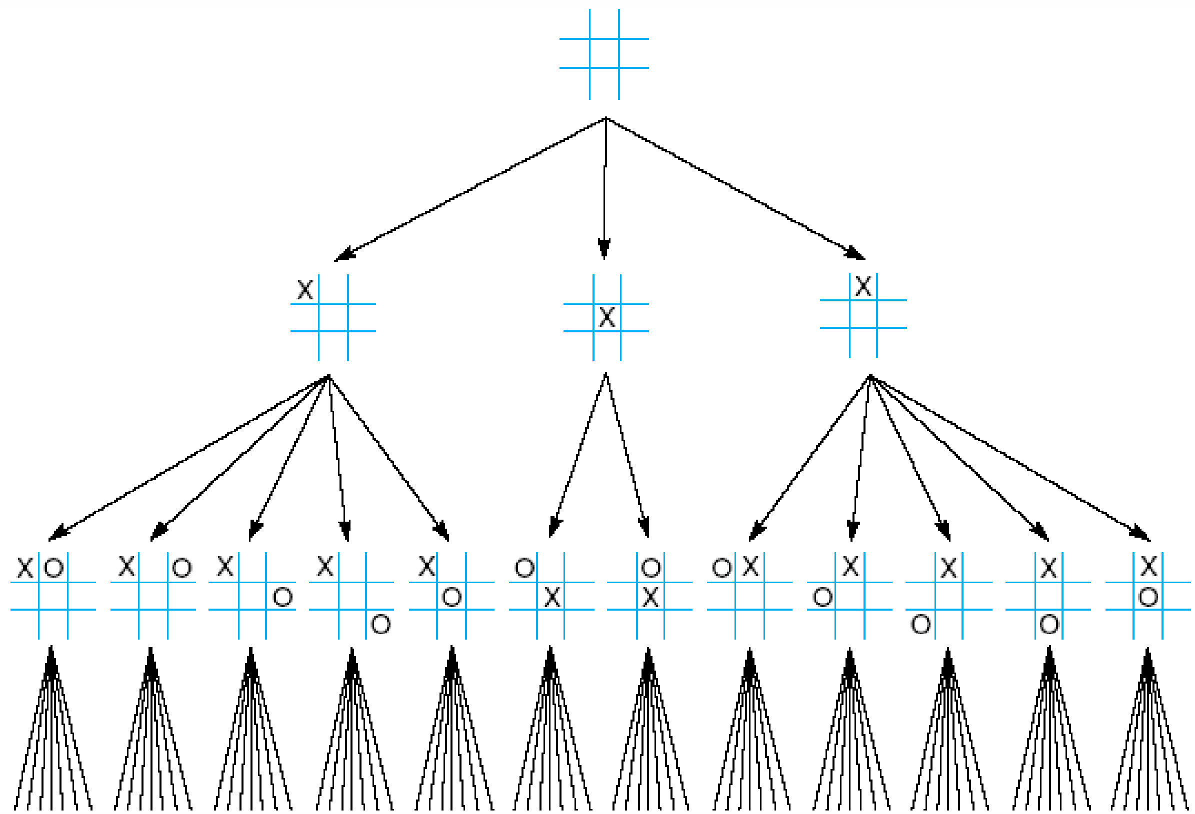
Game Playing-MINMAX Alg



Game Playing

- Major topic of AI requiring intelligence
- What is a game?
  - Sequence of choices
  - Where each choice is made from a number of available discrete alternatives
  - Each sequence ends in an outcome
  - Two players game with alternate moves
    - MAX player
    - MIN player
  - Each player has opposite goals
  - Two types
    - Perfect information game---chess,tic-tac-toe
    - Imperfect information game---- cards games

Game playing -MINMAX

- Searching a game tree using OR graph with two types of nodes
  - Min Node: *select minimum cost successor*
  - Max Node: *select maximum cost successor*
  - Terminal nodes: winning or loosing states
  - Depth first depth limited search
  - Easy to design and understand
  - Typical characteristic of these games
    - *Look ahead* in future position to succeed
    - Exhaustive search can be done if there is no constraint on time and space
    - Usually impossible to search upto terminal nodes

Game playing and state space

- Game playing can be represented as normal search procedure where goal is to win or to loose
- Game can begin in an initial state and end in win for one, loss for another or draw

Sate Space problem	Game problems
States	Legal Board Positions
Rules	Legal Moves
Goals	Winning positions

- Main
  - Minimizing the maximum possible loss
  - Maximizing the minimum gain
  - One player is computer—maximizer
  - Other player –human--minimizer

Game plying and state space

- Two levels –maximizing level and minimizing level
- Node at each level are termed as MAX and MIN
- MAX node will try to maximize its own game and minimize its opponent (MIN's) game
- Any one can play first,   we assume MAX node will start
- Game trees are labeled MIN or MAX alternatively at each level or ply
- Cost of leave nodes is calculated using evaluation function
- Once leaf nodes are labeled, the non leaf node are labeled as either
  - Maximum (successor nodes) for maximizer
  - Minimum(successor nodes) for minimizer

