

# Game theory

## Nim game, Minimax, Alpha-Beta pruning

## beOI Training

OLYMPIADE BELGE D'INFORMATIQUE  
BELGISCHE INFORMATICA-OLYMPIADE

February 17, 2017

# Table of Contents

Basics and decision trees

Subtraction game

Nim game

Minimax algorithm

Alpha-Beta pruning

# What is game theory?

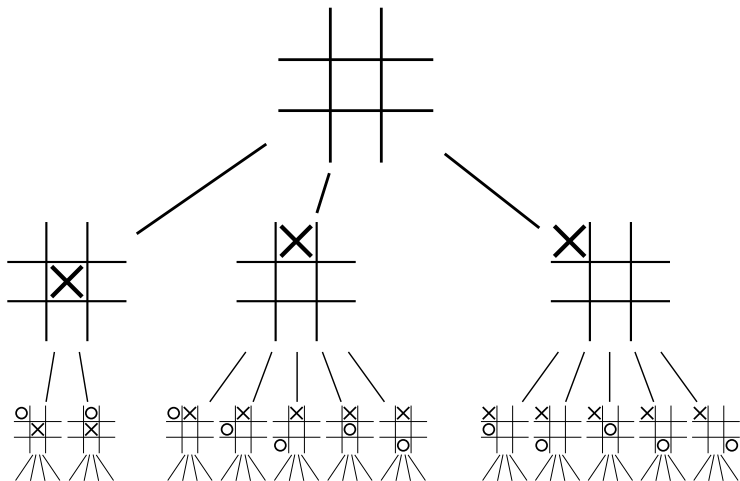
Modelling strategic situations:

- ▶ with conflict or cooperation
  - ▶ chess, football, pictionary, ...
- ▶ decisions based on personal goals
  - ▶ beating the opponent, maximizing points, ...
- ▶ influenced by the choice patterns of other players
  - ▶ if someone plays predictably, you can use it against them
- ▶ might involve randomness
  - ▶ dice rolls, card draws, ...

Goal: compute choices, optimal strategies, expected gains

# Decision trees

A useful tool to examine decisions and their consequences.



# Utility vectors and zero-sum games

Utility vectors:

- ▶ define the “gains” of an end state
- ▶ one entry per player:  $(2, 3, -2)$ ,  $(0, 7)$ , ...
- ▶ determine the choices of players
  - ▶ player 1 will choose  $(\mathbf{3}, 5)$  over  $(2, 3)$
- ▶ but not always!
  - ▶ should player 2 choose  $(0, \mathbf{6}, 5)$  or  $(2, \mathbf{6}, 3)$ ?

Main focus: two-player zero-sum games:

- ▶ only two players: no complicated interactions
- ▶ zero-sum: our benefits are the opponent's losses
  - ▶  $(5, -5)$ ,  $(-3, 3)$ ,  $(0, 0)$ , ... (or just  $5, -3, 0$ )
- ▶ the value of a choice is always well-defined

# Table of Contents

Basics and decision trees

Subtraction game

Nim game

Minimax algorithm

Alpha-Beta pruning

# Subtraction game: rules

“Jeu des alumettes” in French, ??? in Dutch

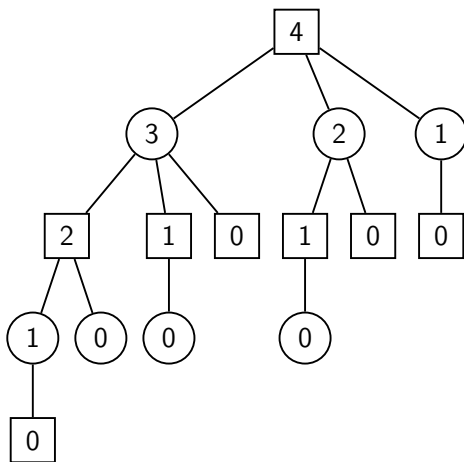
- ▶ At the beginning,  $n$  matches on the table
- ▶ Two players play in alternance
- ▶ At each turn they remove  $1, 2, \dots, k$  matches
- ▶ The player that takes the last match wins

Example with  $k = 3$ :

- ▶ At the beginning,  $n = 8$  matches.
- ▶ Player A takes 2; remaining: 6.
- ▶ Player B takes 1; remaining: 4.
- ▶ Player A takes 3; remaining: 1.
- ▶ Player B takes 1 and wins.

