Le Mans Université

Licence Informatique *2ème année*

Conduite de Projet

Othello

David Merlat
Nathan Manson
Samuel Duran

GitHub Othello

19 avril 2019

Table des matières

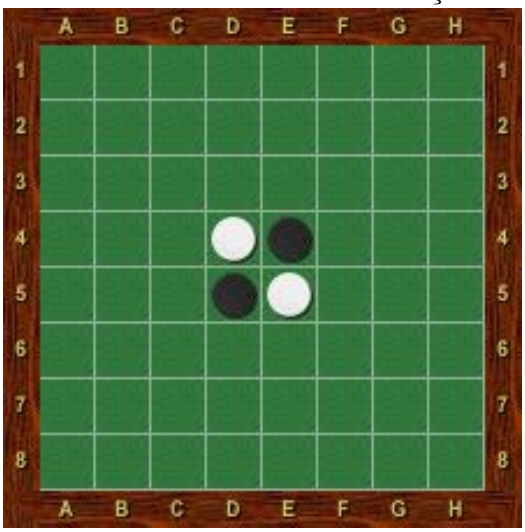
1 Introduction

Ce rapport porte sur notre projet de langage C pour la deuxième année du license. Il nous a été demandé de programmer un jeu d'Othello. Nous allons d'abord en énoncer les règles, puis voir les contraintes qui nous avaient été imposées. Dans une seconde partie, nous verrons comment nous nous sommes répartis la tâche, et nous viendrons ensuite à l'explication de chacune de ses parties. Pour conclure nous verrons où ce projet nous a mené, ce qu'il nous a apporté, ce qui nous a posé problème et ce que nous pourrions améliorer dans le programme.

1.1 Règles du jeu

L'Othello (ou reversi) est un jeu de société se jouant en un contre un. Il se joue sur un plateau de 8 cases par 8 et en tour par tour.

Le plateau se présente comme suit au début de la partie.
(Schéma tiré du site de la Fédération Française d'Othello[1])



Le joueur noir commence. L'objectif du jeu est d'avoir plus de pions que son adversaire à la fin de la partie. Chaque tour, le joueur doit prendre au moins un pion de l'adversaire ou passer son tour. Afin de réaliser une prise, il faut encadrer un ou plusieurs pions adversaires alignés avec un pion de sa propre couleur et le pion que l'on vient de placer. Lorsque des pions sont retournés, les pions qui se retrouvent encadrés par les pions nouvellement retournés ne sont pas pris. Les prises peuvent s'effecter dans vers le haut, le bas, la droite, la gauche et les diagonales. Une partie se termine quand le plateau est totalement remplit ou qu'un des deux joueurs n'a plus de pions.

1.2 Contraintes

Pour la réalisation de ce projet nous avions des contraintes définies par le sujet. Tout d'abord, il fallait que le programme fonctionne, ce qui coule de source. Ajouté à cela, il nous fallait la possibilité d'avoir deux types d'affichages, l'un pour le terminal, l'autre avec une interface grapgique, ce qui nécessitait que l'on utilise les librairies SDL. Ensuite, il nous fallait pouvoir jouer contre une IA, mettant à profit les algorithmes vus en cours. Finalement la possibilité de jouer en réseau et en local était la dernière contrainte imposée.

2 Répartitions des tâches

Pour ce projet, nous avons choisis de nous séparer le travail, ainsi chaque personne travaillerait principalement sur une partie précise du code et ne s'avanturerai pas trop du côté du code des autres. C'est donc David qui s'est occupé de la base du programme, les fonctions pour le déroulement d'une partie, l'interface du terminal et la plupart des fonctionnalités qui allaient avec cet affichage. Nathan quand à lui s'est occupé d'ajouter l'interface graphique sur le squelette tandis que Samuel créait les fonctions pour le jeu en réseau. Une fois ceci fait, on Samuel et David ont travaillé sur l'Intelligence Artificielle qui devait affronter le joueur.

3 Dévelopement du projet

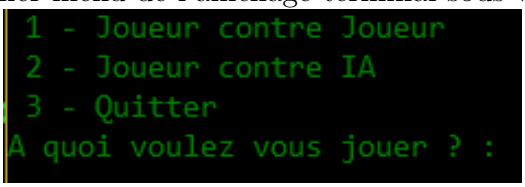
Pour le dévelopement du programme, nous avons eu affaire à différentes problématiques telles que comment stocker le plateau ou encore comment réaliser le tour correctement pour chaque mode de jeu, dans cette partie, nous allons voir comment ces parties ont été réalisées.

