**Projet *Teeko*   
IA41 P15**

**Sommaire**

1. **Rappel de l’énoncé**
2. **Analyse du Problème**
3. **Méthodes proposées**
4. **Situations traitées**
5. **Résultats**
6. **Difficultés**
7. **Améliorations possibles**
8. **Rappel de l’énoncé**

Tout d’abord un peu d’histoire : Teeko est un [jeu de stratégie combinatoire abstrait](http://fr.wikipedia.org/wiki/Jeu_de_stratégie_combinatoire_abstrait) inventé par le magicien [John Scarne](http://fr.wikipedia.org/w/index.php?title=John_Scarne&action=edit&redlink=1) en 1937, revu en 1952, puis une dernière fois au cours des années 60. Le mot " teeko " vient des autres jeux qui ont inspiré l'inventeur : le "T" de tic-tac-toe, le "E" de chess ("échecs"), le "K" de checkers ("damier") et le "O" de bingo. À quoi Scarne rajouta un "E" pour des raisons phonétiques.

Le Teeko se joue à deux et repose sur :

* Un plateau de jeu composé de vingt-cinq cases (cinq par cinq)
* Quatre pions de couleur noire (joueur "Noir")
* Quatre pions de couleur blanche (joueur "Blanc")
* Le but du jeu est d’aligner (horizontal, vertical, diagonal) ses pions ou de les positionner en carré.



Maintenant les règles du jeu :

* Le plateau est vide en début de partie.
* Tour à tour, les joueurs posent un de leurs pions sur une intersection libre (une case libre)
* emplacements de 1 à 45 sur la figure ci-dessous.
* Une case est libre si elle ne contient pas déjà un pion.
* A la fin de cette phase de pose, si aucun joueur n'a obtenu de configuration gagnante, les joueurs déplacent à tour de rôle l'un de leurs pions.
* Un pion déplacé ne peut l'être que sur un emplacement libre adjacent à des pions.
* On peut donc déplacer un pion sur une case dont au moins une de ces cases adjacentes n’est pas vide.
* Dès qu'un joueur réalise une configuration gagnante, la partie s'arrête.

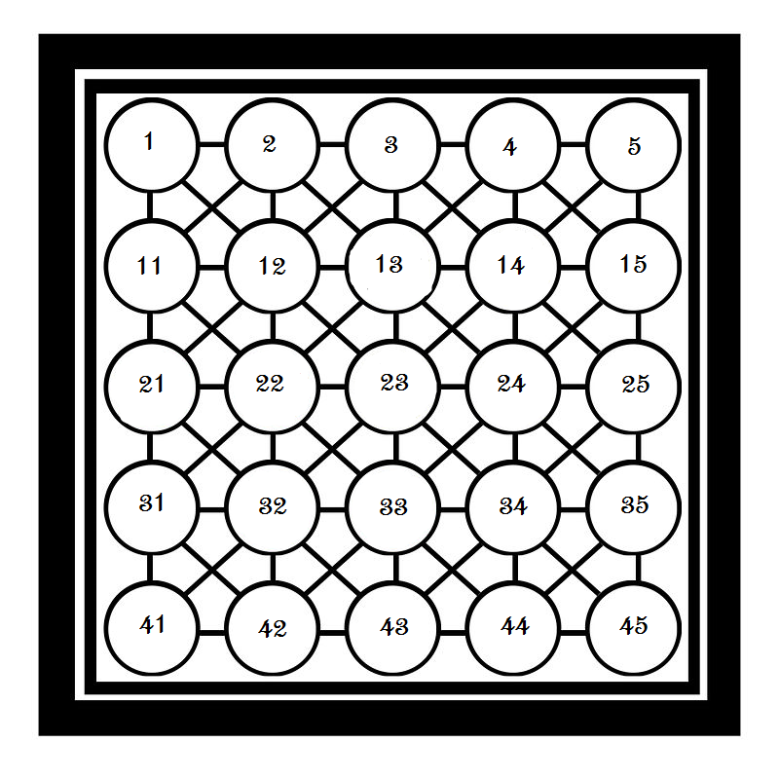
1. **Analyse du Problème**

Nous nous fixons comme objectif d’implémenter le jeu du Teeko permettant de jouer une partie humain contre humain et également une partie humain contre ordinateur ou ordinateur contre ordinateur.

On a représenté le plateau en donnant seulement 2 listes : celle du joueur et celle de l'adeversaire. Ces listes contiennent les emplacements des pions de chaque joueur. Donc, les numéros qui ne sont dans aucune liste représentent des cases vides. Un mouvement est représenté par le changement d'un élément du joueur actif.

Le problème de l'écriture du programme a d'abord été découpé en sous-problèmes :

* Définir la représentation des données (le plateau de jeu)
* Ecrire la fonction d’affichage
* Ecrire la fonction d’évaluation qui permet d’attribuer un score à tous les coups possibles des joueurs
* Algorithme utilisé pour l’IA : alpha beta

1. **Méthodes proposées**

Nous avons décidé de représenter notre plateau par 2 deux listes croissantes de quatre chiffres. Le numéro de la place d’un pion est déterminé par le tableau ci-contre. Par exemple dans le tableau ci-dessous la liste du joueur noire est [13, 14, 23, 24]. A l'initialisation, ces listes sont deux listes vides.