# Subtraction game: decision tree

$\square ?$  = A plays       $\circ ?$  = B plays

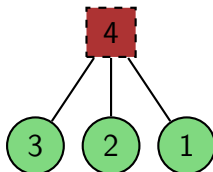
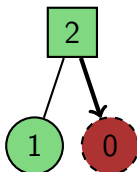
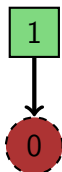




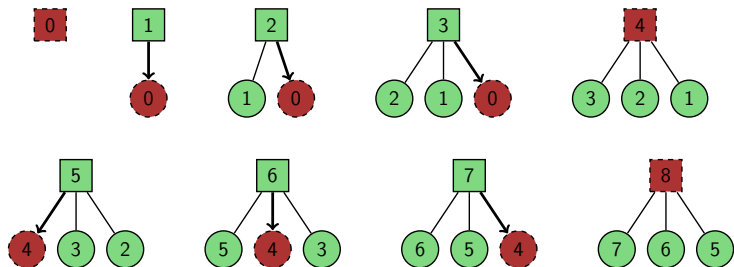
# Winning states

- ▶ A winning state is a state such that the *current player* has winning strategy
- ▶ A state is winning iff one of the moves leads to a losing state for the opponent
- ▶ If all the next states are winning for the opponent, then we can't win!

= winning      = losing      = choice



# Subtraction game: winning states



Observations:

- ▶ All multiples of  $4 = k + 1$  are losing states
- ▶ All other states are winning because they point to a multiple of  $k + 1$

# Subtraction game: optimal play

If the number of matches is a multiple of  $k + 1$ :

- ▶ all moves are bad;
- ▶ if the opponent plays perfectly you will certainly lose.

Otherwise:

- ▶ only one move is correct;
- ▶ remove matches *so that you leave* a multiple of  $k + 1$ ;
- ▶ if you play perfectly you will certainly win.

Examples of perfect moves (for  $k = 3$ ):

- ▶  $1 \rightarrow 0$     $2 \rightarrow 0$     $3 \rightarrow 0$
- ▶  $5 \rightarrow 5$     $6 \rightarrow 4$     $7 \rightarrow 4$
- ▶ ...

# Table of Contents

Basics and decision trees

Subtraction game

Nim game

Minimax algorithm

Alpha-Beta pruning

# Nim game: rules

Similar to subtraction game but:

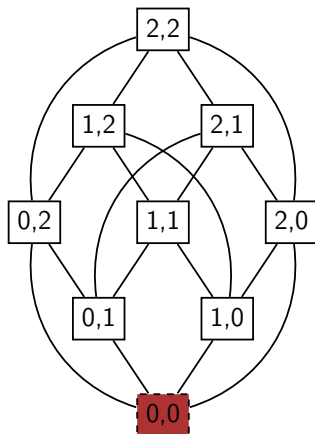
- ▶ More than one heap: sizes  $(2, 3)$ ,  $(2, 1, 9)$ .
- ▶ You can remove as many matches as you want but *from a single heap*.
- ▶ The player that empties the last heap wins.

Example with three heaps:

- ▶ At the beginning, there are three heaps:  $(3, 5, 4)$ .
- ▶ Player A takes 2 on heap 1; remaining:  $(1, 5, 4)$ .
- ▶ Player B empties heap 2; remaining:  $(1, 0, 4)$ .
- ▶ Player A takes 3 on heap 3; remaining:  $(1, 0, 1)$ .
- ▶ Player B empties heap 1; remaining:  $(0, 0, 1)$ .
- ▶ Player A empties heap 3 and wins.

## Nim game: two heaps

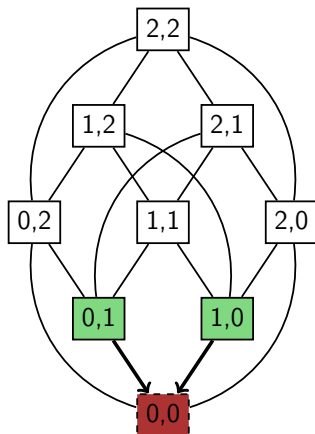
Compute the winning/losing states in a bottom-up way.



