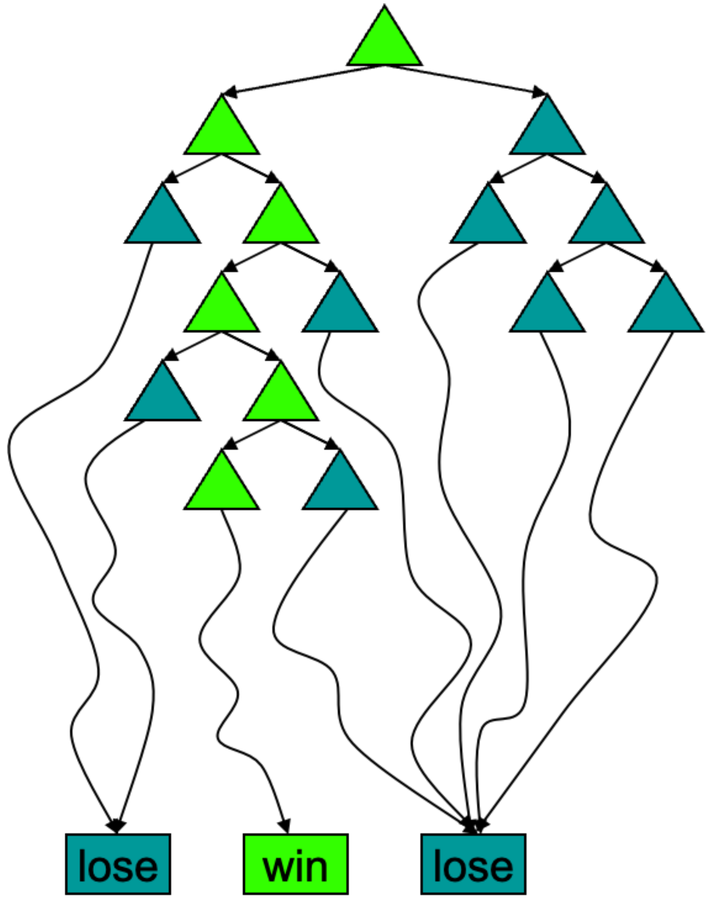
- "Unpredictable" opponent → specifying a move for every possible opponent reply
- Time limits → unlikely to find optimal solution, must approximate

ADVERSARIAL SEARCH

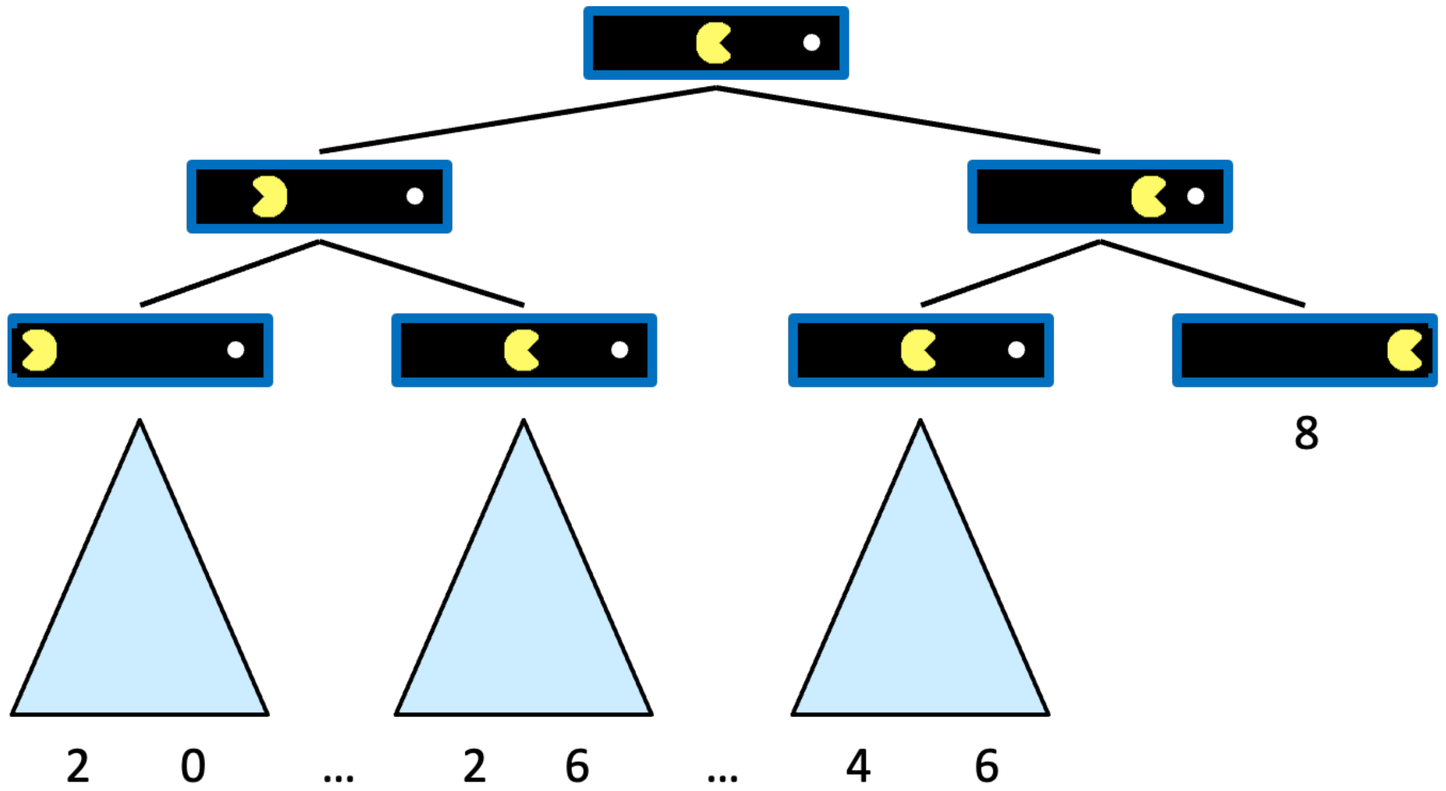


DETERMINISTIC SINGLE-PLAYER?

