# Les automates cellulaires, implémentation

Nguyen Phuc-Thien Thomas

17 avril 2013

**Résumé**

Ce document a été rédigé dans le cadre de mon stage à l'Université de Genève. Plus précisément au centre universitaire d'informatique, où on travaille notamment sur la simulation numérique de divers phénomènes physiques ou biologiques. Dans le cas présent, c'est simultanément les deux, puisque c'est la simulation de sang dans un anévrisme qui est le sujet central du stage. Une simulation qui demande à la fois de la physique, avec la mécanique des fluides, et à la fois de la biologie, pour la physiologie du sang.

Nous n'allons cependant pas commencer par montrer et utiliser des équations aux dérivées partielles qu'exige la mécanique des fluides. Nous allons plutôt aprendre le mécanisme des simulations au niveau informatique, de manière intuitive. Et tout ceci commence dans le petit monde... des automates cellulaires.

## 1 Qu'est-ce que c'est ?

Un automate cellulaire [4, 1] est une grille régulière de cellules contenant chacune un état parmi un ensemble fini, qui évolue au cours du temps. L'état d'une cellule au temps $t + 1$ dépend de celui d'un certain nombre de cellules nommé *voisinage* au temps $t$. À chaque nouvelle étape, les même règles sont appliquées à chaque cellules de la grille, ce qui produit une nouvelle génération de cellule, dépendant entièrement de la génération précédente.

Deux automates cellulaires seront exposées ici afin de visualiser ce que c'est.

## 2 Exemples

### 2.1 Un automate élémentaire

L'un des automates les plus simples consiste en une grille d'une seule dimension, où chaque cellule ne peut prendre que deux états (par exemple noir/blanc), avec pour chaque cellule un voisinage de trois cellules : elle-même et les deux qui lui sont adjacentes.

Puisque seuls deux états sont possibles, et que le voisinage est constitué de trois cellules, il existe $2^3 = 8$ motifs différents possibles. Comme chaque motif peut produire soit une cellule noire, soit une blanche, il y a donc $2^8 = 256$ règles possibles pour ce type d'automates.

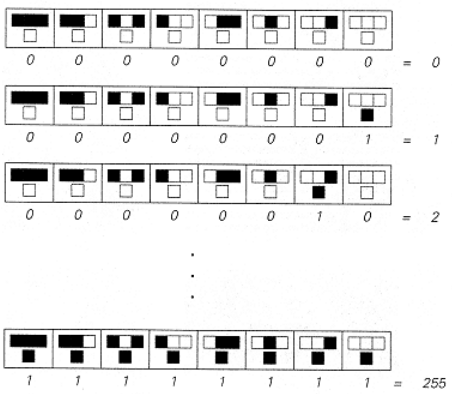
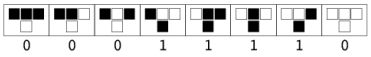
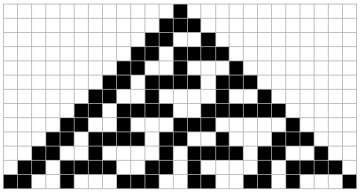



FIGURE 1 – 256 configurations possibles. Chaque état peut être représenté par un nombre, 0 pour blanc et 1 pour noir.

Par exemple, un automate qui obéit à la règle suivante (aussi appelée *règle 30*) :




Produira le résultat suivant, si on part d'une grille composée uniquement de cases blanches, sauf une :



Chaque ligne au temps $t+1$ est le résultat de la précédente au temps $t$. Les couleurs des cases au temps $t+1$ sont fonction des trois voisins présents à la ligne précédente au temps $t$. Par exemple, si tous les trois voisins au temps $t$ sont noirs, la case résultante au temps $t+1$ est blanche. Idem si le voisin à gauche et du milieu sont noirs, si tous sauf celui du milieu est noir, ou s'ils sont tous blancs. Dans les autre cas, par exemple avec la case de gauche noire et les autres blanches, la nouvelle case sera noire.

