Adversarial Search

Tomáš Svoboda

Department of Cybernetics, Vision for Robotics and Autonomous Systems,
Center for Machine Perception (CMP)

March 19, 2018

Games, man vs. algorithm

- Deep Blue
- Alpha Go
- Deep Stack
- Why Games, actually?

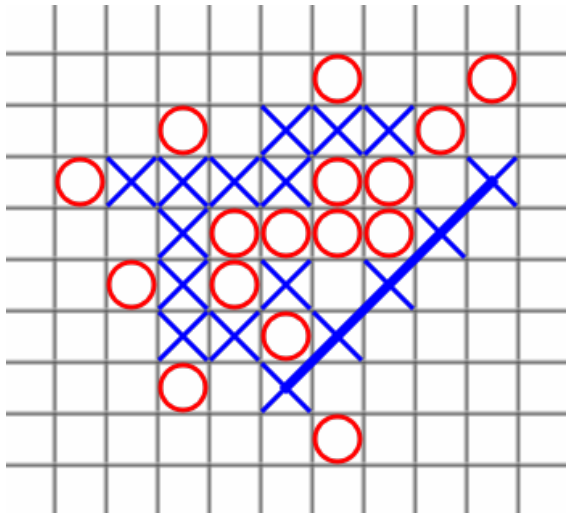
Games are interesting for AI *because* they are hard (to solve).

