

BACCALAURÉAT

SESSION 2026

Épreuve de l'enseignement de spécialité

NUMÉRIQUE et SCIENCES INFORMATIQUES

Partie pratique

Classe de Première de la voie générale

Sujet : Le Jeu de la Vie de Conway

DURÉE DE L'ÉPREUVE : 2 heures

Le sujet comporte 6 pages numérotées de 1 / 6 à 6 / 6
Dès que le sujet vous est remis, assurez-vous qu'il est complet.

Cette situation d'évaluation comporte ce document ainsi que des fichiers de codes présents sur l'ordinateur à la disposition du candidat. Le candidat doit restituer ce document avant de sortir de la salle d'examen. Le candidat doit agir en autonomie et faire preuve d'initiative tout au long de l'épreuve.

En cas de difficulté, le candidat peut solliciter l'examineur afin de lui permettre de continuer sa tâche. Des moments privilégiés pour solliciter l'examineur sont indiqués dans le document sous la forme d'appels professeur.

L'examineur peut intervenir à tout moment, s'il le juge utile.

Le **Jeu de la Vie** est un *automate cellulaire* inventé par le mathématicien John Horton Conway en 1970. Il se déroule sur une grille rectangulaire dont chaque case, appelée **cellule**, est soit **vivante** (valeur 1) soit **morte** (valeur 0).

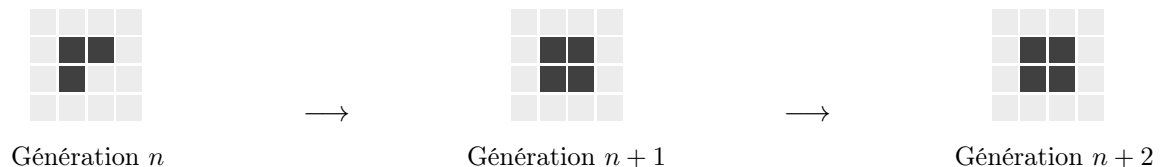
Chaque cellule possède au plus 8 voisins : les cases adjacentes horizontalement, verticalement et en diagonale. Les cellules situées sur les bords de la grille possèdent moins de 8 voisins.

À chaque étape (appelée **génération**), toutes les cellules évoluent *simultanément* selon les règles suivantes :

- **Naissance** : une cellule morte ayant exactement 3 voisins vivantes devient vivante ;
- **Survie** : une cellule vivante ayant 2 ou 3 voisins vivantes reste vivante ;
- **Mort** : dans tous les autres cas, la cellule meurt ou reste morte.

En Python, une grille est représentée par une liste de listes d'entiers 0 et 1. Considérons l'exemple suivant, où trois cellules vivantes forment un L :

```
grille = [[0, 0, 0, 0],  
          [0, 1, 1, 0],  
          [0, 1, 0, 0],  
          [0, 0, 0, 0]]
```



À la génération suivante, la cellule (2,2) naît car elle possède exactement 3 voisins vivantes. Les trois cellules déjà vivantes survivent car elles ont chacune 2 voisins vivantes. On obtient un **bloc** : une structure stable qui ne change plus.

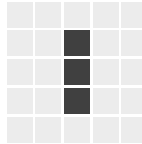
Le fichier `logique.py` contient les fonctions liées aux règles du jeu. Le fichier `jeu_de_la_vie.py` gère la visualisation avec Pygame et importe le module `logique`.

Partie 1 : Logique du jeu (logique.py)

On travaille dans le fichier `logique.py`. Les résultats s'affichent dans le terminal.

Question 1

On considère la grille suivante appelé **blinker** (*Clignotant*).



```
grille = [[0, 0, 0, 0, 0],
          [0, 0, 1, 0, 0],
          [0, 0, 1, 0, 0],
          [0, 0, 1, 0, 0],
          [0, 0, 0, 0, 0]]
```

Sans exécuter de programme :

- Déterminer** à la main le nombre de voisines vivantes des cellules (2, 2), (1, 2) et (2, 1). Pour chacune, indiquer la règle qui s'applique (naissance, survie ou mort).
- En raisonnant de la même manière pour toutes les cellules de la grille, **dessiner** la grille complète à la génération suivante.
- Que peut-on conjecturer sur le comportement de ce motif?



Appeler le professeur pour lui présenter votre réponse ou en cas de difficulté.

Question 2

Écrire en Python le code de la fonction `est_dans_grille` qui prend en paramètres une grille et deux entiers `i` et `j`, et qui renvoie `True` si les coordonnées (i, j) correspondent à une cellule existante dans la grille, `False` sinon.

Question 3

Écrire en Python le code de la fonction `compter_voisins` qui prend en paramètres une grille et deux entiers `i` et `j` représentant les coordonnées d'une cellule, et qui renvoie le nombre de voisines vivantes de cette cellule (un entier entre 0 et 8).

Puis, **compléter** la grille `BLINKER` dans le fichier `logique.py` pour y placer un clignotant horizontal (trois cellules vivantes côte à côte en ligne 2, colonnes 1, 2 et 3).

