```
(*fonctions manipulation de liste *)
type 'a t_liste = 'a list;;

let l_cv() : 'a t_liste= [];;

let l_ape(x,l): 'a t_liste = x::l;;

let rec l_ade(x,l): 'a t_liste = match l with
[] -> [x]
| y::l1 -> y::l_ade(x,l1);;

let l_ev(l: 'a t_liste) = match l with
[] -> true
| _ -> false;;

let l_pe(l : 'a t_liste) = match l
with [] -> failwith "erreur fonction l_pe, pas de premier element dans
une liste vide"
| x::_ -> x;;

let l_spe(l : 'a t_liste): 'a t_liste = match l with
[] -> failwith "erreur fonction l_spe, pas de premier element a supprimer
dans une liste vide"
| x::l1 -> l1;;

let  rec l_de(l : 'a t_liste) = match l with
[] -> failwith "erreur fonction l_de, pas de dernier element dans une
liste vide"
| [x] -> x
| _::l1 -> l_de(l1);;

let  rec l_sde(l : 'a t_liste): 'a t_liste = match l with
[] -> failwith "erreur fonction l_de, pas de dernier element dans une
liste vide"
| [x] -> []
| x::l1 -> x::l_sde(l1);;



(* Déclaration des types *)
type t_configuration = bool*int ;;



(*fonctions de manipulation *)

let joueur(c) = match c with
    (j,_) -> j;;

let nombreAllumettes(c) = match c with
    (_,n) -> n;;

let retirerAllumettes(c,nb) = match c with
    (j,n) -> (j,n-nb);;

let estJoueur1 ( c ) = joueur(c);;   (* si c'est le joueur 1 on aura true
car on a associé true à joueur1 sinon on aura false *)
```

```
let tableVide(c) = match nombreAllumettes(c) with
    0-> true
  |_ -> false ;;

let tourJoueur (c) = match (c) with
    true -> print_string" joueur 1"
  |false -> print_string" joueur 2";;


(*fonctions demandées*)
let rec  affiche_conf_aux(n) =
  print_string"|";
  let h= n-1 in
  if
    h!=0
  then
    affiche_conf_aux(h);;


let affiche_configuration(c) =
  print_string"     Au";
  tourJoueur(joueur(c));
  print_string" :";
  let x = nombreAllumettes(c) in
  affiche_conf_aux(x)
;;

affiche_configuration(false,5);;

(* fonction de manupulation du type t_arbre_configuration *)
type t_arbre_configuration = A of
t_configuration*float*t_arbre_configuration list;;

let a_cree(c,f,l) = A(c,f,l);;

let a_config(a) = match a with
    A(c,_,_) -> c;;

let a_qualite(a) = match a with
    A(_,f,_)-> f ;;

let a_changeQual(a, q) = match a with
    A(c,f,l) -> A(c,q,l);;

let a_changeList(a, newl) = match a with
    A(c,f,l) -> A(c,f,newl);;

let a_list(a) = match a with
    A(_,_,l) -> l ;;

let rec a_construction(c) =
  a_cree(c, -1.0, rempli_listes(c))
and
    rempli_listes(c) =
  let n = nombreAllumettes(c) in
  let j = joueur(c) in
  let l1 = rempli_sous_arbres(j, n-1, l_cv()) in
  let l2 = rempli_sous_arbres(j, n-2, l1) in
```

```
  rempli_sous_arbres(j, n-3, l2)
and
    rempli_sous_arbres(j, n, l) =
  if
    n>0
  then
    l_ade(a_construction(not(j), n), l)
  else
    l;;

a_construction(false,4);;

let rec l_max_aux(l, v) =
  if l_ev(l)
  then
    v
  else
    let new_v = a_qualite(l_pe(l)) in
    if
      new_v>v
    then
      l_max_aux(l_spe(l),new_v)
    else
      l_max_aux(l_spe(l),v);;

let l_max(l) =
  l_max_aux(l, 0.0);;

let rec l_min_aux(l,v) =
  if l_ev(l)
  then
    v
  else
    let new_v = a_qualite(l_pe(l)) in
    if
      new_v<v
    then
      l_min_aux(l_spe(l), new_v)
    else
      l_min_aux(l_spe(l), v);;

let l_min(l) =
  l_min_aux(l, 1.0);;

let rec a_eval(arbre) =
  if
    (l_ev(a_list(arbre)))
  then
    if
      (joueur(a_config(arbre)))
    then
      a_changeQual(arbre, 0.0)
    else
      a_changeQual(arbre, 1.0)
  else
    let sousArbres = a_evalSousArbres(a_list(arbre),l_cv()) in
    if
      (joueur(a_config(arbre)))
```

```
    then
      a_changeQual(a_changeList(arbre, sousArbres), l_max(sousArbres))
    else
      a_changeQual(a_changeList(arbre, sousArbres), l_min(sousArbres))
and
    a_evalSousArbres(l, newl) =
  if
    l_ev(l)
  then
    newl
  else
    let elem = l_pe(l) in
    a_evalSousArbres(l_spe(l), l_ade(a_eval(elem), newl));;

a_eval(a_construction(true,4));;

let rec l_newMax_aux(l, v, c) =
  if l_ev(l)
  then
    c
  else
    let new_v = a_qualite(l_pe(l)) in
    let new_c = a_config(l_pe(l)) in
    if
      new_v>v
    then
      l_newMax_aux(l_spe(l),new_v, new_c)
    else
      l_newMax_aux(l_spe(l),v,c);;

let l_newMax(l) =
  l_newMax_aux(l, 0.0, a_config(l_pe(l)));;

let rec l_newMin_aux(l,v,c) =
  if l_ev(l)
  then
    c
  else
    let new_v = a_qualite(l_pe(l)) in
    let new_c = a_config(l_pe(l)) in
    if
      new_v<v
    then
      l_newMin_aux(l_spe(l), new_v, new_c)
    else
      l_newMin_aux(l_spe(l), v, c);;

let l_newMin(l) =
  l_newMin_aux(l, 1.0, a_config(l_pe(l)));;

let a_trouveCoup(c) =
  let arbre = a_eval(a_construction(c)) in
  let joueur = joueur(c) in
  let nb = nombreAllumettes(c) in
  if
    (joueur)
  then
    let bestCoup = l_newMax(a_list(arbre)) in
```

```
    nb-nombreAllumettes(bestCoup)
  else
    let bestCoup = l_newMin(a_list(arbre)) in
    nb-nombreAllumettes(bestCoup);;

a_trouveCoup(true,4);;

let rec computerVsComputer(c:t_configuration) =
  begin
    affiche_configuration(c);
    if
      nombreAllumettes(c)=1
    then
      begin
      print_string "   Partie terminée-";
      if
        joueur(c)
      then
        print_string "Le joueur 2 a gagné"
      else
        print_string "Le joueur 1 a gagné"
      end
    else
      computerVsComputer(not(joueur(c)),nombreAllumettes(c)-
a_trouveCoup(c))
  end;;

computerVsComputer(true, 15);;
```