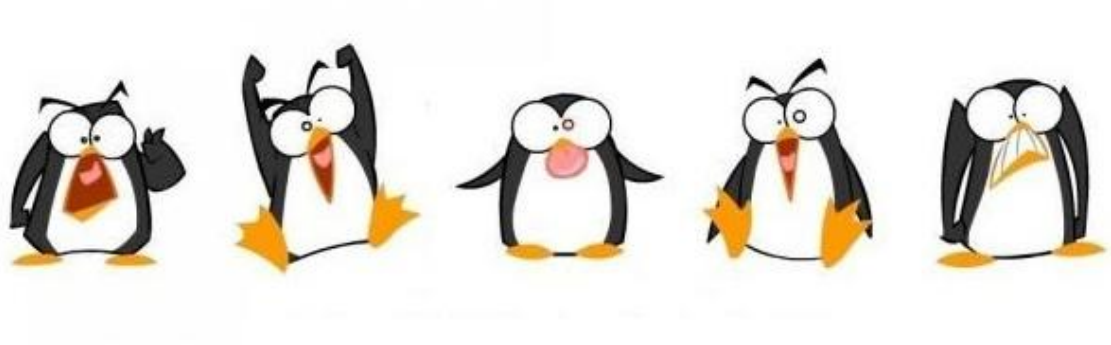
VEILLY Florent
FERRAND Sébastien
TIRAND Yannick
JUNG Nicolas

Rapport : Projet d'Unix

Calcul distribué appliqué à l'évaluation des coups pour une IA d'échec



SOMMAIRE

I. Présentation générale

1. Introduction

De nos jours, les Intelligences Artificielles (IA) prolifèrent dans beaucoup de domaines. Le jeu d'échecs n'y fait pas exception, où des IA ont été développées afin de vaincre les meilleurs joueurs du monde.

Cependant, cette IA rencontre des limites. En effet, la courbe des possibilités de coups étant exponentielle, on atteint vite la limite de puissance de calcul d'un ordinateur.

A cela, deux solutions : augmenter la puissance des ordinateurs (couteux et tout de même limité), ou mettre en place un système de calcul distribué.

Cette dernière idée va être retenue pour ce projet, mettant en œuvre de nombreuses notions vues en cours : lecture de fichier, gestion de threads, …

Pour une vaste famille de jeu, le théorème du minimax de Von Neumann assure l'existence d'une méthode combinatoire permettant d'estimer de manière fiable le meilleur coup pour une position donnée. Les échecs rentrent dans le cadre de cette théorie avec l'utilisation d'algorithmes de type MinMax.

Il amène l'ordinateur à passer en revue toutes les possibilités pour un nombre limité de coups et à leur assigner une valeur qui prend en compte les bénéfices pour le joueur et son adversaire. Le meilleur choix est alors celui qui minimise les pertes du joueur tout en supposant que l'adversaire cherche au contraire à les maximiser.

On constate rapidement que la complexité d'un tel algorithme est exponentielle de type $O(n^p)$ avec n le nombre maximum d'actions/coups légaux à chaque étape et p le nombre de coups calculé à l'avance (profondeur maximal de l'arbre). En prenant en compte que le nombre moyen de coups possibles aux échecs avoisine les 20, on se retrouve très rapidement avec des temps de calculs énormes.

