Lycée Beaussier — La Seyne
NSI Terminale — Année .....

TP en deux parties

D'après Stephan Vanzuijlen

Introduction

Le jeu de la vie est un automate cellulaire imaginé par John Horton Conway en 1970 qui est probablement le plus connu de tous les automates cellulaires.
Le jeu de la vie est un jeu de simulation au sens mathématique plutôt que ludique.
Bien que n'étant pas décrit par la théorie des jeux, certains le décrivent comme un « jeu à zéro joueur ».

Le jeu se déroule sur une grille à deux dimensions, théoriquement infinie (mais de longueur et de largeur finies et plus ou moins grandes dans la pratique), dont les cases qu'on appelle des « cellules », par analogie avec les cellules vivantes peuvent prendre deux états distincts : « vivante » ou « morte ».

Une cellule possède huit voisins, qui sont les cellules adjacentes horizontalement, verticalement et diagonalement.
Celles des bords n'en possèdent que 5 et celles des coins que 3 sur une grille finie..

À chaque étape, l'évolution d'une cellule est entièrement déterminée par l'état de ses huit voisines de la façon suivante :

- une cellule morte possédant exactement trois voisines vivantes devient vivante (elle naît) ;
- une cellule vivante possédant deux ou trois voisines vivantes le reste, sinon elle meurt.

Le programme proposé est structuré autour de deux classes : Cellule et Grille

La classe Cellule

On définit une cellule par ses états(de vie) (actuel, futur) et ses voisines.

```
class Cellule:
    def_init_(self):
        self._actuel = False
        self._futur = False
        self._voisins = None
```

? QUESTION 1:

Quand on crée une cellule c=Cellule(), quels sont ses états?

Les méthodes :

Voici une première méthode, qui renvoie l'état actuel de la cellule

```
def est_vivant(self):
    return self._actuel
```

Voici deux méthodes, qui stocke et renvoie les voisines d'une cellule

```
def set_voisins(self,v):
    self._voisins = v
def get_voisins(self):
    return self._voisins
```

? QUESTION 2:

Compléter la méthode ci-contre pour qu'elle fasse naître la cellule à la prochaine génération (c-a-d : modifier son état futur...)

```
def naitre(self):
    ..............
```

? QUESTION 3:

Compléter la méthode ci-contre pour qu'elle fasse mourir la cellule à la prochaine génération.

```
def mourir(self):
    ..............
```

? QUESTION 4:

Compléter la méthode ci-contre pour qu'elle fasse évoluer l'état la cellule à la prochaine génération.

```
def basculer(self):
    ..................
```

Voici une méthode qui permet de déterminer l'état futur d'une cellule :

```
def calcule_etat_futur(self):
    nb_vivants = sum([c.est_vivant() for c in self._voisins])
    if  nb_vivants != 2 and nb_vivants !=  3:
        self.mourir()
    elif nb_vivants == 3:
        self.naitre()
    else:
        self._futur = self._actuel
```

? QUESTION 5:

Quel est le type de la variable nb_vivants et que représente-t-elle?

Voici une méthode qui permet d'afficher un caractère en fonction de l'état de la cellule:

```
    def_str_(self):
        if  self.est_vivant():
            res =chr(11048)
        else:
            res = " "
        return res
```

La classe Grille

On définit une grille par un tableau (largeur × hauteur) qui contient des cellules.

```
class Grille:
    def_init_(self,largeur,hauteur):
        self.largeur = largeur
        self.hauteur = hauteur
        self.matrix = [[Cellule() for i  in range(self.hauteur)] for j  in
                        range(self.largeur)]
```

l'attribut matrix est donc une liste de listes

? QUESTION 6:

En imaginant matrix comme un tableau (lignes×colonnes).

- l'attribut largeur correspond aux nombres de
- l'attribut hauteur correspond aux nombres de

Les méthodes :

Les cellules du tableau sont atteignables par leur coordonnées **(i,j)**

Voici une première méthode, qui renvoie si une cellule est dans la grille

```
    def dans_grille(self,i,j):
        return 0 <= i  < self.largeur and 0 <= j  < self.hauteur
```

Voici une seconde méthode

```
    def getXY(self,i,j):
        if self.dans_grille(i,j):
            return self.matrix[i][j]
        else:
            return  None
```

? QUESTION 7:

Que renvoie-t-elle?

Voici les méthodes qui permettent de déterminer les voisines d'une cellule dans le tableau

```
@staticmethod
def est_voisin(i,j,x,y):
    return max(abs(x-i),abs(y-j)) == 1

def get_voisins2(self,x,y):
    v=[]
    for i in range(x-1,x+2):
        for j in range(y-1,y+2):
            if self.dans_grille(i,j) and Grille.est_voisin(x,y,i,j):
                v.append(self.getXY(i,j))
    return v
```

La méthode est_voisin permet de savoir si la cellule de coordonnées **(i,j)** est voisine de la cellule de coordonnées (x,y)
C'est une méthode statique que l'on peut appliquer à tout objet grille..(ce n'est pas au programme de NSI)

? QUESTION 8:

Que renvoie la méthode get_voisins2 et que contient-elle?

La méthode affecte_voisins à pour objectif:

- pour chaque cellule du tableau
  - d'affecter à l'attribut _voisins des cellules la liste de leurs voisines

```
def affecte_voisins(self):
    for i in range(self.largeur):
        for jin range(self.hauteur):
            v = ..............
            ........................
```

? QUESTION 9:

Compléter le code ci-dessus
La première ligne à compléter récupère la liste des voisines de la cellule en cours.

La seconde ligne à compléter affecte à l'attribut _voisins de la cellule en cours la liste de ses voisines

Voici les méthodes qui permettent d'afficher la grille, de remplir aléatoirement la grille de cellules vivantes et de lancer le jeu de la vie.

```
    def_str_(self):
        res=""
        for i  in range(self.largeur):
            for j  in range(self.hauteur):
                res += str(self.getXY(i,j))
            res  +=  "\n"
        return res

    def remplir_alea(self,taux):
        for i  in range(self.largeur):
            for j  inrange(self.hauteur):
                if random.random() <= (taux/100):
                    self.getXY(i,j).naitre()
                    self.getXY(i,j).basculer()

    def jeu(self):
        for i  in range(self.largeur):
            for jin range(self.hauteur):
                c = self.getXY(i,j)
                c.calcule_etat_futur()
```

Et pour finir, voici la méthode qui permet d'actualiser les états des cellules pour la génération suivante:

```
    def actualise(self):
        for .....:
            for .....:
                ........................
```

? QUESTION 10:

Compléter le code pour qu'il fasse basculer l'état de chaque cellules de la grille.

Faites fonctionner le programme

Voici le lien qui vous donne le programme que vous devez compléter avec vos réponses. le fichier .py (pour spyder, eduPython) ou directement sur repl.it

- lien vers le fichier .py
- lien direct vers repl.it

Vous prendrez soin de spécifier les classes.