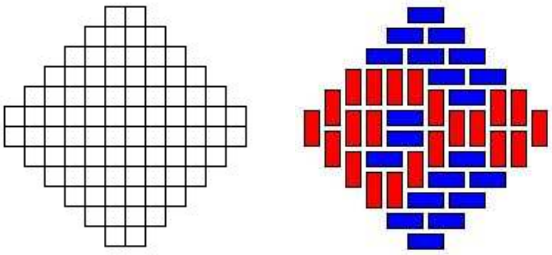
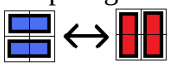
# Pavage du diamant aztèque par des dominos

## De l'ordre au désordre, et du désordre à l'ordre. Lien avec les mots de Schroeder.



Dans un autre document,[1] nous avions programmé la construction d'un pavage aléatoire du « diamant aztèque » par des dominos, en procédant par grossissements progressifs du diamant aztèque (voir ci-dessus un dessin de diamant aztèque de longueur $2L = 12$), suivant la méthode proposée par T. de la Rue et E. Janvresse.[2]

Nous allons reprendre et développer ce thème en utilisant une autre méthode, celle due à Sen-Peng Eu et Tung-Shan Fu [EU2005] qui ont établi un lien entre un pavage et des dessins de « chaînes de montagnes », appelées plus savamment mots de Schroeder. Nous allons utiliser cette méthode pour construire un pavage relativement aléatoire pour des diamants aztèques de dimension moyenne. Mais l'avantage essentiel de cette méthode est de permettre de passer d'un pavage aléatoire au pavage où tous les dominos sont horizontaux, c'est-à-dire du désordre à l'ordre.[3]

Enfin, nous indiquons comment construire un pavage aléatoire par le moyen le plus simple qui soit, en partant d'un pavage tout horizontal, et en provoquant au hasard une succession de rotations de la forme 

## 1. Pavage du diamant aztèque et chaînes de montagnes

Partons d'un pavage quelconque du diamant aztèque d'ordre $L$ par des dominos, comme sur la *figure 1*. Puis prenons les points situés au milieu des segments verticaux de la bordure sud-ouest du diamant. En remplaçant chaque type de domino, tous de dimension 2 sur 1, par un vecteur, ce qui donne trois cas (deux cas pour un domino vertical et un cas pour un domino horizontal, des chemins en forme de lignes brisées peuvent être construits à partir des $L$ points de la bordure sud-ouest, par jonction des vecteurs correspondant aux dominos concernés (*figure 2*), étant entendu que si le premier domino est vertical, on choisit le vecteur diagonal dirigé vers le haut.

Ces chemins à base de trois traits forment des chaînes de montagnes, ce qui signifie que leur point de départ et leur point d'arrivée sont au même niveau, et que jamais le chemin ne descend au-dessous

[1] *Pavage au hasard d'un diamant aztèque par des dominos*, dans *combinatoire, probabilités, travaux complémentaires*, sur mon site *audibertpierre.fr* ou encore *ai.univ-paris8.fr/~audibert*.

[2] *Pavages aléatoires par touillage de dominos*, sur le site *Images des Mathématiques, CNRS 2009*.

[3] Nous avions indiqué comment faire dans [AUD2009].

de ce niveau.[4] Ces chaînes de montagnes sont aussi à une distance verticale d'au moins une unité entre elles.[5] Cela signifie qu'elles ne se touchent ni ne se croisent. Ainsi, à chaque pavage d'un diamant aztèque d'ordre *L* par des dominos correspondent *L* chaînes de montagnes telles que nous les avons définies.

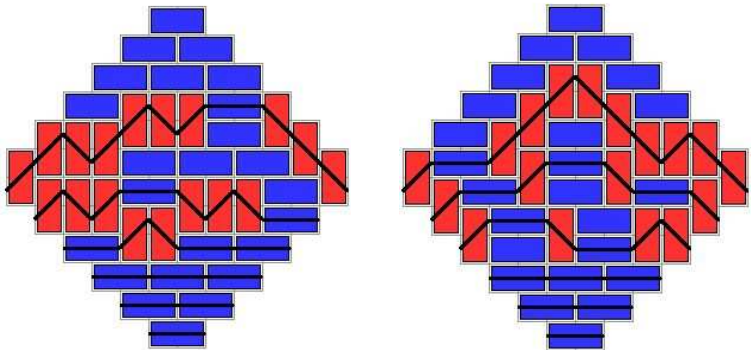


*Figure 1* : Deux pavages d'un diamant aztèque d'ordre 6, et leurs six chaînes de montagne issues de la bordure sud-ouest

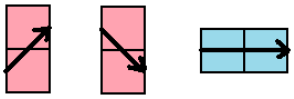


*Figure 2* : Les vecteurs associés aux trois types de dominos, deux étant diagonaux et un horizontal, de coordonnées respectives (1, 1), (1, - 1), (2, 0)

Inversement, partons de *L* chaînes de montagnes partant de la bordure sud-ouest du diamant aztèque, celles-ci étant au moins à une distance unité les unes des autres. Cela donne *L* rubans de dominos qui couvrent partiellement le diamant aztèque (*figure 3*). On constate alors que la zone non

[4] Démontrons-le pour le ruban central de dominos, associé à la chaîne de montagnes de longueur 2*L*. Supposons que la case de départ est noire, à la hauteur 0, et appelons domino *M* un domino dont le vecteur est (1, 1), allant d'une case noire à une case blanche, et domino *D* un domino dont le vecteur est (1, - 1), allant d'une case blanche à une case noire.

1) Sur le ruban central, il y autant de dominos *M* que de domino *D*, ce qui signifie qu'en partant du niveau 0 on arrive au niveau 0.. En effet s'il y avait un domino *M* de plus que de dominos *D*, la longueur du chemin serait impaire, et le domino final serait à la hauteur 1. Mais la case finale du diamant aztèque à la hauteur 1 est à la distance 2*L* paire de la case initiale. Contradiction. S'il y avait deux dominos *M* de plus que de dominos *D*, on arriverait à la hauteur 2, avec une longueur parcourue 2*L* – 2. Mais la case finale du diamant aztèque située à la hauteur 2 est à la distance 2*L* – 1 de la case initiale. Contradiction. Et ainsi de suite selon qu'il y a un nombre pair ou un nombre impair de différence entre le nombre de dominos *M* et le nombre de dominos *D*, le nombre de dominos *M* étant supérieur. Mais ce raisonnement n'est plus valable lorsqu'il y a plus de dominos *D*.

S'il y a un domino *D* de plus que de dominos *M*, il y a à la hauteur - 1 un nombre impair de dominos, et un nombre pair au-dessous, rendant impossible le pavage sous le ruban central. Et de même s'il y en a plus.

2) Le ruban central ne descend jamais au-dessous du niveau 0. En effet, les dominos *M* et les dominos *D* peuvent être associés deux par deux lorsqu'ils sont à la même hauteur *h*. Entre ces couples de dominos à la même hauteur, il y a toujours un nombre pair de cases vides au-dessus de la hauteur *h*. Mais si un domino descend au-dessous du niveau 0, il a à sa gauche une case noire vide de plus que de cases blanches vides, rendant le pavage par des dominos impossible en-dessous du ruban central.

Le nombre de cases vides en-dessous du ruban central est donc pair, et celui de celles au-dessus aussi, puisque le diamant aztèque a un nombre pair de cases. Ce que l'on a fait avec le ruban central peut être refait avec les autres rubans de dominos de longueur 2*L* – 2, 2*L* – 4, etc., toujours avec un nombre pair de cases vides en dessous. Par la même occasion, le nombre de cases vides entre deux rubans successifs est aussi pair.

[5] C'est évident, puisque les extrémités d'un vecteur de domino sont toujours à une distance verticale 1/2 de la bordure du domino.

couverte du diamant aztèque ne peut être remplie que par des dominos horizontaux. Le pavage ainsi obtenu est unique.

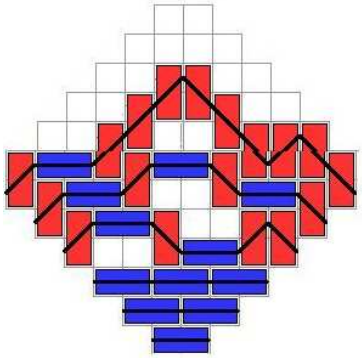
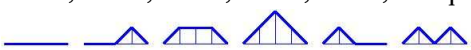


*Figure 3* : Les chaînes de montagne et les rubans de dominos correspondants. La zone non couverte, en blanc, ne peut être remplie que par des dominos horizontaux

Finalement, il y a bijection entre un pavage et ces *L* chaînes de montagne qui ne se touchent ni ne se croisent. Il suffit alors de compter le nombre de façons de construire ces *L* chaînes pour avoir le nombre de pavages possibles et aboutir à la formule $2^{L(L+1)/2}$. C'est cette méthode élégante qu'ont utilisé Sen-Peng Eu et Tung-Shan Fu [EU2005] pour démontrer la formule, en se servant des travaux de I. Gessel et G. Viennot [GES1985].

Dans ce qui suit, nous allons voir comment passer d'un pavage donné à la famille des *L* chaînes de montagnes qui le caractérise, et inversement comment passer des chaînes de montagnes au pavage correspondant. Cela sera fait en plusieurs étapes.

## 2. Chaînes de montagnes et mots de Schroeder

Une chaîne de montagnes de longueur $N = 2L$ peut être écrite sous forme d'un mot de même longueur à base de trois lettres. Notons 0 un pas horizontal de longueur unité (les paliers horizontaux s'écrivant 00), 1 un pas diagonal vers le haut, et 2 un pas diagonal vers le bas. Dans ce contexte, les mots correspondant à toutes les chaînes de montagnes de longueur *N* peuvent être écrits dans l'ordre alphabétique, les trois lettres 0, 1, 2 étant prises dans cet ordre. Ainsi les mots de longueur 4 sont, dans l'ordre : 0000, 0012, 1002, 1122, 1200, 1212, ce qui correspond à cette succession de chaînes de montagnes : 

Le programme qui permet de construire ces mots de longueur donnée *N* est donné en annexe, à la fin de ce document. Chaque mot est placé à tour de rôle dans un tableau *a*[*N*], à partir du mot 000…000, et jusqu'au dernier mot 1212…1212. Un exemple de résultat du programme est indiqué sur la *figure 4*.