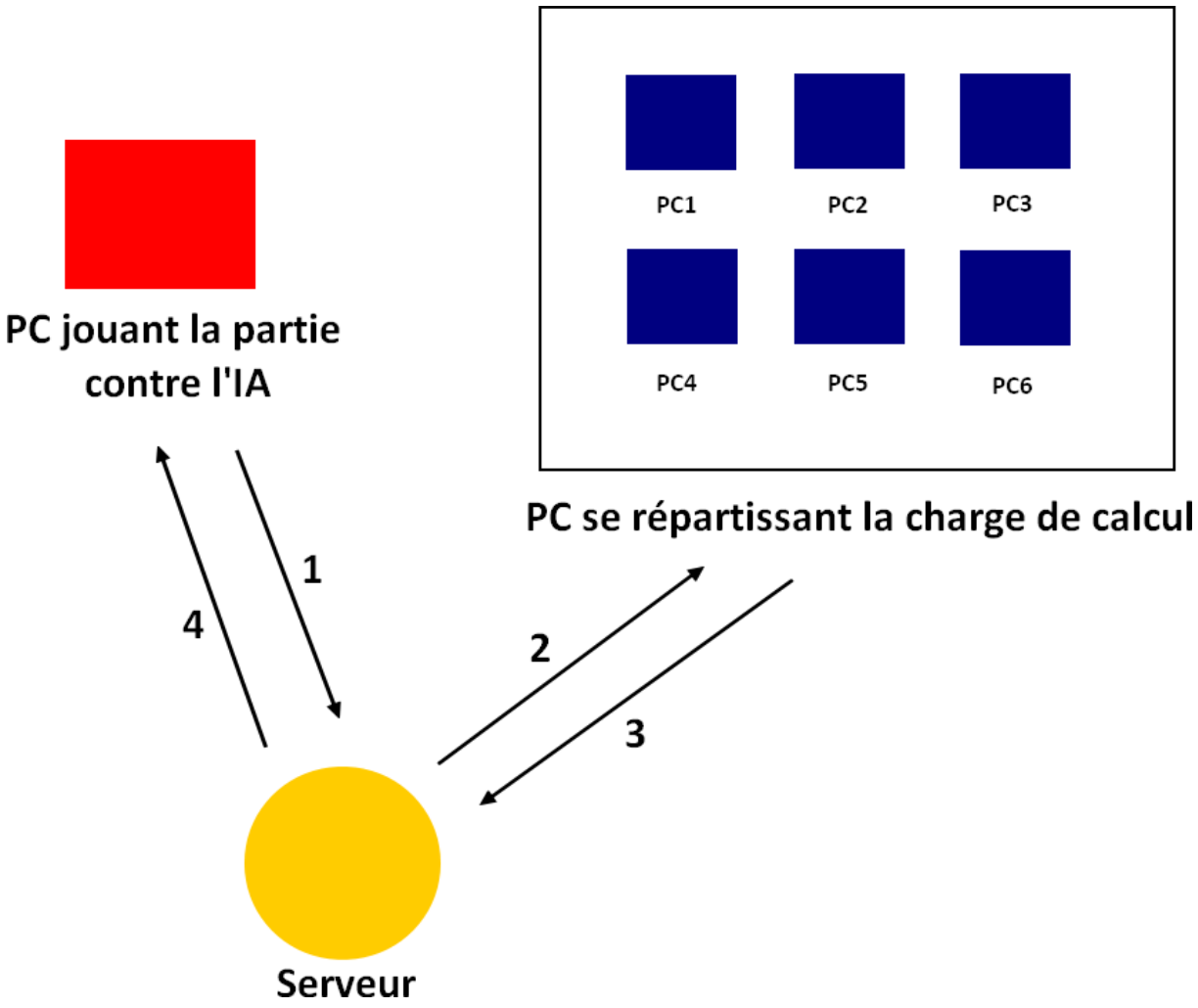
Profondeur de l'arbre	$O(1)$	$O(n^p)$
p = 1	1μs	20μs
p = 2	1μs	400μs
p = 5	1μs	3.2s
p = 10	1μs	118j

On remarque également que cet algorithme est très facilement parallélisable. Il suffit de répartir la charge de calcul de chaque branche racine de l'arbre dans des unités de calculs spécifiques. On se retrouve ainsi avec une nouvelle complexité de type $O(n^p/q)$avec q le nombre d'ordinateur effectuant les calculs en parallèle. Dans le cas ou q est supérieur au

nombre de branches issus de la racine, il suffit de rentrer dans les branches inférieurs de l'arbre pour répartir le calcul ce qui réduit ainsi la profondeur p à calculer.

Ainsi le but de notre projet est de **réaliser une application de calcul distribué destiné à répartir la charge de calcul de l'algorithme MinMax sur différents ordinateurs mis en réseaux.**

2. Architecture générale



1 : Envoie de la partie en cours. Le serveur récupère la partie divise l'arbre en fonction du nombre de clients connectés.

2 : Le serveur envoie la partie en cours ainsi que la portion d'arbre que le client doit traiter.

