TP 5 : Règle de parité et Jeu de la Vie

Cours de modélisation numérique

Vendredi 24 mars 2017

Le but de cette série est d'implémenter un automate cellulaire évoluant selon la *règle de parité* ainsi que celle du *Jeu de la Vie*. Dans le cas du premier, nous essayerons de mettre à jour quelques propriétés un peu plus théoriques.

La règle de parité

La règle de parité, introduite par E. Fredkin au début des années 70, est une dynamique qui, malgré sa simplicité, offre un comportement très riche. Avec cette règle, les cellules ne peuvent avoir que deux états : 0 ou 1. L'état d'une cellule au temps $t+1$ est défini par l'état de ses quatre voisins au temps t selon la fonction suivante :

$$\psi_{t+1}(i,j) = \psi_t(i-1,j) \oplus \psi_t(i+1,j) \oplus \psi_t(i,j-1) \oplus \psi_t(i,j+1)$$

où l'opérateur $\oplus$ représente la somme modulo 2. Dans le cas de cette règle le voisinage considéré est celui de von Neumann (fig. 1). Nous considérerons ici des conditions de bord périodiques, c'est-à-dire que la grille peut être envisagée comme un tore.

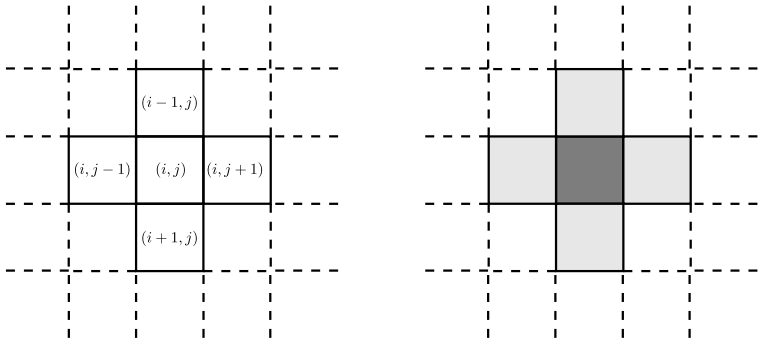

FIGURE 1 – *Illustration du voisinage de von Neumann utilisé pour la règle de parité*

Le Jeu de la Vie

Le Jeu de la Vie est un automate cellulaire proposé par John H. Conway en 1970 qui doit sa popularité jamais démentie à son caractère censément réaliste. Comme dans la règle de parité, les cellules ne peuvent prendre que des valeurs binaires 0 (mort) ou 1 (vivant), mais au contraire de celle-ci on travaille ici avec le voisinage de Moore, c'est-à-dire que chaque cellule interagit avec ses huit voisins (figure 2).

La dynamique est définie comme suit :

- une cellule vivante avec moins de deux voisins vivants meurt
- une cellule vivante avec deux ou trois voisins vivants vit
- une cellule vivante avec plus de trois voisins vivants meurt
- une cellule morte avec exactement trois voisins vivants vit

Cette règle est appliquée de manière synchrone à toutes les cellules de l'automate. Comme précédemment, la topologie est périodique.

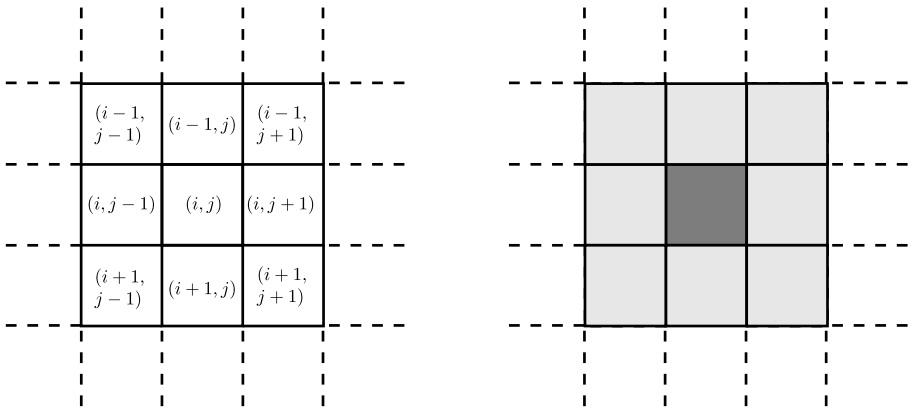

FIGURE 2 – *Illustration du voisinage de Moore utilisé pour le Jeu de la Vie*

Travail à faire

Implémentation

On vous demande d'implémenter ces deux automates, en prenant en compte les quelques contraintes suivantes :

