TDT4136 Introduction to Artificial Intelligence

Chapter 5 - Adversarial Search

Odd Erik Gundersen
odderik@idi.ntnu.no

Norwegian University of Science and Technology

2016-10-03

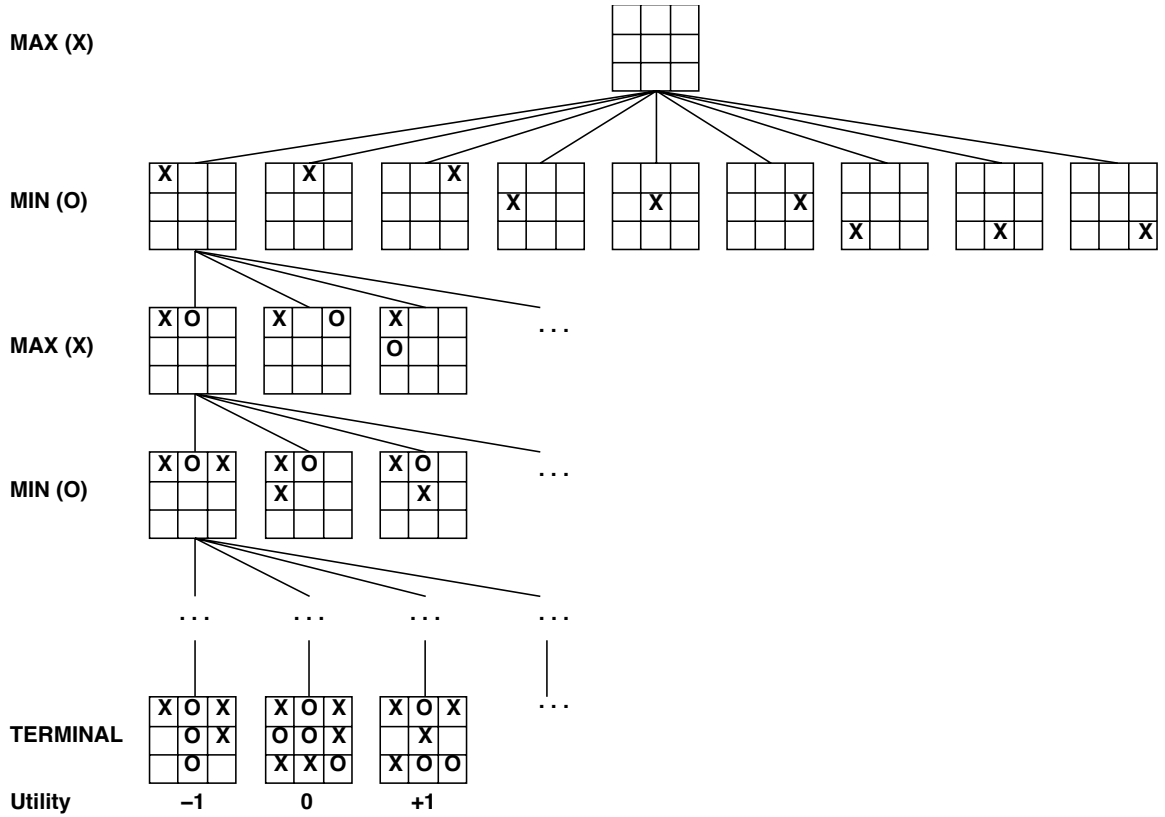
NTNU
Innovation and Creativity

Types of games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

- Perfect Info - state of playing arena and other players holdings known at all times.
- Imperfect Info - some info about arena or players not available.
- Deterministic - outcome of any action is certain.
- Stochastic - some actions have probabilistic outcomes, e.g. dealt cards, rolled dice, etc.

Game tree (2-player, deterministic, turns)



Games vs. search problems

The search tree is different from standard search. Here it is a two-ply tree where at each alternating level one of the players has the control/decisions.

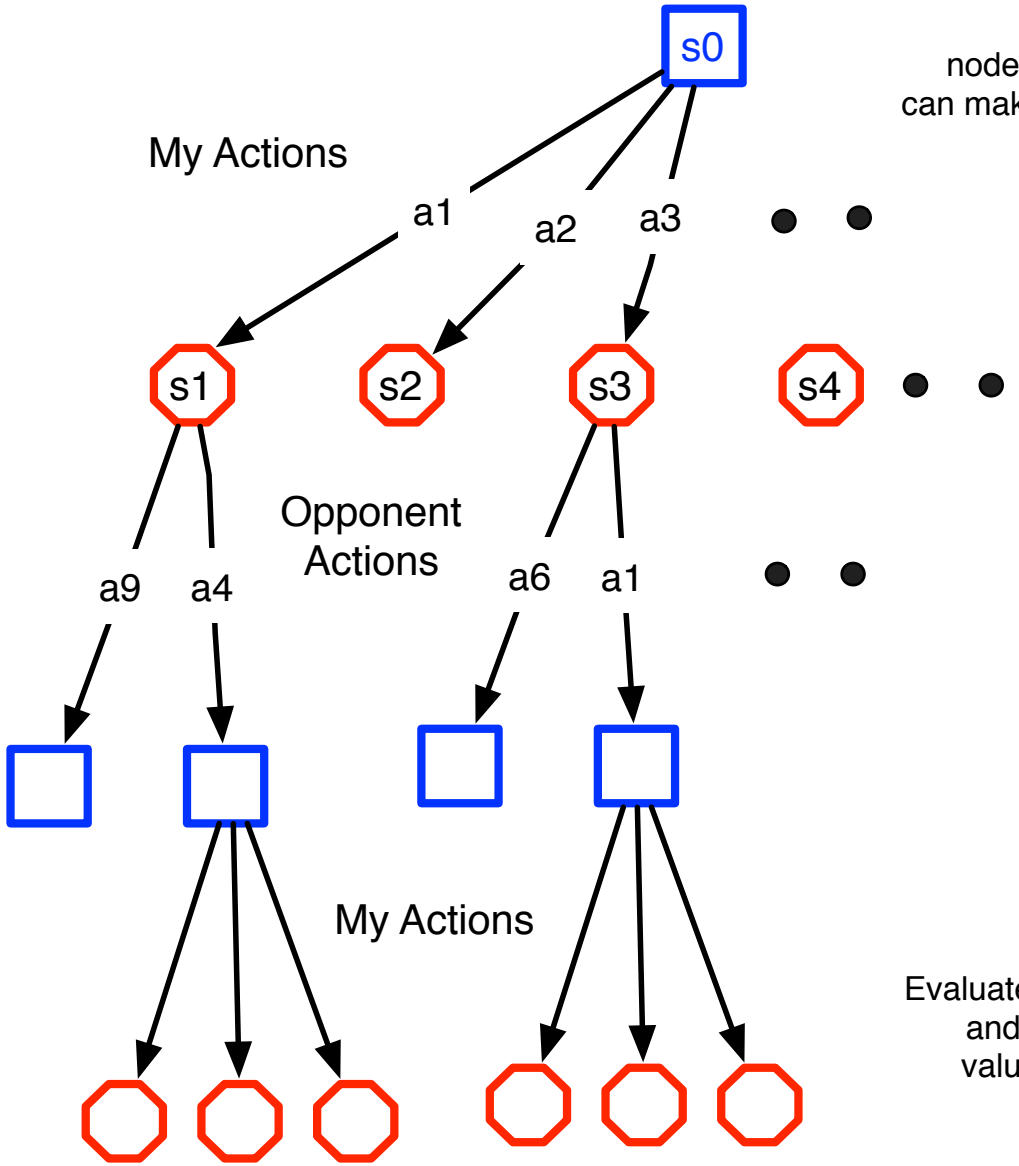
The player we want to win the game starts the game

Each leaf in the search tree is assigned a utility value - often 1 win, -1 lose, 0, draw

The search is about maximizing the utility of the computer/first player (called MAX)

We assume that the opponent is "unpredictable" $\Rightarrow$ solution is a strategy specifying a move for every possible opponent reply

Adversarial Search Trees



Similar to an OR node in AND/OR Search. I can make ANY of these choices.

Similar to AND nodes in AND/OR . I must account for ALL of the opponent's possible moves.

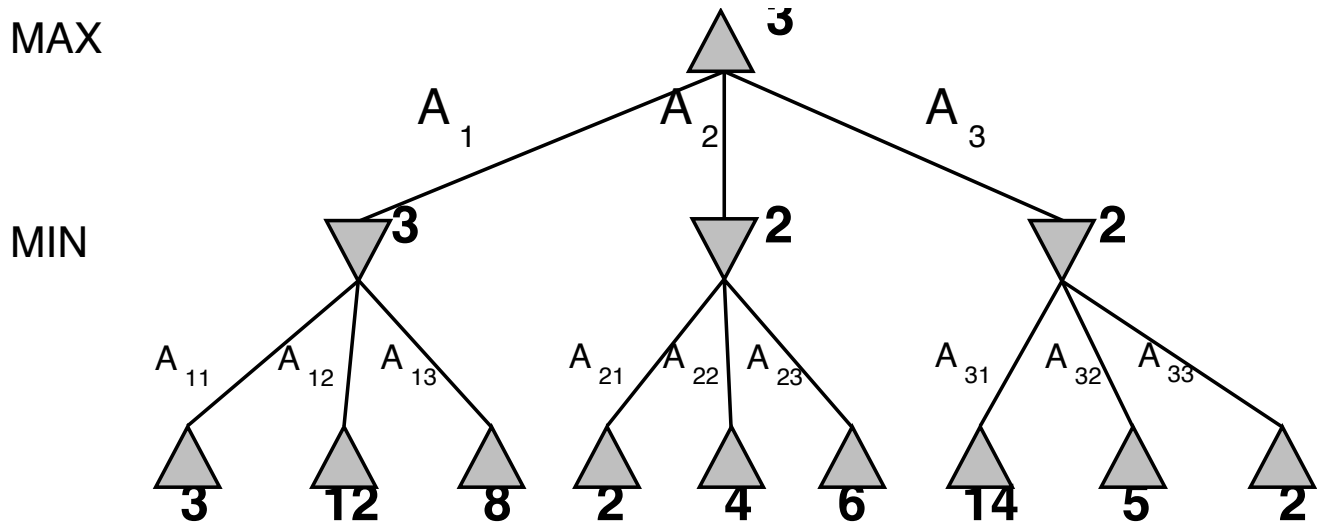
Evaluate bottom states and propagate values upward.

Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value
= best achievable payoff against best play

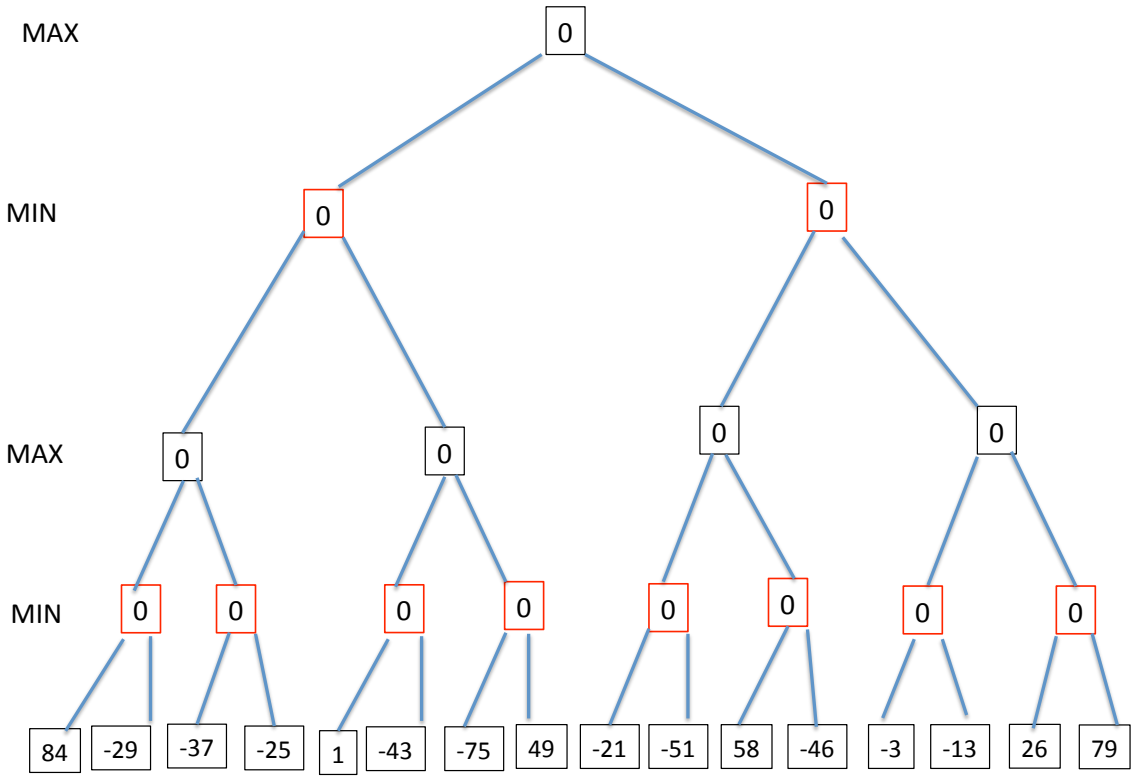
2-ply game:



Basic Process of Adversarial Search

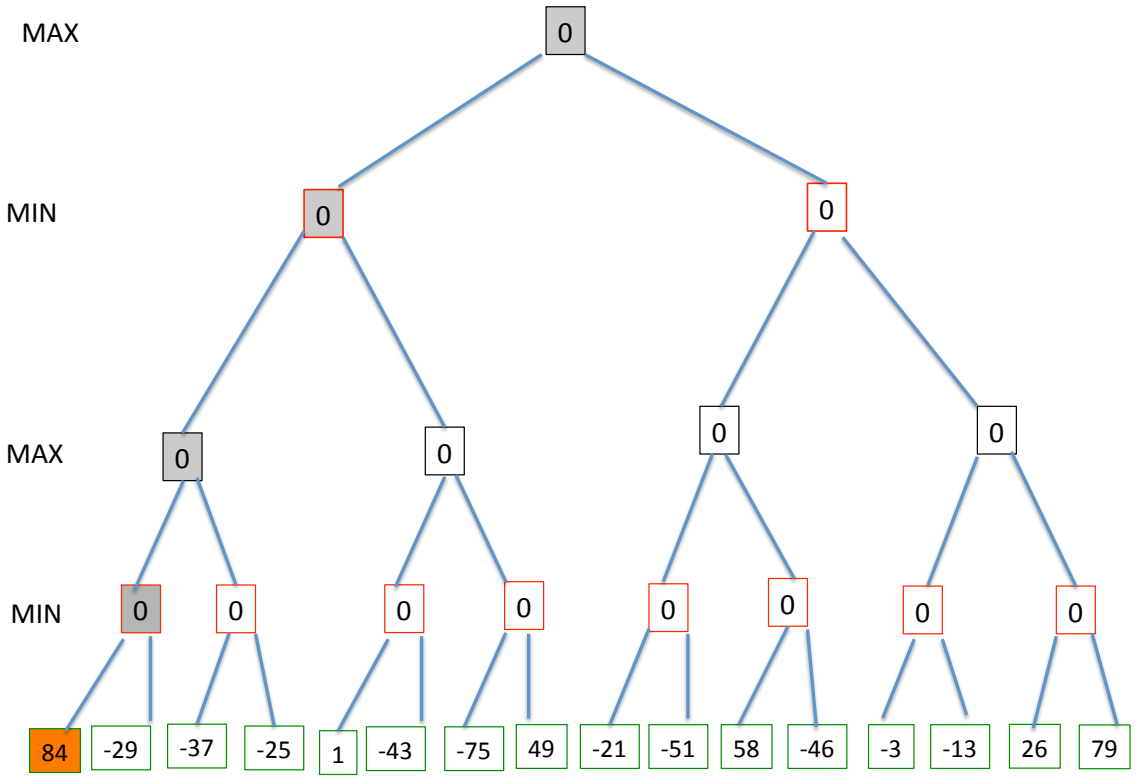
1. From the start state, generate a search tree in a depth-limited, depth-first manner, alternating between own and opponent's possible moves.
2. Only use heuristics/eval functions to evaluate the promise of the bottom-level nodes; then propagate values upwards, combining them using MAX and MIN operators.
3. Return to the root and choose the action, A_{best}, leading to the highest-rated child state, S_{best}.
4. Apply A_{best} to the current game state, producing S_{best}.
5. Wait for the opponent to choose an action, which then produces the new game state, S_{new}.
6. Generate a **whole new** search tree, with root state = S_{new}.
7. GO TO step 2.

