Compte rendu – Devoir 3

Objectif

Nous cherchons à implémenter le jeu de vie vu lors du TD5 de SR01 en utilisant Tkinter pour réaliser l'interface graphique du jeu.

Le jeu de la vie évolue sur un damier infini. Chaque case est occupée par une cellule qui peut être vivante ou morte. Nous modéliserons les cellules vivantes par une case rouge et les cellules mortes resterons « vides » soient blanches.

A chaque génération, chaque cellule peut naître, mourir, ou rester dans son état.

Chaque cellule a exactement 8 voisins.

Les règles qui permettent de passer d'une génération à l'autre sont les suivantes :

- Une cellule vivante ayant exactement 2 ou 3 voisins vivants survit la génération suivante ;
- Une cellule vivante ayant au moins 4 cellules voisines vivantes meurt d'étouffement à la génération suivante ;
- Une cellule vivante ayant au plus une cellule voisine vivante meurt d'isolement à la génération suivante ;
- Une case vide, soit une cellule morte, ayant exactement 3 voisins vivants, naîtra à la génération suivante.

Afin d'implémenter ce jeu, on considère le damier infini comme une matrice "torique". Le damier sera représenté par une matrice dont les bords droit et gauche sont reliés entre eux, ainsi que les bords supérieur et inférieur.

Fonctions d'implémentation du jeu

initMatrice

Cette fonction permet de générer aléatoirement une matrice dont les caractéristiques sont fixées dans les arguments (tailles et pourcentage de vie). Pour cela on commence par créer un tableau à deux dimensions de la taille voulu. Puis à l'aide du module python `random`, on peut tirer aléatoirement la valeur de chaque case selon la distribution voulue.

nombreVoisins

Les cases de la matrice ne comportant que des « 0 » et des « 1 », le nombre de voisins en vie d'une case correspond à la somme des valeurs de chacune des cases voisines. C'est dans cette fonction que l'idée d'une matrice torique est importante : il faut qu'une case sur la première ligne soit considérée comme voisine d'une case sur la dernière ligne. On utilise pour cela la fonction modulo (voir l'implémentation python).

newGeneration

Cette fonction permet de créer une nouvelle matrice, correspondant au passage à la génération suivante depuis la matrice actuelle passée en argument.

Pour cela, on boucle sur chaque case de l'ancienne matrice, et en fonction du nombre de voisins et des règles définies, on rentre la nouvelle valeur (« 0 » ou « 1 ») dans la nouvelle matrice dans la case correspondante.

Fonctions pour l'interface graphique

drawGrid

Cette fonction permet de générer un quadrillage comportant le nombre de cases correspondant à la taille de la matrice choisie. Pour cela, on calcule la taille à allouer à chaque case, puis on dessine ligne par ligne le quadrillage grâce aux méthodes de la classe `Canvas` de Tkinter. On utilise une boucle for pour tracer le bon nombre de lignes en les décalant à chaque itération de la taille d'une case.

fillGrid

Cette fonction prend en paramètre la matrice à afficher et le `Canvas` sur lequel on va afficher les éléments graphiques. On commence donc d'abord par effacer tous les rectangles rouges du `Canvas` s'il y en a, puis on parcourt à l'aide d'une boucle for chaque case de la matrice. Si la valeur contenue est un « 1 », correspondant à une cellule en vie, on dessine un carré rouge aux bonnes coordonnées.

initialise

Cette fonction nous permet d'initialiser la grille en récupérant les informations sur les `scale` de l'interface. L'initialisation crée une matrice à l'aide de la fonction `initMatrice` vue précédemment en passant en paramètres les éléments récupérés des `scale`, on dessine ensuite la grille et on la remplie en suivant la matrice créée. Pour terminer, on active les boutons pour lancer et arrêter le jeu.

start

Cette fonction est lancée lors du clic gauche sur le bouton de l'interface **Lancer**.

On met la variable globale running à TRUE et on lance la fonction `run`.

run

Si la variable globale running est à TRUE alors on crée une nouvelle matrice correspondant à la génération suivante à l'aide de la fonction `newGeneration`, puis on remplit la grille avec cette nouvelle matrice. On initialise une variable speed en récupérant la vitesse voulue dans le `scale` de l'interface. On décide que plus la valeur du `scale` est grande plus la génération suivante sera affichée rapidement. Ainsi, on utilise l'inverse de la vitesse récupérée par le `scale` multipliée par 1000. Après le temps défini par speed en ms, on relance la fonction `run`.