More: Adversarial Learning

Video: Adversing visual segmentation

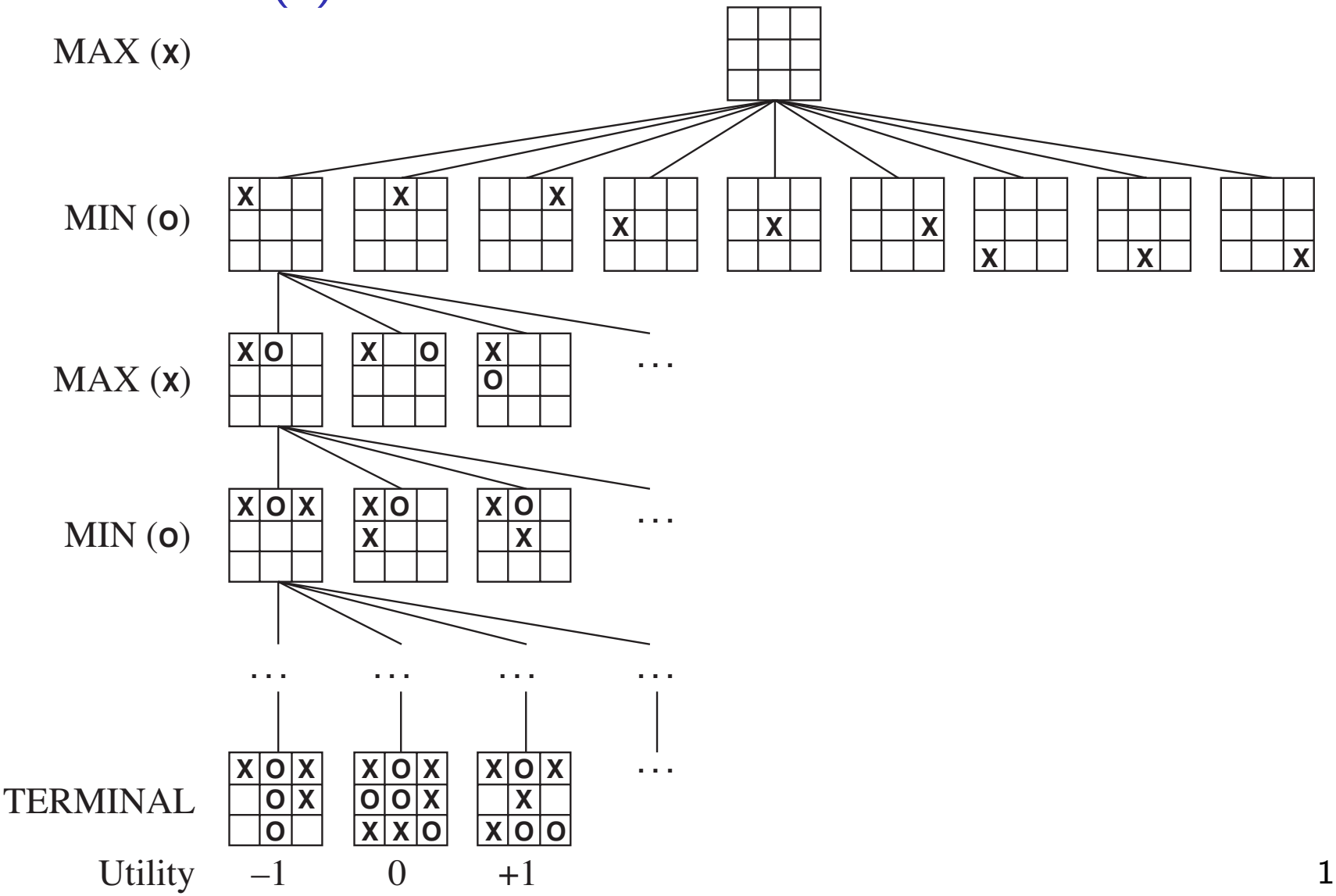
Elements of the game

- S_0: The initial state
- PLAYER(s). Which player has to move in s.
- ACTIONS(s). What are the legal moves?
- RESULT(s, a). Transition, result of a move.
- TERMINAL-TEST(s). Game over?
- UTILITY(s, p). What is prize? Examples for some games ...



https://commons.wikimedia.org/wiki/File:Tic-tac-toe_5.png

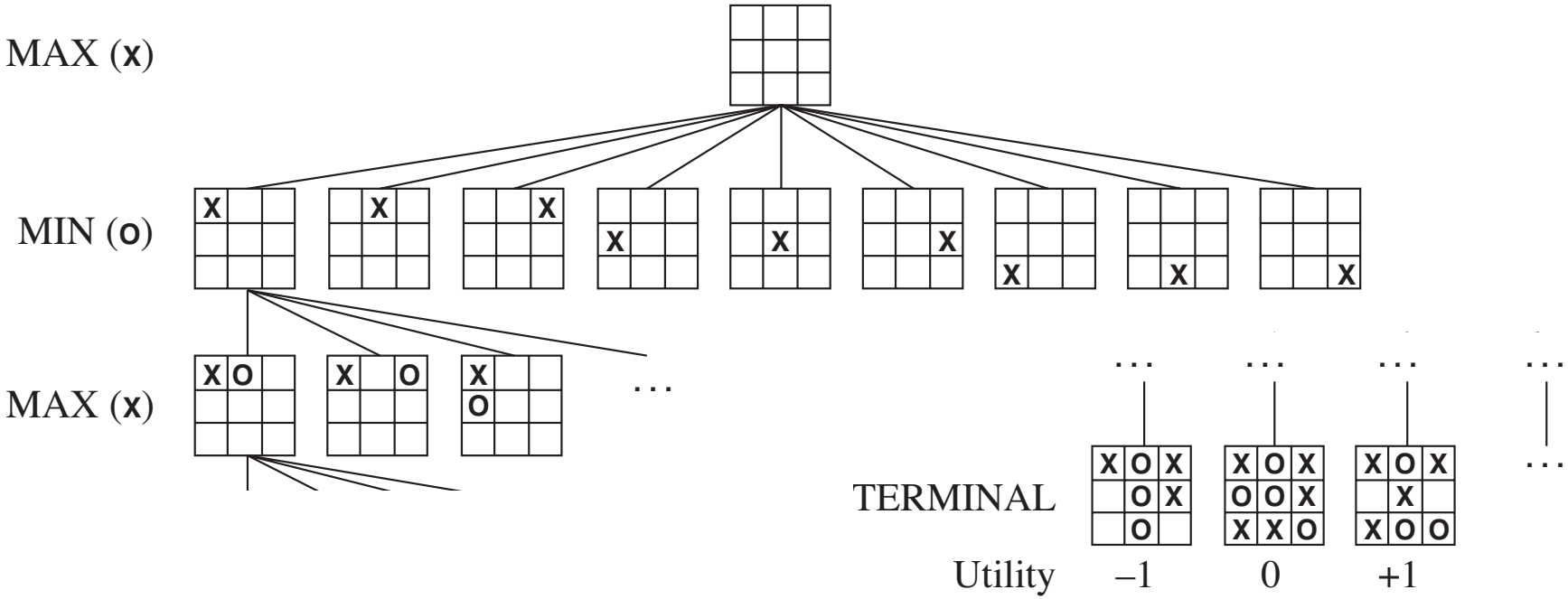
Game Tree(s)