(We make no distinction between players anymore.)

## Nim game: two heaps

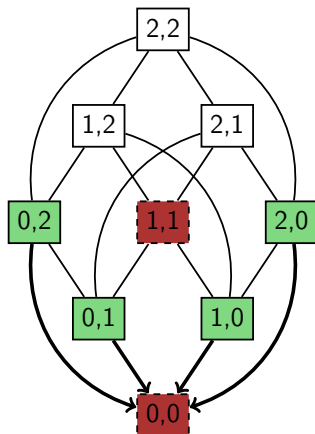
Compute the winning/losing states in a bottom-up way.



(We make no distinction between players anymore.)

## Nim game: two heaps

Compute the winning/losing states in a bottom-up way.

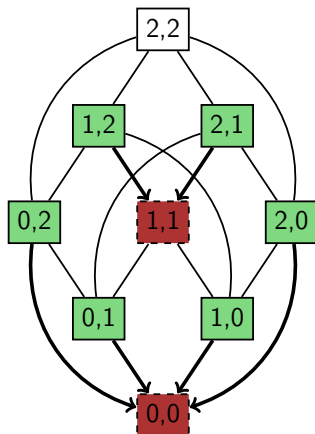


(We make no distinction between players anymore.)



## Nim game: two heaps

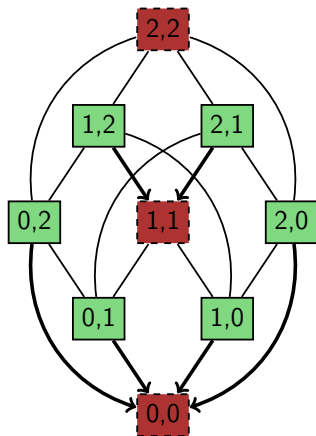
Compute the winning/losing states in a bottom-up way.



(We make no distinction between players anymore.)

## Nim game: two heaps

Compute the winning/losing states in a bottom-up way.



(We make no distinction between players anymore.)

# Nim game: the search of a losing criterion

For two heaps, we can see that losing states have heaps of equal size:

- ▶ if they are  $\neq$ , you can make them  $=$ ;
  - ▶  $(5, 2) \rightarrow (2, 2)$     $(3, 7) \rightarrow (3, 3)$     $(4, 0) \rightarrow (0, 0)$
- ▶ if they are  $=$ , they will always become  $\neq$ ;
  - ▶  $(2, 2) \rightarrow (1, 2), (0, 2), (2, 1), (2, 0)$
- ▶ the game always ends at  $(0, 0)$ .

What about more heaps? Any idea?

Examples of losing states:  $(0, 0, 0)$ ,  $(0, 1, 1)$ ,  $(1, 2, 3)$ ,  $(1, 4, 5)$ ,  
 $(1, 6, 7)$ ,  $(2, 4, 6)$ ,  $(3, 5, 6)$ ,  $(1, 1, 3, 3)$ ,  $(1, 3, 5, 7)$ ,  $(2, 3, 6, 7)$ ,  
 $(4, 5, 6, 7)$ ,  $(3, 3, 4, 4)$ , ...

# Nim game: key invariant

A state  $(n_1, \dots, n_l)$  is losing iff  $n_1 \oplus \dots \oplus n_l = 0$ . **WHAT?**

For example:  $(3, 5, 6)$  is losing because

$$3_{10} \oplus 5_{10} \oplus 6_{10} = 011_2 \oplus 101_2 \oplus 110_2 = 000_2$$

We call this the *nim-sum*.

We need to prove this:

- ▶ The ending situation has a zero nim-sum (clear).
- ▶ During one move:
  - ▶ When the nim-sum is zero, we cannot keep it zero.
  - ▶ When the nim-sum is not zero, we can make it zero.

