version 0.2
last update: 2015.03.29/18:12

Games and strategies

Mihai Oltean
moltean@cs.ubbcluj.ro
www.cs.ubbcluj.ro/~moltean

What kind of games we are dealing with?

- Two players (*A* and *B*)
- Players are moving alternately
- Free Moves
- Full information
- Result: Win or Loose.
- Strategy = set of rules defining moves.
- We are searching for a *Perfect Winning Strategy* for *A*! (*A* wins no matter *B*.)

What kind of strategies we use?

- Symmetry
- Pairs
- Parity
- Dynamic Programming
- And/Or Trees
- MiniMax (with Alpha-Beta cutoffs)
- Others…

Symmetry

- Array of *n* cells (*n* odd)
- Each player fills 1 cell at a move
- Wins the player which makes to appear at least 3 consecutive filled cells.

Dynamic Programming (DP)

- Decompose the problem in subproblems.
- Solve the small subproblems.
- Combine the subproblems in order to obtain the solution for bigger and bigger subproblems.
- Ex: subset sum.

Dynamic Programming for game strategies

- Decompose the game in smaller subgames.
- Find a perfect winning strategy for the smallest games.
- Combine the strategies of the smaller subgames in order to obtain a strategy for larger subgames.

Practical implementation of DP

- Label each subgame with either **T** (true) of **F** (false) depending on whether the player to move has a perfect winning strategy from that position.
- A bigger subgame is labeled with:
  - **T** if $\exists$ at least 1 subgame labeled with **F** to move into.
  - **F** if all subgame which can be reach with 1 move are labeled with **T**.

Example 1 – Extract coins
1 dimensional DP

- A stack with *N* coins. Each coin has 1$ or 2$ value.
- Two players A and B.
- A move - extract any number of consecutive coins having the same value.
- The player performing the last move is the winner of the game.
- A moves first.

Practical example

1
1
2
1
1
2
2

T
T
F
T
F
T
T
F

The player A has a perfect winning strategy.

Example 2 - Tzeanşîdzî game
2 dimensional DP

- 2 stacks of objects
- 2 players A and B
- A move consists of extracting:
  - either any number of objects from a single stack
  - or the same number of objects from both stacks.
- A moves first
- The player performing the last move is the winner

