Teaser

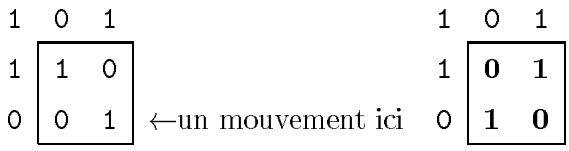
Le but du jeu de teaser[1] consiste à partir de la situation :

```
0 0 0                                   1 1 1
0 1 0    pour obtenir la nouvelle situation :   1 0 1
0 0 0                                   1 1 1
```

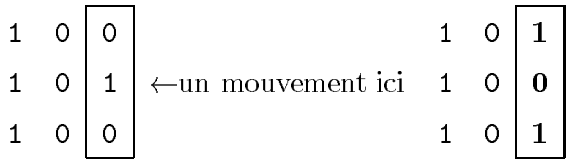
Les seuls "mouvements" légaux consistent à changer les 1 en 0. Par conséquent, le premier mouvement consiste nécessairement dans la modification du 1 central. Lorsque l'on change un 1 en 0, la modification du jeu respecte les règles suivantes :

1. Un mouvement au centre modifie les états des éléments placés en "+" :

2. Un mouvement d'un coin inverse chaque élément d'un carré de 2×2 :



3. Un mouvement au milieu d'un bord modifie le bord entier :



Vous devez réaliser ce jeu en langage C, en utilisant le compilateur de l'environnement UNIX. L'ordinateur ne joue pas à se jeu, il se contente d'afficher les états successifs en fonction de la demande du joueur humain. Il teste la validité du mouvement demandé et si la partie est finie.

[1] Référence : Nico, William I., "Shooting Stars", *Byte*, May, 1976, pp. 42-48
People's Computer Center, *What To Do After You Hit Return*, People's Computer Compagny, 1975, p. 54.

Correction du programme Teaser

1 Analyse

1.1 Lecture du sujet

Le programme `teaser` présente, à l'utilisateur du jeu, une situation de départ sous forme d'un tableau de 3×3, contenant des zéros dans chaque case excepté pour la case centrale qui contient `1`.

Le programme attend que le joueur indique une case contenant `1` puis réaffiche une nouvelle situation en fonction des règles.

Lorsque le tableau est rempli de `1`, sauf au centre, le joueur à gagné.

L'analyse des données conduit à leur représentation.

1.2 Reformulation jusqu'à la levée des ambiguïtés

1.2.1 Reformulation initiale

Le tableau se trouve dans un repère cartésien $(0, \vec{i}, \vec{j})$ et les différentes cases sont repérables par leurs coordonnées.

y \ x	0	1	2
0	*0*	*0*	*0*
1	*0*	*1*	*0*
2	*0*	*0*	*0*

Les éléments représentés par des `1` ou des `0` ne sont nécessaires que pour l'affichage.

Les mouvements sont un nombre de 3.

1.3 Regrouper les idées

Il est possible de regrouper l'ensemble des termes utilisés dans cette analyse en quatre catégories :

1	2	3	4
carré 3×3	convertir	élément	ligne
tableau cartésien	inverser	case	colonne
gagné	situation	état	coin
perdu	mouvement	$1 \rightarrow 0$	bord
		règles	croix
			millieu

Les définitions du groupe 4 ne semblent pas modifiables.

Dans le groupe 1 le repérage cartésien ne fait pas partie des données du problème. C'est une *interprétation*. Il est possible de proposer un repérage indiciel en utilisant une représentation sous forme d'un vecteur :

$$\begin{array}{cccc} 0 & 1 & 2 & 3 \\ & 4 & 5 & 6 \\ & 7 & 8 & 9 \end{array}$$

Cette représentation rappelle le pavé numérique. Cependant l'ordre des touches du pavé n'est pas le même que les indices du tableau précédent :

$$\begin{array}{cccc} 0 & 1 & 2 & 3 \\ & 4 & 5 & 6 \\ & 7 & 8 & 9 \end{array} \Rightarrow \begin{array}{|c|c|c|} \hline 7 & 8 & 9 \\ \hline 4 & 5 & 6 \\ \hline 1 & 2 & 3 \\ \hline \end{array}$$

Dans le groupe 3 chaque case du tableau contient soit la valeur *zéro*, soit la valeur *un*. Il s'agit donc d'une valeur booléenne. Nous aurons à faire passer le contenu de la case d'une valeur à l'autre. Parmi toutes les façons de représenter un ensemble booléens nous pouvons retenir :