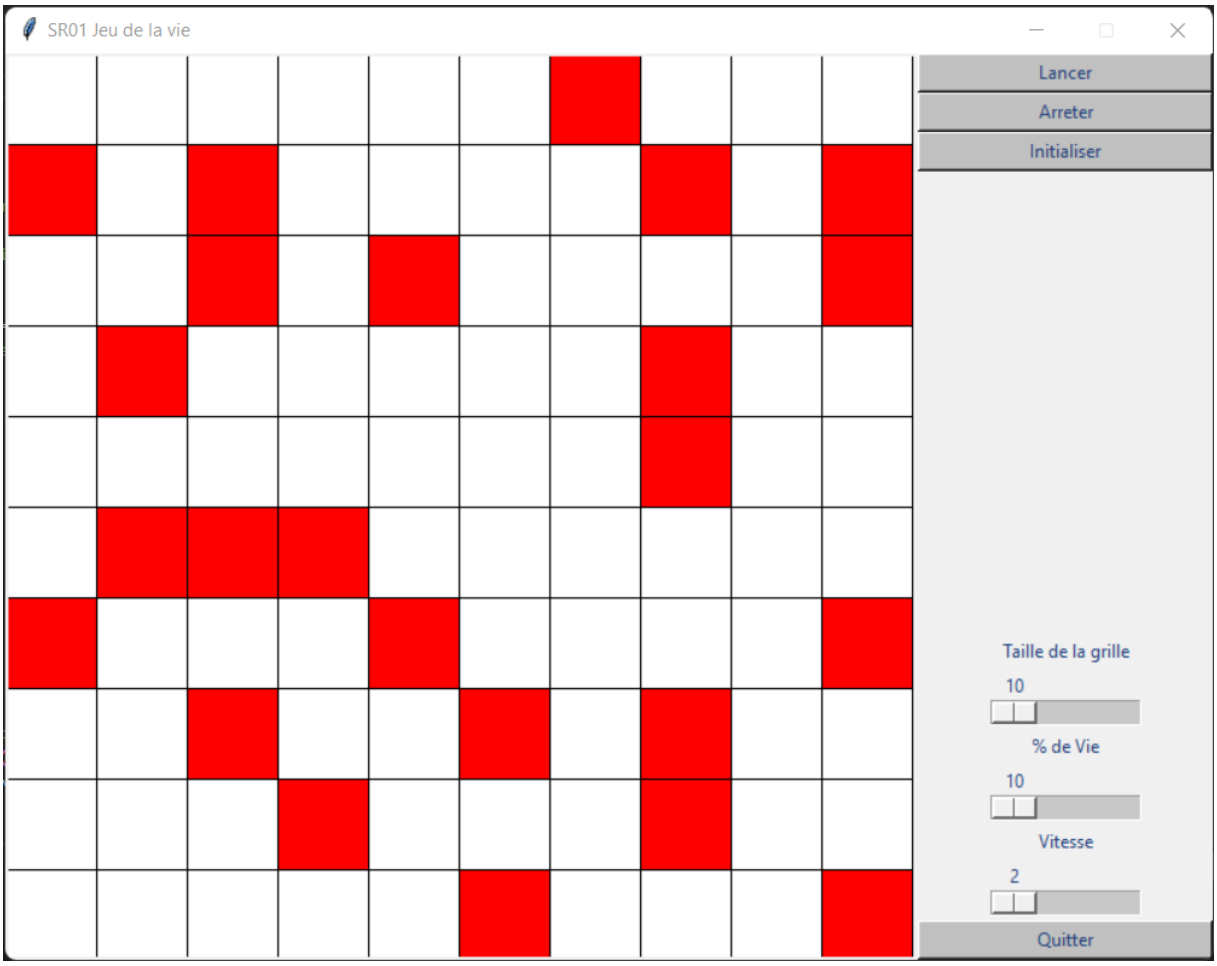
onerun

Cette fonction est lancée par un clic droit sur le bouton **Lancer**, elle permet d'afficher uniquement la génération suivante. Elle est identique à la fonction `run` à l'exception que la fonction n'est pas relancée après un certain temps.

stop

Cette fonction est lancée par un clic gauche sur le bouton **Arreter** de l'interface, elle met la variable globale running à FALSE pour que la fonction `run` (qui nécessite running à TRUE) ne soit pas relancée.

Interface graphique



On a cherché à se rapprocher le plus possible de l'interface graphique du sujet. En effet, nous avons créé deux frames, `gridFrame` à gauche pour la grille de jeu et `buttonFrame` à droite pour les boutons.

Dans `buttonFrame`, on pense à empêcher le clic sur **Lancer** et **Arreter** au lancement de la fenêtre grâce à state=DISABLED, ils seront réactivés dès que le jeu sera initialisé pour la première fois.

Nous avons ajouté sur le bouton **Lancer** un clic droit qui permet de lancer une unique génération grâce à la fonction `onerun`.

Lorsque le jeu est lancé grâce au bouton **Lancer**, la modification des valeurs des `scales` 1 et 2 (taille et pourcentage de vie) grâce à l'interface graphique n'influe pas sur la grille, si le jeu est lancé, on ne peut donc pas modifier la taille de la grille et le pourcentage de vie, si on le souhaite il faut initialiser un nouveau jeu. Sachant que si on initialise un nouveau jeu, pendant qu'un autre est en train de tourner, l'ancien jeu est arrêté et le nouveau jeu est affiché sans être lancé.

Cependant, la modification des valeurs du `scale` 3, soit la vitesse, pendant qu'un jeu tourne, influe immédiatement sur ce jeu. Ainsi, à tout moment d'un jeu, on peut changer quand on le souhaite la vitesse d'apparition d'une nouvelle génération.

Ouverture sur d'autres structures

Le jeu de la vie, de par sa grande simplicité originelle (seulement deux règles simples) mais à la fois sa capacité à créer des structures très complexes a attiré de nombreux chercheurs de domaines variés. Ces recherches ont abouti à la découverte de nombreuses structures intéressantes (configurations fixes, périodiques, en mouvement, à croissance infinie). Nous avons donc implémenté 4 de ces structures qui nous ont semblées amusantes et simples à réaliser : le « canon à planeurs », la « galaxie de Kok », le « pulsar » et le « pentadécathlon ».

Dans l'idée d'un easter egg, ces structures sont initialisables avec un clic droit sur le bouton **Initialiser**.

Ce clic droit affiche un menu flottant avec toutes les fonctions pour initialiser la grille avec ces structures intéressantes.

Les fonctions canonplanneur, galaxiekok, pulsar et penta suivent le même schéma et sont commentées dans le code. On initialise une matrice avec les cellules vivantes et mortes voulues pour dessiner la génération 0. On dessine et on remplit la grille avec cette matrice avant d'activer les boutons pour lancer et arrêter le jeu.