



Algorithmique et programmation en C

Mini - Projet

Automates cellulaires

Version 1.0

Last update: 15/01/2013

Use: Students/Staff

Author: Laurent GODEFROY

SOMMAIRE

1	PREAMBULE	3
2	GENERALITES THEORIQUES SUR CE PROJET	3
2.1	<i>AUTOMATES A DEUX ETATS</i>	<i>4</i>
2.2	<i>AUTOMATES A TROIS ETATS</i>	<i>7</i>
3	CODAGE EN C DES AUTOMATES A DEUX ETATS.....	9
3.1	<i>STRUCTURES DE DONNEES.....</i>	<i>9</i>
3.2	<i>LE CAS D'UN VOISINAGE DE MOORE</i>	<i>10</i>
3.2.1	Initialisation et affichage	10
3.2.2	Changement de génération	12
3.2.3	Evolution sur plusieurs générations	12
3.2.4	Quelques automates bien particuliers.....	14
3.3	<i>LE CAS D'UN VOISINAGE DE VON NEUMANN</i>	<i>15</i>
4	CODAGE EN C DES AUTOMATES A TROIS ETATS	15
4.1	<i>MODIFICATIONS A APPORTER.....</i>	<i>15</i>
4.2	<i>UN AUTOMATE BIEN PARTICULIER</i>	<i>16</i>
5	BAREME INDICATIF.....	16

1 PREAMBULE

Cet examen est **individuel**. Toute forme de plagiat ou utilisation de codes disponibles sur internet ou tout autre support, même de manière partielle, est strictement interdite et se verra sanctionnée d'un 0, d'une mention « cheater », et le cas échéant d'un conseil de discipline.

Vous devrez envoyer les codes sources de votre projet par mail à votre formateur avant le dimanche 10 février à 23h59, heure locale. Au delà de cette date et heure votre note sera de 0. Vous comprimerez ces codes sources dans une archive au format « .zip ».

Ce mini-projet donne lieu à des soutenances qui se dérouleront la semaine du 11 février.

Les soutenances sont également individuelles. Elles durent **10 minutes** pendant lesquelles vous montrerez à votre examinateur le bon fonctionnement de votre programme en en faisant la démonstration. Si vous n'avez pas implémenté le projet dans sa totalité, vous exposerez les parties fonctionnelles.

Pour appuyer votre présentation, vous devrez préparer un fichier de type Powerpoint, dans lesquels vous expliquerez les points du code que vous jugez les plus importants et significatifs. Il n'est pas nécessaire d'envoyer ce fichier à votre examinateur, ce dernier le découvrira le jour de la soutenance.

Un barème indicatif vous est donné dans la dernière partie de ce sujet.

2 GENERALITES THEORIQUES SUR CE PROJET

Le but de ce mini-projet est d'implémenter en langage C deux classes d'automates cellulaires bien connues des informaticiens théoriques et des mathématiciens : les automates de la famille du « jeu de la vie » et ceux de la famille du « Brian's brain ». Dans cette partie nous allons vous présenter les notions théoriques dont vous aurez besoin par la suite. Vous aurez peut-être besoin de plusieurs lectures pour bien assimiler toutes ces règles, prenez donc le temps nécessaire à une bonne compréhension avant de commencer les codes demandés dans les parties suivantes.

Pour nous un automate cellulaire sera un tableau à deux dimensions, a priori infini, dont chaque case contiendra une cellule pouvant prendre plusieurs états. Chaque changement de génération s'accompagnera d'une modification des états de toutes les cellules selon des règles bien précises.

Les plus curieux d'entre vous trouveront de nombreux compléments à ce sujet sur le site

