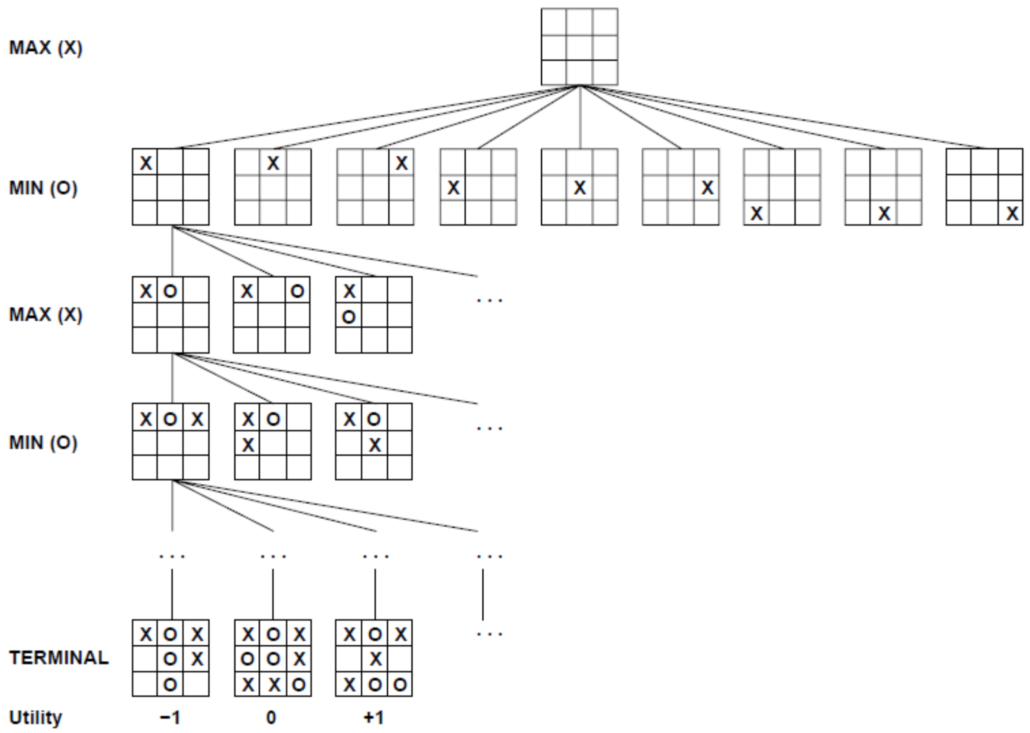
Adversarial Search

Games

- Games are competitive environments where agents' goals are in conflict.
- In this course, we focus on: **zero-sum games**.
- It is deterministic, full observable environments in which the two players (Max and Min) act alternately.
- Terminal game states have a utility *u*. Max tries to maximize *u*, Min tries to minimize *u*. So, the utility for Min is the exact opposite of the utility for Max. A terminal state is reached after a finite number of steps.

Example: tic-tac toe game



The tree of tic-tac-toe game have 9!= 362,880 terminal nodes. But for chess there are 10^{40}.

To compute the optimal strategy, we use *Minimax algorithm*.

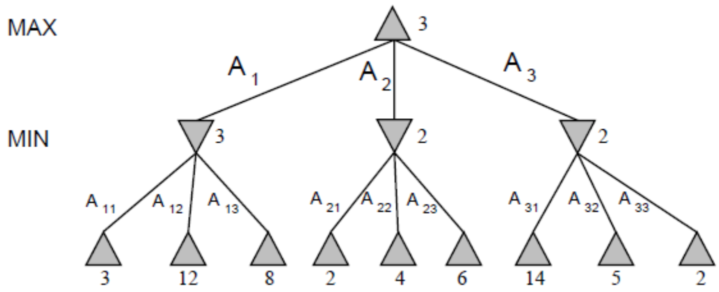
Minimax Algorithm

It is a Depth-first search in game tree, with Max in the root.

1- Apply utility function to terminal positions.
2- for each inner node *n* in the tree, compute the utility *u(n)* of *n* as follows:
  - If it's Max's turn: Set *u(n)* to the maximum of the utilities of *n*'s successor nodes.
  - If it's Min's turn: Set *u(n)* to the minimum of the utilities of *n*'s successor nodes

3- Selecting a move for Max at the root: Choose one move that leads to a successor node with maximal utility.

Another Example




```
function Minimax-Decision(s) returns an action
  v ← Max-Value(s)
  return an action yielding value v in the previous function call

function Max-Value(s) returns a utility value
  if Terminal-Test(s) then return u(s)
  v ← −∞
  for each a ∈ Actions(s) do
    v ← max(v,Min-Value(ChildState(s, a)))
  return v

function Min-Value(s) returns a utility value
  if Terminal-Test(s) then return u(s)
  v ← +∞
  for each a ∈ Actions(s) do
    v ← min(v,Max-Value(ChildState(s, a)))
  return v
```

Alpha-Beta ($\alpha - \beta$) Pruning.