Win or Loss at any node j

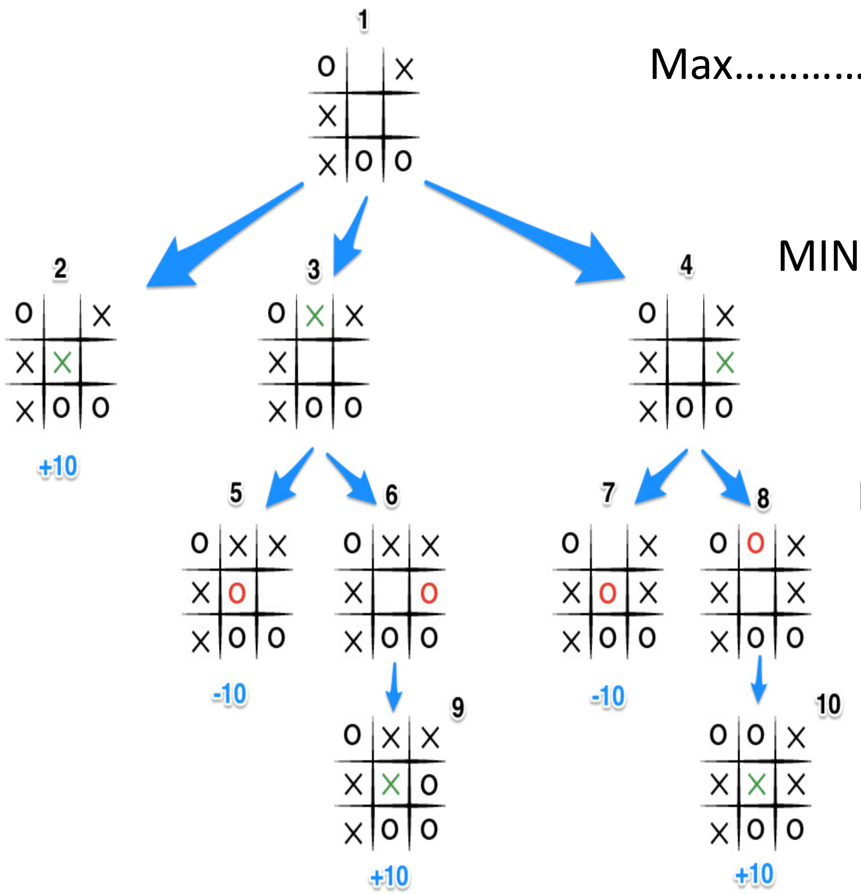
- If terminal nodes are labeled W (+10) for win L(-10) for maximizer or draw
- Labeling procedure for a non terminal node j is as follows
- For non terminal MAX node

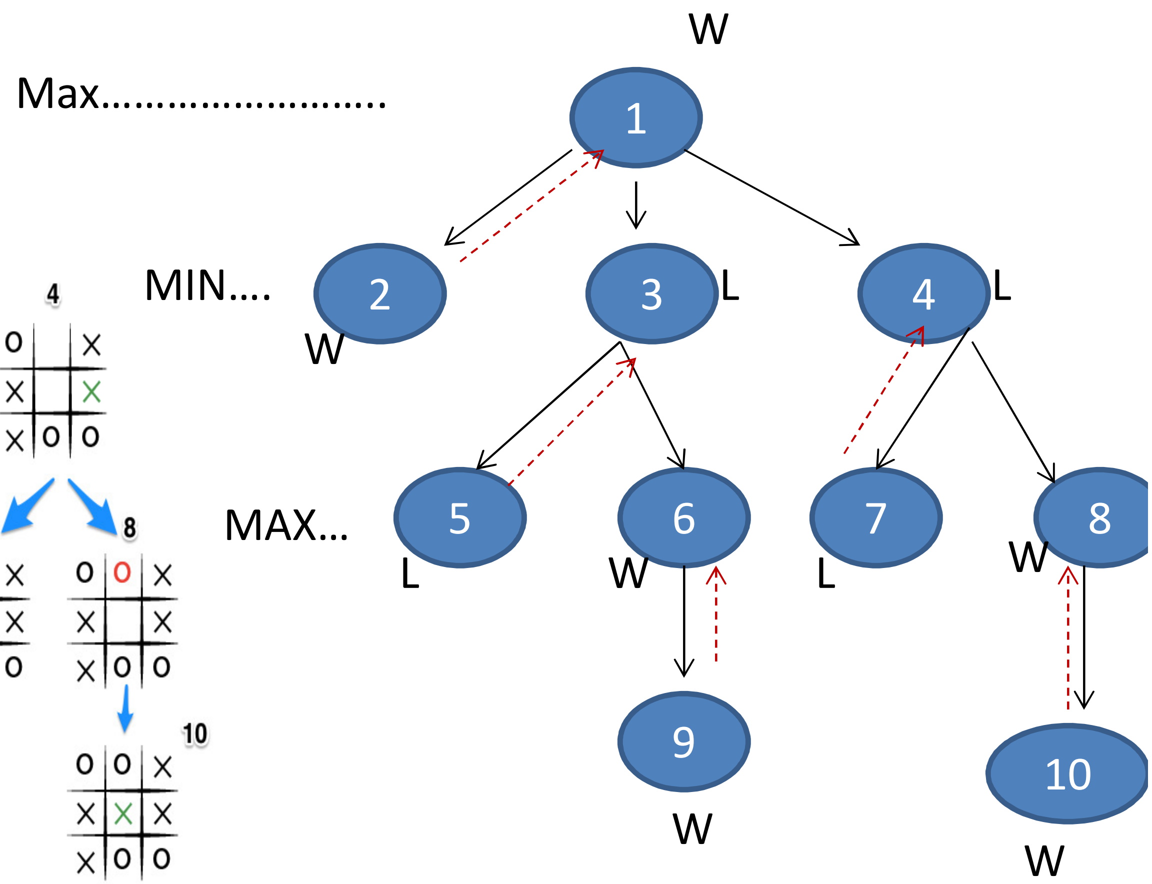
Status(j) =
- W if any of j's successor is a win
- L is all the j's successor are loss
- draw is any one of successor of j is draw and none is win