- 0, 1. Dans ce cas si x et $\overline{x} \in \{0,1\}$

$$\overline{x} = 1 - x$$

- -1, +1. Dans ce cas si x et $\overline{x} \in \{-1,+1\}$

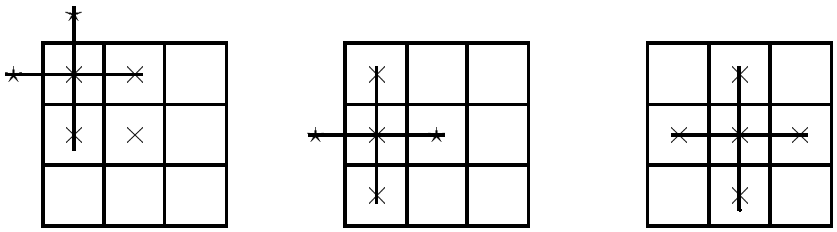
$$\overline{x} = -x$$

c'est à dire que $\overline{x}$ est l'opposé de x. Ce qui simplifie un peu les calculs.

Dans le groupe 2 l'idée d'avoir à gérer 3 types de mouvements, chacun étant différents peut être remise en cause. La recherche d'un seul mouvement général simplifierait la programmation.

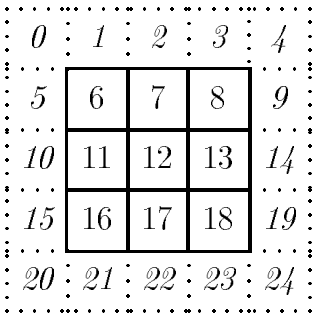
1.4 Changement de contexte

Une représentation graphique "élargie" des mouvements montre que le mouvement en croix suffit à effectuer l'ensemble des 3 mouvements :

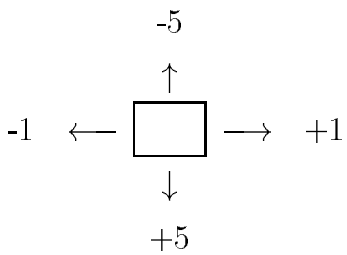


La case centrale demandera un traitement particulier.

Pour que ce mouvement soit réalisable, il est nécessaire de disposer d'un tableau plus grand où les opérations seront *effectives*. Le centre de ce tableau seulement représentera l'espace du jeu, et l'idée d'utiliser un vecteur au lieu d'une matrice à 2 dimensions est conservée :



Compte-tenu de la dimension 5×5 de ce tableau, chaque case de la partie centrale est contigüe avec 4 cases voisines dont les indices sont :



1.5 Choix facilitant la réalisation

En faisant correspondre les touches pressées, le vecteur de touches et le vecteur contenant l'état des cases, on obtient :

Clavier	Haut			Milieu			Bas		
Touches	7	8	9	4	5	6	1	2	3
Indices du tableau	6	7	8	11	12	13	16	17	18

0	1	2	3	4	5	**6**	**7**	**8**	9	10	**11**	**12**	**13**	14	15	**16**	**17**	**18**	19	20	21	22	23	24

2 Détermination des fonctions

Teaser

Le tableau du jeu,
seules les 9 cases centrales
sont visibles.

```
int fond[25] =
  {-1,  -1,  -1,  -1,  -1,
   -1,  -1,  -1,  -1,  -1,
   -1,  -1,   1,  -1,  -1,
   -1,  -1,  -1,  -1,  -1,
   -1,  -1,  -1,  -1,  -1};
```

---

int **coup** ∈ {0-9}

---

int **touche** =

{0, 6, 7, 8, 11, 12, 13, 16, 17, 18} ;

tant que *pas* fini

affiche (`fond`)

Si *gagnant* (`fond`)

alors	**sinon**
fini	*saisir* (`coup`)
	Si `coup = 0` **alors fini**
	change (`touche[coup]`, `fond`)
	traiter le case centrale

gagnant (tablo)

Si `tablo [12]` = 1

alors	**sinon**

(Ni gagné, ni perdu) return	`plein = VRAI`
	`vide = VRAI`
	pour chaque `touche`
	Si `tablo[touche]` ≠ -1
	alors tableau non vide
	pour chaque `touche`
	Si `tablo[touche]` ≠ 1
	alors tableau non plein
	Si `plein = VRAI`
	alors *affiche* Bravo ; **fini**
	Si `vide = VRAI`
	alors *affiche* Perdu ; **fini**

change (x, table)

