Projet ICC-P : Jeu 2048

1. Présentation du projet

Dans ce projet, vous allez implémenter une version simplifiée du fameux jeu 2048 pour smartphone et y jouer dans le terminal. Vous implémenterez différentes parties du jeu séparément avant de tout assembler dans un programme complet. Vous pourrez ensuite soumettre votre code à un correcteur automatique qui exécutera de nombreux tests sur votre code, en vous attribuant des points partiels pour chaque test réussi.

2. Description du jeu

2048 est un jeu de puzzle pour un seul joueur, joué sur une grille 4x4. Voici un site web où vous pouvez jouer au jeu: https://2048game.com/. Au début de la partie, deux tuiles apparaissent aléatoirement sur la grille, chacune affichant le nombre 2. Ensuite en fusionnant les tuiles de même valeur, l'utilisateur peut faire apparaitre des nouvelles tuiles avec des puissances de 2 (2, 4, 8, 16, ...).

Objectif

Le but du jeu est de combiner des tuiles portant le même nombre pour créer des tuiles de valeur plus élevée, idéalement jusqu'à atteindre la tuile 2048. Cependant, le joueur peut continuer à jouer au-delà si des coups restent possibles.

Fonctionnement du jeu

- Le joueur utilise les touches **W**, **A**, **S**, **D** sur le clavier pour déplacer toutes les tuiles vers le **haut**, la **gauche**, le **bas** ou la **droite**.
- Chaque pression de touche déclenche un coup qui met à jour toute la grille selon les règles de déplacement et de fusion du jeu. Ces règles déterminent comment les tuiles glissent, quand les fusions ont lieu, et comment l'état final de chaque ligne ou colonne est formé. Une description complète et précise de cette procédure est donnée dans la section décrivant la fonction `move`.

Dans le jeu standard, une nouvelle tuile ajoutée après un coup peut être une 2 ou une 4. Par simplicité, ce projet ajoute uniquement une tuile de valeur **2** après chaque coup valide.

Implémentation spécifique au projet

Dans ce projet, plusieurs valeurs sont fournies sous forme de **variables globales**. Une variable globale est une variable définie une fois au niveau supérieur du programme et qui peut être lue à l'intérieur de n'importe quelle fonction.

- Vous **pouvez lire** les variables globales à l'intérieur de vos fonctions en utilisant leur nom comme pour des variables "classiques".
- Vous **ne devez pas modifier** ces variables globales à l'intérieur de vos fonctions. Toutes les fonctions doivent se comporter correctement en utilisant les valeurs existantes.
- Vous **pouvez temporairement changer** les valeurs de ces variables lorsque vous testez votre code (par exemple en utilisant une valeur différente pour `MAX_TILE`), mais ces modifications ne doivent pas être faites à l'intérieur des fonctions que vous soumettez.

Les variables globales importantes sont :

- `BLANK = 0` (Représente une case vide dans la grille.)
- `GRID_SIZE = 4` (Le jeu utilise une grille fixe de taille `GRID_SIZE` x `GRID_SIZE`.)
- `VALID_KEYS = ['W', 'A', 'S', 'D', 'Q']` (Les seules entrées utilisateur autorisées.)
- `MAX_TILE = 64` (La valeur maximale de tuile autorisée dans le jeu.)

Bien que le jeu original 2048 continue jusqu'à l'apparition d'une tuile de valeur 2048, ce projet utilise la variable `MAX_TILE` pour définir quand le jeu doit s'arrêter. Vous pouvez supposer que toutes les valeurs de tuiles seront toujours des puissances de deux comprises entre 2 et `MAX_TILE`. `MAX_TILE` est initialement fixé à 64 dans ce projet, mais nous pourrons utiliser des valeurs plus élevées pour tester si votre code reste correct avec des valeurs plus grandes. La valeur maximale que nous pourrons utiliser pour tester votre soumission est 2048.

Victoire

La partie est gagnée lorsque le joueur obtient au moins une tuile de valeur `MAX_TILE`. Dans ce cas, le jeu se termine et le texte "You Win!" est affiché dans le terminal.

Défaite

La partie est perdue lorsqu'aucun coup n'est possible :

- La grille est pleine (aucune case vide), et
- Aucune paire de tuiles adjacentes n'a la même valeur (aucune fusion possible).

Dans ce cas, le jeu se termine et le texte "Game Over!" est affiché dans le terminal.

