Compte rendu : Projet Othello

DUEZ Valentine - 12107343 valentine.duez-faurie@etu.univ-grenoble-alpes.fr
ERAMIL Kadir - 12007772 - kadir.eramil@etu.univ-grenoble-alpes.fr
LUGINBUHL Valentin - 12214021 - valentin.luginbuhl@etu.univ-grenoble-alpes.fr
Université Grenoble Alpes

PROJET
Intelligence artificielle
M1 MIASHS IC S7.

Année universitaire 2024-2025

Contents

1 Introduction

Dans le cadre de notre première année de Master MIASHS (Mathématiques et Informatique Appliquées aux Sciences Humaines et Sociales), parcours Informatique et Cognition, nous devons mener, durant le premier semestre, un projet dans notre cours "Intelligence Artificielle".
Dans ce projet, nous devons coder un jeu ou une application qui comporte une stratégie. Ce projet vise à nous familiariser avec les algorithmes utilisés en Intelligence Artificielle, à développer nos compétences informatiques en développant une application pour deux joueurs, ainsi qu'en implémentant un ou des algorithmes pour que deux IA puissent s'affronter ou qu'un humain puisse affronter l'IA .
Pour développer ce projet, nous devons chercher des algorithme précis capable de résoudre le plus efficacement notre problème, et développer une interface graphique. Notre groupe a choisit le jeu "Othello" pour ce projet. Nous allons vous présenter dans ce document, le déroulement de notre projet et le développement de celui-ci.

2 Présentation du jeu

L'Othello est un jeu de stratégie ou deux joueurs s'affrontent. L'un a les pions noirs et l'autre les pions blancs. Les joueurs posent leurs pions sur une grille de 8x8. Le but est d'avoir le plus de ses pions sur le plateau. Le jeu se termine lorsque le plateau est plein ou bien lorsque les deux joueurs ne peuvent plus jouer. Le plateau de départ se présente de cette manière :

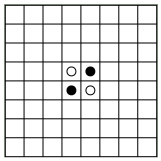

Figure 1: Plateau de départ

Le joueur qui possède les pions noirs commence toujours, puis les deux joueurs jouent chacun leur tour. Si le joueur suivant ne peut plus jouer l'autre joueur rejoue si cela lui est possible. Le joueur ne peut poser son pion qu'à côté ou en diagonale d'un pion déjà posé. Et il faut que le pion qu'il pose soit aligné à un de ses autres pions et qu'il encadre au moins un pion adverse.
Nous avons choisit ce jeu car il est facile à prendre en main et il est facile de développer des stratégies gagnantes. Comme le fait de prendre les angles, de ne pas sortir du carré central 4x4 au risque de ne pas pouvoir poser sur les bords et donc d'encadrer plus facilement les autres pions et bien d'autres.

3 Gestion de projet

Nous avons utilisé l'application Trello pour lister les taches principales, les détailler et les attribuer, ainsi que pour envoyer une notification bi-mensuels à notre professeur pour qu'il constate notre avancement (cf Annexes Trello 2 3).
Nous avons aussi utilisé un diagramme de Gantt pour mieux prévoir le temps que les taches allaient nous prendre, qui a été plutôt bien respecté. Cependant, vers la fin du projet, nous avons repoussé certains délais, car le rendu final a été repoussé (Cf Annexes Diagramme de Gantt 4 5).
Pour des raisons que nous expliquerons ci-dessous nous avons préféré commencer par l'interface graphique puis Humain vs Humain.
Comme il nous était compliqué de coder en même temps nous avons choisi de faire du pair-programming ce qui explique qu'une seule personne push sur le git.

4 Déploiement

Une logique d'intégration et de déploiement continue à été intégré au projet. La branche "Main" sur GitHub est connectée à un déploiement via le service Vercel grâce à des "jobs" décrit pour les GitHub Actions. A chaque mise à jour de la branche, le projet est déployé sur un serveur et accessible via un lien.
Afin d'éviter tout problème de déploiement, les tests contenus dans le dossier "Tests" sont lancés automatiquement lors d'une modification de la branche "Main" suite à une acceptation de Merge Request par exemple. La partie tests sera décrite plus bas, mais cela permet d'éviter toute régression du code lors de l'implémentation et permet aussi de vérifier une dernière fois que les tests de la fonctionnalités ajoutés sont bien valides. Le déploiement via Vercel est dépendant de la réussites des tests, si un seul tests échoue, le déploiement de la nouvelle version est annulé, cela garanti au mieux une version accessible via le lien stable et fonctionnel.