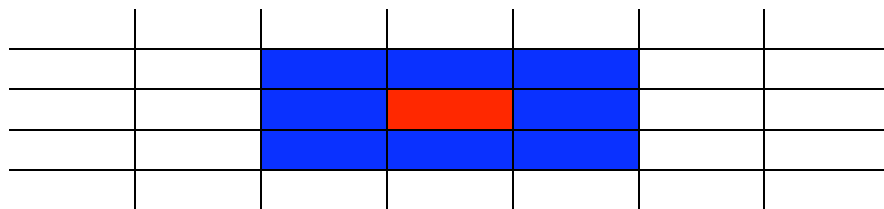
<http://www.mirekw.com/>

2.1 AUTOMATES A DEUX ETATS

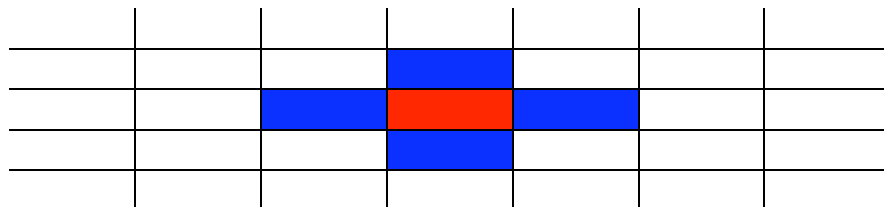
Dans cette famille d'automates, les cellules ne possèdent que deux états : mort ou vivant. Une cellule change d'état d'une génération à la suivante en fonction du nombre de cellules vivantes qui l'entourent.

Pour définir cette notion de « cellules qui en entourent une autre », on distinguera deux types de « voisinages ». L'un comportant 8 cases, celui de Moore, et l'autre comportant 4 cases, celui de Von Neumann.

- Voisinage de Moore : la cellule rouge à 8 voisins qui sont représentés en bleu.



- Voisinage de Von Neumann : la cellule rouge à 4 voisins qui sont représentés en bleu.



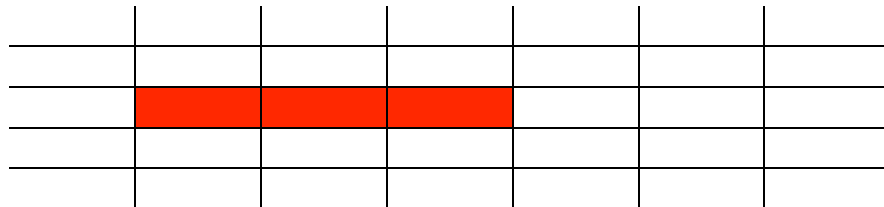
Pour définir un automate, on commence donc par se fixer un des deux types de voisinage ci-dessus. Ensuite on se donne des règles de naissance et de survie. La règle de naissance indiquera à quelle condition une cellule morte pourra devenir vivante à la génération suivante, et celle de survie à quelle condition une cellule vivante restera vivante à la génération suivante.

Pour représenter ces règles on adoptera la notation classique anglo-saxonne : B.../S... Ainsi l'on fera succéder à la lettre B le nombre de cellules vivantes entourant une cellule morte pour qu'elle devienne vivante à la génération suivante, et à la lettre S le nombre de cellules vivantes entourant une cellule vivante pour qu'elle reste vivante à la génération suivante.

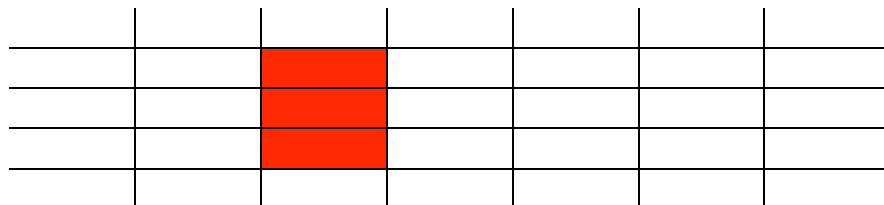
Voici un exemple pour clarifier les choses : l'automate de John Conway, qui utilise un voisinage de Moore et les règles B3/S23. Cela signifie donc que :

- Une cellule morte deviendra vivante si elle est entourée de 3 cellules vivantes parmi ses 8 voisines.
- Une cellule vivante restera vivante si elle est entourée de 2 ou 3 cellules vivantes parmi ses 8 voisines.
- Voici une évolution où l'on a représenté les cellules vivantes en rouge et celles mortes en blanc.