- For non terminal MIN node

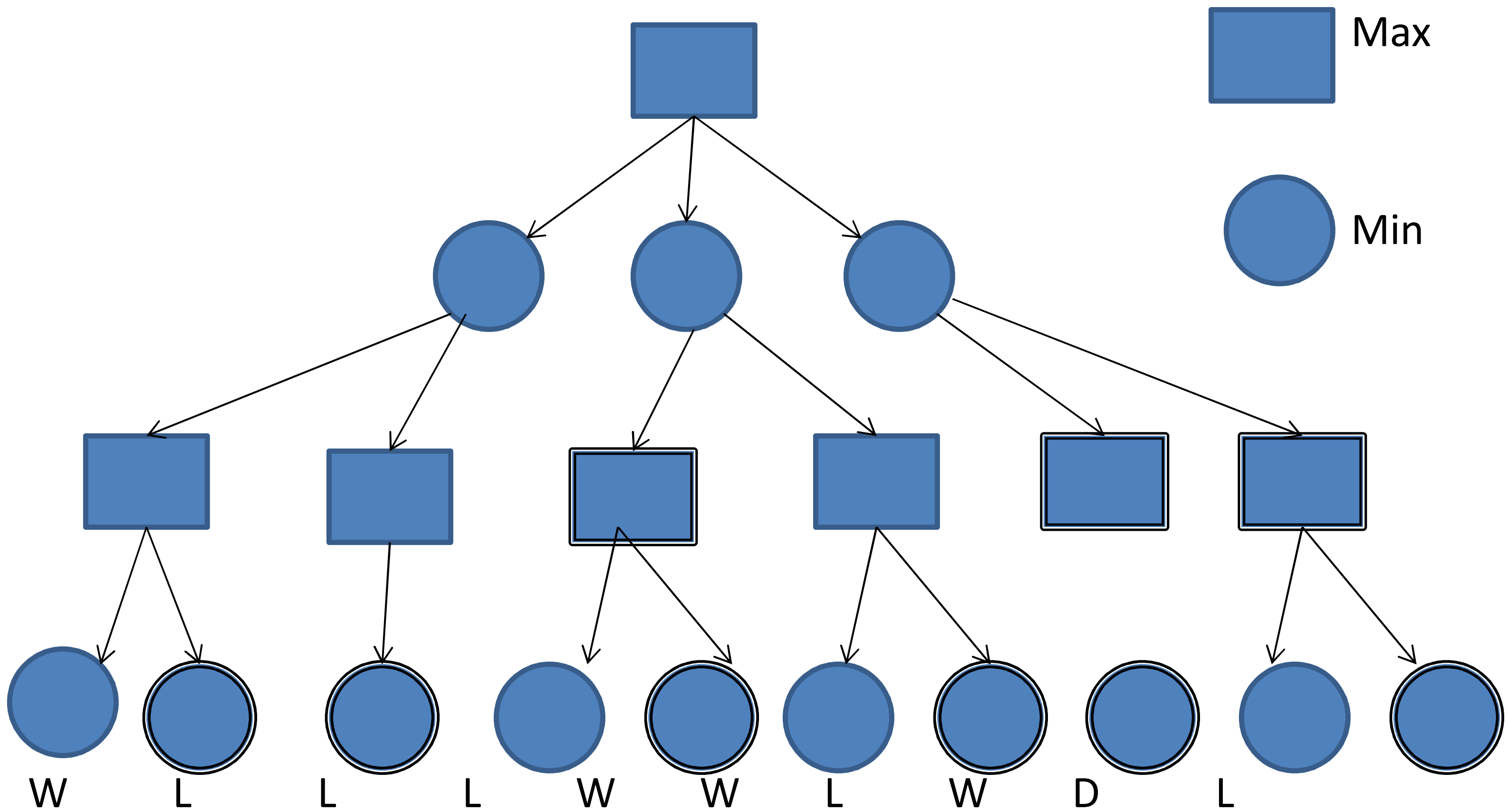
Status(j) =
- L if any of j's successor is a loss
- W is all the j's successor are win
- draw is any one of successor of j is draw and none is loss

Tic-tac-toe


1
2
3
4
+10
5
6
7
8
-10
-10
9
10
+10
+10


Max........................
MIN....
MAX...
W
1
2
3
4
W
L
L
5
6
7
8
L
W
L
W
9
10
W
W

Find the label as W/L/D at root node if W and L are win and Loss for Max resp



Game Playing

- Two main reason that game appears to be a good domain for machine intelligence:
  - Provide a structured task in which it is easy to find success or failure
  - Did not require large amount of knowledge
- Second reason is true only for small games (tic-tac-toe)
- Example Chess
  - Average branching factor = 35
  - On an average game one player can make 50 moves
  - To examine a complete tree we would have to examine 35^{100} positions