5 Interface graphique

Nous avons choisit de commencer par l'interface graphique puisque la logique de la grille étant déjà modélisée grâce au travail pratique n°2 fournit par le professeur. Le framework utilisé pour l'aspect graphique est VueJS. Ce dernier est relativement léger et apporte un minimum de fonctionnalités comme les composants HTML ou le routing, ce qui est suffisant pour créer les deux pages dont nous avons besoin. L'application est découpée en plusieures parties, une vue "Menu" qui représente la page d'accueil avec les règles du jeu et les différents modes de jeux, "Player" (Human vs IA), "MultiPlayer" (Human vs Human) et "Spectator" (IA vs IA). avec, pour chacun, une icône associée pour améliorer l'affordance du menu et pour que l'utilisateur sélectionne plus facilement le mode de jeu souhaité (Cf Annexe Menu 6).
Chaque mode ramène sur une seconde vue "Board" qui est composé d'un composant "Grid" qui comporte toute la logique de la grille de jeu de l'othello. Nous avons choisit de reprendre les codes couleurs de l'othello avec le vert, les pions noirs et les pions blancs. Nous n'avons pas fait le plateau de la couleur verte habituelle pour une meilleur lisibilité de la grille. Nous avons ajouté une modalité pour que les coups possibles soient affichés grâce avec un petit rond gris au milieu des cases ou le coup est possible ainsi qu'un chronomètre pour le confort de l'utilisateur(Cf Annexe Plateau de jeu 7).
Enfin, lorsque la partie est terminée, le nombre de chaque pions est compté et le gagnant est annoncé à l'aide d'un message type (Cf Annexe Fin de partie 8).
Les pions sont posés en cliquant sur la case associé. Lorsqu'il y a deux joueurs pour le mode multi-joueurs, ils doivent cliquer tour à tour.

6 Multi-player : Humain vs Humain

6.1 Explications

La version de l'othello que nous avons développé permet à 2 joueurs de s'affronter. Chacun son tour, et à l'aide du clic de la souris, les joueurs peuvent jouer les coups autorisés par les règles du jeu. Les coups possibles sont affichés en gris à l'aide d'algorithme de recherches.
Pour ce faire, sur la case libre, c'est à dire une case sans pion, on cherche sur chaque direction (côtés et diagonales) si un pion de la même couleur est aligné, si c'est le cas on regarde s'il y a des pions capturables entre les deux pions du joueur. Si ce n'est pas le cas, on cherche sur la case suivante jusqu'à la limite du plateau de jeu. On vérifie de cette manière que le coup du joueur est valide. Si aucun coup n'est valide pour le joueur actif, il passe son tour (Cf Annexe Plateau de jeu 7).
Lorsque le joueur pose un pion, on vérifie si son coup est valide et on change la couleur des pions de l'adversaire pris entre les pions dans toutes les directions à l'aide du même algorithme de recherche que précédemment. On recalcule les coups possibles pour le prochain joueur pour le tour suivant. A chaque fois qu'un pion est posé, une nouvelle grille est créée et redessinée dans le DOM. Si le coup est invalide il n'est pas cliquable.
La partie s'arrête quand les deux joueurs n'ont plus de coup valide ou que tous les pions ont été posés. On compte le nombre total de pions de chaque joueur, le joueur qui possède le plus de pions sur le plateau est le gagnant annoncé (Cf Annexe Fin de partie 8).