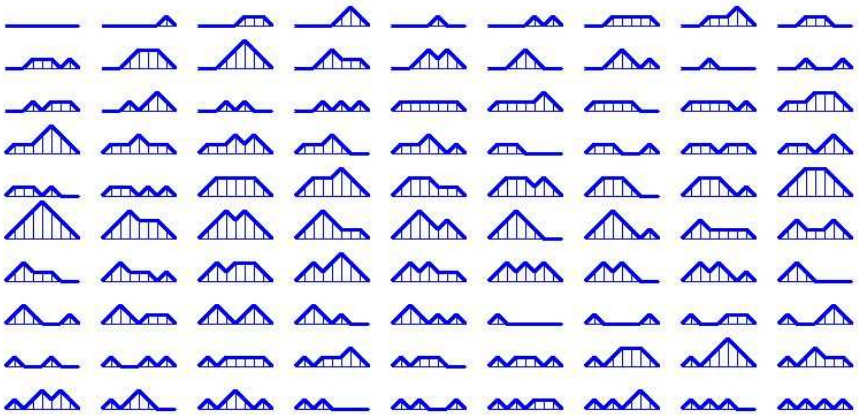


*Figure 4* : Les 90 chaînes de montagnes de longueur 8

## 3. Mots de Schroeder aléatoires

Rappelons que le nombre $u(n)$ de mots de Schroeder, avec $n$ désignant la demi-longueur du diamant aztèque (appelée $L$ auparavant), obéit à la relation de récurrence :

$$u(n) = u(n-1) + \sum_{k=1}^{n-1} u(k) - u(n-1-k) \text{ avec } u(1) = 1.$$

Grâce à ce calcul, et en numérotant les mots de Schroeder obtenus par énumération, il est possible de prendre un numéro $h$ au hasard, entre 0 et $u(L) - 1$, pour obtenir un mot aléatoire. Il suffit alors de procéder à l'énumération des mots de Schroeder jusqu'à ce numéro $h$.

Mais on sait que le mot moyen, correspondant à l'équiprobabilité des mots, comme on l'a vu pour les chaînes de montagnes sans palier ([AUD2009]) est une chaîne désespérément plate. Cela ne correspond pas à ce que doit notamment être le mot correspondant au ruban central des dominos du diamant aztèque, si l'on veut un pavage au hasard. En effet, un pavage aléatoire équiprobable doit avoir un nombre analogue de dominos de même orientation (verticaux à ouest et à l'est, horizontaux au nord et au sud) aux quatre coins du diamant aztèque, cela pour des raisons de symétrie évidentes. Avec un ruban central de dominos assez plat, ayant un nombre limité de dominos verticaux, le nombre de dominos horizontaux situés au-dessus de lui est trop élevé.

Aussi a-t-on intérêt choisir une énumération des chaînes de montagnes qui commencent par des montées, ce qui correspond à la deuxième méthode d'énumération exposée en annexe (*figure 20*). Il suffit alors de choisir le nombre $h$ dans la zone située au début de l'énumération, au moins pour le ruban central. [6] On aboutit ainsi au programme suivant, avec la fonction *schroederhasard*(*LL*) avec *LL* = *L* pour le ruban central, et une valeur inférieure pour les autres chaînes.

```
void schroederhasard(int LL)
{ int i, j,pospivot,nb1,flag, count=0;
  int n; int cumul,u[100],h;
  u[0]=1;       /* calcul préliminaire du nombre u(LL)de mots de Schroeder de longueur 2 LL */
  for(n=1;n<=LL;n++)
   { cumul=0; for(i=0;i<n;i++)  cumul+=u[i]*u[n-1-i];          cumul+=u[n-1];
     u[n]=cumul;
   }
  if (L==LL) h=rand()%(u[LL]/6);  /* choix de h parmi les premiers mots pour le ruban central */
  else if (L-1==LL) h=rand()%(u[LL]/2);
  else h=rand()%(u[LL]);

  for(i=0;i<2*LL;i++) if (i<LL) a[i]=0;else a[i]=1;  /* premier mot */
  for(;;)    /* les mots successifs sont placés à tour de rôle dans le tableau a[] */
    { if (count==h) break; count++;  /* arrêt lorsque l'on arrive au mot numéro h */
      flag=0;
      for(i=0;i<2*LL;i++) if (! (a[i]==2 )) { flag=1; break;}
      if (flag==0) break;
      i=2*LL-1;nb1=0;
          while(a[i-1]>=a[i]) { if (a[i]==1) nb1++;  i--;       }
      if (a[i]==1 && a[i-1]==0 && nb1>=1)
        { pospivot=i-1;  a[pospivot]=1;a[pospivot+1]=0;
          for(j=2*LL-1;j>=pospivot+2;j--) if (j>2*LL-nb1-1) a[j]=1; else a[j]=0;
```

[6] Dans le programme qui suit, nous avons notamment mis la contrainte sur le ruban central :
*if (L==LL) h=rand()%(u[LL]/6);*
Cette méthode marche pour une valeur de $L$ de l'ordre de 7 ou 8. Elle a l'avantage de limiter le nombre d'essais à faire pour trouver le ruban central. Pour de plus fortes valeurs de $L$, le nombre de chaînes de montagnes augmente très vite, il atteint déjà 1 037 718 pour $L$ = 10, et l'on doit réaménager le programme. On devra ajouter des contraintes au ruban central, en lui imposant de commencer par des montées et de finir par des descentes, et de mettre des contraintes analogues aux rubans suivants. En quelque sorte, on provoque le phénomène de cercle arctique.

```
          }
       else if (a[i]==1 && a[i-1]==0 && nb1==0)
         { pospivot=i-1;  a[pospivot]=2; a[pospivot+1]=2;
         }
       else if (a[i]==2 && a[i-1]==0 && nb1>1)
         { pospivot=i-1;  a[pospivot]=1; a[pospivot+1]=0;
           a[pospivot+2]=0;
           for(j=2*LL-nb1-1;j>=pospivot+3;j--) if (j>=(pospivot+3+2*LL-nb1)/2) a[j]=1; else a[j]=0;
         }
       else if (a[i]==2 && a[i-1]==0 && nb1==1)
         { pospivot=i-1;  a[pospivot]=2; a[pospivot+1]=2;
           for(j=2*LL-1;j>=pospivot+2;j--) if (  j>(2*LL+pospivot)/2) a[j]=1; else a[j]=0;
         }
       else if (a[i]==2 && a[i-1]==1)
         { pospivot=i-1;  a[pospivot]=2; a[pospivot+1]=2;
           for(j=2*LL-1;j>=pospivot+2;j--)  if (j>=(2*LL-nb1+pospivot+1)/2) a[j]=1; else a[j]=0;
         }
     }
   compteurV=0;  /* calcul du nombre de montées et descentes de la chaîne de montagnes */
   for(i=0;i<2*LL;i++) if (a[i]==0 || a[i]==1) compteurV++;
}
```

Muni de cette fonction, nous allons donner une première méthode permettant de construire un pavage aléatoire du diamant aztèque.

## 4. Pavage aléatoire du diamant aztèque

Le programme qui permet de construire un pavage aléatoire se déroule en plusieurs étapes, en faisant intervenir une succession de fonctions :

```
dessindiamantazteque();
do
   { SDL_Flip(screen) ; SDL_FillRect(screen,0,white);  /* on efface tout */
     compteurVV=0;
     rubancentral();
     remplissagehaut();
    compteurVV=compteurV;
    for(i=1;i<L;i++)
       { do
           { schroederhasard(L-i);chaine(L-i,i); fini=1;
             for(j=0;j<2*(L-i);j++)
             if (yc[i][j]>yc[i-1][j+1]) {fini=0; break;}
           }
         while (fini==0);
         compteurVV+=compteurV;   /* compteurV est le nombre de dominos verticaux du ruban central */
         dessinchaine(L-i,i); dessinruban(L-i,i);
       }
     remplissagefinal();
   }
    while (compteurVV<=L*(L+1)/2-3);  /* compteurVV est le nombre de dominaux verticaux */
    SDL_Flip(screen);pause();
```

* La fonction *dessindiamantazteque*() détermine les coordonnées des points de la bordure du diamant aztèque de demi-longueur *L* donnée. Puis elle le dessine.

* Puis on entre dans une boucle où des pavages sont construits au hasard jusqu'à ce que le nombre de dominos verticaux, noté *VV*, soit proche de celui des dominos horizontaux.

* La fonction *rubancentral()* construit la succession centrale des dominos.

* La fonction *remplissagehaut()* place des dominos horizontaux partout au-dessus du ruban central.

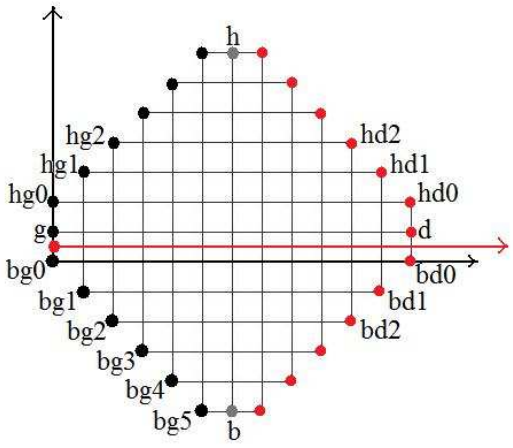
* Une boucle *for()* va ensuite construire les chaînes de montagnes autres que celle du ruban central, et dessiner les successions de dominos correspondantes. Lors du choix au hasard de ces chaînes, plusieurs essais sont effectués jusqu'à ce que la chaîne en construction ne traverse pas la chaîne précédemment construite.

