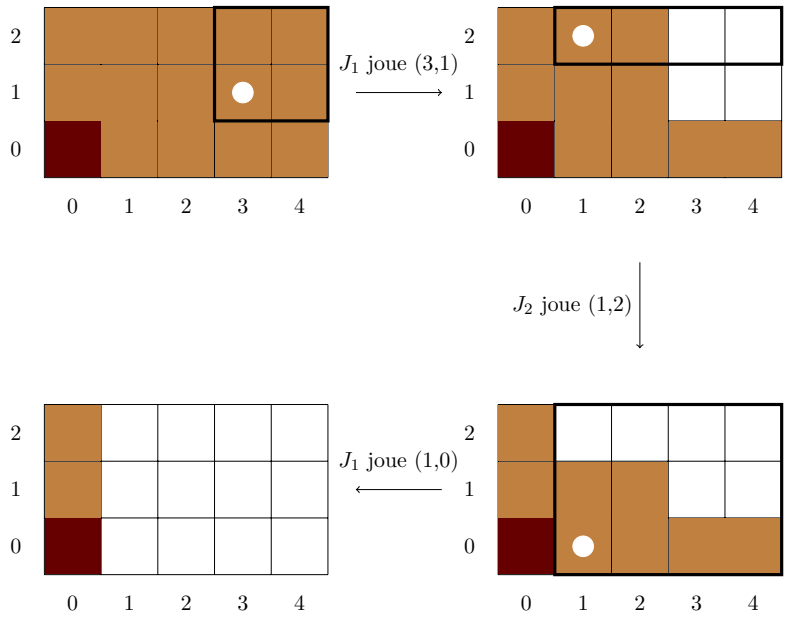
TP : Le jeu de Chomp

Nous suivons en grande partie le TP de Mickaël Péchaud, dont l'original est à l'adresse : http://mpechaud.fr
Nous allons jouer à un jeu à 2 joueurs, sans match nul, suffisamment simple pour que la notion de stratégie gagnante apparaisse naturellement.

I) Règle du jeu

Le jeu du chocolat empoisonné (ou *chomp* en anglais) se joue à deux joueur·se·s, noté·e·s dans la suite J_1 et J_2.
On dispose d'une tablette de chocolat, et l'on suppose que le carré en bas à gauche est empoisonné.
À tour de rôle, chaque joueur·se, en commençant par J_1, doit manger l'un des carrés (repéré par ses coordonnées), ainsi que tous ceux qui se trouvent au dessus et/ou à droite. La personne qui mange le carré empoisonné meurt et a donc perdu. L'autre joueur·se, qui est resté en vie, a gagné. On ne peut pas passer son tour.
Le schéma ci-dessous montre le début d'une partie qui démarre d'une tablette 5×3.



Une « position » est un état de la tablette : nous avons 4 positions successives ci-dessus.

II) Pratique

Exercice 1

1) Dans l'exemple précédent, à partir de la dernière position, J_2 peut-il jouer un coup qui lui permet de gagner à coup sûr ?

2) Formez des binômes, découpez des carrés, et jouez quelques parties avec différentes tailles de grilles, 2×3, 4×3, 4×4, pour vous familiariser avec le jeu.

3) Désormais, on s'intéresse au caractère gagnant ou perdant d'une position donnée. La définition rigoureuse est ci-dessous.

Que pensez-vous des positions suivantes :

a) Ligne $n \times 1$ ou $1 \times n$ (où $n \geqslant 1$) ?

b) Position en « L » avec les deux branches de même longueur

c) Position en « L » avec les deux branches de longueurs différentes

d) Tablette carrée 

Définition 1
Les définitions de « position gagnante » et « position perdante » sont les suivantes :

- Une position est *gagnante* si la personne qui joue à partir de cette position peut parvenir à gagner en fin de partie quels que soient les coups de son adversaire : si c'est à J_1 de jouer, il existe un coup pour J_1, tel que, quel que soit le coup suivant de J_2, il existe un coup pour J_1 etc.. jusqu'au moment où J_2 est obligé de manger le carré empoisonné.
- Une position est *perdante* si la personne qui joue à partir de cette position ne peut que perdre en fin de partie quel que soit les coups qu'elle joue : si c'est à J_1 de jouer, quel que soit le coup de J_1, il existe un coup suivant de J_2 tel que, quel que soit le coup de J_1 etc.. jusqu'au moment où J_1 est obligé de manger le carré empoisonné.

