ISEN ALL IS DIGITAL! BREST | yncréa

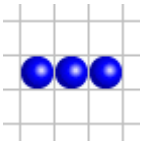
TP N°2.1 - Python
Le jeu de la vie

Thibault Napoléon
thibault.napoleon@isen-ouest.yncrea.fr

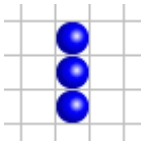
1 Le jeu de la Vie

Le but de l'application est de développer un automate cellulaire permettant de simuler l'organisation de la vie grâce à un ensemble de cellules. L'automate est basé sur une grille à deux dimensions de taille fixe (17x17 cellules dans notre cas) où l'évolution est pilotée par un opérateur. A chaque étape, l'évolution d'une cellule est entièrement déterminée par l'état de ses huit voisines selon des règles bien établies. L'automate cellulaire que vous devez réaliser est régit par les deux règles suivantes :

- 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.



(a)



(b)

Ainsi, la configuration 1a donne au tour suivant la configuration 1b qui redonne ensuite la première.

2 Exemple de déroulement

Entête et questions d'initialisation

Au lancement de l'automate, un entête s'affiche et deux questions sont posées à l'opérateur :

- Quelle est votre nom ?
- Quel est le nom du fichier à lire ?

```
-----------------
| Jeu de la vie |
-----------------
Quelle est votre nom ? Tibo
Quel est le nom du fichier à lire ? Data/Cross.txt
```

Entête de l'automate avec les deux questions posées à l'opérateur (en rouge : les réponses de l'opérateur)

2.1 Affichage initial de l'automate

L'automate affiche alors son état actuel (initialisation) avant de poser une question à l'utilisateur :

- De combien d'itérations voulez-vous avancer (-1 pour finir) ?

```
Isen - Itération numéro : 0!


- - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - -
- - - - @ @ @ - - - - - - - - - - -
- - - - @ - - @ - - - - - - - - - -
- - @ @ @ - - @ @ @ - - - - - - - -
- - @ - - - - - - @ - - - - - - - -
- - @ - - - - - - @ - - - - - - - -
- - @ @ @ - - @ @ @ - - - - - - - -
- - - - @ - - @ - - - - - - - - - -
- - - - @ @ @ @ - - - - - - - - - -
- - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - -
De combien d'itérations voulez-vous avancer (-1 pour finir) ?! 3
```

rawcode*Affichage de l'automate initial avec la question sur le nombre d'itérations (en rouge : la réponse de l'opérateur)*

Affichage de l'évolution de l'automate

L'automate affiche ensuite son état pour chaque itération demandée par l'opérateur et pose une question à celui-ci :

- De combien d'itérations voulez-vous avancer (-1 pour finir) ?

L'évolution de l'automate (1 itération) correspond à l'application des deux règles citées précédemment sur chaque cellule de celui-ci.

```
Isen - Itération numéro : 1

- - - - - - - - - - - - - - - - - -!
- - - - - @ @ - - - - - - - - - - -!
- - - - @ @ @ @ - - - - - - - - - -!
- - - - - - - - - - - - - - - - - -!
- - @ - @ - - @ - @ - - - - - - - -!
- @ @ - - - - - - @ @ - - - - - - -!
- @ @ - - - - - - @ @ - - - - - - -!
- - @ - @ - - @ - @ - - - - - - - -!
- - - - - - - - - - - - - - - - - -!
- - - - @ @ @ @ - - - - - - - - - -!
- - - - - @ @ - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - -!
```

```
- - - - - - - - - - - - - - - - - - - -!

Isen - Itération numéro : 2

- - - - - - - - - - - - - - - - - - - -!
- - - - @ - - @ - - - - - - - - - - -!
- - - - @ - - @ - - - - - - - - - - -!
- - - @ @ - - @ @ - - - - - - - - - -!
- @ @ @ - - - - @ @ @ - - - - - - - -!
- - - - - - - - - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - - -!
- @ @ @ - - - - @ @ @ - - - - - - - -!
- - - @ @ - - @ @ - - - - - - - - - -!
- - - - @ - - @ - - - - - - - - - - -!
- - - - @ - - @ - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - - -!

Isen - Itération numéro : 3

- - - - - - - - - - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - - - -!
- - - - @ @ @ @ - - - - - - - - - - -!
- - - - @ - - @ - - - - - - - - - - -!
- - @ @ @ - - @ @ @ - - - - - - - - -!
- - @ - - - - - - @ - - - - - - - - -!
- - @ - - - - - - @ - - - - - - - - -!
- - @ @ @ - - @ @ @ - - - - - - - - -!
- - - - @ - - @ - - - - - - - - - - -!
- - - - @ @ @ @ - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - - - -!
- - - - - - - - - - - - - - - - - - - -!

De combien d'itérations voulez-vous avancer (-1 pour finir) ? -1
```

