Automates cellulaires

PSI 2022/2023

Les automates cellulaires ont été inventés par Stanislaw Ulam au laboratoire de Los Alamos dans les années 1940. Un automate cellulaire consiste en une grille régulière de « cellules » contenant chacune un « état » choisi parmi un ensemble fini (par exemple 0 pour mort et 1 pour vivant) et qui peut évoluer au cours du temps. L'état d'une cellule au temps $t+1$ est fonction de l'état au temps t d'un nombre fini de cellules appelé son « voisinage » selon les "règles du jeu". À chaque nouvelle unité de temps, les mêmes règles sont appliquées simultanément à toutes les cellules de la grille, produisant une nouvelle « génération » de cellules.

Étudiés en mathématiques, en physique théorique et en informatique, le modèle des automates cellulaires est remarquable par l'écart entre la simplicité de sa définition et la complexité que peuvent atteindre certains comportements macroscopiques : l'évolution dans le temps de l'ensemble des cellules ne se réduit pas (simplement) à la règle locale qui définit le système. À ce titre il constitue un des modèles standards dans l'étude des systèmes complexes. Nous nous proposons dans la suite d'étudier deux automates particulièrement simples.

I Le jeu de la vie

Dans les années 1970, un automate cellulaire à deux dimensions et à deux états, nommé le jeu de la vie, inventé par John Conway de l'université de Cambridge, connut un grand succès, particulièrement parmi la communauté scientifique naissante.

I.1 Règles du jeu

Le cadre du jeu est un quadrillage carré. Il existe deux états possibles :

— éteinte ou morte : couleur blanche (valeur de la case : 0),
— allumée ou vivante : couleur noire (valeur de la case : 1).

Les règles qui régissent l'évolution de ces cellules sont simples. Le projet s'intéresse à l'évolution de ces cellules par rapport au temps, en procédant de génération en génération. Le voisinage de ces cellules est défini par ses huit cases adjacentes.

A chaque génération, la règle est la suivante :

- une cellule ”vivante” reste en vie, si elle a deux ou trois voisins vivants, parmi son voisinage direct,
- une cellule meurt de solitude si elle possède moins de deux voisins vivants, ou elle meurt étouffée si elle possède plus de trois voisins vivants,
- une cellule morte renaît si elle a exactement trois voisins vivants.

I.2 Configurations initiales

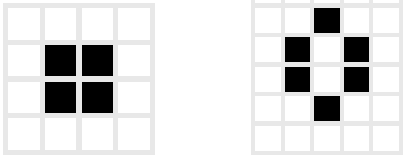
I.2.1 Configuration aléatoire

Le système initial peut être une configuration dont l'état des cellules est défini aléatoirement à l'aide des fonctions *random*.

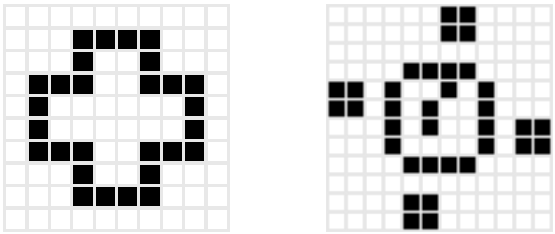
I.2.2 Formes de bases

Il existe des formes de base ayant des propriétés amusantes

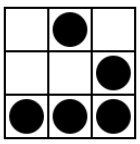
Les formes stables Il s'agit de formes qui ne varient pas au cours des générations. En voici deux exemples : le ”bloc” et la ”ruche”.



Les oscillateurs Ce sont des formes périodiques qui reprennent leur apparence initiale après quelques générations. Ci-dessous : ”la croix” et ”l'horloge”.

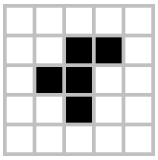


Les navires Ce sont des structures qui se déplacent dans le jeu sans se déformer. Ci-dessous, un ”planeur”.



I.3 Programmation

On prendra pour configuration de départ la configuration dite "pentomino R" ci-dessous. On la modélise à l'aide d'un tableau :



$$P = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Pour voir comment évolue cette configuration suivant les règles du jeu de la vie, il faut compter le nombre de voisins vivants pour l'ensemble des cases du tableau. Par exemple, la case de coordonnées (2,2) est dans l'état vivant 1 et possède 4 voisins vivants. Soit $V_{i,j}$ le nombre de voisins vivants de la case de coordonnées (i, j).

1. Ecrire une fonction `init(N)` prenant en argument un entier N et renvoyant un tableau numpy de taille $N \times N$ contenant uniquement des zéros et un pentomino en haut à gauche du tableau. On pourra utiliser la fonction `np.zeros((N,N))` de la librairie `numpy` qui initialise un tableau rempli de zéros.

