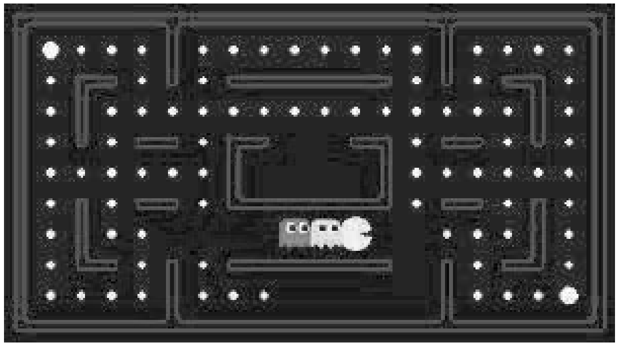
Games I



Course plan

Search problems

Markov decision processes

Adversarial games

Constraint satisfaction problems

Markov networks

Bayesian networks

Reflex **States** **Variables** **Logic**

Low-level → High-level

Machine learning

A simple game

Example: game 1

You choose one of the three bins.

I choose a number from that bin.

Your goal is to maximize the chosen number.

A

-50 50

B

1 3

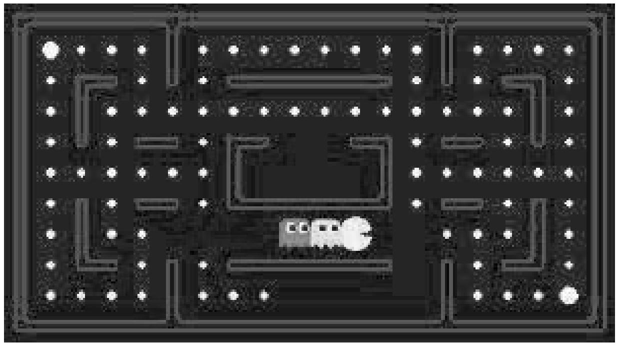
C

-5 15

Roadmap

Modeling	**Learning**
Modeling Games	Temporal Difference Learning
Algorithms	**Other Topics**
Game Evaluation	Simultaneous Games
Expectimax	Non-Zero-Sum Games
Minimax	
Expectiminimax	
Evaluation Functions	
Alpha-Beta Pruning	

Games: modeling

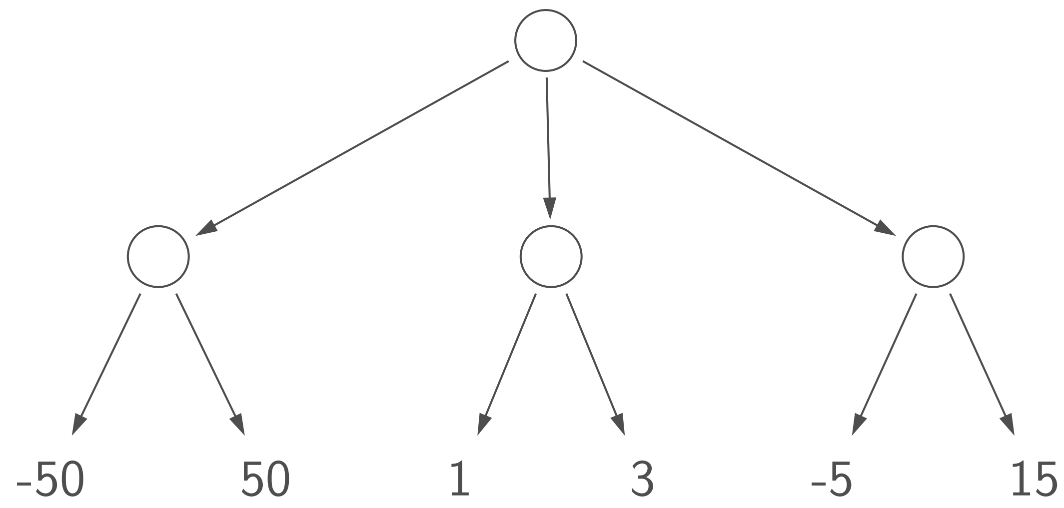


Game tree

Key idea: game tree

Each node is a decision point for a player.

Each root-to-leaf path is a possible outcome of the game.



Two-player zero-sum games

Players $= \{\text{agent}, \text{opp}\}$

Definition: two-player zero-sum game

s_{start}: starting state

Actions(s): possible actions from state s

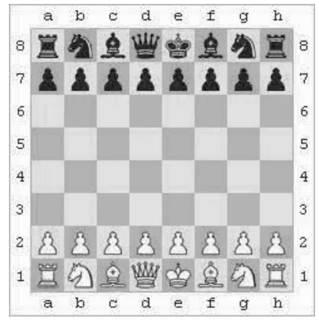
Succ(s, a): resulting state if choose action a in state s

IsEnd(s): whether s is an end state (game over)

Utility(s): agent's utility for end state s

Player$(s) \in$ Players: player who controls state s

Example: chess



Players $=$ {white, black}

State s: (position of all pieces, whose turn it is)

Actions(s): legal chess moves that Player(s) can make

IsEnd(s): whether s is checkmate or draw

Utility(s): $+\infty$ if white wins, 0 if draw, $-\infty$ if black wins

Characteristics of games

- All the utility is at the end state

- Different players in control at different states

The halving game

Problem: halving game

Start with a number N.

Players take turns either decrementing N or replacing it with $\lfloor \frac{N}{2} \rfloor$.

The player that is left with 0 wins.

