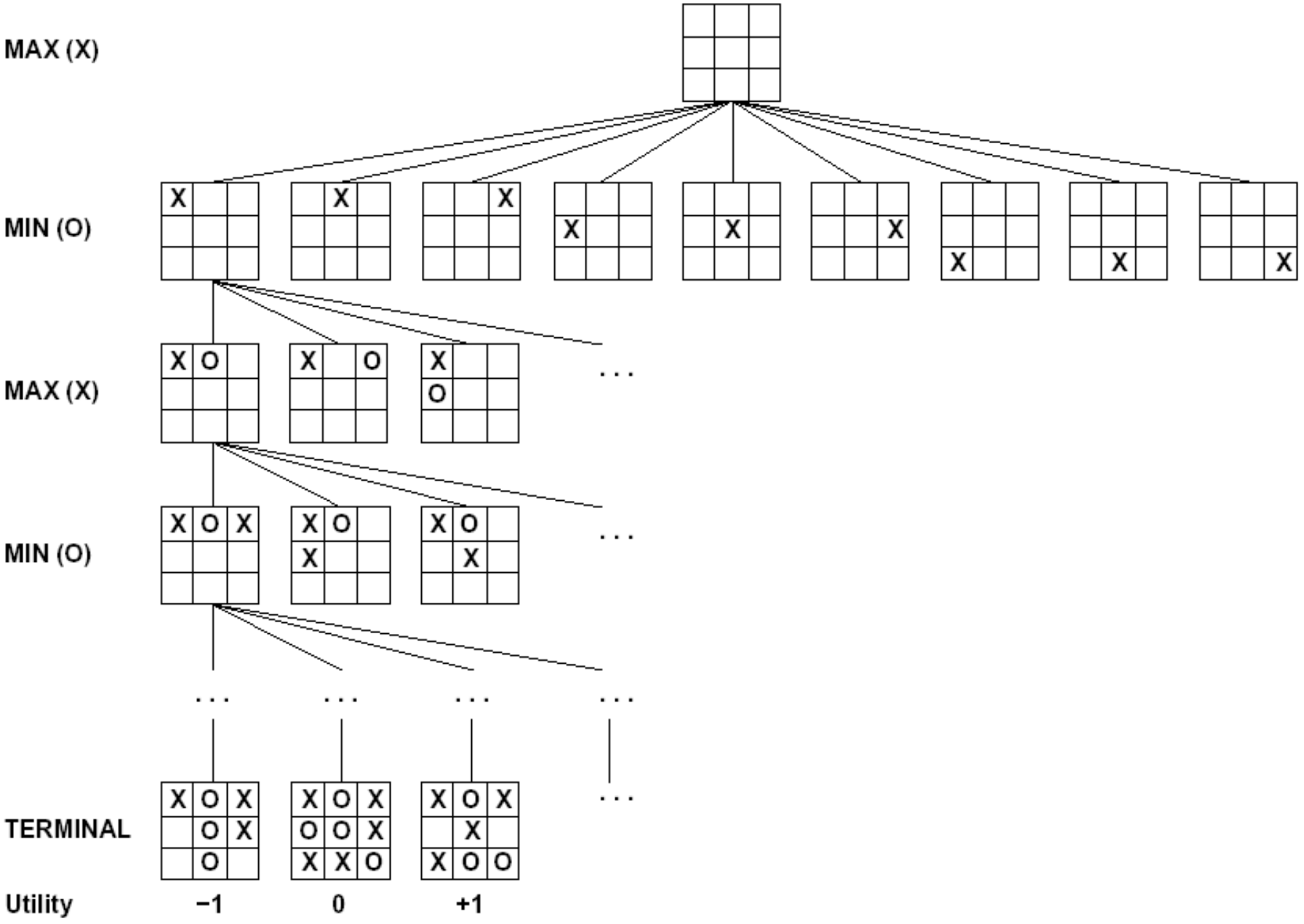
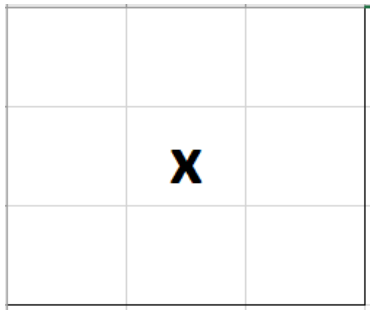
Tic-Tac-Toe-Example for Games Algorithms

