Concours CPGE EPITA-IPSA-ESME
2024

Corrigé de l'épreuve de Sciences du numérique MP-PC-PSI-PT-TSI

Le sujet est composé d'un problème et d'un questionnaire à choix multiples. Le questionnaire à choix multiples devra être inséré dans votre copie. Les fonctions à produire dans ce sujet devront être rédigées en langage Python et se limiter aux éléments décrits ci-après. Il est possible d'écrire des fonctions auxiliaires non explicitement demandées à condition de les documenter et de les définir avant d'en faire usage.

Éléments du langage Python autorisés

Types de base

- Opérations sur les entiers (`int`) : `+`, `-`, `*`, `//`, `**`, `%` avec des opérandes positifs.
- Opérations sur les flottants (`float`) : `+`, `-`, `*`, `/`, `**`.
- Opérations sur les booléens (`bool`) : `not`, `or`, `and`.
- Comparaisons `==`, `!=`, `<`, `>`, `<=`, `>=`.

Types structurés

- Structures indicées immuables (chaînes, n-uplets) :
  - longueur `len` ;
  - accès ;
  - concaténation `+` ;
  - répétition `*` ;
  - extraction de tranche (slicing).
- Listes :
  - création par description complète, par compréhension ou par `append` successifs ;
  - longueur `len` ;
  - accès ;
  - concaténation `+` ;
  - extraction de tranche (slicing) ;
  - répétition `*` ;
  - `pop` sans paramètres.
- Dictionnaires : création, accès, insertion, longueur `len`.

Structures de contrôle

- Instruction d'affectation avec `=`, `+=` ou `-=` et dépaquetage de n-uplets : `(x,y)=(4,2)`.
- Instruction conditionnelle : `if`, `elif`, `else`.
- Boucle `while` (sans `else`). `break`, `return` dans un corps de boucle.
- Boucle `for` (sans `else`) et itération sur `range(a, b)`, une chaîne, un n-uplet, une liste, un dictionnaire au travers des méthodes `keys` et `items`.
- Définition d'une fonction `def f(`p_1`, ..., `p_n`)`, `return`.

Remarques particulières :

- Le mot clé `in` pourra être utilisé dans une boucle `for` (`for x in ...`) mais pas pour effectuer un test d'appartenance (`if x in ...`).
- L'usage de modules n'est pas autorisé ; en particulier l'usage de la fonction `deepcopy` du module `copy` est proscrit.

Problème – Jeu de reversi

Déroulé du jeu et premières fonctions

Le jeu de reversi se compose d'un plateau carré de $n = 8$ cases de côté ainsi que de jetons blancs et noirs. Deux joueurs s'affrontent à tour de rôle en posant un jeton sur le plateau. Le joueur qui possède les jetons noirs commence la partie et sera appelé J_1 tandis que son adversaire sera appelé J_2.

On considère dans ce problème que le plateau de jeu est initialement rempli comme représenté par la figure 1.

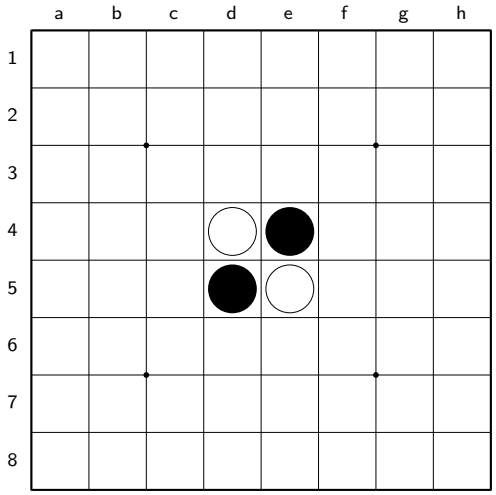


FIGURE 1 – Plateau de départ du jeu

Les lignes sont traditionnellement numérotées de 1 à 8 et les colonnes des lettres a à h. Dans ce sujet l'implémentation considèrera que les lignes et les colonnes sont numérotées de 0 à 7, en préservant l'ordre de nommage de la numérotation traditionnelle. Ainsi la case $(8, a)$ sera représentée informatiquement par le couple `(7,0)`.

On décide de représenter le plateau de jeu `p` par une liste de listes. Le contenu de la case `(i,j)` sera représenté par la valeur de `p[i][j]`. Cette valeur sera `0` lorsque la case est vide, `1` lorsqu'un jeton noir y est présent et `-1` lorsqu'un jeton blanc s'y trouve. Ainsi la valeur de `p[3][4]` représente donc la valeur de la case $(4, e)$ de valeur `1`.

