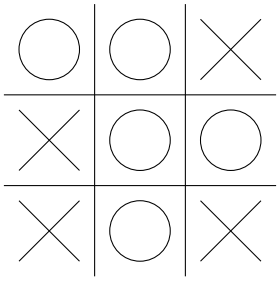
TP8 : Théorie des jeux, apprentissage

1 Jeu du morpion

Le but du TP est d'étudier le jeu de morpion dans le cadre du cours.

On présente rapidement ce jeu très classique. Deux joueurs jouent à tour de rôle, chacun avec son symbole `X` ou `O` sur une grille de 9 cases (3×3). Le premier joueur ayant aligné 3 de ses symboles a gagné. Voici un exemple de situation :



On représente une situation de jeu par une chaîne de caractères contenant les lettres `X` ou `O` en majuscules ainsi que des espaces. La lecture de cette chaîne de caractères permet de reconstruire, ligne par ligne, le plateau de jeu. Dans l'exemple ci-dessus :

```
jeu =  'OOXXOOXOX'
```

1.1 Premiers outils

1. Proposer une fonction `plateau(jeu)` prenant en argument une situation de jeu et affichant le plateau avec une ligne `---` en premier. Par exemple :

   ```
   >>> plateau('OOXXOOXOX')
   ---
   OOX
   XOO
   XOX
   ```

   Vous pourrez à tout moment utiliser cette fonction pour représenter un état de jeu.

2. Proposer une fonction `est_plein(jeu)` renvoyant le booléen répondant à la question « Le plateau est plein ? ». Par exemple :

   ```
   >>> est_plein('XX0 0XXX0')
   False
   >>> est_plein('XXOOOXXX0')
   True
   ```

3. Proposer une fonction `nbr2str(j)` prenant en argument le numéro du joueur j et renvoyant sous forme de chaîne de caractères `'X'` pour le joueur 1 et `'O'` pour le joueur 2.
4. Proposer une fonction `str2nbr(ch)` réalisant le travail inverse où `ch` est la chaîne de caractère `'X'` ou `'O'`.
5. Proposer une fonction `est_gagnant(jeu)` renvoyant un tuple `(bool,j)` avec `bool` le booléen répondant à la question « Un joueur a gagné ? » et `j` le joueur concerné.
6. Proposer une fonction `est_fini(jeu)` prenant en argument une situation de jeu et renvoyant le booléen répondant à la question « Le jeu est fini ? ».

Dans la suite, une position du jeu sera représenté par un tuple contenant l'état du jeu, puis un entier entre 1 et 2 indiquant le numéro de joueur à qui c'est le tour de jouer :

```
('XX0   X O', 2)
```

1.2 Graphe du jeu

7. Créer la fonction `coups(pos)` prenant en argument une position (tuple) et renvoyant une liste de toutes les positions différentes atteignables depuis la position. Par exemple :

```
>>> coups((('XXO   X O', 2))
[('XXOO  X O', 1),('XXO O X O', 1),('XXO  OX O', 1),('XXO   XOO', 1)]
```

Pour réaliser le dictionnaire représentant le graphe du jeu, on réalise un parcours du graphe (au choix, en largeur ou en profondeur). Le dictionnaire contiendra :

— pour clés : une position ;

— pour valeurs : la liste des positions atteignables en une seule itération depuis la position.

8. Créer la fonction `graphe(x0)` réalisant le parcours du graphe depuis `x0` où `x0` est une position au sens défini précédemment.

La commande `graphe((' '*9,1))` permet de créer le graphe complet lorsque le joueur 1 commence.

1.3 Attracteurs

Dans toute la suite, on considère que le joueur 1 joue en premier. Le but est de créer l'attracteur du joueur 1. On note dans la suite `G` le graphe complet du jeu.