Un définition récursive, équivalente, et plus rigoureuse (sans « etc ») :

- La position où ne reste que le carré empoisonné est perdante.
- Pour une position P donnée,
  - s'il existe un coup jouable permettant de passer de P à une position perdante, alors P est gagnante,
  - si tout coup jouable à partir de P amène dans une position gagnante, alors P est perdante.

III) Formalisation du jeu

Exercice 2
On considère une tablette $n \times m$, $n, m \in \mathbb{N}^2$. Justifiez qu'une partie se termine toujours, et qu'il y a à la fin un·e gagnant·e. En combien de coup maximum une partie se termine-t-elle ?

Exercice 3 (graphe associés à un jeu)
Nous allons construire un graphe représentant les positions et les coups possibles. Ce graphe s'appelle l'*arène* associée au jeu :

- Les sommets $s \in \mathcal{S}$ du graphe sont les positions possibles.
- Le couple $(s_1, s_2) \in \mathcal{S}^2$ est une arête s'il existe un coup permettant de passer de s_1 à s_2.

Ainsi, l'illustration au début du TP est un morceau du graphe cherché : les sommets sont des positions, les flèches entre les positions représentent des arêtes (= les coups). Mais ce n'est qu'un morceau : il manque des sommets (toutes les positions possibles ne sont pas représentées) et il manque donc des arêtes – mais il manque même des arêtes entre les sommets présents : par exemple, il existe une arête entre la première et la dernière position.

On considère désormais une petite tablette, 2×3. On notera $\mathcal{S}$ l'ensemble des sommets, et $\mathcal{A}$ l'ensemble des arêtes.

1) Tracer l'arène associée au jeu de Chomp(2,3).

2) Est-ce qu'il y a un ou des sommets dont ne part aucune arête ? Que représente ce sommet ?
Un tel sommet est appelé sommet sans successeur.

3) Le graphe peut-il avoir des cycles ?
Rappel : un cycle est une suite de sommets $(s_0, \dots, s_n)$ tels que

- $s_0 = s_n$
- $\forall i \in [\![0, n-1]\!]$, $(s_i, s_{i+1}) \in \mathcal{A}$.

Indication : *Associer une valeur entière $n(s)$ à chaque sommet $s \in \mathcal{S}$, de façon simple. Déterminer une propriété du couple $(n(s_1), n(s_2))$ pour toute arête $(s_1, s_2) \in \mathcal{A}$.*

4) Montrer qu'un graphe sans cycle [1] possède forcément des sommets sans successeur.

1. Un tel graphe est dit *acyclique*.

Exercice 4 (graphe bipartie associés à un jeu)
Désormais, on veut distinguer les positions où c'est à J_1 de jouer, de celles de où c'est à J_2.

1) Tracer l'arène associée au jeu de Chomp(2,3), en remplaçant, pour les sommets, les dessins de positions par des numéros (ici, $[\![0, 8]\!]$).

On numérotera par 0 la position de départ, et 8 la position sans successeur. On gardera une table de correspondance numéro/dessin.

Graphe biparti associé au jeu : Le nouveau graphe associé au jeu aura des sommets contenant l'information de la position (donc le numéro $n \in [\![0, 8]\!]$) et du joueur qui contrôle la position : le sommet s'appellera n_a ou n_b, selon que c'est à J_1 ou J_2 de jouer.
Exemples : La position de départ 0 ne peut pas être contrôlée par J_2, donc seul le sommet 0_a existe. Comme J_1 et J_2 peuvent gagner, 8_a et 8_b existent.
On note $\mathcal{S}_a$ les sommets d'indices a, contrôlés par J_1, et $\mathcal{S}_b$ ceux d'indice b, contrôlés par J_2.
Remarquons que :

- $\mathcal{S} = \mathcal{S}_a \cup \mathcal{S}_b$ et $\mathcal{S}_a \cap \mathcal{S}_b = \emptyset$.
- Il n'y a aucune arête entre deux sommets de $\mathcal{S}_a$, ou entre deux sommets de $\mathcal{S}_b$.

Les deux points ci-dessus sont la définition d'un graphe « biparti ».

2) Tracer le graphe biparti associé à Chomp(2,3). Traditionnellement, on note les sommets sur 2 colonnes, les sommets de $\mathcal{S}_a$ puis ceux de $\mathcal{S}_b$.

