1 Préparation : autour des grilles

1.1 Grilles de jeu

Un grand nombre de jeux se jouent sur une grille à 2 dimensions.

On représente une grille par une liste de listes de caractères.

On rappelle les bonnes pratiques en vigueur dans l'UE : on ne modifie pas les valeurs des paramètres. On recopie le paramètre valeur par valeur dans une variable locale puis on modifie cette variable locale.

1. Écrire une fonction `init_grille` qui prend en paramètre la dimension d'une grille carrée et un caractère, et renvoie une grille contenant ce caractère dans chacune de ses cases.

```
$$$ g = init_grille(3, '*')
$$$ g[0]
['*', '*', '*']
$$$ g[1]
['*', '*', '*']
$$$ g[2]
['*', '*', '*']
```

2. Écrire une fonction `copie_grille` qui prend en paramètre une grille et renvoie sa copie case par case. **Attention** Vous devez créer une liste qui contiendra les lignes (ou les colonnes) de la nouvelle grille puis **créer et remplir élément par élément** chacune des listes représentant les colonnes (ou les lignes).

```
$$$ g = init_grille(3, '*')
$$$ g2 = copie_grille(g)
$$$ len(g2)
3
$$$ g[1]
['*', '*', '*']
$$$ g[2]
['*', '*', '*']
```

3. Écrire une fonction `positionne` qui prend en paramètre une grille carrée, un couple d'indices dans cette grille et un caractère, et renvoie une nouvelle grille dans laquelle le caractère a été positionné à ces indices. **Attention** vous devez commencer par faire une copie.

```
$$$ g = init_grille(3, '*')
$$$ g2 = positionne(g, (1, 2), 'o')
$$$ g2[1][2]
'o'
$$$ g2[2][1]
'*'
```

4. Écrire une fonction `positionne_diagonales` qui prend en paramètre une grille carrée et un caractère, et positionne ce caractère dans les cases sur les 2 diagonales. **Attention** aux bonnes pratiques.

```
$$$ g = init_grille(3, '*')
$$$ g = positionne_diagonales(g, 'o')
$$$ g[1][1]
'o'
$$$ g[2][2]
'o'
$$$ g[0][0]
'o'
$$$ g[2][0]
'o'
$$$ g[0][2]
'o'
$$$ g[1][2]
```

```
'*'
$$$ g[2][1]
'*'
```

5. Écrire une fonction `nb_cases_avec_car` qui prend en paramètre une grille carrée et un caractère, et renvoie le nombre de cases contenant ce caractère.

```
$$$ g = init_grille(3, '*')
$$$ g = positionne_diagonales(g, 'o')
$$$ nb_cases_avec_car(g, 'o')
5
```

2 Jeu de la vie

D'après un ancien projet utilisé en L1 SESI.

2.1 Présentation

Peut-on reproduire la «vie» (au sens de structures qui évoluent, se déplacent, … et créent elles-mêmes d'autres structures) à l'aide de règles très simples appliquées à des «cellules» ? C'est le défi qu'a lancé J.H. Conway en proposant un automate cellulaire simple intitulé le «jeu de la vie» en 1970. Les automates cellulaires sont définis sur une grille de cellules : les cellules se trouvent dans un état donné et leur état est modifié dans le temps en fonction de leur voisinage. Ces automates cellulaires offrent des modèles simples permettant de simuler des systèmes complexes (en biologie, en physique, en cryptographie, pour la modélisation du trafic autoroutier…). Dans le jeu de la vie, chaque cellule d'une grille à deux dimensions possède un des deux états : *vivante* (=1) ou *morte* (=0). L'état d'une cellule évolue au cours du temps en fonction de trois règles (voir figure 1) impliquant les états des huit cellules qui lui sont immédiatement adjacentes :

- R1 : une cellule morte possédant exactement trois cellules voisines vivantes, naît ;
- R2 : une cellule vivante possédant deux ou trois cellules voisines vivantes reste vivante ;
- R3 : une cellule vivante ne possédant pas deux ou trois cellules voisines vivantes meurt (par isolement ou par surpeuplement).