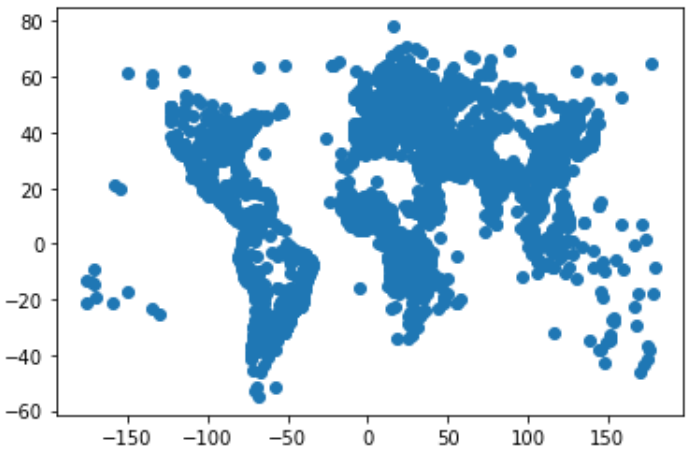
9. Créer la fonction `init_attracteur(G)` où `G` est le graphe complet et renvoyant la liste des états gagnants du joueur 1.
10. Créer la fonction `attracteur_1(G)` qui prend en argument le graphe complet `G` et qui renvoie l'attracteur du joueur 1. On pourra utiliser le cours et créer des fonctions intermédiaires similaires à `Existence` et `PourTout`.

2 Classification et continents

Le but de cette partie est d'utiliser l'algorithme des k-moyennes ainsi que le TP sur les bases de données pour effectuer une classification des grandes villes par continent.

2.1 Représentation graphique des villes

1. En reprenant le TP sur les bases de données données, créer une liste `liste_villes` qui contient les tuples de la forme `('Stanley', 'FALK', 'Falkland Islands', None, -51.42, -57.51, None)` par exemple.
2. Créer deux listes `longitude` et `latitude` qui contiennent respectivement les latitudes et longitudes des villes de la base de données.
3. Représenter, notamment à l'aide de `plt.plot(latitude, longitude,"o")` (ceci permet de tracer des points) les villes sur un graphique. On obtient le graphique suivant :



2.2 Clustering

On va désormais tester l'algorithme des k-moyennes pour regrouper les villes par continent.

4. Écrire une fonction `listes_egales(l1,l2)` prenant en argument deux listes `l1` et `l2` qui renvoie `True` si les deux listes sont identiques et `False` sinon. On considérera uniquement des listes de coordonnées de points du plan (c'est-à-dire des listes de listes à deux éléments).
5. Écrire une fonction `indice_plus_proche(X, liste)` prenant en argument une liste de coordonnées d'un point `X` et une liste de points `X` qui renvoie l'indice de l'élément de la liste le plus proche de `X`. On pourra simplement considérer la distance de la somme des carrées des différences latitude/longitude.
6. Créer les fonctions `mise_a_jour_centres(centres,liste)` et `kmoyennes(liste,k)` qui prennent en argument une liste de coordonnées de centres, une liste de coordonnées de point et un entier k et qui renvoie respectivement le calcul des nouveaux centres (les barycentres) et une liste de centres après l'application de l'algorithme des k-moyennes. Pour l'initialisation, on pourra utiliser un tirage aléatoire avec `sample(liste,k)` où la fonction `sample` sera chargée avec le module `random`.
7. En utilisant le code :

```
def dessine_donnees_avec_moyennnes(listeX, centres):
    formes=['k', 'g', 'b', 'deeppink', 'gold','chocolate','y','darksalmon']
    for X in listeX:
        classe_de_X=indice_plus_proche(X,centres)
        plt.plot([X[0]],[X[1]], marker = 'o', color = formes[classe_de_X])
    for k in range(len(centres)):
        plt.plot(centres[k][0],centres[k][1],'r^')
    plt.show()
```

représenter graphiquement la situation avec des entiers k entre 1 et 8. Qu'en conclure ?

On donne un exemple de résultat possible avec $k = 6$: