Jeu des tablettes

Le jeu de tablettes est un **jeu à 2 joueurs jouant chacun à tour de rôle** inspiré du Babylone. Il est constitué de tablettes de couleurs à empiler selon des règles simples jusqu'à ce qu'il ne soit plus possible d'agir, le joueur ne pouvant plus rien faire ayant perdu.

Dans le jeu classique, il y a C=4 couleurs et N=3 tablettes par couleur.
Au départ, chaque tablette est posée seule sur la table. Il est possible d'empiler deux piles si au moins l'une des 2 conditions suivantes est respectée :

- Elles ont la même taille
- La tablette supérieure est de la même couleur

Un peu de vocabulaire lié à la théorie des jeux :

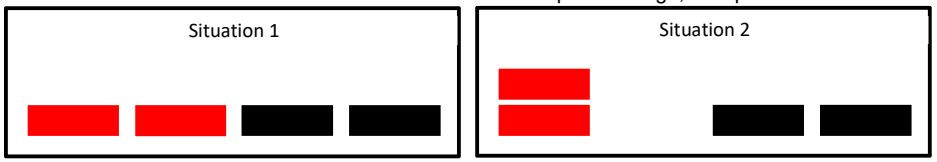
A chaque étape du jeu, les joueurs jouant à tour de rôle, le nombre de piles diminue de 1 et conduit donc à une nouvelle situation. Ce jeu ne présente donc pas de cycle (**acyclique**) et toute partie est finie par la présence de positions sans successeurs formant un sous ensemble à atteindre pour chaque joueur, on parle de **jeu d'accessibilité**. Parmi les 3 types d'états finaux (J1 gagne, J2 gagne, match nul), on remarque que le match nul n'existe pas dans le jeu des tablettes.
Par ailleurs, ce jeu est dit **à information totale** : à tout instant, chacun des joueurs à une information complète de l'état du jeu (rien n'est caché).
Les décisions de jeu sont prises en fonction de la situation présente sans tenir compte des situations passées de la partie ou des parties précédentes : **jeu sans mémoire**.
Dans une situation, une décision amène toujours à la même situation (**jeu sans hasard ou déterministe**), et ne dépend que de la situation et non du joueur (**jeu impartial**).
On dit que c'est **un jeu à somme nulle**, c'est-à-dire que la somme des gains et des pertes est égale à 0. Le gain de l'un constitue obligatoirement une perte pour l'autre. Dans le cas de ce jeu, cela veut dire que si l'un gagne, l'autre perd.

Partie 1 - Lister les coups possibles

On représente une situation de jeu par une liste x de listes à deux entiers [ci,hi] correspondant à chaque pile pi=x[i] avec :

- ci la couleur (que l'on définira par entier positif)
- hi la hauteur de la pile (nombre de tablettes d'une pile)

Soient les 2 situations de jeu suivantes
Nombre de tablettes /couleur : N=2 Couleurs : ci=0 pour le rouge, ci=1 pour le noir



Q1- Donner **l'état de la liste x** dans les 2 situations de jeu proposées ci-dessus.
Q2- On souhaite déterminer l'ensemble des situations de jeu possibles X stockées dans une liste LX à partir d'une situation initiale x en empilant deux piles pi et pj (pour tous les i et j nécessaires) lorsque c'est possible :

- Empilement de pi sur pj
- Si le résultat est différent du premier empilement, empilement de pj sur pi

Rappel : la méthode sort des listes L.sort() permet de trier L de manière lexicographique, soit si L contient des listes, selon la première composante, puis pour la même première composante, selon la seconde etc.
Pour rendre facile la détection de situations semblables dans la suite (ex : [[1,2],[0,1],[0,1]]= [[0,1],[0,1],[1,2]]), on triera les listes X avant de les insérer dans les listes résultats.

Créer la **fonction empile(x, i, j)** prenant en paramètres la situation x et les indices i,j de 2 piles pi et pj tels que i≠j. La fonction retourne la liste de listes LXij des situations atteignables par empilement lorsque les règles sont respectées de pi sur pj et de pj sur pi si différent.