Les joueurs seront représentés par la couleur de leurs jetons, ainsi J_1 sera représenté par la valeur `1` et J_2 par la valeur `-1`.

Question 1 Écrire une fonction `plateau_depart` sans paramètre et retournant le plateau de la figure 1.

```
def plateau_depart():
    p = []
    for i in range(8):
        p.append([0]*8)
    p[3][3] = -1
    p[4][4] = -1
    p[3][4] = 1
    p[4][3] = 1
    return p
```

On appellera **joueur actif** le joueur dont c'est le tour de jouer. Ce joueur doit poser un jeton sur une **case jouable**, c'est-à-dire une case libre lui permettant de faire une capture (voir ci-après) ; dans le cas où il n'existe pas de telle case, l'opposant rejoue. Si plus personne ne peut jouer, le jeu s'arrête. La figure 3 présente une configuration du plateau de jeu où les coups jouables par J_1 ont été représentés par un •.

On parle de **capture**, lorsqu'il existe un alignement d'au moins un jeton adverse (en ligne, en colonne ou en diagonale) compris entre un jeton du joueur actif et le jeton qu'il s'apprête à poser. Lorsqu'une capture a lieu, l'ensemble des jetons adverses de l'alignement sont **retournés** ; c'est-à-dire qu'ils changent de couleur et donc de propriétaire. La **direction** d'une capture sera orientée à partir du jeton posé et dans la direction de l'alignement des jetons à retourner. On représentera les directions de capture par un couple `(dx,dy)` conformément au tableau de la figure 2 :

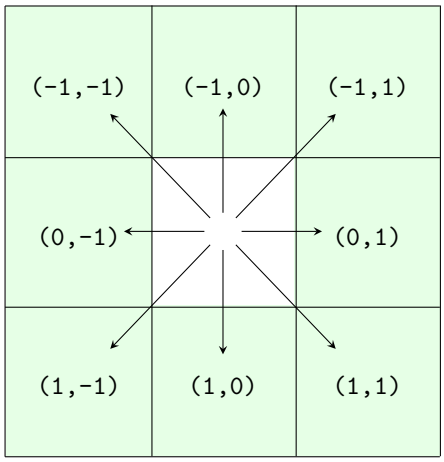


FIGURE 2 – Directions des alignements à partir du jeton posé

Dans la configuration de la figure 3, J_1 peut par exemple :

- poser un jeton noir en $(2, e)$. Il capturera alors les 2 jetons adverses situés en $(3, e)$ et en $(4, e)$ (selon la direction `(1,0)`).
- poser un jeton noir en $(3, d)$. Dans ce cas il y aura une capture selon deux alignements : l'alignement entre $(3, d)$ et $(5, d)$, de direction `(1,0)`, capturera un jeton et celui entre $(3, d)$ et $(5, f)$, de direction `(1,1)`, en capturera un autre. La figure 4 représente le plateau une fois ces captures effectuées.

On notera qu'une capture peut donc s'effectuer selon plusieurs directions.

