Corrigé feuille d'exercices n°15
Jeu de la vie : sujet oral ENS juillet 2003

Pour formater la sortie des résultats, il est utile et recommandé d'utiliser la bibliothèque «printf» en standard dans Caml.

```
#open "printf";;
```

Question 1 : On choisit une méthode itérative pour calculer u_n. La fonction `Q1 u0 n` calcule u_n à partir de la donnée de u_0

```
let Q1 u0 n =
let temp = ref u0  in
for i = 1 to n do
   if (!temp * 16383 < 0) then failwith "erreur de débordement";
   temp := (!temp * 16383) mod 59047;
done;
!temp;;

Q1 12565 996;;   (* résultat trouvé = 41385 *)
Q1 12565 9996;;  (* résultat trouvé = 25658 *)
```

conclusion :

n	996	9996
u_n	41385	25658

Question 2 : On reprend la fonction précédente en ajoutant un compteur dans la boucle d'itération pour compter le nombre de v_i égaux à 1, c'est-à-dire, le nombre de u_i congru à 0 modulo 3.

```
let Q2 u0 n =
let temp = ref u0
   and compt = if n mod 3 = 0 then ref 0 else ref 1 in
for i = 1 to n do
   if (!temp * 16383 < 0) then failwith "erreur de débordement";
   temp := (!temp * 16383) mod 59047;
   if !temp mod 3 = 0 then incr compt;
done;
!compt;;

Q2 12565 9999;;   (* résultat trouvé = 3299 *)
```

Question 3 : Sachant que $u_1 = 14553$ qui est multiple de 3, on en déduit que la case $(1, 0)$ qui est associé à $v_{1+0.k} = v_1 = 1$ contient toujours une cellule vivante.

Question 4 : On commence par créer une fonction d'initialisation du tableau représentant l'univers. On peut penser à utiliser la fonction de la question 1 pour générer le tableau à l'instant de départ :

```
    let init k =
    let t = make_matrix k k false in
    for i = 0 to k-1 do
       for j = 0 to k-1 do
  if Q1 1024 (i+k*j) mod 3 = 0 then t.(i).(j) <- true
       done;
    done;
    t;;
```

Malheureusement, la machine s'écroule pour $k = 100$ car on obtient ainsi un algorithme de remplissage en $O(k^3)$. Il faut absolument éviter des calculs redondants. On peut remarquer que l'on remplit les cases avec les $(u_i \bmod 3)$ si on balaye le tableau ligne par ligne.

```
let init k =
let t = make_matrix k k false and u = ref 0 in
u := 12565; (* c'est le u0 de l'énoncé *)
for j = 0 to k-1 do
   for i = 0 to k-1 do
       if !u mod 3 = 0 then t.(i).(j) <- true;
       u := !u*16383 mod 59047;
   done;
done;
t;;
```

Ici la complexité est en $O(k^2)$ pour le remplissage du tableau, La première solution menant à une complexité en $O(n^3)$.

```
let CompteCellule t =
let compt = ref 0 and k = vect_length t in
for i = 0 to k-1 do
   for j = 0 to k-1 do
      if t.(i).(j) then incr compt;
   done;
done;
!compt;;
```

```
let copie t1 t2 = (* copie le tableau t1 dans t2 *)
let k = vect_length t1 in
for i=0 to k-1 do
  for j=0 to k-1 do
      t2.(i).(j) <- t1.(i).(j)
  done;
done;;

let eval b = if b then 1 else 0;;

let CompteVoisins k t i j =
let compt = ref 0 in
   for l = -1 to 1 do
     for m = -1 to 1 do
          compt := !compt + (eval t.((i+k+l) mod k).((j+k+m) mod k))
     done;
   done;
 !compt - (eval t.(i).(j));;

let IncrementeUniversFini temp t =
let k = vect_length temp in
for i = 0 to k-1 do
   for j = 0 to k-1 do
      temp.(i).(j) <- false;
      let nbvoisins = CompteVoisins k t i j in
         if not(t.(i).(j)) && nbvoisins = 3 then temp.(i).(j) <- true;
         if t.(i).(j) then
            (if  (nbvoisins = 2 || nbvoisins = 3)
                  then temp.(i).(j) <- true
                  else temp.(i).(j) <- false;);
    done;
 done;;

 let evolution t fint =
```

```
let k = vect_length t in
let temp = make_matrix k k false in
printf "temps = 0 nb cellules = %d\n" (CompteCellule t);
for tt = 1 to fint do
  IncrementeUniversFini temp t;
  copie temp t;
  if mem tt [1;10;100;1000;10000]
      then printf "temps = %d nb cellules = %d\n" tt (CompteCellule t);
done;;

let test k = evolution (init k) 10000;;

test 9;;
```