3.1 Squelette du programme

Pour ce programme, nous avons choisi de gérer le plateau avec une matrice de chaînes de caractères de 8 par 8. Ces chaînes de caractères contiennent différents affichages en fonction de l'OS sur lequel le programme est exécuté. En effet sur certains OS certains caractères sont inconnus en version terminale. Nous avions donc les constantes NOIR, BLANCHE et VIDE qui sont utilisées tout le long du programme, autant pour l'affichage que pour les tests sur les cases. Nous avons donc beaucoup utilisé la fonction strcmp pour gé-

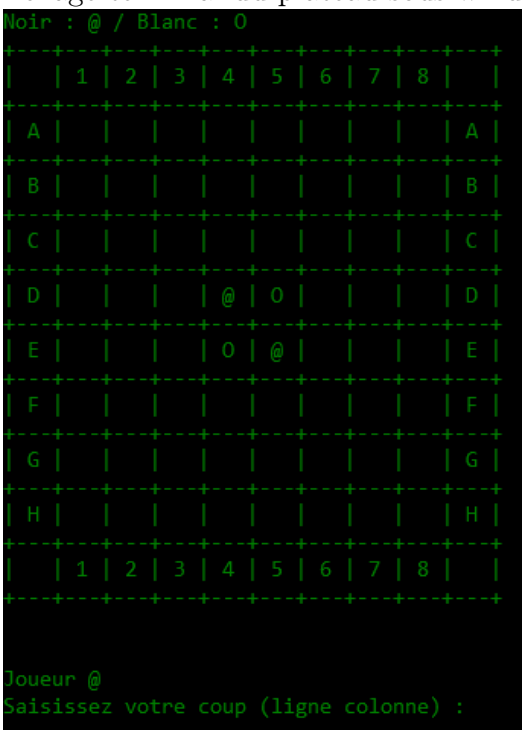
rer le plateau et les variables. Le programme commence par l'affichage d'un menu.

Le premier menu de l'affichage terminal sous windows



Chaque menu nous emmène vers un menu suivant jusqu'au début de la partie. Lorsqu'un menu est séléctionné, une variable du main est modifiée, ce qui va changer le déroulement du programme. Entre autre, le jeu en réseau, contre un joueur local, ou contre une IA. Lors du jeu contre une IA, le joueur peut décider de la couleur de l'IA. Une fois tout les menus parcourus, il est demandé à l'utilisateur s'il souhaite charger une sauvegarde. Si oui, il lui suffit de saisir le nom de la sauvegarde dont le format est txt. Le format des fichiers de sauvegarde ne peut pas être changé par l'utilisateur. Le programme initialise ensuite le plateau avec les paramètres fournis et une fois cette portion du code exécutée, le programme se lance dans une boucle. A chaque ittération de la boucle, le programme va vérifier que la partie n'est pas finie et qu'un coup est possible dans la configuration actuelle. Pour vérifier si la partie est finie, le programme vérifie qu'il y a au moins un pion de chaque couleur et que le plateau n'est pas plein. S'il détecte que la partie est finie, il renvéra alors un entier correspondant au vainqueur : 1 si c'est le joueur Noir qui gagne, 2 si c'est le Blanc et 3 si la partie se conclut sur une égalité. Pour vérifier qu'un coup est possible, le programme va vérifier pour toutes les cases si on pourrait jouer dans cette case avec la couleur donnée. Et s'il en trouve au moins une, alors le coup est possible. Si il n'y a pas de coup possible, la couleur du joueur change et le cycle recommence. Tant que la partie n'est pas finie, le plateau sera affiché et il sera demandé au joueur de saisir son coup.

L'affichage terminal du plateau sous windows



A chaque instant, la couleur du joueur actuel est affichée. Ainsi que le coup précédent si il y en a un. Lorsque le joueur saisi son coup, il peut être soit valide, soit invalide. Les raisons pour qu'un coup soit invalide sont les suivantes :

-le coup est en dehors du plateau.

-le coup est n'est pas dans une case vide.