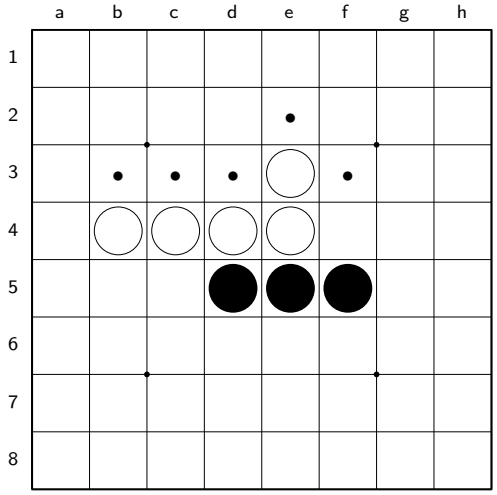


FIGURE 3 – Cases valides pour J_1

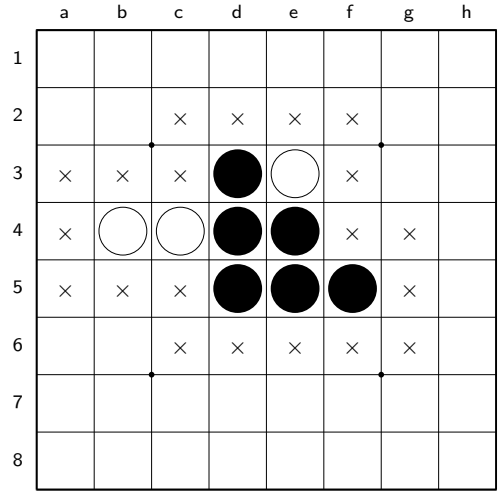
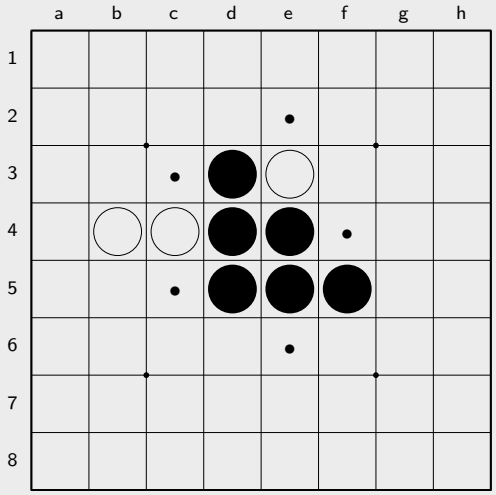


FIGURE 4 – Cases frontières après un coup de J_1 en $(3, d)$

Le jeu se termine lorsque plus aucun joueur ne peut jouer, en général cela se produit lorsque le plateau est entièrement rempli mais il est également possible que J_1 et J_2 n'aient plus de case jouable. Est considéré comme gagnant le joueur possédant le plus de jetons de sa couleur. On définit alors le **score** d'un plateau comme étant la différence entre le nombre de ses jetons noirs et le nombre de ses jetons blancs.

Question 2 Donner les 5 coups jouables de J_2 dans la configuration décrite par la figure 4.

Représentons les coups jouables en marquant les cases d'un • :



Question 3 Écrire une fonction `score` qui étant donné un plateau `p` retourne son score.

```
def score(p):
    s=0
    for l in p:
        for c in l:
            s+=c
    return s
```

Question 4 Écrire une fonction `cases_voisines` qui étant donné une case `(x,y)` retourne la liste de ses cases voisines (en ligne, colonne et diagonale). Ainsi `cases_voisines (0,1)` retournera `[(0,0),(0,2),(1,0),(1,1),(1,2)]`.

```
def cases_voisines(c):
    x,y=c
    return [(k,l) for k in [x-1,x,x+1]
                  for l in [y-1,y,y+1]
                      if (not (k==x and l==y)) and 0<=k<=7 and 0<=l<=7]
```

Question 5 Écrire une fonction `cases_libres` prenant en paramètre un plateau `p` et une liste de cases `l`. Cette fonction retournera la liste des cases de `l` ne contenant pas de jeton sur le plateau `p`.

```
def cases_libres(p,l):
    return [(i,j) for (i,j) in l if p[i][j]==0]
```

On considère à présent les cases vides ayant au moins une case voisine occupée par un jeton. Ces cases seront appelées **cases frontières** (voir figure 4). On remarquera que les cases jouables font nécessairement partie des cases frontières.

Question 6 Justifier que le nombre de cases frontières augmente d'au plus 6 cases à chaque jeton posé.

On considère le graphe non orienté dont les sommets sont les cases non vides du plateau et dont les arêtes relient les sommets dont les cases sont voisines. Ce graphe est connexe car il est connexe au début du jeu et qu'à chaque tour de jeu on ne peut déposer de pions que sur une case voisine d'une case déjà occupée.

De plus lorsqu'un joueur pose un jeton, la case de la frontière où le jeton a été posée est retiré de la frontière et on ajoute ses cases voisines libres ; il y en a au plus 7 (une des cases voisines est nécessairement occupée puisque le graphe est connexe). Finalement la frontière ne peut augmenter que d'au plus 6 cases par tour de jeu.