Practical example

	0	1	2	3	4	5
0	F	T	T	T	T	T
1	T	T	F	T	T	T
2	T	F	T	T	T	T
3	T	T	T	T	T	F
4	T	T	T	T	T	T
5	T	T	T	F	T	T

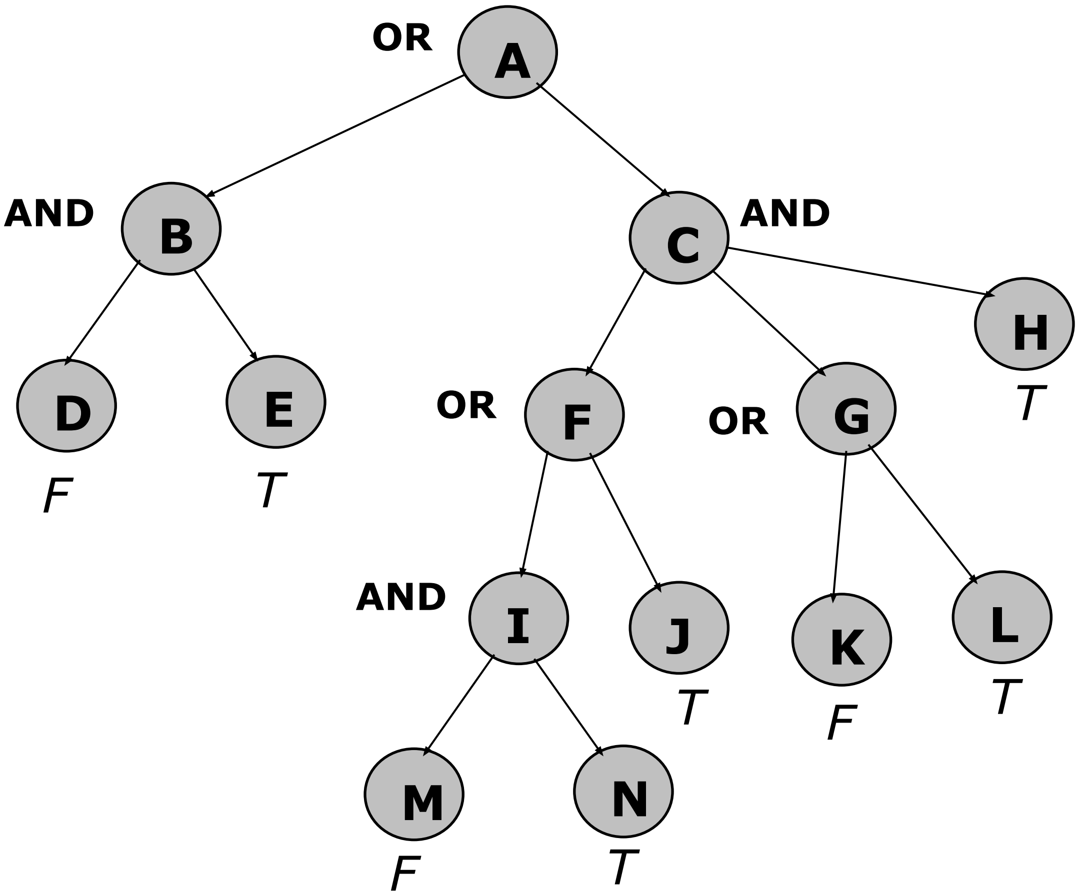
Difficulties with DP

- Not enough memory!

And/Or trees

- Label each node of the game tree with either **T** or **F** depending on whether the first player (*A*) has (or not) a perfect winning strategy for that subgame.
- Leaves of the tree are automatically labeled (with **F** and **T**).
- Rules for labeling internal nodes:
  - (for *A*) Label with **T** if at least one child is labeled with **T**. – OR rule.
  - (for *B*) Label with **T** if all children are labeled with **T**. – AND rule.

Example



And/Or Algorithm

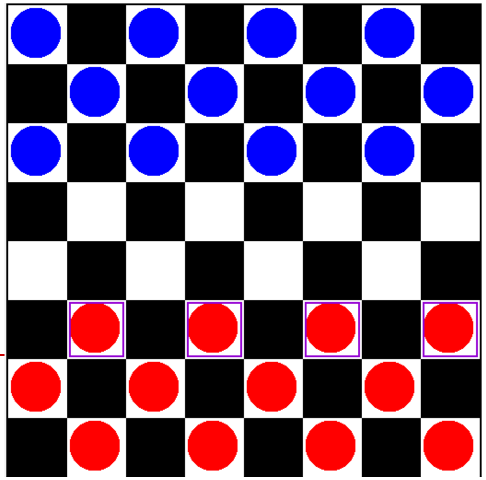
```
bool AndOr(Node N, int level){
          //level = 0 for the node in the top of the tree.
     if N is terminal {
          if the first player won
               return true;
          else return false;}
     else{
          if (level % 2){//B is about to move; AND
               bool result = true;
               for each child Ni of N
                    result = result && AndOr(Ni, level+1);
          }
          else{// A is about to move; OR
               bool result = false;
               for each child Ni of N
                    result = result || AndOr(Ni, level+1);
          }
          return result;
     }
}
```

Strength of And/Or trees

- Can be applied to any kind of game!

Weaknesses of And/Or Trees

- Out of memory!
- Out of time!
  - Chess: 10^{100} nodes.
    - Deep Blue can explore $2*10^8$ positions/second -> 10^{83} years!
  - GO: 10^{320} nodes.
  - Checkers: $5*10^{20}$ nodes



MiniMax search

- Not the entire tree is explored.
- A maximum search depth is fixed at the beginning of the game.
  - How far we go?
    - too deep and the search will take far too long,
    - too shallow and we may miss paths that make early sacrifices for later gain.
- Depth-first search is employed.
- We have to establish the quality of each state!

