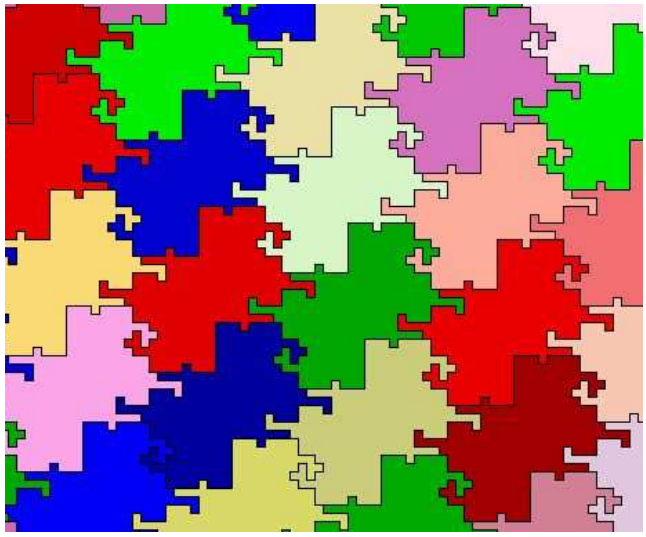
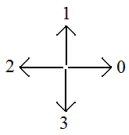
Pavages par polyominos à symétrie centrale



Comment réaliser un pavage du plan avec des pavés identiques dont la bordure est une succession de petits traits horizontaux et verticaux de même longueur, ces pavés se déduisant les uns des autres par translation ? Nous supposerons aussi que ces pavés présentent une symétrie centrale, et nous allons dégager des procédés algorithmiques qui permettent de construire de tels pavages.

1. Polyominos à symétrie centrale

De tels pavés sont appelés polyominos. Leur surface est formée de carrés de longueur unité accolés les uns aux autres côté contre côté, sans trou, et tels que chaque carré ait au moins un carré qui lui soit voisin avec un côté en commun[1]. Cela revient à dire que le périmètre du polyomino est une ligne brisée de traits horizontaux et verticaux qui se referme sur elle-même sans se recouper ni se toucher. En numérotant 0, 1, 2, 3 les quatre directions horizontales et verticales, la bordure du polyomino s'écrit sous forme d'un mot à base des quatre lettres 0, 1, 2, 3. La longueur de ce mot est paire lorsque le polyomino présente une symétrie centrale, puisqu'une moitié du mot se déduit de l'autre moitié par rotation de 180°. Plus précisément, à cause de cette rotation, la deuxième moitié du mot se déduit de la première en ajoutant 2 [modulo 4] à chaque lettre. Le mot est de la forme *m m'* où *m'* se déduit de *m* en ajoutant 2 [4] (*figure 1*).



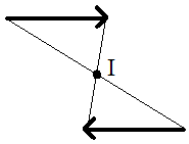


Figure 1 : Symétrie centrale de centre *I* (ou rotation de 180° autour de *I*). Un trait 0 devient 2, 1 devient 3, 2 devient 0, 3 devient 1.

Remarquons qu'un mot de la forme *m m'* donne un polyomino, et qu'un polyomino s'écrit sous la forme *m m'*, mais à un décalage cyclique près, le début du mot pouvant être choisi quelconque. On a aussi le même polyomino en lisant le mot à l'envers, ou encore en ajoutant 1 à chaque lettre, ou bien 2 ou bien 3, ce qui revient à tourner le polyomino d'un multiplie de 90°. Un exemple de polyomino à symétrie centrale est donné sur la *figure 2*.

[1] On dit que le polyomino est 4-connexe : chaque carrré qui le forme admet au moins un carré voisin au nord, au sud, à l'ouest ou à l'est.

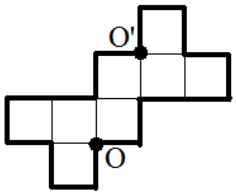


Figure 2 : Polyomino à symétrie centrale. Le mot correspondant, en partant du point *O* et en tournant dans le sens trigonométrique est : 010012123 232230301, la deuxième moitié de *O'* à *O* se déduisant de la première de *O* à *O'* en ajoutant 2 ramené modulo 4. Le centre de la symétrie est le milieu de [*OO'*].

Encore convient-il que la frontière du polyomino ne se recoupe ni ne se touche : aucun cycle ne doit se produire. Quelques faux polyominos sont montrés sur la *figure 3*.

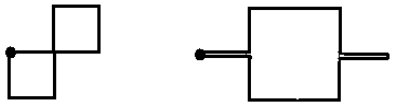


Figure 3 : *A gauche* le mot 0103 2321 donnant un faux polyomino à symétrie centrale, *à droite* le mot 010030 232212 donne aussi un faux polyomino.

2. Frise de polyominos

En vue du pavage futur, commençons par accoler deux polyominos, et supposons que la bordure commune est de longueur paire. Prenons *O* comme milieu de cette partie commune. En répétant ce procédé de collage, on obtient une frise infinie de polyominos (*figure 4*). Remarquons que si la partie commune n'est pas de longueur paire, il est possible de doubler les dimensions du polyomino pour retomber dans le cas d'une zone commune de longueur paire. Remarquons aussi qu'il existe plusieurs façons d'accoler les polyominos de façon à former une frise. Il suffit que les zones communes soient diamétralement opposées, cette zone commune pouvant se réduire à un seul côté unitaire.

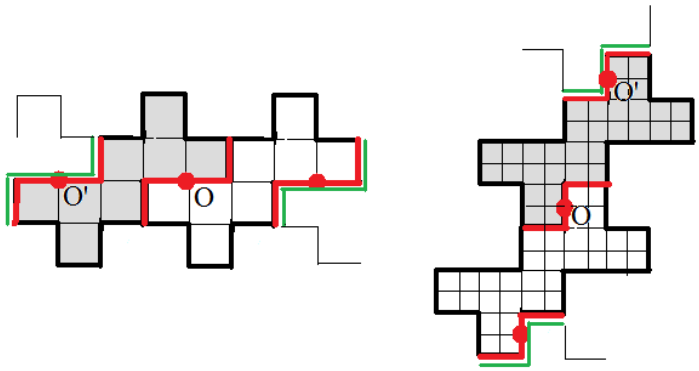


Figure 4 : *A gauche* polyominos accolés, avec la frontière commune en rouge. La répétition de ce collage donne une frise de polyominos. *A droite*, la partie commune a une longueur impaire, avec son centre *O* qui n'est pas un sommet du quadrillage. Pour avoir une partie commune de longueur paire et *O* sur le quadrillage, il suffit de prendre des carrés de longueur moitié par rapport à la figure de gauche, ce qui revient à doubler chaque lettre du mot.

Le demi-mot *m*, qui va de *O* à *O'*, point diamétralement opposé à *O*, s'écrit *m = d M f*, où *d* est la moitié de la partie commune, et l'on prend *f* à la fin qui a la même longueur que *d*. Le mot du polyomino est donc *d M f d' M' f'*. Nécessairement une zone de collage avec un polyomino voisin se fait autour de *O'*, et le polyomino voisin a son origine *O* en *O'*, ce qui signifie que son début *d* vient se plaquer sur le *f* du polyomino. Comme la lecture de *d* sur le polyomino voisin se fait en sens inverse de celle de *f* sur le polyomino, *f* se déduit de *d* en pratiquant une lecture en sens inverse et en ajoutant

2, comme par exemple pour d = 01, on a f = 32 (*figure 4 à gauche*) ou encore pour d = 011, f = 332 (*figure 4 à droite*). A son tour, la fin f' du mot du polyomino s'obtient en ajoutant 2 à f, ce qui donne $f' = \underline{d}$ (d lu à l'envers), et c'est la deuxième moitié du collage. Dans l'exemple de la *figure 4 à gauche*, on obtient le mot :

$\underline{01}\ 21232\ \underline{32}\ \ \underline{23}\ 03010\ \underline{10}$ soit une zone de collage qui est $f'\ d$ = 10 01.
$d\quad M\quad f\quad d'\quad M'\quad f'$

La zone de collage $f'd$ est donc symétrique par rapport à O.