# Nim game: losing situation

*“When the nim-sum is zero, we cannot keep it zero.”*

- ▶ Before the move:  $n_1 \oplus \cdots \oplus n_l = 0$
- ▶ We take matches from heap  $i$ :  $n_i \rightarrow n'_i$
- ▶ After the move:

$$\begin{aligned} & n_1 \oplus \cdots \oplus n'_i \oplus \cdots \oplus n_l \\ &= n_1 \oplus \cdots \oplus (n_i \oplus n_i \oplus n'_i) \oplus \cdots \oplus n_l \\ &= (n_1 \oplus \cdots \oplus n_i \oplus \cdots \oplus n_l) \oplus (n_i \oplus n'_i) \\ &= n_i \oplus n'_i \\ &\neq 0 \end{aligned}$$

# Nim game: winning situation

*“When the nim-sum is not zero, we can make it zero.”*

- ▶ Before the move:  $n_1 \oplus \cdots \oplus n_l = s \neq 0$
- ▶ If we can replace some  $n_i$  with  $n_i \oplus s$ , we win!
- ▶ But it has to verify  $n_i \oplus s < n_i$  (we cannot add matches).
- ▶ Let  $1 \ll j$  be the largest bit in  $s$ . We just have to take  $n_i$  that contains the bit  $1 \ll j$ . This implies  $n_i \oplus s < n_i$ .
- ▶ Example: state (7, 8, 13).

$$7_{10} \oplus 8_{10} \oplus 13_{10} = 0111_2 \oplus 1000_2 \oplus 1101_2 = 0010_2$$

Only  $7_{10} = 0111_2$  has the required bit.

$$0111_2 \oplus 0010_2 = 0101_2 = 5_{10} < 7_{10}$$

Next state: (5, 8, 13).

# Win/lose games: general conclusions

Frequent properties of simple win-or-lose games:

- ▶ Few losing states, many winning states
  - ▶ because one losing child is enough
- ▶ Losing states defined by a specific property
  - ▶  $n$  divisible by  $k + 1$ ,  $n_1 \oplus \dots \oplus n_l = 0$ , ...
- ▶ The property can always be *restored* when broken
  - ▶ this defines the winning strategy
- ▶ But it can never be *kept* by a move
  - ▶ the losing player can never turn the game around

Exercise: solve this subtraction/Nim mashup.

- ▶ Several heaps with sizes  $(n_1, \dots, n_l)$ .
- ▶ You can take  $1, 2, \dots, k$  matches from a single heap.

# Table of Contents

Basics and decision trees

Subtraction game

Nim game

**Minimax algorithm**

Alpha-Beta pruning

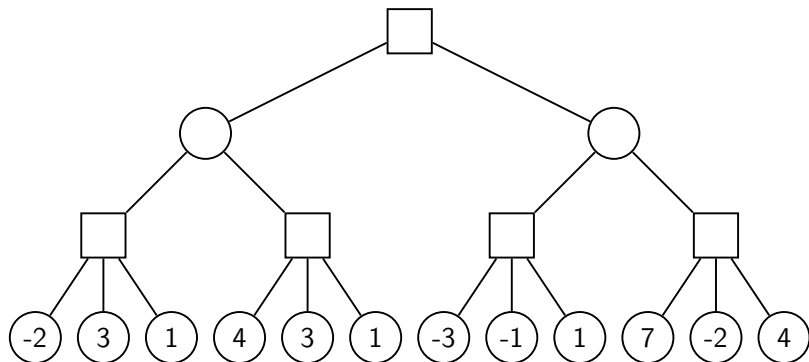


# Minimax principles

- ▶ So far we've only considered win/lose games
  - ▶ utilities:  $+1 = \text{win}$ ,  $-1 = \text{lose}$ .
- ▶ What about games with scores?
  - ▶ example: utility is the difference of the scores.
- ▶ Each player will take the choice that
  - ▶ maximizes his utility
  - ▶ or minimizes the utility of his opponent (zero-sum)
- ▶ Let's fix player A as reference
  - ▶ player A will always maximize the value
  - ▶ player B will always minimize the value

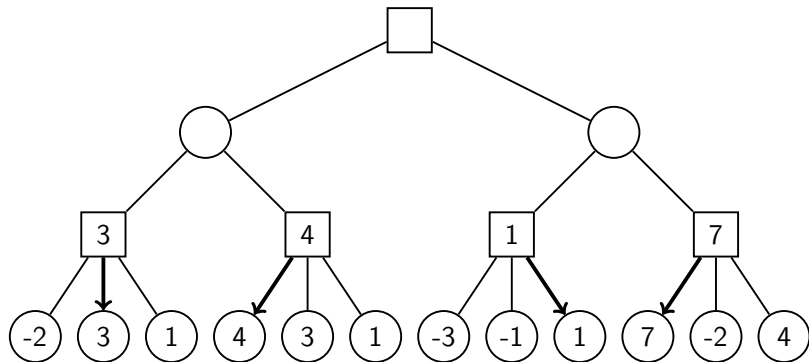
# Minimax example

$\square ?$  = A plays  $\Rightarrow$  maximize       $\circ ?$  = B plays  $\Rightarrow$  minimize