To improve the minimax algorithm, we *prune* unnecessary parts of the tree which have no influence on the solution.

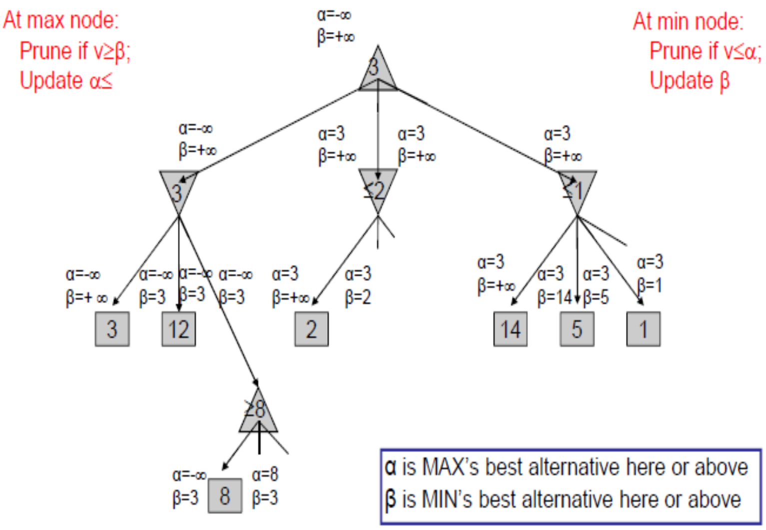
Where,

α = The best choice we have found at any choice point for Max; initially set to $-\infty$
β = the best choice we have found at any choice point for Min ; initially set to $+\infty$

Example: let the successors of A2 in the above figure have x and y values, then the root value is:

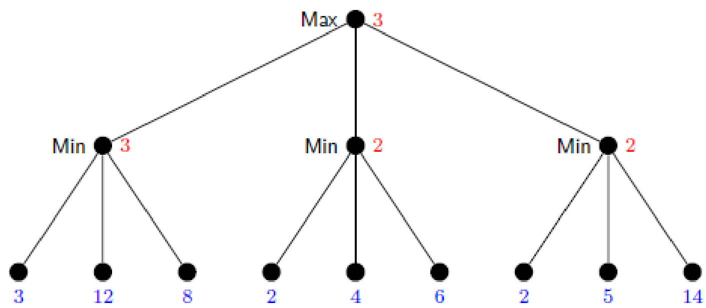
$$\begin{aligned} \text{MINIMAX}(root) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\ &= \max(3, \min(2,x,y), 2) \\ &= \max(3, z, 2) \qquad \text{where } z = \min(2,x,y) \leq 2 \\ &= 3. \end{aligned}$$

```
function Alpha-Beta-Search(s) returns an action
  v ← Max-Value(s, −∞, +∞)
  return an action yielding value v in the previous function call
function Max-Value(s, α, β) returns a utility value
  if Terminal-Test(s) then return u(s)
  v ← −∞
  for each a ∈ Actions(s) do
    v ← max(v, Min-Value(ChildState(s, a), α, β))
    α ← max(α, v)
    if v ≥ β then return v /* Here: v ≥ β ⇔ α ≥ β */
  return v
function Min-Value(s, α, β) returns a utility value
  if Terminal-Test(s) then return u(s)
  v ← +∞
  for each a ∈ Actions(s) do
    v ← min(v, Max-Value(ChildState(s, a), α, β))
    β ← min(β, v)
    if v ≤ α then return v /* Here: v ≤ α ⇔ α ≥ β */
  return v
```



- Prune if $\alpha \geq \beta$

Q\ How many nodes does alpha-beta prune out here?



Evaluation functions.

Alpha-Beta pruning still unpractical. Thus we need to cut off the search early (if time or memory is limited) and use the value of ***evaluation functions*** *(Eval)* instead of the actual utility values *u* of the terminal states.

Evaluation functions cuts off search at a certain depth and compute the value of an **evaluation function** for a state instead of its minimax value.

- A common evaluation function is a weighted sum of *features*:

$$Eval(s) = w_1\ f_1(s) + w_2\ f_2(s) + \ldots + w_n\ f_n(s)$$

  Where w_i is the weight, and f_i is the feature.

  - For chess, w_k may be the **value** of a piece (pawn = 1, knight = 3, rook = 5, queen = 9) and $f_k(s)$ may be the numbers of each kind of pieces.