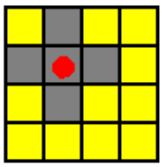
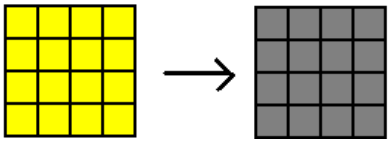
Le jeu du tout éteint (*light out game*)

1) Principe du jeu

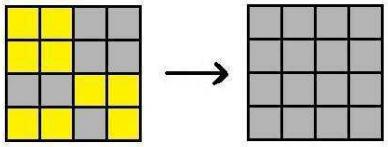


On se place dans un rectangle de longueur L et de hauteur H, possédant $N = L\,H$ cases. Ces cases peuvent être soit éteintes, soit allumées. Nous noterons ces deux états 0 et 1. On a la règle d'évolution suivante : lorsque l'on appuie sur une case, celle-ci change d'état ainsi que ses quatre voisines dans les directions sud, est, nord, ouest, du moins celles qui existent dans les limites du rectangle. Les cellules du coin n'ont que deux voisines, et celles de la bordure sauf les coins en ont trois. Au départ toutes les cases, ou une partie d'entre elles sont allumées. Le jeu consiste à trouver sur quelles cases appuyer pour finir en ayant toutes les cases éteintes.

Par exemple, pour $L = H = 4$:



début fin Passage du tout allumé au tout éteint



Passage d'une partie allumée au tout éteint

Les questions qui se posent sont :

- Est-ce possible ou non ?
- Si c'est possible, comment faire, et combien y a-t-il de solutions ?

2) Système d'équations associé au jeu

Numérotons 0, 1, 2, 3,…, N^2–1, les cellules de gauche à droite et de haut en bas. Selon son état, chaque cellule porte un nombre : 0 lorsqu'elle est éteinte, 1 lorsqu'elle est allumée. Si une cellule est touchée, elle change d'état ainsi que ses voisines. Si elle est touchée deux fois, c'est comme s'il ne s'était rien passé. Le fait de toucher un nombre pair de fois une même cellule revient à ne pas la toucher du tout, et la toucher un nombre impair de fois revient à la toucher une seule fois.

Numérotation des cellules pour $L = H = 2$

Commençons par un cas simple, avec $L = H = 2$. Associons à chaque cellule i un vecteur $\mathbf{V}_i$ où la cellule et ses voisins sont à 1, et le reste à 0, soit

$$\boldsymbol{V}_0 = \begin{pmatrix}1\\1\\1\\0\end{pmatrix} \quad \boldsymbol{V}_1 = \begin{pmatrix}1\\1\\0\\1\end{pmatrix} \quad \boldsymbol{V}_2 = \begin{pmatrix}1\\0\\1\\1\end{pmatrix} \quad \boldsymbol{V}_3 = \begin{pmatrix}0\\1\\1\\1\end{pmatrix}$$

Maintenant, donnons-nous une configuration initiale, par exemple :

Cette configuration initiale est définie par le vecteur $\mathbf{B} = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}$

Puis entrons dans le jeu. Chaque fois que nous appuyons sur une touche, la configuration change, et elle s'écrit toujours sous forme de vecteur. Procédons à l'évolution suivante :

- Appuyons sur la case 0. On obtient la configuration $\begin{pmatrix}0\\1\\1\\0\end{pmatrix}$. Elle est obtenue en additionnant

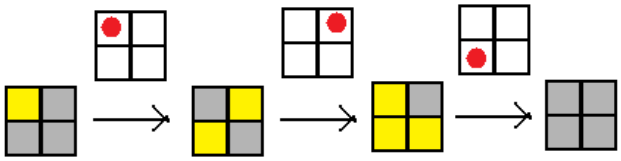
$\mathbf{B}$ et $\mathbf{V_0}$ modulo 2 (avec notamment 1 + 1 = 0) : $\begin{pmatrix}1\\0\\0\\0\end{pmatrix}+\begin{pmatrix}1\\1\\1\\0\end{pmatrix}=\begin{pmatrix}0\\1\\1\\0\end{pmatrix}$

- Appuyons sur la case 1. On obtient la configuration $\begin{pmatrix}1\\0\\1\\1\end{pmatrix}$, obtenue en additionnant

$\mathbf{(B + V_0)}$ et $\mathbf{V_1}$ modulo 2 :

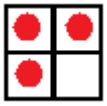
- Appuyons sur la case 2. On obtient la configuration $\mathrm{B} + M\begin{pmatrix}x_0\\x_1\\x_2\\x_3\end{pmatrix}=\begin{pmatrix}0\\0\\0\\0\end{pmatrix}$, obtenue en additionnant

$\mathbf{(B + V_0 + V_1)}$ et $\mathbf{V_2}$ modulo 2 : $\begin{pmatrix}1\\0\\1\\1\end{pmatrix}+\begin{pmatrix}1\\0\\1\\1\end{pmatrix}=\begin{pmatrix}0\\0\\0\\0\end{pmatrix}$. C'est gagné, tout est éteint.



Evolution du jeu

Le fait de toucher les cases 0, 1 et 2 est une solution pour ce jeu. Remarquons que l'ordre dans lequel on touche ces cases est sans importance, puisque seul joue le nombre de fois où elles sont touchées.



Une solution du jeu

Appelons alors x_k le nombre de fois où la cellule k a été touchée, ramené modulo 2 : x_k vaut soit 0 soit 1. Dans l'exemple précédent, nous avons résolu l'équation vectorielle (où $\mathbf{0}$ est le vecteur nul) :

$\mathbf{B} + x_0\,\mathbf{V_0} + x_1\,\mathbf{V_1} + x_2\,\mathbf{V_2} + x_3\,\mathbf{V_3} = \mathbf{0}$,

et nous avons trouvé une solution qui est $(x_0, x_1, x_2, x_3) = (1, 1, 1, 0)$, tous les calculs étant faits dans le champ $\mathbf{Z_2}$ des nombres en binaire modulo 2. En prenant la matrice M ayant pour vecteurs colonnes les vecteurs $\mathbf{V}_i$, cela s'écrit aussi :

$$B + M\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \text{ avec } M = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}, \text{ ou en résumé } \mathbf{B} + M\,\mathbf{X} = \mathbf{0} \text{ avec } \mathbf{X} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

Ce que nous avons fait sur un exemple se généralise. Il s'agit de résoudre le système d'équations :

$\mathbf{B} + M\,\mathbf{X} = \mathbf{0}$ où les vecteurs ont N coordonnées, la matrice M étant de N sur N. Ajoutons $\mathbf{B}$ des deux côtés. Comme $2\mathbf{B} = 0$, on a le système équivalent :

$M\,\mathbf{X} = \mathbf{B}$.