- Deterministic, single player, perfect information:
  - Know the rules
  - Know what actions do
  - Know when you win
  - E.g. Freecell, 8-Puzzle, Rubik's cube
- … it's just search!
- Slight reinterpretation:
  - Each node stores the best outcome it can reach
  - This is the maximal outcome of its children
  - Note that we don't store path sums as before
- After search, can pick move that leads to best node

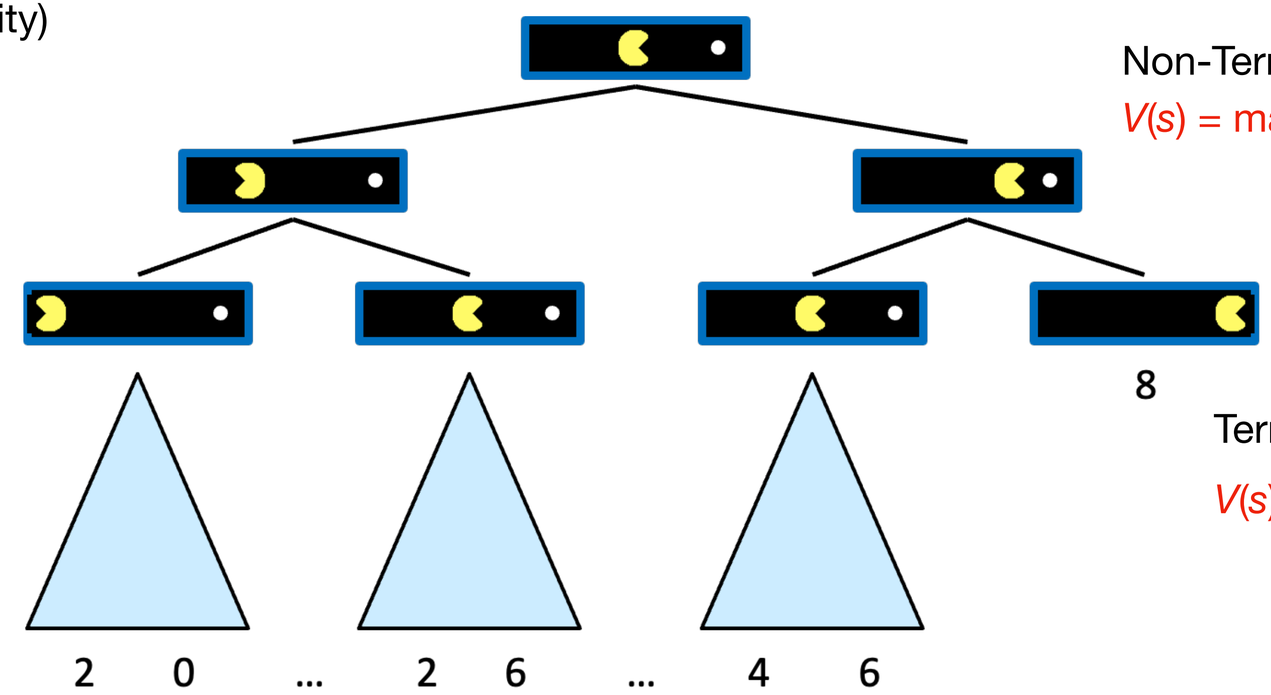


PACEMAN SINGLE -PLAYER SEARCH TREE



VALUE OF A STATE

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



Non-Terminal States:

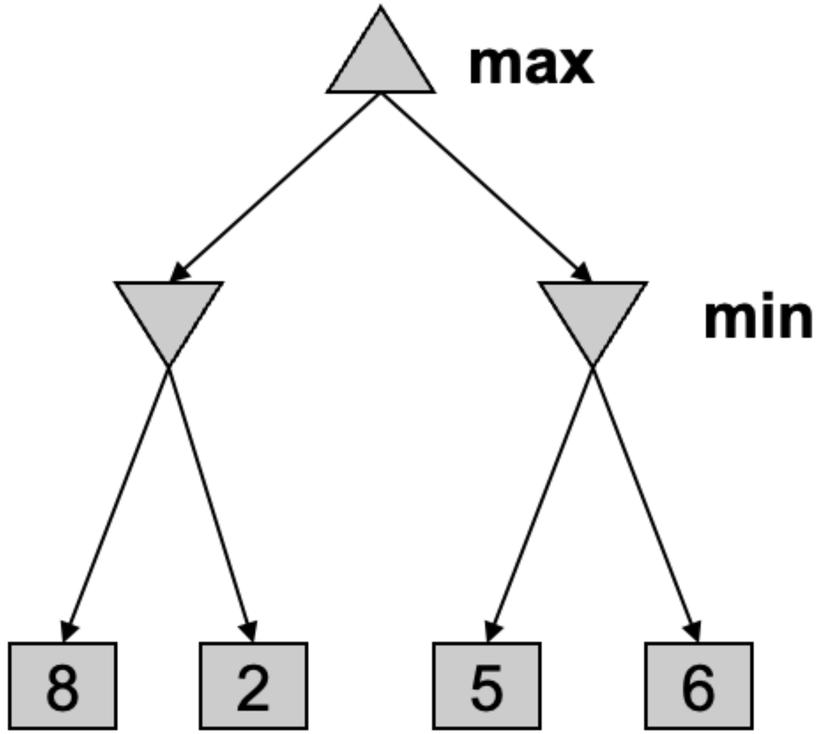
$V(s) = \max V(s')$

Terminal States:

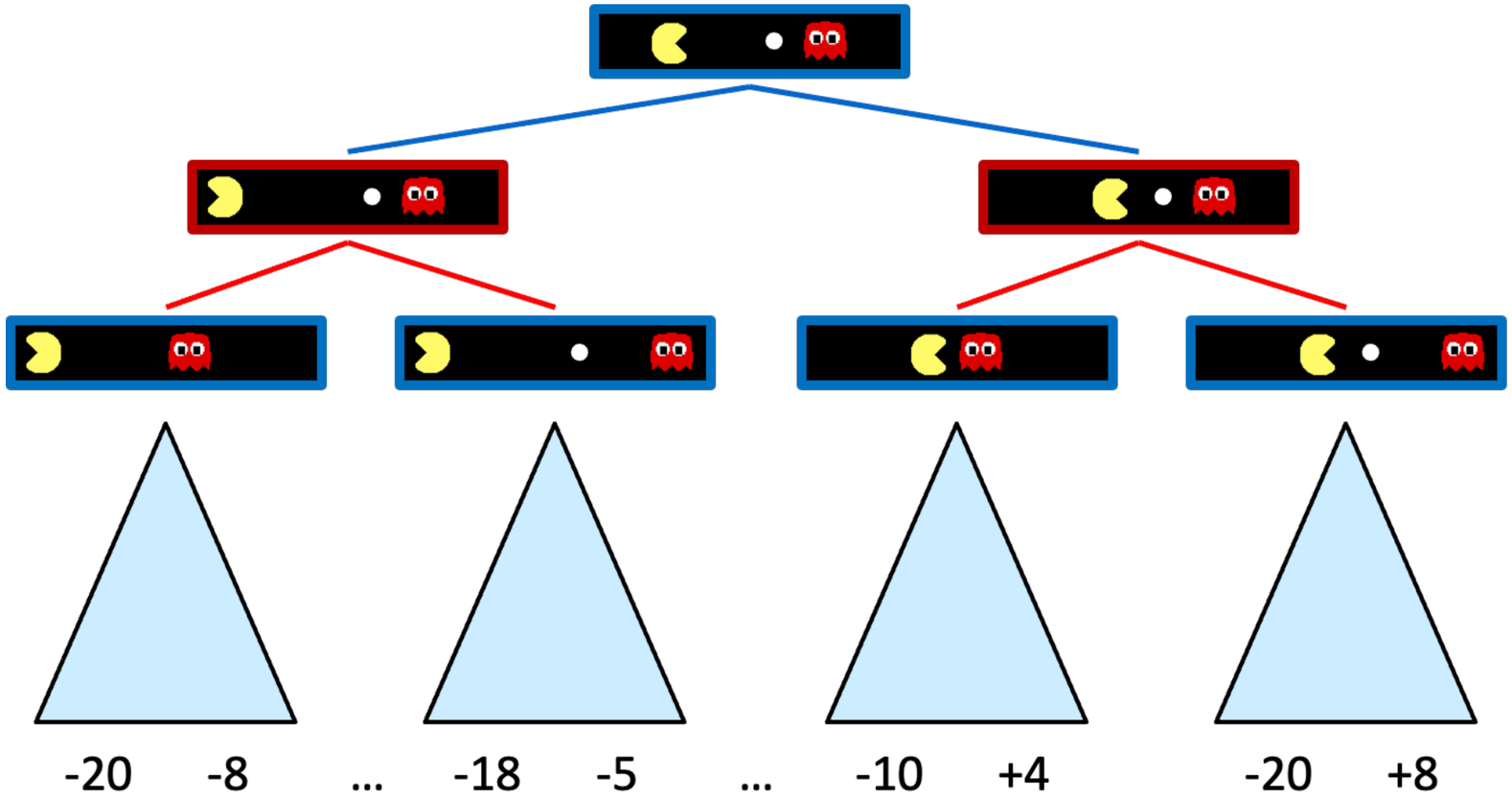
$V(s)$ = known

DETERMINISTIC TWO-PLAYER

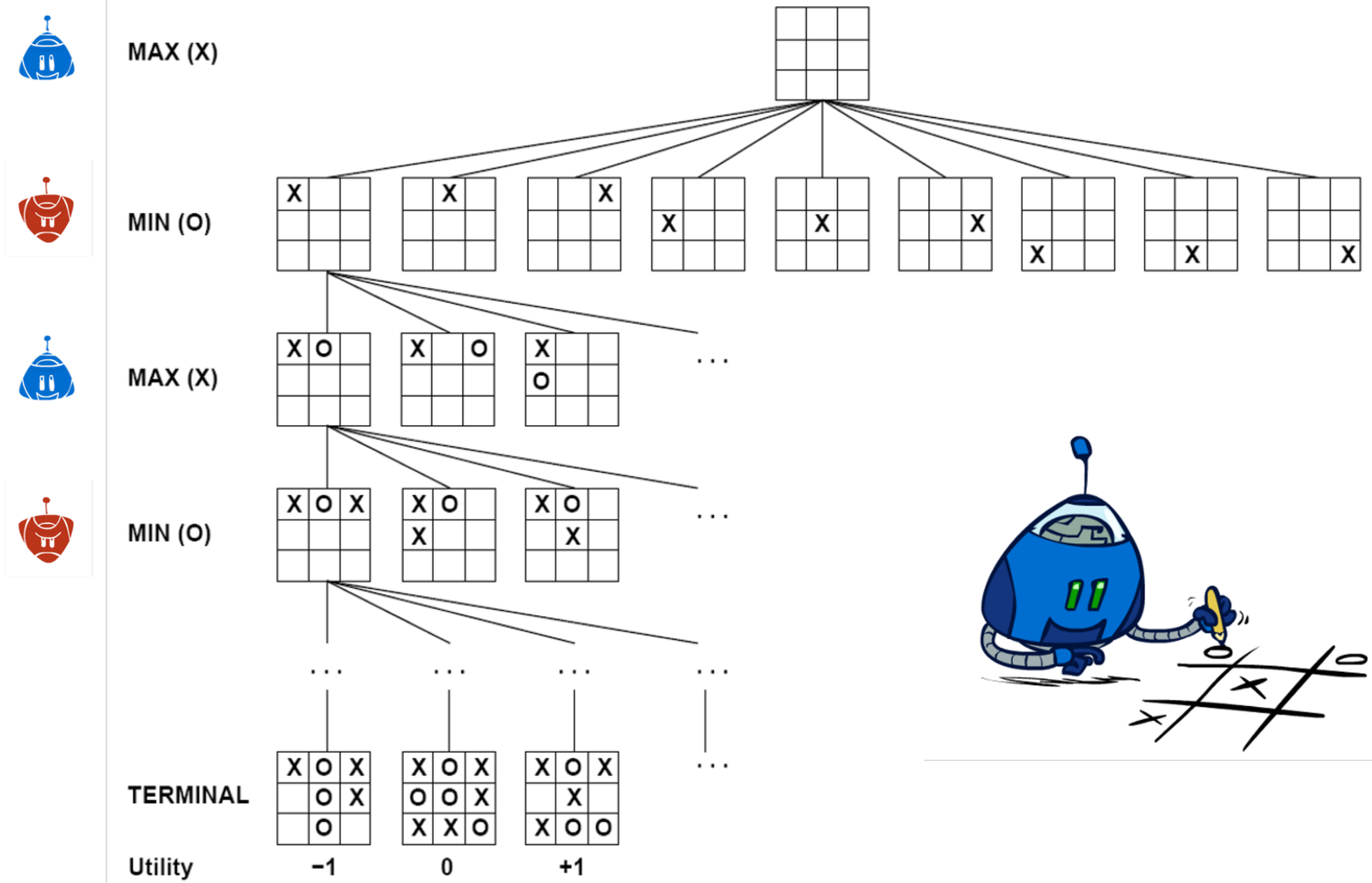
- E.g. tic-tac-toe, chess, checkers
- Minimax search
  - A state-space search tree
  - Players alternate
  - Each layer, or ply, consists of a round of moves
  - Choose move to position with highest minimax value = best achievable utility against best play
- Zero-sum games
  - One player maximizes result
  - The other minimizes result