3. Conception du jeu

3.1 Représentation de la grille

La représentation interne de la grille 2048 utilisera une **liste 2D (liste de listes)** en Python. Cette structure modélise le plateau de taille `GRID_SIZE` x `GRID_SIZE` comme une matrice où chaque sous-liste représente une **ligne** du jeu.

- La grille est une liste contenant `GRID_SIZE` sous-listes (lignes).
- Chaque sous-liste contient `GRID_SIZE` valeurs entières (colonnes).
- Les cases vides sont représentées par la constante `BLANK`.
- Les cases non vides contiennent des puissances de deux entières (2, 4, 8, 16, etc.).
- La coordonnée $(0, 0)$ correspond à la case **en haut à gauche**.
- La coordonnée $(0, 3)$ correspond à la case **en haut à droite**.
- La coordonnée $(3, 0)$ correspond à la case **en bas à gauche**.
- La coordonnée $(3, 3)$ correspond à la case **en bas à droite**.

Exemple de grille interne avec `BLANK` = 0, `GRID_SIZE` = 4 et `MAX_TILE` = 64 :

```
grid = [
    [2, 0, 0, 4],
    [4, 4, 0, 0],
    [0, 32, 8, 0],
    [0, 0, 0, 16]
]

top_left = grid[0][0]    # 2
top_right = grid[0][3]   # 4
bottom_left = grid[3][0] # 0
bottom_right = grid[3][3] # 16
```

3.2 Modules / fonctions du jeu

Le projet est structuré autour de plusieurs fonctions et modules, chacun responsable d'une partie spécifique de la logique du jeu.

Vous devrez implémenter un ensemble de fonctions décrites dans ce document. Chaque fonction a un comportement bien défini, des paramètres précis, ainsi qu'un nombre de points qui lui est associé. Le correcteur automatique exécutera de nombreux tests sur chacune de ces fonctions, et vous obtiendrez des points partiels pour chaque test réussi. Vous ne disposez que d'un nombre limité de soumissions, utilisez-les donc judicieusement.

- `check_grid(grid)` — (donnée) — vérifie la validité de la grille
- `display_grid(grid)` — (donnée) — dessine la grille dans le terminal de manière formatée
- `init_grid()` — 5 pts — crée une grille vide
- `get_user_input()` — 15 pts — lis et valide le coup de l'utilisateur (W/A/S/D/Q)
- `move(grid, direction)` — 25 pts — deplace et fusionne les tuiles selon le coup
- `get_empty_positions(grid)` — 10 pts — renvoie la liste des positions vides

- `add_new_tile(grid)` — 5 pts — ajoute une nouvelle tuile de valeur 2 à une position vide
- `can_move(grid)` — 10 pts — vérifie si au moins un coup est possible
- `game_won(grid)` — 5 pts — vérifie si une tuile de valeur MAX_TILE est présente
- `main()` — 25 pts — fonction principale qui assemble toutes les parties du jeu

Le nombre total de points pour ce projet est de 100 points.

4. Fonctions fournies et fonctions à implémenter

Pour ce projet, un fichier modèle `game_2048.py` vous est donné. **Important :** ne modifiez pas les noms de fonctions fournis, car le correcteur automatique attend ces noms précis. Vous pouvez cependant définir des fonctions auxiliaires si cela vous aide dans votre implémentation. Utilisez les constantes définies dans le fichier modèle (par exemple `BLANK`, `GRID_SIZE`, `MAX_TILE`) et ne codez pas ces valeurs en dur : votre code doit continuer à fonctionner si `GRID_SIZE` est changé à 5 ou si `MAX_TILE` est changé à 2048.

Pour ce projet vous devez implémenter les fonctions du fichier `game_2048.py`. Des points seront attribués fonction par fonction, de manière partielle, en fonction des tests réussis pour chaque fonction.

4.1 Fonctions fournies

Les fonctions suivantes sont déjà implémentées pour vous :

- `check_grid(grid)` — (given) — vérifie la validité de la grille
- `display_grid(grid)` — (given) — affiche la grille de manière formatée

`check_grid(grid)` — 0 points

Cette fonction vérifie que la grille est valide pour que les autres fonctions puissent fonctionner correctement. Elle vérifie que la grille est de taille `GRID_SIZE` x `GRID_SIZE`, et que tous les nombres sont soit `BLANK`, soit des puissances de 2 comprises entre 2 et `MAX_TILE` (inclus).

