

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

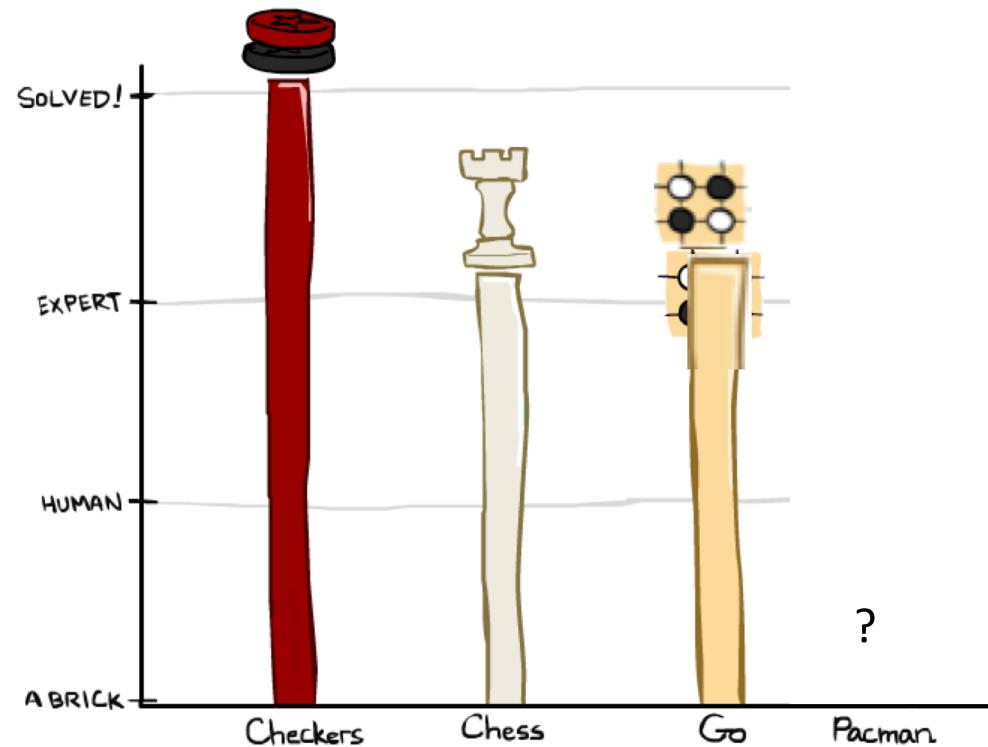


Outline

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

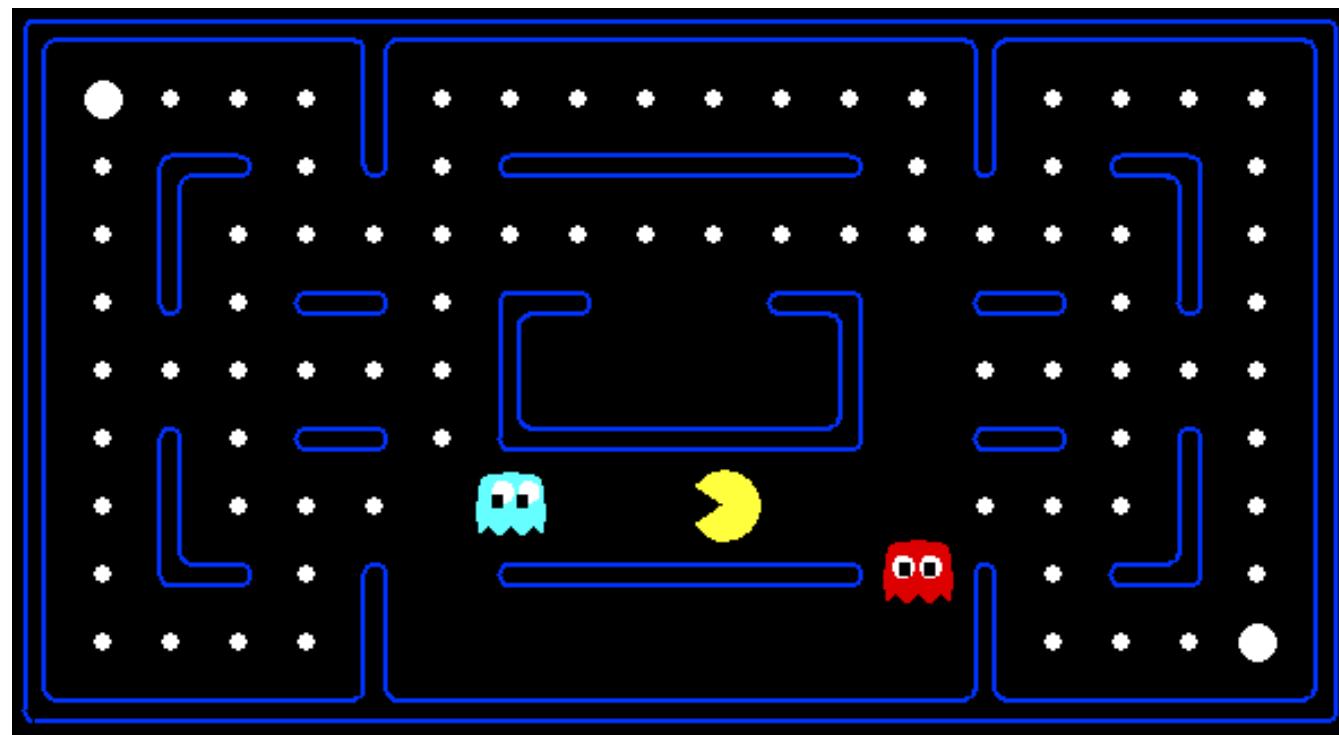
Game Playing State-of-the-Art

- **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)

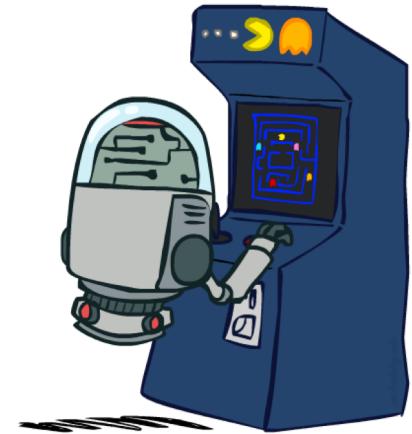
- **Multiagent environments**
 - Any given agent needs to consider the actions of other agents and how they affect their welfare.
- Unpredictability of other agents can introduce many possible contingencies into the agent's problem-solving process.
- *Cooperative* and *competitive* multi-agent environments?
- 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 as 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: $P=\{1\dots N\}$ (usually take turns)
 - Actions: A (may depend on player / state)
 - Transition Function: $S \times A \rightarrow S$
 - Terminal Test: $S \rightarrow \{t, f\}$
 - Terminal Utilities: $S \times P \rightarrow R$
- Solution for a player is a **policy**: $S \rightarrow A$



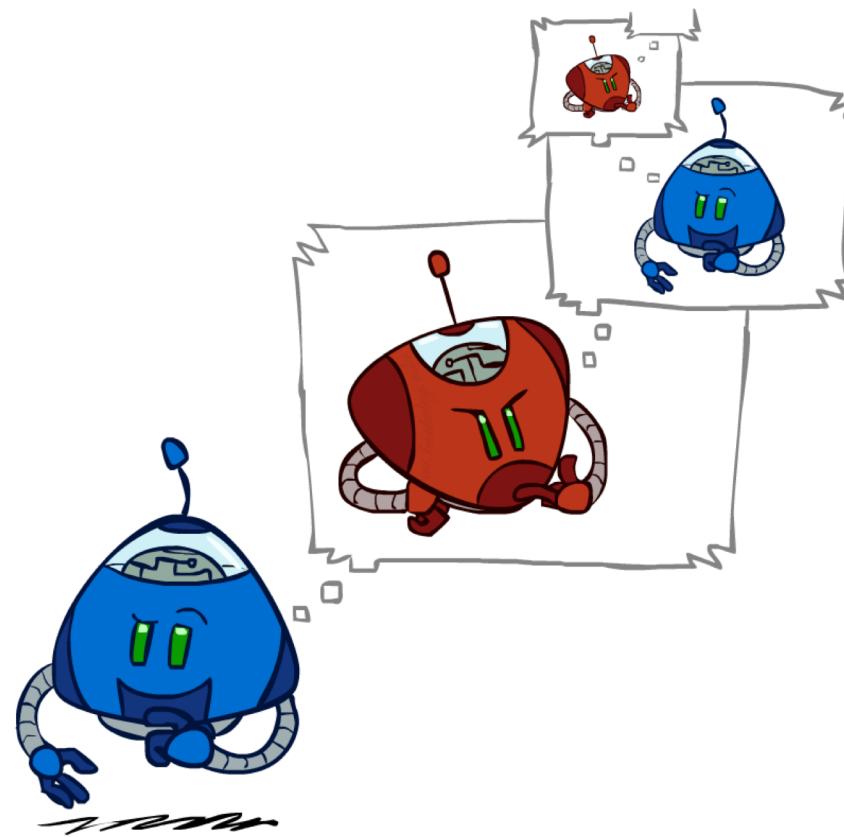
Zero-Sum Games



- 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

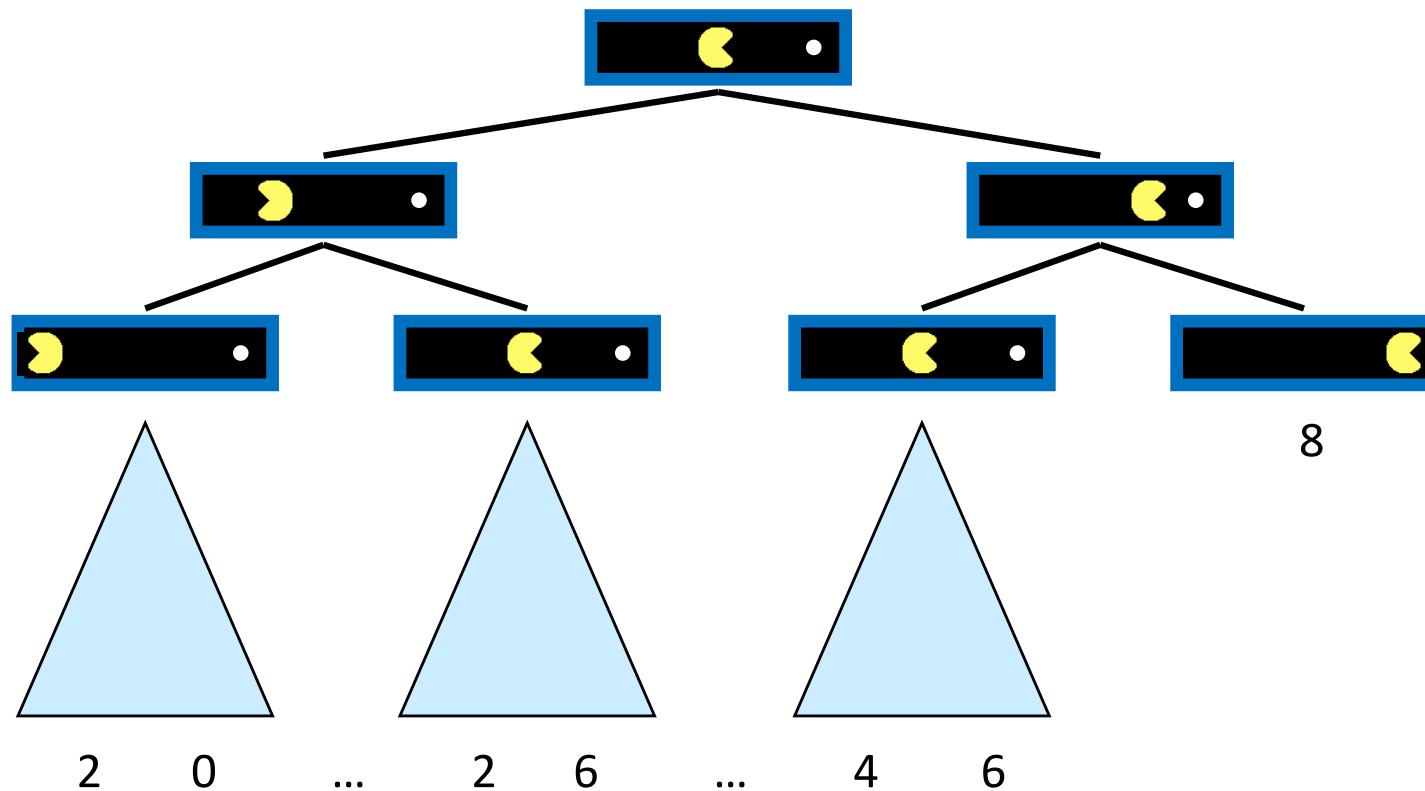
- General Games
 - Agents have independent utilities (values on outcomes)
 - Cooperation, indifference, competition, and more are all possible

