Lecture 6: Adversarial Search

Artificial Intelligence
CS-UY-4613-A / CS-GY-6613-I

Julian Togelius
julian.togelius@nyu.edu

Extra! Hear all about it!

- Adversarial problems and games
- Minimax
- Alpha-beta pruning
- Heuristics and considerations
- After the break: Algorithms to implement

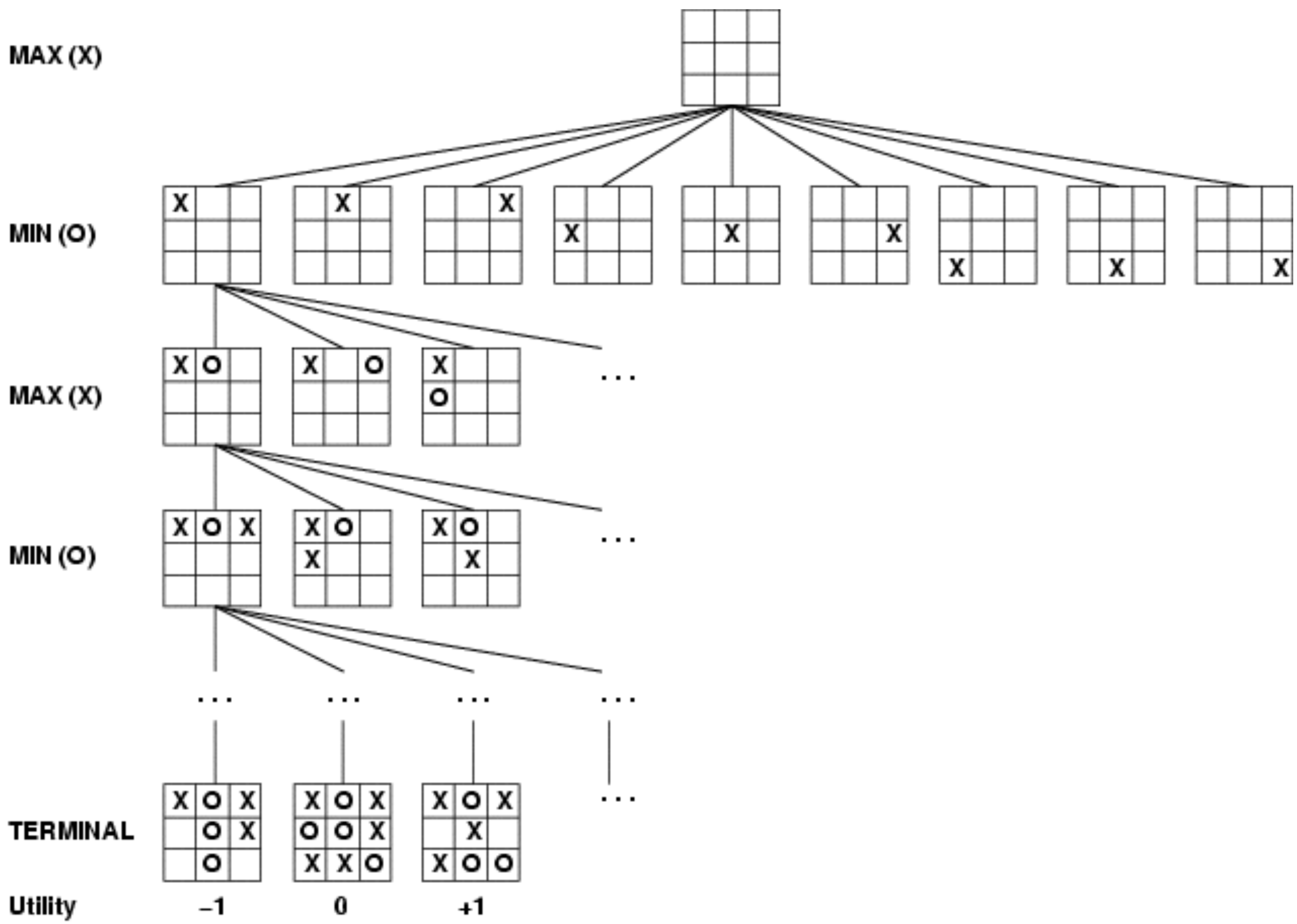
Adversarial problems?