En continuant l'automate, on s'apercevra que le motif dessiné est très complexe et ne semble montrer aucune régularité. On peut donc se dire qu'un phénomène complexe peut être engendré par une chose toute simple, un petit point dans une grille qui devient un motif compliqué après l'application répétée d'une règle ! Il existe également des règles qui produisent des motifs réguliers, voire une grille totalement unicolore.

Ajouter des états ou des voisins font littéralement exploser le nombre de règles possibles. Par exemple, rien qu'en ajoutant un état *gris*, il existe $3^3 = 27$ motifs différents possibles (pour un voisinage de trois cellules). Comme chaque motif peut produire trois états possibles, il y a donc $3^{27} = 7625597484987$ règles possibles pour ce type d'automates. En ajoutant un voisin supplémentaire, par exemple la cellule qui est à deux cases à la droite de la principale, il y aurait $3^{3^4} = 3^{81} \approx 4.43 \times 10^{38}$ possibilités !

## 2.2 Le jeu de la vie

C'est un simple automate, cette fois en deux dimensions, où chaque cellule peut prendre deux états (mort ou vivant) et possède neuf voisins : les huit cellules qui sont autour et elle-même. Les règles sont les suivantes :

- Une cellule morte possédant exactement trois voisines vivantes devient vivante ;
- Une cellule vivante possédant deux ou trois voisines le reste, sinon elle meurt.

Par exemple, la configuration  devient  .

Bien que les règles soit simples, ce « jeu »permet le développement de motifs très complexes. Durant l'exécution des règles, plusieurs structures particulières peuvent apparaître, notamment des *structures stables*, des *oscillateurs*, ou encore des *vaisseaux*. Exemples de ces structures :

**Structures stables :** ces structures ne changent jamais d'un instant à l'autre, à moins qu'un parasite arrive à proximité de la structure. Par exemple, le bloc de quatre cellules :  , ou encore le « bateau » :  .

**Oscillateurs :** ce sont des motifs qui se transforment de manière cyclique avant de retrouver leur état initial. Le premier exemple donné est un oscillateur sur deux périodes : la configuration  devient  , qui redevient  , et ainsi de suite.

**Vaisseaux :** ce sont des structures pouvant se décaler en gardant la même forme après un certain nombre de générations. Par exemple, ce motif, nommé « glider », retrouve son état initial, mais translaté, après quatre périodes :  →  →  →  →  .

**Autres :** il en existe bien d'autres, comme les « puffeurs »(vaisseaux laissant des débris), les canons (oscillateurs qui émettent des vaisseaux), etc. Des animations sont disponibles sur internet pour voir ce que cela donne.

# 3 Utilité

Les automates cellulaires sont notamment étudiés en mathématiques et informatique théorique. En effet, comme nous l'avons vu, des règles simples peuvent engendrer des grilles extrêmement complexes, voire aléatoires. Elles peuvent également avoir des applications en physique, où elles modélisent divers phénomènes comme la dynamique des fluides avec par exemple les modèles de HPP ou FHP [1]. La page du groupe montre quelques animations de divers automates [3].

Elles peuvent être facilement implémentées en informatique. Une implémentation du jeu de la vie sera présentée dans ce document.

1. Initiales des concepteurs de ces modèles. Respectivement Hardy, Pommeau, Pazzis et Frisch, Hasslacher, Pommeau. Ce sont des automates cellulaires comme les autres, avec des règles de voisinage et de collision qui simulent de manière élémentaire un fluide au niveau microscopique.

# 4 Implémentation du jeu de la vie

## 4.1 Méthode 1

Pour commencer, nous allons créer une matrice de 100 par 100, dont les valeurs des cellules (mortes / vivantes) sont générées aléatoirement. Cette matrice est donc la *situation initiale* de l'automate.

