Adversarial Search

Chapter 6

Section 1 – 4

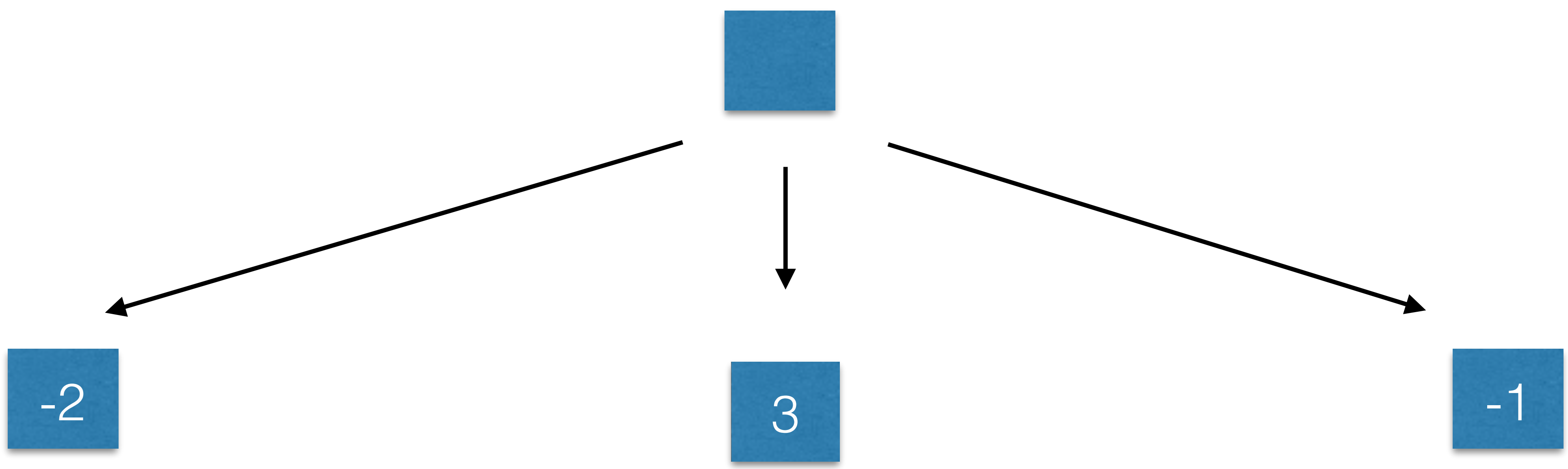
Outline

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

Games vs. search problems

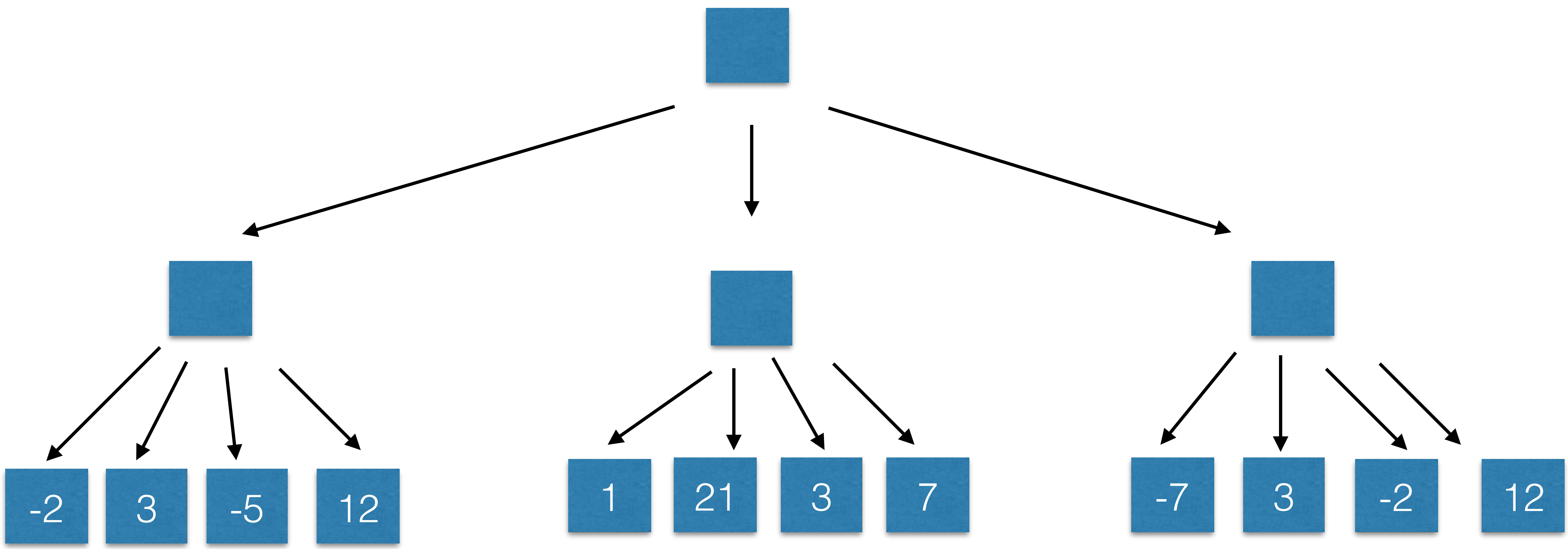
- "Unpredictable" opponent -> specifying a move for every possible opponent reply
- Time limits -> unlikely to find goal, must approximate