3. Pavage de polyominos

La frise obtenue a maintenant comme bordures inférieure et supérieure $MMM\ldots$ et $M'M'M'\ldots$, M' se déduisant de M en ajoutant 2. Mais si on lit le mot M' dans l'autre sens, cela revient encore à ajouter 2 et d'autre part à pratiquer un effet miroir, ce qui donne finalement le mot miroir de M (M lu à l'envers), noté $\underline{M}$. Par exemple, pour M = 21232, $\underline{M}$ = 23212.

Pour que la frise puisse se développer pour donner un pavage[2], il convient que la bordure inférieure $MMM\ldots$ se colle sur la bordure supérieure $\underline{MMM}\ldots$, ce qui impose que les mots M et $\underline{M}$ soient conjugués, c'est-à dire identiques à un décalage cyclique près. Cela signifie que M s'écrit sous la forme $M = P_1\ P_2$, et comme son conjugué est $P_2\ P_1$, et que $\underline{M} = \underline{P_2}\ \underline{P_1}$, cela signifie que $P_1 = \underline{P_1}$ et $P_2 = \underline{P_2}$. Les mots P_1 et P_2 sont des palindromes : ils sont les mêmes qu'on les lise dans un sens ou dans l'autre. Ainsi :

Un polyomino à symétrie centrale permet un pavage du plan lorsque le mot correspondant est de la forme *m m'* avec *m'* se déduisant de *m* en ajoutant 2 [4], et avec $m = d\ P_1\ P_2\ f$, où P_1 et P_2 sont des palindromes et *f* se déduit de *d* en ajoutant 2 et en lisant à l'envers.

Exemple : Polyomino qui s'écrit $m\ m'$ avec $m = \underline{012}\ \underline{212}\ \underline{3223}\ \underline{032}$
$d\quad P_1\quad P_2\quad f$

Une partie du pavage est montrée sur la *figure 5*. Les vecteurs de base des translations qui permettent le pavage sont **OO'** et **OO''**.

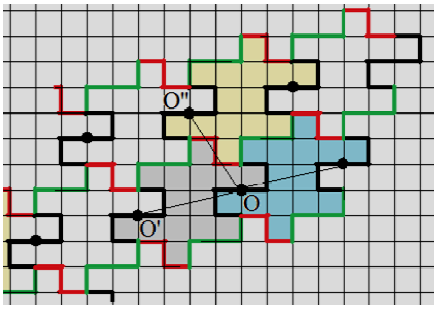


Figure 5 : Pavage à base d'un polyomino à symétrie centrale

A ce stade, nous pouvons réécrire le mot $m\ m' = d\ M\ f\ d'\ M'\ f'$ (avec $M = P_1\ P_2$) en faisant un décalage cyclique, de façon à obtenir le mot :

$m\ m' = f'\ d\ M\ \ f\ d'\ M'$

[2] Il est entendu qu'un tel pavage est constitué de copies translatées du même polyomino.

Comme on l'a vu, *f'* est aussi *d* lu à l'envers, et à son tour *f'd* est un palindrome, que nous notons P_0. Finalement le mot associé au polyomino peut s'écrire *m m'* avec $m = P_0\, P_1\, P_2$ et *m'* se déduisant de *m* en ajoutant 2 à chaque lettre.[3]

Un polyomino à symétrie centrale permet de paver le plan par translations lorsque le mot correspondant est de la forme *m m'* avec *m'* se déduisant de *m* en ajoutant 2 [4], et avec $m = P_0\, P_1\, P_2$, où P_0, P_1 et P_2 sont des palindromes. Cela sous réserve que la bordure du polyomino ainsi obtenue ne présente aucun cycle.

Remarques

1) Dans notre raisonnement, nous avions supposé que la zone de collage entre deux polyominos successifs constituant la frise était de longueur paire, ce qui conduit pour le pavage final à un palindrome P_0 de longueur paire. Mais cela reste valable pour un palindrome P_0 de longueur impaire.

En effet, en cas d'une zone de collage de longueur impaire, nous avons proposé de doubler les dimensions du polyomino pour se ramener au cas précédent, ce qui revient à doubler chaque lettre du mot de bordure. Dans ces nouvelles conditions *d* et *f'* sont de longueur impaire, la zone de collage *f'd* a une longueur paire, mais avec *f'* qui découle de *d* en ajoutant 2 et en lisant à l'envers, *f'd* est un palindrome dont chacune des lettres est répétée deux fois. En revenant aux dimensions initiales, le palindrome P_0 a une longueur impaire.

Exemple : Considérons ce polyomino (*figure 6*), avec une zone de collage de longueur impaire. Commençons par prendre un quadrillage deux fois plus petit. Le mot de bordure à partir du point *O* s'écrit *m m'* avec *m* = 100 00111100 33000033 223 et *m'* = 322 22333322 11222211 001. On est bien dans les conditions d'un pavage. Après un décalage cyclique de trois crans le mot de bordure devient *m m'* avec *m* = 001100 00111100 33000033 et *m'* qui s'en déduit en ajoutant 2. En revenant aux dimensions initiales, le mot de bordure s'écrit *m m'* avec *m* = 010 0110 3003, soit trois palindromes avec le premier P_0 de longueur impaire.

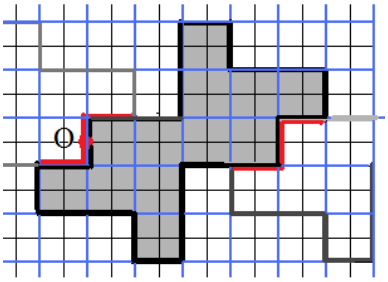


Figure 6 : Polyomino à zone de collage impaire, en rouge autour de *O*, dans le quadrillage bleu. En prenant un quadrillage deux fois plus petit, la zone de collage devient paire.

2) En prenant les points correspondant aux extrémités des palindromes d'un polyomino, ces points sont les sommets d'un hexagone à côtés opposés parallèles et de même longueur (*figure 7*).

[3] Un autre moyen d'arriver à ce résultat consiste à utiliser une propriété due à D. Beauquier et M. Nivat [BEA1991] : Un polyomino pave le plan (par translations) si et seulement si le mot de bordure s'écrit *X Y* Z *X' Y' Z'*, où *X'* s'obtient en ajoutant d'abord 2 [4] aux lettres de *X* puis en lisant le mot obtenu à l'envers, et de même avec *Y'* et *Z'*. En imposant en plus que le polyomino présente une symétrie centrale, cela impose que la moitié de la bordure *X' Y' Z'* se déduise de *X Y Z* en ajoutant 2, ce qui impose que *X' Y' Z'* donne le même mot en le lisant à l'envers, d'où *X' Y' Z'* sont des palindromes, et de même *X*, *Y* et *Z*.

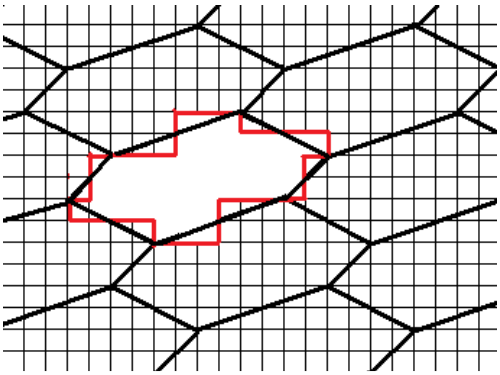


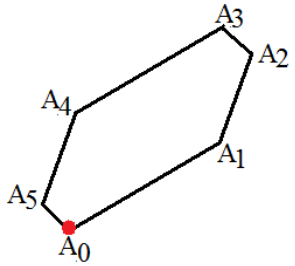
Figure 7 : Le polyomino en rouge et l'hexagone sous-jacent en noir, ce qui va permettre le pavage du plan.

