Univers de murs et arbre *BSP*

Le contexte

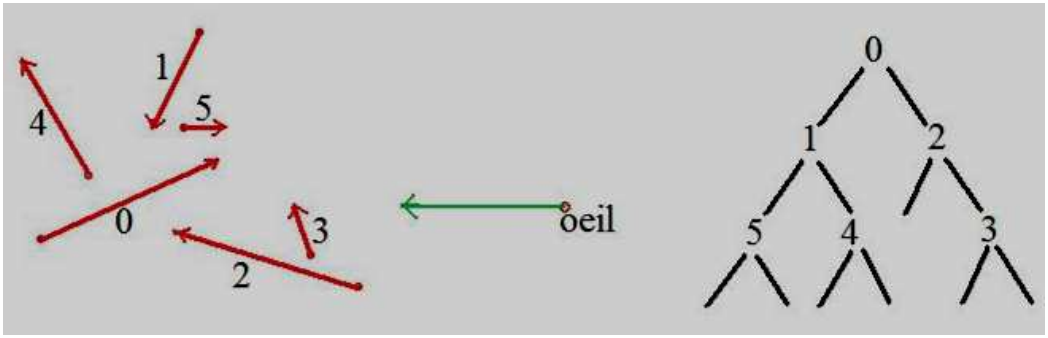
Plaçons-nous dans le cas simple où le paysage est constitué de plusieurs murs rectilignes posés sur le sol et que l'on regarde en leur faisant face. Il s'agit principalement d'un problème en deux dimensions. L'objectif est de dessiner les parties de murs visibles par l'œil de l'observateur.

Arbre *BSP* (*binary search partitioning*)

Les données initiales sont les *N* murs, numérotés de 0 à *N*-1, et considérés comme des segments orientés avec une extrémité origine (*xe*1[], *ye*1[]) et une extrémité finale (*xe*2[],*ye*2[]). On connaît aussi la position de l'œil (*xoeil*, *yoeil*) et sa direction de vision grâce au vecteur ***V****oeil*. Notre univers de murs va être enregistré dans une structure de données en forme d'arbre binaire, et l'on verra ensuite pourquoi cela facilite son dessin. La méthode consiste prendre chaque mur l'un après l'autre et à construire un arbre binaire dont les nœuds contiennent chaque numéro des murs. Après avoir mis dans la racine de l'arbre le mur 0, on fait descendre chacun des autres murs dans le sous-arbre gauche ou le sous-arbre droit selon qu'il est à gauche ou à droite des murs enregistrés dans les nœuds déjà construits, jusqu'à la première place vide. Ce procédé est identique à celui du tri par arbre binaire (cf. *Algorithmique chapitre* 6, à revoir au préalable avant de se lancer dans l'arbre *BSP*). Mais cela suppose une situation où chaque mur est soit à gauche soit à droite, et non pas de part et d'autre. Nous allons d'abord traiter ce contexte le plus simple, avant de passer au cas général.

Gauche ou droite, exclusivement

On met le mur 0 dans la racine de l'arbre. Puis le mur numéro 1 est placé à gauche ou à droite sous la racine selon qu'il est à gauche ou à droite du mur 0. Puis on fait descendre le mur 2 à gauche ou à droite des nœuds déjà placés, selon sa position, etc. Cela donne un arbre qui concentre l'information sur la position relative des murs. Notons que cet arbre est complètement indépendant de la position de l'œil. Voici un exemple de murs avec l'arbre associé :



Dans le programme correspondant, on commence par entrer la structure des nœuds :

```
struct cell {int n;  struct cell * droite; struct cell * gauche;};
```

Puis on construit l'arbre :

```
racine=(struct cell *) malloc(sizeof(struct cell));
racine->n=0; racine->gauche=NULL; racine->droite=NULL;
for(i=1;i<N;i++)
 {  crochety=NULL;crochetx=racine;
    newcell=(struct cell *) malloc(sizeof(struct cell));
    newcell->n=i; newcell->droite=NULL; newcell->gauche=NULL;
    while(crochetx!=NULL)
     {  crochety=crochetx;      numero=crochetx->n;
        xa=xe1[crochetx->n];ya=ye1[crochetx->n];
        xxa=xe2[crochetx->n];yya=ye2[crochetx->n];
        gauchedroite=dequelcote(xe1[i],ye1[i],xe2[i],ye2[i],xa,ya,xxa,yya);
        if (gauchedroite==1) crochetx=crochetx->gauche;
        else crochetx=crochetx->droite;
     }
   if (gauchedroite==1) crochety->gauche=newcell;
   else crochety->droite=newcell;
 }
parcoursarbre(racine);
```