Dans la fonction «`evolution`», on calcule l'univers à l'instant $t+1$ à partir de l'univers à l'instant t. On parcourt toutes les cases du tableau, pour chaque, on calcule le nombre de ses voisins et en fonction de ce nombre,

conclusion :

k	t=0	t=1	t=10	t=100	t=1000	t=10000
9	31	25	15	3	3	3
20	163	125	80	8	8	8
20	578	573	395	158	44	44
100	3299	3674	2239	963	456	309

Évaluation de la complexité : en considérant que le temps d'analyse pour chacun des points du tableau est constant, on peut dire que la complexité de l'algorithme précédent est en $O(t * k^2)$ où t est le temps et k^2 la dimension de l'univers. En effet : Le calcul de la configuration de l'univers à l'instant $(t+1)$ à partir de la configuration de l'univers à l'instant t se fait en $O(n^2)$ et il faut répéter t fois l'opération pour arriver au résultat.

Compte tenu du fait que beaucoup de cellules disparaissent rapidement, on pourrait envisager de mémoriser l'emplacement de cellules vivantes et ne visiter que les cellules vivantes et leurs voisines pour passer d'un instant t à l'instant $t+1$.

Question 5

Le nombre de configurations pour l'univers est fini égal à 2^{k*k} donc nécessairement il existe deux valeurs de temps distinctes t_0 et t_1 telles que l'univers soit dans la même configuration pour ces deux instants ($|t_1 - t_0| \leq 2^{(k^2)}$).

Question 6

On va utiliser une heuristique pour limiter la place mémoire utilisée. On décide de caractériser un univers à l'aide de huit valeurs qui sont :

- le nombre de cellules,
- la somme des indices de lignes des cellules vivantes
- la somme des indices de colonnes des cellules vivantes
- $\sum_{(i,j)\text{cellules vivantes}} |i-j|$
- les abscisses et ordonnées minimales et maximales des cellules vivantes.

```
let caract t =
let k = vect_length t in
let compt1 = ref 0 and compt2 = ref 0 in
let compt3 = ref 0 and compt4 = ref 0 in
let imin = ref k and imax = ref 0 in
let jmin = ref k and jmax = ref 0 in
for i = 0 to k-1 do
   for j = 0 to k-1 do
       if t.(i).(j)
```

```
            then (incr compt1; compt2 := !compt2 + 1;
                compt3 := !compt3 + j;
                compt4 := !compt4 + (abs (i-j));
                if i < !imin then imin := i;
                if i > !imax then imax := i;
                if j < !jmin then jmin := j;
                if j > !jmax then jmax := j;
                );
   done;
done;
(!compt1,!compt2,!compt3,!compt4,!imin,!imax,!jmin,!jmax);;

(* Pour la fonction suivante, on suppose en plus que la période
sera inférieure à 20 et que $t1$ sera inférieur à 10000.  *)

 exception solution of int*int;;

 let periode t =
 let k = vect_length t in
 let temp = make_matrix k k false in
 let memoire = make_vect 10000 (caract t) in
 try
 for tt = 1 to 10000 do
    IncrementeUniversFini temp t;
    copie temp t;
    memoire.(tt) <- (caract t);
    for i = max 1 (tt-20) to tt-1 do
       if memoire.(i) = memoire.(tt) then raise (solution(i,tt-i))
    done;
    clear_graph (); dessineg t;
 done;
 with solution (a,b) -> printf "période = %d et temps = %d" b a;;

 let test5 k = periode (init k);;

 test5 9;;
```

Pour chaque instant t, on fait évoluer notre univers comme dans les fonctions des questions précédentes et en plus on stocke la caractérisation de notre univers décrite plus haut dans le tableau `memoire`. On compare ensuite cette valeur avec les 20 qui la précèdent. S'il y a égalité, il y a de fortes chances pour être à l'instant t_1. On arrête le programme et on retourne la valeur de la période et t_0.

conclusion :

k	période	t_0
9	2	35
20	1	66
40	2	635
100	2	1268

Évaluation de la complexité : La complexité est en $O(t_1 * k^2)$ résulte du calcul successif des configurations et du test d'égalité de la dernière configuration avec les vingt précédentes par l'intermédiaire de leur représentation compacte.

