## L’objet TreeSTD

Une arborescence est représenté par 4 attributs :

* un objet de type couple qui contient la Map d’un plateau et le nombre de points gagnés par l’IA.
* un entier qui indique le numéro du noeud dans l’arborescence
* un objet de type Tree qui représente la partie gauche si le TreeSTD est un noeud
* un objet de type Tree qui représente la partie droite si le TreeSTD est un noeud

Afin de calculer le meilleur coup possible à jouer pour l’IA, un arbre sera construit avant chaque tour de celui-là. Cet arbre contiendra 3 niveaux qui seront :

* la racine qui est le plateau de jeu dans l’état où c’est à l’IA de jouer
* un niveau supplémentaire qui contiendra tous les coups possibles pour l’IA
* un autre niveau supplémentaire qui contiendra les coups possibles pour le joueur.

Ainsi, avec ces 3 niveaux, il sera possible de calculer le coup qui permettra à l’IA de gagner le plus de points ou d’en perdre le moins.

## Construction de l’arborescence

Fonction construire\_arborescence(p : plateau) : arborescence

*{Fonction qui permet d’appeler toutes les fonctions de recherche sur le plateau passé en paramètre pour créer l’arborescence des coups possibles}*

Début

var TreeSTD arborescence <- vide

Pour chaque cellule c appartenant à l’IA faire

*{Construction de la structure de l’arborescence, ajout de tous les noeuds sans la valeur des points}*

Début Pour

arborescence <- recherche(p, c, arborescence, 1)

*{Ici le 1 permet d’indiquer que la recherche de coups possibles se fait pour l’IA}*

Pour chaque cellule c1 appartenant au joueur faire

Début Pour

arborescence <- recherche(p, c1, arborescence, 0}

*{Ici le 0 permet d’indiquer que la recherche de coups possibles se fait pour le joueur}*

Fin Pour

Fin Pour

Pour chaque feuille f de arborescence faire

*{Calcul des points gagnés ou perdus par l’IA}*

Début Pour

var noeud\_pere <- noeud pere de f

valeur de f <- calcul\_point(f, noeud\_pere)

Fin Pour

Retourner (arborescence)

Fin

Cet algorithme permet de construire l’objet de type TreeSTD correspondant au plateau passé en paramètre. Il recherche dans un premier temps tous les coups possibles pour l’IA pour ensuite rechercher tous les coups possible pour le joueur en fonction du coup d’avant. Pour finir, il calcule la valeur de chaque feuille de l’arborescence. Cette valeur est le nombre de points perdus ou gagnés par l’IA pour chaque combinaison de coups possible.

## L’algorithme minimax

### Explication

L’algorithme Minimax est un algorithme applicable que les jeux à 2 joueurs, à somme nulle et à information complète. Dans le cas du jeu Othello, cet algorithme est totalement applicable pour trouver le meilleur coup possible pour l’IA. En effet, l’Othello se joue à 2, est à somme nulle car les points gagnés par un joueur sont des points perdus par le joueur adverse et est à information complète, les deux joueurs ayant accès aux mêmes informations.

Cet algorithme permettra de trouver le coup pour l’IA qui lui permettra de gagner le plus de points, ou du moins, en perdre le moins.

Fonction minimax (s: noeud, T: arborescence) : réel

*{Algorithme qui permet de trouver la combinaison qui permet de minimiser les pertes}*

Début

var val réel

Si s est une feuille alors

Début Si

val <- k(s)

Sinon

Si s est de type Max alors

Début Si

val <- - infini

Pour tout successeur s’ de s faire

Début Pour

val <- maximum(val, minimax(s’, T))

Fin Pour

Sinon

val <- +infini

Pour tout successeurs s’ de s faire

Début Pour

val <- minimum(val, minimax(s’, T))

Fin Pour

Fin Si

Retourner(val)

Fin Si

Fin

### Implémentation

Pour des soucis de clarté, il a été décidé de créer un objet avec un type différent pour chaque algorithme possible. Ainsi, c’est l’objet Minimax qui, avec l’arborescence passer en paramètre, calcule le meilleur coup possible ainsi que le nombre de point associé à ce coup grâce à la fonction de cette classe Minimax.