Année 2022/2023
Résolution de Problèmes
Master 1, MIV

Série de Travaux Pratiques N°3
Modélisation et réalisation du jeu Mancala

Le Mancala est un jeu de semis appelé aussi Awalé, d'origine africaine. Le Mancala se joue sur un plateau (board) qui a un certain nombre de petites fosses (pits), généralement 6 fosses de chaque côté, et une grande fosse, appelée magasin (store), à chaque extrémité. Chacune des fosses contient un certain nombre de graines (seeds), généralement 4. Une visualisation du plateau Mancala est illustrée dans la Figure 1.

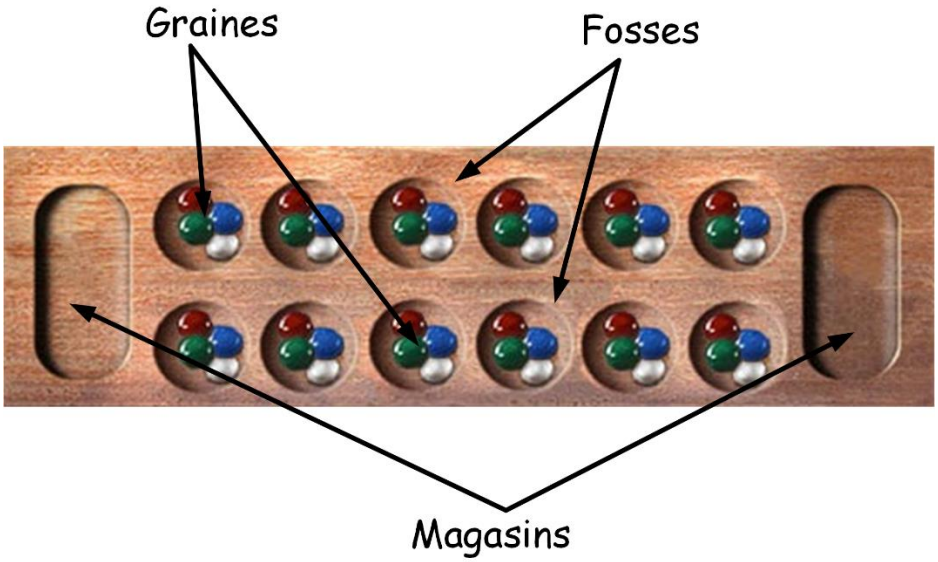

Figure 1 Mancala Board

1. **Organisation du jeu:** L'organisation du jeu Mancala est illustrée dans la Figure 2.
   - Les joueurs sont assis l'un en face de l'autre avec le plateau du jeu Manacala entre les deux. Les six fosses sur le côté de chaque joueur lui appartiennent ;
   - Chaque joueur place 4 graines dans chacune de ses 6 fosses ;
   - Le magasin de chaque joueur se trouve sur sa droite. Le magasin est initialement vide.

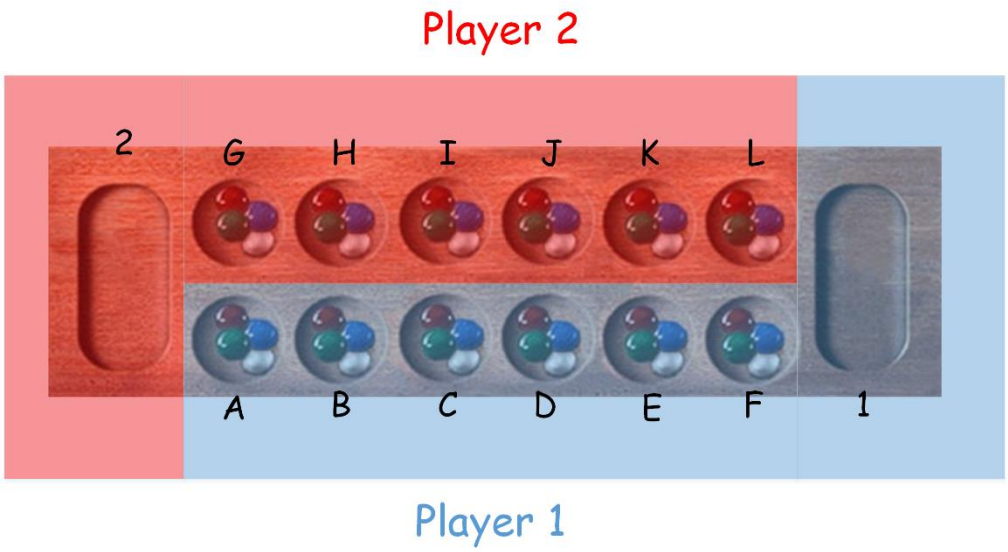

Figure 2 Organisation du jeu Mancala

2. **Objectif du jeu :** L'objectif du jeu est de capturer le plus de graines que l'adversaire.

3. **Règles du jeu :** Les règles de ce jeu sont très simples :
   - Le premier joueur choisit une fosse de son côté du plateau et ramasse toutes ses graines ;
   - Dans le sens inverse des aiguilles d'une montre, le joueur dépose une pierre dans chaque fosse jusqu'à ce qu'ils n'aient plus de graines dans sa main;
   - Le joueur peut déposer une graine dans n'importe quelle fosse du plateau (y compris son magasin) à l'exception du magasin de l'adversaire ;
   - Si la dernière graine déposée atterrit dans le magasin du joueur, ce joueur peut jouer un tour supplémentaire ;
   - Si la dernière graine déposée atterrit dans une fosse vide du côté du joueur, cette graine et toutes les graines dans la fosse du côté opposé (c'est-à-dire une fosse de l'adversaire) vont à ce joueur, et sont placées dans son magasin ;
   - Lorsqu'un joueur n'a plus de graines dans toutes les fosses de son côté, la partie se termine, et le joueur adverse peut prendre toutes les graines restantes et les placer dans son propre magasin ;
   - Le joueur avec le plus de graines dans son magasin, gagne.