Adversarial Search



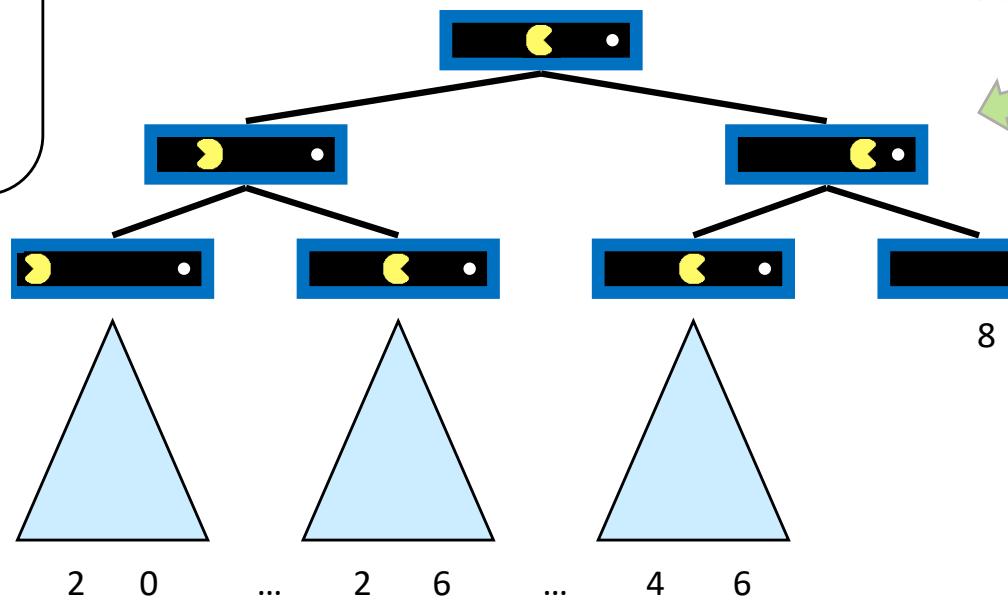
Single-Agent Trees

(Does not consider adversaries)



Value of a State

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



Non-Terminal

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



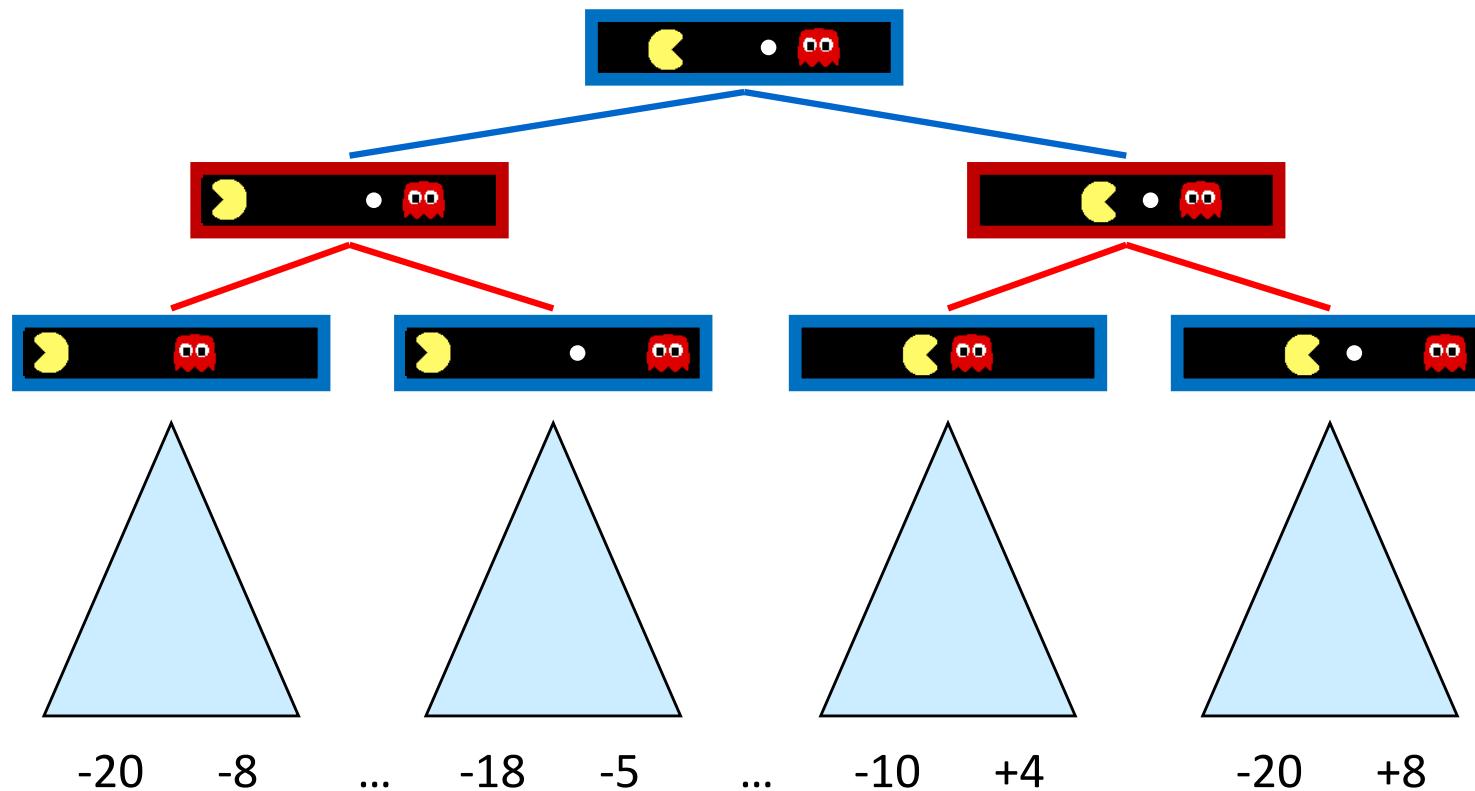
8

Terminal States:

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



Adversarial Game Trees (considers adversaries)



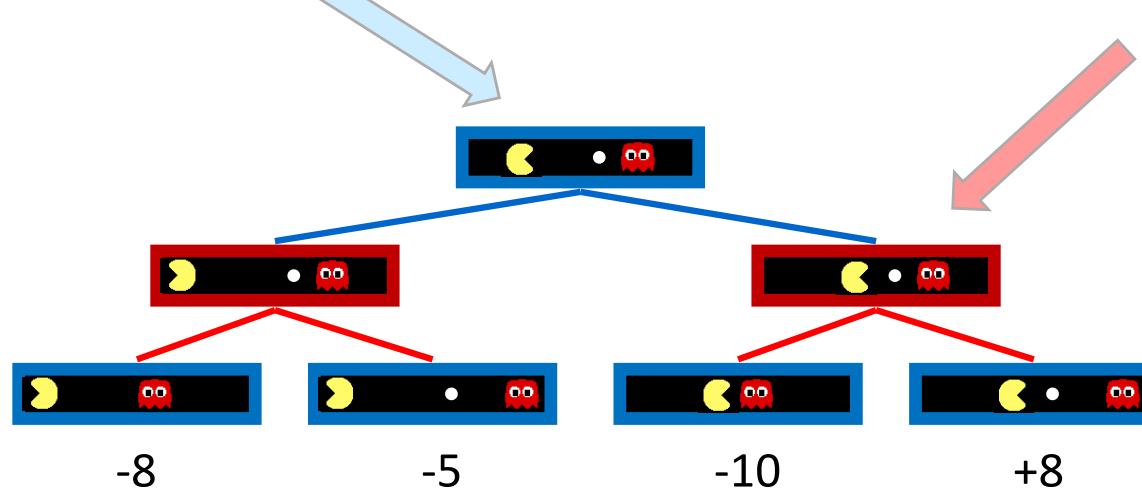
Minimax Values

States Under Agent's Control:

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

States Under Opponent's Control:

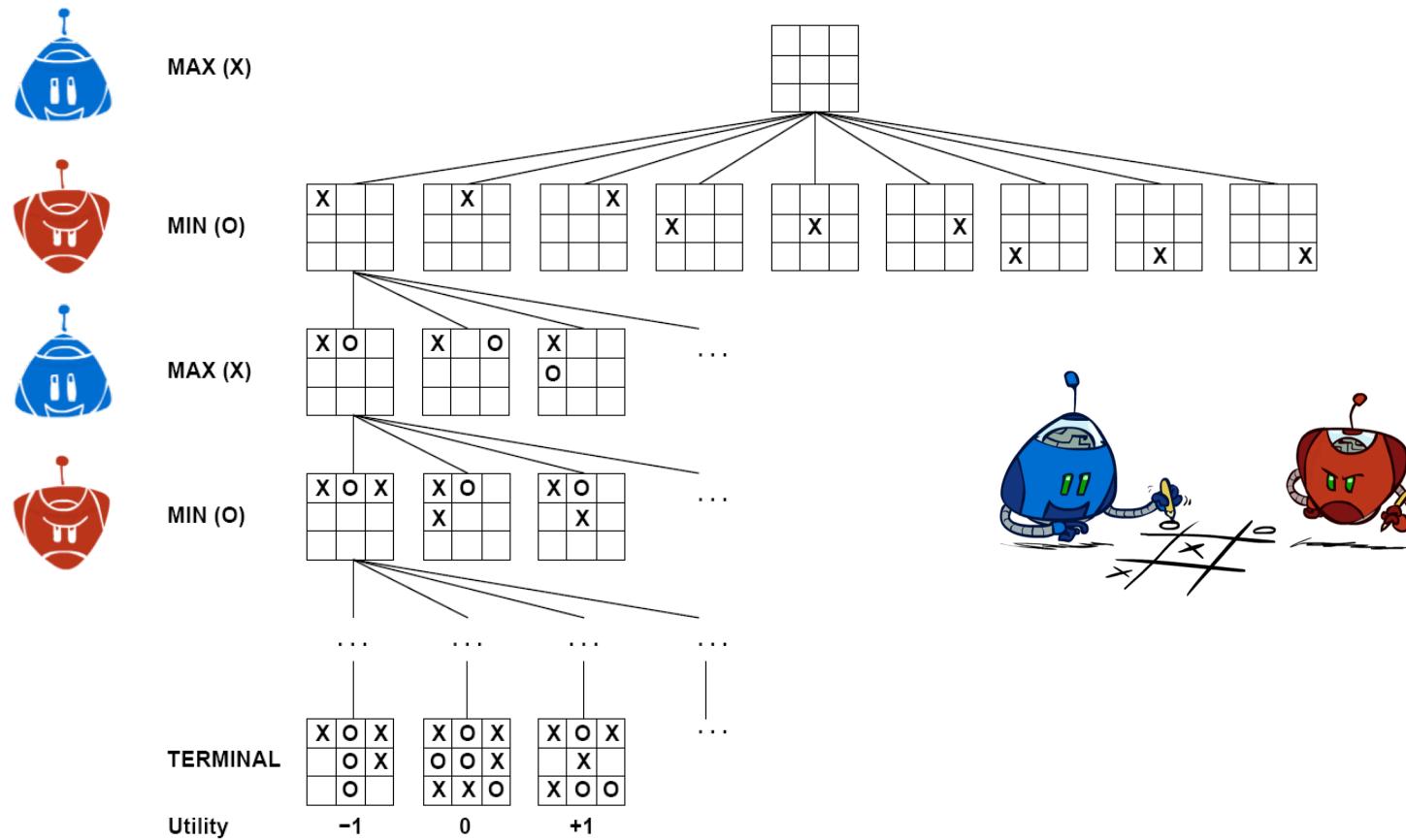
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

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

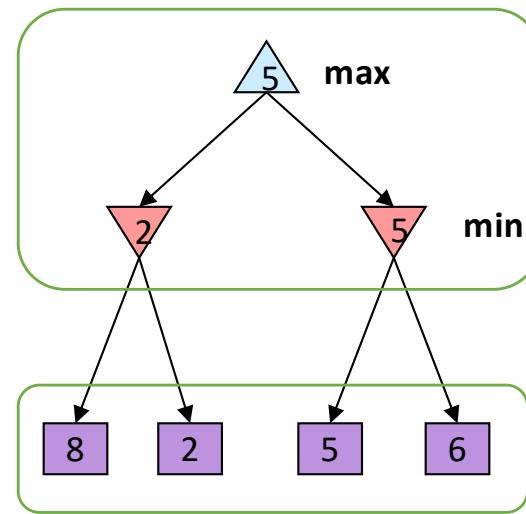
Tic-Tac-Toe Game Tree



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 values:
computed recursively



Terminal values:
part of the game

Minimax algorithm

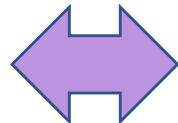
```
function MINIMAX-DECISION(state) returns an action
    v  $\leftarrow$  MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v
```

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v  $\leftarrow -\infty$ 
    for a, s in SUCCESSORS(state) do
        v  $\leftarrow \text{MAX}(\iota, \text{MIN-VALUE}(s))$ 
    return v
```

```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v  $\leftarrow \infty$ 
    for a, s in SUCCESSORS(state) do
        v  $\leftarrow \text{MIN}(\iota, \text{MAX-VALUE}(s))$ 
    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)
```

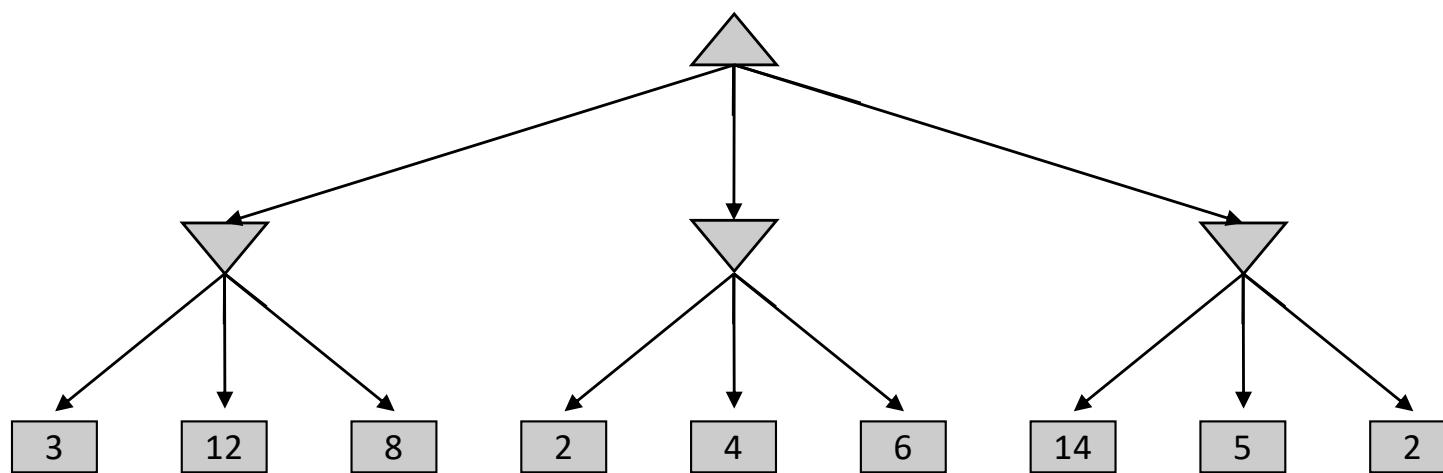
```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

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

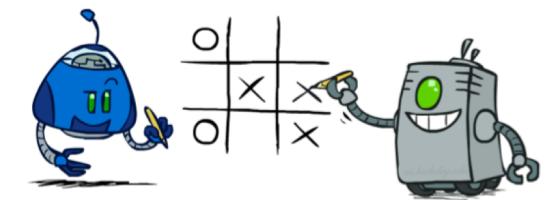
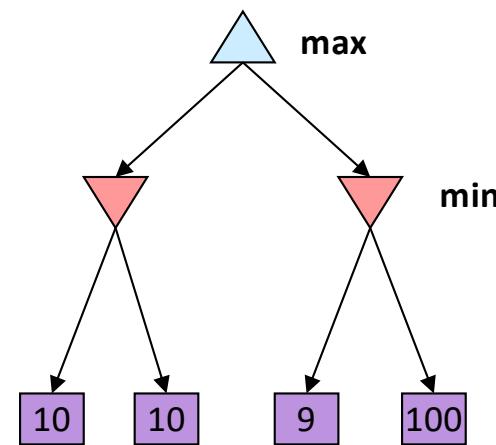
$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

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

Minimax Example



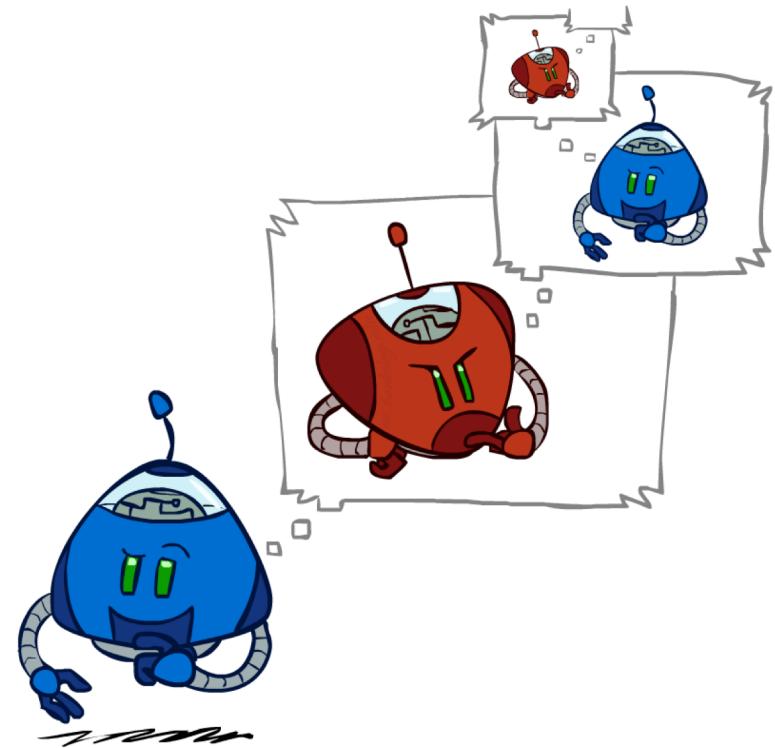
Minimax Properties



Optimal against a perfect player. Otherwise?

Minimax Efficiency

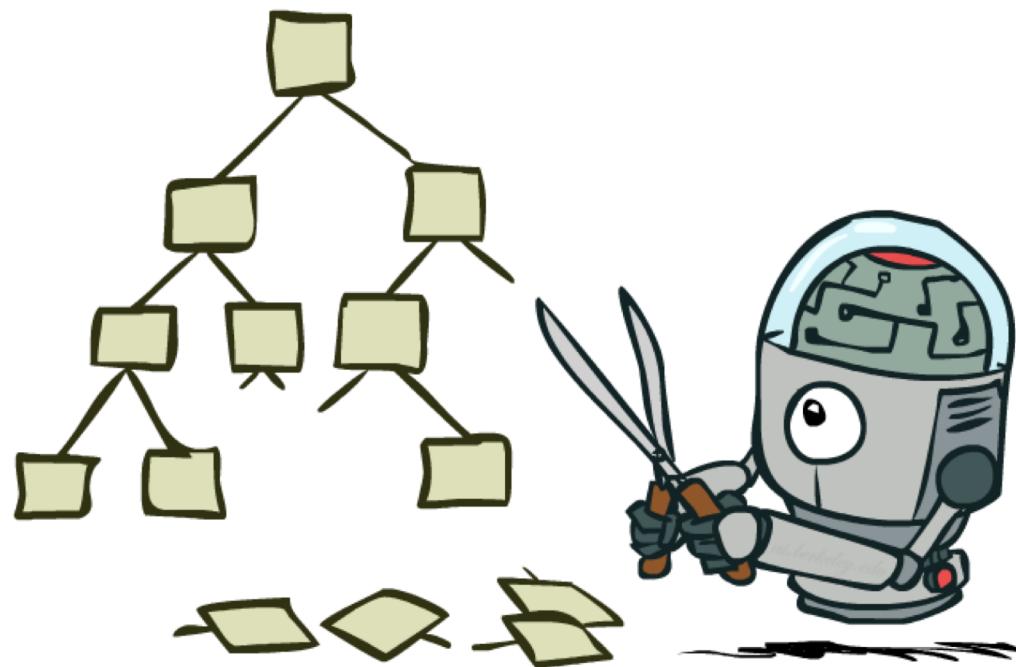
- How efficient is minimax?
 - Just like (exhaustive) DFS
 - Time: $O(b^m)$
 - Space: $O(bm)$
- Example: For chess, $b \approx 35$, $m \approx 100$
 - Exact solution is completely infeasible
 - But, do we need to explore the whole tree?