ADVERSARIAL GAME TREE



TIC-TAC-TOE GAME TREE

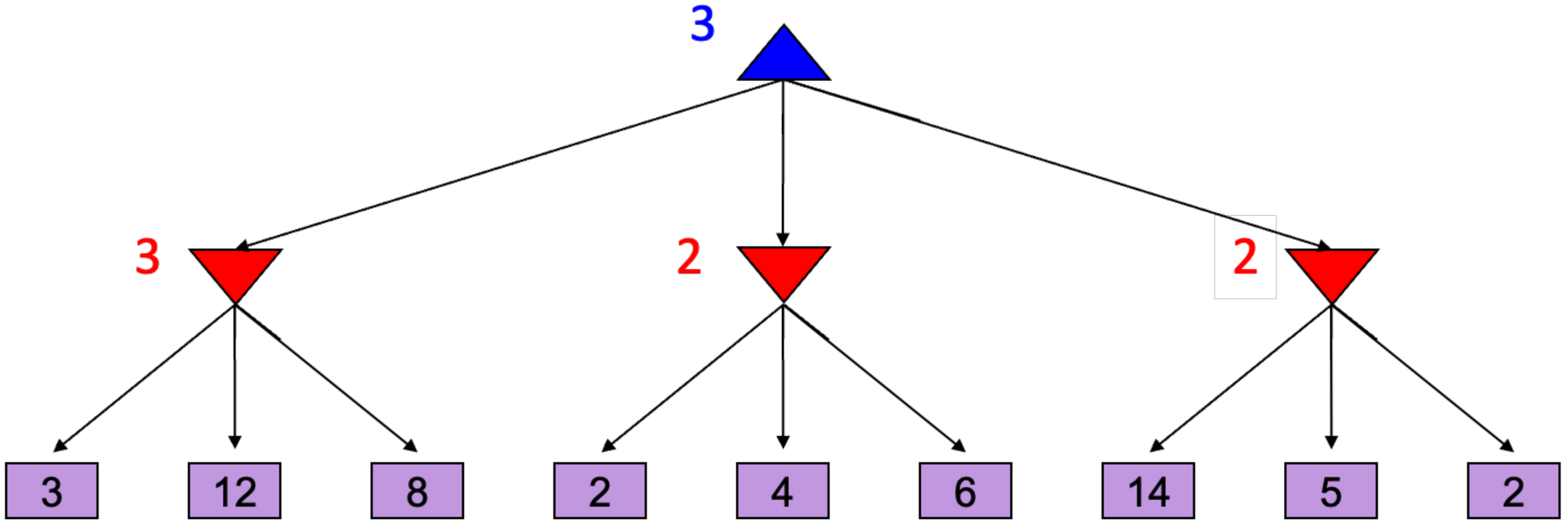


MINIMAX SEARCH

```
function minimax-decision(s) returns an action
    return the action a in Actions(s) with the highest
     minimax_value(Result(s,a))
```

```
function minimax_value(s) returns a value
    if Terminal-Test(s) then return Utility(s)
    if Player(s) = MAX then return max_{a in Actions(s)} minimax_value(Result(s,a))
    if Player(s) = MIN then return min_{a in Actions(s)} minimax_value(Result(s,a))
```

MINIMAX EXAMPLE



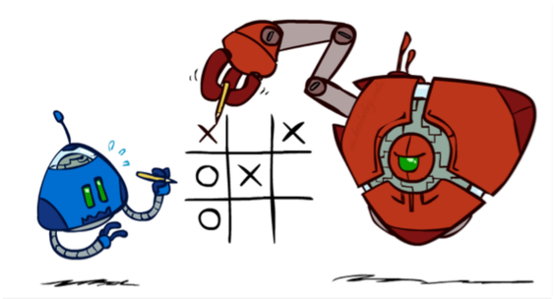
MINIMAX PROPERTIES

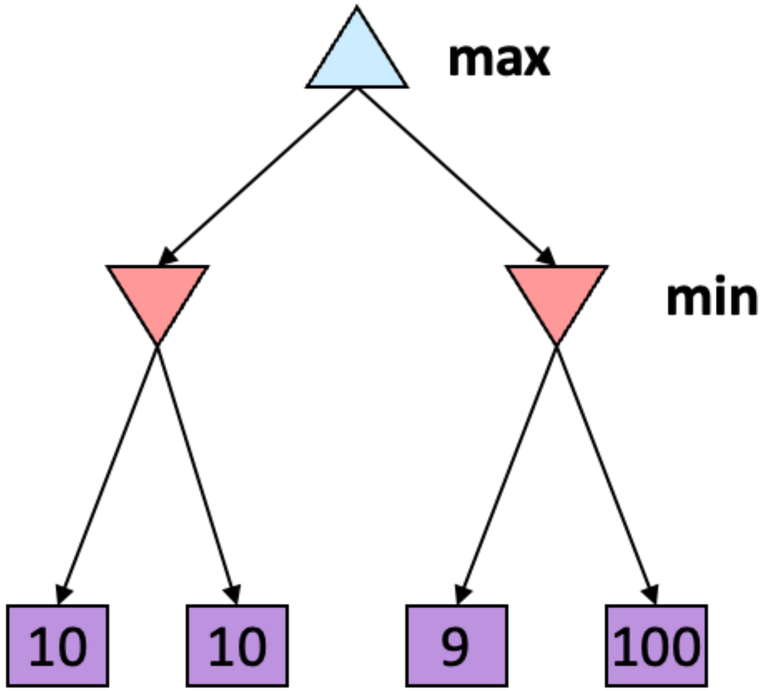
- Optimal against a perfect player. Otherwise?

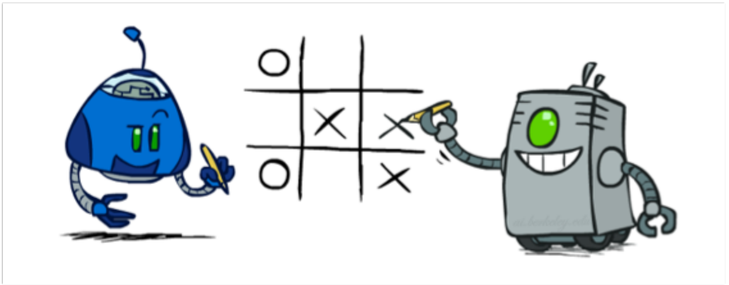
MINIMAX PROPERTIES

- Optimal against a perfect player. Otherwise?

.







