Dynamique des automates cellulaires

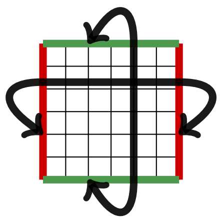
Ce TP tient lieu de contrôle continu.
Le travail est à effectuer seul ou en binôme. Tout regroupement de taille supérieure est exclu.

Il est à déposer sur ecampus au plus tard le lundi 30 avril sous la forme d'une archive compressée comprenant :

1. **le programme**
2. **un fichier `readme.txt` avec les noms des deux membres du binôme**
3. **un rapport au format `pdf` présentant les résultats d'expérimentation.**

Objectif. Visualiser la dynamique d'une famille d'automates cellulaires définissant un modèle simplifié de cellules en milieu excitable.

Tous les automates envisagés ici sont à deux dimensions. L'espace consiste en une grille carrée de taille `tail` × `tail`. Cet espace est torique : les bords horizontaux bas et haut de la grille sont adjacents, de même les bords verticaux gauche et droite.



Chaque cellule (case) de la grille n'a qu'une mémoire finie. Avec un nombre d'états borné par `nbEtats`, les états de chaque cellules sont numérotés de `0` à `nbEtats - 1`.

1 Préliminaire.

Récupérer le fichier `ac.py` qui contient la classe `CA2D`. Cette classe dispose, entre autres,

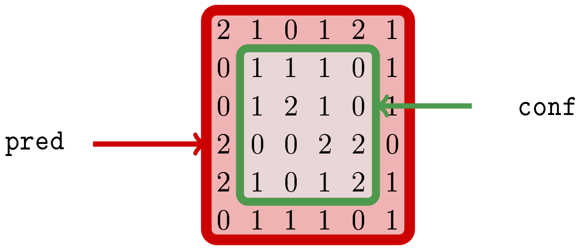
- d'un attribut `tail` qui définit la taille de la grille carrée
- d'un attribut `nbEtats` qui spécifie les états possibles de chaque cellule
- d'un attribut `conf` qui représente la configuration courante, c-à-d, l'ensemble des états de chaque cellule de la grille, sous la forme d'un tableau numpy à deux dimensions.

Qu 1. Modifier la méthode `initAlea` de la classe `CA2D` de façon à ce que la configuration initiale soit choisie aléatoirement.

```
In [1]:  ac = CA2D(6,4)
In [2]:  print(ac)
1 3 1 1 0 2
2 0 3 2 3 3
1 1 1 1 1 2
3 2 0 0 1 3
2 1 2 0 2 2
1 1 0 3 0 3
```

À partir de cette configuration initiale, les cellules évoluent de façon locale, uniforme et synchrone. Du fait que les cellules évoluent en parallèle, on introduit un attribut `pred` pour enregistrer la configuration courante `conf` avant de calculer la nouvelle configuration.

Considération technique Comme l'espace est un tore, pour traiter l'évolution des cellules du bord, on peut coller les `r` cellules du bord gauche de `conf` à droite de `pred`, les `r` cellules du bord droit de `conf` à gauche de `pred`, les `r` cellules du bord haut de `conf` en bas de `pred` et les `r` cellules du bord bas de `conf` en haut de `pred`. C'est ce que réalise la méthode `calclPred` de `CA2D`. L'extraction du voisinage d'une cellule peut alors se faire de manière uniforme que cette cellule soit ou non sur un bord.



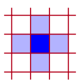
,

2 Le modèle de Greenberg-Hastings

En premier lieu, on veut simuler l'évolution de l'automate de Greenberg-Hastings.
Cet automate est un automate cellulaire à `3` états : quiescent (`0`), excité (`1`) et réfractaire (`2`). Le système évolue selon le principe suivant : à chaque étape, une cellule excitée devient réfractaire, une cellule réfractaire devient quiescente et une cellule quiescente devient excitée si elle a au moins une de ses `4` voisines dans l'état excité.

Formellement, l'automate cellulaire de Greenberg-Hastings a les caractéristiques suivantes :

- Le réseau est de dimension `2`
- Le voisinage est celui de von Neumann : le voisinage de chaque cellule est constitué d'elle-même et de ses `4` voisines adjacentes

- Il y a trois états `0`, `1` et `2`
- La règle de transition locale est définie comme suit :
  - si la cellule est dans l'état `1` alors elle passe dans l'état `2`
  - si la cellule est dans l'état `2` alors elle passe dans l'état `0`
  - si la cellule est dans l'état `0` alors

$$\begin{array}{ll} \text{elle passe dans l'état } \texttt{1}\,, & \text{si au moins une de ses } \texttt{4} \text{ voisines est dans l'état } \texttt{1} \\ \text{elle reste dans l'état } \texttt{0}\,, & \text{sinon} \end{array}$$