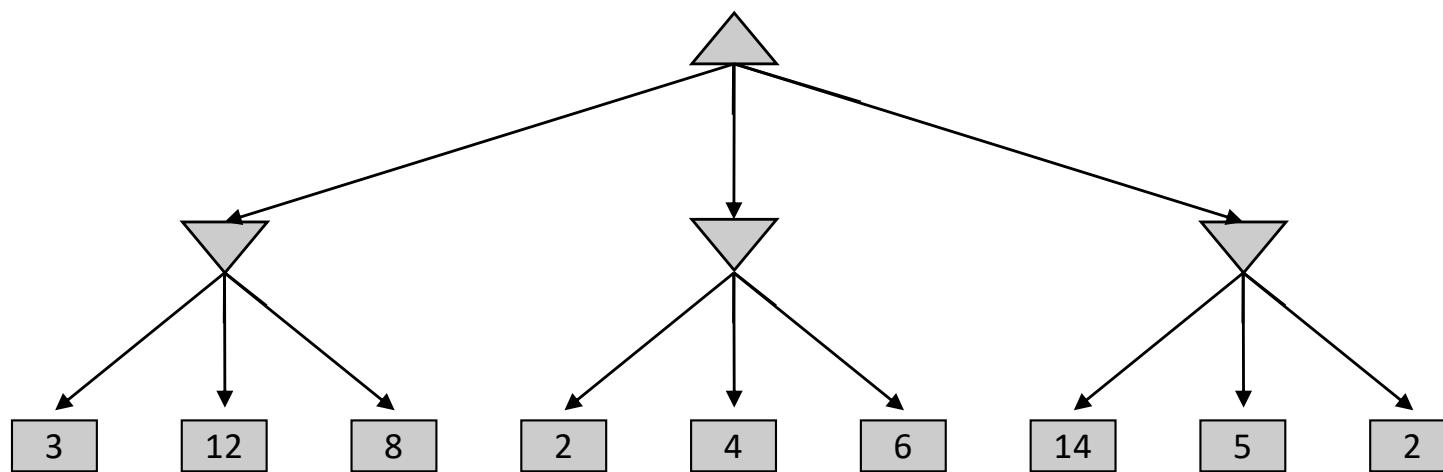
Resource Limits



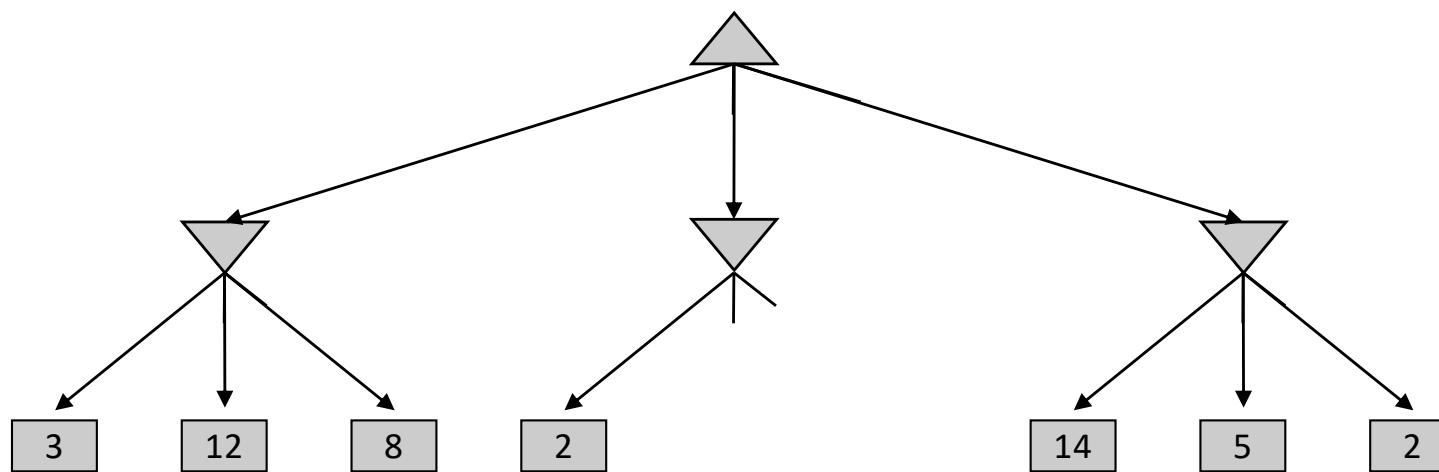
Game Tree Pruning



Minimax Pruning Example

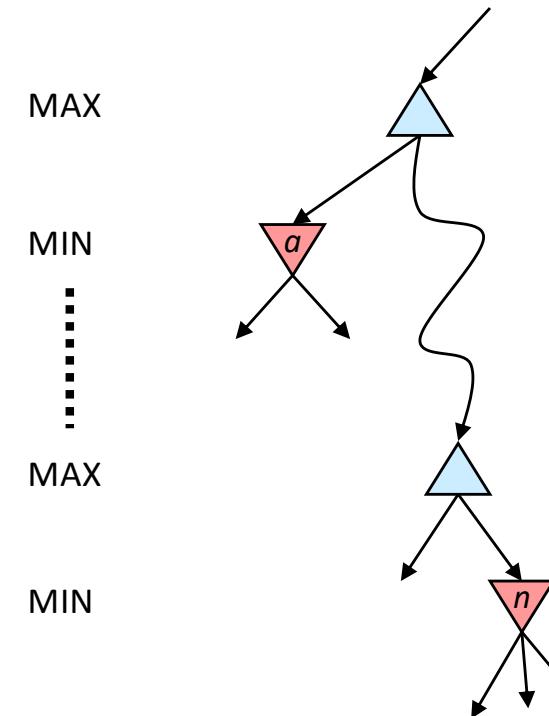


Minimax Pruning



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 \leftarrow MAX-VALUE(*state*, $-\infty$, $+\infty$)