* La fonction *remplissagefinal*() remplit de dominos horizontaux les parties restées vides du diamant aztèque.

Enfin la grande boucle du programme provoque des essais de pavages jusqu'à ce que le nombre de dominos verticaux (*compteurVV*) soit proche de celle des dominos horizontaux.

Nous allons maintenant entrer dans le détail de ces fonctions.

### 4.1. Construction du diamant aztèque




Les points de la bordure du diamant aztèque d'ordre $L$ sont notés comme indiqué sur le dessin ci-contre pour $L = 6$. On commence par utiliser le repère en noir d'origine *bg0*, pour obtenir les coordonnées de ces points, chaque petit carré du diamant aztèque ayant une longueur unité. Comme on le verra plus tard les points les plus importants sont les points *bg*(*i*) de la bordure en bas à gauche, avec *i* allant de 0 à $L - 1$. Puis on pratique une translation verticale de l'axe des *x* de façon que l'origine du repère soit au milieu du segment [*bg0 g*], d'où le nouvel axe horizontal en rouge. On passe enfin aux coordonnées écran en se donnant une origine (*xorig*, *yorig*) et un *zoom*. Il suffit de joindre les points obtenus pour avoir le quadrillage du diamant aztèque. On en déduit le programme de construction (les coordonnées ayant été déclarées en global).

```
void dessindiamantazteque(void)
   {int i;
    xorig=10; yorig=450;
    for(i=0;i<L;i++)  { xbg[i]=i;    ybg[i]=-i;  xhg[i]=i; yhg[i]=2+i;}
    for(i=0;i<L;i++) line(xorig+zoom*xbg[i],yorig-zoom*ybg[i]+zoom/2,
                          xorig+zoom*xhg[i],yorig-zoom*yhg[i]+zoom/2,gray);
     xb=L;yb=ybg[L-1]; xh=L;yh=yhg[L-1];
     line(xorig+zoom*xb,yorig-zoom*yb+zoom/2,  xorig+zoom*xh,yorig-zoom*yh+zoom/2,gray);
     for(i=0;i<L;i++)  { xbd[i]=2*L-i;    ybd[i]=-i;  xhd[i]=2*L-i; yhd[i]=2+i;}
     for(i=0;i<L;i++) line(xorig+zoom*xbd[i],yorig-zoom*ybd[i]+zoom/2,
                          xorig+zoom*xhd[i],yorig-zoom*yhd[i]+zoom/2,gray);
     for(i=0;i<L;i++) line(xorig+zoom*xbg[i],yorig-zoom*ybg[i]+zoom/2,
                          xorig+zoom*xbd[i],yorig-zoom*ybd[i]+zoom/2,gray);
     for(i=0;i<L;i++) line(xorig+zoom*xhg[i],yorig-zoom*yhg[i]+zoom/2,
                          xorig+zoom*xhd[i],yorig-zoom*yhd[i]+zoom/2,gray);
     xg=0;yg=1; xd=2*L;yd=1;
     line(xorig+zoom*xg,yorig-zoom*yg+zoom/2,  xorig+zoom*xd,yorig-zoom*yd+zoom/2,gray);
   }
```

### 4.2. Famille de chaînes de montagnes

Rappelons qu'un pavage du diamant aztèque est caractérisé par L chaînes de montagnes, dont la longueur varie de 2L à 2, leur origine étant à chaque fois décalée d'une unité en abscisse ainsi qu'en ordonnée, avec comme condition que les chaînes ne se touchent ni ne se traversent. Pour gagner de la place en vue de leur dessin sur l'écran, nous allons leur donner à toutes la même ordonnée à l'origine, en faisant seulement varier leur abscisse au départ. Dans ces conditions, les chaînes peuvent maintenant se toucher mais pas se traverser (*figure 5*).

Pour une chaîne de longueur *LL*, avec *LL* variant de *L* à 1, avec son origine décalée d'un nombre *k* par rapport à la première (notons que $k = L - LL$), le mot associé a été enregistré dans le tableau *a*[] grâce à la fonction précédemment vue *schroederhasard*(). Il suffit alors de déterminer les coordonnées (*xs*, *yc*) des extrémités de chaque pas de la chaîne numéro *k* selon les valeurs 0, 1 ou 2 du tableau *a*[]. D'où la fonction *chaine()* :

```
void chaine(int LL, int k)
{ int i;
  xs[k][0]=k;yc[k][0]=0;    /* coordonnées du point initial 0 de la chaîne numéro k*/
  for(i=1;i<=2*LL;i++)
   { if (a[i-1]==2) {  xs[k][i]=xs[k][i-1]+1; yc[k][i]=yc[k][i-1];}
     else if(a[i-1]==0) { xs[k][i]=xs[k][i-1]+1; yc[k][i]=yc[k][i-1]+1; }
     else if(a[i-1]==1) { xs[k][i]=xs[k][i-1]+1; yc[k][i]=yc[k][i-1]-1; }
   }
}
```

On en déduit le dessin des chaînes sur l'écran grâce à la fonction *dessinchaine*() :

```
void dessinchaine(int LL, int k)
{ int i;
  xxorig=xorig; yyorig=100+4*k;
  for(i=1;i<=2*LL;i++)
  linewithwidth(xxorig+zoom*xs[k][i-1],yyorig-zoom*yc[k][i-1],
                xxorig+zoom*xs[k][i],yyorig-zoom*yc[k][i],1,black);
  yyorig=100;
}
```

Dans le dessin, on a introduit une légère variation verticale des origines des chaînes de montagnes, de façon à mieux les voir.

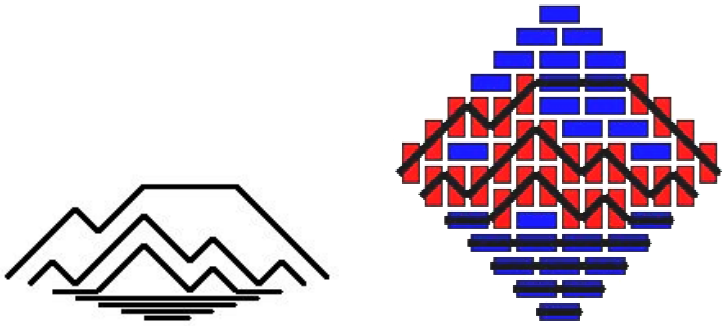


*Figure 5* : A gauche, les chaînes qui peuvent se toucher mais pas se traverser (coordonnées *xs*, *yc*), à droite les mêmes chaînes dans le diamant aztèque (coordonnées *xs*, *ys*)

C'est dans le programme principal (voir le début du *paragraphe 4*) que les tests sont effectués, grâce aux coordonnées des points (*xs*, *yc*), pour qu'une chaîne nouvellement construite ne traverse pas la chaîne précédemment construite. Cela explique aussi que la première chaîne, celle du ruban central, qui n'a aucun prédécesseur, doive être construite à part, avant toutes les autres.

Il s'agit maintenant de remettre ces chaînes dans le diamant aztèque et de les remplacer par un pavage de dominos.

### 4.3. Rubans de dominos associés aux chaînes

On passe des coordonnées (*xs*, *yc*) des chaînes de montagnes dessinées à part, à celles (*xs*, *ys*) de ces chaînes placées dans le diamant aztèque. Puis en suivant leur cheminement, on dessine les dominos qui leur correspondent, ce que fait la fonction *dessinruban*() :

```
void dessinruban(int LL, int k)
  { int i;
  for(i=0;i<=2*LL;i++) {ys[k][i]=yc[k][i]-k;}
     for(i=1;i<=2*LL;i++)
     { if (a[i-1]==2) {rectangle(xorig+zoom*xs[k][i-1]+2,yorig-zoom*ys[k][i-1]-zoom/2+2,
                                 xorig+zoom*xs[k][i+1]-2,yorig-zoom*ys[k][i+1]+zoom/2-2, blue,black);
                       i++;
                      }
       else if (a[i-1]==0) rectangle(xorig+zoom*xs[k][i-1]+2,yorig-zoom*ys[k][i-1]+zoom/2-2,
                                 xorig+zoom*xs[k][i]-2,yorig-zoom*ys[k][i]-zoom/2+2, red,black);
       else if (a[i-1]==1) rectangle(xorig+zoom*xs[k][i-1]+2,yorig-zoom*ys[k][i-1]-zoom/2+2,
                                 xorig+zoom*xs[k][i]-2,yorig-zoom*ys[k][i]+zoom/2-2, red,black);
     }
 }

void rectangle(int x1,int y1, int x2, int y2, Uint32 cr,Uint32 c)
{ line(x1,y1,x2,y1,c);line(x1,y2,x2,y2,c);line(x1,y1,x1,y2,c);line(x2,y2,x2,y1,c);
  floodfill((x1+x2)/2,(y1+y2)/2,cr,c);
}
```

### 4.4. Le ruban central de dominos

Il s'agit maintenant d'utiliser les fonctions précédentes pour tracer en premier lieu le ruban central de domino,

```
void rubancentral(void)
{ do  schroederhasard(L);
   while (compteurV<=L);   /* on impose d'avoir au moins une moitié de dominos verticaux */
   chaine(L,0); dessinchaine(L,0); dessinruban(L,0);
}
```

### 4.5. Remplissage au-dessus du ruban central

On parcourt chaque ligne horizontale de la partie haute du diamant aztèque, à partir de la bordure en haut à gauche, les lignes successives étant prises de bas en haut. A chaque fois, on considère les points comme indiqué sur la *figure 6*, de coordonnées ($xhg(i) + j$, $yhg(i) + 1/2$) avant le zoom sur l'écran. Puis on teste si les points décalés de (1/2, - 1/2) sont en blanc tandis que ceux décalés de (1/2, - 3/2) ne le sont pas - cela pour éviter d'être au dessous du ruban central, auquel cas on place un domino horizontal puisque la place est libre.