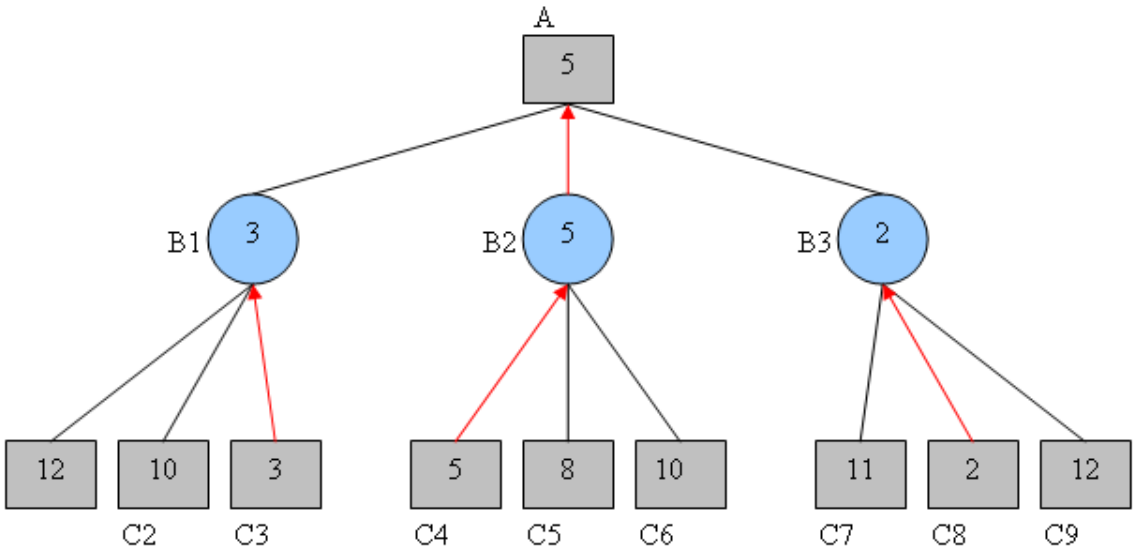
3 : Chaque client renvoie le meilleur coup qu'il a défini. Le serveur récupère parmi ces coups le meilleur.

4 : Le serveur renvoie le meilleur coup à jouer.

3. L'Algorithme MinMax

L'algorithme minimax est très simple : on visite l'arbre de jeu pour faire remonter à la racine une valeur qui est calculée récursivement de la façon suivante :

- minimax(p) = f(p) si p est une feuille de l'arbre où f est une fonction d'évaluation de la position du jeu
- minimax(p) = MAX(minimax(O_1), ..., minimax(O_n)) si p est un nœud Joueur avec fils O_1, ..., O_n
- minimax(p) = MIN(minimax(O_1), ..., minimax(O_n)) si p est un nœud Opposant avec fils O_1, ..., O_n



Dans le schéma ci-dessus, les nœuds gris représentent les nœuds joueurs et les bleus les nœuds opposants. Pour déterminer la valeur du nœud A, on choisit la valeur maximum de l'ensemble des nœuds B (A est un nœud joueur). Il faut donc déterminer les valeurs des nœuds B qui reçoivent chacun la valeur minimum stockée dans leurs fils (nœuds B sont opposants). Les nœuds C sont des feuilles, leur valeur peut donc être calculée par la fonction d'évaluation.