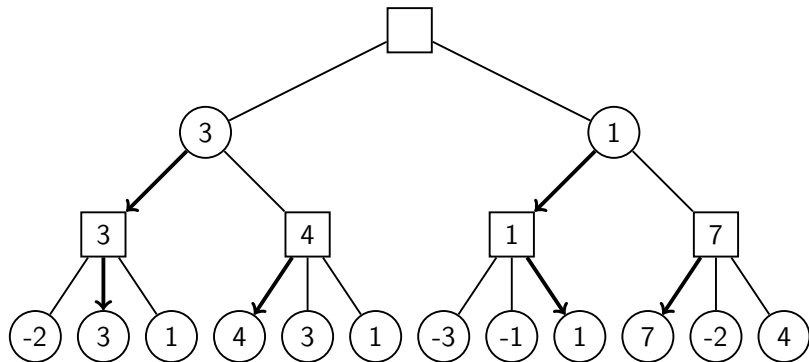
# Minimax example

$\square ?$  = A plays  $\Rightarrow$  maximize       $\circ ?$  = B plays  $\Rightarrow$  minimize



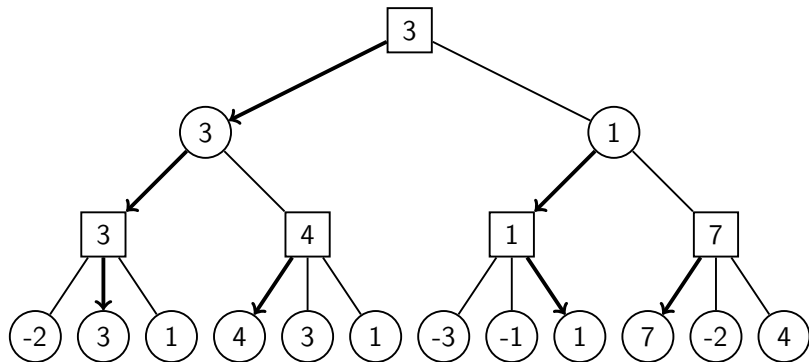
# Minimax example

$\square ?$  = A plays  $\Rightarrow$  maximize       $\circ ?$  = B plays  $\Rightarrow$  minimize



# Minimax example

$\square ?$  = A plays  $\Rightarrow$  maximize       $\circ ?$  = B plays  $\Rightarrow$  minimize



# Minimax implementation

## Implementation 1: recursive, DFS-style

```
int minimax(state u) {  
    if (terminal(u)) // the game has ended  
        return score(u);  
    int best = -INF;  
    for (state v : nextStates(u)) // possible moves  
        best = max(best, -minimax(v));  
    return best;  
}
```

- ▶ Doesn't separate the “min” and “max” steps
- ▶ Just takes the opposite of the opponent's score

**Complexity:**  $O(b^d)$ , if  $d$  is depth and  $b$  branches every time.

# Minimax with DP

## Implementation 2: add DP memoization

```
map<state, int> dp; // make it an array if possible

int minimax(state u) {
    if (terminal(u))
        return score(u);
    if (dp.count(u)) // already computed before
        return dp[u];
    // [...] find best move
    return (dp[u] = best); // don't forget to save
}
```

**Complexity:**  $O(s)$ , where  $s$  is the number of possible states.

# Table of Contents

Basics and decision trees

Subtraction game

Nim game

Minimax algorithm

Alpha-Beta pruning



# Very large search spaces

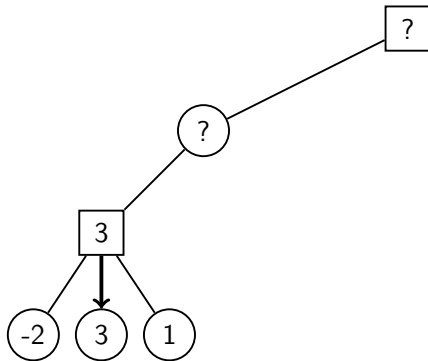
Sometimes there are just too many states to explore!  
(chess, Go, draughts, ...)

