# Design of AI for games

## Students Survival Pack

IUT de Sénart Fontainebleau
Département Informatique

# Different types of games

## Perfect Information

### Deterministic
checkers, othello, chess, go, diplomacy

### With Chance
backgammon, monopoly, risk

## Imperfect Information

### Deterministic
battleships, stratego, diplomacy

### With Chance
bridge, poker, scrabble, risk

# Different types of games

## Perfect Information

### Deterministic
checkers, othello, chess, go, diplomacy

### With Chance
backgammon, monopoly, risk

## Imperfect Information

### Deterministic
battleships, stratego, diplomacy

### With Chance
bridge, poker, scrabble, risk

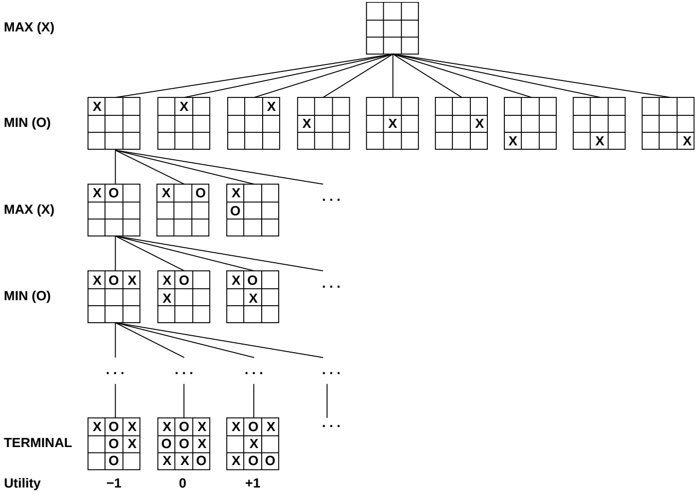
# Game tree

- Nodes: game configuration
- Successors: configuration reachable in one ply
- Leafs: are endgame position labelled by a payoff function, *e.g.* Loss$=-1$, Draw$=0$ and Win$=+1$.

**Remark**

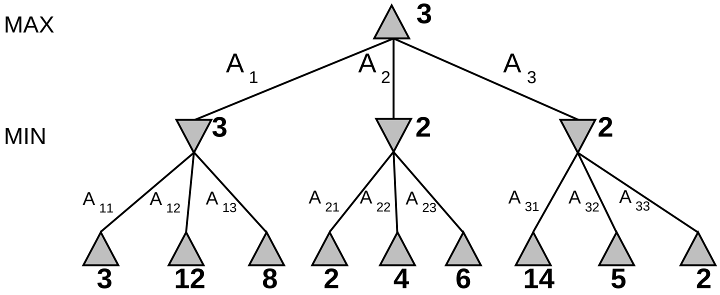
The game tree of a minimax game is finite.

# Example: Tic-Tac-Toe



# Minimax

- Perfect play for deterministic, perfect-information games.
- strategy: choose move to position with highest minimax value
- It is the best achievable payoff against best play.



# The algorithm

**Minimax**(`currentState`)

**if** `currentState` is a terminal state **then**
  **return** *payoff*(`currentState`)
**else if** I am to move **then**
  **return** $\max_{\texttt{nextState}}\{\textbf{Minimax}(\texttt{nextState})\}$
**else**
  **return** $\min_{\texttt{nextState}}\{\textbf{Minimax}(\texttt{nextState})\}$
**end if**

# With one function for each player

**ExploreMax**(`currentState`)

**if** `currentState` is a terminal state **then**
  **return** *payoff*(`currentState`)
**else**
  **return** $\max_{\texttt{nextState}} \{\textbf{ExploreMin}(\texttt{nextState})\}$
**end if**

# With one function for each player

**ExploreMin**(currentState)

**if** currentState is a terminal state **then**
  **return** *payoff*(currentState)
**else**
  **return** $\min_{\texttt{nextState}} \{\textbf{ExploreMax}(\texttt{nextState})\}$
**end if**

# With one function for each player

First call

**ExploreMax**(`currentState`)

**ExploreMax**

**ExploreMin**

Remark

- the two functions are calling each other recursively until a leaf of the game tree is reached
- the values are propagated upward

# Example

- Try some applets available on the web.
- For example with `minimax` on a complete tree with nodes of degree 3 and height 2 (beware the variables are called differently in the applet and in the rest of this lecture: their $b$ denotes de degree and their $d$ the depth).
- You can also enter evaluation for the nodes, try for example

$$1, 0, 1, 1, -1, 0, 0, -1, 0.$$

- Note that, there is no pruning mechanism implemented (e.g. when reaching Win).
- If you use the algorithm `Minimax alpha beta`, some pruning occurs.

# With one function for each player

First call

**ExploreMax**(`currentState`)

**ExploreMax**

**ExploreMin**

Remark

- the two functions are calling each other recursively until a leaf of the game tree is reached
- the values are propagated upward

# Further remarks

## Warning

- the tree is never fully stored on the computer
- some parts of it are stored in RAM
- Essentially by the call stack of the two functions.

## Implementation issues

- the max and min in these functions are implemented by loops of recursive calls
- to avoid pointless computing, if we find a maximal value in a max loop (or a minimal value in a min loop) we can stop the loop early.
- in an object oriented framework, the passing of state as arguments will probably not be needed

# Cutting off search

## Implementation Issues

- With two functions **ExploreMax** and **ExploreMin**, note that both will need a cut-off test
- If a cut-off is added, we also need to implement an evaluation function f.
- In a two player game, it is recommended to keep this function $f$ symmetric.
- In this case, the simplest evaluation function is $+\infty$ for a win, $-\infty$ for a loss and 0 otherwise (essentially every game that has not been simulated until the end is treated as a draw by the AI in its analysis).