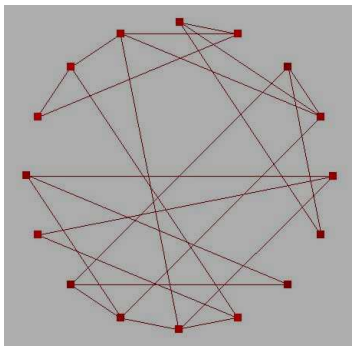
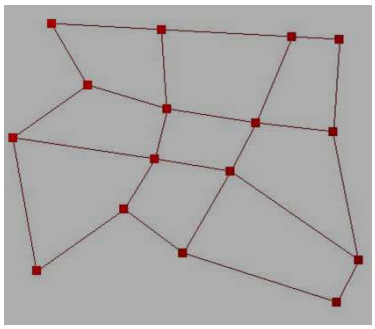
Planarité

Le jeu *Planarity* (cf. *planarity.net*), développé par John Tantalo, est à la fois simple, par son mode d'emploi et sa programmation, et captivant, voire exaspérant comme un supplice de Tantale. Il s'agit de démêler un graphe pour que finalement il devienne planaire, c'est-à-dire que ses arêtes ne se coupent jamais entre elles.



le graphe initial

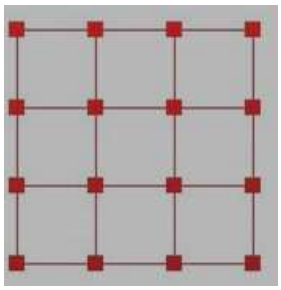


le même graphe, devenu planaire

Nous en donnons ici une version simplifiée, que nous allons programmer. Nous allons commencer par construire un graphe planaire simple, pour nous un réseau carré, puis nous y mettrons du désordre, en déplaçant les sommets au hasard. C'est là que le jeu commence : à partir de ce graphe désordonné, le joueur devra déplacer les sommets manuellement, avec l'aide de la souris, jusqu'à obtenir à la fin un graphe planaire. En signe de succès, un grand cercle apparaîtra alors sur l'écran.

Etapes préliminaires

1) Construire un réseau carré, avec N sommets par côté, soit N^2 sommets au total.

0  3

4 7

8 11

12 15

Un réseau carré, avec N=4, et NN=16 sommets numérotés de 0 à 15

Il s'agit d'entrer le graphe en machine. Pour chaque sommet i, ses voisins vont être enregistrés dans le tableau $v[i][]$, ainsi que leur nombre $nbv[i]$. Mais les sommets ont quatre, trois ou deux voisins, selon leur position. Ainsi les sommets qui ont un voisin à leur droite sont ceux qui ne sont pas sur la bordure de droite, ils portent un numéro i tel que $(i+1)\%N$ est différent de 0. Ceux qui n'ont pas de voisin au-dessus d'eux sont ceux qui ne sont pas sur la bordure supérieure, soit $i>=N$. Ceux qui ne sont pas sur la bordure de gauche, soit $i\%N$ différent de 0, ont un voisin à gauche, et ceux qui ne sont pas sur la

dernière ligne, $i < NN - N$, ont un voisin au-dessous. Cela permet d'enregistrer les voisins de chaque sommet, grâce au programme suivant, où le tableau *nbv*[] est supposé mis à 0 au départ :

```
for(i=0;i<NN;i++)
   {
      if ((i+1)%N!=0) v[i][nbv[i]++]=i+1;
      if (i>=N) v[i][nbv[i]++]=i-N;
      if (i%N!=0) v[i][nbv[i]++]=i-1;
      if (i<NN-N) v[i][nbv[i]++]=i+N;
   }
```

2) Donner un numéro à chaque arête du graphe et enregistrer ses deux sommets extrêmes dans *extr*1[] et *extr*2 [] avec *extr*1 < *extr*2. Le nombre des arêtes sera placé dans *NA*.

Il suffit de parcourir la liste des voisins de chaque sommet *i*, où les arêtes entre *i* et chaque voisin *v*[*i*][*j*]j seront prises pourvu que *i* soit inférieur au voisin.

```
k=0;
for(i=0;i<NN;i++) for(j=0;j<nbv[i];j++) if (i<v[i][j])
   {extr1[k]=i; extr2[k]=v[i][j];k++;}
NA=k;
```

Puis on a les étapes suivantes à traiter. Les morceaux de programme qui leur correspondent se trouvent dans le programme global ci-dessous.

