Exercice 1 (6 points)

Cet exercice porte sur la programmation orientée objet et la récursivité.

Le jeu puissance 4 se joue à deux joueurs dans une grille de 6 lignes et 7 colonnes. Une couleur de pion, blanc ou noir, est attribuée à chaque joueur. Les joueurs jouent à tour de rôle. Lorsque c'est à son tour de jouer, un joueur choisit une colonne dans laquelle il introduit un pion de sa couleur. La grille de jeu est placée verticalement de sorte que le pion introduit tombe dans la colonne choisie et bute soit sur le bas de la grille, soit sur un autre pion déjà placé. Le but du jeu pour chaque joueur est d'aligner au moins quatre pions de sa couleur dans n'importe quelle direction (horizontalement, verticalement, en diagonale). Le premier à y parvenir a gagné.

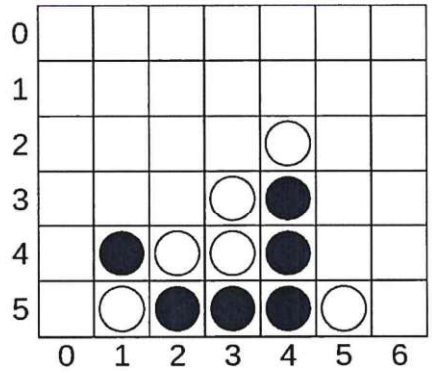

Figure 1. Une partie gagnée par le joueur blanc

Le but de l'exercice est d'implémenter un algorithme appelé min-max permettant à un joueur d'optimiser ses chances de victoire selon un principe simple : à chaque coup possible du joueur, on associe un score calculé en fonction de tous les futurs coups possibles, les siens, mais aussi ceux de son adversaire. Le coup qui a le meilleur score est celui qu'il faut choisir.

Pour simuler l'ensemble des coups possibles, on utilise un arbre dont les nœuds correspondent à un coup. Chaque fils d'un nœud correspond à une possibilité de coup suivant le coup considéré.

Partie A : grille de jeu et score associé

Dans la suite, les deux joueurs seront notés 1 et 2.

Pour gérer la grille de jeu, on va écrire une classe `Grille`. L'unique attribut d'un objet instance de la classe `Grille` est un tableau nommé `grille` de 6 lignes et 7 colonnes, représenté en Python par une liste de listes. Ce tableau contient des entiers qui ne peuvent prendre que trois valeurs : 0, 1 ou 2. La valeur 0 signifie que la case du jeu correspondante est vide. La valeur 1 qu'un pion du joueur 1 se trouve dans la case et la valeur 2 qu'un pion du joueur 2 s'y trouve. La convention de numérotation des lignes et des colonnes dans le tableau est présentée sur la figure 1.

1. Écrire la méthode `__init__(self)` de la classe `Grille` qui définit l'attribut `grille` comme un tableau rempli de 0.

2. Recopier et compléter les lignes 4, 6, 7 et 9 de la méthode `joue(self, colonne, joueur)` suivante de la classe `Grille`. Son but est de tenter de placer un pion du joueur `joueur` dans la colonne dont le numéro est `colonne`. Si le coup est possible, c'est-à-dire si la colonne ne contient pas déjà six pions, alors le pion est placé dans la grille au bon endroit et la fonction renvoie `True`. Si le coup n'est pas possible, la fonction renvoie simplement `False`.

   Remarque importante : la recherche d'une case où placer le pion se fait de bas en haut.

```
1 def joue(self, colonne, joueur):
2     ligne = 5  # on part de la rangée la plus basse
3     while ligne!=-1 and self.grille[ligne][colonne]!=0:
4         ligne = ...
5     if ligne != -1:
6         self.grille[ligne][colonne] = ...
7         return ...
8     else:
9         return ...
```

Voici un exemple de tableau représentant la grille de jeu obtenue à la fin du troisième coup :

```
[[0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 1, 0, 0, 0],
 [0, 0, 1, 2, 0, 0, 0]]
```

3. Écrire le code Python nécessaire pour créer cette grille de jeu en utilisant uniquement les méthodes de la classe `Grille`. Elle sera stockée dans une variable `jeu1`.