Par exemple avec $L = H = 2$, soit $N = 4$, si l'on part du tout allumé, le système à résoudre s'écrit[1] :

$$\begin{aligned} x_0 + x_1 + x_2 &= 1 \\ x_0 + x_1 + x_3 &= 1 \\ x_0 + x_2 + x_3 &= 1 \\ x_1 + x_2 + x_3 &= 1 \end{aligned}$$

Dans le jeu du tout éteint, la configuration finale est le vecteur $\mathbf{0}$. Mais si l'on veut arriver à une configuration finale quelconque $\mathbf{C}$, il convient de résoudre $\mathbf{B} + M\,\mathbf{X} = \mathbf{C}$.

3) Résolution du système d'équations sur ordinateur

La résolution d'un système linéaire de N équations à N inconnues se fait par la méthode du pivot de Gauss. Commençons par traiter l'exemple simple précédent, avec $L = H = 2$ et le vecteur $\mathbf{B} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

$$\begin{cases} x_0 + x_1 + x_2 = 1 \\ x_0 + x_1 + x_3 = 0 \\ x_0 + x_2 + x_3 = 0 \\ x_1 + x_2 + x_3 = 0 \end{cases} \text{ ou sous forme simplifiée } \left(\begin{array}{cccc|c} 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{array}\right)$$

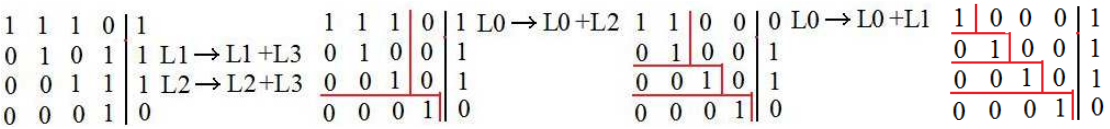
Le pivot de la Ligne 0 étant 1 au début, on ajoute cette équation à celles en dessous ayant aussi un 1 dans la même colonne 0, de façon à obtenir un premier échelon. On recommence sur la matrice des colonnes 1 à 3, mais comme la Ligne 1 commence par un 0, il convient de l'échanger avec la ligne suivante qui commence par 1. Avec ce nouveau pivot 1, on ajoute la Ligne 1 à celles du dessous qui commencent aussi par 1, de façon à obtenir un nouvel échelon. Et ainsi de suite. On constate que l'on a à la fin un système triangulaire. On est assuré qu'il existe une solution unique.

$$\left(\begin{array}{cccc|c} \textcircled{1} & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{array}\right) \begin{array}{l} \\ L1 \to L1+L0 \\ L2 \to L2+L0 \\ \\ \end{array} \left(\begin{array}{cccc|c} 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{array}\right) L1 \leftrightarrow L2 \left(\begin{array}{cccc|c} 1 & 1 & 1 & 0 & 1 \\ 0 & \textcircled{1} & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{array}\right) \begin{array}{l} \\ \\ \\ L3 \to L3+L1 \end{array}$$

$$\left(\begin{array}{cccc|c} 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & \textcircled{1} & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{array}\right) \begin{array}{l} \\ \\ \\ L3 \to L3+L2 \end{array} \left(\begin{array}{cccc|c} 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right)$$

[1] On peut vérifier qu'il existe dans ce cas une solution unique, à savoir la solution : $x_1 = x_2 = x_3 = x_4 = 1$. Il suffit de toucher chaque case une fois.

Pour la trouver, on va faire une remontée qui va aboutir à l'obtention d'une matrice diagonale, et même ici la matrice identité :



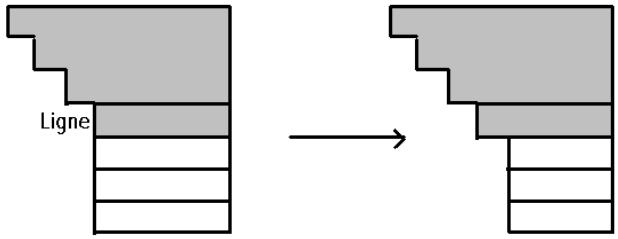
Le système est devenu $x_0 = 1$, $x_1 = 1$, $x_2 = 1$, $x_3 = 0$. C'est la solution unique du problème.

On va programmer la résolution d'un tel système dans le cas général, en utilisant la méthode du pivot de Gauss, pour ce système de *N* équations à *N* inconnues (*en cas de besoin, voir cours mathématiques et informatique chapitre 6*). On commence par entrer la configuration initiale, ainsi que la matrice *M* du système, grâce aux fonctions suivantes :

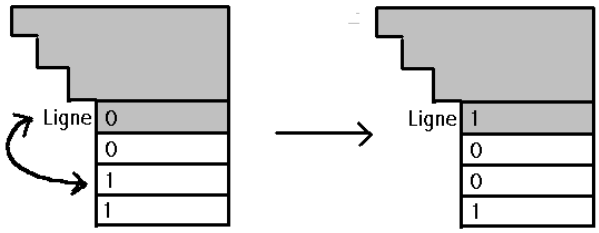
```
/**********    CONFIGURATION INITIALE    **********/
void configurationinitiale(void)
  { int i;
    for(i=0;i<N;i++)  b[i]=1;
    b[3]=0; b[5]=0;
  }
/**********      CONSTRUCTION DE LA MATRICE M  DU SYSTEME        ************/
void matriceinitiale(void)
{int i,ligne,colonne;
for(i=0;i<N;i++)
 { M[i][i]=1;
   if (i%L!=0) M[i][i-1]=1;    if ((i+1)%L!=0) M[i][i+1]=1;
   if (i>=L) M[i][i-L]=1;      if (i<N-L) M[i][i+L]=1;
 }
printf("\n"); /* affichage de la matrice */
for(ligne=0;ligne<N;ligne++)
 { printf("\n");
   for(colonne=0;colonne<N;colonne++) printf(" %d",M[ligne][colonne]);
   printf("  %d",b[ligne]);
 }
getchar();
}
```

La fabrication de la matrice *M* mérite quelques explications. Déclarée en global, la matrice *M* [][] a ses coefficients à 0 initialement. Chaque case *i* étant mise à 1, on commence par mettre à 1 la diagonale de la matrice. Puis on prend les voisins de la case *i*, qui peuvent être au nombre de 4, de 3 ou de 2, selon la position de la case. La case *i* admet un voisin de gauche si elle n'est pas sur la première colonne ($i\%L$ différent de 0). Elle a un voisin de droite si elle n'est pas sur la dernière colonne ($(i + 1)\%L$ différent de 0). Elle a un voisin au-dessus si elle n'est pas sur la première ligne ($i \geq L$), et un voisin au-dessous si elle n'est pas sur la dernière ligne ($i < N - L$). Ces cases voisines sont mises à 1.