3) Déterminer les coordonnées *x*[] et *y*[] de chaque sommet.

4) Placer un carré colorié en rouge autour de chaque sommet *i*, soit *pointcarre*[*i*], avec les coordonnées de son coin en haut à gauche dans *position.x* et *position.y*. Chaque sommet possède sa propre nuance de couleur *rouge*[*i*]. La couleur *rouge*[*NN*] sera utilisée pour colorier les arêtes.

5) Procéder au dessin du réseau carré obtenu (à noter la présence de la fonction *fini*() qui teste si le graphe est planaire, et dans le cas favorable dessine un grand cercle bleu sur l'écran, ce qui se produit bien dans le cas présent. La fonction *fini*() sera expliquée plus tard).

6) On va maintenant mettre du désordre dans le graphe précédent. Commencer par faire une permutation $i \rightarrow p[i]$ sur les numéros des sommets. Puis placer les sommets ainsi permutés sur un cercle, à intervalles réguliers. Par exemple si la permutation fait passer de 012 à 201, les points placés sur le cercle seront 201 dans cet ordre, et leurs coordonnées sont enregistrées dans *x*[] et *y*[]. Il ne reste plus qu'à remettre des carrés rouges autour de ces sommets avec leur couleur spécifique *rouge*[] puis de tracer les arêtes de jonction dans ce nouveau contexte.

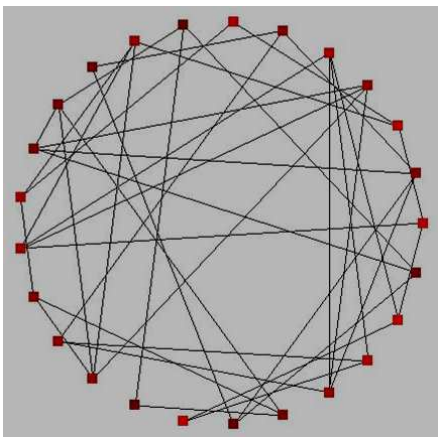
```
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <SDL/SDL.h>
#define OUI 1
#define NON 0
#define N 4
```

```
#define NN (N*N)
#define pas 50
#define R 200
#define pi 3.14159
void ligne(int x0,int y0, int x1,int y1, Uint32 c);   /*  fonctons utilisées  */
void pause(void);
void cercle( int xo, int yo, int RR, Uint32 couleur);
int secoupent(int q, int qq);
int fini(void);
Uint32 getpixel(int xe, int ye);
Uint32 blanc,rouge[NN+1],vert,bleu;SDL_Surface *ecran,*pointcarre[NN];
SDL_Rect position[NN];
int v[NN][4],nbv[NN],x[NN],y[NN],dejafait[NN],p[NN],NA,
      extr1[NN*NN],extr2[NN*NN];

int main ( int argc,char *argv[])
{
 int i,j,k,h,flag, continuer=OUI;   SDL_Event event;
for(i=0;i<NN;i++)          /********   1   ********/
    {  if ((i+1)%N!=0) v[i][nbv[i]++]=i+1;
       if (i>=N) v[i][nbv[i]++]=i-N;
       if (i%N!=0) v[i][nbv[i]++]=i-1;
       if (i<NN-N) v[i][nbv[i]++]=i+N;
    }
k=0;                           /********   2   ********/
for(i=0;i<NN;i++) for(j=0;j<nbv[i];j++) if (i<v[i][j])
   {extr1[k]=i; extr2[k]=v[i][j];k++;}
NA=k;
   for(i=0;i<NN;i++)       /********   3   ********/
      {   y[i]=200+pas*(i/N);  x[i]=300+pas*(i%N);   }
   SDL_Init( SDL_INIT_VIDEO );
   ecran= SDL_SetVideoMode(800, 600, 32,SDL_HWSURFACE | SDL_DOUBLEBUF);
   blanc=SDL_MapRGB(ecran->format,255,255,255);
   vert=SDL_MapRGB(ecran -> format, 0,255, 0);
   bleu=SDL_MapRGB(ecran -> format, 0,0,255);
   SDL_FillRect(ecran,NULL,blanc);
   for(i=0;i<NN;i++)       /********   4   ********/
    { pointcarre[i]= SDL_CreateRGBSurface(SDL_HWSURFACE,10,10,32,0,0,0,0);
      rouge[i]=SDL_MapRGB(ecran -> format, 255-5*i, 0, 0);
    }
   rouge[NN]=SDL_MapRGB(ecran -> format, 255-20*NN, 0, 0);
   for(i=0;i<NN;i++)
    {  SDL_FillRect(pointcarre[i],NULL,rouge[i]);
       position[i].x = x[i]-5; position[i].y = y[i]-5;
    }
   for(i=0;i<NN;i++)       /********   5   ********/
      SDL_BlitSurface(pointcarre[i], 0, ecran, &position[i]);
   for(i=0;i<NN;i++) for(j=0;j<nbv[i];j++) if (i<v[i][j] )
     ligne(x[i],y[i],x[v[i][j]],y[v[i][j]],rouge[NN]);
   if (fini()==OUI) cercle(400,300,400,bleu);
   SDL_Flip(ecran); pause();
   SDL_FillRect(ecran,NULL,blanc);
   srand(time(NULL));    /********   6   ********/
   for(i=0;i<NN;i++)
    {  do   h=rand()%NN;  while (dejafait[h]==OUI);
```

