Corrigé du TP sur le jeu de solitaire

1) On crée une pile vide p, et on étudie chaque site (i, j) et chaque direction. Par exemple, on ajoute (i, j), `'d'` à p si et seulement si $i < 5$, (i, j) et $(i + 1, j)$ contiennent une boule et $(i + 2, j)$ n'en contient pas.

```
let coups_possibles (n,m) = let p = ref [] in
     for i = 0 to 6 do for j = 0 to 6 do
        if m.(i).(j) = 1 then
           begin
           if j < 5 then
              if m.(i).(j+1) = 1 && m.(i).(j+2) = 0 then
                 p : =  ((i,j),'h')::(!p);
           if j>1 then
              if m.(i).(j-1) = 1 && m.(i).(j-2) = 0 then
                 p : =  ((i,j),'b')::(!p);
           if i < 5 then
              if m.(i+1).(j) = 1 && m.(i+2).(j) = 0 then
                 p : =  ((i,j),'d')::(!p);
           if i>1 then
              if m.(i-1).(j) = 1 && m.(i-2).(j) = 0 then
                 p : =  ((i,j),'g')::(!p);
           end
     done; done;
     !p;;
```

2) Il faut veiller à créer une nouvelle matrice, pour ne pas modifier la position initiale[1]. On commence donc par recopier la position, et on modifie ensuite les trois sites concernés par le coup étudié.

```
let nouvelle_position pos c = let n,m = pos and (a,b),d = c in
     let m1 = Array.make_matrix 7 7 0 in
        for i = 0 to 6 do for j = 0 to 6 do
           m1.(i).(j) < - m.(i).(j)
        done; done;
        let d1,d2 = match d with
           | 'd' -> 1,0
           | 'g' -> (-1),0
           | 'b' -> 0,(-1)
           | _ -> 0,1 in
        m1.(a).(b)  < - 0;
        m1.(a + d1).(b + d2)  < -0;
        m1.(a + 2*d1).(b + 2*d2)  < -1;
        n-1,m1;;
```

3) Nous aurons besoin de la fonction `ajouter: 'a -> 'a list list -> 'a list list` qui, appliquée à un élément a de type `'a` et à une liste $[l_1; l_2; \ldots; l_k]$ de listes d'objets de type `'a` associe la liste $[a :: l_1; a :: l_2; \ldots; a :: l_k]$ obtenue en ajoutant a en tête de chacune des listes l_i.

```
let rec ajouter a = function
     | [] -> []
     | p::l1 -> (a::p)::(ajouter a l1);;
```

1. On peut aussi travailler en place sur la matrice du jeu, à condition de supprimer les modifications quand on revient en arrière. Cela peut faire l'objet d'une question 6 !

La fonction `solutions` s'écrie alors facilement :

```
let rec solutions pos = match pos with
     | 1,m -> [[]]
     | n,m -> let p = ref (coups_possibles (n,m)) in
              let pile = ref [] in
                 while !p  < > [] do
                    let c = List.hd !p in
                       p : =  List.tl !p;
                       pile: =  (ajouter c (solutions (nouvelle_position (n,m) c)))@(!pile)
                 done;
                 !pile;;
```

On peut, avant même de lancer quelques calculs, se persuader de l'inefficacité de cette procédure, qui est beaucoup trop gourmande en espace mémoire et en temps de calcul.

```
> solutions pos2;;
- : ((int*int)*char) list list  =
     [[(3,4),‘d‘;(3,2),‘h‘;(2,4),‘d‘;(5,4),‘g‘;(3,4),‘h‘];
      [(3,4),‘d‘;(3,2),‘h‘;(2,4),‘d‘;(5,4),‘g‘;(3,5),‘b‘];
      [(3,4),‘g‘;(3,2),‘h‘;(4,4),‘g‘;(1,4),‘d‘;(3,4),‘h‘];
      [(3,4),‘g‘;(3,2),‘h‘;(4,4),‘g‘;(1,4),‘d‘;(3,5),‘b‘]]

> solutions pos3;;
- : ((int*int)*char) list list  =
     [[(2,3),‘g‘;(4,3),‘g‘;(3,1),‘h‘;(3,3),‘g‘;(0,3),‘d‘;(3,5),‘b‘;(2,3),‘d‘;(4,3),‘d‘];
      [(2,3),‘g‘;(4,3),‘g‘;(3,1),‘h‘;(3,3),‘g‘;(0,3),‘d‘;(3,5),‘b‘;(2,3),‘d‘;(5,3),‘g‘];
      [(2,3),‘g‘;(4,3),‘g‘;(3,1),‘h‘;(3,3),‘g‘;(3,5),‘b‘;(0,3),‘d‘;(2,3),‘d‘;(4,3),‘g‘];
      [(2,3),‘g‘;(4,3),‘g‘;(3,1),‘h‘;(3,3),‘g‘;(3,5),‘b‘;(0,3),‘d‘;(2,3),‘d‘;(5,3),‘g‘];
      [(2,3),‘g‘;(4,3),‘g‘;...];...]
```

Il y a 32 solutions différentes à partir de `Pos3` et 8 à partir de `Pos4`. Par contre, l'instruction `solutions Pos5;;` ne donne pas de résultat après 2 heures de calculs.