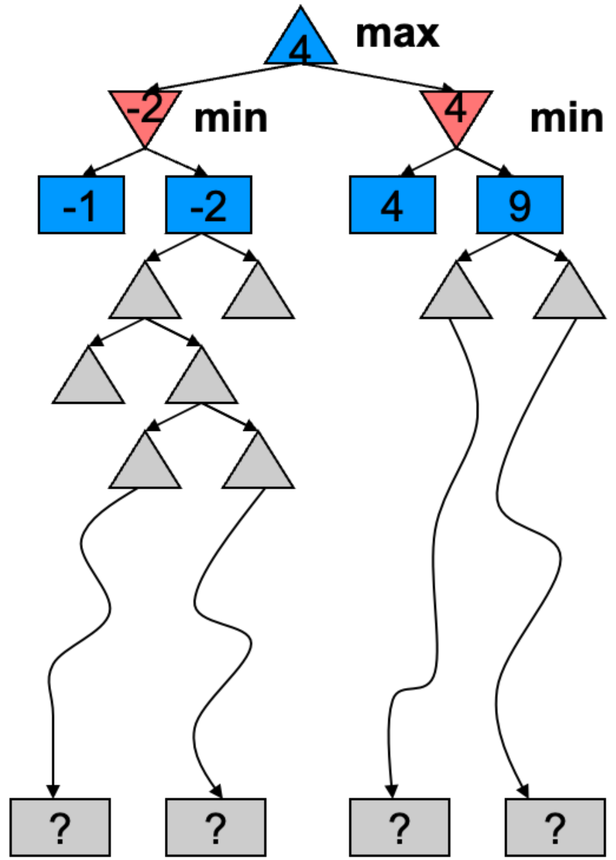
MINIMAX PROPERTIES

- Optimal against a perfect player. Otherwise?
- Time complexity?
  - $O(b^m)$
  - m = maximum depth of search tree, b = branching factor
- Space complexity?
  - O(bm)
- For chess, b ~ 35, m ~ 100
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?

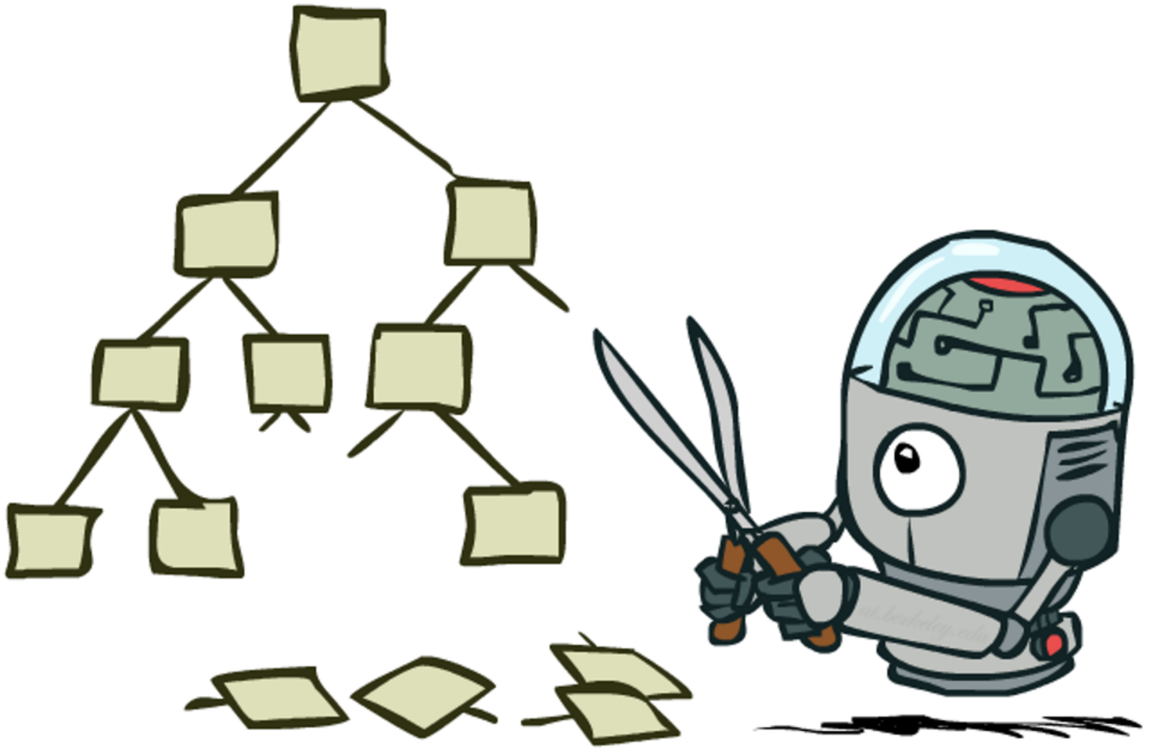
> The whole question of making an automaton play any game depended upon the possibility of the machine being able to ***represent all the myriads of combinations*** relating to it.
>
> - Charles Babbage

RESOURCE LIMITS

- Cannot search to leaves
- Limited search
  - Instead, search a limited depth of the tree
  - Replace terminal utilities with an eval function for non-terminal positions
- Guarantee of optimal play is gone
- More plies makes a BIG difference
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - α-β reaches about depth 8 – decent chess program



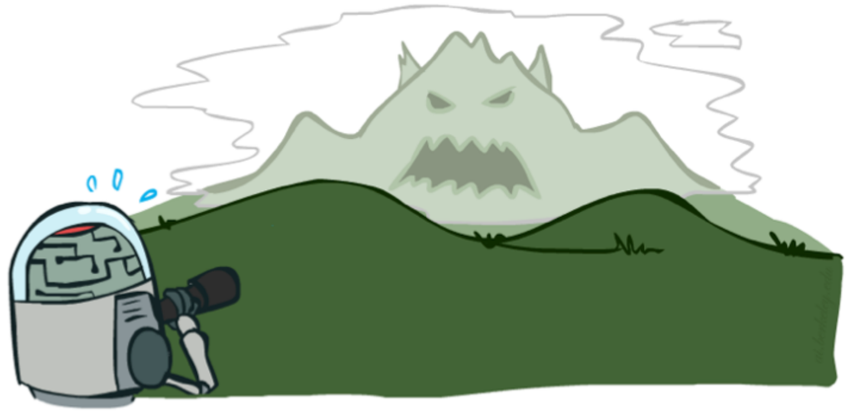
GAME TREE PRUNING

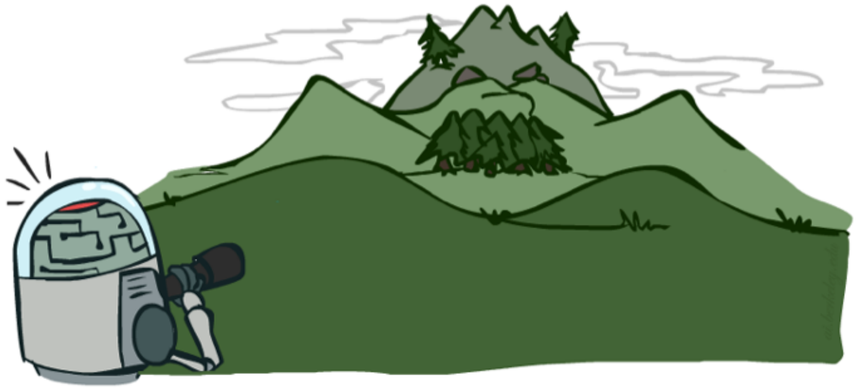


DEPTH MATTERS

- Evaluation functions are always imperfect
- Deeper search => better play (usually)
- Or, deeper search gives same quality of play with a less accurate evaluation function

.