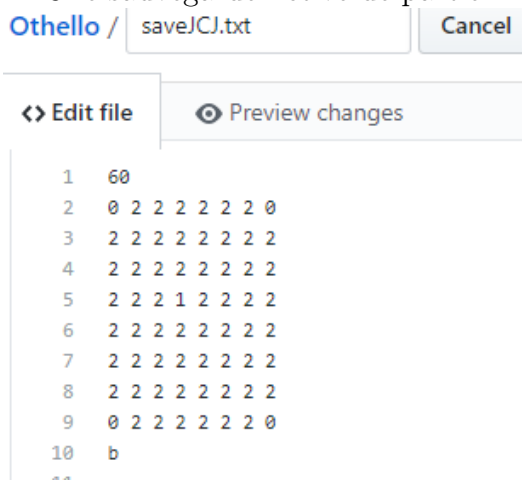
-le coup n'est pas a cote d'une case adverse.

-le coup ne prend pas de pion adverse.

Toutes ces conditions sont vérifiées à la suite et dans cet ordre, en effet, si le coup est en dehors du plateau, il est inutile de vérifier les autres, et s'il n'est pas sur une case vide, les verifications suivantes ne servent à rien et ainsi de suite. Si le coup est valide, il est appliqué. Pour cela, différentes fonctions vont parcourir la matrice et vérifier des valeurs. Nous avons utilisé un tableau d'entiers qui contient soit 1, soit 0, soit -1. Pour chaque direction en partant de la coordonnée saisie par l'utilisateur, le tableau va être rempli. Pour chaque case dans cette direction, on va placer la valeur correspondante de la couleur. Puis le tableau va être analysé. Si il y a une suite de la couleur adverse sans interruption puis un pion de sa propre couleur, alors il y a prise de pion, elles sont donc appliquées par une fonction qui remplace les pions adverses par ses propres pions. Si il y a un espace ou pas de pion de

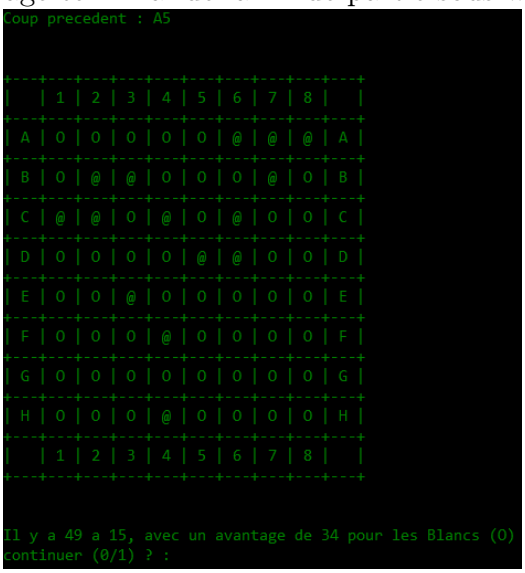
la couleur adverse avant de trouver un pion de sa propre couleur, alors le tableau est réinitialisé et on passe à la direction suivante. Quand toutes les directions ont été traitées, on incrémente le tour et on change le joueur. Il y a cependant une saisie différente qui ne représente pas une case : si dans la version terminale au lieu d'entrer une coordonnée vous rentrez "sa" (pour sauvegarde), le programme vous demande de saisir un nom de fichier et il va ensuite enregistrer la partie avec le schéma suivant : d'abord le numéro du tour, en traitant la matrice ligne par ligne, de gauche à droite, puis la couleur du joueur dont c'est le tour. Le chargement d'une partie est une fonction miroir qui initialise les variables en utilisant un fichier txt créé par le programme. La possibilité d'avoir plusieurs sauvegardes différentes n'est possible que dans la version terminale.

Une sauvegarde fictive de partie



Si la partie est finie, alors on passe directement à l'affichage des résultats, contenant le plateau final et donnant le vainqueur ainsi que le nombre de points de chaque joueur.

L'affichage terminal de la fin de partie sous windows



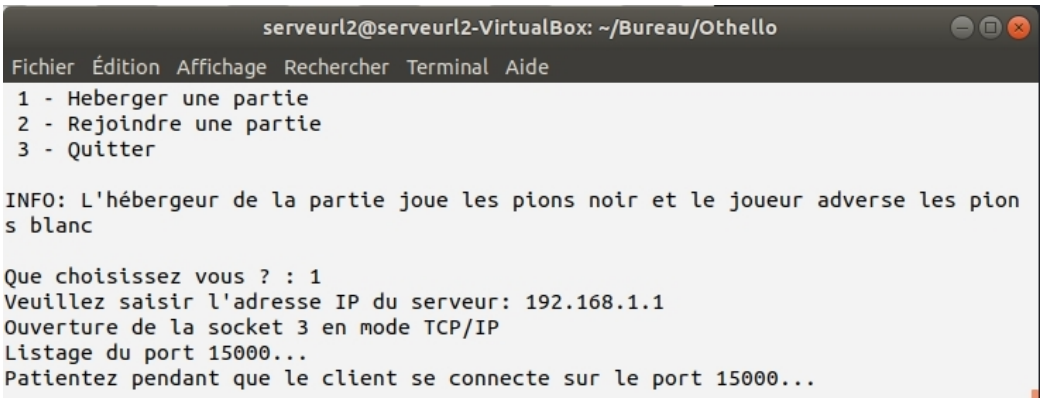
Il est ensuite demandé à l'utilisateur s'il souhaite refaire une partie. Si oui, alors le programme relance une partie avec les mêmes paramètres, sinon il termine l'exécution.