Cette fonction est déjà implémentée pour vous. Vous pouvez l'utiliser sans vous soucier de son code interne. Son implémentation utilise les exceptions pour signaler les entrées invalides. Vous n'avez pas besoin de comprendre les détails des exceptions pour ce projet, car il s'agit d'un sujet plus avancé.

Important : vous devez appeler cette fonction au début de chaque fonction qui prend une grille en argument. C'est une pratique standard en programmation de vérifier les arguments d'entrée et de s'assurer qu'ils sont valides. Vous devez uniquement appeler la fonction `check_grid()` au début de votre fonction. Si l'entrée est invalide, votre programme plantera avec le message d'erreur approprié. Inspirez-vous de ce qui est fait dans `display_grid()`.

`display_grid(grid)` — 0 points

Cette fonction affiche une chaîne multilignes représentant la grille dans le terminal, sous forme de tableau avec des cases de largeur fixe. Elle appelle `check_grid(grid)` pour vérifier la validité de la grille avant de l'afficher.

Chaque case est affichée comme un cadre de **4 caractères de large** et de **3 lignes de haut**. Les tuiles sont centrées dans leurs cases afin de garder la grille visuellement alignée. Les cases vides (valeur `BLANK`) sont affichées comme des cases vides.

Exemple de résultat avec `BLANK` = 0, `GRID_SIZE` = 4 et `MAX_TILE` = 64 :

```
grid = [
    [2, 4, 0, 8],
    [0, 0, 64, 0],
    [2, 0, 16, 32],
    [0, 8, 0, 0]
]

display_grid(grid) ->

+----+----+----+----+
|    |    |    |    |
| 2  | 4  |    | 8  |
|    |    |    |    |
+----+----+----+----+
|    |    |    |    |
|    |    | 64 |    |
|    |    |    |    |
+----+----+----+----+
|    |    |    |    |
| 2  |    | 16 | 32 |
|    |    |    |    |
+----+----+----+----+
|    |    |    |    |
|    | 8  |    |    |
|    |    |    |    |
+----+----+----+----+
```

4.2 Fonctions à implémenter

Les fonctions suivantes doivent être implémentées par vous. Pour chaque fonction, l'énoncé précise les entrées, les sorties et les règles détaillées à respecter.

`init_grid()` — 5 points

Cette fonction crée une nouvelle grille vide.

Entrée : aucune.

Sortie : Une nouvelle grille de taille `GRID_SIZE` x `GRID_SIZE` (liste de listes) où chaque position est initialisée avec `BLANK` pour représenter une case vide.

```
[
    [0, 0, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0]
]
```

`get_user_input()` — 15 points

Cette fonction lit le coup du joueur au clavier et le valide. Elle accepte les lettres en majuscules ou en minuscules.

Le joueur doit entrer l'une des touches suivantes pour déplacer les tuiles :

- **W ou w** → déplacement vers le haut
- **A ou a** → déplacement vers la gauche
- **S ou s** → déplacement vers le bas
- **D ou d** → déplacement vers la droite

Cependant, le joueur peut aussi décider d'arrêter la partie en entrant :

- **Q ou q** → quitter la partie

La fonction doit :

1. Afficher ce message à l'utilisateur : `"Enter move (W/A/S/D/Q): "`.
2. Vérifier si l'entrée fait partie des touches valides `['W','A','S','D','Q']`, en utilisant la constante `VALID_KEYS`.
3. Si l'entrée est valide, renvoyer la direction choisie sous forme d'un unique caractère **majuscule**.
4. Si l'entrée est invalide, afficher le message d'erreur `"Invalid input, try again."`.

L'action de quitter la partie sera gérée par la fonction principale `main()`. Lorsque l'utilisateur entre `'Q'`, la fonction doit simplement renvoyer `'Q'`.

Attention : votre fonction doit être robuste, c'est-à-dire qu'elle doit aussi accepter les lettres valides en minuscules, et elle doit également accepter des entrées avec des espaces superflus. Cela signifie que `" a "` est une entrée valide. **Astuce :** les fonctions `.upper()` et `.strip()` peuvent être utiles.

Exemple d'interaction :

```
Enter move (W/A/S/D/Q): x
Invalid input, try again.
Enter move (W/A/S/D/Q):   a

-> get_user_input() returns 'A'
```

`move(grid, direction)` — 25 points

Cette fonction doit appliquer l'ensemble des règles de déplacement et de fusion de 2048 à la grille dans la direction donnée. Elle doit renvoyer une **nouvelle** grille `GRID_SIZE` x `GRID_SIZE`. La **grille d'origine ne doit pas être modifiée**.

