Introduction to Artificial Intelligence

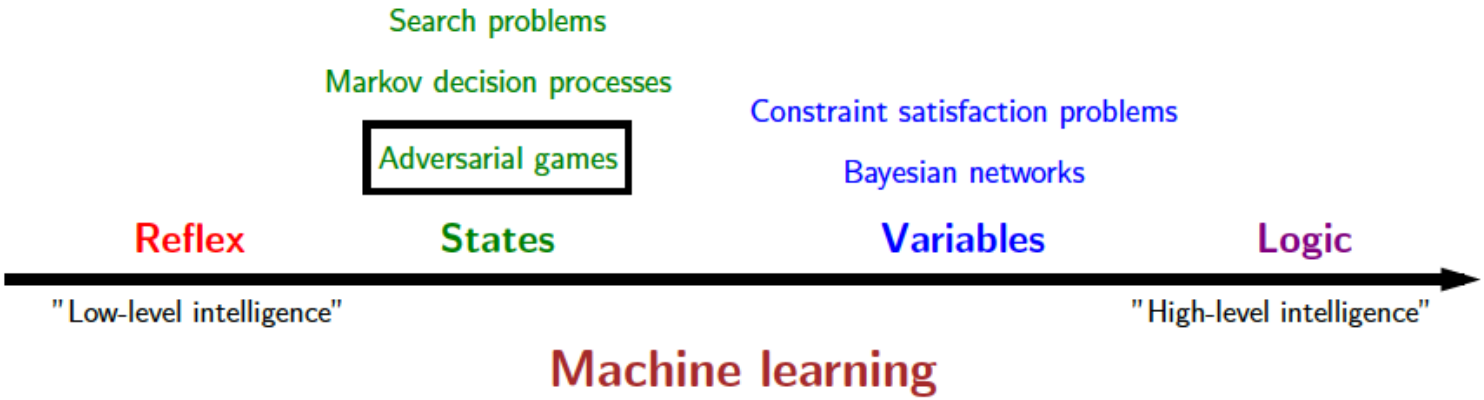
Computational games

Céline Hudelot (with Jean-Philippe Poli)

CentraleSupélec Summer School 2019
Artificial Intelligence

CentraleSupélec

Where we are !

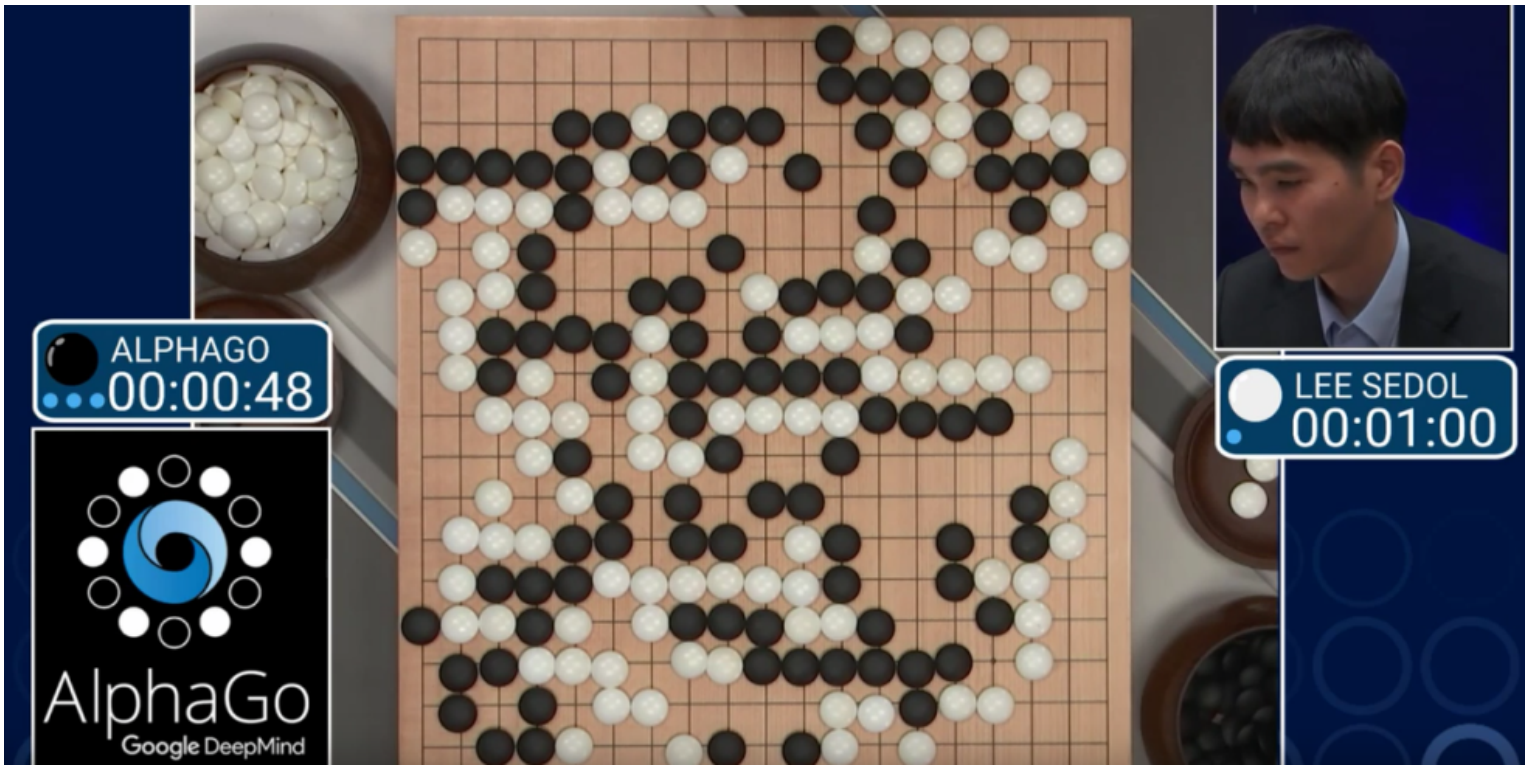

Search problems
Markov decision processes
Adversarial games
Constraint satisfaction problems
Bayesian networks
Reflex
States
Variables
Logic
"Low-level intelligence"
"High-level intelligence"
Machine learning

An important episode ! ! !

In 1997, Deeper Blue wins Kasparov, the world chess champion !

Another important episode ! ! !

In 1990, Goliath is an AI which has the level of a beginner at the Go Game. In 2016, Alpha Go wins Lee Sedol, one of the best player in the world.



Why studying games ?

A proof of intelligence ?

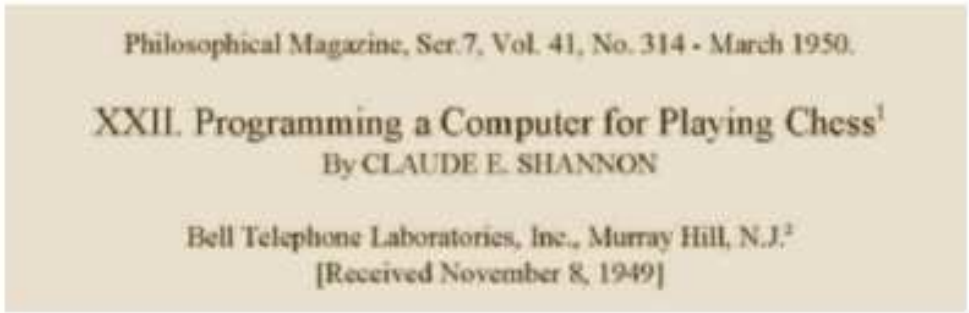
- Almost all the humans play.
- Results can be popularized easily

An AI showcase

Progress in a two player game can be measured easily.

- Games are popular to demonstrate new ideas in AI.
- Approaches are used in others fields.

History : a funding article in AI



Philosophical Magazine, Ser.7, Vol. 41, No. 314 - March 1950.

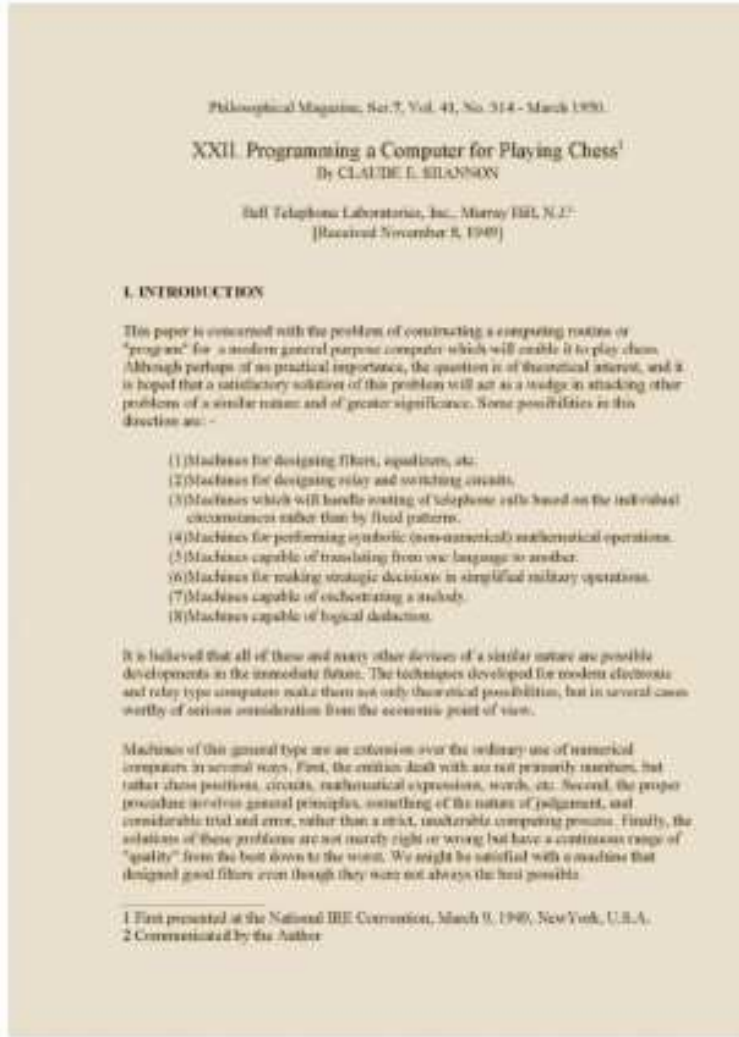
XXII. Programming a Computer for Playing Chess[1]

By CLAUDE E. SHANNON

Bell Telephone Laboratories, Inc., Murray Hill, N.J.[2]

[Received November 8, 1949]

Le **premier article d'I.A.** est un article sur les jeux (Shannon, 1950).



Philosophical Magazine, Ser.7, Vol. 41, No. 314 - March 1950.

XXII. Programming a Computer for Playing Chess[1]

By CLAUDE E. SHANNON

Bell Telephone Laboratories, Inc., Murray Hill, N.J.[2]

[Received November 8, 1949]

1. INTRODUCTION

This paper is concerned with the problem of constructing a computing routine or "program" for a modern general purpose computer which will enable it to play chess. Although perhaps of no practical importance, the question is of theoretical interest, and it is hoped that a satisfactory solution of this problem will act as a wedge in attacking other problems of a similar nature and of greater significance. Some possibilities in this direction are:-

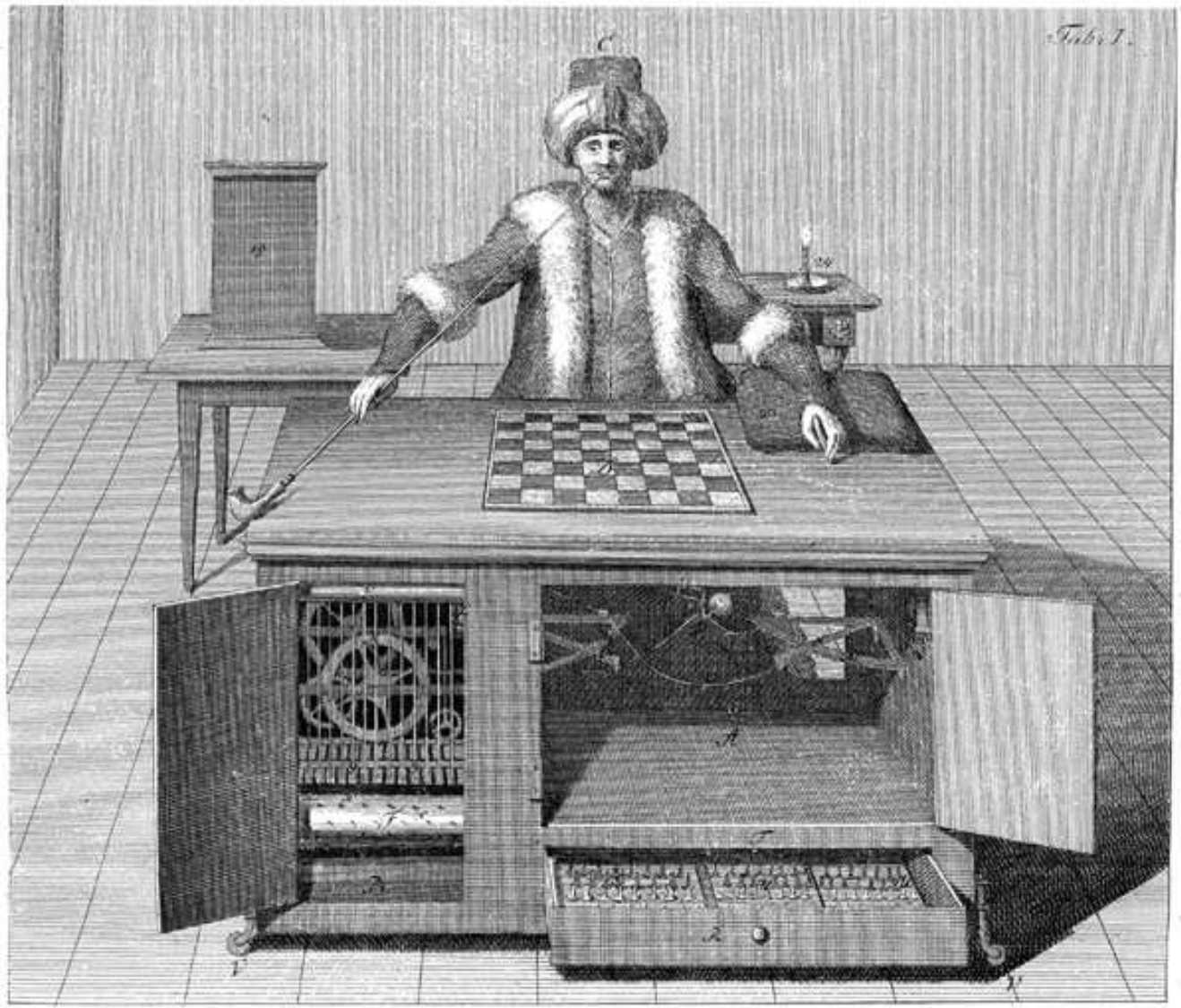
(1) Machines for designing filters, equalizers, etc.
(2) Machines for designing relay and switching circuits.
(3) Machines which will handle routing of telephone calls based on the individual circumstances rather than by fixed patterns.
(4) Machines for performing symbolic (non-numerical) mathematical operations.
(5) Machines capable of translating from one language to another.
(6) Machines for making strategic decisions in simplified military operations.
(7) Machines capable of orchestrating a melody.
(8) Machines capable of logical deduction.

It is believed that all of these and many other devices of a similar nature are possible developments in the immediate future. The techniques developed for modern electronic and relay type computers make them not only theoretical possibilities, but in several cases worthy of serious consideration from the economic point of view.

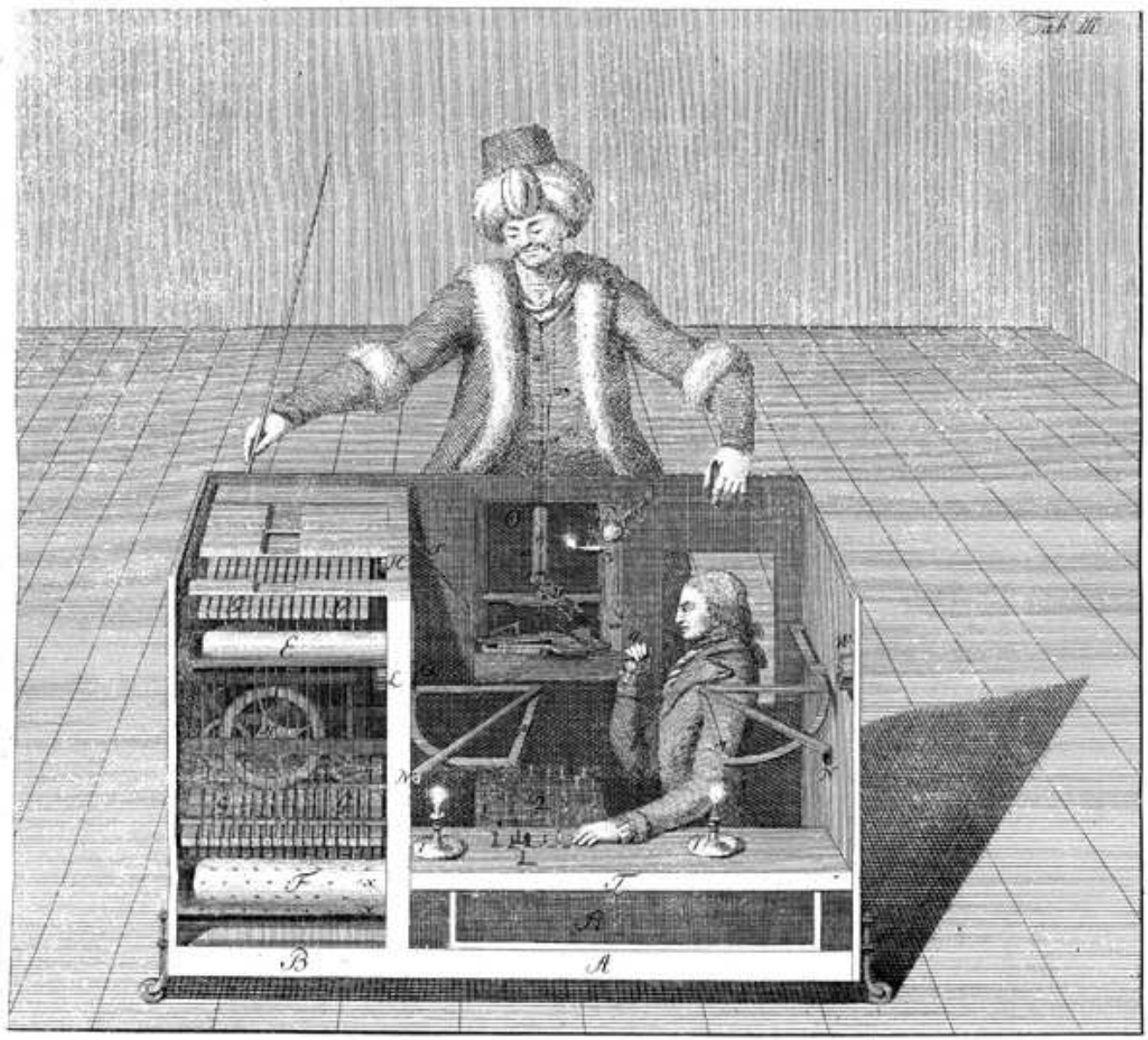
Machines of this general type are an extension over the ordinary use of numerical computers in several ways. First, the entities dealt with are not primarily numbers, but rather chess positions, circuits, mathematical expressions, words, etc. Second, the proper procedure involves general principles, something of the nature of judgment, and considerable trial and error, rather than a strict, unalterable computing process. Finally, the solutions of these problems are not merely right or wrong but have a continuous range of "quality" from the best down to the worst. We might be satisfied with a machine that designed good filters even though they were not always the best possible.