Minimax example



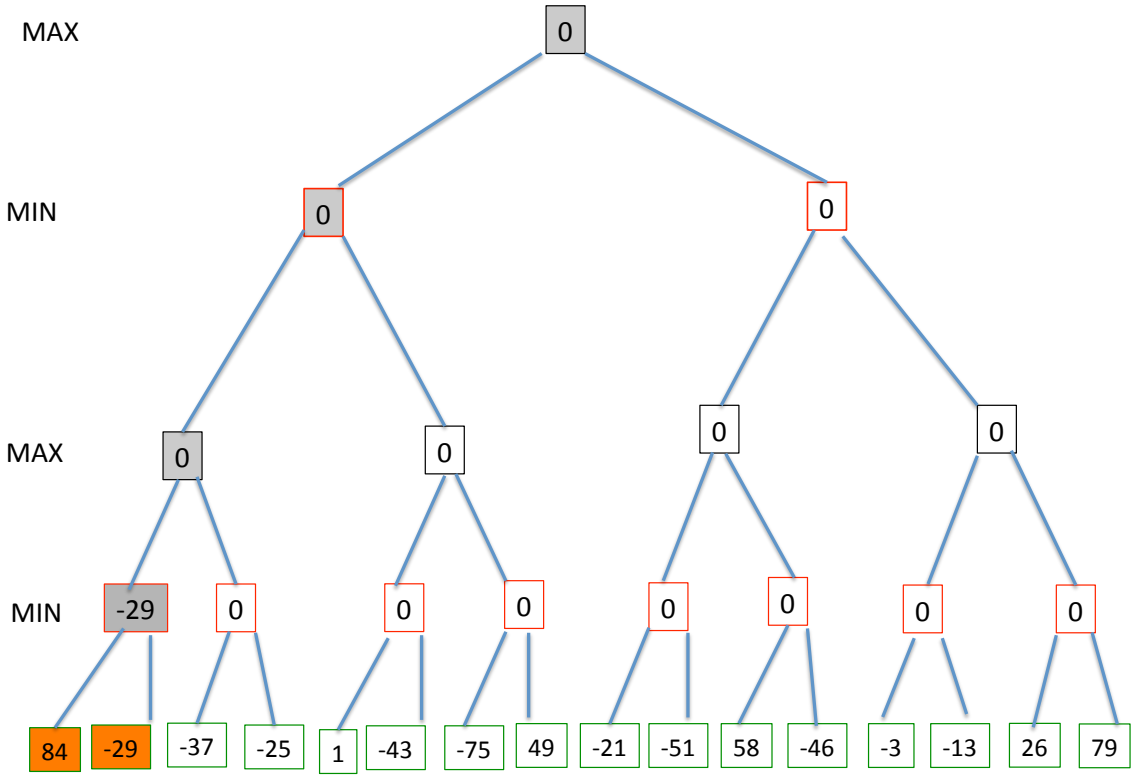
© Patrick Winston

Minimax example



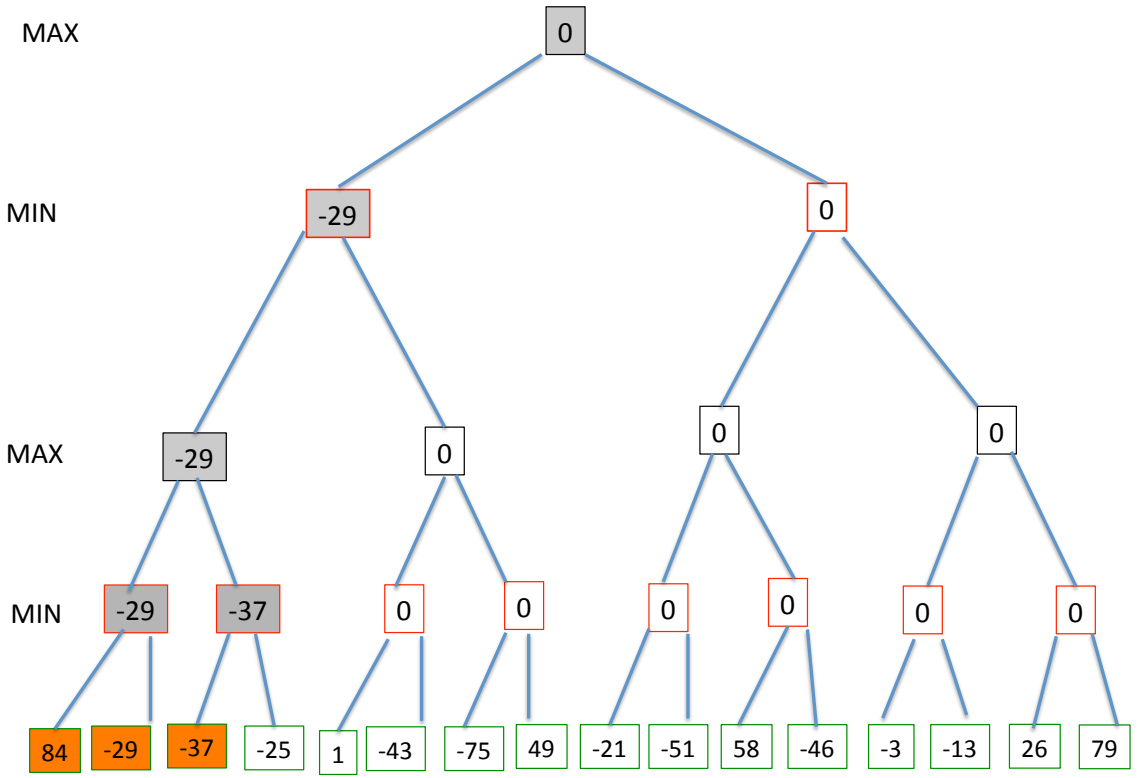
© Patrick Winston

Minimax example



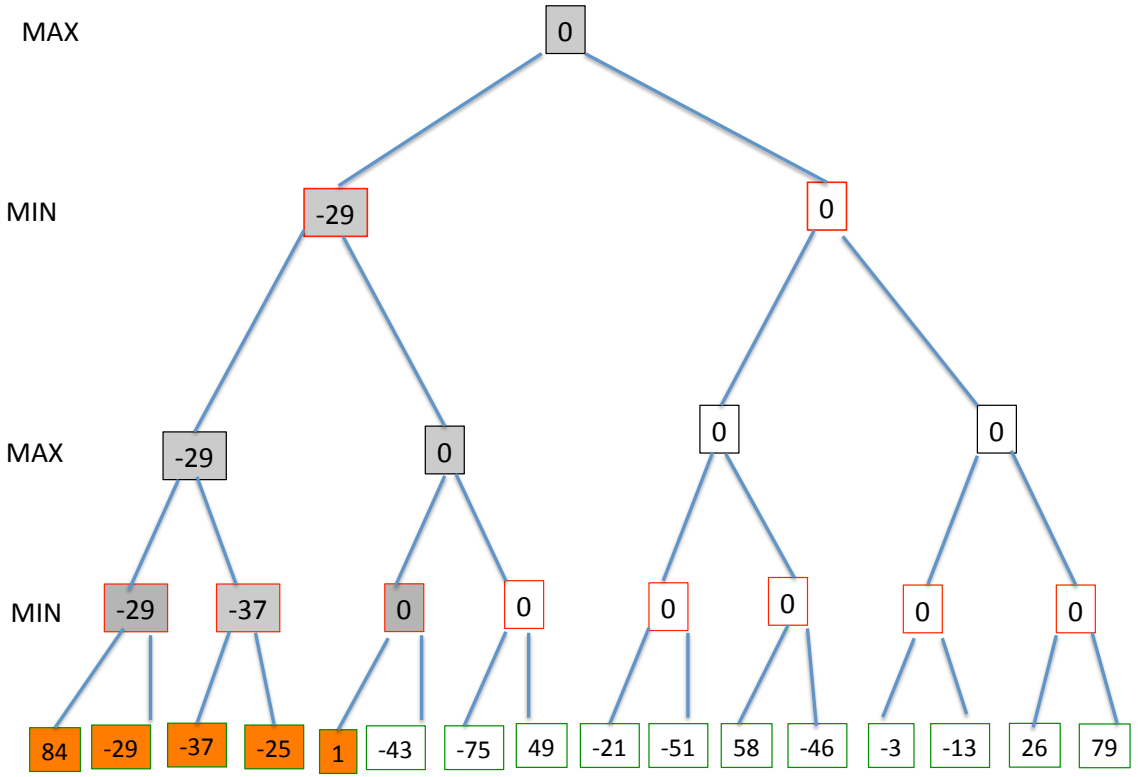
© Patrick Winston

Minimax example



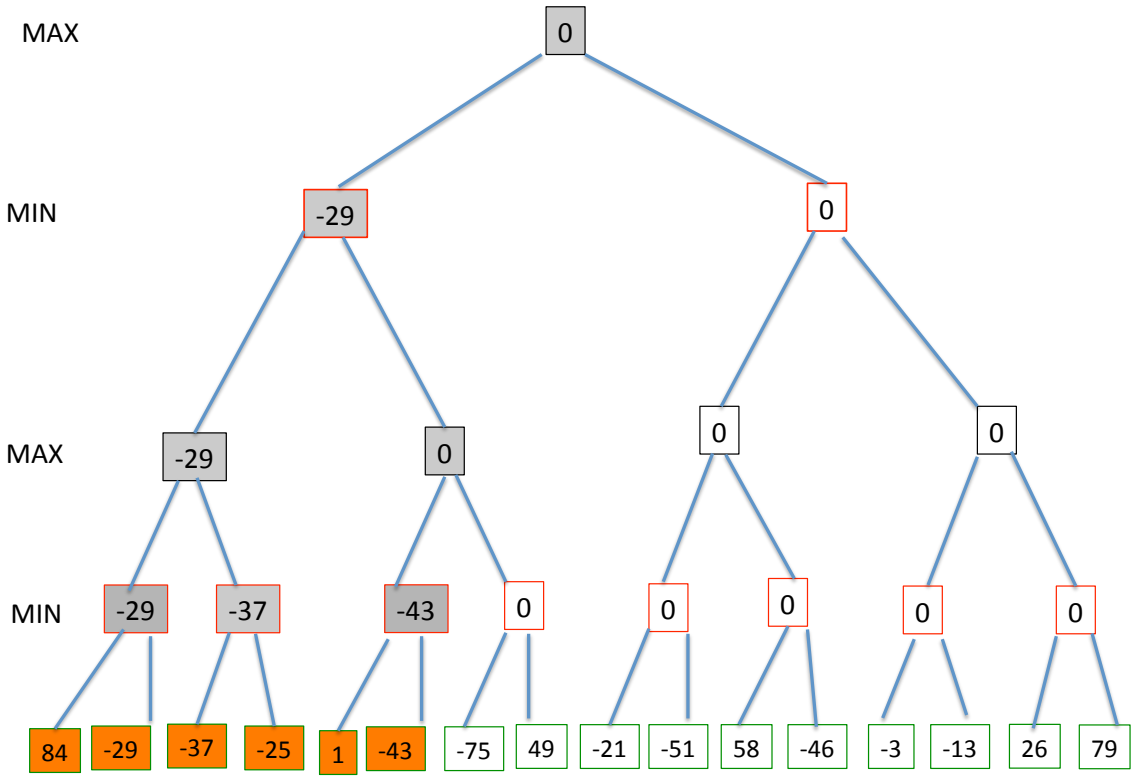
© Patrick Winston

Minimax example



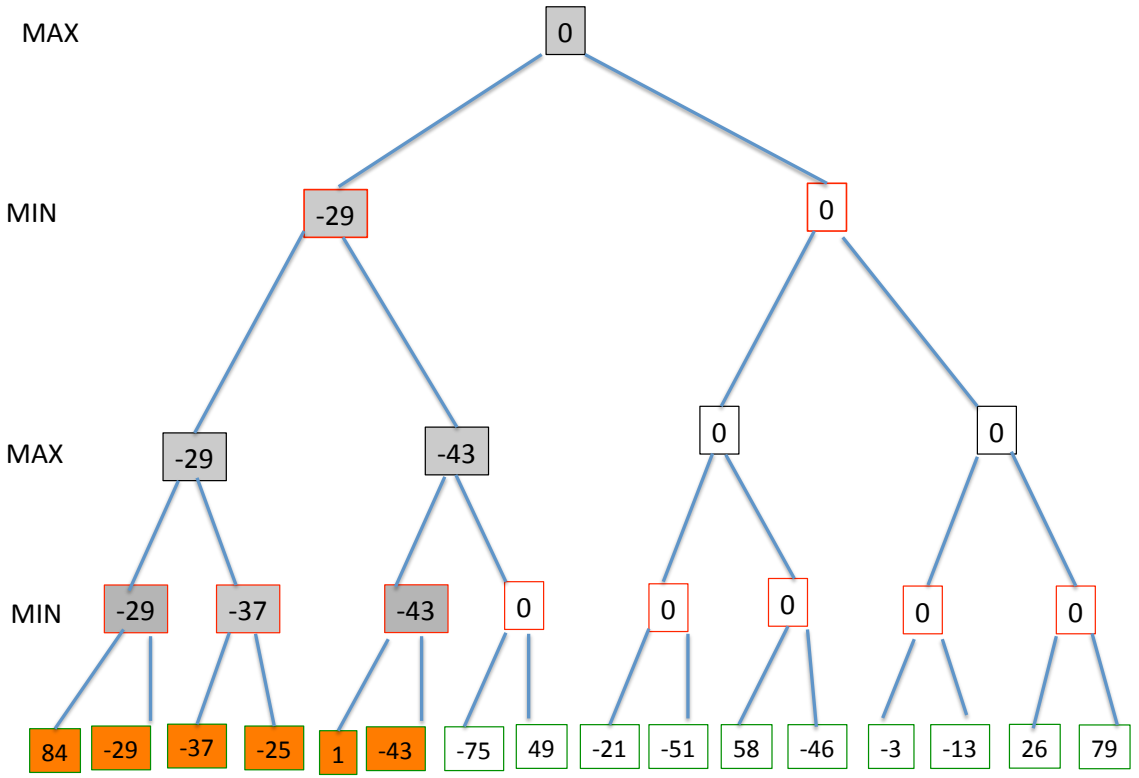
© Patrick Winston

Minimax example



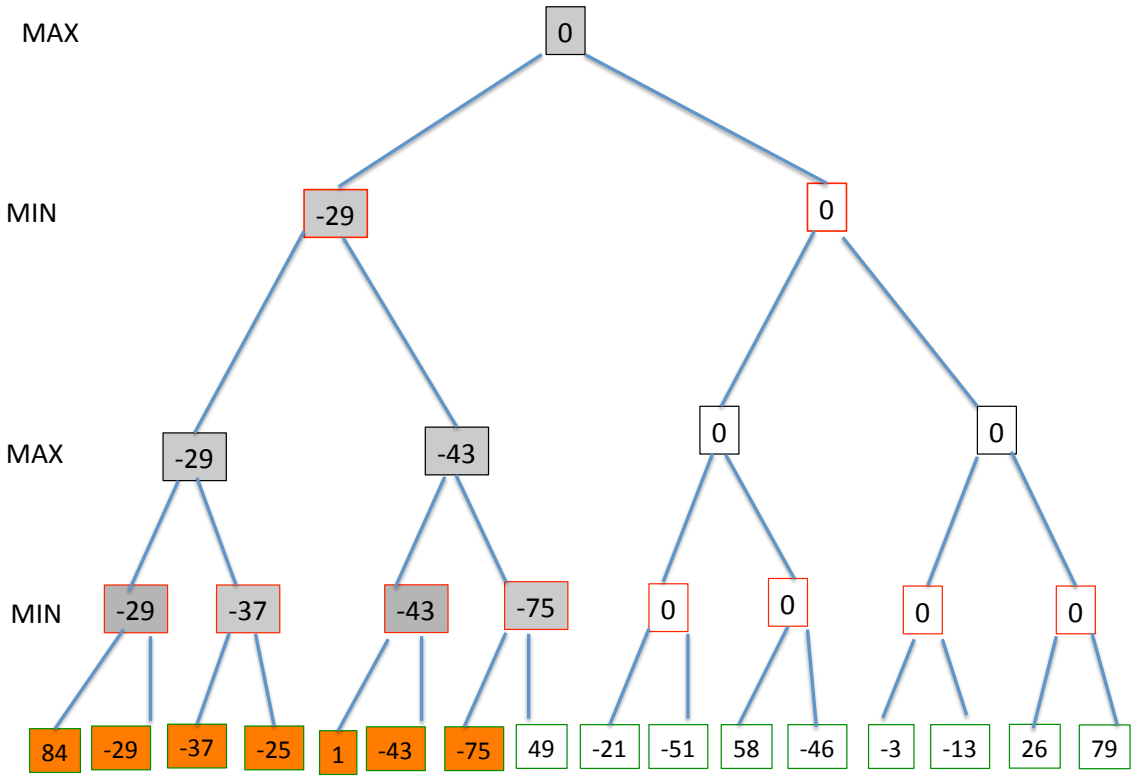
© Patrick Winston

Minimax example



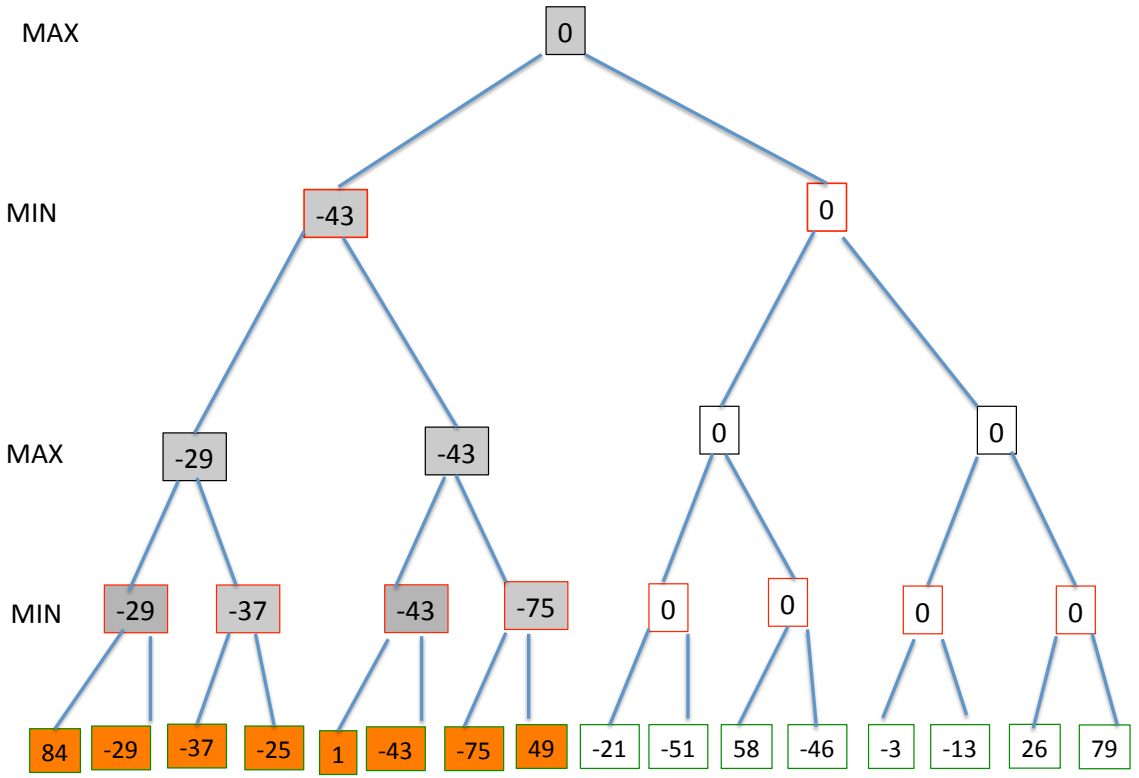
© Patrick Winston

Minimax example



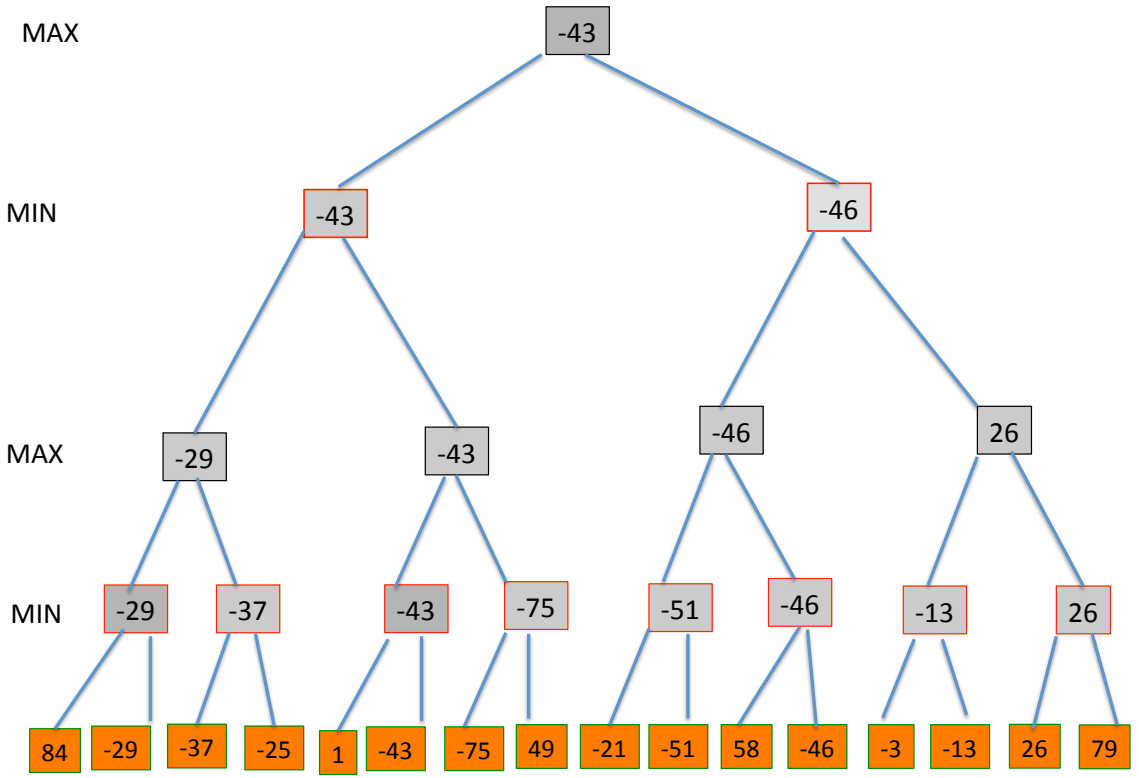
© Patrick Winston

Minimax example



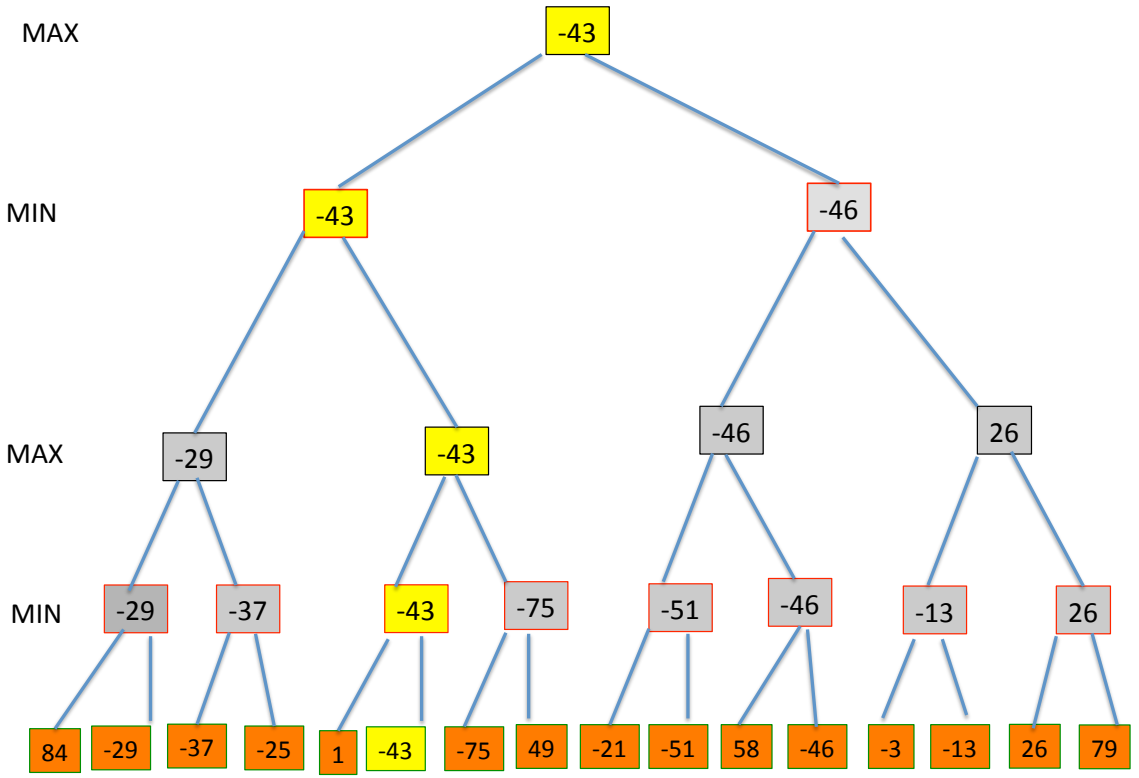
© Patrick Winston

Minimax example



© Patrick Winston

Minimax example



© Patrick Winston

Minimax algorithm

Adversarial analogue of DFS

```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game

    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))