Exécuter la fonction de test `test_compter_voisins()` pour vérifier la réponse.



Appeler le professeur pour lui présenter votre réponse ou en cas de difficulté.

Question 4

Écrire en Python le code de la fonction `cellule_suivante` qui prend en paramètres une grille et deux entiers `i` et `j`, et qui renvoie l'état de la cellule (i, j) à la génération suivante : 1 si elle sera vivante, 0 sinon. On appliquera les trois règles données en introduction en utilisant la fonction `compter_voisins`.

Question 5

Écrire en Python le code de la fonction `generation_suivante` qui prend en paramètre une grille et qui renvoie une **nouvelle** grille correspondant à la génération suivante, en appliquant les trois règles données en introduction. La grille d'origine ne doit pas être modifiée. Exécuter la fonction de test `test_generation_suivante()` pour vérifier la réponse.



Appeler le professeur pour lui présenter votre réponse ou en cas de difficulté.

Question 6

Écrire en Python le code de la fonction `afficher` qui prend en paramètre une grille et qui l'affiche dans le terminal en utilisant le caractère ■ (cellule vivante) et le caractère · (cellule morte), suivis d'un espace. Tester en appelant `afficher(BLINKER)`.

Question 7

Compléter le code à trous de la fonction `simuler` qui prend en paramètres une grille et un entier `n`, et qui simule `n` générations en affichant chaque étape dans le terminal. Exécuter `simuler(BLINKER, 20)` et constater l'évolution. On considère la figure suivante, déjà codée dans le fichier sous le nom de `GLIDER`



Un planeur (*glider*)

Exécuter `simuler(GLIDER, 20)` et constater l'évolution.



Appeler le professeur pour lui présenter votre réponse ou en cas de difficulté.

Partie 2 : Visualisation avec Pygame (jeu_de_la_vie.py)

On travaille dans le fichier `jeu_de_la_vie.py`. Ce fichier importe le module `logique` et contient un programme principal qui ouvre une fenêtre Pygame et affiche une grille. Certaines parties du code sont commentées : elles seront décommentées et complétées au fil des questions.

Question 8

Écrire en Python le code de la fonction `dessiner_grille` qui prend en paramètres l'écran Pygame et une grille, et qui dessine un rectangle coloré pour chaque cellule. On rappelle que dans Pygame, le coin supérieur gauche de la fenêtre correspond au point $(0,0)$. Pour une cellule de coordonnées (i,j) :

- l'abscisse en pixels vaut $j * \text{TAILLE_CELLULE}$;
- l'ordonnée en pixels vaut $i * \text{TAILLE_CELLULE}$.

Lancer le programme. On doit voir la grille du blinker s'afficher de manière statique dans la fenêtre.

Question 9

Dans le programme principal, **repérer** la ligne commentée correspondant à la question 7. La **décommenter** et la **compléter** pour que la grille soit mise à jour à chaque tour de boucle en appelant la fonction `generation_suivante`.

Lancer le programme et observer : le blinker doit maintenant s'animer.



Appeler le professeur pour lui présenter votre réponse ou en cas de difficulté.

Question 10

On souhaite pouvoir mettre la simulation en pause et la relancer avec la touche `ESPACE`.

Ajouter une variable `en_pause` initialisée à `True` au début du programme principal.

Décommenter et **compléter** le bloc correspondant à la question 8 dans la gestion des événements.

Modifier la ligne de la question 7 pour que la grille ne soit mise à jour que lorsque la simulation n'est pas en pause.

Lancer le programme : la simulation doit démarrer en pause et se lancer ou s'arrêter à chaque appui sur `ESPACE`.

Question 11

On souhaite pouvoir dessiner des cellules à la souris lorsque la simulation est en pause.

Décommenter et **compléter** le bloc correspondant à la question 9 : lors d'un clic gauche en pause, le programme doit calculer les coordonnées (i,j) de la cellule cliquée à partir de la position de la souris, puis inverser son état (vivante \rightarrow morte ou morte \rightarrow vivante).

Tester : dessiner un planeur en cliquant sur les cellules, puis lancer la simulation avec `ESPACE` et **observer** le comportement.



Appeler le professeur pour lui présenter votre réponse ou en cas de difficulté.

Question 12

Choisir au moins une des améliorations suivantes et l'implémenter :

- a) **Grille torique.** Modifier la fonction `compter_voisins` dans `logique.py` pour que la grille boucle sur elle-même : le bord droit est voisin du bord gauche, le bord haut est voisin du bord bas. On utilisera l'opérateur modulo (%).
- b) **Couleur selon l'âge.** Modifier la fonction `dessiner_grille` pour que la couleur d'une cellule vivante varie en fonction du nombre de générations consécutives pendant lesquelles elle est restée vivante.
- c) **Compteur de population.** Afficher dans la fenêtre Pygame le numéro de génération et le nombre total de cellules vivantes à l'aide de `pygame.font`.



Appeler le professeur pour lui présenter votre réponse ou en cas de difficulté.