Game playing I

Chapter 5.1-4

Adapted from slides kindly shared by Stuart Russell

Announcements

Grades for Python Tutorial (P0) will be up after I get some D2L questions answered

Glad to see folks digging into P1 (search). Due Thu 9-27 at 5pm

Hint: you'll probably want a simple Node class to manage your search tree

Wed: more pacman - bring your questions; talk about quiz next Monday

Put off Expectimax till next week

Project P2: Multi-Agent Pac-Man - coming soon!

Office hours for me on Thursday 2:30-3:30 and Friday 2:00-3:00, ECST 121

Outline

◇ Games

◇ Perfect play
  – minimax decisions
  – α–β pruning

◇ Resource limits and approximate evaluation

Games vs. search problems

Specify a move in each state, rather than sequence of actions like in search

"Unpredictable" opponent $\Rightarrow$ solution is a strategy (policy) specifying a move for every possible opponent reply

Policy maps states to actions

Time limits $\Rightarrow$ unlikely to find goal, must approximate

Types of games

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

Zero-sum vs room for cooperation

Deterministic, zero-sum, perfect info

- Know the rules
- Know what actions do
- Zero-sum: Gain for one player is loss for other (really "constant sum")
- One player maximizes result, the other minimizes result

This is the focus for today

Game Playing State-of-the-Art

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: Until recently: human champions refused to compete against computers, who are too bad. In go, $b > 300$. A classic challenge for AI since long ago.

Go

Great game: handicaps!

Very hard for AI

Big board

Many moves possible

Difficult evaluation function

But in the last few years, best go program (Zen) became better than perhaps 99% of online human players (6d on KGS Go Server).

Monte Carlo (randomized) search/evaluation.

Pacman

Unknown :)

Game vs Search Reinterpretation

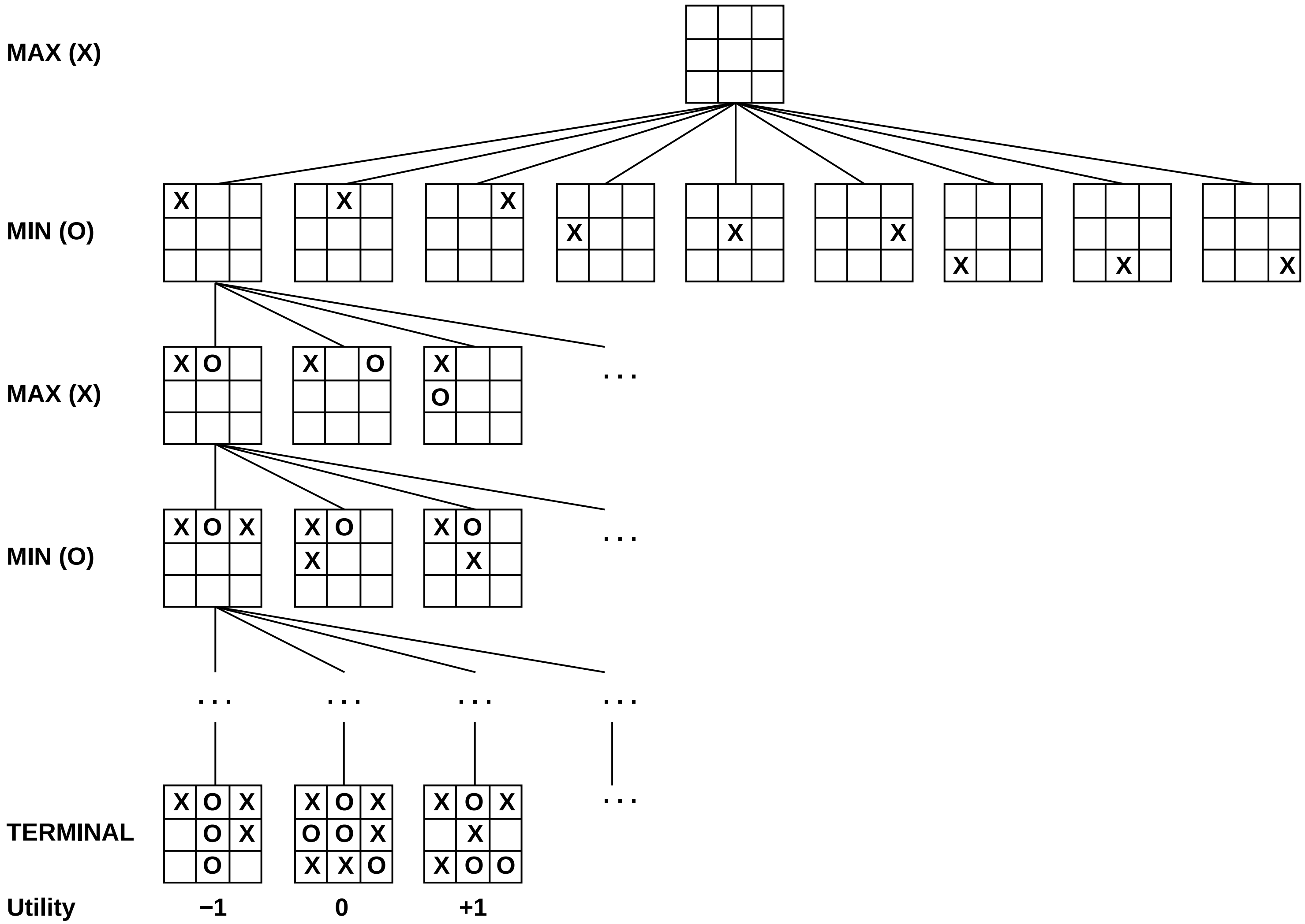
Each node stores a value: the best outcome it can reach