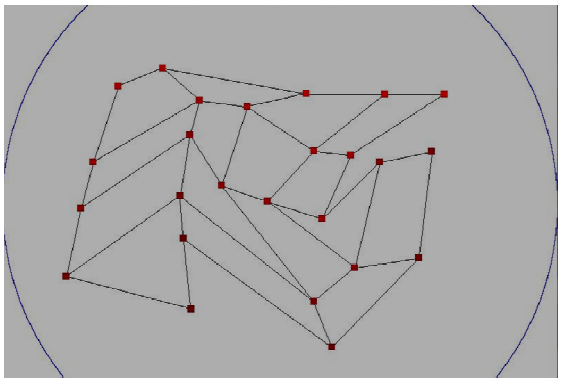
```
    p[i]=h; dejafait[h]=OUI;
  }
for(i=0;i<NN;i++)
  {  x[p[i]]=400+R*cos(2.*pi*(float)i/(float)NN);
     y[p[i]]=300- R*sin(2.*pi*(float)i/(float)NN);
  }
for(i=0;i<NN;i++)    {  SDL_FillRect(pointcarre[i],NULL,rouge[i]);
                           position[i].x = x[i]-5; position[i].y = y[i]-5;
                        }
for(i=0;i<NN;i++)     SDL_BlitSurface(pointcarre[i], 0, ecran, &position[i]);
for(i=0;i<NN;i++) for(j=0;j<nbv[i];j++) if (i<v[i][j] )
   ligne(x[i],y[i],x[v[i][j]],y[v[i][j]],rouge[NN]);
SDL_Flip(ecran);
```

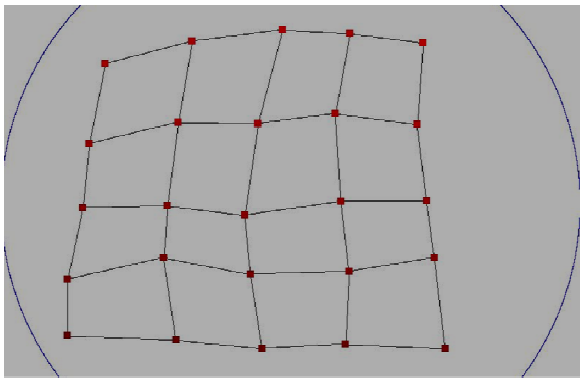
Le jeu



Après ce qui précède, voici ce que voit le joueur sur l'écran. Il va maintenant utiliser la souris pour bouger les points, et tenter d'obtenir finalement le graphe sous forme planaire.

Comment faire bouger un point ? Il suffit de déplacer la souris de façon que la flèche qui lui correspond sur l'écran vienne se placer sur un carré rouge représentant un point. En appuyant sur un bouton de la souris, le point se trouvera agrippé à la souris. En déplaçant celle-ci, le point suivra le même mouvement. Une fois arrivé là où on veut mettre le point, il suffit de relâcher le bouton de la souris, le point se fixe et la souris reprend sa liberté pour aller vers un autre point à déplacer. En cas de victoire, le graphe étant devenu planaire, un grand cercle bleu apparaîtra sur l'écran. Le joueur pourra encore améliorer le résultat en redessinant le graphe sous forme du réseau carré.