Nous allons maintenant donner deux façons de construire ces polyominos à symétrie centrale donnant lieu à des pavages. La première est particulière, dans la mesure où les chemins en marches d'excalier qui forment le contour du polyomino sont assez proches des côtés de l'hexagone sous-jacent. La deuxiième est plus générale et conduit à des formes plus variées de polyominos.

4. Cas particulier : le polyomino est proche de l'hexagone sous-jacent

Nous commençons par construire au hasard l'hexagone à côtés opposés parallèles et égaux, en évitant que ses côtés ne se recoupent. Puis nous remplacerons chacun de ses côtés par une ligne brisée en escalier proche de chaque côté, de façon à obtenir un polyomino qui ressemble à l'hexagone. Enfin nous dessinerons le pavage.

Première étape : construction de l'hexagone



L'hexagone est noté A_0 A_1 A_2 A_3 A_4 A_5. Le point A_0 est pris comme origine du repère. On commence par prendre le point A_1 au hasard, avec des coordonnées toutes deux positives, ce qui ne nuit en rien à la généralité de la construction. Comme nous l'expliquerons plus tard, les coordonnées du vecteur $\mathbf{A_0A_1}$, tout comme celles des autres vecteurs associés aux côtés, ne peuvent pas être toutes les deux impaires, seuls les autres cas sont possibles. Puis on construit au hasard le point A_2 en faisant en sorte qu'il soit à gauche du vecteur $\mathbf{A_0A_1}$, toujours sans perte de généralité. Enfin on construit au hasard le point A_3, et les points A_4 et A_5 se déduisent des précédents, ce qui constitue la fin du tracé. Mais il convient d'éviter que les segments $[A_0A_1]$ et $[A_2A_3]$ ne se coupent, de même que les segments $[A_1A_2]$ et $[A_3A_4]$, et aussi les segments $[A_2A_3]$ et $[A_4A_5]$. On répète donc le choix au hasard du point A_3 jusqu'à ce qu'il en soit ainsi. On en déduit le programme de la fonction *hexagone*(). Il est supposé que les variables intervenant dans cette fonction sont déclarées en global.

```
void hexagone(void)
{ int h;
  filldisc(xorig,yorig,6,red); /* point A0 sur l'écran */
  xA1=4+rand()%20;    /* premier trait A0 A1, le point A1 ayant pour coordonnées xA1 et yA1, et le vecteur
                         A0A1 ayant pour coordonnées X1 et Y1, celles-ci étant égales à celles du point A1 */
  if (xA1%2==1) yA1 = 4+2*(rand()%10); else yA1=4+rand()%20; /* coordonnées entre 4 et 23 */
  X1=xA1;Y1=yA1;
  xeA1=xorig+zoom*xA1,yeA1=yorig-zoom*yA1;  /* coordonnées écran de A1 */
  do      /* deuxième trait A1 A2, le vecteur A1A2 ayant pour coordonnées X2 et Y2, positives ou négatives */
    { X2=4+rand()%20; h= 2*(rand()%2)-1; if (h==-1)X2=-X2;  ./* h vaut 1 ou – 1 */
      if (abs(X2)%2==1) Y2 = 2*(rand()%10); else Y2=rand()%20;
      h= 2*(rand()%2)-1; if (h==-1)Y2=-Y2;
      xA2=xA1+X2; yA2=yA1+Y2; /* coordonnées de A2 */
```

```
    }
  while(gauche(0,0,xA1,yA1,xA2,yA2)==0); /* A2 à gauche du vecteur A0A1 */
  xeA2=xorig+zoom*xA2; yeA2=yorig-zoom*yA2;
  do     /* troisième trait A2 A3, le vecteur A2A3 ayant pour coordonnées X3 et Y3    */
    { X3=4+rand()%20;h= 2*(rand()%2)-1; if (h==-1)X3=-X3;
      if (abs(X3)%2==1) Y3 = 2*(rand()%10); else Y3=rand()%20;
      h= 2*(rand()%2)-1; if (h==-1)Y3=-Y3;
      xA3=xA2+X3; yA3=yA2+Y3;xA4=xA3-xA1;yA4=yA3-yA1; xA5=xA4-X2;yA5=yA4-Y2;
     }
   while (secoupent(0,0,xA1,yA1,xA2,yA2,xA3,yA3)==1
            || secoupent(xA1,yA1,xA2,yA2,xA3,yA3,xA4,yA4)==1
            || secoupent(xA4,yA4,xA5,yA5,xA2,yA2,xA3,yA3)==1);
   xeA3=xorig+zoom*xA3; yeA3=yorig-zoom*yA3;
   xeA4=xorig+zoom*xA4; yeA4=yorig-zoom*yA4;
   xeA5=xorig+zoom*xA5; yeA5=yorig-zoom*yA5;
   circle(xeA1,yeA1,4,red);line(xorig,yorig,xeA1,yeA1,black); /* tracé de l'hexagone et de ses sommets */
   circle(xeA2,yeA2,4,red); line(xeA1,yeA1,xeA2,yeA2,black);
   circle(xeA3,yeA3,4,red);line(xeA2,yeA2,xeA3,yeA3,black);
   circle(xeA4,yeA4,4,blue);line(xeA3,yeA3,xeA4,yeA4,black);
   circle(xeA4,yeA4,4,red);line(xeA4,yeA4,xeA5,yeA5,black);
   circle(xeA5,yeA5,4,red);line(xeA5,yeA5,xorig,yorig,black);
}
```

Avec l'appoint des fonctions *gauche*(*x0, y0, x1*, *y1*, *x2*, *y2*) pour savoir si le point numéro 2 est à gauche du vecteur **01**, et de la fonction *secoupent*() pour savoir si les segments 01 et 23 se coupent :

```
int gauche (int x0, int y0, int x1, int y1, int x2, int y2)
  {  int dx1,dy1,dx2,dy2,det;
     dx1=x1 -  x0 ;dy1= y1 - y0 ; dx2 = x2 - x0 ; dy2 = y2 -y0 ;
     det= dx1*dy2 - dy1*dx2 ;
     if (det>0) return 1;
     return 0;
  }
int secoupent (int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3)
  {  int dx1,dy1,dx2,dy2,dx3,dy3,dx4,dy4,dx5,dy5,dx6,dy6;
     int det1,det2,det3,det4;
     dx1=x1- x0 ;dy1= y1-y0 ; dx2 = x2-x0 ; dy2 = y2-y0 ;dx3 = x3-x0 ; dy3= y3-y0 ;
     det1= dx1*dy2 - dy1*dx2 ;det2= dx1*dy3 - dy1*dx3 ;
     dx4=x3- x2 ;dy4= y3-y2 ; dx5 = x0-x2 ; dy5 = y0-y2 ;dx6 = x1-x2 ; dy6= y1-y2 ;
     det3= dx4*dy5 - dy4*dx5 ;det4= dx4*dy6 - dy4*dx6 ;
     if (det1*det2<0 && det3*det4<0) return 1;
     return 0;
  }
```

Deuxième étape : remplacement de chaque côté de l'hexagone par un chemin en marches d'escalier

Comment remplacer un vecteur associé à un côté par un cheminement en marches d'escalier qui soit proche du côté et qui soit aussi un palindrome ? Prenons le cas du premier côté $[A_0A_1]$ dont le vecteur associé a pour coordonnées X_1 et Y_1, celles-ci étant toutes deux positives. Les traits horizontaux ou vecticaux qui forment les marches d'escalier doivent être symétriques par rapport au milieu du segement $[A_0A_1]$, et nous allons les construire en avançant pas à pas à partir de la gauche tout en reculant en même temps à partir de la droite. On voit aussitôt qu'il est impossible de le faire si les deux coordonnées X_1 et Y_1 du vecteur sont impaires, comme on le voit sur la *figure 8*.