```

```
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
```

Properties of minimax

Complete?? Only if tree is finite

Optimal?? Yes, against an optimal opponent.

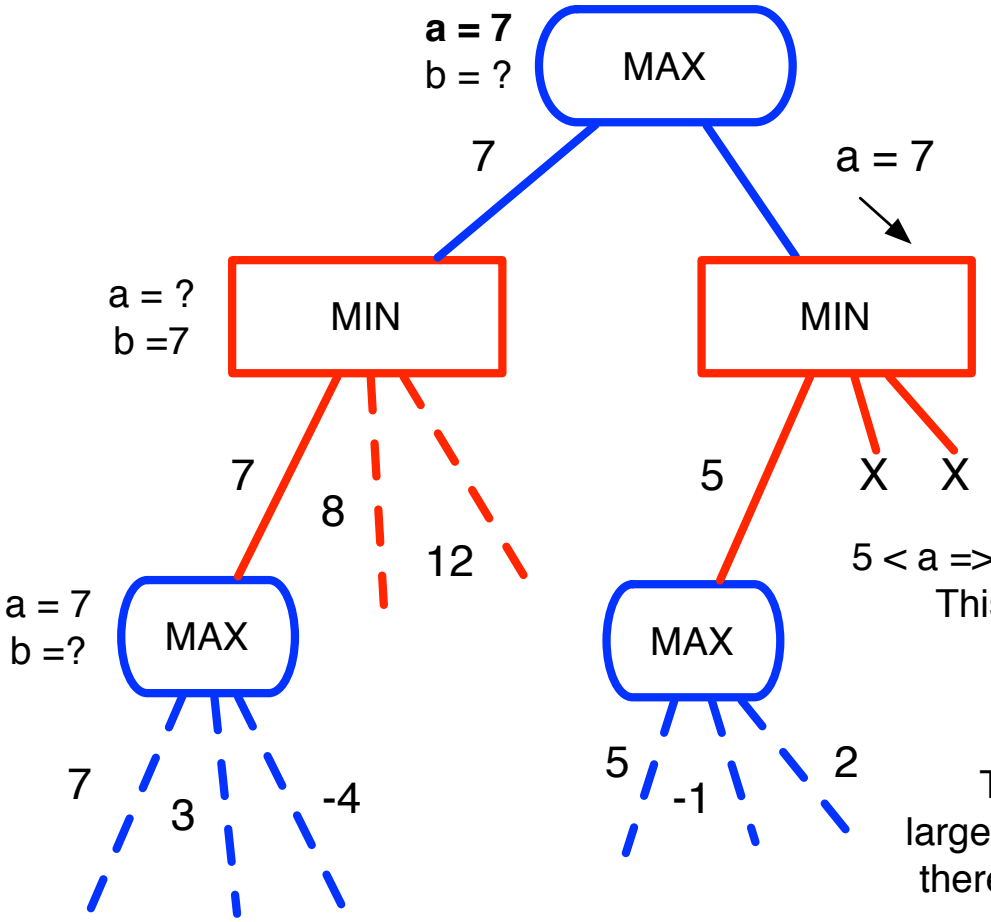
Time complexity?? $O(b^m)$

Space complexity?? $O(bm)$ (depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
$\Rightarrow$ exact solution completely infeasible

But do we need to explore every path?

Alpha-Beta Pruning



5 < a => Stop generating children. This node's value cannot win at the root. Prune it!

There could be many large subtrees down here, but there is no need to generate them, since they cannot affect the choice made at the root.

Alpha and Beta

Alpha

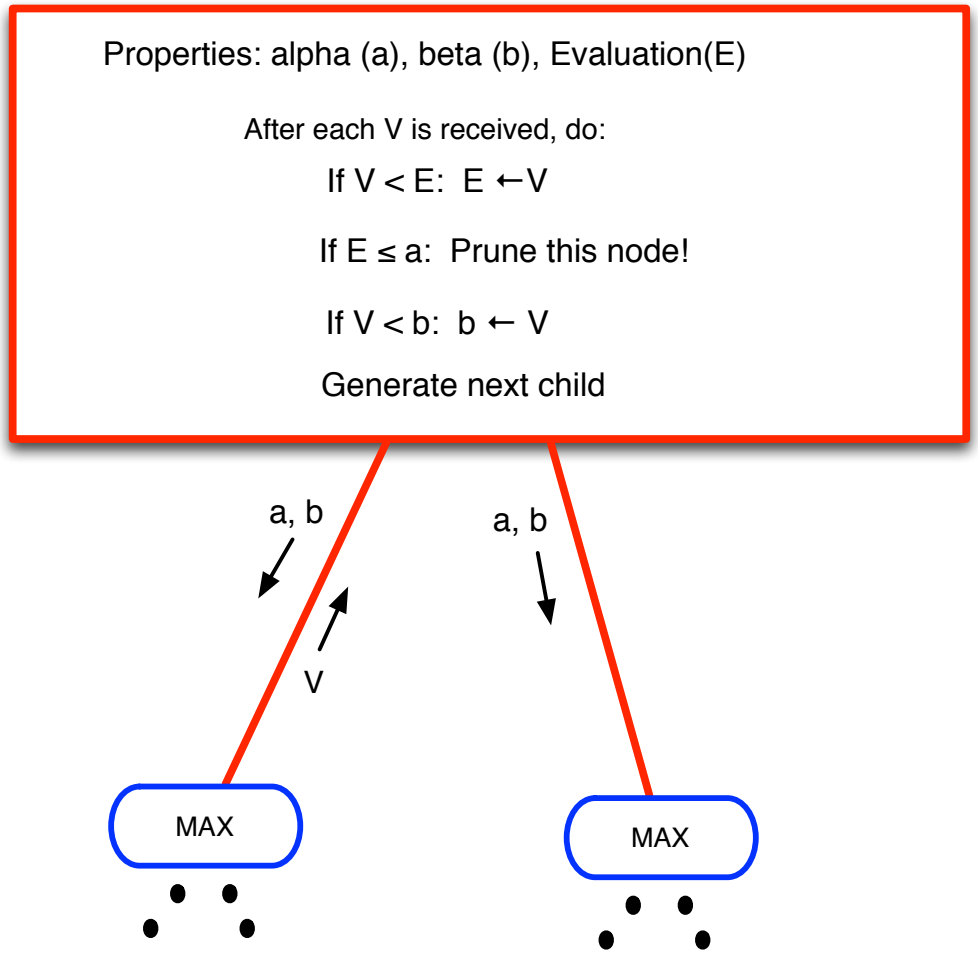
- A modifiable property of a MAX node.
- Indicates the lower bound on the evaluation of that MAX node.
- Updated whenever a LARGER evaluation is returned from a child MIN node.
- Used for pruning at MIN nodes.