This is the maximal outcome of its children (the max value)

Note that we don't have path sums as before (utilities at end)

After search, can pick move that leads to best node

Game tree (2-player, deterministic, turns)

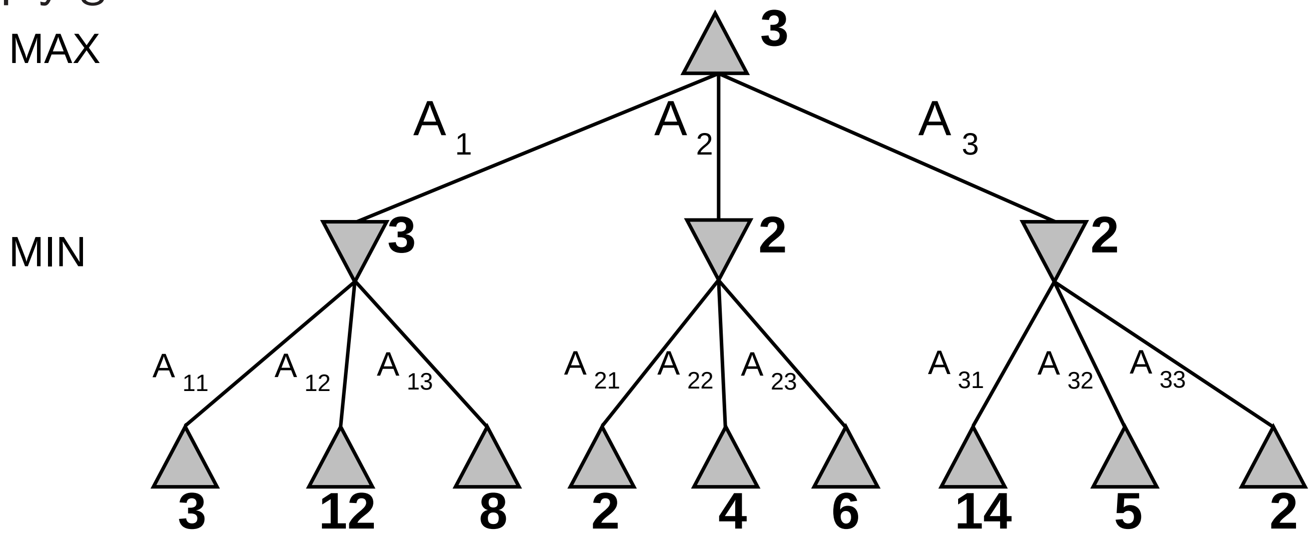


Minimax

Perfect play for deterministic, perfect-information games

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

E.g., 2-ply game:



Thanks to Dan Klein

Computing Minimax Values

- Two recursive functions:
  - max-value maxes the values of successors
  - min-value mins the values of successors

---

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 max = $-\infty$

  For each successor of state:

    Compute value(successor)

    Update max accordingly

  Return max

Properties of minimax

Complete??

