PDP: Reversi
Cahier des besoins

Chemoune Alaeddine Delengaigne Hadrien
Pivetaud Victor Schlick Charlie

2 février 2018

1 Introduction

Le *Reversi* est un jeu de société qui se joue à deux joueurs, avec des pions noirs et blancs, sur un plateau composé de 64 cases. Ce jeu est apparu en 1883 en Angleterre sous le nom de *Reversi*, mais son invention reste contestée. La version moderne est connue sous le nom d'*Othello* et a été brevetée en 1971 au Japon par *Goro Hasegawa* et diffère légèrement de son ancêtre, notamment par le placement des pions en début de partie[1](les joueurs plaçaient chacun à leur tour les pions de départ sur les 4 cases centrales, ce qui pouvait aboutir à la même configuration que l'Othello ou à une configuration avec les pions blanc et noir en parallèle). Depuis 1977 un championnat du monde est organisé chaque année. Les sommaires règles de base offrent la possibilité aux meilleurs joueurs de prévoir plusieurs coups à l'avance, ce qui permet de développer des stratégies complexes.

Au début d'une partie, quatre pions sont posés au centre du plateau, deux faces blanches et deux faces noires en diagonale. Le but du jeu est d'avoir le plus de pions de sa couleur. Pour prendre des pièces à son adversaire, il faut mettre ses pions aux extrémités d'une ligne de pions adverses. Chaque joueur pose donc un nouveau pion sur le plateau à chaque tour. Seuls les coups qui permettent de prendre au moins une pièce à l'adversaire sont autorisés. La partie se termine lorsque le plateau est plein ou lorsque plus aucun coup n'est jouable, le gagnant est alors le joueur qui possède le plus de pions de sa couleur sur le plateau.

2 Description et analyse de l'existant

De nombreux programmes de *Reversi* ont été développés et déjà avant la défaite du champion Takeshi Murakami contre le programme Logistello en 1997(avec un score de 6-0) on pouvait affirmer que les programmes étaient meilleurs que les humains à ce jeu.La grande problématique de recherche sur les jeux de plateau est de parvenir à les résoudre. On génère un arbre dont les sommets représentent les états du jeu et les chemins des coups possibles (il est aussi nécessaire de noter qui doit jouer dans le cas où un des joueurs est forcé de passer son tour). On va ainsi pouvoir représenter tous les enchaînements de coups. Résoudre un jeu signifie que l'on est capable d'aller au bout de l'arbre, ainsi on est toujours capable de contrer l'adversaire. Cela ne veut pas dire que l'on a parcouru tous les sommets, on peut simplifier l'arbre représentant tous les coups (par exemple dans le Reversi les 4 possibles sont symétriques, la position choisie n'importe pas et on peut donc ne regarder qu'un seul des 4 sous-arbres).

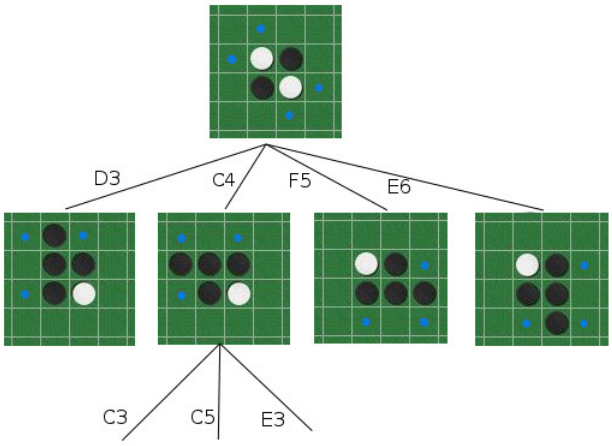


FIGURE 1 – Exemple d'arbre que l'on peut générer. A partir de la position de départ du jeu, les sommets de l'arbre sont les états et les chemins des coups possibles.

Même si le *Reversi* sur un plateau de 8x8 n'a pas encore été résolu, des solutions ont été trouvées pour des parties en 4x4 et 6x6[2].