Beta

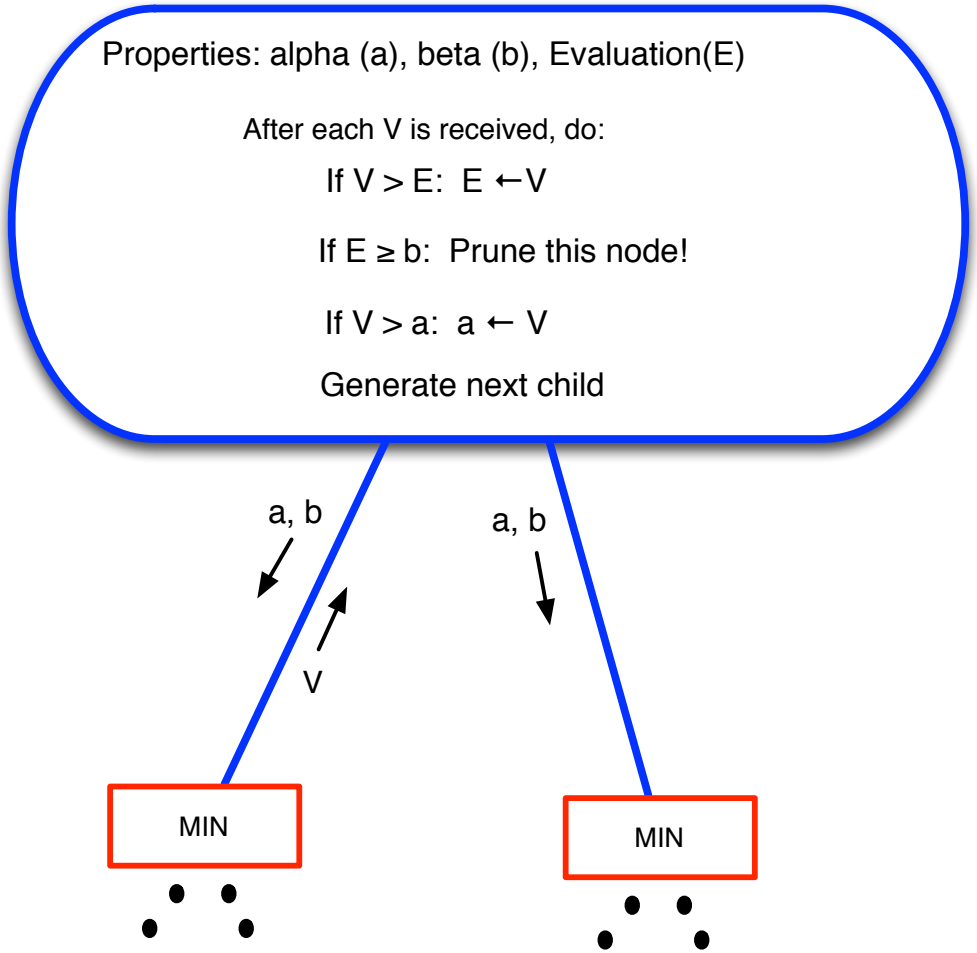
- A modifiable property of a MIN node.
- Indicates the upper bound on the evaluation of that MIN node.
- Updated whenever a SMALLER evaluation is returned from a child MAX node.
- Used for pruning at MAX nodes.

* Both alpha and beta are passed between MIN and MAX nodes.

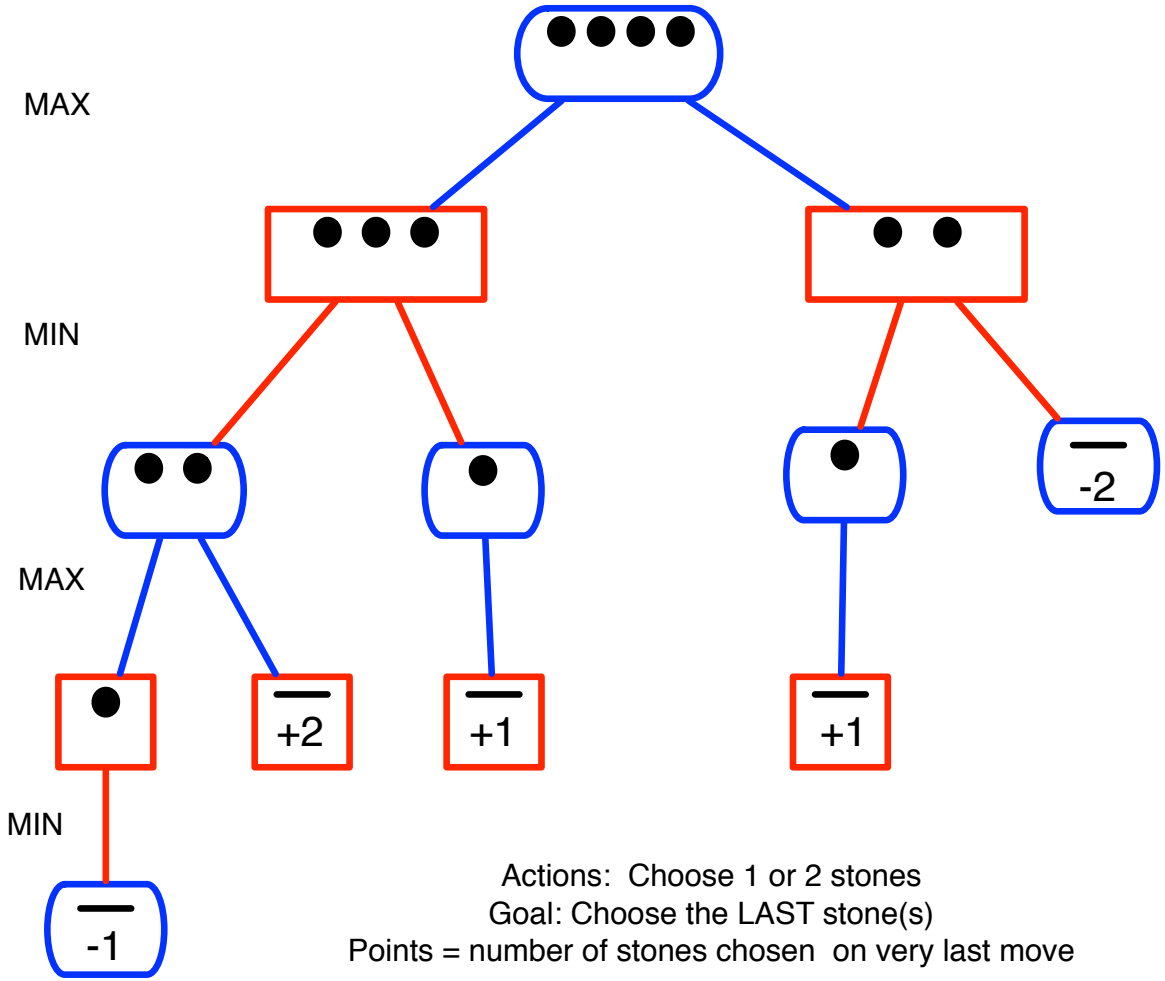
A Nanosecond in the Life of a MIN node



A Nanosecond in the Life of a MAX node

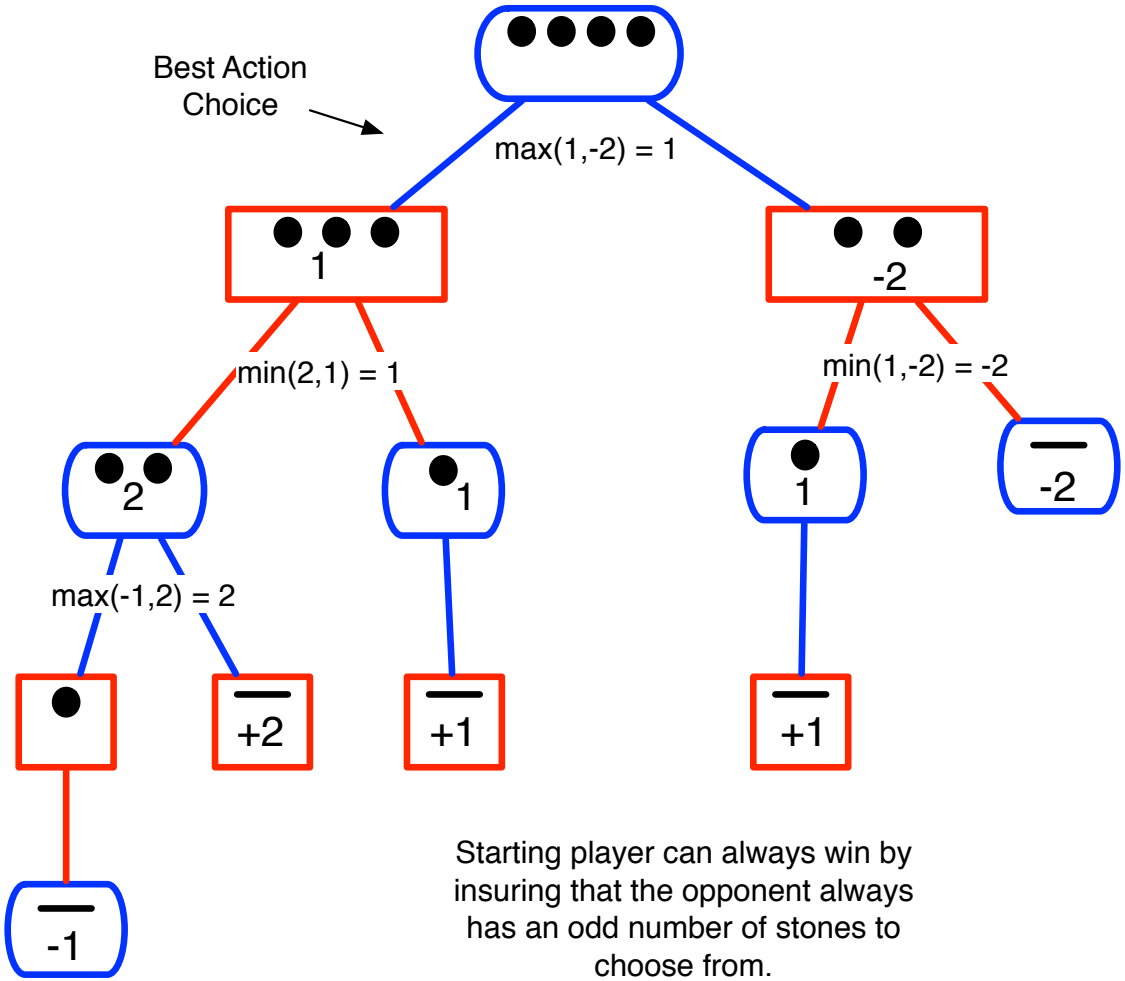


The Game of Nim



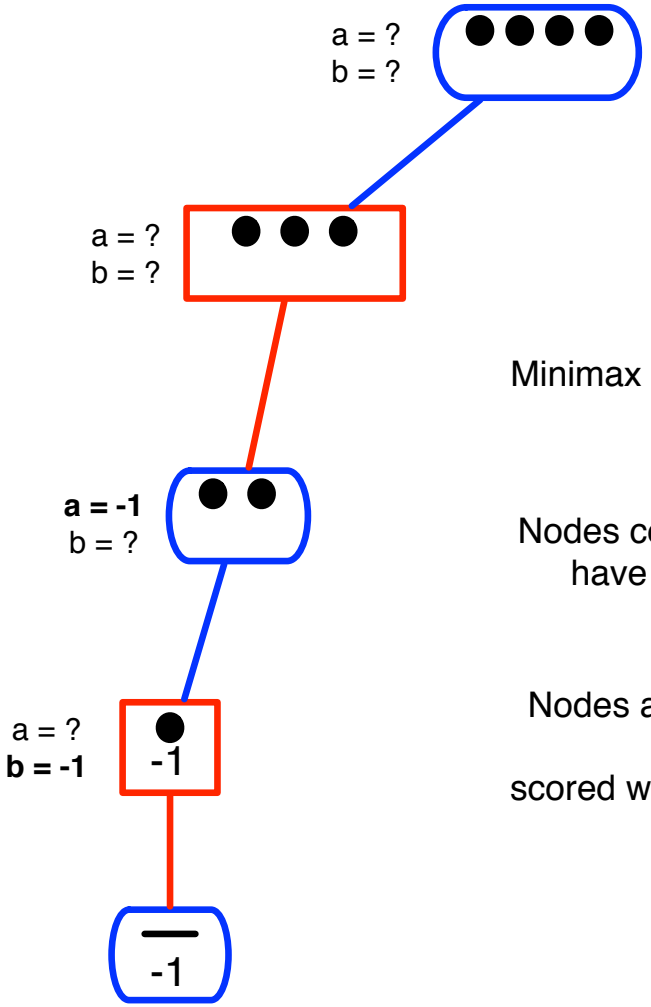
Actions: Choose 1 or 2 stones
Goal: Choose the LAST stone(s)
Points = number of stones chosen on very last move

Minimax Applied to Nim



Starting player can always win by insuring that the opponent always has an odd number of stones to choose from.