Question 7 Écrire une fonction `cases_frontieres` prenant en paramètre un plateau `p`, une liste de cases frontieres `l` et une case jouable `c`. Cette fonction retournera la liste des cases frontières du plateau de jeu obtenu après avoir joué en case `c` sur le plateau `p`. On fera en sorte que la liste produite ne contienne pas de doublon.

```
def cases_frontieres(p,l,c):
    v = cases_libres(p,cases_voisines(c))
    old = [(i,j) for (i,j) in l if (i,j)!=c]
    new = [e for e in v if e not in old]
    return old + new
```

Question 8 Écrire une fonction `deplacer` qui étant donné une case `c` et une direction `d` retourne la case obtenue après déplacement d'une case dans la direction `d`. On autorisera dans cette question les cases situées en dehors du plateau de jeu, ainsi `deplacer (0,7) (-1,1)` retournera `(-1,8)`.

```
def deplacer(c,d):
    (x,y)=c
    (dx,dy)=d
    return (x+dx,y+dy)
```

Question 9 Écrire une fonction `est_direction_possible` prenant en paramètre un plateau `p`, une case de la frontière `c`, une direction `d` et le joueur actif `j` (`1` pour J_1 et `-1` pour J_2). Cette fonction retournera `True` si le joueur actif dispose d'une capture selon la direction `d` en jouant en case `c` et `False` sinon.

```
def est_direction_possible(p,c,d,j):
    (x,y) = deplacer(c,d)
    jetons_a_retourner = False
    while 0<=x<=7 and 0<=y<=7 and p[x][y]==-j:
        jetons_a_retourner = True
        (x,y) = deplacer((x,y),d)
    return (jetons_a_retourner and 0<=x<=7 and 0<=y<=7 and p[x][y]==j)
```

Question 10 Écrire une fonction `directions_possibles` prenant en paramètre un plateau `p`, une case de la frontière `c` et le joueur actif `j`. Cette fonction retournera la liste des directions permettant une capture si le joueur actif jouait en case `c`.

```
def directions_possibles(p,c,j):
    directions = [(dx,dy) for dx in [-1,0,1] for dy in [-1,0,1] if dx!=0 or dy!=0]
    return [d for d in directions if est_direction_possible(p,c,d,j)]
```

Question 11 Écrire une fonction `cases_jouables` prenant en paramètre un plateau de jeu `p`, la liste des cases frontières `l` et le joueur actif `j`. Cette fonction retournera la liste des cases jouables pour le joueur considéré.

```
def cases_jouables(p,l,j):
    cases = []
    for c in l:
        if directions_possibles(p,c,j)!=[]:
            cases.append(c)
    return cases
```

Question 12 Écrire une fonction `capturer` prenant en paramètre un plateau `p`, une case jouable `c`, une direction `d`. Cette fonction retournera `None` et modifiera en place le plateau de jeu `p` de façon à retourner les cases adverses selon l'alignement partant de la case `c` dans la direction `d`.

```
def capturer(p,c,d):
    (x,y)=deplacer(c,d)
    j_adv = p[x][y]
    while p[x][y]==j_adv:
        p[x][y]=-p[x][y]
        (x,y)=deplacer((x,y),d)
    return None
```

Question 13 Écrire une fonction `copier_plateau` prenant en paramètre un plateau `p` et retournant une copie du plateau.

```
def copier_plateau(p):
    plateau=[]
    for l in p:
        tmp = []
        for c in l:
            tmp.append(c)
        plateau.append(tmp)
    return plateau
```

Question 14 Écrire une fonction `poser_jeton` prenant en paramètre un plateau `p`, une case jouable `c` et le joueur actif `j`. Cette fonction retournera le plateau obtenu à partir du plateau `p` après que le joueur `j` ai joué un jeton en case `c`. On fera attention à ne pas modifier le plateau `p` passé en paramètre.

```
def poser_jeton(p,c,j):
    plateau = copier_plateau(p)
    plateau[c[0]][c[1]] = j
    for d in directions_possibles(plateau,c,j):
        capturer(plateau,c,d)
    return plateau
```

Implémentation d'une intelligence artificielle

On souhaite à présent concevoir des fonctions permettant de jouer seul contre la machine.

Question 15 En considérant que le temps nécessaire à l'étude d'un plateau de jeu est de $0{,}2$ millisecondes, que l'on dispose en moyenne de 7 coups jouables par tour de jeu et que l'âge de l'univers est d'environ 4×10^{17} secondes, estimer le nombre de coups à l'avance qu'il aurait été possible de prévoir complètement si on avait commencé l'exploration dès la création de l'univers.

