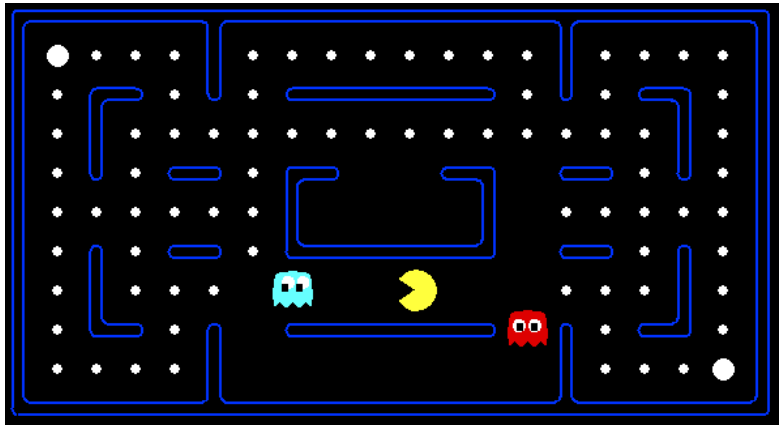


Adversarial Search

Minimax

Artificial Intelligence

Jay Urbain, Ph.D.

Credits: Stuart Russel, Peter Norvig, AIMA

Dan Klein, Pieter Abbeel, University of California, Berkeley

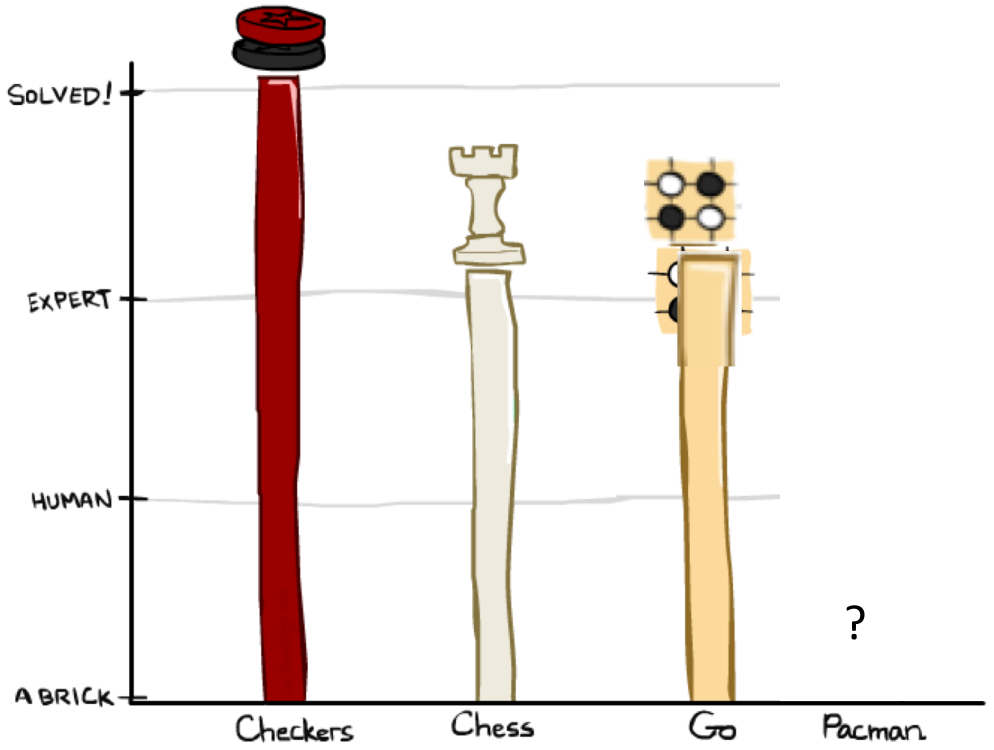
Dilbert Teaches Game Theory

Outline

- Optimal decisions
- Minimax
- α-β pruning
- Imperfect, real-time decisions

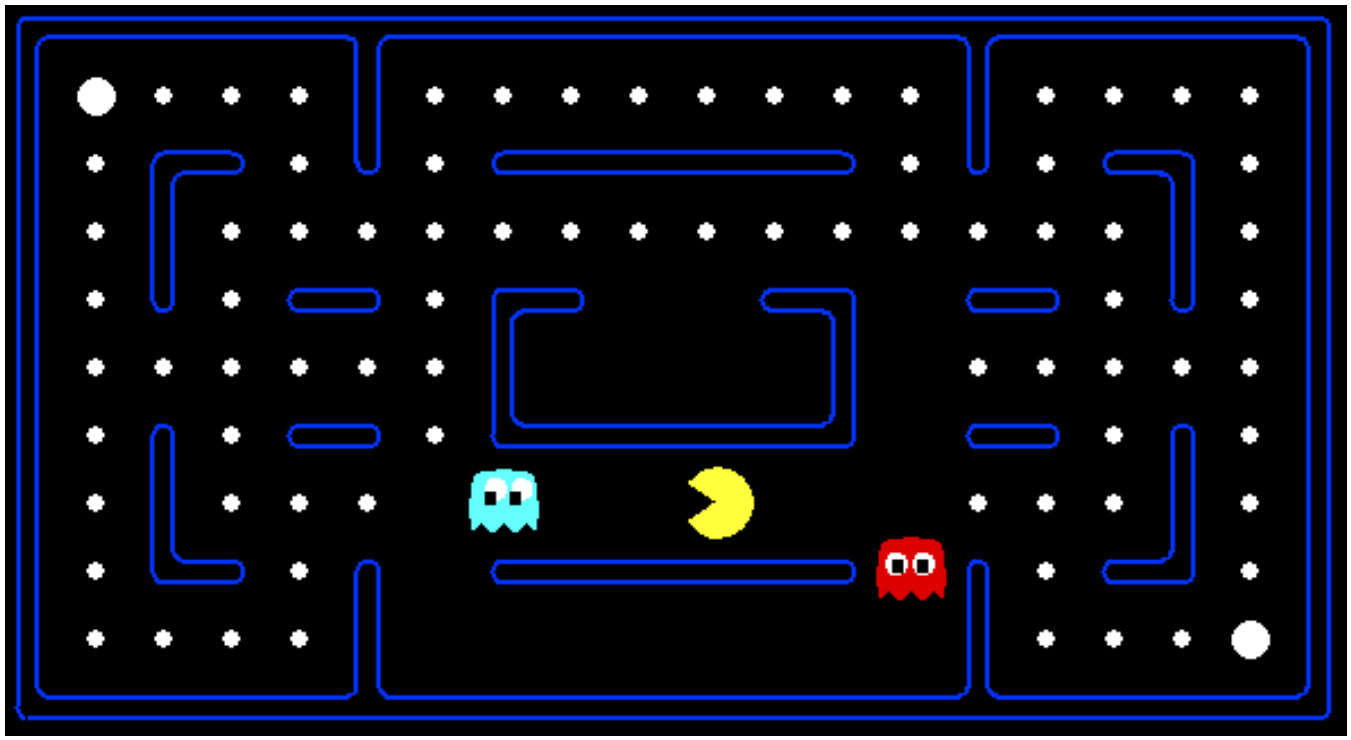
Game Playing State-of-the-Art

- **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?**



"It always seems impossible until it's done." — Nelson Mandela

Behavior from Computation



Demo
python pacman.py --frameTime 0 -p ReflexAgent -k 2

Games
(Adversarial Search Problem)

- Multiagent environments
  - Any given agent needs to consider the actions of other agents and how they affect their welfare.
- Unpredictability of other agents can introduce many possible contingencies into the agent's problem-solving process.
- *Cooperative* and *competitive* multi-agent environments?
- Competitive environments in which the agents' goals are often in conflict - give rise to *adversarial search problems* – often called *games*.

