(option informatique-1ère année)

# Corrigé Colle d'info n°19

```
#open "graphics";;

type case = Libre | Mur | Caisse | Garage | Caisse_garée | Héros | Héros_garé;;

type Mouv = Nord | Sud | Ouest | Est ;;

let nl = 11 and nc = 19;;

let t = make_matrix nl nc Libre;;
```

(* Question 1 *)

```
let laby_valide t =
  let nb_héros = ref(0) and nb_caisses = ref(0) and nb_garages = ref(0) in
     for i = 0 to nl-1 do
       for j = 0 to nc-1 do
           match t.(i).(j) with
             Héros -> incr nb_héros
           | Garage -> incr nb_garages
           | Caisse -> incr nb_caisses
           | Héros_garé -> incr nb_héros; incr nb_garages
           |   _    -> ()
       done;
      done;
(!nb_héros = 1) && (!nb_garages = !nb_caisses);;
```

(* Question 2 *)

```
exception trouvé of int * int;;

let place_héros t =
try
     for i = 0 to nl-1 do
       for j = 0 to nc-1 do
           match t.(i).(j) with
             Héros | Héros_garé -> raise (trouvé (i,j))
           |          _                  -> ()
       done;
      done;
(-1,-1)    (* Ce cas est exclu si le labirynthe est valide *)
with trouvé (i,j) -> (i,j);;
```

(* Question 3 *)

```
let print_laby t =
   for i = 0 to nl-1 do
     for j = 0 to nc-1 do
         match t.(i).(j) with
             Héros -> print_char 'H'
           | Héros_garé -> print_char 'h'
           | Caisse -> print_char 'C'
           | Garage -> print_char 'G'
           | Caisse_garée -> print_char 'c'
```

```
          | Mur -> print_char 'M'
          |   _    -> print_char ' '
       done;
      print_newline();
      done;;
```

(* Question 4 *) (* On s'arrête si l'une des caisses n'est pas garée ou si l'un des garages est vide *)

```
exception non_fini;;

let laby_fini t =
try
     for i = 0 to nl-1 do
       for j = 0 to nc-1 do
           match t.(i).(j) with
             Caisse | Garage -> raise non_fini
           |   _    -> ()
       done;
      done;
true
with non_fini -> false;;
```

(* Question 5 *)

```
let mouv_licite mouv =
  let (i,j) = place_héros t in
  match mouv with
    Ouest -> begin
             match t.(i).(j-1) with
               Libre | Garage -> true
             | Mur  -> false
             | Caisse | Caisse_garée -> (t.(i).(j-2) = Libre) || (t.(i).(j-2) = Garage)
             | _  -> failwith "mouv_licite : erreur dans le labyrinthe"
             end
   |  Est  -> begin
             match t.(i).(j+1) with
               Libre | Garage -> true
             | Mur  -> false
             | Caisse | Caisse_garée -> (t.(i).(j+2) = Libre) || (t.(i).(j+2) = Garage)
             | _  -> failwith "mouv_licite : erreur dans le labyrinthe"
             end
   | Sud -> begin
             match t.(i+1).(j) with
               Libre | Garage -> true
             | Mur  -> false
             | Caisse | Caisse_garée -> (t.(i+2).(j) = Libre) || (t.(i+2).(j) = Garage)
             | _  -> failwith "mouv_licite : erreur dans le labyrinthe"
             end
   | Nord   -> begin
             match t.(i-1).(j) with
               Libre | Garage -> true
             | Mur  -> false
             | Caisse | Caisse_garée -> (t.(i-2).(j) = Libre) || (t.(i-2).(j) = Garage)
             | _  -> failwith "mouv_licite : erreur dans le labyrinthe"
             end
;;
```

(* Question 6 *)

```
let liste_mouv_licites () =
let l = [Sud; Est; Nord; Ouest] in
let rec test = function
   x::xs when (mouv_licite x) -> x::(test xs)
  | x::xs -> test xs
  | [] -> []
in test l;;
```