Properties of minimax

Complete?? Only if tree is finite (chess has specific rules for this).
NB a finite strategy can exist even in an infinite tree!

Optimal??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)

Optimal?? Yes, against an optimal opponent. Otherwise??

Time complexity??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)

Optimal?? Yes, against an optimal opponent. Otherwise??

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

Space complexity??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)

Optimal?? Yes, against an optimal opponent. Otherwise??

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?

Resource limits

Standard approach:

- Use Cutoff-Test instead of Terminal-Test

  e.g., depth limit

- Use Eval instead of Utility

  i.e., evaluation function that estimates desirability of position

Iterative deepening search:

- Start with a shallow search, get a possible answer
- Keep searching deeper until you run out of time

Suppose we have 100 seconds, explore 10^4 nodes/second

$\Rightarrow$ 10^6 nodes per move $\approx 35^{8/2}$

$\Rightarrow$ α–β reaches depth 8 $\Rightarrow$ pretty good chess program

Iterative deepening search

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution
    inputs: problem, a problem

    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
    end
```

Iterative deepening search $l = 0$

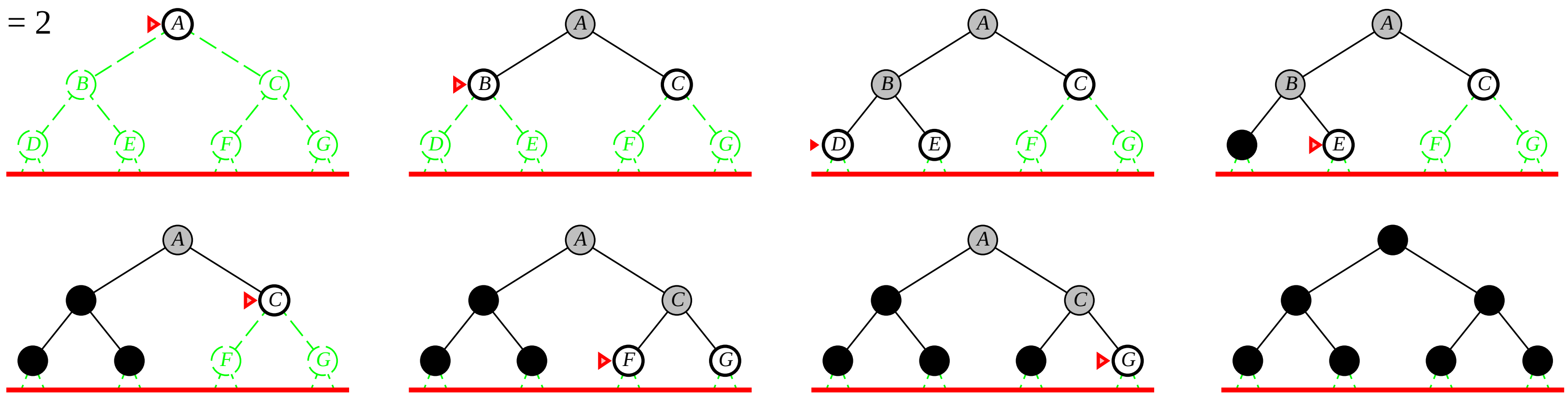
Limit = 0

Iterative deepening search $l = 1$

Limit = 1

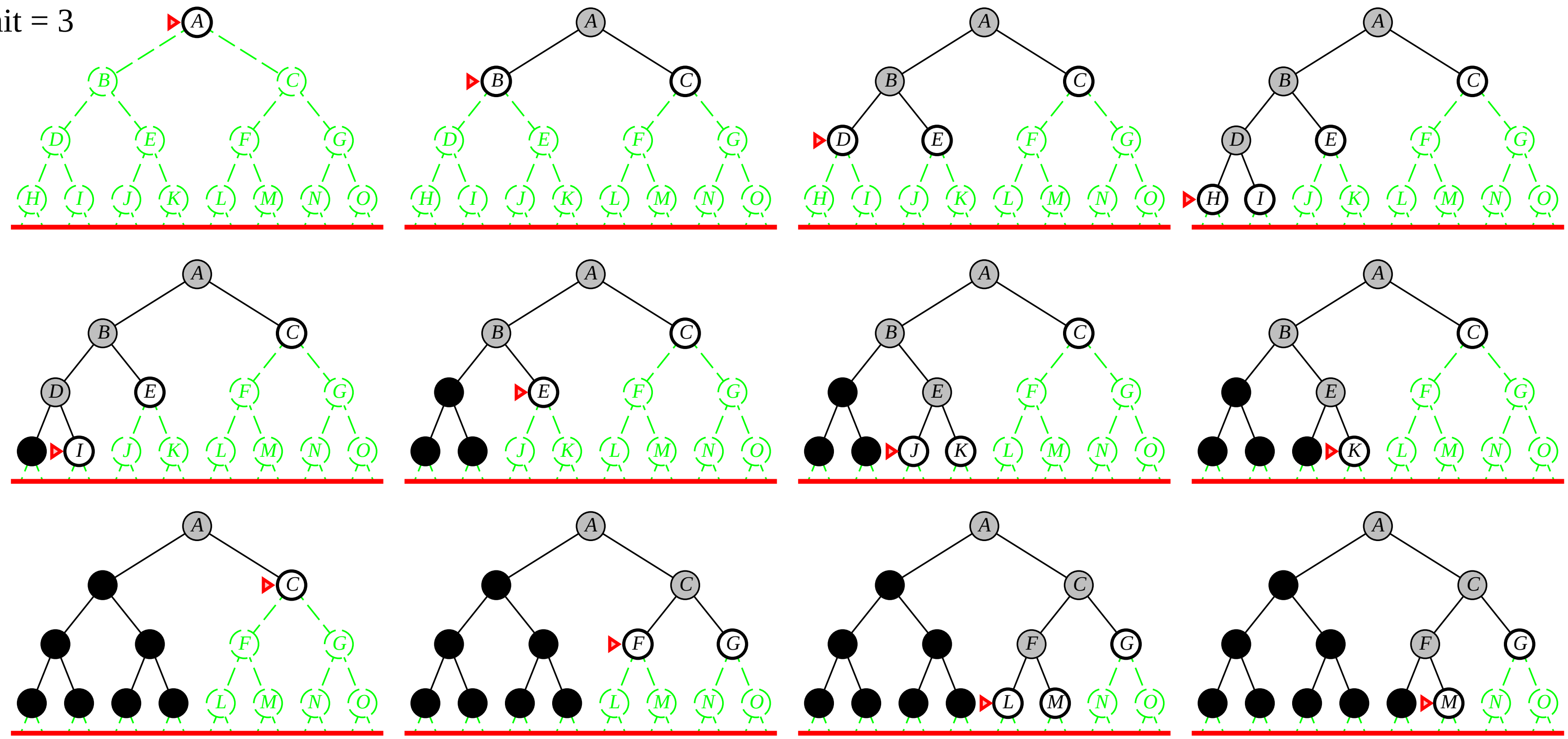
Iterative deepening search $l = 2$

Limit = 2

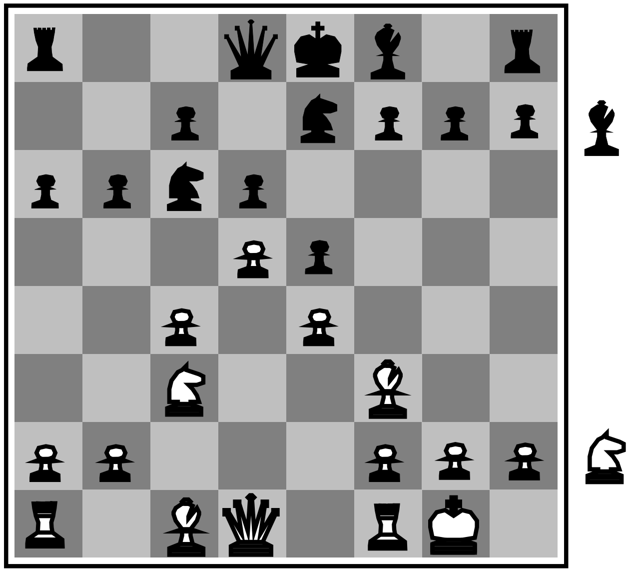


Iterative deepening search $l = 3$

Limit = 3

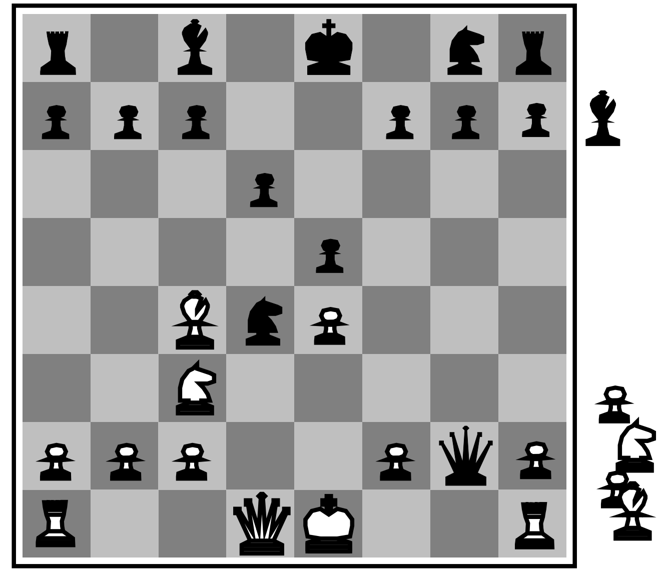


Evaluation functions



Black to move

White slightly better



White to move

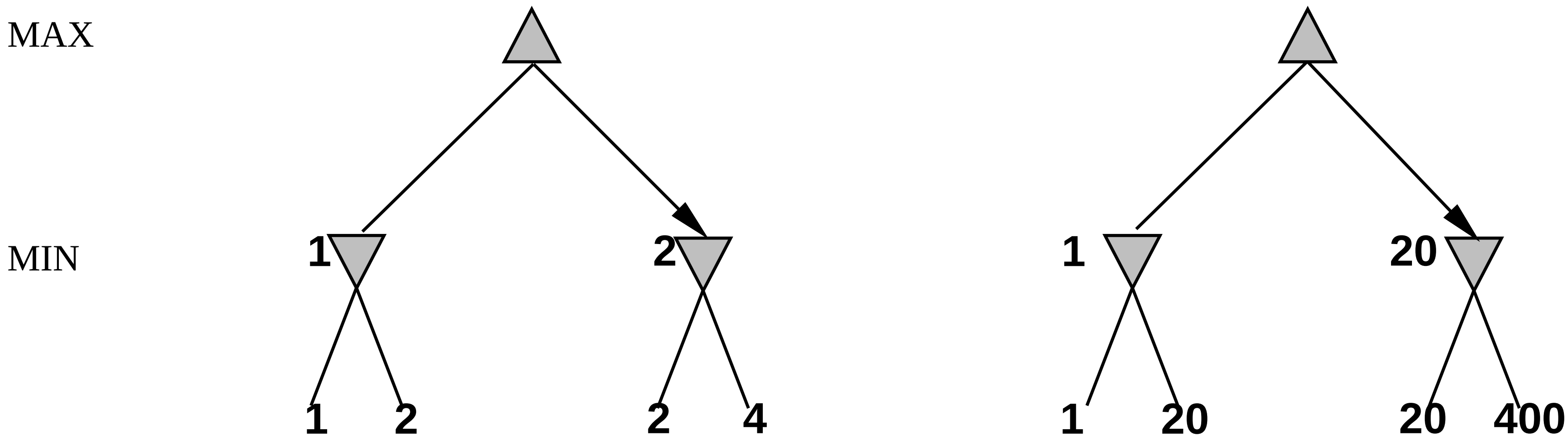