Afin de trouver rapidement le meilleur coup (celui qui donne le plus de chance de gagner la partie), les programmes ont besoin dans un premier

temps d'une technique de parcours d'arbre efficace, mais aussi d'heuristique. De nombreuses techniques de parcours existent, parmi les plus utilisées on peut citer :

— *Minimax Alpha-beta pruning*[3], le grand avantage de cette méthode est qu'elle élimine des branches au fur et à mesure de son déroulement. Elle est donc assez optimisée pour les arbres représentants les coups possibles d'un jeu de plateau.

— *Negamax* Une variante simplifiée de minimax, lors du calcul de la perte maximum on part du principe que la valeur d'un coup pour un joueur donné est égale à la négation de la valeur de l'autre joueur. Ainsi au lieu de calculer le minimum et maximum pour chaque joueur et chaque coup on dit que $max(a, b) == -min(-a, -b)$.

Comme dit précédemment, on ne sait pas, à l'heure actuelle, aller jusqu'au bout de l'arbre des coups possibles, c'est pour cette raison que nous avons besoin d'une fonction d'évaluation. Le but de cette fonction est d'attribuer une valeur au sommet représentant les états du jeu. Plusieurs techniques sont possibles[4].

— IAGO : Cette méthode consiste à utiliser les connaissances que l'on a du jeu de Reversi[5], en effet on peut dégager des critères augmentant les chances de gagner. Par exemple les bords et les coins sont des positions très intéressantes à avoir car elles sont difficiles ou impossibles à retourner. En réduisant le nombre de coups possibles pour l'adversaire on peut aussi l'obliger à jouer des coups nous favorisant (par exemple nous permettre de prendre possession d'un coin). En étant capable d'extraire ces informations on peut évaluer chaque coup.

— LOGISTELLO : La méthode utilisée par le programme ayant battu le champion du monde se base sur des parties jouées. En faisant analyser les coups joués et le résultat de la partie à un ordinateur on peut obtenir une heuristique basée sur des parties réelles plutôt que deviner les éléments importants pour gagner.

Enfin, afin de représenter le plateau d'une façon permettant des manipulations rapides, une structure spécifique a été inventée : les Bitboards[6]. Cette structure de données est un tableau de bit représentant la présence ou non des pions sur la case correspondante. Le principal avantage de cette stucture est qu'elle permet la manipulation du plateau par des opérations bit à bit, opérations pouvant être effectuées très rapidement par les processeurs. Le deuxième avantage est que cette information prend très peu de place en mémoire. Dans le cas du Reversi, le plateau peut être représenté par deux

bitboards, un codant les positions des pions blancs et l'autre les positions des pions noirs.

3 Description des besoins

3.1 Fonctionnels

3.1.1 Liste des besoins liés à la simulation de jeu :

Déterminer les coups possibles à jouer : Dans les règles du Reversi[7], peu ou aucun coups sont possibles à chaque tour, nous devons donc pouvoir les déterminer pour respecter les règles du jeu et pour savoir si le coup qui est demandé est légal ou non.

Actualiser le plateau après un coup : Lorsque l'on rajoute un pion au plateau il faut changer la couleur des pions qui ont été capturés avant de changer de joueur.

Reconnaître une fin de partie : Pour savoir quand arrêter le jeu nous devons reconnaître une fin de partie. Il n'y a soit plus de case libre soit plus de coups légaux disponibles pour les deux joueurs.

3.1.2 Liste des besoins liés à l'interface utilisateur :

Démarrer une partie : Au lancement du programme il faut qu'une partie se lance selon les paramètres passés par l'utilisateur, ces options permettront de :

— spécifier la taille
— reprendre une partie sauvegardée dans un fichier texte
— faire jouer les blancs et/ou noir par une IA
— lancer le mode contest
— afficher l'aide

Afficher l'état de la partie sur la console : Il faut pouvoir afficher le plateau de jeu quelle que soit sa taille, en dessous d'une certaine limite, et d'une manière qui soit compréhensible par le joueur.

Déterminer les coups possibles à jouer et les afficher : Il faut pouvoir prévoir les coups légaux qui permettent de prendre des pions à l'adversaire et les rajouter sur l'affichage, pour que le joueur puisse savoir où il a le droit de jouer.

Choisir un coup : Le joueur doit pouvoir placer un pion, ce qui aura pour effet si le coup est légal, d'actualiser l'état de la partie, ou le cas échéant, de redemander une position au joueur.