- Evaluation-Function
- Alpha-Beta-Pruning
- Monte-Carlo Tree Search

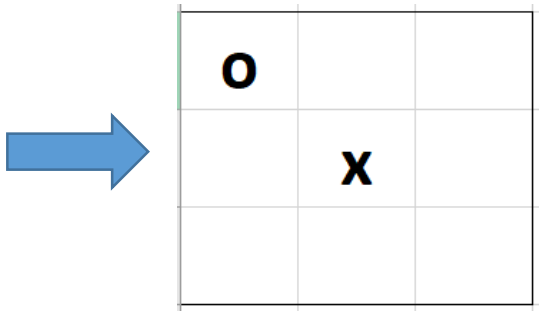


Evaluation Function

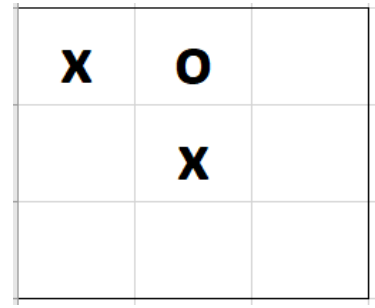
(Number of own chances to win) – (number of opposite chances)



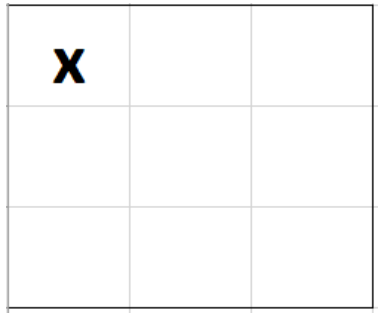
8 – 4 = 4



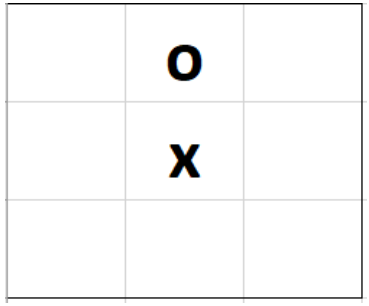
5 – 4 = 1



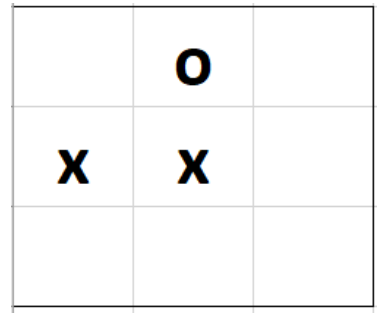
6 – 2 = 4



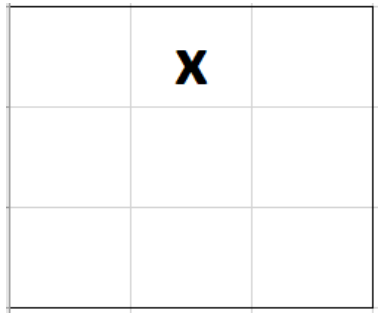
8 – 5 = 3



6 – 4 = 2



6 – 3 = 3

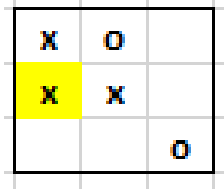


8 – 6 = 2

Improvement (idea similar to Quiescense Search):