Black winning

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

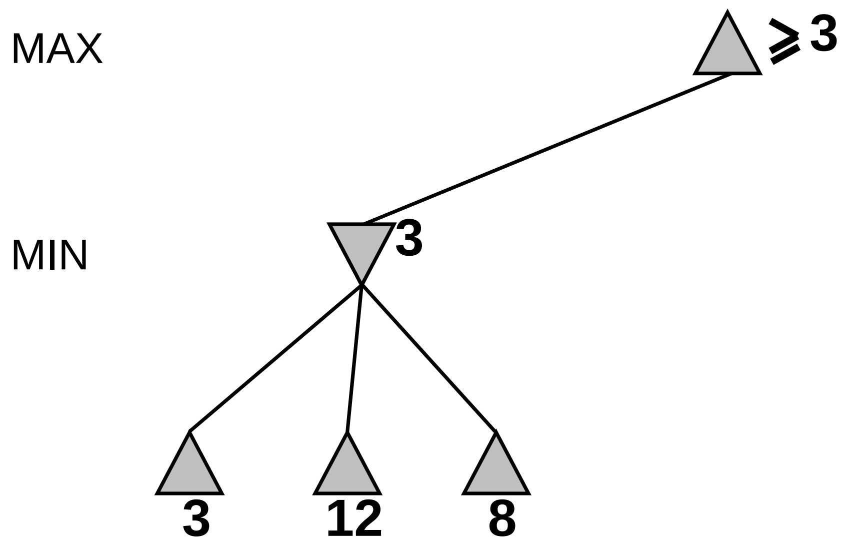


Behaviour is preserved under any **monotonic** transformation of EVAL

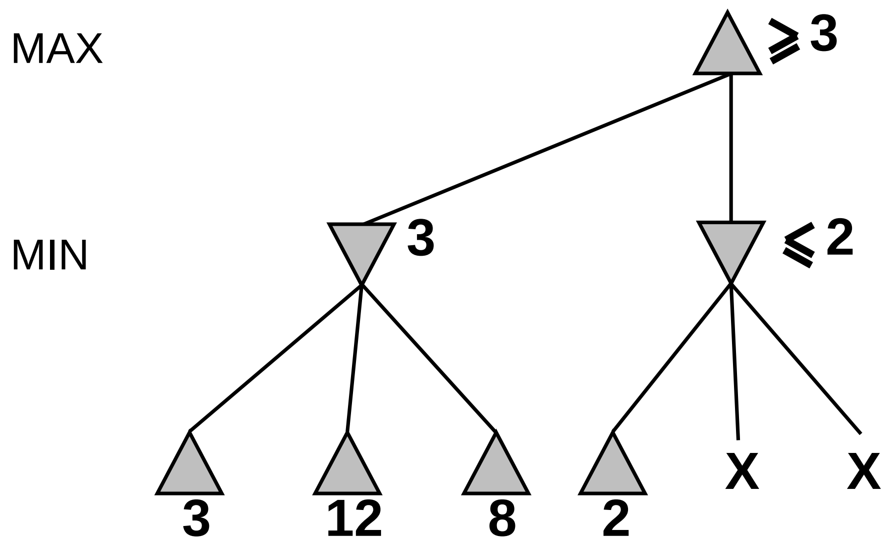
Only the order matters:

payoff in deterministic games acts as an ordinal utility function

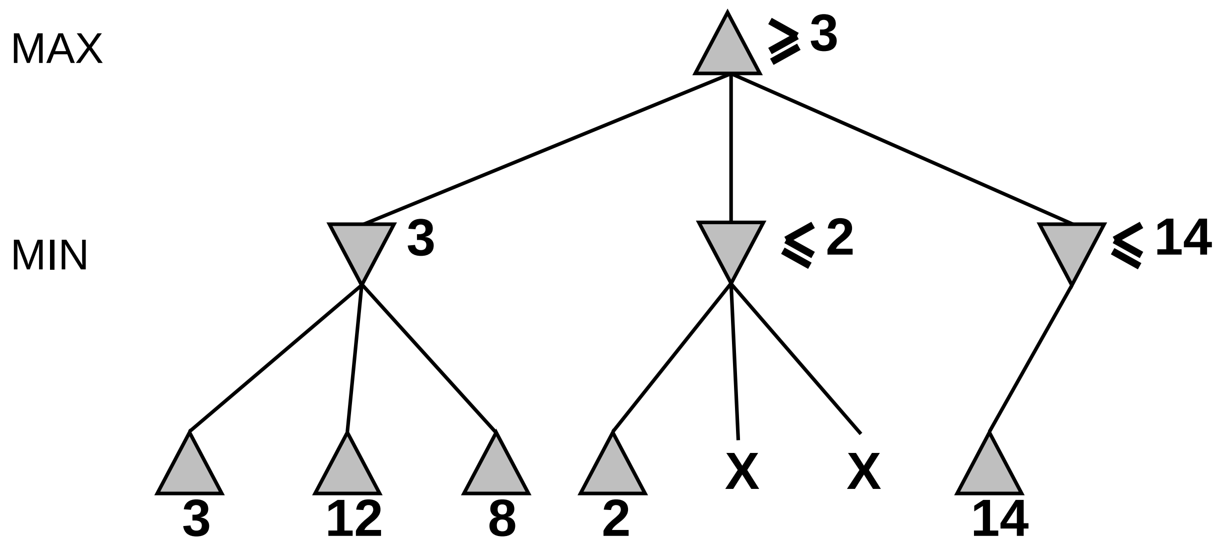
α–β pruning example



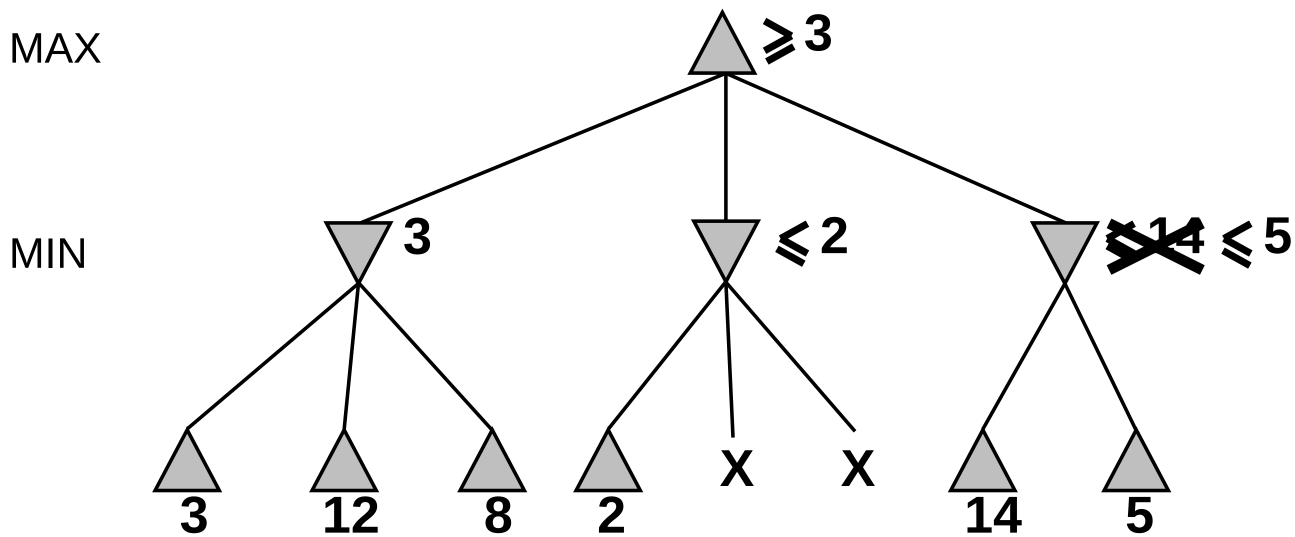
α–β pruning example