Question 7

```
#open "stack";;

(* fonction qui ajoute dans la pile "p" les voisins du point de
 * coordonnées (i,j) qui sont des cellules vivantes et qui ne
 * sont pas "cochées" dans le tableau ti :
 *   - (i,j) coordonnées du point étudié,
 *   - p : pile des coordonnées des points à étudier,
 *   - t tableau de la représentation de l'univers
 *   - ti tableau mémorisant les points déjà étudiés
 *   - compt : compteur des points de l'îlot *)

 let AjouteVoisin t i j pile =
 let k = vect_length t in
   for l = -1 to 1 do
     for m = -1 to 1 do
let a' = (i+k+l) mod k and b' = (j+k+m) mod k in
   if t.(a').(b') then push (a',b') pile;
done;
   done;;

 let NbCoiliens t i j =
 let k = vect_length t in
 let tab = make_matrix k k false in
 copie t tab;
 let pile = new() and compt = ref 0 in
 push (i,j) pile;
 while length pile <> 0 do
    let (a,b) = pop pile in
    if tab.(a).(b)
       then  (tab.(a).(b) <- false; incr compt;
      AjouteVoisin tab a b pile);
done;
!compt;;

for k = 0 to 3 do
   printf "nb de cellules de l'îlot de (1,0) pour k = %d est égal à %d\n"
             tk.(k) (NbCoiliens (init tk.(k)) 1 0);
done;;
```

Pour compter le nombre de cellules dans un îlot donné par un point initial, on le «marque» pour signalé qu'il a été pris en compte puis on empile tous les cellules vivantes qui sont ses voisines en les marquant. On dépile une cellule et on réitère le procédé sur cette cellule. On s'arrête quand la pile est vide. On va interroger au plus 8 fois un cellule et le nombre de cellules visités sera proportionnel au nombre de cellules de l'îlot. Notre algorithme a donc une complexité en $O(p)$ où p est le nombre de cellules de l'îlots.

conclusion :

k	9	20	40	100
nb cellules de l'îlot	1	141	200	53

Question 8

```
    let NbCoiliensBis t i j =
    let pile = new() and compt = ref 0 in
    push (i,j) pile;
    while length pile <> 0 do
       let (a,b) = pop pile in
       if t.(a).(b)
  then  (t.(a).(b) <- false; incr compt;
 AjouteVoisin t a b pile);
   done;
   !compt;;

   let NbIlots t =
   let k = vect_length t in
   let nbilots = ref 0 in
   for i=0 to k-1 do
      for j=0 to k-1 do
  if NbCoiliensBis t i j <> 0 then incr nbilots;
      done;
   done;
   !nbilots;;

   (* Pour chaque case de l'univers, on regarde si il y a
   une cellule vivante.
   Si c'est le cas, on incrémente le compteur d'îlots et on tue
   toutes les cellules vivantes qui sont dans la même composante
   connexe. Complexité en O(n^2) *)

    let EvolutionNbIlots t t0 =
     let k = vect_length t in
     let temp = make_matrix k k false in
     copie t temp;
     printf "temps = 0 nb îlots = %d\n" (NbIlots temp);
     for tt = 1 to t0 do
       IncrementeUniversFini temp t;
       copie temp t;
       if mem tt [1;10;t0] then
          printf "temps = %d nb ilots = %d\n" tt (NbIlots temp);
     done;;

    EvolutionNbIlots (init 9) 35;;
    EvolutionNbIlots (init 20) 66;;
    EvolutionNbIlots (init 40) 635;;
    EvolutionNbIlots (init 100) 1268;;
```

Évaluation de la complexité : Pour compter le nombre d'îlots, on parcourt toutes les cellules du tableau. Mais quand une cellule a été comptabilisé comme élément d'un îlot, elle a été «effacée» c'est-à-dire qu'il n'y aura rien à faire. On trouve une complexité en $O(k^2)+O(p)$ où p est le nombre de cellules vivantes. Or $p \leq k^2$, d'où une complexité en $O(k^2)$ pour chaque instant t et donc une complexité globale en $O(t_0.k^2)$.