1 First presented at the National IRE Convention, March 9, 1949, New York, U.S.A.
2 Communicated by the Author.

The first machine that plays chess : 1769 ?



The first machine that plays chess : 1769 ?



Today, a machine that mimics the mechanical turk and a principle for collective working

Image of the Mechanical turk from the game Fritz 9

And for Humans ?

Game, the field of the machines ?

- Games for which computers >> Humans :
  - Checkers
  - Reversi
  - Scrabble
- Games for which computers are at the world level :
  - Chess
  - Backgammon
  - Go (e.g. Mogo, INRIA `http://www.lri.fr/~teytaud/taiwanopen2009.html`, `http://www.inria.fr/saclay/ressources/culture-scientifique/mogo`)
- Challenges :
  - Bridge

Games we will study

Characteristics

- Rules are well known and can be formalized.
- Two players
- Zero-sum games : the utility of the agent is negative the utility of the opponent
- Open games : the state of the game is fully-observed

Game problems

Characteristics

- Search problems in presence of other agents, whose utility is not generally aligned with ours - **adversarial search**)
  - Introduction of an uncertainty : opponent strategies are unknown
  - All the moves are not controlled by the computer
- Complexity : games are often too complex to tackle them by an exhaustive search. (Example : search tree in chess)
- The goal of the search is to choose the best move at each step.

Types of games

Game classification

	Deterministic	**With random**
Complete Information	Chess, checkers, reversi, go	monopoly, backgammon
Incomplete Information		poker, bridge

In this course

Characteristics

- Two players (people take turns)
- **The state of the game is fully-observed** by the two players at each step : deterministic and fully observable environment .
- The agent utility functions return zero or equal values at the end of the game.

Video games

- More graphical and realistic issues than strategic ones
- Important real time constraints
- *Cheating* to made the game realistic

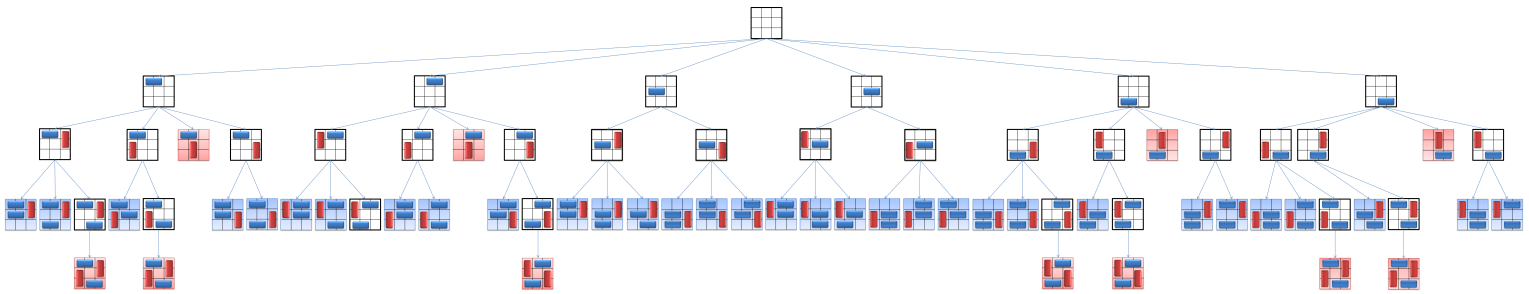
First step : formulation as a search problem

- **State** : position of the items of the game and identity of the player.
- **Initial state** ($\mathtt{s}_{start}$) : initial position of the items of the game and identity of the player that begins
- **Actions** ($\mathtt{Actions}(s)$) : set of possible actions from the state s.
- **Successor function** ($\mathtt{Succ}(s, a)$) : function that gives the next step resulting from the action a (chosen action) on state s.
- **End state** ($\mathtt{IsEnd}(s)$) : says if state s is an end state or not : one of the player has win or equality.
- **Utility function** ($\mathtt{utility}(s)$) : returns a value that evaluates the end states, i.e. utility of the agent for an end state s : $+1$ (win), -1 (lose) et 0 (equality)

Game tree

Tree of the game possible states :

- Each node is a decision point for a player.
- Each root-to-leaf path is a possible outcome of the game.



We will use tree-based algorithms (i.e. backtracking...)

Definitions

Game tree

Tree that consists in nodes representing the possibilities in a two-player game. There is an alternation between the possibilities of the two-players.

Turn

The different levels of the tree correspond to the different turns.

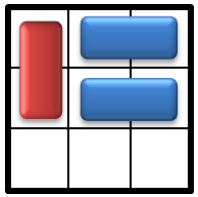
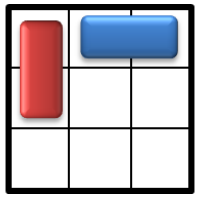
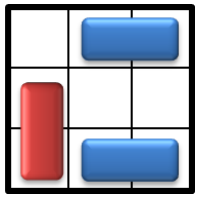
End Nodes

End of the game : winning configuration, losing configuration or equal configuration.

Color dominoes

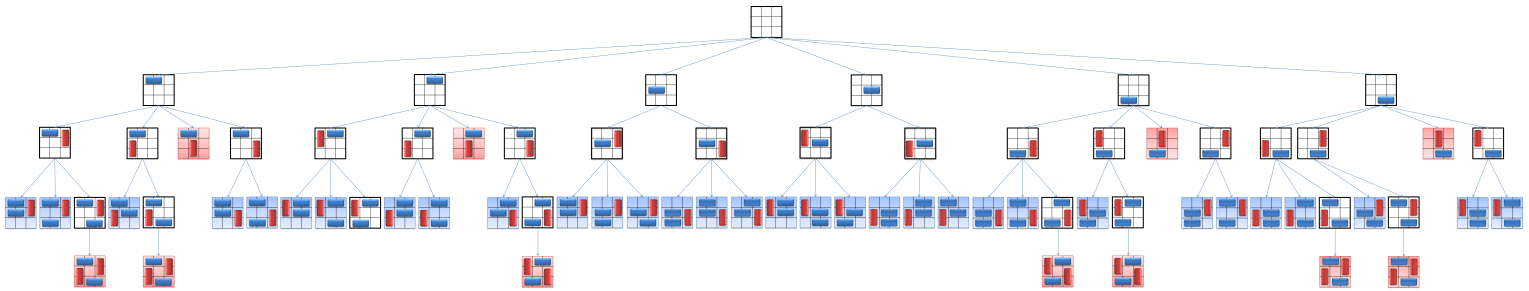
Principle

On a grid of size nxn, we have to be the last to pose our domino 2x1.
Player 1 plays on rows and player 2 on columns.

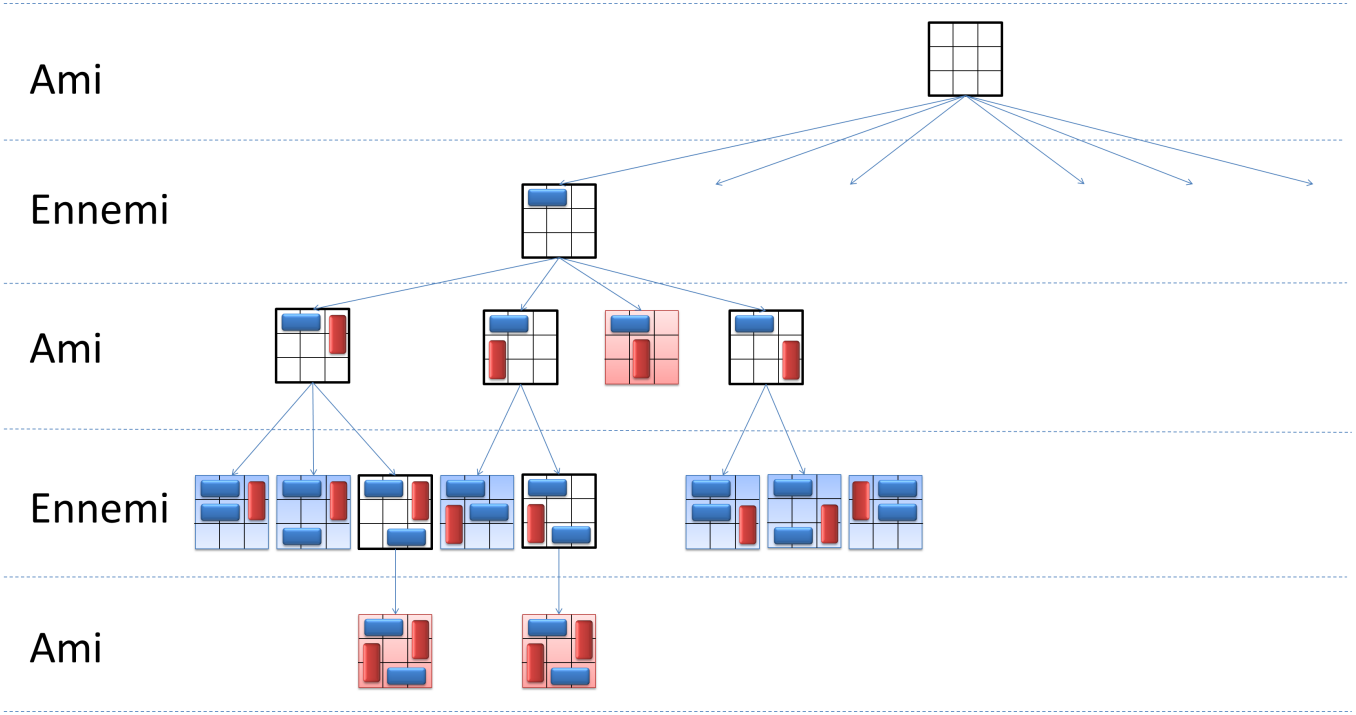




Exploration of the search tree

- The simulation of all the possible game shots from the initial state enables to build the game tree.



Exploration of the search tree

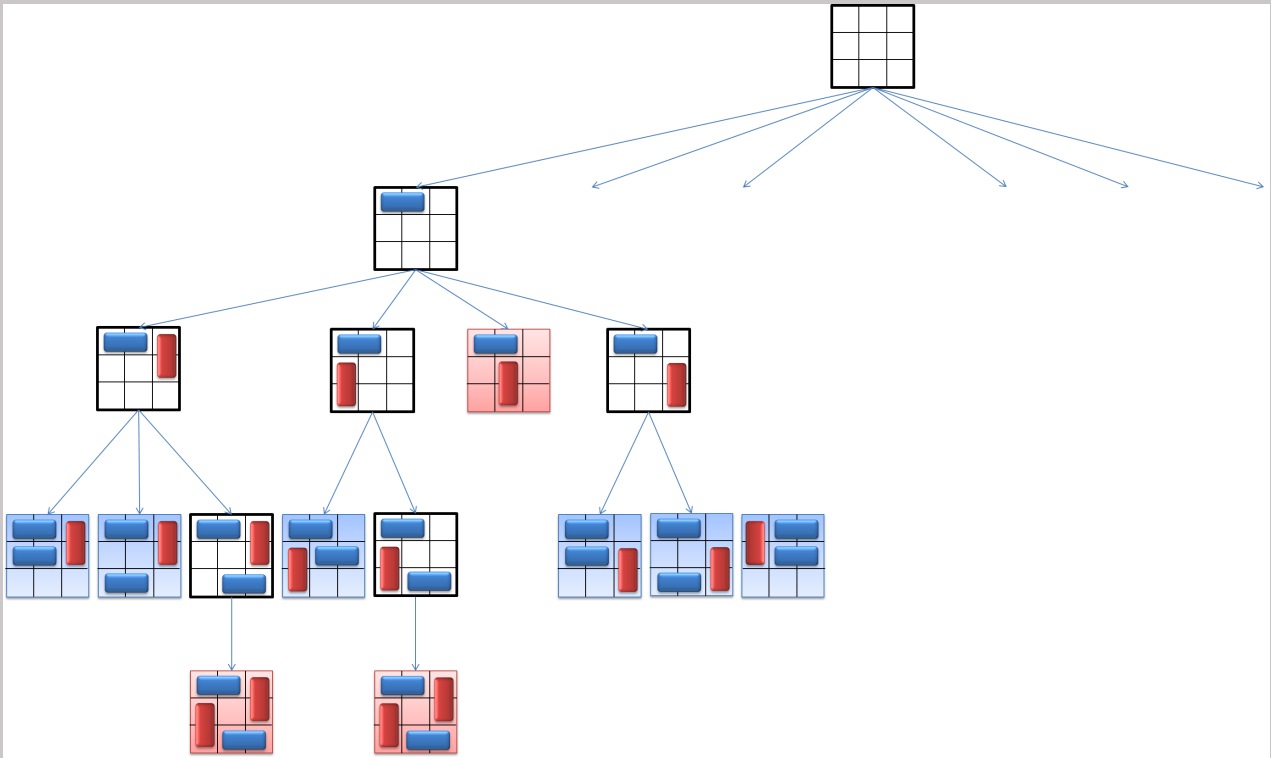


Exploration of the search tree

Breath-first traversal of the tree.

Exploration of the search tree

Complete exploration of the search tree



For a grid of size n

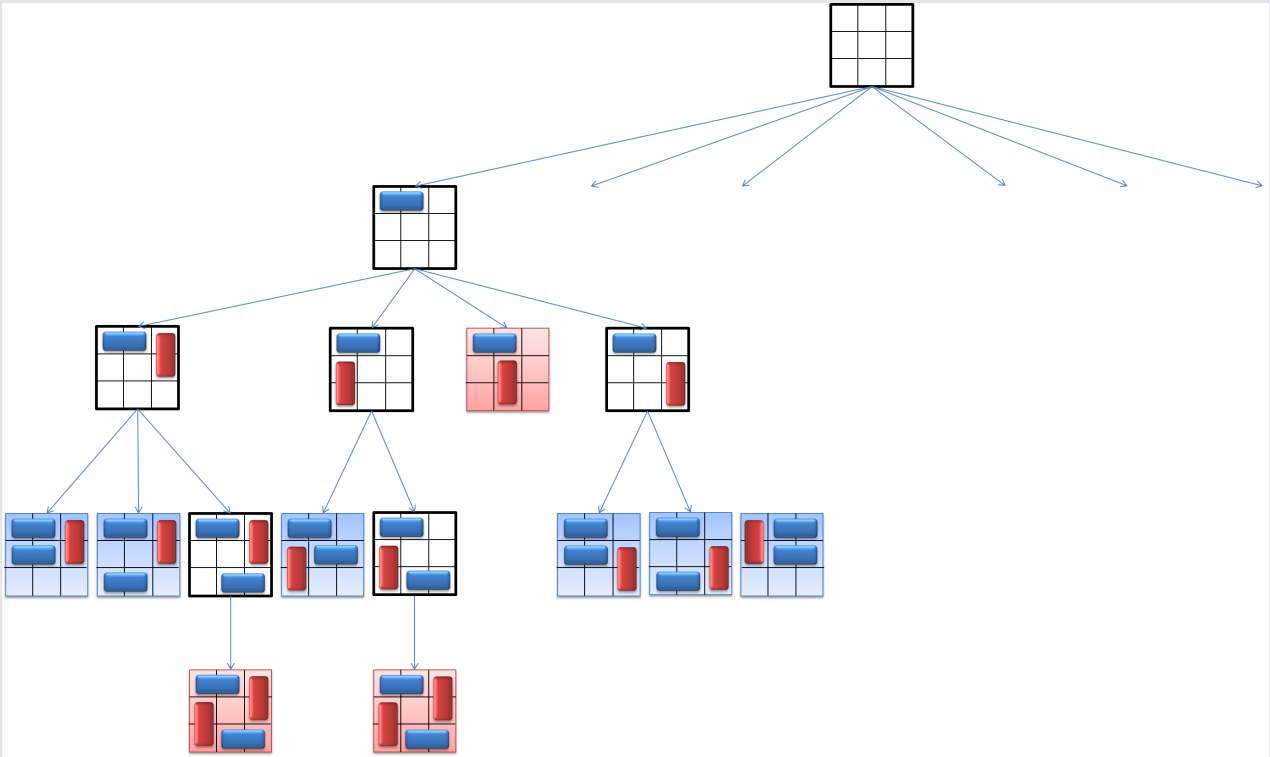
Combinatorial issue for size > 6. How improves the search ?