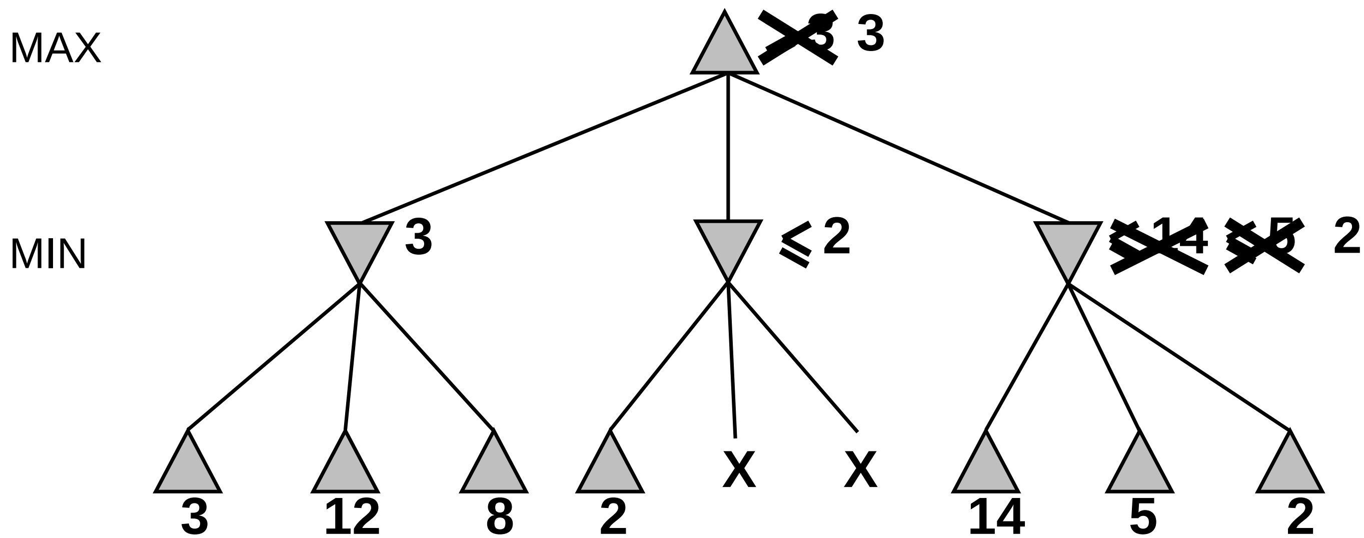
α–β pruning example



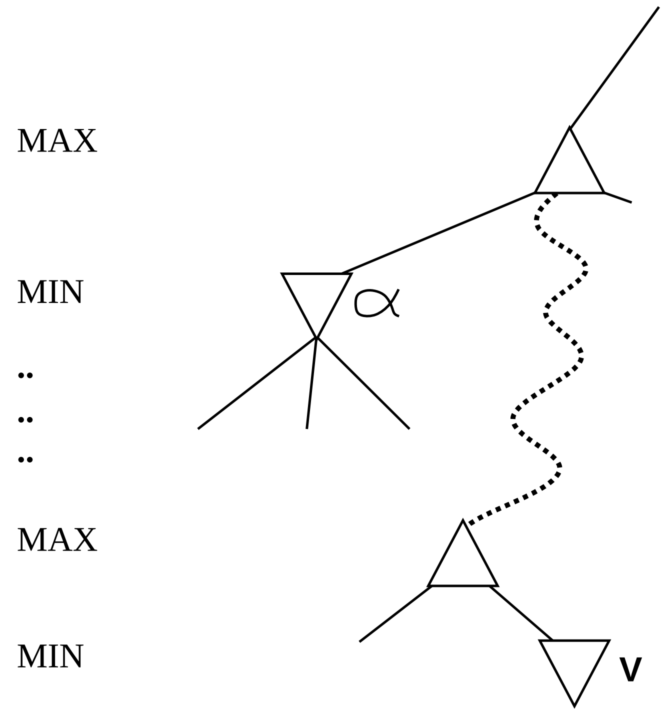
α–β pruning example



α–β pruning example



Why is it called α–β?



α is the best value (to MAX) found so far off the current path

If V is worse than α, MAX will avoid it $\Rightarrow$ prune that branch

Define β similarly for MIN

The α–β algorithm Min / Max

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

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!