Projet Reversi

Lorenzo MARNAT & Hugo MOHAMED

13 mai 2018

Table des matières

1 Fonctionnalités du programme

1.1 Mode de jeu

Le reversi est un jeu à 2 joueurs :
— l'un est contrôlé par le joueur humain,
— l'autre est contrôlé par l'ordinateur

Le joueur humain utilise les pions noirs, tandis que l'ordinateur utilise les pions blancs. Ces couleurs ne peuvent pas être échangés et le joueur humain commence toujours.

La partie se termine quand le plateau est plein ou si tous les pions sont d'une même couleur. Si l'un des joueurs, humain ou ordinateur, ne peut pas jouer, l'opposant joue jusqu'à ce que le premier joueur puisse de nouveau placer un pion.

1.2 Utilisation

Le jeu se compile depuis le terminal avec la commande :

```
make reversi
```

ou

```
make
```

La commande

```
make all
```

permet de compiler le jeu ainsi que le fichier latex.

Le jeu se lance ensuite avec la commande :

```
reversi -n x -p y
```

L'option -n permet d'entrer le niveau de l'intelligence artificielle, elle prend en argument une valeur de 0 à 3 selon le niveau souhaité. Cette option est **obligatoire** et le jeu s'arrêtera en cas d'absence de cette option.

L'option -p permet d'indiquer la profondeur de construction de l'arbre des possibilités de l'IA. Cette option est utilisé par l'IA de niveau 2 et l'IA de niveau 3. Cette valeur doit être strictement supérieure à zéro. Cette option n'est **pas obligatoire**, par défaut la profondeur sera de 5.

1.3 Arbre de jeu

Les niveaux 1 à 3 de l'intelligence artificielle utilisent un arbre de jeu. Cet arbre contient la configuration actuelle à la racine, chacun de ses noeuds représente un coup possible alternativement pour l'IA et pour le joueur humain. La structure du noeud est la suivante :

```
typedef struct noeud
{
    plateau position;
    int valeur\_plateau;
    int nb\_fils;
    struct noeud **tab_fils;
} noeud;
```

Elle contient les informations suivantes :

— position : contient une configuration du plateau,
— valeur_plateau : indique le nombre de points que rapporterait la configuration du plateau (par rapport à la configuration père),
— nb_fils : indique le nombre de fils du noeud,
— tab_fils : contient les adresses vers les fils du noeud.

Le module arbre permet de créer un arbre vide ou un arbre de jeu avec comme unique noeud la racine, de rajouter des fils à un arbre et enfin de vérifier si un arbre est vide ou non.

1.4 Niveaux de l'intelligence artificielle

L'intelligence artificielle possède 4 niveaux, numérotés de 0 à 3. Plus le niveau est haut, plus l'IA est performante. Les détails d'implémentation des différents niveaux suivent ci-dessous.

Niveau 0 L'IA de niveau 0 joue de manière aléatoire. À chacun de ses tours, elle choisit une position du plateau :

— si la position est valide, elle y place son pion,
— sinon, elle choisit une autre position jusqu'à en trouver une valide.

La position est généré aléatoirement par :

```
int x = rand() % 8;
int y = rand() % 8;
```

Cette IA est très peu performante pour plusieurs raisons. Tout d'abord, elle place son pion indifféremment du nombres de points que cela lui rapporte ou du nombres de points que cela pourra rapporter au joueur humain les tours suivants. De plus, le choix de la position étant totalement aléatoire, il se peut que l'IA tente plus d'une dizaine de coups avant de trouver une position valide et peut essayer plusieurs fois une même position invalide.

Niveau 1 L'IA de niveau 1 construit un arbre des possibilités de profondeur 2 : à chacun de ses tours, elle regarde tous ses coups possibles, puis les coups possibles de l'adversaire et une deuxième fois ses coups possibles. L'arbre est construit récursivement jusqu'à ce que la profondeur de 2 soit atteinte. La construction d'un sous-arbre s'arrête avant si l'IA rencontre une configuration où l'un des deux joueurs n'a plus de coups valides.

Une fois l'arbre construit, l'IA va le parcourir de ses feuilles jusqu'à la racine en utilisant l'algorithme MinMax. À chaque profondeur, elle sélectionne le fils rapportant le plus de points si c'est à son tour de placer un pion, ou le fils en rapportant le moins si c'est un coup de l'adversaire. Les valeurs sont ainsi remontés jusqu'aux fils directs de la racine parmi lesquels l'IA choisit le meilleur coup possible (celui rapportant le plus de points).

La profondeur de l'arbre ne peut pas être modifiée, si une valeur est entrée au lancement avec l'option -p, cette dernière ne sera pas prise en compte par l'IA.

Niveau 2 L'IA niveau 2 construit l'arbre de la même manière que l'IA niveau 1 et utilise le même algorithme que l'IA niveau 1. L'arbre est construit de la même manière, à la différence près que la profondeur peut en théorie varier de 1 à $+\infty$ selon la valeur entrée en argument avec l'option -p. Dans la pratique, il est préférable de rentrer une profondeur de 7 au maximum pour que le jeu soit fluide. Avec une profondeur de 3 ou plus, l'IA niveau 2 choisit ses coups bien plus efficacement que l'IA niveau 1. En effet, plus la profondeur est grande, plus l'IA choisit des coups rentables sur le long terme.