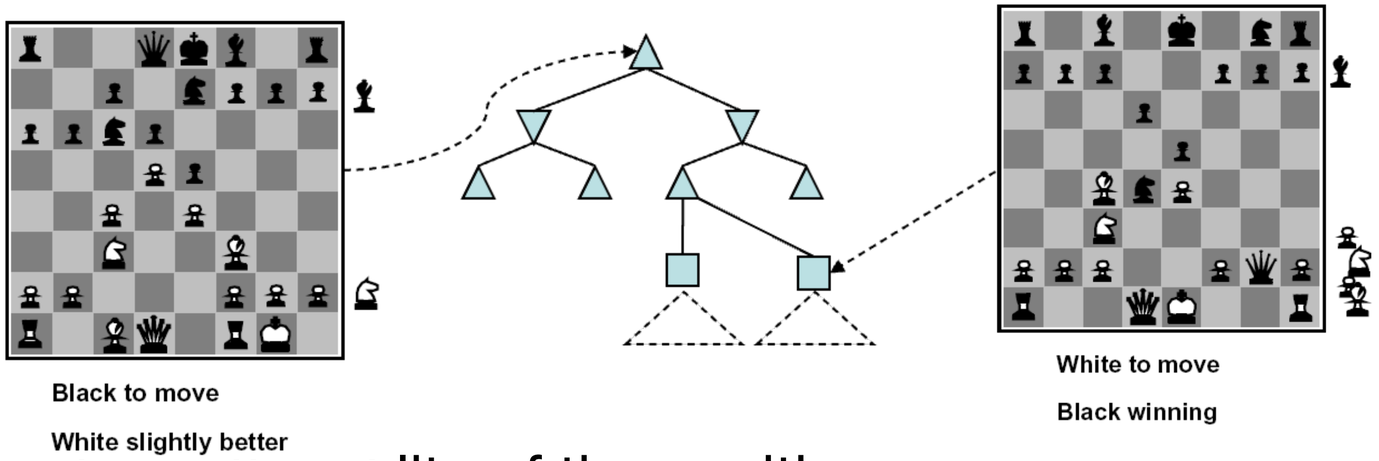


ITERATIVE DEEPENING

- Iterative deepening uses DFS as a subroutine:
    1. Do a DFS which only searches for paths of length 1 or less. (DFS gives up on any path of length 2)
    2. If “1” failed, do a DFS which only searches paths of length 2 or less.
    3. If “2” failed, do a DFS which only searches paths of length 3 or less.
- ….and so on.
- This works for single-agent search as well!
- Why do we want to do this for multiplayer games?

EVALUATION FUNCTION

- Function which scores non-terminals



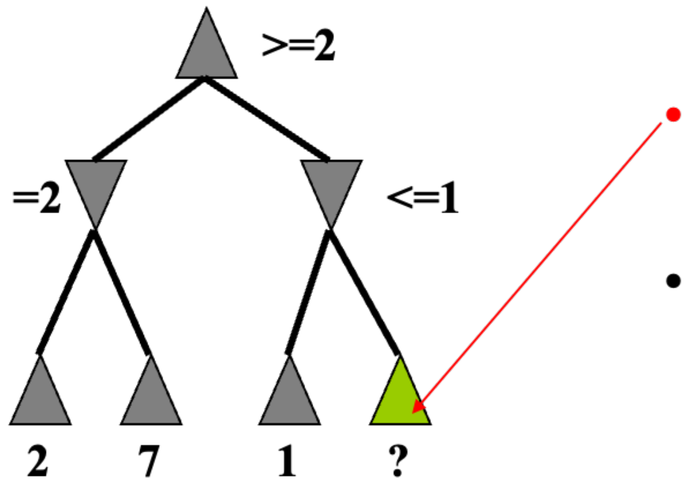
- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features:

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

  - e.g. $f_1(s)$ = (num white queens – num black queens), etc.
  - Or a more complex nonlinear function (e.g., NN) trained by self-play RL

α-β PRUNING

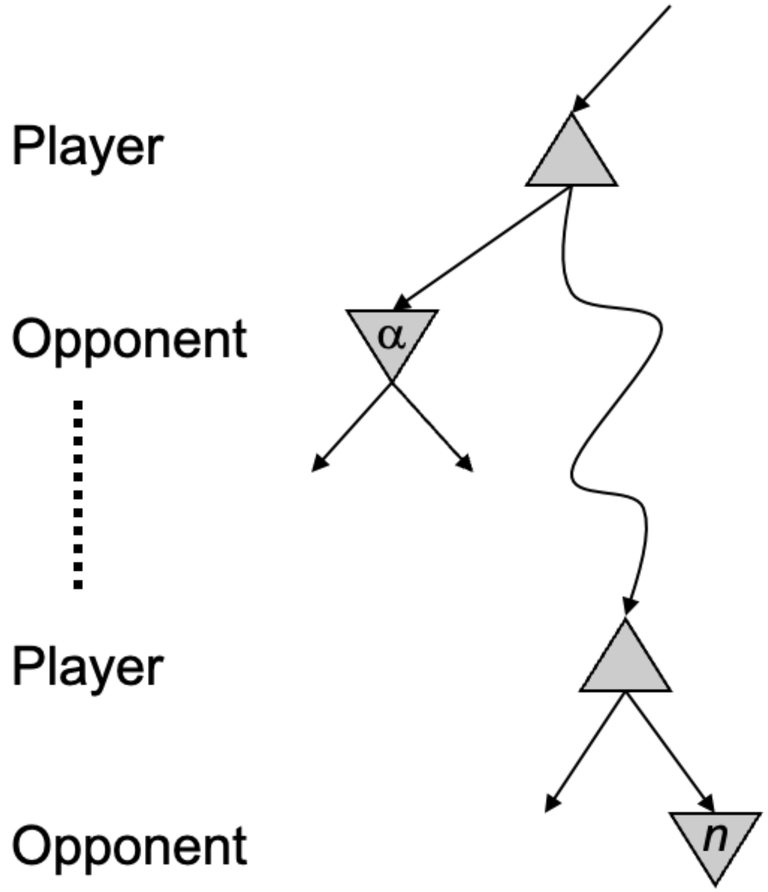
- A way to improve the performance of the Minimax Procedure
- Basic idea: “If you have an idea which is surely bad, don’t take the time to see how truly awful it is” ~ Pat Winston



- We don’t need to compute the value at this node.
- No matter what it is it can’t effect the value of the root node.