2. Ecrire une fonction `voisins(i,j)` qui renvoie `Vij`, le nombre de cellules vivantes autour de la case de coordonnées (i, j). Pour les cellules situées sur les bords, on prendra des conditions aux bords périodiques. En particulier,si i (ou j) est égal à $N-1$, alors $i+1$ (ou $j+1$) est égal à 0.

3. Ecrire une fonction `update(Pij,Vij)` qui renvoie la nouvelle valeur de `P[i,j]` en fonction de son ancienne valeur `Pij` et de `Vij`

4. Pourquoi est-il nécessaire de stocker les nouvelles valeurs de `P[i,j]` dans une copie et ne pas modifier directement les valeurs du tableau `P` ?

5. La fonction `jeu_vie(Temps)` écrite ci-dessous modélise le jeu de la vie sur `Temps` itérations. A vous de remplir les trous !

```python
import matplotlib.pylab as plt
from matplotlib import cm
from copy import deepcopy
import numpy as np

def jeuvie(Temps):
    P = init()
    plt.figure(1) # ouvre une fenetre graphique dans Windows
    plt.xticks([]) # efface l'axe des abscisses
    plt.yticks([]) # efface l'axe des ordonnees
    imag=plt.imshow(P,interpolation='nearest',cmap=cm.binary) # affiche l'image initiale en noir et blanc
    plt.draw()
    N = len(P)
    t=0
    while t<Temps:
        Pn = deepcopy(P) # fait une copie temporaire de P
       ..............
       ..............
       ..............
       ..............
        P = deepcopy(Pn) # recopie les nouvelles valeurs dans P
        imag.set_data(P) # On fait un update de l'image
        plt.pause(1e-6)
        plt.draw()
```

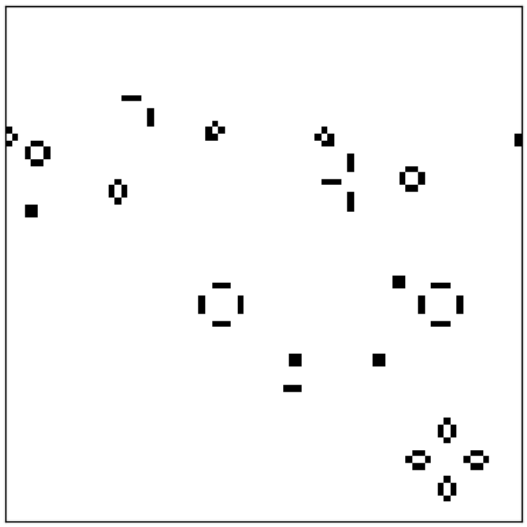


FIGURE 1 – *Evolution du pentomino sur un carré de taille* 80×80 *après 1100 itérations.*

II Simulation d'un feu de forêt

Le mot percolation vient du latin *percolatio* qui signifie filtration. Ce terme se rencontre dans un grand nombre de situations, il évoque les notions de propagation dans un milieu de structures aléatoires partiellement interconnectés. Les exemples les plus courants sont : la conduction dans un milieu granulaire formés de grains conducteurs et isolants, la perméabilité d'une roche poreuse pour l'extraction du pétrole ou de l'eau dans un filtre à café, la propagation des épidemies ou encore la propagation des incendies. C'est ce dernier phénomène que nous allons étudier ici.

II.1 Règles de propagation du feu

Le système d'étude est un carré de côté n. Un arbre est représenté par un 1 et un espace vide par un 0. Les arbres de la forêt ne sont pas plantés de manière homogène mais de manière aléatoire. On suppose que la forêt est de densité d, c'est à dire que pour chaque site, il y a une probabilité d d'avoir un arbre et une probabilité $1-d$ d'avoir un espace vide. Une cellule du système d'étude est prise au hasard et est mise en feu. Elle prend alors la valeur 100.

Un automate cellulaire simule la propagation des flammes. Les règles de propagation du feu sont les suivantes : une cellule contenant un arbre prend feu à l'instant t si l'une au moins de ces 8 voisines est en feu au temps $t-1$. Les cellules en feu à l'instant $t-1$ tombent en cendre à l'instant t. Elles prennent alors la valeur 10. Les cellules vides ne peuvent pas prendre feu. Il s'agit là d'un atomate cellulaire à quatre états : celulle vide (0), arbre vivant (1), arbre en feu (100), arbre mort (10).

Ainsi, pour savoir si un arbre possède un voisin en feu, il suffit de sommer la valeur des différentes cellules voisines (comme pour le jeu de la vie) ; si cette somme est supérieure ou égale à 100, la cellule considérée prend nécessairement feu à la génération suivante.