[live solution: `HalvingGame`]

Policies

Deterministic policies: $\pi_p(s) \in \text{Actions}(s)$

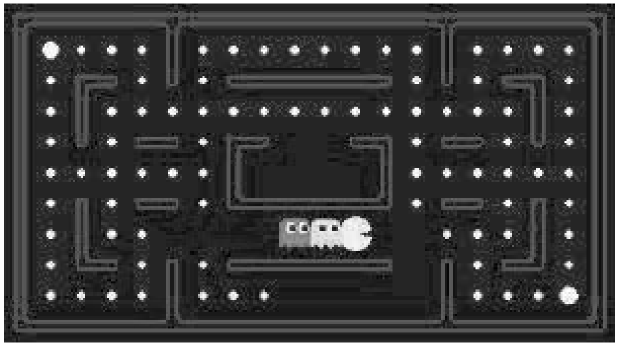
action that player p takes in state s

Stochastic policies $\pi_p(s, a) \in [0, 1]$:

probability of player p taking action a in state s

[live solution: policies, main loop]

Games: game evaluation

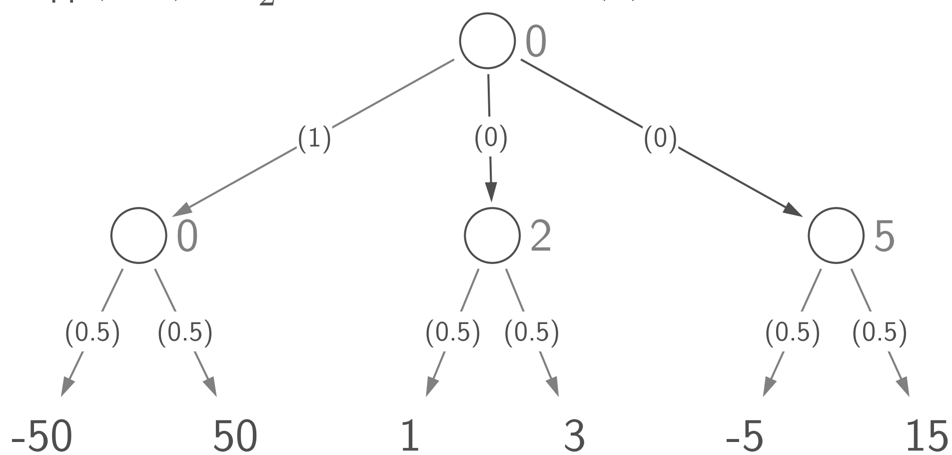


Game evaluation example

Example: game evaluation

$\pi_{\text{agent}}(s) = \text{A}$

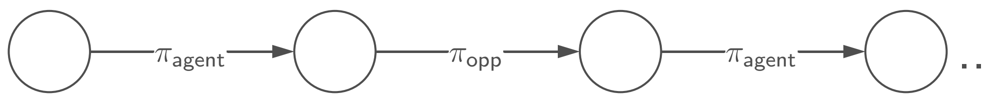
$\pi_{\text{opp}}(s, a) = \frac{1}{2}$ for $a \in \text{Actions}(s)$



$\boxed{V_{\text{eval}}(s_{\text{start}}) = 0}$

Game evaluation recurrence

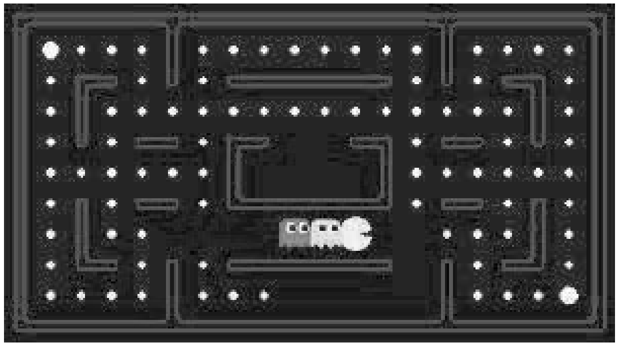
Analogy: recurrence for policy evaluation in MDPs



Value of the game:

$$V_{\text{eval}}(s) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \sum_{a \in \text{Actions}(s)} \pi_{\text{agent}}(s,a) V_{\text{eval}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{agent} \\ \sum_{a \in \text{Actions}(s)} \pi_{\text{opp}}(s,a) V_{\text{eval}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{opp} \end{cases}$$

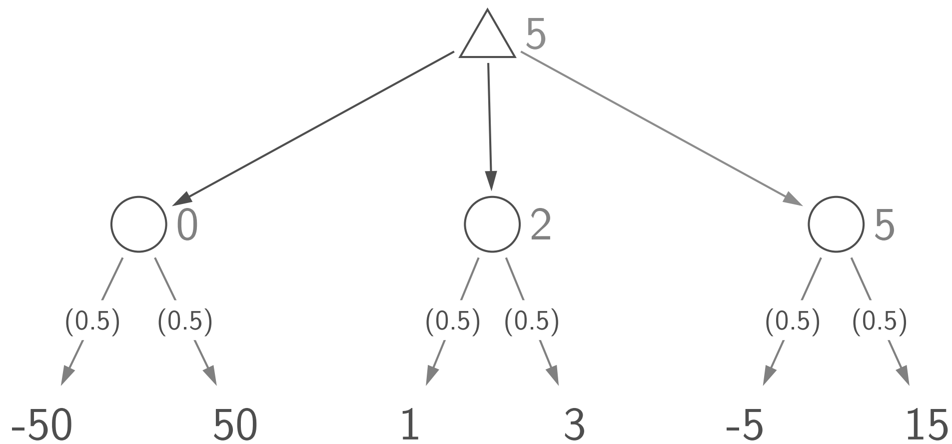
Games: expectimax



Expectimax example

Example: expectimax

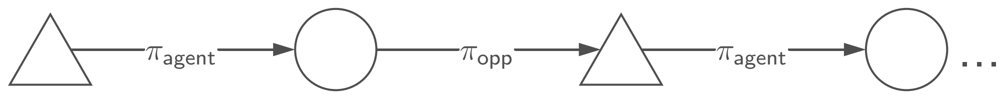
$\pi_{\text{opp}}(s, a) = \frac{1}{2}$ for $a \in \text{Actions}(s)$