4) L'analyse donnée par l'énoncé conduit directement à la fonction :

```
let rec solution pos = match pos with
    | 1,m -> []
    | n,m -> let rec essai = function
                 [] -> failwith "pas de solution"
                 |c::l -> try c::(solution (nouvelle_position pos c))
                          with Failure "pas de solution" -> essai l in
             essai (coups_possibles (n,m));;
```

On obtient cette fois assez rapidement une solution :

```
> let p3 = solution pos3;;
- : p3 : ((int*int)*char) list  =
     [(4,3),‘d‘;(2,3),‘d‘;(3,5),‘b‘;(3,3),‘d‘;(6,3),‘g‘;(3,1),‘h‘;(4,3),‘g‘;(2,3),‘g‘]

> let p4 = solution pos4;;
- : p4 : ((int*int)*char) list  =
     [(3,4),‘g‘;(2,6),‘b‘;(4,6),‘g‘;(2,3),‘h‘;(2,6),‘b‘;
      (1,4),‘d‘;(3,5),‘b‘;(3,3),‘d‘;(4,5),‘b‘;(5,3),‘g‘]
```

Même à partir de la position `Pos0`, qui est la position initiale habituelle, une dizaine de minutes de patience permet d'obtenir une solution particulière :

```
> let p0 = solutions pos0;;
- : p0 : ((int*int)*char) list  =
     [(5,3),‘g‘;(4,5),‘b‘;(6,4),‘g‘;(6,2),‘h‘;(4,3),‘h‘;(4,6),‘b‘;(4,2),‘d‘;(4,0),‘h‘;
      (3,4),‘d‘;(6,4),‘g‘;(3,6),‘b‘;(3,4),‘d‘;(3,2),‘d‘;(6,2),‘g‘;(3,0),‘h‘;(3,2),‘d‘;
      (1,4),‘d‘;(2,6),‘b‘;(2,4),‘d‘;(5,4),‘g‘;(3,4),‘b‘;(2,2),‘d‘;(5,2),‘g‘;(2,0),‘h‘;
      (2,3),‘b‘;(0,2),‘d‘;(3,2),‘g‘;(0,4),‘b‘;(0,2),‘d‘;(2,1),‘h‘;(2,3),‘g‘]
```

5) Nous utilisons la même analyse que dans la question 4 :

```
let rec solution2 pos1 pos2 = match pos1,pos2 with
    | (n1,m1),(n2,m2) when n1 = n2 -> if m1 = m2 then [] else failwith "pas de solution"
    | _,_ -> let rec essai = function
                 [] -> failwith "pas de solution"
                 |c::l -> try c::(solution2 (nouvelle_position pos1 c) pos2)
                          with Failure "pas de solution" -> essai l in
             essai (coups_possibles pos1);;
```

Nous obtenons par exemple :

```
> let p21 = solution2 Pos2 Pos1;;
- : p21 : ((int*int)*char) list  =
     [(3,4),‘g‘;(3,2),‘h‘;(4,4),‘g‘;(1,4),‘d‘;(3,5),‘b‘]

> solution2 Pos0 Pos7;;
- : Uncaught exception: Failure "pas de solution"
```

Pour terminer, si `p` est une partie et `pos` une position, l'instruction `jouer p pos` permet de suivre la partie `p` à partir de la position initiale `pos` : une fois l'instruction lancée, il faut ouvrir la fenêtre graphique et le passage d'une étape à la suivante se fait en appuyant sur une touche du clavier (la bille déplacée devient tout d'abord bleue, avant de se déplacer en sautant une bille).

```
let jouer p pos = let pref = ref p and posref = ref pos in
  dessin pos;
  let a = ref(Graphics.read_key()) in
     while (!pref)  < > [] do
        let c = List.hd !pref in let (x,y) = centre(fst(c)) in
              Graphics.set_color Graphics.blue ;
              Graphics.moveto (x-h/2) (y-h/2);
              Graphics.lineto (x+h/2) (y-h/2);
              Graphics.lineto (x+h/2) (y+h/2);
              Graphics.lineto (x-h/2) (y+h/2);
              Graphics.lineto (x-h/2) (y-h/2);
              Graphics.set_color Graphics.black;
              a: =  Graphics.read_key();
              posref : =  nouvelle_position (!posref) c;
              pref : =  List.tl !pref;
              dessin (!posref);
              a: = Graphics.read_key();
    done;;
```

On peut ensuite lancer les instructions :

```
> jouer p3 pos3;;

> jouer p21 pos2;;

> jouer p0 pos0;;
```