 **Rapport TP - Algo Minmax, application au tictactoe**

Balayssac Antoine, Rassat Théo

Semestre 2 - Année 2018/2019

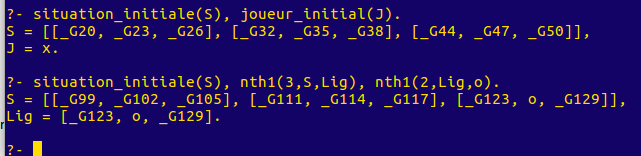
**Introduction :**

Au cours de ce tp, nous avons implémenté l’algorithme Minmax pour la convention Negamax. Le but est de rechercher le coup optimal dans un arbre de jeu à 2 joueurs depuis la situation initiale, pour une profondeur variant de 1 à 7 coups.

Nous suivrons au cours de ce compte-rendu de TP les étapes suivies lors de l’implémentation, en répondant aux questions posées, en y ajoutant éventuellement un screenshot de code. Dans ce même mail, sont fournis nos codes sources.

1. **Familiarisation avec le problème du TicTacToe 3x3**

**A - Quelle interprétation donnez-vous aux requêtes suivantes?**

****

* La première requête nous permet de récupérer la situation initiale dans la variable S. Elle est alors représentée par une matrice de variable non liée. Ce qui est possible dans ce jeu car au TicTacToe on doit instancier la grille. Les joueurs pourront ensuite lier ces variables par leur signe respectif (x,o). On ne crée ainsi pas de nouvelle grille à chaque modification.

Le joueur qui commence est récupéré par la seconde partie de la requête dans la variable J. Cette convention arbitraire est donnée dans le fichier tictactoe.pl. Les joueurs sont matérialisés par le signe “x” ou “o”. C’est le joueur x qui commence.

* La seconde requête récupère la situation initiale dans la variable S comme précédemment. Ensuite, grâce au prédicat Prolog nth1, on récupère premièrement la 3ème ligne de la matrice S dans la variable Lig et on lie ensuite le 2ème élément de celle-ci avec “o”. (nth1 commence à “compter” à 1). Cela modélise donc le fait que le joueur 0 a joué sur la situation initiale en 3x2.

**B - Alignements**

Il s’agit ici, de définir un prédicat alignement qui pour une matrice donnée, renvoi tout ces alignements possibles : lignes, colonnes et diagonales.

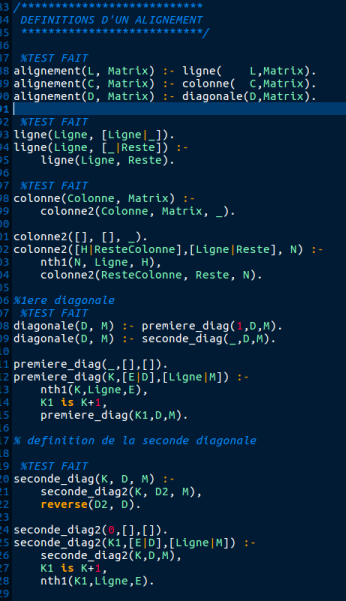
Pour accéder à une ligne, on avance récursivement dans la matrice en récupérant chacune de ses “sous-listes” (cf code ci-dessous)

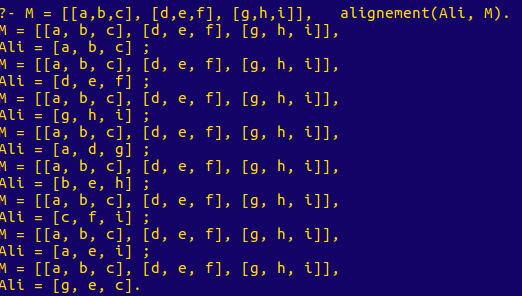
Pour une colonne, nous devons avancer récursivement dans la matrice, et récupérer sur chaque ligne, chaque élément N et l’ajouter à une nouvelle liste où les Nème éléments des autres lignes de la matrice seront aussi ajoutés. (cf code ci-dessous)

Pour récupérer les diagonales, il faut récupérer la première et la deuxième distinctement. Pour la première, nous devons parcourir la matrice sur ce même principe et incrémenter un compteur à chaque étape afin d’ajouter le 1er élément de la première ligne, puis le 2ème élément de la seconde et ainsi de suite jusqu’à N à la même liste qui composera la diagonale.

Pour la seconde, le principe est similaire, mais avant d’incrémenter le compteur il faut d’abord se placer sur la dernière ligne de la matrice et ensuite remonter, afin d’ajouter à une nouvelle liste qui sera la seconde diagonale, le premier élément de la Nème ligne, le second de la N-1ème et le Nème de la première ligne.

On obtiendra ainsi par ces 3 prédicats l’ensemble des alignements possibles. En voici le code :



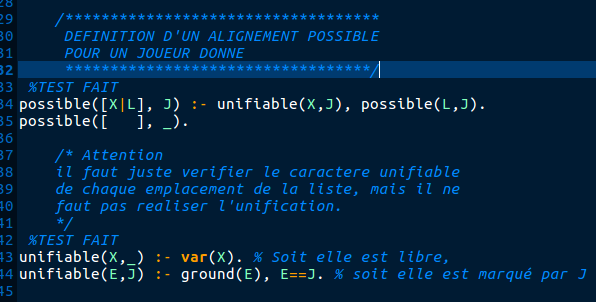


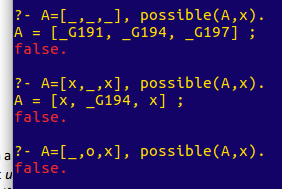
Nous obtenons bien les 8 alignements d’une matrice NxN. Les lignes, puis les colonnes et enfin les diagonales sont obtenus.

On veut maintenant savoir si pour un alignement A et un joueur J donné, ce joueur J peut encore gagner sur cet alignement, c’est à dire, si toute les cases de l’alignement sont encore libre ou appartiennent déjà à J.

Le prédicat possible va donc avancer récursivement sur chaque case de l’alignement et appeler le prédicat unifiable pour chacune de celle-ci qui renverra true si la variable est libre (prédicat var) ou qu’elle vaut déjà J.

En voici le code et son comportement :



 Le comportement suivant est obtenu. Il est valide car notre première requête nous renvoi une solution pour un alignement avec des variables libre, la 2ème pour un alignement composé de variables libres et du symbole de J et la 3ème, aucune solution car cet alignement n’est plus gagnable pour J. (présence d’un symbole adverse)

Nous définissons maintenant les prédicats alignement\_gagnant(A,J), alignement\_perdant(A,J) qui valide qu’un alignement est totalement lié à un joueur ou pour perdant, à son adversaire. On utilise le prédicat ground qui renvoi true si la variable donnée est tot

1. **Conception du XML schema, étape clée**
2. **Documents XML et validation du modèle**
   1. **Ecriture des fichier XML**

* 1. **Validation et correction du XSD**

1. **Transformation en fichier HTML via XSLT**

**Conclusion :**