```
(************************************************************************)
(*                  Jeu de la reine contre les 8 pions                  *)
(************************************************************************)

type 'a option = None | Some of 'a ;;

type joueur = Reine | Pions ;;

type position = { colonne : int ; (* 0..7 *)
                  ligne   : int   (* 0..7 *)
} ;;

type configuration = {
  aqui : joueur ;         (* a qui de jouer *)
  reine : position ;      (* ou est la reine *)
  mangee : bool ;         (* la reine a ete mangee *)
  nb_pions : int ;        (* combien y a-t-il de pions *)
  pions : position vect   (* ou sont-ils 0..n-1 *)
} ;;


(************************************************************************)
(*                          Interface graphique                         *)
(************************************************************************)

#open "graphics";;

let W, H =
  open_graph "" ;
  let x,y = size_x(), size_y() in
  close_graph () ;
  x,y
;;

let pas = (min W H) / 10 ;;

let off_x = (W - 8*pas) / 2 and off_y = (H - 8*pas) / 2 ;;

let affiche_case_vide i j =
  let x = off_x + pas*i and y = off_y +pas*j in
  set_color blue ;
  moveto x y ;
  lineto (x+pas) y ;
  lineto (x+pas) (y+pas) ;
  lineto x (y+pas) ;
  lineto x y
;;

let affiche_case i j ty =
  affiche_case_vide i j ;
  match ty with
      None -> ()
    | Some Reine ->
        moveto (off_x+i*pas+15) (off_y+j*pas+15) ;
        draw_string "R"
    | Some Pions ->
        draw_circle (off_x+i*pas+pas/2) (off_y+j*pas+pas/2) (4*pas/10)
;;

let affiche_configuration
  { aqui=j ; reine=posr ; mangee=m ; nb_pions=n ; pions=posp } =

  clear_graph () ;
```



```
  for i = 0 to 7 do
    for j = 0 to 7 do
      affiche_case_vide i j
    done
  done ;
  if not m then affiche_case posr.colonne posr.ligne (Some Reine) ;
  for i = 0 to n-1 do
    affiche_case posp.(i).colonne posp.(i).ligne (Some Pions)
  done ;
  moveto 10  10 ;
  draw_string (match j with Reine -> "à la reine de jouer"
                          | Pions -> "aux pions de jouer")
;;


(************************************************************************)
(*                              Deplacements                            *)
(************************************************************************)

let configuration_initiale =
  let v = make_vect 8 { colonne = 0 ; ligne = 0 } in
  for i = 0 to 7 do
    v.(i) <- { colonne = i ; ligne = 1 }
  done ;
  { aqui = Pions ;
    reine = { colonne = 4 ; ligne = 7 } ;
    mangee = false ;
    nb_pions = 8 ;
    pions = v }
;;

let list_0_n n =
  let rec aux i =
    if i > n then [] else i :: (aux (succ i))
  in aux 0
;;

let rec list_n_7 n =
  if n > 7 then [] else n :: (list_n_7 (succ n))
;;

let deplac_reine plateau pos =
  let pl = ref [] in
  let i = pos.colonne and j = pos.ligne in
  let x = ref 0 and y = ref 0 in

  (* a droite *)
  x := succ i; y := j ;
  while (!x<8 & plateau.(pred !x).(j)=false) do
    pl := (!x,j) :: !pl ; incr x done ;

  (* a gauche *)
  x := pred i; y := j ;
  while (!x>=0 & plateau.(succ !x).(j)=false) do
    pl := (!x,j) :: !pl ; decr x done ;

  (* en haut *)
  x := i; y := succ j ;
  while (!y<8 & plateau.(i).(pred !y)=false) do
    pl := (i,!y) :: !pl ; incr y done ;

  (* en bas *)
  x := i; y := pred j ;
```

```
  while (!y>=0 & plateau.(i).(succ !y)=false) do
    pl := (i,!y) :: !pl ; decr y done ;

  (* en haut a droite *)
  x := succ i ; y := succ j ;
  while (!x<8 & !y<8 & plateau.(pred !x).(pred !y)=false) do
    pl := (!x,!y) :: !pl ; incr x ; incr y done ;

  (* en bas a droite *)
  x := succ i ; y := pred j ;