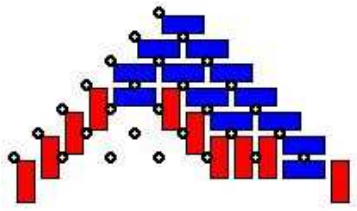


*Figure 6* : Parcours de la moitié haute du diamant aztèque, suivant les points marqués par des cercles

```
void remplissagehaut(void)
   { int i,xx,j;
   for(i=0;i<L;i++)
    { xx=xhg[i];
      for(j=0;j<2*L-2*i;j+=2)
         { if (getpixel(xorig+zoom*(xx+j)+zoom/2,yorig-zoom*yhg[i]+zoom/2+zoom/2)==white
           && getpixel(xorig+zoom*(xx+j)+zoom/2,yorig-zoom*yhg[i]+zoom/2+zoom/2+zoom)!=white)
              { rectangle( xorig+zoom*(xx+j)+2,yorig-zoom*yhg[i]+zoom/2+2,
                xorig+zoom*(xx+j+2)-2,yorig-zoom*yhg[i]+zoom+zoom/2-2, blue,black);
              }
          }
      }
   }
```

### 4.6. Remplissage final

Une fois tous les rubans de dominos dessinés à partir des chaînes de montagnes, il reste des zones vides à remplir par des dominos horizontaux. On procède comme on vient de le faire pour le remplissage du haut, en parcourant chaque ligne horizontale du diamant aztèque, mais on se contente de tester s'il existe une place vide (en blanc).

```
void remplissagefinal(void)
{ int i,xx,j;
   for(i=0;i<L-1;i++)   /* parcours de la partie basse du diamant aztèque */
    { xx=xbg[i];
      for(j=0;j<2*L-2*i;j+=1)
      if (getpixel(xorig+zoom*(xx+j)+zoom/2,yorig-zoom*ybg[i]+zoom/2-zoom/2)==white)
       { rectangle( xorig+zoom*(xx+j)+2,yorig-zoom*ybg[i]+zoom/2-2,
                 xorig+zoom*(xx+j+2)-2,yorig-zoom*ybg[i]-zoom+zoom/2+2, blue,black);
       }
    }
 for(i=0;i<L;i++)   /* parcours de la partie haute */
    { xx=xhg[i];
      for(j=0;j<2*L-2*i;j+=2)
        {if (getpixel(xorig+zoom*(xx+j)+zoom/2,yorig-zoom*yhg[i]+zoom/2+zoom/2)==white)
              rectangle( xorig+zoom*(xx+j)+2,yorig-zoom*yhg[i]+zoom/2+2,
                 xorig+zoom*(xx+j+2)-2,yorig-zoom*yhg[i]+zoom+zoom/2-2, blue,black);
        }
     }
}
```

## 5. Du désordre à l'ordre

A partir d'un pavage quelconque du diamant aztèque, comme on vient de le faire, nous allons voir comment revenir à un pavage où tous les dominos sont horizontaux, avec toutes les chaînes de montagnes aplanies à l'altitude 0. Plaçons-nous dan le contexte où les chaînes démarrent toutes au même niveau et peuvent se toucher sans se traverser (points (*xs*, *yc*)). Il existe deux façons d'aplanir progressivement les chaînes de montagnes [AUD2009] :

1) Toute pointe d'une chaîne peut être remplacée par un palier horizontal, sans jamais traverser la chaîne située en dessous (*figure 7*).

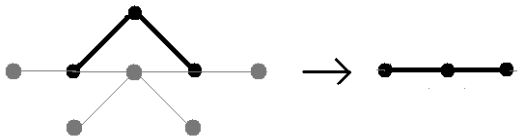


*Figure 7* : Une pointe (en noir) devient un palier. La chaîne située au-dessous (plusieurs cas sont dessinés en gris) n'interdit jamais cette transformation.

2) Tout palier peut être transformé en une crevasse (une pointe dirigée vers le bas) pourvu que celle-ci ne traverse pas la chaîne située au-dessous (*figure 8*).

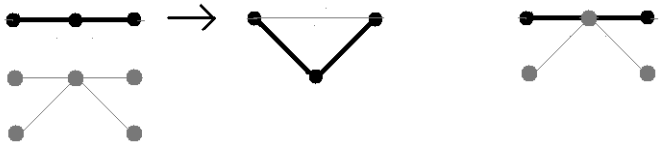


*Figure 8* : A gauche un palier transformé en crevasse, et à droite un cas où la modification est impossible.

Ces deux types de modifications sont appliquées en succession : on commence par aplanir les pointes de toutes les chaînes de montagnes concernées, puis on crée des crevasses là où c'est possible, puis on recommence ces deux modifications, et ainsi de suite jusqu'à avoir toutes les chaînes aplanies (*figure 9*).

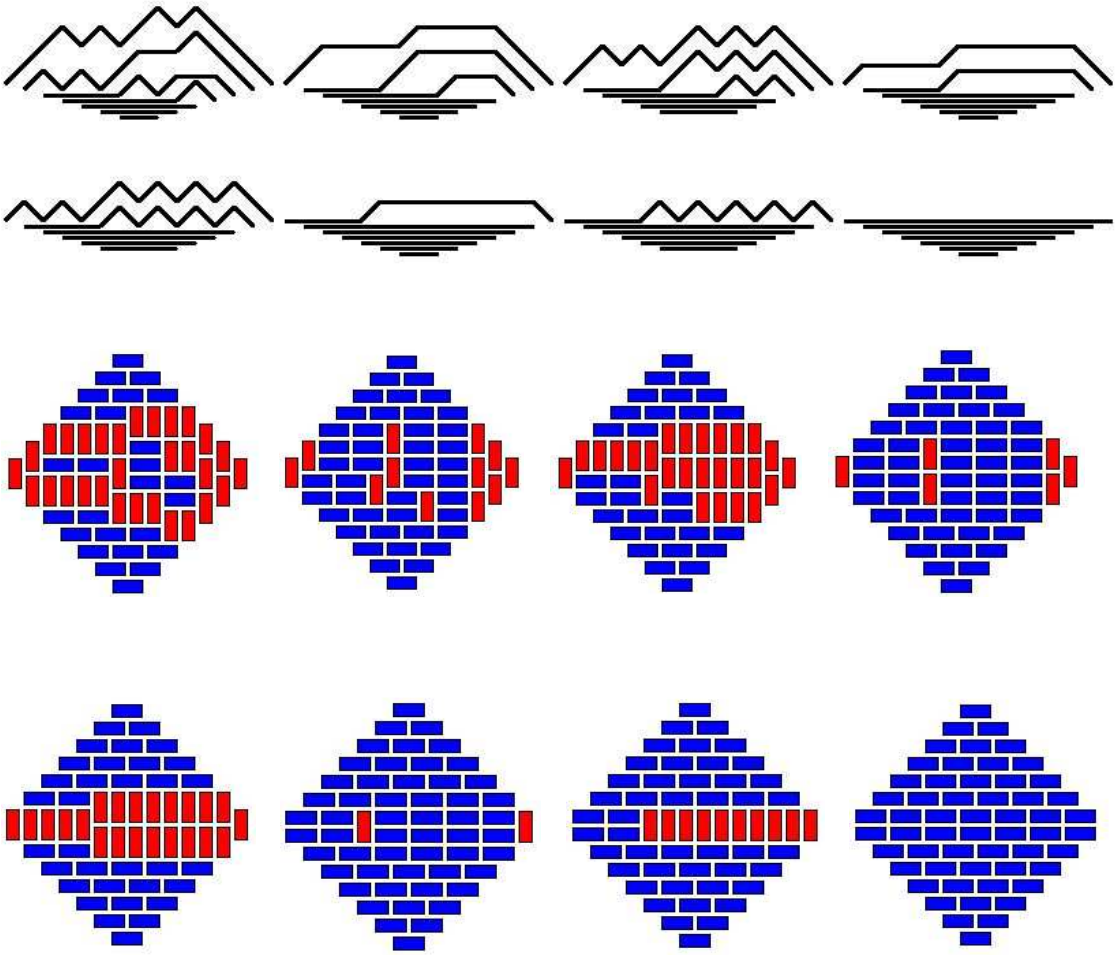


*Figure 9* : En haut, l'évolution des chaînes de montagnes (repère *xc*, *yc*), en bas l'évolution correspondante du pavage, jusqu'au tout horizontal