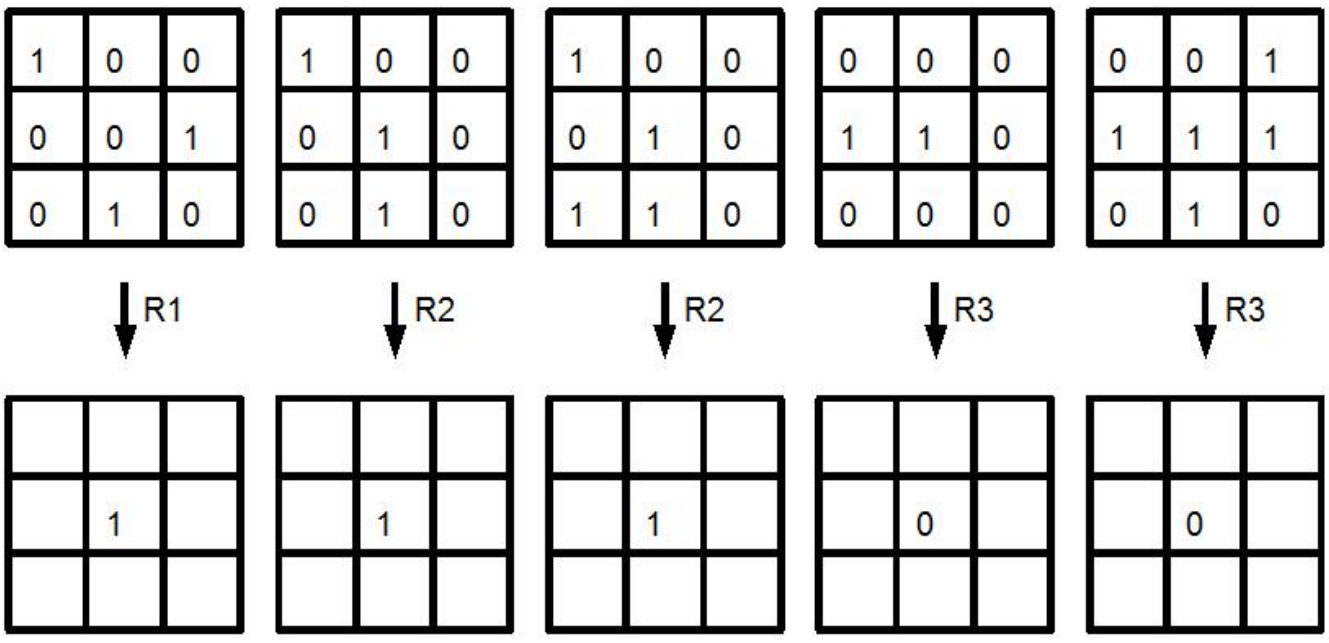

Figure 1: Illustration des trois règles d'évolution du jeu de la vie appliquées à la cellule centrale dans différents voisinages

Ainsi, l'évolution globale du système va se faire de manière automatique entre deux itérations (passage du temps

t au temps t+1, ou du pas de simulation t au pas de simulation t+1), et va dépendre de la grille initiale choisie ainsi que des règles locales. Comme les règles sont déterministes (les mêmes états en entrée donneront toujours les mêmes états en sortie), il suffit au «joueur» de choisir la configuration de départ puis de laisser l'ordinateur faire évoluer la grille sur autant de pas de simulation que souhaités.

On peut alors observer des phénomènes d'extinction, des structures stables (qui n'évoluent plus dans le temps), des structures périodiques et mêmes des structures qui se déplacent (nommées *vaisseaux*). En 1970, J.H. Conway avait offert 50$ à qui trouverait une structure qui puisse en créer d'autres à l'infini. C'est un groupe d'étudiants du MIT qui a trouvé la solution peu après… (voir exemples à suivre). Il a même été montré par la suite que l'on pouvait concevoir un ordinateur (au sens d'une machine de Turing) à partir du jeu de la vie !

Pour simuler un espace infini, nous considérerons que les deux dimensions de la grille sont périodiques : à savoir, la colonne à «gauche» (respectivement, à «droite») de la première (resp. dernière) colonne est la dernière (resp. première) colonne ; et de même pour la première et la dernière ligne.

2.2 Travail à réaliser

Il est impératif de réfléchir au découpage du code pour que les fonctionnalités du jeu soient testables et testées.

On utilisera une grille carrée, et comme support de la grille une liste de listes dont les cases contiennent :

- soit l'entier 1 (la case contient une cellule),
- soit l'entier 0 (la case ne contient pas de cellule).

La fonctionnalité essentielle consiste, étant donnée une grille peuplée de cellules, à calculer une nouvelle grille après un pas de simulation, en respectant les règles énoncées ci-dessus.

On pourra utiliser comme configuration initiale :

- soit une configuration aléatoire obtenue à partir d'un taux d'occupation de la grille ;
- soit une configuration fournie dans le fichier `configurations_init_jeu_vie.py`.

Ce fichier contient des variables globales :

- `CONFIGS` contient une liste de configurations : l'utilisateur pourra saisir un indice de cette liste pour choisir une configuration donnée ;
- les autres variables représentent une configuration sous la forme d'un couple dont :
  - le premier élément est la taille de la grille ;
  - le 2ème élément est une liste de couples, chaque couple correspondant à la position (ligne, colonne) d'une cellule dans la grille (entiers compris entre 0 inclus et la taille de la grille exclue).

Le programme principal pourra donner le choix entre le type de configuration initiale (fournie/aléatoire), entre une simulation pas à pas ou par pas de taille fixée, et affichera la grille atteinte en fin de simulation

En lançant des simulations pour les configurations fournies, on pourra regarder :

- quelles sont les structures qui amènent à une extinction ?
- quelles sont celles qui sont stables ? périodiques ?
- quelles sont celles qui n'amènent à aucun comportement régulier ?
- quelles sont enfin celles qui correspondent à des vaisseaux ?

Pour celles et ceux qui souhaitent aller plus loin, il est possible de s'intéresser à l'évolution du taux d'occupation de la grille. Le programme devra alors pouvoir afficher, à la fin de la simulation, une courbe présentant l'évolution du taux d'occupation en fonction des pas de temps. On pourra s'appuyer pour cela sur la bibliothèque MatPlotLib, voir

- *http://matplotlib.org/*
- et notamment *http://matplotlib.org/examples/pylab_examples/simple_plot.html*