Puis on appliquer la méthode du pivot de Gauss. A chaque grande étape, on a une ligne, notée *Ligne*, telle qu'au dessus le système est déjà triangulaire. On prendra ainsi chaque valeur de *Ligne*, de la *Ligne* 0 à l'avant-dernière. A chaque fois il s'agit de transformer les lignes qui sont au-dessous pour leur enlever leur colonne de gauche :



Le premier coefficient de la ligne *Ligne* est *a*[*Ligne*][*Ligne*]. Dans un premier temps, il convient de s'assurer qu'il n'est pas nul, et dans ce cas c'est lui qui va jouer le rôle de pivot. Sinon, si ce coefficient est nul, il faut échanger cette *Ligne* avec une ligne située au-dessous dont le premier coefficient *a*[*ligne*][*Ligne*] est égal à 1. A condition qu'il en existe une.



D'où la fonction suivante :

```
/***************        ECHANGE ENTRE DEUX LIGNES       *****************/
int echangelignes(int Ligne)
{ int flag,ligne,i;
  flag=0;
  for(ligne=Ligne+1; ligne<N;ligne++) if (M[ligne][Ligne]!=0)
    { for(i=Ligne;i<N;i++) aux[i]=M[Ligne][i];
      for(i=Ligne;i<N;i++) M[Ligne][i]=M[ligne][i];
      for(i=Ligne;i<N;i++) M[ligne][i]=aux[i];
      auxb=b[Ligne];b[Ligne]=b[ligne]; b[ligne]=auxb;
      flag=1; break;
    }
 return flag;
}
```

Cette fonction ramène 1 lorsque l'échange a pu avoir lieu et 0 sinon, ce qui signifie alors que toute la colonne concernée est formée de zéros.

Si tout va bien on procède à la modification des lignes qui suivent la *Ligne* où l'on est, avec la fonction :

```
/***************        MODIFICATION DES LIGNES          *****************/
void changementdeslignessuivantes(int Ligne)
{ int ligne,colonne;
  for(ligne=Ligne+1;ligne<N;ligne++)  if (M[ligne][Ligne]!=0)
    { for(colonne=Ligne;colonne<N;colonne++)
       M[ligne][colonne]=(M[ligne][colonne]+M[Ligne][colonne])%2;
       b[ligne]=(b[ligne]+b[Ligne])%2;
    }
}
```

Si le procédé continue jusqu'à l'avant-dernière ligne, sans que l'on tombe sur une colonne de zéros, le système devient parfaitement triangulaire, et l'on a une solution unique. Pour l'obtenir, on est amené à faire une nouvelle triangularisation de bas en haut, ce qui va donner une matrice diagonale, avec des 0 partout sauf sur la diagonale avec des 1. Ce qui est fait par cette fonction :

```
/********      CALCUL DE LA SOLUTION QUAND ELLE EST UNIQUE       ********/
void solutionunique(void)
{int ligne,Ligne,colonne,i;
 for(Ligne=N-1;Ligne>0;Ligne--)
   {
    if (M[Ligne][Ligne]==0)
    for(ligne=Ligne-1; ligne>=0;ligne--)  if (M[ligne][Ligne]!=0)
      {
       for(i=Ligne;i>=0;i--) aux[i]=M[Ligne][i];
       for(i=Ligne;i>=0;i--) M[Ligne][i]=M[ligne][i];
       for(i=Ligne;i>=0;i--) M[ligne][i]=aux[i];
       auxb=b[Ligne];b[Ligne]=b[ligne]; b[ligne]=auxb;
       break;
      }
    for(ligne=Ligne-1;ligne>=0;ligne--) if (M[ligne][Ligne]!=0)
      {
       for(colonne=Ligne;colonne>=0;colonne--)
       M[ligne][colonne]=(M[ligne][colonne]+M[Ligne][colonne])%2;
       b[ligne]=(b[ligne]+b[Ligne])%2;
      }
    matrice(Ligne);
   }
 for(i=0;i<N;i++) printf("\nx%d=%d",i,b[i]);
}
```

Avec la fonction annexe qui affiche, si on le désire, la matrice à chaque étape de son évolution :

```
/***************       AFFICHAGE DE LA MATRICE       ******************/
void matrice(int Ligne)
{ int ligne,colonne;
   printf(" ***  Ligne=%d",Ligne); /**printf("\n");*/
   for(ligne=0;ligne<N;ligne++)
     { printf("\n");
       for(colonne=0;colonne<N;colonne++) printf(" %d",M[ligne][colonne]);
       printf("  %d",b[ligne]);
     }
getchar();
}
```

Maintenant traitons le cas où le processus se bloque lorsque l'on tombe sur une colonne de 0. Cela se produit si la fonction d'échange de lignes ramène 0 (dans *flag*). Il se trouve que dans le contexte du problème, toutes les colonnes qui suivent (sauf éventuellement celles des *b*[]) sont aussi toutes à 0. Cela peut être testé. Dans le programme principal nous plaçons une variable *fflag* qui vaut 0 si toutes les colonnes à partir de celles où l'on est ne contiennent que des 0, et *fflag* à 1 sinon, ce dernier cas ne se produisant pas (s'il se produisait on afficherait PROBLEME ![2]).

Dans le cas où les dernières lignes du système, au nombre de $N - Ligne$, n'ont que des coefficients 0, à gauche du signe =, mais qu'il y a au moins un 1 dans la colonne des *b*[], il n'y a aucune solution au problème, et cela est affiché.

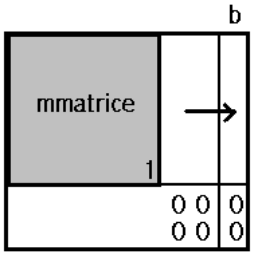
Plaçons-nous maintenant dans le cas où les dernières lignes du système, au nombre de $N - Ligne$, n'ont que des coefficients 0, aussi bien à gauche qu'à droite du signe =. Les $N - Ligne$ dernières

[2] Il y aurait problème si l'on avait par exemple un système de la forme

```
1 1 1 0   1
0 1 1 0   0
0 0 0 1   1  ← Ligne
0 0 0 1   0
```

Il conviendrait alors de procéder à un échange des colonnes 2 et 3, ce que notre programme ne fait pas.

inconnues du système sont arbitraires, elles valent soit 0 soit 1, ce qui donne $2^{N-Ligne}$ solutions au système. Toutes ces solutions sont obtenues en fabriquant les nombres en binaire de longueur *N – Ligne*. Chacune de ces solutions arbitraires, correspondant aux inconnues non principales, est envoyée à droite dans les *Ligne* – 1 premières équations, d'où de nouvelles valeurs pour *b*[]. Puis on détermine les valeurs des inconnues principales dans la partie *mmatrice* de la matrice concernée, en procédant à sa diagonalisation.