Le programme en découle.

```
void remiseaplat(void)
{int i,j,k,LL,fin,debut[L],q,ii,jj;
 do
  { fin=1;
    xxorig+=205; if (xxorig>650) { yyorig+=100; xxorig=xorig;}
    /********** premier type de modifications **********/
    for(k=0;k<L;k++)  /* on prend chaque chaîne à tour de rôle */
       { LL=L-k;
         for(i=0;i<=2*LL;i++) ys[k][i]=yc[k][i]-k;
         for(i=0;i<2*LL-1;i++)
         if (yc[k][i+1]-yc[k][i]==1 && yc[k][i+1]-yc[k][i+2]==1) /* on tombe sur un palier */
            { yc[k][i+1]=yc[k][i];
              effacerrectangle(xorig+zoom*xs[k][i],yorig-zoom*ys[k][i]+zoom/2,
                               xorig+zoom*xs[k][i+2],yorig-zoom*ys[k][i+2]-zoom/2-zoom);
              SDL_Flip(screen); SDL_Delay(50);
              rectangle(xorig+zoom*xs[k][i]+2,yorig-zoom*ys[k][i]-zoom/2+2,
                                    xorig+zoom*xs[k][i+2]-2,yorig-zoom*ys[k][i+2]+zoom/2-2, blue,black);
              rectangle(xorig+zoom*xs[k][i]+2,yorig-zoom*ys[k][i]-zoom/2-zoom+2,
                                xorig+zoom*xs[k][i+2]-2,yorig-zoom*ys[k][i+2]-zoom+zoom/2-2, blue,black);
            }
         for(i=1;i<=2*LL;i++)  {   ys[k][i]=yc[k][i]-k;  }
       }
    for(i=0;i<=2*L;i++) if (yc[L][i]>0) {fin=0; break;} /* test de fin */

    /********** deuxième type de modifications **********/
    if (fin==0)
       {for(k=0;k<L-1;k++)
          { LL=L-k;
            for(i=0;i<=2*LL;i++) ys[k][i]=yc[k][i]-k;
           q=0;
           for(i=0;i<2*LL-1;i++)
           if (yc[k][i+1]==yc[k][i] && yc[k][i+1]==yc[k][i+2]) { debut[q++]=i; i++;} /* on a un palier */
           for(j=0;j<q;j++)       /* debut[q] est le début de chaque palier de la chaîne k */
           if (yc[k][debut[j]+1]-yc[k+1][debut[j]]>=1) /* cas où la modification est possible */
             { ii=debut[j];
               yc[k][ii+1]=yc[k][ii]-1;
               effacerrectangle(xorig+zoom*xs[k][ii],yorig-zoom*ys[k][ii]-zoom/2,
                                xorig+zoom*xs[k][ii+2],yorig-zoom*ys[k][ii+2]+zoom/2+zoom);
               SDL_Flip(screen); SDL_Delay(50);
               rectangle(xorig+zoom*xs[k][ii]+2,yorig-zoom*ys[k][ii]-zoom/2+2,
                              xorig+zoom*xs[k][ii+1]-2,yorig-zoom*ys[k][ii+2]+zoom+zoom/2-2, red,black);
               rectangle(xorig+zoom*xs[k][ii+1]+2,yorig-zoom*ys[k][ii]-zoom/2+2,
                               xorig+zoom*xs[k][ii+2]-2,yorig-zoom*ys[k][ii+2]+zoom+zoom/2-2, red,black);
             }
           for(i=1;i<=2*LL;i++)  { ys[k][i]=yc[k][i]-k; }
         }
       }
   for(i=0;i<=2*L;i++) if (yc[L][i]>0) {fin=0; break;} /* test de fin */
  }
while(fin==0);
}
void effacerrectangle(int x1,int y1, int x2, int y2)
{ int i,j,aux;
   if (y1>y2) { aux=y1;y1=y2;y2=aux;}
   for(i=x1; i<=x2; i++)   for(j=y1;j<=y2; j++) putpixel(i,j,white);
}
```

## 6. Une autre façon de créer un pavage aléatoire

C'est l'occasion de passer de l'ordre au désordre. Nous partons d'un pavage tout horizontal, puis en provoquant un grand nombre de rotations au hasard dans des carrés 2 sur 2 du diamant aztèque (*figure 10*), on arrive finalement à un pavage aléatoire. Cette méthode est particulièrement simple, son écueil est de ne pas savoir quel est le nombre minimal de rotations à faire pour être assuré d'un pavage aléatoire. Sur de grands diamants aztèques, le procédé devient long. Par exemple pour un diamant aztèque d'une longueur de l'ordre de 50, on a fait 800 000 rotations.

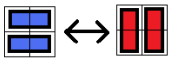


*Figure 10* : Les rotations permises dans le pavage

Le programme principal se présente ainsi :

```
dessindiamantazteque();  SDL_Flip(screen);pause();   SDL_FillRect(screen,0,white);
carres();
touthorizontal();SDL_Flip(screen);
pavagehasard(80000); SDL_Flip(screen);pause();   /* 80000 est le nombre de rotations */
chainesmontagnes();
```

* La fonction *dessindiamantazteque*() a déjà été faite précédemment (4.1.). Il convient d'effacer ce dessin avant de lancer les fonctions suivantes.

* La fonction *carres*() détermine tous les carrés 2 sur 2 à l'intérieur du diamant aztèque, et donne leur nombre.

* La fonction *touthorizontal*() construit le pavage formé de dominos tous horizontaux.

* La fonction *pavagehasard*(*q*) provoque *q* rotations au hasard dans le pavage. On obtient ainsi un pavage aléatoire.

* La fonction *chainesmontagnes*() construit les *L* chaînes de montagnes qui caractérisent le pavage, à partir du pavage. Il s'agit du procédé inverse de celui qui consiste à construire un pavage à partir des chaînes de montagnes, tel que nous l'avions fait dans le paragraphe 4. A partir de là on peut si on le désire relancer la fonction de remise à plat vue au paragraphe 5.

Voyons maintenant ces nouvelles fonctions dans le détail.

### 6.1. Détermination des carrés 2 sur 2 dans le diamant aztèque de longueur 2*L*

La fonction *carres*() calcule les coordonnées des centres de ces carrés, soit *xce*[*k*], yce[*k*] pour le carré numéro *k*, ainsi que le nombre de ces carrés *nombrecarres* qui seront disponibles ensuite pour les rotations. Les centres obtenus sont indiqués sur la *figure 11*.

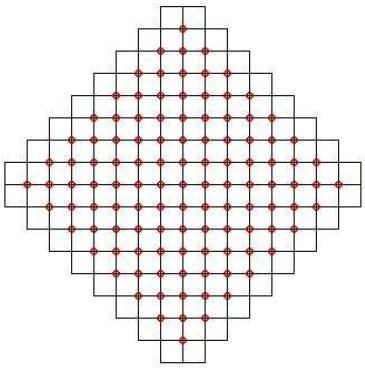


*Figure 11* : Les centres des carrés 2×2 inscrits dans le diamant aztèque

```
void carres(void)
{int i,j,k;
  k=0;
  for(j=0;j<2*L-1;j++)   /* centres des carrés sur la ligne centrale */
    { xce[k]=xg+1+j; yce[k]=yg;
      xece[k]=xorig+zoom*xce[k];yece[k]=yorig-zoom*yce[k]+zoom/2;
      cercle(xece[k],yece[k],3,red); k++;
    }
   for(i=0;i<L-1;i++) for(j=0;j<2*L-3-2*i;j++)  /* centres des carrés sur les lignes du bas */
    { xce[k]=xbg[i]+2+j; yce[k]=ybg[i];
      xece[k]=xorig+zoom*xce[k];yece[k]=yorig-zoom*yce[k]+zoom/2;
      cercle(xece[k],yece[k],3,red); k++;
    }
   for(i=0;i<L-1;i++) for(j=0;j<2*L-3-2*i;j++)  /* centres des carrés sur les lignes du haut */
    { xce[k]=xhg[i]+2+j; yce[k]=yhg[i];
      xece[k]=xorig+zoom*xce[k];yece[k]=yorig-zoom*yce[k]+zoom/2;
      cercle(xece[k],yece[k],3,red); k++;
    }
   nombrecarres=k;
}
```

## 6.2. Le pavage tout horizontal

La fonction *touthorizontal*() utilise, pour gagner du temps, des fonctions déjà construites auparavant (*chaine*(), *dessinruban*(),*remplissagehaut*()). Elle commence par dessiner la ligne centrale des dominos horizontaux. Puis elle remplit ce qui est au-dessus, et enfin ce qui est au-dessous.

```
void touthorizontal(void)
{ int i;
  for(i=0;i<2*L;i++) a[i]=2;  /* les mots de Schroeder sont mis à 2 (horizontal)*/
  chaine(L,0); dessinruban(L,0);  /* ruban central des dominos */
  remplissagehaut();
  for(i=1;i<L;i++)  /* pavage tout horizontal au-dessous du ruban central */
    {chaine(L-i,i);   dessinruban(L-i,i);}
}
```

## 6.3. Pavage au hasard

La fonction *pavagehasard(q*) provoque *q* rotations de carrés 2×2, deux dominos horizontaux devenant verticaux ou vice versa. Encore faut-il savoir quels sont les carrés que l'on peut faire tourner, à savoir ceux qui contiennent exactement deux dominos. C'est le rôle de la fonction *carrestournants*(). Celle-ci fait un parcours de tous les carrés du diamant aztèque, au nombre de *nombrecarres*, et détermine parmi eux ceux qui peuvent être tournés, au nombre de *count*, en enregistrant leur numéro dans le tableau *ct*[]. Cela étant fait, la fonction *rotation*() se charge de choisir un de ces carrés au

hasard, de numéro *hc*, en évitant que l'on utilise deux fois de suite ce numéro, pour éviter de ne rien faire. Puis elle transforme les deux dominos horizontaux en dominos verticaux, ou l'inverse.

```
void pavagehasard(int q)
{ int i;
   oldhc=-1;
   for(i=0;i<q;i++)
     { carrestournants( );
       rotation();
     }
 }

void carrestournants(void)
{ int i;
  count=0;
  for(i=0;i<nombrecarres;i++)  /* test pour savoir si l'on a deux dominos horizontaux (bleus) dans le carré */
  if (getpixel(xece[i]+zoom/2,yece[i])==white &&
      getpixel(xece[i]-zoom/2,yece[i])==white &&
      getpixel(xece[i],yece[i]+zoom/2)==blue &&
      getpixel(xece[i],yece[i]-zoom/2)==blue)
         { ct[count]=i; count++;}
   for(i=0;i<nombrecarres;i++)  /* test pour savoir si l'on a deux dominos verticaux (rouges) dans le carré */
   if (getpixel(xece[i]+zoom/2,yece[i])==red &&
      getpixel(xece[i]-zoom/2,yece[i])==red &&
      getpixel(xece[i],yece[i]+zoom/2)==white &&
      getpixel(xece[i],yece[i]-zoom/2)==white)
         { ct[count]=i; count++;}
}

void rotation(void)
 { int i;
   do { hc=rand()%count; }  /* tirage au hasard d'un carré qui peut tourner */
   while (hc==oldhc); /* en évitant de faire tourner deux fois de suite le même carré */
   nhc=ct[hc];
   if (getpixel(xece[nhc],yece[nhc]-zoom/2)==blue) /* on passe de bleu à rouge */
     { effacerrectangle(xece[nhc]-zoom,yece[nhc]-zoom,xece[nhc]+zoom,yece[nhc]+zoom);
       rectangle(xece[nhc]-zoom+2,yece[nhc]-zoom+2, xece[nhc]-2,yece[nhc]+zoom-2,red,black);
       rectangle(xece[nhc]+zoom-2,yece[nhc]+zoom-2, xece[nhc]+2,yece[nhc]-zoom+2,red,black);
     }
  else    /* on passe de rouge (vertical) à bleu (horizontal) */
    { effacerrectangle(xece[nhc]-zoom,yece[nhc]-zoom,xece[nhc]+zoom,yece[nhc]+zoom);
      rectangle(xece[nhc]-zoom+2,yece[nhc]-zoom+2, xece[nhc]+zoom-2,yece[nhc]-2,blue,black);
      rectangle(xece[nhc]-zoom+2,yece[nhc]+2, xece[nhc]+zoom-2,yece[nhc]+zoom-2,blue,black);
    }
  oldhc=hc;
}
```

Un résultat du programme est donné sur la *figure 12*.