Finir une partie : La partie se terminera par un affichage des scores et la possibilité de recommencer une partie, ou d'arrêter le programme sera proposée au joueur.

Sauvegarder une partie dans un fichier texte : Si l'on souhaite continuer la partie plus tard, nous devons être capable de sauvegarder l'état actuel dans un simple fichier texte avant de fermer le programme. L'écriture de ce fichier doit respecter la syntaxe fixée par le client (Voir figure 2) qui sera expliquée plus bas.

Charger une partie à partir d'un fichier de sauvegarde : Une fois qu'une partie est sauvegardée, si l'on souhaite la continuer nous devons pouvoir placer en entrée de notre programme le nom d'un fichier de sauvegarde qui respecte la syntaxe (Voir figure 2) et continuer la partie à partir de l'état de jeu actuel.

Respecter la syntaxe du fichier de sauvegarde (voir 2) pour que le programme arbitre du client puisse l'utiliser :

- Les lignes vides, espaces et tabulation ne sont pas pris en compte, seuls les lignes non vides sont significatives.
- La première ligne significative indique le joueur courant('X' pour le joueur noir et 'O' pour le joueur blanc).
- Chaque lignes significatives qui suit représente une ligne du plateau, avec pour chaque case,'_' si la case est vide, 'X' si il y a un pion noir et 'O' si il y a un pion blanc.
- Chaque fin de ligne est défini par un '\n'.
- Le caractère '#' indique un commentaire jusqu'au prochain '\n'
- Les lignes ne contenant que des commentaires sont donc des lignes considérées vides.
- Il faut respecter des tailles carrées et paires tels que 2x2, 4x4, 6x6, 8x8...

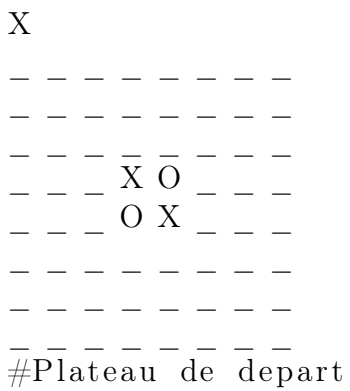


FIGURE 2 – Exemple fichier sauvegarde

Choisir la taille du plateau : On doit pouvoir choisir la taille du plateau de jeu, il faut cependant que le plateau de jeu soit un carré de nombre pair pour que la partie soit jouable.

Avoir un mode "Contest" : Ce mode permettra au client d'utiliser notre intelligence artificielle au cours d'une partie. Il faut pouvoir prendre en entrée une partie en cours et afficher sur le terminal le coup que l'on souhaite jouer. Le client utilisera lui son programme arbitre pour faire jouer notre intelligence artificielle.

3.1.3 Liste des besoins liés à l'intelligence artificielle :

Pour réaliser notre intelligence artificielle, nous utiliserons des algorithmes de parcours d'arbres[8] qui étudieront les différents coups possibles afin de choisir le meilleur. Il est donc nécessaire d'implémenter les besoins suivants :

Créer les fils d'un nœud Chaque nœud de l'arbre se composera d'une configuration de jeu et du joueur qui devra jouer. Car une configuration de jeu seule ne permet pas de savoir quel joueur vient de jouer. Nous devons donc à partir d'un noeud, pouvoir déterminer tous ses fils (l'ensemble des coups qui suivront la configuration actuelle).

Fixer une profondeur limite à notre arbre Il est impossible de générer entièrement l'arbre des coups possibles du Reversi, il est donc necessaire de fixer une limite à la profondeur de notre arbre si l'on souhaite le générer.

Fusionner les branches identiques Des coups différents peuvent parfois mener à des configurations de jeux identiques ou symmétriques. Afin de limiter la taille de notre arbre, on peut les fusionner lorsqu'on les détecte.

Parcourir notre arbre des coups Différentes techniques de parcours d'arbre existent, certaines ont besoin d'un arbre déjà créé (le Minimax[9] par exemple), d'autres le construisent au fur et à mesure (alpha beta prunning[3] par exemple) afin d'éviter de créer les noeuds que l'on a pas besoin d'étudier. Nous devons les étudier pour trouver la performante.