(* la liste n'est jamais vide car le héros peut toujours revenir en arrière sauf dans le cas où la position de départ ne lui permet pas de bouger *)
(* Question 7 *)

```
let déplace mouv =
let (i,j) = place_héros t in
begin
match t.(i).(j) with (* On libère la case occupée par le héros *)
      Héros   -> t.(i).(j) <- Libre
 | Héros_garé -> t.(i).(j) <- Garage;
end;
match mouv with
  Nord ->
         begin
          match t.(i-1).(j) with
             Libre -> t.(i-1).(j) <- Héros
          | Garage -> t.(i-1).(j) <- Héros_garé
          | Caisse -> begin
                           match t.(i-2).(j) with
                               Libre -> t.(i-2).(j) <- Caisse
                            | Garage -> t.(i-2).(j) <- Caisse_garée
                            |   _       -> failwith "déplace : mouvement illicite";
                    end;
                           t.(i-1).(j) <- Héros;
          | Caisse_garée
                      -> begin
                           match t.(i-2).(j) with
                               Libre -> t.(i-2).(j) <- Caisse
                            | Garage -> t.(i-2).(j) <- Caisse_garée
                            |   _       -> failwith "déplace : mouvement illicite";
                            end;
                           t.(i-1).(j) <- Héros_garé
         |     _     -> failwith "déplace : mouvement illégal"
          end
|  Sud ->
         begin
          match t.(i+1).(j) with
             Libre -> t.(i+1).(j) <- Héros
          | Garage -> t.(i+1).(j) <- Héros_garé
          | Caisse -> begin
                           match t.(i+2).(j) with
                               Libre -> t.(i+2).(j) <- Caisse
                            | Garage -> t.(i+2).(j) <- Caisse_garée
                            |   _       -> failwith "déplace : mouvement illicite";
                           end;
                           t.(i+1).(j) <- Héros;
          | Caisse_garée
```

```
                    -> begin
                         match t.(i+2).(j) with
                             Libre -> t.(i+2).(j) <- Caisse
                           | Garage -> t.(i+2).(j) <- Caisse_garée
                           |   _       -> failwith "déplace : mouvement illicite";
                          end;
                         t.(i+1).(j) <- Héros_garé;
        |     _     -> failwith "déplace : mouvement illégal"
         end
| Est ->
        begin
        match t.(i).(j+1) with
            Libre -> t.(i).(j+1) <- Héros
         | Garage -> t.(i).(j+1) <- Héros_garé
         | Caisse -> begin
                         match t.(i).(j+2) with
                             Libre -> t.(i).(j+2) <- Caisse
                           | Garage -> t.(i).(j+2) <- Caisse_garée
                           |   _       -> failwith "déplace : mouvement illicite";
                         end;
                         t.(i).(j+1) <- Héros;
         | Caisse_garée
                     -> begin
                         match t.(i).(j+2) with
                               Libre   -> t.(i).(j+2) <- Caisse;
                             | Garage  -> t.(i).(j+2) <- Caisse_garée;
                             |     _      -> failwith "déplace : mouvement illicite";
                          end;
                         t.(i).(j+1) <- Héros_garé
        |     _     -> failwith "déplace : mouvement illégal"
         end
| Ouest ->
        begin
         match t.(i).(j-1) with
            Libre -> t.(i).(j-1) <- Héros
         | Garage -> t.(i).(j-1) <- Héros_garé
         | Caisse -> begin
                         match t.(i).(j-2) with
                             Libre -> t.(i).(j-2) <- Caisse
                           | Garage -> t.(i).(j-2) <- Caisse_garée
                           |   _       -> failwith "déplace : mouvement illicite";
                         end;
                         t.(i).(j-1) <- Héros;
         | Caisse_garée
                     -> begin
                         match t.(i).(j-2) with
                                Libre -> t.(i).(j-2) <- Caisse
                             | Garage -> t.(i).(j-2) <- Caisse_garée
                             |    _       -> failwith "déplace : mouvement illicite";
                          end;
                          t.(i).(j-1) <- Héros_garé
        |     _     -> failwith "déplace : mouvement illégal"
         end
;;
```

(* Question 8 : *)

```
type Mouv = Nord | Sud | Est | Ouest | NO | NE | SE | SO;;

let libre (i,j)  =
match t.(i).(j) with
  Libre | Garage -> true
|    _  -> false;;

let mouv_gen_licite mouv =
let (i,j) = place_héros t in
match mouv with
  Nord | Sud | Est | Ouest -> mouv_licite mouv
| NO -> (libre(i-1,j) || libre (i,j-1)) && libre(i-1,j-1)
| NE -> (libre(i-1,j) || libre (i,j+1)) && libre(i-1,j+1)
| SO -> (libre(i+1,j) || libre (i,j-1)) && libre(i+1,j-1)
| SE -> (libre(i+1,j) || libre (i,j+1)) && libre(i+1,j+1);;
```