Exemples d'exécution :

```
# tests de la fonction empile
x = [ [0,1] , [0,1] , [1,1] , [1,1] ] # situation 1
res1 = empile( x, 0, 1 )
print( res1 )   # affiche [[[0, 2], [1, 1], [1, 1]]]

x = [ [0,1],[0,1],[1,1],[1,1] ]  # situation 1
res2 = empile(x,0,3)
print(res2)     # affiche [[[0, 1], [0, 2], [1, 1]], [[0, 1], [1, 1], [1, 2]]]

x = [[0,2],[1,1],[1,1]]
res3 = empile(x,0,1)
print(res3)     # affiche []
```

Q3- Créer la **fonction coups(x)** qui prend en paramètre une situation x (liste) et retourne une liste de listes LX de toutes les situations différentes atteignables depuis x.
Vous utiliserez la fonction empile de la Q2 pour mettre en place cette fonction.
Il faudra veiller à ne pas avoir de doublons dans LX : il ne peut pas y avoir 2 fois la même situation finale.

Exemples d'exécution :

```
x = [ [0,2],[0,3],[1,1],[1,2] ]
lesCoups1 = coups(x)
print( lesCoups1 )
# affiche [[[0, 5], [1, 1], [1, 2]], [[0, 3], [0, 4], [1, 1]], [[0, 3], [1, 1],
[1, 4]], [[0, 2], [0, 3], [1, 3]]]

x = [[0,1],[0,1],[1,1],[1,1]]
lesCoups2 = coups(x)
print( lesCoups2 )
# affiche [[[0, 2], [1, 1], [1, 1]], [[0, 1], [0, 2], [1, 1]], [[0, 1], [1, 1],
[1, 2]], [[0, 1], [0, 1], [1, 2]]]
```

Partie 2 : Création du graphe du jeu

Q4- Créer une **fonction init(C, N)** prenant en paramètre le nombre de couleurs C et le nombre de tablettes par couleur N et renvoyant la liste x0 situation initiale du jeu triée, par ordre lexicographique.

Exemple d'exécution :

```
test = init(3,4)
print(test)
# affiche [[0, 1], [0, 1], [0, 1], [0, 1], [1, 1], [1, 1], [1, 1], [1, 1], [2, 1],
[2, 1], [2, 1], [2, 1]]
```

Q5- On souhaite mettre en place un dictionnaire pour représenter l'ensemble des situations de jeu possibles à partir d'une situation initiale x0.
Mais les dictionnaires ne peuvent pas avoir pour clé des listes (car mutables) mais peuvent avoir pour clé des tuples. Mais, ces tuples ne peuvent eux-mêmes pas contenir de listes.
Jusqu'à présent nous avons travaillé avec des listes de listes.
Nous allons donc créer une fonction permettant de transformer ces listes de listes de listes …en tuples de tuples de tuples …que nous désignerons comme **T**uple avec une majuscule. Nous allons donc créer une fonction Tuple transformant une liste de listes de listes etc. en tuples de tuples de tuples etc.
La fonction tuple(…) définie en Python permet de transformer une liste simple en tuple.
Cf https://waytolearnx.com/2019/04/convertir-une-liste-en-tuple-python.html pour un exemple d'utilisation. Attention à ne pas remplacer cette fonction tuple de python (sans majuscule), qui sera utile ici.

Créer **une fonction récursive Tuple(L)** réalisant cette transformation de liste en Tuple.
Exemple d'exécution

```
L = [ [ [1,2],[3,4,5] , [[6]]] ]
lesTuples = Tuple( L )
print(lesTuples)  # affiche (((1, 2), (3, 4, 5), ((6,),)),)
```

Remarque : Il est normal que la transformation d'une liste de listes en tuples fasse apparaître une virgule.

```
L1 = [ [1] ]
lesTuples2 = Tuple( L1 )
print(lesTuples2) # affiche  ((1,),)
```

Q6- Pour constituer le dictionnaire représentant le graphe du jeu, on réalise un parcours en largeur. Pour cela, on utilisera une collection deque afin d'améliorer la complexité en temps de ce parcours.
Le dictionnaire comportera :

- Pour clés : Une situation du jeu x transformée en Tuple
- Pour valeurs : Une liste de listes LX de toutes les situations atteignables X à partir de x

Créer la **fonction graphe(C , N)** réalisant le parcours en largeur des possibilités.