How do we assign a quality to each state?

- Difficult task!!!
- We use an heuristic which will actually depend on our experience with that game.

Example – Chess game

- Count the number of pieces left on the table. Compute the difference.
- Count the strength of the pieces that you have.
- Count the number of pieces that are attacking your pieces.
- ....

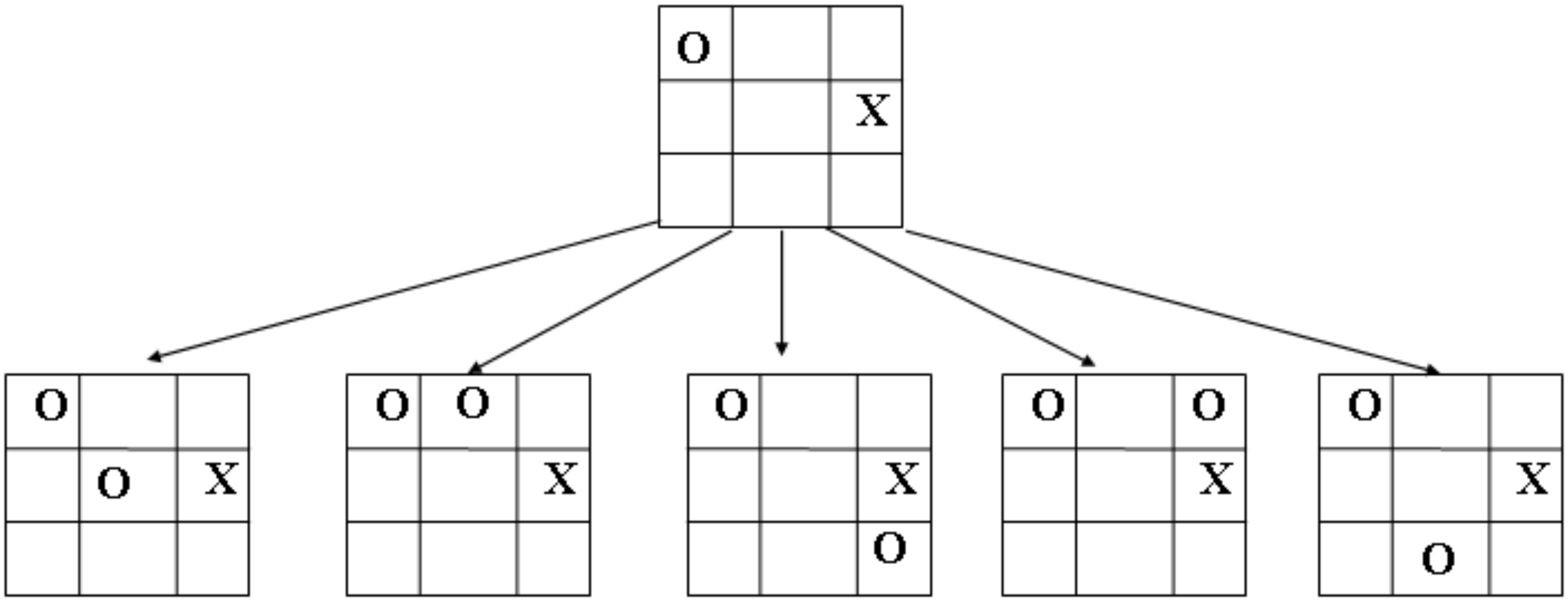
Deep Blue’s heuristics

- More than 50 highly-tuned heuristics provided by chess grandmasters.
- Different heuristics for different stages of the game.