Attribuer une valeur à une configuration de jeu Lorsque l'on parcourt notre arbre, nous devons attribuer à chaque état du jeu une valeur afin de pouvoir séléctionner le meilleur coup. Là encore, différentes techniques existent. Par exemple, le nombre de pions que l'on possède après un coup ou le nombre de coup différent que l'on pourra faire pour avoir plus de choix ou encore, le nombre de pions stables (qui ne peuvent pas être repris par le joueur adverse). Il nous faudra donc en implémenter plusieurs afin de les comparer et choisir la meilleure ou d'en utiliser plusieurs.

3.2 Non fonctionnels

- Utilisation de deux bitboards[10] pour représenter les pions noirs et blancs
- Actualisation de l'affichage en moins d'une seconde
- IA suffisamment rapide (10 secondes sur les ordinateurs du Crémi)

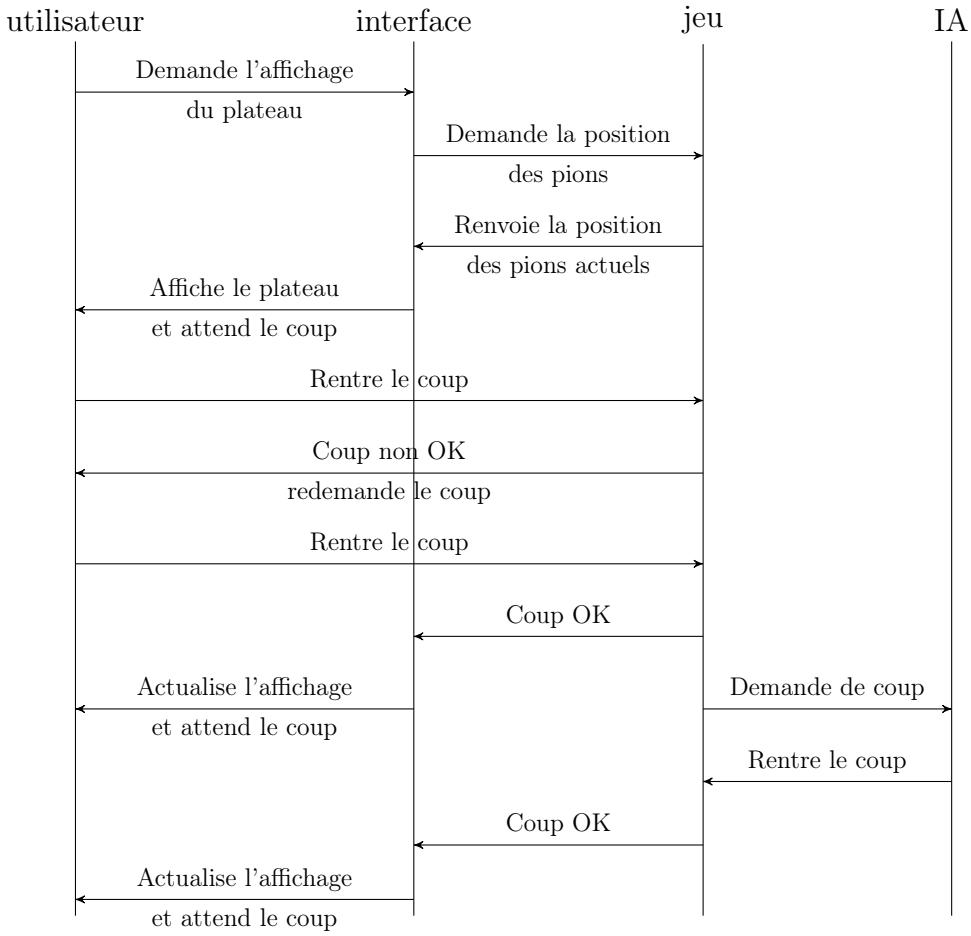
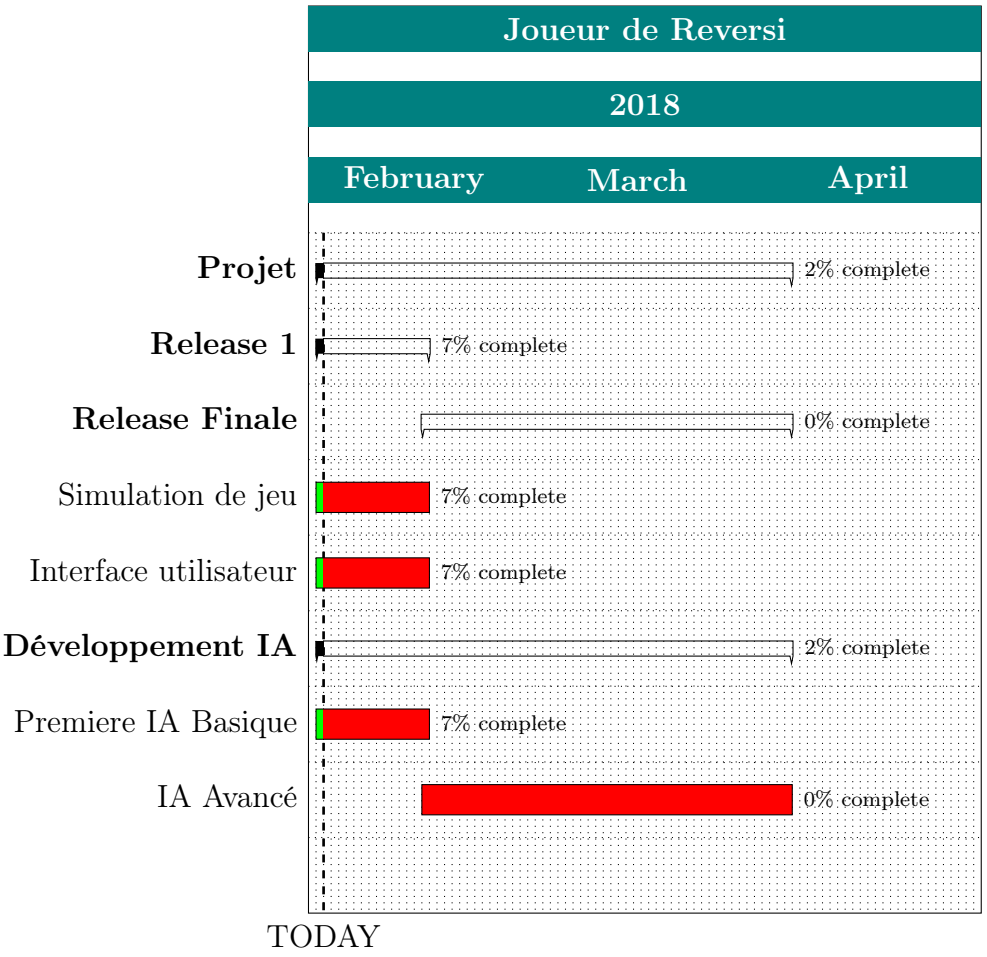