Init state, ACTIONS function, and RESULT function defines game tree

[1] Image from the AIMA book [1]

How do we play - **max** for me, **min** for the opponent.



$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

Minimax algorithm

function MINIMAX(state) **returns** an action
  return $\underset{a \in \text{Actions}(s)}{\text{arg}\mathbf{max}}$ MIN-VALUE(RESULT(state, a))
end function

---

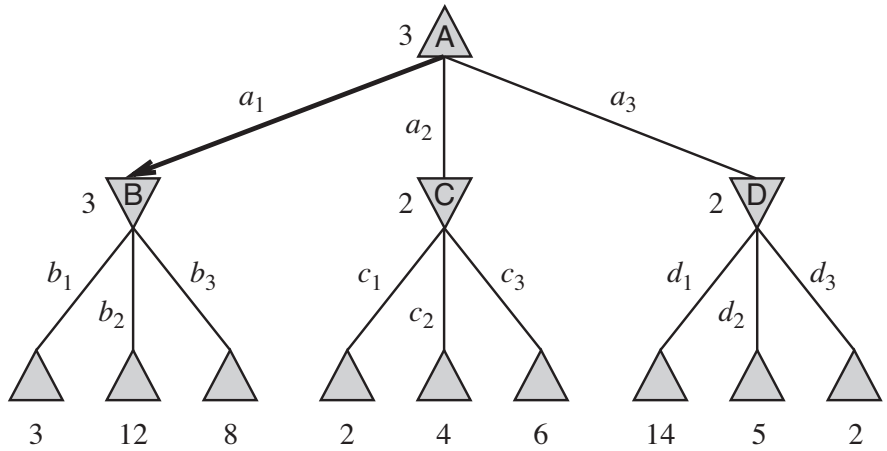
function MIN-VALUE(state) **returns** a utility value v
  if TERMINAL-TEST(state) **then return** UTILITY(state)
  end if
  $v \leftarrow \infty$
  for all ACTIONS(state) **do**
    $v \leftarrow$ **min**(v, MAX-VALUE(RESULT(state,a)))
  end for
end function