$$\boxed{V_{\text{exptmax}}(s_{\text{start}}) = 5}$$

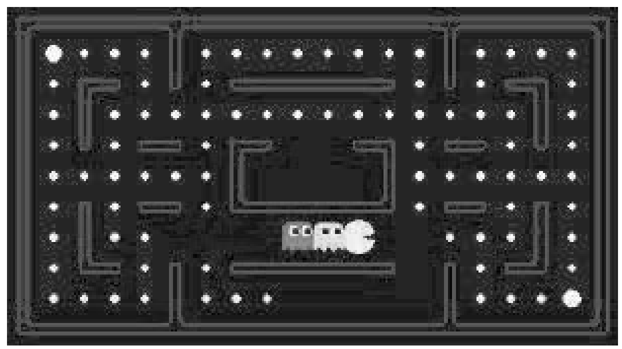
Expectimax recurrence

Analogy: recurrence for value iteration in MDPs



$$V_{\text{exptmax}}(s) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} V_{\text{exptmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{agent} \\ \sum_{a \in \text{Actions}(s)} \pi_{\text{opp}}(s,a) V_{\text{exptmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{opp} \end{cases}$$

Games: minimax

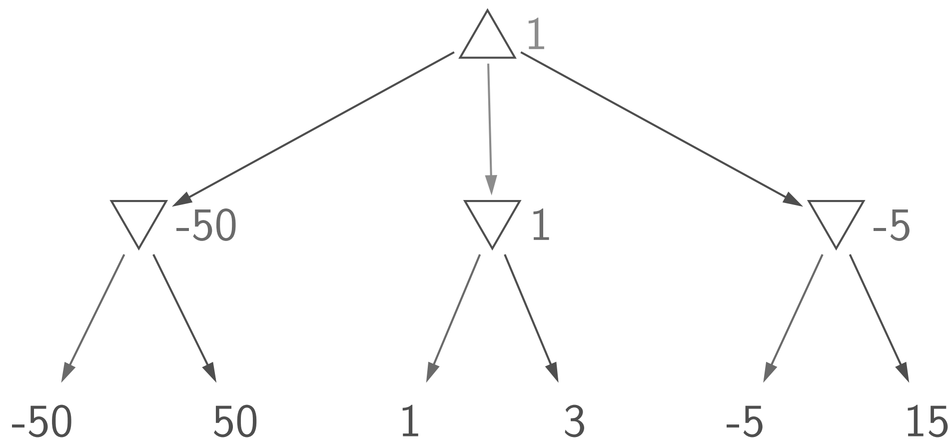


Problem: don't know opponent's policy

Approach: assume the worst case

Minimax example

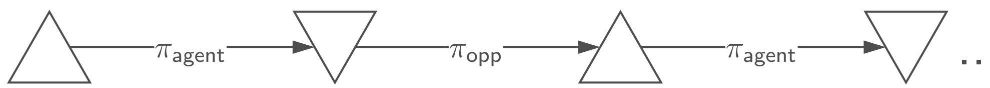
Example: minimax



$$\boxed{V_{\text{minmax}}(s_{\text{start}}) = 1}$$

Minimax recurrence

No analogy in MDPs:

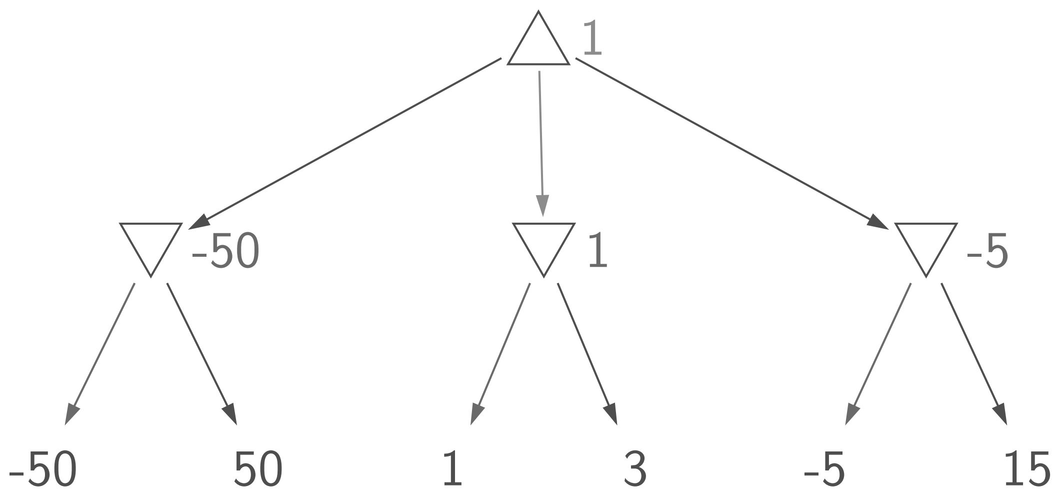


$$V_{\text{minmax}}(s) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} V_{\text{minmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{agent} \\ \min_{a \in \text{Actions}(s)} V_{\text{minmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{opp} \end{cases}$$

Extracting minimax policies

$$\pi_{\max}(s) = \arg \max_{a \in \text{Actions}(s)} V_{\text{minmax}}(\text{Succ}(s, a))$$

$$\pi_{\min}(s) = \arg \min_{a \in \text{Actions}(s)} V_{\text{minmax}}(\text{Succ}(s, a))$$



The halving game

Problem: halving game

Start with a number N.

Players take turns either decrementing N or replacing it with $\lfloor \frac{N}{2} \rfloor$.

The player that is left with 0 wins.