3.2 Réseau

La partie Réseau a pour objectif de pouvoir jouer une partie en LAN entre deux utilisateurs. L'un des utilisateurs hébergera la partie tandis que l'autre sera le client. Les machines doivent appartenir aux mêmes adresses réseau, la donnée communiquée est la case jouée par le joueur, c'est un tableau d'entiers de taille 2 qui contient l'abscisse et l'ordonnée de la case.

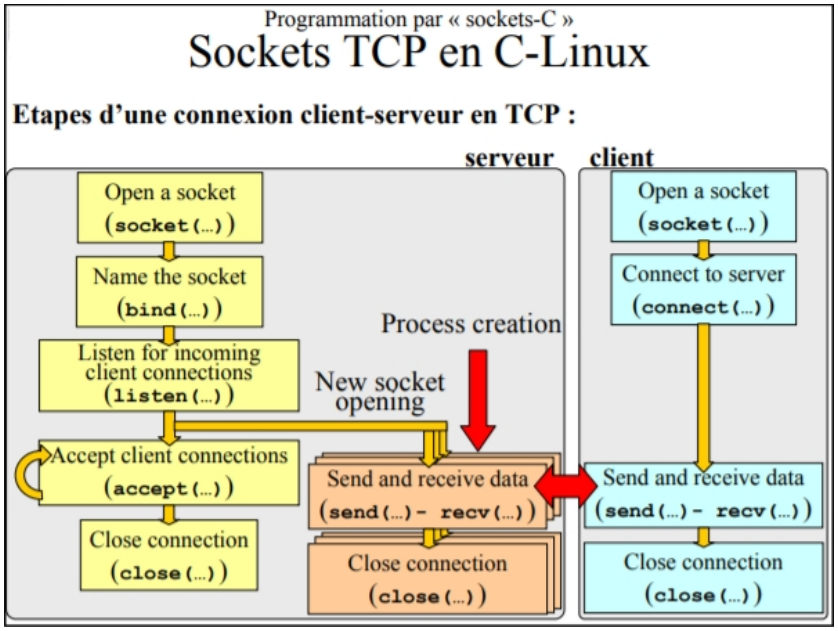
Le réseau en C fonctionne à l'aide de la librairie socket dont on utilise TCP pour maintenir la donnée à destination, autrement dit, plus fiable qu'UDP pour le transport de paquets. D'ailleurs les tutoriels disponible sur UMTICE ainsi que les ressources pour apprendre la librairie optent pour TCP, ce qui nous a influencé sur ce choix.

Le programme réseau fonctionne à partir d'un menu qui opère différents choix suivant la figure ci-dessous :



Affichage du menu sur le serveur et de son adresse IP.

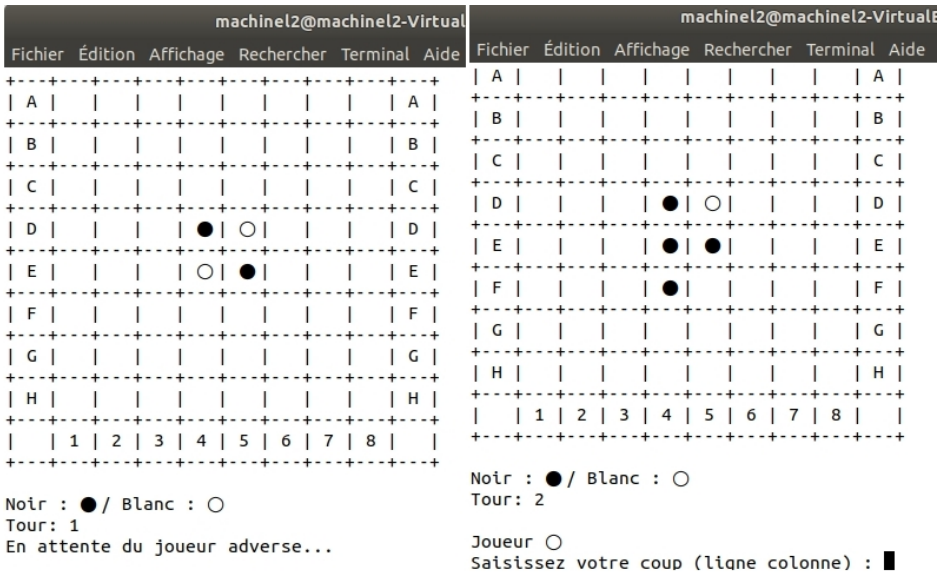
Tout d'abord on définit si la machine est client ou serveur, s'il prend le rôle de serveur, on doit saisir son adresse IP et il en va de même pour le client. Un numéro de port doit être requisitionné, il doit être au-dessus de 1024 et ne doit pas prendre un numéro déjà occupé, par défaut le numéro sera 15000. La socket du serveur se lie avec son adresse IP et son port et passe en écoute, dans l'attente de la connexion du client. Les opérations comme send, recv, shutdown et close seront rappelées par la suite.



