TP N° 5 : (TP Noté) Jeu de la vie

Pour ce travail vous devez déposer un **unique** fichier au format `.ipynb`, dont le nom est `tp_note_hmla310_group_?_prenom_nom.ipynb`, le tout en minuscule. Vous remplirez votre nom, prénom et le groupe qui vous concerne (?=A,B ou C) de manière adéquate ; voir aussi le fichier `TP-Note-draft.ipynb` fourni) sur le site du cours HLMA310 pour bien démarrer.

Vous devez charger votre fichier sur Moodle, avant le vendredi 12/10/2018, 23h55.

La note totale est sur **20** points répartis comme suit :

— qualité des réponses aux questions : **14** pts,

— qualité de rédaction et d'orthographe : **1** pts,

— qualité des graphiques (légendes, couleurs) : 1 pt

— style PEP8 valide : **2** pts,

— qualité d'écriture du code (noms de variable clairs, commentaires, code synthétique, etc.) : **1** pt

— Notebook reproductible (i.e., "Restart & Run all" marche correctement sur la machine du correcteur) et absence de bug : **1** pt

Les personnes qui n'auront pas soumis leur devoir sur Moodle avant la limite obtiendront **zéro**.

Rappel : aucun travail par mail ne sera accepté !

Exercice 1. Chaîne de caractères

Télécharger le fichier `TP-Note-draft.ipynb` sur le site du cours, pour obtenir une ébauche du fichier au format attendu.

1) (**0.5pt**) Créer la chaîne de caractères `filename` correspondant au nom de votre fichier et qui aura le format suivant : `tp_note_hmla310_groupe_?_prenom_nom.ipynb` (le tout en minuscule, avec ?=A, B ou C). On pourra se servir du fichier `TP-Note-draft.ipynb` disponible.
2) (**0.5pt**) Créer une variable `taille_str` qui compte le nombre de caractères dans la chaîne de caractères `filename`.
3) (**0.5pt**) Créer une variable `ma_graine` qui vaut le reste de la division euclidienne de `taille_str` par 5 (exemple pour le cas `prenom=joseph, nom=salmon, group=a` ce nombre est 3).

Exercice 2. Le jeu de la vie

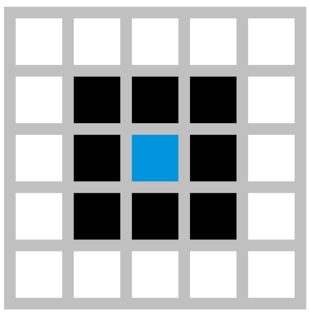


Figure 1 – Cellule (au centre, en bleu), et ses voisins (autour, en noir)

Le **jeu de la vie** est un automate cellulaire mis au point par le mathématicien britannique John Horton Conway en 1970. Il constitue l'exemple le plus connu d'un automate cellulaire. Le "jeu" est en fait un jeu à zéro joueur, ce qui signifie que son évolution est déterminée par son état initial et ne nécessite aucune intervention de la part de joueurs. On interagit avec le jeu de la vie en créant une configuration initiale ; il ne reste plus alors qu'à observer son évolution.

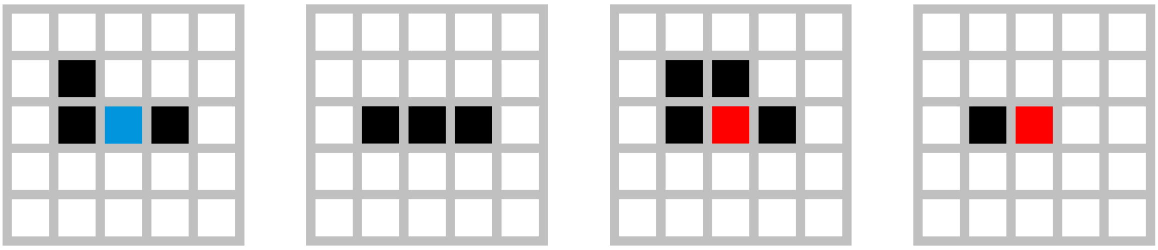
L'univers du jeu est une grille orthogonale bidimensionnelle infinie (dans la suite du TP on la supposera que la grille est carrée et de taille finie pour éviter toute difficulté ; **on supposera aussi que le pourtour de la grille est toujours inactif/mort**) de cellules carrées, chacune d'entre elles se trouvant dans l'un des deux états possibles :

