Annexe : Listing du programme de création de pavages

```
(*Tipe - Creation de pavages et affichage*)

type tuile = int*int*int*int;; (*Bas,Droite,Haut,Gauche*)

type jeu_tuile= tuile array;; (*Jeu de tuile donné sous forme
                                [|1;2;3;4|] *)
type motif= tuile array array;
type vecteur=int*int;; (* l-vecteur sous forme (2,4) *)
type voisinnage=vecteur list;; (* liste des conctraintes, en
                                    (x+1,y),... donné sous [v1;v2] *)
type relation=(voisinnage->bool);; (*ne sert à rien*)
type position= Haut | Bas | Droite | Gauche;;



(***Graphiques***)
#load"Graphics.cma";;
open Graphics;;


(*Définitions d'une squelette de tuile*)
let dessin long=      (*Prend en argument la longueur d'un côté*)
  set_color black;    (*trace les côtés en noir *)
  let x,y=current_point() in
  let inter=long/5 in (*créé des bords vides de 1/6 de côte, pour
                        mettre plus tard les couleurs*)
  lineto (x+long) y;
  lineto (x+long) (y+long);
  lineto (x) (y+long);
  lineto x y;
  lineto (x+inter) (y+inter);
  lineto (x+long-inter) (y+inter);
  lineto (x+long) y;
 lineto (x+long-inter) (y+inter);
  lineto (x+long-inter) (y+long-inter);
  lineto (x+long) (y+long);
  lineto (x+long-inter) (y+long-inter);
  lineto (x+inter) (y+long-inter);
  lineto x (y+long);
  lineto (x+inter) (y+long-inter);
  lineto (x+inter) (y+inter);;
```

```
let destuile cl1 cl2 cl3 cl4 taille x y= (*couleur ba dr ha ga
                                          longueur_côté posx posy*)
  let inter=taille/5 in
  moveto x y;
  dessin taille;
  set_color cl1;
  fill_poly
  [|(x+2,y+1);(x+taille-2,y+1);(x+taille-inter,y+inter-1);(x+inter,y+inter-1)|];
  set_color cl2;
  fill_poly
    [|(x-1+taille,y+2);(x+taille-1,y+taille-2);(x+taille-inter+1,y+taille-inter);(x+taille-
inter+1,y+inter)|];
  set_color cl3;
 fill_poly
  [|(x+taille-2,y+taille-1);(x+taille-inter,y+taille-inter+1);(x+inter,y+taille-
inter+1);(x+2,y+taille-1)|];
 set_color cl4;
 fill_poly
  [|(x+inter-1,y+taille-inter);(x+1,y+taille-2);(x+1,y+2);(x+inter-1,y+inter)|];
 moveto (x+1) (y+1);
 set_color black;
 lineto (x+1) (y+taille-1);
 lineto (x+taille-1) (y+taille-1);
 lineto (x+taille-1) (y+1);
 lineto (x+1) (y+1)
;;
open_graph "500x500";;
destuile  red  yellow green blue 100 200 75;;

clear_graph ();;



(**********Explications**********)
(*Programme remplissage d'un motif de taille 'n':
  -Créatoin d'une matrice n*n,
  -Appel récursif de l'algorithme de remplissage:
  - On envoie une copie de la matrice, ainsi que i j la nouvelle case
  devant être remplie, ainsi que le reste des tuiles à tester
  -si il ne reste qu'une tuile, on renvoie si il marche le résultat du
  test suivant
  -si i=0 la tuile est tout en haut => on ne teste pas le haut, si j=0
  la tuile est sur le bord gauche =>on ne teste pas la gauche,
  sinon on teste gauche et haut.