Voici la fonction qui donne toutes les solutions l'une après l'autre :

```
/*************       AFFICHAGE DES SOLUTIONS MULTIPLES      *************/
void solutions(int LL, int nombre)
{int nombresolutions,i,q,j,ligne,colonne,Ligne,cumul,x[N],n;
  nombresolutions=pow(2,nombre);
 for(n=0;n<nombresolutions;n++)
  {
    for(i=0;i<N;i++) for(j=0;j<N;j++) M[i][j]=MM[i][j];
    for(i=0;i<N;i++) b[i]=bb[i];
    q=n;
    for(i=0;i<nombre;i++) { x[N-1-i]=q%2; q=q/2;}
    for(ligne=0;ligne<LL;ligne++)
      { cumul=0;
        for(j=0;j<nombre;j++)  cumul+=x[N-1-j]*M[ligne][N-1-j];
        b[ligne]=(b[ligne]+cumul)%2;
      }
    printf("\n");
    for(j=0;j<nombre;j++) printf(" x%d=%d ",N-nombre+j,x[N-nombre+j]);
    mmatrice(nombre);
    for(Ligne=LL-1;Ligne>0;Ligne--)
      { if (M[Ligne][Ligne]==0)
        for(ligne=Ligne-1; ligne>=0;ligne--)
        if (M[ligne][Ligne]!=0)
          {
           for(i=Ligne;i>=0;i--) aux[i]=M[Ligne][i];
           for(i=Ligne;i>=0;i--) M[Ligne][i]=M[ligne][i];
           for(i=Ligne;i>=0;i--) M[ligne][i]=aux[i];
           auxb=b[Ligne];b[Ligne]=b[ligne]; b[ligne]=auxb;
           break;
          }
        for(ligne=Ligne-1;ligne>=0;ligne--)
        if (M[ligne][Ligne]!=0)
          {
           for(colonne=Ligne;colonne>=0;colonne--)
           M[ligne][colonne]=(M[ligne][colonne]+M[Ligne][colonne])%2;
           b[ligne]=(b[ligne]+b[Ligne])%2;
          }
       printf(" *** Ligne=%d",Ligne);  mmatrice(nombre);
      }
    printf("\n Solution %d\n",n+1);
    for(i=0;i<N-nombre;i++) printf(" x%d=%d",i,b[i]);
    for(j=0;j<nombre;j++) printf(" x%d=%d",N-nombre+j,x[N-nombre+j]);
    getchar();
```

```
  }
printf("FINI"); getchar();
}
```

Avec l'affichage de la matrice carrée principale *mmatrice* :

```
/***********       AFFICHAGE DE LA PARTIE DE LA MATRICE       ***********/
void mmatrice(int n)
{
int ligne,colonne;
/**printf("\n");*/
for(ligne=0;ligne<N-n;ligne++)
 { printf("\n");
   for(colonne=0;colonne<N-n;colonne++) printf(" %d",M[ligne][colonne]);
   printf("  %d",b[ligne]);
 }
getchar();
}
```

A noter que lorsqu'une solution est trouvée avec la diagonalisation de la matrice, il faut revenir à la matrice triangulaire initiale pour chercher la solution suivante, d'où le tableau *MM*[][] qui conserve ses valeurs.

Il reste le programme principal, où l'on prend chaque *Ligne* l'une après l'autre :

```
int main()
{ int ligne,colonne,i,j; int Ligne,flag,dernieligne,fflag;
   configurationinitiale();
   matriceinitiale();
   for(Ligne=0;Ligne<N-1;Ligne++)
     { if (M[Ligne][Ligne]==0)  /* 0 en début de ligne */
         { flag=echangelignes(Ligne);
           if (flag==0)  /* on n'a trouvé aucun 1 dans les lignes suivantes */
            { matrice(Ligne);    fflag=0;
              for(ligne=Ligne;ligne<N;ligne++)  for(colonne=Ligne+1;colonne<N;colonne++)
              if (M[ligne][colonne]==1) {fflag=1; break;}
              for(ligne=Ligne;ligne<N;ligne++)  if (b[ligne]==1)
                 { printf(" PAS DE SOLUTIONS");
                   getchar();  exit(0);   /* pas de solutions, c'est terminé */
                 }
              if (fflag==0)
                { printf("\n\nNombre de solutions=%d    fini",(int)pow(2,N-Ligne));getchar();
                  dernieligne=Ligne;
                  for(i=0;i<N;i++) for(j=0;j<N;j++)MM[i][j]=M[i][j];  for(i=0;i<N;i++) bb[i]=b[i];
                 solutions(dernieligne,N-Ligne);
                  break;
                }
             if (fflag==1) {printf("*** PROBLEME ***");getchar();}
           }
         }
       changementdeslignessuivantes(Ligne);   matrice(Ligne);
       if (Ligne==N-2)
         { printf("\n Une seule solution. On va la donner");getchar();
           solutionunique(); getchar();
         }
     }
getchar(); return 0;
}
```