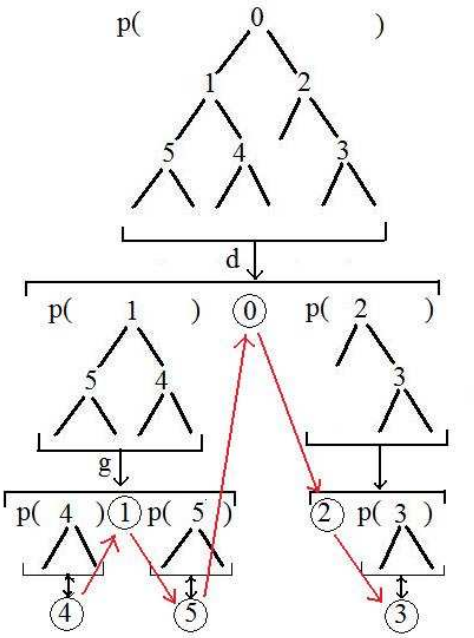
A noter la présence de la fonction *dequelcote*() qui teste de quel côté se trouve le segment en cours de descente *xe*1[*i*],*ye*1[*i*],*xe*2[*i*],*ye*2[*i*] par rapport au segment *xa*, *ya*, *xxa*, *yya* correspondant au nœud de l'arbre. Cette fonction ramène 1 ou -1 selon qu'on est à gauche ou à droite. Enfin il y a la fonction *parcoursarbre*() qui va nous permettre de comprendre l'intérêt de l'arbre BSP.

Parcours de l'arbre

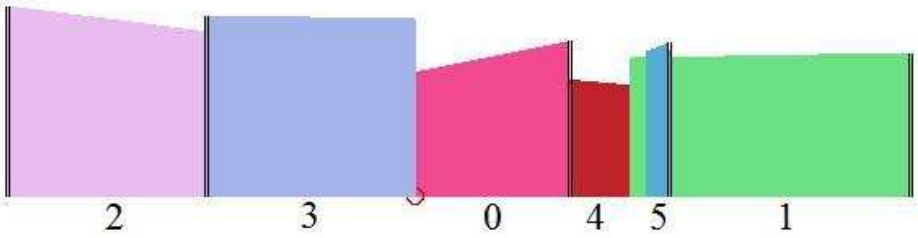
Faisons maintenant intervenir la position de l'œil. Si l'œil est à droite du mur numéro *i*, on doit d'abord dessiner les murs qui sont à gauche de ce mur *i* (et qui sont derrière par rapport à l'œil) avant de dessiner ce mur, et dessiner ce mur avant ceux qui sont situés à sa droite, cela afin de préserver les parties visibles des murs. Cela correspond à un parcours infixé du sous-arbre dont le mur *i* est la racine. Par contre si l'œil est à gauche du mur *i*, on doit commencer par tracer les murs à droite, aussi va-t-on échanger le sous-arbre gauche et le sous-arbre droit de l'arbre de racine *i*, de façon à faire ensuite comme auparavant un parcours infixé. Autrement dit, la position de l'œil ne fait que provoquer quelques interversions dans l'arbre, ce qui est facile à opérer, tout comme le parcours infixé.

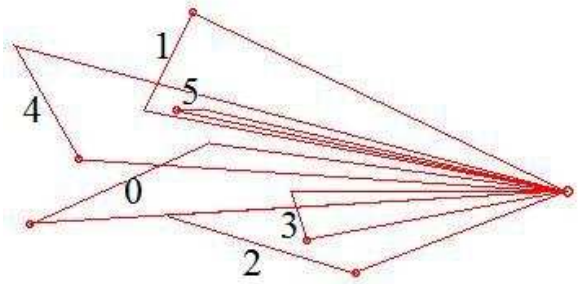
```
void parcoursarbre(struct cell * r)
{  if (r!=NULL)
     {  if (oeilgauche(r)==NON)      /*  fonction oeilgauche() à faire  */
          { parcoursarbre (r->gauche);   dessiner le mur   parcoursarbre(r->droite); }
        else
          {  parcoursarbre (r->droite);   dessiner le mur    parcoursarbre(r->gauche); }
      }
}
```

Voici comment marche ce parcours dans l'exemple choisi :



Les murs sont affichés dans l'ordre 4 1 5 0 2 3.