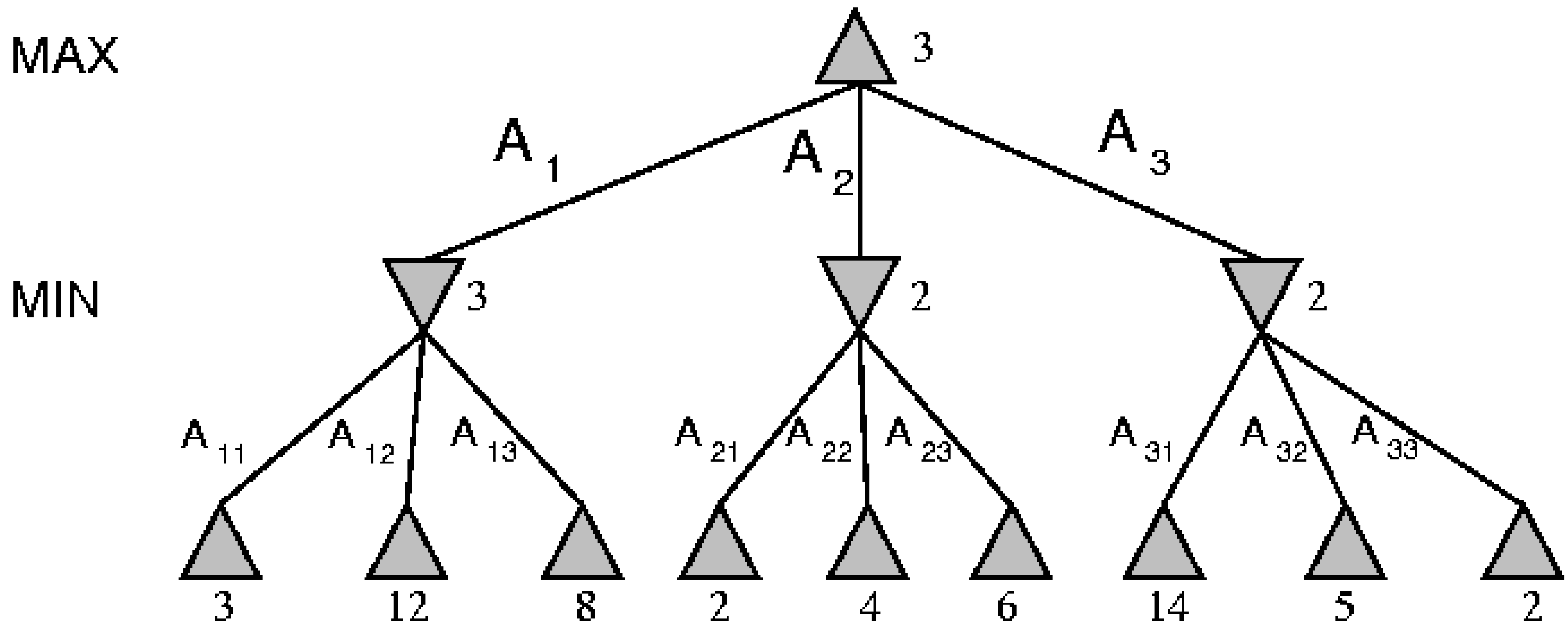
Game Playing

- Use *Plausible move generator* in which small number of promising move are generated
- With such a generator, test procedure can spend more time in evaluating these nodes.
- Search procedure will find a path till goal is reached
- Goal: winning / loosing state
- Even with plausible generator, in games like chess, it is not possible to search till goal is found.

Game Playing

- Like heuristics , use **static evaluation function** to find the best node
  - Tic-tac-toe = number of winning moves for X- number of winning moves for O
  - Chess: add number of blacks(w), add number of whites(W) quotient W/B
- Two concern parameter in game search is **depth** and **branching factor**
- Also called **zero sum game** as
  if A gets more B will get less
  If A win B will loose