[live solution: `minimaxPolicy`]

Face off

Recurrences produce policies:

$$\begin{aligned} V_{\textsf{exptmax}} &\Rightarrow \pi_{\textsf{exptmax}(7)}, \pi_7 \text{ (some opponent)} \\ V_{\textsf{minmax}} &\Rightarrow \pi_{\max}, \pi_{\min} \end{aligned}$$

Play policies against each other:

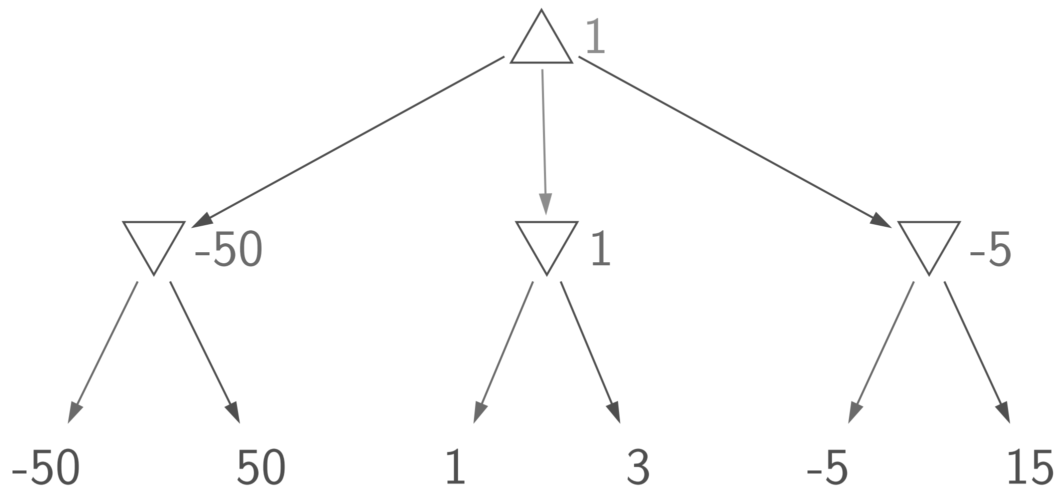
	$\pi_{\min}$	π_7
$\pi_{\max}$	$V(\pi_{\max}, \pi_{\min})$	$V(\pi_{\max}, \pi_7)$
$\pi_{\textsf{exptmax}(7)}$	$V(\pi_{\textsf{exptmax}(7)}, \pi_{\min})$	$V(\pi_{\textsf{exptmax}(7)}, \pi_7)$

What's the relationship between these values?

Minimax property 1

Proposition: best against minimax opponent

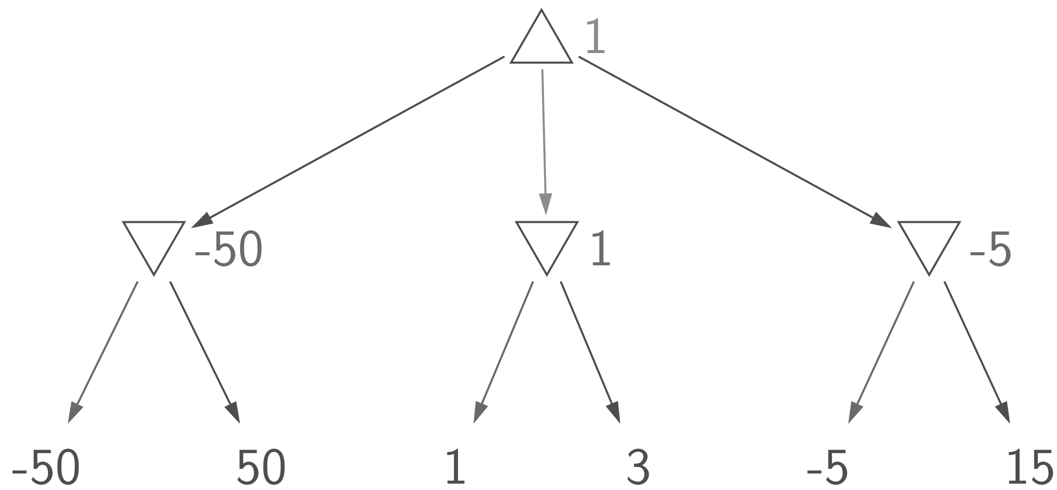
$V(\pi_{\max}, \pi_{\min}) \geq V(\pi_{\text{agent}}, \pi_{\min})$ for all π_{agent}



Minimax property 2

Proposition: lower bound against any opponent

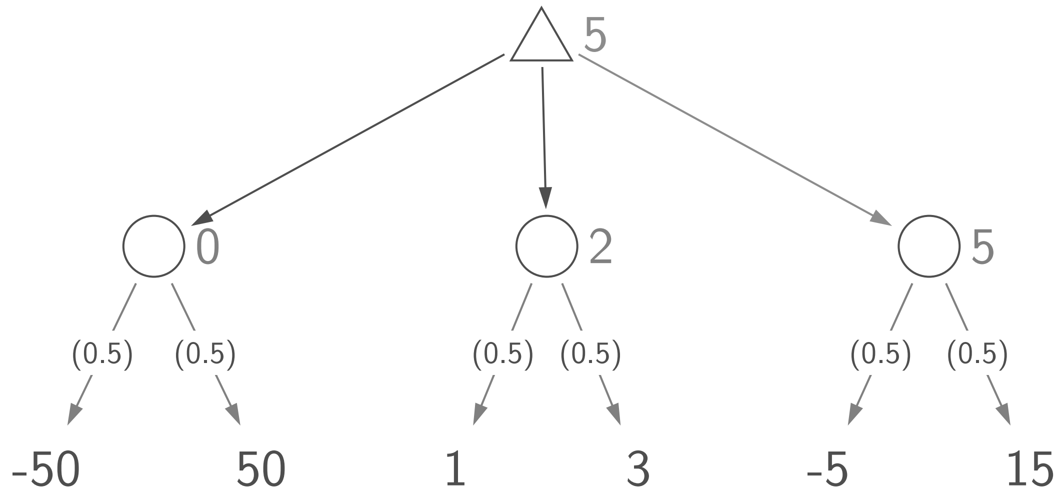
$V(\pi_{\max}, \pi_{\min}) \leq V(\pi_{\max}, \pi_{\text{opp}})$ for all π_{opp}