minimax - basic idea

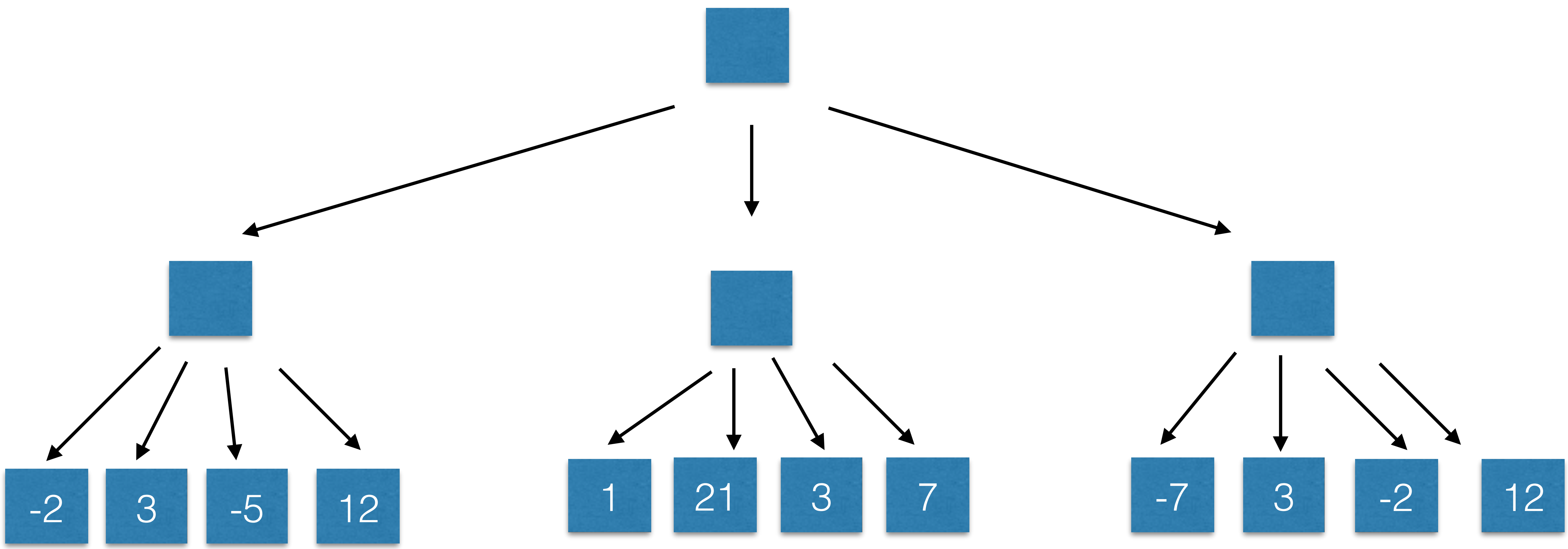


1 - turn game

2 turn game - opponents move

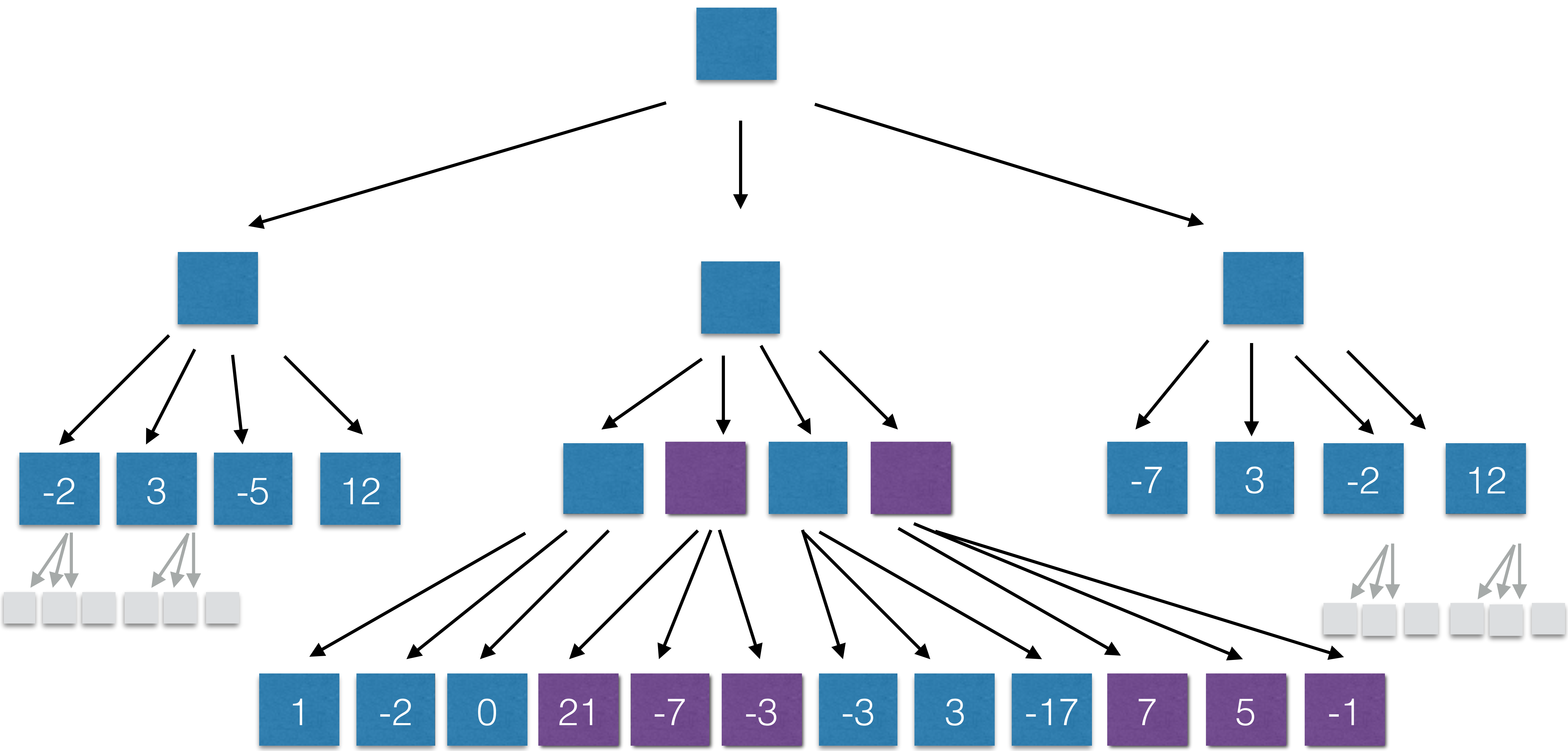