6.2 Tests

Nous avons testé manuellement les parties Humain vs Humain de notre jeu. Nous avons effectué 75 parties (25 parties par personnes de notre groupe).
Durant ces parties, nous avons vérifié que les points gris étaient cliquables et que ils ne représentaient que des coups possibles, tandis que les coups pas en gris (cases vide ou pions noir et blanc) étaient tous incliquables et que les coups étaient tous invalides.
Nous avons aussi vérifié que lorsqu'un joueur ne peut pas jouer, c'est alors bien à l'autre joueur de poser son pion.
Et qu'enfin le gagnant s'affiche uniquement lorsque la partie est finie, c'est à dire lorsque les joueurs n'ont plus de pions ou bien que il ne reste aucun coup valide aux deux joueurs. Et que le gagnant affiché est bien celui avec un nombre de pions > 24 pions et s'il n'y en a pas on affiche bien pour la fin de partie qu'il y a une égalité.

7 IA

7.1 Recherche de l'IA

Algorithme	**Principe**	**Points positifs / négatifs**	**Compatibilité avec Othello**
Minimax (Alpha-Beta Pruning)	Explore les coups en maximisant les gains pour un joueur et en minimisant ceux de l'adversaire. Optimisé par l'élagage Alpha-Beta pour réduire les branches inutiles.	+ Décisions optimales si suffisamment profond. - Gourmand en ressources, dépend de la qualité de l'évaluation.	Adapté
A* (A-star)	Trouve le chemin optimal dans un graphe en combinant coût actuel et estimation restante.	+ Optimal avec heuristique admissible. - Inefficace pour jeux avec un adversaire	Incompatible
BFS (Breadth-First Search)	Explore tous les coups couche par couche jusqu'à atteindre une certaine profondeur. Mais s'arrête dès qu'elle trouve une solution.	+ Solution garantie pour petit espace. - Inefficace pour jeux complexes. Très coûteux pour une solution pas forcément optimale.	Peu compatible.
DFS (Depth-First Search)	Explore tous les coups en profondeur, et s"arrête dès qu'elle trouve une solution, pas forcément optimale.	+ Peu-être peu gourmand en mémoire. - Se perd dans branches non optimales. Très coûteux pour une grille aussi grande pour une solution pas forcément optimale. La solution peut-être très loin et il faudrait des branches très profondes.	Peu compatible
AC-3 (Arc-Consistency)	Résout les problèmes de satisfaction de contraintes (CSP) en rendant les relations entre variables cohérentes.	+ Simplifie les problèmes CSP. - Inefficace pour jeux dynamiques.	Incompatible
IA Aléatoire	Choisit un coup au hasard parmi les options possibles.	+ Simple et rapide. - Aucun aspect stratégique.	Compatible mais non compétitive

Table 1: Comparaison des algorithmes d'IA pour Othello.

7.2 Implémentation de l'IA

Nous avons donc fait le choix d'implémenter une IA Minimax α-β Pruning avec un élagage pour économiser de la mémoire et qu'elle soit moins coûteuse. Pour ce faire, nous avons structuré l'algorithme de la manière suivante :

7.2.1 Fonctionnement de l'algorithme Minimax

L'algorithme Minimax fonctionne en explorant toutes les possibilités de jeu jusqu'à une certaine profondeur, pour évaluer quelles sont les meilleures décisions à prendre. Il part du principe que les deux joueurs jouent de manière optimale, c'est-à-dire qu'ils font les choix les plus avantageux pour eux-mêmes. Quand c'est au tour de l'IA, son objectif est de maximiser son score en choisissant les coups qui lui donnent le plus d'avantages. En revanche, lorsque c'est au tour de l'adversaire, l'algorithme suppose qu'il va essayer de minimiser les gains de l'IA en jouant les coups qui lui sont les plus défavorables. C'est cette alternance entre maximisation et minimisation qui permet à Minimax de simuler des parties stratégiques et optimales.

7.2.2 Optimisation avec α-β Pruning

Pour rendre l'algorithme plus rapide et efficace, nous avons ajouté une optimisation appelée élagage α-β. Cette méthode permet de ne pas explorer inutilement toutes les possibilités en éliminant celles qui ne peuvent pas influencer le résultat final. Avec l'élagage α, si une branche propose un résultat moins avantageux que ce qui a déjà été trouvé ailleurs, elle est immédiatement ignorée. De même, l'élagage β permet d'écarter une branche si elle offre une solution meilleure que ce que l'adversaire peut contrer. Grâce à cette optimisation, l'algorithme évite de faire des calculs inutiles, ce qui est particulièrement utile lorsque l'arbre de jeu est très complexe ou profond.