Exploration of the search tree : improvments

Do not explore common sub-graphs

- Idea : in some game, a same state can be reached by several paths.
- Symetric states can be removed

Dominoes



Exploration of the search tree : improvments

Do not explore common sub-graphs : some issues

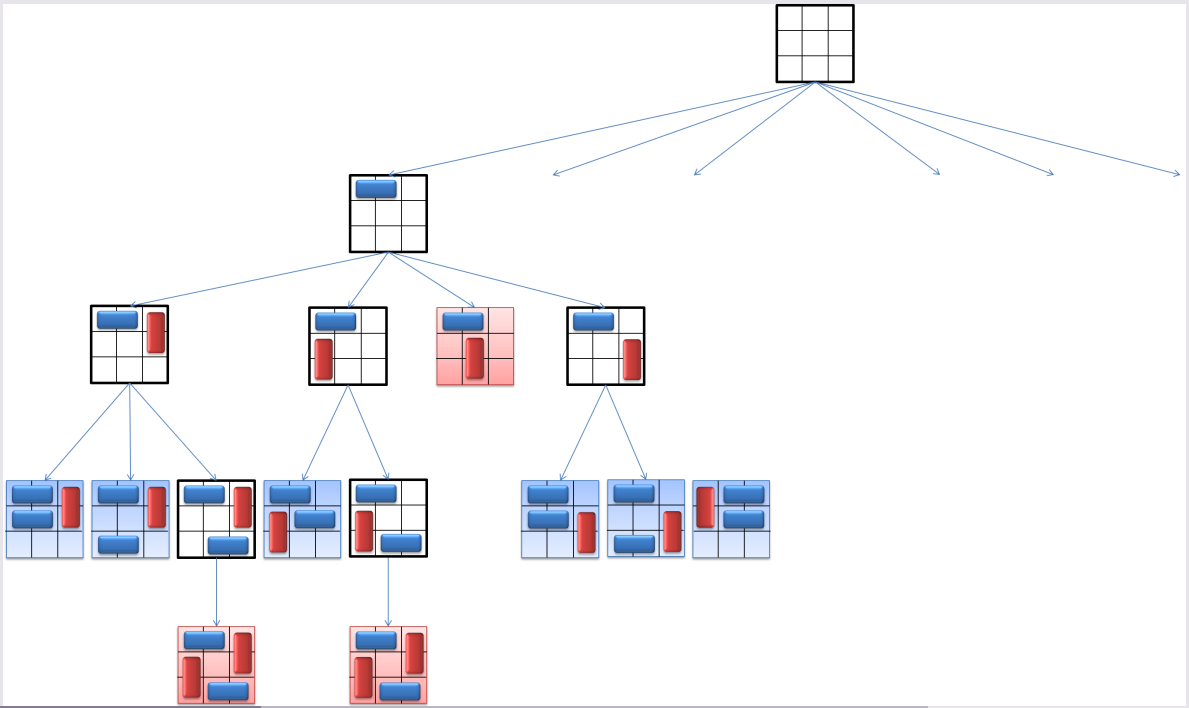
- Explored nodes have to be found.
- Nodes have to be kept in memory.
- Many time is spent in checking nodes.

Exploration of the search tree : improvments

Pruning

- Idea : If a branch is successful, do not extend the neighborhood branches.
- If a branch leads to a defeat, do not extend the neighborhood branches.

Dominoes



Exploration of the search tree : improvments

Pruning

- Find a winning strategy
- Enable the pruning of parts of the tree
- but it is mandatory to go until the leaves of the tree (not tractable in practice)

Introduction of heuristics

Idea

- The game tree will be developed until a depth p. The leaves are not necessary end states.
- Node and positions have to be evaluated.The information *lost* or *win* is not sufficient.

Definition

Heuristics are evaluation functions that map a position of the game to a real value.

Dominoes

Example

- The search tree is extended until a given depth.
- The leaf position is evaluated.
- The leaf information is propagated into the tree.

Introduction of heuristics

New problem

- Find the branch that maximizes the heuristic values after the p next shots.
- The two players follow the heuristics

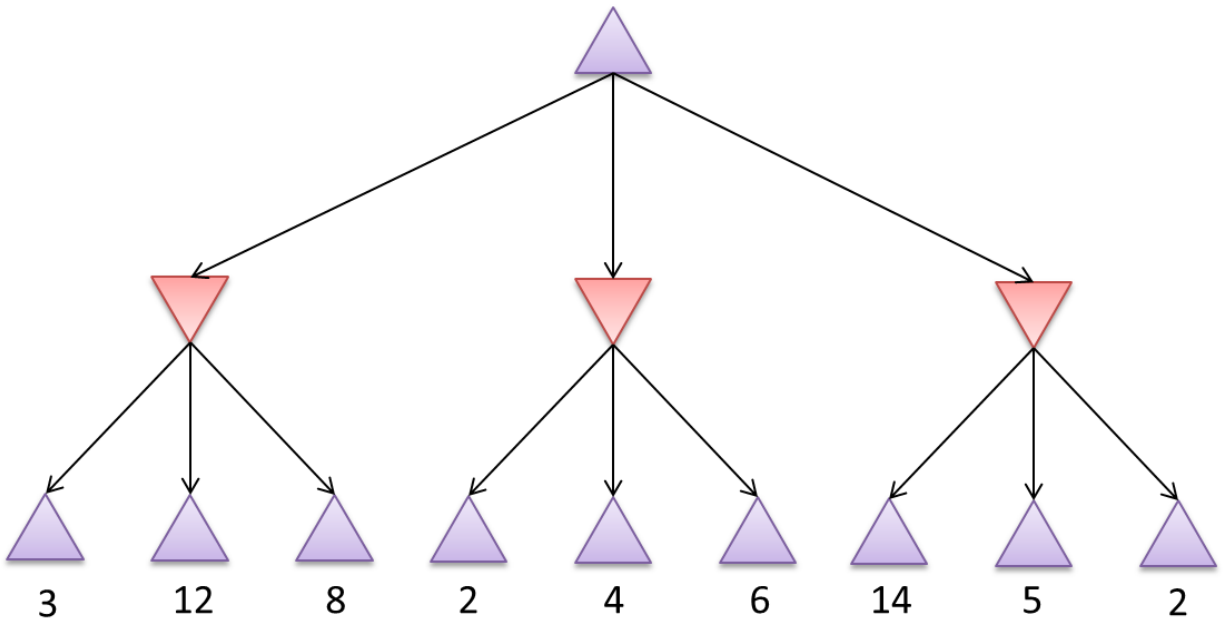
How to propagate the best value to the root ?

MiniMax Algorithm

Principle

- One of the player is named Max. The utility values are expressed according to Max.
- The principle of MiniMax Algorithm is to extend completely the search tree, to evaluate each leaf with the utility function and then to propagate the value with the hypothesis that each player wants to win (each player chooses the best shot).
  - Minimum value at the node MIN
  - Maximum value at the node MAX

MiniMax Algorithm : principle



Principle

- At the initial state, Max can choose three actions
- Then, Min can choose between tree actions that depend of the action of Max in the previous shot.

MiniMax Algorithm

Minimax

Utility value at the end of the game if the two players are playing optimally.

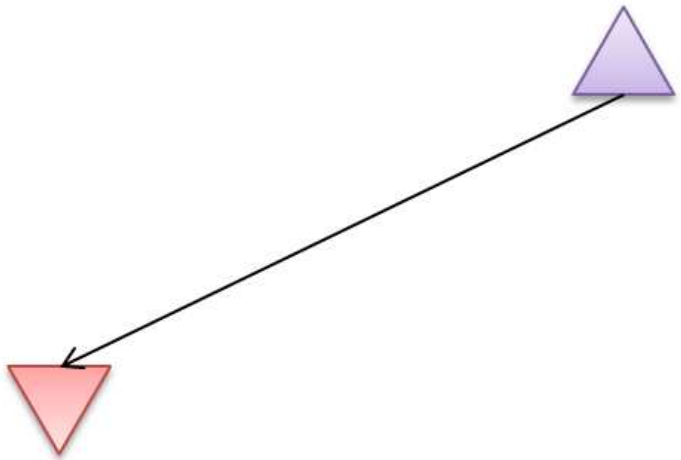
Optimal shot

- Max will go to the state (or node) that has a maximal minimax
- Min will go to the state (or node) that has a minimal minimax.

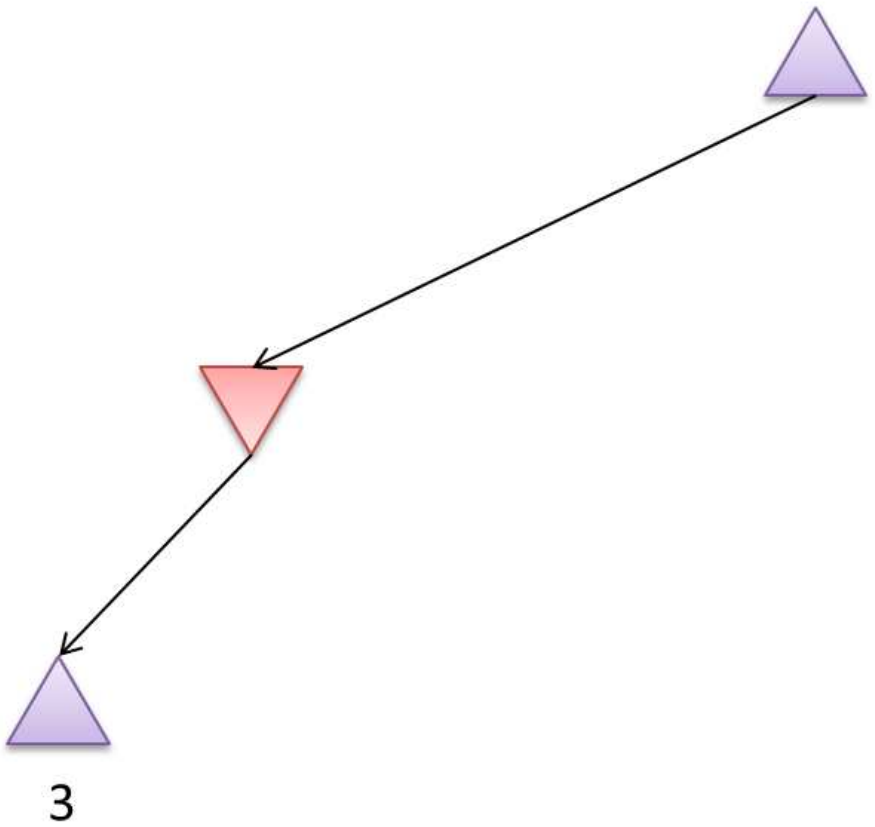
From the leaves, the minimax value of each node can be computed.