Qu 2. Définir la méthode `voisins(self, lig, col)` de la classe `GreenbergHastings` qui retourne la liste des états des `4` voisines adjacentes à la cellule `(lig, col)` qui sont mémorisées dans l'attribut `pred`.

```
In [2]:  ac = GreenbergHastings(4)
In [3]:  print(ac)
1 1 1 0
1 2 1 0
0 0 2 2
1 0 1 2
In [4]:  ac.calculPred()
In [5]:  print(ac.pred)
[[2, 1, 0, 1, 2, 1],
[0, 1, 1, 1, 0, 1],
[0, 1, 2, 1, 0, 1],
[2, 0, 0, 2, 2, 0],
[2, 1, 0, 1, 2, 1],
[0, 1, 1, 1, 0, 1]]
In [6]:  ac.voisins(1,2)
Out[6]:  array([1, 2, 2, 0])
In [7]:  ac.voisins(0,0)
Out[7]:  array([1, 0, 1, 1])
```

Qu 3. Définir la méthode `transition(self, lig, col)` de la classe `GreenbergHastings` qui met à jour l'état de la cellule `(lig, col)`. Précisément, cette fonction calcule `self.conf[lig, col]` le nouvel état de la cellule `(lig, col)` à partir de son état et des états de son voisinage.

Qu 4. Afficher les `5` premières étapes de l'évolution de l'automate de Greenberg-Hastings sur une entrée aléatoire.

3 Interlude

Le fichier `application.py` fournit les ingrédients pour visualiser l'évolution du système avec `matplotlib`.

4 Généralisation du modèle de Greenberg-Hastings

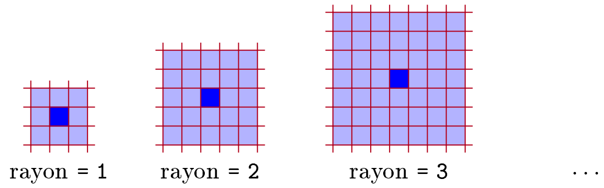
On veut maintenant généraliser le modèle précédant en paramétrant à la fois le nombre d'états, le voisinage et le seuil à partir duquel une cellule passe de l'état quiescent `0` à l'état excité `1`.

L'automate cellulaire de Greenberg-Hastings généralisé se définit comme suit.

- Il est spécifié par *trois paramètres* :
  1. le nombre d'états `nbEtats`,
  2. le rayon du voisinage `rayon`
  3. le seuil d'excitation `seuil`.
- Ses caractéristiques sont alors les suivantes :
  - Le réseau est de dimension `2`

- Le voisinage est celui de Moore d'ordre `rayon` : le voisinage de chaque cellule est constitué d'elle-même et de ses voisines à distance au plus `rayon`

$\mathcal{V}_{\texttt{rayon}}$ `= {(x, y) : max(x, y) <= rayon}`

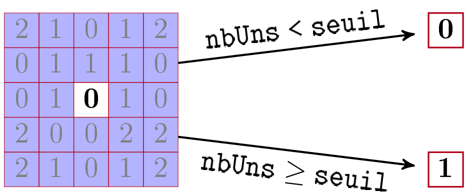


- Il y a `nbEtats` états numérotés `0`, `1`, ..., `nbEtats - 1`
- La règle de transition locale est définie par :
  * si la cellule est dans un état `i` non nul alors elle passe dans l'état `(i+1)%nbEtats`

`0`   `1` ⟶ `2` ⟶ `3` - - - → `nbEtats - 1` → `0`

  * si la cellule est dans l'état `0` alors

    elle passe dans l'état `1`, si le nombre de ses voisines dans l'état `1` dépasse `seuil`
    elle reste dans l'état `0`, sinon



Qu 5. Écrire une classe `GHGeneralise` qui hérite de la classe `CA2D` et qui permet de simuler l'évolution de cet automate. À l'initialisation, les paramètres sont la taille de la grille, le nombre d'états, le rayon et le seuil spécifiques l'automate.

Qu 6. Définir une instance de la classe `GHGeneralise` qui a pour paramètres `nbEtats = 8`, `rayon = 3`, `seuil = 5`. Quels phénomènes d'émergence observe-t-on au cours de son évolution ?

Qu 7. En faisant varier les paramètres `nbEtats`, `rayon`, `seuil` et `tail`, proposer d'autres automates dont la dynamique présente également des phénomènes d'émergence.

Qu 8. Faire un bilan des expérimentations réalisées.