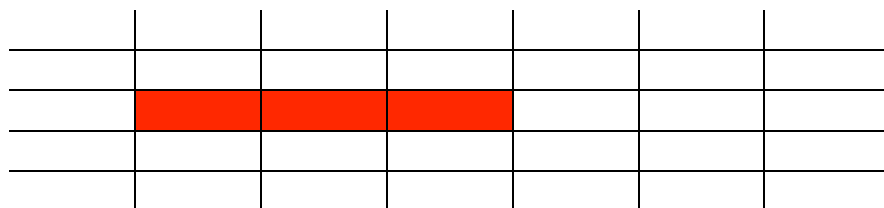
Génération 1 :



Génération 2 :



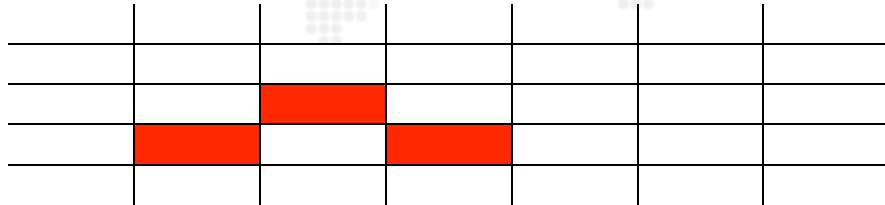
Génération 3 :



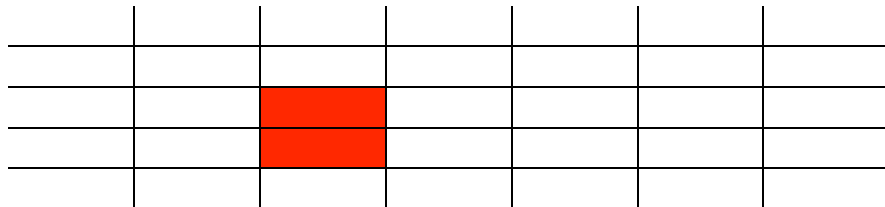
And so on !

- Autre exemple d'évolution :

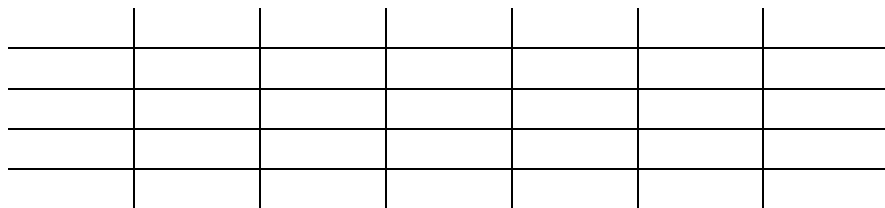
Génération 1 :



Génération 2 :



Génération 3 :



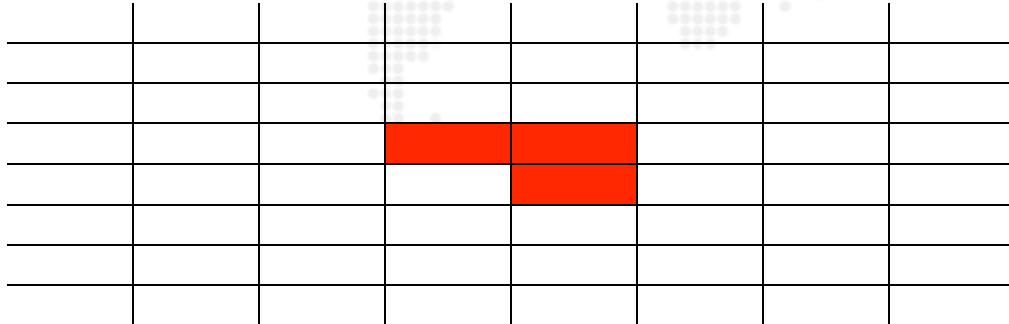
Tout le monde est mort ...

Il est évident sur ces exemples d'évolutions que la configuration initiale des cellules vivantes aura une importance capitale sur l'évolution de la population.