MiniMax : step by step

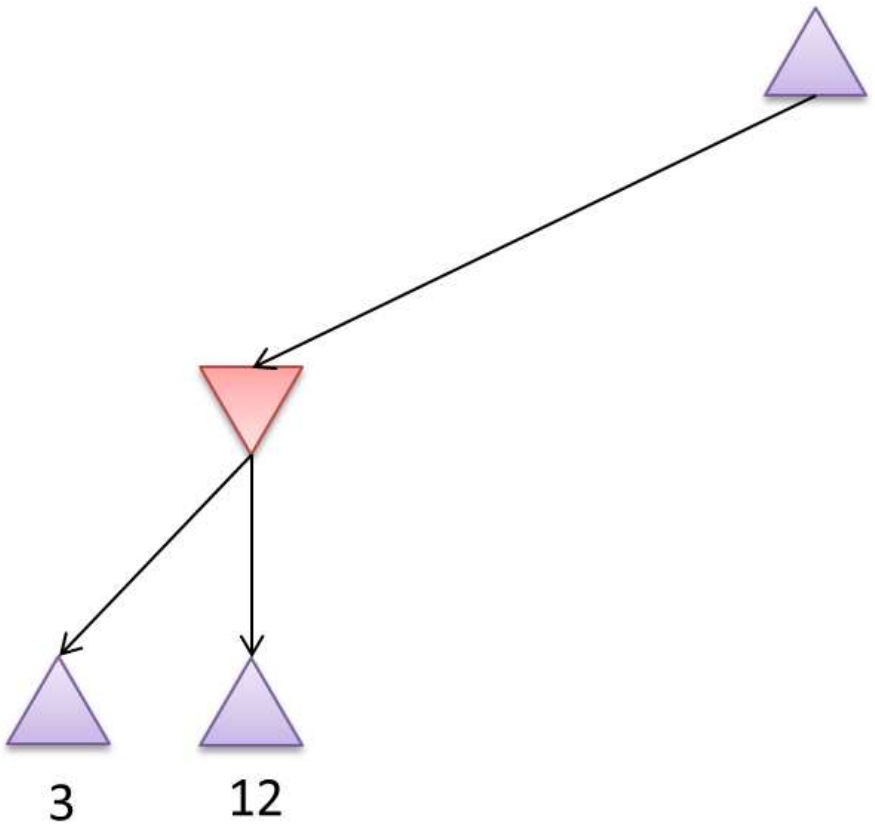
MiniMax : step by step



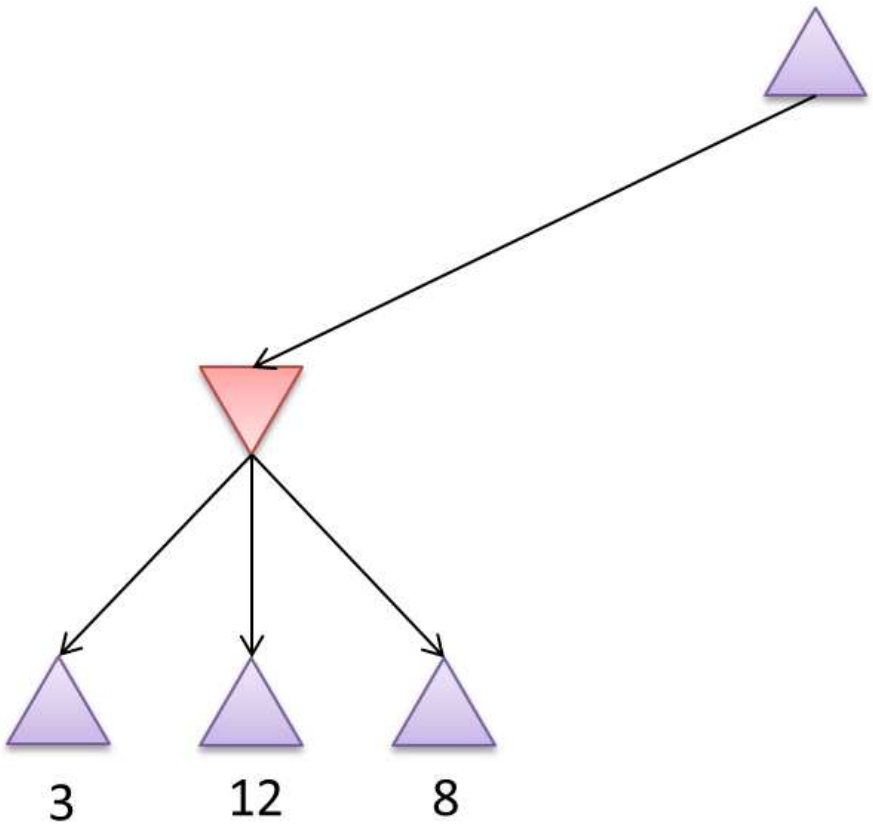
MiniMax : step by step



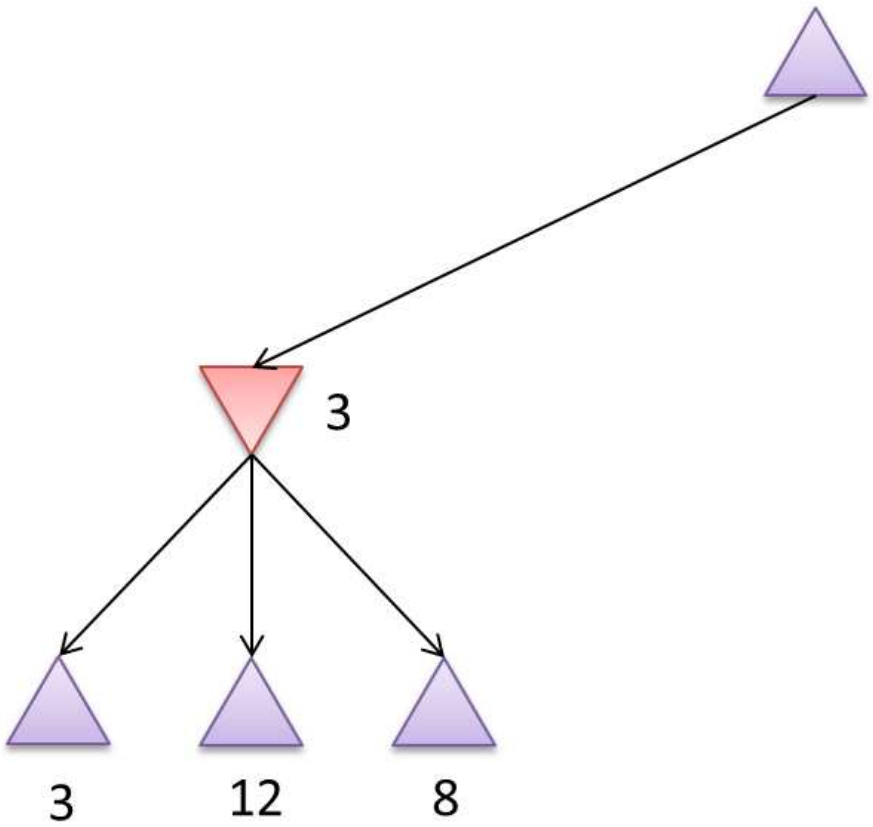
MiniMax : step by step



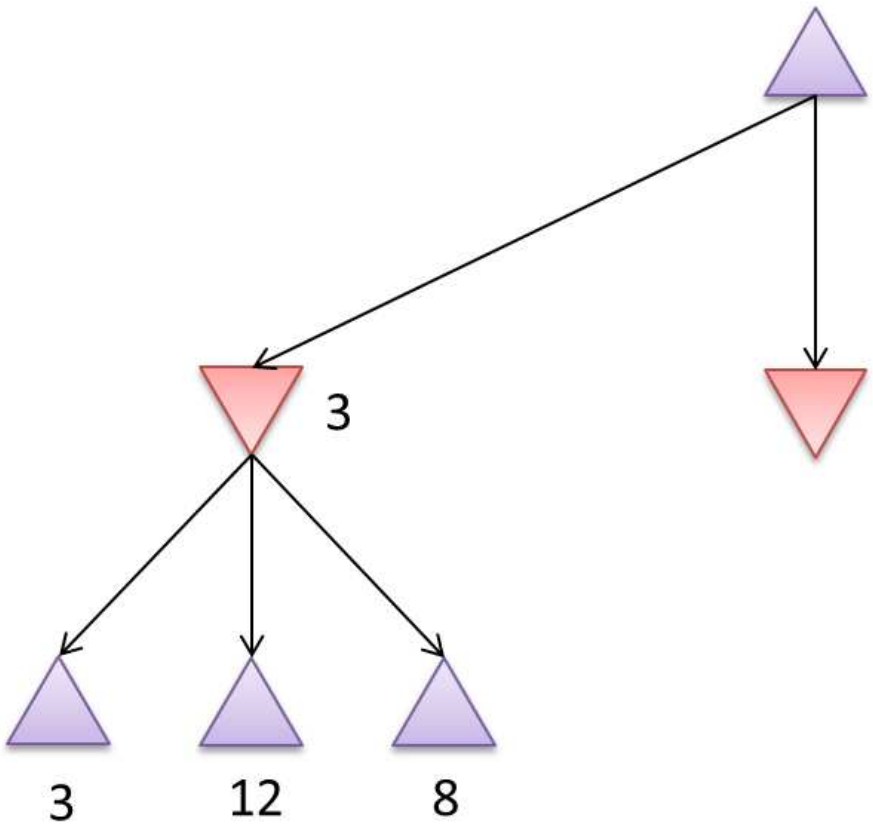
MiniMax : step by step



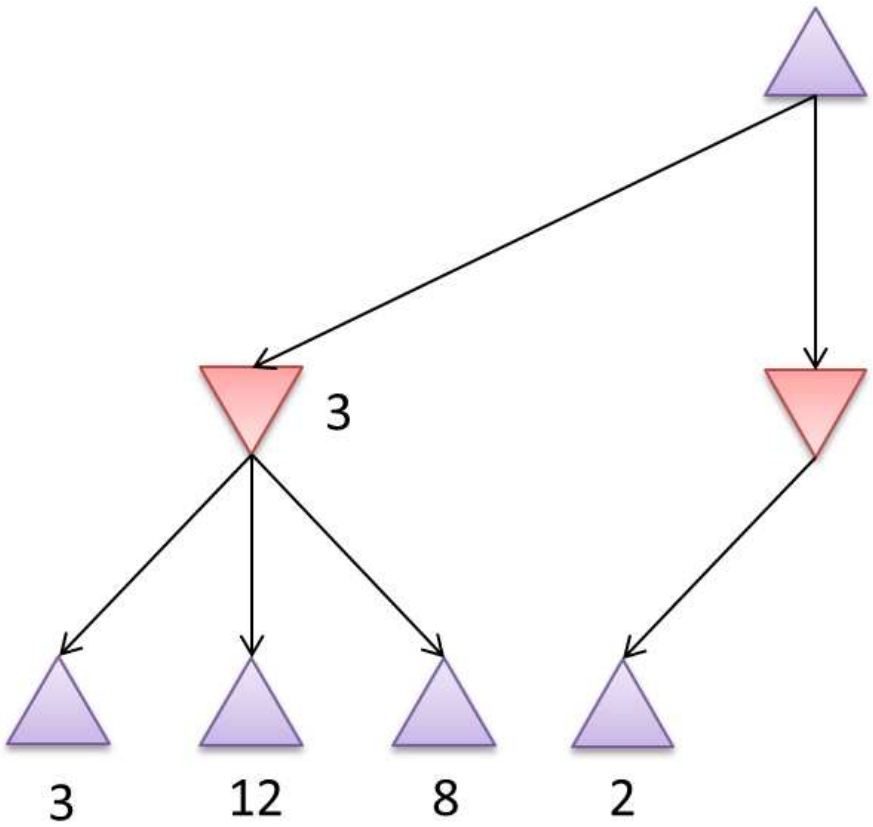
MiniMax : step by step



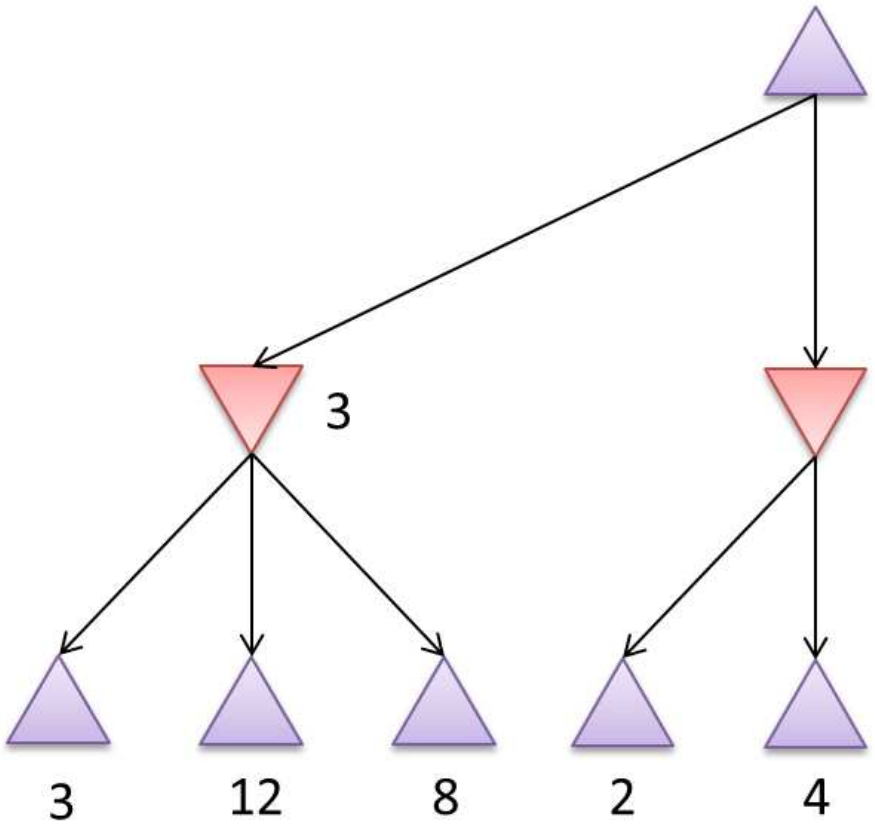
MiniMax : step by step



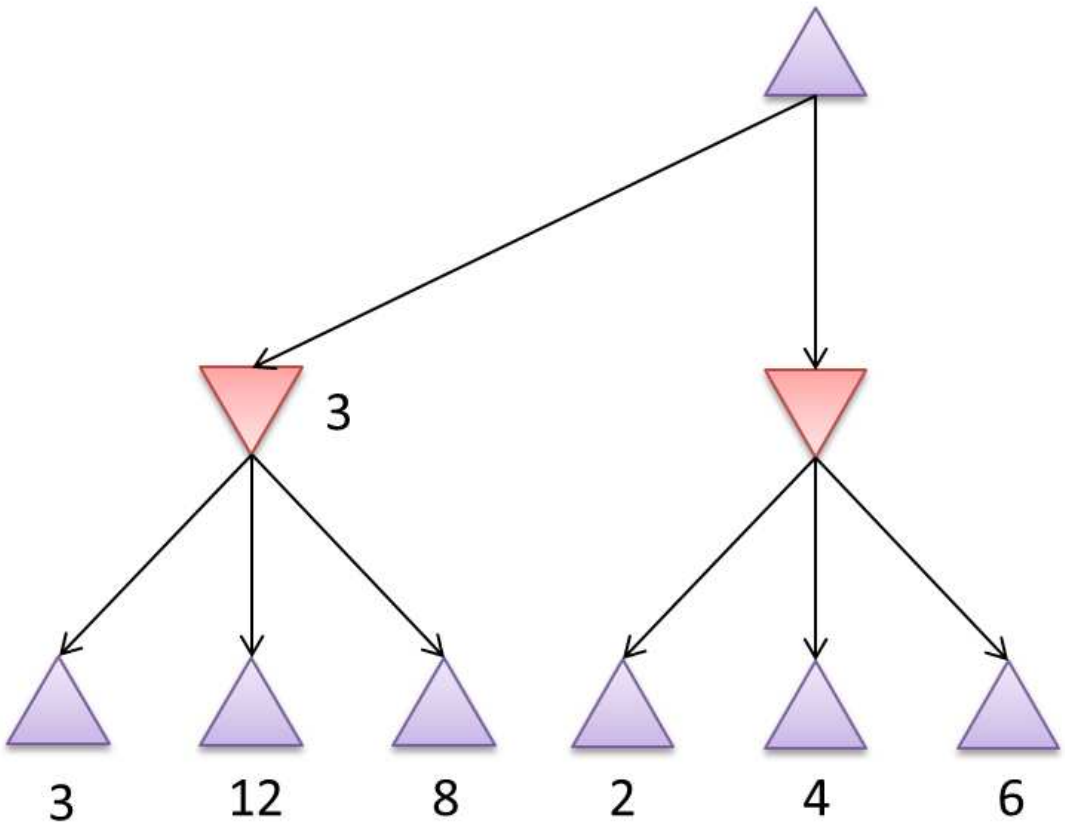
MiniMax : step by step



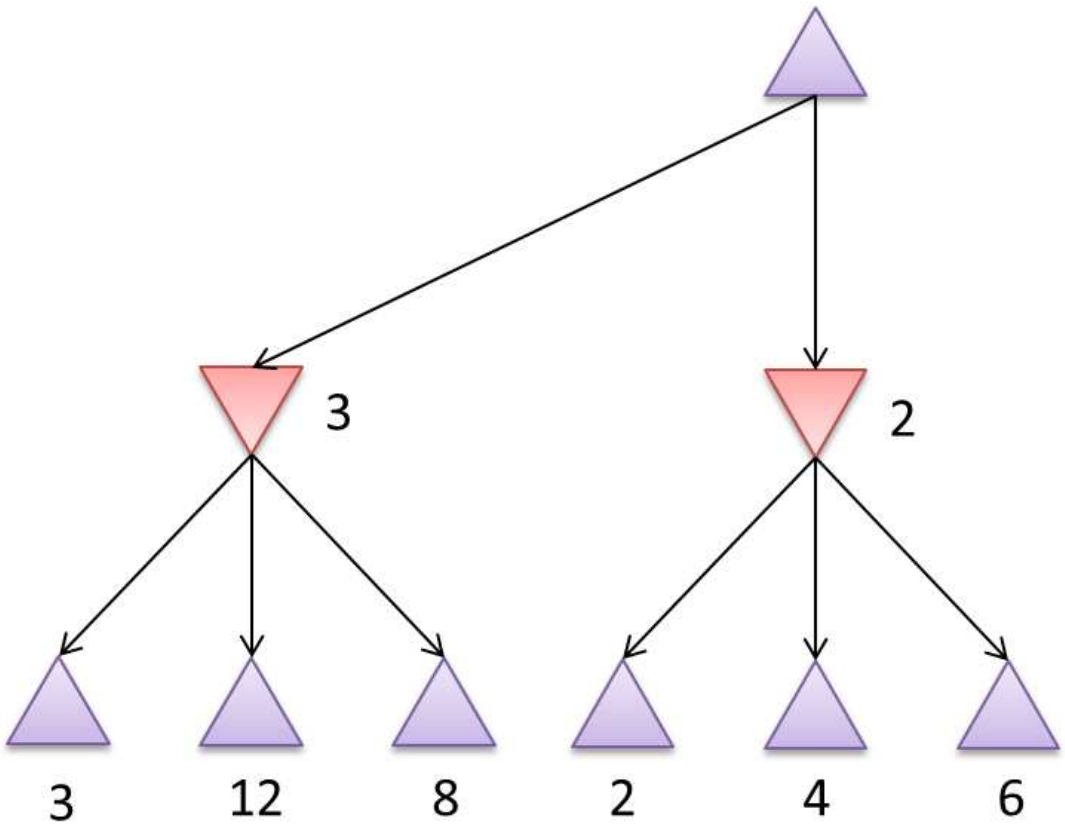
MiniMax : step by step



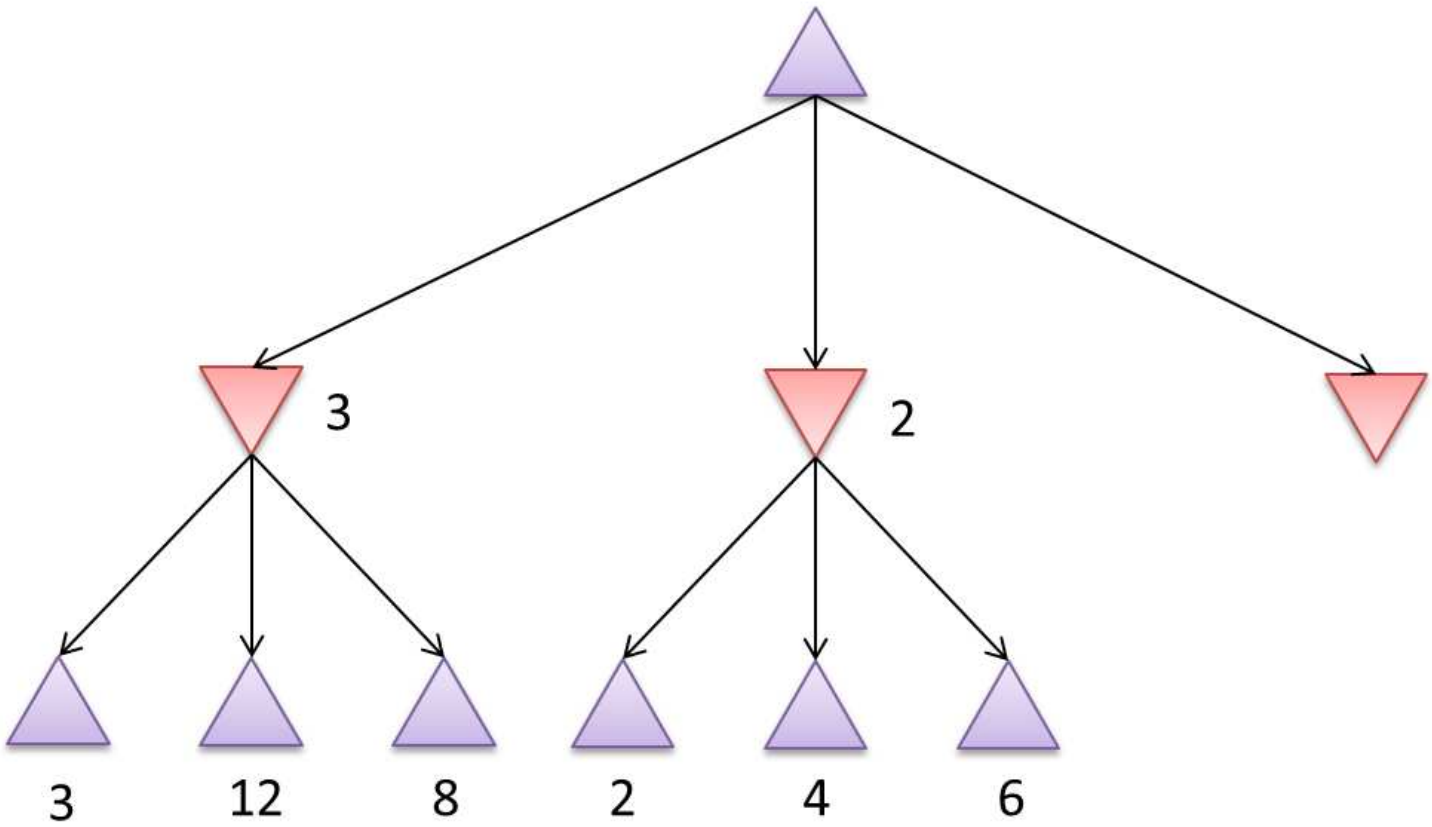
MiniMax : step by step



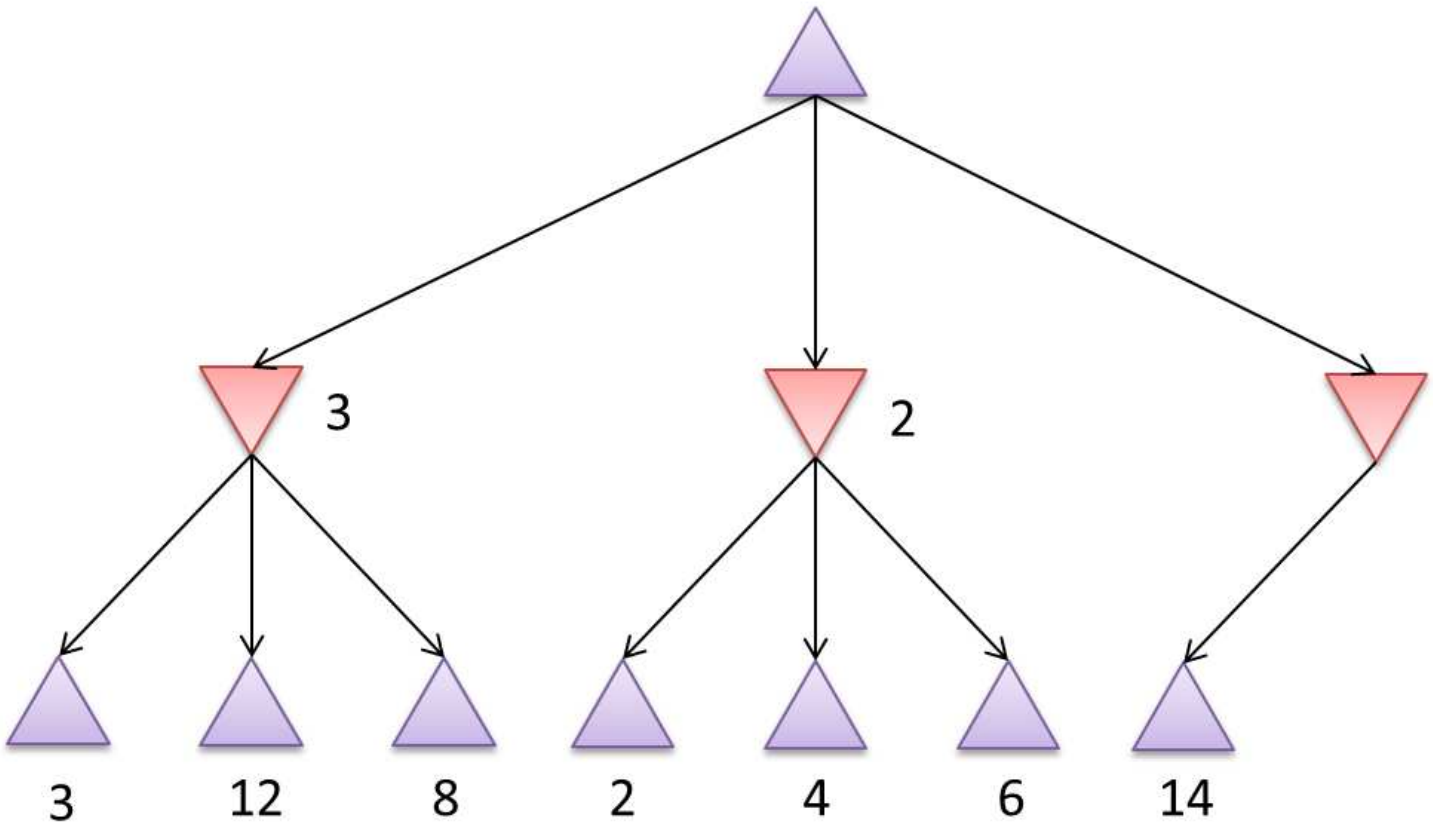
MiniMax : step by step



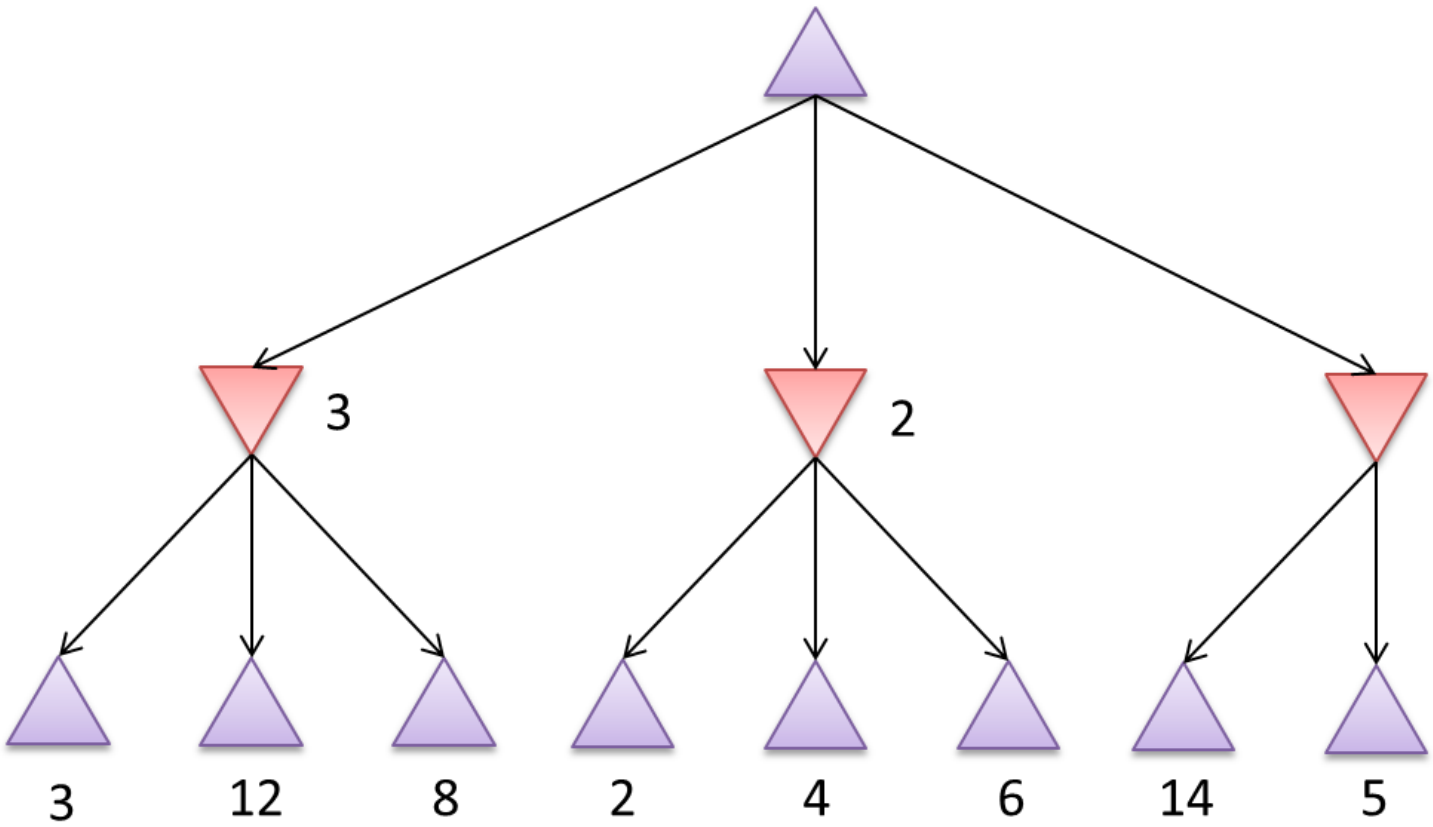
MiniMax : step by steps



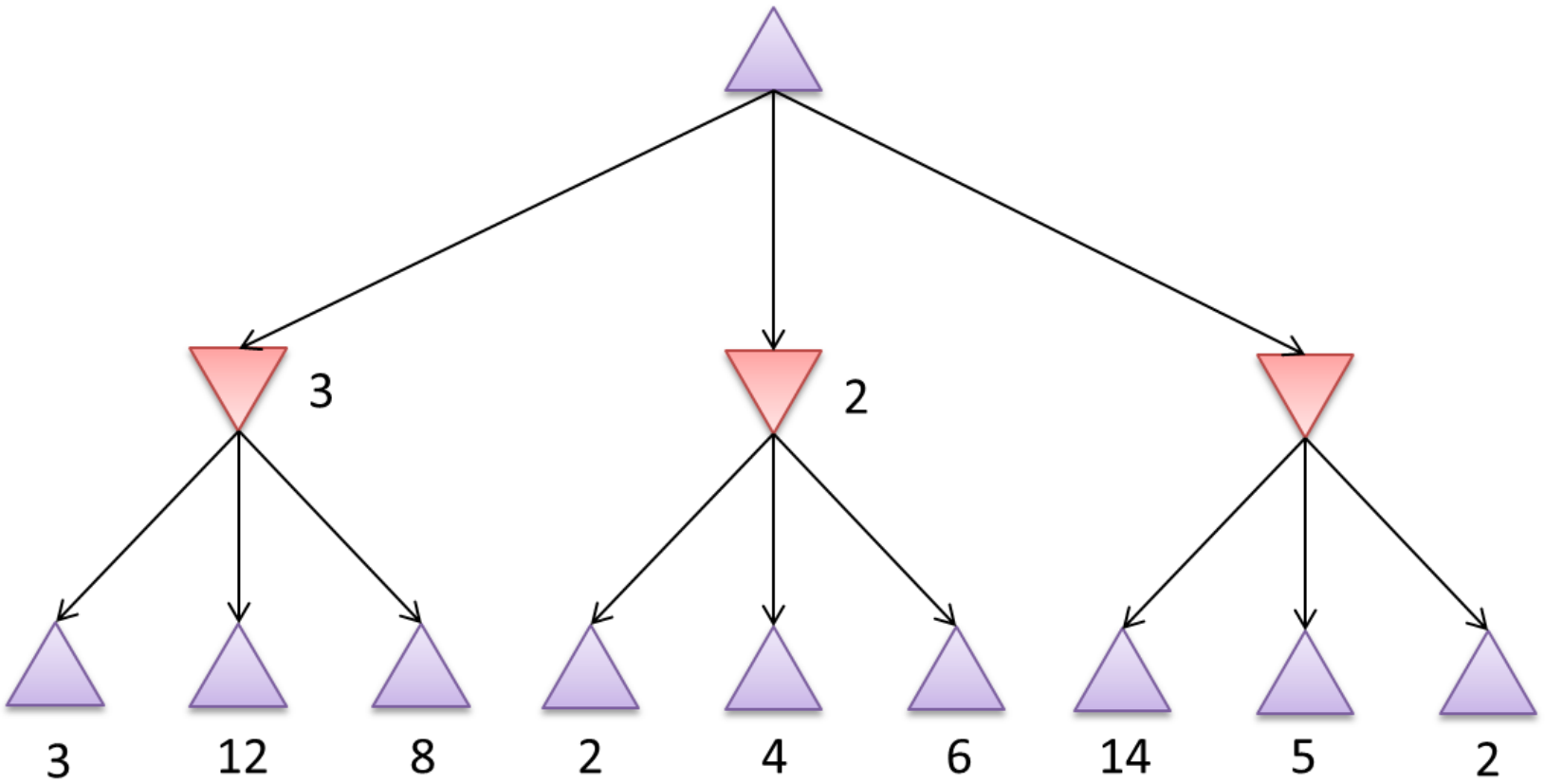
MiniMax : step by step



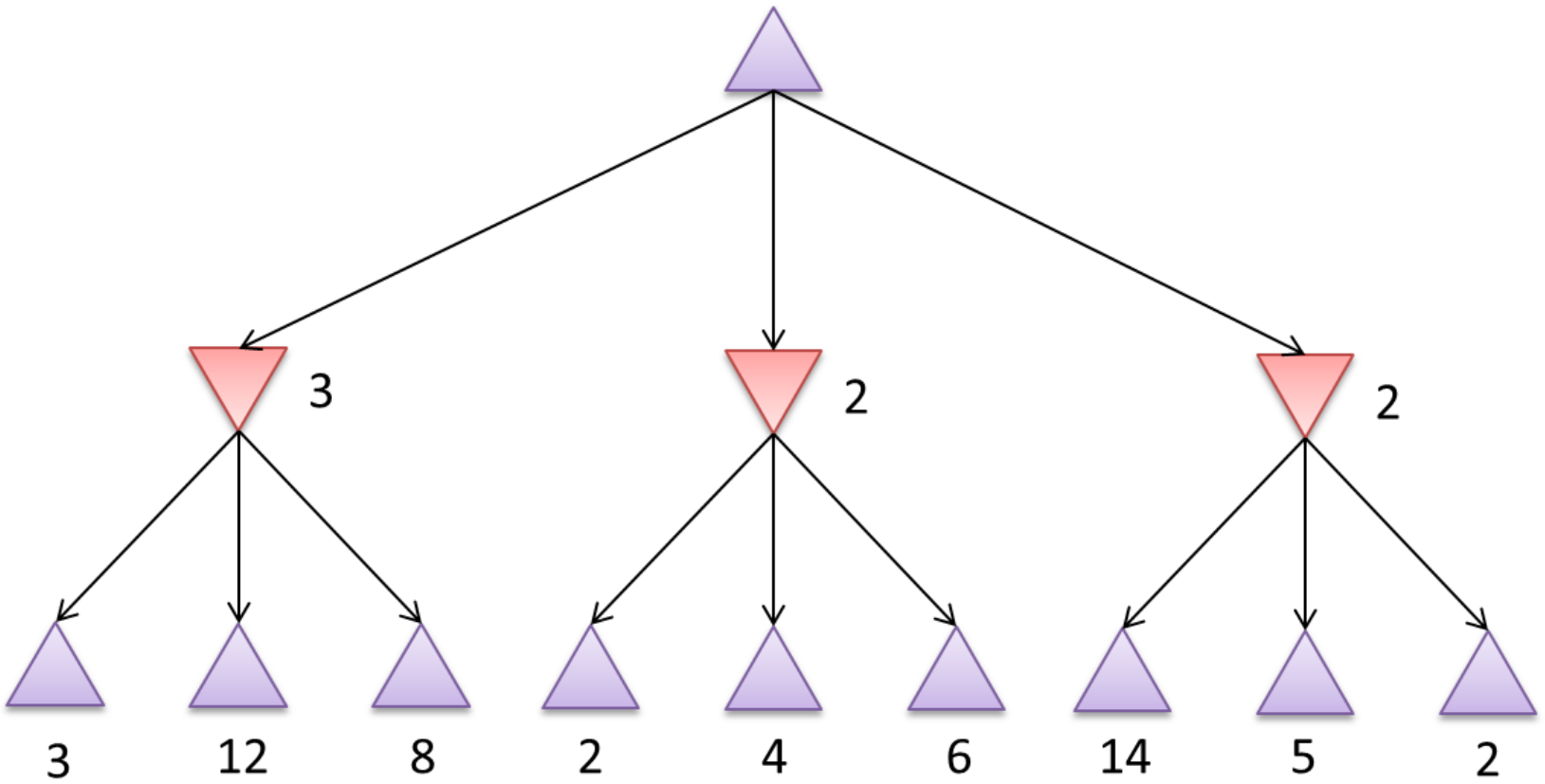
MiniMax : step by step



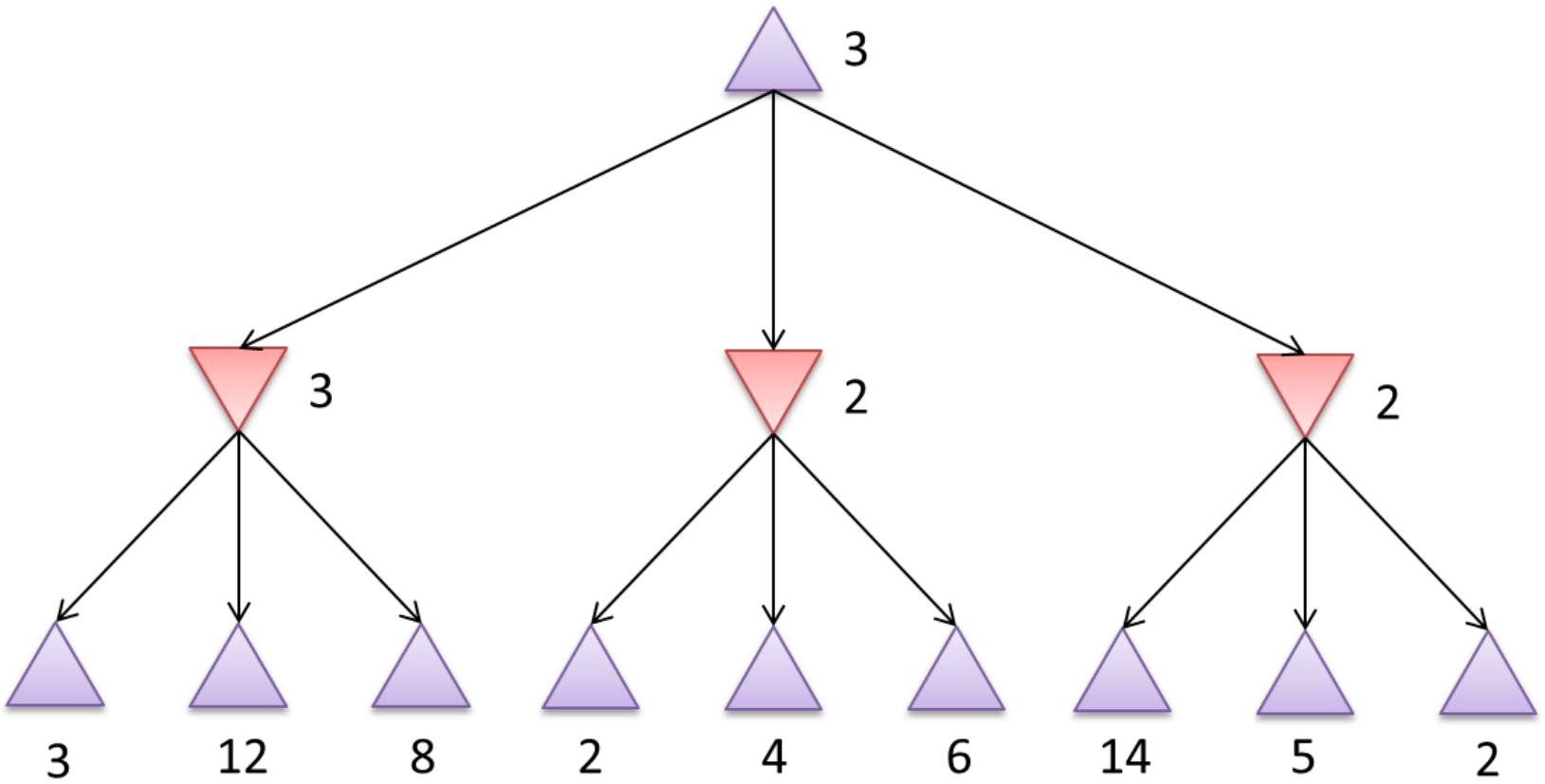
MiniMax : step by step



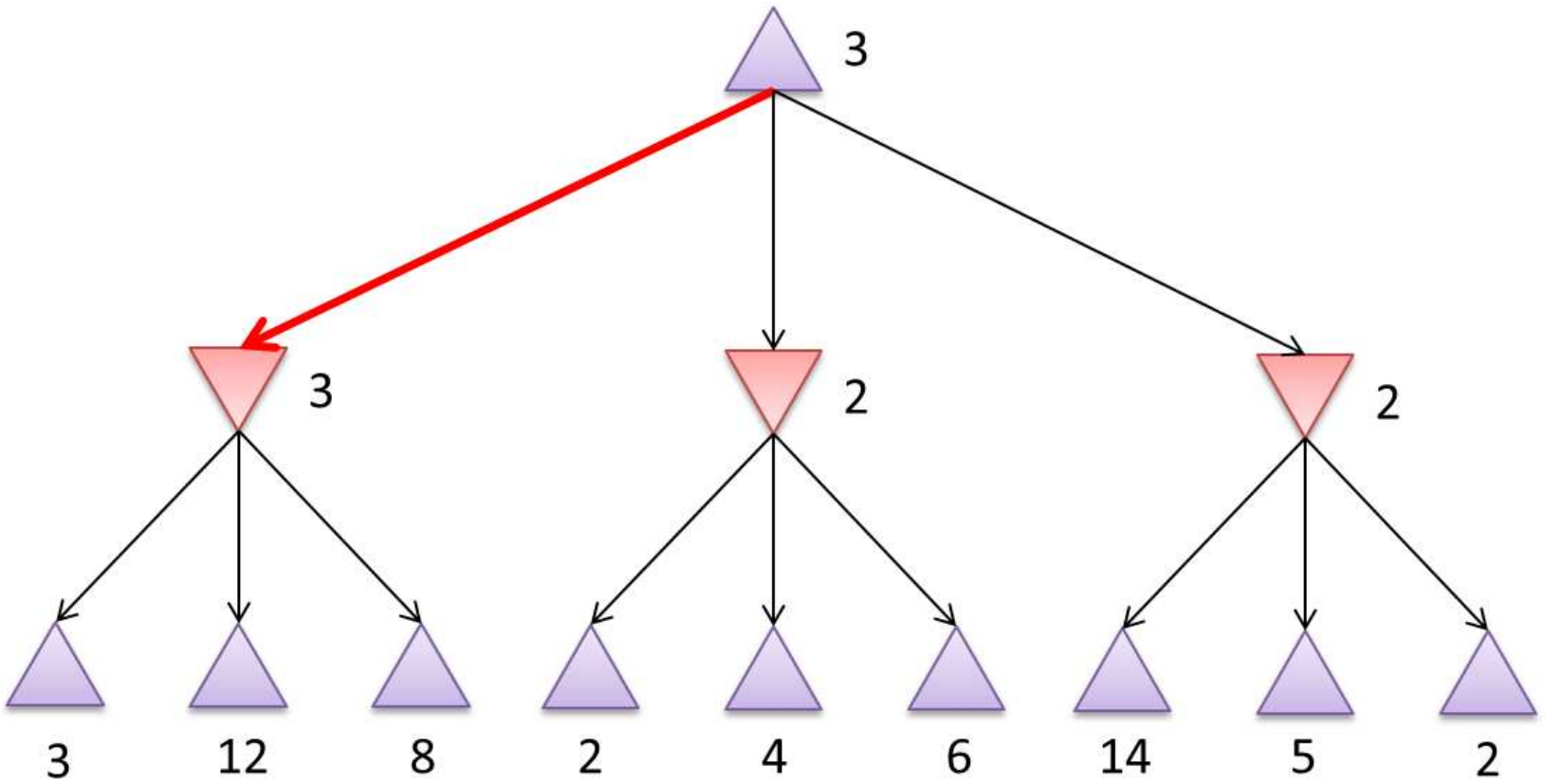
MiniMax : step by step



MiniMax : step by step



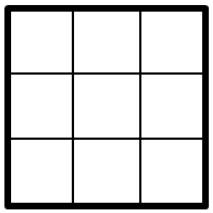
MiniMax : step by step



Evaluation function for a state of the game

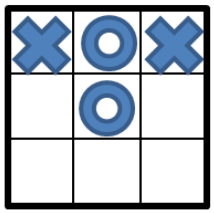
- $e(s) = +\infty$ if s is a gain situation for Max
- $e(s) = -\infty$ if s is a gain situation for Min
- $e(s)$ is a measure of good state s for Max .

e(s) = # lignes/cols/diags ouvertes pour Max
- # lignes/cols/diags ouvertes pour Min



8 – 8 = 0

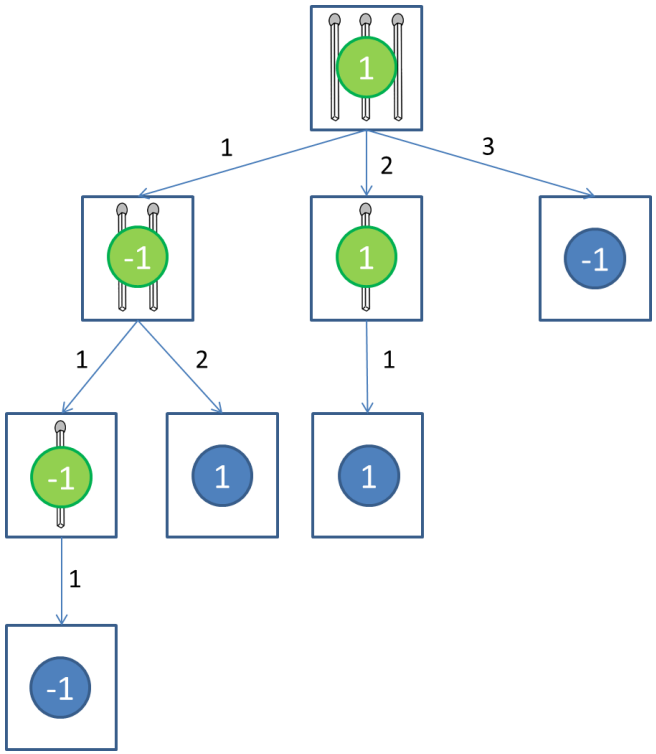
6 – 4 = 2



3 – 3 = 0

The Nim game

The evaluations are propagated to the root



MiniMax

```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game
    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

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
```

- MiniMax-Decision returns the action that Max must play
- Max-Value (MaxMin) returns the minimax of a state where Max has to play
- Min-Value (MinMax) returns the minimax of a state where Min has to play

Properties

Properties

- Completness
  Yes if the tree is finite
