INFO-H-304 2023-2024

Compléments de Programmation et d'Algorithmique

Projet

Reversi / Othello

1 Introduction

Le jeu de Reversi, connu sous le nom d'Othello dans sa version commerciale, est un jeu de plateau opposant deux joueurs, nommés "noir" et "blanc" [wik(2017)]. Le plateau de jeu est constitué d'une grille carrée de 64 cases (8 sur 8), sur lesquels les joueurs peuvent disposer des pions bicolores, noirs d'un côté et blancs de l'autre. La position de départ du plateau est illustrée à la figure 1.

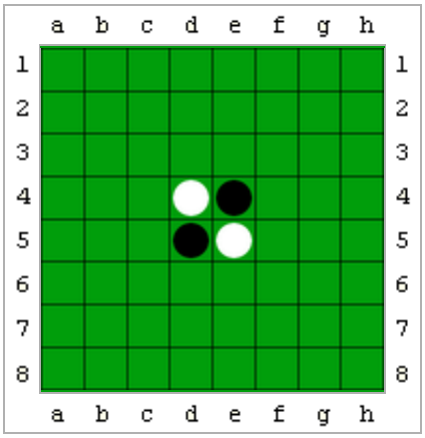

Figure 1 : Position de départ du jeu Reversi / Othello.

A partir de là, chaque joueur, **en commençant par le noir**, pose l'un après l'autre un pion de sa couleur sur le plateau de sorte à capturer au moins un pion de la couleur de son adversaire.

La capture de pions survient lorsqu'un joueur place un de ses pions à l'extrémité d'un alignement de pions adverses contigus et dont l'autre extrémité est déjà occupée par un de ses propres pions. Les alignements considérés peuvent être une colonne, une ligne, ou une diagonale. Si le pion nouvellement placé vient fermer plusieurs alignements, il capture tous les pions adverses des lignes ainsi fermées. La capture se traduit par le retournement des pions capturés. Ces retournements n'entraînent pas d'effet de capture en cascade : seul le pion nouvellement posé est pris en compte.

Lors de chaque tour, si un joueur ne peut pas capturer de pion adverse, quelle que soit la position où il placerait son pion, il doit passer son tour. Si aucun des deux joueurs ne peut capturer de nouveaux pions, la partie se termine. Le joueur ayant le plus grand nombre de pions de sa couleur sur le plateau remporte alors la partie.

```
  a b c d e f g h
1 . . . . . . . . 1
2 . . . . . . . . 2
3 . . . . . . . . 3
4 . . . O X . . . 4
5 . . . X O . . . 5
6 . . . . . . . . 6
7 . . . . . . . . 7
8 . . . . . . . . 8
  a b c d e f g h
```

Figure 2 : Représentation en ASCII de la position de départ (`O`=blanc, `X`=noir).

2 But du projet et cahier des charges

Le but principal de ce projet est d'écrire un programme permettant de jouer au Reversi. Plus précisément, le programme devrait satisfaire le cahier des charges suivant :

Langage	— C++ et sa *Standard Template Library* — Aucune utilisation de paquets tiers, si ce n'est éventuellement pour l'interface graphique (voir plus bas) — Même si une version avec interface graphique est fournie, il devrait aussi être possible de compiler le programme sans aucun paquet tiers (et donc le cas échéant sans interface graphique) — Pour la version optionnelle avec interface graphique, veuillez préciser les éventuels paquets à installer dans la machine virtuelle du cours pour pouvoir la compiler

Makefile	— Présence d'un Makefile à la racine de votre dépôt git pour faciliter la compilation des différentes version de votre projet :
	  — `projetprelim` : version préliminaire
	  — `projet` : version finale standard, n'utilisant aucun paquet tiers
	  — (optionnel) `projetgraph` : version finale avec interface graphique
	— Le Makefile devrait donc contenir une cible par version du projet, de sorte que les commandes `make projetprelim`, `make projet` et `make projetgraph` génèrent les exécutables correspondant à la racine de votre dépôt git.

