ECOLE
POLYTECHNIQUE
DE BRUXELLES

Projet : Reversi/Othello

INFO-H-304 : Compléments de Programmation et d'Algorithmique

Authors :

Adel Nehili, Nathan Van Begin, Julien Rizk

Group 13

Teacher :

Jérémie Roland

Année académique 2020 - 2021

Contents

1 Introduction **2**

2 Adapation Des Règles **2**

3 Choix de la structure du programme **3**

4 Optimisation de l'IA **4**

1 Introduction

Dans le cadre du projet d'INFO-H304, il a été demandé aux étudiants de reconstituer le jeu "Reversi" et par après d'implémanter une Intelligence Artificielle (IA). Pour ce faire, il a fallu interpréter les règles du jeu et les adapter au language C++, et ce en orienté objet pour permettre un travail en groupe plus aisé. De plus, cette même IA devra être capable de jouer contre d'autre IA ou joueurs. Pour remplir cette condition, il a été choisit d'utiliser des fichiers dans lesquels l'adversaire écrira les positions.

2 Adapation Des Règles

L'adaptation des règles du jeu s'est faite en 3 étapes:

La création du tableau 8x8 avec tous les pions initiaux: Pour créer un tableau de pions, il a été décidé de créer une classe Board qui prend un tableau de int dans le quel les valeurs 1 et -1 représentent respectivement les pions noires et blancs. De là, les valeurs initiales des 4 pions de départ ont été manuellement codées.

La mise en place de détecteur de nouvelles positions valables. La solution apportée est codée sous la forme d'une fonction executeMove de Game dont le travail est de vérifier si la case sélectionnée est valable, c'est-à-dire une case vide et faisant gagner au moins un pion. De plus, cette fonction permettra à Board de changer les cases du tableau en fonction. Pour ce faire, la fonction doit recevoir en paramètre le int du joueur et la postion séléctionnée afin de vérifier sur toutes les lignes traversant la position si il existe effectivement un pion allié permettant de gagner des points.

```
void EXECUTE_MOVE(int matrix[8][8], int player, int row, int col){
    bool MoveIsValid = false;
    if (matrix[row][col] == 0) {
        for (int i = -1; i <= 1; i++) {
            for (int j = -1; j <= 1; j++) {
                int k = 1;
                while ( col+k*i >= 0 && col+k*i <= 7 && row+k*j >= 0 && row+k*j <= 7 && !(i==0 && j==0)) {
                    // tant que les indices restent bien dans les limites du tableau, et que i et j ne sont pas nu
                    if (matrix[row+k*j][col+k*i] == -player) {
                        k++;
                    } else if (matrix[row+k*j][col+k*i] == player && k>1) {
                        for (int l = k-1; l >= 0; l--) {
                                matrix[row+l*j][col+l*i] = player;
                        }
                        MoveIsValid = true;
                        break;
                    } else {
                        break;
                    }
                }
            }
        }
    }
}
```

La mise en place d'un élément vérifiant que la partie ne soit pas terminée: Enfin, il fallait déterminer si il réstait encore des zones valables pour le joueur en cours. Dès lors, deux fonctions ont été rajoutées: isValidMove et validMoveExists Le rôle de la première est semblable à la fonction executeMOve à ceci prêt qu'elle ne fait que lire le tableau selon des vecteur partant de la position sléctionnée. Par après, validMoveExists va lire l'entièrement du tableau et appellera isValidMove jusqu'à trouver une zone valable. Cependant, la fonction appelée a comme première vérification, que la zone séléctionnée soit assignée à la valeur 0. Dès lors, la lecture du tableau sera de plus en plus rapide plus il y aura de pions.

3 Choix de la structure du programme

Afin de permettre à l'IA de jouer contre d'autres joueurs, il a été choisit d'utiliser des fichiers dans lesquels l'ordinateur pourra lire les actions de l'adversaire. Cependant, puisqu'il y a un adversaire, il faut tenir en compte une chose, le faite que les actions de l'un ne soient jouées que si l'autre a belle et bien envoyé son action. Par exemple, tant que l'adversaire n'envoie pas d'actions valides l'IA attendra. Dès lors, le problème de fin de partie apparait puisqu'à la fin, il est possible que l'adversaire n'envoie pas de commande spécifique comme un [THE END]. Dès lors, une solution serait de vérifier à la fin de chaque tour si les deux

joueurs peuvent encore placer des pions et ce afin de s'assurer de ne pas tourner alors que l'adversaire ne peut plus jouer.

Une stratégie au jeu est d'interpréter le fait que certaines cases aient plus de "valeur" que d'autres. Pour ce faire, une matrice de poids a été préalablement définie, les cases les plus lourdes sont les plus interressantes à avoir. Ainsi, l'implémentation d'une matrice de poids permettra à l'IA de choisir ses cases judicieusement afin de gagner le plus de points sur toute une ligne.

4 Optimisation de l'IA

De base, l'algorithme utilisé pour la conception de l'intelligence artificielle est l'implémentation d'une fonction récursive appelée minimax qui permet à l'intelligence artificielle de regarder tous les choix possibles pour à la fin choisir le meilleur coup en fonction de la profondeur choisie. La fonction minimax va parcourir toutes les branches de l'arbre et pour pouvoir connaitre le chemin idéal. Cette fonction va tout d'abords parcourir une branche et va garder en mémoire la valeur du noeud si celle-ci est favorable. Sinon, elle ne va juste pas garder le noeud et passer à une autre branche.

Le problème par rapport à cette méthode, est que l'exécution dure trop longtemps. Une méthode trouvée pour régler ce problème est l'utilisation de l'alpha-bêta pruning qui a pour but de réduire le temps d'exécution en supprimant certaines branches de l'arbre. Il est prévu d'améliorer le code dans le sens d'ajouter cette fonctionnalité en plus pour garantir une plus grande chance de gagner en ajoutant dans ce cas là une plus grande profondeur, c'est-à-dire prévoir plus de coups à l'avance car l'exécution sera plus rapide.