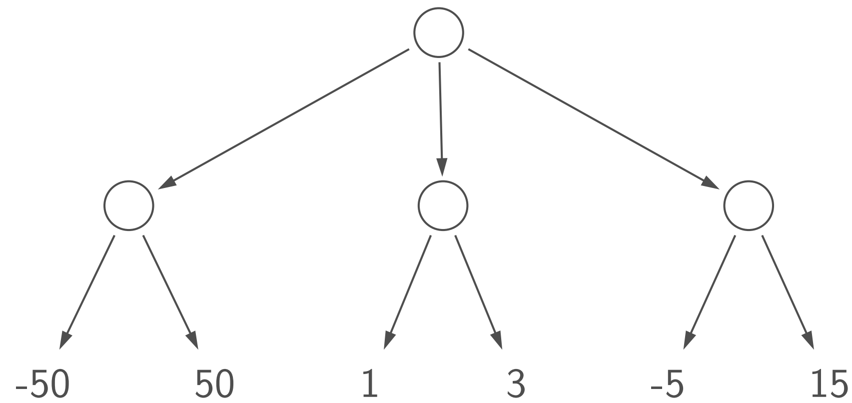
Minimax property 3

Proposition: not optimal if opponent is known

$V(\pi_{\max}, \pi_7) \leq V(\pi_{\text{exptmax}(7)}, \pi_7)$ for opponent π_7

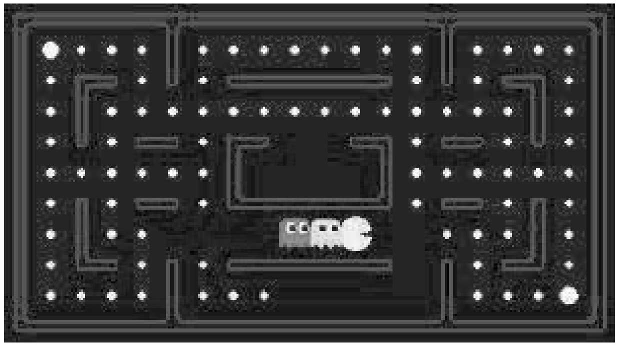


Relationship between game values



	$\pi_{\min}$		π_7
$\pi_{\max}$	$V(\pi_{\max}, \pi_{\min})$ 1	$\leq$	$V(\pi_{\max}, \pi_7)$ 2
	$\geq$		$\leq$
$\pi_{\text{exptmax}(7)}$	$V(\pi_{\text{exptmax}(7)}, \pi_{\min})$ -5		$V(\pi_{\text{exptmax}(7)}, \pi_7)$ 5

Games: expectiminimax



A modified game

Example: game 2

You choose one of the three bins.

Flip a coin; if heads, then move one bin to the left (with wrap around).

I choose a number from that bin.

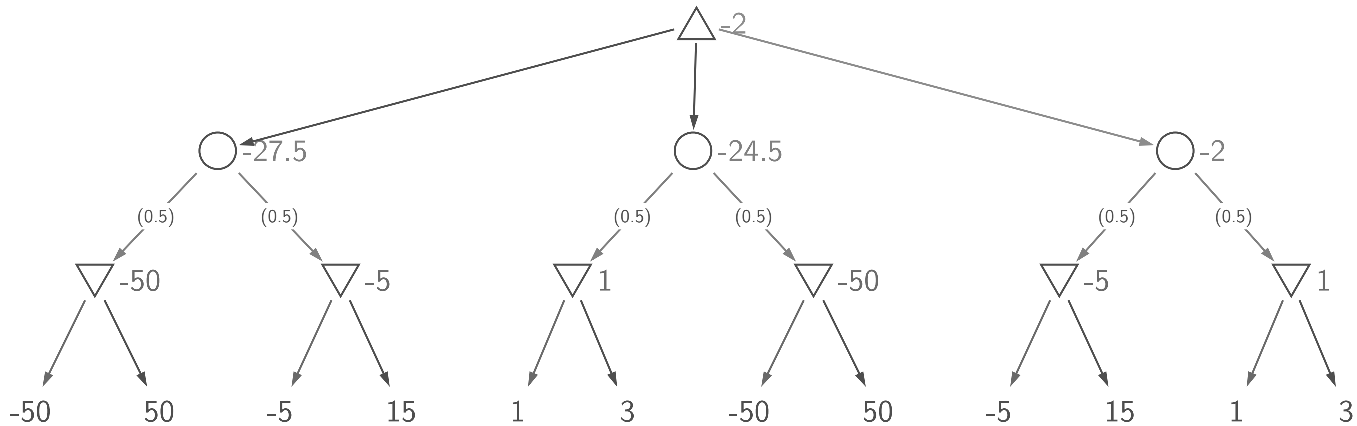
Your goal is to maximize the chosen number.

A	B	C
-50 50	1 3	-5 15

Expectiminimax example

Example: expectiminimax

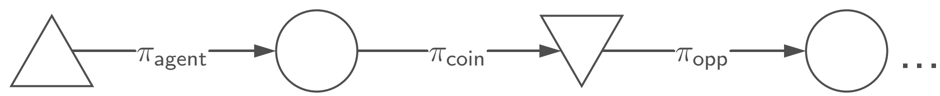
$\pi_{\text{coin}}(s, a) = \frac{1}{2}$ for $a \in \{0, 1\}$



$$\boxed{V_{\text{exptminmax}}(s_{\text{start}}) = -2}$$

Expectiminimax recurrence

Players $= \{\text{agent}, \text{opp}, \text{coin}\}$



$$V_{\text{exptminmax}}(s) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} V_{\text{exptminmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{agent} \\ \min_{a \in \text{Actions}(s)} V_{\text{exptminmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{opp} \\ \sum_{a \in \text{Actions}(s)} \pi_{\text{coin}}(s,a) V_{\text{exptminmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{coin} \end{cases}$$