- So far, problems have been deterministic with a known forward model
    - Though the methods would work with noise and/or imperfect models
- Adversarial: there is another agent, which works against you
    - The opponent is not predictable
- Game trees rather than search trees

Paradigm case

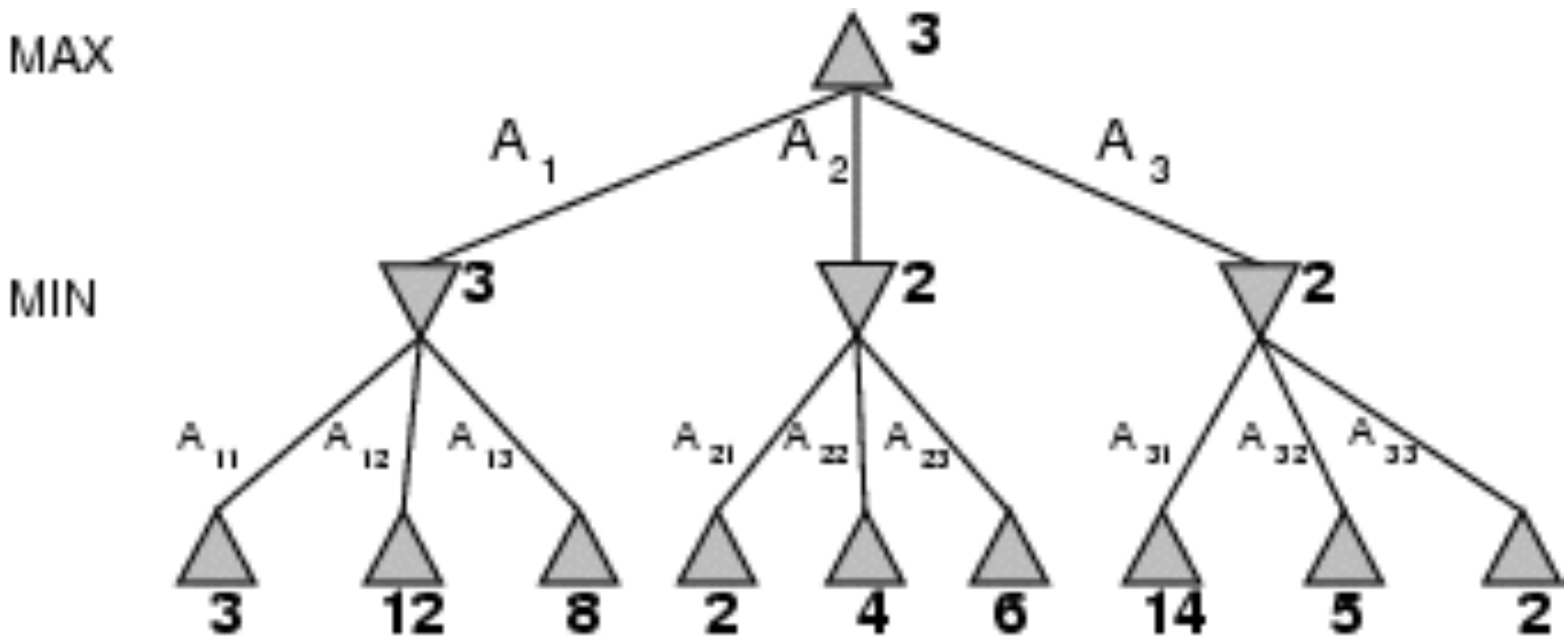
Other cases

Two-player game tree



Minimax

- Perfect play for deterministic games
- Choose move to state with highest minimax value (best payoff against perfect opponent)



```
function MINIMAX-DECISION(state) returns an action
    v ← 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 ← −∞
    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
```

```
MinMax (GamePosition game) {
  return MaxMove (game);
}

MaxMove (GamePosition game) {
  if (GameEnded(game)) {
    return EvalGameState(game);
  }
  else {
    best_move <- {};
    moves <- GenerateMoves(game);
    ForEach moves {
       move <- MinMove(ApplyMove(game));
       if (Value(move) > Value(best_move)) {
          best_move <- move;
       }
    }
    return best_move;
  }
}

MinMove (GamePosition game) {
  best_move <- {};
  moves <- GenerateMoves(game);
  ForEach moves {
     move <- MaxMove(ApplyMove(game));
     if (Value(move) > Value(best_move)) {
        best_move <- move;
     }
  }

  return best_move;
}
```