L'IA niveau 2 fonctionne donc de la même manière que l'IA niveau 1 lorsque sa profondeur est de 2. Les commandes :

`reversi -n 1`

et

`reversi -n 2 -p 2`

produisent exactement le même résultat.

Niveau 3 L'IA niveau 3 construit l'arbre de la même manière que les IA de niveau 1 et 2. Cependant, il choisit son coup plus rapidement que ses prédécesseurs, à profondeur égale. Cela est dû à un algorithme de parcours de l'arbre plus efficace : l'algorithme AlphaBeta. Cet algorithme permet d'éliminer certaines branches de l'arbre dont on sait qu'elles seront moins rentables que certaines branches déjà explorées.

L'IA niveau 3 aura tendance à choisir les mêmes coups que l'IA niveau 2 mais elle jouera plus rapidement que cette dernière.

Pondération du plateau Afin d'améliorer la qualité des coups des IA de niveau 1, 2 et 3, nous avons augmenté le nombre de points que rapportent certaines zones du plateau.

Ainsi, une case peut rapporter de 1 à 3 points dans les cas suivants :

— si c'est un coin du plateau, la case rapporte 3 points,

— si c'est un bord du plateau, la case rapporte 2 points,

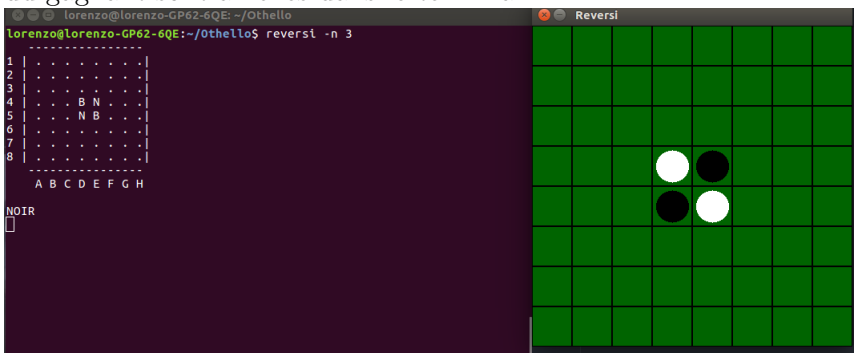
— si ce n'est ni un coin ni un bord, la case rapporte 1 point.

En effet, les bords et les coins sont des positions avantageuses pour le joueur qui y place ses pions : les pions situés sur un bord sont difficiles à capturer tandis que ceux placés dans un coin ne peuvent pas être capturés par la suite.

1.5 Affichage et placement du pion

Le jeu est affiché à l'écran grâce à la bibliothèque graphique MLV. Lorsque que c'est au tour du joueur humain, celui-ci doit cliquer sur une case de la fenêtre MLV pour placer son pion. La fenêtre est mise à jour après chaque pion noir ou blanc placé.

Le terminal affiche de son coté une version simplifiée du plateau (avec des lettres pour les pions, B pour les blancs, N pour les noirs). Cela permet de s'assurer que la fenêtre MLV est bien à jour mais aussi et surtout de pouvoir regarder les configurations précédentes du plateau. Le terminal affiche aussi tous les messages d'erreur (coup invalide, problème au lancement du jeu, etc) ainsi que la couleur du joueur à qui c'est le tour. À la fin de la partie, les scores et l'annonce du gagnant sont affichés dans le terminal.



2 Difficultés rencontrées et solutions

2.1 Construction de l'arbre

La principale difficulté a été la construction de l'arbre. L'arbre est construit récursivement, pour chaque coup possible on regarde tous les coups possibles de l'opposant jusqu'à atteindre la profondeur maximale. Du moment que les coups prévus étaient alternativement un pion du premier joueur et un pion de l'autre joueur, tout fonctionnait comme prévu. Le problème était que l'IA se retrouvait régulièrement dans des cas où elle ne pouvait pas prévoir de coup noir ou blanc car il n'y avait pas de coups valides pour cette couleur. L'IA ne pouvait donc pas prévoir plusieurs coups consécutifs d'un même joueur. Le problème était encore plus important dans un cas où le coup capturait l'entièreté des pions adversaires, on se retrouvait alors dans un cas où la construction de l'arbre était impossible.

Nous avons essayé de changer la couleur s'il n'y avait pas de coup valide pour la couleur actuelle (si on ne peut pas placer un pion d'une couleur, on place un pion de l'autre couleur) mais cela faussait une partie de la suite de cette branche de l'arbre. De plus, nos algorithmes de choix du coup se basent sur la profondeur pour savoir si on est dans un cas où on doit choisir la valeur minimale ou maximale (si la profondeur est paire on est dans un cas Max, si elle est impaire on est dans un cas Min). Les différentes solutions envisagées, tel que indiquer dans la structure du noeud si c'est un noeud de type Max ou Min, ralentissaient grandement la vitesse de construction et de parcours de l'arbre de jeu.