Summary so far

Primitives: max nodes, chance nodes, min nodes

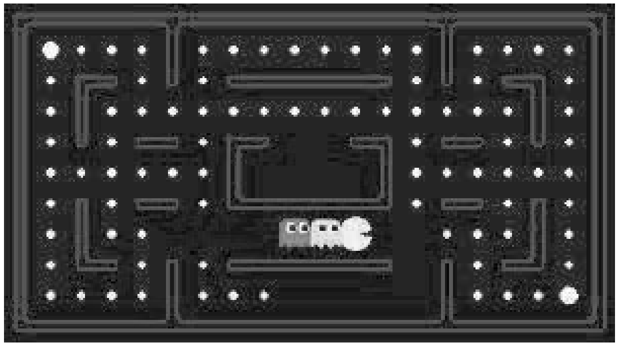
Composition: alternate nodes according to model of game

Value function $V_{\dots}(s)$: recurrence for expected utility

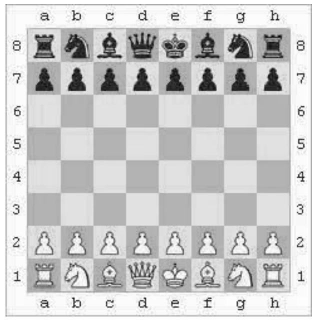
Scenarios to think about:

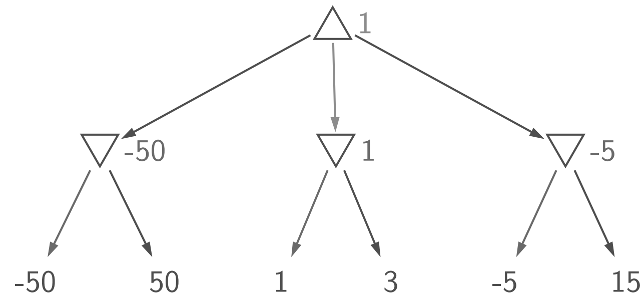
- What if you are playing against multiple opponents?
- What if you and your partner have to take turns (table tennis)?
- Some actions allow you to take an extra turn?

Games: evaluation functions



Computation





Approach: tree search

Complexity:

- branching factor b, depth d ($2d$ plies)
- $O(d)$ space, $O(b^{2d})$ time

Chess: $b \approxeq 35$, $d \approxeq 50$

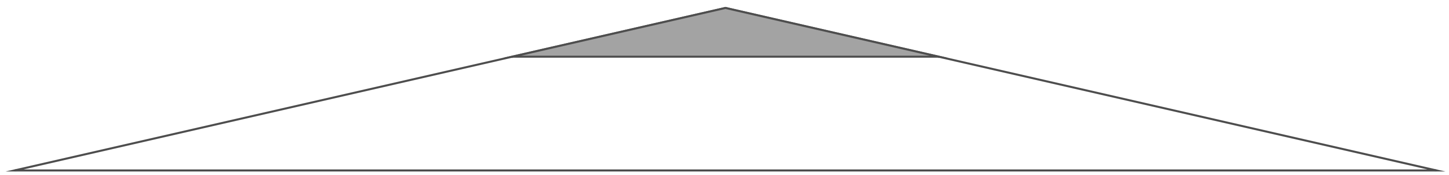
2551552067298685292412115015142558763019041448816101932417677844077146725823993736584373298704355578978233619563773665328554329789767507463693618774414062 5

Speeding up minimax

- Evaluation functions: use domain-specific knowledge, compute approximate answer

- Alpha-beta pruning: general-purpose, compute exact answer

Depth-limited search



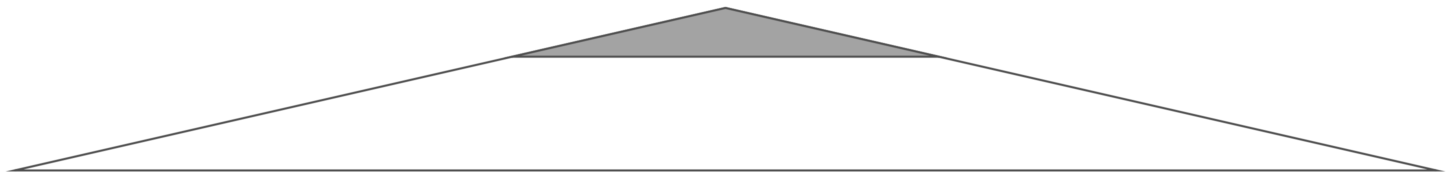
Limited depth tree search (stop at maximum depth d_{max}):

$$V_{\text{minmax}}(s, d) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \text{Eval}(s) & d = 0 \\ \max_{a \in \text{Actions}(s)} V_{\text{minmax}}(\text{Succ}(s, a), d) & \text{Player}(s) = \text{agent} \\ \min_{a \in \text{Actions}(s)} V_{\text{minmax}}(\text{Succ}(s, a), d - 1) & \text{Player}(s) = \text{opp} \end{cases}$$

Use: at state s, call $V_{\text{minmax}}(s, d_{\text{max}})$

Convention: decrement depth at last player's turn

Evaluation functions

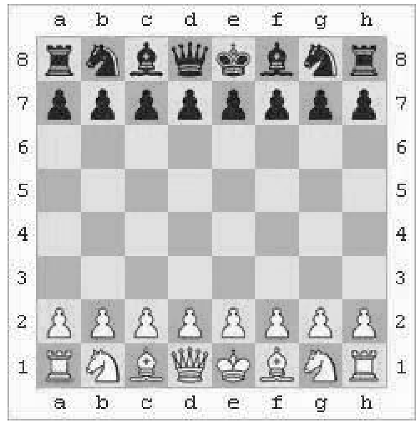


Definition: Evaluation function

An evaluation function $\text{Eval}(s)$ is a (possibly very weak) estimate of the value $V_{\text{minmax}}(s)$.