- Complexity in time
  $o(b^m)$ (breath-first search) b : branching factor. m : horizon of search
- Space : $o(bm)$
- Optimality : Yes if the opponent is optimal.

In practice, the full tree can not be explored.

Improvments of the MiniMax algorithm

Two approches

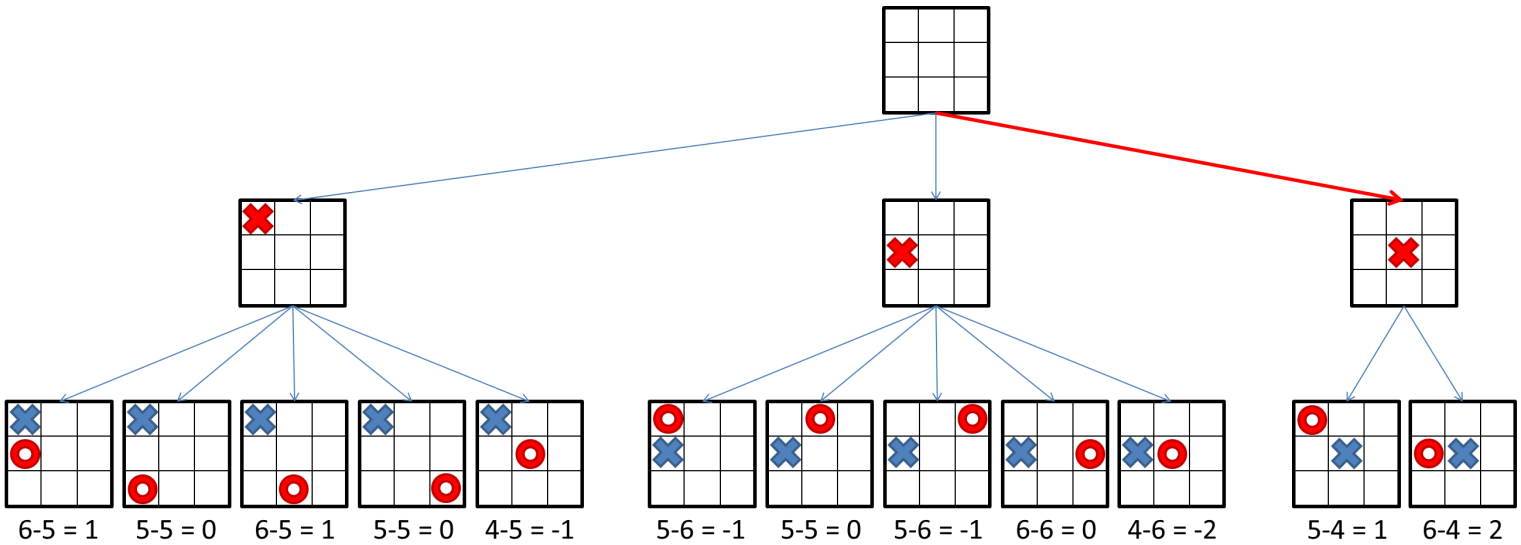
- Limitation of the depth of the exploration
- Search tree pruning : Alpha-Beta

MiniMax with limited depth

Algorithme

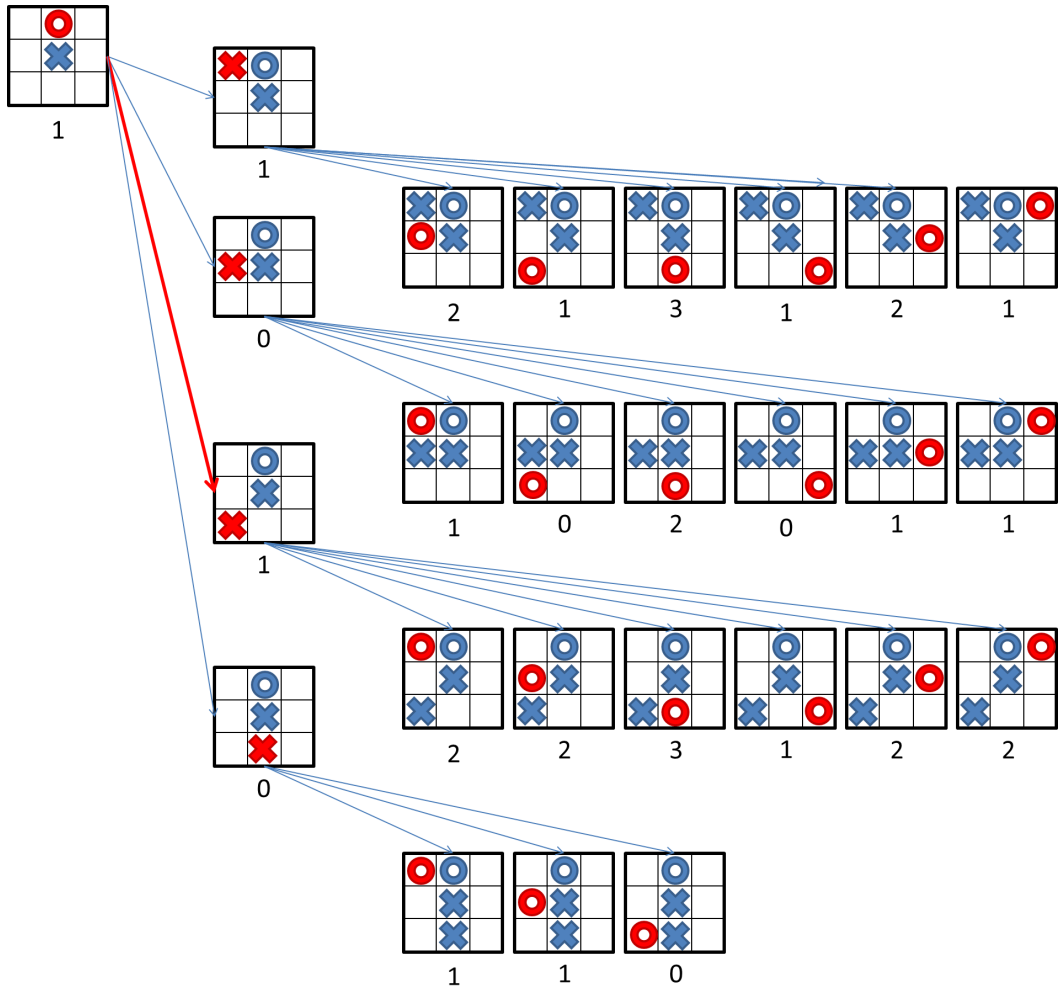
1. The search tree is extended until a given depth h (horizon).
2. Computing of the evaluation function for each leaf.
3. Propagation of the values to the root.
   - 3.1 A Max node receives the maximal value of its successor evaluations
   - 3.2 A Min node receives the minimal value of its successor evaluations
4. Choose of the move to the node Min that has the highest value

Example : Tic Tac Toe with horizon 2



From http ://cui.unige.ch/DI/cours/IA

Example : Tic Tac Toe with horizon 2



From http ://cui.unige.ch/DI/cours/IA

MiniMax with limited depth

For chess

- 4 shots : beginning player
- 8 shots : experimented player
- 12 shots : : Deep Blue

MiniMax

Importance of the evaluation function

A linear function that weighs each item of the game.

Importance of the evaluation function

$$\begin{aligned} f(P) = \; & 200(K-K') + 9(Q-Q') + 5(R-R') + 3(B-B'+N-N') + (P-P') \\ & -\frac{1}{2}(D-D'+S-S'+I-I') \\ & +0.1(M-M') \ldots \end{aligned}$$

avec :

- K, Q, R, B, N, P represent the number of kings, queens, towers, fools, riders and pawns.
- D, S, I represent the white pawns that are isolated, doubled or in front.
- M is a mobility measure(number of moves that a white pawn can do).

Alpha-Beta Algorithm : pruning

Pruning

We want to find the same evaluation function of the root node without a complete extension of the tree. Un-useful parts of the tree will be pruned.

Idea

We consider a node n in the tree such as the player can play in n. If it exists a better node m than n (from the parent node of n or higher in the tree), n will never be played.