Représentation du fonctionnement de la socket TCP.

Dès que la machine client est connectée, il joue d'office les pions blancs tandis que le serveur les pions noirs, cette information est précisée dans le menu. Durant le déroulement de la partie,

lorsque que le tour est impair, le serveur envoie (send) son coup alors que le client attend sa reception (recv). À l'inverse, lorsque le tour est pair, le client envoie (send) son coup alors que le serveur attend sa reception (recv).



Représentation du jeu du client en fonction du tour.

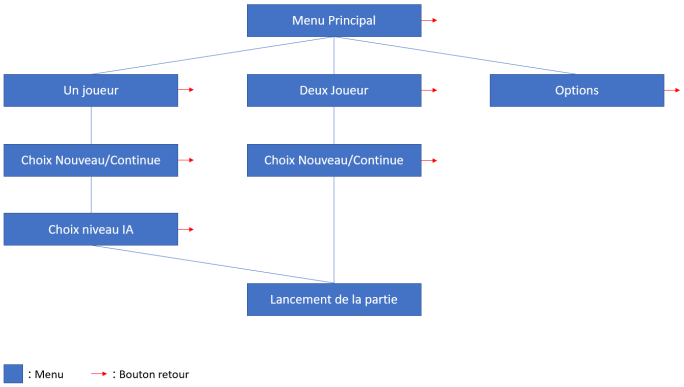
Quand la partie est terminée, le serveur arrête la transmission de la socket par shutdown, le serveur et le client se déconnectent en fermant leur socket. Il n'y a pas d'option pour relancer la partie tout en préservant un score.

3.3 Interface graphique

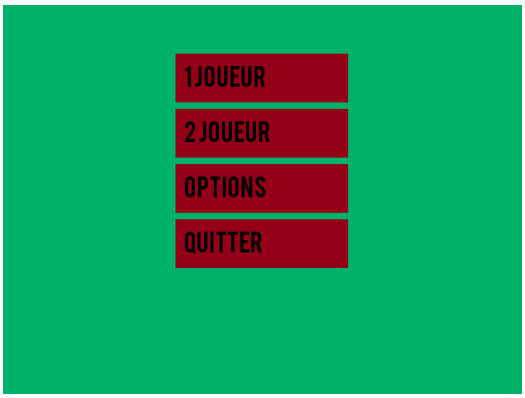
L'interface graphique est gérée grâce à la librairie C SDL (Simple DirectMedia Layer), qui permet d'afficher dans une fenêtre diverse formes et Images. SDL est une librairie multiplateforme très documentée et possède toutes sortes de librairies supplémentaires qui peuvent être ajoutées à SDL. La version de SDL utilisée ici est la version 2.0, ainsi qui la librairie SDL_TTF permettant l'affichage de texte en plus des formes proposées par SDL.

La première étape de l'affichage du jeu de l'Othello est l'affichage des menus. Ils sont la première chose que le joueur voit, et doivent le guider simplement vers les différentes options. L'organisation des menus suit une forme d'arbre, car chaque menu permet d'accéder à un sous menu, et ainsi de suite.

L'arbre des menus et organisé comme suit.