- ▶ Solution 1: cut off the tree and *evaluate* the situation
  - ▶ e.g. cut at depth 4, evaluate, and run minimax
  - ▶ evaluation based on heuristics (imperfect estimations)
  - ▶ inexact result  $\Rightarrow$  not okay for us
- ▶ Solution 2: eliminate states without changing the result
  - ▶ prune complete parts of the trees
  - ▶ *prove* that the unvisited states are not part of the optimal play

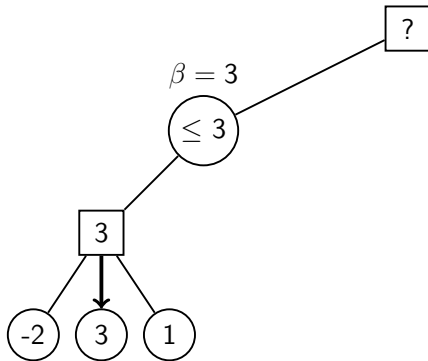
# Alpha-beta definition

- ▶ Let's assume we compute A's utility
  - ▶ A wants to maximize the score
  - ▶ B wants to minimize the score
- ▶ During the minimax search, we maintain two parameters:
  - ▶  $\alpha$  = maximum score that A is assured of
  - ▶  $\beta$  = minimum score that B is assured of
- ▶ In other words,  $\alpha$  and  $\beta$  are such that:
  - ▶ we know the final result is in  $[\alpha, \beta]$
  - ▶  $\alpha$  is as big as possible
  - ▶  $\beta$  is as small as possible