(* Question 9 :
Il peut y avoir deux solutions, l'une ou les deux nécessitant de pousser des caisses, il peut y avoir une bonne et une mauvaise solution en fonction de la stratégie choisie par le joueur et qui ne peut être devinée par le programme *)
(* Question 10 *)

```
let tacc = make_matrix nl nc (999999);;

let rec accés_laby (i,j) n =
tacc.(i).(j) <- n;
begin (* déplacement vers le nord *)
match t.(i-1).(j) with
  Libre | Garage when tacc.(i-1).(j) > n
       -> accés_laby (i-1,j) (n+1)
  | _  -> ()
end;
begin (* déplacement vers le sud *)
match t.(i+1).(j) with
  Libre | Garage when tacc.(i+1).(j) > n
       -> accés_laby (i+1,j) (n+1)
  | _  -> ()
end;
begin (* déplacement vers l'est *)
match t.(i).(j+1) with
  Libre | Garage when tacc.(i).(j+1) > n
       -> accés_laby (i,j+1) (n+1)
  | _  -> ()
end;
begin (* déplacement vers l'ouest *)
match t.(i).(j-1) with
  Libre | Garage when tacc.(i).(j-1) > n
       -> accés_laby (i,j-1) (n+1)
  | _  -> ()
end;;

accés_laby (place_héros t) 0;;
```

(* Question 13 :
On part de la position d'arrivée qui est marquée de $n$, on considère alors une case adjacente marquée de $(n-1)$, on stocke le déplacement et on recommence ainsi de suite jusqu'à 0 *)

```
let chemin (i,j) =
let rec Chemin (i,j) n l =
if n = 0
   then l
   else begin
        if tacc.(i-1).(j) = n-1
        then Sud::(Chemin (i-1,j) (n-1) l)
        else
           if tacc.(i+1).(j) = n-1
           then Nord::(Chemin (i+1,j) (n-1) l)
           else
               if tacc.(i).(j-1) = n-1
               then Est::(Chemin (i,j-1) (n-1) l)
               else
                   if tacc.(i).(j+1) = n-1
                   then Ouest::(Chemin (i,j+1) (n-1) l)
                   else failwith "erreur";
        end
in
match tacc.(i).(j) with
  999999 -> failwith "chemin : position inaccessible"
 |  n    -> rev(Chemin (i,j) n []);;
```

(* Question 14 *)

```
let lire_laby () =
let fichier = open_in "laby" in
for i = 0 to nl-1 do
    let s = input_line fichier in
    for j = 0 to nc-1 do
         match s.[j] with
           'H' -> t.(i).(j) <- Héros
         | 'h' -> t.(i).(j) <- Héros_garé
         | 'C' -> t.(i).(j) <- Caisse
         | 'G' -> t.(i).(j) <- Garage
         | 'c' -> t.(i).(j) <- Caisse_garée
         | 'M' -> t.(i).(j) <- Mur
         | ' ' -> t.(i).(j) <- Libre
         |  _  -> failwith "lire_laby : caractère illicite"
    done;
done;
close_in fichier;;
```

(* Question 15 *)

```
let joue () =
lire_laby();
while not(laby_fini t) do
    print_laby t;
    match read_key () with
      '8' | 'N' | 'n'  -> if mouv_licite Nord then déplace Nord
    | '2' | 'S' | 's'  -> if mouv_licite Sud then déplace Sud
    | '4' | 'O' | 'o'  -> if mouv_licite Ouest then déplace Ouest
    | '6' | 'E' | 'e'  -> if mouv_licite Est then déplace Est
    | x when int_of_char x = 30  ->  if mouv_licite Nord then déplace Nord
    | x when int_of_char x = 31  ->  if mouv_licite Sud then déplace Sud
    | x when int_of_char x = 28  ->  if mouv_licite Ouest then déplace Ouest
    | x when int_of_char x = 29  ->  if mouv_licite Est then déplace Est
    | _ -> ()
    done;
print_laby t;
print_string "félicitation, vous avez gagné";;
```

(* Question 16 :
Il faut garder en mémoire la liste des déplacements du Héros en conservant en mémoire la direction du déplacement (Nord, Sud, Est ou Ouest) et si le héros a poussé d'une caisse pendant son déplacement. Il reste à construire ensuite une fonction anti_déplace qui permet de revenir un coup en arrière *)