Projet algorithmique

Ce projet **individuel** est décomposé est deux parties : la conception d'un jeu de puissance 4 dans le terminal et le calcul d'une image fractale de l'ensemble de Mandelbrot.

Consignes pour le rendu

Pour toutes vos fonctions, sauf indication contraire, vous devrez produire une **docstring** contenant au moins :

- la liste des arguments (avec leur type et s'il y a lieu leurs restrictions)
- le type de retour
- une **doctest** contenant des tests appropriés (certains tests sont déjà proposés pour vous faciliter la tâche).

Vous déposerez sur moodle dans l'espace dédié au projet :

- Un fichier `puissance4.py` avec le code complété.
- Un fichier `ensemble_mandelbrot.py` avec le code complété.

au plus tard pour le :

Dimanche 15 janvier 2023

Attention à :

- bien rendre 2 fichiers et non pas un dossier compressé,
- respecter le nom des fonctions demandées.
- faire tous vos tests, affichages et définitions de variables dans la conditionnelle :

```
if __name__ == "__main__":
    # tests, etc.
```

Il faut que l'import de vos fichiers ne provoque aucun affichage !

Il y aura des tests automatiques pour faciliter la correction : un non-respect des consignes entraînera une pénalité sur la note.

Avertissement sur le travail à plusieurs

Ce projet est à rendre individuellement. Vous pouvez vous entraidez sur ce projet, **mais tout le code que vous produisez doit être personnel !**

Il y aura des vérifications de similarité de code, et en cas de suspicion de fraude, je ferais un signalement à la commission disciplinaire de l'université [1].

1. voir ce lien pour davantage d'informations

1 Puissance 4

Le Puissance 4 est un jeu de société à 2 joueurs sans hasard, dans lequel les joueurs placent tour à tour des jetons dans une grille (originellement de 6 lignes et 7 colonnes).

On ne peut placer un jeton dans une colonne que si celle-ci n'est pas pleine, le jeton tombe alors à la position la plus basse possible.

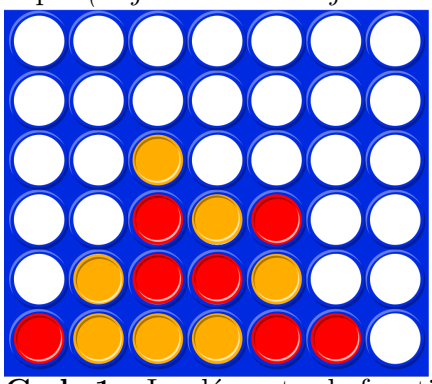
La partie se termine lorsque l'un des joueurs parvient à aligner 4 jetons (verticalement, horizontalement ou en diagonale) : dans ce cas, il remporte la partie ; ou que la grille est complétée sans qu'il y ait de vainqueur : il y a alors match nul. [2]

L'objectif de cette première partie du projet est d'implémenter ce jeu avec un affichage dans le terminal (on n'utilisera pas d'affichage graphique).

Conventions de représentation Pour implémenter le jeu, on utilisera les conventions suivantes :

- Les joueurs sont numérotés 1 et 2.
- La grille de jeu peut avoir des dimensions variables, pas forcément 6 lignes et 7 colonnes. *(Vos fonctions devront donc être génériques dans leur fonctionnement.)*
- Une grille de `n` lignes et `m` colonnes sera représentée par un tableau de taille `n` contenant des tableaux d'entiers de taille `m` : la première ligne de la grille correspond à la plus basse dans le jeu.
- Les valeurs de la grille peuvent être :
  — 1 si l'emplacement correspondant est occupé par un jeton du joueur 1,
  — 2 si l'emplacement correspondant est occupé par un jeton du joueur 2,
  — 0 si l'emplacement correspondant est vide.

Exemple *(le joueur 1 a les jetons rouges)* :



est représentée par :

```
grille = [[1,2,2,2,1,1,0],
          [0,2,1,1,2,0,0],
          [0,0,1,2,1,0,0],
          [0,0,2,0,0,0,0],
          [0,0,0,0,0,0,0],
          [0,0,0,0,0,0,0]]
```

Code 1. Implémentez la fonction `generer_grille_vide(nb_col,nb_lig)` prenant en paramètre deux entiers positifs `nb_col` et `nb_lig` représentant respectivement le nombre de colonnes et le nombre de lignes de la grille de jeu, et retournant une représentation d'une grille vide.