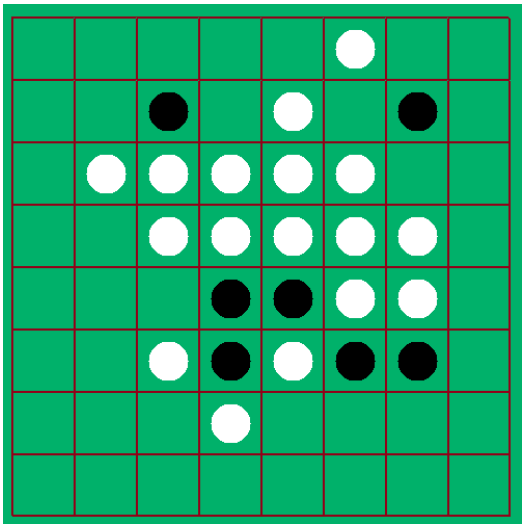


Le menu principal est affiché dès le lancement du programme, il est la première fonction appelée après l'initialisation de SDL. Puis, chacun des menus est appelé en fonction de quels menus l'utilisateur utilise. Les menus se présentent tous sous la même forme ; un certain nombre de bouton (relatif au menu actuel), puis le bouton retour. Le fond de l'écran est uniforme et ne change pas tout du long de l'exécution du programme. Les boutons sont des rectangles, avec du texte correspondant à leur fonction. Leur dimension est définie par rapport à la taille de l'écran, obtenue grâce aux fonctions de SDL.



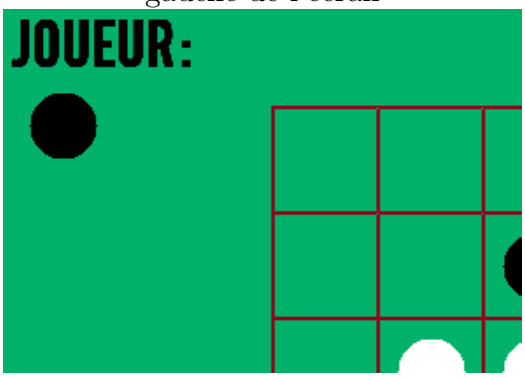
Une fois que l'utilisateur lance une partie, le logiciel remplace la fenêtre affichée actuellement par le fond uniforme, pour supprimer les menus, puis affiche le plateau de jeu. Le plateau de jeu est affiché au centre de l'écran, dynamiquement par rapport à la taille de la fenêtre. Il est constitué de deux éléments : le plateau, et les pions. Le plateau, est constitué de 64 cases, disposées en carré, selon une dimension de 8 cases par 8 cases. La taille des cases est elle aussi dynamique par rapport à la taille de l'écran. Les pions, eux, correspondent à la matrice du jeu d'Othello. Ils peuvent être de couleur Noire ou Blanche, et leur taille est gérée dynamiquement, de manière

à toujours rentrer dans les cases du plateau. SDL ne permettant pas de tracer de cercles, ils sont tracés en utilisant l'algorithme de tracé d'arc de cercle de Brensenham[2]

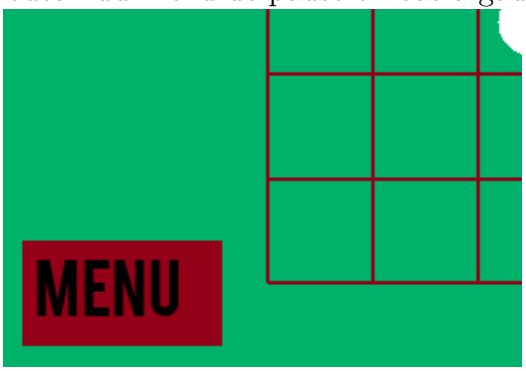


La partie d'Othello se déroule de manière classique, en suivant les règles du jeu de base. Les joueurs jouent chacun leur tour et prennent des pions adverses.

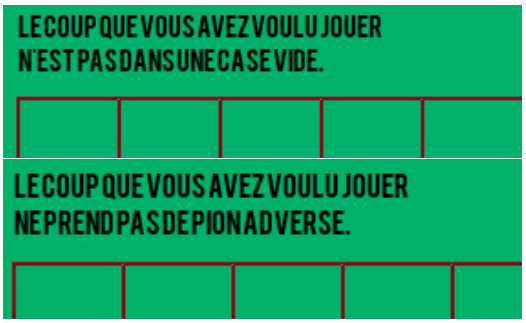
Durant toute la partie, la couleur du joueur actuel est affichée en haut à gauche de l'écran



Le bouton du Menu de pause en bas à gauche.