Properties of Minimax

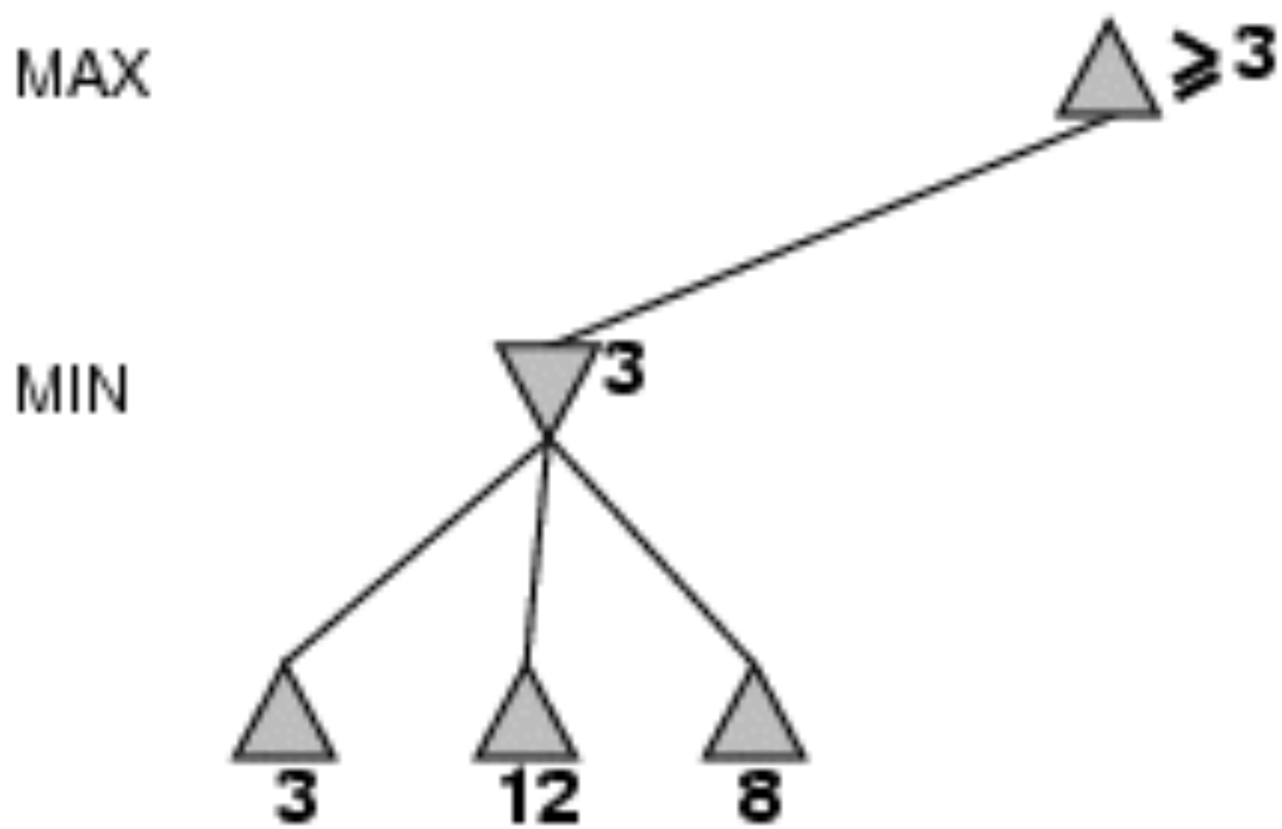
- *Complete?* Yes (if tree is finite)
- *Optimal?* Yes (against an optimal opponent)
- *Time complexity?* $O(b^m)$
- *Space complexity?* $O(bm)$ (depth-first exploration)