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 $\leftarrow -\infty$

for *a, s* in SUCCESSORS(*state*) **do**

v $\leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$

if *v* $\geq \beta$ **then return** *v*

$\alpha \leftarrow \text{MAX}(\alpha, v)$

return *v*

The α - β algorithm

```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
    inputs: state, current state in game
             $\alpha$ , the value of the best alternative for MAX along the path to state
             $\beta$ , the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
     $v \leftarrow +\infty$ 
    for a, s in SUCCESSORS(state) do
         $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta))$ 
        if  $v \leq \alpha$  then return v
         $\beta \leftarrow \text{MIN}(\beta, v)$ 
    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,  $\alpha$ ,  $\beta$ ):  
    initialize v = - $\infty$   
    for each successor of state:  
        v = max(v, value(successor,  $\alpha$ ,  $\beta$ ))  
        if v  $\geq \beta$  return v  
         $\alpha$  = max( $\alpha$ , v)  
    return v
```

```
def min-value(state ,  $\alpha$ ,  $\beta$ ):  
    initialize v = + $\infty$   
    for each successor of state:  
        v = min(v, value(successor,  $\alpha$ ,  $\beta$ ))  
        if v  $\leq \alpha$  return v  
         $\beta$  = min( $\beta$ , 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

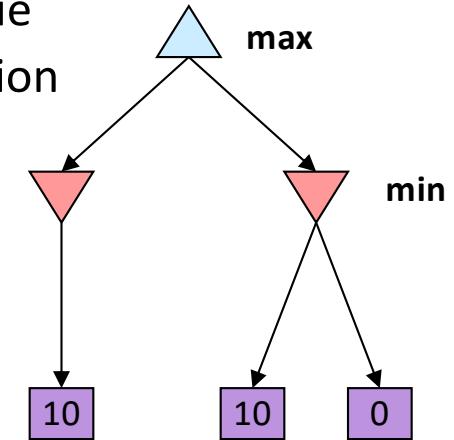
- Important: children of the root may have the wrong value
 - So the most naïve version won't let you do action selection

- Good child ordering improves effectiveness of pruning

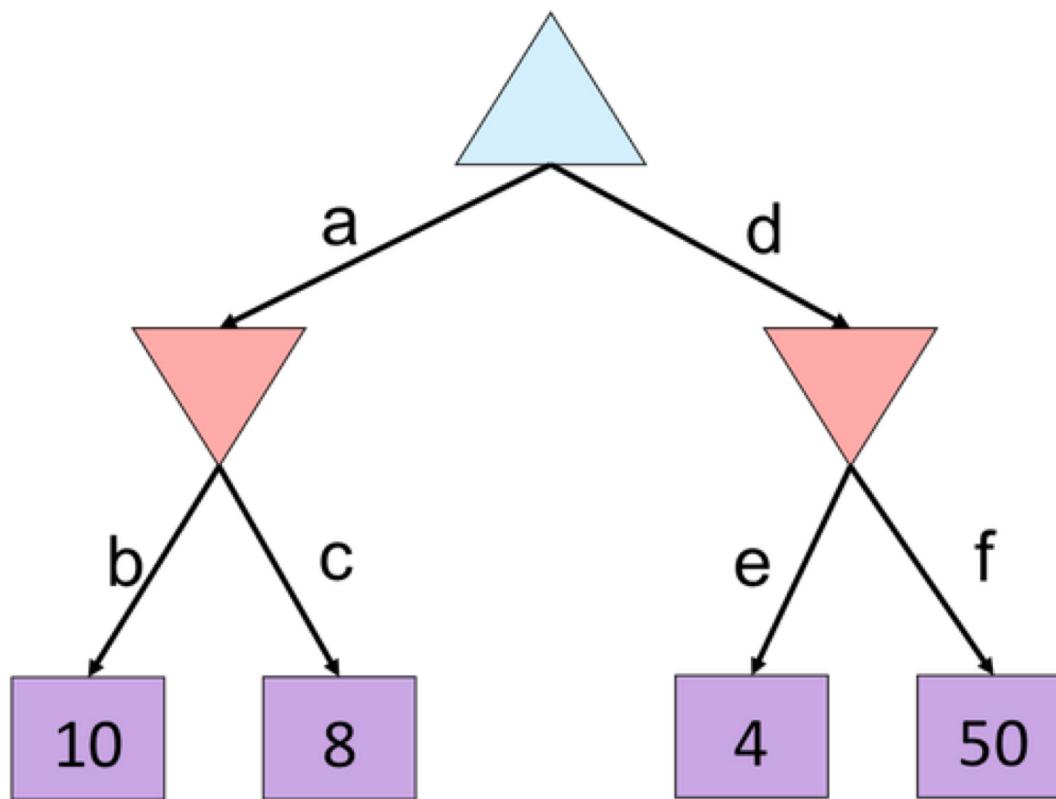
- With “perfect ordering”:

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

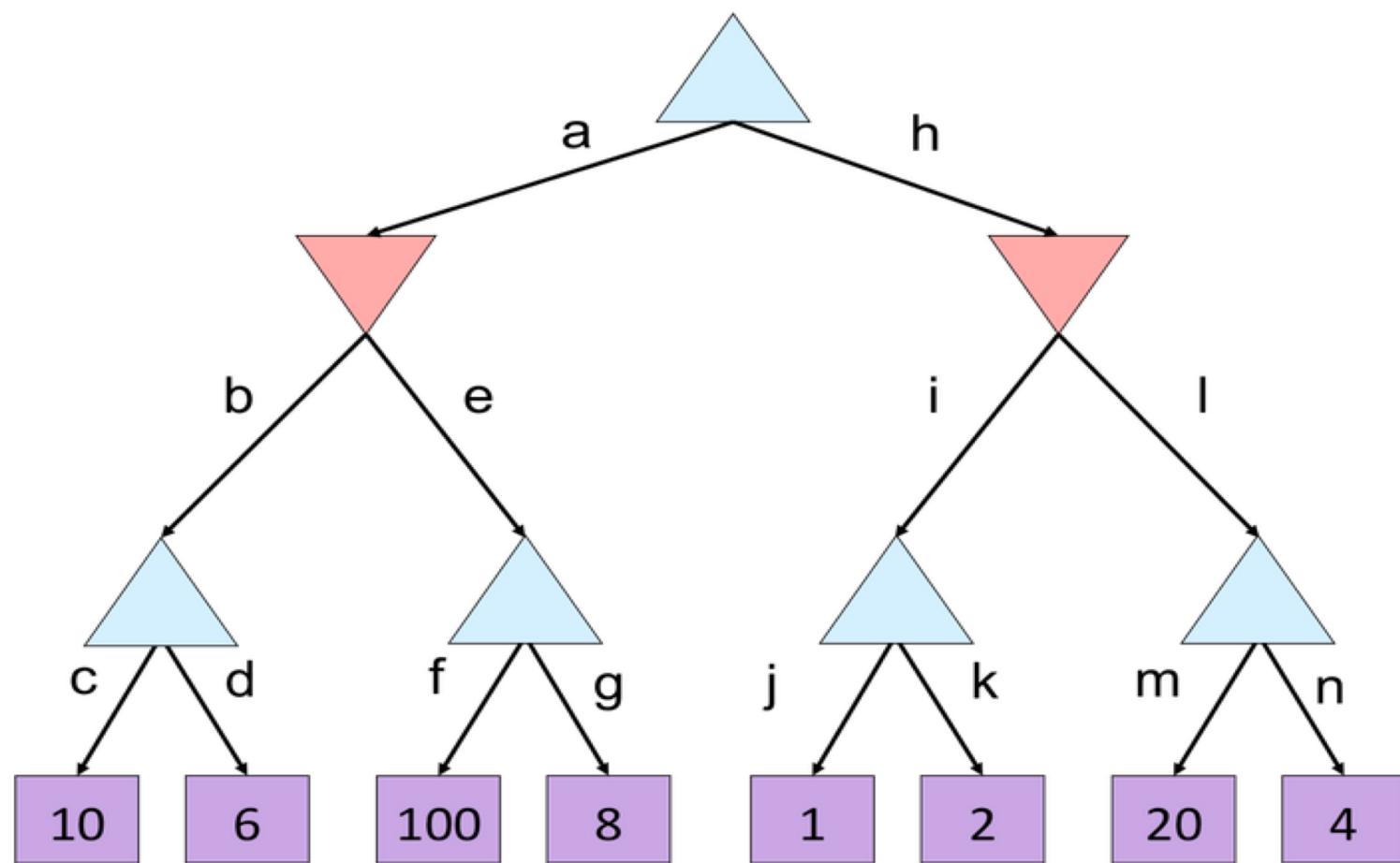
- This is a simple example of **metareasoning** (computing about what to compute)



Alpha-Beta Quiz

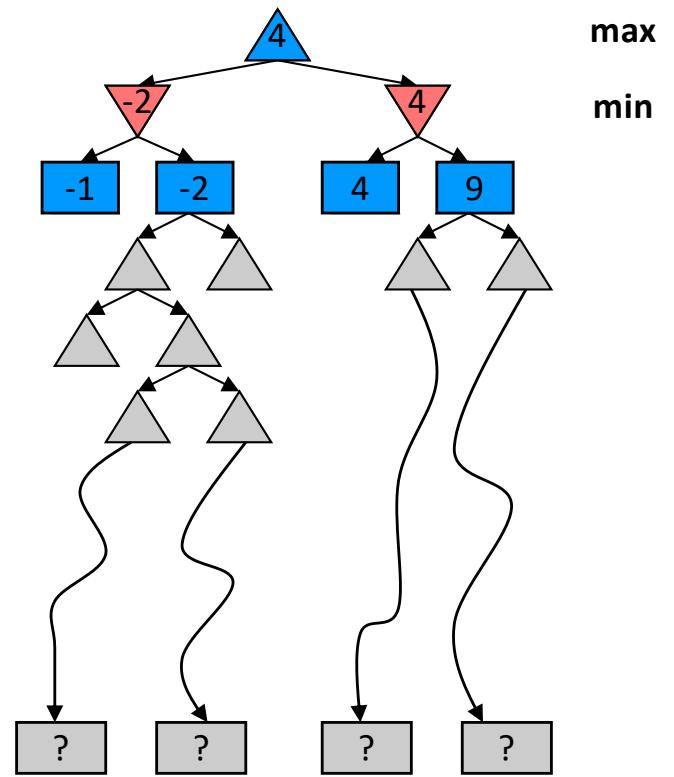


Alpha-Beta Quiz 2

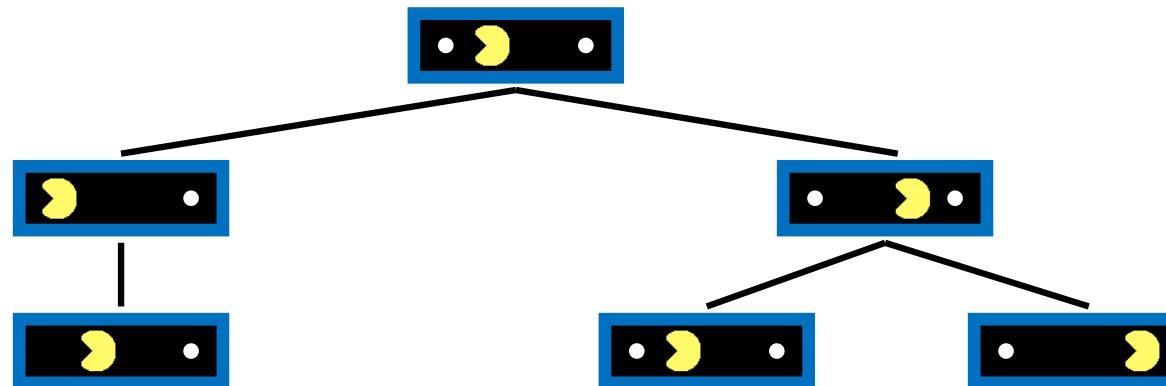


Resource Limits

- 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

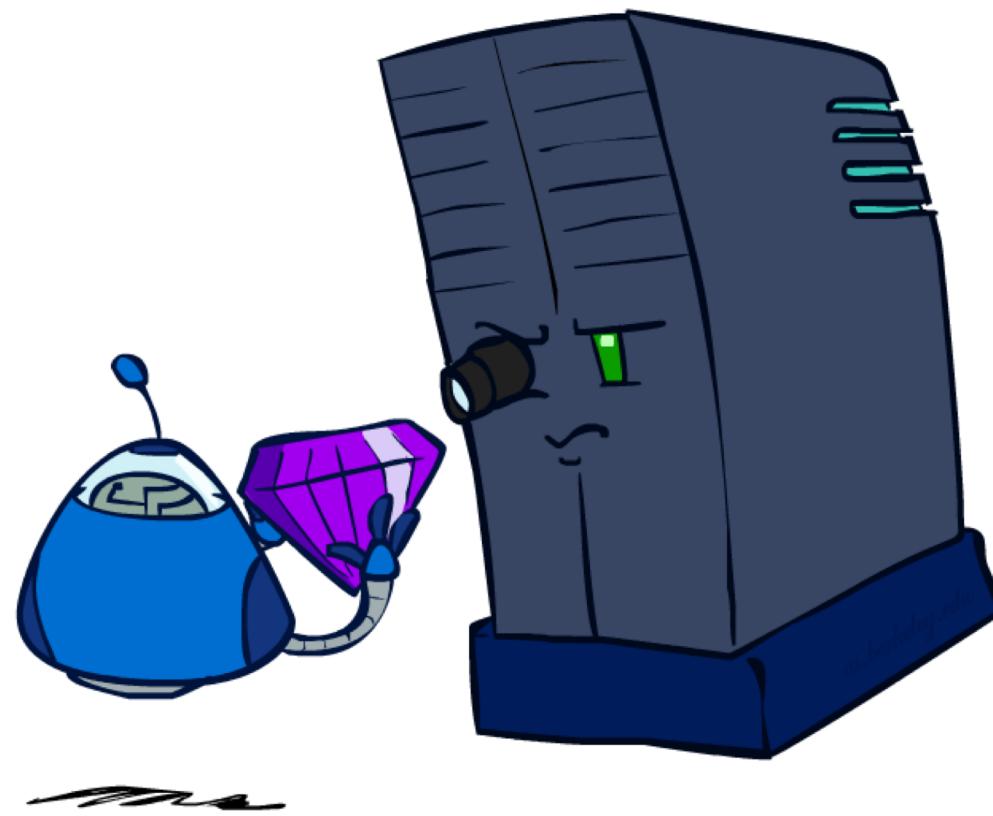


Why Pacman Starves



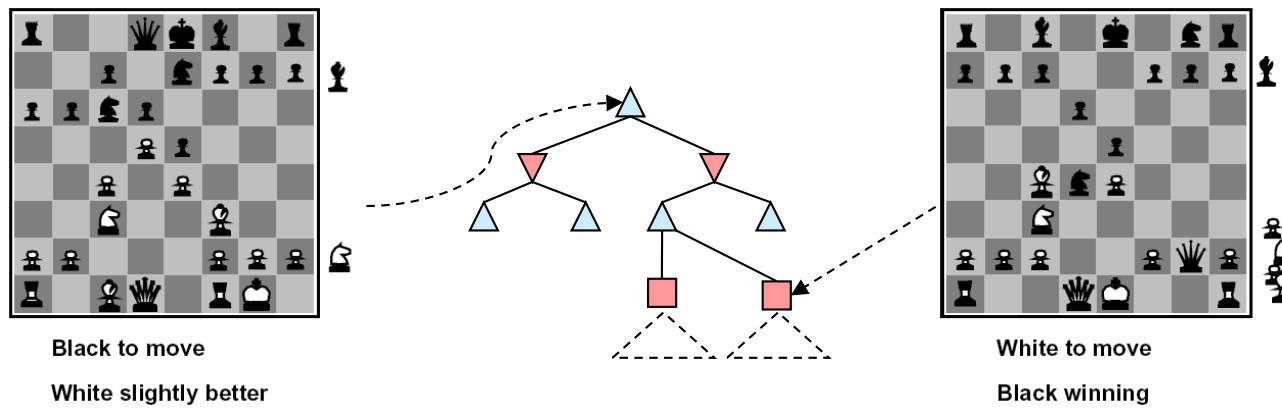
- A danger of replanning agents!
 - He knows his score will go up by eating the dot now (west, east)
 - He knows his score will go up just as much by eating the dot later (east, west)
 - There are no point-scoring opportunities after eating the dot (within the horizon, two here)
 - Therefore, waiting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



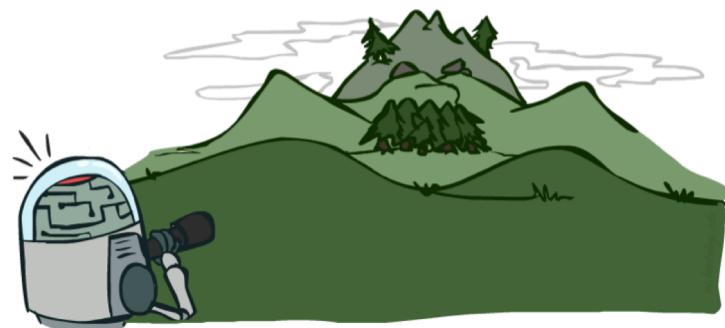
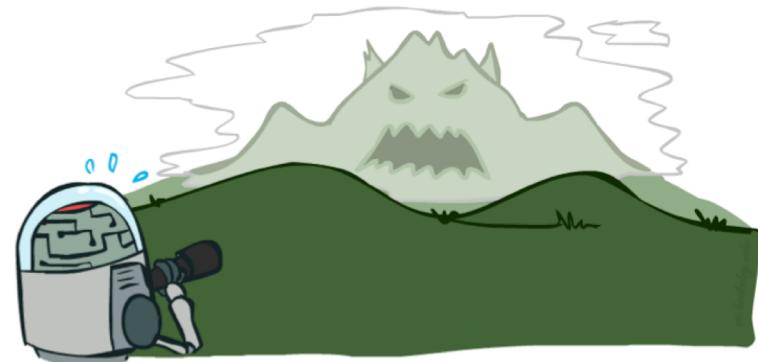
- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

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

- e.g. $f_1(s) = (\text{num white queens} - \text{num black queens})$, 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!

Optimal Decisions in Games