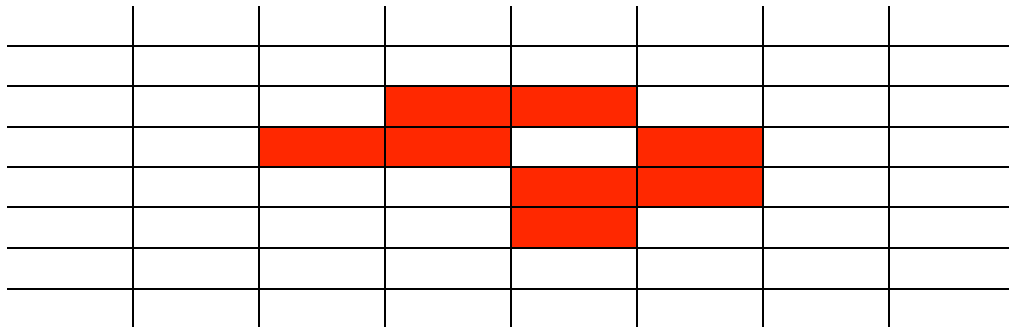
Voici un autre exemple d'automate : celui d'Edward Fredkin, qui utilise un voisinage de Von Neuman et les règles B13/S13. Cela signifie donc que :

- Une cellule morte deviendra vivante si elle est entourée de 1 ou 3 cellules vivantes parmi ses 4 voisines.
- Une cellule vivante restera vivante si elle est entourée de 1 ou 3 cellules vivantes parmi ses 4 voisines.
- Voici un exemple d'évolution où comme précédemment l'on a représenté les cellules vivantes en rouge et celles mortes en blanc.

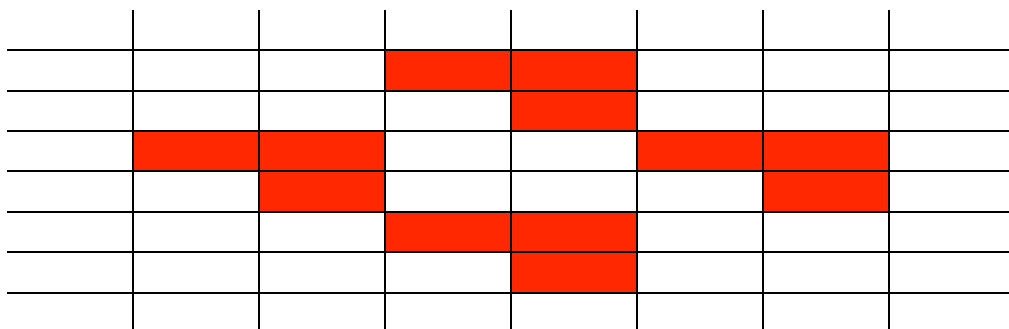
Génération 1 :



Génération 2 :



Génération 3 :



Et ainsi de suite...

2.2 AUTOMATES A TROIS ETATS

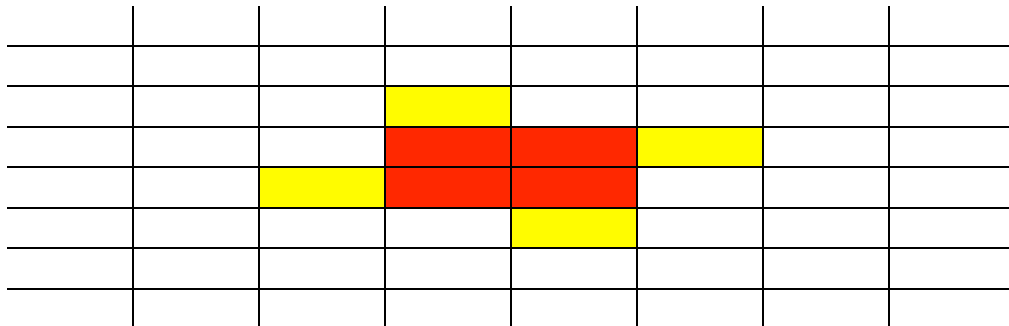
Dans cette famille d'automates, les cellules possèdent trois états : mort, vivant ou fantôme. Une cellule change d'état d'une génération à la suivante en fonction du nombre de cellules vivantes qui l'entourent. **On ne considèrera ici que des voisinages de Moore.**

On adoptera pour décrire le devenir des cellules mortes et vivantes le même type de notation que dans le cas des automates à deux états, à cette nuance près qu'une cellule vivante si elle ne remplit pas les règles de survie devient fantôme et non pas morte. De plus, une cellule fantôme deviendra nécessairement morte à la génération suivante quel que soit le nombre de cellules vivantes de son voisinage.