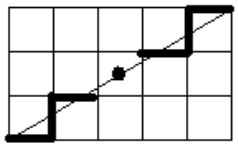


Figure 8 : Exemple du vecteur (5, 3), et construction symétrique gauche-droite du chemin en marches d'excalier, aboutissant à une impossibilité.

Par contre, tous les autres cas sont possibles. Ils sont illustrés sur la *figure 9*.

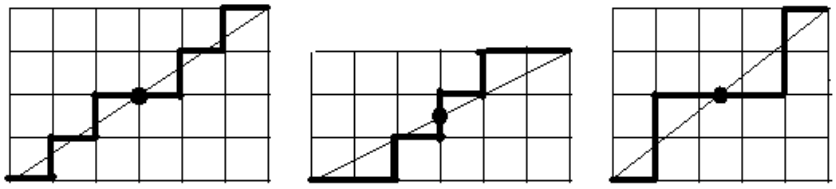


Figure 9 : *A gauche*, le vecteur (6, 4) a ses deux côtés paires, *au centre* le vecteur (6, 3) a une abscisse paire et une ordonnée impaire, *à droite* le vecteur (5, 4) a son abscisse impaire et son ordonnée paire.

A chaque étape de la construction gauche-droite, on détermine la pente du segment joignant le point où l'on est arrivé à gauche et le point où l'on est en même temps à droite. Si cette pente est inférieure à 1, on décide de faire un trait horizontal à gauche ainsi qu'à droite, et l'on diminue la distance horizontale notée *xx* de 2 avant de passer à l'étape suivante. Si elle est supérieure ou égale à 1, on trace un trait vertical vers le haut à gauche et vers le bas à droite, et l'on diminue la distance de séparation verticale *yy* de 2. Et ainsi de suite jusqu'à ce que l'on tombe sur une distance horizontale *xx* ou verticale *yy* nulle. Il ne reste plus qu'à combler la zone centrale. Par exemple, sur la *figure 9 à gauche*, lorsque l'on tombe sur *yy* = 0, on a *xx* = 2, et l'on trace le segment horizontal final de longueur 2. Sur la *figure 9 an centre ou à droite*, on tombe soit sur *xx* = 0, auquel cas on trace un dernier trait vertical de jonction, ou sur *yy* = 0, auquel cas on ajoute un trait horizontal.

La fonction correspondante *escalier*() enregistre dans un tableau *a*[] le mot associé au chemin en marches d'escalier. Ce mot est formé de 0 et de 1 dans le cas où les coordonnées du vecteur sont toutes deux positives. La longueur du mot, qui est aussi le nombre de traits horizontaux et verticaux, est enregistrée dans la variable *longueur*. Pour traiter les cas où les deux coordonnées ne sont pas positives, on les particularise grâce à une variable *flag*, puis on rend positives les deux coordonnées pour se ramener au cas précédent, et enfin, suivant la valeur de *flag* on modifie les 0 en 2 ou les 1 en 3 suivant les cas.

```
void escalier(int xx, int yy)  /* escalier à partir du vecteur xx, yy */
{  int k,i,flag;
   k=0 ;
   if (yy==0 && xx>=0) { longueur=xx; for(i=0;i<xx; i++) a[i]=0;} /* cas particuliers */
   else if (yy==0 && xx<0) { longueur=-xx; for(i=0;i<-xx; i++) a[i]=2;}
   else if (yy>=0 && xx==0) { longueur=yy; for(i=0;i<yy; i++) a[i]=1;}
   else if (yy<0 && xx==0)  {longueur=-yy; for(i=0;i<-yy; i++) a[i]=3;}
   else
    {
      if (xx>0 && yy>0) flag=0; /* distinction de quatre cas suivant les signes des coordonnées */
     else if (xx>0 && yy<0) { yy=-yy;flag=1;}
     else if (xx<0 && yy>0) {xx=-xx; flag=2;}
     else if (xx<0 && yy<0) { yy=-yy;xx=-xx; flag=3;} /* les coordonnées sont rendues posivives */
     do         /* construction de proche en proche avec les coordonnées positives */
       { if ((float)yy/(float)xx <1.) { a[k]=0;xx-=2;k++; }
         else { a[k]=1;yy-=2;k++;}
       }
     while (yy!=0 && xx!=0);
```

```
    longueur = 2*k;
    if (yy==1 && xx==0) { a[k]=1; longueur++;}
    else if (xx==1 && yy ==0) {a[k]=0; longueur++;}
    else if (xx==2 && yy==0) {a[k++]=0; longueur = 2*k;}
    else if (xx==0 && yy==2) {a[k++]=1; longueur = 2*k;}
    for(i=0; i<k;i++) a[longueur-1-i]=a[i];  /* on enregistre la deuxième moitié du mot */

    if (flag==1) for(i=0;i<longueur;i++) {if (a[i]==1) a[i]=3;}
    else if (flag==2) {for(i=0;i<longueur;i++) { if (a[i]==0) a[i]=2;}}
    else if (flag==3) {for(i=0;i<longueur;i++)  {a[i]=a[i]+2; }}
   }
}
```

Chaque fois que le mot *a*[] est enregistré avec sa *longueur* pour un vecteur comme (X_1, Y_1), on dessine le chemin correspondant sur l'écran, grâce à la fonction *dessin*() qui convertit les chiffres du mot en traits.

```
void dessin(int xe0,int ye0,Uint32 c)  /*( xe0, ye0) est le point origine du vecteur concerné sur l'écran */
{ int i,xe,ye,newxe,newye;
  xe=xe0;ye=ye0;   /* (xe, ye) est le point courant */
  for(i=0;i<longueur;i++)  /* longueur est une variable globale, correspondant à la longueur du chemin */
    { if (a[i]==0)  { newxe=xe+zoom; newye=ye;line(xe,ye,newxe,newye,c);xe=newxe;ye=newye;}
      else if (a[i]==1)  { newxe=xe; newye=ye-zoom; line(xe,ye,newxe,newye,c);xe=newxe;ye=newye;}
      else if (a[i]==2) { newxe=xe-zoom; newye=ye;line(xe,ye,newxe,newye,c);xe=newxe;ye=newye;}
      else if (a[i]==3) { newxe=xe; newye=ye+zoom; line(xe,ye,newxe,newye,c);xe=newxe;ye=newye;}
    }
}
```

Troisième étape : le programme principal

Le programme principal consiste à appeler d'abord la fonction *hexagone*(). Puis on dessine les chemins en marches d'excalier associés aux trois vecteurs $\mathbf{A_0A_1}$, $\mathbf{A_1A_2}$ et $\mathbf{A_2A_3}$, grâce aux fonctions *escalier*() et *dessin*(), et à chaque fois on enregistre dans un nouveau tableau *m*[] le mot associé aux trois chemins mis en succession, en gérant sa longueur *L*. Mais cela ne suffit pas. Encore convient-il que les traits de jonctions entre deux chemins associés à deux vecteurs successifs ne se recouvrent pas. La dernière lettre *dernier1* du premier vecteur doit être différente de celle du premier vecteur *premier2* du deuxième vecteur augmentée de 2 modulo 4, sinon les deux traits seraient confondus. De même pour la dernière lettre du deuxième vecteur et la première du troisième, et aussi pour la dernière du troisième et la première du quatrième. On répète donc le choix au hasard de l'hexagone jusqu'à ce qu'il en soit ainsi. Il ne reste plus qu'à construire le contour complet du polyomino, grâce à la fonction *tracecomplet*().

```
do
 { SDL_FillRect(screen,0,white);
   hexagone();
   escalier(X1,Y1); dessin(xorig,yorig,red);
   L=longueur;
   for(i=0;i<L;i++) m[i]=a[i];
   dernier1=a[L-1];
   escalier(X2,Y2); dessin(xeA1,yeA1,green);
   premier2=a[0];
   for(i=0;i<longueur;i++) m[i+L]=a[i];
   L+=longueur;
   dernier2= m[L-1];
   escalier(X3,Y3); dessin(xeA2,yeA2,blue);
   premier3=a[0];
   dernier3=a[longueur-1];
   for(i=0;i<longueur;i++) m[i+L]=a[i];
```

```
   L+=longueur;
 }
while ( dernier1==(premier2+2)%4 || dernier2==(premier3+2)%4 || dernier3==dernier1);
tracecomplet();
SDL_Flip(screen);   pause();SDL_FillRect(screen,0,white);
pavage();
SDL_Flip(screen);   pause();
```