If: Immediate-Win then 100

elseif: Double-Double then 10

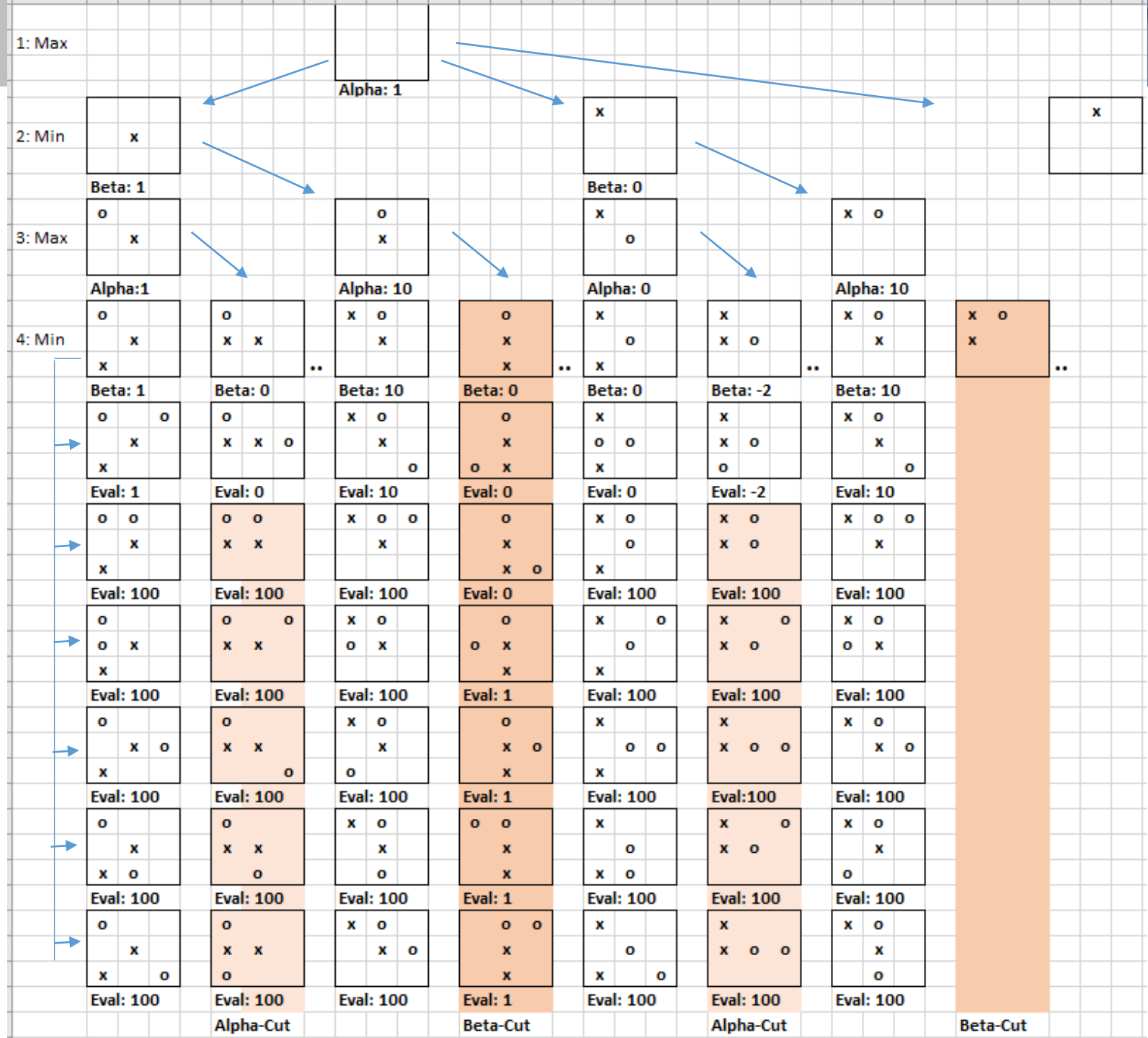


else: (Number of own chances to win) – (number of opposite chances)

Example

- … for Alpha-Cut (light orange)
- … for Beta-Cut (full orange)

- With search depth of 4



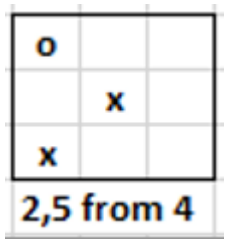
Monte Carlo Tree Search

```
function MONTE-CARLO-TREE-SEARCH(state) returns an action
  tree ← NODE(state)
  while IS-TIME-REMAINING() do
    leaf ← SELECT(tree)
    child ← EXPAND(leaf)
    result ← SIMULATE(child)
    BACK-PROPAGATE(result, child)
  return the move in ACTIONS(state) whose node has highest number of playouts
```

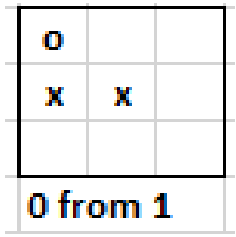
Selection strategy:

$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(\text{PARENT}(n))}{N(n)}}$$