* L’affichage du plateau se fait grâce à la fonction *display\_board(+BoardBlanc,+BoardNoir)*. Un B signifie un jeton blanc, N signifie un jeton noir et X une case vide.
* *score(+BoardBlanc,+BoardNoir,+Init, ?ScoreTotal)* qui calcule la valeur d’une position suivant la position de chaque pion.
* *getPossibleMovementL(+Joueur, +Adversaire, -ListeMvmt)* qui retourne tous les mouvements possibles d’un joueur.
* *ordonner(+L, -O)* qui ordonne la liste L dans l'ordre croissant et donne O
* *play()* qui lance le jeu et donne le choix du mode de jeu
* *alphaBeta(+Joueur1,+Adv,+Profondeur,?MouvementJoueur1,?ValeurDuCoup)* fonction alphabéta qui calcule le meilleur coup que l’IA peut faire. Le coup sera plus ou moins bien suivant la Profondeur qui représente la taille de l’arbre parcouru, elle est définie par le niveau que l’utilisateur a donné au début.

L’Algorithme AlphaBeta

Pour faire jouer l’ordinateur correctement, nous avons besoin d’évaluer et choisir le meilleur coup. Pour cela deux algorithmes sont à notre disposition : le minimax et l’alphabeta.

Nous avons décidé d’implémenter l’alphabeta car ce dernier est plus optimisé.

* Principe du MiniMax :

Cette technique amène l’ordinateur à passer en revue toutes les possibilités pour un nombre limité de coups (une profondeur) et à leur assigner une valeur qui prend en compte les bénéfices pour le joueur et pour son adversaire. Le meilleur choix étant alors celui qui maximise ses bénéfices et minimise ceux de son adversaire.

Cet algorithme s’applique au jeu où s’opposent deux joueurs. De plus, on suppose qu’aucune partie de ce jeu ne comporte un nombre infini de coups, mais qu’à chaque coup il existe un nombre fini de possibilités.

En fait, on parcourt l’arbre de jeu pour faire remonter à la racine une valeur dite valeur de jeu qui est calculée récursivement de la façon suivante (f est la fonction d’évaluation) :

MiniMax (coup) = f (coup) si coup est une position terminale

MiniMax (coup) = max (MiniMax(h1),...,MiniMax(hn)) avec hn les fils du coup

MiniMax (coup) = min (MiniMax(j1),...,MiniMax(jn)) avec jn les fils mais c’est le coup de l’autre joueur

* Principe de l’AlphaBeta:

L’alphabeta permet l’élagage de l’arbre. Cet élagage permet de réduire le nombre de nœuds évalués par l’algorithme MiniMax. L’algorithme MiniMax effectue en effet une exploration complète de l’arbre de jeu jusqu’à une profondeur donnée, alors qu’une exploration partielle de l’arbre est généralement suffisante. Lors de l’exploration, il n’est pas nécessaire d’examiner les sous-arbres qui conduisent à des configurations dont la valeur ne contribuera sûrement pas au calcul du gain à la racine de l’arbre.

L’élagage AlphaBeta évite d’évaluer des nœuds de l’arbre dont on est sûr que leur qualité sera inférieure à un nœud déjà évalué.

1. **Situations traitées**

Nous avons décidés de faire 3 modes de jeu :

* joueur vs joueur
* joueur vs IA
* IA vs IA

L’IA peut avoir 2 niveaux de difficultés que l’utilisateur définira au début.

1. **Résultats**

Nous allons vous présenter le mode de jeu joueur vs IA avec une IA de niveau 2.

1. **Difficultés**

Nous avons essayé de créer une interface en Prolog puis ensuite en C++ mais dans les deux cas nous n’avons pas réussi. Nous avons aussi rencontrés plusieurs difficultés liés à la connaissance du Prolog. Nous nous sommes également posé de nombreuses questions sur la fonction d’évaluation (comment rendre les coups de l’IA intelligents).

1. **Améliorations possibles**

Il serait possible d’implémenter une meilleure interface graphique qui représente le plateau de jeu. Le joueur n’aurait qu’à cliquer sur le pion qu’il veut déplacer et sur la case vide adjacente sur laquelle il désire le poser.

On peut aussi améliorer notre IA pour la rendre plus performante et plus rapide.

**Conclusion**

Ce projet est une application concrète des notions vues en cours, il nous a permis de solidifier nos connaissances en Prolog. De plus, nous avions tous déjà joué au jeu de société teeko ce qui a rendu ce projet d’autant plus intéressant. Nous avons pu aussi mieux comprendre le principe de l’intelligence artificielle grâce aux recherches effectuées pour implémenter l’AlphaBeta.

Il nous a également permis d’améliorer notre travail en groupe. Bien qu’au moment d’écrire ce rapport, la modélisation n’est pas encore achevée, nous a avons définis précisément les tâches à accomplir, ce qui nous permet d’effectuer une meilleure répartition des tâches entre les membres et ainsi de savoir parfaitement dans quelle direction continuer.