université de BORDEAUX

Master CSI 2024-2025

Rapport de Projet

Reversi

U.E. : Programmation en C

Réalisé par :
Sadick Youssouf Charaf

Enseignant responsable de l'U.E :
Emmanuel Fleury

Date : 18 février 2026

Table des matières

Introduction

Reversi, également connu sous le nom d'Othello, est un jeu de plateau stratégique où les joueurs tentent de capturer le plus grand nombre possible de pions adverses en les encadrant. Ce projet propose une implémentation complète de ce jeu en langage C, avec des fonctionnalités modernes et une intelligence artificielle avancée. Dans notre version, les pions noirs et blancs sont représentés respectivement par les caractères 'X' appelé aussi ($BLACK\_DISC$) et 'O' ($WHITE\_DISC$), et le plateau est modélisé sous forme de matrice carrée. Les joueurs peuvent choisir la taille du plateau au démarrage (par défaut : 8x8). Les cases contenant un '*' ($HINT\_DISC$) indiquent les coups possibles pour le joueur actif, rendant le jeu plus intuitif. L'implémentation inclut une IA basée sur les algorithmes Minimax et Minimax alpha-bêta, offrant différents niveaux de difficulté. Ces algorithmes permettent à l'IA de prendre des décisions stratégiques et de jouer efficacement contre un adversaire humain. Ce rapport présente les améliorations algorithmiques, les détails techniques de l'implémentation et les efforts fournis pour assurer la qualité, la modularité et la robustesse du code. Le projet a été réalisé dans le cadre du cours de Programmation C, sous la direction de M. Emmanuel Fleury, à l'Université de Bordeaux.

1 Améliorations Algorithmiques

1.1 Algorithme Minimax

L'algorithme Minimax est une méthode utilisée pour prendre des décisions dans les jeux à information complète, en particulier dans les jeux de stratégie comme Reversi. L'objectif de cet algorithme est d'explorer toutes les possibilités de jeu et de choisir le coup optimal en maximisant le score du joueur tout en minimisant celui de l'adversaire.

L'algorithme fonctionne selon un principe récursif, en générant un arbre de décisions où chaque nœud représente un état du jeu et chaque branche représente un coup possible. Le joueur essaye de maximiser son score tandis que l'adversaire cherche à minimiser le score du joueur. La fonction `minimax()` implémente cet algorithme en explorant récursivement toutes les branches de l'arbre de décisions jusqu'à une profondeur maximale définie par la constante `MAX_DEPTH`. Cette profondeur limite le temps de calcul pour éviter des évaluations trop longues et coûteuses.

La fonction `score_heuristic()` permet d'évaluer chaque état du jeu en retournant la différence entre les scores des pions du joueur actuel et ceux de l'adversaire. Cette évaluation est utilisée pour guider l'algorithme vers les états les plus avantageux pour le joueur. Plus précisément :

- Si le joueur est le `BLACK_DISC`, l'heuristique retourne la différence entre le nombre de pions noirs et blancs sur le plateau.
- Si le joueur est le `WHITE_DISC`, l'heuristique retourne la différence entre le nombre de pions blancs et noirs.

1.2 Optimisation avec Élagage Alpha-Bêta

L'algorithme Minimax, bien que puissant, peut devenir coûteux en termes de temps de calcul lorsqu'il explore un grand nombre de possibilités. L'élagage Alpha-Bêta est une optimisation de cet algorithme qui permet de réduire considérablement le nombre de nœuds à explorer en coupant certaines branches inutiles.

Dans cette version optimisée, deux variables *alpha* et *beta* sont utilisées pour maintenir les meilleurs scores trouvés pour les joueurs Maximiseur (l'IA) et Minimiseur (l'adversaire)

respectivement. L'algorithme `minimax_ab()` implémente cette optimisation en explorant les coups possibles tout en maintenant les valeurs de *alpha* et *beta*. Si, à un moment donné, il devient évident qu'un certain nœud ne pourra pas améliorer les résultats de l'IA (ou de l'adversaire), cette branche est abandonnée, réduisant ainsi le nombre de coups évalués.

1.3 Optimisation des Stratégies de l'IA

Plusieurs optimisations ont été introduites pour améliorer les performances de l'IA et rendre ses décisions plus intelligentes et plus efficaces. Bien que certains points sont très importants dans le jeu de reversi, mais malheureusement, ils ne sont pas tous pris en compte dans l'implémentation actuelle. certains points j'ai essayé mais je n'ai pas réussi à les implémenter correctement.