avec la fonction qui trace le contour :

```
void tracecomplet(void)
{ int i,xe,ye,newxe,newye;
for(i=0;i<L;i++)  m[i+L]=(m[i]+2)%4; /* deuxième moitié du contour */
L=2*L;  /* L est la longueur totale du contour */
xe=xorig;ye=yorig;
for(i=0;i<L;i++)
if (m[i]==0)
  { newxe=xe+zoom; newye=ye;line(xe,ye,newxe,newye,blue);xe=newxe;ye=newye;}
else if (m[i]==1)
  { newxe=xe; newye=ye-zoom; line(xe,ye,newxe,newye,blue);xe=newxe;ye=newye;}
else if (m[i]==2)
  { newxe=xe-zoom; newye=ye;line(xe,ye,newxe,newye,blue);xe=newxe;ye=newye;}
else if (m[i]==3)
  { newxe=xe; newye=ye+zoom; line(xe,ye,newxe,newye,blue);xe=newxe;ye=newye;}
}
```

Quatrième étape : construction du pavage

Dans le programme principal précédent, nous avons rajouté à la fin une fonction *pavage*() qui dessine le pavage par les polyominos suivant une double translation. On peut choisir comme vecteurs de base des translations les vecteurs $\mathbf{V}_1$ ($X_1 + X_2$, $Y_1 + Y_2$) et $\mathbf{V}_2$ ($xA_2 - xA_4$, $yA_2 - yA_4$) comme indiqué sur la *figure 10*. Les polyominos du pavage sont ensuite obtenus par duplication en faisant des translations de vecteurs $k\,\mathbf{V}_1 + k'\,\mathbf{V}_2$, k et k' étant des entiers relatifs.

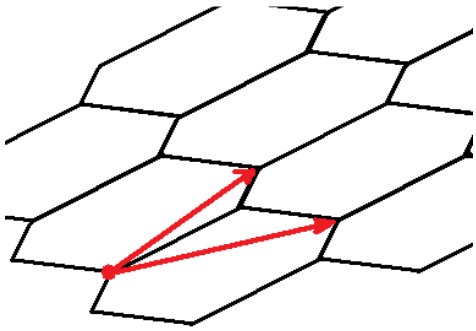


Figure 10 : Les deux vecteurs de base pour les translations

```
void pavage(void)
{int i,xV1,yV1,xV2,yV2,k,kk,N,xo,yo,xe,ye,newxe,newye;
N=25;
xV1=xA1+X2; yV1=yA1+Y2; /* vecteurs de base des translations */
xV2=xA2-xA4; yV2=yA2-yA4;
for(k=-N;k<=N;k++) for(kk=-N;kk<=N;kk++) /* combinaisons des vecteurs de base */
   { xo=xorig+(k*xV1+kk*xV2)*zoom;  /* origine A0 du polyomino après translation */
     yo=yorig-(k*yV1+kk*yV2)*zoom;
     xe=xo;ye=yo;
     for(i=0;i<L;i++)  /* dessin du contour du polyomino translaté */
       { if (m[i]==0) {newxe=xe+zoom;newye=ye;}
         else if (m[i]==1) {newxe=xe;newye=ye-zoom; }
```

```
          else if (m[i]==2) {newxe=xe-zoom;newye=ye; }
          else if (m[i]==3) {newxe=xe;newye=ye+zoom; }
          if (xe>zoom && xe<800-zoom && ye>zoom && ye<600-zoom) /* pour rester dans l'écran */
         line(xe,ye,newxe,newye,black);
         xe=newxe;ye=newye;
        }
    }
}
```

Quelques résultats sont donnés sur la *figure 11*.

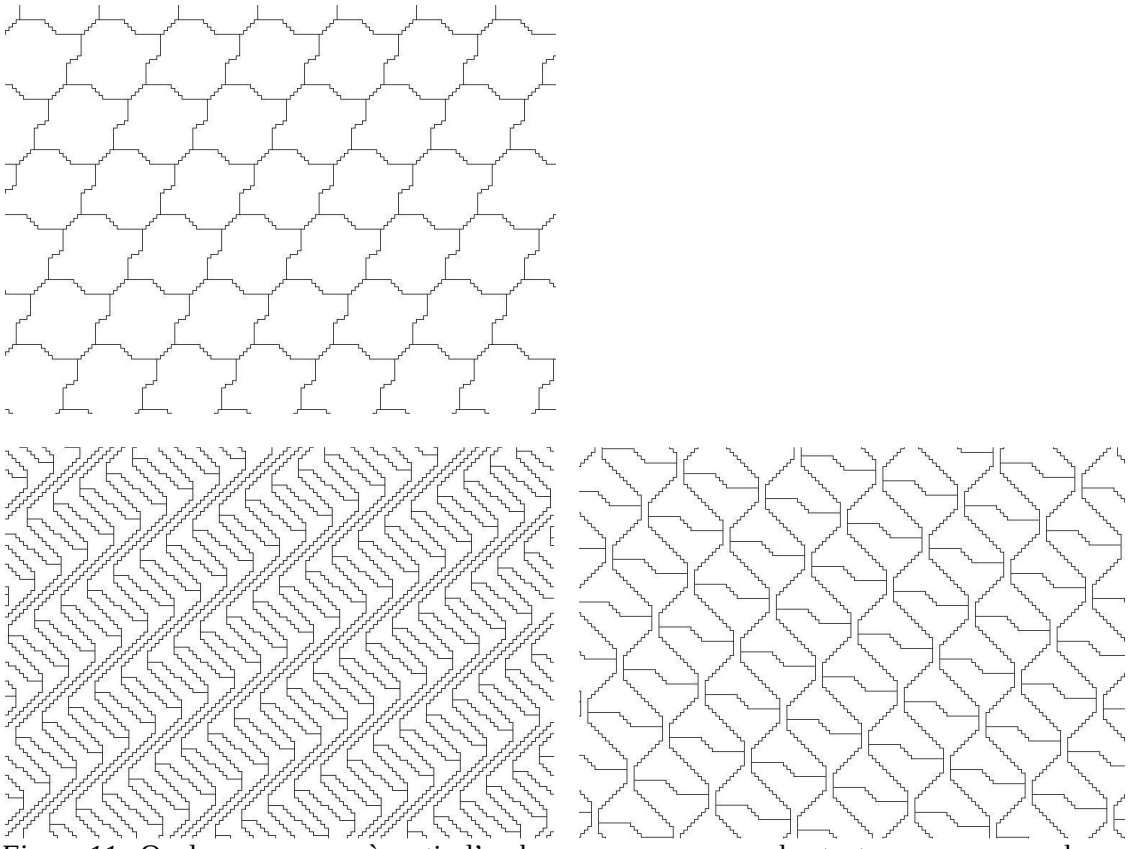


Figure 11 : Quelques pavages, à partir d'un hexagone convexe en haut, et non convexe en bas.

