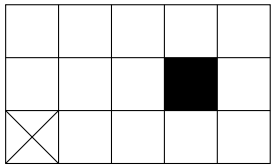
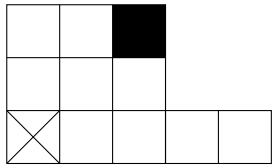
Un jeu d'accessibilité : la tablette de chocolat [1]

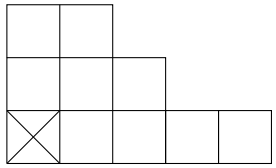
Imaginons une tablette de chocolat avec `n` lignes et `p` colonnes . On suppose que le carré situé tout en bas et tout à gauche est empoisonné. Deux joueurs, Honoré et Diégo, jouent à tour de rôle. À chaque coup, le joueur choisit l'un des carrés de la plaque et mange ce carré ainsi que ceux situé en haut et à droite de ce carré. Le jeu continue tant qu'il reste des carrés à manger. Celui qui a mange le carré empoisonné perd.



(a) Une tablette avec 3 lignes et 5 colonnes. Honoré choisit la case 2-ième ligne 4-ième colonne



(b) Diégo choisit la case 1ère ligne, 3ième colonne



(c) La partie continue tant qu'il reste des carrés à manger.

1. Justifier qu'il s'agit bien d'un jeu d'accessibilité.
2. Téléchargez le fichier `TP5.py` qui contient des fonctions codant ce jeu, et jouer une partie avec votre voisin (en actualisant la liste `L` avec vos prénoms). Attention, cela marchera sous Spyder mais pas sous Pyzo.

Ensembles en Python :

- Les ensembles en Python se comporte comme ceux en maths : pas de répétition, l'ordre ne compte pas.
- Pour définir l'ensemble vide on écrit [2] `E=set()`,
- Pour écrire un ensemble par extension, on fait `E=`$\{1,2,3\}$
- Par compréhension : `E=`$\{$`k*k for k in range(10) if k!=8`$\}$.
- À un ensemble `E`, on rajoute un élément par `E.add(8)`.
- Si `A` et `B` sont deux ensembles `A-B` désigne l'ensemble $A\backslash B$.
- On peut boucler sur un ensemble comme sur une liste `for e in E:`.
- $\{1,1,3,1,5\} == \{1,3,5\}$ est un booléen qui vaut `True`. En revanche, $[1,1,3,1,5] == [1,3,5]$ est un booléen qui vaut `False`.
- `len(E)` renvoie le nombre d'éléments de l'ensemble `E`.

3. Lisez rapidement les fonctions du fichier et comprenez ce qu'elles font.
4. Rejouer une partie avec une seule ligne ou une seule colonne mais avec au moins deux carrés de chocolat, celui qui commence doit pouvoir gagner automatiquement.
5. Coder une fonction heuristique `H(tab)` de la façon suivante : si la tablette ne contient que le carré `(0,0)` renvoyer -1 (synonyme de défaite), si la tablette ne contient qu'une seule ligne ou une seule colonne renvoyer 1 (synonyme de victoire) et 0 dans tous les autres cas (synonyme qu'on ne sait pas trop où on va).
6. Écrire l'algorithme Minimax avec cette heuristique (s'aider du cours).
7. Utiliser l'algorithme Minimax3 et Minimax7 pour les comparer avec la stratégie hasard.

On souhaite maintenant utiliser l'algorithme de l'attracteur pour construire une autre stratégie. On va construire le graphe du jeu où les sommets seront de la forme `(tab,j)` ou `tab` est l'état de la tablette et `j` désigne le joueur qui doit jouer.

8. Créer une fonction `ListeVoisins(s)` qui renvoie la liste des voisins d'un sommet `s`.
9. Créer une fonction récursive qui va créer le graphe par un parcours en profondeur (adapter l'algorithme du cours).

> **Attention les ensembles ne sont pas hashables**
> Ici, comme une position contient un ensemble de carrés de chocolat, on ne peut pas être une clé d'un dictionnaire. Car un ensemble comme une liste ne sont pas hasables. À la place, on va juste créer la liste des sommets et on va noter `G` cette liste. Si on a besoin de la liste des sommets de `s` on utilisera la fonction `ListeVoisins`.

10. Créer la fonction qui calcule l'attracteur (adapter l'algorithme du cours).
11. Écrire la fonction `StrategieA` qui va utiliser le calcul de l'attracteur de la façon suivante : si la position n'est pas dans l'attracteur renvoyer un coup au hasard, sinon renvoyer un coup qui reste encore dans l'attracteur.
12. Comparer cette stratégie aux autres.

1. N'en n'abusez pas à Noël !
2. Et non `E={}` qui définit un dictionnaire vide.