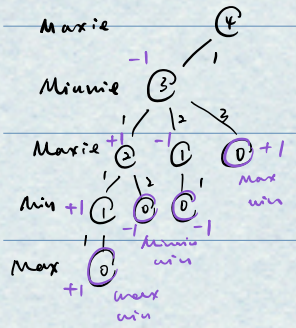
Game

Nim (take 1/2/3 in turn - empty)



Maxie win, assign +1
Minnie win, assign -1

perfect estimator
player can win 100% iff
n mod 4 ≠ 1

Estimator

In practice, tree too large! Instead, ① expand tree to some depth
② use estimator to assign values

structure	signature	functor	signature	functor
Nim	GAME	Minimax	PLAYER	Referee
Connect 4		AlphaBeta		VerboseRef
Checkers		Human		
Chess				

```
signature GAME =
sig
    datatype player = Minnie | Maxie
    datatype outcome = Winner of player | Draw        (* concrete *)
    datatype status = Over of outcome | In_play
    type state
    type move                                          (* abstract *)
    val start : state
    val move : state -> move Seq.seq
    val make_move : state * move -> state
    val status : state -> status
    val player : state -> player
    datatype est = Definitely of outcome | Guess of int   (* concrete *)
    val estimate : state -> est
end
```

REQ: status In_play

```
structure Nim : GAME =
struct
    datatype player = Minnie | Maxie
    datatype outcome = Winner of player | Draw
    datatype status = Over of outcome | In_play
```

```
datatype state = State of int * player
datatype move = Move of int
val start = State (15, Maxie)
fun moves (State (n,_)) = Seq.tabulate (fn k => Move (k+1)) (Int.min (n,3))
fun flip Maxie = Minnie
  | flip Minnie = Maxie
fun make_move (State (n,p), Move k) = State (n-k, flip p)
datatype est = Definitely of outcome | Guess of int
fun estimate (State (n, p)) =
    if n mod 4 = 1 then Definitely (Winner (flip p))
    else Definitely (Winner p)
or just estimate _ = Guess 0   No need to be useful!

fun player (State (_, p)) = p
fun status (State (0, p)) = Over p
  | _                     = In_play
```

Player

```
signature PLAYER =
sig
  structure Game : GAME
  val next_move : Game.state -> Game.move
end

functor HumanPlayer (G: GAME) : PLAYER =
struct
    structure Game = G
    fun next_move s =
        let
            fun
        in
          (case parse (s, read()) of
              SOME m => m
            | NONE   => next_move s)
        end
end

functor MiniMax (Setting : SETTINGS) : PLAYER =
struct
    structure Game = Setting.Game
    structure G = Game
    type edge = G.move * G.est
    fun env (m,v) = m
    fun evl (m,v) = v

    fun leq (x,y) = ...   G.est * G.est -> bool
```

fun max (e1, e2) = if leq (evl e2, evl e1) then e1 else e2

fun choose G. Maxie = Seq. reduce max

end