5. Cas général

Cette méthode est très différente de la précédente. Maintenant nous nous donnons les trois longueurs -nombre de traits horizontaux et verticaux- L_0, L_1, L_2 des trois palindromes qui caractérisent le polyomino à symétrie centrale, les trois autres s'en déduisant. Comme il s'agit de palindromes, ces longueurs sont nécessairement paires. A partir de ces trois longueurs, nous constuisons au hasard les palindromes. Evidemment, il existe de nombreuses contraintes, à savoir que les dessins correspondant aux trois palindromes ne se touchent ni ne se recoupent eux-mêmes ni les uns les autres, et que les trois palindromes qui terminent le contour ne recoupent pas les trois premiers. En particulier, il convient que le premier trait d'un palindrome ne se confonde pas avec le dernier trait du précédent. Ce qui donne un programme principal qui s'écrit ainsi :

```
toutvabien=0;   /* variable toutvabien qui vaut 0 (pour NON) ou 1 (pour OUI) */
while(toutvabien==0)  /* tant que cela ne va pas */
   { SDL_FillRect(screen,0,white);  /* on efface tout avant chaque nouvel essai */
     x[0]=0; y[0]=0;x[1]=1; y[1]=0; m[0]=0;m[L0-1]=0;  /* premier trait horizontal vers la  droite */
     toutvabien=1;
     palindrome0();
```

```
    if (toutvabien==1) premierelettre1();
    if (toutvabien==1) palindrome1();
    if (toutvabien==1) premierelettre2();
    if (toutvabien==1) palindrome2();
    if (toutvabien==1) deuxiememoitie();
  }
SDL_Flip(screen);  pause();  SDL_FillRect(screen,0,white);srand(time(NULL));
pavage();  SDL_Flip(screen);  pause();
```

Ce programme indique que l'on ne commence une nouvelle étape de la construction que si tout va bien jusque là. Autrement dit, dès qu'un problème se pose, une nouvelle construction est relancée au hasard à partir du début. Avant d'aboutir à un polyomino valable, de multiples essais vont donc être faits, et cela d'autant plus que les longueurs L_0, L_1, L_2 sont grandes. Dans notre programme nous les avons prises entre 20 et 30.

Prenons maintenant chacune des fonctions intervenant dans le programme principal.

Premier palindrome, numéroté 0

Rappelons que dans le programme principal nous avons choisi un premier trait horizontal dirigé vers la droite, ce qui ne nuit en rien à la généralité de la construction. Dans le mot *m*[] associé au contour, cela impose *m*[0] = 0 et *m*[*L*0 – 1] = 0. Par la même occasion on a enregistré les corrdonnées calcul des extrémités du premier trait, avec *x*[0] = 0, *y*[0] = 0, *x*[1] = 1, *y*[1] = 0. Toutes ces variables sont déclarées en global. On construit le palindrome 0 au hasard en faisant en sorte que sa moitié gauche soit symétrique de sa moitié droite, le mot correspondant étant enregistré dans *m*[]. Puis on enregistre les coordonnées des extrémités de chaque trait dans *x*[] et *y*[], grâce à la fonction *pointsuivant*(). On réitère cette opération jusqu'à ce que le palindrome ne présente aucune boucle, grâce à la fonction *cycle*().

Enfin, on teste si l'extrémité du palindrome n'est pas un cul-de-sac (*figure 12*), ce qui bloquerait le palindrome suivant dès son premier trait. Si un tel blocage arrive, on met la variable *toutvabien* à 0, ce qui provoque la relancee de la boucle principale du programme principal.

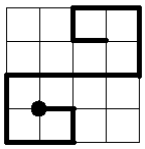


Figure 12 : Cas où le palindrome se termine en cul-de-sac, ce qui empêche tout prolongement du contour.

```
void palindrome0(void)
{ int i,g,d;
  do
     { d=L0-2; /* d pour à droite */
       for(g=1;g<=L0/2;g++)  /* g pour : à gauche */
           { m[g]=rand()%4; m[d]=m[g]; d--; }
       for(i=0;i<L0;i++) pointsuivant(i);
      }
   while(cycle(0,L0)==1 );
   for(i=0;i<L0;i++)
   linewithwidth(xorig+zoom*x[i],yorig-zoom*y[i],xorig+zoom*x[i+1],yorig-zoom*y[i+1],1,black);
   if (getpixel(xorig+zoom*(x[L0]+1),yorig-zoom*y[L0])==black
       && getpixel(xorig+zoom*(x[L0]-1),yorig-zoom*y[L0])==black
       && getpixel(xorig+zoom*x[L0],yorig-zoom*(y[L0]+1))==black
       &&  getpixel(xorig+zoom*x[L0],yorig-zoom*(y[L0]-1))==black)
   toutvabien=0;
}
```

Avec l'appoint de la fonction *pointsuivant*(*i*) qui à partir du point numéro *i* de coordonnées x_i et y_i détermine le point suivant x_{i+1} et y_{i+1} :

```
void pointsuivant(int i)
{ if (m[i]==0) {x[i+1]=x[i]+1; y[i+1]=y[i];}
  else if (m[i]==1) {x[i+1]=x[i]; y[i+1]=y[i]+1;}
  else if (m[i]==2) {x[i+1]=x[i]-1; y[i+1]=y[i];}
  else if (m[i]==3) {x[i+1]=x[i]; y[i+1]=y[i]-1;}
}
```

Et de la fonction *cycle* (*idebut*, *ifin*) qui détermine si, entre les deux points numérotés *idebut* et *ifin*, il existe deux points de coordonnées égales, ramenant 1 (OUI) si cela se produit, ce qui correspond à la présence d'une boucle, ou 0 (NON) s'il n'existe aucun recoupement.

```
int cycle(int idebut,int ifin)
{ int i,j;
   for(i=idebut;i<ifin-1;i++)  for(j=i+1;j<ifin;j++)
   if (x[i]==x[j] && y[i]==y[j]) return 1;
   return 0;
}
```

Première lettre du palindrome 1

Comme nous avons évité de tomber dans un cul-de-sac à la fin du palindrome 0, le premier trait du palindrome 1 va être obtenu sans contrainte, il suffit que son point final ne tombe pas sur le palindrome 0 colorié en noir.

```
void premierelettre1(void)
{ int h,xx,yy;
  do { h=rand()%4; /* un des quatre voisins */
       if (h==0) {xx=x[L0]+1; yy=y[L0];}
       else if (h==1) {xx=x[L0]; yy=y[L0]+1;}
       else if (h==2) {xx=x[L0]-1; yy=y[L0];}
       else if (h==3) {xx=x[L0]; yy=y[L0]-1;}
     }
  while (getpixel(xorig+zoom*xx,yorig-zoom*yy)==black); /* éviter une boucle avec le palindrome 0 */
  P1[0]=h; P1[L1-1]=h;  /* enregistrement dans le tableau P1[] */
  line(xorig+zoom*x[L0],yorig-zoom*y[L0],xorig+zoom*xx,yorig-zoom*yy,red);
}
```

Palindrome 1

Le programme du palindrome 1 ressemble à celui du palindrome 0, avec des contraintes supplémentaires. D'abord il convient que son tracé ne se recoupe pas. Mais il faut aussi que son tracé ne recoupe pas (ou ne touche pas) le tracé du palindrome 0. Ce cas se présente assez souvent, lorsque le palindrome 0 a son point final entouré d'une cavité où le palindrome 1 ne peut pas s'épancher complètement. Si cela arrive, on recommence tout. Comme on l'a fait précédemment, on vérifie aussi que la première lettre du palidrome 2 doit avoir de la place pour se placer, sinon on recommence tout.

```
void palindrome1(void)
{ int g,d,i;
    do
     {
       d=L1-2;
       for(g=1;g<=L1/2;g++) /* on commence avec g = 1 car on connaît déjà la première lettre */
           { P1[g]=rand()%4; P1[d]=P1[g]; d--;  }
       for(i=L0;i<L0+L1;i++) m[i]=P1[i-L0]; /* on envoie le talbeau P1[] dans le mot de contour m[] */
```

```
        for(i=L0;i<L0+L1;i++)   pointsuivant(i);
       }
   while(cycle(L0,L0+L1)==1); /* tant que le palindrome 1 présente une boucle */
   for(i=L0;i<L0+L1;i++)
   linewithwidth(xorig+zoom*x[i],yorig-zoom*y[i],xorig+zoom*x[i+1],yorig-zoom*y[i+1],1,red);
   if (cycle(0,L0+L1)==1) toutvabien=0; /* si les deux palindromes se recoupent, on recommence tout */
   if (getpixel(xorig+zoom*(x[L0]+1),yorig-zoom*y[L0])!=white  /* a-t-on un cul de sac ? */
       && getpixel(xorig+zoom*(x[L0]-1),yorig-zoom*y[L0])!=white
       && getpixel(xorig+zoom*x[L0],yorig-zoom*(y[L0]+1))!=white
       &&  getpixel(xorig+zoom*x[L0],yorig-zoom*(y[L0]-1))!=white)
   toutvabien=0;
}
```