Minimax

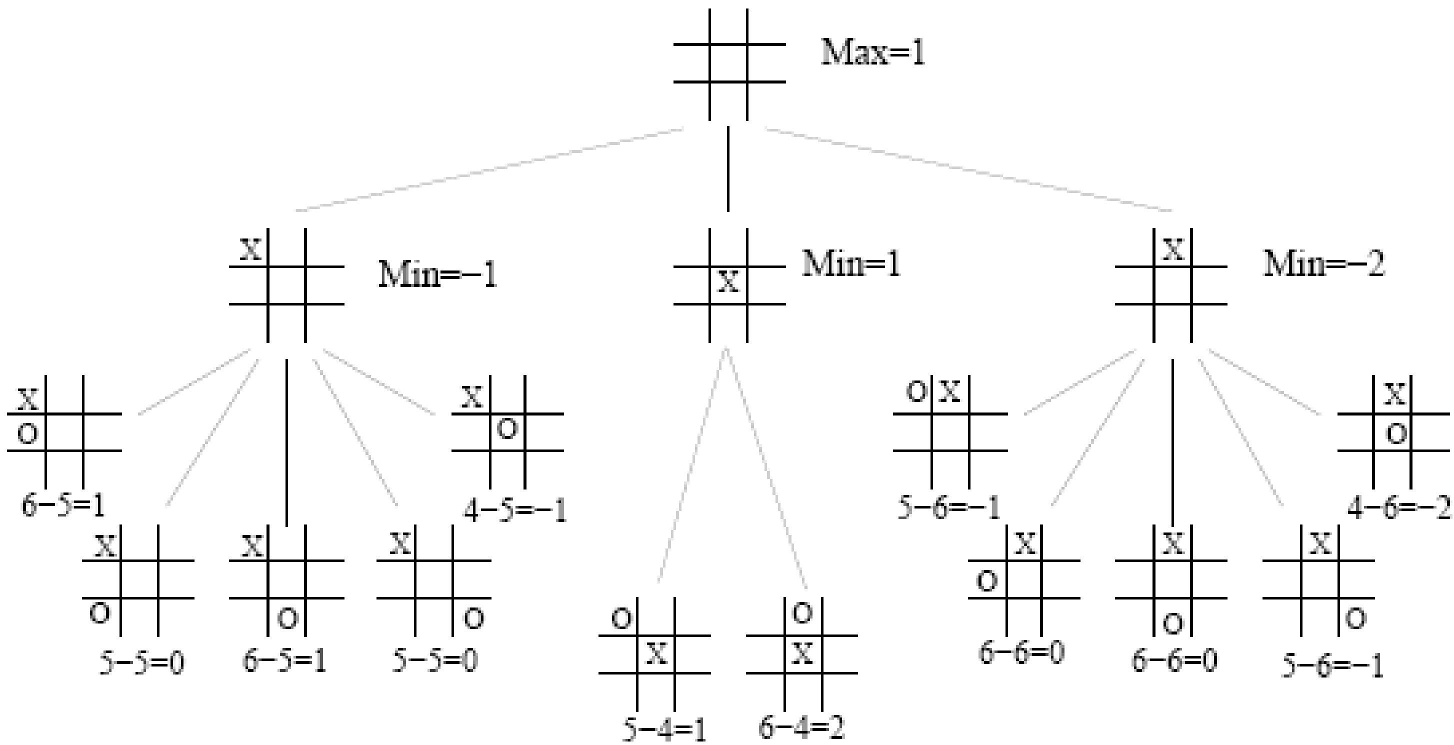


- Minimax-value(n): utility for MAX of being in state n, assuming both players are playing optimally =
  - Utility(n), if n is a terminal state
  - $\max_{s \in \text{Successors}(n)}$ Minimax-value(s), if n is a MAX state
  - $\min_{s \in \text{Successors}(n)}$ Minimax-value(s), if n is a MIN state

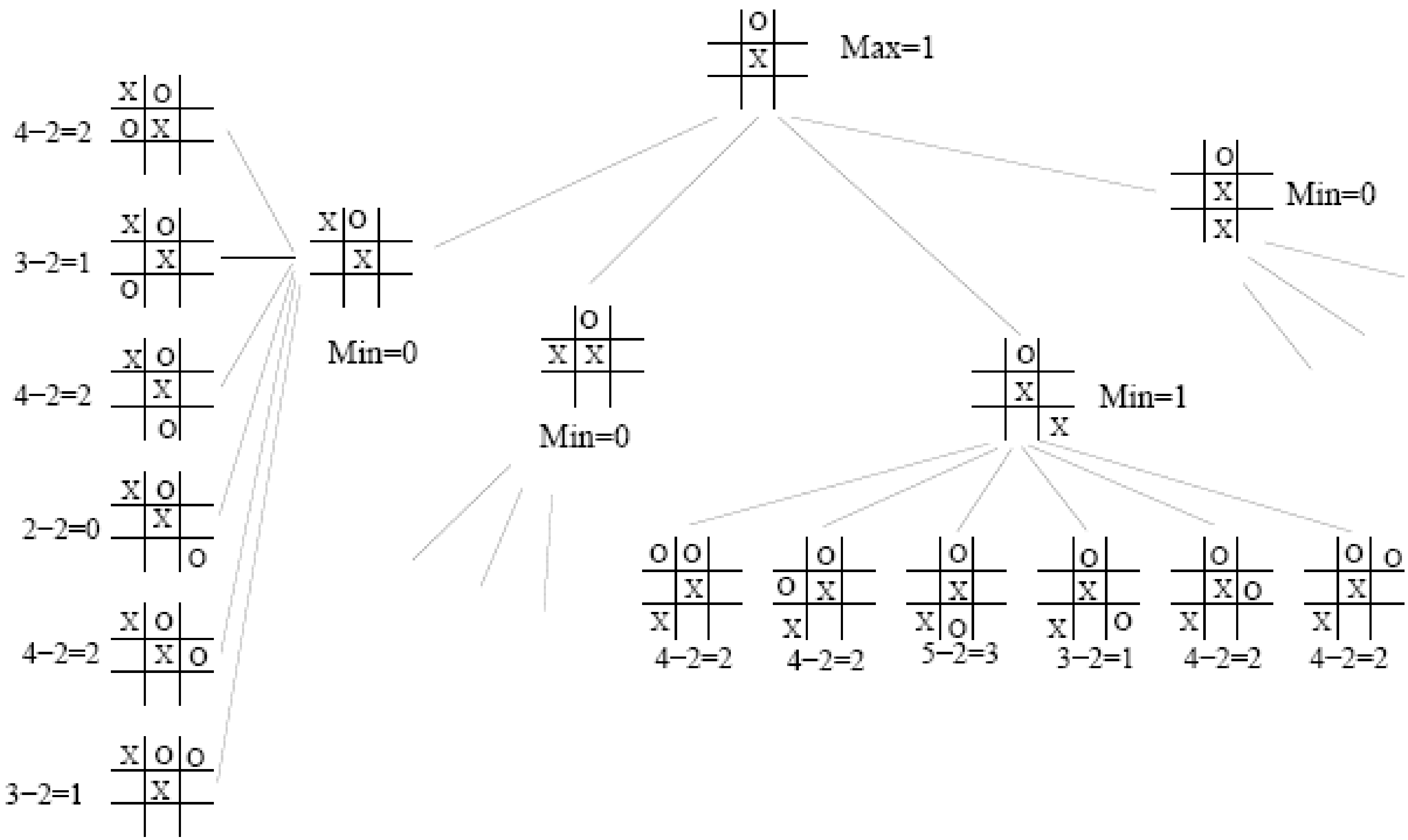
Example: tic-tac-toe

- e (evaluation function → integer) = number of available rows, columns, diagonals for MAX - number of available rows, columns, diagonals for MIN
- MAX plays with “X” and desires maximizing e.
- MIN plays with “0” and desires minimizing e.
- Symmetries are taken into account.
- A depth limit is used (2, in the example).