Ayant obtenu l'ordre dans lequel on va prendre les murs, il reste à la dessiner sur l'écran. Traçons pour cela un arc de cercle de centre l'œil, cet arc étant coupé en *O* par le vecteur ***Voeil*** donnant la direction de la vision. C'est cet arc qui va jouer le rôle d'écran. Pour chaque mur, on détermine les deux angles (positifs ou négatifs) que font ses deux extrémités avec la direction de l'œil, ce qui donne leur distance (la longueur de l'arc) au point *O* sur l'écran. On ajoute enfin un effet de perspective en diminuant la hauteur du mur entre ses deux extrémités selon qu'elles sont plus ou moins loin de l'œil. Remarquons que notre écran est courbe, ce qui n'est pas le cas de l'écran d'ordinateur, mais la déformation est faible. On pourrait d'ailleurs la rectifier en prenant le sinus des angles.

Cas général

Nous allons maintenant nous placer dans le cas général où un mur n'est pas forcément à gauche ou à droite d'un autre mur, mais se trouve de part et d'autre. Il convient alors de découper le mur suivant sa partie à gauche, qui conserve le numéro du mur complet initial, et sa partie à droite, qui devient un nouveau mur, avec un nouveau numéro (qui suit celui du mur de gauche), et qui s'insère dans la liste des murs. On appellera *NN* le nombre de ces murs après découpage, étant entendu que *NN* est égal à *N* au début, mais qu'il augmente de un à chaque découpage d'un mur en deux. Les numéros des murs en cours d'évolution sont placés dans un tableau *a*[]. Si le mur découpé a sa partie gauche dans *a*[*i*], sa partie droite sera placée dans *a*[*i*+1]. La fonction *dequelcote*(), qui jusqu'ici ramenait 1 ou -1 selon que le mur était à gauche ou à droite d'un autre mur, va maintenant ramener aussi 2 o -2, selon que le mur passe de gauche à droite ou de droite à gauche.

Le problème de l'intersection

Pour permettre le découpage éventuel d'un mur, il s'agit de chercher le point où se fait la coupure. Prenons *C* (*X*, *Y*) et *D* (*XX*, *YY*) de part et d'autre du vecteur ***AB*** avec *A* (*x*, *y*) et *B* (*xx*, *yy*), et déterminons le point d'intersection *K* de (*AB*) avec [*CD*]. Considérons d'abord le cas où *C* est à gauche et *D* à droite de ***AB*** : det(***AB***, ***AC***)>0 et det(***AB***, ***AD***)<0. Le point *B* est le barycentre des points *A*, *C* et *D*, avec comme masses respectives les aires des triangles *CBD*, *ADB* et *ABC*, ou leur double, qui sont les déterminants des vecteurs correspondants. Le point *K* est le barycentre de *C* et *D* avec les coefficients $d1 = \det(\boldsymbol{AB}, \boldsymbol{AC})$ et $-d2 = -\det(\boldsymbol{AB}, \boldsymbol{AD})$. Cela se traduit par :

$\boldsymbol{CK} = (d1/(d1 - d2))\ \boldsymbol{CD}$, d'où $xk = (d1/(d1 - d2))(XX-X)+x$, et même pour *yK*.

On fera de même lorsque *C* est à droite et *D* à gauche de ***AB***.

Il y a maintenant un problème subtil. Imaginons que l'on ait pris les coordonnées des extrémités des murs en entiers. Jusque là tout va bien. Mais en cas de problème d'intersection, si l'on prend aussi les coordonnées du point *K* en entiers, il se produit une légère erreur d'arrondi, et ensuite le test pour savoir si la partie de mur est à gauche ou à droite risque de rater : la partie gauche peut être testée comme empiétant sur la droite. Il est donc obligatoire de tout prendre en flottants, et de laisser une petite marge d'erreur, du style *determinant*<=0.00001 au lieu de *determinant*<=0.