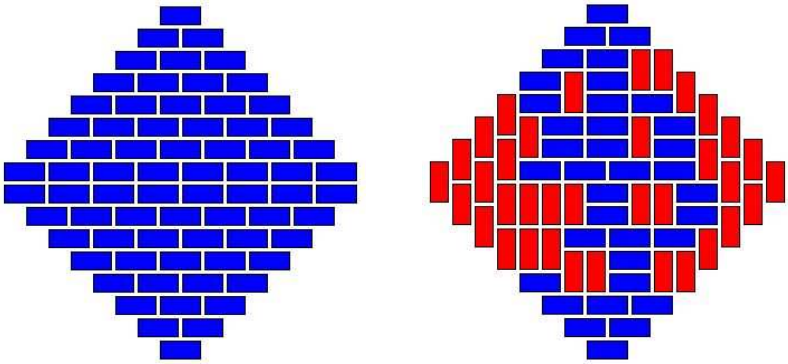


*Figure 12* : Passage de l'ordre au désordre sur un diamant aztèque avec $L$ = 8, après 80 000 rotations au hasard

### 6.4. Fabrication des chaînes de montagnes à partir du pavage aléatoire

On part de chaque point de la bordure sud-ouest du diamant aztèque. Puis on parcourt de gauche à droite les dominos qui se succèdent, en prenant comme point courant (*xei*, *yei*). Si l'on tombe sur un domino horizontal bleu, on enregistre deux 0 dans le tableau *a*[] qui va donner le mot de Schroeder associé. Si l'on tombe sur un domino rouge vers le haut, on place un 1 dans *a*[], et s'il est vers le bas on met 2 dans *a*[]. A la fin, il suffit d'appliquer les fonctions *chaine*() et *dessinchaine*() vues au *paragraphe 4.2.* afin de dessiner, hors du diamant aztèque, les chaînes qui peuvent se toucher mas pas se traverser. Par la même occasion, on dessine aussi les chaînes à l'intérieur du diamant aztèque, là où elles ne peuvent ni se toucher ni se traverser.

```
void chainesmontagnes(void)
{ int k,i, xei,yei;
  for(k=0;k<L;k++)
    { xei=xorig+zoom*xbg[k]+zoom/2; yei=yorig-zoom*ybg[k]; i=0;
      do
        {
          if (getpixel(xei,yei)==blue)  /* si l'on tombe sur un domino horizontal */
            { linewithwidth(xei-zoom/2,yei, xei+zoom+zoom/2,yei,1,black); /* dessin dans le diamant */
              xei+=2*zoom; a[i]=2;a[i+1]=2; i++;  /* mot de Schroeder dans a[] */
            }
          else if (getpixel(xei,yei)==red && getpixel(xei,yei+zoom/2)==white) /* domino vertical haut */
            {  linewithwidth(xei-zoom/2,yei, xei+zoom/2,yei-zoom,1,black);
               xei+=zoom; yei-=zoom; a[i]=0;
            }
          else if (getpixel(xei,yei)==red && getpixel(xei,yei-zoom/2)==white)  /* domino vertical bas */
            {  linewithwidth(xei-zoom/2,yei, xei+zoom/2,yei+zoom,1,black);
               xei+=zoom;yei+=zoom; a[i]=1;
            }
          i++;
        }
      while( getpixel(xei,yei)!=white);
      chaine(L-k,k);dessinchaine(L-k,k); /* chaîne dessinée en dehors du diamant aztèque */
    }
}
```

Des résultats sont donnés sur la *figure 13*.

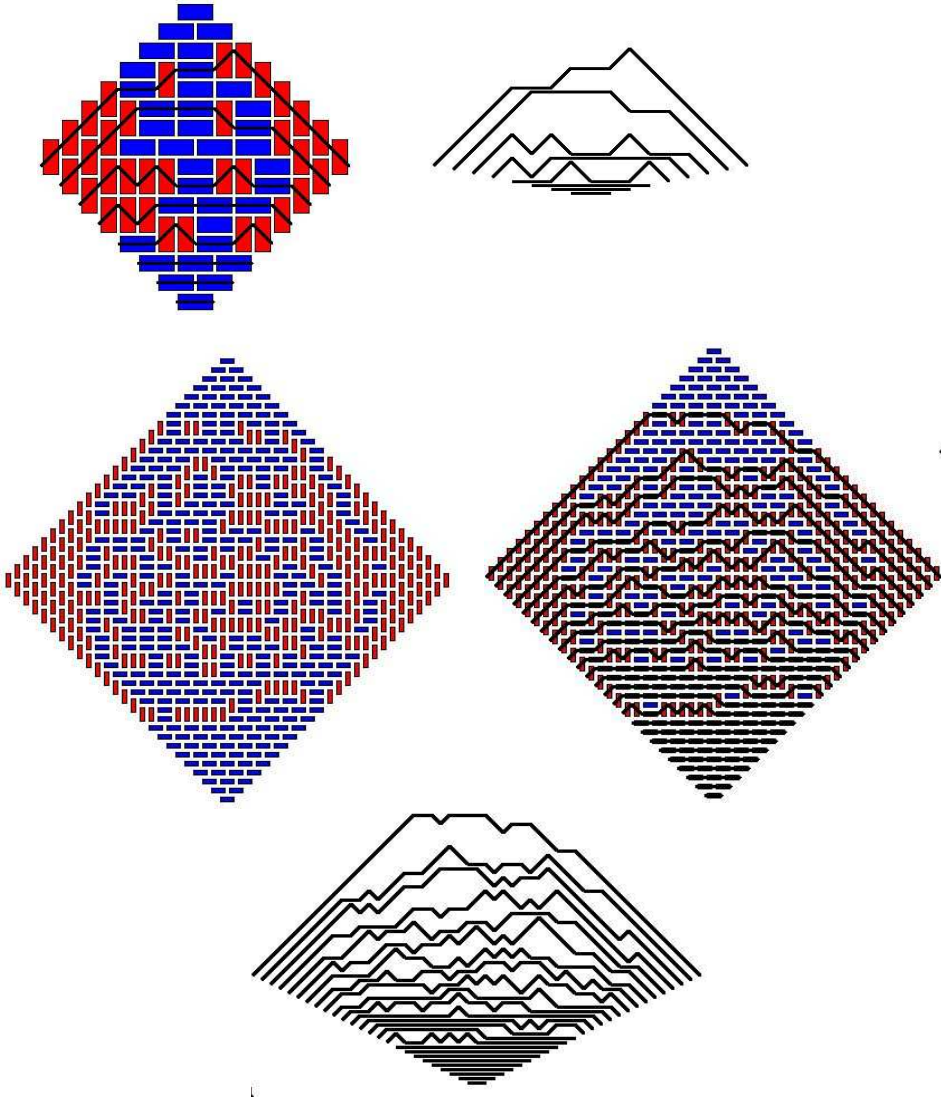


*Figure 13* : Pavages aléatoire pour $L = 8$ en haut, et $L = 25$ en bas, avec leurs chaînes de montagnes associées

# *Annexe : mots de Schroeder, et famille de chaînes de montagnes*

## 1. Première méthode

Nous nous contentons ici de redonner le programme d'énumération des mots de Schroeder de longueur $N$ ($N = 2L$) faite dans [AUD2009], telle qu'elle est esquissée dans le *paragraphe 2* précédent.

```
xorig=10; yorig=pas*N/2+5;
  for(i=0;i<N;i++) a[i]=0;
  for(;;)
   { drawing();  count++;
     flag=0;
     for(i=0;i<N;i+=2) if (! (a[i]==1 && a[i+1]==2)) { flag=1; break;}
     if (flag==0) break;
      i=N-1;
     while (! ((a[i]==0 && a[i-1]==0 && i>=1)
         || (a[i]==2 && a[i-1]==2 && a[i-2]==1 && i>=2))) i--;
```

```
    if (a[i]==0 && a[i-1]==0)
      { pospivot=i-1;  a[pospivot]=1;
        nb1=0; nb2=0;
        for(j=N-1;j>pospivot;j--)
              { if (a[j]==1) nb1++ ;
                if (a[j]==2) nb2++;
              }
        if (nb1==0)  for(j=pospivot+1;j<N;j++) a[j]=2;
        else
          { for(j=N-1;j>N-1-(nb2-nb1+1);j--) a[j]=2;
            for(k=pospivot+1;k<=j;k++) a[k]=0;
            }
       }
    else
      { pospivot=i-2;  a[pospivot]=2;
        nb1=0; nb2=0;
        for(j=N-1;j>pospivot;j--)
              { if (a[j]==1) nb1++ ;
                if (a[j]==2) nb2++;
              }
        for(j=pospivot+1;j<N;j++) a[j]=0;
        for(j=N-1;j>N-1-(nb2-nb1-2);j--) a[j]=2;
      }
    }
```

Avec la fonction dessinant les chaînes :

```
void drawing(void)
{ int x,y,i;

  x=xorig;y=yorig;
  line(xorig,yorig,xorig+N*pas,yorig,blue);
  for(i=0;i<N;i++)
    { line (x,y,x,yorig,blue);
      if (a[i]==0) {x+=pas; linewithwidth (x,y,x-pas,y,1,blue); }
      else if (a[i]==1) {x+=pas;y-=pas; linewithwidth(x,y,x-pas,y+pas,1,blue);}
      else if (a[i]==2) {x+=pas;y+=pas; linewithwidth (x,y,x-pas,y-pas,1,blue);}
    }
  xorig+=N*pas+20; if (xorig>800-N*pas) {yorig+=N*pas/2+5;xorig=10;}
  if (yorig>600-N*pas) { SDL_Flip(screen);pause();SDL_FillRect(screen,0,white);
                         xorig=10; yorig=pas*N/2;
                        }
}
```

## 2. Deuxième méthode

Nous donnons maintenant une autre méthode, en changeant l'ordre des lettres. Un pas diagonal vers le haut sera noté 0, un pas vers le bas noté 1, et un pas horizontal sera noté 2. L'ordre alphabétique dans lequel sont mis les mots de Schroeder place en premiers ceux qui commencent par des montées. Cela est mieux adapté à notre problème, où l'on doit construire au hasard le ruban central des dominos du diamant aztèque. On pourra le choisir parmi les premiers mots de Schroeder, sans avoir besoin de les chercher tous.