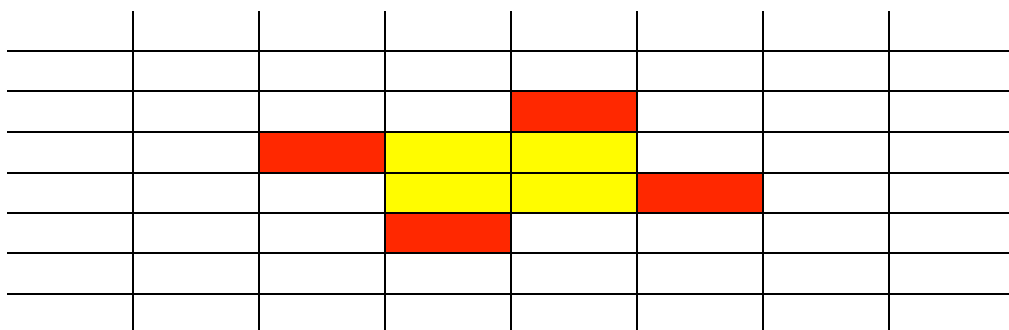
Voici un exemple d'un tel automate : celui de Brian Silverman, qui utilise donc un voisinage de Moore et les règles B2/S. Cela signifie ainsi que :

- Une cellule morte deviendra vivante si elle est entourée de 2 cellules vivantes parmi ses 8 voisines.
 - Une cellule vivante deviendra nécessairement fantôme.
 - Une cellule fantôme deviendra nécessairement morte.
- Voici un exemple d'évolution où l'on a représenté les cellules vivantes en rouge, celles mortes en blanc et les fantômes en jaune.

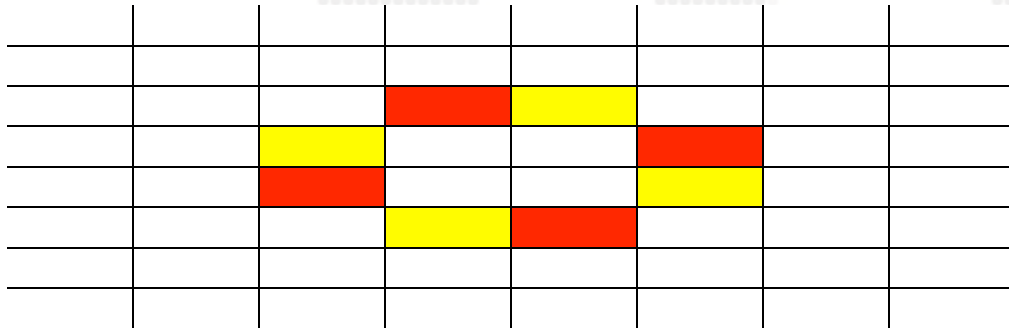
Génération 1 :



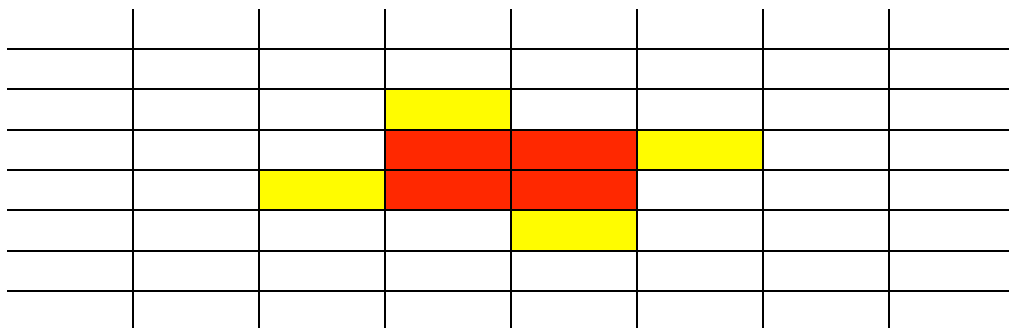
Génération 2 :



Génération 3 :



Génération 4 :



Ces généralités étant terminées, vous aller maintenant réaliser en langage C une application permettant à un utilisateur, après qu'il ait saisi les dimensions du tableau, la configuration initiale des cellules et les règles de naissance/survie, de voir évoluer sa population.

Dans la partie 3 vous traiterez la cas des automates à deux états, et dans la partie 4 celui des automates à trois états. Dans chaque cas la structure du code vous est imposée, vous n'aurez donc qu'à implémenter les fonctions demandées.