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

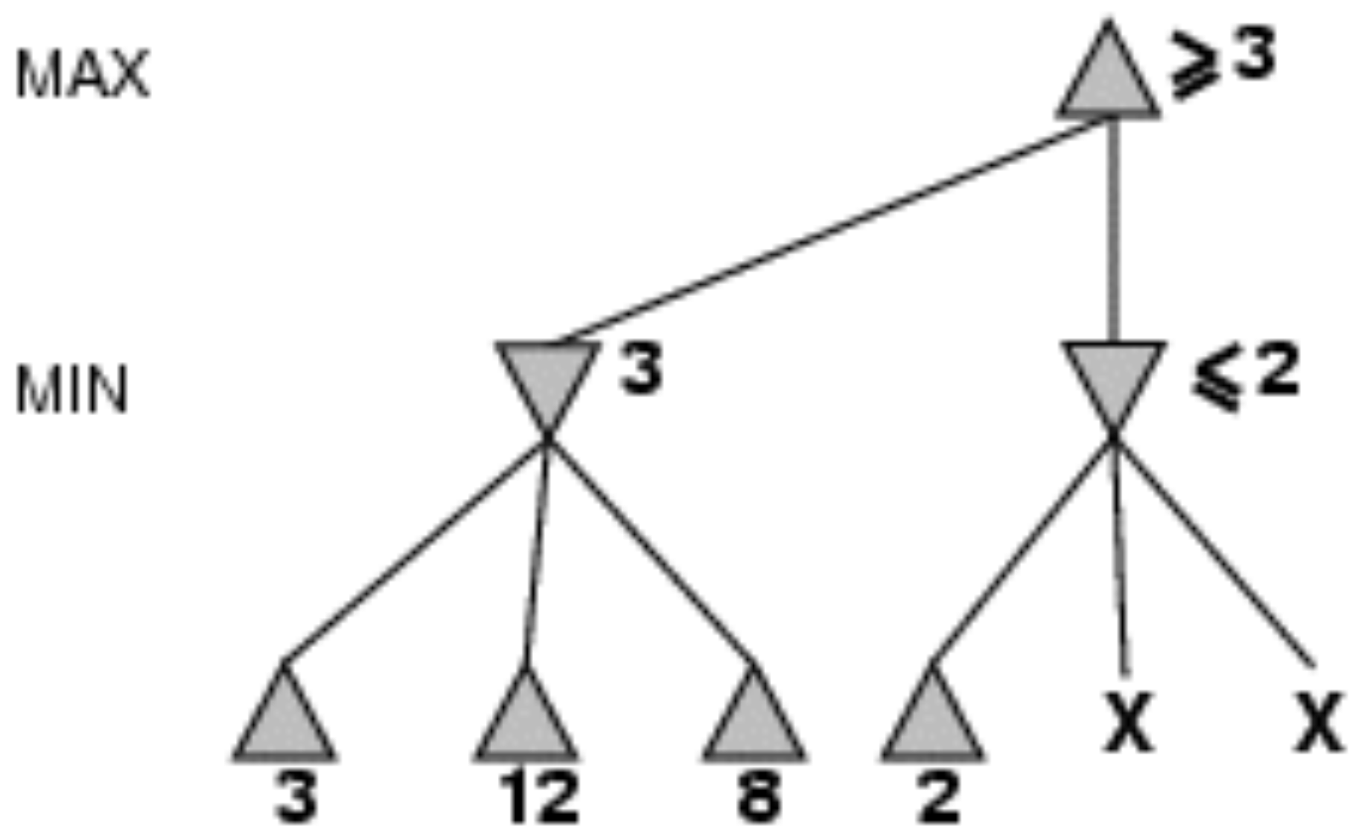
α-β pruning

- Can we improve this?
- Idea: don't consider branches of the tree that cannot lead to a better outcome than those that we have already explored