α-β PRUNING

- General case (pruning children of MIN node)
  - We're computing the MIN-VALUE at some node *n*
  - We're looping over *n*'s children
  - *n*'s estimate of the children's min is dropping
  - Who cares about *n*'s value? MAX
  - Let ***α*** be the best value that MAX can get so far at any choice point along the current path from the root
  - If *n* becomes worse than ***α***, MAX will avoid it, so we can prune *n*'s other children (it's already bad enough that it won't be played)
- Pruning children of MAX node is symmetric
  - Let **β** be the best value that MIN can get so far at any choice point along the current path from the root



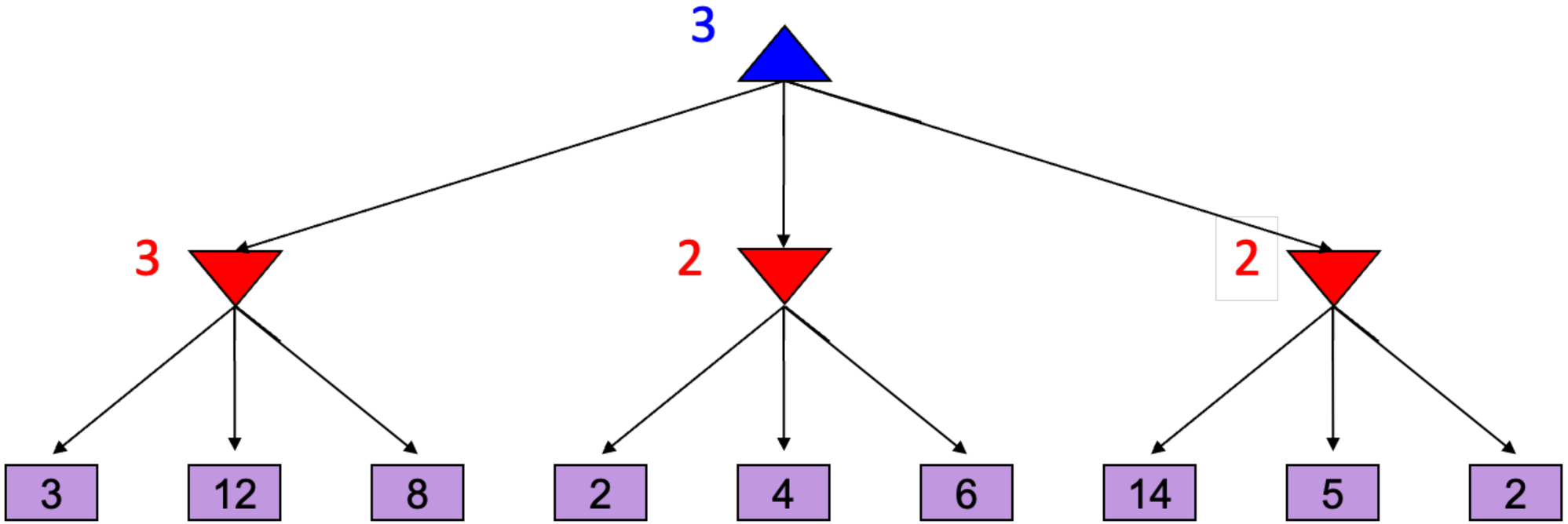
α-β PRUNING ALGORITHM

α: MAX's best option on path to root
β: MIN's best option on path to root

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

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

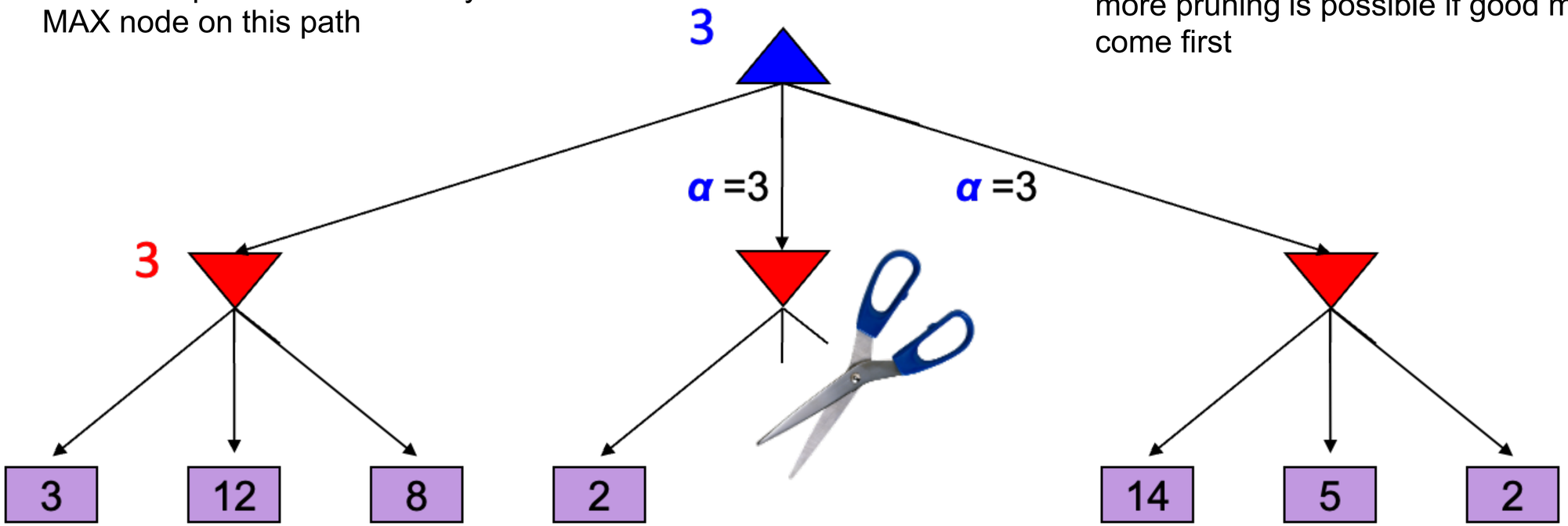