À présent, on va écrire une méthode de la classe `Grille` qui associe à une grille un score en fonction des pions déjà placés.

On suppose qu'on a déjà écrit une fonction `valeur_case(ligne, colonne)` qui renvoie le nombre d'alignements de quatre cases contenant la case `(ligne, colonne)`. Par exemple, l'appel `valeur_case(0, 1)` renvoie 4. En effet, il y a quatre alignements de quatre cases qui contiennent la case `(0, 1)` et qui sont représentés à la figure 2.

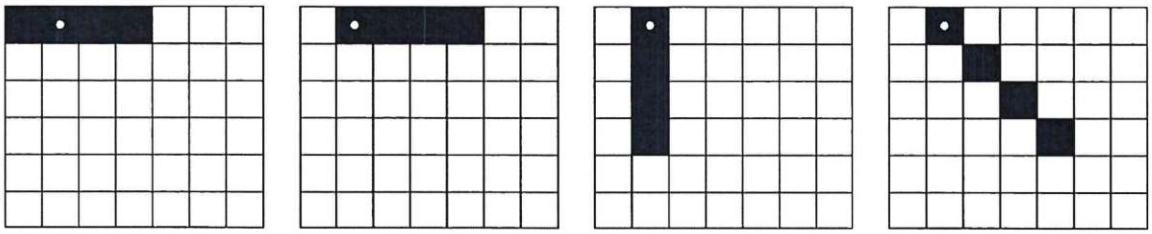

Figure 2. Alignements contenant la case (0, 1).

Le score d'une grille est calculé en additionnant les valeurs de toutes les cases occupées par un pion du joueur 2 et en soustrayant les valeurs de toutes les cases occupées par un pion du joueur 1, la valeur d'une case étant obtenue à l'aide de la fonction `valeur_case`.

4. Donner le score associé à la grille de jeu `jeu1` donnée à la question 3 en expliquant bien le calcul effectué.

5. Écrire la méthode `score(self)` qui renvoie le score associé à la grille de jeu représentée par l'objet.

Partie B : algorithme min-max et création de l'arbre de coups

Dans la suite, on associe un arbre de coups à un joueur et à une grille de jeu. Il permet de simuler tous les coups possibles pour le joueur et son adversaire et de calculer les scores associés à chaque coup en respectant l'algorithme min-max décrit ci-dessous :

- on ne calcule qu'un nombre maximum donné de coups à l'avance. On note `niveau_max` ce nombre qui correspond donc au niveau de profondeur maximale atteint par l'arbre, avec la convention que sa racine est au niveau 0. `niveau_max` est une valeur qu'on considérera comme une constante accessible en tant que variable globale ;

- si un nœud correspond à un coup gagnant pour un des deux joueurs, son score est égal à $-(100 + 10 \times (niveau\_max - niveau))$ pour le joueur 1 et $(100 + 10 \times (niveau\_max - niveau))$ pour le joueur 2, où `niveau` désigne le niveau de profondeur du nœud dans l'arbre de coups ;

- si un nœud se trouve au niveau `niveau_max`, son score est égal au score de la grille de jeu, calculé grâce à la méthode `score` écrite dans la partie A ;

- enfin, dans tous les autres cas, le score d'un nœud est le minimum des scores de ses fils s'il s'agit d'un coup du joueur 1 et le maximum des scores de ses fils s'il s'agit d'un coup du joueur 2.

Afin d'optimiser ses chances de victoire, quand c'est à son tour de jouer, le joueur 1 doit choisir le nœud de niveau 1 correspondant au coup ayant le score le plus petit tandis qu'à son tour, le joueur 2 doit choisir le coup ayant le score le plus grand.

Pour construire l'arbre de coups, on va utiliser une classe `Noeud`.

Une instance de la classe `Noeud` a trois attributs :

- `colonne` qui vaut soit `-1`, soit le numéro de la colonne où le coup est joué, de 0 à 6 ;
- `score` qui correspond au score associé au coup ;
- `suivants` qui est la liste de ses nœuds fils.

La racine d'un arbre de coups est un nœud ne représentant pas un coup, son attribut colonne est initialisé à -1, ses fils représentent tous les premiers coups possibles pour le joueur 1.