— vivant

— mort

Chaque cellule interagit avec ses huit voisins, qui sont les cellules directement adjacentes horizontalement, verticalement ou en diagonale, comme indiqué sur la Figure 1. Ã chaque pas dans le temps, les transitions suivantes se produisent :

a) Toute cellule morte ayant exactement 3 voisins vivants devient une cellule vivante (**naissance**), *cf.* Figure 2a
b) Toute cellule vivante avec 2 ou 3 voisins vivants vit inchangée jusqu'à la génération suivante (**équilibre**), *cf.* Figure 2b
c) Toute cellule vivante ayant 4 voisins vivants (**mort par étouffement**), *cf.* Figure 2c
d) Toute cellule vivante ayant 0 ou 1 voisin vivant décède (**mort par isolement**), *cf.* Figure 2d.



(a) Naissance (b) Équilibre (c) Étouffement (d) Isolement

FIGURE 2 – Quatres configurations possibles pour la cellule centrale : naissance (bleu), équilibre, mort par étouffement et par isolement (rouge).

Le modèle initial constitue la "graine" du système. La première génération est créée en appliquant les règles ci-dessus simultanément à chaque cellule de la graine - les naissances et les décès se produisent simultanément. Ainsi chaque génération est une fonction de la précédente. Les règles continuent d'être appliquées de manière répétée pour créer d'autres générations[1].

ATTENTION : La "carte" (on utilisera soit une liste de liste, soit une matrice) est équipée d'une frontière de 0 qui permet d'accélérer un peu en évitant d'avoir des tests spécifiques pour les frontières lors du comptage du nombre de voisins. Ainsi la "couronne" du tour de la carte sera fixée à zéro.

Implémentation sans numpy

On va fournir dessous le code pure Python pour coder ce jeux. Dans la suite on va coder les cellules vivantes par des 1 et les cellules mortes par des 0. Tout d'abord on définit la fonction `calcul_nb_voisins` :

```
def calcul_nb_voisins(Z):
    forme = len(Z), len(Z[0])
    N = [[0, ] * (forme[0]) for i in range(forme[1])]
    for x in range(1, forme[0] - 1):
        for y in range(1, forme[1] - 1):
            N[x][y] = Z[x-1][y-1]+Z[x][y-1]+Z[x+1][y-1] \
                    + Z[x-1][y] + 0          +Z[x+1][y]  \
                    + Z[x-1][y+1]+Z[x][y+1]+Z[x+1][y+1]
    return N
```

1. Plus de détails pour les curieux : https://fr.wikipedia.org/wiki/Jeu_de_la_vie

4) (**1pt**) Appliquer la fonction précédente à la liste (de liste) `Z` suivante, et expliquer ce que représente la sortie obtenue `N=calcul_nb_voisins(Z)`.

```
Z = [[0,0,0,0,0,0],
     [0,0,0,1,0,0],
     [0,1,0,1,0,0],
     [0,0,1,1,0,0],
     [0,0,0,0,0,0],
     [0,0,0,0,0,0]]
```

Définir ensuite la fonction `iteration_jeu` comme suit : et rajouter une *docstring* pour cette fonction décrivant les entrées / sorties et ce que retourne la fonction :

```
def iteration_jeu(Z):
    forme = len(Z), len(Z[0])
    N = calcul_nb_voisins(Z)
    for x in range(1,forme[0]-1):
        for y in range(1,forme[1]-1):
             if Z[x][y] == 1 and (N[x][y] < 2 or N[x][y] > 3):
                 Z[x][y] = 0
             elif Z[x][y] == 0 and N[x][y] == 3:
                 Z[x][y] = 1
    return Z
```

5) (**2pts**) Dans cette question on se propose pour la liste `Z` ci-dessus d'afficher les étapes du jeu de 0 à 9 itérations, en utilisant une boucle `for`. On utilisera la fonction `subplot` de `matplotlib` pour afficher sur 2 lignes et 5 colonnes ces 10 matrices. De plus on devra transformer ces listes en array pour pouvoir utiliser la fonction `imshow` de `matplotlib`.