conclusion :

t	0	1	10	t_0
9	2	3	3	1
20	10	16	12	2
40	45	42	40	11
100	363	176	261	68

Question 9

```
    let AjouteVoisinX t i j pile =
     let k = vect_length t in
       for l = -2 to 2 do
 for m = -2 to 2 do
    let a' = (i+k+l) mod k and b' = (j+k+m) mod k in
       if t.(a').(b') then push (a',b') pile;
    done;
       done;;

     let NbCoiliensX t i j =
     let k = vect_length t in
     let tab = make_matrix k k false in
     copie t tab;
     let pile = new() and compt = ref 0 in
     push (i,j) pile;
     while length pile <> 0 do
let (a,b) = pop pile in
if tab.(a).(b)
   then  (tab.(a).(b) <- false; incr compt;
  AjouteVoisinX tab a b pile);
    done;
    !compt;;

    for i = 0 to 3 do
       printf "k = %d et Nbiliens = %d\n" tk.(i) (NbCoiliensX (init tk.(i)) 1 0);
    done;;
```

L'algorithme est le même qu'à la question 7 sauf que la fonction de voisinage a été modifiée pour obtenir des X-voisinages.

conclusion :

k	9	20	40	100
nb cellules de l'île	31	163	578	3299

Question 10

```
    let NbCoiliensXbis t i j =
    let pile = new() and compt = ref 0 in
    push (i,j) pile;
    while length pile <> 0 do
       let (a,b) = pop pile in
       if t.(a).(b)
  then  (t.(a).(b) <- false; incr compt;
 AjouteVoisinX t a b pile);
   done;
   !compt;;

   let NbIles t =
   let k = vect_length t in
   let nbiles = ref 0 in
   for i=0 to k-1 do
      for j=0 to k-1 do
  if NbCoiliensXbis t i j <> 0 then incr nbiles;
      done;
   done;
   !nbiles;;

   (* Évolution au cours du temps *)
```

```
let EvolutionNbIles t t0 =
 let k = vect_length t in
 let temp = make_matrix k k false in
 copie t temp;
 printf "temps = 0 nb îles = %d\n" (NbIles temp);
 for tt = 1 to t0 do
    IncrementeUniversFini temp t;
    copie temp t;
    if mem tt [1;10;t0] then
 printf "temps = %d nb îles = %d\n" tt (NbIles temp);
 done;;

EvolutionNbIles (init 9) 35;;
EvolutionNbIles (init 20) 66;;
EvolutionNbIles (init 40) 635;;
EvolutionNbIles (init 100) 1268;;
```

Évaluation de la complexité : L'algorithme étant identique à la question 8, la complexité est toujours en $O(t_0 * k^2)$. Le nombre de cases exploré sera plus élevé dans le cas des X-voisinages. On peut penser que le temps d'exécution sera à peu près 3 fois plus élevé ($3 \sim \frac{25}{8}$).

conclusion : tableau du nombre d'îles en fonction du temps et de la dimension de l'univers.

t	0	1	10	t_0
k=9	1	1	1	1
k=20	1	1	2	2
k=40	1	1	1	8
k=100	1	2	25	61

Question 11

```
    (* On choisit de représenter un univers en stockant les coordonnées
     *  des points en lesquels il y a une cellule vivante *)
    let init_infini k =
    let l = ref [] and u = ref 0 in
    u := 12565; (* c'est le u0 de l'énoncé *)
    for j = 0 to k-1 do
       for i = 0 to k-1 do
         if !u mod 3 = 0 then l := (i,j)::!l;
         u := !u*16383 mod 59047;
       done;
    done;
    !l;;

(* pour compter le nombre de voisins du point $(i,j)$ dans un univers infini donné
sous forme de liste *)
let rec CompteVoisinsInfini i j = function
    [] -> 0
  | (a,b)::xs -> if abs(a-i)<=1 && abs(b-j) <= 1 && (a,b) <> (i,j)
    then 1 + CompteVoisinsInfini i j xs
    else CompteVoisinsInfini i j xs;;

(* Pour calculer l'univers à l'instant $t+1$ à partir celui à l'instant $t$ *)
let IncrementUnivers l =
 let nl = ref [] and ll = ref l in  (* nl : liste des points vivants à t+1 *)
    while !ll <> [] do
let (i,j) = hd !ll in
   let nbvoisins = CompteVoisinsInfini i j l in
if  (nbvoisins = 2 || nbvoisins = 3)
```

```
    then nl := (i,j)::!nl;
   for k = i-1 to i+1 do
      for m = j-1 to j+1 do
  if not(mem (k,m) l) && not(mem (k,m) !nl) && (CompteVoisinsInfini k m l = 3)
     then nl := (k,m)::!nl;
      done;
  done;
ll := tl !ll;
    done;
 !nl;;

let EvolutionInfini l fint =
printf "temps = 0 nb cellules = %d\n" (list_length l);
let l = ref l in  (* liste représentant l'univers courant *)
for tt = 1 to fint do  (* tt indice de temps *)
    l:=IncrementUnivers !l;
     clear_graph (); DessineInfini !l; (* pour le <<débugage>> *)
    if mem tt [1;10;100;1000;10000] then printf "temps = %d nb cellules = %d\n" tt (list_length !l);
done;;


EvolutionInfini (init_infini 9) 10000;;
```

conclusion : tableau du nombre de cellules en fonction du temps et de l'univers de départ caractérisé par k.