Voici un exemple de résultats de ce programme pour $L = 3$ et $H = 2$ ($N = 6$). *A gauche* on trouve l'évolution de la mise en échelons de la matrice, avec le blocage final sur deux lignes de 0, ce qui indique qu'il y a quatre solutions. Puis *à droite*, on cherche ces solutions une par une, et envoyant les deux inconnues non principales à droite du =, et en diagonalisant la matrice 4x4 *mmatrice*.

```
1 1 0 1 0 0  1
1 1 1 0 1 0  1
0 1 1 0 0 1  1
1 0 0 1 1 0  0
0 1 0 1 1 1  1
0 0 1 0 1 1  0
***  Ligne=0
1 1 0 1 0 0  1
0 0 1 1 1 0  0
0 1 1 0 0 1  1
0 1 0 0 1 0  1
0 1 0 1 1 1  1
0 0 1 0 1 1  0
***  Ligne=1
1 1 0 1 0 0  1
0 1 1 0 0 1  1
0 0 1 1 1 0  0
0 0 1 0 1 1  0
0 0 1 1 1 0  0
0 0 1 0 1 1  0
***  Ligne=2
1 1 0 1 0 0  1
0 1 1 0 0 1  1
0 0 1 1 1 0  0
0 0 0 1 0 1  0
0 0 0 0 0 0  0
0 0 0 1 0 1  0
***  Ligne=3
1 1 0 1 0 0  1
0 1 1 0 0 1  1
0 0 1 1 1 0  0
0 0 0 1 0 1  0
0 0 0 0 0 0  0
0 0 0 0 0 0  0
***  Ligne=4
1 1 0 1 0 0  1
0 1 1 0 0 1  1
0 0 1 1 1 0  0
0 0 0 1 0 1  0
0 0 0 0 0 0  0
0 0 0 0 0 0  0

Nombre de solutions=4
```

```
x4=0  x5=0
1 1 0 1  1
0 1 1 0  1
0 0 1 1  0
0 0 0 1  0
*** Ligne=3
1 1 0 0  1
0 1 1 0  1
0 0 1 0  0
0 0 0 1  0
*** Ligne=2
1 1 0 0  1
0 1 0 0  1
0 0 1 0  0
0 0 0 1  0
*** Ligne=1
1 0 0 0  0
0 1 0 0  1
0 0 1 0  0
0 0 0 1  0

Solution 1
x0=0 x1=1 x2=0 x3=0 x4=0 x5=0
```

```
x4=1  x5=0
1 1 0 1  1
0 1 1 0  1
0 0 1 1  1
0 0 0 1  0
*** Ligne=3
1 1 0 0  1
0 1 1 0  1
0 0 1 0  1
0 0 0 1  0
*** Ligne=2
1 1 0 0  1
0 1 0 0  0
0 0 1 0  1
0 0 0 1  0
*** Ligne=1
1 0 0 0  1
0 1 0 0  0
0 0 1 0  1
0 0 0 1  0

Solution 3
x0=1 x1=0 x2=1 x3=0 x4=1 x5=0
```

```
x4=0  x5=1
1 1 0 1  1
0 1 1 0  0
0 0 1 1  0
0 0 0 1  1
*** Ligne=3
1 1 0 0  0
0 1 1 0  0
0 0 1 0  1
0 0 0 1  1
*** Ligne=2
1 1 0 0  0
0 1 0 0  1
0 0 1 0  1
0 0 0 1  1
*** Ligne=1
1 0 0 0  1
0 1 0 0  1
0 0 1 0  1
0 0 0 1  1

Solution 2
x0=1 x1=1 x2=1 x3=1 x4=0 x5=1
```

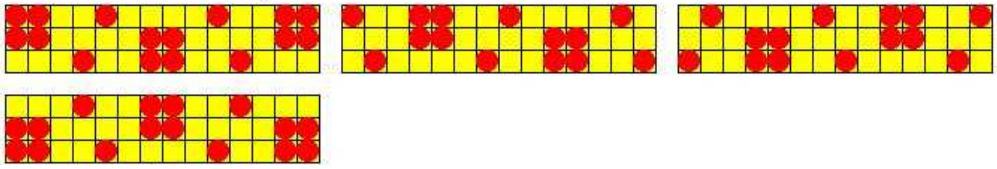
```
x4=1  x5=1
1 1 0 1  1
0 1 1 0  0
0 0 1 1  1
0 0 0 1  1
*** Ligne=3
1 1 0 0  0
0 1 1 0  0
0 0 1 0  0
0 0 0 1  1
*** Ligne=2
1 1 0 0  0
0 1 0 0  0
0 0 1 0  0
0 0 0 1  1
*** Ligne=1
1 0 0 0  0
0 1 0 0  0
0 0 1 0  0
0 0 0 1  1

Solution 4
x0=0 x1=0 x2=0 x3=1 x4=1 x5=1
```

4) Quelques résultats lorsque tout est allumé au départ

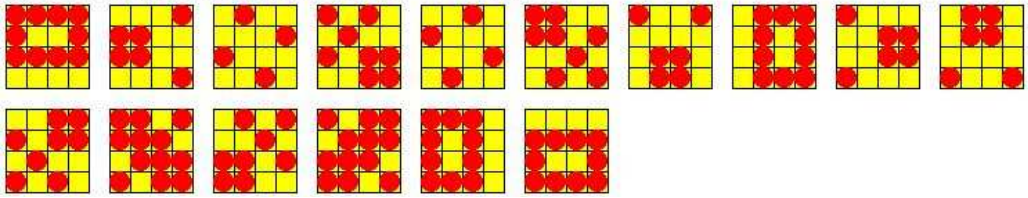
On constate qu'il y a toujours au moins une solution. Voici quelques exemples, à partir du programme précédent auquel on ajoute le dessin des solutions.

a) Nombre de solutions pour L = 14 et H = 3 : 4



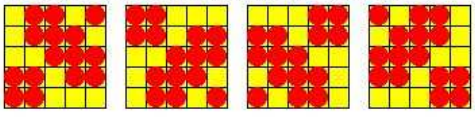
(deux solutions aux symétries près du rectangle)

b) Nombre de solutions pour L = 4 et H = 4 : 16



(5 solutions aux symétries près du carré)

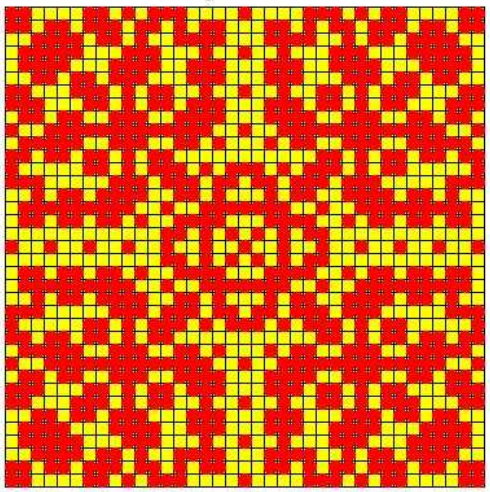
c) Nombre de solutions pour L = 5 et H = 5 : 4



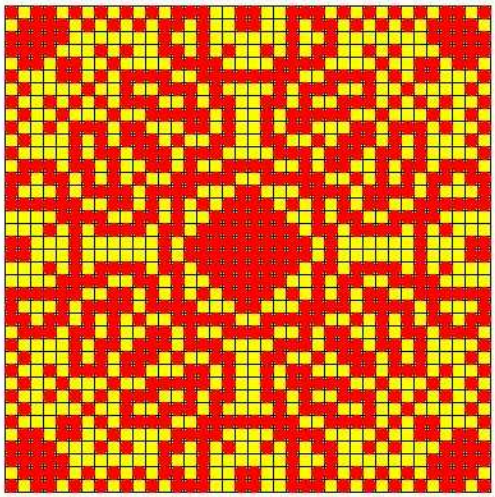
(une seule solution aux symétries près du carré)

d)