IV) Python

Nous allons commencer à écrire des fonctions permettant de tester si une position est gagnante ou perdante. Nous choisissons pour ce faire de coder une position sous forme d'une liste d'entiers contenant le nombre de carrés de chocolats dans chaque ligne, les lignes étant parcourues de bas en haut. Ainsi, les quatre positions successives données sur la première page correspondent aux listes suivantes :

- `[5, 5, 5]`
- `[5, 3, 3]`
- `[5, 3, 1]`
- `[1, 1, 1]`

Par convention, une telle liste ne contiendra jamais de 0. On écrira donc `[2, 1]` au lieu de `[2, 1, 0, 0]`.
Dans la suite de l'énoncé, le terme *position* désignera une telle liste.
Par ailleurs, un *coup* sera codé par le couple de coordonnées correspondant au carré choisi.

1) Compléter la fonction `tablette(m, n)` qui prend en argument deux entiers non-nuls, et renvoie la position correspondant à une tablette à m colonnes et n lignes.

2) Tester la fonction `affiche(pos)`, qui prend en argument une position, et l'affiche à l'aide de chaînes de caractères – □ désignant un carré commestible, et ■ le carré empoisonné.

3) Compléter la fonction `coups_jouables(pos)`, qui prend en argument une position et renvoie la liste des coups jouables – i.e. la liste des coordonnées de tous les carrés de la position. Par convention, $(0, 0)$ n'est pas un coup jouable.

4) Compléter la fonction `enleve_0(l)`, qui prend en argument une liste `l` et la modifie par effet de bord en enlevant tous les éventuels 0 qu'elle contient à sa fin.

5) Compléter la fonction `joue(pos, coup)`, qui prend en argument une position de départ et un coup jouable pour cette position, et renvoie une nouvelle position correspondant à la position obtenue en jouant le coup à partir de la position de départ. Il est conseillé avant de se lancer dans l'écriture du code d'effectuer quelques essais à la main.
Quelle est la complexité de votre fonction ?

6) En utilisant la définition récursive de « position gagnante », compléter la fonction `gagnante(pos)` qui prend en argument une position, et teste si elle est gagnante (elle renverra un booléen). Penser à la tester sur quelques exemples.

7) **a)** Justifier que votre fonction termine.

b) Majorer (très grossièrement) le nombre de positions accessibles dans une partie démarrant depuis une tablette $n \times m$.[2]

c) Afin de compter le nombre d'appels récursifs effectués par votre fonction, ajouter au début de celle-ci les lignes suivantes, qui permettent d'incrémenter un compteur global, défini à l'extérieur de la fonction :

```
global compteur
compteur += 1
```

Tester alors par exemple avec

```
compteur = 0
print(gagnante(tablette(4, 5)))
print(compteur)
```

Comparer le résultat obtenu avec celui de la question précédente, et commenter.

Afin d'éviter les calculs redondants, on souhaite mémoriser le caractère gagnant ou perdant des positions déjà rencontrées afin de ne pas avoir à le recalculer (mémoïsation).
Pour cela, on va utiliser un *dictionnaire*

- dont les clés correspondent aux positions
- dont les valeurs sont des booléens indiquant si la position est gagnante ou perdante.

Il est impossible d'utiliser une liste `pos` comme clé d'un dictionnaire, donc on utilisera à la place un tuple ou une chaîne, que vous pourrez obtenir à l'aide des fonctions `tuple(pos)` ou `str(pos)`.
Ainsi, les entrées du dictionnaire ressembleront à

```
{
'2' : True,
'2-1' : False,
'2-2' : True,
...
}
```

8) Compléter la fonction `gagnante_memo(pos, dict)`, qui prend en argument une position et un dictionnaire, en vous inspirant de la fonction `gagnante`. Cependant, la nouvelle fonction devra commencer par tester si la position apparaît déjà dans le dictionnaire, auquel cas il suffira de renvoyer la valeur correspondante. Si ça n'est pas le cas, la position et sa valeur devront être rajoutées au dictionnaire avant de renvoyer le résultat. `dict` sera donc modifié par effet de bord.

9) Testez votre fonction, et comparez empiriquement son temps d'exécution à celui de `gagnante`. Quelle est sa complexité ?

2. Calculer exactement ce nombre est un bon exercice de dénombrement.