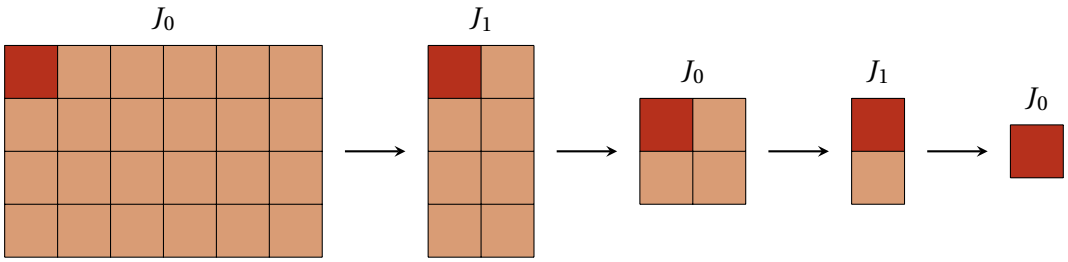
# TP 4 INFO — Théorie des jeux combinatoires

## Partie I — Étude d'un jeu

Deux joueurs $J_0$ et $J_1$ jouent avec une tablette en chocolat composée de $p$ lignes et $q$ colonnes de carreaux. Chaque joueur choisit à tour de rôle soit de manger une partie des dernières lignes en bas de la tablette, soit de manger une partie des dernières colonnes à droite de la tablette. Le carreau du coin supérieur gauche est empoisonné : le joueur qui se trouve dans l'obligation de le manger perd la partie. Le joueur $J_0$ est le premier à jouer.

**Exemple 1 :** Voici une partie avec une tablette de taille $(p,q) = (4,6)$ dans laquelle le joueur $J_1$ gagne.




### I.A - Étude d'un cas particulier

Dans cette sous-partie, on suppose que $(p,q) = (2,3)$.

**Question 1 :** Représenter le graphe biparti associé à ce jeu.

**Question 2 :** Déterminer les positions gagnantes pour le joueur $J_0$.

**Question 3 :** Le joueur $J_0$ admet-il une stratégie gagnante ? Si oui, déterminer en une.

### I.B - Étude du cas général

Pour représenter les graphes en Python, on utilise un dictionnaire dont les clés sont les sommets du graphe et les valeurs associées sont les listes des successeurs. On représente chaque sommet du graphe biparti $G = (S, A)$ du jeu par un triplet $(a,b,k) \in [\![1,p]\!] \times [\![1,q]\!] \times \{0,1\}$ où le couple $(a,b)$ est la taille de la tablette restante et $k$ est l'indice du joueur contrôlant cet état du jeu. La fonction ci-dessous permet de générer le graphe biparti du jeu.

```python
def gen_graphe_biparti(p:int, q:int) -> dict:
    dico = {}
    for a in range(1,p+1):
        for b in range(1,q+1):
            for k in range(2):
                dico[(a,b,k)] = []
                for i in range(1,a):
                    dico[(a,b,k)].append((i,b,1-k))
                for j in range(1,b):
                    dico[(a,b,k)].append((a,j,1-k))
    return(dico)
```

Par exemple, pour représenter le graphe de la partie précédente, on peut utiliser la commande suivante.

```
G = gen_graphe_biparti(2, 3)
```

Le graphe transposé $G^{\mathrm{T}}$ d'un graphe $G = (S, A)$ est obtenu en conservant tous les sommets de $S$ et en inversant le sens des arêtes de $A$.

**Question 4 :** Écrire une fonction `transpose(G)` prenant en argument un graphe $G$ et renvoie le graphe transposé de $G$.

Dans la suite, on considérera le graphe transposé $H$ du graphe biparti $G = (S, A)$ associé au jeu.

```
H = transpose(G)
```

Dans la suite, pour représenter un ensemble $E$, on utilise un dictionnaire dont les clés sont les éléments de l'ensemble $E$ et les valeurs sont vides. Par exemple, l'ensemble $F_0$ (respectivement $F_1$) des sommets finaux correspondant à une victoire de $J_0$ (respectivement $J_1$) pour ce jeu se représente de la manière suivante.

```
F0 = {(1,1,1) : None}
F1 = {(1,1,0) : None}
```

**Question 5 :** Construire l'ensemble $S_0$ (respectivement $S_1$) des sommets représentant les états du jeu contrôlés par le joueur $J_0$ (respectivement $J_1$).

Dans les questions suivantes, nous allons rédiger des fonctions en Python permettant de calculer l'attracteur de chaque joueur.

**Question 6 :** Écrire une fonction `foncttion_attracteur(E:dict, k:int) -> dict` prenant en argument une partie $E$ de $S$ et un entier $k \in \{0, 1\}$ et qui renvoie la partie $E'$ de $S$ définie par :

$$E' = E \cup \left\{ s \in S_k \;\middle|\; \exists s' \in E,\ (s, s') \in A \right\} \cup \left\{ s \in S_{1-k} \;\middle|\; \forall s' \in S_k,\ (s, s') \in A \Rightarrow s' \in E \right\}.$$

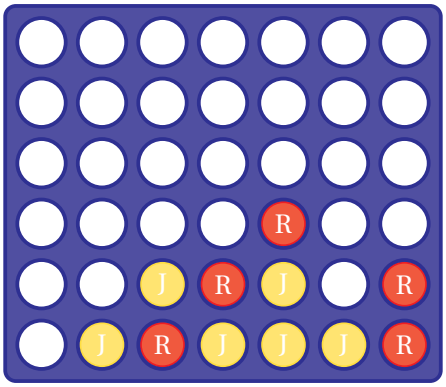
**Question 7 :** Écrire une fonction `attracteur(k:int) -> dict` prenant en argument un entier $k \in \{0, 1\}$ et qui renvoie l'attracteur du joueur $J_k$.