- L'automate doit avoir la forme d'une matrice carrée. Vous devrez utiliser les structures de `numpy`. Comme dans la série 2, à l'exception de la boucle temporelle, évitez d'utiliser des boucles `for` pour accéder à l'état des voisins d'une cellule.
- Vous devrez utiliser une *table de lookup* (LUT). Une LUT est une structure de données stockée en mémoire, employée pour remplacer un calcul par une opération plus simple de consultation. Dans le cas de cette série le calcul

à remplacer est celui de l'état d'une cellule au temps t en fonction de son voisinage au temps $t-1$. Le plus simple consiste à utiliser un tableau dont chaque indice correspond à un état possible du voisinage. Vous devez donc trouver une manière d'encoder les différents états en un entier qui sera utilisé comme indice pour trouver la valeur correspondante à cet état dans un tableau.

— Votre programme doit prendre en entrée le nom d'un fichier décrivant l'état initial de votre automate [1], et produire en sortie un fichier du même format représentant l'état de l'automate au temps t_{max}. Le nombre d'itérations t_{max} doit être laissé au choix de l'utilisateur.

Questions sur la règle de parité

Faites évoluer la règle de parité pour un nombre d'itérations étant une puissance de 2 ($2^k, k = 1, 2, 3, \ldots$). Si votre implémentation est correcte, vous devriez observer que le motif initial est translaté quatre fois de 2^k sites, une fois dans chaque direction. Démontrez mathématiquement que un motif initial d'une cellule est dupliqué quatre fois (haut, bas, gauche, droite) après 2 itérations.

Faites ensuite évoluer un automate avec la règle de parité dont la longueur L d'un côté est une puissance de 2 avec un nombre d'itérations égal à $L/2$. Décrivez ce que vous observez et tentez de l'expliquer en vous aidant de la réponse à la question précédente. L'argument suffit, il n'est pas nécessaire de procéder à une démonstration mathématique.

Avec ces résultats, pouvez-vous proposer une manière astucieuse pour connaître l'état d'un AC évoluant selon cette règle au temps t sans avoir à calculer toutes les itérations intermédiaires ?

Rendu

Vous disposez de deux semaines pour rendre le rapport et le code sur chamilo, c'est-à-dire jusqu'au vendredi 7 avril. Veuillez vous référer au document `guidelines` disponible sur chamilo pour la rédaction du TP.

1. Le format de ce fichier doit être le suivant :
— chaque ligne du fichier correspond à une rangée de la matrice représentant l'automate
— chaque élément de la matrice est séparé par un espace
— les éléments de la matrice prennent valeur 1 ou 0

Annexe : Configurations du Jeu de la Vie

Afin de tester diverses configurations intéressantes de l'automate, il est conseillé d' implémenter un décompositeur analytique [2] capable d'importer une configuration du format `lif`. Dans ce format les 0 sont représentés par des « . » et les 1 par des « ∗ ». Toute ligne commençant par un « # » est un commentaire et ne doit donc pas être interprétée. Par exemple le contenu suivant :

```
# un motif stable
.
**.
**.
.
```

produit le motif de la figure 3. Vous pouvez tester votre automate avec diverses

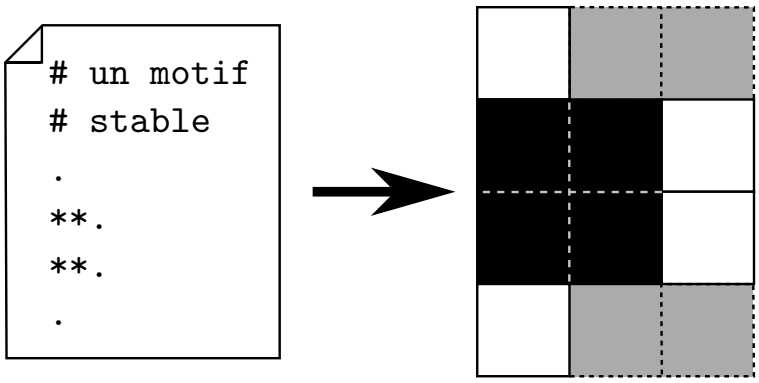

FIGURE 3 – *Un motif stable du jeu de la vie. A gauche le motif au format* `lif`*. A droite les case noires représentent les cellules vivantes et les case blanches les cellules mortes. Les valeurs des cellules grises (vivantes ou mortes) ne changent pas les propriétés du motif.*

configurations disponibles à l'adresse suivante :

`http://www.radicaleye.com/lifepage/patterns/contents.html`.

Pour chaque configuration, le site propose un *applet* Java illustrant le comportement de l'automate.

2. Que vous connaissez probablement sous le nom anglais de *parser*.