Rapport du projet Othello

FRENZEL Loïc
HENRY Romaric

3 Juin 2025

Table des matières

1 Introduction

Le projet Othello a pour but de créer une IA[1] plus ou moins simple, selon les algorithme utilisé. Il fallait donc codé le jeu dans un premier temps pour ensuite permettre de choisir parmis plusieurs mode de jeu :

- Joueur contre Joueur
- Joueur contre IA
- IA contre IA

Le partie devait originellement s'éxécuté sur terminal. Nous avons fais le chois d'ajouter une interface graphique pour d'une certaine manière rendre le projet plus agréable et permettre de vaguement simplifié les changements de tour[2].

Plusieurs algorithmes devait ensuite être implémenter pour l'IA avec différentes évaluations. Nous n'avons pas eu le temps de tout faire, nous nous sommes donc arrêté à l'élagage $\alpha\beta$. La raison sera vue en détail dans la suite du document.

[1] Ici le terme IA fait référence au IA des années 70-80 des algorithme, qui sont en faite des algorithme qui ont une recherche intéligente

[2] Il s'avairera que dans le futur ce ne fut pas le choix le plus judicieuxdu moins aussi tôt dans le projet

2 Les bases du jeu, l'architecture

2.1 Les premier pas

Comme dit dans l'introduction au début du projet nous avons commencé sur une architecture simple qui éxécuté le jeu sur terminal (Cf le diagramme de classe)

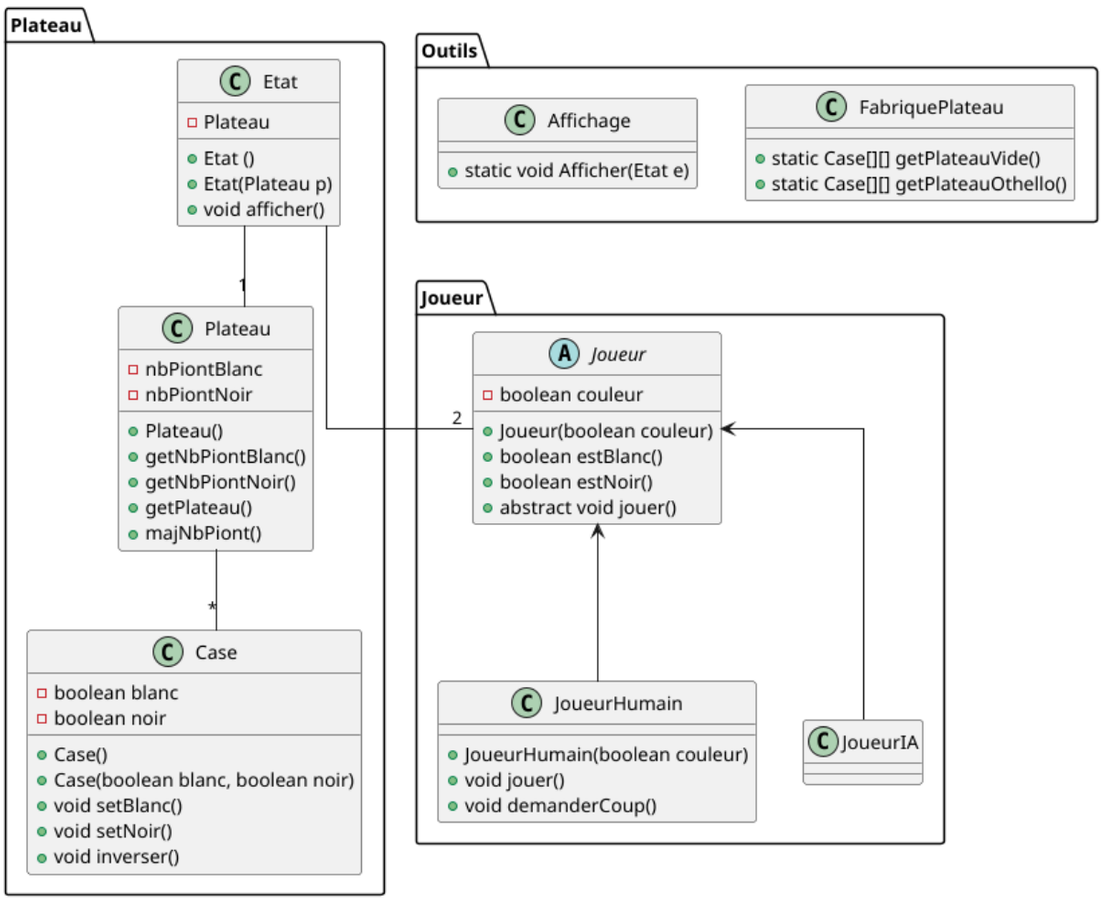

Figure 1: Premier diagramme de classe

Donc cette première architecture fut les bases de notre projet. Une des premières et denières fois qu'on a bien réfléchis avant de codé.