Entrées :

- `grid` : une grille `GRID_SIZE` x `GRID_SIZE` sous forme de liste de listes d'entiers. Les cases vides contiennent `BLANK`.
- `direction` : un des caractères majuscules `'W'`, `'A'`, `'S'` ou `'D'` (présents dans `VALID_KEYS`) représentant respectivement **haut**, **gauche**, **bas** et **droite**.

Sortie : Une nouvelle grille `GRID_SIZE` x `GRID_SIZE` reflétant le résultat de l'application du mouvement. Si le mouvement ne change pas la grille (aucune tuile ne bouge ni ne fusionne), la nouvelle grille renvoyée doit être identique à la grille d'entrée.

Règles précises Les étapes ci-dessous décrivent le comportement requis des mécanismes du jeu, et non la stratégie d'implémentation.

Votre code peut être structuré comme vous le souhaitez, mais la grille résultante doit toujours respecter les règles données ici.

1. **La fonction ne doit pas modifier la grille d'origine.** Elle doit renvoyer une nouvelle grille représentant l'état mis à jour après le coup. Utilisez `copy.deepcopy()` pour copier la grille.
2. **Toutes les tuiles se déplacent aussi loin que possible dans la direction choisie.** Une tuile continue de glisser jusqu'à ce que l'une des conditions suivantes survienne :

- elle atteint le bord de la grille, ou
- elle devient adjacente à une autre tuile qui empêche tout mouvement supplémentaire.

Les tuiles ne traversent jamais d'autres tuiles, que ces tuiles aient déjà bougé ou qu'elles fusionnent plus tard dans ce même coup.

3. **Après que toutes les tuiles ont glissé, les tuiles adjacentes de même valeur peuvent fusionner.** La fusion doit respecter les règles suivantes :
   - Seules deux tuiles de même valeur, directement voisines dans la direction du mouvement, peuvent fusionner. La fusion commence du côté vers lequel on se déplace.
   - Une fusion produit une tuile unique dont la valeur est la somme des deux tuiles d'origine.
   - Chaque tuile ne peut participer qu'à une seule fusion par coup. Cette restriction s'applique aux tuiles d'origine et aux tuiles nouvellement créées qui ne peuvent pas fusionner à nouveau avant le coup suivant.
   - Après une fusion, la tuile résultante occupe la position la plus avancée dans la direction du mouvement, et elle bloque d'autres tuiles de même valeur pour éviter plusieurs fusions successives vers cette même position pendant le même coup.
4. **Après toutes les fusions possibles, les tuiles doivent être à nouveau compactées dans la direction du mouvement.** Il ne doit rester aucun « trou » entre les tuiles dans la direction choisie. Toutes les cases vides doivent se trouver du côté opposé à la direction de déplacement.
5. **Chaque ligne ou colonne concernée doit être dans un état entièrement résolu.** Chaque ligne (ou colonne) doit vérifier :
   (a) toutes les tuiles ont été déplacées aussi loin que possible dans la direction choisie ;
   (b) toutes les fusions valides ont été effectuées en respectant la règle « au plus une fusion par tuile et par coup » ;
   (c) aucune fusion supplémentaire n'est possible pour ce coup.

Exemples :

```
grid = [
    [2, 0, 2, 0],
    [4, 4, 4, 4],
    [0, 0, 0, 0],
    [0, 2, 16, 2]
]

move(grid, 'A') returns ->
[
    [4, 0, 0, 0],
    [8, 8, 0, 0],
    [0, 0, 0, 0],
    [2, 16, 2, 0]
]

...
```

```
grid = [
    [2, 0, 2, 4],
    [2, 0, 2, 4],
    [4, 4, 8, 4],
    [0, 32, 8, 0]
]

move(grid, 'W') returns ->
[
    [4, 4, 4, 8],
    [4, 32, 16, 4],
    [0, 0, 0, 0],
    [0, 0, 0, 0]
]
```

Remarque : vous êtes autorisé et encouragé à créer des fonctions auxiliaires si cela vous aide à structurer votre code.

`get_empty_positions(grid)` — 10 points

Cette fonction auxiliaire parcourt toute la grille et renvoie une liste des positions vides. L'ordre dans lequel les positions sont renvoyées n'a pas d'importance.