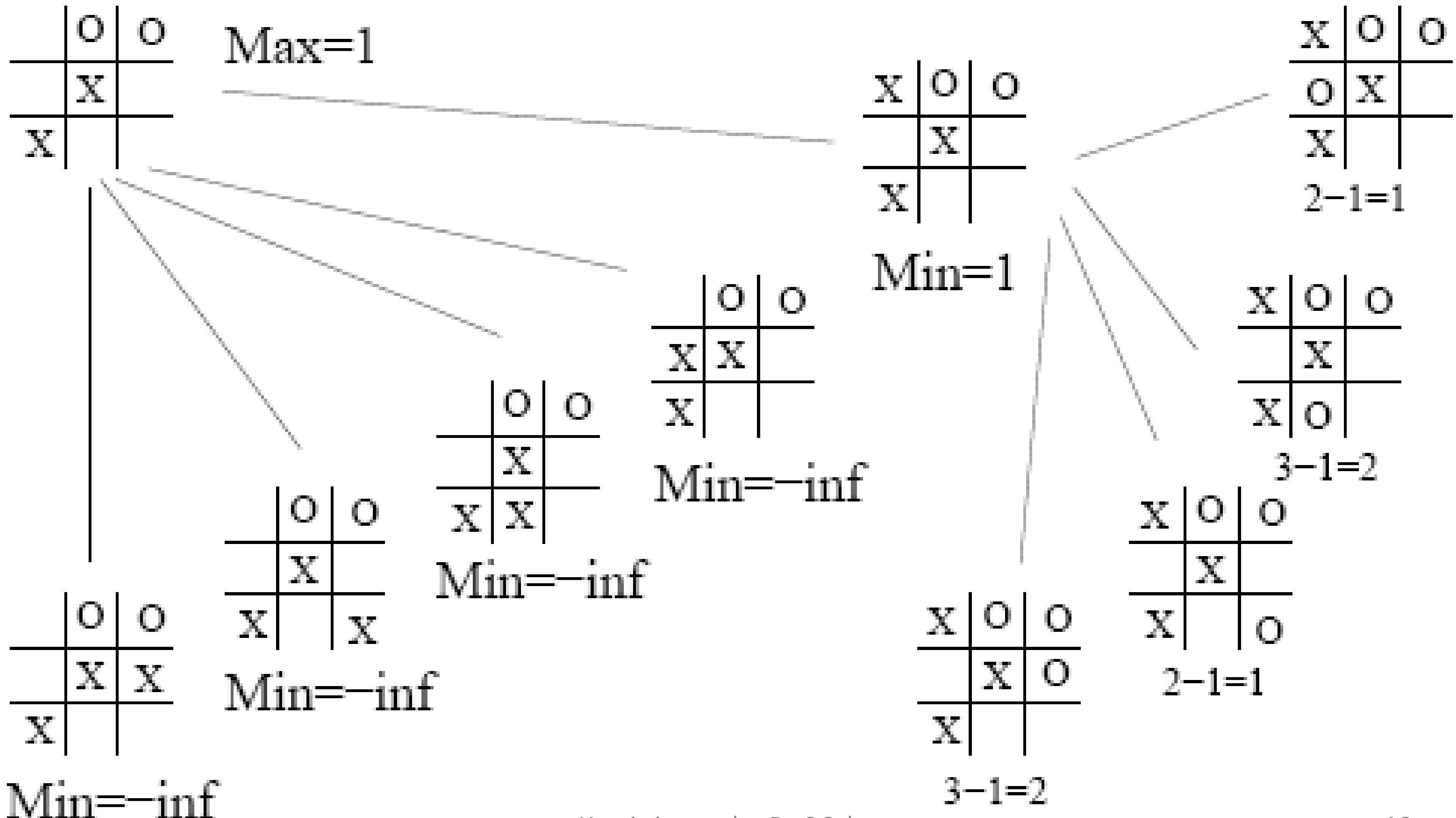
Example: tic-tac-toe



Example: tic-tac-toe



Example: tic-tac-toe



The minimax algorithm

- It is depth first depth limited search
- Uses *plausible move generator* to generate the promising successor(moves/nodes)
- Uses *Static evaluation function* with large values indicating ‘good’ values for us

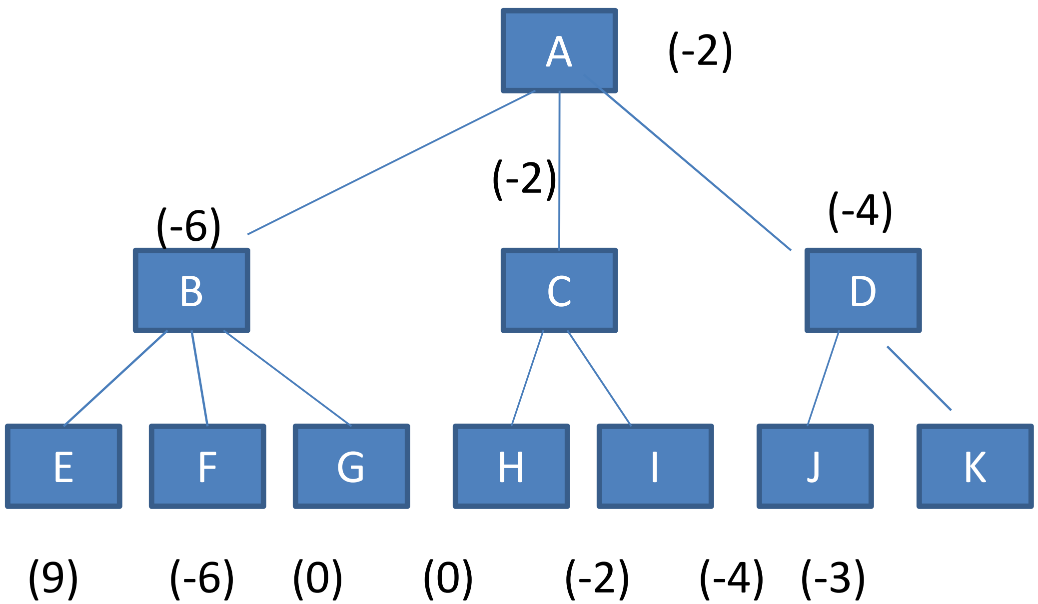
  +10 -------------------------- -10

  Win for us win for opponent

- Value generated by static evaluation function is backed up to parents node
- Since static evaluation function is an estimate , it is better to carry search more than one ply in game tree –”*look ahead”*