Ainsi, nous avons choisi de privilégier la fluidité du programme en arrêtant la construction des branches pour lesquelles on arrivait dans un cas où un joueur était bloqué pour un ou plusieurs tours. Cela nous permet de ne pas avoir a géré le type de noeud sur lequel on se trouve (Min ou Max), on sait directement de quel type il est et de quel type sont tous les noeuds du même niveau grâce à la profondeur. Ce choix de construction, en plus d'améliorer la vitesse de jeu de l'IA, ne fausse que de manière très infime la qualité des coups de l'IA. En effet, on peut facilement observer que :

— si le joueur humain n'a aucun coup possible, c'est souvent que le coup de l'IA précédent a apporté un grand nombre de points, ce coup aura donc une grande chance d'être choisit,
— si l'IA n'a aucun coup possible, c'est souvent que le coup du joueur lui a rapporté un grand nombre de points, ce coup sera donc pratiquement jamais choisit par l'IA.

En conclusion, même si les branches où un joueur se retrouve bloqué ne sont pas complètes, cela ne fausse pratiquement pas les choix de l'IA : un coup où l'IA bloque le joueur humain aura de grandes chances d'être choisi et un coup bloquant l'IA ne sera pratiquement jamais choisit.

2.2 Qualité du choix de coup de l'IA

Un autre problème handicapant grandement l'IA était la priorisation des positions sur le plateau. L'IA choisissait son coup indifféremment de l'aspect stratégique d'une position, elle choisissait la position avec le meilleur compromis entre le nombre de points que cela lui rapporte et le nombre de points rapportés à l'adversaire. Hors, un joueur aguerri de Reversi sait que

certaines positions, tels que les coins et les bords, sont des positions très importantes à capturer, même si cela n'apporte que peu de points sur le coup.

Ainsi, nous avons pondéré les coins et les bords : dans le calcul de la qualité d'une position, les coins rapportent 3 points et les bords 2 points, là où les autres cases du plateau rapportent 1 point. De cette manière, l'IA capturera presque toujours les coins quand elle le peut et privilégiera les bords du plateau aux autres cases.

3 Améliorations possibles

3.1 Gestion de la mémoire

Lors de la construction de l'arbre, chaque noeud va contenir un état du plateau, soit un tableau d'entiers de 8 par 8. Cela a pour conséquence que l'arbre prend une très grande place en mémoire. Ainsi, quand la profondeur dépasse 7, le jeu est très lent et il arrive que des problèmes de mémoire soient rencontrés.

Il serait donc envisageable d'optimiser le stockage des états du plateau dans l'arbre. Par exemple, on pourrait ne stocker dans le noeud que les coordonnées du pion rajouté et ainsi de passer de 64 à 2 entiers indiquant l'état du plateau. En réduisant la mémoire prise par l'arbre, on pourrait construire des arbres avec une profondeur plus importante, améliorant en même temps la qualité des coups de l'IA.

3.2 Modes de jeu

Actuellement, le jeu ne comporte pas de mode permettant de faire des parties à deux joueurs humains. Un mode joueur contre joueur serait tout à fait envisageable.

Aussi, dans certaines versions de l'Othello, il existe un mode à plus de 2 joueurs (de 2 à 4). Une version où il est possible de jouer à 3 ou 4, avec potentiellement plusieurs IA, serait intéressante. Fondamentalement, les algorithmes utilisés ne seraient pas si différents de ceux utilisés dans cette version.

4 Annexe : répartition du travail

Le groupe est composé de Lorenzo MARNAT et de Hugo MOHAMED. La répartition des tâches a été la suivante :

Lorenzo

- Mise en place des règles de base du jeu : capture des pions, validité d'une position et calcul des points,
- Création des IA de niveau 1 et 2,
- Rédaction du rapport

Hugo

- Création des IA de niveau 0 et 3,
- Mise en place de l'aspect graphique avec MLV : affichage du plateau, prise en compte du clic du joueur,
- Mise en page du code : commentaires, clarté...

La répartition du travail c'est faite naturellement en fonction de nos facilités et de nos disponibilités. Nous avons l'habitude de travailler ensemble et connaissons nos forces.

Le fait d'être à deux nous a permis de nous répartir le travail : pendant que l'un faisait un niveau de l'IA, l'autre faisait un autre niveau ou améliorait les autres aspects du jeu. Nous avions décidé avant de ce que l'on allait faire, ainsi nous savions où nous allions et étions assuré que nos créations respectives seraient compatibles les unes les autres. Nous avons donc pu avancer deux fois plus vite que si le projet avait été fait seul. De plus, être à deux nous a permis de nous entraider et de corriger les erreurs de l'autre si on en trouvait. Enfin, le temps gagner de cette manière a pu être utilisé pour les autres projets et les révisions.