**Question 8 :** En utilisant votre fonction précédente, conjecturer l'ensemble des positions gagnantes dans le cas général pour les deux joueurs, puis proposer une stratégie gagnante pour le joueur en admettant une.

# Partie II Algorithme min-max

Dans cette partie, on s'intéresse au jeu Puissance 4. Le but du jeu est d'aligner une suite de 4 pions de même couleur dans une grille rectangulaire composée de 6 lignes et 7 colonnes. Chaque joueur dispose de 21 pions d'une même couleur. Les deux joueurs placent tour à tour un pion dans la colonne de leur choix, le pion coulisse alors jusqu'à la position la plus basse possible dans ladite colonne, à la suite de quoi c'est à l'adversaire de jouer. Le vainqueur est le joueur qui réalise en premier un alignement (horizontal, vertical ou diagonal) consécutif d'au moins 4 pions de sa couleur. Si toutes les cases de la grille de jeu sont remplies et qu'aucun des deux joueurs n'a réalisé un tel alignement, la partie est déclarée nulle.

Dans la suite, on suppose que $J_0$ joue avec des pions jaunes et que $J_1$ joue avec des pions rouges.




Une position du jeu Puissance 4

Pour jouer une partie contre l'ordinateur, vous pouvez utiliser les instructions suivantes.

```
from module_puissance4 import *
partie()
```

**Question 9 :** Gagner une partie contre l'ordinateur.

Normalement, c'était très facile : l'ordinateur joue aléatoirement dans une des colonnes possibles. Nous allons utiliser l'algorithme min-max pour améliorer la stratégie suivie par l'ordinateur.

On commence par construire une heuristique pour ce jeu. Pour ce faire, on attribue à chaque case du jeu le nombre d'alignements possibles de quatre jetons contenant cette case. Les valeurs attribuées sont reportées dans le tableau ci-dessous.

| | | | | | | |
|---|---|---|---|---|---|---|
| 3 | 4 | 5 | 7 | 5 | 4 | 3 |
| 4 | 6 | 8 | 10 | 8 | 6 | 4 |
| 5 | 8 | 11 | 13 | 11 | 8 | 5 |
| 5 | 8 | 11 | 13 | 11 | 8 | 5 |
| 4 | 6 | 8 | 10 | 8 | 6 | 4 |
| 3 | 4 | 5 | 7 | 5 | 4 | 3 |

Tableau des valeurs attribuées à chaque case

Ensuite, pour calculer l'heuristique d'une position (non finale) du jeu, on calcule somme des valeurs des cases contrôlées par $J_0$ à laquelle on retranche la somme des valeurs des cases contrôlées par $J_1$. Par exemple, l'heuristique de la position $s$ représentée au début de cette partie est

$$h(s) = (4+7+5+4+8+8) - (5+3+10+4+11) = 3.$$

**Question 10 :** Quel premier choix doit effectuer le joueur $J_0$ afin de maximiser l'heuristique après 1 coup ? après 2 coups ?

Pour représenter une position du jeu Puissance 4 en Python, on considère une liste de listes dont les éléments sont `"J"` pour case jaune, `"R"` pour case rouge ou `"O"` pour case ouverte. Par exemple, la position dessinée au début de cette partie est représentée par la liste ci-dessous.

```
grille = [
["O","O","O","O","O","O","O"],
["O","O","O","O","O","O","O"],
["O","O","O","O","O","O","O"],
["O","O","O","O","R","O","O"],
["O","O","J","R","J","O","R"],
["O","J","R","J","J","J","R"]]
```

Dans le fichier Python du TP, l'heuristique décrite ci-dessus est déjà définie sous le nom `h(grille)` . De même, la fonction `min_max(grille, joueur, profondeur)` permettant de mettre en place l'algorithme min-max est déjà rédigée.

**Question 11 :** Dans votre fichier Python du TP, étudier le fonctionnement puis commenter les lignes du code de la fonction `min_max(grille, joueur, profondeur)`.

Pour terminer, on considère une constante indiquant la profondeur d'exploration de l'arbre que devra effectuer le joueur $J_1$ avec l'algorithme min-max pour choisir son prochain coup.

```
PROFONDEUR_EXPLORATION = 4
```

**Question 12 :** Écrire une fonction `IA_min_max(grille) -> int` prenant en entrée l'état du jeu représenté par `grille` et qui renvoie l'indice de la colonne que devra jouer l'ordinateur (i.e. le joueur $J_1$) en suivant l'algorithme min-max avec la profondeur `PROFONDEUR_EXPLORATION`.

Vous pouvez jouer contre l'ordinateur avec cette nouvelle I.A. en utilisant la commande suivante.

```
partie(IA_min_max)
```

**Remarque 1 :** On peut augmenter la valeur de `PROFONDEUR` pour améliorer le niveau de l'ordinateur.