Analogy: $\text{FutureCost}(s)$ in search problems

Evaluation functions



Example: chess

Eval(s) $=$ material $+$ mobility $+$ king-safety $+$ center-control

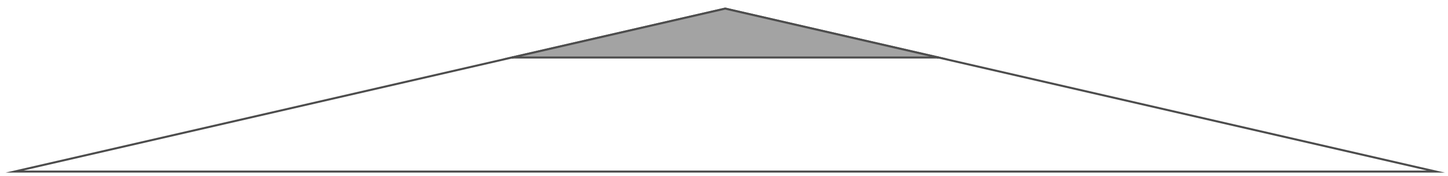
material $= 10^{100}(K - K') + 9(Q - Q') + 5(R - R') +$
$3(B - B' + N - N') + 1(P - P')$

mobility $= 0.1($num-legal-moves $-$ num-legal-moves$')$

...

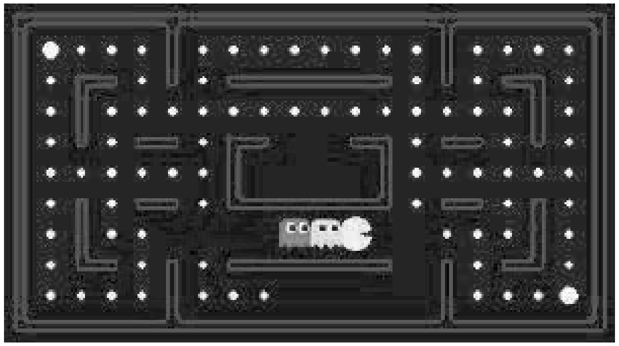
Summary: evaluation functions

Depth-limited exhaustive search: $O(b^{2d})$ time



- $\text{Eval}(s)$ attempts to estimate $V_{\text{minmax}}(s)$ using domain knowledge
- No guarantees (unlike A*) on the error from approximation

Games: alpha-beta pruning



Pruning principle

Choose A or B with maximum value:

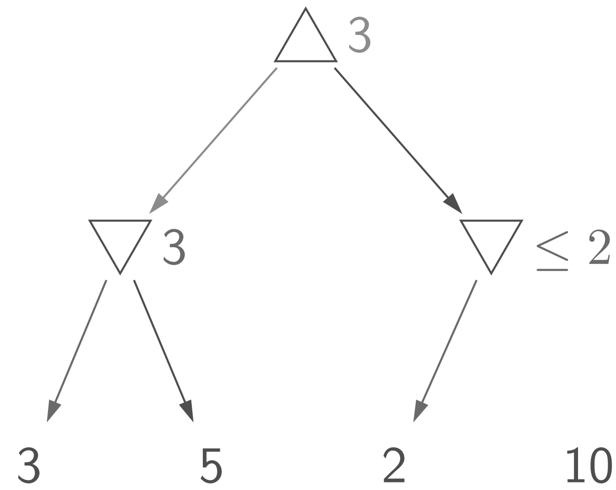
A: [3, **5**]   B: [**5**, 100]

Key idea: branch and bound

Maintain lower and upper bounds on values.

If intervals don't overlap non-trivially, then can choose optimally without further work.

Pruning game trees



Once we see 2, we know that value of right node must be ≤ 2

Root computes $\max(3, \leq 2) = 3$

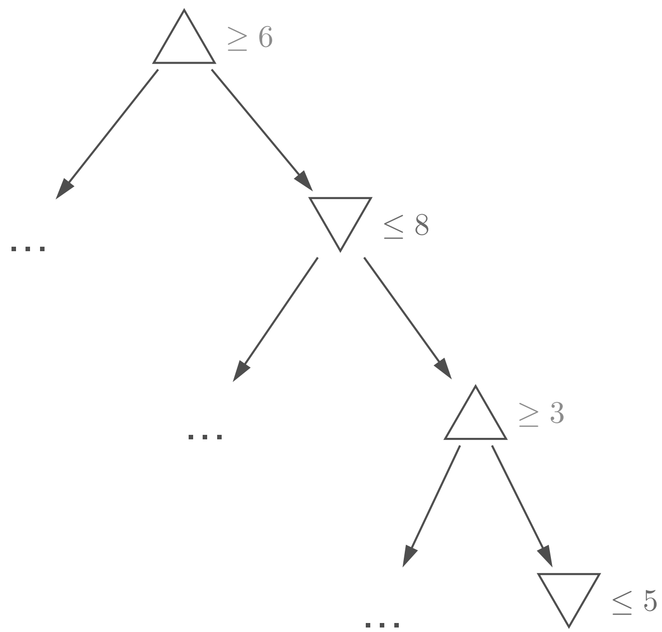
Since branch doesn't affect root value, can safely prune