Code 2. Implémentez la fonction `affiche_grille(grille)` prenant en paramètre une grille et **affichant** la grille à l'écran selon les conventions suivantes :

- les jetons du joueur 1 sont représentés par des `X` (x en majuscule),
- les jetons du joueur 2 sont représentés par des `O` (o en majuscule),
- les emplacements vides sont représentés par des espaces,
- les séparateurs de colonnes sont des barres verticales `|` (alt gr + 6),

2. Pour une description plus complète du jeu, voir https://fr.wikipedia.org/wiki/Puissance_4

- les séparateurs de lignes sont des tirets (touche 6),
- des `+` sont placés aux croisements des lignes verticales et horizontales de la grille,
- les numéros des colonnes sont inscrits en dernière ligne.

Exemple :

L'appel de
`affiche_grille(grille)`
avec

```
grille = [[1,1,2,1,2,1,1],
          [2,1,2,2,1,2,2],
          [0,2,1,1,0,1,2],
          [0,1,2,2,0,0,0],
          [0,1,2,1,0,0,0],
          [0,0,0,0,0,0,0]]
```

Provoque l'affichage à l'écran de :

```
+-+-+-+-+-+-+-+
|␣|␣|␣|␣|␣|␣|␣|
+-+-+-+-+-+-+-+
|␣|X|O|X|␣|␣|␣|
+-+-+-+-+-+-+-+
|␣|X|O|O|␣|␣|␣|
+-+-+-+-+-+-+-+
|␣|O|X|X|␣|X|O|
+-+-+-+-+-+-+-+
|O|X|O|O|X|O|O|
+-+-+-+-+-+-+-+
|X|X|O|X|O|X|X|
+-+-+-+-+-+-+-+
␣0␣1␣2␣3␣4␣5␣6
```

Code 3. Implémentez la fonction `peut_jouer(grille,colonne)` prenant en paramètre une grille `grille` et un entier `colonne` représentant un numéro de colonne, et retournant `True` si un joueur peut jouer dans la colonne `colonne` (`False` sinon).

Votre fonction devra également vérifier si le numéro de colonne donné est possible.

Code 4. Implémentez la fonction `joue(grille,colonne,joueur)` prenant en paramètre une grille `grille` et deux entiers `colonne` et `joueur` représentant respectivement un numéro de colonne et un numéro de joueur, et modifiant `grille` en ajoutant un jeton du joueur `joueur` dans la colonne `colonne`.

On considérera que le coup à jouer est possible.

(Pour cette fonction, produisez uniquement une docstring avec les commentaires, il n'est pas nécessaire de produire de doctest.)

Code 5. Implémentez 4 fonctions :

- `a_gagne_vert(grille,joueur)` : prenant en paramètre une grille `grille` et un numéro de joueur `joueur` et retournant `True` si le joueur `joueur` à gagné par une combinaison de 4 jetons alignés verticalement dans la grille, `False` sinon.
- `a_gagne_hor(grille,joueur)` : prenant en paramètre une grille `grille` et un numéro de joueur `joueur` et retournant `True` si le joueur `joueur` à gagné par une combinaison de 4 jetons alignés horizontalement dans la grille, `False` sinon.
- `a_gagne_diag1(grille,joueur)` : prenant en paramètre une grille `grille` et un numéro de joueur `joueur` et retournant `True` si le joueur `joueur` à gagné par une combinaison de 4 jetons alignés en diagonale montante dans la grille, `False` sinon.
- `a_gagne_diag2(grille,joueur)` : prenant en paramètre une grille `grille` et un numéro de joueur `joueur` et retournant `True` si le joueur `joueur` à gagné par une combinaison de 4 jetons alignés en diagonale descendante dans la grille, `False` sinon.

Puis implémentez la fonction générale `a_gagne(grille,joueur)` prenant en paramètre une grille `grille` et un numéro de joueur `joueur` et retournant `True` si le joueur `joueur` à gagné par une combinaison de 4 jetons alignés dans la grille, `False` sinon.