Alpha-Beta Algorithm : pruning

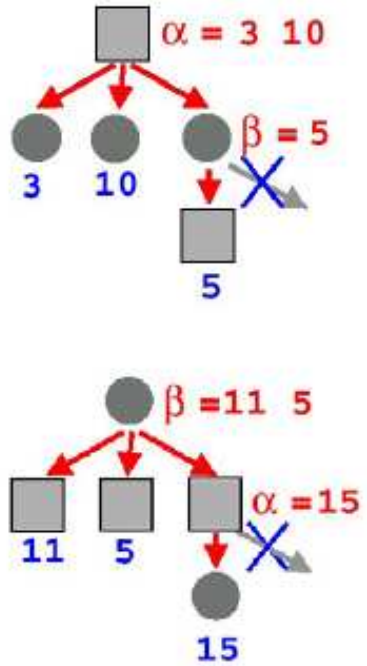
Principle

- Extension of the tree until a depth h.
- Do not generate the successors of a node if it seems clear that this node will not be chosen.
  - Each Max node keeps an α-value, the value of its best found successors at this time. The best choice for Max on the path. α-value is increasing.
  - Each Min node keeps a β-value, the value of its best found successors at this time. β-value is decreasing
- The tree is pruned if α-value is superior to β-value.

Alpha-Beta Algorithm

Two rules

1. Stop the search of a Max node if its α-value $\geq$ β-value of its parent node.
2. Stop the search of a Min node if its β-value $\leq$ α-value of its parent node.



Alpha-Beta Algorithm

Algorithm 1: Algorithm MaxValue

Function *MaxValue (etat, α, β)*/* Evaluation FRIEND level */

Data:

- *state* : current game
- α : best evaluation FRIEND
- β : best evaluation ENNEMY

begin

  if *IsLeaf(state)* **then**

    return *evaluate(state)*/* Evaluation with heuristics */

  end

  forall *successor s of sate* **do**

    $\alpha \longleftarrow \max(\alpha, \mathit{MinValue}(s, \alpha, \beta))$

    if $\alpha \geq \beta$ **then**

      return β/* pruning β */

    end

  end

  return α

end

Alpha-Beta Algorithm

Algorithm 2: Algorithm MinValue

Function *MinValue (state,* α*,* β*)*`/* Evaluation ENNEMY level */`

Data:

- *state* : current game
- α : best evaluation FRIEND
- β : best evaluation ENNEMY

begin

  if *EstFeuille(state)* **then**

    return *evaluate(state)*`/* Evaluation with heuristics */`

  end

  forall *successor s of state* **do**

    $\beta \longleftarrow \min(\beta, \mathit{MaxValue}(s, \alpha, \beta))$

    if $\alpha \geq \beta$ **then**

      return β`/* Pruning` α `*/`

    end

  end

  return α

end

Alpha-Beta Algorithm

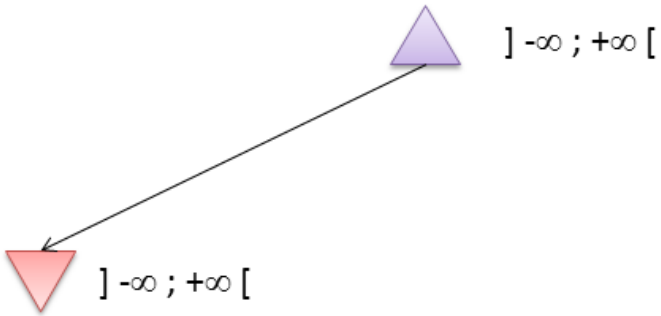
Principe

- To evaluate a game, we apply *MaxValue* with the current state of the game.
- Initially $\alpha = -\infty$ and $\beta = +\infty$