---

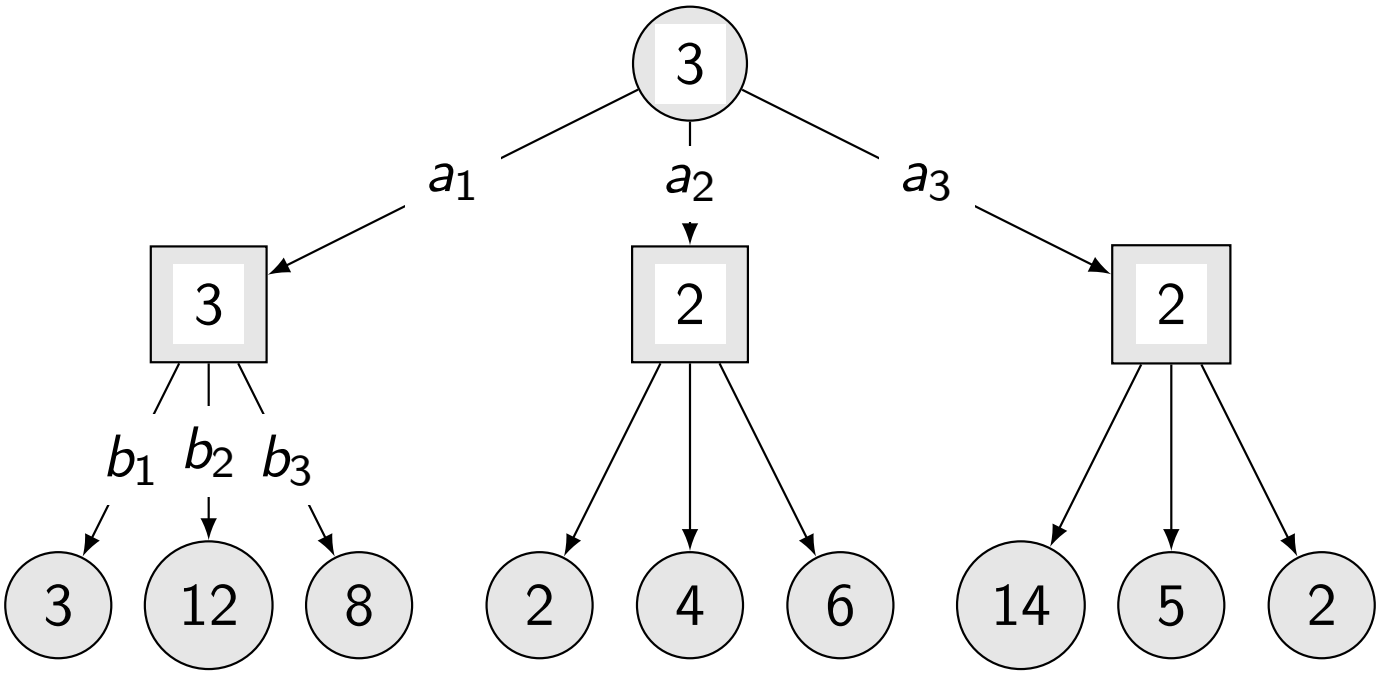
function MAX-VALUE(state) **returns** a utility value v
  if TERMINAL-TEST(state) **then return** UTILITY(state)
  end if
  $v \leftarrow -\infty$
  for all ACTIONS(state) **do**
    $v \leftarrow$ **max**(v, MIN-VALUE(RESULT(state,a)))
  end for
end function

A two ply game, down to terminal and back again . . .

```
function MINIMAX(s) returns a                                   MAX
    argmax_{a∈Actions(s)} MINVAL(RES(s, a))
end function
-------------------------------------------------------------   MIN
function MINVAL(s) returns v
    if TERMINAL(s) then UTIL(s)
    end if
    v ← ∞
    for all ACTIONS(s) do
        v ← min(v, MAXVAL(RES(s, a)))
    end for
end function
-------------------------------------------------------------
function MAXVAL(s) returns v
    if TERMINAL(s) then UTIL(s)
    end if
    v ← −∞
    for all ACTIONS(s) do
        v ← max(v, MINVAL(RES(s, a)))
    end for
end function
```