Dans ce contexte, le premier mot s'écrit: 00001111, ici pour $N = 8$. Comment passer d'un mot au suivant, étant entendu que chaque mot est placé à tour de rôle dans un tableau $a[N]$ ? Rappelons que lors de ce passage, un bloc fixe de lettres est maintenu à gauche, le plus long possible, et la première lettre qui change est appelée le pivot. Pour trouver le pivot, il suffit de parcourir le mot de droite à gauche jusqu'à obtenir la première descente, où une lettre est strictement inférieure à la suivante. Trois cas se présentent alors, selon qu'on tombe en premier sur 01, ou 02, ou 12.

1) On tombe sur **01** et 0 est le pivot. Le facteur 01 est alors transformé en 10 sous réserve que le nombre de 1 situés derrière le bloc 01 soit supérieur à 0. Après ce changement, on met à la suite du mot et jusqu'à la fin du mot, le sous-mot le plus petit, c'est-à-dire un bloc de 0 devant un bloc de 1, de façon qu'à la fin on retombe au niveau du sol. Sinon, lorsque le bloc 01 est déjà au niveau du sol, il est transformé en 22, ce cas ne pouvant se produire que si le bloc 01 est à la fin du mot. (*figure 14*).

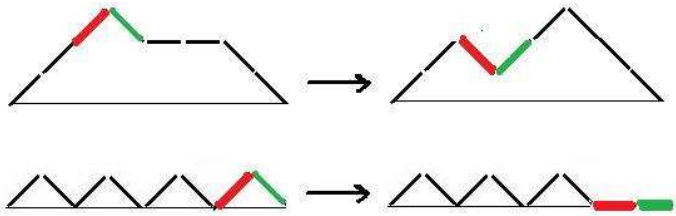


*Figure 14* : Passage d'un mot au suivant. Le parcours de droite à gauche fait tomber sur une première descente en 01, le pivot 0 étant mis en rouge. Le bloc 01 devient 10, ou 22, et l'on a le mot suivant.

2) On tombe sur **02** et 0 est le pivot. Le facteur 02 est transformé en 10 si le nombre de 1 situés après ce bloc est supérieur à 1, et sinon il devient 22. Au-delà, on met un bloc de 0 devant un bloc de 1 (*figure 15*).

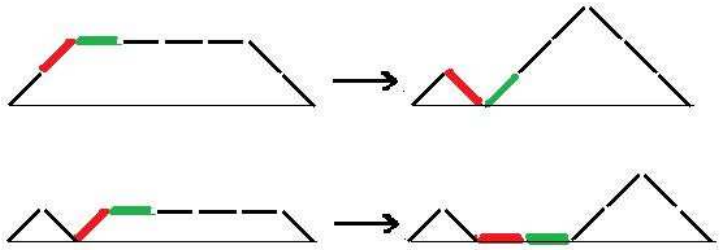


*Figure 15* : Le bloc 02 devient 10, ou 22. Dans le premier cas, le mot 00222211 devient 01000111. Dans le deuxième cas, le mot 01022221 devient 01220011.

3) On tombe sur le bloc **12**, 1 étant le pivot. Ce bloc est alors transformé en 22, avec derrière un bloc de 0 devant un bloc de 1 (*figure 16*).

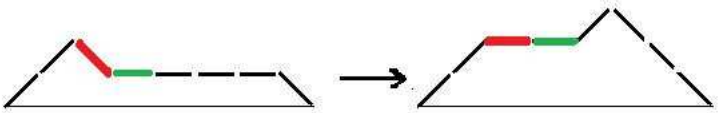


*Figure 16* : Le bloc 12 devient 22. Le mot 00122221 devient 00220111.

```
xorig=10; yorig=pas*N/2+5;
for(i=0;i<N;i++) if (i<N/2) a[i]=0;else a[i]=1; /* premier mot */
for(;;)
 { dessinchaine();  count++; /* le compteur de mots, count, est déclaré en global, donc mis à 0 */
   flag=0;
   for(i=0;i<N;i++) if (! (a[i]==2 )) { flag=1; break;}
   if (flag==0) break;  /* test d'arrêt de la boucle for, le mot final étant 2222…22 */

    i=N-1 ; nb1=0;  /* on part de la fin, à la recherché du pivot */
   while(a[i-1]>=a[i])    { if (a[i]==1) nb1++;    i--;} /* on attend la première descente */
   if (a[i]==1 && a[i-1]==0 && nb1>=1)
      { pospivot=i-1;  a[pospivot]=1;a[pospivot+1]=0;
        for(j=N-1;j>=pospivot+2;j--) if (j>N-nb1-1) a[j]=1; else a[j]=0;
      }
   else if (a[i]==1 && a[i-1]==0 && nb1==0)
      { pospivot=i-1;  a[pospivot]=2; a[pospivot+1]=2;
      }
```

```
      else if (a[i]==2 && a[i-1]==0 && nb1>1)
        { pospivot=i-1;  a[pospivot]=1; a[pospivot+1]=0;
          a[pospivot+2]=0;
          for(j=N-nb1-1;j>=pospivot+3;j--) if (j>=(pospivot+3+N-nb1)/2) a[j]=1; else a[j]=0;
        }
      else if (a[i]==2 && a[i-1]==0 && nb1==1)
        { pospivot=i-1;  a[pospivot]=2; a[pospivot+1]=2;
          for(j=N-1;j>=pospivot+2;j--) if (  j>(N+pospivot)/2) a[j]=1; else a[j]=0;
        }
      else if (a[i]==2 && a[i-1]==1)
        { pospivot=i-1;  a[pospivot]=2; a[pospivot+1]=2;
          for(j=N-1;j>=pospivot+2;j--)  if (j>=(N-nb1+pospivot+1)/2) a[j]=1; else a[j]=0;
        }
   }

void dessinchaine(void)
{ int x,y,i;
  x=xorig;y=yorig;
  line(xorig,yorig,xorig+N*pas,yorig,blue);
  for(i=0;i<N;i++)
    { line (x,y,x,yorig,blue);
      if (a[i]==2) {x+=pas; linewithwidth (x,y,x-pas,y,1,blue); }
      else if (a[i]==0) {x+=pas;y-=pas; linewithwidth(x,y,x-pas,y+pas,1,blue);}
      else if (a[i]==1) {x+=pas;y+=pas; linewithwidth (x,y,x-pas,y-pas,1,blue);}
    }
  xorig+=N*pas+20; if (xorig>800-N*pas) {yorig+=N*pas/2+5;xorig=10;}
  if (yorig>600-N*pas) { SDL_Flip(screen);pause();SDL_FillRect(screen,0,white);
                         xorig=10; yorig=pas*N/2;
                        }
}
```

Un résultat du programme est donné sur la *figure 17*.

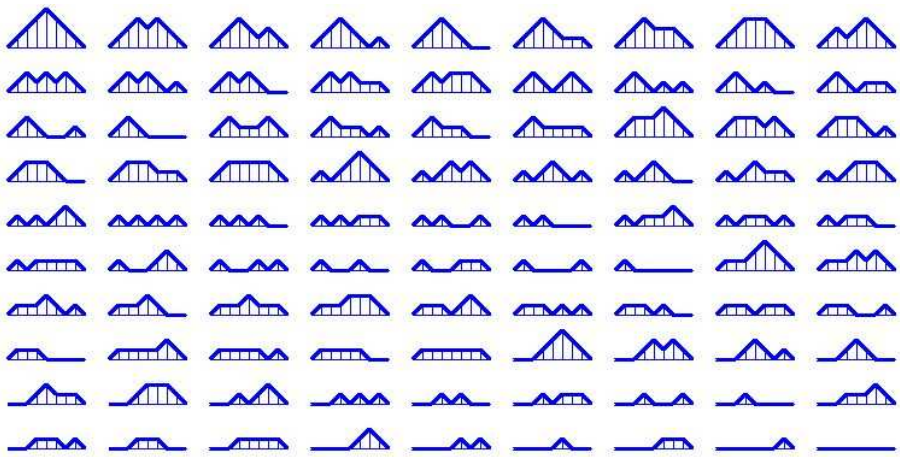


*Figure 17* : Les 90 mots de longueur 8, sous forme de chaînes de montagnes. A comparer avec la *figure 4*, où les lettres choisies ne sont pas les mêmes.

### 3. Famille de chaînes de montagnes

A titre d'exercice, nous allons maintenant construire les familles de chaînes de montagnes, avec leur demi-longueur *LL* allant de *L* à 1, démarrant toutes au niveau 0, et qui peuvent se toucher sans se traverser (*figure 18*). Ce sont justement celles qui correspondent aux pavages du diamant aztèque.