Chaque case vide est représentée par un **tuple** (x, y) où :

- x est l'**indice de ligne** (0 à 3, de haut en bas),
- y est l'**indice de colonne** (0 à 3, de gauche à droite).

Une case est considérée comme vide si sa valeur est `BLANK`.

Exemple :

```
grid = [
    [2, 0, 0, 4],
    [0, 2, 2, 16],
    [0, 4, 64, 0],
    [4, 0, 32, 0]
]

print(get_empty_positions(grid)) ->

[(0, 1), (0, 2), (1, 0), (2, 0), (2, 3), (3, 1), (3, 3)]
```

`add_new_tile(grid)` — 5 points

Cette fonction doit ajouter une nouvelle tuile dans une case vide de la grille. Vous pouvez utiliser la fonction intégrée de Python `random.choice(list)` pour choisir aléatoirement un élément dans une liste. La fonction `add_new_tile()` doit toujours ajouter une tuile de valeur `2` à la position vide choisie. La fonction ne doit pas modifier les cases non vides.

`can_move(grid)` — 10 points

Cette fonction vérifie si au moins un coup est possible sur la grille. Elle doit renvoyer `True` s'il existe au moins une case vide ou une paire de tuiles adjacentes ayant la même valeur, et `False` sinon.

Exemples :

```
grid = [
    [2, 4, 2, 8],
    [8, 16, 8, 2],
    [4, 2, 16, 4],
    [2, 8, 4, 2]
]

can_move(grid) # returns False


grid = [
    [2, 4, 2, 8],
    [8, 0, 8, 2],
    [4, 2, 16, 4],
    [2, 8, 4, 2]
]

can_move(grid) # returns True


grid = [
    [2, 4, 2, 8],
    [8, 16, 8, 2],
    [4, 2, 16, 4],
    [4, 8, 4, 2]
]

can_move(grid) # returns True
```

`game_won(grid)` — 5 points

Cette fonction vérifie s'il existe au moins une tuile de valeur `MAX_TILE` dans la grille. Si une telle tuile existe, la fonction renvoie `True`, indiquant que le joueur a gagné la partie. Sinon, elle renvoie `False`.

Exemples :

```
MAX_TILE = 64
grid = [
    [2, 4, 2, 8],
    [8, 0, 8, 2],
    [4, 2, 16, 4],
    [2, 8, 4, 2]
]

game_won(grid) # returns False

MAX_TILE = 64
grid = [
    [2, 4, 2, 8],
    [8, 16, 8, 2],
    [4, 2, 64, 4],
    [2, 8, 4, 2]
]

game_won(grid) # returns True

MAX_TILE = 128
grid = [
    [2, 4, 2, 8],
    [8, 16, 8, 2],
    [4, 2, 64, 4],
    [2, 8, 4, 2]
]

game_won(grid) # returns False
```

Assemblage : `main()` — 25 points

La fonction principale est responsable d'appeler toutes les autres fonctions pour permettre de jouer au jeu. Au début de son exécution, elle doit créer une grille vide et y ajouter deux tuiles `2` à des positions aléatoires. Ensuite, dans la boucle principale du jeu, tant que le joueur a encore un coup disponible, elle doit afficher la grille, demander un nouveau coup au joueur, appliquer ce coup et ajouter une nouvelle tuile à la grille. Si le joueur entre `'Q'`, vous devez afficher "Quitting the game." et terminer le programme.

Important : une nouvelle tuile ne doit être ajoutée que si le mouvement a effectivement modifié

la grille. Si le joueur essaie de se déplacer dans une direction où aucune tuile ne peut bouger (par exemple, toutes les tuiles sont déjà poussées à gauche et il appuie sur `'A'`), la grille reste inchangée et aucune nouvelle tuile ne doit être ajoutée.

Si le joueur atteint un état où au moins une tuile `MAX_TILE` est présente, vous devez arrêter le jeu et afficher "You Win!". Si le joueur atteint un état où il ne peut plus se déplacer, vous devez arrêter le jeu et afficher "Game Over!" dans le terminal. Avant d'afficher le message final de victoire ou de défaite, la grille doit être affichée une dernière fois.

Important : tous les affichages (par exemple les messages, la mise en forme de la grille, etc.) doivent correspondre à ce qui est attendu dans les tests. Utilisez les constantes textuelles définies dans le fichier fourni.