  while (!x<8 & !y>=0 & plateau.(pred !x).(succ !y)=false) do
    pl := (!x,!y) :: !pl ; incr x ; decr y done ;

  (* en haut a gauche *)
  x := pred i ; y := succ j ;
  while (!x>=0 & !y<8 & plateau.(succ !x).(pred !y)=false) do
    pl := (!x,!y) :: !pl ; decr x ; incr y done ;

  (* en bas a gauche *)
  x := pred i ; y := pred j ;
  while (!x>=0 & !y>=0 & plateau.(succ !x).(succ !y)=false) do
    pl := (!x,!y) :: !pl ; decr x ; decr y done ;

  !pl
;;

let deplac_pions cf =
  let pl = ref [] in
  for i=0 to cf.nb_pions-1 do

    if cf.pions.(i).ligne < 7 &
       not (cf.pions.(i).colonne=cf.reine.colonne
          & cf.pions.(i).ligne = pred cf.reine.ligne) then begin
      let v = copy_vect cf.pions in
      v.(i) <- { colonne = cf.pions.(i).colonne ;
                 ligne   = succ cf.pions.(i).ligne } ;
      let p = { aqui = Reine ;
                reine = cf.reine ;
                mangee = false ;
                nb_pions = cf.nb_pions ;
                pions = v } in
      pl := p :: !pl
    end ;

    (* pion i mange en haut a droite *)
    if cf.pions.(i).colonne = pred cf.reine.colonne
     & cf.pions.(i).ligne = pred cf.reine.ligne then begin
      let v = copy_vect cf.pions in
      v.(i) <- { colonne = succ cf.pions.(i).colonne ;
                 ligne   = succ cf.pions.(i).ligne } ;
      let p = { aqui = Reine ;
                reine = cf.reine ;
                mangee = true ;
                nb_pions = cf.nb_pions ;
                pions = v } in
      pl := p :: !pl
    end ;

    (* pion i mange en haut a gauche *)
    if cf.pions.(i).colonne = succ cf.reine.colonne
     & cf.pions.(i).ligne = pred cf.reine.ligne then begin
      let v = copy_vect cf.pions in
      v.(i) <- { colonne = pred cf.pions.(i).colonne ;
```



```
                 ligne   = succ cf.pions.(i).ligne } ;
      let p = { aqui = Reine ;
                reine = cf.reine ;
                mangee = true ;
                nb_pions = cf.nb_pions ;
                pions = v } in
      pl := p :: !pl
    end ;

  done ;
  !pl
;;

let deplac cf =
  let plateau = make_matrix 8 8 false in
  for i = 0 to cf.nb_pions-1 do
    plateau.(cf.pions.(i).colonne).(cf.pions.(i).ligne) <- true
  done ;
  let i' = ref 0 in
  match cf.aqui with
      Reine ->
        let posl = deplac_reine plateau cf.reine in
        map (fun (x,y) ->
               let n = if plateau.(x).(y) then pred cf.nb_pions
                                          else cf.nb_pions in
               let v = make_vect n { colonne=0; ligne=0 } in
               i' := -1 ;
               for i = 0 to cf.nb_pions-1 do
                 if not (cf.pions.(i).colonne=x
                         & cf.pions.(i).ligne=y) then begin
                   incr i' ;
                   v.(!i') <- cf.pions.(i)
                 end
               done ;
               { aqui = Pions ;
                 reine = { colonne = x ; ligne = y } ;
                 mangee = false ;
                 nb_pions = n ;
                 pions = v } ) posl

    | Pions ->
        if cf.nb_pions = 0 then
          []
        else
          deplac_pions cf
;;


(*************************************************************************)
(*                      Jeu, par la strategie min/max                    *)
(*************************************************************************)

let vect_exists =
  let r = ref false in
  fun p v ->
  r := false ;
  for i = 0 to (vect_length v)-1 do
    if (p v.(i)) then
      r := true
  done ;
  !r
;;
```

```
let promotion cf =
  vect_exists (fun p -> p.ligne = 7) cf.pions
;;

(* La fonction d'evaluation.