  - si la case peut être remplie par la tuile, on relance l'algo sur
  1) le prog avec le tableau initial 2) le prog avec la nouvelle
  tuile, i+1 j+1 jdt_original *)
```

```
let peut_aller (tuile : tuile) i j (mat : motif)=
  (*vérifie si la tuile peut aller dans la matrice, à la position i j*)
  let b,d,h,g=tuile in
  match i,j with
    |0,0->true (*si 0,0, c'est l'origine, pas de contrainte*)
    |0,_->let _,d1,_,_=mat.(i).(j-1) in d1=g (*pas de contrainte vers
                                              le haut*)
    |_,0->let b1,_,_,_=mat.(i-1).(j) in b1=h (*pas de contrainte vers
                                              la gauche*)
    |_->let _,d1,_,_=mat.(i).(j-1) and b1,_,_,_=mat.(i-1).(j) in d1=g
          && b1=h;;

let caz_suiv mat i j=
  (*On remplit la matrice de bas en haut, de gauche à droite*)
  let n=Array.length mat in
  if (i,j)=((n-1),(n-1)) then (n,n) (*cas inutile, mais qui sert plus bas*)
  else if (j)=(n-1) then (i+1,0) (*arrivé en bout de ligne, on renvoie
                                   à gauche sur la ligne dessous*)
  else (i,j+1);;


let copi_mat mat= (*programme de copie de matrices, parceque
                    Array.copy me faisait bugger*)
  let n=Array.length mat in
let res=Array.make_matrix n n mat.(0).(0) in
for i=0 to (n-1) do
  for j=0 to (n-1) do
res.(i).(j)<-mat.(i).(j)
  done;
done;
res;;



let rec remplir_case mat i j jdt k=
  (*renvoie la liste de toutes les possibilité des construction
    possibles à la case i,j à partir d'une matrice donnée*)
  let t=Array.length mat and n=Array.length jdt in
  if k=n then [] (*si on a parcouru tout le jdt, il ny a plus de possibilité*)
  else let res=remplir_case mat i j jdt (k+1) in (*on teste les autres
                                                   tuiles possibles*)
  if peut_aller jdt.(k) (i) (j) mat then
    begin (*si elle y va, on la rajoute*)
      let copi=copi_mat mat in
      copi.(i).(j)<-jdt.(k);
      copi::res
    end
  else res;;
```

```
let rec remplir_recu mat jdt i j=
  (*Recurrence qui renvoie exhaustivement les restricitions possibles*)
  let n=Array.length mat in
  if (i,j)=(n,n) then [mat]
  else  let (inew,jnew)=caz_suiv mat i j in (*la nouvelle tuile a
  testée sera*)
      let lst_res=remplir_case mat i j jdt 0 in
      let rec traitement lst= match lst with
          |[]->[]
          |t::q->(remplir_recu t jdt inew jnew)@(traitement q) in
      traitement lst_res;;



let remplir_un_motif largeur jdt= (*renvoie le premier motif trouvé*)
  let mat=Array.make_matrix largeur largeur (jdt.(0)) in
  let lst_result=remplir_recu mat jdt 0 0 in
  match lst_result with
    |[]->Array.make_matrix largeur largeur (0,0,0,0) (*Aucun pavage*)
    |t::q-> t;;
let remplir_tout_motif largeur jdt= (*renvoie la liste des motifs trouvés*)
  let mat=Array.make_matrix largeur largeur (jdt.(0)) in
  remplir_recu mat jdt 0 0 ;;

let a=remplir_tout_motif 4
  [|(1,2,1,1);(2,2,2,2);(3,1,3,2);(4,2,1,2)|];;

let b=remplir_un_motif 5 [|(1,2,3,4);(3,3,3,2);(4,2,1,2)|];;

let a1=remplir_un_motif 8
  [|(1,2,1,1);(2,2,2,2);(3,1,3,2);(4,2,1,2)|];;


let nb_to_coul nb= (*Ca se voit*)
  match nb with
    |1->red
    |2->green
    |3->blue
    |4->yellow
    |5->cyan
    |6->magenta
    |_->black;;
```

```
let dessin_pavage mot=
 (* set_color (rgb 200 200 200);
  fill_rect 0 0 500 500; *)
  let n=Array.length mot in
  let long=500/(n) in
  for i=0 to (n-1) do
    for j=0 to (n-1) do
      let b,d,h,g=mot.(i).(j) in
