TP 5
Jeu de la vie

L'objet de ce TP est d'implémenter simplement le « Jeu de la vie » imaginé par John Horton Conway en 1970, et de faire un peu de statistique autour de celui-ci.

1 Introduction

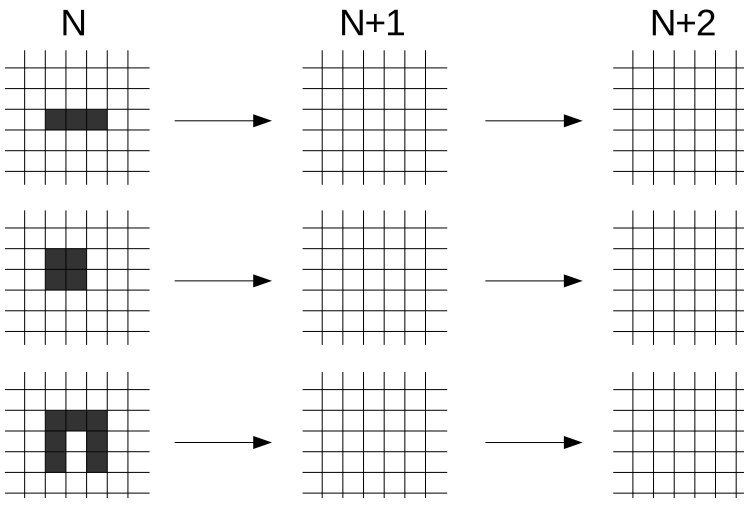
Le jeu de la vie n'est pas à proprement parler un jeu, car il ne nécessite aucun joueur. Étant donnée une configuration initiale donnée, le jeu évolue automatiquement suivant des règles simples.

On se donne une grille de cellules pouvant être dans deux états : mort ou vivant. Chaque cellule possède 8 voisines (gauche, droite, haut, bas et les 4 diagonales). Pour passer de la génération N à la génération $N+1$, l'état de chaque cellule de la grille est modifié selon les règles suivantes :

1. si la cellule est **vivante** et possède **2 ou 3** voisines, elle reste vivante ; sinon, elle meurt
2. si la cellule est **morte** et possède **exactement 3** voisines, elle devient vivante ; sinon, elle reste morte

La terminologie proche de la biologie s'explique par la manière surprenante dont évoluent des motifs simples, mais également par l'interprétation qu'on peut faire des règles ci-dessus : les cellules meurent en cas de sous-population ou surpopulation (règle 1), et peuvent se reproduire (règle 2).

Q1 Appliquer les règles ci-dessus aux motifs ci-dessous.



2 Implémentation

On représente la grille de cellules par un tableau de taille $N \times N$, composé uniquement de 0 et de 1, respectivement pour une cellule morte ou vivante. La première dimension désignera l'axe des abscisses, orienté vers la droite, et la deuxième l'axe des ordonnées, orienté vers le haut. Dans tout le TP on prendra `N = 80`.

Q2 Écrire une fonction `init_jeu_u(N)` prenant en argument un entier N et retournant un tableau de taille $N \times N$ représentant une grille de cellules mortes, sauf au centre où figure le motif « U inversé » de la question précédente. On posera `G = init_jeu_u(N)`

Q3 Écrire une fonction `afficher_jeu(grille)` prenant en argument la grille du jeu et affichant à l'aide de la fonction `plot` du package `matplotlib.pyplot` les cellules vivantes de la grille, qu'on représentera par un carré noir. La tester sur la grille `G`.

Q4 Écrire une fonction `compter_voisins(grille, i, j)` retournant le nombre de voisines vivantes de la cellule (i, j), où i et j désignent respectivement la première et la deuxième coordonnée du tableau `grille`.

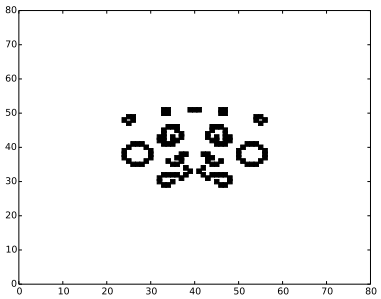
Remarque : on pourra pour simplifier supposer que (i, j) ne désigne pas une cellule du bord de la grille

Q5 Écrire une fonction `iterer_jeu(grille)` appliquant les règles données en introduction pour mettre à jour la grille du jeu. Tester la fonction sur la grille `G`, et comparer avec la troisième figure de la question 1.

Remarque : on pourra ignorer les cellules du bord de la grille

Q6 Modifier la fonction `iterer_jeu(grille)` en `iterer_jeu(grille, k, afficher = False)` pour qu'elle mette à jour k fois la grille, et prenne en argument un booléen indiquant si celle-ci doit afficher la grille après chaque itération. Tester la fonction sur la grille `G`, pour $k = 110$, pour obtenir la configuration ci-dessous.

Remarque : on pourra utiliser la fonction `pause` *du package* `matplotlib.pyplot`*, avec un délai très faible (e.g.* 10^{-15}*)*



3 Un peu de statistiques

Q7 Modifier la fonction `iterer_jeu` de sorte qu'elle renvoie un tuple `(vPop, vMorts, vNaissances)`, où `vPop`, `vMorts`, `vNaissances` sont respectivement des tableaux à `nb_iter` éléments contenant le nombre de cellules vivantes (population), nombre de morts, nombre de naissances à chaque itération.

Exemple : `vMorts[4]` *indique le nombre de morts lors de la* 5^e *mise à jour de la grille,* `vPop[6]` *la population après la* 7^e *mise à jour*

Q8 Tracer sur un même graphique les courbes montrant l'évolution de la population, du nombre de naissances et de décès de cellules à partir de la configuration initiale jusqu'à la 200^e génération.