Minimax with Alpha-Beta Pruning

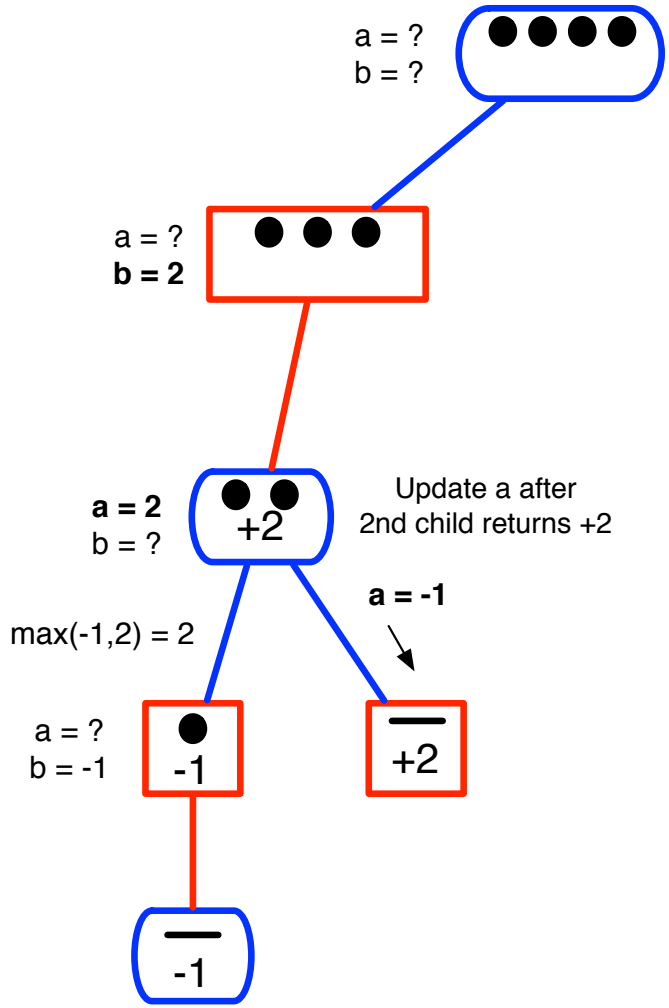


Minimax with alpha-beta pruning is depth-first.

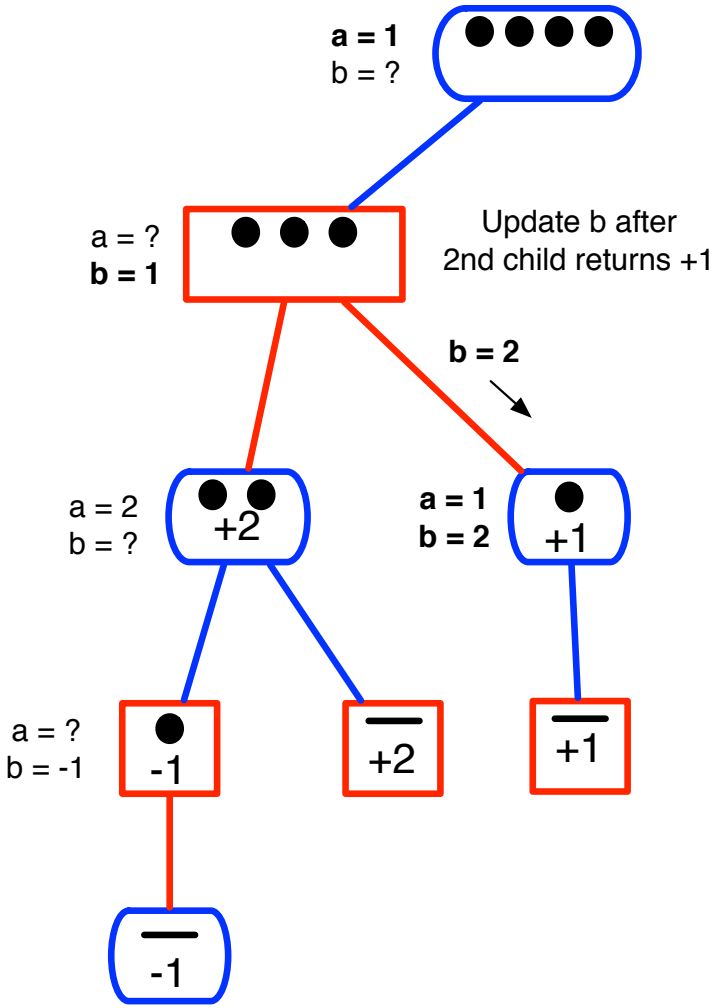
Nodes containing final states have concrete scores.

Nodes at max-depth that are not final states are scored with an evaluation function.

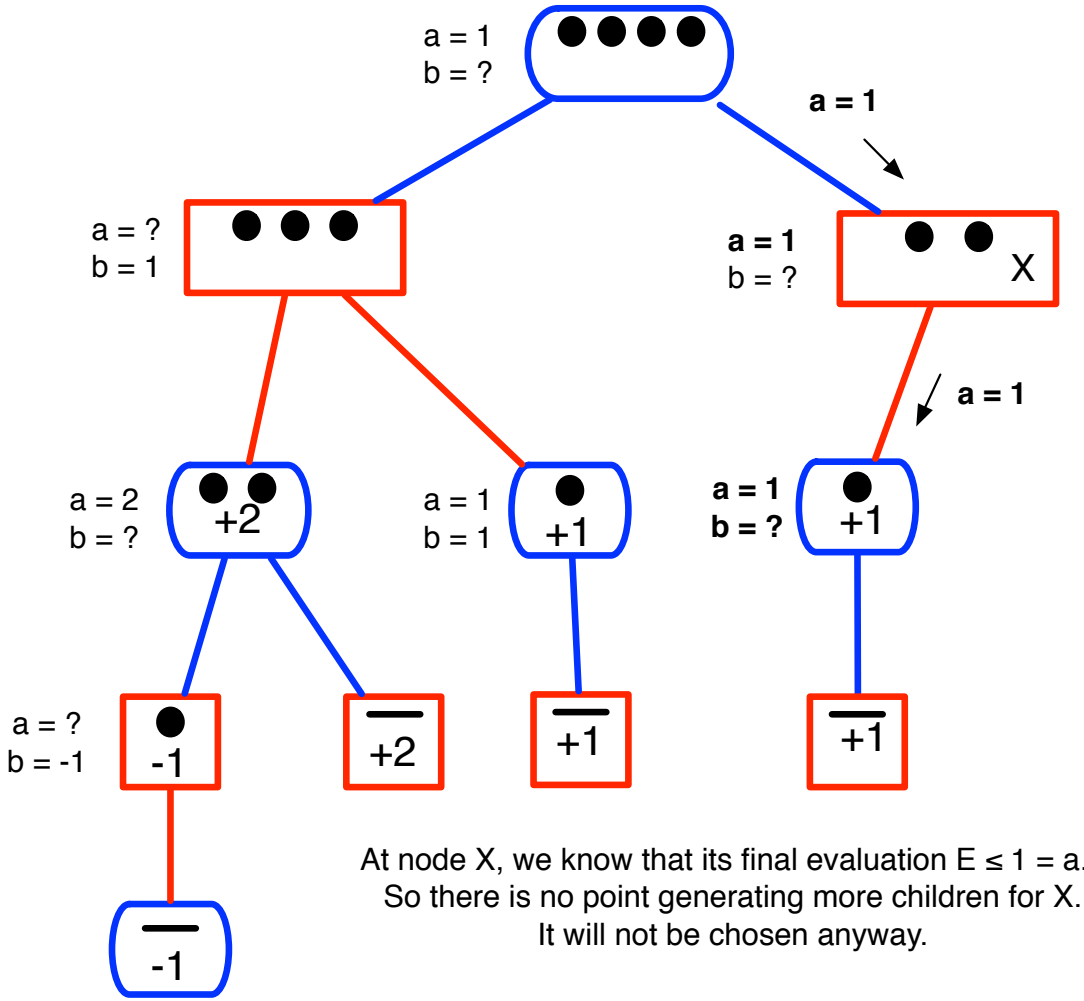
Minimax with Alpha-Beta Pruning



Minimax with Alpha-Beta Pruning



Minimax with Alpha-Beta Pruning



At node X, we know that its final evaluation E ≤ 1 = a.
So there is no point generating more children for X.
It will not be chosen anyway.

The α–β algorithm

function ALPHA-BETA-DECISION(*state*) **returns** an action
  return the *a* in ACTIONS(*state*) maximizing MIN-VALUE(RESULT(*a*, *state*))

---

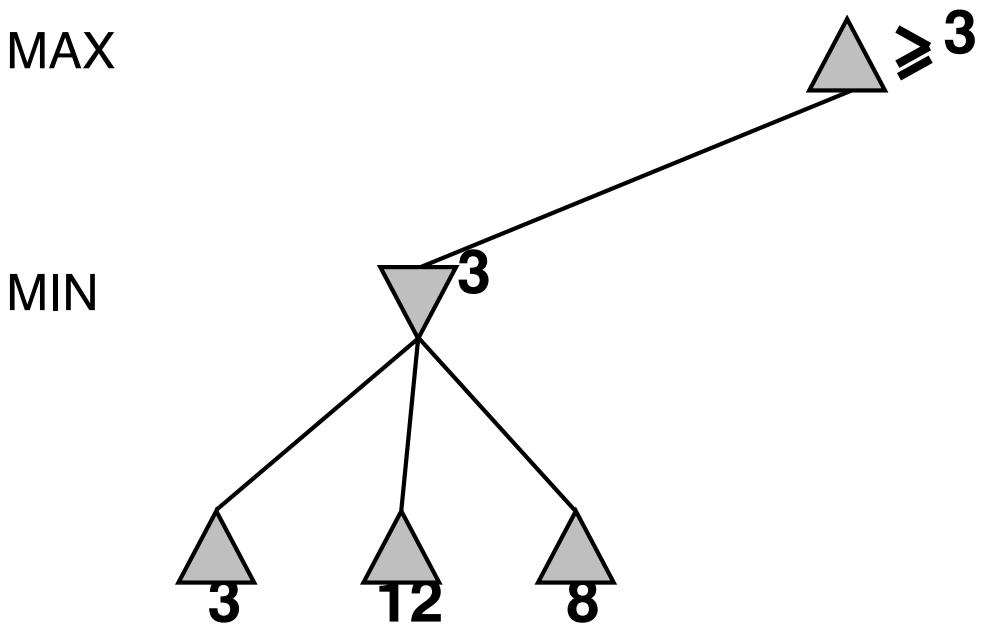
function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for *a*, *s* in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