The minimax algorithm

- Our goal is to maximize our move
- Opponent goal is to minimize our move
- Hence the name – minmax algorithm



Initial call to minmax will be minmax(current,0,p1)-if player 1 starts otherwise minmax(current,0,p2)

The minimax algorithm

•Requirement:

• MOVEGEN (Position Player) – generates list of nodes representing the moves that can be made by player in position

•STATIC(Position , Player) – returns a number representing goodness of position with respect to player

•When to stop recursion ?

Uses function DEEP ENOUGH which returns

true if search can be stopped

False otherwise

•Return value by the algorithm

Value : backed up value

Path itself

Hence a structure containing both the values is returned

```
Minmax (position, depth, player )
{
step1:If  DEEP-ENOUGH(position,player) then return structure
            VALUE=STATIC(position , player)
            PATH=nil
Step2:Else
       {
            SUCCESSORS=MOVE_GEN(position,player)
Step3:        if (SUCCESSOR= empty)  return structure // same as DEEP_ENOUGH

Step4:         if (! Empty(SUCCESSORSvalue ))
             {  BEST_SCORE=min that STATIC can generate
                For every SUCC of SUCCESSOR
     4(a)         {   RESULT_SUCC= MINMAX(SUCC, depth+1,OPPOSITE(player)
     4(b)                     NEW-VALUE= - VALUE(RESULT_SUCC)
      4(c)                if NEW-VALUE > BEST_SCORE
         (i)           {     BSET_SCORE=NEW-VALUE
            (ii)                 BEST-PATH= SUCC + PATH(RESULT_SUCC)
                                    path from CURRENT to SUCC .
                       }// end if
                    }// end for
```

```
Step5:              value=BEST_SCORE
                      path=BEST_PATH
                 return structure
              }// end if
        }// else
  }// minmax
```

Minmax(A,0,p1)

1 false

2 move-gen(A,p1)

3 F

4 succ=(B,C,D)

for B

result_suc=minmax(B,1,p2)

1 F

2 move-gen(b,p2)

3 F

4 succ=(E,F,G)

for E

result_succ=minmax(E,2,p1)

1. T

val=static(E,p1)=9

path=nil

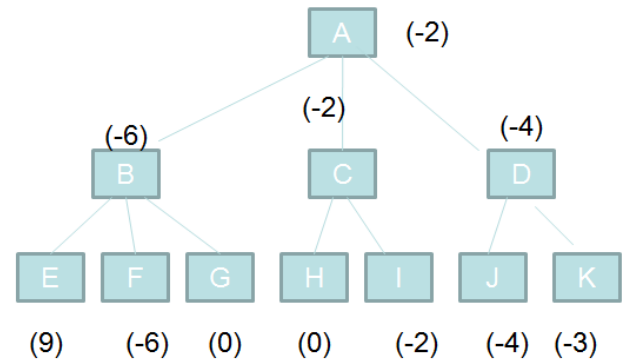
return (path=nil,val(9))