Une seule solution pour L = 37 et H = 37



Une seule solution pour L = 38 et H = 38



e) Nombre de solutions pour une grille carrée *L* sur *L* pour le jeu *tout allumé vers tout éteint*

```
L =  2   1
L =  3   1
L =  4    16
L =  5     4
L =  6   1
L =  7   1
L =  8   1
L =  9   256
L = 10   1
L = 11    64
L = 12   1
L = 13   1
L = 14    16
L = 15   1
L = 16   256
L = 17     4
L = 18   1
L = 19   65536
L = 20   1
L = 21   1
L = 22   1
L = 23   16384
L = 24    16
L = 25   1
L = 26   1
L = 27   1
L = 28   1
L = 29   1024
L = 30   1048576
L = 31   1
L = 32   1048576
L = 33   65536
L = 34    16
L = 35    64
L = 36   1
L = 37   1
L = 38   1
L = 39   4294967296
```

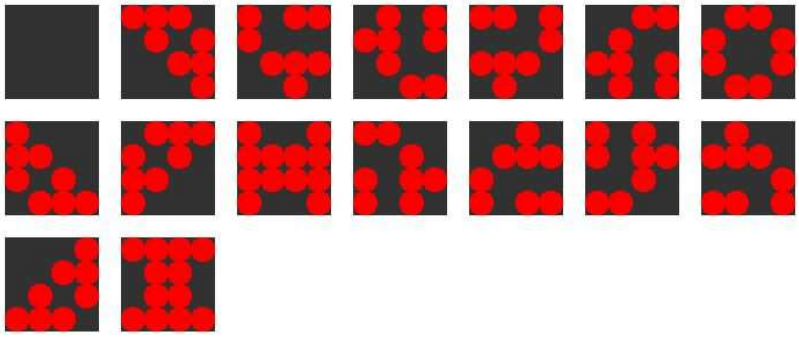
5) Précisions théoriques

a) Configurations neutres et noyau de la matrice *M*

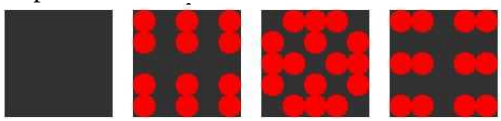
Lorsque l'on part d'une configuration **B** et que l'on appuie sur les cases correspondant à un vecteur **X**, on obtient à la fin la configuration **B** + *M* **X**, comme on l'a vu au 1°. Supposons que l'on veuille arriver à **B**, ce qui veut dire que l'on a une configuration qui est la même à la fin qu'au début, cela impose pour **X** de vérifier **B** + *M* **X** = **B**, ou encore *M* **X** = 0. Les vecteurs **X** solutions de ce système sont ce que l'on peut appeler les configurations neutres puisqu'en appuyant sur leurs cases il ne se produit finalement aucun changement. Vérifiant *M* **X** = 0, elles forment ce que l'on appelle le noyau de la matrice *M* (ou son espace nul). Pour les obtenir, il suffit de reprendre le programme précédent en mettant tous les *b*[] à 0.

Exemples :

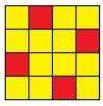
- pour un carré de 4 sur 4, on obtient ces 16 solutions :

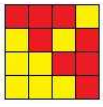


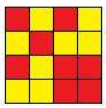
- pour un carré de 5 sur 5, on a ces 4 solutions :



Restons sur ce dernier exemple, en appelant les configurations neutres obtenues 0, $\mathbf{d}_1$, $\mathbf{d}_2$, $\mathbf{d}_3$. Elles sont toutes engendrées par les combinaisons de deux d'entre elles, soit $a\ \mathbf{d}_1 + b\ \mathbf{d}_3$,[3] avec *a* et *b* valant 0 ou 1. Notamment on vérifie que $\mathbf{d}_1 + \mathbf{d}_3 = \mathbf{d}_2$: en appuyant sur les touches rouges de $\mathbf{d}_1$ puis sur celles de $\mathbf{d}_3$, on obtient évidemment une configuration invariante, et l'on constate que c'est $\mathbf{d}_2$. Le noyau de *M* est de dimension 2, et il donne naissance par combinaisons à $2^2 = 4$ configurations. Pour les mêmes raisons la dimension du noyau de *M* pour le carré 4x4 est 4 puisque l'on obtient par combinaisons $2^4 = 16$ configurations.

Prenons maintenant une solution du problème *tout allumé vers tout éteint*, par exemple pour le carré 4x4 la solution 

Ajoutons lui une configuration invariante, soit 

Cela revient à appuyer sur les touches rouges de ces deux configurations successivement, ce qui donne la troisième configuration 

[3] Si l'on fait le produit scalaire des deux vecteurs $\mathbf{d}_1$ et $\mathbf{d}_3$, on obtient une somme avec un nombre pair de 1, ce qui donne 0. Les deux vecteurs sont orthogonaux et forment une base orthogonale du noyau de *M*.

A la fin on arrive encore au tout éteint, grâce à la première configuration, la deuxième ne changeant rien. On trouve encore une solution du problème *tout allumé vers tout éteint*. Ainsi, en ajoutant à une solution du problème les 16 configurations invariantes on trouve toutes les solutions du problème *tout allumé vers tout éteint*.

Dans certains cas, le noyau de M se réduit au vecteur nul, qui est la configuration neutre évidente. Cela signifie que la matrice M est inversible, de déterminant différent de 0 (on est dans le champ $\mathbf{Z}_2$). La méthode du pivot de Gauss aboutit à une diagonalisation de la matrice qui devient la matrice identité, avec des 1 sur sa diagonale, et des 0 ailleurs. Il existe une solution unique au problème, quel que soit $\mathbf{B}$ dans $M\,\mathbf{X} = \mathbf{B}$. Autrement dit, toute configuration initiale permet d'aboutir au tout éteint, et il n'existe qu'une façon d'y arriver (à condition de ne pas appuyer plusieurs fois sur la même case). Mais ce n'est plus le cas lorsque le noyau de M n'est pas réduit à $\mathbf{0}$.

b) Configurations gagnantes et espace image des configurations

Sur une surface de $N = L\,H$ cases, il existe 2^N configurations initiales $\mathbf{B}$ possibles (puisque l'on choisit soit 0 soit 1 dans chaque case). Mais combien parmi elles sont-elles gagnantes, c'est-à-dire permettant d'arriver au tout éteint ? Pour les trouver il suffit de prendre $M\,\mathbf{X}$ avec tous les vecteurs $\mathbf{X}$ possibles, au nombre de 2^N aussi. Les vecteurs $\mathbf{B} = M\,\mathbf{X}$ obtenus sont les configurations gagnantes. Ces vecteurs constituent ce que l'on appelle l'espace image de M.