Pour rappel sur le parcours en largeur :
https://fr.wikipedia.org/wiki/Algorithme_de_parcours_en_largeur

Intelligence artificielle
Théorie des jeux
Jeu des tablettes

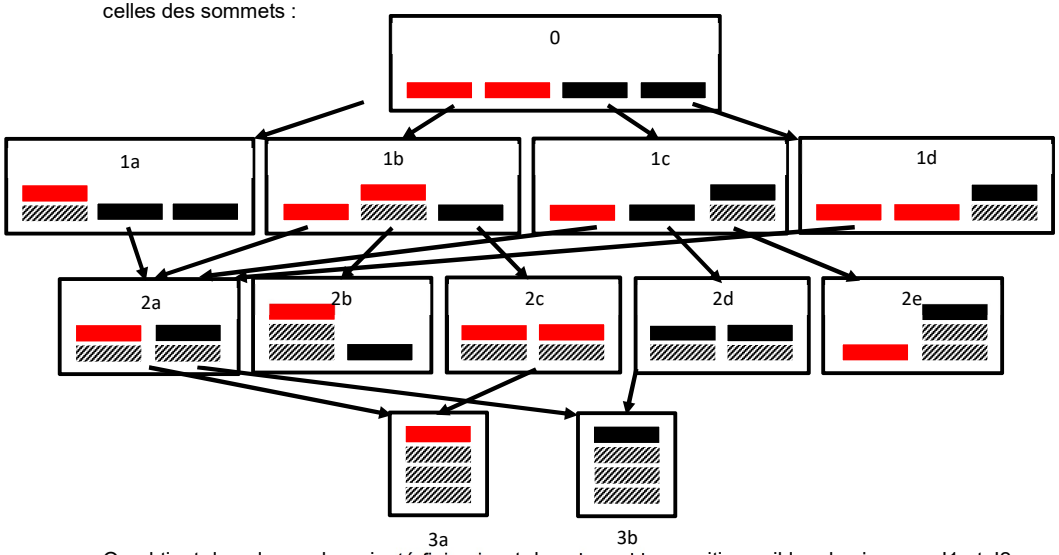
Exemple d'exécution :

```
C,N = 2,2
Graphe = graphe(C,N)
print(Graphe)
# affiche (avec mise en forme)
{((0, 1), (0, 1), (1, 1), (1, 1)): [[[0, 2], [1, 1], [1, 1]],
                                    [[0, 1], [0, 2], [1, 1]],
                                    [[0, 1], [1, 1], [1, 2]],
                                    [[0, 1], [0, 1], [1, 2]]],
 ((0, 2), (1, 1), (1, 1))        : [[[0, 2], [1, 2]]],
 ((0, 1), (0, 2), (1, 1))        : [[[0, 3], [1, 1]],
                                    [[0, 2], [0, 2]],
                                    [[0, 2], [1, 2]]],
 ((0, 1), (1, 1), (1, 2))        : [[[0, 2], [1, 2]],
                                    [[1, 2], [1, 2]],
                                    [[0, 1], [1, 3]]],
 ((0, 1), (0, 1), (1, 2))        : [[[0, 2],
                                    [1, 2]]],
 ((0, 2), (1, 2))                : [[[0, 4]],
                                   [[1, 4]]],
 ((0, 3), (1, 1))                : [],
 ((0, 2), (0, 2))                : [[[0, 4]]],
 ((1, 2), (1, 2))                : [[[1, 4]]],
 ((0, 1), (1, 3))                :  [],
 ((0, 4),): [],
 ((1, 4),): []
 }
```