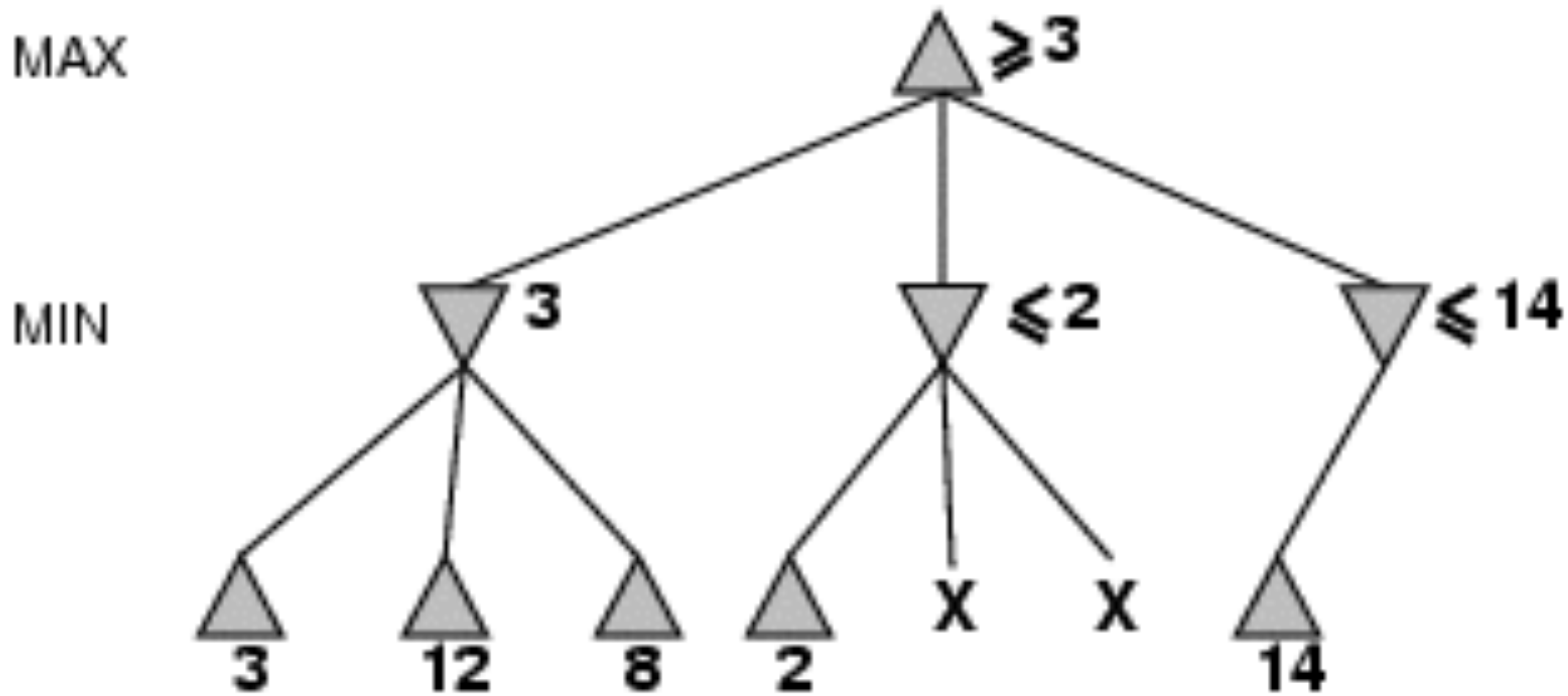
α-β pruning



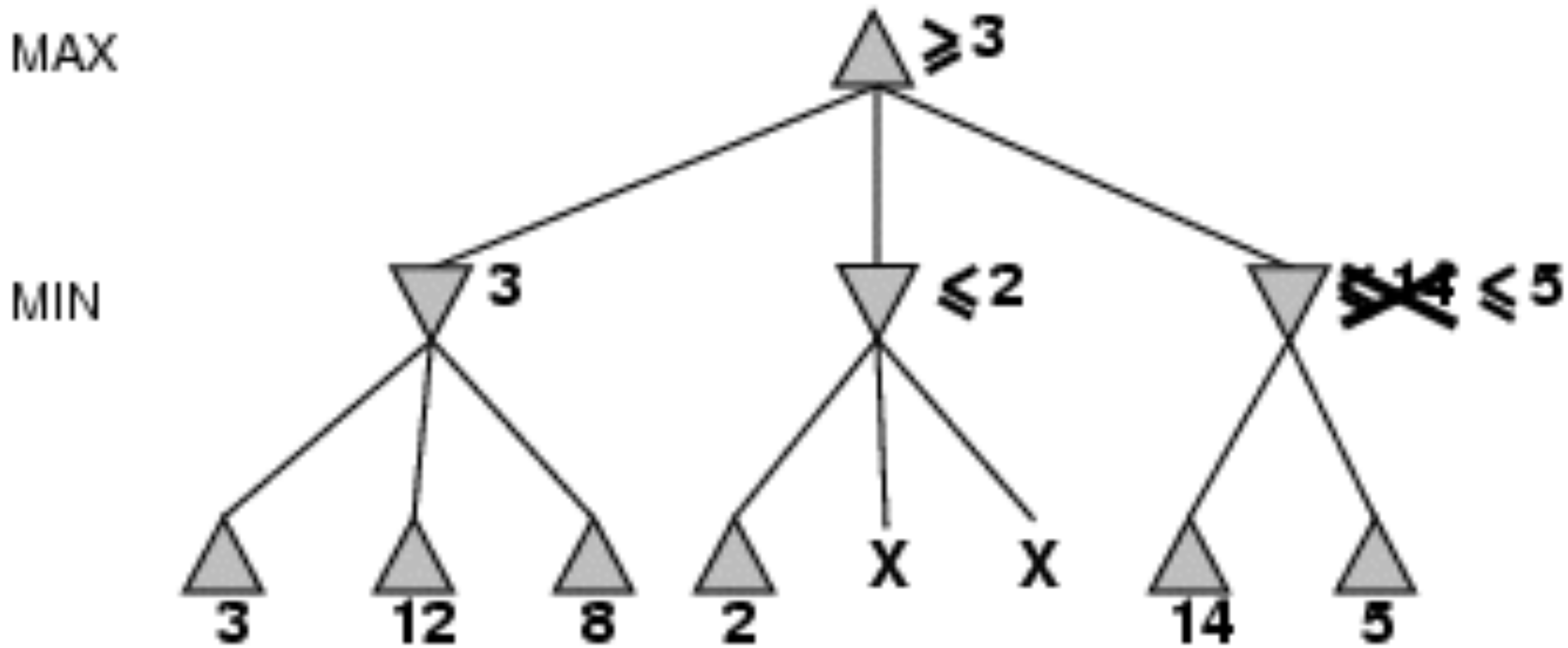
α-β pruning



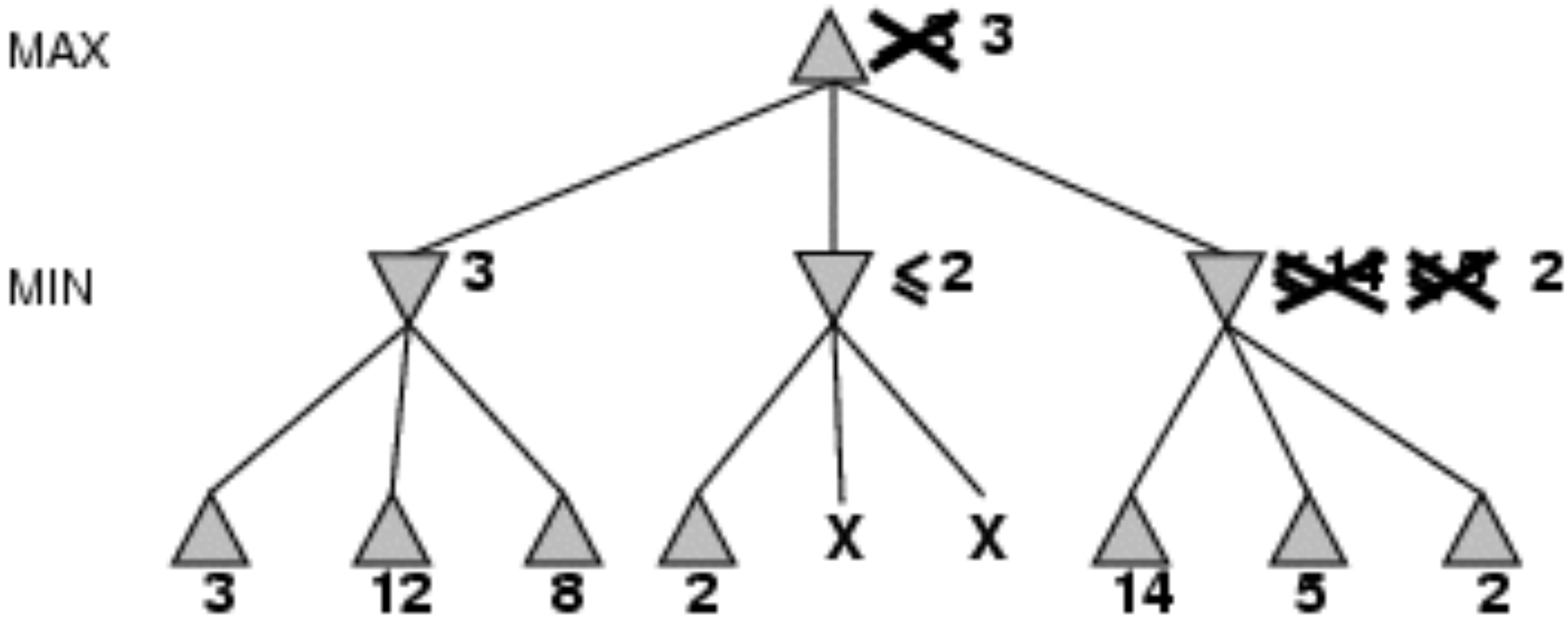
α-β pruning