Il existe un lien bien connu entre l'espace image et le noyau de M : la somme de leurs dimensions est égale à N, dimension de l'espace vectoriel dans lequel on se trouve.

Par exemple, dans le carré 5x5, où le noyau de M est de dimension 2, l'espace image est de dimensions $25 - 2 = 23$. Cela signifie qu'il existe 2^{23} configurations gagnantes, parmi les 2^{25} configurations initiales possibles. Ainsi, un quart des configurations initiales sont gagnantes. Pour les mêmes raisons, dans le carré 4x4, un seizième des configurations initiales sont gagnantes. Et lorsque le noyau de M est réduit à $\mathbf{0}$, de dimension 0, on retrouve le fait que toutes les configurations initiales sont gagnantes.

Dans le cas où le noyau de M n'est pas $\mathbf{0}$, comment savoir si une configuration initiale est gagnante ou pas ? La résolution du système par la méthode de Gauss permet de répondre. Mais il existe un autre moyen de le savoir, sans résoudre le système. On sait que la matrice M est symétrique, et l'on dispose des propriétés de ce type de matrice dans un champ comme $\mathbf{Z}_2$.[4] Ainsi, une telle matrice est toujours diagonalisable. C'est d'ailleurs ce que nous avons constaté par la méthode de Gauss, la diagonale étant formée de 1 et éventuellement de quelques 0 (lorsque le noyau n'est pas nul).[5]

D'autre part, on sait que pour une matrice symétrique, l'espace image est le complément orthogonal du noyau.[6] Cela signifie que tout vecteur de l'espace image est orthogonal à chaque vecteur de la base du noyau. Autrement dit, le produit scalaire de toute configuration gagnante avec chaque vecteur de base du noyau est nul. Et comme le produit scalaire s'obtient en additionnant les produit des coordonnées respectives des deux vecteurs, il est nul si et seulement si les cases allumées (valant 1) de la configuration initiale qui sont en commun avec les cases de chaque configuration neutre est un nombre pair.

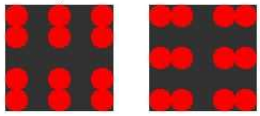
---

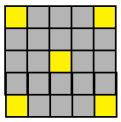
[4] Dans les cours d'algèbre, les propriétés des matrices symétriques sont étudiées dans le champ (ou corps) $\mathbf{R}$ des nombres réels, mais cela vaut aussi dans tout champ.

[5] Les valeurs propres sont 1 ou 0, les vecteurs propres associés aux valeurs propres 0 forment la base du noyau, et ceux de valeur propre 1 forment la base de l'espace image.

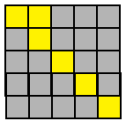
[6] C'est une conséquence de cette propriété des matrices symétriques à coefficients dans un champ : les sous-espaces propres associés à deux valeurs propres distinctes sont orthogonaux.

Exemple du carré 5x5, avec ses deux configurations neutres de base $\mathbf{d}_1$ et $\mathbf{d}_3$:



- Configuration initiale : 

Elle a quatre cases (allumées) en commun, celles des coins, avec chacune des deux configurations neutres. C'est une configuration gagnante.

- Configuration initiale : 

Elle a deux cases en commun avec $\mathbf{d}_1$ et trois en commun avec $\mathbf{d}_3$, ce n'est pas une configuration gagnante. On n'arrivera jamais au tout éteint.

c) Cas particulier du jeu *tout allumé vers tout éteint*

On a cette propriété :

La configuration initiale *tout allumé* est toujours gagnante, quelles que soient les dimensions du rectangle *LxH* des cases.

Démonstration

La matrice *M* est non seulement symétrique, mais elle n'a que des 1 sur sa diagonale. Montrons d'abord que $\mathbf{X}^T M \mathbf{X} = \mathbf{X}^T \mathbf{X}$, $\mathbf{X}$ étant un vecteur colonne quelconque, et $\mathbf{X}^T$ son transposé (vecteur ligne).

Commençons par prendre une matrice symétrique *S* avec seulement deux éléments symétriques non nuls, tous les autres éléments étant nuls (notamment ceux de la diagonale). On vérifie aussitôt que $\mathbf{X}^T S \mathbf{X} = 0$. Cela se généralise à toute matrice symétrique *S* dont tous les éléments symétriques de part et d'autre de la diagonale sont non nuls ou nuls, et où ceux de la diagonale sont tous nuls. Là encore, on a $\mathbf{X}^T S \mathbf{X} = 0$. La matrice *M* est de la forme $I + S$, *I* étant la matrice identité (des 1 sur sa diagonale uniquement), d'où $\mathbf{X}^T M \mathbf{X} = \mathbf{X}^T (I + S) \mathbf{X} = \mathbf{X}^T I \mathbf{X} = \mathbf{X}^T \mathbf{X}$.

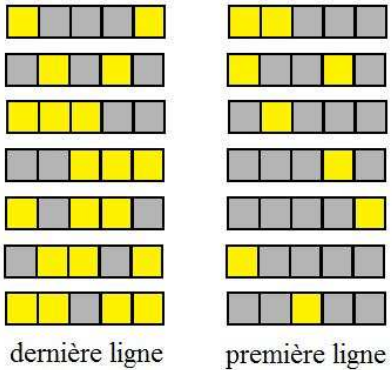
Choisissons maintenant $\mathbf{X}$ comme étant une configuration neutre, soit $M\mathbf{X} = \mathbf{0}$. Alors $\mathbf{X}^T M \mathbf{X} = 0$, d'où $\mathbf{X}^T \mathbf{X} = 0$. Cela prouve que toute configuration neutre possède un nombre pair de cases (à 1). La configuration initiale *tout allumé* a donc un nombre pair de cases en commun avec chaque configuration neutre. C'est une configuration gagnante.

A ce stade, nous savons résoudre le problème du tout éteint dans tous les cas possibles, du moins si l'on dispose d'un ordinateur qui fait les calculs. Mais comment faire si l'on joue manuellement à ce jeu ? Nous allons donner un algorithme qui permet de s'en sortir si l'on dispose d'un carré 5 sur 5.

6) Algorithme de l'extinction en descente[7] dans le carré 5x5

Partons d'une configuration initiale, avec certaines cases allumées. Commençons par mettre la ligne 0 en noir en utilisant la ligne 1 suivante. Il suffit pour cela d'appuyer sur la case en dessous de chaque case lumineuse. Quand la ligne 0 est en noir, on fait de même pour la ligne 1 en s'aidant de la ligne 2. Et l'on continue ainsi jusqu'à ce que l'avant-dernière ligne soit noire, en s'aidant de la dernière ligne. Mais cette dernière n'est pas noire, sauf exception. Utilisons alors les résultats suivants (nous ne les démontrons pas ici) :

- Si la dernière ligne est un de ces 7 cas (*colonne de gauche ci-dessous*), alors on reprend la première ligne (ligne 0) et on allume les cases indiquées sur le dessin correspondant à chacun des 7 cas dans la *colonne de droite* ci-dessous. Puis on recommence le processus d'extinction ligne par ligne. A la fin, tout est éteint.