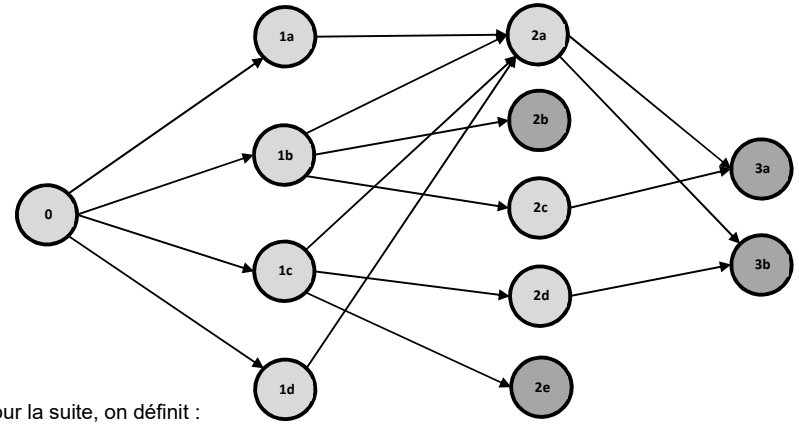
Le dictionnaire obtenu représente le graphe orienté fini G=(S,A) dans lequel certains sommets (clés) sont contrôlés par le joueur J1 (S1) et d'autres par le joueur J2 (S2). L'ensemble (G , S1 , S2) est appelé **arène**.
Chaque **sommet** représente une configuration/**situation/position** du jeu.
On appelle **arc/arête** a=(si,sj) ∈ A la possibilité pour un joueur Ji de passer du sommet si ∈ Si au sommet sj ∈ Sj en un coup, c'est une **décision**. Les arcs relient uniquement des sommets entre S1 et S2.
On définit les ensembles F1 et F2 avec F1∩F2=Ø, les objectifs des deux joueurs (**sommets gagnants/états finaux/terminaux** pour J1 et J2).

On appelle alors **jeu d'accessibilité** l'ensemble de l'arène et de F1 et F2 ((G, S1, S2), F1, F2).

On peut représenter le graphe pour C=2 et N=2, en ne tenant pas compte des couleurs autres que celles des sommets :



On obtient donc le graphe orienté fini suivant dans lequel les positions cibles des joueurs J1 et J2 (positions sans successeurs) sont représentées en foncé :



Pour la suite, on définit :

Pos_0 = [[0,1],[0,1],[1,1],[1,1]]	Pos_2b = [[0,3],[1,1]]
Pos_1a = [[0,2],[1,1],[1,1]]	Pos_2c = [[0,2],[0,2]]
Pos_1b = [[0,1],[0,2],[1,1]]	Pos_2d = [[1,2],[1,2]]
Pos_1c = [[0,1],[1,1],[1,2]]	Pos_2e = [[0,1],[1,3]]
Pos_1d = [[0,1],[0,1],[1,2]]	Pos_3a = [[0,4]]
Pos_2a = [[0,2],[1,2]]	Pos_3b = [[1,4]]

Intelligence artificielle
Théorie des jeux
Jeu des tablettes

Soient les instructions suivantes :

```
N1 = len(Graphe)
N2 = sum([len(Graphe[x]) for x in Graphe])
```

Q7- Que représentent **N1 et N2** ?
Vérifier les prédictions sur le graphe généré précédemment.

On propose le tableau suivant :

	N =	1	2	3	4
C = 1	Sommets				
	Arêtes				
	Temps (s)				
C = 2	Sommets				
	Arêtes				
	Temps (s)				
C = 3	Sommets				4220
	Arêtes				23 487
	Temps (s)				6560
C = 4	Sommets				
	Arêtes				
	Temps (s)				

Q8- Utiliser le programme mis en place afin de compléter les cases non grisées

Remarques :

- Le temps indiqué est relatif à l'exécution sur l'ordinateur de test. Vous n'obtiendrez probablement pas la même valeur.
- Le calcul pour C=3 et N=4 peut ne pas passer suivant le processeur
- Les cases grisées sont des cases pour lesquels l'ordinateur sur lequel le programme a été testé n'a pas été capable de trouver le graphe par manque de mémoire RAM (et après une longue attente)