En prévoyant n coups à l'avance, on devra étudier 7^n plateaux de jeu, ce qui prendra $7^n \times 0,2 \times 10^{-3}$ secondes. Or $7^n \times 0,2 \times 10^{-3} = 4 \times 10^{17} \Leftrightarrow n = \dfrac{\ln\left(2 \times 10^{21}\right)}{\ln(7)} \simeq 25,2$. On aurait donc pu prévoir au plus 25 coups à l'avance.

Faute de temps, il n'est pas envisageable (du moins dès le début du jeu) d'explorer tous les plateaux possibles. Cela nous conduit à devoir évaluer la qualité d'un plateau de jeu alors qu'il reste des coups jouables. On se dote d'une **fonction d'évaluation** f s'appliquant à un plateau de jeu et retournant un entier représentant une **évaluation** quantifiée du plateau. Plus la note est élevée, plus elle est avantageuse pour J_1 ; dans le cas contraire elle est avantageuse pour J_2.

Une première idée pour évaluer un plateau serait d'utiliser le résultat de la fonction `score`. La présence de jetons sur certaines positions stratégiques du plateau est également un indicateur intéressant à prendre en considération. Par exemple, tout jeton posé dans un coin du plateau ne pourra plus être retourné. Dès lors on choisit d'évaluer un plateau par la fonction d'évaluation $f : p \mapsto \sum_{i=0}^{7}\sum_{j=0}^{7} p[i][j].F[i][j]$ où $F[i][j]$ est une valeur quantifiant l'importance de la capture de la case située en ligne i et colonne j. Ainsi si on considère que la capture des coins du plateau est importante, on attribuera une forte valeur à `F[0][0]`, `F[0][7]`, `F[7][0]` et `F[7][7]`. Dans ce sujet on ne cherche pas à déterminer F ; on prendra alors simplement f comme paramètre de nos fonctions.

Question 16 Écrire une fonction `ia_naive` qui étant donné un plateau `p`, la liste des cases frontières `l`, une fonction d'évaluation `f` et le joueur actif `j`, retourne un couple `(c,val)` composé de `c`, la case jouable conduisant le joueur actif à un plateau de valeur maximale, et de `val`, la valeur attribuée à ce coup. On suppose que cette fonction ne sera appelée que dans le cas où le joueur actif dispose d'un coup jouable et on fera attention à ne pas modifier le plateau `p` passé en paramètre.

```
def ia_naive(p,f,l,j):
    cases   = cases_jouables(p,l,j)
    valeurs = []
    for c in cases :
        plateau = copier_plateau(p)
        plateau = poser_jeton(plateau,c,j)
        valeurs.append(f(plateau))
    i_max = 0
    for i in range(1,len(cases)):
        if j*valeurs[i_max]<j*valeurs[i]:
            i_max = i
    return cases[i_max],valeurs[i_max]
```

On souhaite à présent développer un algorithme envisageant tous les plateaux possibles, n coups à l'avance. La figure 5 présente un arbre d'exploration à 3 coups d'avance. Les nœuds de l'arbre, représentés par un •, correspondent chacun à une configuration du plateau. Lorsqu'il est possible de passer d'un plateau à un autre en jouant en case $c_{i,j}$, on relie les deux nœuds correspondant par une arête d'étiquette $c_{i,j}$. Les $c_{i,j}$ représentent donc les cases jouables pour le joueur actif.