- Si la dernière ligne n'est pas un des 7 cas, le problème n'a pas de solution, la configuration initiale n'est pas gagnante.

Annexe : Jeu sur l'écran d'ordinateur

Pour les amateurs du jeu, voici un programme (en langage C-SDL) qui permet de jouer dans un carré de dimensions quelconque, en cliquant sur les cases dessinées sur l'écran pour les faire changer d'état. La configuration initiale est prise au hasard, mais on peut modifier cette situation si on le désire.

```
/******   Jeu du tout éteint à partir d'une configuration prise au
                        hasard dans un carré de N sur N                    ********/
#include <SDL/SDL.h>
#include <time.h>
#define N 5   /** N est la longueur du côté du carré */
#define pas 30  /** cases de dimensions pas sur pas */
#define xorig 50
#define yorig 50
#define OUI 1
#define NON 0
void changercouleur(int n);
SDL_Surface* ecran, * lignev[N+1], *ligneh[N+1], *carre[N*N];
SDL_Rect position;SDL_Event event;
int continuer,i,xcase,ycase,numerocase,lumiere[N*N];
Uint32 blanc,jaune,noir,couleur;

int main ( int argc, char** argv )
```

[7] Cette méthode est appelée *chasing* (poursuite) dans les références en anglais)

```
{ srand(time(NULL));
  for(i=0;i<N*N;i++) lumiere[i]=rand()%2; /** configuration initiale */
   SDL_Init( SDL_INIT_VIDEO );
   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);
   jaune=SDL_MapRGB(ecran->format,255,255,0);
   SDL_FillRect(ecran,NULL, blanc);
   for(i=0;i<N+1;i++)
    {lignev[i]=SDL_CreateRGBSurface(SDL_HWSURFACE,1,N*pas,32,0,0,0,0);
     position.x=xorig+pas*i;
     position.y=yorig;
     SDL_FillRect(lignev[i],NULL, couleur);
     SDL_BlitSurface(lignev[i], 0, ecran, &position);
    }
   for(i=0;i<N+1;i++)
    {ligneh[i]=SDL_CreateRGBSurface(SDL_HWSURFACE,N*pas,1,32,0,0,0,0);
     position.x=xorig;
     position.y=yorig+pas*i;
     SDL_FillRect(ligneh[i],NULL, couleur);
     SDL_BlitSurface(ligneh[i], 0, ecran, &position);
    }

   for(i=0;i<N*N;i++)
    {carre[i]=SDL_CreateRGBSurface(SDL_HWSURFACE,pas-2,pas-2,32,0,0,0,0);
     if (lumiere[i]==OUI) couleur=jaune;
     else couleur=noir;
     position.x=xorig+pas*(i%N)+1;
     position.y=yorig+pas*(i/N)+1;
     SDL_FillRect(carre[i],NULL, couleur);
     SDL_BlitSurface(carre[i], 0, ecran, &position);
    }
   SDL_Flip(ecran);
   continuer=OUI;
   while (continuer==OUI)
    {SDL_WaitEvent(&event);
     if (event.type==SDL_QUIT) continuer=NON;
     else if(event.type==SDL_MOUSEBUTTONDOWN
          && event.button.button == SDL_BUTTON_LEFT)
       { if(event.button.x>xorig && event.button.x<xorig+pas*N
            && event.button.y>yorig && event.button.y<yorig+pas*N)
          { xcase=(event.button.x-xorig)/pas;
            ycase=(event.button.y-yorig)/pas;
            numerocase=ycase*N+xcase;
            changercouleur(numerocase);
          }
         SDL_Flip(ecran);
       }
    }
   return 0;
}

void changercouleur( int n)
{  int vd,vg,vb,vh;
   lumiere[numerocase]=(lumiere[numerocase]+1)%2;
   if (lumiere[numerocase]==OUI) couleur=jaune;else couleur=noir;
   position.x=xorig+pas*(numerocase%N)+1;
   position.y=yorig+pas*(numerocase/N)+1;
   SDL_FillRect(carre[numerocase],NULL, couleur);
   SDL_BlitSurface(carre[numerocase],0, ecran, &position);
```

```
    if ((numerocase+1)%N!=0)
      {vd=numerocase+1;
       lumiere[vd]=(lumiere[vd]+1)%2;
       if (lumiere[vd]==OUI) couleur=jaune;else couleur=noir;
       position.x=xorig+pas*(vd%N)+1;
       position.y=yorig+pas*(vd/N)+1;
       SDL_FillRect(carre[vd],NULL, couleur);
       SDL_BlitSurface(carre[vd],0, ecran, &position);
      }
    if (numerocase%N!=0)
      {vg=numerocase-1;
       lumiere[vg]=(lumiere[vg]+1)%2;
       if (lumiere[vg]==OUI) couleur=jaune;else couleur=noir;
       position.x=xorig+pas*(vg%N)+1;
       position.y=yorig+pas*(vg/N)+1;
       SDL_FillRect(carre[vg],NULL, couleur);
       SDL_BlitSurface(carre[vg],0, ecran, &position);
      }
    if (numerocase>=N)
      {vh=numerocase-N;
       lumiere[vh]=(lumiere[vh]+1)%2;
       if (lumiere[vh]==OUI) couleur=jaune;else couleur=noir;
       position.x=xorig+pas*(vh%N)+1;
       position.y=yorig+pas*(vh/N)+1;
       SDL_FillRect(carre[vh],NULL, couleur);
       SDL_BlitSurface(carre[vh],0, ecran, &position);
      }
    if (numerocase<N*N-N)
      {vb=numerocase+N;
       lumiere[vb]=(lumiere[vb]+1)%2;
       if (lumiere[vb]==OUI) couleur=jaune;else couleur=noir;
       position.x=xorig+pas*(vb%N)+1;
       position.y=yorig+pas*(vb/N)+1;
       SDL_FillRect(carre[vb],NULL, couleur);
       SDL_BlitSurface(carre[vb],0, ecran, &position);
      }
  }
```

Références bibliographiques :

M. Anderson, T. Fell, *Turning Lights Out with Linear Algebra*, Mathematics Magazine, vol. 71, n° 4, 1998.

Jaap’s Puzzle Page, *Lights out*, 2007.