Remise du code	— Le code source du projet devra être remis via un dépôt git créé dès le début du projet et mis à jour tout au long de son développement — Le code considéré comme soumis sera celui présent sur le dépôt git au moment de l'échéance (intermédiaire ou finale)
Fonctionnalités	— Après son lancement avec la commande `./projet`, le programme commence par offrir trois options pour chaque joueur : joueur humain (H), intelligence artificielle (A) ou fichier (F) — Le programme exécute ensuite la partie entre les deux joueurs suivant les modalités décrites ci-dessous, qui dépendent du type de joueur — Pour l'intelligence artificielle, une limite de 20 secondes doit être respectée pour le calcul de chaque mouvement
Rapport	— En plus de votre programme, vous devrez également fournir via le dépôt git un court rapport expliquant le fonctionnement de votre programme, y compris les éléments suivants : — Description de la structure du programme en classes, fonctions, etc. (il est également conseillé d'utiliser des commentaires directement dans le code pour le rendre plus lisible) — Éléments de réflexion et choix en résultant pour votre structure de données et votre algorithme

3 Dépôt git du projet

La première opération à effectuer pour commencer à travailler sur le projet est de forker et cloner le dépôt git du projet en suivant les instructions mentionnées dans le README de ce dépôt. Ce dépôt sera utilisé par tous les membres du groupe de projet tout au long de son développement. Veuillez dès lors faire des commits réguliers au fur et à mesure de votre travail, la bonne utilisation

de ce dépôt git faisant partie des critères d'évaluation du projet. Une guidance sera organisée pour vous aider à vous lancer dans l'utilisation de git.

4 Interface

Dans sa version de base (`projet`), l'interface devrait utiliser uniquement le terminal. En vue de représenter le plateau de jeu dans le terminal, la solution la plus simple est sans doute d'utiliser une représentation ASCII comme illustré à la Figure 2, d'autres variantes étant acceptables tant qu'elles sont lisibles.

De manière optionnelle, vous pouvez également fournir, en plus de la version de base, une version avec interface graphique. Cela peut se limiter à une représentation graphique de tableau de jeu dans une fenêtre externe, l'interaction avec le programme se faisant toujours via la terminal, ou une interface graphique interactive où un peut par exemple directement cliquer sur le plateau de jeu pour indiquer ses mouvements.

N'oubliez pas d'indiquer explicitement les éventuels paquets supplémentaires à installer pour pouvoir compiler votre programme avec interface graphique sur la machine virtuelle (la version de base devrait compiler sans installer quoi que ce soit).

5 Types de joueurs

Dans la description qui suit, nous allons distinguer les joueurs et l'utilisateur du programme. L'utilisateur du programme est la personne qui interagit avec celui-ci en entrant des commandes dans le terminal et en recevant des informations en retour via ce même terminal. L'identité de chacun des deux joueurs dépend du type de joueur sélectionné en début de partie, parmi les trois options suivantes :

Joueur humain (H) : Dans ce cas, le joueur est l'utilisateur (s'il y a deux joueurs humains, ceux-ci prennent tour à tour le rôle de l'utilisateur en se relayant devant l'ordinateur).

Intelligence artificielle (A) : Ici, le joueur est le programme lui-même.

Fichier (F) : Le joueur est soit un utilisateur humain interagissant avec le programme non pas via le terminal, mais en donnant ses instructions via un fichier, soit un autre programme éditant automatiquement ce fichier (il peut s'agir d'une autre instance du même programme, ou du programme d'un autre groupe d'étudiants). Le programme ne devrait pas permettre de choisir deux joueurs de type fichier (ce cas de figure est peu intéressant et complique inutilement l'implémentation)

Lorsque le programme est lancé, il devrait donc commencer par demander à l'utilisateur le type de chaque joueur, **en commençant par le joueur noir** et en suivant la séquence suivante (il est important de respecter cette séquence pour permettre des tests automatisés de votre programme) :

1. Le programme demande d'abord à l'utilisateur d'indiquer le type du joueur en tapant "H" pour Humain, "A" pour IA et "F" pour fichier. Il attend ensuite que la réponse soit entrée dans le terminal. Une fois la réponse entrée par l'utilisateur et confirmée en tapant "Enter", le programme peut poser une seconde question suivant le type de joueur :
   - H : Le programme demande le nom du joueur.
   - A : Pas de question supplémentaire (le nom du joueur sera généré par le programme, il peut s'agir d'un nom fixe ou d'un nom aléatoire, à votre choix).
   - I : Le programme demande le chemin du dossier dans lequel se trouveront les fichiers utilisés pour interagir avec le joueur (le nom du joueur est provisoirement laissé vide et sera lu dans le fichier lors du début de la partie, voir ci-dessous).