Il s'agit maintenant de créer deux fonctions permettant de faire les tests nécessaires (état de la cellule testée et celle des cellules voisines) afin de déterminer si une cellule à une case donnée donnée sera vivante (état 1) ou morte (état 0) à la génération suivante. Commençons par la première fonction. Comme cet automate est en deux dimensions, il faudra créer deux boucles pour parcourir et tester l'état de l'ensemble des cases $(i; j)$ du tableau. Une autre matrice (matrice finale) de même dimensions que la première, avec des valeurs vides sera créée avant la boucle, on y écrira par dessus les cellules qui sont en vie à la génération suivante.

Ensuite, il faut créer la fonction qui sera appelée durant la boucle permettant de tester le voisinage pour déterminer la vie ou la mort de la cellule. Cela se fait en récupérant les valeurs de chaque voisin à tester (0 / 1), et en les additionnant pour compter le nombre de cellules vivantes, ce qui se fait par exemple en envoyant à cette fonction une matrice de $3 \times 3$ cases (où la cellule à tester est au centre), ou bien en envoyant individuellement les huit valeurs autour.

À partir de ça, il suffit d'écrire des conditions *if* à l'intérieur de la boucle afin de faire fonctionner les règles de l'automate : si la cellule testée est vivante, et qu'exactement deux ou trois de celles qu'il y a autour également, elle le reste, on marque alors un 1 dans la case $(i; j)$ de la matrice finale. Idem si la cellule est morte et qu'exactement trois de ses voisins soient vivants. Sinon, on ne fait rien, puisqu'on a déjà défini la matrice finale comme étant rempli de zéros. Notez que si, pour la fonction qui teste le nombre de voisins vivants, vous avez choisi d'envoyer une matrice de 3 par 3, vous devez faire attention à ne pas compter par erreur la cellule au centre, par exemple en testant si la valeur est égale à 3 ou 4 et non 2 ou 3.

Le programme se structure donc à la création d'une matrice initiale, d'une matrice de même dimensions vide, de deux boucles imbriquées permettant de tester l'état de chaque cellule, de leur voisins, puis de déterminer leur vie ou leur mort à la période suivante et écrire ces états dans la matrice finale.

Enfin, nous pouvons permettre la possibilité de ne lancer non pas une itération, mais plusieurs en une fois, en créant une fonction qui prend pour argument la matrice initiale et le nombre de périodes T désiré, et qui enclenche une boucle exécutant T fois les instructions écrites précédemment.

## 4.2 Méthode 2

L'algorithme que nous avons vu fonctionne bien, mais des complications peuvent apparaître, notamment pour la gestion des cellules aux bords de la matrice. La matrice d'un automate cellulaire est en effet périodique, par exemple un vaisseau traversant le bord du bas se retrouvera en haut. Il existe une autre méthode, permettant de s'affranchir de ce problème, et d'écrire un code bien plus court et simple. Le seul inconvénient de cet algorithme est qu'il ne peut pas vraiment être implémenté facilement dans les langages de bas niveau tel que le C++ ou le Java. Il faut en effet un langage gérant les matrices et offrant dès le départ de nombreuses fonctions toutes faites, comme le MatLab.

Ici, il consiste à créer des *copies décalées* de la matrice initiale, de sorte que les voisins se retrouvent dans ces matrices temporaires à la place de la cellule centrale. Ainsi, il faudra en créer huit copies, une pour la case en haut à gauche, une pour la case en-dessus, etc. Ces huit cas de décalage peuvent par ailleurs être stockées dans un autre tableau. Ensuite, il s'agit d'additionner toutes ces matrices, ce qui aura pour effet de faire correspondre à chaque cellule de la matrice initiale le nombre exact de voisins. La suite est donc triviale, la matrice finale peut être déterminée facilement par la double boucle et quelques conditions, et la fonction permettant plusieurs périodes reste inchangée.

Cette structure peut être généralisée, au sens où, pour implémenter d'autres types d'automates cellulaires, il suffira de changer les règles de décalage, ainsi que les conditions à tester. Par exemple, pour la règle de parité, un autre type d'automate, il suffit de ne décaler que dans les quatre directions, sans les diagonales, à la place des huit nécessaires pour le jeu de la vie, et de changer la condition à tester (la somme des quatre voisins doit être impaire pour que la cellule prenne la valeur 1).