Games in AI

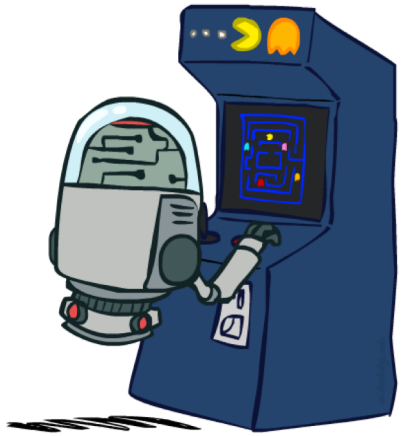
- Game theory (Economics) – views multiagent environments a game regardless of whether the agents are *cooperative* or *competitive*.

- Environments:
  - Deterministic or stochastic?
  - Turn-taking?
  - One, two, or more players?
  - Zero-sum?
  - Fully or partially observable?

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

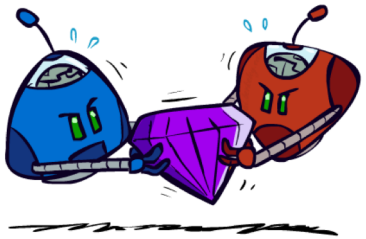
Deterministic Games

- 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

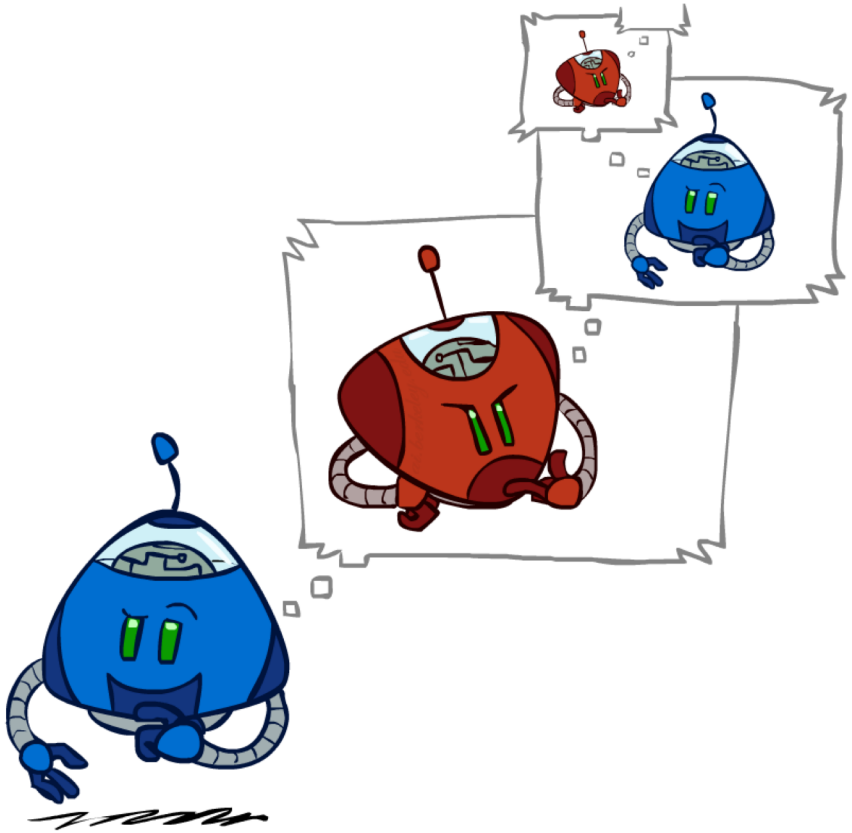
Zero-Sum Games



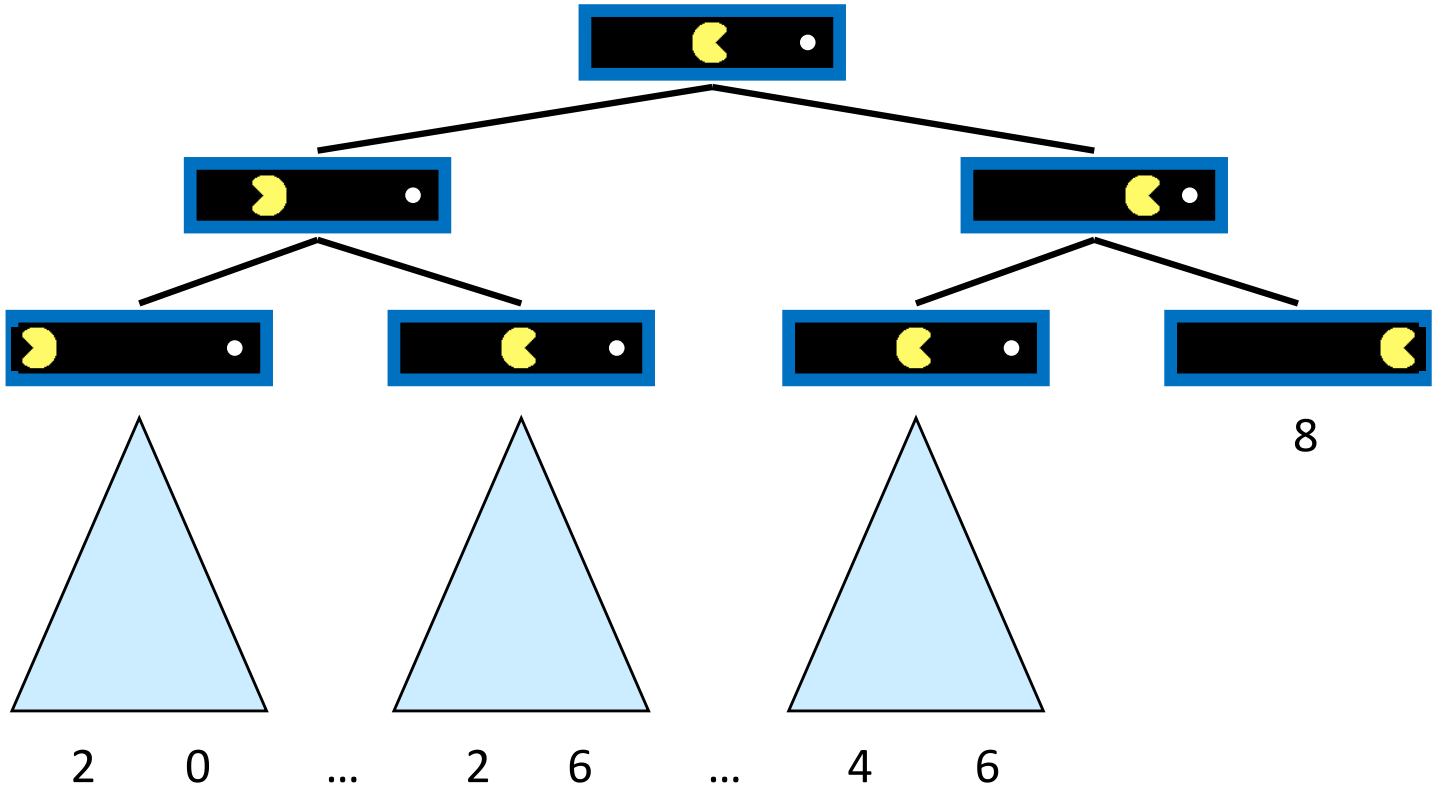
- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition

- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, and more are all possible

Adversarial Search

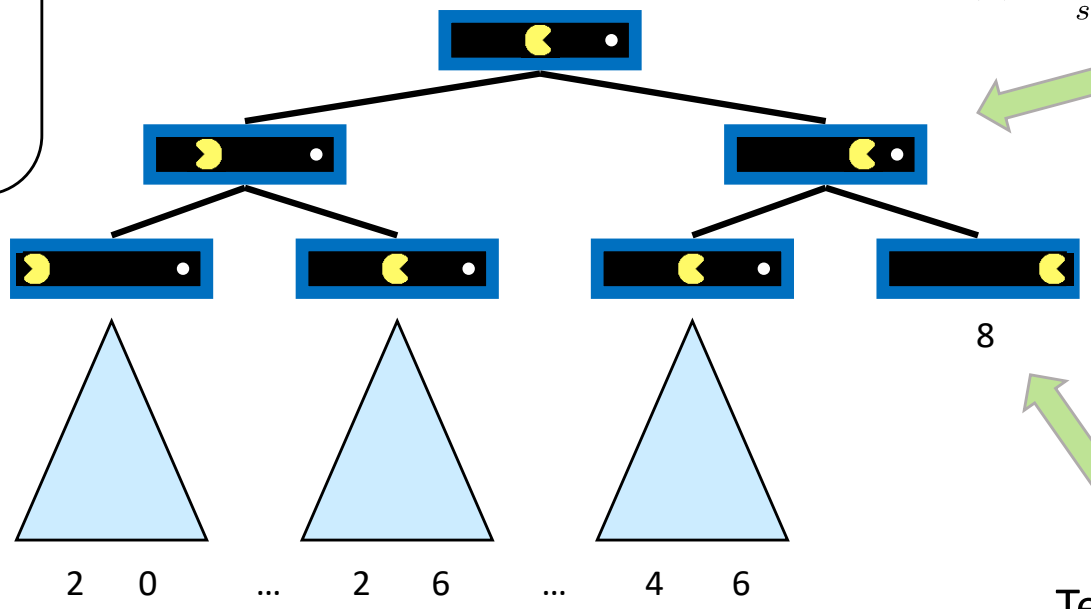


Single-Agent Trees
(Does not consider adversaries)