Une fois tout cela fait, vous pouvez jouer au jeu en lançant le programme, soit en utilisant le bouton d'exécution de VS Code, soit avec la commande suivante dans le dossier contenant le fichier du jeu :

```
python game_2048.py
```

5. Instructions et détails de soumission

Pour ce projet, vous devez implémenter les fonctions dans le fichier `game_2048.py` fourni.
Important : ne modifiez pas les noms de fonctions fournis, sinon le correcteur automatique ne les reconnaîtra pas et vous obtiendrez 0 point. Vous êtes toutefois autorisé à définir des fonctions auxiliaires si cela vous aide. Utilisez les constantes définies dans le fichier modèle lorsque c'est pertinent, car votre code sera testé avec différentes valeurs de ces constantes ! Ne codez aucune valeur en dur. Par exemple, votre code doit rester entièrement fonctionnel si `GRID_SIZE` est changé à 5 ou si `MAX_TILE` est changé à 2048.

Pour tester votre implémentation, nous fournissons un fichier `tests.py` qui teste les fonctions individuellement avec des exemples donnés dans l'énoncé. Tous les tests de ce fichier ne couvrent pas forcément tous les tests utilisés par le correcteur automatique, mais ils sont fortement recommandés.

Pour utiliser le fichier `tests.py`, exécutez-le en utilisant le bouton d'exécution de VS Code ou avec la commande suivante dans le dossier contenant les fichiers de test et de jeu :

```
python tests.py
```

Aucune bibliothèque autre que les modules standards `random` et `copy` de Python (déjà importées dans le modèle) n'est autorisée pour ce projet. Toutes les fonctions peuvent et doivent être implémentées uniquement à l'aide des fonctions vues en cours ou mentionnées dans ce document.

Formatage

Un formatage lisible et cohérent est une partie *requise* de ce projet. Une logique correcte ne suffit pas : votre code doit aussi suivre les conventions de mise en forme et de nommage décrites ci-dessous. Si votre fichier n'est pas correctement mis en forme, vous perdrez des points de formatage

même si votre programme fonctionne.

Pourquoi le formatage est important

Une mise en forme cohérente garantit que votre code est facile à lire, facile à déboguer et plus simple à corriger. L'objectif n'est pas de rendre votre code « joli », mais de veiller à ce qu'il respecte un style Python standard, de sorte que n'importe quel lecteur puisse le comprendre sans effort.

autopep8 corrigera automatiquement l'indentation, retirera les espaces superflus et coupera les lignes trop longues conformément à la configuration du projet. Il ne modifiera **pas** votre logique, ne renommerera pas vos variables et ne réécrira pas vos fonctions, donc l'utiliser est toujours sans risque.

Règles de mise en forme obligatoires

Votre soumission doit respecter *les deux* exigences suivantes :

1. **Votre fichier doit déjà correspondre à la configuration autopep8 du projet.** Avant de soumettre, formatez votre code en utilisant le fichier `setup.cfg` fourni. Si le formatage change de nombreuses lignes, c'est le signe que votre mise en forme initiale ne respectait pas les règles.
2. **Votre code doit utiliser des noms cohérents et des commentaires utiles.**
   - Utilisez des noms en *snake_case* pour vos variables et fonctions (par exemple `user_input`, `total_count`, `move_line`).
   - Réservez les variables à une seule lettre (`i`, `j`, `k`) pour les indices de boucle simples.
   - Ajoutez des commentaires pour expliquer les parties non évidentes de votre code, mais évitez les commentaires redondants qui répètent exactement ce que dit déjà le code.

Si vos noms ne sont pas cohérents, si vous utilisez trop de variables à une seule lettre, ou si vous omettez des commentaires essentiels, vous perdrez des points de mise en forme.

Utilisation de `autopep8` dans VS Code

Le projet inclut un fichier `setup.cfg` qui définit les règles de mise en forme.

Pour l'utiliser dans VS Code, installez l'extension autopep8, puis, dans le fichier que vous souhaitez formater, utilisez :

- Shift + Alt + F (Windows/Linux)
- Shift + Option + F (Mac)

Rappel : pour installer une extension dans VS Code, ouvrez la vue Extensions en cliquant sur l'icône appropriée dans la barre latérale de gauche ou en appuyant sur Ctrl + Shift + X (Cmd + Shift + X sur Mac). Recherchez "autopep8" et cliquez sur "Install" sur le résultat approprié.

Si autopep8 ne change rien après utilisation du raccourci clavier, c'est que votre fichier est déjà correctement formaté.