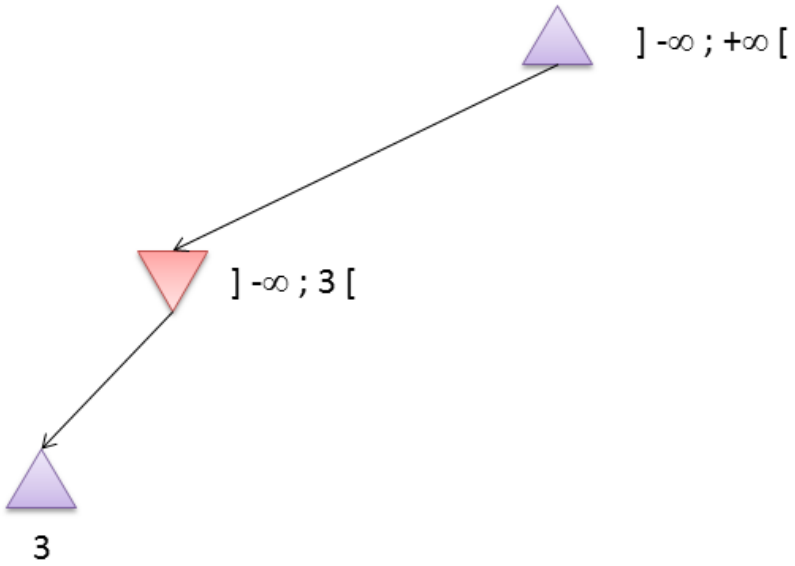
Example 1

] -∞ ; +∞ [

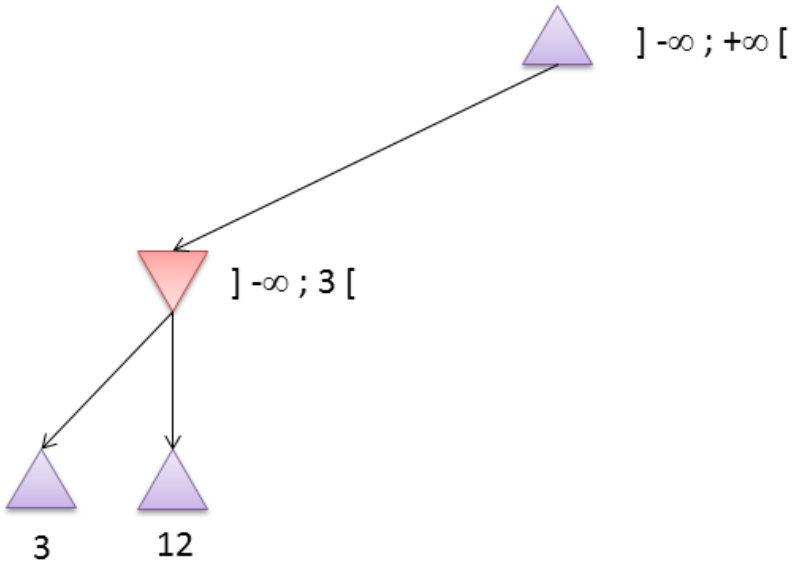
Example 1



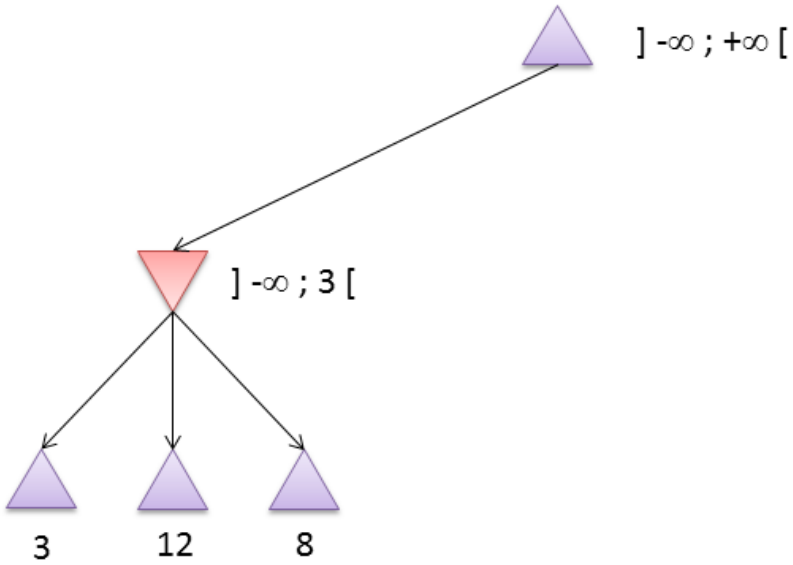
Example 1



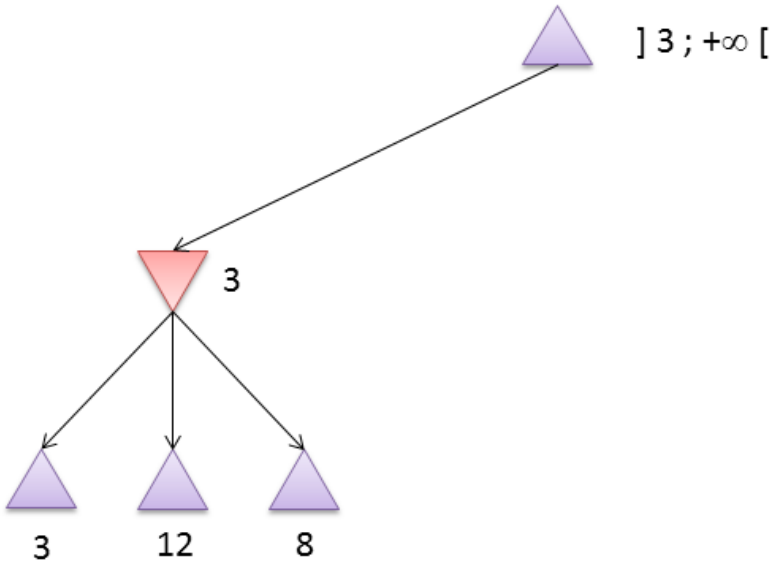
Example 1



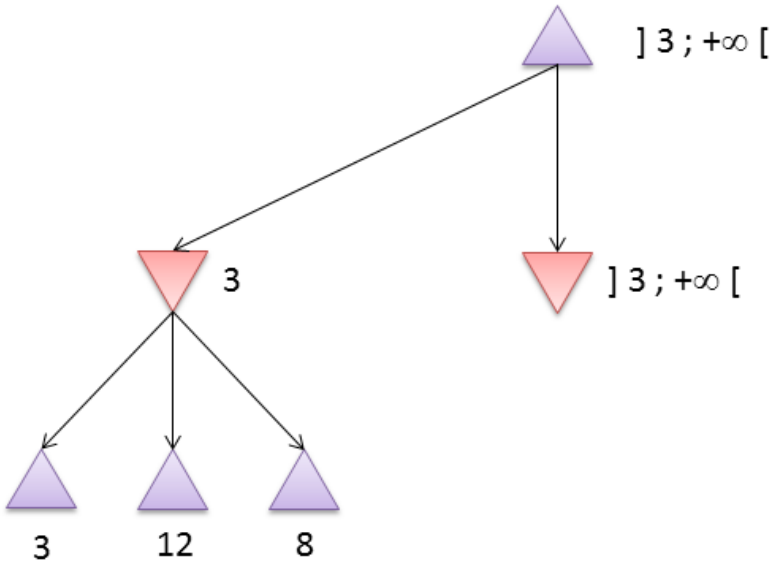
Example 1



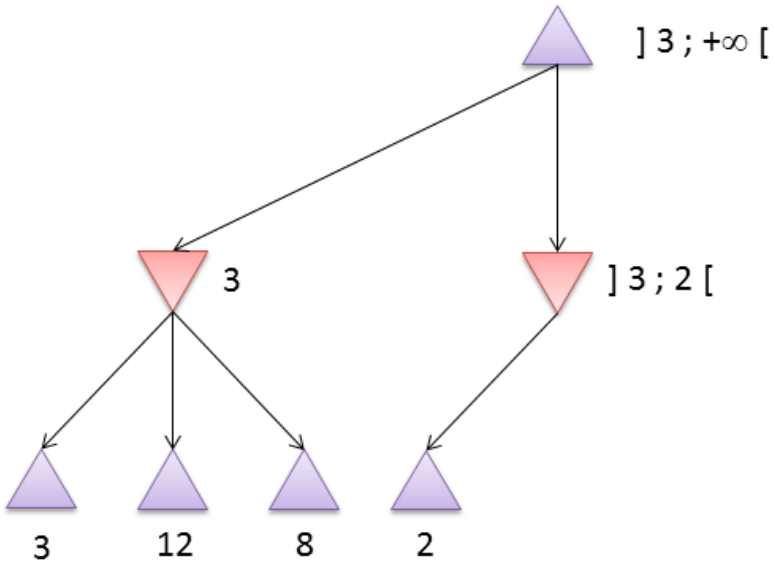
Example 1



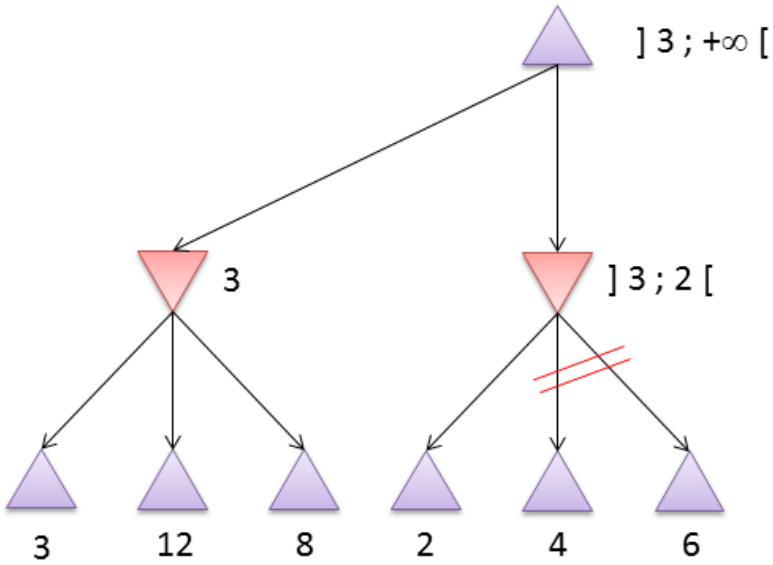
Example 1



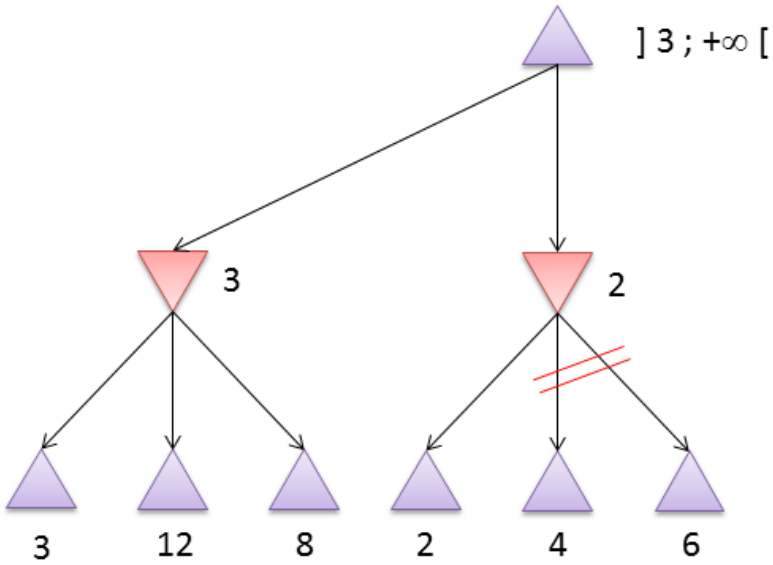
Example 1



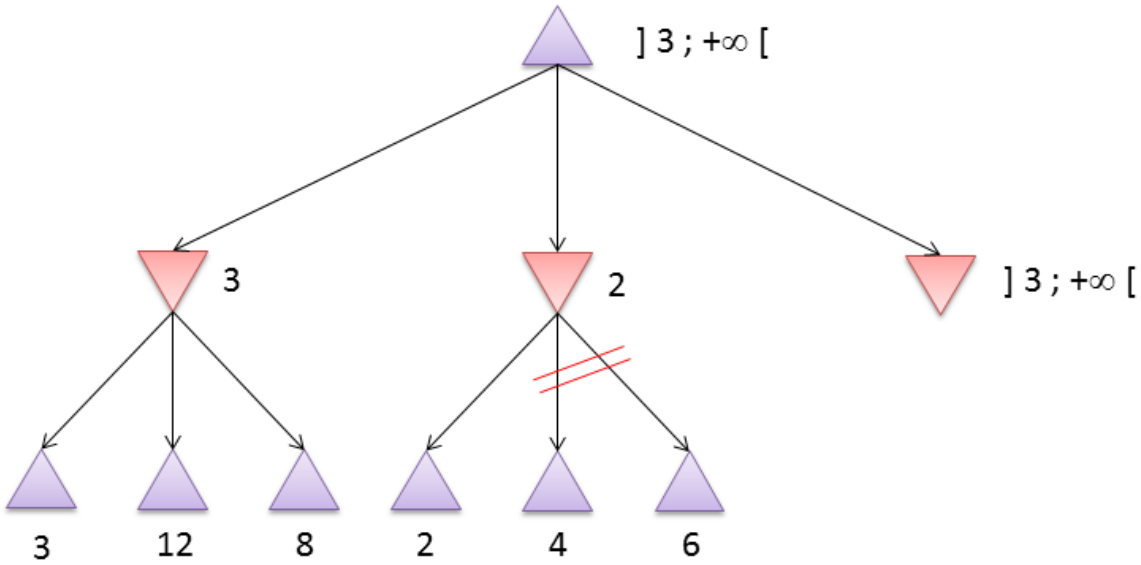
Example 1



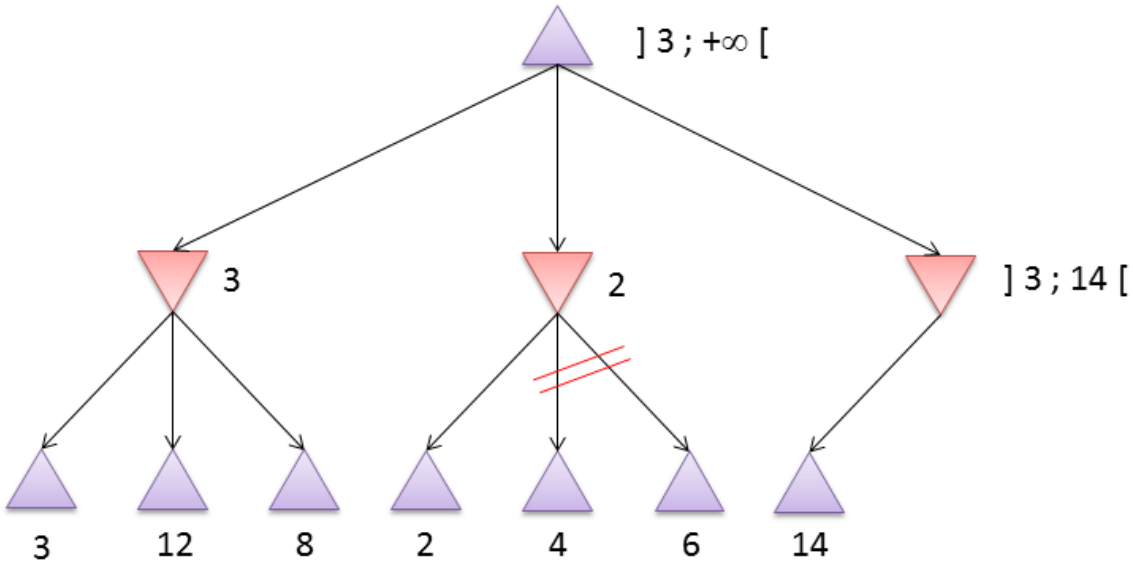
Example 1



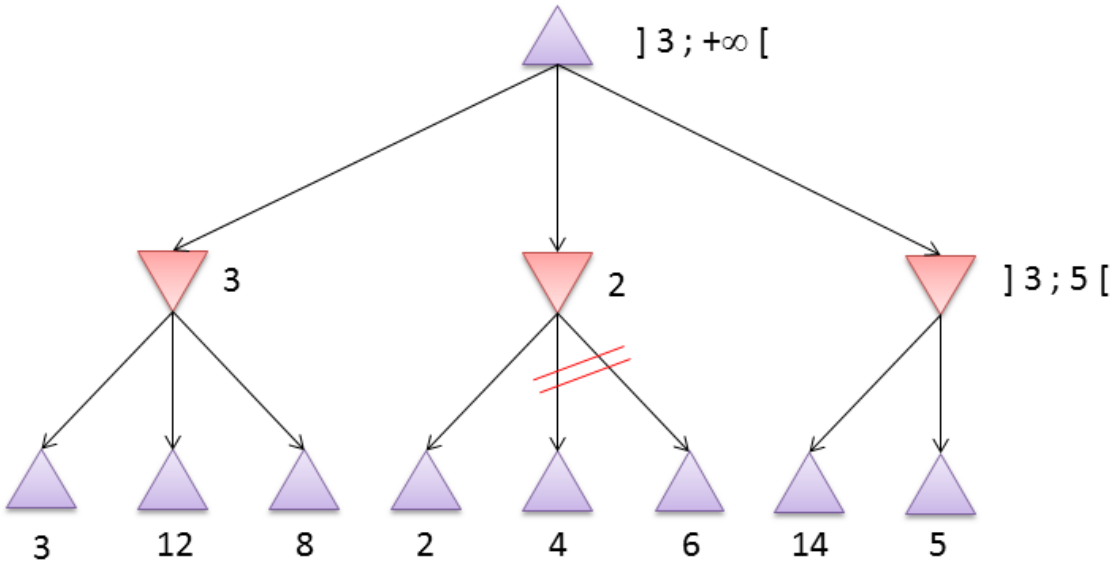
Exemple 1



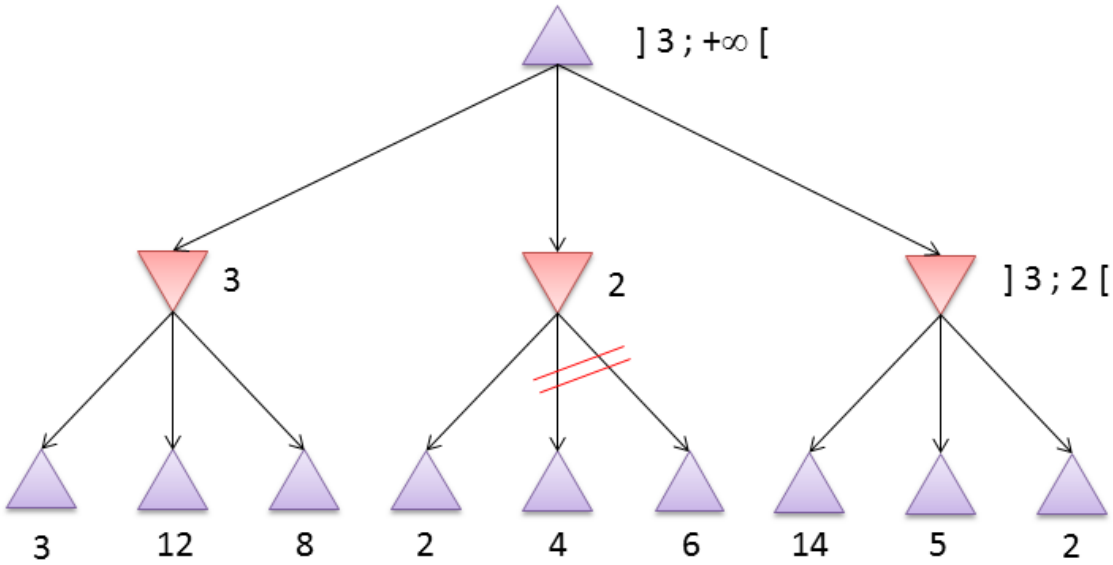
Example 1



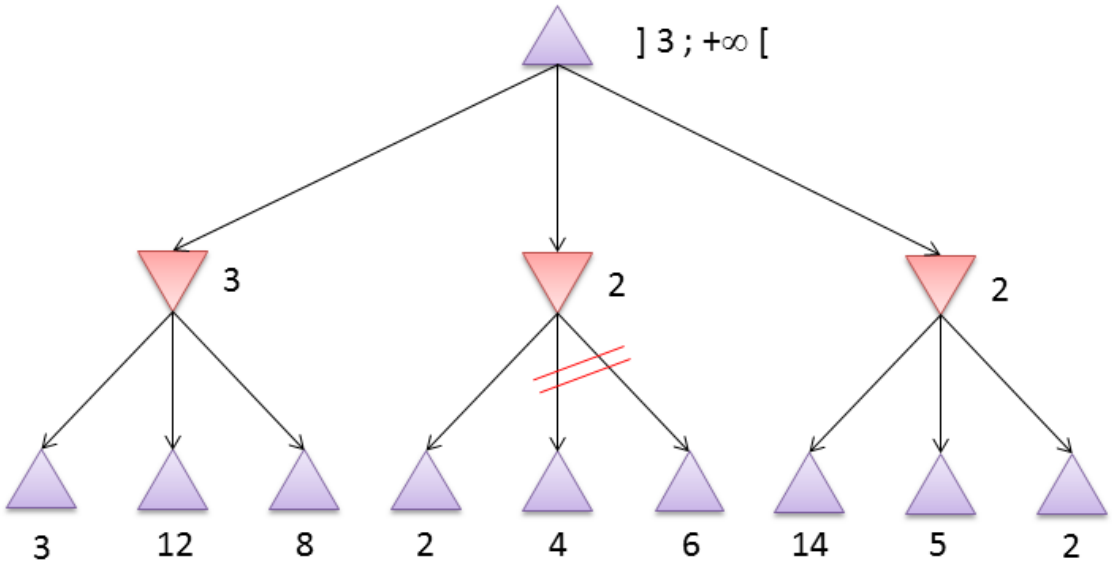
Example 1



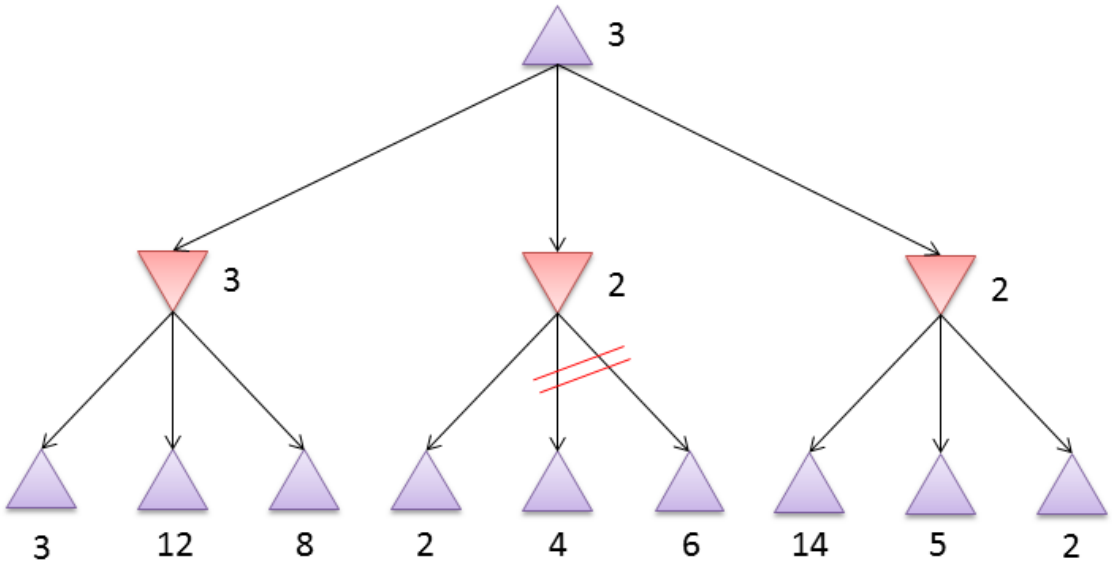
Example 1



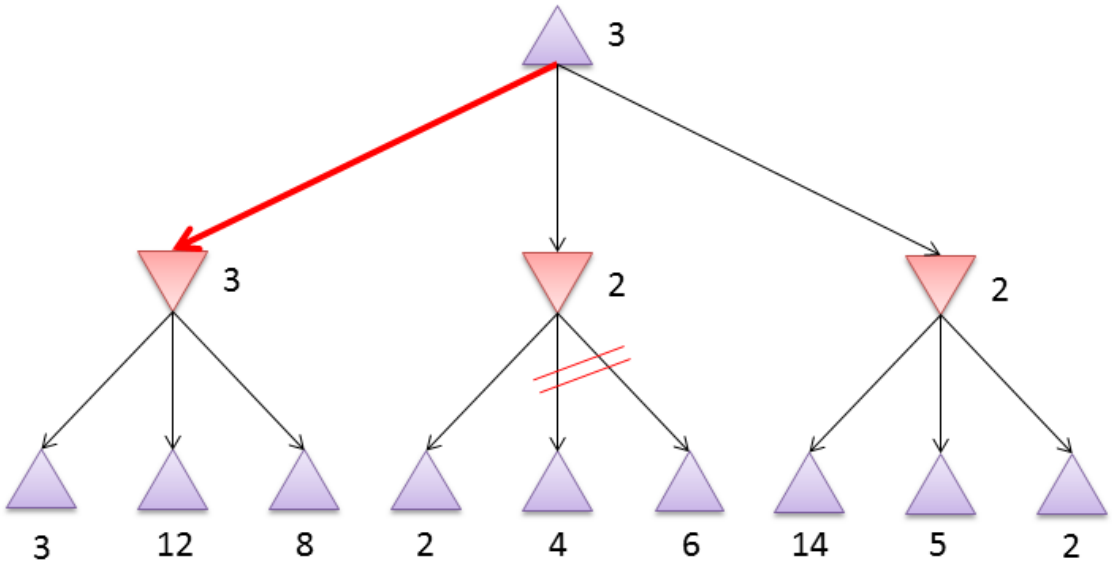
Example 1



Example 1



Example 1



To resume

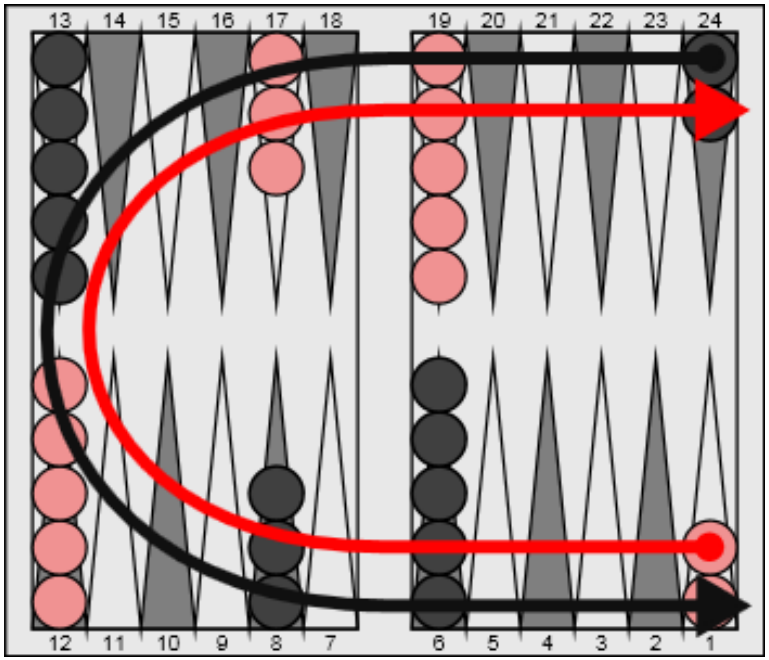
- The alpha value of a node MAX is a lower bound of the propagated value
- The beta value of a node MIN is a higher bound of the propagated value
- Updating of the alpha-beta of the parent of a node N when the search below N is finished or stopped.
- Stop the search below a node MAX if its alpha value is superior to the beta value of a parent MIN of N
- Stop the search below a node MIN if its beta value is inferior to the alpha value of a parent MAX of N

Comparison of the algorithms

Number of explored nodes and execution time for the Reversi Game

p	minimax	$\alpha - \beta$	SSS*
1	3 - 0.37	3 - 0.36	3 - 0.36
2	14 - 1.38	13 - 1.4	11 - 1.19
3	61 - 6.0	37 - 3.9	35 - 3.7
4	349 - 32.5	150 - 14.77	95 - 10.0
5	2050 - 185.7	418 - 41.4	292 - 19.2
6	13773 - 1213.5	1830 - 172.9	1617 - 117.3

Backgammon



Backgammon

- Goal : All the queens have to be put outside of the board.
- The possible moves are determined by some dices
- Two dices give the possible moves.

Backgammon

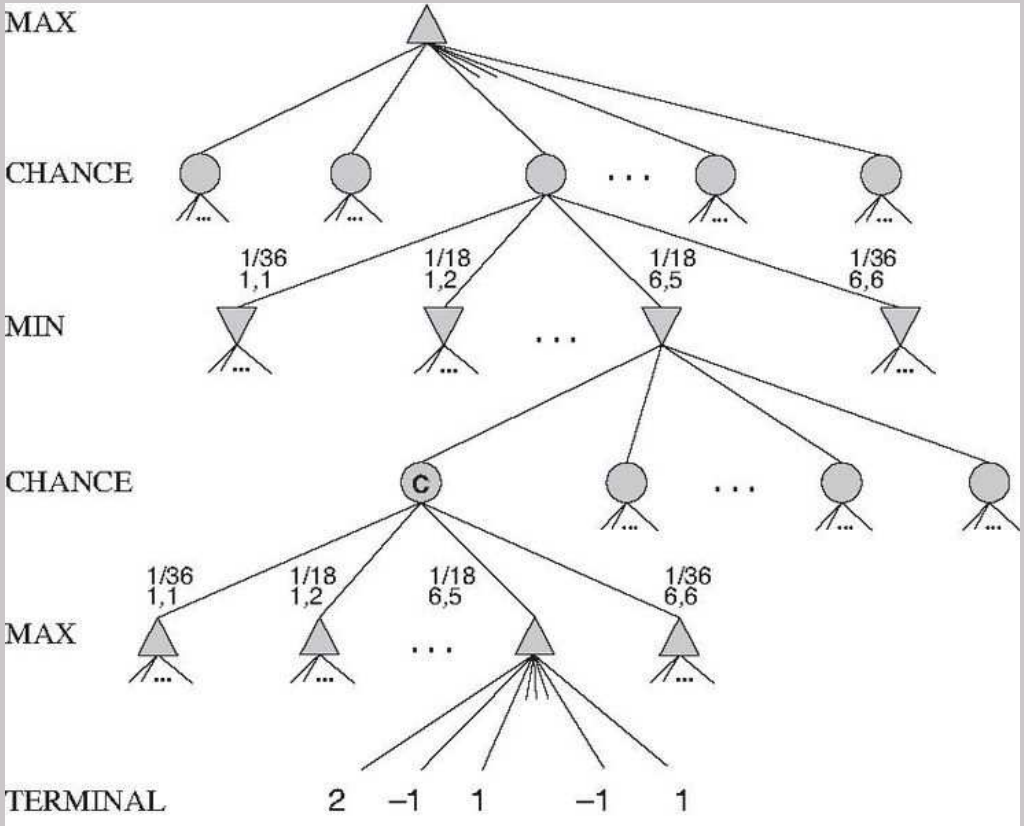
Issue

How modify the previous algorithms to take into account the hazard of the dices ?

Games with hazard

Idea

We have layers MIN and MAX, but also a CHANCE layer to represent the hazard



Games with hazard

ExpectMiniMax

The algorithm is the same than MiniMax except that the return value is :

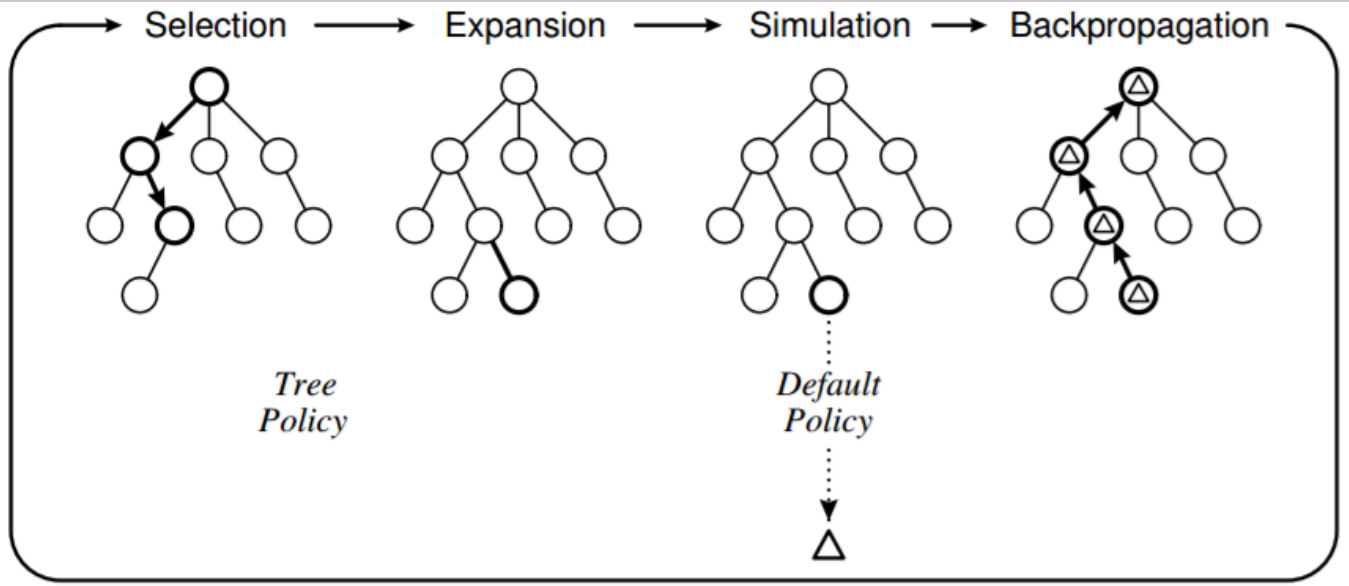
$$\textsc{ExpectMiniMax}(\text{n})= \begin{cases} evaluation(n) & \text{si } n \text{ is an end node} \\ max_{s \in successeurs(n)} \textsc{ExpectMiniMax}(\text{s}) & \text{if } n \text{ is a node MAX} \\ min_{s \in successeurs(n)} \textsc{ExpectMiniMax}(\text{s}) & \text{if } n \text{ is a node MIN} \\ sum_{s \in successeurs(n)} P(s).\textsc{ExpectMiniMax}(\text{s}) & \text{if } n \text{ is a node CHANCE} \end{cases}$$

Monte-Carlo Tree Search

Use

When the search space is too big or when the evaluation function is too complex

Principle



Monte-Carlo Tree Search

4 steps

- Selection : from the root, a choice is made recursively to go until a node that is urgent to extend among the extensible nodes.