II.2 Programmation

On importera les modules suivants :

```
1 import matplotlib.pyplot as plt
2 from copy import deepcopy
3 import numpy as np
4 import random as rd
```

On pourra suivre le plan suivant :

1. Ecrire une fonction `init(n,d)` qui initialise une forêt de taille $n \times n$ et de densité d. Pour cela, pour chaque site aléatoire, on tirera un nombre aléatoire compris entre 0 et 1 à l'aide de la fonction `random()` de la librairie `random`. Si ce nombre est inférieur à d, on plantera un arbre. Sinon, on ne fera rien.

2. Ecrire une fonction `nombre_feu(P)` qui renvoie le nombre de cellules en feu à l'instant t.

3. Ecrire une fonction `nombre_brules(P)` qui renvoie le nombre de cellules brulées à l'instant t.

4. Ecrire une fonction `depart_feu(P)` qui tire deux nombres au hasard i et j compris entre 0 et $n-1$ et met en feu la cellule d'indice (i, j). On pourra utiliser la fonction `randint(0,n-1)` du module `random`.

5. Ecrire une fonction `somme(i,j)` qui renvoie la somme des valeurs contenues dans les cellules voisines de (i, j). On ne prendra plus de conditions aux bords périodiques et on veillera à traiter les cas particuliers des cellules situées dans les coins et sur les bords.

6.

7. Ecrire une fonction `update(Pij,Vij)` qui renvoie la nouvelle valeur de `P[i,j]` en fonction de son ancienne valeur `Pij` et de la somme des valeurs voisines

8. On considère la fonction `image(P)` ci-dessous. Qu'effectue-elle ?

```
def image(P):
    xA,yA = [],[]
    xF,yF = [],[]
    xE,yE = [],[]
    n = len(P)
    for i in range(n):
        for j in range(n):
            if P[i,j] == 1:
                xA.append(i)
                yA.append(j)
            elif P[i,j] == 100:
                xF.append(i)
                yF.append(j)
            elif P[i,j] == 10:
                xE.append(i)
                yE.append(j)

    plt.plot(xA,yA,'g+') # arbres vivants representes par des plus verts (green +)
    plt.plot(xF,yF,'rx') # arbres en feu representes par des croix rouges (red x)
    plt.plot(xE,yE,'ko') # arbres morts representes par des ronds noirs (black o)
    plt.pause(1e-6)
    plt.draw()
```

9. Remplir les trous de la fonction `incendies(n,d)` ci-dessous qui simule l'évolution d'un feu de forêt.

```
def incendie(n,d):

    P =init(n,d)
    plt.figure(1)
    plt.xticks([])
    plt.yticks([])
    depart_feu(P) # On allume le feu
    nfeu = nombre_feu(P)
    image(P) # On affiche la foret initiale

```

```
    while .............:

        Pn = np.zeros((n,n))

  ..........................
  ..........................
  ..........................
  ..........................


        P = deepcopy(Pn)
        image(P)
```

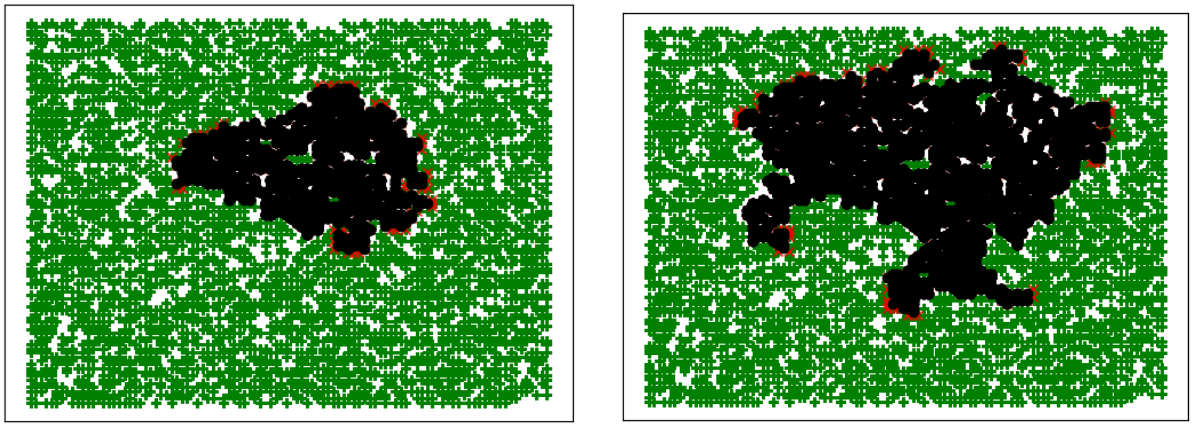


FIGURE 2 – *Evolution d'un feu de forêt pour une densité* $d = 0.45$