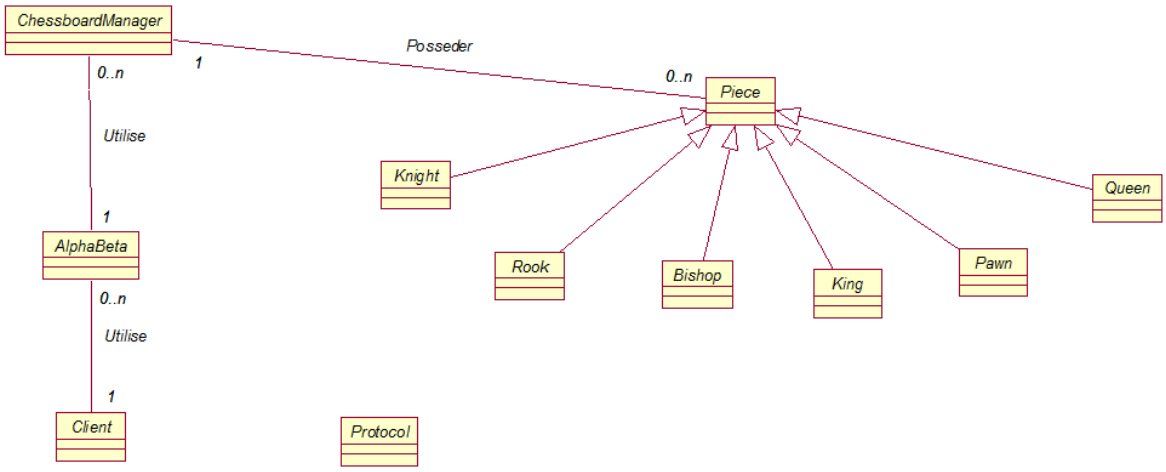
Le nœud A prend donc la valeur 5. Le joueur doit donc jouer le coup l'amenant en B2. En observant l'arbre, on comprend bien que l'algorithme considère que l'opposant va jouer de manière optimale : il prend le minimum. Sans ce prédicat, on choisirait le nœud C1 qui propose le plus grand gain et le prochain coup sélectionné amènerait en B1. Mais alors on prend le risque que l'opposant joue C3 qui propose seulement un gain de 3.

En pratique, la valeur théorique de la position P ne pourra généralement pas être calculée. En conséquence, la fonction d'évaluation sera appliquée sur des positions non terminales.

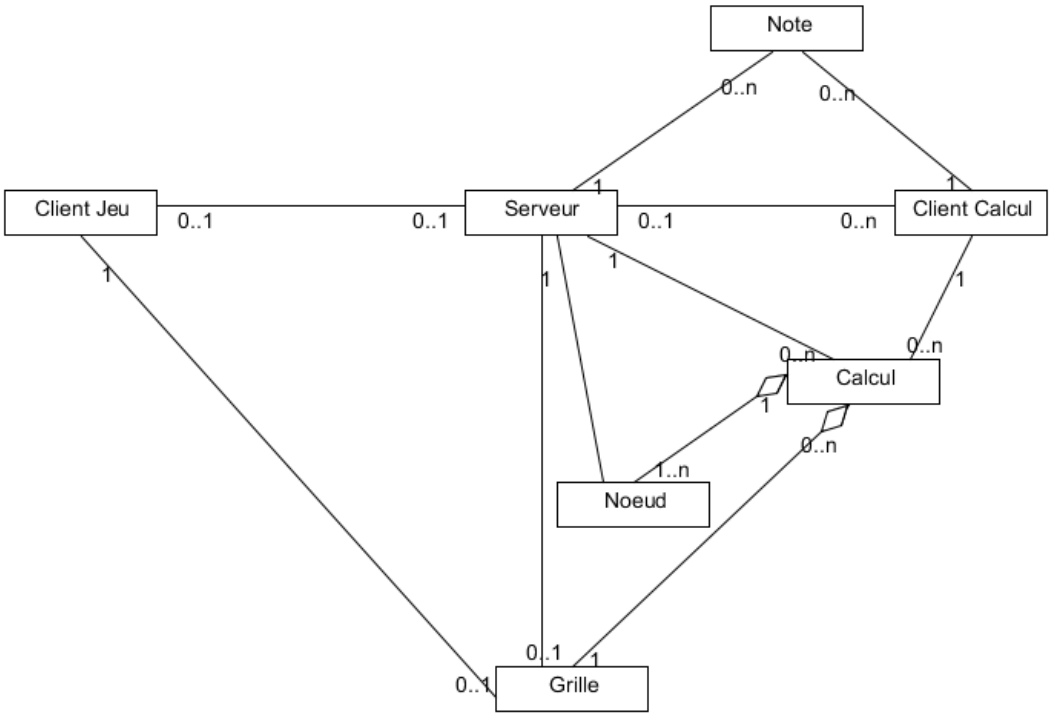
On considérera que plus la fonction d'évaluation est appliquée loin de la racine, meilleur est le résultat du calcul. C'est-à-dire qu'en examinant plus de coups successifs, nous supposons obtenir une meilleure approximation de la valeur théorique donc un meilleur choix de mouvement.