7.2.3 Fonction d'évaluation

Pour déterminer la qualité des positions intermédiaires, comme lorsque la profondeur maximale de recherche est atteinte ou à la fin du jeu, nous avons développé une fonction d'évaluation (heuristique). Cette fonction donne un score à chaque position en fonction d'un critère. Le critère est le critère est l'importance des coins : ces positions sont stratégiquement cruciales, car une fois capturées, elles ne peuvent plus être reprises par l'adversaire. Elles reçoivent donc une forte valorisation dans le calcul du score.

7.2.4 Déroulement de l'algorithme

D'abord, l'algorithme identifie tous les coups valides pour le joueur actuel. Ensuite, il simule chacun de ces coups en modifiant temporairement l'état du plateau pour voir les conséquences de cette action. Après cela, l'algorithme s'appelle lui-même de manière récursive, en alternant entre maximisation (lorsque

c'est au tour de l'IA) et minimisation (lorsque c'est au tour de l'adversaire), pour explorer les positions suivantes. Lorsque la profondeur maximale d'exploration est atteinte ou qu'il n'y a plus de coup possible, l'algorithme utilise une fonction d'évaluation pour attribuer un score à chaque position. Une fois toutes les options explorées, il retourne le coup qui offre le score le plus élevé pour l'IA.

7.2.5 Limites et ajustements

Une des principales limites de notre implémentation vient de la profondeur d'exploration de l'algorithme. Actuellement, l'algorithme explore un nombre limité de niveaux en profondeu pour éviter que l'ordinateur ne soit submergé par le nombre de calculs à effectuer. En effet, plus on explore en profondeur, plus les ressources nécessaires augmentent, ce que nos machines actuelles auraient du mal à gérer. Cela dit, avec des ordinateurs plus puissants ou des optimisations supplémentaires, il serait possible d'aller plus loin dans l'exploration et d'améliorer significativement les décisions prises par l'IA.

D'un autre côté, il y a aussi de la marge pour rendre la fonction d'évaluation plus sophistiquée. Pour l'instant, l'IA se base sur des critères comme la capture des coins et le contrôle des pions, mais on pourrait intégrer d'autres stratégies. Par exemple, l'IA pourrait essayer de maximiser le nombre de pions récupérés à chaque coup, ce qui serait particulièrement utile dans certaines phases du jeu. Une autre idée serait de combiner plusieurs heuristiques : par exemple, donner de l'importance à la fois aux coins et au nombre de pions capturés. Ces améliorations permettraient à l'IA de mieux jongler entre des stratégies à court et à long terme, et de devenir plus efficace face à des adversaires humains.

8 Player : Humain VS IA

Nous commençons par implémenter une simple joueur aléatoire. Pour tester si le tour par tour avec une IA marchait, pour ce faire nous prenons la liste des coups possibles pour le joueur qui n'est pas humain et l'IA en joue aléatoirement un.
Nous avons ajouté un temps de latence de "réflexion" pour que cela aille moins vite pour le joueur Humain.

8.1 Explications

Pour ce format, nous avons donc notre MiniMax qui affronte un Humain, nous avons donc remplacé notre algorithme aléatoire par l'algorithme décrit dans la sous-section 7.2.

8.2 Tests

Pas de test spécifique pour cette partie. Nous avons juste joué une cinquantaine de partie pour avoir un maximum de scénarios possibles et de s'assurer qu'il n'y a aucun bug. Après plusieurs parties face à l'IA nous nous sommes rendus compte qu'il était plus compliqué de gagner lorsque nous ne commencions pas.

9 Spectator : IA VS IA

9.1 MiniMax VS MiniMax

9.1.1 Explications

Pour ce format, nous implémentons pour les joueurs noirs et blancs l'IA MiniMax. Le problème étant que puisque les deux joueurs ont la même IA, que l'élagage est le même pour chaque partie et que le coup choisit est le même à chaque partie : les parties sont toutes les mêmes. Sur 20 parties, nous obtenons à chaque fois le même résultat : une égalité (Cf Fin de partie MiniMax VS MiniMax 10). Ce qui est tout à fait logique puisque l'emplacement de début est toujours le même.