FIGURE 3 – Diagramme de séquence du lancement du lancement du programme

4 Diagramme de Gant



5 Bibliographie

Références

[1] Ted Landau. Othello : Brief & Basic. 1990.

[2] Joel Feinstein. Perfect play in othello from two alternative starting positions. `https://web.archive.org/web/20091101013931/http://www.feinst.demon.co.uk/Othello/6x6sol.html`. [archive date : NOV 01 2009].

[3] Edwards, J D, Hart, and TP. The Alpha-Beta Heuristic. 1963.

[4] Michael Buro. The evolution of strong othello programs. pages 81–88, 2003.

[5] Fédération Française d'Othello. Principes stratégiques. http://www.ffothello.org/othello/principes-strategiques/. [access date january 2018].

[6] Cameron Browne. Bitboard methods for games. 37 :67–84, 06 2014.

[7] Le comptoir des jeux. Règle du reversi. http://www.lecomptoirdesjeux.com/regle-reversi.htm. [access date january 2018].

[8] sensAgent. Définition - arbre (informatique). http://dictionnaire.sensagent.leparisien.fr/Arbre%20(informatique)/fr-fr/. [access date january 2018].

[9] baeldung. Introduction to minimax algorithm. http://www.baeldung.com/java-minimax-algorithm. [access date january 2018].

[10] Franck Zibi. An introduction to bitboards. http://www.fzibi.com/cchess/bitboards.htm. [access date january 2018].