TP N°06 : Le Mancala.

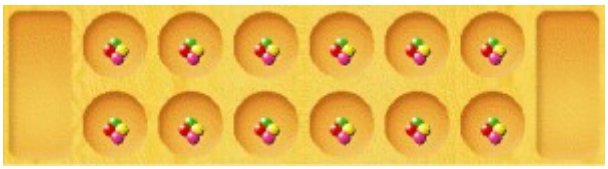
Le but de ce TP qui durera deux séances est d'implémenter un jeu à deux joueurs et de trouver une bonne stratégie pour ce jeu à l'aide de l'algorithme du minimax. Il est impossible de construire l'arbre complet pour ce jeu, cela nécessiterait beaucoup trop de sommets et un temps de calcul trop long. Il faudra se satisfaire d'un arbre sur les n prochains coups avec $n \approx 7$.

Descriptif du jeu

Le Mancala est un jeu dont les règles peuvent varier selon les usages, nous utiliserons, dans ce TP, les règles de la version du site BrainKing :

Position de départ et but du jeu

La partie se joue sur le plateau de 6 x 2 (avec deux greniers de chaque côté) et chaque trou contient 4 graines au départ. La figure suivante montre la position initiale :



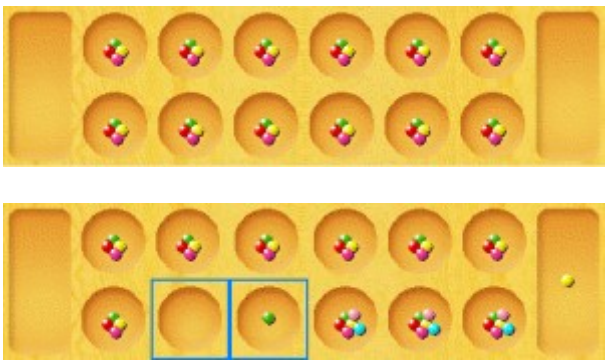
Le but du jeu est d'obtenir plus de points que son adversaire en déplaçant des graines de son camp ou en capturant celles de l'adversaire.

Placement des graines

Le joueur, à qui c'est le tour de jouer, choisit sur un trou (sur la rangée la plus proche de lui) qui contient au moins une graine. Cette action prendra toutes les graines du trou sélectionné et les placera une par une dans les trous suivants, dans le sens inverse des aiguilles d'une montre. Le grenier du joueur [1] (à droite du plateau de son point de vue) est utilisé aussi et lorsqu'une graine est placée dedans, le joueur obtient 1 point. La figure suivante montre un exemple du premier coup ; le joueur a retiré 4 graines du trou repéré, placé 1 graine dans les 4 trous suivants (incluant son grenier) et a obtenu 1 point :

Si la dernière graine (du coup en cours) est placé dans le grenier, le joueur continue de jouer et choisi un autre trou non-vide. La figure suivante montre ce coup :

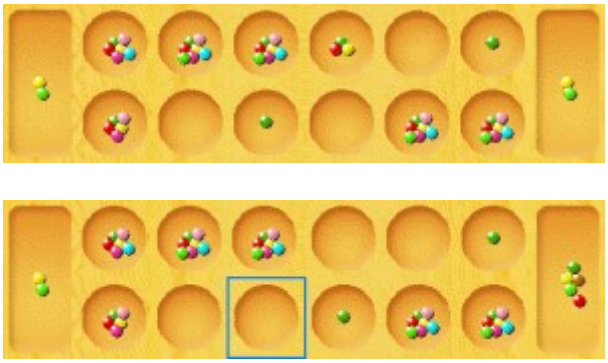
1. *à ne pas confondre avec le joueur du grenier*



la sélection du premier coup (celui qui contient la graine maintenant) a mis la dernière graine dans le grenier, aussi le joueur a vidé le second trou (celui qui est à présent vide). Si la dernière graine ne tombe pas dans son grenier, c'est à l'autre joueur de jouer.

Comment prendre des graines

Si la dernière graine (du coup en cours) est placé dans un trou vide (du côté du joueur), toutes les graines de la même colonne de la rangée opposée sont prises et placées dans le grenier du joueur. La figure suivante montre une prise (avant puis après) :



Fin de la partie

La partie se termine si un des joueurs n'a plus de coup légal, il n'y a plus de graine dans sa rangée. Lorsque cela se produit, toutes les graines qui reste dans les trous adverses sont ajoutées au score de l'adversaire. Le joueur avec le plus grand nombre de points gagne la partie.

