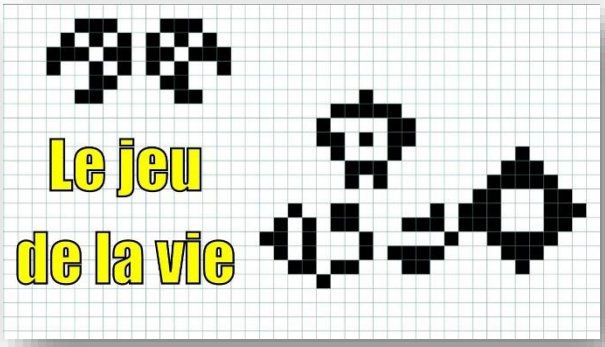
Le jeu de la vie

Miniprojet 2

I. Le principe

Le jeu de la vie a été inventé par le mathématicien britannique John H. Conway (1937-2020). C'est un exemple de ce qu'on appelle un **automate cellulaire**.



Une cellule peut être dans deux états : **vivante** ou **morte**. La dynamique du jeu s'exprime par les règles de transition suivantes :

- une cellule vivante reste vivante si elle est entourée de 2 ou 3 voisines vivantes et meurt sinon ;
- une cellule morte devient vivante si elle possède exactement 3 voisines vivantes.

La notion de « voisinage » dans le jeu de la vie est celle des 8 cases qui peuvent entourer une case donnée. (on parle de voisinage de Moore.

L'objectif de ce projet est de :

- ✓ réaliser une modélisation objet du problème.
- ✓ Implémenter les classes en programmation objet
- ✓ intégrer votre travail à une IHM

II. Modélisation objet

Q1. Quelles classes peut-on dégager de ce problème au premier abord ?

Q2. Quelles sont quelques-unes des méthodes qu'on pourrait leur donner ?

Q3. Dans quelle classe pouvons-nous représenter simplement la notion de voisinage d'une cellule ? Et le calculer.

Q4. Une cellule est au bord si x=0, x=L-1, y=0 ou y=H-1. Combien de voisins possède une cellule qui n'est pas au bord ? Combien de voisins possède une cellule qui est au bord ?

Q5. Que pourrions-nous aussi comme voisin de droite de la case en haut à droite de la grille ? Et comme voisin du haut ?

III. Implémentation des cellules

Classe Cellule

Q6. Implémenter tout d'abord une classe `Cellule` avec comme attributs :

- ✓ Un booléen `actuel` initialisé à `False` ;
- ✓ Un booléen `futur` initialisé à `False` ;
- ✓ Un booléen `voisins` initialisés à `None` ;

Ces attributs seront considérés comme « privés ». La valeur `False` signifie que la cellule est morte et `True` qu'elle est vivante.

Q7. Ajouter les méthodes suivantes :

- ✓ `est_vivant()` qui renvoie l'état actuel (vrai ou faux) ;
- ✓ `set_voisins()` qui permet d'affecter comme voisins, la liste passée en paramètre ;
- ✓ `get_voisins()` qui renvoie la liste des voisins de la cellule ;
- ✓ `naitre()` qui met l'état futur de la cellule à `True` ;
- ✓ `mourir()` qui permet l'opération inverse ;
- ✓ `basculer()` qui fait passer à l'état futur la cellule dans l'état actuel ;

Q8. Ajouter à la classe `Cellule` une méthode `__str__()` qui affiche une croix (un X) si la cellule est vivante et un tiret (un -) sinon. Expliquer brièvement l'utilité d'une telle méthode en Python.

Q9. Ajouter une méthode `calcule_etat_futur()` dans la classe `Cellule` qui permet d'implémenter les règles d'évolution du jeu de la vie en préparant l'état futur à sa nouvelle valeur.

Classe Grille

Q10. Créer la classe `Grille` et y placer les attributs suivants considérés comme « public » :

- ✓ `largeur` ;
- ✓ `hauteur` ;
- ✓ `matrice` : un tableau de cellules à 2 dimensions (implémenté en python par une liste de liste).

Q10. Ajouter les méthodes suivantes :

- ✓ `dans grille()` qui indique si un point de coordonnées i et j est bien dans la grille ;
- ✓ `setXY()` qui permet d'affecter une nouvelle valeur à la case (i, j) de la grille ;
- ✓ `getXY()` qui permet de récupérer la cellule située dans la case (i, j) de la grille ;
- ✓ `get_largeur()` qui permet de récupérer la largeur de la grille ;
- ✓ `get_hauteur()` qui permet de récupérer la hauteur de la grille ;
- ✓ `est_voisin()` une méthode statique qui vérifie si les cases (i, j) et (x, y) sont voisines dans la grille ;

```
@staticmethod
def est_voisin(i, j, x, y):
    # à compléter
```

Une méthode statique peut être appelée à l'extérieur et à l'intérieur de la classe sans dépendre de l'objet instancié. Ici, on l'appellera avec `Grille.est_voisin(i, j, x, y)`

Q10. Ajouter une méthode `get8voisin()` qui renvoie la liste des voisins d'une cellule.

Q11. Fournir une méthode `__str__()` qui permet d'afficher la grille sur un terminal.

Q12. On veut remplir aléatoirement la `Grille` avec un certain taux de `Cellule` vivantes. Fournie à cet effet une méthode `remplir_alea()` avec le taux (en %) en paramètre.

Jeu

Q13. Concevoir une méthode `Jeu()` permettant de passer en revue toutes les `Cellule` de la `Grille`, de calculer leur état futur, puis une méthode `actualise()` qui bascule toutes les cellules de la `Grille` dans leur état futur.

Q14. Programme principal :

```
if __name__=='__main__':
    # à compléter
```

Terminer l'implémentation du jeu de la vie avec un affichage en console en utilisant les méthodes précédentes.

On pensera à instancier, remplir aléatoirement, calculer les voisinages, puis jouer en marquant une pause grâce à la méthode `sleep` du module `time`.

On donne la méthode suivante qui permet d'effacer l'écran dans un terminal ANSI :

```
def effacer_ecran():
    print("\u001B[H\u001B[J")
```

(Q14. Optionnel : proposer une interface graphique utilisant une bibliothèque graphique de type Tkinter***)***

Travail à rendre :

- ✓ Un document répondant aux questions sur la modélisation d'objet
- ✓ Le code Python