# TD 11 : le Jeu de la Vie
**Mars-Avril 2012**

Dans ce TD on parlera d'automates cellulaires : des modèles de systèmes dynamiques en apparence très simples, mais qui se révèlent très puissants dans des domaines parfois inattendus. Vous manipulerez des listes à deux dimensions, et vous n'utiliserez rien de plus méchant que des boucles for.

## Le jeu de la vie

Le jeu de la vie, imaginé par John Conway à la fin des années 40, est ce qu'on appelle un automate cellulaire. Il s'agit d'un système (qui n'est d'ailleurs pas particulièrement *ludique*) composé d'une grille infinie de cases (typiquement $Z^2$), qui évolue au fil du temps selon des règles très simples.

Chaque case est définie à un instant $t$ par son état : mort ou vivant, et évolue selon l'état de ses 8 voisines de la façon suivante :

1. *Naissance* : Si la case est morte à l'instant $t$ et qu'elle a exactement 3 voisines vivantes, alors elle devient vivante à l'instant $t+1$.
2. *Survie* : Une case vivante à $t$ le reste à $t+1$ si et seulement si elle a 2 ou 3 voisines vivantes.

Ce qui est amusant, c'est que de ces règles simplistes surgissent énormément de structures macroscopiques régulières, comme nous allons le voir.

## Rappels

Vous aurez dans ce TD à créer et manipuler des configurations sous forme de listes à deux dimensions. Une liste à deux dimensions est simplement une liste dont chaque élément est aussi une liste (comme `[[5,6],[7,8]]`).

On accède à ces listes avec la notation habituelle : sur la liste ci-dessus, `l[2][1]` renvoie 7.

Pour créer des listes à deux dimensions, on peut imbriquer un opérateur `seq` dans un autre : `[seq([seq(i*j,i=1..10)], j=1..10)]` renvoie la table de multiplication entre 1 et 10.

## Questions

### Quelques outils pour la suite

On ne va pas essayer de représenter une grille infinie en Maple, mais on va travailler sur un *tore*, c'est-à-dire un carré où l'on conviendra que les cases tout en haut sont voisines avec celles tout en bas, et pareil sur les cotés (comme le labyrinthe de Pacman). On représentera ces carrés par des listes de listes, ou chaque case vivante vaut 1, et chaque case morte 0.

**Question 1** Ecrire `vide:=proc(n)` et `randConfig:=proc(n)`, qui renvoient des configurations de taille $n \times n$ respectivement vide et tirée au hasard.

**Question 2** Ecrire `densite:=proc(config)`, qui calcule le nombre de cases vivantes dans une configuration donnée.

**Question 3** En déduire `estVide:=proc(config)`, qui détermine si tout le monde est mort dans la configuration ou non. Noter qu'une configuration vide reste vide ; il n'y a pas de naissances spontanées.

**Question 4** Écrire `compteVoisins:=proc(config,i,j)`, la procédure qui compte le nombre de voisins vivants de la case $(i, j)$.

**Évaluation paresseuse** Pour gérer les bords du carré, vous pouvez profiter de l'évaluation *paresseuse* des booléens par Maple : pour tester la valeur de la formule **F=A and B**, si A est faux, Maple sait que F sera nécessairement faux et n'essaiera pas d'évaluer B.

De fait, même si i et j sont hors du carré, des tests comme : `if (i<=n) and (i>=1) and (j<=n) and (j>=1) and (config[i][j]=1)` n'échoueront pas en Maple.

**Question 5 (optionnelle)** Ecrire `trace:=proc(config)`, qui affiche de façon pas trop moche la configuration donnée. Vous pouvez utilisez `plot` ou `print`. Soyez créatifs.

Une solution très simple mais peu lisible se résume en : `print(Matrix(config))`

### Le Jeu de la Vie

**Question 6** Ecrire `destinCase:=proc(config,i,j)`, qui calcule le destin une étape plus loin de la case $(i, j)$.

**Question 7** En déduire `evolution:=proc(config)`, qui renvoie une *nouvelle* configuration, l'évolution à une unité de temps de l'argument.

**Question 8** Ecrire `load:=proc(motif,n)`, qui crée une configuration vide de taille $n \times n$ au centre (approximativement) de laquelle est placée le motif :

1. Le *block* : `[ [1,1],[1,1] ]`
2. Le *blinker* : `[ [0,1,0],[0,1,0],[0,1,0] ]`
3. Le *glider* : `[ [0,1,0],[0,0,1],[1,1,1] ]`

**Question 9** Donner `life:=proc(config,z)`, qui calcule le futur d'une configuration après $z$ étapes.

*Optionnel* : si vous avez fait la question 5, faites afficher à vos programmes les configurations successives au cours du calcul (rappelez vous de `display` vu au TD 7, si vous utilisez `plot`).

**Question 10** Les trois motifs de la question 8 avaient chacun leur particularité. Que sont-elles ? *(NB : avec un peu de patience vous pouvez éventuellement faire cette question à la main)*

**Question 11** Modifier le `life` de la question 9 pour, après $z$ étapes, tracer le graphe d'évolution de la densité de cases vivantes.

## Pour aller plus loin

Le Jeu de la Vie est un sujet réellement très riche, et il y a énormément de choses à découvrir après cette maigre introduction : quel genre de choses peut-on calculer avec le jeu de la vie ? sait-on calculer les générations successives plus rapidement qu'avec l'algorithme naïf vu ici ? etc... Si ce genre de choses vous intéresse, je vous suggère :

- *Golly* : un petit logiciel libre qui fait tourner le jeu de la vie ultra rapidement, avec l'algorithme Hashlife. Il dispose en plus d'une grande bibliothèque de motifs préchargés souvent très exotiques (comme un jeu de la vie dans le jeu de la vie).
- *The irRegular Game of Life* : un petit jeu en flash très amusant qui propose une quarantaine de niveaux-puzzles autour des règles du jeu de la vie.
- *Wireworld* : un autre automate cellulaire, sur lequel des gens ont créé un "Wireworld computer" assez impressionnant qui calcule et affiche les nombres premiers successifs.