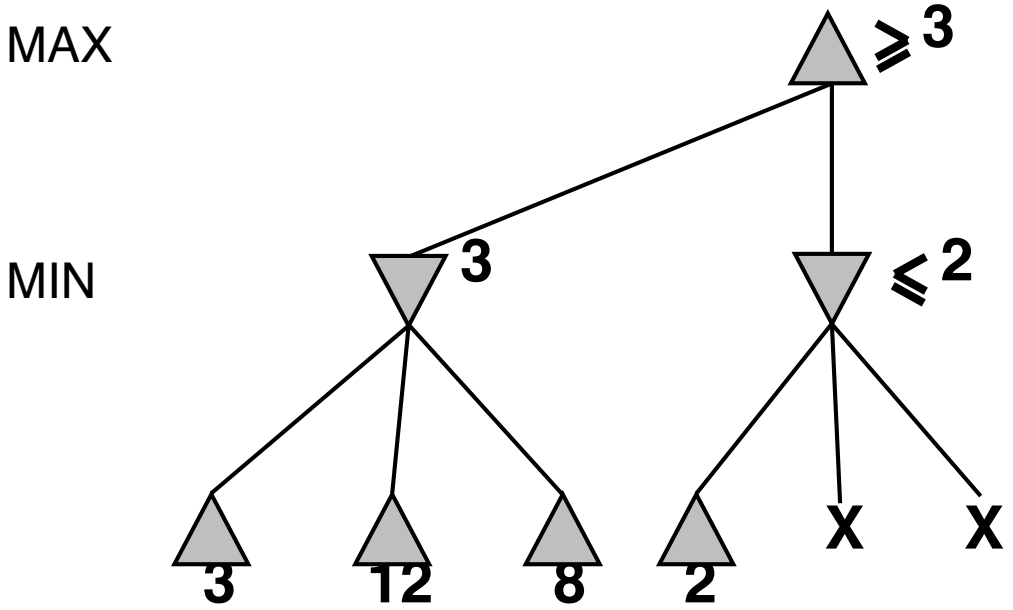
---

function MIN-VALUE(*state*, α, β) **returns** *a utility value*
  same as MAX-VALUE but with roles of α, β reversed

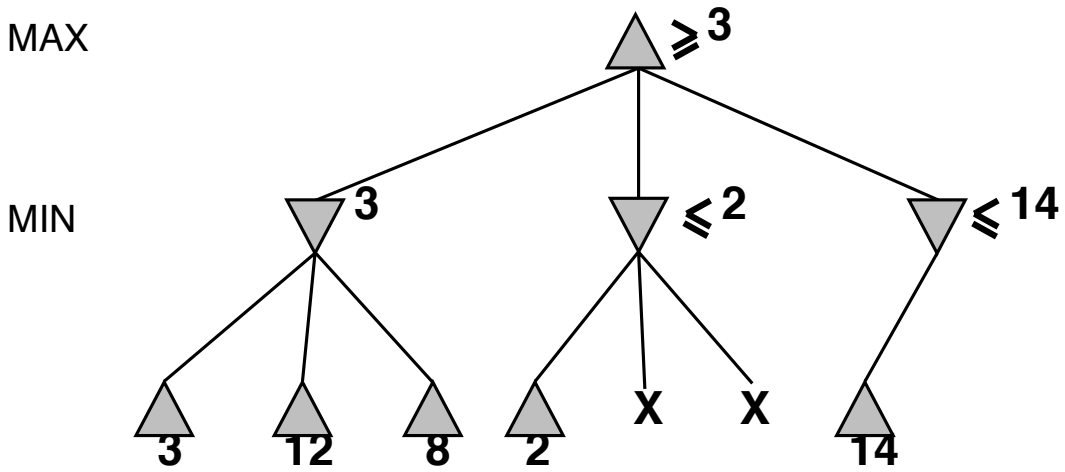
α–β pruning example



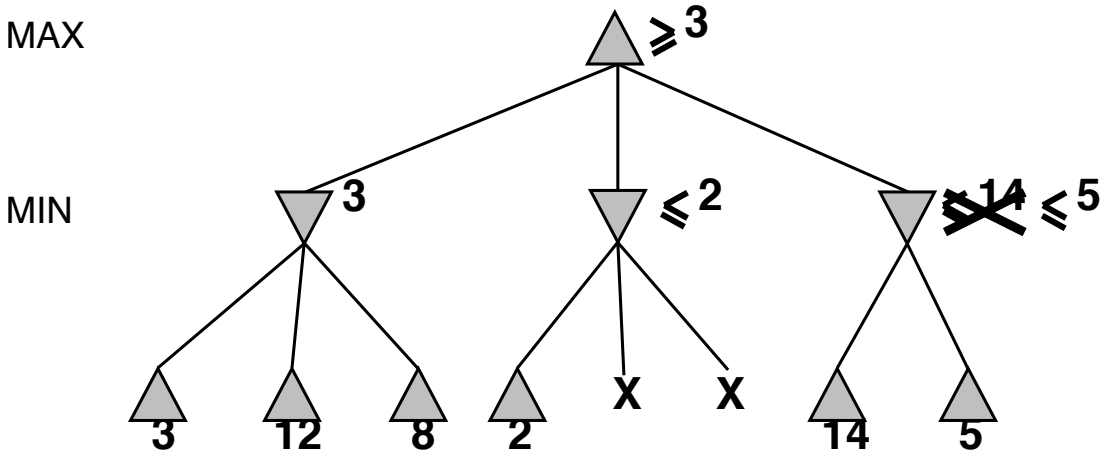
α–β pruning example



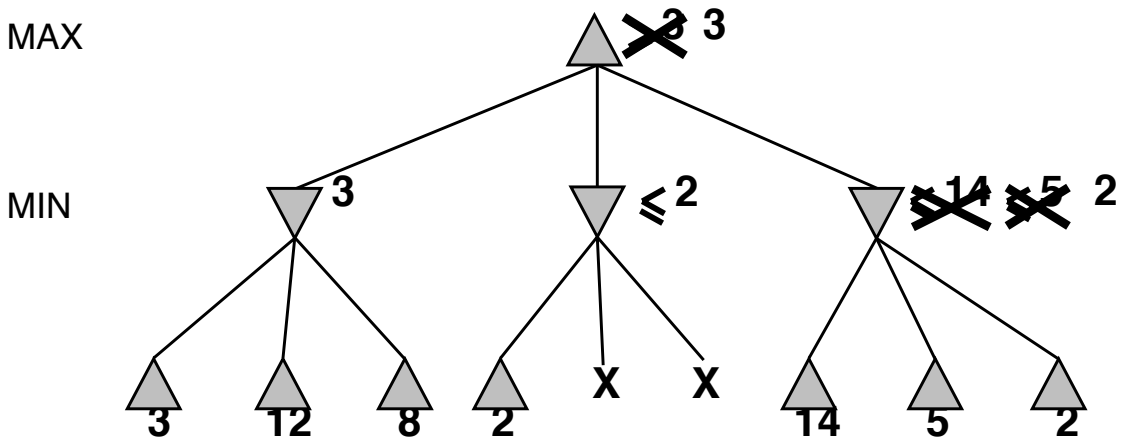
α–β pruning example



α–β pruning example



α–β pruning example



Properties of α–β

Pruning does not affect final result

Good move ordering improves effectiveness of pruning

With "perfect ordering," time complexity = $O(b^{m/2})$
$\Rightarrow$ doubles solvable depth

A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

Unfortunately, 35^{50} is still impossible!

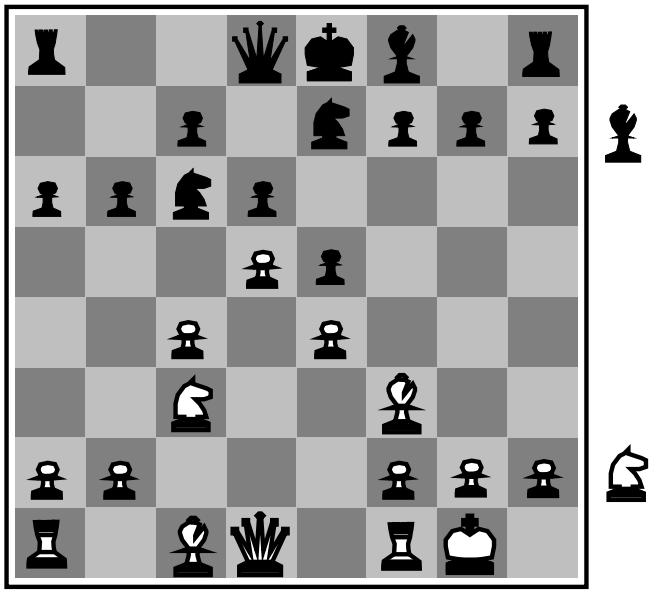
Resource limits

Standard approach:

- Use CUTOFF-TEST instead of TERMINAL-TEST
  e.g., depth limit (perhaps add quiescence search)
- Use EVAL instead of UTILITY
  i.e., evaluation function that estimates desirability of position

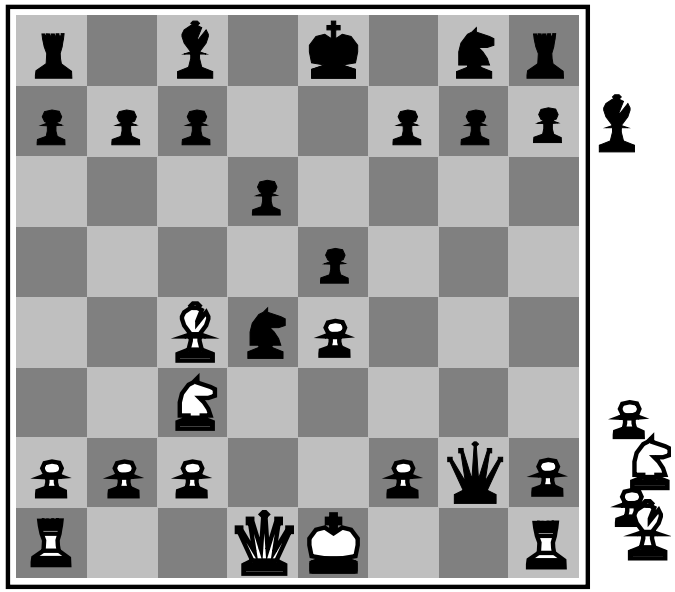
Suppose we have 180 seconds, explore 10^6 nodes/second
$\Rightarrow 18^8$ nodes per move $\approx 35^{10/2}$
$\Rightarrow \alpha$–β reaches depth 10 $\Rightarrow$ expert level chess program

Evaluation functions



Black to move

White slightly better



White to move

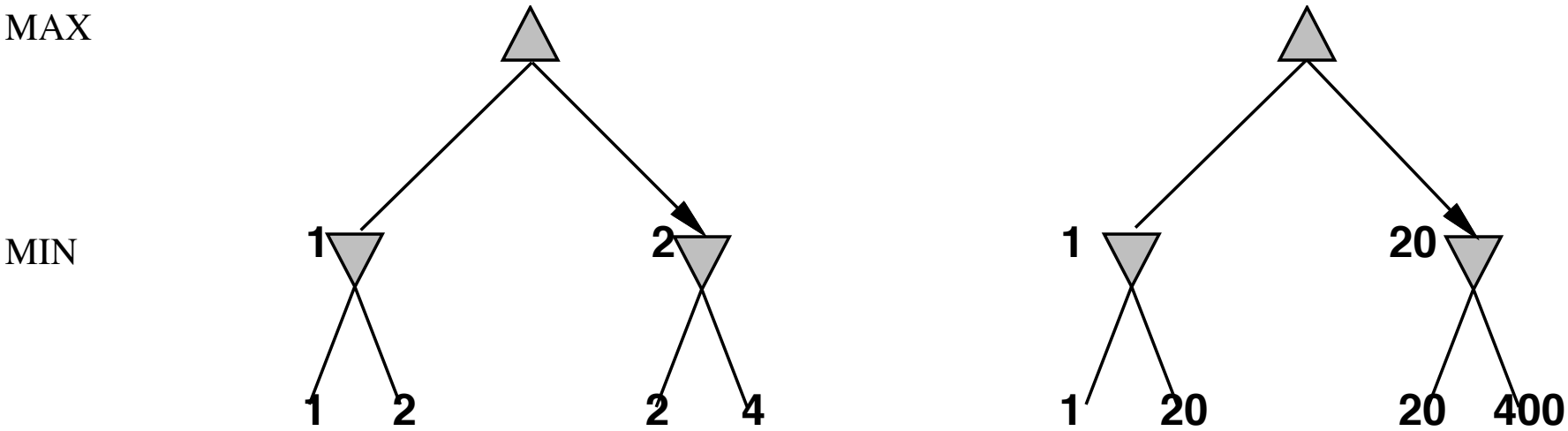
Black winning

For chess, typically linear weighted sum of features

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

e.g., $w_1 = 9$ with
$f_1(s) =$ (number of white queens) – (number of black queens), etc.