Alpha-beta pruning

Key idea: optimal path

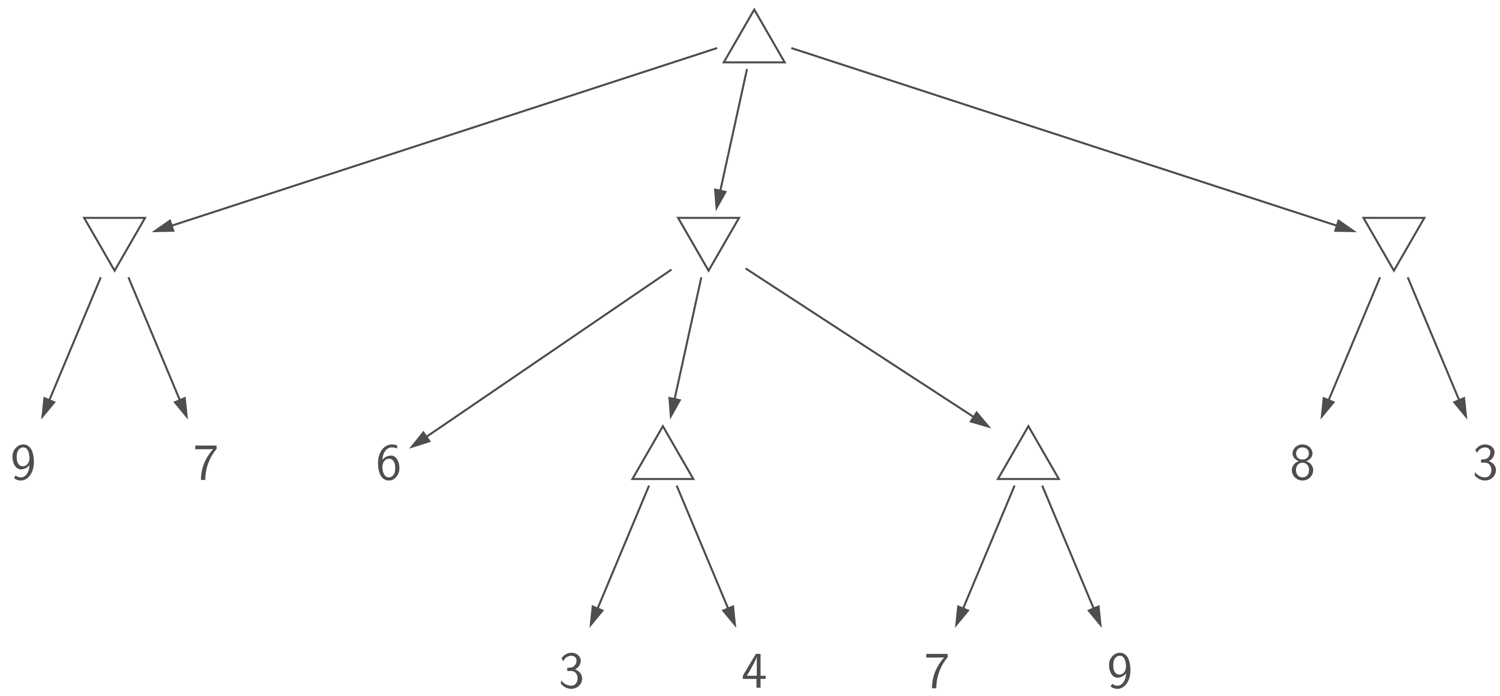
The optimal path is path that minimax policies take.

Values of all nodes on path are the same.



- a_s: lower bound on value of max node s
- b_s: upper bound on value of min node s
- Prune a node if its interval doesn't have non-trivial overlap with every ancestor (store $\alpha_s = \max_{s' \preceq s} a_{s'}$ and $\beta_s = \min_{s' \preceq s} b_{s'}$)

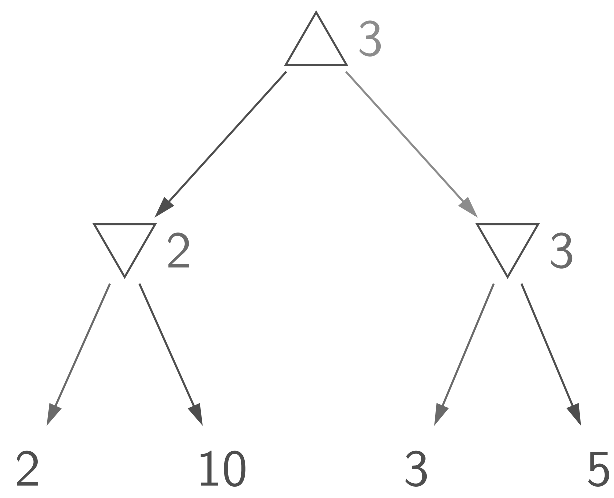
Alpha-beta pruning example



Move ordering

Pruning depends on order of actions.

Can't prune the 5 node:



Move ordering

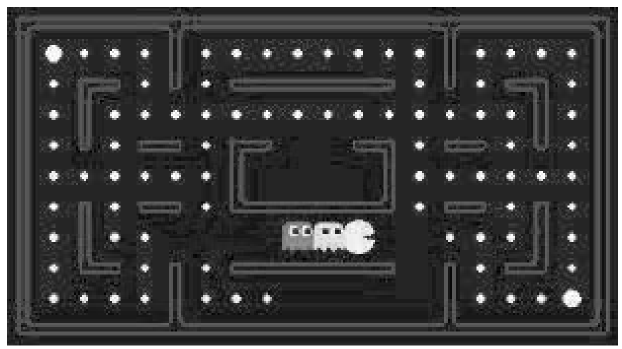
Which ordering to choose?

- Worst ordering: $O(b^{2\cdot d})$ time
- Best ordering: $O((\sqrt{b-\frac{3}{4}}+\frac{1}{2})^{2\cdot d}) \simeq O(b^{2\cdot 0.5d})$ time
- Random ordering: $O(b^{2\cdot 0.75d})$ time when $b=2$
- Random ordering: $O((\frac{b-1+\sqrt{b^2+14b+1}}{4})^{2\cdot d})$ for general b

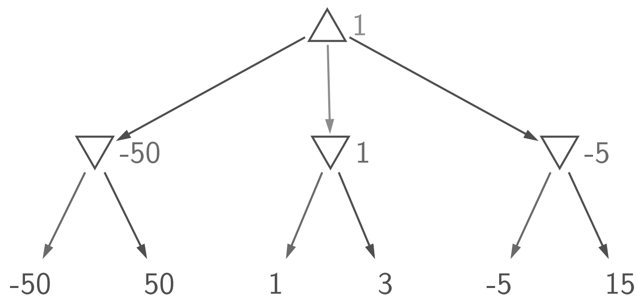
In practice, can use evaluation function $\text{Eval}(s)$:

- Max nodes: order successors by decreasing $\text{Eval}(s)$
- Min nodes: order successors by increasing $\text{Eval}(s)$

Games: recap



Summary



- Game trees: model opponents, randomness
- Minimax: find optimal policy against an adversary
- Evaluation functions: domain-specific, approximate
- Alpha-beta pruning: domain-general, exact