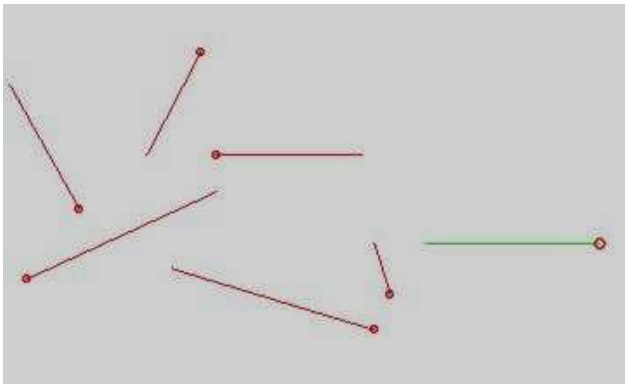
On aboutit ainsi au programme final.

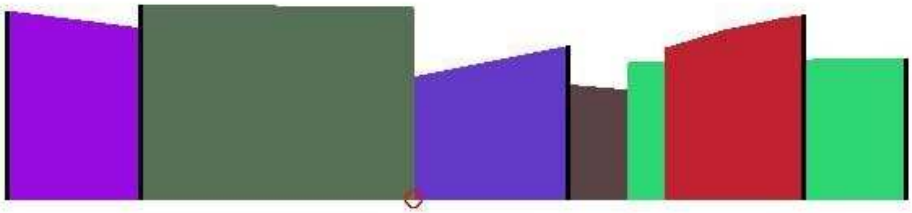
```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <SDL/SDL.h>
#define OUI 1
#define NON 0
#define N 6
#define z 100.   /* zoom  */
```