Example – X and 0’s quality function

- If a player has two collinear board places and the 3rd space in the line is empty, then award that player 200 points. (In other words, if you almost have one complete line that is not blocked have 200 points).
- If a player has two nearly complete lines then score 300.
- If a player has a complete line score 600.
- Add onto these values:
  - The number of possible lines that can be completed by this player from this state.

Example



Values of each new state

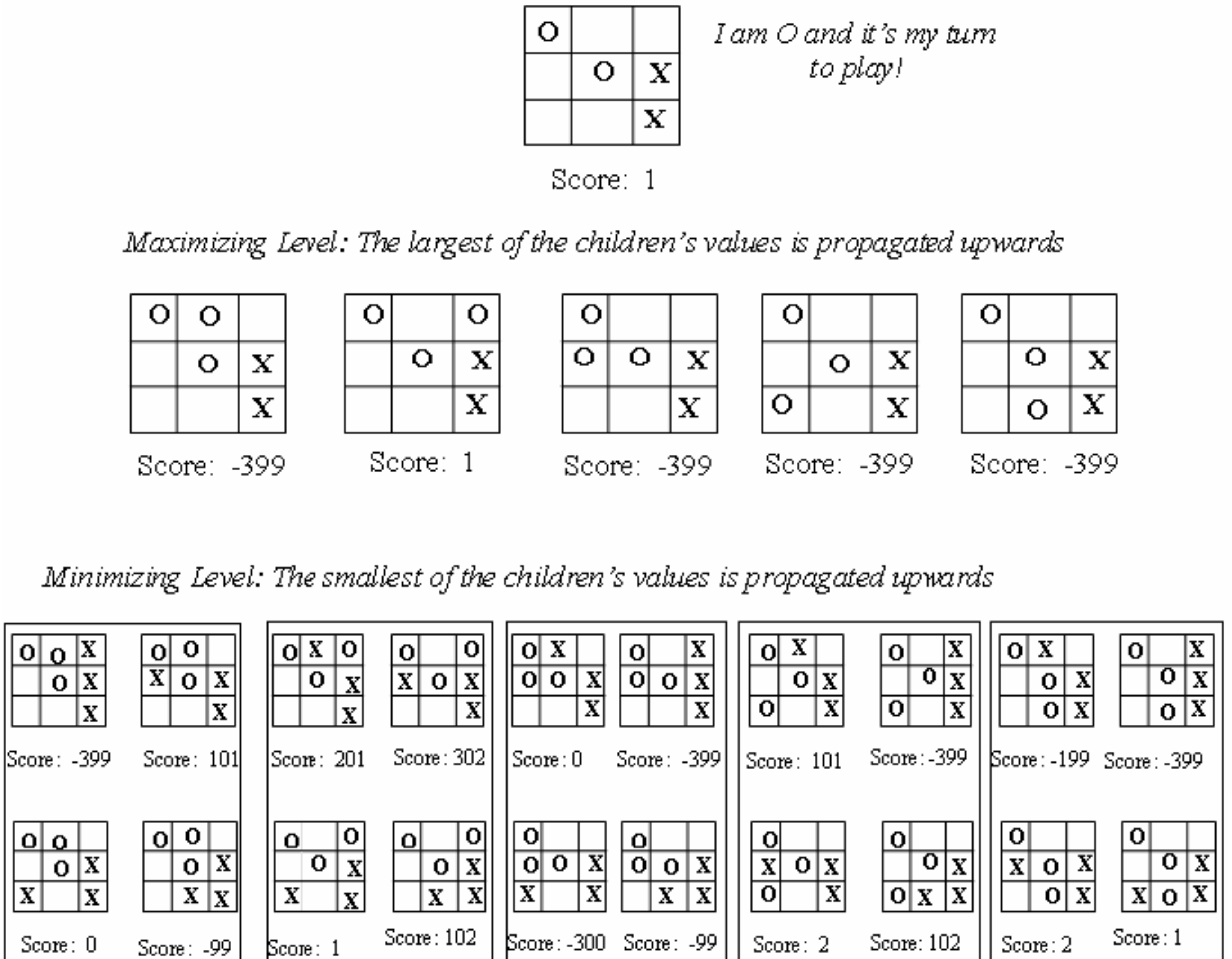
New State	1	2	3	4	5
O Sum	200+5	200+4	200+4	200+4	5
O Total	205	204	204	204	5
X Total	1	2	1	1	2
	------	------	------	-------	----
Total	204	202	203	203	3

MiniMax Algorithm – basic ideas

- First player will try to maximize its gain (also called MAX player).
- Second player will try to minimize the gain of the first player (called MIN player).