6) (**1pt**) Que remarquez-vous entre l'itération 0 et l'itération 4 ? Que se passe-t-il après l'itération 7 ?

Implémentation avec numpy

7) (**1pt**) Pour le vecteur `vec` suivant, exprimer ce vaut le vecteur `nb_vect` défini comme suit.

```
vect = np.array([0,1,0,0,1,1])
nb_vect = np.zeros(vect.shape)
nb_vect[1:-1] += (vect[:-2] + vect[2:])
```

Rem : on ne se s'intéresse pas au bord du vecteur, tout comme on ne intéresse pas au bord de la matrice représentant le jeu de la vie.

8) (**3pts**) Se servir la question précédente (et le `slicing` donc) pour créer une fonction `calcul_nb_voisins_np`, cette fois sur des `array`, qui prend en entrée un matrice `Z` et qui ressort le nombre de voisins pour chaque entrée (et qui vaut zéro sur le pourtour). On utilisera donc 8 types de slicing pour obtenir le nombre de voisins.

9) (**1pt**) Créer une fonction `iteration_jeu_np`, similaire à `iteration_jeu` mais qui prend comme entrée sortie des `numpy array` et non plus des listes de listes. On se servira bien sûr de la question précédente pour cela.

10) (**0.5pt**) Créer une fonction `jeu_np` qui prend en entrée une matrice initiale `Z_in` et un nombre d'itérations `nb_iter` et sort une matrice (de même taille que `Z_in`) décrivant l'état du jeu de la vie après `nb_iter` itérations.

11) (**3pts**) Afficher un film (avec la commande `animation.FuncAnimation` de `matplotlib`) qui représente les itérations du jeu de la vie quand on initialise avec la matrice `Z_huge` suivante :

```
Z_huge = np.zeros((100, 100))
Z_np = np.array(
    [[0, 0, 0, 0, 0, 0],
     [0, 0, 0, 1, 0, 0],
     [0, 1, 0, 1, 0, 0],
     [0, 0, 1, 1, 0, 0],
     [0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0]])
Z_huge[10:16, 10:16] = Z_np
```

On pourra consulter notamment l'exemple suivant :
https://jakevdp.github.io/blog/2012/08/18/matplotlib-animation-tutorial/, ainsi que l'aide de `matplotlib`. Il pourra être utile aussi pour cette question d'utiliser la commande magique : `%matplotlib notebook` pour l'affichage des vidéos (ou bien d'utiliser `from IPython.display import HTML`).

12) (**1pt**) Reprendre la question précédente en initialisant de la manière suivante : créer un matrice aléatoire de taille 100×100, remplie de 1 et de 0, et dont la proportion de 1 est (en espérance) égale à `(1+ ma_graine) * 10 / 100` (*e.g.*, pour `prenom=joseph, nom=salmon` ce nombre vaut `prop_active = 0.4`).

13) (**1pt**) Proposer et afficher avec `plt.imshow()` des matrices (et 10 itérations d'un jeu initialisé avec elles) de taille 50×50, ayant les propriétés suivantes :

   — trois matrices simples qui représentent des jeux qui sont fixes dans le temps (configuration stables)

   — une matrice qui représente un jeu dont l'état oscille avec une période de deux (et qui ne comporte que des valeurs non nulles).

14) Question bonus (**2pt**) : Comment modifier `jeu_np` et `iteration_jeu_np, calcul_nb_voisins_np` pour faire ce jeux non plus sur une grille carrée (avec une bordure "éteinte"), mais sur un tore, "à la Pac-Man". Reprendre la question 12) avec cette nouvelle règle.