- Expansion : one or several sons are added to the previous chosen node according the available actions
- Simulation : simulations of games to have a score.
- Back propagation : The statistics of the node of the considered branch are updated.

Each node has in memory :

- the number of times it was visited
- the obtained score

Monte-Carlo Tree Search

Strategies to chose the next shot

Some actions are chosen from the root according to a criteria among whom :

- Max : we choose the action that leads to the node that has the highest reward ;
- Robustness : we choose the action that leads to the node that is the most visited ;
- Max and Robustness : we choose the action that leads to the node that is the most visited and that has the highest reward.
- ...

Monte-Carlo methods

How to choose the node to extend ?

Exploration/exploitation dilemna

- Making more simulations for the shots that seem to be better (to decrease the value of their evaluation),
- But we also want to explore other nodes that could be better.

Monte-Carlo methods

UCB critera (Upper Confidence Bound)

Choose the action that maximizes the criteria UCB :

$$\mu_i + C \times \sqrt{\log(p) \div p_i}$$

où

- μ_i is the mean of the scores of the games that begin by this shot,
- p is the number of played games
- p_i is the number of games that begin by this shot
- C is a constant.

How to choose C in practice ?

C depends on the game.
C high is in favor of exploration.

Alpha GO

Although progress has been steady,
it will take many decades of research and development
before world-championship calibre GO programs exist.

Jonathan Schaeffer, 2001

Alpha GO, 2016, wins againts Lee Seedol

Alpha GO

Principle

- based on MCTS
- use of neural networks to choose the node to extend, to simulate some games and to evaluate the game.
- use of human game play and automatic game play

Alpha GO

Shot selection (network policy)

Two deep neural networks + 1 for initialization :

- P_π : small network trained on human game plays, estimation of $P(a|s)$, which will be used during the simulations to find the next shot to play (response time : 2 μs)
- P_σ : bigger network trained in true game plays, estimation of $P(a|s)$
- P_ρ : network that has the same topology as P_σ, initialized as P_σ, then trained by reinforcement learning on game plays against itself to maximize its victories. (response time : 3 ms), used in the selection phase

Gain estimation (value policy)

A deep network to compute $V(s)$, the value of the game, estimated from game plays against itself.

Alpha GO

In practice

- 30 millions of positions played by humans have been used.
- V(s) as criteria of MCTS enable to win against all the GO software but simulation is necessary against world champions.
- Basic versions : 40 threads, 48 CPUs, 8GPUs (to play, not to learn)
- Distributed version : 40 threads, 1202 CPUs, 176 GPUs
- AlphaGO wins 494/495 games against the other programs.