   Il n'y a qu'une seule fonction d'evaluation, puisqu'une configuration
   contient l'information "c'est a X de jouer".

   La fonction d'evaluation rend un flottant entre 0 et 1.
   1 signifie la victoire et 0 la defaite. *)

let eval cf =
  match cf.aqui with
      Reine ->
        if cf.nb_pions = 0 then
          1.0
        else if cf.mangee or (promotion cf) then
          0.0
        else begin
          (** fonction du max des hauteurs des pions et du nb de pions *)
          let s = ref 1 in
          for i = 0 to cf.nb_pions-1 do
            if cf.pions.(i).ligne > !s then s := cf.pions.(i).ligne
          done ;
          1.0 -. 0.1 *. (float_of_int !s)
              -. 0.01 *. (float_of_int cf.nb_pions)
        end

    | Pions ->
        if cf.nb_pions = 0 then
          0.0
        else if cf.mangee or (promotion cf) then
          1.0
        else
          (* uniquement fonction de leur nombre *)
          0.1 *. (float_of_int cf.nb_pions)
;;


include "minmax";;

let joue c = fst (minmax_n eval deplac 1 c) ;;

let rec boucle_auto c =
  affiche_configuration c ;
  if c.nb_pions = 0 or c.mangee or (promotion c) then begin
    moveto 320 10 ; draw_string "fin de partie" ;
    c
  end else
    let touche = read_key () in
      if touche = 'q' then
        c
      else
        boucle_auto (joue c)
;;

let pause n = for i = 0 to n do done ;;

let licite c (i,j) (i',j') =
  let est_pion =
    vect_exists (fun p -> p.colonne = i & p.ligne = j) c.pions in
  let mange =
    c.reine.colonne=i' & c.reine.ligne=j' in
```



```
  let croque =
       (i'=(succ i) & j'=(succ j) & mange)
    or (i'=(pred i) & j'=(succ j) & mange) in
  let avance =
    i'=i & j'=(succ j) & (c.reine.colonne<>i' or c.reine.ligne<>j') in

  if est_pion & (avance or (croque & mange)) then
    let v' = copy_vect c.pions in
    for k = 0 to c.nb_pions-1 do
      if v'.(k).colonne=i & v'.(k).ligne=j then
        v'.(k) <- { colonne=i' ; ligne=j' }
    done ;
    let c' = { aqui = Reine ;
               reine = c.reine ;
               mangee = mange ;
               nb_pions = c.nb_pions ;
               pions = v' } in
    true, c'
  else
    false, c
;;

let rec deplac_souris c =
  moveto 500 10 ; draw_string "de" ;
  let cl = deplac c in
  while not (button_down ()) do done ;
  let x,y = mouse_pos () in
  let i,j = (x-off_x)/pas, (y-off_y)/pas in
  pause 100000 ; moveto 520 10 ; draw_string "a" ;
  while not (button_down ()) do done ;
  let x,y = mouse_pos () in
  let i',j' = (x-off_x)/pas, (y-off_y)/pas in
  let ok, c' = licite c (i,j) (i',j') in
  if ok then
    c'
  else begin
    clear_graph () ;
    affiche_configuration c ;
    deplac_souris c
  end
;;

let rec boucle_joueur c =
  affiche_configuration c ;
  if c.nb_pions = 0 or c.mangee or (promotion c) then begin
    moveto 320 10 ; draw_string "fin de partie" ;
    c
  end else
  match c.aqui with
      Reine -> boucle_joueur (joue c)
    | Pions -> boucle_joueur (deplac_souris c)
;;

let test boucle =
  open_graph "" ;
  let c = boucle configuration_initiale in
  read_key () ;
  close_graph () ;
  c
;;
```