- **Prise en compte stratégique des positions :** Les coins et les bords du plateau sont essentiels dans le jeu de Reversi car ils offrent un meilleur contrôle sur l'espace de jeu. L'IA favorise ces positions dans ses évaluations heuristiques pour s'assurer de leur domination.
- **Gestion adaptative de la profondeur :** L'IA explore moins profondément les branches en début de partie afin d'économiser des ressources. En fin de partie, lorsque le nombre de coups possibles est plus réduit, la profondeur de l'exploration est augmentée pour maximiser les chances de victoire.
- **Stratégie de mobilité :** L'IA privilégie les coups qui augmentent le nombre de mouvements possibles dans les tours suivants. Cela lui permet de conserver une certaine flexibilité et d'empêcher l'adversaire de bloquer rapidement ses options.
- **Réduction des explorations inutiles :** En éliminant rapidement les branches d'arbre inutiles grâce à l'élagage alpha-bêta, l'IA économise du temps de calcul et se concentre uniquement sur les coups qui peuvent réellement influencer le résultat final.

1.4 Algorithme Minimax et Élaboration Alpha-Bêta

L'algorithme **Minimax**, combiné à l'élagage alpha-bêta, aide l'IA à prendre des décisions plus rapidement en réduisant le nombre de coups à explorer. En élaguant certaines branches de l'arbre de décision, l'algorithme se concentre uniquement sur les coups les plus prometteurs. Cela permet à l'IA de choisir le meilleur coup en explorant moins de possibilités, tout en maintenant la qualité du jeu.

2 Implémentation Technique

2.1 Structures de Données

Pour une manipulation efficace des données, le projet utilise les structures suivantes :

- **Bitboards :** Représentation compacte du plateau avec des opérations rapides pour calculer les mouvements et les captures. Les **Bitboards** ont été choisis pour une représentation efficace des plateaux, permettant des opérations logiques rapides (AND, OR, XOR, ...) pour vérifier les coups et calculer les scores
- **Structures personnalisées :**
  - struct board_t : Représente l'état du plateau, le joueur actif, et les bitboards pour les pions noirs et blancs.
  - struct move_t : Modèle un mouvement avec une ligne et une colonne.
  - struct score_t : Stocke les scores des pions noirs et blancs.
  - ...

2.2 Structure du Code

Le projet suit une approche modulaire avec les fichiers principaux suivants :
— **board.c :** Gestion du plateau (création, affichage, les mouvements valides, ...).
— **player.c :** Logique des joueurs, y compris IA et humain.
— **reversi.c :** Boucle principale du jeu, configuration des joueurs et gestion des parties.

2.3 Optimisations

— **Optimisation mémoire :** Allocation dynamique limitée aux besoins du plateau.
— **Pré-calculs :** Scores et mouvements possibles pré-calculés pour améliorer les performances.
— **Élagage alpha-bêta :** Réduction du nombre de branches explorées lors de la recherche de l'IA.

2.4 Exemple de Fonction

Voici un extrait de la fonction d'évaluation du score :

```
int score_heuristic(const board_t *board, disc_t player) {
    score_t score = board_score(board);
    return (player == BLACK_DISC) ? score.black - score.white : score.white -
}
```

Cette fonction calcule une heuristique de score pour le joueur donné. Elle utilise la fonction `board_score()` pour obtenir les scores actuels des pions noirs et blancs sur le plateau, puis retourne la différence entre ces scores en fonction du joueur actuel.

3 Qualité, Modularité et Robustesse du Code

3.1 Tests

Des tests exhaustifs ont été réalisés :
— **Tests unitaires :** Vérification des fonctions individuelles (validation des mouvements, calcul des scores, etc.).
— **Tests d'intégration :** Validation des interactions entre les modules.
— **Cas limites :** Tests sur des configurations extrêmes comme un plateau presque plein ou des mouvements impossibles.

3.2 Outils Utilisés

— **Valgrind :** Détection et résolution des fuites de mémoire.
— **Gprof :** Analyse des performances pour identifier les fonctions les plus coûteuses en temps.
— time : Mesure du temps d'exécution pour évaluer les performances. Réduction des explorées par l'élagage alpha-bêta, ce qui améliore les performances de l'IA.
— ...