2 turn game - opponents move

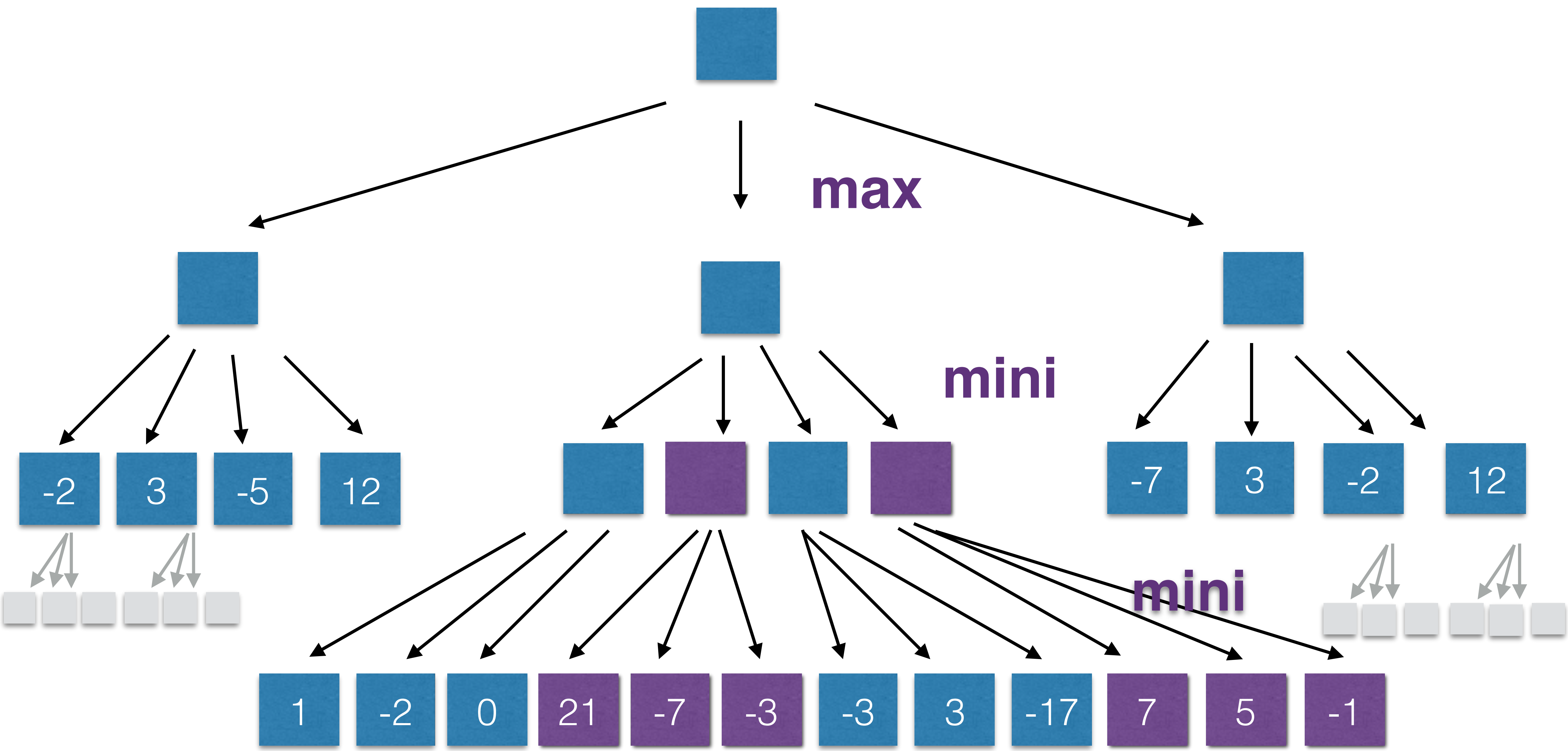


Our thinking is “*if I go here then my opponent will go there*”