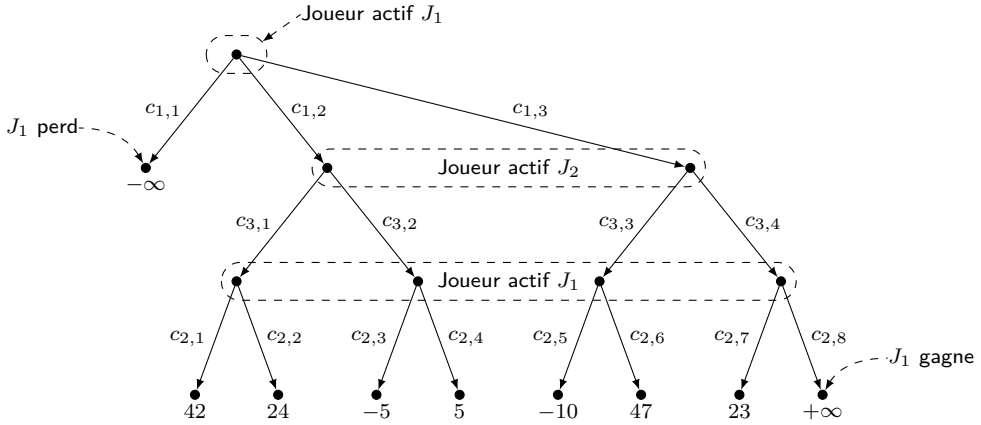


FIGURE 5 – Exemple d'arbre d'exploration de profondeur 3

Dans cet exemple, J_1 commence à jouer et dispose de 3 cases possibles pour poser son jeton. S'il joue en case $c_{1,1}$, on suppose que la partie s'arrête (par exemple lorsque plus aucun joueur ne dispose de coup jouable) et que J_1 dispose de moins de jetons que J_2, auquel cas il perd la partie. Il peut aussi jouer en case $c_{1,2}$ ou en case $c_{1,3}$. Dans ce cas, J_2 joue à son tour et dispose dans tous les cas de 2 cases jouables. J_1 peut alors jouer à son tour une troisième fois. On a alors envisagé 3 coups d'avance et on arrête ici notre exploration.

Afin de décider du meilleur coup à jouer, on doit évaluer les 9 feuilles de l'arbre. Dans le cas où J_1 perd, on évaluera le plateau à $-\infty$, dans le cas où J_1 gagne, on évaluera le plateau à $+\infty$. Dans les autres cas, on fera appel à la fonction d'évaluation f pour évaluer le plateau de jeu. Pour des raisons d'implémentation on choisira pour $+\infty$ la valeur 10000 et pour $-\infty$ la valeur -10000 en faisant l'hypothèse que la fonction f renvoie des entiers strictement compris entre ces deux valeurs.

Lors du tour de chaque joueur, l'algorithme considèrera que chaque joueur joue le coup qui lui est le plus avantageux au regard de l'évaluation du plateau obtenue. La figure 6 représente l'arbre des évaluations déduites (de bas en haut) par l'algorithme. Finalement dans notre exemple le joueur J_1 aura intérêt à jouer en case $c_{1,3}$.

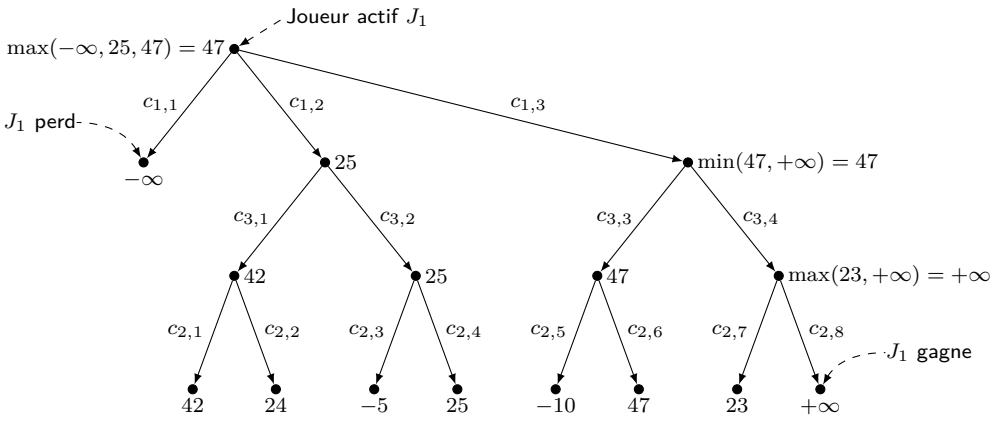


FIGURE 6 – Exemple d'arbre d'exploration de profondeur 3

Question 17 Écrire une fonction `ia_minmax` prenant en paramètre un plateau `p`, la liste des cases frontières `l`, une fonction d'évaluation `f`, le joueur actif `j` et un entier `n` non nul. Cette fonction devra évaluer la situation du jeu avec `n` coups d'avance et retourner un couple composé du meilleur coup à jouer et de l'évaluation de ce coup. Cette fonction tiendra compte du fait que la partie peut s'arrêter avant de pouvoir jouer `n` coups. On fera l'hypothèse que lors de l'exploration, tous les joueurs ont toujours au moins un coup jouable.

```
def ia_minmax(p,f,l,j,n):
    cases   = cases_jouables(p,l,j)
    valeurs = []
    for c in cases :
        plateau = copier_plateau(p)
        plateau = poser_jeton(plateau,c,j)
        if n==1:
            valeur = f(plateau)
        else:
            (case,valeur) = ia_minmax(plateau,f,cases_frontieres(plateau,l,c),-j,n-1)
        valeurs.append(valeur)
    i_max = 0
    for i in range(1,len(cases)):
        if j*valeurs[i_max]<j*valeurs[i]:
            i_max = i
    return cases[i_max],valeurs[i_max]
```