Value of a State

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



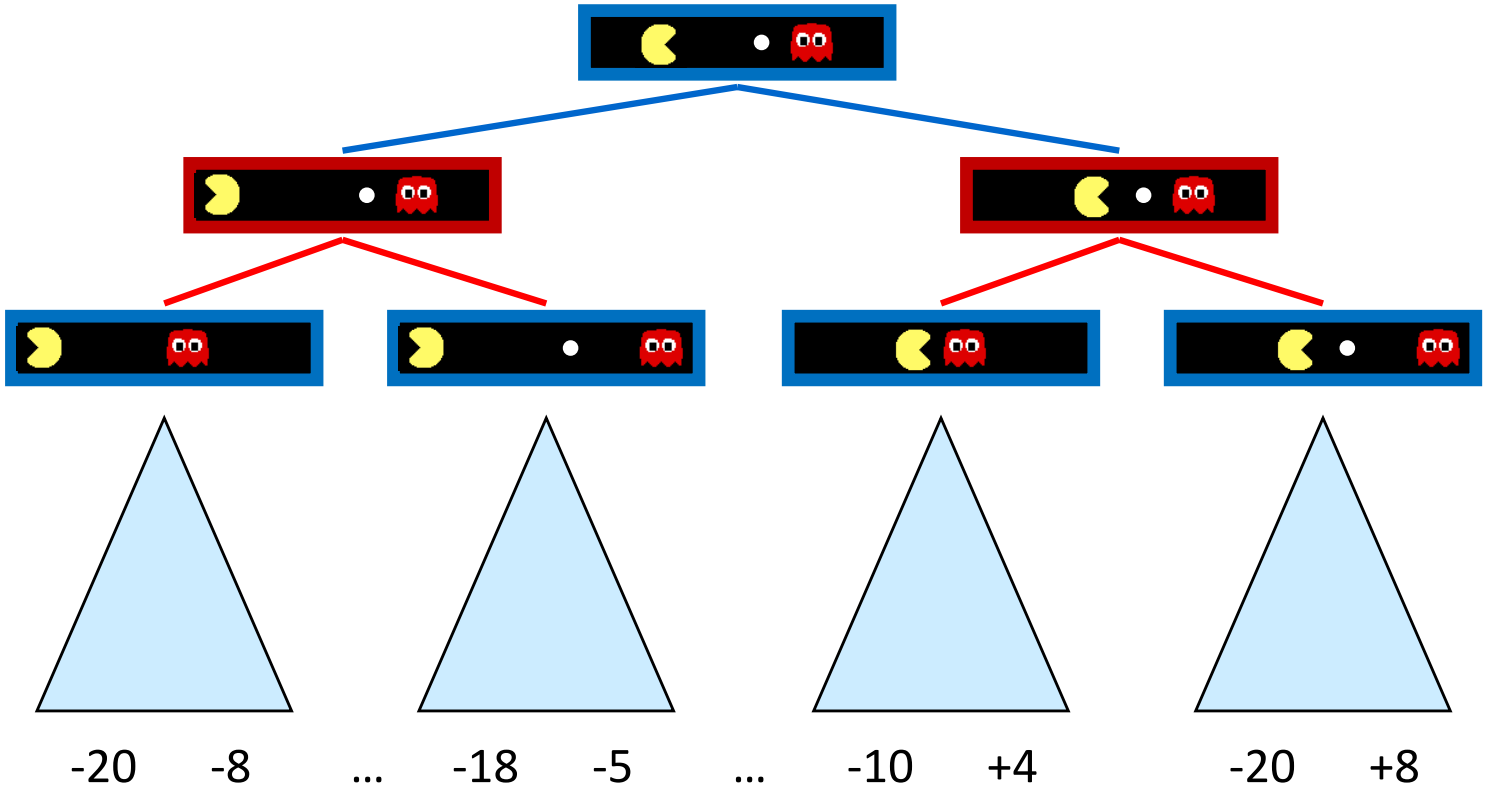
Non-Terminal

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

Terminal States:

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

Adversarial Game Trees (considers adversaries)



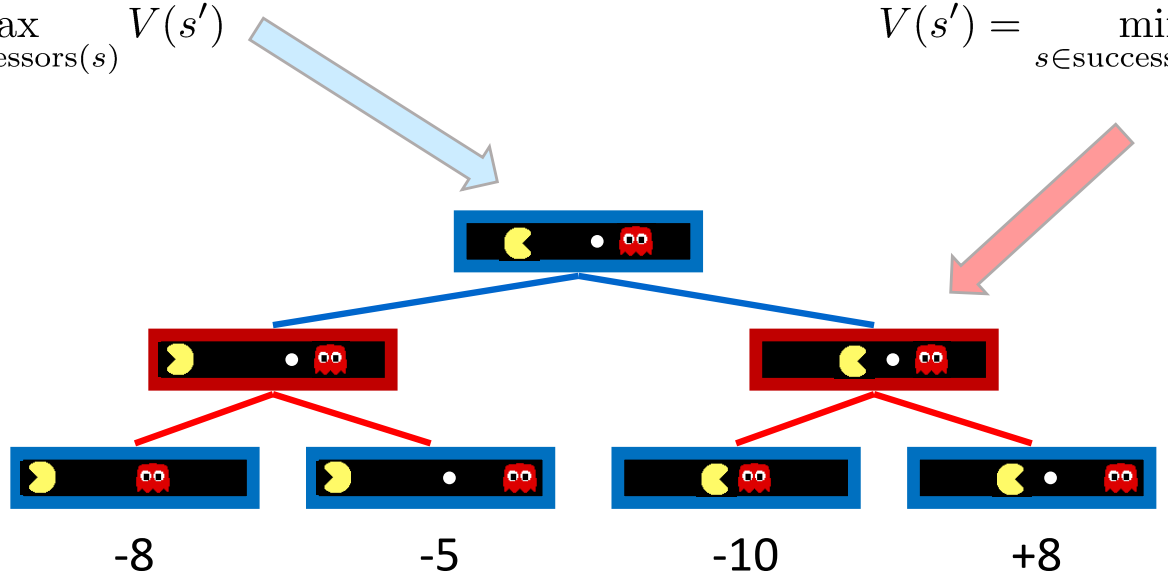
Minimax Values

States Under Agent's Control:

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

States Under Opponent's Control:

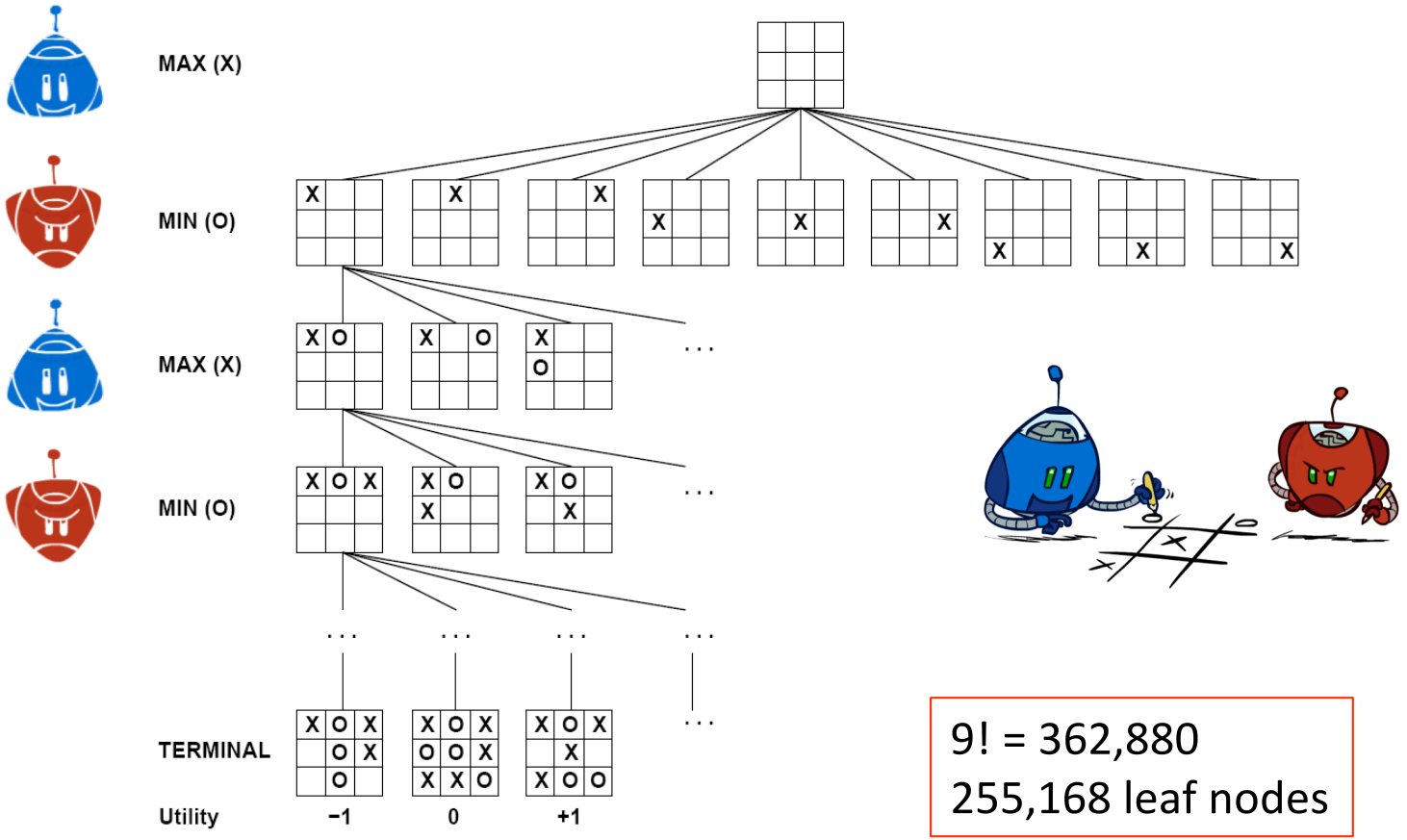
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

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

Tic-Tac-Toe Game Tree

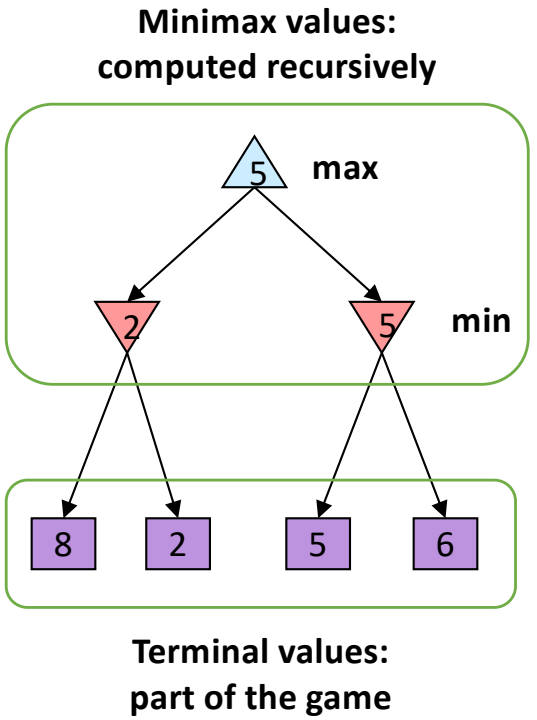


9! = 362,880
255,168 leaf nodes

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

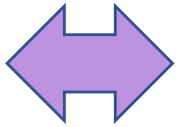
```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

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

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

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 (with 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)
```