Quels que soient les types de joueurs, un tour commence toujours de la même manière :

1. Le programme affiche tout d'abord dans le terminal le dernier mouvement effectué (par exemple : "Le joueur noir a placé son pion en d3"), sauf dans les deux cas particuliers suivants :

— Pour le tout premier tour de la partie, le programme indique “Début de la partie entre `xxx` (noir) et `yyy` (blanc)”, en remplaçant `xxx` et `yyy` par les noms des joueurs.

— Si le joueur précédent a passé son tour, le programme indique par exemple : “Le joueur noir a passé son tour”.

Dans tous les cas (toujours quel que soit le type de joueur), une fois ce message affiché, le programme demande à l'utilisateur de confirmer qu'il a bien reçu l'information en tapant “Entrée”.

2. Dès que l'utilisateur a tapé sur “Entrée”, le programme affiche à l'écran la nouvelle configuration du plateau de jeu et détermine si la partie est terminée (aucun des joueurs ne peut faire de mouvement valide) ou non.

   — Si la partie est terminée, le programme affiche le vainqueur et le score de chaque joueur (nombre de pions blancs et noirs).

   — Sinon, le programme annonce le tour du joueur suivant en affichant par exemple : “Tour du joueur noir”.

   Notez que pour des raisons de simplicité dans la gestion des différents types de joueur, le programme annonce le tour d'un joueur même si celui-ci est forcé de passer son tour. Il y a donc toujours une alternance entre les tours du joueur noir et ceux du joueur blanc.

3. Finalement, le programme donne la main au joueur dont il vient d'annoncer le tour, et attend sa réponse suivant les modalités décrites ci-dessous.

5.1 Joueur humain (H)

Pour un tel joueur, le programme attend simplement que le joueur entre son mouvement dans le terminal, en tapant deux caractères suivis de la touche “Entrée”. Les deux caractères indiquent la colonne et la ligne où le joueur veut placer son pion, par exemple “d3” est un mouvement valide pour le joueur noir si on se trouve dans la configuration de départ indiquée à la Figure 1. Pour passer son tour, le joueur entre la séquence spéciale “00” (zéro-zéro).

Une fois que le joueur a entré son mouvement, le programme vérifie sa validité (le mouvement “00” étant valide si aucun autre mouvement n'est valide).

— Si le mouvement est invalide, le programme affiche une erreur, annule le mouvement et indique au joueur qui vient de jouer de recommencer.

— Si le mouvement est valide, le programme l'accepte et passe au tour suivant.

5.2 Intelligence artificielle (A)

Pour un tel joueur, le programme calcule lui-même le mouvement à jouer, en essayant de trouver, parmi tous les mouvement valides pour ce joueur, celui qui maximise ses chances de gagner à terme la partie. Si aucun mouvement n'est valide, le programme choisit automatiquement le mouvement “00” (passer son tour).

En vue de développer une stratégie pour chercher un mouvement optimal, un extrait du livre *Artificial Intelligence : A Modern Approach* [Russell et Norvig(2010)] vous sera fourni. Vous pouvez utiliser toute autre source pour vous documenter sur le sujet, mais veuillez noter que le but est de développer un algorithme de recherche efficace capable de calculer des mouvements en temps réel. Pour cette raison, il n'est pas autorisé d'utiliser une base de données de mouvements prédéfinis.

Attention : une limite de 20 secondes est imposée pour le calcul de chaque mouvement (ce temps pouvant dépendre de la machine sur laquelle est exécutée le code, il devrait être possible d'ajuster un paramètre dans le code pour réduire le temps de calcul si nécessaire).

5.3 Fichier (F)

Dans ce cas, le programme interagit avec le joueur via des fichiers externes. Lorsqu'un tel joueur est sélectionné en début de partie, le programme commence par demander le chemin du dossier

dans lequel se trouvera les fichiers utilisés pour interagir avec le joueur. Ce dossier devrait exister et contenir deux fichiers initialement vides, nommés `blanc.txt` et `noir.txt`.

En début de partie, un premier échange est effectué pour communiquer le nom des joueurs. Si le joueur de type Fichier est le joueur blanc, l'échange se passe comme suit :

1. Le programme écrit à la première ligne du fichier `noir.txt` le nom du joueur noir (de type Humain ou Intelligence artificielle [1]), suivi d'un retour à la ligne, en vue de le communiquer au joueur blanc de type Fichier. Il attend ensuite que le joueur blanc communique son propre nom en l'écrivant dans le fichier `blanc.txt`.
2. Dès qu'une nouvelle ligne est apparue dans le fichier `blanc.txt`, le programme lit la ligne pour prendre connaissance du nom du joueur blanc.