Évolution de l'automate avec la question sur le nombre d'itérations (en rouge : la réponse de l'opérateur)

3 Programme à écrire

Pour réaliser ce programme, deux stratégies s'offrent à vous :

- Développer votre code fait maison.

- Utiliser les conseils proposés dans la suite de ce document.

Quel que soit votre choix, veillez à respecter le déroulement présenté ci-dessus et pensez à faire des classes, des fonctions et à commenter votre code.

La classe *Cellule*

Le but de cette classe est de définir une cellule de l'automate. Par la suite, les classes *CelluleVivante* et *CelluleMorte* hériterons de *Cellule*.

Travail à réaliser

Créez le fichier *Cellule.py* dans lequel vous écrirez le code correspondant à la classe *Cellule*. Cette classe comporte un unique attribut :

- *___estVivante* : de type booléen, il vaut vrai si la cellule est vivante et faux sinon.

Cette classe contient un constructeur, une méthode et un accesseur ("*getter*") dont voici la description :

- Constructeur *___init___* : Il permet d'initialiser l'attribut *___estVivante* avec le paramètre *estVivante*.
- Méthode afficher : Elle permet d'afficher une cellule. Pour cela elle affiche "@ " si celle-ci est vivante et "- " sinon. Attention à ne pas oublier l'espace après le '@' et le '-'.
- Accesseur *estVivante* : C'est un *getter* pour *___estVivante*.

La classe *CelluleVivante*

Le but de cette classe est de définir une cellule vivante de l'automate. Cette classe hérite de la classe *Cellule* définie précédemment.

Travail à réaliser

Créez le fichier *CelluleVivante.py* dans lequel vous écrirez le code correspondant à la classe *CelluleVivante*. Cette classe comporte juste un constructeur dont voici la description :

- *___init___* : Il appel simplement le constructeur de la classe mère (*Cellule*) pour créer une cellule vivante.

La classe *CelluleMorte*

Le but de cette classe est de définir une cellule morte de l'automate. Cette classe hérite de la classe *Cellule* définie précédemment.

Travail à réaliser

Créez le fichier *CelluleMorte.py* dans lequel vous écrirez le code correspondant à la classe *CelluleMorte*. Cette classe comporte juste un constructeur dont voici la description :

- *___init___* : Il appel simplement le constructeur de la classe mère (*Cellule*) pour créer une cellule morte.

La classe *Operateur*

Le but de cette classe est de gérer un opérateur. En l'occurrence elle permet de définir son nom et de lui demander de rentrer un nombre ou une chaine.

Travail à réaliser

Créez le fichier *Operateur.py* dans lequel vous écrirez le code correspondant à la classe *Operateur*. Cette classe comporte un unique attribut :

- ___*nom* : de type *string* il contient le nom de l'opérateur.

Cette classe contient un constructeur, trois méthodes et un accesseur ("*getter*") dont voici la description :

- Constructeur ___*init*___ : Il permet d'initialiser le nom de l'opérateur à une valeur par défaut que vous choisirez.
- Méthode *demanderNom* : Elle permet de demander un nom à l'utilisateur (comme spécifié dans la partie 2 : "*Exemple de déroulement*") et de le stocker dans l'attribut ___*nom* de la classe.
- Méthode *demanderNombre* : Elle permet de demander un nombre à l'opérateur supérieur ou égal à -1 l'utilisateur (comme spécifié dans la partie 2 : "*Exemple de déroulement*") et de le renvoyer.
- Méthode *demanderChaine* : Elle permet de demander une le nom de fichier à l'opérateur (comme spécifié dans la partie 2 : "*Exemple de déroulement*") et de la renvoyer.
- Accesseur *getNom* : C'est un getter pour ___*nom*.