3 CODAGE EN C DES AUTOMATES A DEUX ETATS

Il vous est fortement recommandé de lire l'intégralité de cette partie avant de commencer à coder. Les travaux demandés sont mis en évidence avec une couleur bleue.

3.1 STRUCTURES DE DONNEES

Dans le « main », il sera déclaré plusieurs variables et tableaux afin de stocker les informations nécessaires :

- Un tableau à deux dimensions d'entiers sera déclaré et dimensionné de façon dynamique après que l'utilisateur en ait saisi sa taille. C'est dessus qu'évoluera notre population. Chaque case du tableau sera amenée à valoir soit « 0 » soit « 1 » selon que la cellule correspondante sera morte ou vivante.
- Deux entiers correspondants aux valeurs des dimensions du tableau.
- Deux tableaux statiques d'entiers à une dimension comportant chacun 9 éléments. L'un modélisera les règles de naissance et l'autre les règles de survie, en considérant un voisinage de Moore. Chaque case de ces tableaux sera amenée à valoir soit « 0 » soit « 1 » selon que le nombre de cellules vivantes du voisinage induise une naissance dans un cas, et une survie dans l'autre. Par exemple la règle B3/S23 sera modélisée par un tableau B valant :

0	0	0	1	0	0	0	0	0
---	---	---	---	---	---	---	---	---

Et un tableau S valant :

0	0	1	1	0	0	0	0	0
---	---	---	---	---	---	---	---	---

Dans les deux cas l'indice i correspond à la présence de i cellules vivantes dans le voisinage et la valeur de la case indique pour B si une naissance aura lieu et pour S si une survie sera possible.

- Sur le même principe deux tableaux statiques à une dimension comportant chacun 5 éléments modélisant les règles de naissance et de survie en considérant cette fois ci un voisinage de Von Neumann.
- Un entier indiquera le nombre de générations voulu par l'utilisateur.

3.2 LE CAS D'UN VOISINAGE DE MOORE

3.2.1 Initialisation et affichage

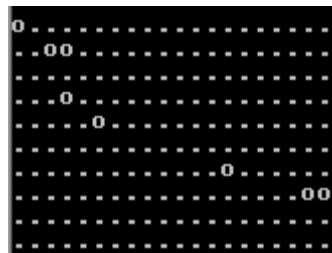
Déclarer dans un « .h » et implémenter dans un « .c » les fonctions ci-dessous :

```
void dimensionnement (int *n,int *m);  
void initialisation(int** tab, int n, int m);  
int nb_generation();  
void affiche(int** tab, int n, int m);  
void affiche_fichier(int** tab, int n, int m,char* nom);  
void saisie_B_M(int B[9]);  
void saisie_S_M(int S[9]);
```

Quelques précisions :

- La fonction « dimensionnement » fait saisir à l'utilisateur deux entiers supérieurs ou égaux à 1 (ce seront les dimensions de notre tableau) et affecte ces valeurs à ses paramètres.
- La fonction « initialisation » prend en paramètres un tableau à deux dimensions d'entiers ainsi que ses dimensions. Elle commence par affecter toutes les cases du tableau avec la valeur « 0 ». Elle demande ensuite à l'utilisateur le nombre de cellules vivantes qu'il souhaite avoir à la première génération. Elle lui fait ensuite saisir les coordonnées de ces cellules, en prenant soin de lui redemander si elles ne sont pas valides (coordonnées « en dehors » du tableau, ou saisie de deux fois les mêmes coordonnées). Elle affecte enfin les cases correspondantes avec la valeur « 1 ». Pour l'utilisateur la numérotation des lignes et des colonnes commencera à 1. A vous de gérer ce fait.
- La fonction « nb_generation » fait saisir à l'utilisateur un entier supérieur ou égal à 1 qu'elle retourne. Ce sera le nombre de générations de notre évolution.
- La fonction « affiche » prend en paramètres un tableau à deux dimensions d'entiers, ainsi que ses dimensions. Elle réalise alors l'affichage du tableau avec la convention suivante : un « 0 », c'est à dire une cellule morte, est représenté par un « . », et un « 1 », c'est à dire une cellule vivante, est représenté par un « o ». On supposera que le tableau passé en paramètre ne contient que des 0 et des 1.