3.3 Robustesse et Modularité

Le code a été conçu pour être robuste et modulaire :
— Validation des entrées utilisateur pour éviter les erreurs.

— Fonctionnalités séparées en modules indépendants pour une maintenance facile.

4 Conclusion

Les améliorations apportées au jeu Reversi incluent des optimisations algorithmiques, une meilleure modularité et une attention particulière à la qualité et aux performances du code. Le projet Reversi implémente une IA efficace et un jeu interactif grâce à des optimisations algorithmiques et des pratiques de développement robustes. Ce projet met en avant l'importance de la modularité, des tests rigoureux et de l'amélioration continue des performances pour obtenir une application fiable et compétitive.

Les principales réalisations incluent :

— Une IA compétitive utilisant Minimax et Alpha-Bêta.

— Une structure de données efficace basée sur des bitboards.

— Une qualité de code assurée par des tests rigoureux et des outils professionnels.

Des améliorations futures pourraient inclure une interface graphique et des niveaux de difficulté personnalisables.

Prérequis et Utilisation du jeu reversi

Prérequis

— Un compilateur C.
— Un terminal pour exécuter les commandes.

Installation

1. Cloner le dépôt Git :

   ```
   git clone git@gitlab.emi.u-bordeaux.fr:prog-2024/sadick-youssouf_charaf/reversi.git
   ```

2. Se rendre dans le répertoire du projet :

   ```
   cd reversi
   ```

3. Compiler le projet avec la commande :

   ```
   make
   ```

Utilisation

— Lancer l'exécutable pour jouer en mode console :
```
./reversi
```

— Afficher l'aide pour connaître les options disponibles :
```
./reversi -h
```

ou
```
./reversi --help
```

— Lancer l'exécutable avec une taille de plateau personnalisée (8 par défaut) la taille est comprise entre 1 et 5, car elle est doublée :
```
./reversi --size 3
```

— Configurer les joueurs IA :
  — Par exemple, pour que le joueur noir utilise une IA aléatoire :
  ```
  ./reversi --black-ai 1
  ```

  — Pour que le joueur blanc utilise l'algorithme Minimax :
  ```
  ./reversi --white-ai 2
  ```

— Modifier le délai de réflexion de l'IA (par défaut 5 secondes) :
```
./reversi --timeout 10
```

Exemple de Jeu

```
  A B C D E F G H
1 _ _ _ _ _ _ _ _
2 _ _ _ _ _ _ _ _
3 _ _ _ * _ _ _ _
4 _ _ * O X _ _ _
5 _ _ _ X O * _ _
6 _ _ _ _ * _ _ _
7 _ _ _ _ _ _ _ _
8 _ _ _ _ _ _ _ _
```

— Entrez une position valide présenté par '*' sur le plateau par exemple comme `D3` ou `d3` pour jouer.
— Entrez `q` ou `Q` pour quitter la partie.
— Lors de la sortie, vous pouvez sauvegarder la partie en répondant `y` (oui) ou `n` (non) lorsque cela vous est demandé.

Annexes

Image du jeu Reversi ou Othello dans la vie réelle

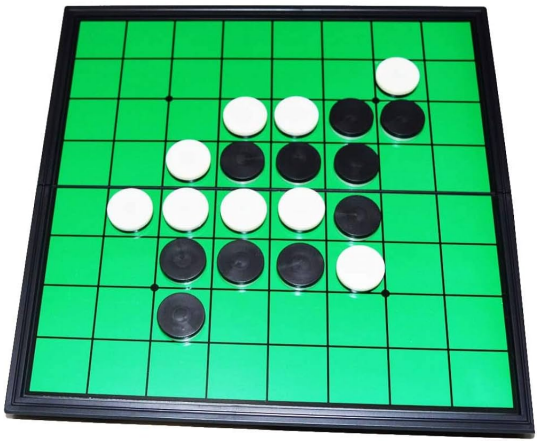

FIGURE 1 – Image représentant le jeu Reversi ou Othello dans la vie réelle.

Cette image trouvée sur : [3].

Références

Références

[1] **Wikipedia - Othello** : `https://fr.wikipedia.org/wiki/Othello_(jeu)`.

[2] **Site personnel d'Emmanuel Fleury** : `https://www.labri.fr/perso/fleury/courses/c_programming/pages/assignments.html`.

[3] **Othello - Jeu sur Google** : `https://tinyurl.com/35p79tye`.