RAPPORT DEVOIR 3

OUEDRAOGO Taoufiq, BRENA–LABEL Yannis

SR01
Automne 2021

Introduction:

Le but de ce devoir était de réaliser le **<jeu de la vie>** sous format python**.** Pour ce faire, on s'est servi de la bibliothèque **Tkinter** de python pour mettre en place l'interface graphique.

Bibliothèques:

Afin d'implémenter ce jeu on a fait appel à plusieurs bibliothèques.
D'abord **Tkinter,** pour réaliser l'interface graphique, positionner les boutons et les scales mais aussi pour représenter chaque cellule par une case dans l'interface.
Ensuite **Numpy**, pour créer la matrice carrée (avec la fonction **zeros**) qui sera utilisée pour représentation dans le GUI. Puisque chaque élément de la matrice représente une case dans l'interface.
Puis, **Random,** qui va nous permettre de définir aléatoirement la valeur de chaque case. En utilisant le module **randrange(0,2)**, qui fournit aléatoirement un entier égal à 0 ou 1. Dans ce devoir, la valeur 1 servira à représenter une cellule vivante tandis que la valeur 0 représentera une cellule morte.
Enfin, la bibliothèque **Time,** qui nous permettra de mettre en arrêt le fonctionnement pendant un certain temps. On l'utilisera pour mettre en arrêt le fonctionnement du jeu grâce au bouton **Arreter** avec la fonction **sleep()**. Nous avons remarqué que notre jeu prenait un peu de temps à se relancer après l'utilisation du bouton **Arreter**.

Fonctions:

-initialiser() : pour initialiser la matrice aléatoirement.

-nb_voisin_vivant(mat, i, j) : qui renvoie le nombre de voisin vivant de la cellule dont les indices de ligne et colonne sont successivement i et j. Cette fonction traite tous les cas, que la cellule soit aux extrémités ou au cœur de la matrice.

-pourcentage_vie(mat) : retourne le pourcentage de cellules vivantes dans la matrice. Ce pourcentage sera affiché dans un des scale.

-survie(mat, l, c) : précise si la cellule de ligne l et de colonne c sera vivante ou non dans la génération suivante, en fonction des conditions de survie qui définissent le jeu.
Si la cellule se trouve dans des conditions de survie alors elle prend la valeur 1 sinon 0. Et donc définit aussi l'état de la cellule dans la génération suivante.

-next_gen(mat, mat_suiv) : renvoie la génération suivante issue de mat. Elle sera stockée dans la matrice mat_suiv.
Dans cette fonction, on va appliquer la fonction **survie()** pour chaque case de la matrice mat.

-afficher_matrice(mat) : permet d'afficher la matrice mat sous format matriciel.

-f(mat, mat_suiv) : Cette fonction va permettre l'implémentation successive du jeu et des différentes générations. Pour chaque case on va créer un rectangle qui sera colorié **bleu** si la case est vivante sinon sera colorié avec la couleur du Canvas. Pour cela on a utilisé la méthode **create.rectangle().**
Ensuite on a fait appel à la méthode **after()** de Tkinter qui permet d'effectuer une fonction incessamment avec un temps de latence.

C’est ce temps de latence qui nous permettra de contrôler la vitesse(on augmente le temps pour diminuer la vitesse et inversement). Il nous était dur de modifier cette vitesse à partir du scale; ainsi nous avons créé des évènements pour le faire (on appuie sur la touche **<a>** pour accélérer et **<b>** pour décélérer). Mais la valeur sera modifiée dans le scale.

```
appuyer la touche <a> pour accélérer
appuyer la touche <d> pour deccélérer
```

Dans cette fonction, on va également définir une nouvelle valeur par défaut pour le scale lié au % de vie. On a donc utilisé la méthode **scale.set()** pour le faire et ainsi le scale sera modifié sans action de notre part. Comme pour la vitesse.

-init() : pour initialiser le damier. Ainsi on redéfinit les valeurs des cases.

-lancer() : pour relancer le jeu. Pour ce faire on a défini une variable **go** qui vaut 0 pour confirmer l’arrêt et 1 pour confirmer le relancement avec la fonction **f** en appuyant sur le bouton **Lancer**.

-arreter() : pour stopper le jeu, en appuyant sur le bouton **Arreter**. On a utilisé la fonction **sleep()** pour endormir le processus qui fonctionne pendant une certaine période. Et le jeu sera relancé soit en appuyant sur le bouton **Lancer** soit si le délai est écoulé. On a eu des difficultés à trouver une méthode autre que celle-là.

-accelerer(event) : pour augmenter la vitesse, on est passé par un événement, ainsi il faudra appuyer la touche **<a>.** Pour accélérer, on a juste réduit le temps d’attente entre 2 générations successives (le **delay** dans la méthode **after()** de la fonction **f**).

-decelerer(event) : pour réduire la vitesse, on est passé par un événement, ainsi il faudra appuyer la touche **<d>.** Pour décélérer, on a juste augmenté le temps d'attente entre 2 générations successives (le **delay** dans la méthode **after()** de la fonction **f**).

Affichage graphique:

On a d'abord créé une **fenêtre**, puis un **Canvas** où sera affiché les différentes cellules.
Puis, des **boutons** pour l'exécution de certaines tâches.
Des **scales,** pour afficher l'état instantané du damier.

CONCLUSION :

Le jeu fonctionne correctement, cependant nous avons rencontré certaines difficultés lors de l'implémentation du jeu notamment le bouton **Arreter** mais des alternatives ont pu être trouvées.
Également, on ne savait pas si les scales servaient à modifier le damier où si plutôt étaient présents pour afficher l'état du damier; cependant nous étions d'abord partis sur la base que les scales permettraient à l'utilisateur de modifier le damier et le fonctionnement du jeu, mais plusieurs problèmes de fonctionnement sont apparus par la suite. C'est pour cela que finalement nous avons gardé les scales uniquement pour afficher à chaque instant l'état du damier, cela nous a paru plus judicieux, parce que modifier par exemple le % de vie pour ensuite modifier le damier est une tâche très complexe pour notre niveau actuel. Et pour modifier la vitesse, nous avons créé des évènements; ce qui semble plus simple.