Code 6. Implémentez la fonction `grille_pleine(grille)` prenant en paramètre

une grille `grille` et retournant `True` si la grille `grille` est remplie.

Code 7. Implémentez la fonction `boucle_principale()` implémentant le jeu selon l'algorithme (haut niveau) suivant :

```
boucle_principale()
  Données : grille : entier[][], joueur_courant : entier
1 début
2 |  grille ← generer_grille_vide
3 |  joueur_courant ← 1
4 |  tant que grille n'est pas pleine faire
5 |  |  Afficher la grille
6 |  |  Demander le prochain coup au joueur courant
7 |  |  tant que le prochain coup n'est pas un coup valable faire
8 |  |  |  Afficher un message d'erreur
9 |  |  |  Afficher la grille
10 |  |  |  Demander le prochain coup au joueur courant
11 |  |  fin
12 |  |  Jouer le coup retenu
13 |  |  si Le joueur courant a gagné alors
14 |  |  |  Afficher un message de victoire
15 |  |  |  Sortir de la fonction
16 |  |  fin
17 |  |  Changer le joueur courant
18 |  fin
19 |  Afficher un message de match nul
20 fin
```

(Il n'est pas nécessaire de produire de docstring/doctest pour cette fonction.)

2 L'ensemble de Mandelbrot

Soient $a, b \in \mathbb{R}^2$, on considère les suites (x_n) et (y_n), de paramètres a et b, définies récursivement par :

$$\begin{cases} x_0 = 0 \\ x_{n+1} = x_n^2 - y_n^2 + a \quad \forall n \in \mathbb{N} \end{cases} \text{ et } \begin{cases} y_0 = 0 \\ y_{n+1} = 2x_n \times y_n + b \quad \forall n \in \mathbb{N} \end{cases}$$

L'ensemble de Mandelbrot est défini par :

$$\mathscr{E} = \{(a, b) \in \mathbb{R}^2 | (x_n^2 + y_n^2)_{n\in\mathbb{N}} \text{ est bornée}\}$$

On peut montrer que si pour un certain indice n, $x_n^2 + y_n^2 > 4$, alors la suite ne sera pas bornée.

Dans ce projet, on appelle *indice de divergence*, l'indice n à partir duquel la valeur de $x_n^2 + y_n^2$ est strictement supérieure à 4.

Ce nombre pouvant être infini (dans certains cas, la suite $(x_n^2 + y_n^2)_{n\in\mathbb{N}}$ est bornée), on se fixe un indice limite $n_{max} = 127$.

Code 8. Implémentez la fonction `prochains_termes(x,y,a,b)` prenant en paramètre deux nombres `x` et `y` représentant les termes x_n et y_n des suites définies ci-dessus et retournant les termes correspondant à x_{n+1} et y_{n+1}.

Code 9. Implémentez **de manière récursive** la fonction `indice_divergence` prenant en paramètres :

- `x,y` : les termes courants de la suite
- `a,b` : les paramètres pour le calcul de la suite
- `n` : l'indice du terme courant calculé.

et retournant :

- -1 si `n` > 127
- l'indice à partir duquel la valeur de $x_n^2 + y_n^2$ est supérieur à 4.

Pour afficher l'ensemble de Mandelbrot sur une image, on *discrétise* une partie du plan : chaque pixel de l'image représente un couple (a, b) qui deviennent les paramètres des suites (x_n) et (y_n).

On affiche ensuite :

- un pixel noir `(0,0,0)` si la suite ne diverge pas (pour $n \leqslant 127$)
- un pixel d'une certaine couleur (par exemple d'une des couleurs présentes dans `color_table`) en fonction de l'indice de divergence.

La discrétisation du plan est déjà gérée dans la fonction `create_image` : vous n'avez pas besoin de la recoder.

Code 10. Implémentez la fonction `get_color(a,b)` retournant la couleur du pixel représentant le couple `a,b` selon la méthode suivante : on pose n l'indice de divergence de la suite dont les paramètres sont `a` et `b`.

- si $n > 127$ (`indice_divergence` renvoie -1) : la couleur renvoyée est $(0, 0, 0)$.
- sinon, on retourne la couleur contenue à l'indice n *modulo* $taille(color\_table)$.

Sur les exemples fournis, vous devez obtenir les figures :