Question 18 Reprendre la fonction précédente pour qu'elle puisse traiter également les cas où un joueur serait contraint de passer son tour et les cas où la partie se termine. On pourra ajouter un paramètre booléen `tp` pour indiquer lors des appels récursifs que le joueur précédent à été contraint de passer son tour faute de coup jouable.

```
def ia_minmax(p,f,l,j,n,tp=False):
    cases = cases_jouables(p,l,j)
    if cases==[]:
        s = score(p)
        if tp or n==1:
            if s>0:
                return None,10000
            elif s==0:
                return None,0
            else:
                return None,-10000
        else:
            return ia_minmax(p,f,l,-j,n-1,True)
    else:
        valeurs = []
```

```
        for c in cases :
            plateau = copier_plateau(p)
            plateau = poser_jeton(plateau,c,j)
            if n==1:
                valeur = f(plateau)
            else:
                (case,valeur) = ia_minmax(plateau,f,cases_frontieres(plateau,l,c),-j,n-1)
            valeurs.append(valeur)
        i_max = 0
        for i in range(1,len(cases)):
            if j*valeurs[i_max]<j*valeurs[i]:
                i_max = i
        return cases[i_max],valeurs[i_max]
```

Evaluation statistique des parties

On associe à tout plateau p son **encodage** défini par la quantité : $\sum_{i=0}^{7}\sum_{j=0}^{7}(p[i][j]+1)3^{8i+j}$

Question 19 Écrire une fonction `encode` qui étant donné un plateau `p` retourne son encodage.

```
def encode(p):
    e=0
    for i in range(8):
        for j in range(8):
            e += (p[i][j]+1)*(3**(8*i+j))
    return e
```

Question 20 Écrire une fonction `decode` qui étant donné un encodage `e` retourne le plateau `p` correspondant.

```
def decode(e):
    p=[]
    for i in range(8):
        ligne = e%(3**8)
        p.append([])
        for j in range(8):
            case = ligne%3
            p[i].append(case-1)
            ligne = ligne//3
        e = e//(3**8)
    return p
```

Question 21 Quel est le plus grand entier atteignable par cet encodage ?

Le plus grand entier atteignable s'obtient en choisissant un plateau remplies de cases noires.
On obtient alors comme valeur $\sum_{i=0}^{7}\sum_{j=0}^{7}2.3^{8i+j} = 2\sum_{i=0}^{7}3^{8i}\sum_{j=0}^{7}3^{j} = 2\frac{1-(3^8)^8}{1-(3^8)}\frac{1-3^8}{1-3} = 3^{64}-1 > 2^{64}$.

On stocke à présent dans une base de données quelques informations sur les parties jouées et on fera l'hypothèse que les encodages de plateaux peuvent, malgré leur taille, être stockés dans notre base de données.

On répartit ces informations en 3 tables dont on présente des extraits ci-après. La première ligne représente les noms des champs et les suivantes les enregistrements. La clé primaire de chaque table est indiquée en première colonne. Les clés étrangères seront soulignées.

Une première table `joueurs` permet de stocker les informations personnelles des joueurs :

idJoueur	nomComplet	dateDeNaissance	adresse	···
⋮	⋮	⋮	⋮	⋮
42	Beth Harmon	1948-11-02	···	···
⋮	⋮	⋮	⋮	⋮
1926	John von Neumann	1903-12-28	···	···
⋮	⋮	⋮	⋮	⋮

Une seconde table `parties` permet de stocker les statistiques d'une partie terminée. `joueurs_idJ1` permet de déterminer le joueur ayant les jetons noirs, `joueurs_idJ2` celui ayant les jetons blancs et `joueurs_idGagnant` permet d'identifier le joueur ayant remporté la partie. Ainsi les valeurs des champs `joueurs_idJ1`, `joueurs_idJ2` et `joueurs_idGagnant` sont des clés étrangères du champ `idJoueur` de la table `joueurs`. Le champ score contiendra le score du plateau de fin de partie.

idPartie	joueurs_idJ1	joueurs_idJ2	joueurs_idGagnant	score
⋮	⋮	⋮	⋮	⋮
12	42	1926	42	7
⋮	⋮	⋮	⋮	⋮