II. Analyse

1. Diagramme de classe de la partie client (calculs)



2. Diagramme de classe de la partie serveur



III. Outils, méthodologie et planification

1. Les outils

Le projet va être entièrement développé sous Netbeans 6.9. Au niveau des librairies, à part la bibliothèque standard C++, pour l'instant nous n'en utilisons aucune.

Pour ce qui est du jeu d'échec en lui-même nous avons récupérer un jeu réalisé en JAVA par nos soin le semestre dernier. Il nous suffira donc juste de le modifier pour qu'il fasse office d'interface de jeu.

2. La méthodologie

Les méthodes de développement utilisées s'apparentent aux méthodes agiles. C'est-à-dire des cycles de développement courts couplés à une revue du code en permanence.

Ces méthodes reposent sur 4 valeurs fondamentales :

- L'équipe : Dans l'optique agile, l'équipe est bien plus importante que les outils (structurants ou de contrôle) ou les procédures de fonctionnement.

- L’application : Il est vital que l'application fonctionne.
- La collaboration : Le client doit être impliqué dans le développement.
- L’acceptation au changement : La planification initiale et la structure du logiciel doivent être flexibles afin de permettre l'évolution de la demande du client tout au long du projet.

Les tests seront faits unitairement par chaque développeur puis une série de nouveaux tests sera effectué au moment de l’intégration finale.

3. Planification

Les taches ainsi que le développement ont été réparti de la manière suivante :

Client de jeu			
N°	Taches	Priorité	Responsable
T1	Modification de la partie client/serveur existante pour communiquer avec le serveur	2	Nicolas Yung
T2	Modification de notre jeu d'échec pour jouer contre le serveur	0	Yannick Tirand

Serveur			
N°	Taches	Priorité	Responsable
T3	Réalisation de la partie client/serveur pour communiquer avec les clients	2	Nicolas Yung
T4	Développement d’une fonction calculant les coups possibles pour une pièce donnée	2	Yannick Tirand & Florent Veilly
T5	Algorithme divisant l'arbre en fonction des clients connectés	2	Yannick Tirand

Clients distribués			
N°	Taches	Priorité	Responsable
T6	Réalisation de la partie client/serveur pour communiquer avec le serveur	2	Nicolas Yung
T7	Développement d’une fonction d'évaluation du jeu d'échec (comptage par point, position,...)	2	Sébastien Ferrand
T8	Implémentation de l'algorithme MinMax (optimisation AlphaBeta)	2	Florent Veilly
T4	Développement d’une fonction calculant les coups possibles pour une pièce donnée	2	Yannick Tirand & Florent Veilly