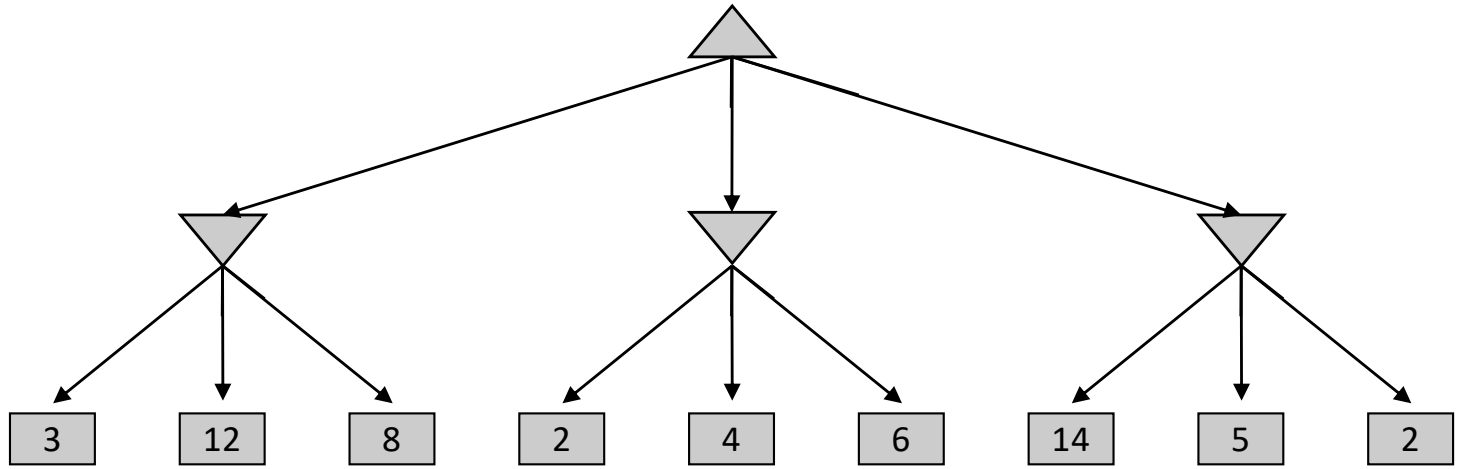
```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

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

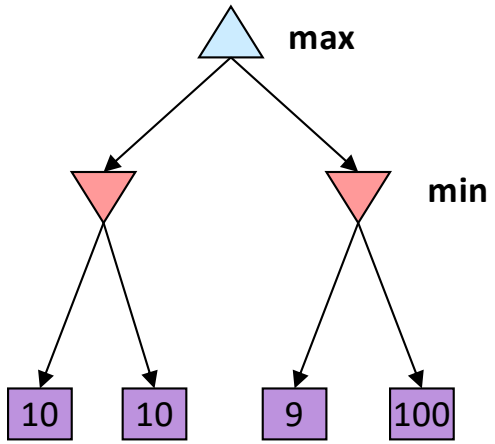
$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

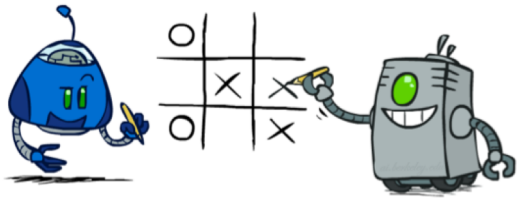
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

Minimax Example



Minimax Properties



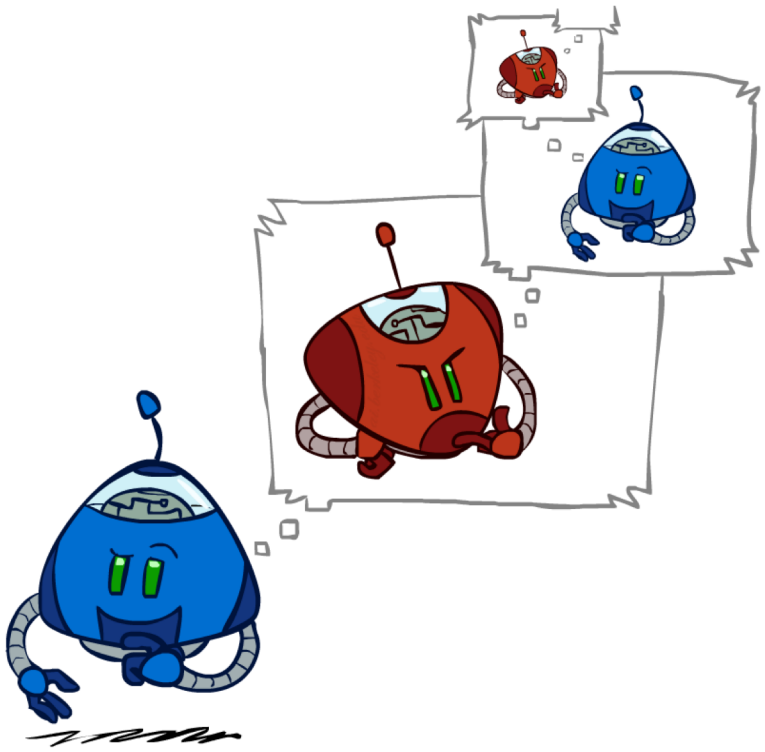


Optimal against a perfect player. Otherwise?

Minimax Efficiency

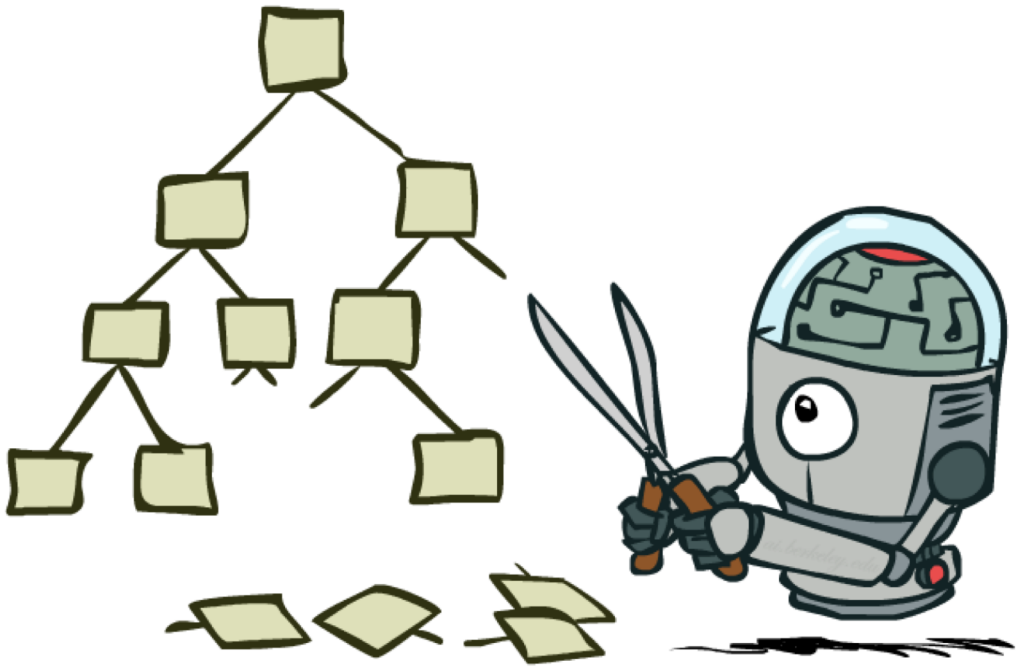
- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: O(bm)

- Example: For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?

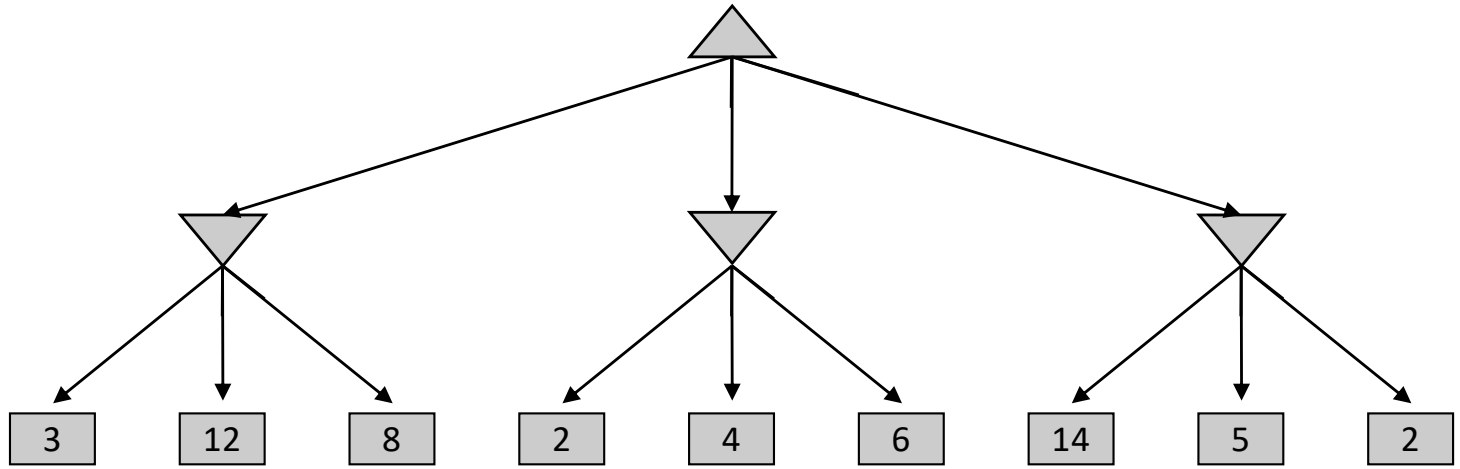


Resource Limits

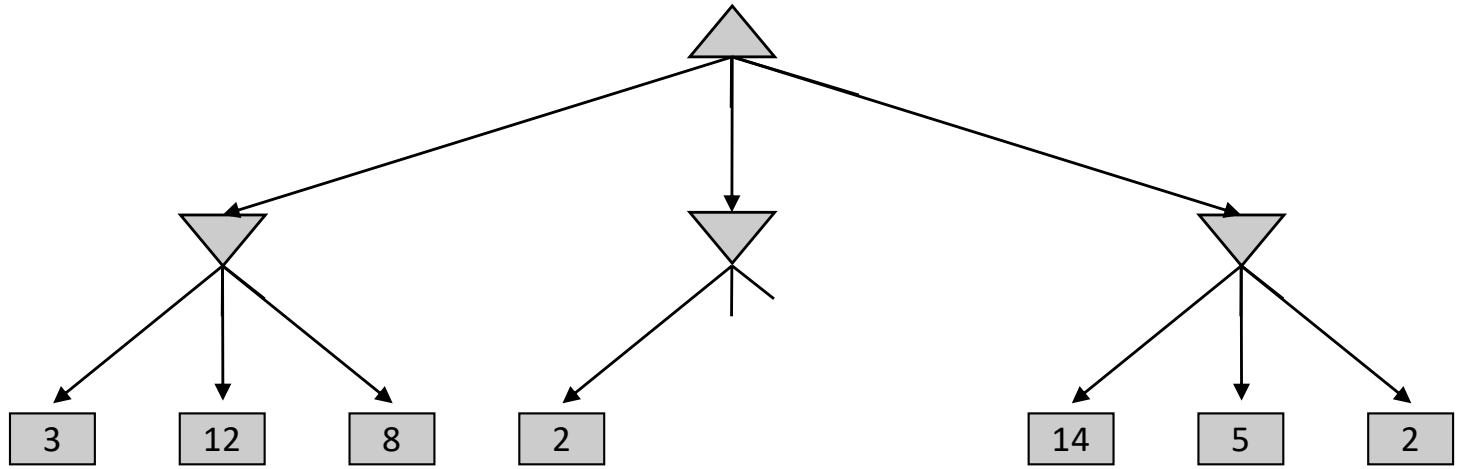
Game Tree Pruning



Minimax Pruning Example

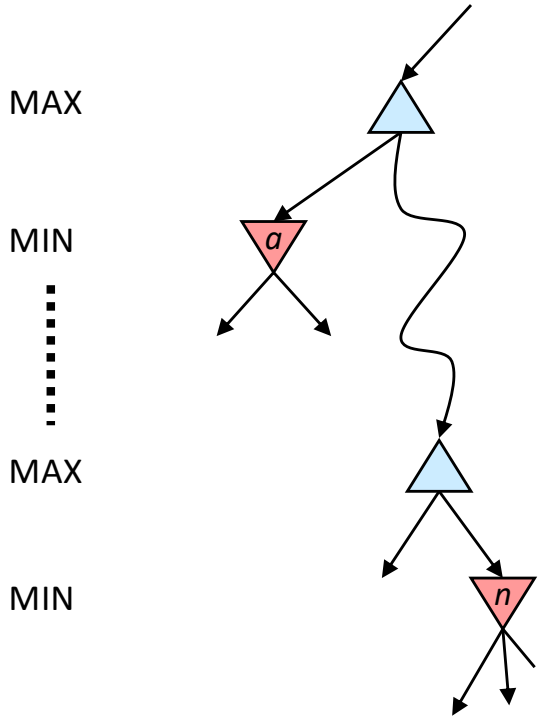


Minimax Pruning



Alpha-Beta Pruning

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



- MAX version is symmetric

The α-β algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
    inputs: state, current state in game

    v ← MAX-VALUE(state, −∞, +∞)
    return the action in SUCCESSORS(state) with value v

---

function MAX-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state

    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

The α-β algorithm