Une troisième table `plateaux` recense les informations des plateaux de jeu pour lesquels l'intelligence artificielle a eu à se prononcer. On stocke alors l'identifiant `idPartie` de la partie concernée dans `parties_idPartie` qui sera donc une clé étrangère, `codePlateau` contiendra l'encodage du plateau analysé, `profondeur` la profondeur de l'analyse, `evaluation` le résultat de l'évaluation du plateau et `joueurActif` vaudra `1` si le joueur actif joue les jetons noirs et `-1` sinon. Enfin, les champs `caseX` et `caseY` contiendront les numéros de ligne et de colonne du meilleur coup à jouer trouvé.

idPlateau	parties_idPartie	codePlateau	profondeur	evaluation	joueurActif	caseX	caseY
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
421	12	3917277946	3	13	-1	3	4
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Question 22 Écrire une requête SQL retournant les numéros de ligne et de colonne du meilleur coup obtenu pour le plateau d'encodage 3917277946 lors du tour du joueur J_1 et à la plus grande profondeur.

```sql
SELECT caseX,caseY FROM plateaux
WHERE codePlateau=3917277946 AND joueurActif=1
ORDER BY profondeur DESC
LIMIT 1
```

Question 23 Écrire une requête SQL qui donne le score moyen de la joueuse « Beth Harmon » lorsqu'elle joue avec les jetons noirs.

```sql
SELECT nomComplet,AVG(score)
FROM parties
JOIN joueurs ON joueurs.idJoueur = parties.joueurs_idJ1
WHERE nomComplet LIKE "Beth Harmon"
GROUP BY nomComplet
```

Question 24 Écrire une requête SQL qui liste, pour les 3 parties avec le meilleur score remportées par J_2, le nom du gagnant et le score de la partie. On fera ici l'hypothèse qu'il n'y a pas de cas d'égalité.

```sql
SELECT nomComplet,score
FROM joueurs JOIN parties ON parties.joueurs_idGagnant = joueurs.idJoueur
WHERE score < 0
ORDER BY score ASC
LIMIT 3
```

QCM – MP-PC-PSI-PT-TSI

Numéro de candidat :
Document à insérer dans votre copie

Dans ce questionnaire à choix multiples, chaque question comporte une ou plusieurs bonnes réponses.
Chaque réponse correcte fait gagner des points, mais chaque réponse fausse annule tous les points de la question.
Les questions sont majoritairement formulées au pluriel par commodité d'expression. Cela n'implique pas nécessairement qu'elles admettent plusieurs réponses correctes.

On considère les deux fonctions Python suivantes :

```
def fonction1(n):
    if n==0:
        return True
    else:
        return fonction2(n-1)
```

```
def fonction2(n):
    if n==0:
        return False
    else:
        return fonction1(n-1)
```

1. Soit `n` un entier naturel, alors `fonction1(n)` :
   - ☐ retourne la même valeur que `fonction2(n)`.
   - ■ retourne la négation de `fonction2(n)`.
   - ■ cherche à déterminer si `n` est pair.
   - ☐ cherche à déterminer si `n` est premier.

2. Une fonction de hachage :
   - ☐ permet de découper un problème en sous problèmes pour le résoudre.
   - ■ associe à une valeur arbitrairement longue une valeur dans un intervalle restreint.
   - ■ permet d'implémenter la structure de dictionnaire.
   - ☐ permet d'effectuer une dichotomie.

3. Lorsque l'on effectue un parcours de graphe en largeur :
   - ■ on utilise habituellement une structure de file.
   - ☐ on utilise habituellement une structure de pile.
   - ■ on a besoin de savoir quels sommets ont été visités.
   - ☐ on a besoin de savoir quelles arêtes ont été empruntées.

4. Une recherche dichotomique d'un élément dans un tableau trié :
   - ■ donne toujours un résultat exact.
   - ☐ est de complexité linéaire en temps.
   - ☐ ne s'applique que lorsque les éléments du tableau sont distincts.
   - ☐ ne s'applique qu'à des tableaux d'entiers.

5. Parmi les algorithmes de tris suivants, quels sont ceux reposant sur des comparaisons entre les éléments à trier :
   - ■ le tri par insertion.
   - ■ le tri par sélection.
   - ☐ le tri par comptage.
   - ■ le tri par partition-fusion.

Fin du corrigé