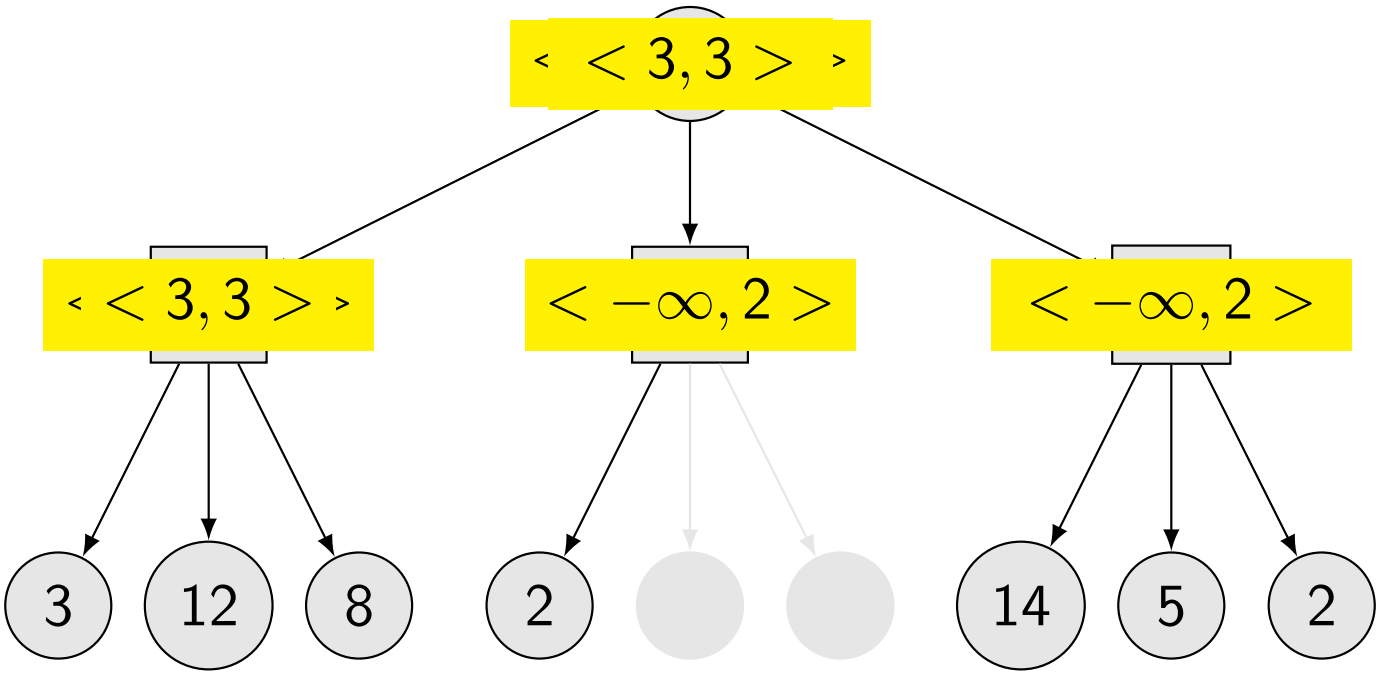
A two ply game, revisited



What is the complexity? How many nodes to visit?

