

Amérique du Nord – 2026 – sujet1 - Correction

Exercice 1 (6 points)

Question 1

```
class Grille:
    def __init__(self):
        self.grille = [[0 for j in range(7)] for i in range(6)]
```

👉 *Commentaire : on crée ici en compréhension une liste de 6 lignes contenant chacune 7 zéros. Chaque case vide est représentée par la valeur 0.*

On pouvait aussi utiliser deux boucles :

```
class Grille:
    def __init__(self):
        self.grille = []
        for i in range(6):
            ligne = []
            for j in range(7):
                ligne.append(0)
            self.grille.append(ligne)
```

Question 2

```
def joue(self, colonne, joueur):
    ligne = 5 # on part de la rangée la plus basse
    while ligne != -1 and self.grille[ligne][colonne] != 0:
        ligne = ligne - 1 # on remonte d'une case
    if ligne != -1:
        self.grille[ligne][colonne] = joueur
        return True # le pion est placé
    else:
        return False # colonne pleine
```

👉 *Commentaires :*

- *La recherche se fait du bas vers le haut afin de simuler la chute du pion dans la colonne.*
- *Si ligne vaut -1, cela signifie que toute la colonne est remplie.*

- La ligne 3 est donnée par l'énoncé : on la recopie telle quelle (écrire while ligne ≥ 0 and ... serait équivalent).

Question 3

```
jeu1 = Grille()
```

```
jeu1.joue(2, 1)
```

```
jeu1.joue(3, 2)
```

```
jeu1.joue(3, 1)
```

👉 *Commentaire : les coups sont joués dans l'ordre afin de reconstruire exactement la grille donnée dans l'énoncé.*

Question 4

Dans la grille :

- le joueur 1 possède deux pions :
 - o en (5,2)
 - o en (4,3)
- le joueur 2 possède un pion :
 - o en (5,3)

$\text{score} = \Sigma \text{valeur_case}(\text{pions joueur 2}) - \Sigma \text{valeur_case}(\text{pions joueur 1})$

Le score vaut : **$\text{score} = \text{valeur_case}(5,3) - \text{valeur_case}(5,2) - \text{valeur_case}(4,3)$**

Pour calculer valeur_case, il faut compter **combien d'alignements de 4 cases passent par cette case** (horizontal, vertical, diagonales):

valeur_case(5, 2), joueur 1 ligne du bas : **5 alignements**

valeur_case(4, 3) joueur 1 : **10 alignements**

valeur_case(5, 3), joueur 2 : **7 alignements**

$\text{score} = \text{valeur_case}(5,3) - \text{valeur_case}(5,2) - \text{valeur_case}(4,3) = 7 - 5 - 10 = -8$

Question 5

```
def score(self):
    total = 0
    for ligne in range(6):
        for colonne in range(7):
            if self.grille[ligne][colonne] == 1:
                total = total - valeur_case(ligne, colonne)
            elif self.grille[ligne][colonne] == 2:
                total = total + valeur_case(ligne, colonne)
    return total
```

👉 *Commentaire : on parcourt toutes les cases de la grille afin d'additionner ou soustraire les valeurs selon le joueur occupant la case.*

Question 6

```
class Noeud:
    def __init__(self, colonne):
        self.colonne = colonne
        self.score = 0
        self.suivants = []
```

👉 *Commentaire : un nœud mémorise la colonne jouée, son score et la liste de ses fils dans l'arbre des coups.*

Question 7

```
def colonne_score_min(self):
    mini = self.suivants[0]                # premier fils

    for noeud in self.suivants:
        if noeud.score < mini.score:
            mini = noeud                  # nouveau minimum

    return (mini.colonne, mini.score)
```

👉 *Commentaires :*

- On suppose que la liste des fils n'est pas vide, ce qui permet d'initialiser mini avec le premier fils.

- Cette méthode sera utilisée lorsque le joueur 1 cherche à minimiser le score.

Question 8

```
def calcule_score(self, niveau, joueur, grille):
    g = grille.gagnant()
    if g == 1:
        self.score = -(100 + 10 * (niveau_max - niveau)) # J1 gagne
    elif g == 2:
        self.score = 100 + 10 * (niveau_max - niveau) # J2 gagne
    elif niveau == niveau_max:
        self.score = grille.score() # feuille
    else:
        for colonne in range(7):
            grille2 = grille.copie_grille()
            if grille2.joue(colonne, joueur):
                nouveau_noeud = Noeud(colonne)
                self.suivants.append(nouveau_noeud)
                nouveau_noeud.calcule_score(niveau + 1, 3 - joueur, grille2)
        if joueur == 1:
            self.score = self.colonne_score_min()[1] # J1 minimise
        else:
            self.score = self.colonne_score_max()[1] # J2 maximise
```

👉 Commentaires :

- si un joueur a gagné, on attribue immédiatement un score très élevé ou très faible.
- plus une victoire est rapide, plus son score est important grâce au terme $\text{niveau_max} - \text{niveau}$
- on crée une copie de la grille afin d'éviter de modifier la grille originale pendant les simulations.
- l'algorithme min-max suppose que le joueur 1 minimise le score tandis que le joueur 2 le maximise.

Question 9

Il n'est pas réaliste d'explorer toutes les parties possibles car **leur nombre est gigantesque**.

À chaque coup il y a jusqu'à **7 colonnes** possibles, et une partie peut durer jusqu'à **42 coups**. Le nombre de nœuds de l'arbre complet est de l'ordre de 7^{42} , soit environ 3×10^{35} nœuds, un nombre astronomique que même le calculateur le plus puissant ne pourrait pas explorer en un temps raisonnable. C'est pourquoi on limite la profondeur à `niveau_max`.

👉 *Commentaire : la taille de l'arbre augmente de manière exponentielle avec la profondeur.*

Question 10

```
def choisit_coup(grille, joueur):
    racine = Noeud(-1)
    racine.calcul_score(0, joueur, grille) # construit tout l'arbre
    if joueur == 1:
        return racine.colonne_score_min()[0]
    else:
        return racine.colonne_score_max()[0]
```

👉 *Commentaire :*

- La racine correspond à l'état actuel du jeu et ne représente pas un coup réel.
 - Après le calcul des scores, on choisit la colonne correspondant au meilleur score pour le joueur considéré.
-