Exemple d'affichage :



- La fonction « affiche_fichier » réalise la même chose que la fonction « affiche », mais cet affichage ne se fait plus à la console mais dans un fichier créé à l'occasion et dont le nom est passé en paramètre.
- La fonction « saisie_B_M » prend en paramètre un tableau de 9 entiers. Elle fait saisir à l'utilisateur ces 9 valeurs qui devront nécessairement être 0 ou 1. Pour chaque indice i entre 0 et 8, la saisie s'accompagnera d'un affichage du style « Une cellule morte ayant i

voisins sera t-elle morte ou vivante à la génération suivante ? ».

- La fonction « saisie_S_M » prend en paramètre un tableau de 9 entiers. Elle fait saisir à l'utilisateur ces 9 valeurs qui devront nécessairement être 0 ou 1. Pour chaque indice i entre 0 et 8, la saisie s'accompagnera d'un affichage du style « Une cellule vivante ayant i voisins sera t-elle morte ou vivante à la génération suivante ? ».

3.2.2 Changement de génération

Déclarer dans un « .h » et implémenter dans un « .c » les fonctions ci-dessous :

```
int nb_voisins_M(int** tab, int i, int j);  
void duplication(int** tab1, int** tab2, int n, int m);  
void generation_suivante_M(int** tab1, int** tab2, int n, int m, int B[9], int S[9]);
```

Quelques précisions :

- La fonction « nb_voisins_M » prend en paramètres un tableau à deux dimensions d'entiers, ainsi que les coordonnées d'une cellule (coordonnées que l'on suppose valides). Elle retourne le nombre de cellules vivantes qui entourent la cellule passée en paramètre. Rappelons que dans cette sous-partie on ne traite que le cas des voisinages de Moore. A vous de gérer la possibilité de se trouver sur le « bord » du tableau.
- La fonction « duplication » prend en paramètres deux tableaux à deux dimensions d'entiers, ainsi que les dimensions de ces tableaux que l'on suppose égales. Elle réalise la copie du premier dans le second.
- La fonction « génération_suivante » prend en paramètres deux tableaux à deux dimensions d'entiers, les dimensions de ces tableaux que l'on suppose égales, ainsi que deux tableaux de 9 entiers que l'on suppose binaires. Le premier tableau représente l'état notre population de cellules à une génération n , et cette fonction affecte au second tableau l'état de la population à la génération $n+1$ selon les règles de naissance et de survie indiquées par les tableaux B et S.

3.2.3 Evolution sur plusieurs générations

Déclarer dans un « .h » et implémenter dans un « .c » la fonction ci-dessous :

```
void jeu_M(int** tab1, int n, int m, int B[9], int S[9], int ng);
```

Quelques précisions :

- La fonction « jeu_M » prend en paramètres un tableau à deux dimensions d'entiers, ses dimensions, deux tableaux de 9 entiers que l'on suppose binaires et un entier naturel. Elle

réalise l'évolution de la population pendant « ng » générations selon les règles de naissance et de survie indiquées par les tableaux B et S. Seront affichées à chaque génération son numéro ainsi que l'état de la population correspondant.