Q9 Écrire une fonction `init_jeu_aleat(N, p)` retournant une grille où l'état de chaque cellule est obtenu en simulant une variable aléatoire de Bernoulli de paramètre p, sauf pour les cellules du bord qui sont mortes.

Q10 Reprendre la question 8 pour des configurations initiales obtenues aléatoirement, pour $p \in \{0.15, 0.5, 0.6, 0.8\}$ (un graphique par valeur de p), jusqu'à la 800^e génération.

Q11 Tracer une courbe montrant l'évolution de la population **moyenne** à partir de 10 configurations initiales générées aléatoirement avec $p = 0.15$, jusqu'à la 250^e génération.

Q12 Écrire une fonction `calculer_matrice_voisins(grille)` calculant la matrice du nombre de voisines vivantes pour chaque cellule de manière un peu astucieuse. Améliorer la fonction `iterer_jeu` en conséquence. Vérifier que l'approche est effectivement plus efficace. Pourquoi ?

Indication : penser à la contribution de chaque cellule vivante à la matrice des voisines

Q13 Tracer sur un même graphique, pour chaque valeur de p donnée à la question 10, la courbe montrant l'évolution de la population **moyenne** (obtenue à partir de 10 configurations initiales générées aléatoirement), jusqu'à la 250^e génération.

4 Pour s'amuser encore un peu

4.1 Ajoutons de la couleur

Nous allons à présent classer les cellules **vivantes** en 4 catégories différents, et on leur attribuera les couleurs suivantes :

- en **rouge**, les cellules qui étaient déjà vivantes à la génération précédente, et vont mourir à la suivante
- en **vert**, les cellules qui viennent de naître, et vivront encore à la génération suivante
- en **jaune**, les cellules qui viennent de naître, mais vont mourir à la génération suivante

— en **noir**, les cellules qui était déjà vivantes à la génération précédente, et vivront encore à la suivante

Pour afficher à présent la grille avec ce code couleur, on décide de garder en mémoire 3 générations successives de la grille, qu'on appellera respectivement configuration $k-1$ (génération précédente), k (génération courante) et $k+1$ (génération suivante).

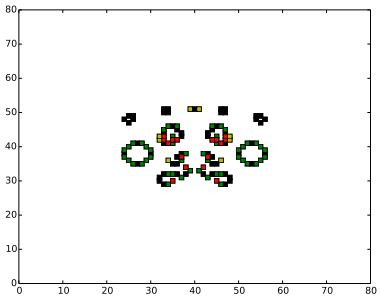
On ajoute donc une dimension à la grille du jeu, de sorte que `G[:,:,0]`, `G[:,:,1]`, `G[:,:,2]` contiennent respectivement les configurations, $k-1$, k, $k+1$.

Q14 Modifier les fonctions d'initialisation pour qu'elles retournent une grille à 3 dimensions, où les `G[:,:,i]` contiennent la même configuration initiale, pour $i \in \{0, 1, 2\}$.

Q15 Modifier la fonction `afficher_jeu` pour qu'elle affiche la configuration courante `G[:,:,1]` en tenant compte des configuration précédente et suivante, avec le code couleur donné précédemment.

Q16 Adapter la fonction `iterer_jeu` aux nouvelles dimensions de la grille, pour qu'elle mette à jour les configurations précédente, courante, et suivante.

Indication : à chaque itération, copier la configuration k dans la $k-1$, puis la configuration $k+1$ dans la k, et mettre à jour la configuration $k+1$ en appliquant les règles du jeu

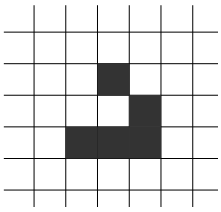


4.2 Des motifs particuliers

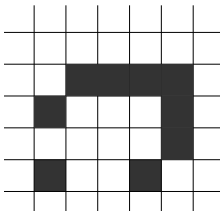
Certains motifs présentent des comportement particuliers lorsqu'ils évoluent : certains sont périodiques, se déplacent à l'identique, ou semblent tirer des projectiles, etc.

Q17 Écrire une fonction `init_jeu_motif(N, motif)` prenant en argument un entier `N` et un tableau à deux dimensions `motif` représentant un motif, qui retourne une grille de taille N du jeu avec le motif au centre. Le motif est supposé de dimension inférieure à N.

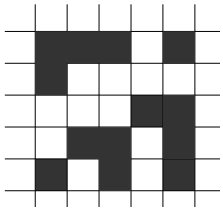
Utiliser cette fonction pour observer l'évolution des motifs ci-après.



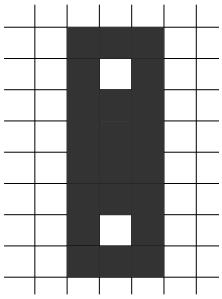
(a) Glider



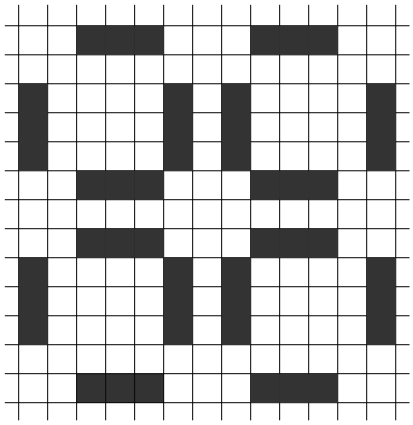
(b) Spaceship



(c) Callahan



(d) Pentadecathlon



(e) Pulsar