Étude

Pour commencer, nous allons créer une étiquette pour représenter l'état de la partie. Par exemple, la partie :

sera représentée par la chaine de caractères : `"5,0,1,0,7,7,2,1,0,3,7,7,7,2"` (les valeurs sont obtenues en comptant le nombre de graine dans chaque trou, en partant du trou le plus en bas à gauche et en tournant dans le sens anti-horaire. De plus, on rajoutera quelle joueur contrôle la

partie (qui doit jouer) : `"0|5,0,1,0,7,7,2,1,0,3,7,7,7,2"` si c'est au joueur 0 (qui contrôle les trous du bas et le grenier droite) de jouer, sinon `"1|5,0,1,0,7,7,2,1,0,3,7,7,7,2"`. L'étiquette de l'image précédente est donc :

`"0|5,0,1,0,7,7,2,1,0,3,7,7,7,2"`

1. Écrire une fonction `etiquette_en_jeu(etiquette)` qui prend une etiquette (chaine de caractères) comme sus-mentionné et qui renvoie un couple `(joueur,jeu)` tel que `joueur` soit une chaine de caractères qui représente le joueur qui contrôle (`'0'` ou `'1'`) et `jeu` soit une liste d'entier donnant le nombre de graines dans le même ordre que étiquette. Pour l'exemple précédent, `joueur='0'` et `jeu=[5,0,1,0,7,7,2,1,0,3,7,7,7,2]`. Par

2. Écrire une fonction `jeu_en_etiquette(couple)` qui effectue l'opération inverse avec `couple=(joueur,jeu)`.

3. Écrire une fonction `terminaux(etiquette)` qui prend en argument une étiquette et renvoie `True` si la partie est terminée (si le sommet est un sommet terminal), `False` sinon.

Pour afficher une partie, on pourra utiliser la fonction `affichage(etiquette)` qui permet d'obtenir l'affichage suivant :

```
>>> affichage('0|4,4,4,4,3,4,1,4,4,2,4,4,4,2')
   |4|4|4|2|4|4|
2  -------------  1
   |4|4|4|4|3|4|
```

4. Écrire une fonction récursive ou itérative `graphemancala(n,etiquette='0|4,4,4,4,4,4,0,4,4,4,4,4,4,0',S=[],A=[],E=[],i=0)` qui prend en argument un entier `n` correspondant aux nombres de coups qu'on veut modéliser dans le graphe en partant de l'état `etiquette`, un triplet `(S,A,E)` correspondant à un graphe étiqueté et un entier i (correspondant aux nombres de coup déjà réalisés lors des appels récursifs) et qui renvoie un triplet (S,A,E) correspondant à un graphe étiqueté du jeu en partant de l'état d'étiquette sur n coups. Je vous conseille de démarrer par la version itérative, qui est à mon avis moins compliqué.

On pourra s'aider du pseudo-code suivant : **version récursive, la version itérative suit**

- Si i=0, on initialise (S,A,E) à ([0],[],[etiquette]).
- Si i=n, on renvoie (S,A,E) (*on arrive à une feuille de l'arbre car on a déjà joué i=n coups*)
- Si l'état associé à etiquette n'est pas terminal,
  - Si c'est au joueur 0 de jouer :
    - Pour chaque coup possible pour le joueur (attention de vérifier qu'il reste des graines/billes dans le trou) :
      - On crée une variable jeu_new correspondant au nouvel état de la table après le coup joué ;
      - On crée une variable joueur_new correspondant au joueur qui doit jouer après le coup joué ;
      - On crée une variable etiquette_new à l'aide des deux variables précédentes et correspondant à la nouvelle étiquette après le coup ;

        - On demande récursivement de créer l'arbre partant du sommet lié à etiquette_new avec i+1 coups joués ;
        - On ajoute le sous-arbre crée à l'arbre (S,A,E) au niveau du sommet correspondant à etiquette. Pour cela, on pourra créer une sous-fonction qui ajoute un sous-arbre à un arbre au sommet choisi.
    - Si c'est au joueur 1 de jouer
        - On effectue les mêmes opérations (*démerdez-vous*) .