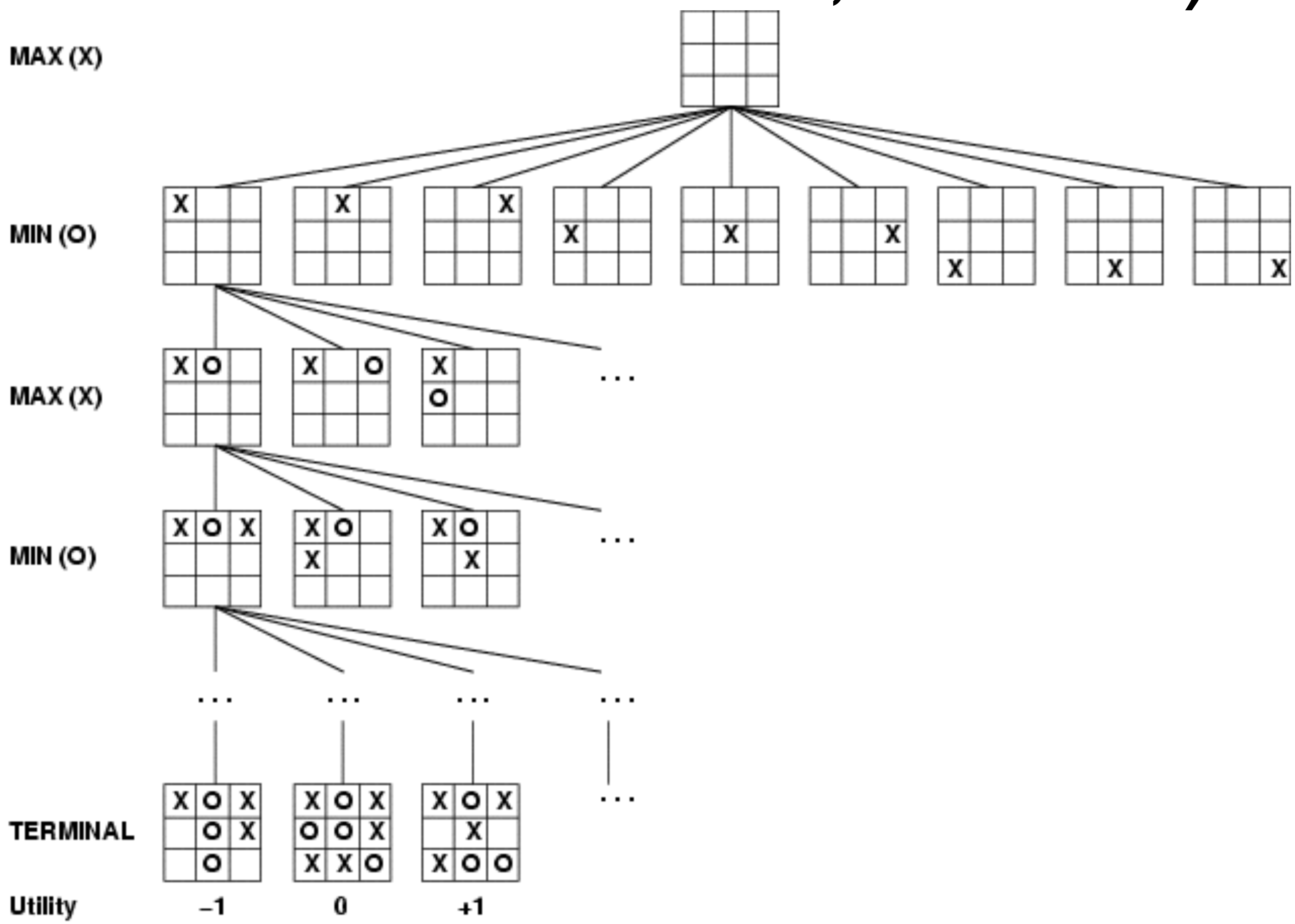
3 turn game - my move



minimax



Game tree (2-player, deterministic, turns)



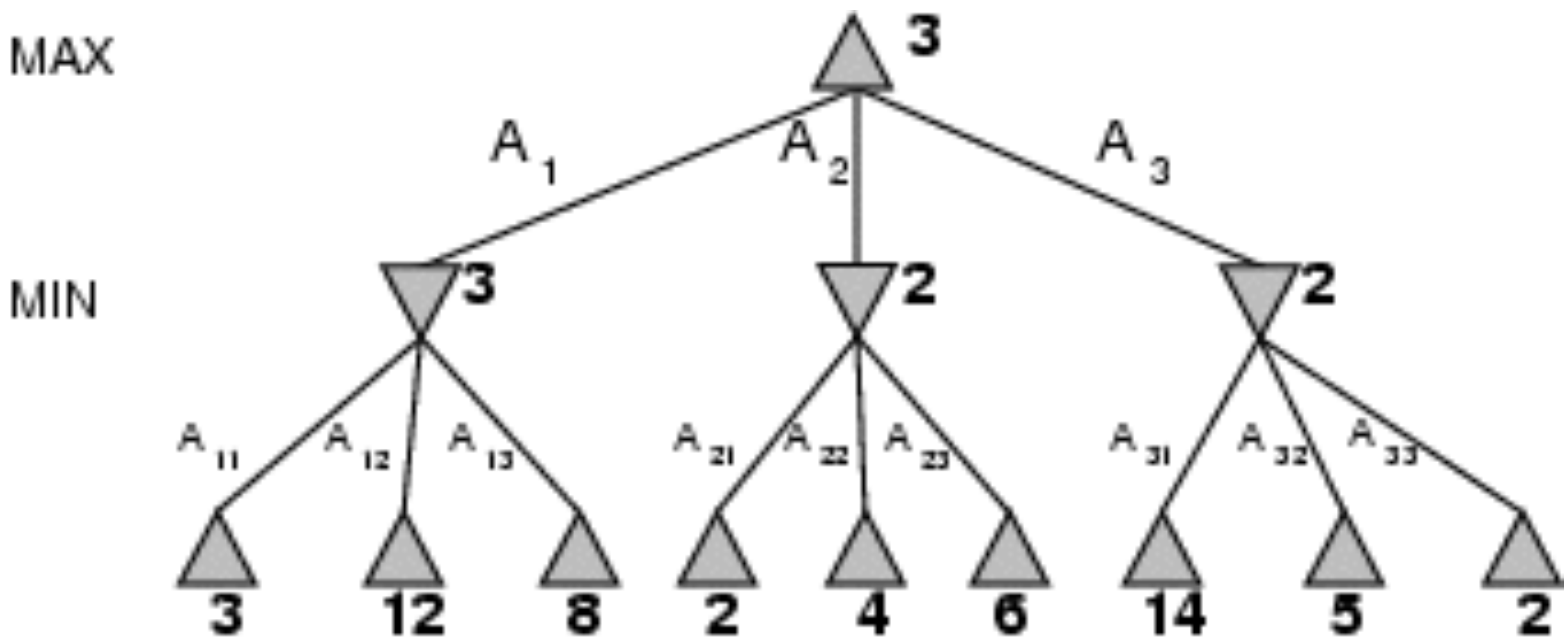
Minimax

- Perfect play for deterministic games

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

- E.g., 2-ply game:

Minimax


MAX
MIN
3
A 1
A 2
A 3
3
2
2
A 11
A 12
A 13
A 21
A 22
A 23
A 31
A 32
A 33
3
12
8
2
4
6
14
5
2

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**
        $v \leftarrow$ MAX(v, MIN-VALUE(s))
    return v