```
function MIN-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(s, α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Alpha-Beta Implementation

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

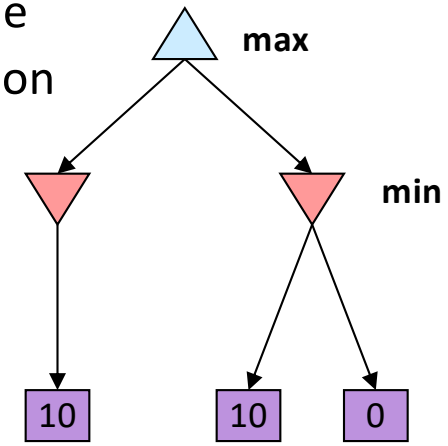
```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
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
```

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

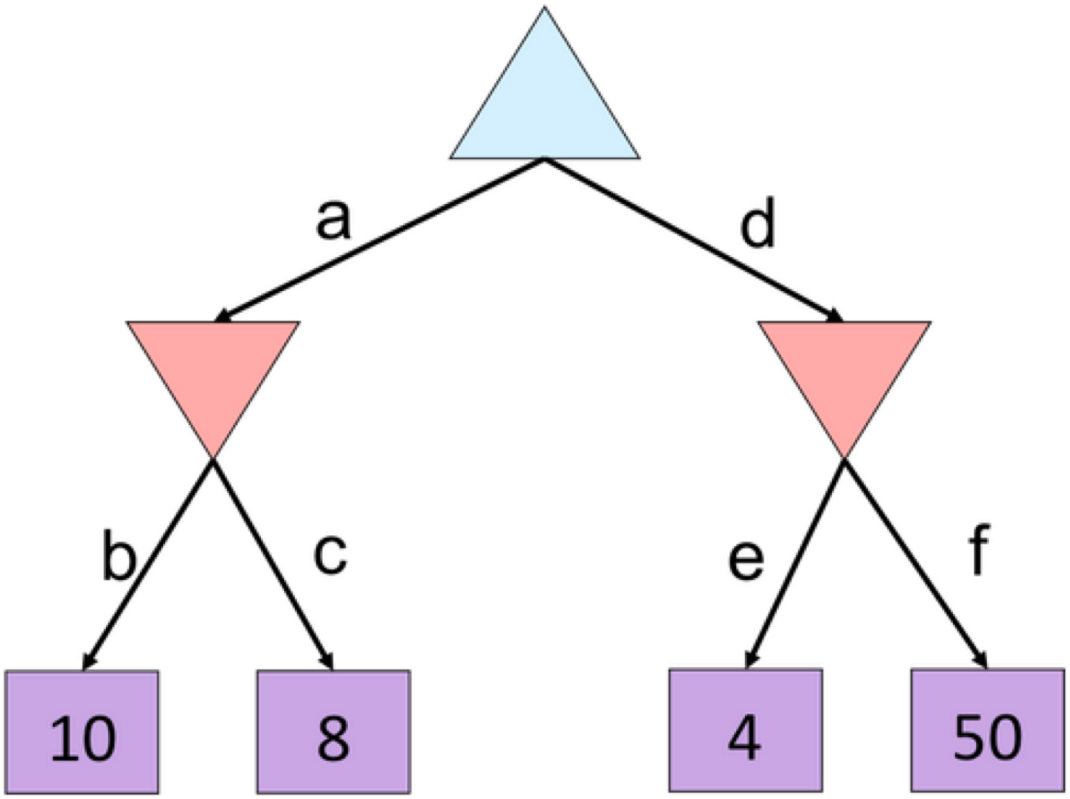


- Good child ordering improves effectiveness of pruning

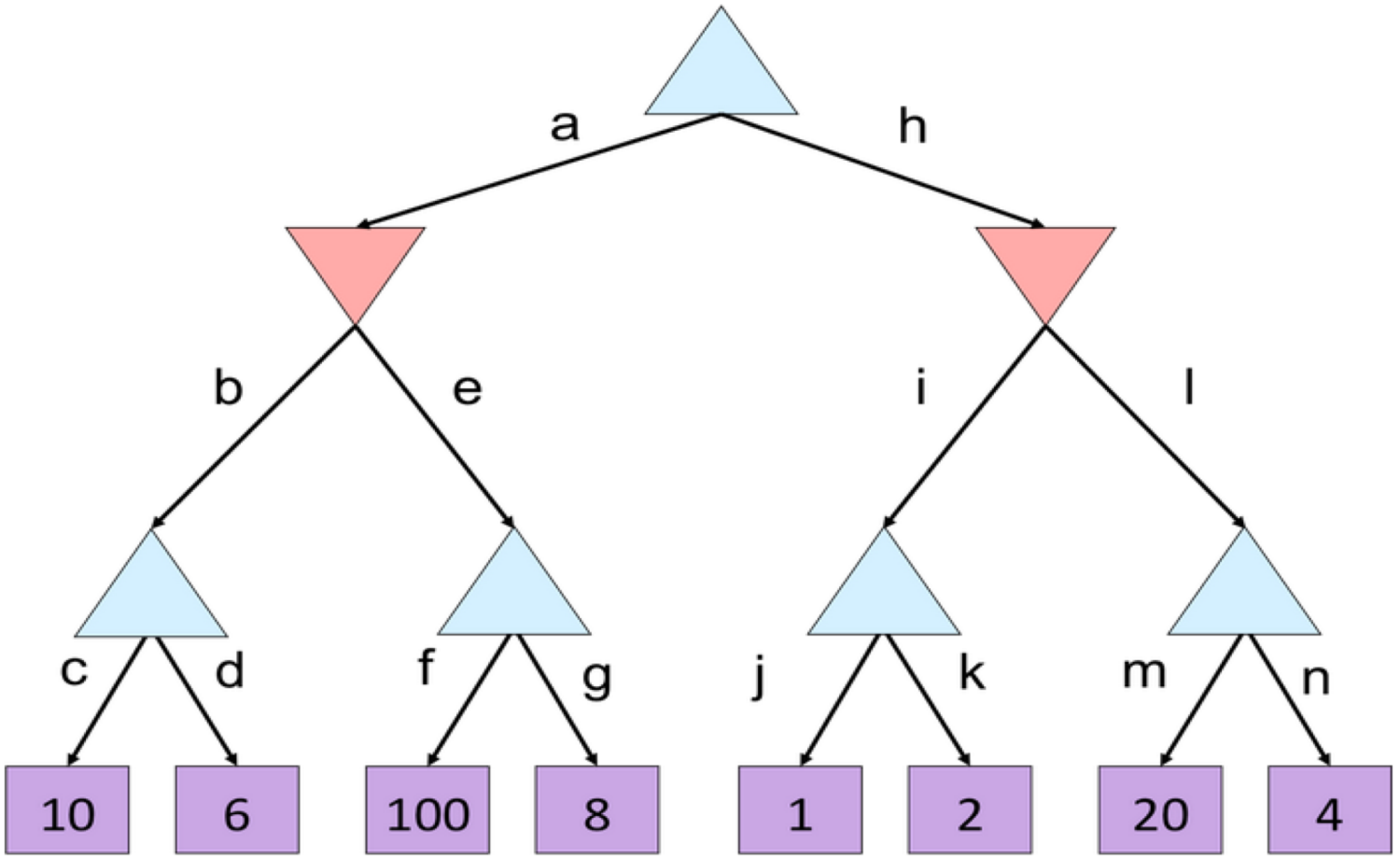
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless…