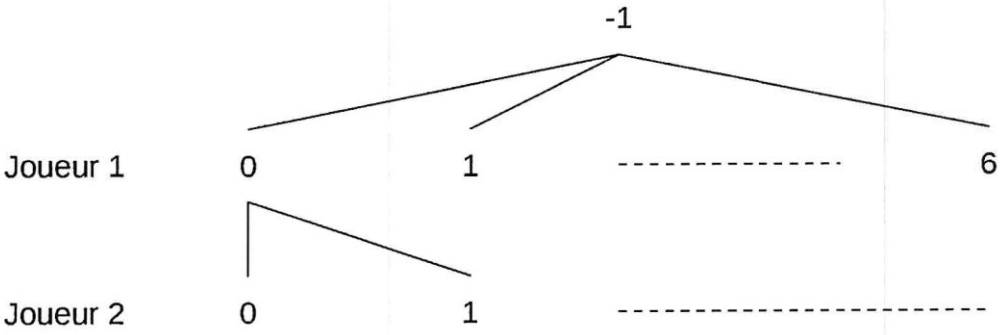

Figure 3. Schéma partiel de l'arbre de coups avec niveau_max = 2 où l'on a indiqué la valeur de l'attribut colonne

6. Compléter la méthode `__init__(self, colonne)` de la classe `Noeud` permettant de créer un nœud de l'arbre symbolisant un coup joué dans la colonne `colonne` avec un score nul et aucun nœud fils.

```
1 class Noeud:
2     def __init__(self, colonne):
3         ...
4         ...
5         ...
```

7. Écrire une méthode `colonne_score_min` de la classe `Noeud` qui renvoie le couple `(colonne, score)` correspondant au numéro de la colonne et au score d'un des fils du nœud pour lequel le score est le plus petit. On suppose que le nœud a au moins un fils.

On suppose dans la suite qu'on a écrit de même une méthode `colonne_score_max` qui renvoie un couple `(colonne, score)` correspondant au numéro de la colonne et au score d'un des fils du nœud pour lequel le score est le plus grand.

On suppose qu'on a ajouté les méthodes suivantes à la classe `Grille` :

- `gagnant(self)` renvoyant le numéro du joueur gagnant (1 ou 2) s'il existe un alignement de quatre de ses pions dans la grille représentée par l'objet, ou 0 s'il n'y a aucun gagnant. On suppose qu'il ne peut y avoir qu'un gagnant ;
- `copie_grille(self)` renvoyant une grille de jeu, copie exacte de la grille représentée par l'objet. Ainsi, si on modifie une copie de la grille, la grille n'est pas modifiée.

8. Recopier et compléter le code de la méthode `calcule_score(self, niveau, joueur, grille)` de la classe `Noeud` suivante, qui attribue un score au nœud représenté par l'objet conformément à l'algorithme min-max décrit plus haut. `niveau` est le niveau de profondeur du nœud dans l'arbre de coups. `joueur` est le numéro du joueur dont le nœud représente le coup. `grille` est un objet `Grille` représentant le jeu juste après que le coup correspondant au nœud a été joué.

```
    def calcule_score(self, niveau, joueur, grille):
        g = grille.gagnant()
        if g == 1:
            self.score = ...
        elif g == 2:
            self.score = ...
        elif niveau == niveau_max:
            self.score = ...
        else:
            for colonne in range(7):
                grille2=grille.copie_grille()
                if grille2.joue(colonne, joueur):
                    nouveau_noeud = ...
                    self.suivants.append(...)
                    nouveau_noeud.calcule_score(...)
            if joueur == 1:
                self.score = ...
            else:
                self.score = ...
```

9. Indiquer pourquoi il n'est pas réaliste d'utiliser cet algorithme pour explorer l'ensemble des parties, ce qui correspond à la valeur 42 pour `niveau_max`.

Partie C : choix du meilleur coup à jouer

10. Utiliser les fonctions précédentes et la classe `Noeud` afin d'écrire une fonction `choisit_coup(grille, joueur)` prenant en arguments une grille de jeu et le numéro d'un joueur et renvoyant le numéro de la colonne où devrait jouer le joueur pour optimiser ses chances de victoire selon le principe de l'algorithme min-max.