Elle décompose bien les différentes classe du modèle. Le package Joueur gère les joueur avec une classe abstraite pour pouvoir faire jouer sans connaitre le type du joueur entre humain et le joueur IA.

Le package outil possède la classe Affichage et la classe FabriquePlateau. La classe Affichage gère tous ce qui est relatif à l'affichage dans le terminal. La classe FabriquePlateau permet d'obtenir le plateau initial mais aussi différents plateau pour les tests.

Le package Plateau gère la plus grosse partie du modèle. Avec les classes Etat, Plateau et Case. La classe Etat connait deux Joueur, qu'importe leur type réel. Elle sait aussi qui est le joueur courant parmis les deux. Elle gère aussi tous ce qui est successeur d'elle même. Elle connait également un plateau. La classe Plateau est là ou est décrit le plateau de jeu, elle connait le nombre de piont blanc et noir, et une collection à deux dimensions de Case. La classe case possède deux booléen : blanc et noir. Si les deux à false, la

case est vide. Si true à l'un ou l'autre ça veut dire qu'il y a un pion à cette endroit. Le fait que ce soit des booléen facilite l'inversion des pion.

2.2 Le début des problèmes

Une classe Jeu piloté le tout et lancé une partie. Mais très vite l'architecture des joueurs perdu sons sens puisque toute les fonctionnalité du joueur humain furent mise dans la classe jeu. Cela posé beaucoup de problème d'évolutivité du code mais aussi de compréhension et de conception. Ce problème resta longtemps dans le projet. Le changement ne sera opéré qu'une fois des bugs majeur sur le déroulé de la partie apparurent.

Très vite après les premières partie lancé, nous avons décider de changer la structure du projet pour ajouter une interface graphique. Cela nous à permis de mettre en pratique ce qui à était vue dans une autre UE. Mais également de permettre dans une moindre mesure simplifié le jeu à deux joueurs. Son impact à était minimisé due à la présence du problème architectural précédement mis en exergue.

Le modèle utilisé ici pour l'interface graphique est le modèle MVC. Il consiste en une ou plusieurs classe qui sont les sujet observé qui vont rafraichir les observateurs, les vue à chacune de leur modification. Mais la relation est dans les deux sens, puisque les controlleurs peuvent détecter des actions de l'utilisateurs et ensuite demandé appliqué des modification sur le sujet observé.