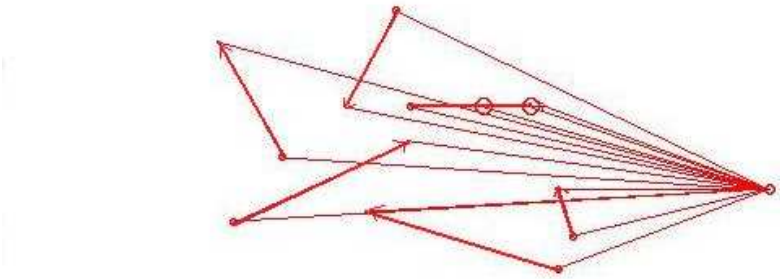
```
void ligne(int x0,int y0, int x1,int y1, Uint32 c);
void pause(void);
void cercle( int xo, int yo, int RR, Uint32 couleur);
int dequelcote(float X, float Y, float XX, float YY,float x,float y, float xx, float yy);
void parcoursarbre(struct cell * r);
int oeilgauche(struct cell * rr);
void dessinmurs(void);
void floodfill( int x,int y, Uint32 cr,Uint32 cb);
Uint32 blanc,noir,rouge,vert,color[100],couleur,c[100];SDL_Surface *ecran;
float xe1[1000],ye1[1000],xe2[1000],ye2[1000]; int a[1000],NN,q;
float  oeilx,oeily,vectoeilx,vectoeily; float det1,det2;
float longoeile,longoeilo, longvectoeil;int X1,Y1,X2,Y2;
float oeilxe,oeilye,oeilxo,oeilyo,xg,yg;  float angle1,angle2;
struct cell {int n; struct cell * droite;struct cell * gauche;};
struct cell * racine;

int main ( int argc,char *argv[])
{  int i,j,resultat,xg,yg,xd,yd,gauchedroite,numero;
   float xint,yint, xc,yc,xxc,yyc,xa,ya,xxa,yya;
   struct cell * crochetx, * crochety,*newcell;
 SDL_Init( SDL_INIT_VIDEO );srand(time(NULL));
 ecran= SDL_SetVideoMode(800, 600, 32,SDL_HWSURFACE | SDL_DOUBLEBUF);
 blanc=SDL_MapRGB(ecran->format,255,255,255);
 noir=SDL_MapRGB(ecran->format,0,0,0);
 rouge=SDL_MapRGB(ecran -> format, 255, 0, 0);
 vert=SDL_MapRGB(ecran -> format, 0,255, 0);
 for(j=0;j<100;j++)
 color[j]=SDL_MapRGB(ecran -> format, rand()%200, rand()%256, rand()%256);
 SDL_FillRect(ecran,NULL,blanc);

xe1[0]=0.20;ye1[0]=0.50; xe2[0]=1.30;ye2[0]=1.00; /* un exemple avec six murs  */
xe1[1]=1.20;ye1[1]=1.80; xe2[1]=0.90;ye2[1]=1.20;
xe1[2]=2.20;ye1[2]=0.20; xe2[2]=1.05;ye2[2]=0.55;
xe1[3]=2.30 ;ye1[3]=0.40; xe2[3]=2.20;ye2[3]=0.70;
xe1[4]=0.50;ye1[4]=0.90; xe2[4]=0.10;ye2[4]=1.60;
xe1[5]=1.30;ye1[5]=1.20; xe2[5]=2.13 ;ye2[5]=1.20;

for(i=0;i<N;i++)
  {  ligne(z*xe1[i],450-z*ye1[i],z*xe2[i],450-z*ye2[i],rouge);
     cercle(z*xe1[i],450-z*ye1[i],2,rouge);SDL_Flip(ecran);
  }
oeilx=3.50;oeily=0.70;cercle(z*oeilx,450-z*oeily,3,rouge);  vectoeilx=-1.00;vectoeily=0;
ligne(z*oeilx,450-z*oeily,z*oeilx+z*vectoeilx,450-(z*oeily+z*vectoeily),vert);
SDL_Flip(ecran);pause();

for(i=0;i<N;i++)   a[i]=i;
NN=N;
racine=(struct cell *) malloc(sizeof(struct cell));
racine->n=a[0]; racine->gauche=NULL; racine->droite=NULL; c[a[0]]=color[0];
for(i=1;i<NN;i++)
{
crochety=NULL;crochetx=racine;
newcell=(struct cell *) malloc(sizeof(struct cell));
newcell->n=a[i]; newcell->droite=NULL; newcell->gauche=NULL;
xc=xe1[a[i]];yc=ye1[a[i]];xxc=xe2[a[i]];yyc=ye2[a[i]];
```

```
if (c[a[i]]==0) c[a[i]]=color[i];  /* couleur finale des murs  */

while(crochetx!=NULL)
  {  crochety=crochetx;
     numero=crochetx->n;
     xa=xe1[crochetx->n];ya=ye1[crochetx->n];
     xxa=xe2[crochetx->n];yya=ye2[crochetx->n];
     xc=xe1[a[i]];yc=ye1[a[i]];xxc=xe2[a[i]];yyc=ye2[a[i]];
     gauchedroite=dequelcote(xc,yc ,xxc ,yyc ,xa,ya,xxa,yya);
     if (gauchedroite==1) crochetx=crochetx->gauche;
     else if (gauchedroite==-1)  crochetx=crochetx->droite;
     else if (gauchedroite==2)
       { xint=det1*(float)(xxc-xc)/(det1-det2) +xc;
         yint=det1*(float)(yyc-yc)/(det1-det2)+yc;
         cercle(z*xint,450-z*yint,5,rouge);
         xe2[a[i]]=xint; ye2[a[i]]=yint;
         for(q=NN-1;q>i;q--)
               { a[q+1]=a[q]; xe1[a[q+1]]=xe1[a[q]]  ; ye1[a[q+1]]=ye1[a[q]]    ;
                 xe2[a[q+1]]=xe2[a[q]]  ; ye2[a[q+1]]=ye2[a[q]] ;
               }
         a[i+1]=NN; xe1[a[i+1]]=xint; ye1[a[i+1]]=yint; xe2[a[i+1]]=xxc; ye2[a[i+1]]=yyc;
         c[a[i+1]]=c[a[i]]; /* on garde la même couleur pour chaque partie du même mur */
         NN++;
       }