La classe *Automate*

Le but de cette classe est de gérer l'automate cellulaire. Elle doit permettre d'initialiser l'automate, de le faire évoluer par itérations successives et de l'afficher. Il est basé sur une grille de cellules vivantes ou mortes.

Travail à réaliser

Créez le fichier *Automate.py* dans lequel vous écrirez le code correspondant à la classe *Automate*. Cette classe comporte trois attributs :

- ___*grille* : de type tableau de 17x17 Cellule permettant de représenter l'automate cellulaire.
- ___*operateur* : de type *Operateur* représentant l'opérateur de l'automate.
- ___*iterations* : de type *int* représentant le numéro de l'itération en cours.

Cette classe contient un constructeur, cinq méthodes privées eutft une méthode publique dont voici la description :

- Constructeur ___*init*___ : Il permet d'initialiser chaque cellule du tableau ___*grille* à 0, l'attribut ___*iterations* à 0 et de créer l'opérateur.
- Méthode privée ___*creerCellule* : Elle permet de remplir la case x, y (paramètres de la méthode) de la grille avec une cellule vivante ou morte (en fonction du paramètre *estVivante*).
- Méthode privée ___*initialiser* : Elle permet d'initialiser ___*grille* avec une grille les informations du fichier dont le nom est passé en paramètre dans *chemin*.
- Méthode privée ___*afficher* : Elle permet d'afficher la grille de l'automate cellulaire (comme spécifié dans la partie 2 : "*Exemple de déroulement*") en affichant : Le nom de l'opérateur, le numéro de l'itération et la grille. Vous utiliserez la méthode *afficher* de chaque cellule pour afficher la grille.
- Méthode privée ___*iterer**Cellule* : Elle permet de calculer la somme des voisines de la cellule de position x, y (paramètres de la méthode) en considérant qu'elles valent 1 si elles sont vivantes et 0 sinon. Elle renvoie :
  - Vrai si la somme vaut 3.
  - Faux si la somme est inférieur à 2 ou supérieur à 3.
  - La valeur de la cellule sinon.

  [label=•]

- Méthode privée *___iterer* : Elle permet de calculer une itération de l'automate cellulaire. Elle commence par créer un tableau temporaire de même taille que la grille. Elle remplit alors ce tableau avec *True* ou *False* en fonction du résultat de la méthode *iterercellule*. Ensuite la grille est mise à jour avec ce tableau temporaire. On incrémente alors le paramètre *___iterations* et on affiche la grille. Finalement, ces instructions sont répétées *nombreDIterations* fois (donné en paramètre de la méthode).
- Méthode publique lancer : Il s'agit de la fonction permettant de créer l'automate et de lancer la boucle d'itérations. Son déroulement doit être le suivant :
  - Afficher l'entête du programme (comme spécifié dans la partie 2 : "*Exemple de déroulement*").
  - Demander le nom de l'opérateur.
  - Initialiser la grille de l'automate cellulaire en appelant la méthode du même nom.
  - Afficher la grille de l'automate cellulaire en appelant la méthode du même nom.
  - Lancer la boucle de vie demandant à l'opérateur un nombre d'itérations (comme spécifié dans la partie 2 : "*Exemple de déroulement*") et exécutant celles-ci grâce à la méthode *iterer*. Cette boucle s'arrête lorsque l'opérateur saisi une valeur différente de -1.

Programme principal

Il s'agit du point d'entrée du programme permettant de lancer l'automate.

Travail à réaliser
Écrivez un programme principal dans le fichier *Run.py*. Pour cela, il vous suffit de créer un automate et de le lancer en appelant la méthode du même nom.