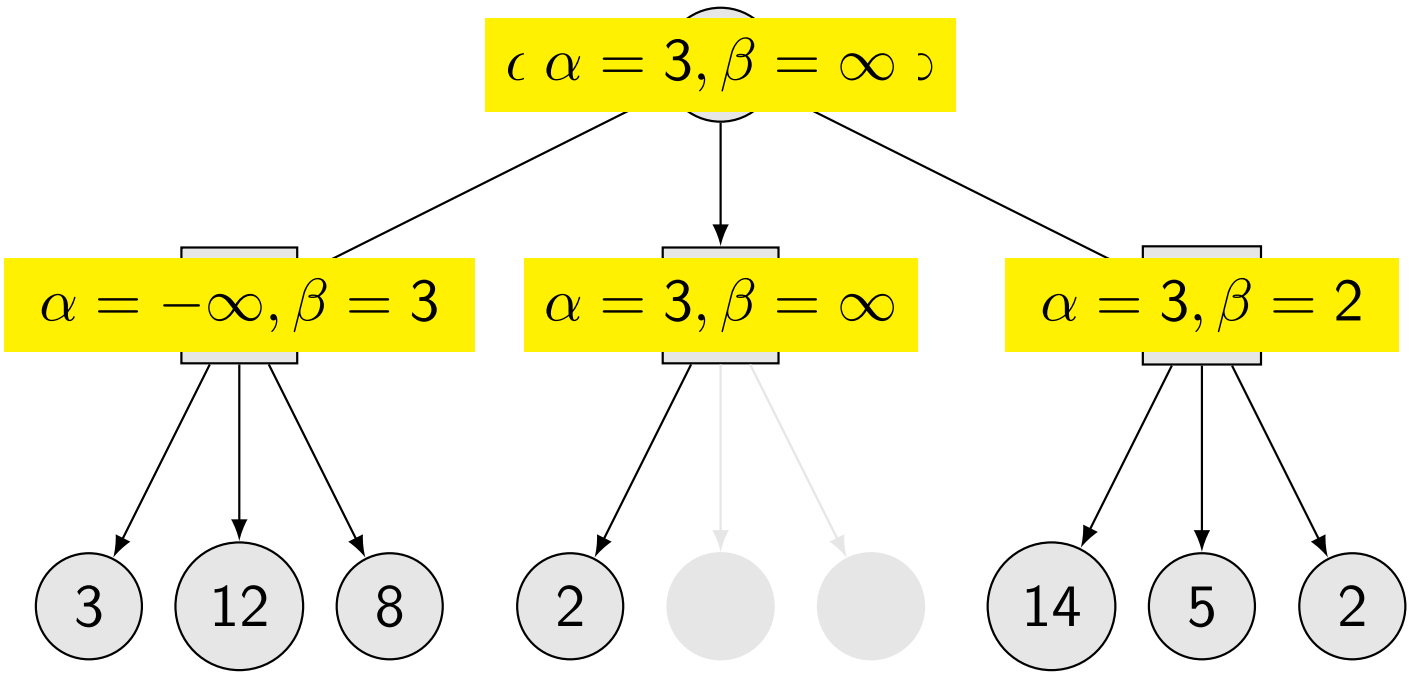
Can we do better? How?

Nodes worth visiting



α-β prunning

α highest value choice found so far for any choice along MAX
β lowest value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