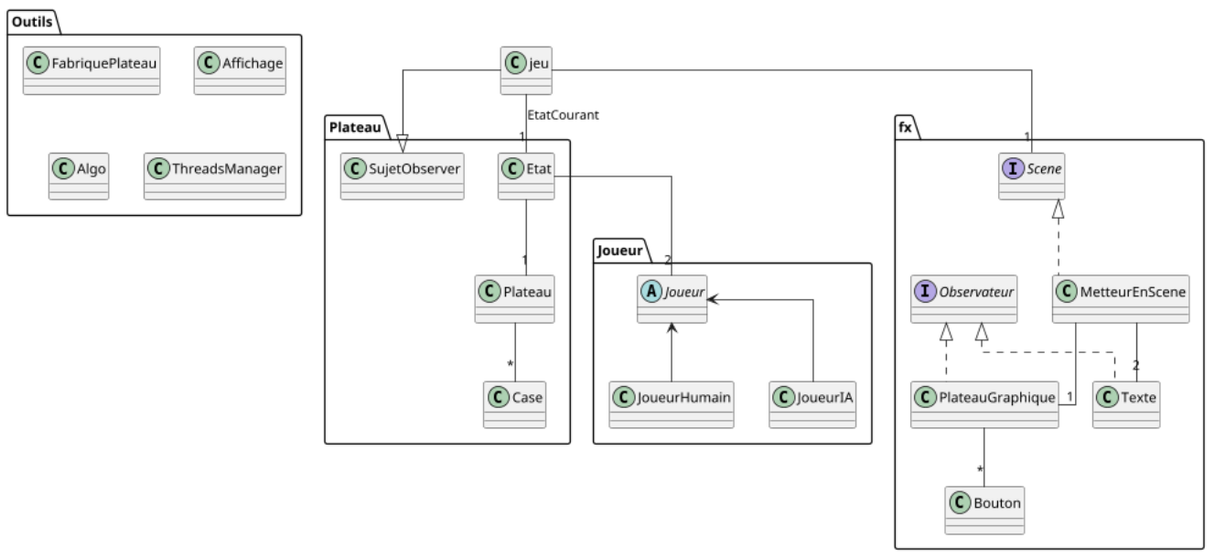

Figure 2: Deuxième diagramme de classe

Ici le sujet observé fut donc la classe Jeu, et l'observateur fut la classe PlateauGraphique. La classe PlateauGraphique est composé d'une grille constitué de bouton. Si le bouton était cliqué une demande de coup était demandé au jeu, cela ne ce faisait que si le joueur courant était un joueur humain. Si le coup était valide, le bouton affiche une image de pion selon la couleur du joueur courant. Sinon une exception est affiché à l'écran, informant que le coup est invalide.

2.3 Des IA et des problèmes

Des problèmes apparurent un peu plus tard quand le joueur IA fut jouable, puisque le plateau graphique ne signalais pas le coup au joueur mais au jeu. Ce qui fait que d'une part la classe joueur perd tout son sens, mais d'autre part qu'il n'y a presque aucune logique de tour, hormis que dans le modèle on change d'état. Mais cela ne se faisait pas bien du à ce soucis de conception.

Pour rester dans ce problème de conception, l'entierté du programme du joueur IA était également réalisé dans la classe jeu... Il avait aussi des problèmes sur le fonctionnement même de l'IA mais cela sera abordé dans une partie futur.

2.4 Une solution...

La décision fut prise de refactorisé le code, pour enfin respecter le principe de JAVA mais aussi d'avoir quelque chose de clair. Les classes des joueurs eurent enfin une vrais utilité, leurs codes qui leur revenaient de droit y furent implémantés.

L'interface graphique à chaque action du joueur sur le plateau, s'il est possible, demande de jouer un coup au joueur courant. Si ce n'est pas le tour d'un joueur humain, le platrau n'est pas réactif. A chaque nouveau tour, le joueur courant regarde s'il a des coups possible, si oui il attend pour le joueur humain, détermnine le meilleur coup pour l'IA. Si aucun coup possible le joueur passe son tour, l'Etat et le joueur courant change.

Malheuresement même après tous cela des bugs perssiste. Mais cela à lieux au niveaux du joueur IA.

2.5 Une complexité des IA

Les IA avec une intarface graphique sont doté d'une petite spécifité. Puisque le calcul du coup qu'elle va jouer est très demandant, elle va entièrement coupé toutes autres actions. Ce qui fait que le joueur humain va jouer un coup puis l'IA va faire le sien après un certain temps, mais on aura pas vue le coup du joueur humain ce placer, due au faite que l'interface fut bloqué puis rafraichis deux fois de suite.

Pour réglé ce problème il faut créer un thread dans le joueur IA qui va s'occuper d'aouter un certain temps d'attente lorsque que la profondeur de recherche et très courte puis de demandé à l'IA de chercher un coup. Il faut aussi ajouter une classe pour géré les thread et les tué une fois qu'ils sont inutiles.

3 Ajout de l'algorithme MinMax

3.1 Algorithme et évaluation

Pour implémenter l'algorithme MinMax nous nous sommes principalement référer sur les documents du cours dans un premier temps nous sommes concentrés sur l'implémentation de la fonction éval0, qui va décider selon le choix de l'utilisateur sur quel type de stratégies nous allons partir. Dans le projet à l'heure actuelle il y'a deux stratégies :

- Le gaint de pions cette première stratégie oriente l'IA à vouloir manger le plus de pions possible.
- La mobilité avec cette stratégie l'IA va plutôt privilégier certaines cases grâçe à un systèmes de pondération sur celle-ci.

Le choix de la stratégie est initialisé par le Joueur dès que le choix est réalisé, lors de l'appel de la fonction eval0 un switch va être fait selon la stratégie du joueur IA et un appel à une fonction éval0 personnalisé va être fait. Pour l'algorithme MinMax en lui même il s'agit d'une adaptation très fidèle fournit dans le cours en pseudo-langage.