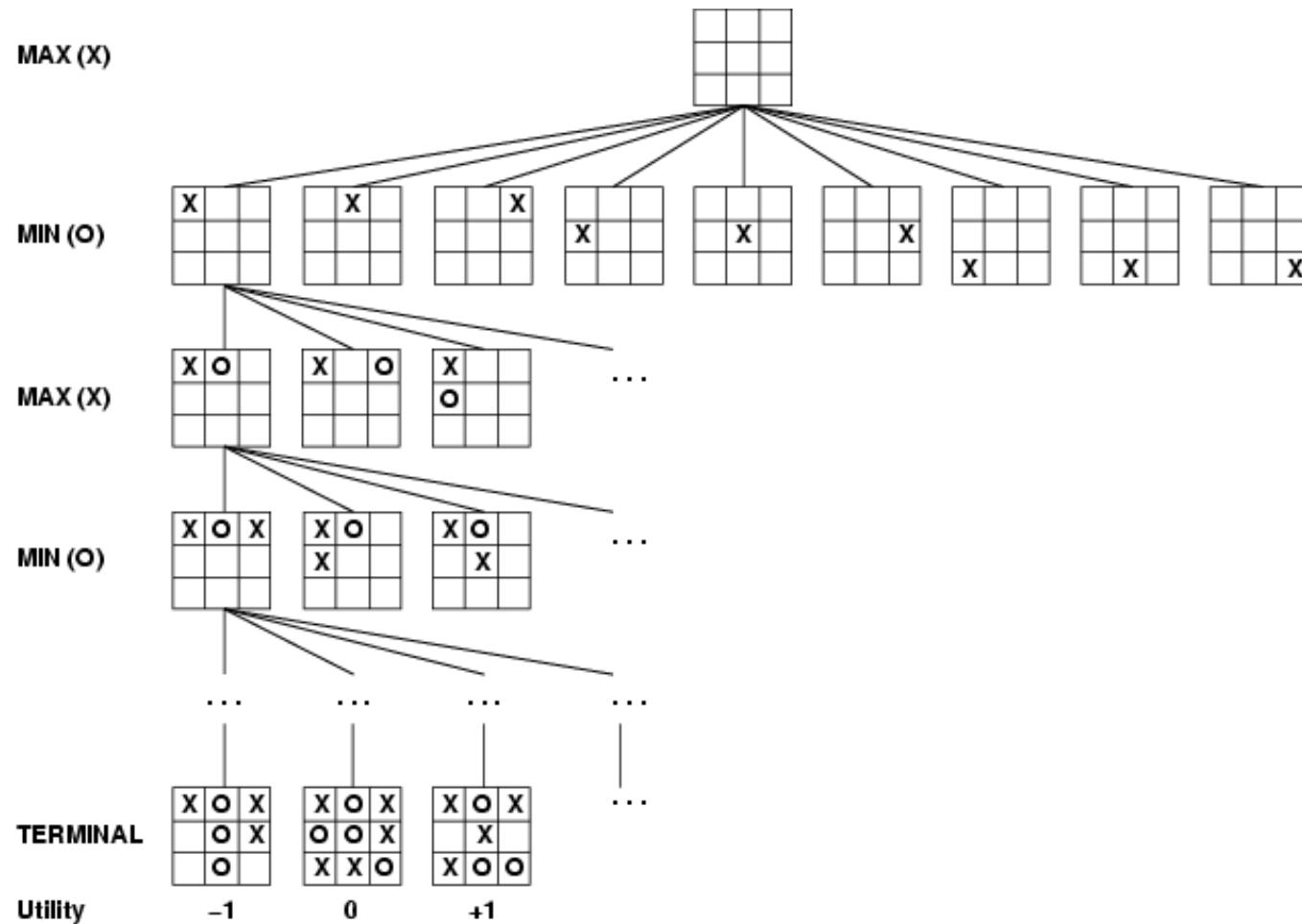
Game search problem:

- Initial state –
 - board position, player to move.
- Successor function –
 - returns list of {move, state} pairs, each indicating a legal move and the resulting state.
- Terminal test –
 - determines when the game is over, i.e., reach *terminal state*.
- Utility function (objective function) –
 - numeric value for terminal states {+1, 0, -1}

Optimal Decisions in Games

Initial state + the *legal moves* for each side define the *game tree*

Game tree (2-player, deterministic, turns)



Optimal Strategies

- Optimal solution is a sequence of moves leading to a goal state.
- Unfortunately, *Max* and *Min*, have equal and opposite objectives.
- Max must find a *contingent strategy*, which specifies Max's move in the initial state.
- Max moves in *successive states* resulting from every possible response by Min.
- *What approach can we use?*

Minimax - Game Theory

- In the theory of *simultaneous games*, a **minimax** strategy is a *mixed strategy* which is part of the solution to a zero-sum game.
- In zero-sum games, the minimax solution is the same as the **Nash equilibrium**:
 - A *solution concept* of a game involving two or more players.
 - Each player is assumed to know the equilibrium strategies of the other players.
 - No player has anything to gain by changing only his own strategy unilaterally.

Minimax Theorem

- For every two-person, zero-sum game with finitely many strategies, there exists a value V and a mixed strategy for each player, such that:
 - Given player 2's strategy, the best payoff possible for player 1 is V , and
 - Given player 1's strategy, the best payoff possible for player 2 is $-V$.
- Equivalently, Player 1's strategy guarantees him a payoff of V regardless of Player 2's strategy, and similarly Player 2 can guarantee himself a payoff of $-V$.

Minimax Theorem

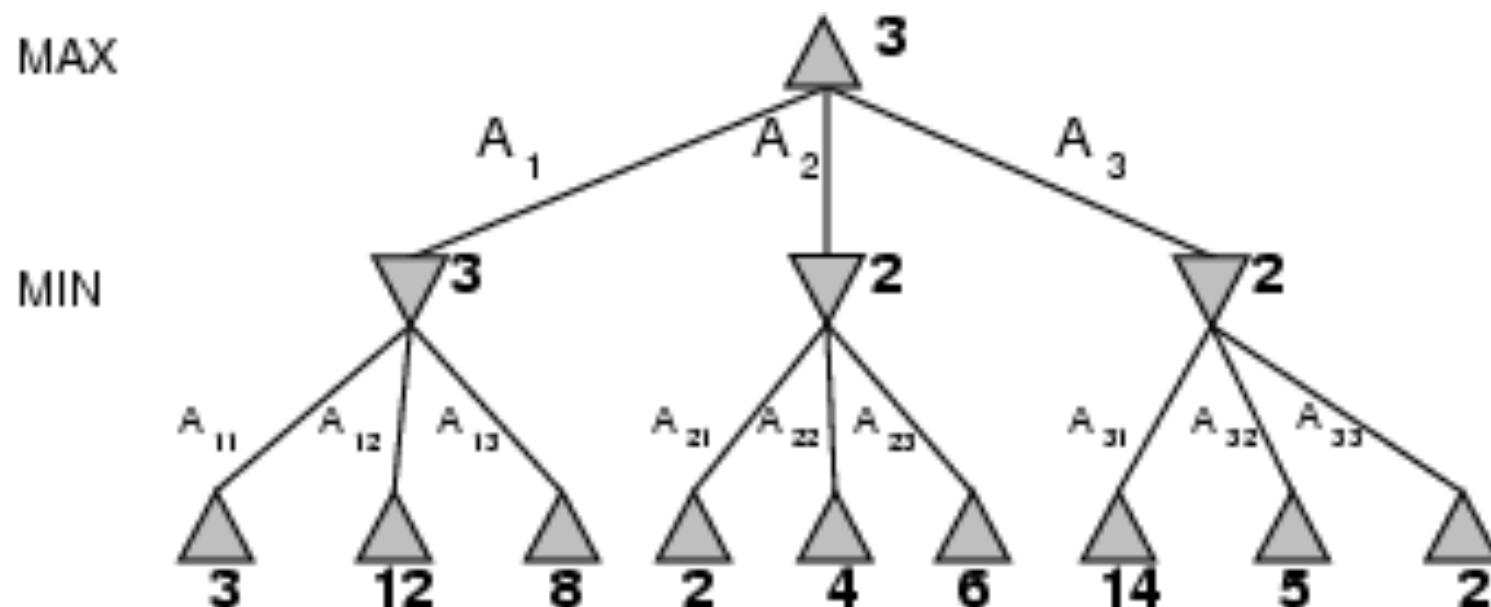
- The name minimax arises because each player minimizes the maximum payoff possible for the other—since the game is zero-sum, he also maximizes his own minimum payoff.
- Theorem established by *John von Neumann*, who is quoted as saying:
 - "As far as I can see, there could be no theory of games ... without that theorem ... I thought there was nothing worth publishing until the *Minimax Theorem* was proved".

Optimal Strategies (cont.)

- Optimal strategy leads to outcomes at *least* as good as any other strategy when one is playing an *infallible opponent*.
- Will do even better with fallible opponent.
- First, consider *optimal strategy*, even though its computationally problematic for all but the simplest games!
- Given a game tree, the optimal strategy can be determined by examining the *minimax* value of each node.

Minimax

- Perfect play for deterministic games
- Idea: choose move with highest *minimax value*
= best achievable payoff against best play
- E.g., 2-ply game:



Minimax algorithm