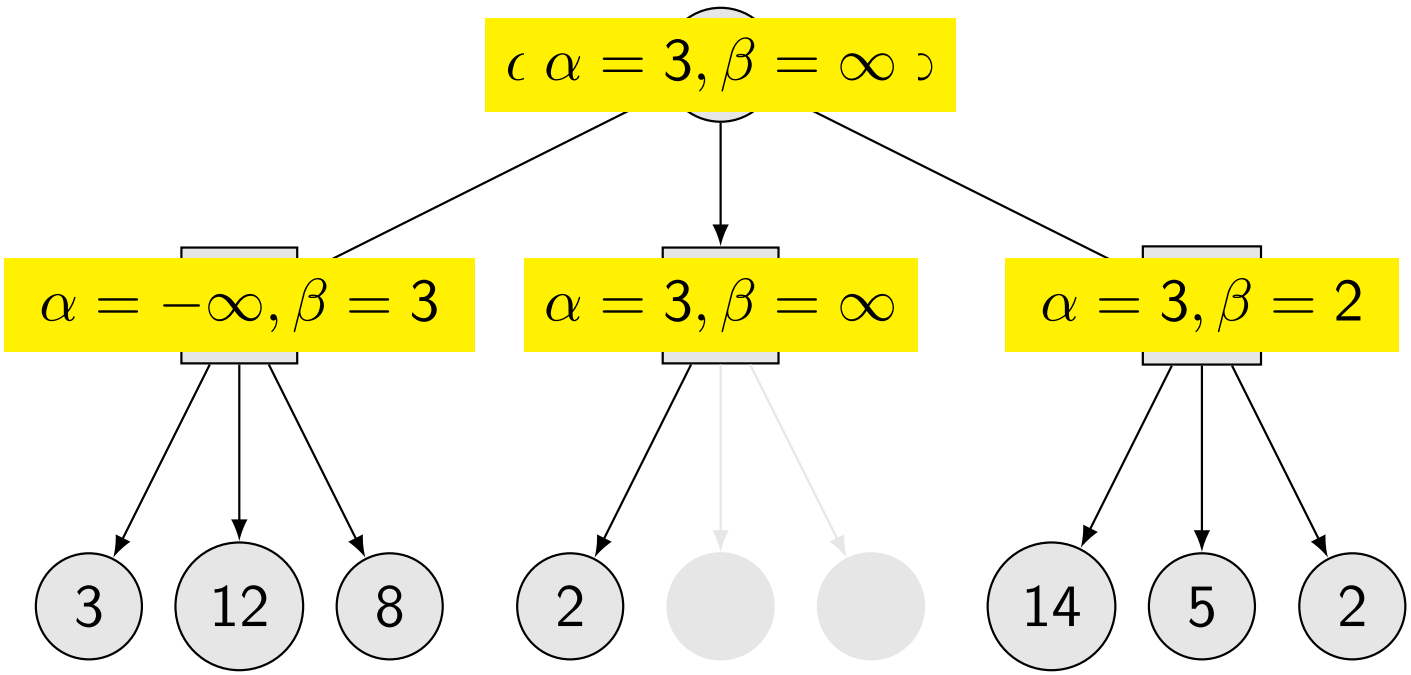
```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, −∞, ∞)
    return the action in ACTIONS(state) with value v
end function
```

```
function MAX-VALUE(state,α, β) returns a utility value v
    if TERMINAL-TEST(state) return UTILITY(state)
    v ← −∞
    for all ACTIONS(state) do
        v ← max(v, MIN-VALUE(RESULT(state,a),α, β))
        if v ≥ β return v
        α ← max(α, v)
    end for
end function
```

```
function MIN-VALUE(state, α, β) returns a utility value v
    if TERMINAL-TEST(state) return UTILITY(state)
    v ← ∞
    for all ACTIONS(state) do
        v ← min(v, MAX-VALUE(RESULT(state,a),α, β))
        if v ≤ α return v
        β ← min(β, v)
    end for
end function
```

Move ordering

α highest value choice found so far for any choice along MAX
β highest value choice found so far for any choice along MIN



Imperfect but real-time decisions

$$
\begin{aligned}
&\textsc{h-minimax}(s, d) = \\
&\qquad \textsc{eval}(s) \quad \text{if} \quad \textsc{cutoff-test}(s, d) \\
&\max_{a \in \textsc{actions}(s)} \textsc{h-minimax}(\textsc{result}(s, a), d + 1) \quad \text{if} \quad \textsc{player}(s) = \textsc{max} \\
&\min_{a \in \textsc{actions}(s)} \textsc{h-minimax}(\textsc{result}(s, a, d + 1)) \quad \text{if} \quad \textsc{player}(s) = \textsc{min}
\end{aligned}
$$

Cutting off search

Replace
if TERMINAL-TEST(s) **then return** UTILITY(s)
with:
if CUTOFF-TEST(s,d) **then return** EVAL(s)

Cutting depends on *d* only, why we need *s* as the input parameter?

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, −∞, ∞)
    return the action in ACTIONS(state) with value v
end function
```

```
function MAX-VALUE(state,α, β) returns a utility value v
    if TERMINAL-TEST(state) return UTILITY(state)
    v ← −∞
    for all ACTIONS(state) do
        v ← max(v, MIN-VALUE(RESULT(state,a),α, β))
        if v ≥ β return v
        α ← max(α, v)
    end for
end function
```

```
function MIN-VALUE(state, α, β) returns a utility value v
    if TERMINAL-TEST(state) return UTILITY(state)
    v ← ∞
    for all ACTIONS(state) do
        v ← min(v, MAX-VALUE(RESULT(state,a),α, β))
        if v ≤ α return v
        β ← min(β, v)
    end for
end function
```

References

[1] Stuart Russell and Peter Norvig.
Artificial Intelligence: A Modern Approach.
Prentice Hall, 3rd edition, 2010.