MiniMax Algorithm

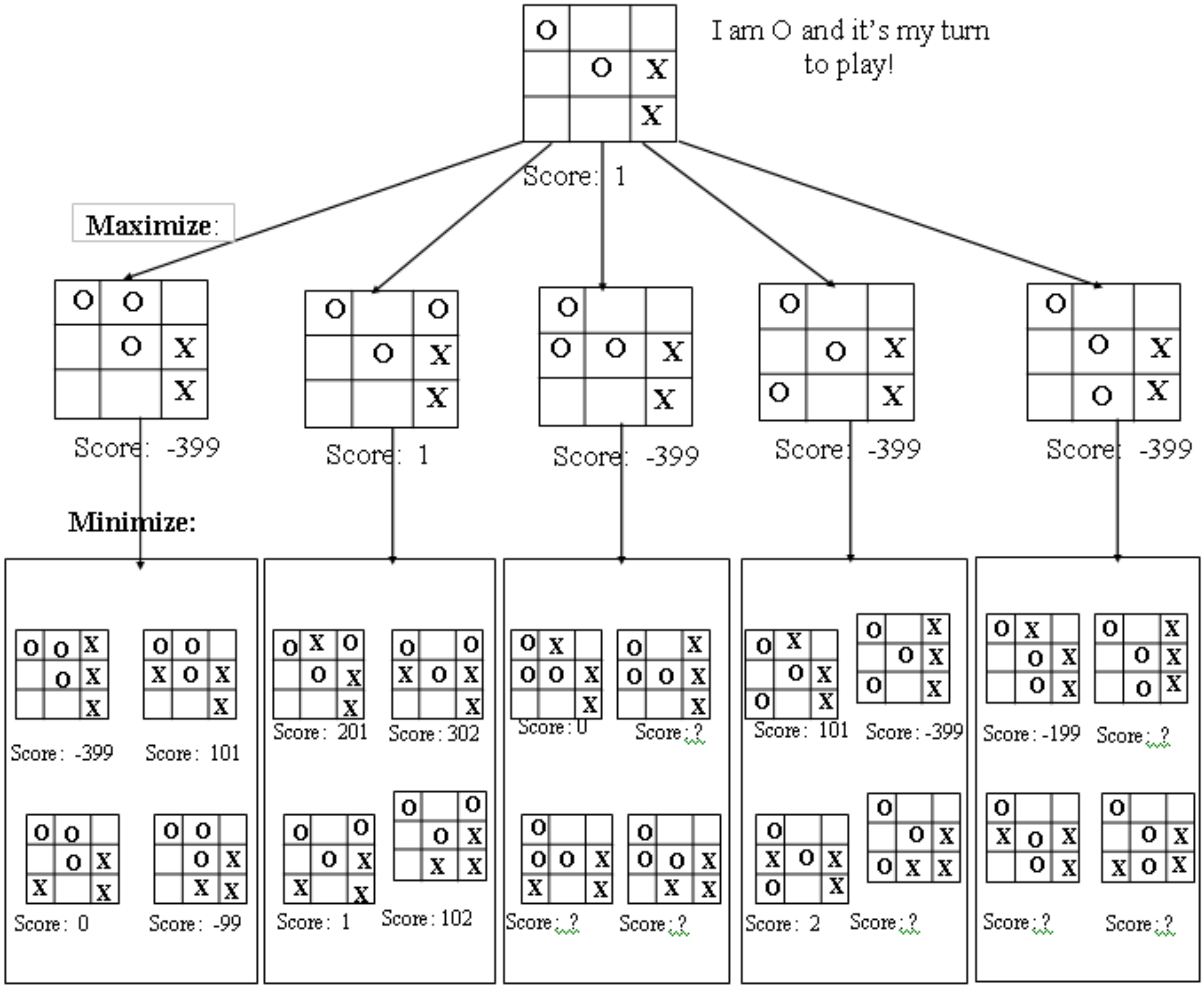
```
int MiniMax(Node N, int level){
        //level = 0 for the node in the top of the tree.
    if (level == MaxLevel)
        return the quality of N computed with the heuristic;
    if (level % 2){//B is about to move; minimize
        int result = MaxInt;
        for each child Ni of N
            result = Min(result, MiniMax(Ni, level+1));
    }
    else{ // A is about to move; Maximize
        int result = -MaxInt;
        for each child Ni of N
            result = Max(result, MiniMax(Ni, level+1));
    }
    return result;
}
```