```
function MINIMAX-DECISION(state) returns an action
```

```
    v  $\leftarrow$  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  $\leftarrow -\infty$ 
```

```
    for a, s in SUCCESSORS(state) do
```

```
        v  $\leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$ 
```

```
    return v
```

```
function MIN-VALUE(state) returns a utility value
```

```
    if TERMINAL-TEST(state) then return UTILITY(state)
```

```
    v  $\leftarrow \infty$ 
```

```
    for a, s in SUCCESSORS(state) do
```

```
        v  $\leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$ 
```

```
    return v
```

Properties of minimax

- Complete? Yes! (if tree is finite)
 - Optimal? Yes! (against an *optimal* opponent)
 - Time complexity? $O(b^m)$ – ouch!
 - Space complexity? $O(bm)$ (depth-first exploration)
-
- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
→ exact solution completely infeasible: 35^{100}
 - *Note: minimax can easily be extended to > 2 players.*

Alpha-beta Pruning

Huge number of games states - $O(b^m)$

- Can't eliminate exponent, but we can cut it in ~half.
- Trick:
 - Possible to compute the correct *minimax* decision without looking at *every* node in the game tree.
 - Using idea of pruning (from informed search) to eliminate (potentially) large parts of tree.
- **Alpha-beta** pruning
 - Apply to *minimax* game tree
 - Prune away branches that cannot possibly influence the final decision.

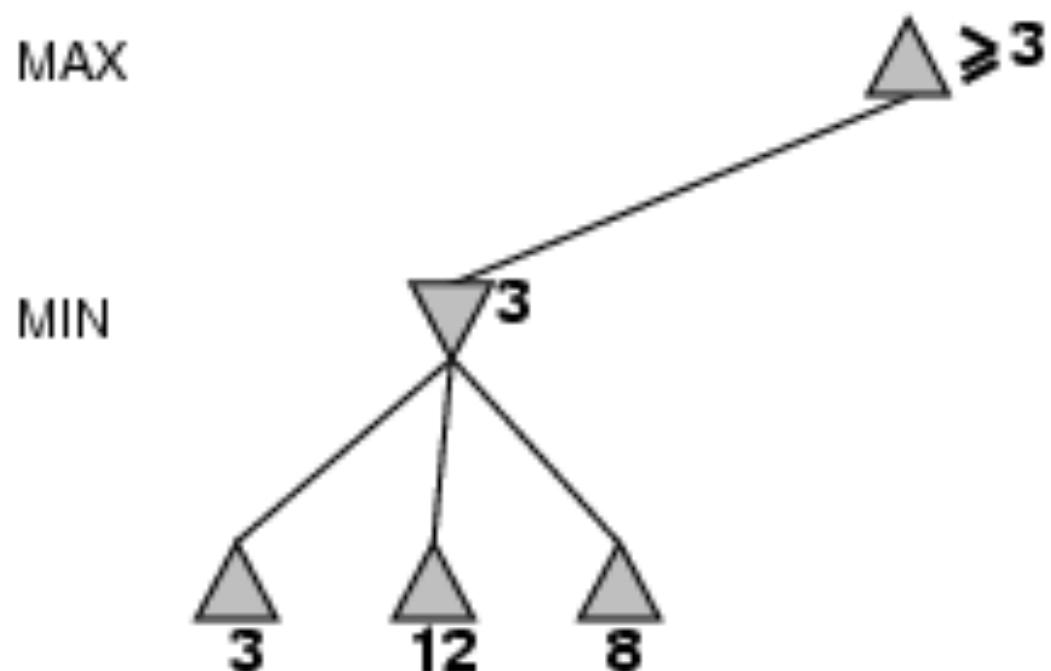
Alpha-beta Pruning (cont.)

Consider simplification of the formula for minimax-value:

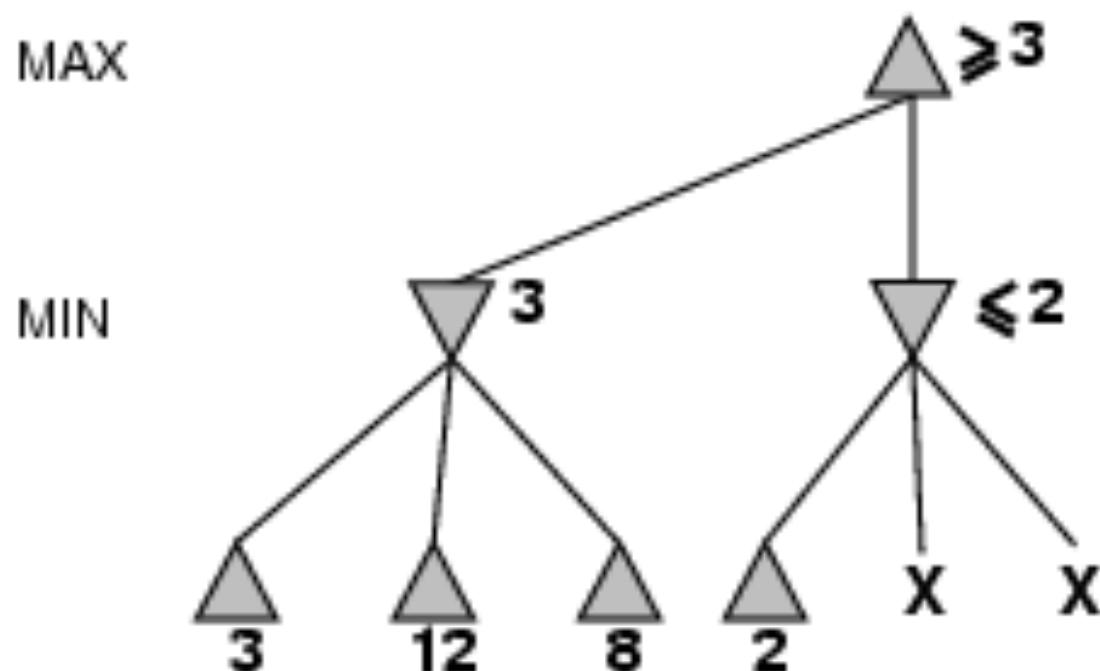
Minimax-value(root)

$$\begin{aligned} &= \max (\min(3,12,8), \min(2,x,y), \min(14,5,2)) \text{ // leaf nodes} \\ &= \max (3, \min (2,x,y), 2) \\ &= \max (3,z,2) \text{ where } z \leq 2 \\ &= 3 \end{aligned}$$

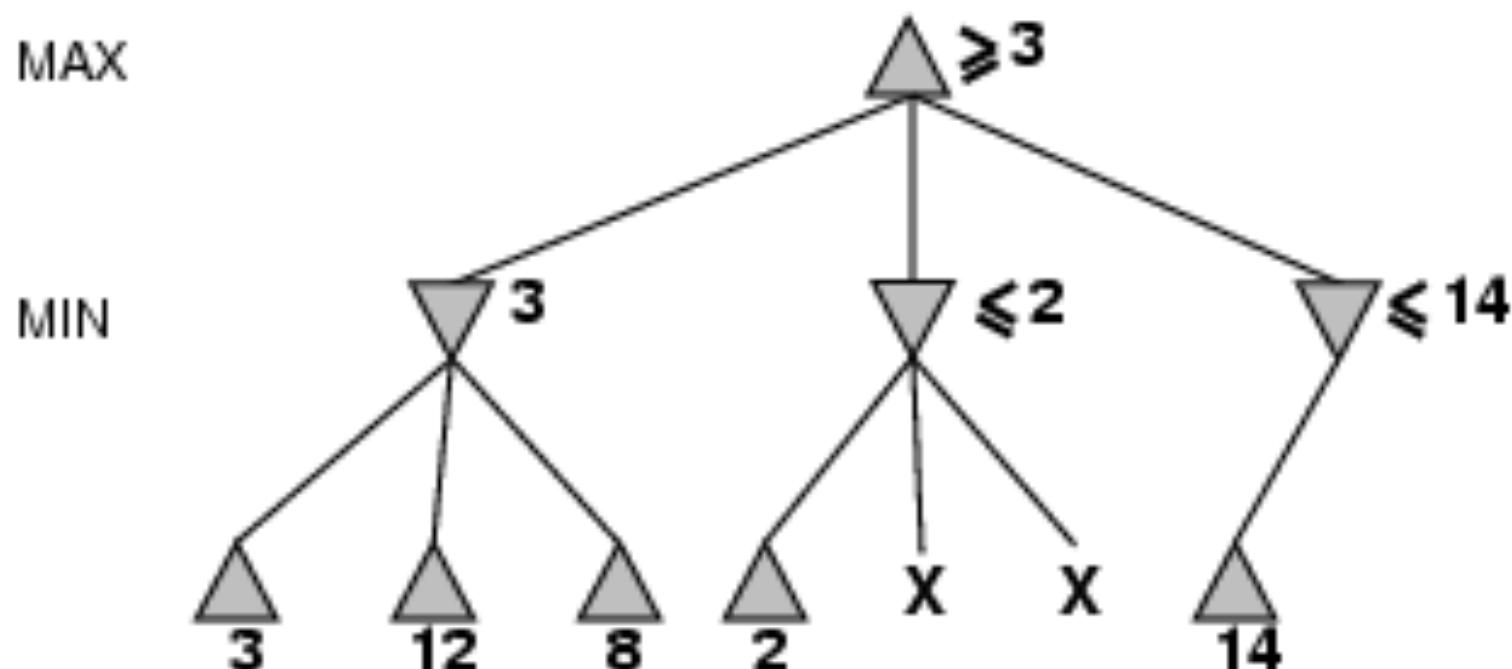
α - β pruning example



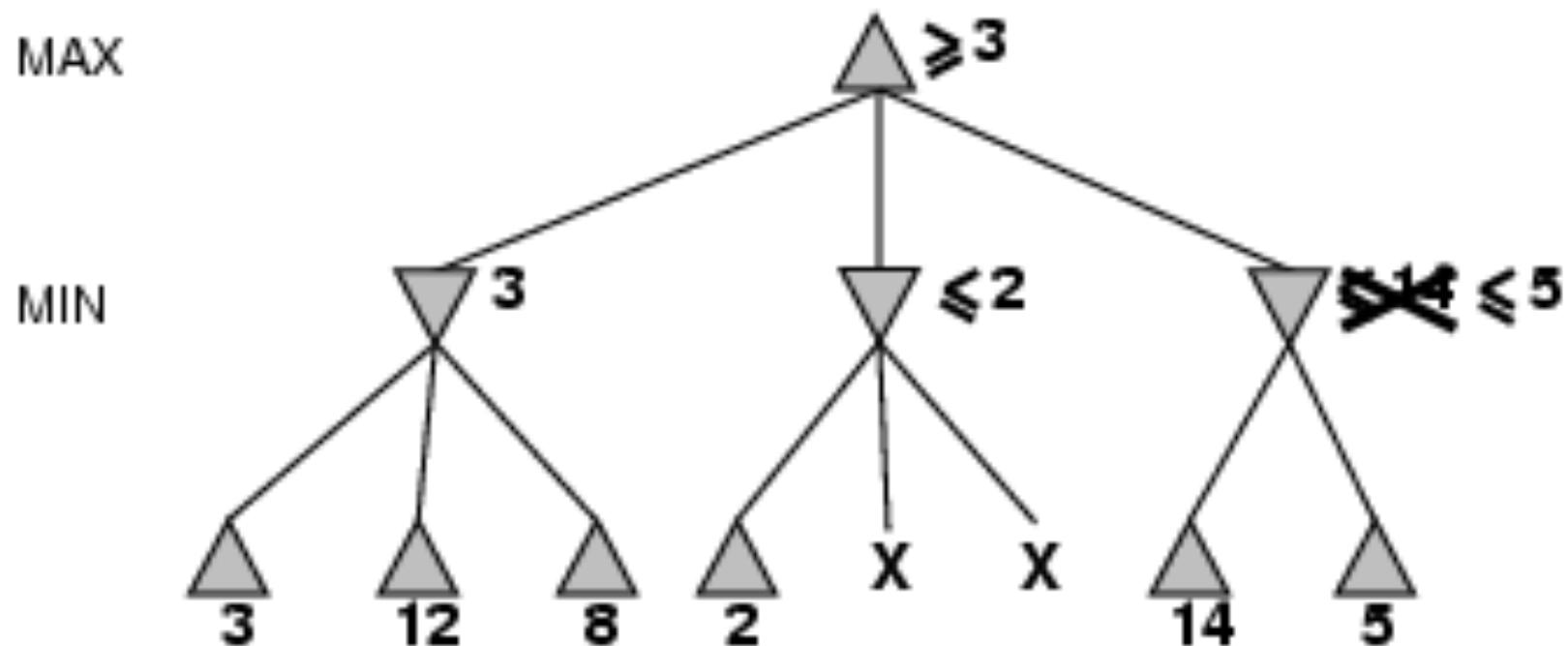
α - β pruning example



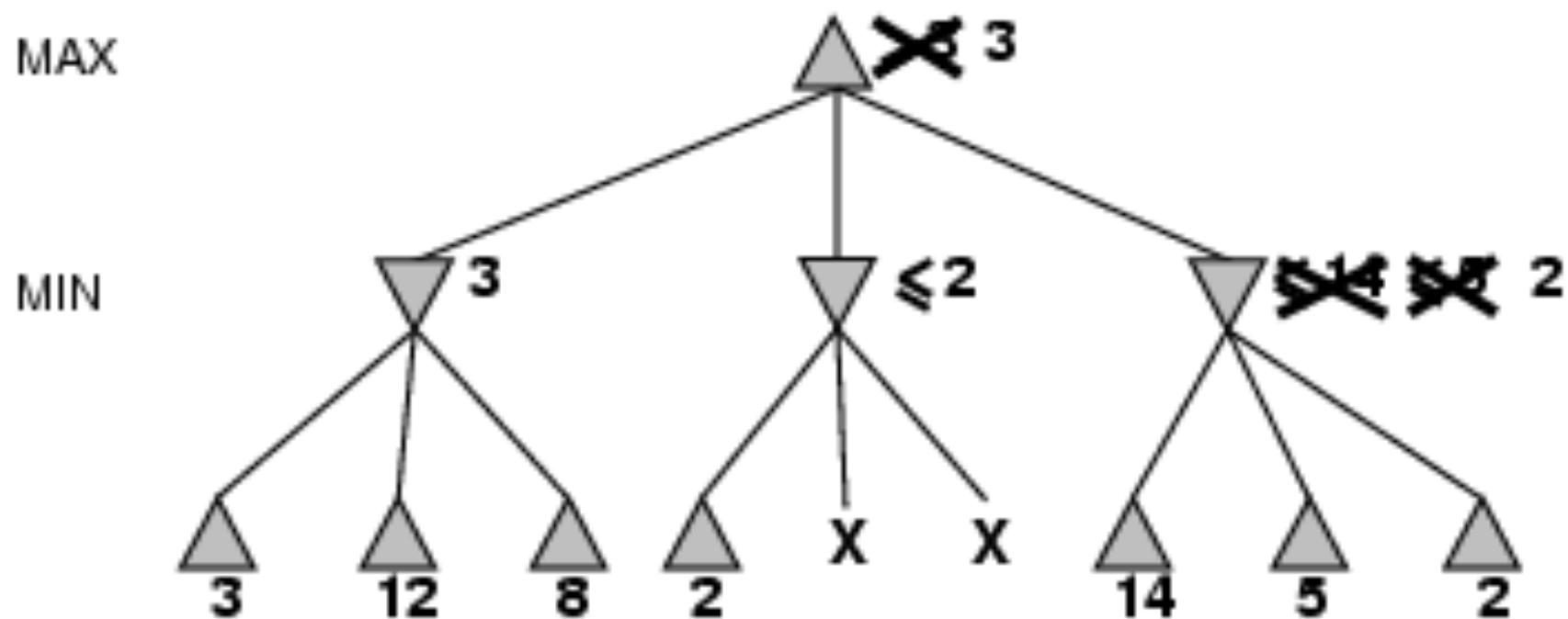
α - β pruning example



α - β pruning example



α - β pruning example

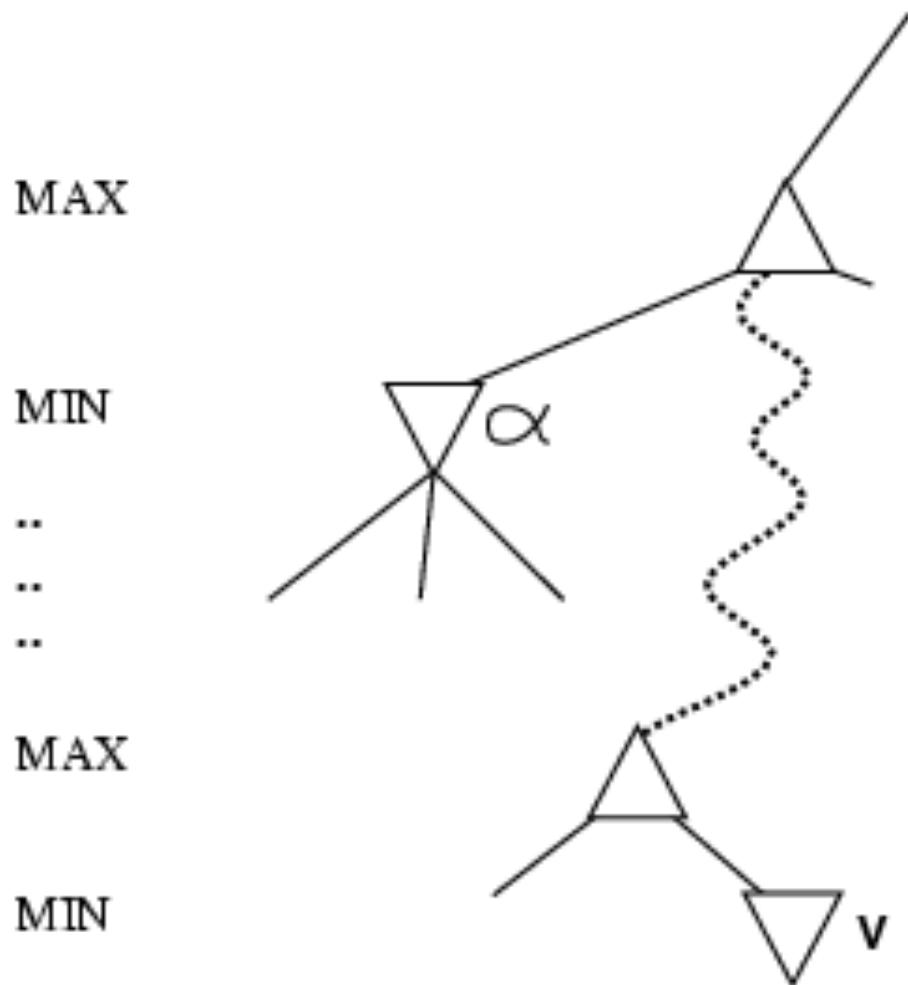


Properties of α - β

- Pruning **does not** affect the final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$
- Example of the value of reasoning about which computations are relevant (*meta-reasoning*)

α - β ?

- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max*
- If v is worse than α , *max* will avoid it
→ prune that branch
- Define β similarly for *min*



The α - β algorithm

function ALPHA-BETA-SEARCH(*state*) **returns** *an action*

inputs: *state*, current state in game

v \leftarrow MAX-VALUE(*state*, $-\infty$, $+\infty$)

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 $\leftarrow -\infty$

for *a, s* in SUCCESSORS(*state*) **do**

v $\leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$

if *v* $\geq \beta$ **then return** *v*

$\alpha \leftarrow \text{MAX}(\alpha, v)$

return *v*

The α - β algorithm