Digression: Exact values don't matter



Behaviour is preserved under any monotonic transformation of EVAL

Only the order matters:

payoff in deterministic games acts as an ordinal utility function

Deterministic games in practice

Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.

Deterministic games in practice

Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.

Chess: Deep Blue defeated human world champion Gary 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.

Deterministic games in practice

Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.

Chess: Deep Blue defeated human world champion Gary 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.

Deterministic games in practice

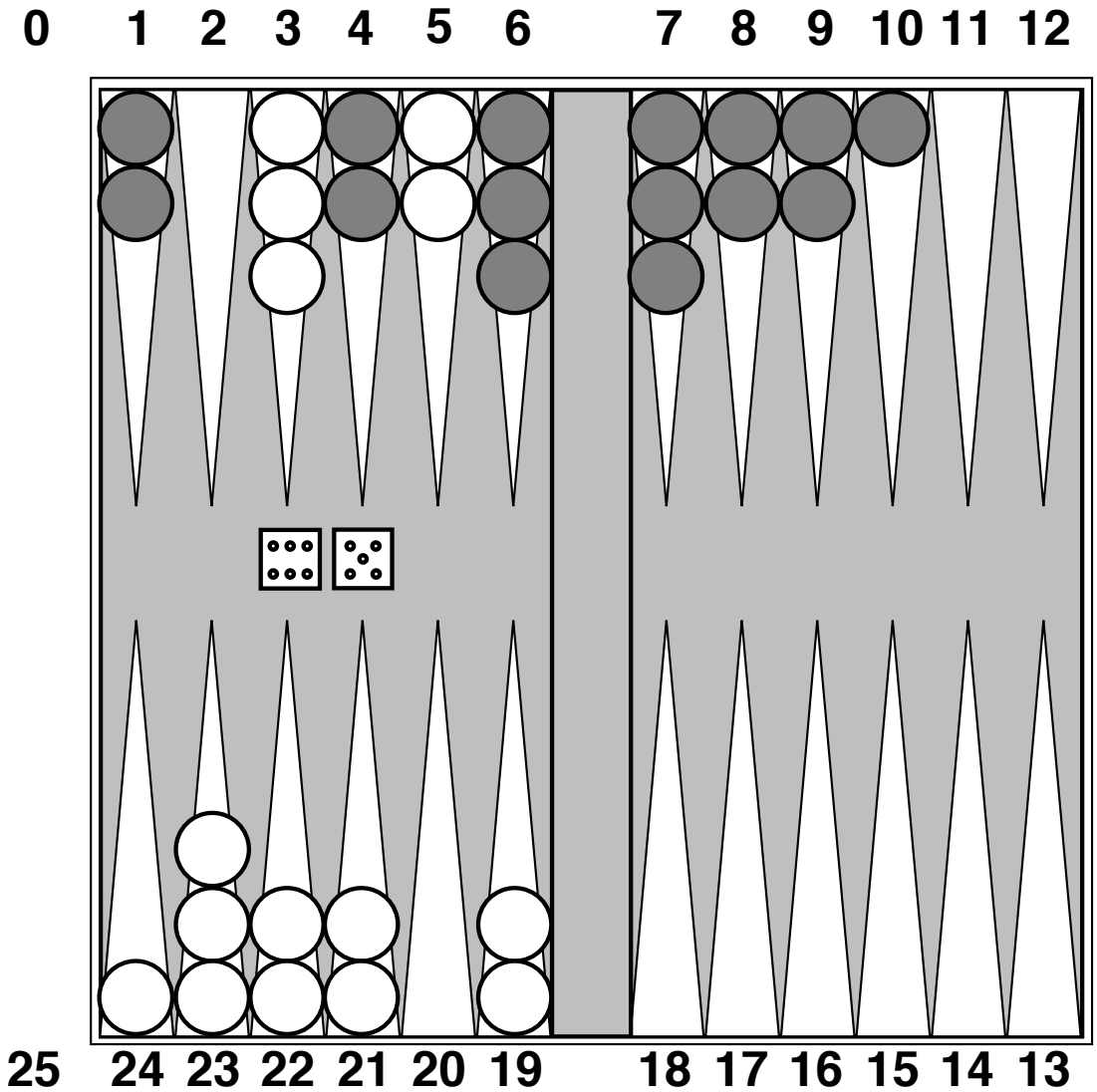
Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.

Chess: Deep Blue defeated human world champion Gary 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.

Go: human champions refuse to compete against computers, who are too bad. In Go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

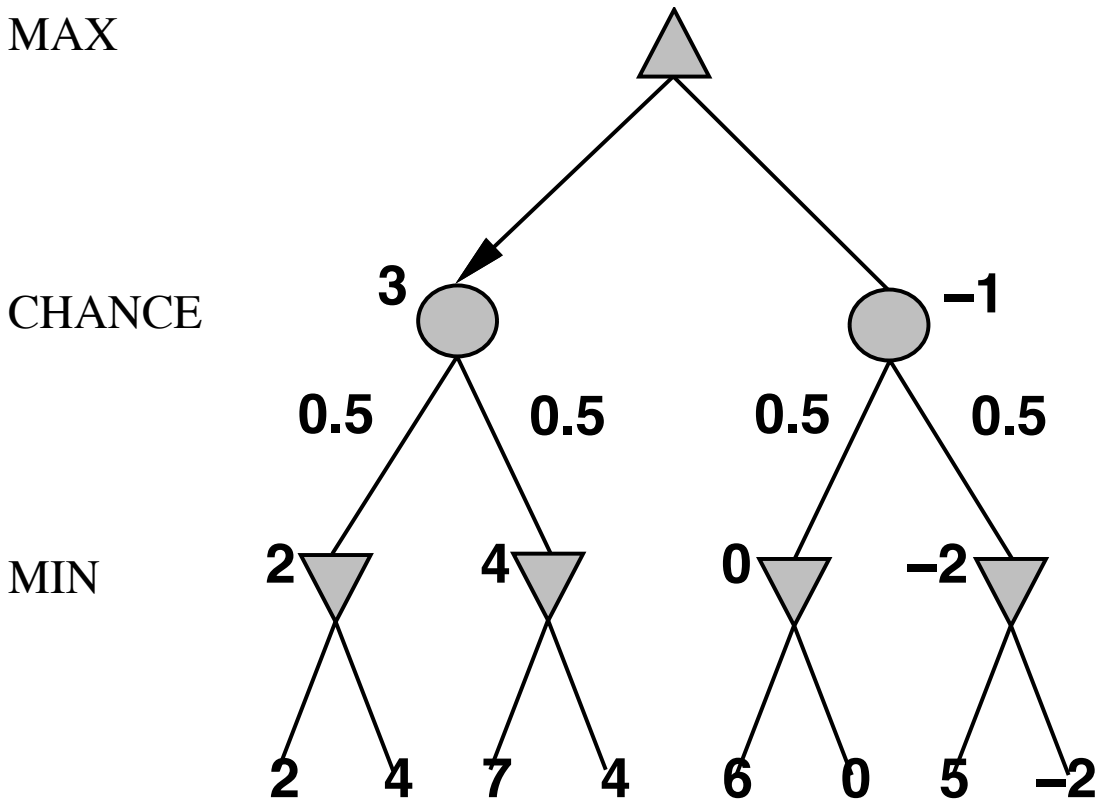
Nondeterministic games: backgammon



Nondeterministic games in general

In nondeterministic games, chance introduced by dice, card-shuffling

Simplified example with coin-flipping:



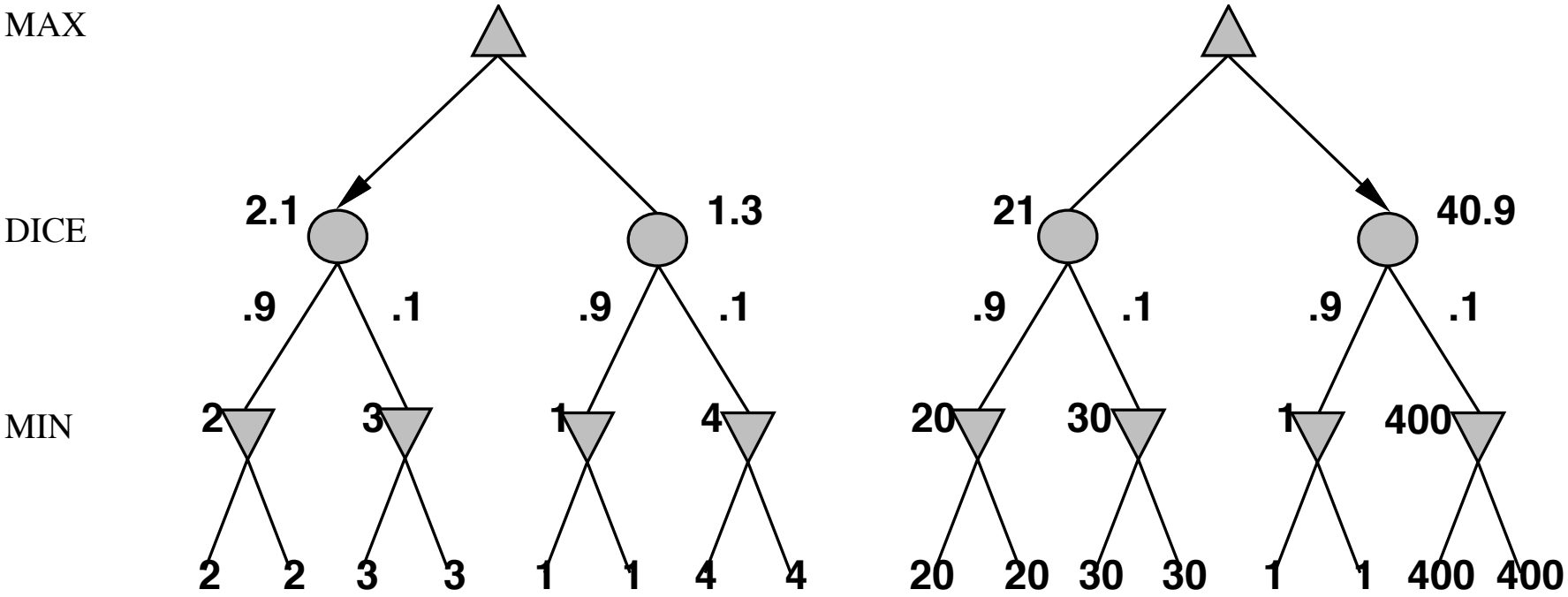
Algorithm for nondeterministic games

EXPECTIMINIMAX gives perfect play

Just like MINIMAX, except we must also handle chance nodes:

```
...
if state is a MAX node then
        return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a MIN node then
        return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a chance node then
        return sum of ..... EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
alternative: ...
```

Digression: Exact values DO matter



EVAL should be proportional to the expected payoff

Summary

Games are fun to work on! (and dangerous)

They illustrate several important points about AI

- ♢ perfection is unattainable ⇒ must approximate
- ♢ good idea to think about what to think about
- ♢ uncertainty constrains the assignment of values to states

Games are to AI as grand prix racing is to automobile design