I am O and it's my turn to play!
Score: 1
Maximizing Level: The largest of the children's values is propagated upwards
Score: -399
Score: 1
Score: -399
Score: -399
Score: -399
Minimizing Level: The smallest of the children's values is propagated upwards
Score: -399
Score: 101
Score: 0
Score: -99
Score: 201
Score: 302
Score: 1
Score: 102
Score: 0
Score: -399
Score: -300
Score: -99
Score: 101
Score: -399
Score: 2
Score: 102
Score: -199
Score: -399
Score: 2
Score: 1

Alpha-Beta Cutoffs

- We want to reduce the number of states that should be explored!
- Some of the branches may be avoided!


I am O and it's my turn to play!
Score: 1
Maximize:
Score: -399
Score: 1
Score: -399
Score: -399
Score: -399
Minimize:
Score: -399
Score: 101
Score: 0
Score: -99
Score: 201
Score: 302
Score: 1
Score: 102
Score: 0
Score: ?
Score: ?
Score: ?
Score: 101
Score: -399
Score: 2
Score: ?
Score: -199
Score: ?
Score: ?
Score: ?

Alpha-Beta Cutoffs

- ALPHA

  The best score that player MAX is guaranteed to obtain.
- BETA

  The minimal score that player MIN is guaranteed to obtain.
- We start with ALPHA = $-\infty$

  and BETA = ∞.

More on AB with cutoffs

- ALPHA value of a node
  - Initially it is the score of that node, if the node is a leaf, otherwise it is -∞. Then at a MAX node it is set to the largest of the scores of its successors explored up to now, and at a MIN node to the alpha value of its predecessor.
- BETA value of a node
  - Initially it is the score of that node, if the node is a leaf, otherwise it is +∞. Then at a MIN node it is set to the smallest of the scores of its successors explored up to now, and at a MAX node to the beta value of its predecessor.
- Score of a node: At MAX node, it is final ALPHA, at MIN is final BETA