9.1.2 Tests

Nous n'avons pas fait de test spécifique à cette partie, nous avons juste run en boucle le programme pour en observer le résultat, puisque les tests de l'IA ont déjà été fait dans la partie *Player : Humain VS IA*.

9.2 Min-Max VS Aléatoire

9.2.1 Explications

Pour éviter de tomber tout le temps sur une égalité nous avons fait le choix de faire jouer notre MiniMax contre un algorithme complètement aléatoire.
Lorsque le MiniMax est appliqué au joueur noir, on obtient 94 victoire des noirs, une égalité et 5 victoire des blancs.
Tandis que lorsque le MiniMax est appliqué au joueur blanc, on obtient 28 victoires des noirs, 8 égalités et 64 victoires des blancs.
Il semble donc que le MiniMax soit assez efficace, mais que pour appliquer les heuristiques correctement le fait de commencer ou non impacte énormément le nombre de victoire.

9.2.2 Tests

Nous n'avons pas fait de test spécifique pour cette partie puisque nous savions que le MiniMax et l'algorithme alétoire marchait. Nous avons juste run 200 parties pour comparer le nombre de victoires selon si l'algorithme MiniMax commence ou non.

10 Tests

Pour expliquer les tests non manuels effectués

10.1 Structure générale des tests

Nous avons un setup commun dans beforeEach :
On initialise une instance de la grille (Grid) et du jeu (Game). Puis nous simulons un DOM avec deux éléments # grid-container et #context.

10.2 Test "Setup

Nous vérifions que l'initialisation du plateau est correcte, c'est à dire que les positions initiales des pions sont correctes.
Nous testons que les positions centrales sont correctes, que le joueur initiale est bien le joueur noir, et que nous avons bien 4 coups valides, c'est à dire les coups black-grey sont bien 4 au début de partie.

10.3 Test "Status"

Nous vérifions que si il n'y a aucun coup valide, le tour est bien passé. Nous testons cela en supprimant tous les coups valides (state = null), et que le joueur courant passe du noir au blanc.
Nous vérifions que le jeu détecte correctement qu'il n'y a plus de cases libres. Nous remplissons toutes les cases de pions noirs (state = 'black'). Ainsi la méthode 'isGameOver' retourne true.
Nous vérifions que le jeu compte correctement les pions pour les deux joueurs. Nous testons cela en vérifiant en début de partie qu'il y a bien 2 pions noirs et 2 pions blancs.
Pour tester l'affichage du gagnant, nous testons en mettant tous les noeuds en noir que le jeu affiche "Black wins!" dans l'élément DOM #context.

10.4 Test "Strategy"

Nous souhaitons vérifier que l'heuristique a bien une préférence pour les coins, c'est à dire que l'IA va privilégier les coins parmi tous les coins disponibles. Nous testons cela en vérifiant que si i une case dans un coin (id $\bar{0}$) est marquée comme coup valide (state = 'black-grey'), elle est incluse dans les coups valides détectés.
Enfin, nous souhaitons vérifier que l'heuristique évite les coups proches des coins, car ce sont des coups peu intéressants, c'est souvent une mauvaise option. Nous testons cela en vérifiant que si un coup valide est près d'un coin (id = 1) et qu'un autre est plus éloigné, le meilleur coup est celui qui minimise les risques (le plus éloigné du coin).

11 Conclusion

Pour les Heuristiques nous les avons gardées puisque nous pensons que ce sont les plus efficaces pour gagner comme essayer de prendre les coins, etc... Nous n'avons pas de réelles heuristiques qui empêchent l'adversaire de gagner.
Pour l'algorithme MiniMax, il a été compliqué d'en voir l'intérêt l'autre que les deux mêmes IA s'affrontent mais on observe une différence claire de pourcentage de victoire que ce soit lorsque MiniMax commence ou non, entre l'algo MiniMax et l'agorithme purement aléatoire. Ce projet nous a permis de mieux comprendre et appréhender les différentes heuristiques et les différents algorithmes de recherche dans le choix de notre algo pour l'IA. De découvrir d'autres algorithmes, même si non pertinents.
Mais aussi de développer nos compétences informatiques,en logique et en réflexion.
Toutefois nous avons fait face à quelques difficultés telles que :

- l'implémentation des heuristiques
- rendre les coups possibles cliquables
- l'implémentation du MiniMax
- nous n'avions pas pensé que le fait de commencer impacterait tant la victoire. Ce aurait du être un point à améliorer pour nos heuristiques.