4. **Meilleur mouvement d'ouverture :** C'est idéal qu'un joueur ouvre le jeu en récoltant les graines de sa troisième fosse, car la dernière graine va atterrir dans son magasin. Cela marque un point et permet au joueur d'avoir un tour supplémentaire. Dans le deuxième tour, si le joueur récolte les graines dans sa fosse la plus à droite, il marque un autre point et dépose la dernière graine dans la troisième fosse de l'adversaire (cela l'empêche de faire de même). Une illustration du meilleur mouvement d'ouverture joué par Player 1 est représentée dans la Figure 3.

Figure 3 Meilleur mouvement d'ouverture

Partie 1 :

Proposer une modélisation du jeu Mancala, et faire son implémentation en Python.

1. **Modélisation d'un état:** pour pouvoir modéliser un état, il faut créer une classe **« MancalaBoard »** où vous allez définir les attributs et les fonctions suivantes :
   - Le jeu se présente sous forme d'un plateau (board), qu'on peut représenter à l'aide d'un dictionnaire dont les clés sont les indices des 12 fosses et 2 magasins, et les valeurs sont le nombre de graines qu'il y a à l'intérieur. Les fosses du joueur 1 sont représentées par des lettres de ***A*** à ***F*** et les fosses du joueur 2 sont représentées par des lettres de ***G*** à ***L***. Les magasins des joueurs 1 et 2 sont représentées par les chiffres correspondants.
   - Si besoin, ajouter deux tuples, un pour stocker les indices (les lettres) des fosses du joueur 1, et un autre pour stocker les indices (les lettres) des fosses du joueur 2;
   - Si besoin, ajouter deux dictionnaires, un pour stocker la fosse opposée, et un autre pour stocker la fosse suivante;
   - Définir la fonction **possibleMoves()** qui va retourner les coups possibles (c'est-à-dire, les indices des fosses du joueur qui contiennent des graines) ;
   - Définir la fonction **doMove()** qui va exécuter un mouvement, et retourner le numéro du joueur qui va jouer le prochain tour (le même joueur ou l'adversaire);

2. **Modélisation du jeu:** Afin de pouvoir modéliser ce jeu, il faut créer les deux classes suivantes :

a) La classe **« Game »** qui va représenter le nœud de l'arbre de recherche. Dans cette classe, vous allez définir ce qui suit:
   - Un attribut **state** qui va représenter l'état (c'est-à-dire une instance de la classe MancalaBoard) ;
   - Un attribut **playerSide**, où vous allez stocker le numéro du joueur choisi par l'utilisateur (player1 ou player2)
   - La fonction **gameOver()** qui va vérifier si la partie est finie (c'est-à-dire, toutes les fosses de l'un des joueurs sont vides). Cette fonction va aussi permettre de récolter toutes les graines qui sont restées dans les fosses de l'adversaire et de les placer dans son magasin ;
   - La fonction **findWinner()** qui va retourner le gagnant de la partie, ainsi que son score;

- La fonction **evaluate()** qui va estimer le gain dans l'algorithme de recherche NegaMax with Alpha-Beta Pruning. Cette fonction va attribuer une valeur à un nœud n dans l'arbre de recherche selon l'équation suivante (Equation 1):

$$value(n) = nbSeedsStore(player1) - nbSeedsStore(player2) \quad (1)$$

b) La classe **« Play »** où vous allez définir:

- La fonction **humanTurn** qui va permettre à l'utilisateur de jouer son tour ;
- La fonction **computerTurn** qui va permettre à l'ordinateur de jouer son tour. Le choix du coup à jouer par l'ordinateur se fait à l'aide de l'algorithme de recherche pour les jeux;
- La fonction de recherche pour les jeux **NegaMaxAlphaBetaPruning.**
  - Dans les jeux simples tels que Fan-Tan et Tic-Tac-Toe, l'ordinateur est capable de développer l'arbre de recherche entier et de trouver la séquence de mouvements qui va entrainer une victoire. Par contre, les jeux comme les Echecs et Mancala ont trop de mouvements et de variations, donc c'est difficile pour un ordinateur d'être en mesure de rechercher d'une manière réaliste toutes les différentes variantes. La raison pour laquelle, on doit préciser la profondeur maximale de l'arbre dans les algorithmes de recherche et d'utiliser la meilleure heuristique comme fonction d'évaluation d'un nœud ;
  - Généralement, **COMPUTER=MAX** et **HUMAN=MIN** ;
  - Le pseudo code de la fonction de recherche NegaMaxAlphaBetaPruning est donné comme suit :

```
NegaMaxAlphaBetaPruning (game, player, depth, alpha, beta)
// game est une instance de la classe Game et player = COMPUTER ou HUMAN
Begin
  If game.gameOver() or depth == 1
  {
    bestValue = game.evaluate()
    bestPit = None
    If player == HUMAN
    {
          bestValue = - bestValue
    }
     return  bestValue, bestPit
  }
  bestValue = -inf
  bestPit = None
  For pit in game.state.possibleMoves(game.playerSide[player])
  {
    child_game = copy(game)
    child_game.state.doMove(game.playerSide[player], pit)
    value, _ = NegaMaxAlphaBetaPruning (child_game, -player, depth-1, -beta, -alpha)
    value = - value
    If value > bestValue
```

```
    {
        bestValue = value
        bestPit =pit
    }
    If bestValue > alpha
    {
        alpha = bestValue
    }
    If beta <= alpha
        Break
    }
  }
  return bestValue, bestPit
End
```