Première lettre du palindrome 2

La fin du programme précédent empêche le premier trait du palindrome 2 de toucher les palindromes précédents. Mais il s'ajoute une nouvelle contrainte : la première lettre étant aussi la dernière du palindrome 2, le trait correspondant à cette dernière lettre ne doit pas se confondre avec le premier trait du palindrome suivant, qui n'est autre que le palindrome 0 auquel on ajoute 2 [4]. Cela impose la contrainte $m[L0+L1] \neq m[L0-1]$. Si l'on n'arrive pas à cette condition au bout d'un certaine nombre d'essais, on recommence tout.

```
void premierelettre2(void)
{int h,xx,yy;
  compteur=0;
  do { h=rand()%4; compteur++;
        if (h==0) {xx=x[L0+L1]+1; yy=y[L0+L1];}
       else if (h==1) {xx=x[L0+L1]; yy=y[L0+L1]+1;}
       else if (h==2) {xx=x[L0+L1]-1; yy=y[L0+L1];}
       else if (h==3) {xx=x[L0+L1]; yy=y[L0+L1]-1;}
      }
 while (h==m[L0-1]);
  P2[0]=h; P2[L2-1]=h;
  if (compteur==100) toutvabien=0;
}
```

Palindrome 2

Le programme est construit suivant le même modèle que les palindromes précédents.

```
void palindrome2(void)
{  int g,d,i;
   do
       {  d=L2-2;
         for(g=1;g<=L2/2;g++)  { P2[g]=rand()%4; P2[d]=P2[g]; d--;}
         for(i=L0+L1;i<L0+L1+L2;i++) m[i]=P2[i-L0-L1];
         for(i=L0+L1;i<L0+L1+L2;i++) pointsuivant(i);
       }
   while(cycle(L0+L1,L0+L1+L2)==1); /* le palindrome ne doit pas se retroucher */
   if (cycle(0,L0+L1+L2)==1) toutvabien=0;
   for(i=L0+L1;i<L0+L1+L2;i++) /* le palindrome ne doit pas retoucher les palindromes précédents */
   linewithwidth(xorig+zoom*x[i],yorig-zoom*y[i],xorig+zoom*x[i+1],yorig-zoom*y[i+1],1,green);
}
```

Deuxième moitié du contour

Il suffit de reprendre le mot *m*[] construit jusqu'à présent en lui ajoutant 2 modulo 4. Encore convient-il que le tracé de cette deusième moitié ne retouche pas la première moitié du contour.

```
void deuxiememoitie(void)
{ int i;
  for(i=0;i<(L0+L1+L2);i++)   m[i+L0+L1+L2]=(m[i]+2)%4;
  for(i=L0+L1+L2;i<2*(L0+L1+L2);i++) pointsuivant(i);
  for(i=L0+L1+L2;i<2*(L0+L1+L2);i++)
  linewithwidth(xorig+zoom*x[i],yorig-zoom*y[i],xorig+zoom*x[i+1],yorig-zoom*y[i+1],1,blue);
  if (cycle(0,2*(L0+L1+L2))==1) toutvabien=0;  /* si la deuxième moitié touche la première, on
                                                  recommence tout */
}
```

Sur la *figure 13*, on trouvera quelques résultats de polyominos ainsi obtenus.

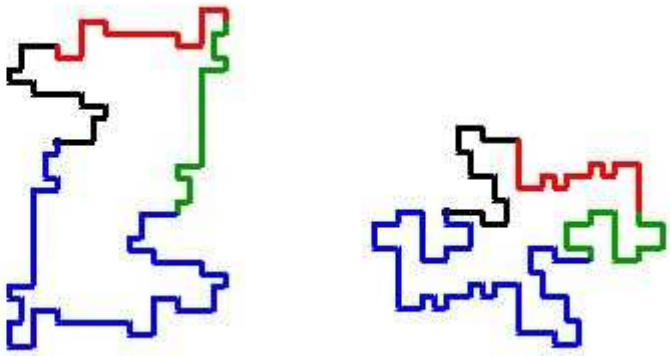
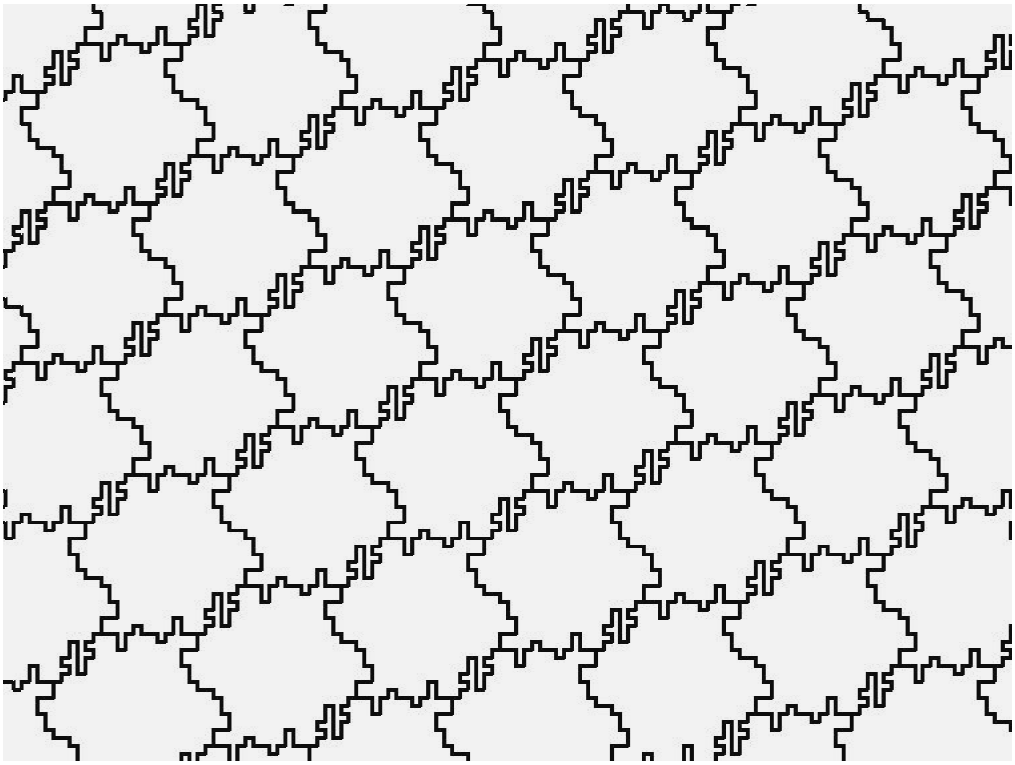