ou du pseudo-code suivant : **version itérative**

- On initialise (S,A,E) à ([0],[],[etiquette]).
- On initialise une liste d'attente à [0], elle contiendra les numéros des sommets qui devront être traités.
- On initialise l'indice lié à cette file à 0, c'est l'indice de l'élément de la liste d'attente en cours de traitement.
- on initialise une liste indiquant la profondeur (le nombre de coups joués) de chaque sommet à [0] (le sommet numéro 0 est de profondeur 0).
- Tant que l'indice ne dépasse pas la taille de la file d'attente
    - On récupère le numéro du sommet.
    - On récupère sa profondeur.
    - On récupère son étiquette et les données liées à cette étiquette.
    - Si le sommet n'est pas terminaux et pas trop profond (pas trop de coups joués) :
        - Si c'est au joueur 0 de jouer :
            - Pour chaque coup possible pour le joueur (attention de vérifier qu'il reste des graines/billes dans le trou) :
            - On crée une variable jeu_new correspondant au nouvel état de la table après le coup joué ;
            - On crée une variable joueur_new correspondant au joueur qui doit jouer après le coup joué ;
            - On crée une variable etiquette_new à l'aide des deux variables précédentes et correspondant à la nouvelle étiquette après le coup ;
            - On donne un numéro de sommet ind_new_sommet au sommet qu'on va crée, puis, on rajoute le sommet, l'arête et l'étiquette correspondant à ce nouveau sommet lié au nouvel état de la partie.
            - on rajoute la profondeur de ce sommet à la liste des profondeurs
            - on rajoute le nouveau sommet à la file d'attente.
        - Si c'est au joueur 1 de jouer
            - On effectue les mêmes opérations (*démerdez-vous*) .

Remarque 6.1. *Dans jeu :*

- *jeu[6] correspond au grenier du joueur 0 ;*
- *jeu[13] correspond au grenier du joueur 1 ;*

- *jeu[i] pour i in range(6) correspond au trou du joueur 0 ;*
- *jeu[i] pour i in range(7,13) correspond au trou du joueur 1 ;*
- *pour un trou jeu[i], le trou en face est jeu[12-i] ;*
- *Il est fort utile de travailler modulo 14 ;*
- *On continue de noter les sommets de 0 à len(S)-1.*

5 Écrire une fonction python `sommets_controlés(S,A,E)` qui renvoie la liste des sommets contrôlés par le joueur 0

6 Il faut une fonction pour évaluer si l'état est favorable au joueur 0 ou non, pour prendre un exemple simple, créer une fonction `valeur(etiquette)` qui prend en argument l'étiquette de l'état de la partie et renvoie le nombre de billes dans le grenier du joueur 0 moins celles du grenier du joueur 1. On oubliera pas d'inclure celles sur le plateau si la partie est terminée.

7 Modifier la fonction `minimax(S,A,S0,F,V)` en `minimax(S:list,A:list,E:list,S0:list,valeur:function)` qui effectue l'algorithme du minimax. On initialisera l'algorithme à l'aide des feuilles de l'arbre (sommet sans successeur) avec la valeur donnée par la fonction valeur. La fonction minimax renverra une liste value des valeurs attribuées à chaque sommet de l'arbre et une etiquette d'un successeur du sommet 0 dont la valeur attribuée est la même que celle du sommet 0. On le choisira aléatoirement parmi tous les successeurs de 0 vérifiant cette propriété.

8 Créer une fonction `jouer(etiquette)` qui prend en argument l'étiquette de l'état du jeu, la fonction demandera au joueur quelle trou il veut jouer entre le numéro 0 au numéro 5 (utiliser la fonction input -> google est votre "ami") on renverra la partie après le coup joué selon le trou choisi.

9 Créer une fonction `game(etiquette)` qui prend en argument l'état de la partie et permet de jouer en tant que joueur 0 à une partie de Mancala contre un joueur 1 utilisant l'algorithme du minimax. On utilisera la fonction `affichage` pour afficher le jeu, on affichera (print) du texte pour préciser les différentes étapes du jeu (*Vous pouvez personnifier le joueur 1, par exemple, l'appeler Seigneur Vador*).