Autre			
N°	Taches	Priorité	Responsable
T9	Intégration	1	Florent Veilly

T10	Documentation utilisateur	0	Sébastien Ferrand

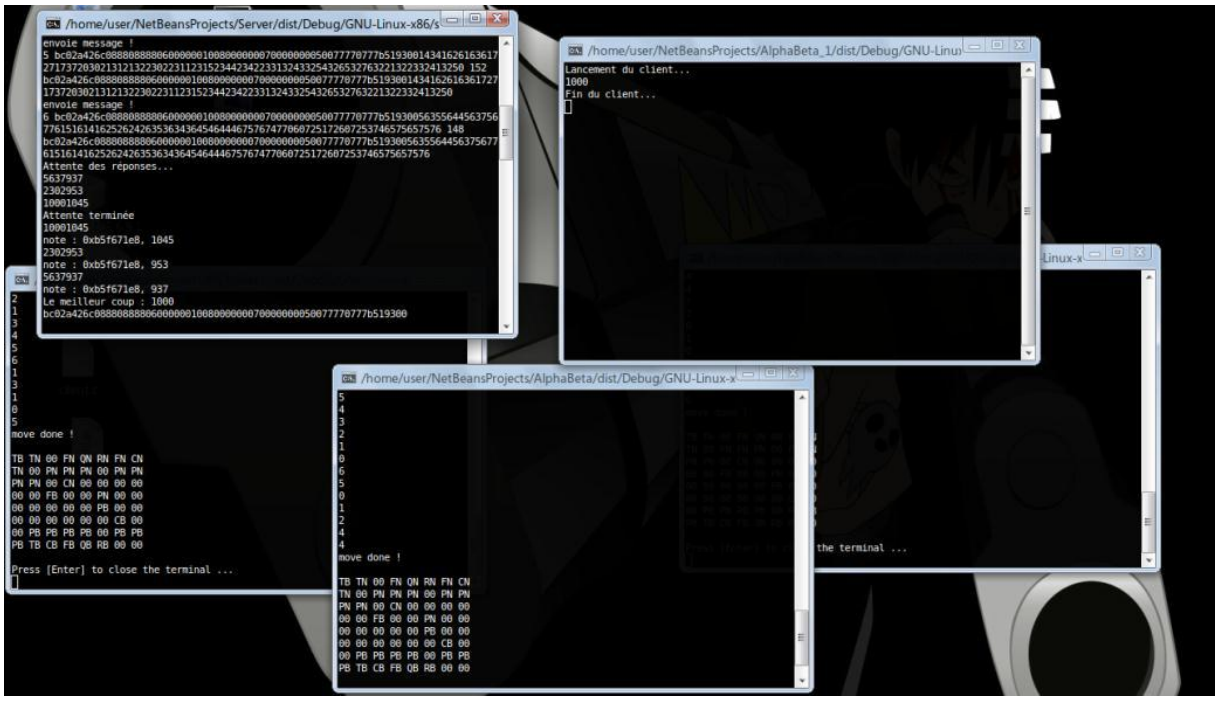
IV. Notions de programmation UNIX abordées

Plusieurs notions d'Unix ont été abordées au cours de ce projet. D'abord nous avons utilisé des threads ainsi que des sockets avec les librairies standards de Linux, en effet, les clients de calcul et celui de jeu sont mis en relation avec un thread dédié par le biais de leur socket. Ce socket est traité comme un descripteur de fichier. Ceci nous a permis de mettre en place une communication dynamique entre les serveurs. Des notions de temps réels ont ainsi émergés notamment dans les temps de réponses entre les clients calculs et le serveur.

V. Documentation utilisateur

Plusieurs modules doivent être lancés dans l'ordre pour tester l'application :

- Tout d'abord le serveur (Server) qui va rediriger les requêtes des clients
- Ensuite les clients de calculs (AlphaBeta) doivent être lancé pour pouvoir se répartir les calculs d'IA (cela devient intéressant à partir de 3 clients lancés)
- Enfin le client simulant le jeu (le module JAVA n'étant pas encore intégré bien que fonctionnel) doit être lancé pour déclencher le calcule des différents modules



VI. Assurance qualité

La qualité au cours de ce projet a été gérée de différentes façons :

- Mise en place d'un document partagé sur la liste des tâches à effectuer via Google Doc.
- Les tests unitaires et fonctionnels ont été maintenus tout au long du projet, même si les environnements de travail différents ne nous ont pas permis d'utiliser de logiciel de maintenance automatique des tests.

VII. Résultats et améliorations possibles

Plusieurs points auraient pu être approfondis au cours de ce projet :

- Pour l'instant le serveur réparti les taches en fonctions des premiers nœuds racines. Il n'y a pas de parcours en profondeur limitant ainsi le nombre de client pouvant se connecter. Toutefois, l'algorithme prévu à l'origine était prévu pour aller chercher à une plus grande profondeur pour permettre de mieux diviser les coups à calculer par les clients en fonction du nombre de ces derniers. Cet algorithme serait implémentable assez facilement.
- Par manque de temps nous n'avons pas pu mettre en place une gestion de la fermeture du programme poussé. Pour l'instant il se réaliser via CTRL+C.
- L'algorithme Negamax aurait pu être multithreadé pour une meilleure utilisation des processeurs multi-cœurs. Nous pourrions utiliser des sémaphores pour ce faire.
- Bien que l'interface de jeu JAVA fonctionne, nous n'avons pas eut le temps de l'implémenter