Le graphe redevenu planaire (à gauche) et retrouvant même sa forme de réseau carré (à droite)

Pour le programme, nous avons repris celui déjà fait pour la *courbe de Bézier en mouvement* (cf. *problèmes complémentaires –algorithmes–*4.). Il s'agit d'un programme évènementiel, où une boucle ne cesse de tourner dans l'attente du moindre évènement, venant essentiellement de la souris. Lors du mouvement d'un point sous l'effet de la

souris, les arêtes liées à ce sommet suivent le mouvement. A chaque instant aussi, le programme teste s'il existe des intersections entre les arêtes, par le biais des fonctions *secoupent*() et *fin*(), que nous expliquerons plus bas. Il en résulte cette partie du programme qui s'ajoute à la précédente.

```
    flag=-1;
    while (continuer==OUI)
     { SDL_WaitEvent(&event);
       if (event.type==SDL_QUIT)     continuer=NON;
       else if (flag==-1 && event.type==SDL_MOUSEBUTTONDOWN )
        { for(i=0;i<NN;i++)
          if ( getpixel(event.button.x,event.button.y)==rouge[i])
                  { flag=i;   SDL_WarpMouse(position[i].x+5,position[i].y+5); break; }
        }
       else if (flag!=-1 && event.type==SDL_MOUSEBUTTONUP)  flag=-1;
       else if( event.type== SDL_MOUSEMOTION)
            {position[flag].x=event.button.x-5;   position[flag].y=event.button.y-5;}

       SDL_FillRect(ecran,NULL,blanc);
       for(i=0;i<NN;i++) for(j=0;j<nbv[i];j++) if (i<v[i][j] )
          ligne(position[i].x+5,position[i].y+5,
                position[v[i][j]].x+5,position[v[i][j]].y+5,rouge[NN]);
       for(i=0;i<NN;i++)    SDL_BlitSurface(pointcarre[i], 0, ecran, &position[i]);
       if (fini()==OUI) cercle(400,300,400,bleu);
       SDL_Flip(ecran);
     }
for(i=0;i<4;i++) SDL_FreeSurface(pointcarre[i]);
return 0;
}
```

Les tests d'intersection

Pour savoir si deux segments [*AB*] et [*CD*] se coupent, on exprime que *C* et *D* sont de part et d'autre de (*AB*) et que *A* et *B* sont de part et d'autre de (*CD*), comme on l'a vu dans le chapitre sur *Orientation et angles* (cf. cours *Graphisme et géométrie*).. Il convient aussi que l'intersection soit franche : si les deux segments ont une extrémité en commun, on considère qu'ils ne se coupent pas, ce qui impose qu'aucun des déterminants utilisés ne soit nul. C'est ce que fait ici la fonction *secoupent*(*q*, *qq*) qui prend comme variables les numéros *q* et *qq* de deux arêtes, et qui ramène si oui ou non les deux arêtes concernées se coupent. On comprend a posteriori l'intérêt d'avoir numéroté les arêtes. Enfin la fonction f*in*() se charge de prendre toutes les paires d'arêtes, et de compter le nombre des intersections. Quand il n'y en a plus, elle dessine un grand cercle sur l'écran.

```
int secoupent(int q, int qq)
{  int a, b, c ,d,abx,aby,acx,acy,adx,ady,cdx,cdy,cax,cay,cbx,cby;
   float det1,det2,det3,det4;
   a=extr1[q]; b=extr2[q];   c=extr1[qq];d=extr2[qq];
   abx=position[b].x-position[a].x; aby=position[b].y-position[a].y;
   acx=position[c].x-position[a].x; acy=position[c].y-position[a].y;
   adx=position[d].x-position[a].x; ady=position[d].y-position[a].y;
   det1=abx*acy-aby*acx;  det2=abx*ady-aby*adx;
   cdx=position[d].x-position[c].x; cdy=position[d].y-position[c].y;
```

```
   cax=-acx; cay=-acy;
   cbx=position[b].x-position[c].x; cby=position[b].y-position[c].y;
   det3=cdx*cay-cdy*cax;  det4=cdx*cby-cdy*cbx;
   if (det1*det2<0.  && det3*det4<0.) return OUI;
   return NON;
}

int fini(void)
{
   int i, e, ee, nbdefois=0;
   for(e=0;e<NA-1;e++) for(ee=e+1;ee<NA;ee++)
   if (secoupent(e,ee)==OUI) nbdefois++;
   for(i=0;i<nbdefois;i++) cercle(700,100,2*(i+1),bleu);
   if(nbdefois==0) return OUI; else return NON;
}
```