---

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

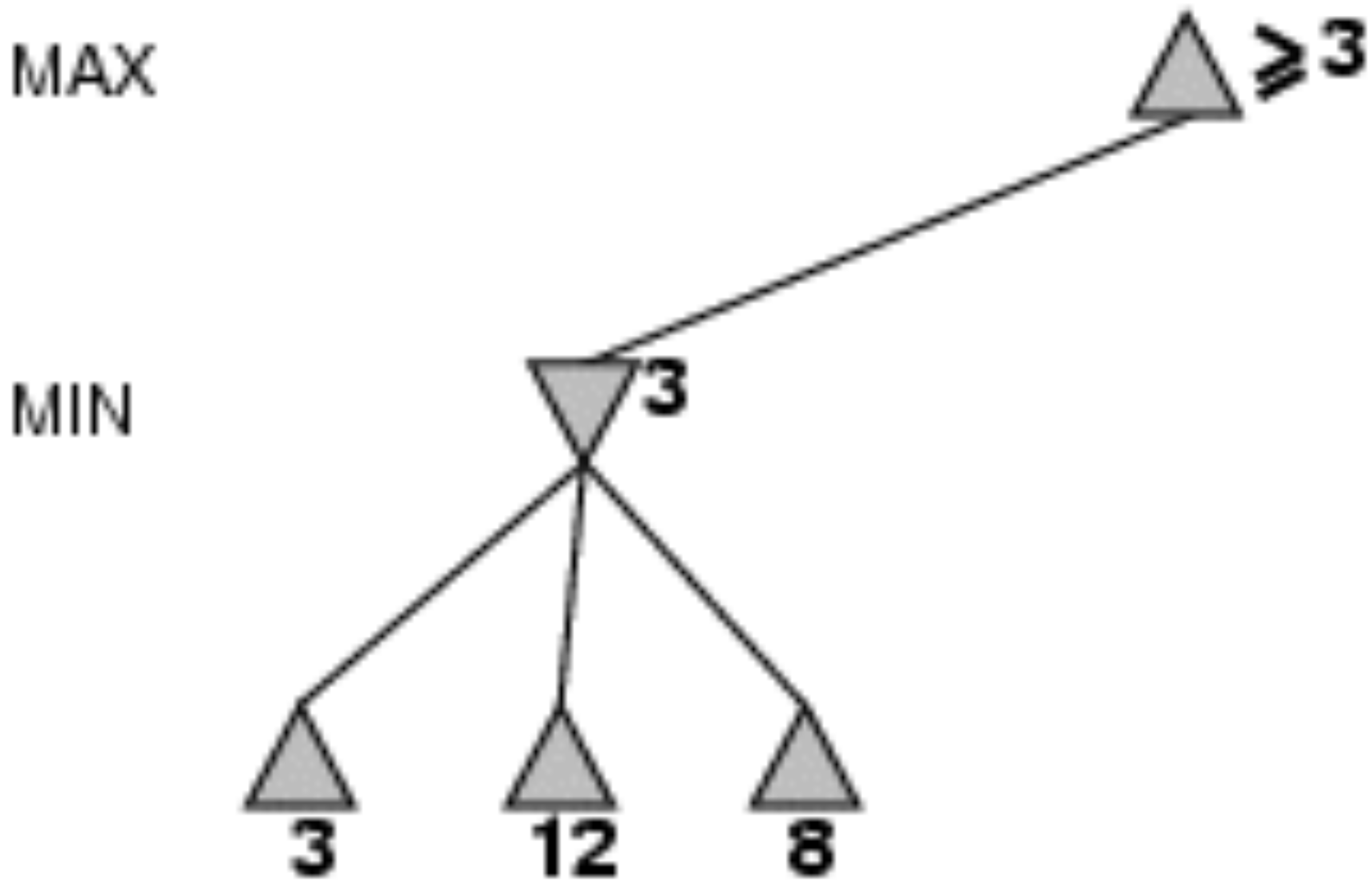
    if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
    $v \leftarrow \infty$
    for a, s in SUCCESSORS(*state*) **do**
        $v \leftarrow$ MIN(v, MAX-VALUE(s))
    return v