Ecrire un « main » où seront déclarées les structures de données nécessaires (voir sous-partie 3.1).
Utiliser les fonctions précédemment écrites pour que la saisie suivante :

```
Saisir votre nombre de lignes : 20
Saisir votre nombre de colonnes : 50
Saisir votre nombre de cellules vivantes : 5
Saisir les coordonnées de votre cellule numero 1 : 6 4
Saisir les coordonnées de votre cellule numero 2 : 6 3
Saisir les coordonnées de votre cellule numero 3 : 6 5
Saisir les coordonnées de votre cellule numero 4 : 5 5
Saisir les coordonnées de votre cellule numero 5 : 4 4
Saisir votre nombre de generations : 41
Une cellule morte ayant 0 voisins sera t-elle morte ou vivante a la generation s
uivante ? 0
Une cellule morte ayant 1 voisins sera t-elle morte ou vivante a la generation s
uivante ? 0
Une cellule morte ayant 2 voisins sera t-elle morte ou vivante a la generation s
uivante ? 0
Une cellule morte ayant 3 voisins sera t-elle morte ou vivante a la generation s
uivante ? 1
Une cellule morte ayant 4 voisins sera t-elle morte ou vivante a la generation s
uivante ? 0
Une cellule morte ayant 5 voisins sera t-elle morte ou vivante a la generation s
uivante ? 0
Une cellule morte ayant 6 voisins sera t-elle morte ou vivante a la generation s
uivante ? 0
Une cellule morte ayant 7 voisins sera t-elle morte ou vivante a la generation s
uivante ? 0
Une cellule morte ayant 8 voisins sera t-elle morte ou vivante a la generation s
uivante ? 0
Une cellule vivante ayant 0 voisins sera t-elle morte ou vivante a la generation
suivante ? 0
Une cellule vivante ayant 1 voisins sera t-elle morte ou vivante a la generation
suivante ? 0
Une cellule vivante ayant 2 voisins sera t-elle morte ou vivante a la generation
suivante ? 1
Une cellule vivante ayant 3 voisins sera t-elle morte ou vivante a la generation
suivante ? 1
Une cellule vivante ayant 4 voisins sera t-elle morte ou vivante a la generation
suivante ? 0
Une cellule vivante ayant 5 voisins sera t-elle morte ou vivante a la generation
suivante ? 0
Une cellule vivante ayant 6 voisins sera t-elle morte ou vivante a la generation
suivante ? 0
Une cellule vivante ayant 7 voisins sera t-elle morte ou vivante a la generation
suivante ? 0
Une cellule vivante ayant 8 voisins sera t-elle morte ou vivante a la generation
suivante ? 0
```

réalise cet affichage à la première génération :

[illegible]

et celui ci à la dernière génération :

[illegible]

3.2.4 Quelques automates bien particuliers

Compléter les « .h » et « .c » de la sous-partie précédente avec les fonctions suivantes :

```
void jeu_conway(int** tab1,int n, int m,int ng);
void jeu_fredkin_M(int** tab1,int n, int m,int ng);
```

Quelques précisions :

- La fonction « jeu_conway » réalise l'automate B3/S23.
- La fonction « jeu_fredkin » réalise l'automate B1357/S1357

Faites évoluer ces automates avec quelques configurations initiales. Que constatez vous ?

3.3 LE CAS D'UN VOISINAGE DE VON NEUMANN

Ecrire, quant cela est nécessaire, de nouvelles fonctions sur le modèle de celles déjà implémentées afin de traiter maintenant le cas d'un voisinage de Von Neumann. On gardera la même architecture de code.

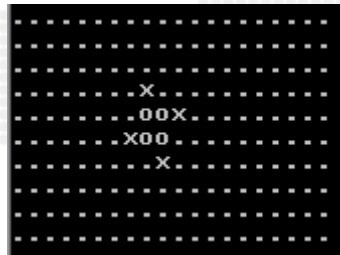
Ecrire également une fonction réalisant l'automate B13/S13.

4 CODAGE EN C DES AUTOMATES A TROIS ETATS

4.1 MODIFICATIONS A APPORTER

Ecrire, quant cela est nécessaire, de nouvelles fonctions sur le modèle de celles déjà implémentées afin de traiter maintenant le cas des automates à trois états. On rappelle une fois de plus qu'ils utilisent nécessairement un voisinage de Moore. On gardera la même architecture de code.

La fonction d'initialisation devra en particulier faire saisir également les coordonnées des cellules fantômes de la première génération et la fonction d'affichage représenter celles ci par un « x ». Voici d'ailleurs un exemple d'affichage :



Cela sous-entend que le tableau représentant la population ne contiendra plus uniquement des « 0 » et des « 1 » mais également des « 2 » pour modéliser les cellules fantômes.

4.2 UN AUTOMATE BIEN PARTICULIER

Ecrire une fonction réalisant l'automate B2/S. L'utiliser pour faire évoluer la population donnée en exemple dans la partie précédente. Que constatez vous ?

5 BAREME INDICATIF

- Soutenance : 20 points (détails à venir)
- Partie 3.2.1. : 7
- Partie 3.2.2. : 9
- Partie 3.2.3. : 8
- Partie 3.2.4. : 4
- Partie 3.3. : 4
- Partie 4.1. : 7
- Partie 4.2. : 1

Ce qui fait un total de 60 points, ramené sur 20 par proportionnalité.