- This is a simple example of metareasoning (computing about what to compute)

Alpha-Beta Quiz

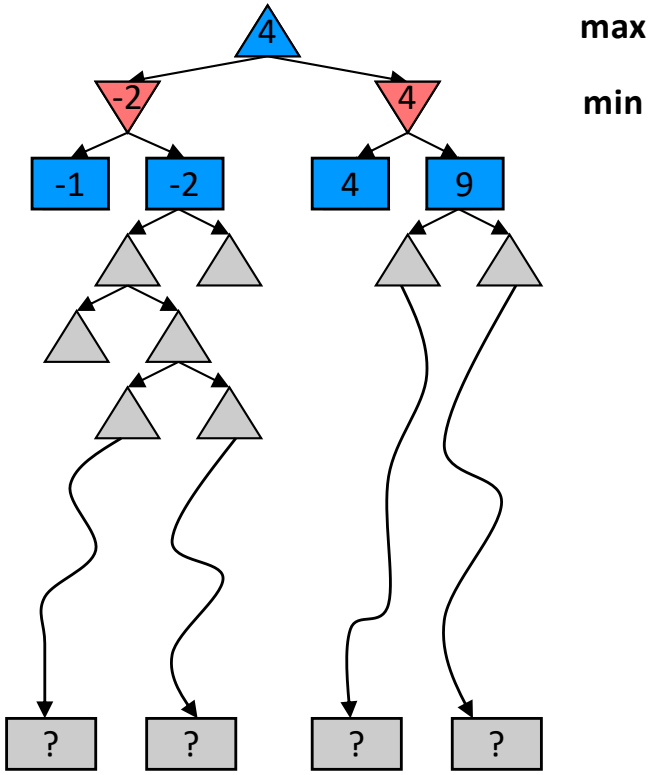


Alpha-Beta Quiz 2

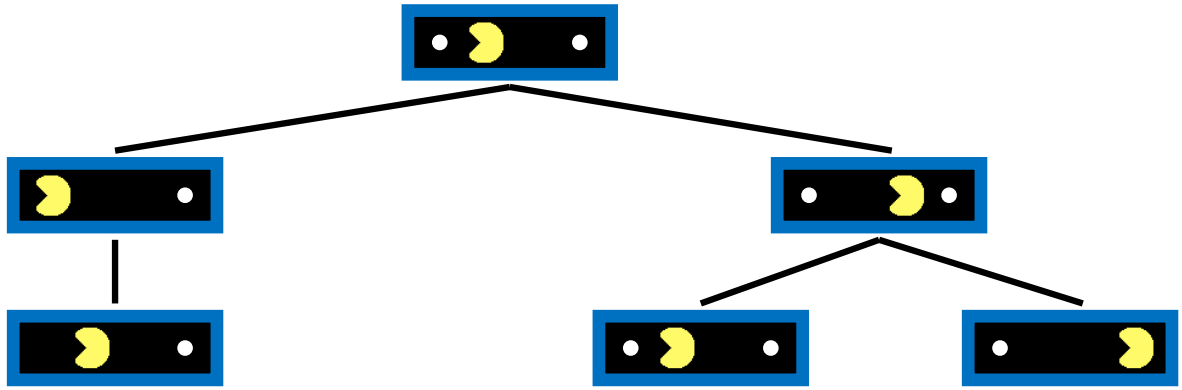


Resource Limits

- Problem: In realistic games, cannot search to leaves!
- Solution: Depth-limited search
  - Instead, search only to a limited depth in the tree
  - Replace terminal utilities with an *evaluation function* for non-terminal positions
- 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
- Guarantee of optimal play is gone
- More plies makes a BIG difference
- Use iterative deepening for an anytime algorithm