12 Annexes

12.1 Dépôt GITHUB

:https://github.com/lelinguine/othello-project

12.2 Trello

:https://trello.com/b/BVCk8acy/projet-othello

12.3 Trello : Organisation et exemple de tâche plus détaillée

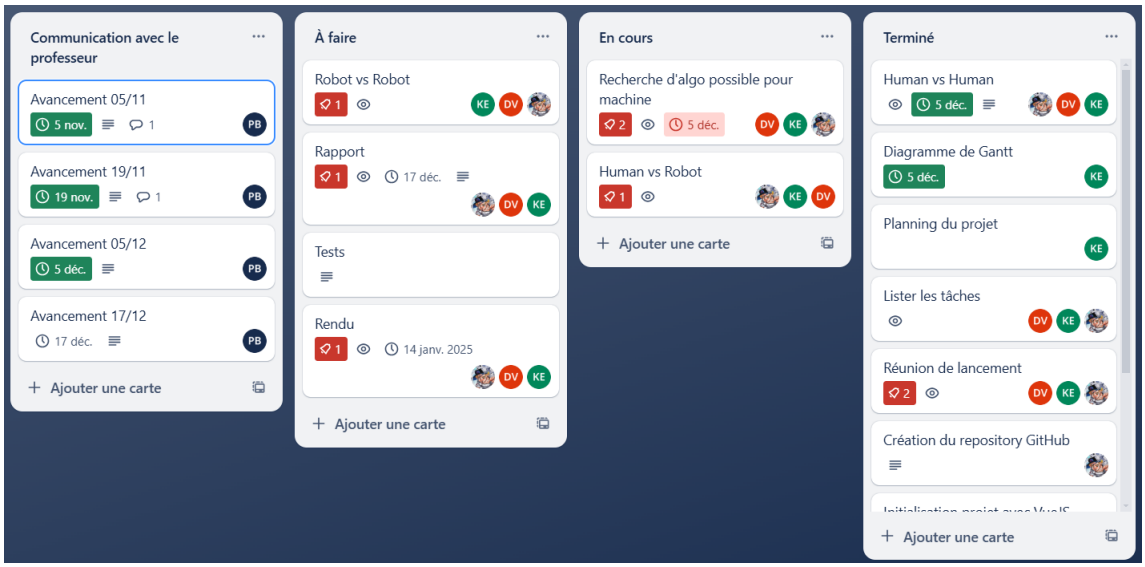

Figure 2: Capture d'écran du trello

Figure 3: Exemple de tâche détaillée

12.4 Diagramme de Gantt

Figure 4: Diagramme de Gantt

12.5 Diagramme de Gantt corrigé

Figure 5: Digramme de Gantt corrigé