`x` est argument

`table` est argument

```
table[x - 5] *= -1 ;
table[x - 1] *= -1 ;
table[x    ] *= -1 ;
table[x + 1] *= -1 ;
table[x + 5] *= -1 ;
```

affiche (tablo)

int `k`

int `i`

`touche` est globale

`k = 7`

tant que `k > 0`

`i = k`

tant que `i < k + 3`

Si `tablo[touche[i]] = 1`

alors	**Sinon**
affiche 1	*affiche* 0

`i = i + 1`

affiche retour à la ligne

`k = k - 3`

```
/*
 * Jeu de Teaser
 */

main()
{
  int sto[10] ;
  int i ;
  char c = '0'    ;

  /* initialisation */

  for (i = 1 ; i < 10 ; i++)      /* image de l'ecran */
    sto[i] = -1 ;
  sto[5] =  1 ;

  affiche (sto)  ;

  while (1)
  {
    scanf ( "%c", &c) ;
    switch (c)
    {
    case 49 :                      /* 1 */
      if (sto[1] == 1)
      {
        sto[1] *=  -1 ;
        sto[2] *=  -1 ;
        sto[4] *=  -1 ;
        sto[5] *=  -1 ;
      }
      affiche (sto)  ;
      break ;

    case 50 :                      /* 2 */
      if (sto[2] == 1)
      {
        sto[1] *=  -1 ;
        sto[2] *=  -1 ;
        sto[3] *=  -1 ;
      }
      affiche (sto)  ;
      break ;

    case 51 :                      /* 3 */
      if (sto[3] == 1)
      {
        sto[2] *=  -1 ;
        sto[3] *=  -1 ;
        sto[5] *=  -1 ;
        sto[6] *=  -1 ;
      }
      affiche (sto)  ;
      break ;

    case 52 :                      /* 4 */
      if (sto[4] == 1)
      {
        sto[1] *=  -1 ;
        sto[4] *=  -1 ;
        sto[7] *=  -1 ;
      }
      affiche (sto)  ;
      break ;

    case 53 :                      /* 5 */
      if (sto[5] == 1)
      {
        sto[2] *=  -1 ;
```



```
        sto[4] *=  -1 ;
        sto[5] *=  -1 ;
        sto[6] *=  -1 ;
        sto[8] *=  -1 ;
      }
      affiche (sto)  ;
      break ;

    case 54 :                       /* 6 */
      if (sto[6] == 1)
      {
        sto[3] *=  -1 ;
        sto[6] *=  -1 ;
        sto[9] *=  -1 ;
      }
      affiche (sto)  ;
      break ;

    case 55 :                      /* 7 */
      if (sto[7] == 1)
      {
        sto[4] *=  -1 ;
        sto[5] *=  -1 ;
        sto[7] *=  -1 ;
        sto[8] *=  -1 ;
      }
      affiche (sto)  ;
      break ;

    case 56 :                      /* 8 */
      if (sto[8] == 1)
      {
        sto[7] *=  -1 ;
        sto[8] *=  -1 ;
        sto[9] *=  -1 ;
      }
      affiche (sto)  ;
      break ;

    case 57 :                      /* 9 */
      if (sto[9] == 1)
      {
        sto[5] *=  -1 ;
        sto[6] *=  -1 ;
        sto[8] *=  -1 ;
        sto[9] *=  -1 ;
      }
      affiche (sto)  ;
      break ;

    case 'Q' :
    case 'q' :
      exit (0) ;
    default  :
      break     ;

    }                              /* fin du switch */
  }
}

int affiche (sto)
int sto [];
{
  int i ;

  for (i = 7 ; i < 10 ; i++)
  {
    if (sto[i] == 1)
      printf ("%d ", 1)  ;
```

```
    else printf ("%d ", 0) ;
  }
  printf ("\n") ;

  for (i = 4 ; i < 7 ; i++)
  {
    if (sto[i] == 1)
      printf ("%d ", 1)  ;
    else printf ("%d ", 0) ;
  }
  printf ("\n") ;

  for (i = 1 ; i < 4 ; i++)
  {
    if (sto[i] == 1)
      printf ("%d ", 1)  ;
    else printf ("%d ", 0) ;
  }
  printf ("\n\n") ;

}
```