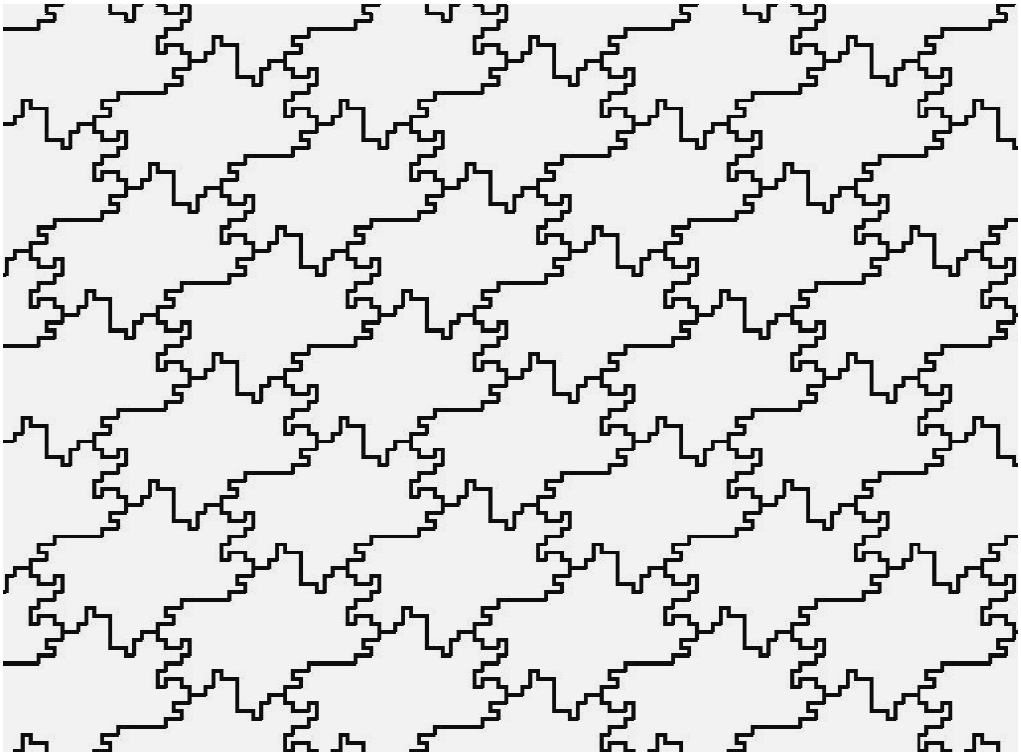
Figure 13 : Deux exemples de polyominos, avec $L_0 = L_1 = L_2 = 24$. Le palindrome 0 est en noir, le palindrome 1 en rouge, le palindrome 2 en vert, et la deuxième moitié du contour est en bleu.

Pavage

Une fois le polyomino construit, il ne reste qu'à construire la pavage correspondant. Le programme est quasiment le même que celui fait au *paragraphe 4*. Des résultats sont donnés sur la *figure 14*.

```
void pavage(void)
{int i,xV1,yV1,xV2,yV2,k,kk,N,xo,yo,xe,ye,newxe,newye;
N=25;
xV1=x[L0+L1]; yV1=y[L0+L1];
xV2=x[L0+L1]-x[L0+L1+L2+L0]; yV2=y[L0+L1]-y[L0+L1+L2+L0];
for(k=-N;k<=N;k++) for(kk=-N;kk<=N;kk++)
   { xo=xorig+(k*xV1+kk*xV2)*zoom;    yo=yorig-(k*yV1+kk*yV2)*zoom;
     xe=xo;ye=yo;
     for(i=0;i<2*(L0+L1+L2);i++)
       { if (m[i]==0) {newxe=xe+zoom;newye=ye;}
         else if (m[i]==1) {newxe=xe;newye=ye-zoom; }
         else if (m[i]==2) {newxe=xe-zoom;newye=ye; }
         else if (m[i]==3) {newxe=xe;newye=ye+zoom; }
         if (xe>zoom && xe<800-zoom && ye>zoom && ye<600-zoom)
        line(xe,ye,newxe,newye,black);
        xe=newxe;ye=newye;
       }
    }
}
```





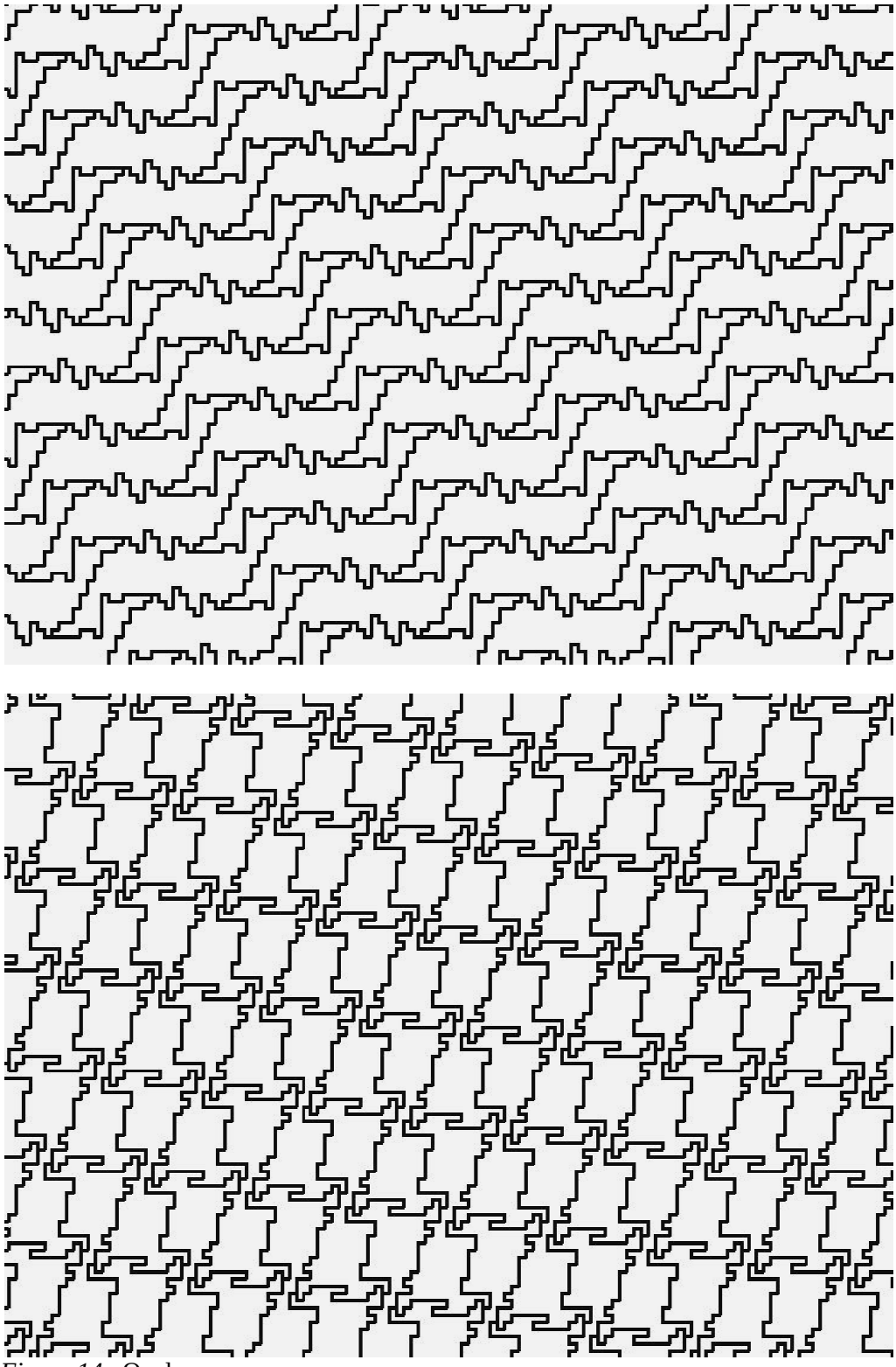


Figure 14 : Quelques pavages

Référence bibliographique

[BEA1991] D. Beauquier, M. Nivat, *On translating one polyomino to tile the plane*, *Discrete Comput. Geom.*, 6, 1991.