```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
    inputs: state, current state in game
             $\alpha$ , the value of the best alternative for MAX along the path to state
             $\beta$ , the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
     $v \leftarrow +\infty$ 
    for a, s in SUCCESSORS(state) do
         $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta))$ 
        if  $v \leq \alpha$  then return v
         $\beta \leftarrow \text{MIN}(\beta, v)$ 
    return v
```

Resource limits

Suppose we have 100 seconds, explore 10^4 nodes/sec
→ 10^6 nodes per move

Standard approach:

- **cutoff test:**
e.g., depth limit (perhaps add *quiescence search*)
- **evaluation function (heuristic)**
= estimated desirability of position

Resource limits

Quiescence search –

- Select values unlikely to exhibit wild swings in value.
- E.g. chess - in positions where favorable captures can be made are *not* quiescent for an evaluation function that just counts material.
- Nonquiescent positions can be expanded further until quiescent positions are reached.

Horizon affect –

- Arises when the program is facing a move by the opponent that causes serious damage and is ultimately unavoidable.
- Hard to avoid.
- E.g., pawn making it across the board & being replaced by queen.
- Can use *singular exclusions* – move that is clearly better than any other move at the current position. Can go beyond normal depth limit since its branching factor = 1. Very effective in chess.

Evaluation functions

- For chess, typically **linear** weighted sum of **features**

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

- e.g., $w_1 = 9$ with

$f_1(s) = (\text{number of white queens}) - (\text{number of black queens})$, etc.

- Can use Machine Learning to optimize your *weights*.
- *More sophisticated – learn moves!*

Cutting off search

MinimaxCutoff is identical to *MinimaxValue* except

1. *Terminal* is replaced by *Cutoff*
2. *Utility* is replaced by *Eval*

Does it work in practice?

$$b^m = 10^6, b=35 \rightarrow m=4$$

4-ply lookahead is a hopeless chess player!

- 4-ply \approx human novice
- 8-ply \approx typical PC, human master
- 12-ply \approx Deep Blue, Kasparov

Deterministic games in practice

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a pre-computed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of *444 billion* positions.
- Chess: Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches *200 million* positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.
- Othello: human champions refuse to compete against computers, who are too good.
- In 2008, thanks to an efficient message-passing parallelization, **MoGo** won one game (out of three) against Catalin Taranu, 5th dan pro, in 9x9 with standard time settings (30 minutes per side).



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