Properties of minimax

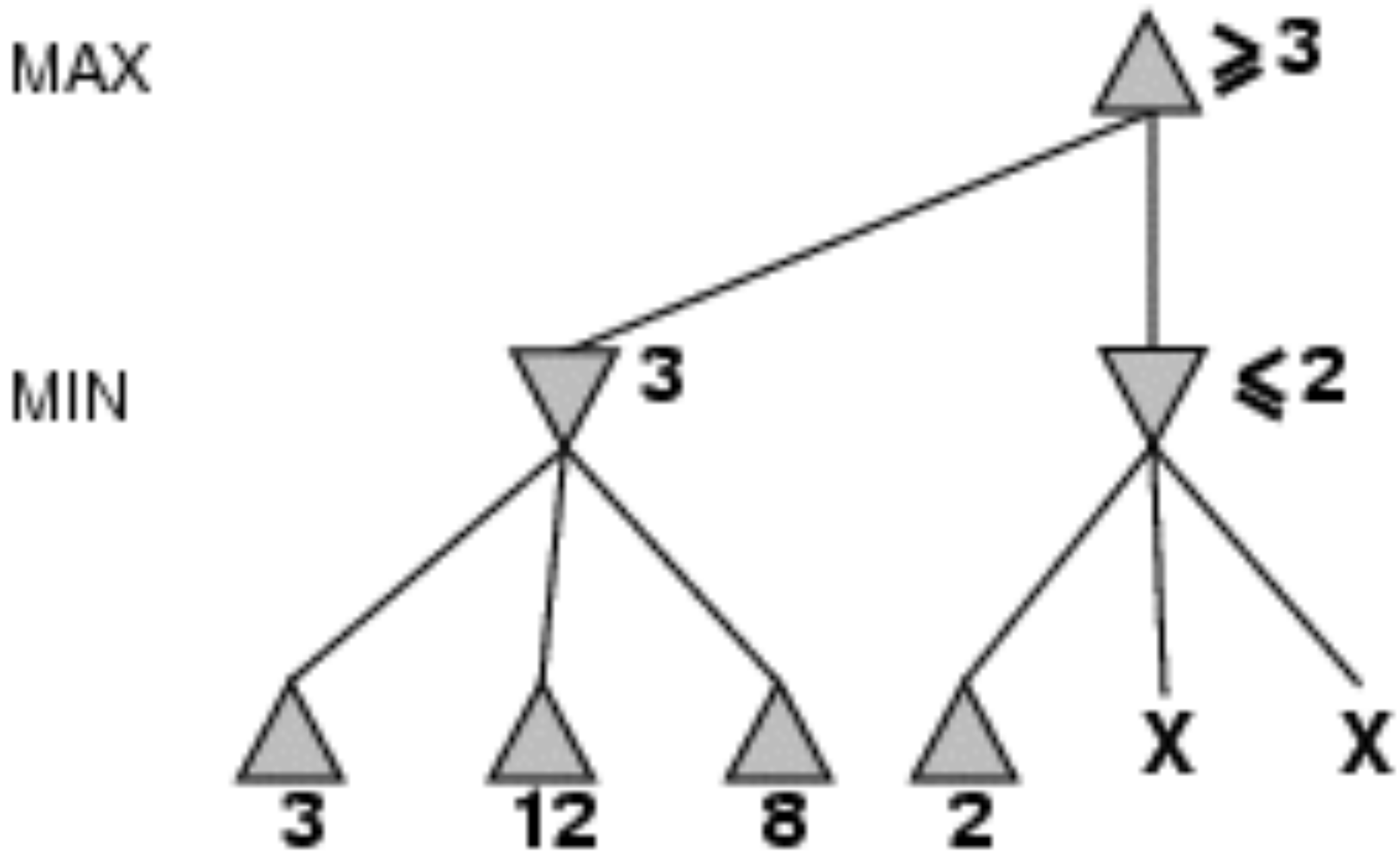
- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
- Time complexity? $O(b^m)$
- Space complexity? O(bm) (depth-first exploration)
- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  -> exact solution completely infeasible

```
2551552067298685292412115015142558763019041448816101932417677844077146725823993736
5843732987043555789782336195637736653285543297897675074636936187744140625L
```