## 4.3 Résultats

*Les deux implémentations ont été faites sous MatLab [2]. Elles fonctionnent aussi bien l'une que l'autre.*

Les quatre dernières figures montrent ce qui peut être obtenu après quelques efforts...

# Références

[1] Wikipédia, articles Automate cellulaire et Jeu de la vie. http://fr.wikipedia.org/, consulté le 16 avril 2013.

[2] Stéphane Balac. Débuter avec MATLAB, 2001. http://perso.univ-rennes1.fr/stephane.balac/matlab/matlab.pdf, consulté le 17 avril 2013. Pour apprendre le MatLab.

[3] Bastien Chopard. Page du Prof. Chopard Bastien, chercheur et professeur à l'UNIGE, contenant un certain nombre d'animations d'automates cellulaires. http://cui.unige.ch/~chopard/CA/Animations/img-root.html, consulté le 17 avril 2013.

[4] Stephen Wolfram. *A new kind of science*. Wolfram Media, 2002.

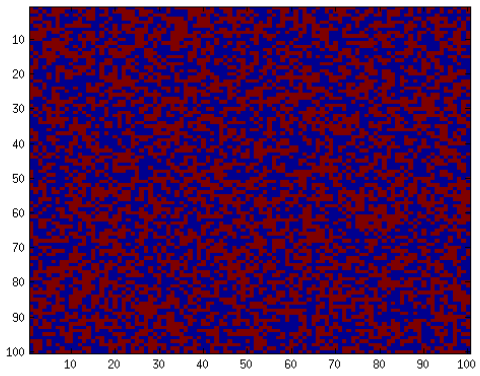



FIGURE 2 – La matrice initiale. Le rouge représente l'état mort (0), le bleu l'état vivant (1)

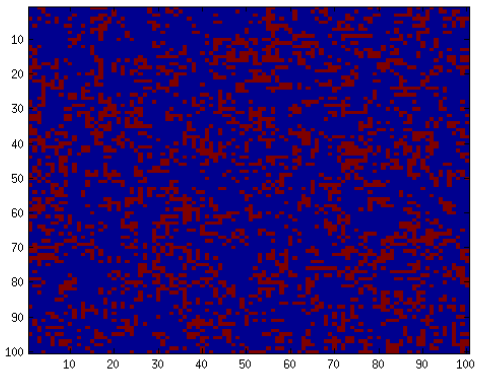



FIGURE 3 – Après une itération

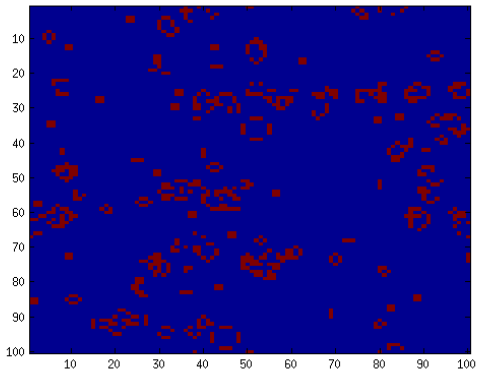



FIGURE 4 – Après cent itérations

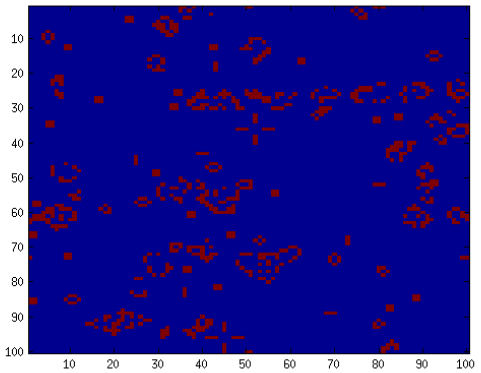



FIGURE 5 – Après cent-une itérations. On peut y voir entre autres les blocs de la génération précédente qui sont restés, et les oscillateurs à trois cellules, ce qui montre bien que le code fonctionne.