α-β pruning



α-β pruning



Properties of α-β

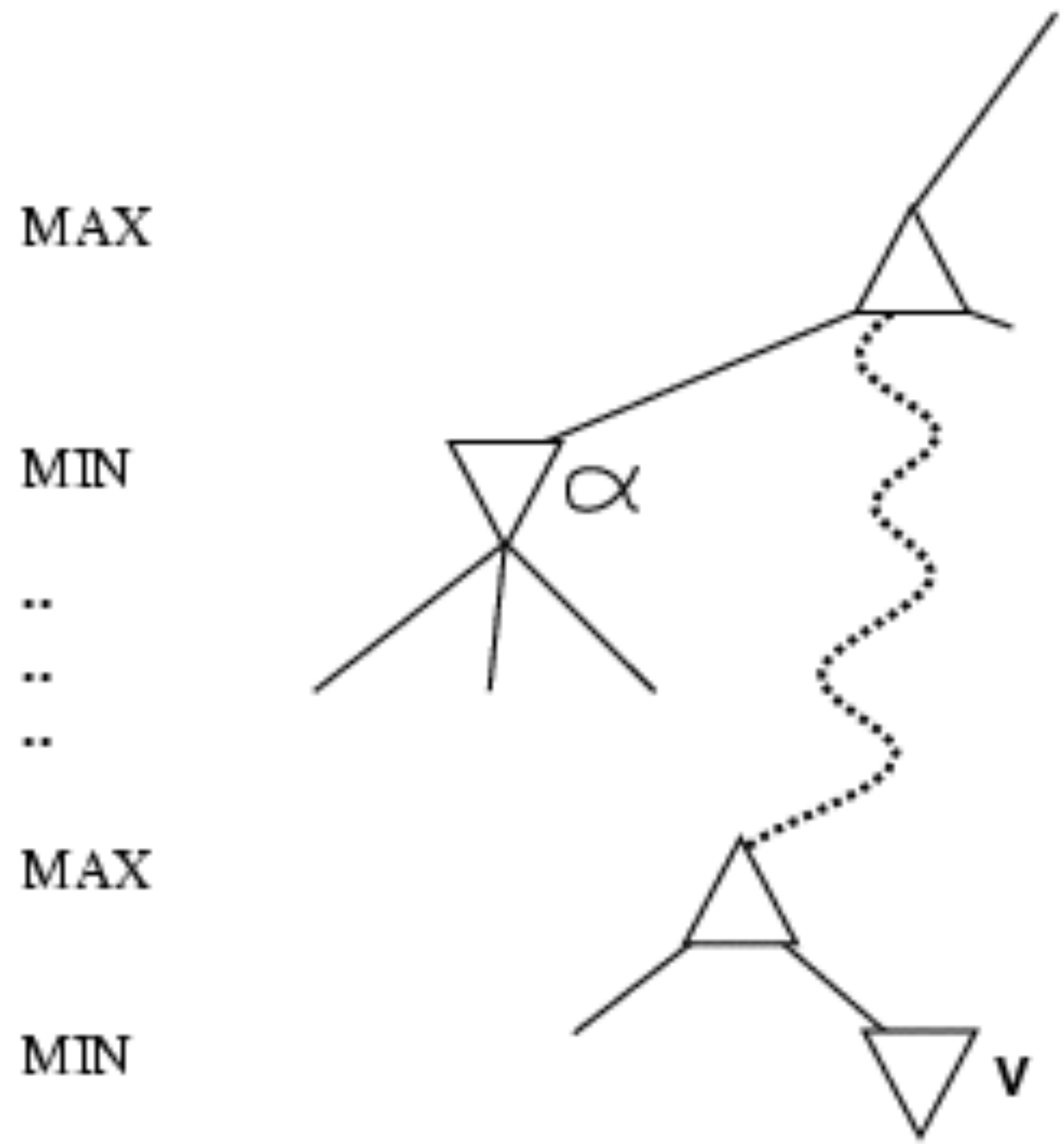
- Pruning does **not** affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering”, time complexity = $O(b^{m/2})$
  - doubles depth of search
- A simple example of the value of reasoning about which computations are relevant (metareasoning)