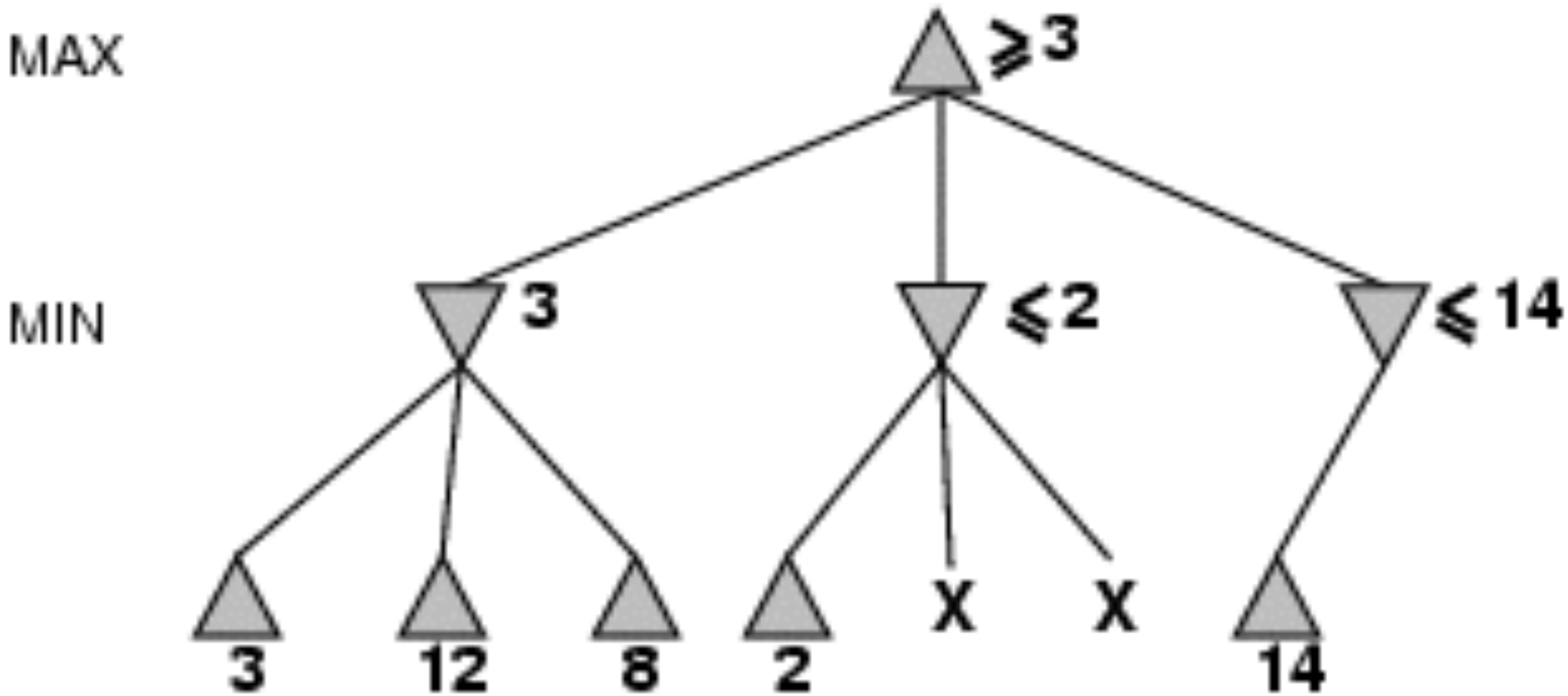
α-β pruning example



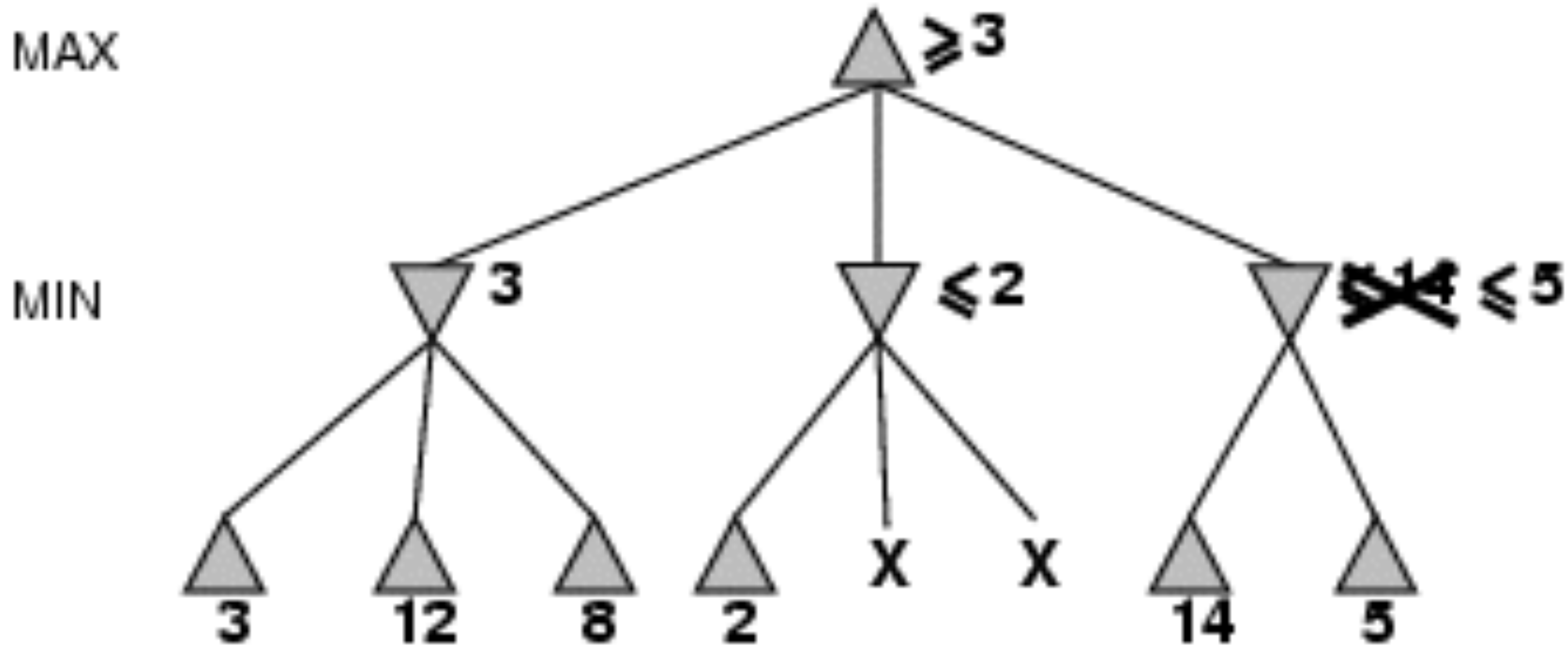
α-β pruning example



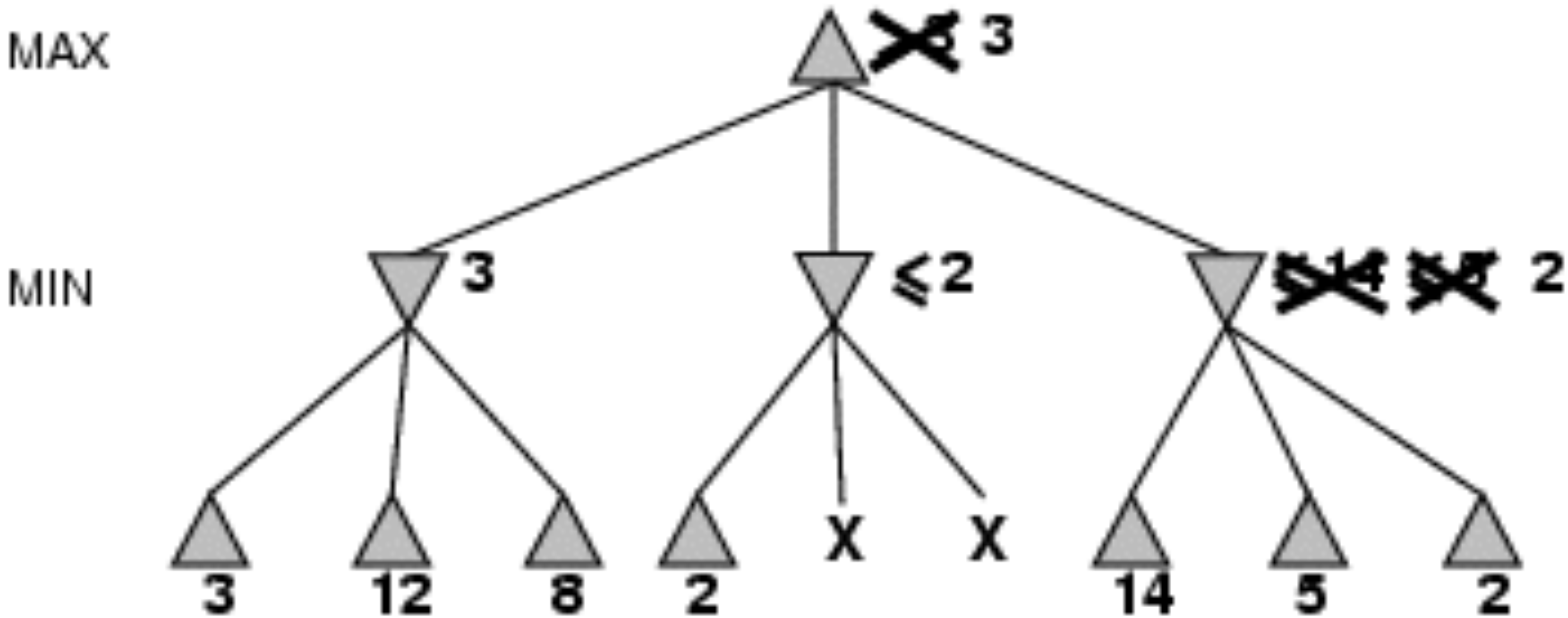
α-β pruning example



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 = $O(b^{m/2})$
  - doubles depth of search
- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

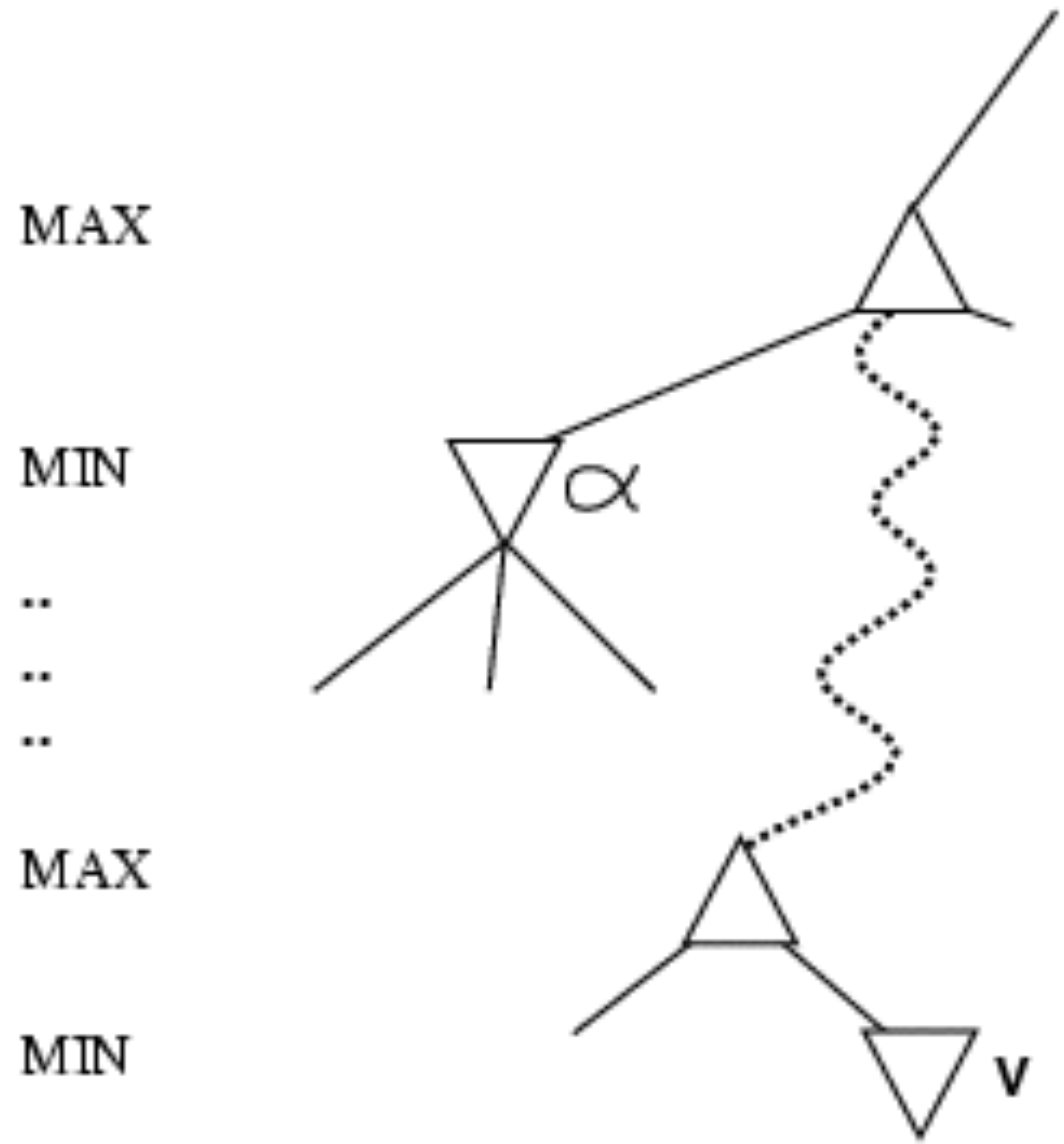
Why is it called α-β?

- 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($state, \alpha, \beta$) **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

The α-β algorithm

function MIN-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**
    $v \leftarrow$ MIN(v, MAX-VALUE(s, α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

Resource limits aka horizon problem

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

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

Evaluation functions

- For chess, typically linear weighted sum of features

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

e.g., $w_1 = 9$ with

- $f_1(s)$ = (number of white queens) – (number of black queens), etc.

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?

$b^m = 106$, $b=35$ -> $m=4$

- 4-ply lookahead is a hopeless chess player!
  - 4-ply ≈ human novice
  - 8-ply ≈ typical PC, human master
  - 12-ply ≈ Deep Blue, Kasparov

Deterministic games in practice

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed 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.

- 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.

Summary

- Games are fun to work on!

- They illustrate several important points about AI

- perfection is unattainable -> must approximate
- good idea to think about what to think about