MINIMAX EXAMPLE



α-β PRUNING EXAMPLE

α = best option so far from any MAX node on this path

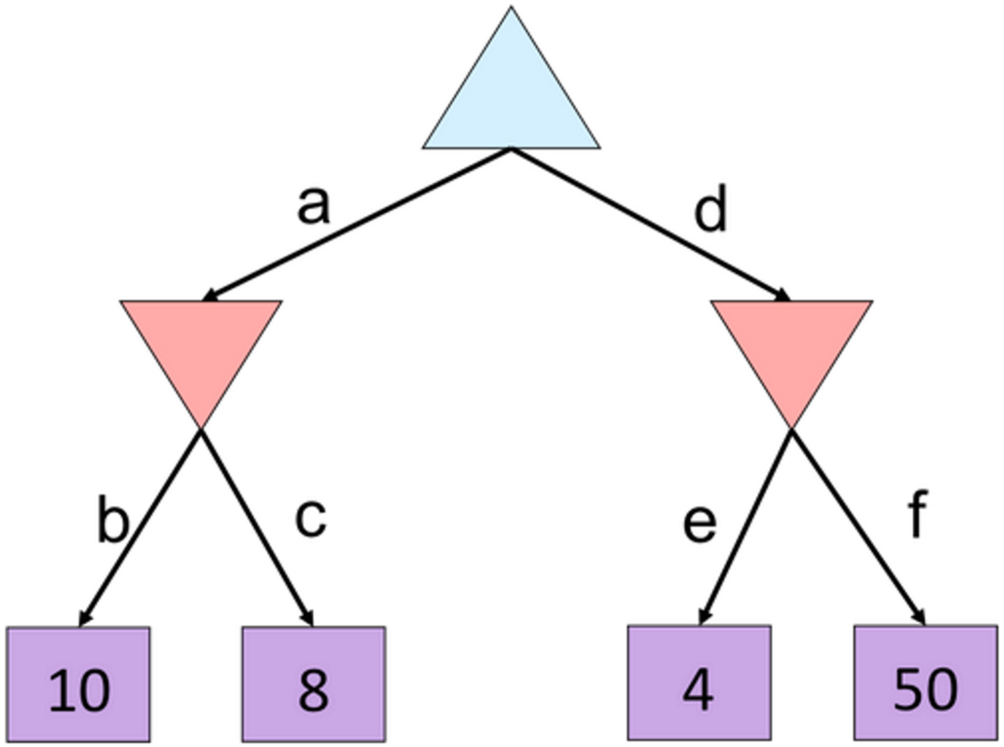
The order of generation matters: more pruning is possible if good moves come first



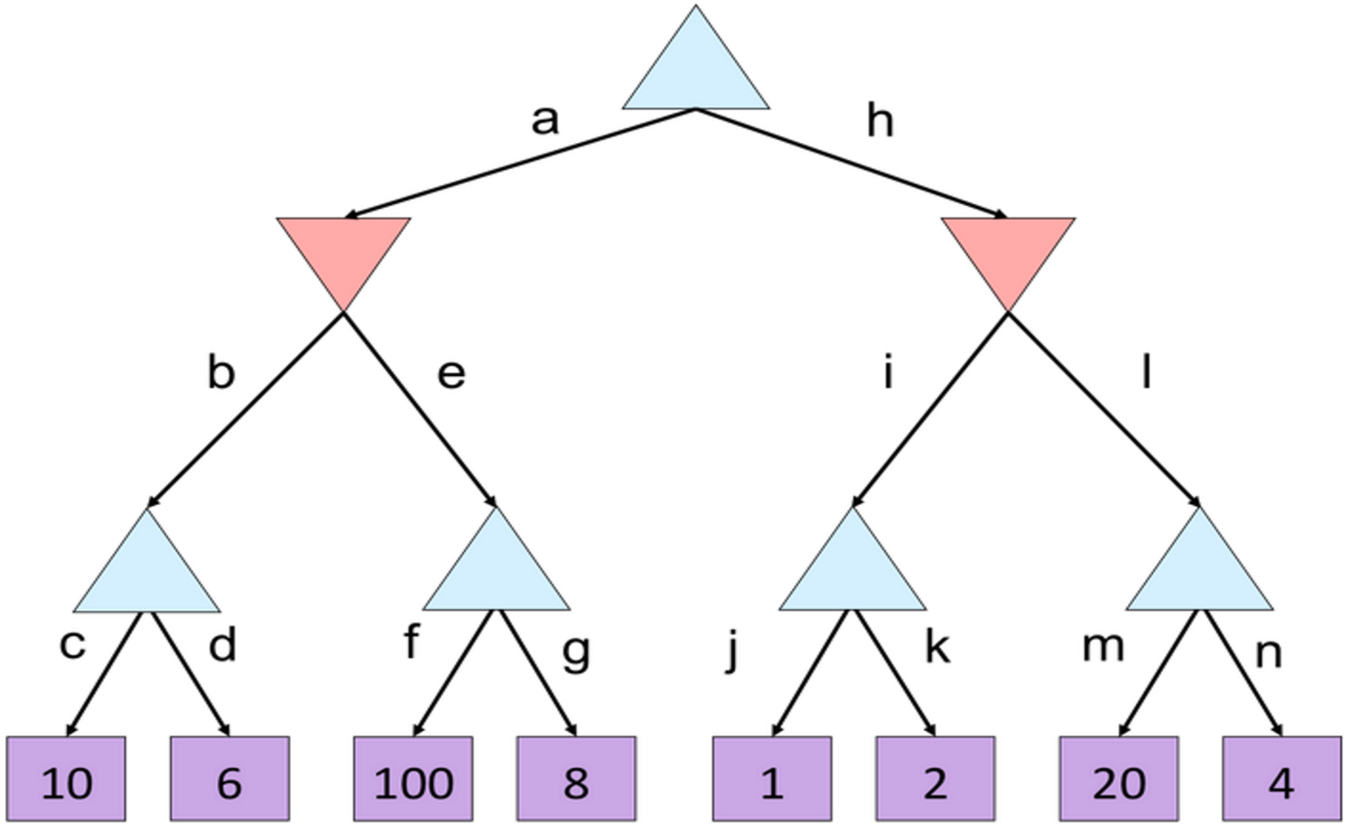
α-β PRUNING PROPERTIES

- Pruning has no effect on final result
- Good move 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!
- A simple example of metareasoning, here reasoning about which computations are relevant
- For chess: only 35^{50} instead of 35^{100}!! Yaaay!!!!!

α-β PRUNING QUIZ



α-β PRUNING QUIZ 2



SUMMARY

- Games require decisions when optimality is impossible
    - Bounded-depth search and approximate evaluation functions
- Games force efficient use of computation
    - Alpha-beta pruning
- Game playing has produced important research ideas
    - Reinforcement learning
    - Iterative deepening
    - Rational metareasoning