t	0	1	10	100	1000	10000
k=9	31	26	32	14	14	14
k=20	163	139	98	58	9	9
k=40	578	560	391	245	150	150

Question 12

```
let voisin (a,b) (c,d) =
  abs(a-c) <= 1 && abs(b-d) <= 1 && (a <> c || b <> d);;

let rec IlesInfini (a,b) l =
  let p = new() in
    l := EmpileVoisin (a,b) p !l;
    while (length p > 0) do
       let (c,d) = pop p in l := EmpileVoisin (c,d) p !l
    done
where
  rec EmpileVoisin (c,d) p = function
       [] -> []
     | (e,f)::xs -> if voisin (c,d) (e,f) then (push(e,f) p; EmpileVoisin (c,d) p xs)
                           else (e,f)::(EmpileVoisin (c,d) p xs);
;;

let NbIlesInfini univers =
let nb = ref 0 and ll = ref univers in
while !ll <> [] do
   incr nb;
   let (a,b) = hd(!ll) in
      ll := tl(!ll);
      IlesInfini (a,b) ll; (* supprime des coîliens de (a,b) *)
done;
!nb;;
```

```
let EvolutionNbIlesInfini l fint =
printf "temps = 0 nb iles = %d\n" (NbIlesInfini l);
let l = ref l in  (* liste représentant l'univers courant *)
for tt = 1 to fint do  (* tt indice de temps *)
    l:=IncrementUnivers !l;
    (* clear_graph (); DessineInfini !l; *)
    if mem tt [1;10;100;1000;10000] then printf "temps = %d nb d'îles = %d\n" tt (NbIlesInfini !
done;;

EvolutionNbIlesInfini (init_infini 9) 10000;;
EvolutionNbIlesInfini (init_infini 20) 10000;;
EvolutionNbIlesInfini (init_infini 40) 10000;;
```

conclusion : tableau du nombre d'îles en fonction du temps et de l'univers de départ caractérisé par k.

t	0	1	10	100	1000	10000
k=9	3	4	2	3	3	3
k=20	14	21	14	11	2	2
k=40	58	50	45	24	31	31

Question 13

```
let Q13 univers =
let n = 10 in
let memoire = make_vect (n+1) [] in
let ll = ref univers in
memoire.(0) <- univers;
for i = 1 to n do
   ll := IncrementUnivers !ll; memoire.(i) <- !ll;
done;
let coeur1 = ref(intersect memoire.(0) memoire.(6)) and
    coeur2 = ref(intersect memoire.(0) memoire.(8)) and
    coeur3 = ref(intersect memoire.(4) memoire.(10)) in
while !coeur1 <> !coeur2 || !coeur2 <> !coeur3 do
  for k = 0 to n-1 do memoire.(k) <- memoire.(k+1) done;
  ll:= IncrementUnivers !ll; memoire.(n) <- !ll;
  coeur1 := intersect memoire.(0) memoire.(6);
  coeur2 := intersect memoire.(2) memoire.(8);
  coeur3 := intersect memoire.(4) memoire.(10);
  clear_graph (); DessineInfini (!ll);
done;
printf "nombres de cellules du coeur = %d\n" (list_length !coeur1);
printf "nombres de glisseurs = %d\n" (NbIlesInfini (subtract !ll !coeur1));
clear_graph (); DessineInfini !ll;
;;

Q13 (init_infini 9);;
```

Pour trouver le cœur et les glisseurs, on fait l'hypothèse que le cœur est de période 2 et qu'à l'instant $t+6$ un glisseur c'est suffisamment déplacé pour qu'il n'intersecte la position qu'il occupait à l'instant t. On repère le cœur comme étant l'intersection des représentations de l'univers aux instants t et $t+6$. Si cette intersection ne varie plus, il y a de forte chance pour avoir le cœur. Les glisseurs sont données par le complémentaire du cœur dans l'ensemble des cellules vivantes et chaque glisseur constitue une île. Il ne reste plus qu'à compter le nombre d'îles pour trouver le nombre de glisseurs.

	nb d'éléments du cœur	nb de glisseurs
k=9	14	0
k=20	9	0
k=40	145	1