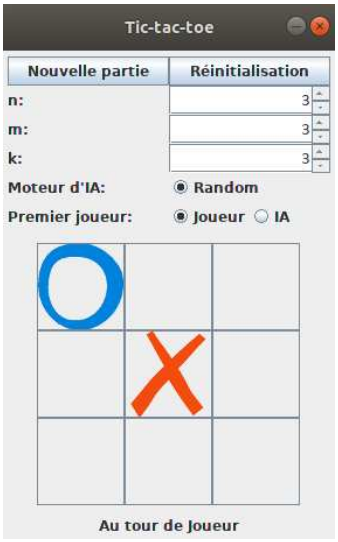
Intelligence artificielle
Master 1 Informatique
TP 1

Tic-tac-toe



On souhaite programmer une intelligence artificielle qui joue au jeu du Tic-tac-toe (Morpion, n, m, k-game) en utilisant l'algorithme MinMax.
Un moteur de jeu et une interface utilisateur ont été développés en langage Java.
Créer une classe `MinMaxPlayer` qui étend la classe `AIPlayer`. L'IA programmé dans cette classe prendra ses décision en suivant l'algorithme MinMax. En particulier, elle devra implanter la méthode `double [][] getDecision (Board board)` qui retourne un tableau à deux dimensions contenant le score attribué à chaque coup possible (la plus grande valeur sera choisie) en fonction de l'état du jeu.
Les trois états terminaux du jeu sont la victoire ou la défaite (obtenue par la méthode `boolean win (int row, int col)` de la classe `Board`), et le match nul quand la grille est pleine et qu'il n'y a pas de gagnant (méthode `boolean isFull ()` de la classe `Board`).
Afficher le nombre de nœuds évalués par l'algorithme.

Élagage

Modifier l'algorithme MinMax pour ajouter deux méthode d'élagage.
La première consiste à détecter si l'évaluation partielle d'un nœud (avant d'avoir évalué tous ses fils) est déjà l'extremum pour le type de nœud : le maximum pour un nœud « Max » (victoire de l'IA) ou le minimum pour un nœud « Min » (défaite de l'IA).
La seconde méthode est l'élagage alpha-bêta.
Afficher le nombre de nœuds évalués par l'algorithme avec l'un ou l'autre des méthodes d'élagage, et avec les deux simultanément.

Profondeur maximale

Modifier l'algorithme MinMax pour ajouter une limite à la profondeur de l'arbre. Pour les nœuds dont l'issue est indéterminée, on rajoutera une valeur possible pour le score, en respectant l'ordre suivant : défaite, indéterminé, égalité, victoire. La profondeur de l'arbre est fixée par une constante (ou en fonction de la taille de la grille).

Doublons

Dans le jeu du Tic-tac-toe, plusieurs chemins peuvent mener vers des nœuds identiques (strictement identique ou identique à une symétrie près). Afin d'éviter d'évaluer plusieurs fois le même nœuds, on peut créer une liste des nœuds déjà visités. Chaque fois que l'on crée un nouveau nœuds, l'algorithme doit d'abord vérifier s'il n'appartient pas déjà à la liste. Plus précisément, on utilisera une table de hachage (`HashMap`) et les nœuds seront indicé par un identifiant unique que l'on peut obtenir via la méthode `long getIndex ()` de la classe `Board`.