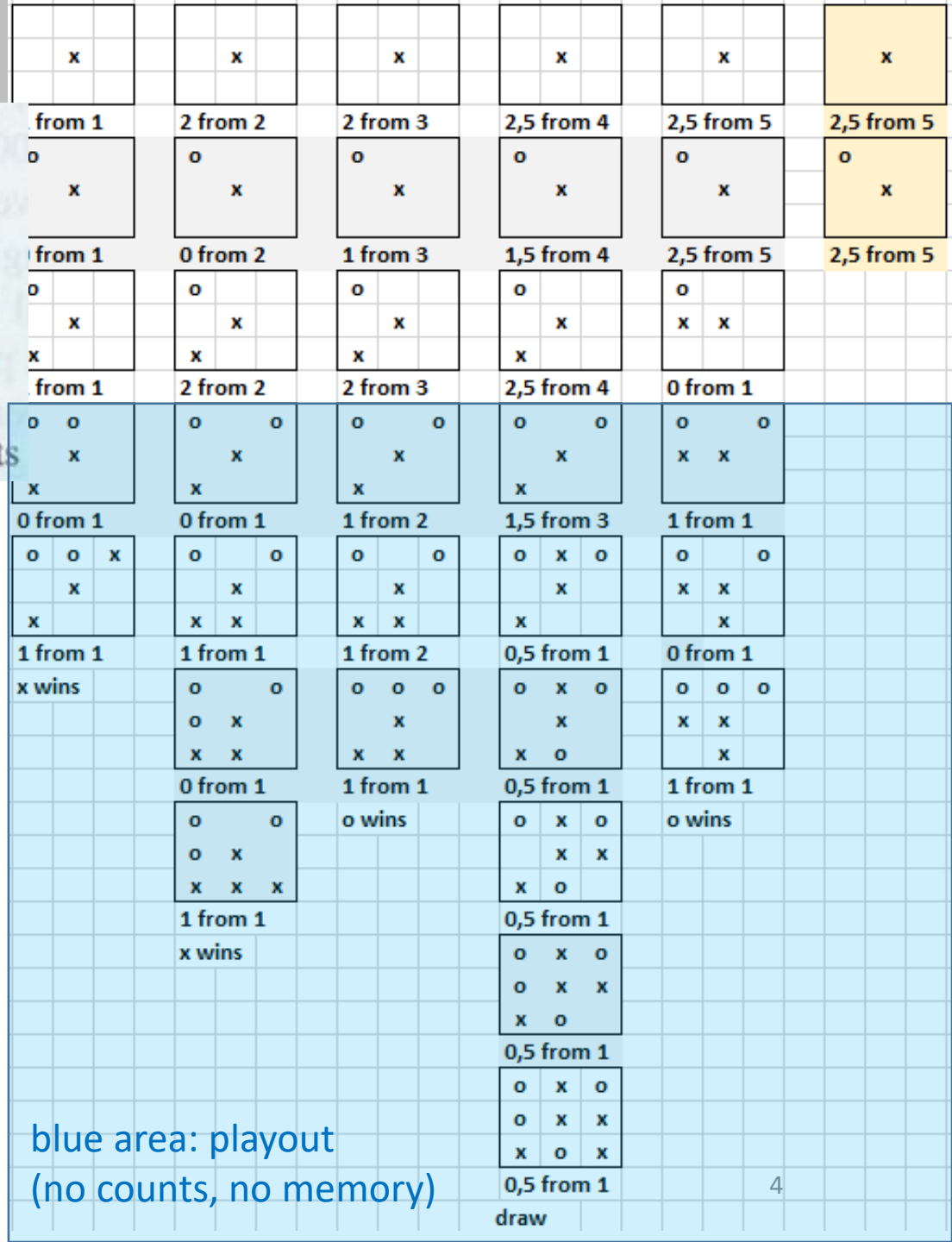
U(n) = Utility of node n
N(n) = Number of visits
C = Constant, e.g √2
U(n) = # Wins



5/8+√2*√(log5/4) ≈
0,625 + 1,4*1,6/4 ≈
0,625 + 0,56 ≈ **1,2**



0 +√2*√(log5/1) ≈
0 + 1,4*1,6/1 ≈
0 + 2,24 ≈ **2,2**



blue area: playout
(no counts, no memory)