INTRODUCTION

à l'**O**bject **O**riented **P**rogramming (OOP) en Python

Partie 2 : Othello

Objectif

L'objectif de cette seconde partie est de mettre en pratique les connaissances acquises en OOP au travers d'un cas d'étude : développement d'un jeu d'Othello.

Modalités

- Activité devant être réalisée en groupe (2 à 3 personnes)
- Durée : 3 à 3.5 jours

Démarche pédagogique

- Vous ne coderez pas cette activité sur un Jupyter Notebook mais sur **VS Code** : (https://code.visualstudio.com/download)
- Vous organiserez votre code en **modules** et **packages** (https://dev.to/codemouse92/dead-simple-python-project-structure-and-imports-38c6)
- Vous créerez un **environnement virtuel** pour ce projet (https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html)
- Vous générerez un fichier **requirements.txt**

Compétences associées à l'activité

- Toutes les compétences ont déjà été validées lors de la 1ère partie.

Règles de l'Othello

- Vous pouvez trouver les règles du jeu ici : https://www.ffothello.org/othello/regles-du-jeu/

Déroulé du projet

Ce projet est découpé en deux étapes :

1. Développement du jeu d'Othello.
2. (Optionnel) Développement d'un adversaire artificiel contre lequel jouer.

1. **Développement du jeu d'Othello.**

L'idée n'est pas de créer un très beau jeu mais plutôt un jeu fonctionnel. Vous le développerez en mode **console** uniquement. Ci-après un exemple de rendu possible avec un **O** pour les blancs et un * pour les noirs (vous pouvez changer la notation par un **X** par exemple)

Un exemple de rendu possible

Organisez votre code en **classes** : par exemple une classe **board, pawn, engine,...**

Afin de rendre le découpage de votre code le plus clair possible, il vous est recommandé de créer **un fichier .py par classe**.

Pensez à vous répartir le travail en équipe !

Nous vous conseillons très fortement de passer par une étape de **planification papier/ stylo** pour écrire le programme. Quelles classes allez-vous créer ? Quels attributs et quelles méthodes dans quelles classes ? Etc.

Il est possible de vous inspirer du langage de modélisation d'UML (https://www.tutorialspoint.com/uml/uml_basic_notations.htm) pour schématiser votre planification. Ce type de langage permet de représenter les différents attributs et méthodes utilisés dans une classe ainsi que les relations de cette classe avec les autres.

Il existe de nombreux outils permettant de gérer simplement des diagrammes UML. Par exemple :

- DotUML (web)[1] : https://dotuml.com/playground.html
- WebSequenceDiagrams (web, diagrammes de séquence uniquement) : https://www.websequencediagrams.com/#
- LucidChart (web, limited free signup) : https://www.lucidchart.com/pages/
- StarUML (application native) : https://staruml.io/

[1] Extension VS Code : https://marketplace.visualstudio.com/items?itemName=BMLSolutions.dotuml

2. **(Optionnel) Développement d'un adversaire artificiel.**

Pour créer un joueur artificiel avec une intelligence basique, des algorithmes basés sur des arbres évaluant les coups possibles sont souvent utilisés.

Dans notre cas l'**algorithme minimax** sera utilisé pour créer un programme capable de jouer à l'Othello (https://fr.wikipedia.org/wiki/Algorithme_minimax).

Vous trouverez ici (https://www.ffothello.org/informatique/algorithmes/) une analyse et des pseudo-codes vous permettant d'implémenter cet algorithme dans le cadre de l'Othello.

Pour les (beaucoup) plus rapides, vous pourrez implémenter l'**élagage alpha-bêta** pour accélérer les calculs sans avoir besoin de limiter la profondeur de recherche.

Ressource supplémentaire :

- Minimax et élagage alpha-bêta : https://www.youtube.com/watch?v=l-hh51ncgDI&t=29s