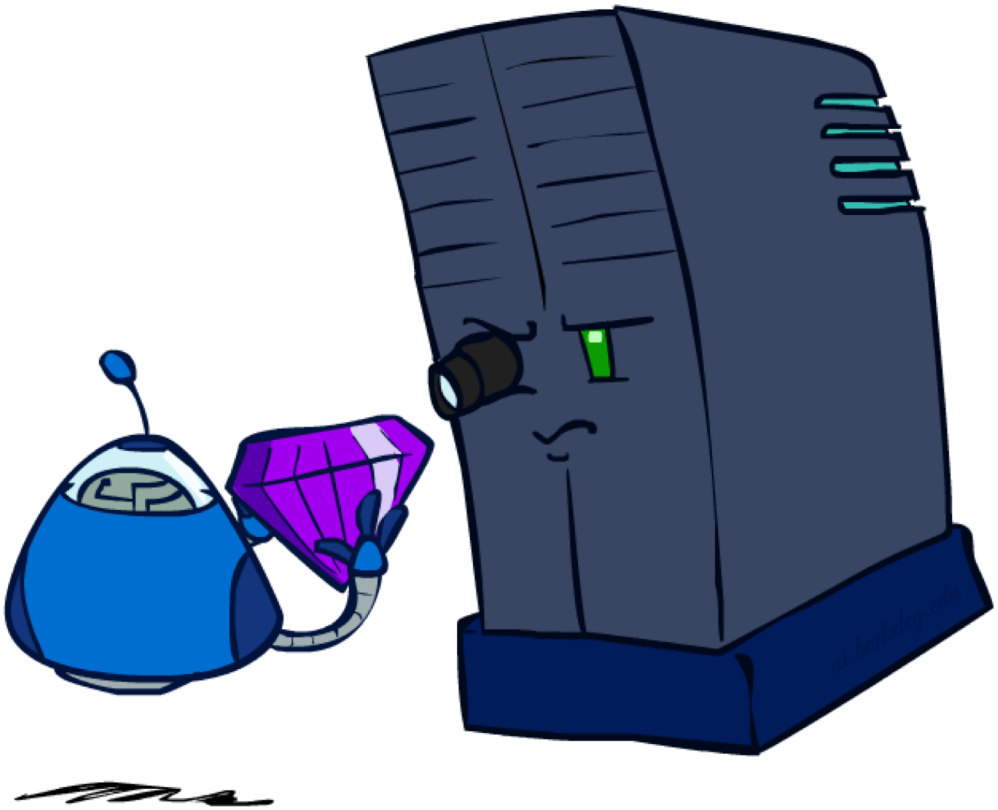


Why Pacman Starves



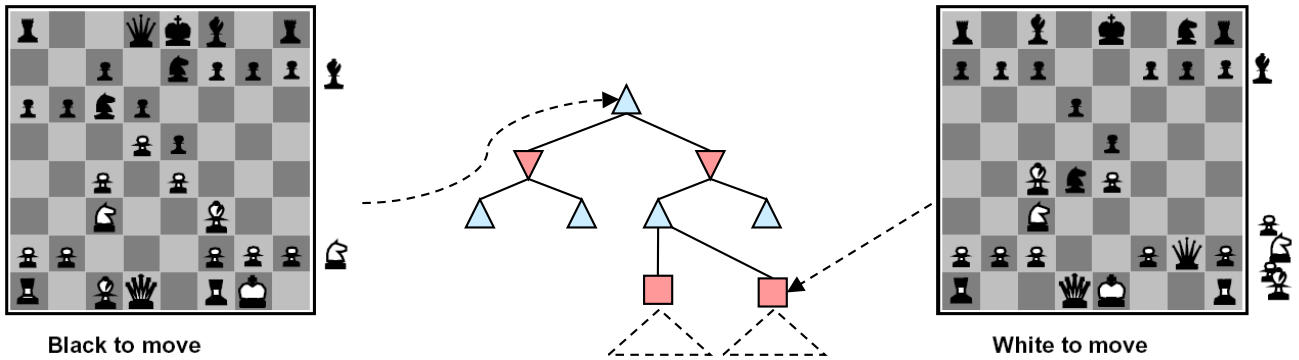
- A danger of replanning agents!
  - He knows his score will go up by eating the dot now (west, east)
  - He knows his score will go up just as much by eating the dot later (east, west)
  - There are no point-scoring opportunities after eating the dot (within the horizon, two here)
  - Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!

Evaluation Functions



Evaluation Functions

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



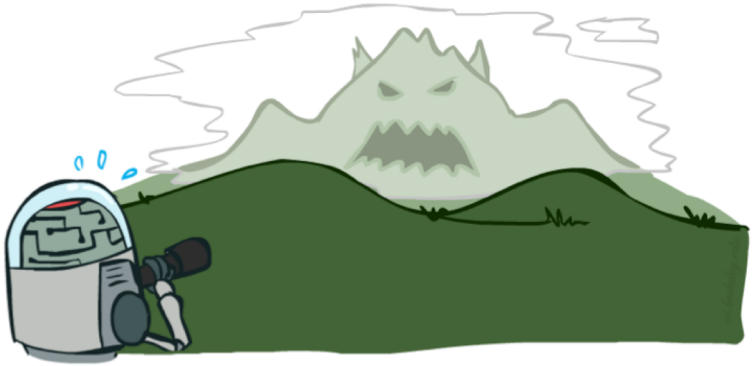
- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

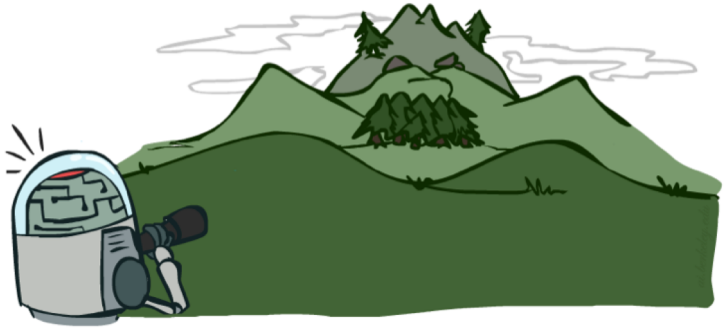
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

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

Depth Matters

- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
- An important example of the tradeoff between complexity of features and complexity of computation





Synergies between Evaluation Function and Alpha-Beta?

- Alpha-Beta: amount of pruning depends on expansion ordering
  - Evaluation function can provide guidance to expand most promising nodes first (which later makes it more likely there is already a good alternative on the path to the root)
    - (somewhat similar to role of A* heuristic, CSPs filtering)

- Alpha-Beta: (similar for roles of min-max swapped)
  - Value at a min-node will only keep going down
  - Once value of min-node lower than better option for max along path to root, can prune
  - Hence: IF evaluation function provides upper-bound on value at min-node, and upper-bound already lower than better option for max along path to root
    THEN can prune

Next Time: Uncertainty!

Summary

- Reviewed games to understand what optimal play means and to understand how to play well in practice.
- Game can be defined by the *initial state* (how the board is set up), legal *actions* in each state, a *terminal test* (game over), and a *utility function* that applies to terminal states.
- For two-player, zero-sum games with perfect information, the *minimax* algorithm can select optimal moves using a depth-first enumeration of the game tree.
- *Alpha-beta* search computes the same optimal move as minimax, but achieves much greater efficiency by eliminating subtrees that are provably irrelevant.
- Usually not feasible to consider the whole game tree, need to *cut the search* off at some point and apply *eval* function that gives a *guesstimate* of the utility of a state.

Summary

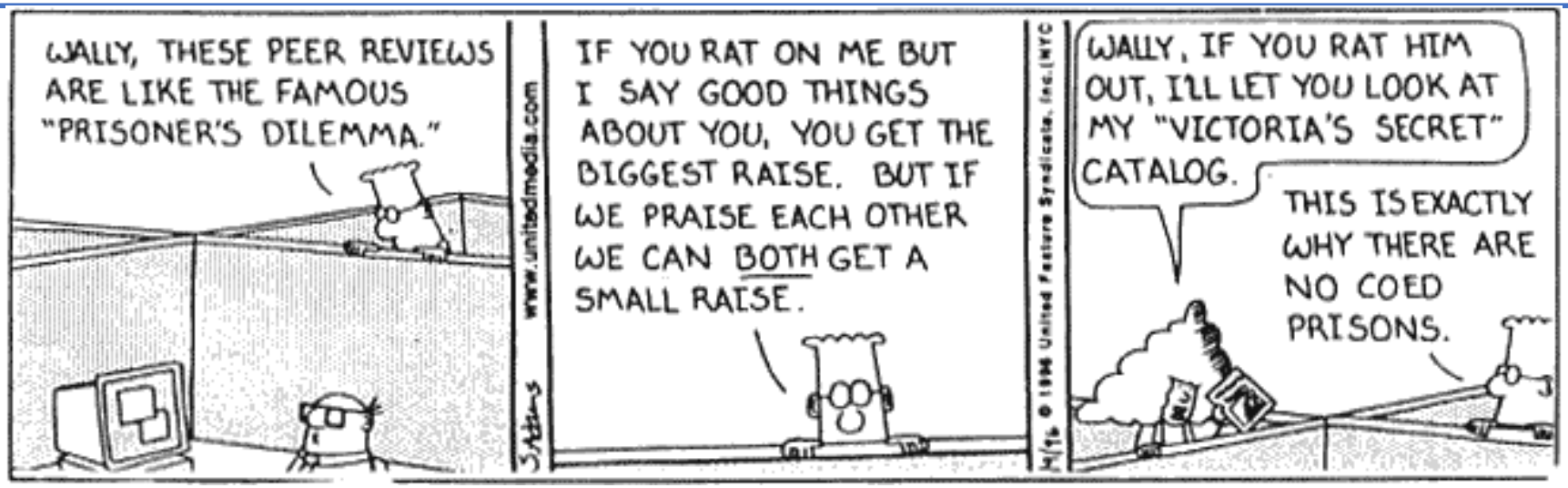
- Games of chance can be handled by an extension to minimax algorithm that values a *chance* node by taking the *average utility* of all its children nodes *weighted by the probability* of each child.

- Optimal play in games of imperfect information (bridge) requires reasoning about the current and future belief states of each player. Can average value of an action over each possible action.

- Programs can match or beat the best human players in many games.

- Games are fun to work on!
  - Illustrate several important points about AI
  - perfection is unattainable → must approximate
  - good idea to think about what to think about

Game Theory


WALLY, THESE PEER REVIEWS ARE LIKE THE FAMOUS "PRISONER'S DILEMMA."
IF YOU RAT ON ME BUT I SAY GOOD THINGS ABOUT YOU, YOU GET THE BIGGEST RAISE. BUT IF WE PRAISE EACH OTHER WE CAN BOTH GET A SMALL RAISE.
WALLY, IF YOU RAT HIM OUT, I'LL LET YOU LOOK AT MY "VICTORIA'S SECRET" CATALOG.
THIS IS EXACTLY WHY THERE ARE NO COED PRISONS.
www.unitedmedia.com
© 1996 United Feature Syndicate, Inc. (NYC