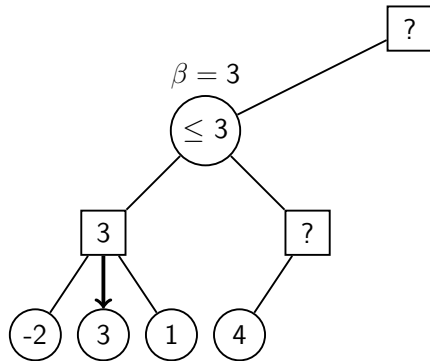
## Alpha-beta example



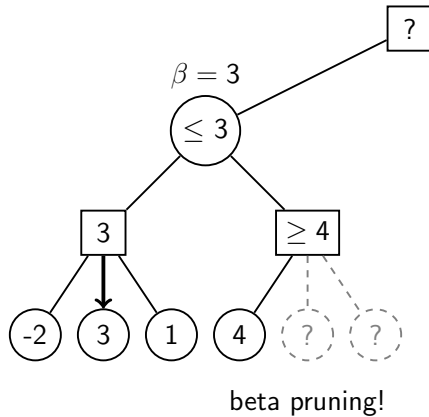
# Alpha-beta example



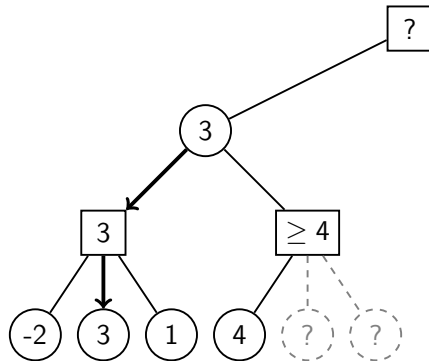
# Alpha-beta example



# Alpha-beta example

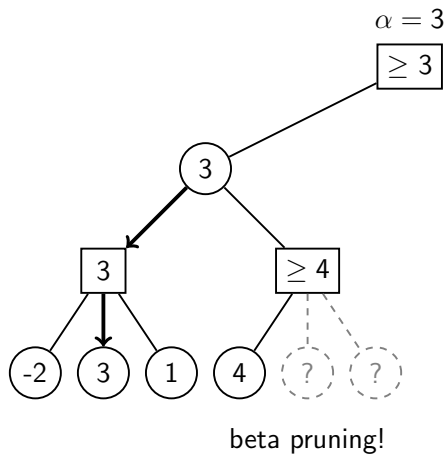


# Alpha-beta example



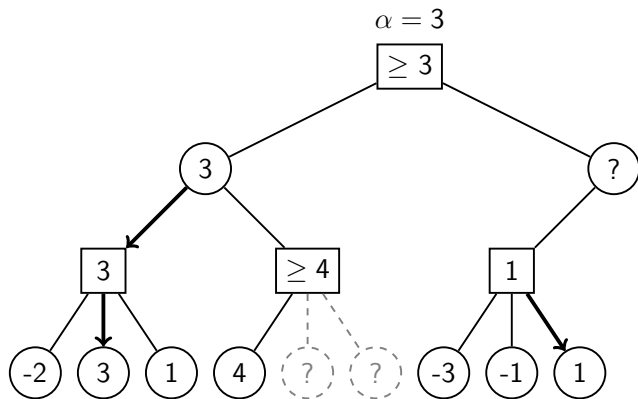
beta pruning!

# Alpha-beta example



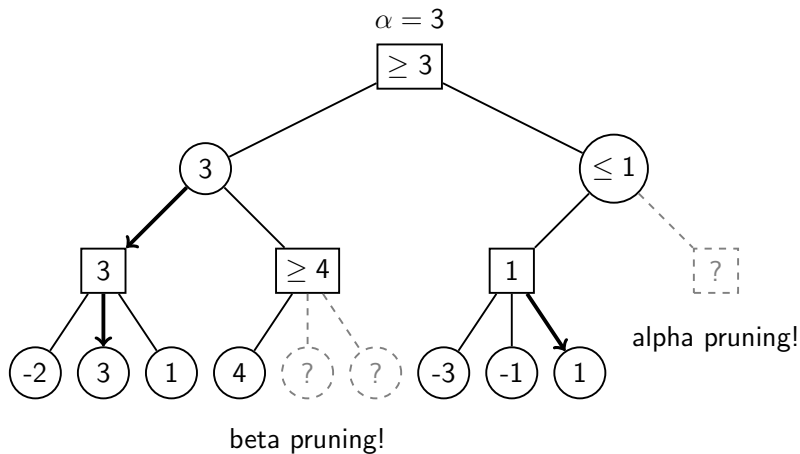


## Alpha-beta example

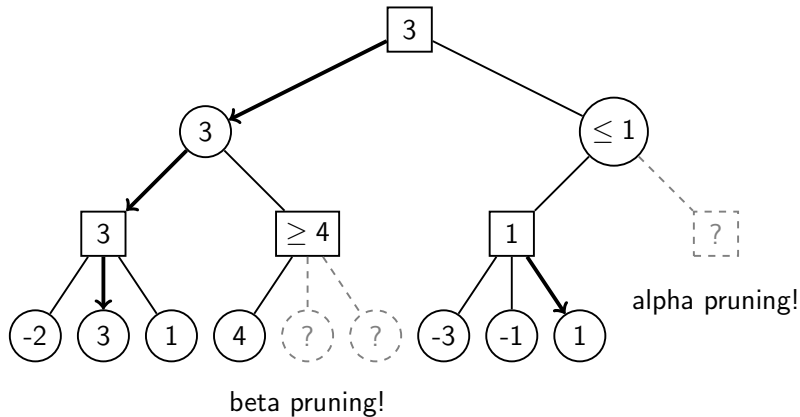


beta pruning!

# Alpha-beta example



# Alpha-beta example



# Alpha-beta implementation

```
int minimax(state u, int alpha, int beta) {  
    if (terminal(u))  
        return score(u);  
    int best = -INF;  
    for (state v : nextStates(u)) {  
        best = max(best,  
                    -minimax(v, -beta, -alpha));  
        alpha = max(alpha, best);  
        if (alpha >= beta) break; // pruning  
    }  
    return best;  
}
```

- ▶ Interval  $[\alpha, \beta]$  becomes  $[-\beta, -\alpha]$  when switching players
- ▶ Cutoff when current best is  $\geq \beta$ 
  - ▶ maybe there is better, but the opponent has a better choice anyway

## Sources of figures

- ▶ <https://commons.wikimedia.org/wiki/File:Tic-tac-toe-game-tree.svg>