Si le joueur de type Fichier est le joueur noir, le programme commence par attendre que son nom soit communiqué dans le fichier `noir.txt`.

1. Dès qu'une nouvelle ligne est apparue dans le fichier `noir.txt`, le programme lit la ligne pour prendre connaissance du nom du joueur noir.

Dans ce cas, il n'est pas encore nécessaire de communiquer le nom du joueur blanc via le fichier correspondant, ceci sera fait lors du premier tour de jeu (voir ci-dessous).

Après cela, la partie peut commencer. Supposons par exemple que le joueur de type Fichier est le joueur noir, que le dossier entré en début de partie est `/home/student/reversi/`, et que c'est maintenant le tour du joueur noir.

1. Le programme commence par indiquer le dernier mouvement du joueur blanc au joueur noir de type Fichier en l'écrivant à la fin du fichier `blanc.txt` du dossier `/home/student/reversi/`, sous la forme d'une ligne reprenant les deux caractères correspondant au dernier mouvement (par exemple "f4" ou "00" si le joueur blanc avait passé son tour). S'il s'agit du premier mouvement, c'est le nom du joueur blanc qui est indiqué dans le fichier à la place du mouvement.
2. Le programme attend alors que l'instance jouant le joueur noir (une autre instance du programme, un autre programme ou un joueur humain manipulant manuellement les fichiers) prenne connaissance du dernier mouvement du joueur blanc via le fichier `blanc.txt` (et donc mette à jour sa connaissance du plateau de jeu) puis communique son propre mouvement en écrivant les deux caractères correspondant à la fin du fichier `noir.txt` du dossier `/home/student/reversi/`.
3. Dès que la nouvelle ligne est apparue dans le fichier `noir.txt`, le programme lit la ligne pour prendre connaissance du mouvement. Le programme vérifie ensuite la validité de ce mouvement.
   - Si celui-ci est valide, le programme l'accepte et passe au tour suivant (ceci inclut le cas où il n'y a pas de mouvement valide et le joueur noir a entré "00" dans le fichier pour passer son tour).
   - Si celui-ci est invalide, le programme affiche une erreur dans le terminal indiquant au joueur de recommencer, attend qu'il entre son nouveau mouvement à la fin du fichier `noir.txt`, vérifie la validité de ce nouveau mouvement, et si nécessaire recommence jusqu'à ce que ce mouvement soit valide.

Notez que ce système est conçu de telle sorte que deux instances différentes du programme (voire les programmes de deux groupes d'étudiants différents) puissent jouer l'une contre l'autre en interagissant via ces fichiers. Il existe bien sûr des solutions plus efficaces pour faire communiquer deux processus, mais elles vont au-delà de ce cours et l'idée est donc de résoudre ce problème avec une solution relativement simple à mettre en oeuvre.

En vue de vous aider à manipuler des fichiers de la sorte, et s'assurer que les différents groupes utilisent des solutions compatibles, un exemple de code et une vidéo explicative seront communiquées ultérieurement.

1. Pour des raisons de simplicités, on peut supposer qu'on n'a jamais deux joueurs de type Fichier qui s'affrontent dans une même partie (ce cas de figure a de toute façon peu d'intérêt). Dans cet exemple, le joueur noir est donc de type Humain ou Intelligence artificielle.

6 Implémentation

Le projet se subdivise naturellement en les tâches suivantes :

— Entrées / sorties : Conception des menus permettant de configurer le jeu en début de partie (type de joueurs, dossier pour un joueur de type fichier), puis de lire et écrire les mouvements des joueurs dans le terminal et/ou dans les fichiers pendant la partie.

— Structure de données : conception d'une structure de données appropriée pour stocker une configuration du tableau de jeu, déterminer les mouvements valides possibles et mettre à jour le plateau de jeu en fonction du mouvement de chaque joueur. Cette structure de données devrait être choisie judicieusement pour rendre l'intelligence artificielle aussi efficace que possible.

— Algorithme : conception de l'intelligence articielle

Notez que les deux premières parties peuvent être commencées indépendamment de la troisième : il suffit de créer un programme où l'option "Intelligence artificielle" pour le choix des joueurs est désactivée. Dans un premier temps, vous pouvez également désactiver l'option "Fichier" pour créer un programme permettant juste à deux joueurs humains de s'affronter, ce qui constitue le cas de figure le plus simple.