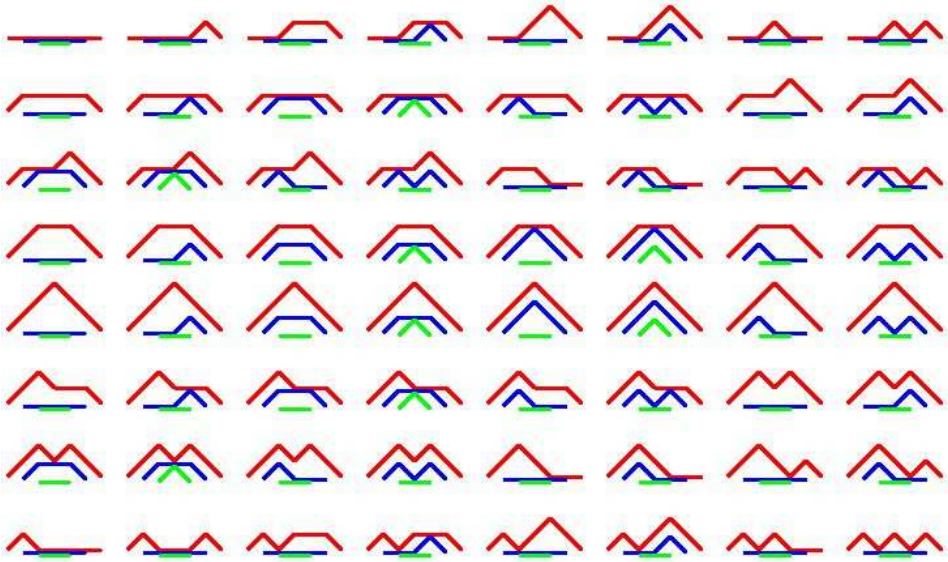


*Figure 18* : Pour $L = 3$, les 64 familles de chaînes de montagnes pouvant se toucher sans se traverser

Le programme principal consiste essentiellement à fabriquer toutes les chaînes de montagnes de longueur $2L$, correspondant à la chaîne la plus haute de chaque famille. Mais chaque fois qu'une telle chaîne est construite, en utilisant ici la première méthode d'énumération, une fonction récursive *schroeder*$(L-1)$ est appelée, qui va tracer toutes les chaînes situées au-dessous. Remarquons que les mots de Schroeder sont enregistrés dans un tableau à double entrée $a[i][j]$ où $j$ correspond au numéro de la chaîne, et $i$ à l'indice de chaque lettre du mot.

```
 xorig=10; yorig=zoom*L+5;
for(i=0;i<2*L;i++) a[i][0]=0;
for(;;)
   {chaine(L,0);  /* la chaîne supérieure, numéro 0 */
     schroeder(L-1);   /* appel de la fonction récursive qui va construire les autres chaînes */
     flag=0;          /* utilisation de la première méthode d'énumération */
     for(i=0;i<2*L;i+=2) if (! (a[i][0]==1 && a[i+1][0]==2)) { flag=1; break;}
     if (flag==0) break;
      i=2*L-1;
     while (! ((a[i][0]==0 && a[i-1][0]==0 && i>=1)
         || (a[i][0]==2 && a[i-1][0]==2 && a[i-2][0]==1 && i>=2))) i--;
     if (a[i][0]==0 && a[i-1][0]==0)
       { pospivot=i-1;  a[pospivot][0]=1;    nb1=0; nb2=0;
         for(j=2*L-1;j>pospivot;j--)
              { if (a[j][0]==1) nb1++ ;     if (a[j][0]==2) nb2++;
              }
         if (nb1==0)  for(j=pospivot+1;j<2*L;j++) a[j][0]=2;
         else
           { for(j=2*L-1;j>2*L-1-(nb2-nb1+1);j--) a[j][0]=2;
             for(k=pospivot+1;k<=j;k++) a[k][0]=0;
             }
       }
     else
       { pospivot=i-2;  a[pospivot][0]=2;     nb1=0; nb2=0;
         for(j=2*L-1;j>pospivot;j--)  { if (a[j][0]==1) nb1++ ;    if (a[j][0]==2) nb2++; }
         for(j=pospivot+1;j<2*L;j++) a[j][0]=0;
         for(j=2*L-1;j>2*L-1-(nb2-nb1-2);j--) a[j][0]=2;
       }
     }
```

Passons à la fonction récursive *schroeder*(LL) appelée au départ dans le programme principal pour $LL = L - 1$, et qui va se rappeler sur les valeurs décroissantes de *LL*, jusqu'au test d'arrêt où $LL = 0$, auquel cas on dessine la famille de chaînes de montagnes correspondante, grâce à la fonction *dessinLchaines*(). La boucle principale *for*(;;) énumère les chaînes de montagnes de longueur 2*LL*, en utilisant la première méthode d'énumération, et lorsqu'elle tombe sur une chaîne qui ne traverse pas la chaîne déjà construite au-dessus d'elle, elle la prend et se rappelle sur les chaînes de longueur inférieure $LL - 1$ par *schroeder*(*LL*-1).

```
void schroeder(int LL)
 {  int i, j,k,pospivot,nb1,nb2,flag,fini;
    if (LL==0) dessinLchaines();   /* test d'arrêt */
    else
       {
         for(i=0;i<2*LL;i++) a[i][L-LL]=0;
         for(;;)
           { chaine(LL,L-LL);
             fini=1;
             for(j=0;j<2*(LL);j++)
             if (yc[L-LL][j]>yc[L-LL-1][j+1]) {fini=0; break;} /* test de traversée */
             if (fini==1) schroeder(LL-1);  /* cas où la chaîne ne traverse pas celle du dessus */
             flag=0;   /* la première méthode d'énumération des mots de Schroeder */
             for(i=0;i<2*LL;i+=2) if (! (a[i][L-LL]==1 && a[i+1][L-LL]==2)) { flag=1; break;}
             if (flag==0) break;
             i=2*LL-1;
             while (! ((a[i][L-LL]==0 && a[i-1][L-LL]==0 && i>=1)
                   || (a[i][L-LL]==2 && a[i-1][L-LL]==2 && a[i-2][L-LL]==1 && i>=2))) i--;
             if (a[i][L-LL]==0 && a[i-1][L-LL]==0)
               { pospivot=i-1;  a[pospivot][L-LL]=1;    nb1=0; nb2=0;
                 for(j=2*LL-1;j>pospivot;j--)
                      { if (a[j][L-LL]==1) nb1++ ;     if (a[j][L-LL]==2) nb2++;
                      }
                 if (nb1==0)  for(j=pospivot+1;j<2*LL;j++) a[j][L-LL]=2;
                 else
                    { for(j=2*LL-1;j>2*LL-1-(nb2-nb1+1);j--) a[j][L-LL]=2;
                      for(k=pospivot+1;k<=j;k++) a[k][L-LL]=0;
                    }
               }
             else
               { pospivot=i-2;  a[pospivot][L-LL]=2;     nb1=0; nb2=0;
                 for(j=2*LL-1;j>pospivot;j--)
                      { if (a[j][L-LL]==1) nb1++ ;    if (a[j][L-LL]==2) nb2++;
                      }
                 for(j=pospivot+1;j<2*LL;j++) a[j][L-LL]=0;
                 for(j=2*LL-1;j>2*LL-1-(nb2-nb1-2);j--) a[j][L-LL]=2;
               }
           }
       }
 }

void chaine(int LL, int k)  /* pour avoir les coordonnées (xs, yc) des points de la chaîne associée
                               au mot a[][] */
{    int i;
   xs[k][0]=k;yc[k][0]=0;
  for(i=1;i<=2*LL;i++)
    { if (a[i-1][k]==0) {  xs[k][i]=xs[k][i-1]+1; yc[k][i]=yc[k][i-1];}
      else if(a[i-1][k]==1) { xs[k][i]=xs[k][i-1]+1; yc[k][i]=yc[k][i-1]+1; }
      else if(a[i-1][k]==2) { xs[k][i]=xs[k][i-1]+1; yc[k][i]=yc[k][i-1]-1; }
    }

}
```

```
void dessinLchaines(void)
{
  int i,K;
  for(K=L; K>=1; K--)
  for(i=1;i<=2*K;i++)
  linewithwidth(xorig+zoom*xs[L-K][i-1],yorig-zoom*yc[L-K][i-1]+2*(L-K),
          xorig+zoom*xs[L-K][i],yorig-zoom*yc[L-K][i]+2*(L-K),1,color[ L-K]);
  xorig+=2*L*zoom+20;
  if (xorig>800-2*L*zoom) {yorig+=2*L*zoom/2+20;xorig=10;}
  if (yorig>600-L*zoom) { SDL_Flip(screen);pause();SDL_FillRect(screen,0,white);
                          xorig=10; yorig=zoom*L+5;
                        }
}

void dessinchaine(int LL, int k)
{ int i;
  for(i=1;i<=2*LL;i++)
  linewithwidth(xorig+zoom*xs[k][i-1],yorig-zoom*yc[k][i-1],
          xorig+zoom*xs[k][i],yorig-zoom*yc[k][i],1,black);
}
```

## *Références bibliographiques*


[AUD2009] P. Audibert, Combien ? Mathématiques appliquées à l'informatique, Lavoisier 2009.

[BOS2013] F. Bosio, M.A.A. van Leeuwen, *A Bijection Proving the Aztec Diamond Theorem by Combing Lattice Paths*, *Electr. J. of Combinatorics*, 20, 2013.

[EU2005] S.-P. EU, T.-S. Fu, *A simple Proof of the Aztec Diamond Theorem*, *Electr. J. of Combinatorics*, 12, 2005.

[GES1985] I. Gessel, G. Viennot, *Binomial determinants, paths, and hook length formulae*, *Advances in Math.* 58, 1985.