```
(************************************************************************)
(*                              Min Max                                 *)
(************************************************************************)

(* minimize : ('a -> 'a -> bool) -> ('b -> 'a) -> 'b list -> 'b * 'a
 *
 * prend une fonction de comparaison, une fonction d'evaluation, une liste,
 * et rend un element de la liste minimizant la fonction d'evaluation,
 * ainsi que la valeur de la fonction d'evaluation en ce point. *)

let minimize cmp eval l =
  let rec trouve ((m,v) as c) = function
      []    -> c
    | x'::l -> let v' = eval x' in
               if cmp v' v then
                 trouve (x',v') l
               else
                 trouve c l
  in match l with
      []   -> raise (Invalid_argument "minimize: empty list")
    | x::l -> trouve (x,eval x) l
;;

(* min et max dans le cas particulier d'evaluation dans float *)

let min (eval:'a->float) l = minimize (prefix <) eval l;;
let max (eval:'a->float) l = minimize (prefix >) eval l;;


(* minmax
     eval : 'a -> float
     suiv : 'a -> 'a list
     dep  : 'a
   renvoie le coup à jouer et son evaluation : 'a (pos. choisie) * float
   profondeur 1. *)

let minmax eval suiv dep =
  let eval1 p = match eval p with
      0.0 -> 1.0
    | 1.0 -> 0.0
    | _   -> snd (min eval (suiv p))
  in
    max eval1 (suiv dep)
;;


(* minmax_n : min-max profondeur n *)

let minmax_n eval suiv n dep =

  let rec eval1 n p = match eval p with
      0.0 -> 1.0
    | 1.0 -> 0.0
    | _   ->
        if n = 0 then
          snd (min eval (suiv p))
        else
          snd (min (fun p -> snd (minmax_rec (pred n) p)) (suiv p))

  and minmax_rec n p =
    match eval p with
        (0.0 | 1.0) as e -> p,e
      | _                -> max (eval1 n) (suiv p)
```



```
  in minmax_rec n dep
;;

(* remarque : minmax = (minmax_n 0) *)

(************************************************************************)
(*                          Jeu des allumettes                          *)
(************************************************************************)

type configuration == bool * int ;;

let (configuration_initiale : configuration) = true, 20 ;;

let (suiv : configuration -> configuration list) = fun (b,n) ->
  map (fun n' -> not b,n') (match n with
    0 -> []
  | 1 -> [0]
  | 2 -> [0 ; 1]
  | n -> [n-3 ; n-2 ; n-1])
;;

let (eval : configuration -> float) = fun (b,n) -> match n with
    0 -> 1.0
  | _ -> 0.5
;;

let joue ((b,n) as c) = fst (minmax_n eval suiv (n/2+1) c) ;;

let affiche_configuration (b,n) =
  print_string (if b then "au joueur 1 : " else "au joueur 2 : ") ;
  for i = 1 to n do
    print_string "|"
  done ;
  print_newline ()
;;

let boucle coup ci =
  let rec boucle_rec ((b,n) as c) =
    if n = 0 then begin
      print_string (if b then "le joueur 1 gagne !"
                         else "le joueur 2 gagne !") ;
      print_newline ()
    end else begin
      affiche_configuration c ;
      boucle_rec (coup c)
    end
  in
    boucle_rec ci
;;
```