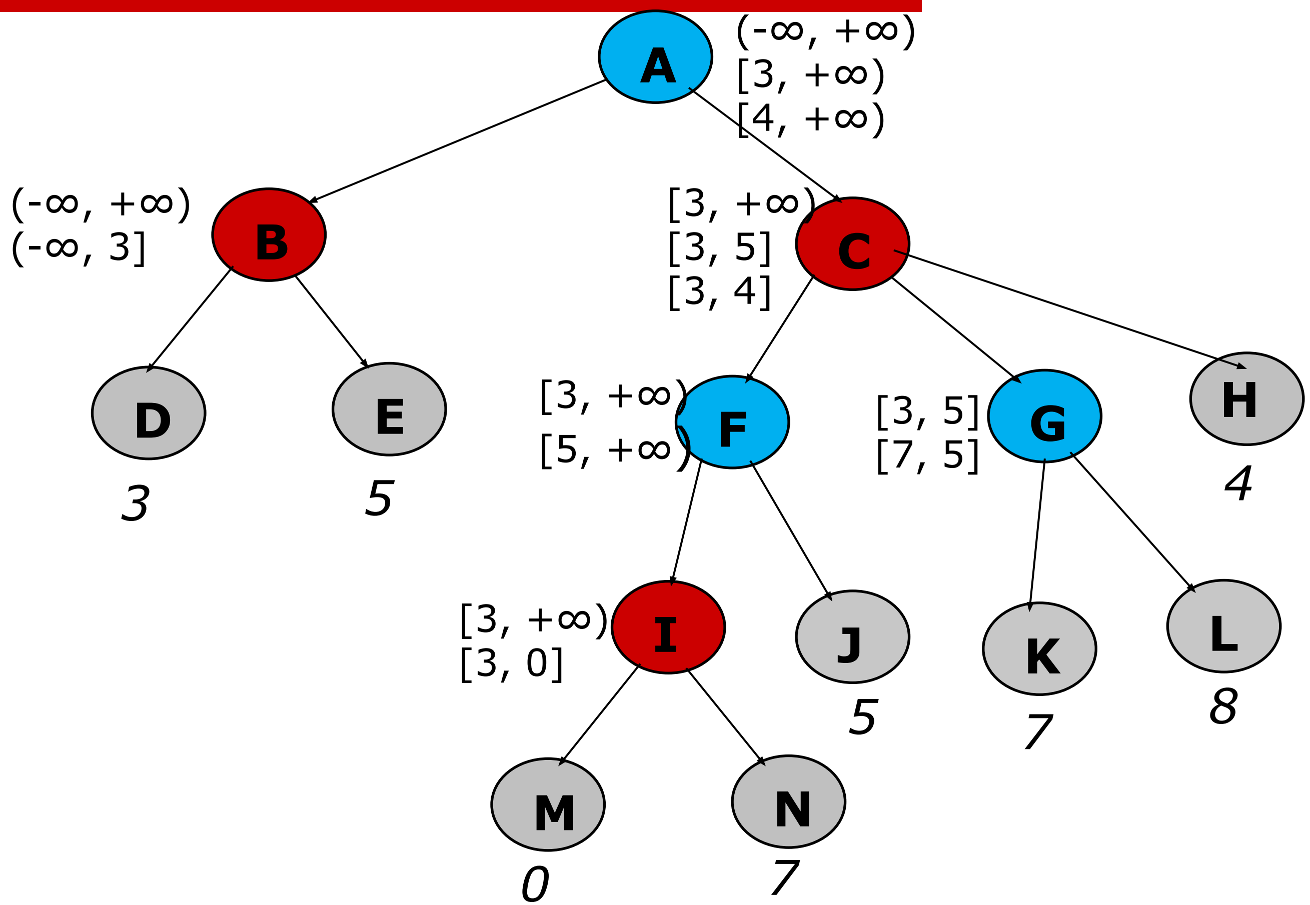
MiniMax with Alpha-Beta

```
function alphabeta(node, depth, α, β, maximizingPlayer)
    if depth = 0 or node is a terminal node
        return the heuristic value of node
    if maximizingPlayer
        v := -∞
        for each child of node
            v := max(v, alphabeta(child, depth - 1, α, β, FALSE))
            α := max(α, v)
            if β ≤ α
                break (* β cut-off *)
        return v
    else
        v := ∞
        for each child of node
            v := min(v, alphabeta(child, depth - 1, α, β, TRUE))
            β := min(β, v)
            if β ≤ α
                break (* α cut-off *)
        return v
```

```
alphabeta(origin, depth, -∞, +∞, TRUE) (* Initial call *)
```

https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning#Pseudocode

Labeled tree



Unvisited nodes: N, L

Strength of MiniMax with AlphaBeta cutoffs

- Can explore larger trees than MiniMax only.