12.6 Menu

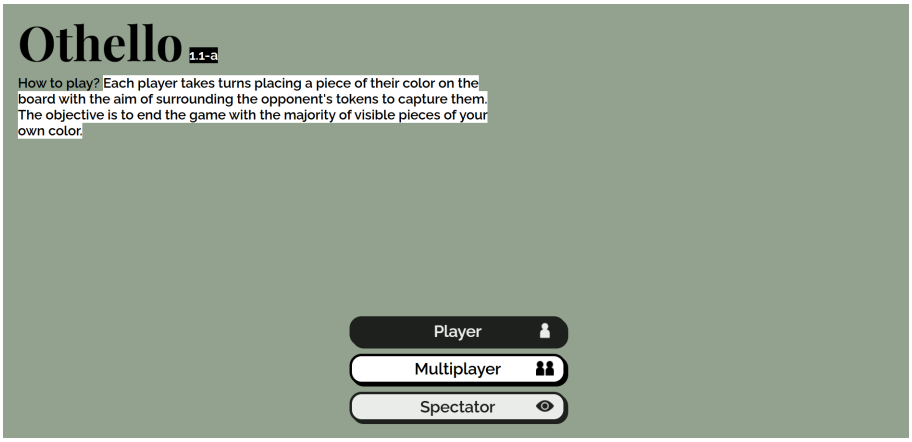

Figure 6: Menu

12.7 Plateau de jeu

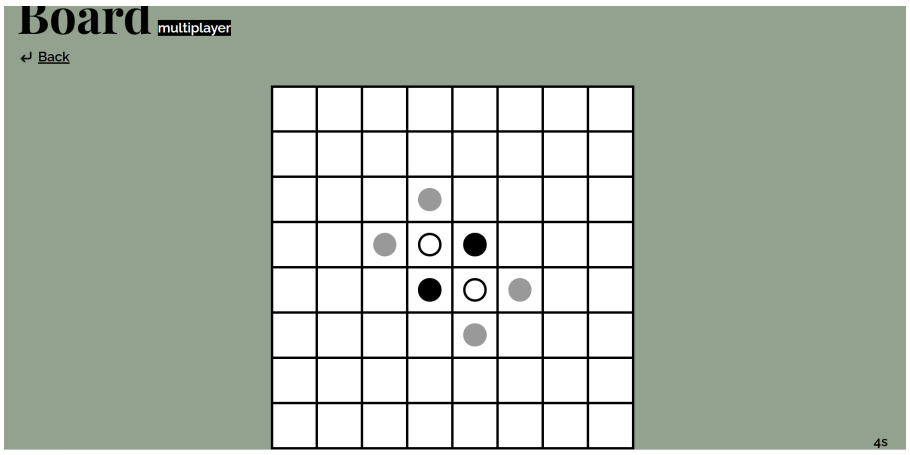

Figure 7: Plateau de jeu

12.8 Fin de partie

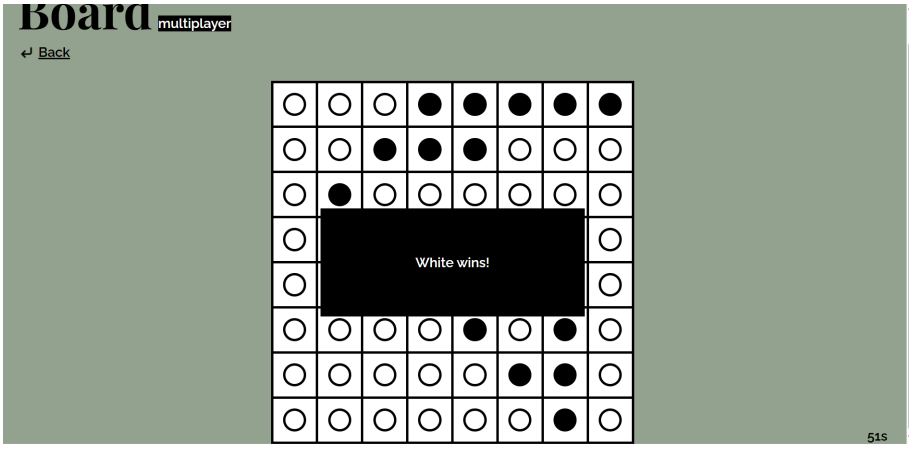

Figure 8: Fin de partie

12.9 HUMAIN VS IA

Figure 9: Menu

12.10 IA VS IA

12.10.1 MiniMax VS MiniMax

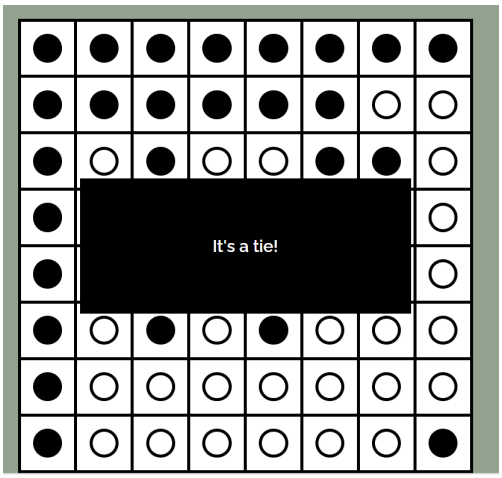

Figure 10: Fin de partie avec MiniMax VS MiniMax