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*



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

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

Still, there is no time…

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

    - a far cry from 35^{100}…

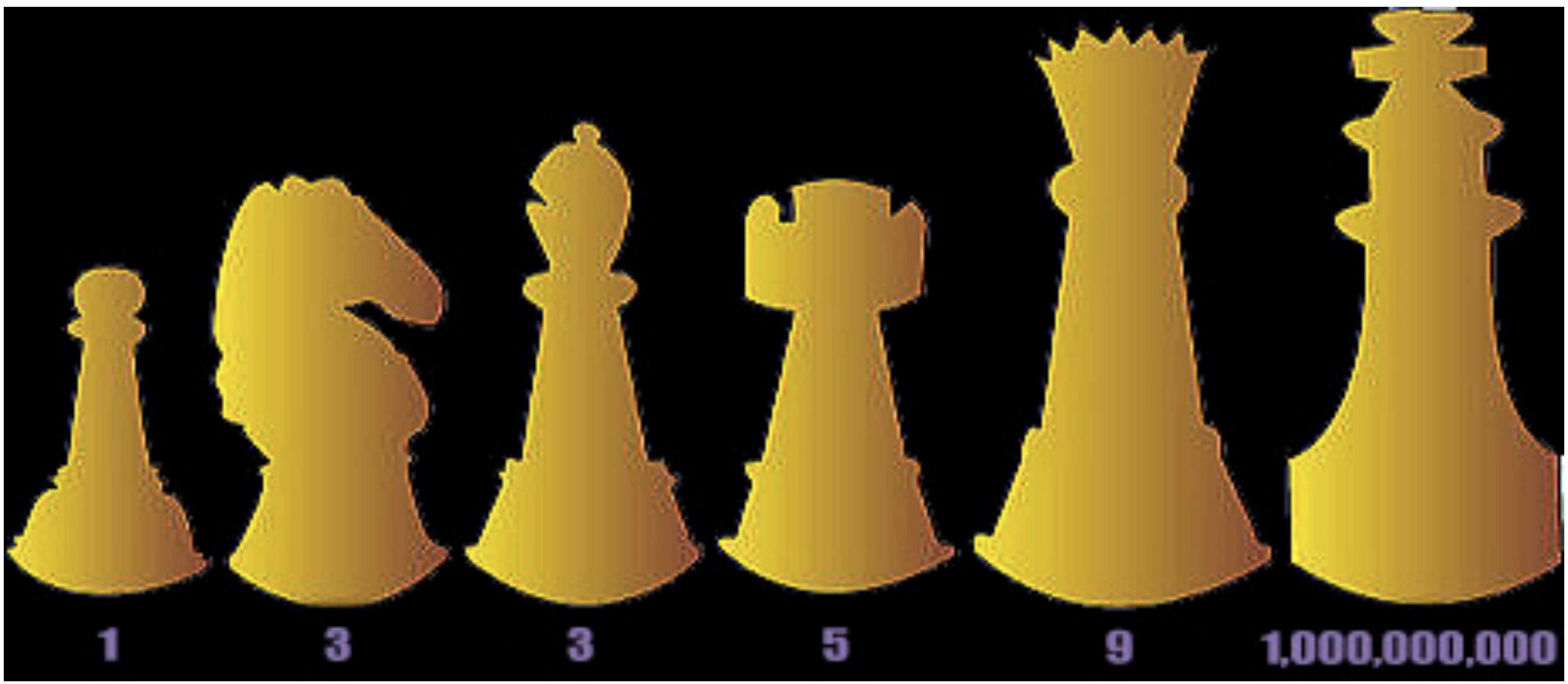
- Standard approach: use an *evaluation function* (heuristic)

- Either at the same depth for all branches, or use a *cutoff test* such as in quiescence search

Evaluation functions

- Simplest: number of white pieces - number of black pieces
- More complex: assign values to piece types
- Even more complex: count number of threats, try to recognize known positions
- Generally: Linear weighted sum of features
- Neural network
- Read more: Blondie24 by David Fogel

Piece weights?



Cutting off search

- MinimaxCutoff is identical to MinimaxValue except
  - Terminal? is replaced by Cutoff?
  - Utility is replaced by Eval
- Does it work in practice?
  - $b^m = 10^6$, b=35 means m=4
- 4-ply lookahead is (in general) a hopeless chess player!
  - 4-ply ≈ human novice
  - 8-ply ≈ old-school Chess program, human master
  - 12-ply ≈ Deep Blue, Kasparov

Some deterministic two-player games

- Chess: Kasparov vs Deep Blue 1997
- Checkers: Chinook almost defeated Tinsley (grand champion) 1994
    - Solved 2007
- Othello: computers vastly better than humans
- Go: computers currently at advanced intermediate level, thanks to MCTS

Discussion on projects

Important

- ***<u>GAME TREE SEARCH: YOU ARE SEARCHING IN STATE SPACE, NOT JUST PHYSICAL SPACE</u>***

- Optimization: could be used to find a path, or to find a controller, or parameters for a search algorithm

- In general, there are several different ways of using each algorithm

- Some of the algorithms will suck. That's OK.

What is the goal/utility?

- Mario: end of level / progress to the right edge of screen
- Pac-Man: clearing the level / score OR pills eaten OR distance to ghosts OR combination
- General Video Game Playing: winning / score OR whatever
- Fighting Game Competition: winning / health OR score

One-step search

- Start with a Game object
- Get set of moves (only four for Pac-Man)
- For each move, make a copy of Game (Game.copy())
- Try each move in its own copy (don’t forget to advanceGame())
- Choose the Game state with highest score