L'algorithme semblait fonctionnait correctement peut être un peu trop, en effet en profondeur 8 l'IA jouer coup sur coup le résultat des parties était que nous gagnions toujours contre-elle mais elle jouait vite, bien trop vite pour cette implémentation de l'algorithme à un tel de degret de profondeur. Après de longues sessions de débuggage il est apparu en représentant les successeurs qu'elle ne les visualiser pas correctement. En effet elle se représentait dans sa liste de successeurs à chaque fois comme le joueur courant c'est pour ça que généralement au bout de trois coups elle se voyait manger entièrement les joueurs noirs ce qui est en réalité tout sauf pratique. Après une correction plus que mérité, nous avons constaté quelque chose de bien plus terrifiant en effet quand nous arrivions en fin de partie et que le joueur humain doit passer son tour alors le jeu l'application fige ce qui est bien embêtant, pire encore les partie IA contre s'arrêtaient au bout de trois coups.

3.2 Refactorisation du code

Pour faire suite à ce que nous disions nous avons du opérer à un refactoring du code pour simplifier la logique du jeu qui empêcher toute forme d'évolutivité. Pour cela nous somme concentré sur le rôle des classes du package joueur. En effet les résponsabilité était mal partagé la classe JoueurIA permettait de jouer depuis celle ci mais la classe JoueurHumain ne faisait rien de tout ça la logique de jeu était capturé directement par l'écouteurs. Le refactoring fut plus que bénéfique pour le jeu dans sa globalité le jeu fonctionne que ce soit en IA contre Joueur(avec un bug ou deux mais très léger) et IA contre IA ou maintenant la partie se déroule totalement.

3.3 Les problèmes contre-attaques

Cepandant après une courte période de paix, coup de tonerre MinMax avait de nouveau le même problème qu'auparavant c'est à dire qu'il se représentait dans ces successeurs pouvoir jouer plusieurs coups d'affiler sauf que cepandant la façon dont nous l'avions fixer ne fonctionnait plus c'est donc en bataillant avec le débuggage que nous sommes arriver à une situation ou l'IA se représente correctement ses successeurs non pas sans ambiguité.

3.4 piste de solution

Une piste de solution pour garantir de la stabilité serait de repartir totalement de l'application terminale créer une boucle de jeu standardisé et bien fonctionnel sur sortie standard, s'assurer de sa stabilité entre les algorithmes et le jeu puis après avoir une version bien stable, passait sur une application graphique. Pourquoi cela n'a pas été fait ? Au timing ou ce sont déclaré les problèmes et les différents bugs il était impossible alors de boulverser totalement l'ordre des classes et la logique de jeu car la boucle de ne fonctionne pas de la même façon sur interface graphique que sur sortie standard. Pour éviter ce qu'il s'est passé il aurait fallut conservé un deuxième Main mais dont la partie se fait directement dans le terminal cela aurait pu être une solution le temps de stabiliser l'interface graphique avec le modèle.

4 algorithme avec élagage α β

4.1 Implémentation tardive de l'algorithme MinMax avec élagage α β

Avec le retard accumulé nous ne pensions pas avoir le temps de réaliser une implémentation de l'algorithme MinMax avec élagage, mais finalement nous l'avons fais bien que nous n'avons pas pu réaliser suffisament de tests pour le comparer avec l'algorithme MinMax classique nous avons pu constaté que l'IA jouait beaucoup mieux sur différentes parties, en profondeur 5, et est beaucoup plus performante au même niveau de profondeur que MinMax classique, que ce soit en terme de temps de calcul et aussi sur la pertinence des coups joués.

5 Conclusion

Pour conclure ce rapport nous sommes au regrets de nous rendre à l'évidence que nous avons pas réussit à aller au bout de ce projet malgré de bonnes intentions de notre part. Les problèmes de conceptions nous ont fait accumuler beaucoup trop de retard. Bien que nous n'ayons pas pu finaliser ce projet nous avons pris conscience de plusieurs choses, l'importance d'une bonne conception en effet dans un premier lieu ce n'est pas du temps perdu de bien réfléchir à comment va se piloter l'application il faut penser à tout avant de se lancer dans le développement, nous aurions du aussi au lieux de nous précipiter vers l'idée de faire une interface graphique mais nous concentré sur le fonctionnement de l'application sur sortie standard et d'attendre un certains seuil de stabilité avant de vouloir se lancer sur une interface graphique. Bien que ce projet est aujourd'hui inachevé, nous comptons bien le poursuivre jusqu'a obtenir une version tel qu'elle aurait du être rendu.