     else if (gauchedroite==-2)
      { xint=det2*(float)(xxc-xc)/(det2-det1)+xc;
            yint=det2*(float)(yyc-yc)/(det2-det1) +yc;
            xe2[a[i]]=xint; ye2[a[i]]=yint; cercle(xint,450-yint,3,vert );
        for(q=NN-1;q>i;q--)
               {
                 a[q+1]=a[q]; xe1[a[q+1]]=xe1[a[q]]  ; ye1[a[q+1]]=ye1[a[q]]    ;
                 xe2[a[q+1]]=xe2[a[q]]  ; ye2[a[q+1]]=ye2[a[q]]    ;
               }
            a[i+1]=NN; xe1[a[i+1]]=xint; ye1[a[i+1]]=yint;
        xe2[a[i+1]]=xxc; ye2[a[i+1]]=yyc;
        c[a[i+1]]=c[a[i]];
        NN++;
       }
  }
if (gauchedroite==1) crochety->gauche=newcell;
else crochety->droite=newcell;
}
parcoursarbre(racine);
SDL_Flip(ecran);pause(); return 0;
}

int dequelcote(float X, float Y, float XX, float YY,float x,float y, float xx, float yy)
{  float vx,vy,vX,vY,vXX,vYY;
   vx=xx-x;vy=yy-y;   vX=X-x;vY=Y-y;   vXX=XX-x;vYY=YY-y;
   det1=vx*vY-vy*vX;  det2=vx*vYY-vy*vXX;
   if (det1>=-0.00001 && det2>=-0.00001) return 1;
   if (det1<=0.00001 && det2<=0.00001) return -1;
   if (det1>=-0.00001 && det2<=0.00001 ) return 2;
   if (det1<=0.00001 && det2>=-0.00001) return -2;
   else return 0;
```

```
}
void parcoursarbre(struct cell * r)
{  if (r!=NULL)
     {  if (oeilgauche(r)==0)
          { parcoursarbre (r->gauche);
            oeilxo=xe1[r->n] -oeilx;  oeilyo=ye1[r->n]-oeily;
            oeilxe=xe2[r->n]-oeilx;  oeilye=ye2[r->n]-oeily;
            longvectoeil=sqrt(vectoeilx*vectoeilx+vectoeily*vectoeily);
            longoeilo=sqrt(oeilxo*oeilxo+oeilyo*oeilyo);
            longoeile=sqrt(oeilxe*oeilxe+oeilye*oeilye);
            ligne(z*oeilx,450-z*oeily,z*xe1[r->n],450-z*ye1[r->n],rouge);
            ligne(z*oeilx,450-z*oeily,z*xe2[r->n],450-z*ye2[r->n],rouge);
            couleur=c[r->n];dessinmurs();
            parcoursarbre(r->droite);
           }
         else
           {  parcoursarbre (r->droite);
              oeilxo=xe1[r->n] -oeilx;  oeilyo=ye1[r->n]-oeily;
              oeilxe=xe2[r->n]-oeilx;  oeilye=ye2[r->n]-oeily;
              longvectoeil=sqrt(vectoeilx*vectoeilx+vectoeily*vectoeily);
              longoeilo=sqrt(oeilxo*oeilxo+oeilyo*oeilyo);
              longoeile=sqrt(oeilxe*oeilxe+oeilye*oeilye);
              ligne(z*oeilx,450-z*oeily,z*xe1[r->n],450-z*ye1[r->n],rouge);
              ligne(z*oeilx,450-z*oeily,z*xe2[r->n],450-z*ye2[r->n],rouge);
              couleur=c[r->n];dessinmurs();
              parcoursarbre(r->gauche);
           }
      }
}
int oeilgauche(struct cell * rr)
{  float vx,vy,vvx,vvy,det;
   vx=xe2[rr->n]-xe1[rr->n];     vy= ye2[rr->n]- ye1[rr->n];
   vvx=oeilx-xe1[rr->n] ;vvy=oeily-ye1[rr->n];
   det=vx*vvy-vy*vvx;
   if ( det>0) return 1; else return 0;
}
void dessinmurs(void)
{
angle1=(oeilxo*vectoeily-oeilyo*vectoeilx) /(longvectoeil*longoeilo);
angle2=(oeilxe*vectoeily-oeilye*vectoeilx)  /(longvectoeil*longoeile);
X1=(int) (700.*angle1);Y1=150-z*longoeilo/4;
X2=(int) (700.*angle2);Y2=150-z*longoeile/4;
cercle(300,200,5,rouge);
ligne(300+X1,200,300+X1,200-Y1,couleur);ligne(300+X2,200,300+X2,200-Y2,couleur);
ligne(300+X1,200,300+X2,200,couleur);ligne(300+X1,200-Y1,300+X2,200-Y2,couleur);
xg=(300+X1+300+X2)/2; yg=(200+200-Y2)/2;
floodfill(xg,yg,couleur,couleur);
if (longoeilo<longoeile)
  { ligne(300+X1,200,300+X1,200-Y1,noir);  ligne(301+X1,200,301+X1,200-Y1,noir);
    ligne(302+X1,200,302+X1,200-Y1,noir);}
else   { ligne(300+X2,200,300+X2,200-Y2,noir);  ligne(301+X2,200,301+X2,200-Y2,noir);
         ligne(302+X2,200,302+X2,200-Y2,noir);}
}
```







(ici un des murs, celui qui est en rouge, a été coupé en trois)