      let cb=nb_to_coul b and cd=nb_to_coul d
                              and ch=nb_to_coul h
                              and cg=nb_to_coul g
          (*Couleur bas, droite, haut, gauche*)
      in destuile cb cd ch cg long (j*long) (500-(i+1)*long)
    done;
  done;;


clear_graph ();;
dessin_pavage a1;;

let rec animation lst_mot=
  match lst_mot with
    |[]->()
    |t::q->dessin_pavage t;
           read_line ();
           animation q;;
animation quasiperio;;
List.length a;;
```

```
(******************Pavage
                    quasipériodique*****************************)
let
   jdtquasi=[|
(4,1,2,3);(4,2,3,1);(3,1,2,4);(4,1,3,2);(4,2,1,3);(2,3,4,1);(1,3,4,2);(3,1,2,3);(2,4,4,3);(4,3,4,1);
(3,3,4,2);(1,2,4,1);(2,1,3,2);(3,4,4,1)|];;
let creerpav lst=
  let nn=List.length lst in
  let n=int_of_float( sqrt(float_of_int nn)) in
  let lste=ref lst in
  let lst_to_tuile lst=
    match lst with
      |[]->(0,0,0,0),[]
      |t::q->jdtquasi.(t-1),q in
  let res=Array.make_matrix  n  n (0,0,0,0)  in
  for i=(n-1) downto 0 do
    for j=0 to (n-1) do
      let t,rest=lst_to_tuile (!lste) in
      lste:=rest;
      res.(i).(j)<-t
    done;
  done;
res;;
clear_graph ();;
let lstquasi=[
1;6;5;4;6;1;6;5;4;1;
13;2;7;8;2;13;2;7;8;1;
14;3;10;9;3;14;3;10;9;1;
1;6;5;4;6;1;6;5;4;1;
6;1;12;11;1;6;1;12;11;1;
1;6;5;4;6;1;6;5;4;1;
13;2;7;8;2;13;2;7;8;1;
14;3;10;9;3;14;3;10;9;1;
1;1;1;1;1;1;1;1;1;1;
1;1;1;1;1;1;1;1;1;1];;
let quasipa=creerpav lstquasi;;
dessin_pavage quasipa;;
(*Affichage des tuiles une par une*)
let affichtuiles jdt=
  let n=Array.length jdtquasi in
  for k=0 to (n-1) do
    let tui=jdt.(k) in let b,d,h,g=tui in
    let cb=nb_to_coul b and cd=nb_to_coul d
                              and ch=nb_to_coul h
                              and cg=nb_to_coul g in
    destuile cb cd ch cg (500/2) 50 50;
          read_line ();
  done;;
clear_graph ();;
affichtuiles jdtquasi;;
(**********************Retour au prog
                        principal*****************************)
```

```
(********    Partie III : les fonctions   *********)

let periodique pav=
  let n=Array.length pav in
  (*Cherchons s'il existe une périodicité*)
  let t=ref 1 and trouve=ref false
                and fini=ref false
                and j=ref 0
                and i=ref (n-1) in

 while (not (!fini)) && (!i>=0) do
   (*si il reste encore des valeurs possibles à tester pour t, et
     qu'on a pas atteint la première ligne*)
    (*On garde les valeurs possibles de t dans une liste, en effet, il
      nous faut seulement des multiples du premier t trouvé*)
    let k=ref 1 and tlst=ref [] in
    while (!k)*(!t)<n do
      tlst:=((!k)*(!t))::(!tlst);
      (k:=!k+1)
    done;
  (*Boucle de recherche de t*)
    while ((!j)+(!t)<n) do
      if pav.(!i).((!j)+(!t))=pav.(!i).(!j) then
           (
             incr j;
             trouve:=true;
           )
      else (
           if (!tlst)=[] then
             (* il n'y a plus de valeurs de t possible, on arrète*)
             (
               j:=n;
               trouve:=false
             )
           else (
             (* sinon, on prend la suivante dans la liste *)
           let suiv::queue=(!tlst) in
           t:=suiv;
           tlst:=queue;
           j:=0;
           trouve:=false;
            )
           )
    done;
    j:=0;
    if not !trouve then fini:=true
    else i:=!i-1;
  done;
  !trouve,(!t);;

periodique quasipa;;
let a2=List.nth a 13 ;;
dessin_pavage a2;;
```