newval=-9

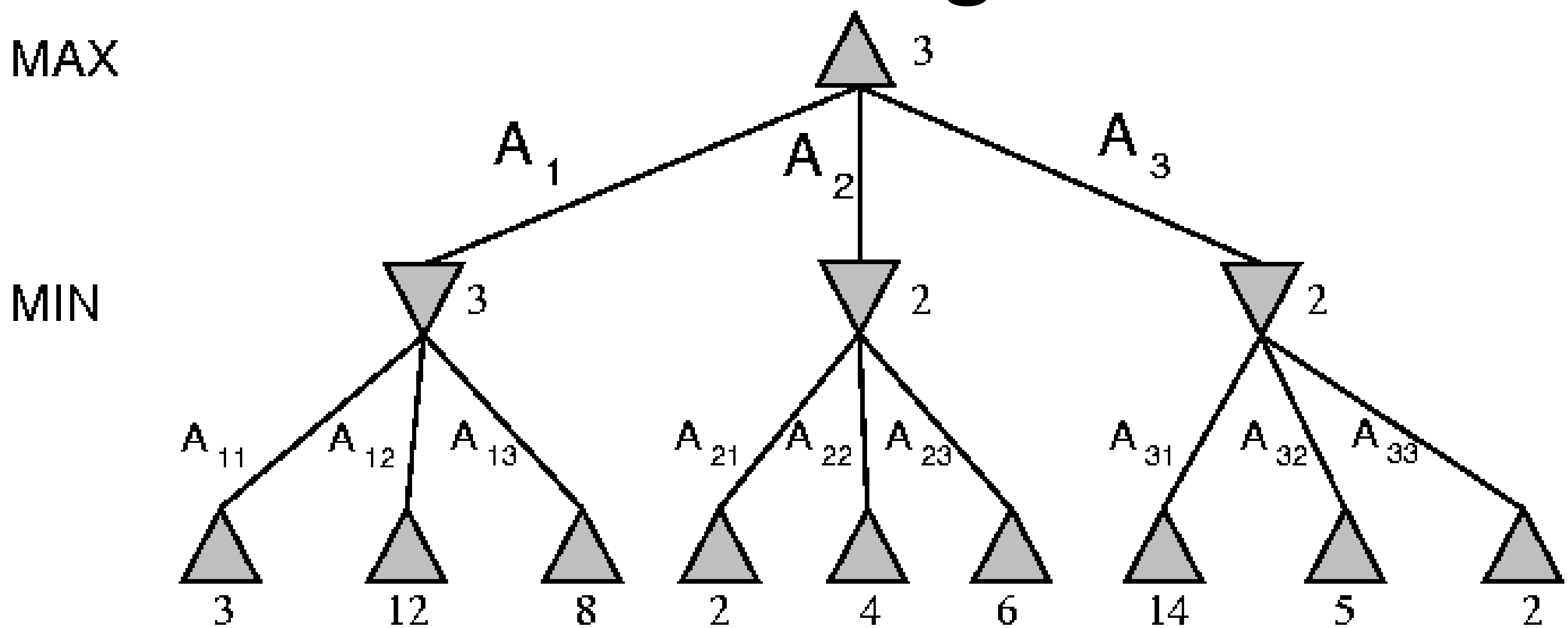
best-score=-9

best-path=E

return(BP,BS)



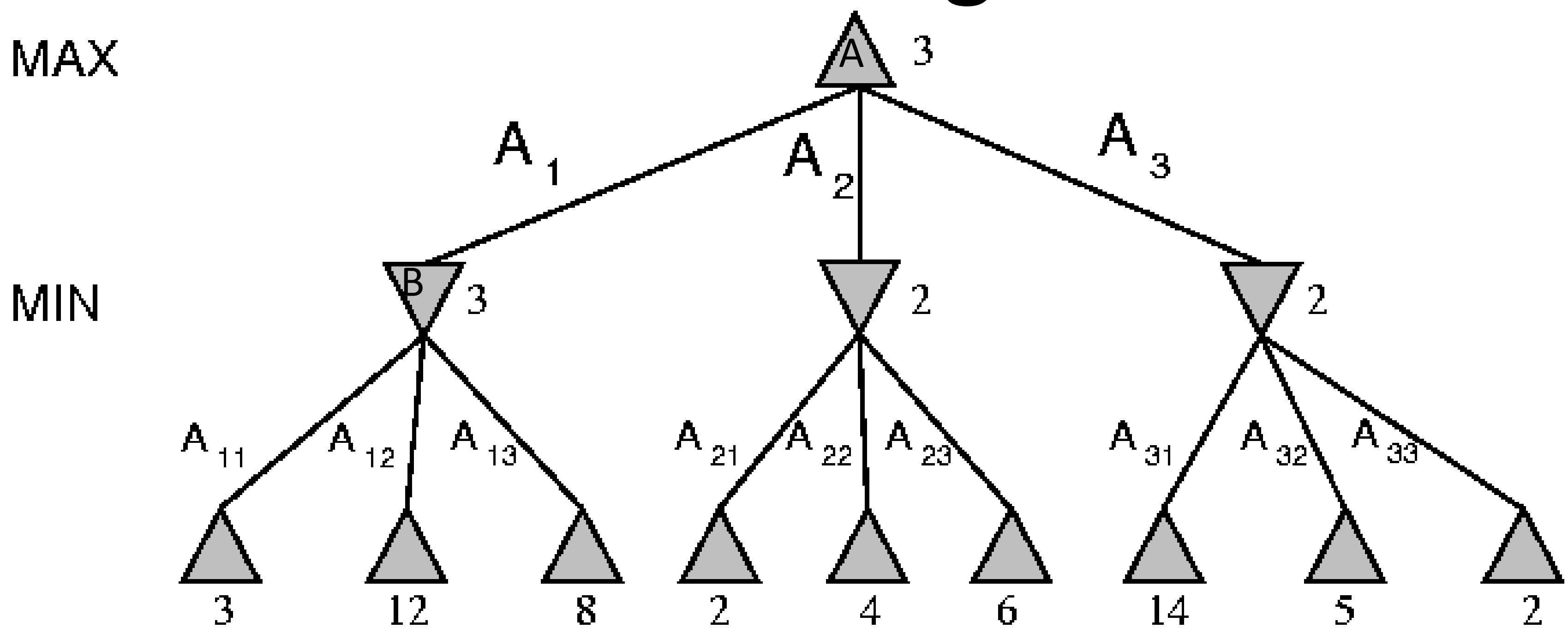
The minimax algorithm



- The algorithm first recurses down to the tree bottom-left nodes
  - and uses the *Utility* function on them to discover that their values are 3, 12 and 8.

23

The minimax algorithm



- Then it takes the minimum of these values, 3, and returns it as the backed-up value of node B.
- Similar process for the other nodes.

The minimax algorithm

- The minimax algorithm performs a complete **depth-first exploration** of the game tree.
- In minimax, at each point in the process, only the nodes along a path of the tree are considered and kept in memory.

The minimax algorithm

- If the maximum depth of the tree is *m*, and there are *b* legal moves at each point, then the **time complexity** is $O(b^m)$.
- The space complexity is:
  - *O(bm)* for an algorithm that generates all successors at once
  - *O(m)* if it generates successors one at a time.

The minimax algorithm: problems

- For real games the time cost of minimax is totally impractical, but this algorithm serves as the basis:
  - for the mathematical analysis of games and
  - for more practical algorithms
- Problem with minimax search:
  - The number of game states it has to examine is exponential in the number of moves.
- Unfortunately, the exponent can’t be eliminated, but it can be cut in half.