Le joueur place son pion en cliquant sur la case de son choix, puis l'autre joueur fait de même, et ainsi de suite. Les joueurs ne peuvent cliquer que sur des cases valides car dans le cas contraire, une erreur sera affichée, et le coup sera ignoré puis le programme sera dans l'attente d'un coup valide. Tout coup en dehors du plateau ou pas sur le bouton Menu sera ignoré sans déclencher d'erreur. Les erreurs sont affichées au-dessus du plateau, en fonction de l'action ayant déclenché l'erreur. Elles sont retirées de l'écran lorsqu'un nouveau a été fait.



A chaque tour de jeu, le programme vérifie si la partie est terminée. Une partie est considérée comme terminée lorsque l'un des deux joueurs ne possède plus de pions, ou que le plateau est plein. Le programme affiche alors, au centre de l'écran, un message correspondant a la condition d'arrêt de la partie.

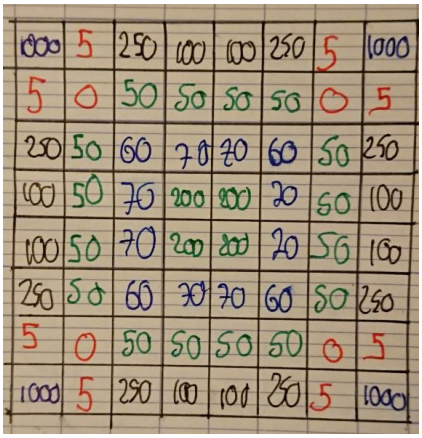
Affichage des différentes possibilités de fin de partie

3.4 Intelligence Artificielle

Pour l'IA, nous avons commencé en lui faisant jouer des coups aléatoires, elle prenait une case au hasard du plateau et regardait si son coup était valide, et elle jouait le premier coup trouvé. C'était le niveau facile de l'IA, car nous avons voulu faire plusieurs niveaux de difficultés. Pour simuler une difficulté nous avons attribué un coefficient de difficulté à chaque niveau. Facile : 1, intermédiaire : 3, difficile : 10 et ultra : 70. Ce coefficient représente un nombre de tours au bout du quel l'IA va jouer un coup aléatoire. Les autres coups devraient donc être choisis de telle sorte que ce soit le meilleur coup possible qui soit joué. Nous avons conscience que l'on aurait pu jouer sur la profondeur de la recherche, car nous avons pensé à un algorithme min max pour trouver le meilleur coup, mais ceci n'a pas abouti.

Cependant, l'idée de l'algorithme min max prend en paramètres le plateau actuel, un plateau à valeurs de force, la couleur, les coordonnées des cases jouables, la profondeur, le nombre de tour et un entier d'évaluation.

Pour détailler, le plateau de force attribue aux cases un entier naturel, plus l'entier est grand et plus la case permet de remporter la partie, la valeur minimale étant 0, 1000 étant le maximum.

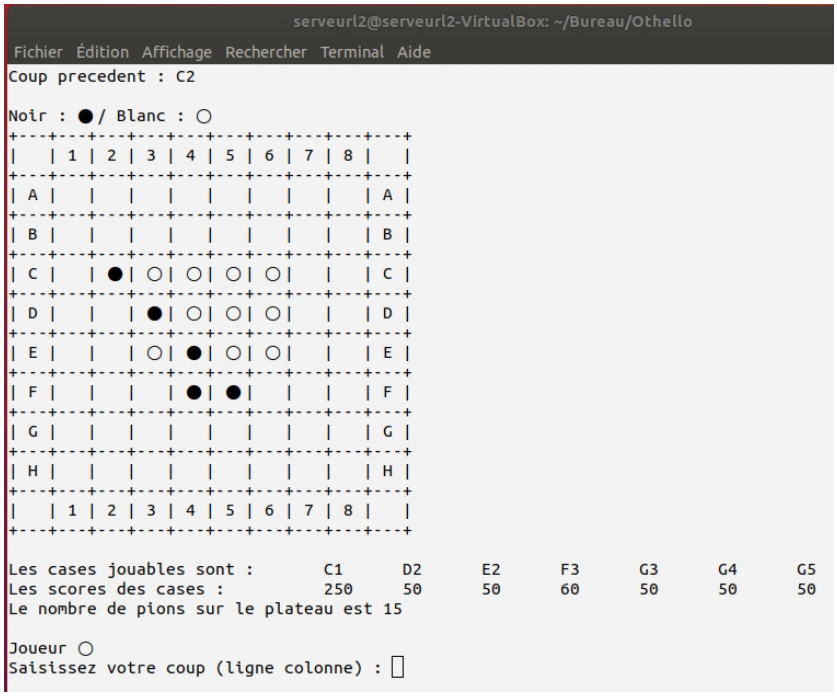


Plateau d'Othello à valeurs de force.