Pensez dès le départ à bien concevoir votre structure de données pour qu'elle s'adapte à toutes les fonctionnalités requises. Avant même son implémentation, vous pouvez déjà fixer les différentes classes à définir, et pour chacune les attributs et fonctions membres nécessaires pour interagir avec le reste du programme (ce qui constitue les fichiers "header" `.h` de chaque classe).

La partie intelligence artificielle peut être implémentée dans un second temps, sachant que celle-ci se base sur des techniques algorithmiques qui ne seront vues que plus tard (recherche exhaustive par backtracking, "branch-and-bound"), mais vous pouvez bien sûr commencer à y réfléchir dès le début.

7 Evaluation

Les critères d'évaluation suivants seront utilisés (voir aussi la grille d'évaluation en annexe) :

1. Respect de toutes les contraintes du cahier des charges
2. Qualité générale du code, y compris les aspects orientés objets (un code bien commenté est nécessaire pour pouvoir vérifier ce critère)
3. Qualité du rapport (fond et forme)
4. Rapidité et qualité de l'intelligence artificielle
5. Utilisation mémoire de votre programme
6. Bonne utilisation de git : mise en place dès le début du projet, commits réguliers, pertinence des fichiers déposés sur le git (code source uniquement, pas de fichiers compilés, etc.)
7. Qualité de la représentation graphique du plateau de jeu

Le premier critère est indispensable pour réussir le projet (note supérieure à la moitié), les critères suivant permettant d'améliorer la note davantage. Pour le dernier critère, une représentation graphique basique mais fonctionnelle est suffisante (comme la solution proposée en Figure 2), mais une interface graphique plus évoluée donnera lieu à un bonus.

Si l'horaire le permet, un tournoi sera organisé pour mettre en compétition les intelligences artificielles des différents groupes, mais pour que celui-ci se passe dans une ambiance plus détendue il aura lieu après l'évaluation et ne contribuera donc pas à la note.¡

8 Plagiat

Comme tout travail à l'université, le plagiat sera sévèrement sanctionné. Dans le cas d'un programme informatique, toute utilisation de code écrit par quelqu'un d'autre sans mention explicite sera considérée comme du plagiat. Cela inclut également le code écrit par d'autres étudiants : vous ne pouvez pas récupérer du code écrit par un membre d'un autre groupe de projet.

En pratique, vous n'avez pas le droit de

— copier-coller (ou copier manuellement) la moindre ligne de code qui vous n'auriez pas écrite vous-même (en particulier du code trouvé sur internet ou écrit par un étudiant hors de votre groupe), sauf exception indiquée ci-dessous ;

— demander à un autre étudiant de manipuler directement tout ou partie de votre code.

Vous avez par contre le droit de

— discuter de votre projet avec d'autres étudiants ou demander oralement de l'aide pour résoudre des problèmes que vous pourriez rencontrer ;

— rechercher de l'aide sur internet sur un problème générique de programmation (syntaxe ou utilisation d'une fonction) ;

— (éventuellement) réutiliser des bouts de code trouvés sur internet pour réaliser des tâches génériques non directement liées au sujet du projet, à condition de bien mentionner vos sources et de préciser les lignes de code concernées, étant entendu que cela ne peut concerner qu'une partie négligeable du code.

9 Echéances

Une version préliminaire du programme, permettant à deux joueurs humains de s'affronter, est due pour le mercredi 15 novembre à 18h.

La version finale du programme, avec toutes les fonctionnalités, accompagnée du rapport de projet est due pour le mercredi 20 décembre à 18h.

Références

[wik(2017)] 2017. URL https://fr.wikipedia.org/wiki/Othello_(jeu).

[Russell et Norvig(2010)] Russell, S. et P. Norvig. 2010, *Artificial Intelligence : A Modern Approach*, Prentice Hall.

INFO-H-304 Évaluation du projet 2023-2024

Groupe : Note: /5

	I	F	S	B	TB	Commentaires
Critères requis : • Compile et s'exécute • Respect du cahier des charges • Joueur humain (H) • Joueur fichier (F) • Joueur intelligence artificielle (A)						
Rapidité et qualité de l'IA						
Utilisation mémoire						
Interface graphique						
Qualité du code : • Structure et aspects orienté-objet						
Utilisation de git						
Rapport : qualité et pertinence des informations (y compris éléments de réflexion algorithmique)						

Commentaires généraux :