La profondeur est un entier qui a une forte relation avec le nombre de tours, elle détermine jusqu'où min-max va calculer l'évaluation.

L'entier d'évaluation suivant la profondeur additionne l'ensemble des valeurs de force de la couleur de l'IA et soustrait l'ensemble des valeurs de force de la couleur du joueur, ce calcul est fait par min-max. Ceci est la première expérience qu'on a voulu réaliser mais le soucis c'est que suivant la profondeur, les valeurs de force reste médianes, nous voulions qu'une grande valeur, démarrant à 100, soit repérable par le min-max.

C'est alors qu'on a découvert par des simulations de jeu en joueur contre joueur que la profondeur idéale pour chercher cette valeur maximal est à peu près à une profondeur de 10. Pour trouver la profondeur suivant la figure ci-dessous, il faut soustraire le nombre de pions sur le plateau par les quatres pions de départ et la racine pour mieux comprendre, la "profondeur" zéro (15 - 4 - 1 = 10).



La case C1 à 250 se démarque des autres case jouables.

Une grande profondeur implique un calcul lent de l'évaluation, pour résoudre ce problème, il nous faut impliquer une fonction qui va influencer le résultat de l'évaluation, c'est-à-dire une fonction qui permet de calculer la valeur de force des pions retournés. Si cette fonction existe, notre min-max pourra choisir la meilleur case jouable et réduire la profondeur nécessaire. Si le résultat de la fonction ne nous convient pas car la profondeur reste toujours trop grande (c'est une supposition) alors on peut rajouter le nombre de pions retournés ce qui différenciera encore plus l'évaluation et optera pour le meilleur choix. Notre progression dans l'IA se situe dans la création d'une fonction qui retourne les coordonnées des cases retournées.

4 Conclusion

4.1 Résultats

Ce projet nous aura permis de travailler en groupe sur un objectif d'envergure. Il nous aura également fait utiliser des librairies que nous n'avions jamais utilisées (SDL, socket...). Nous avons du comprendre ce qui nous était demandé, nous demander comment répondre à cette problématique, répartir le travail et nous fixer des objectifs de résultat et de temps. Nous avons finalement un programme fonctionnel pour la version terminale et avec l'interface

graphique. Il est possible de jouer en réseau avec le terminal. Pour l'Intelligence Artificielle, Il est possible de jouer sans erreurs, mais nous n'avons pas réussi à lui faire trouver le meilleur coup. Nous avons cependant rajouté quelques fonctionnalités qui n'étaient pas demandées, telles que la sauvegarde et le chargement de partie.

4.2 Difficultées rencontrées

Nous avons eu beaucoup de problèmes avec le min max, bien que nous en ayons compris le principe, sa mise en œuvre a été plus que laborieuse. En conséquence, l'IA n'est qu'un jeu aléatoire. Sinon, pour la création du squellette il n'y avait pas grand chose de différent de ce que l'on connaissait déjà donc il n'y a pas vraiment eu de problèmes sur cette partie. Pour l'interface graphique, il fallait comprendre les fonctions de la librairie SDL et se les approprier. Pour la partie réseau, les difficultés rencontrés se trouvent dans la saisie de l'adresse IP, la gestion des erreurs lié à sa saisie et le déroulement de la partie en fonction de la parité du tour. L'intelligence artificielle pour un jeu d'Othello nécessite une grande expérience dans la recherche des théories possibles et une bonne connaissance des stratégies pour produire une IA capable de réaliser à coup sûr le meilleur coup.

4.3 Améliorations possibles

L'interface graphique étant gérée par un programme différent de la version terminale, certaines fonctionnalitées de la version terminale ne sont pas prises en charge par la version SDL. Une amélioration possible de la version SDL, serait donc la prise en charge du jeu en réseau, qui n'est actuellement disponible que sur la version terminale du programme.
Il faudrait éventuellement finir le système de l'IA, ou revoir l'agencement alambiqué du code pour les deux versions. En effet l'IA ne joue que des coups au hasard car les fonctions qui lui font décider du meilleur coup n'ont pas abouti.

5 Bibliographie

Références

[1] Site de la Fédération Française d'Othello. http://www.ffothello.org/

[2] Page Wikipédia de l'Algorithme de tracé d'arc de cercle de Bresenham. http://bit.ly/2PjikU6