



UNIVERSITÉ PARIS EST-CRÉTEIL

3EME SEMESTRE

Rapport projet

PRÉPARÉ PAR

Haddad Amel
Boudjaoui Zayneb

2022/2023

SOMMAIRE

- 01** Introduction
- 02** Présentation du jeu
- 03** Règles du jeu
- 04** Fonctionnalités
- 05** Choix de programmation
- 06** Structures des données
- 07** Diagramme des classes
- 08** Difficultés rencontrées
- 09** Répartition des tâches
- 10** Citation des sources
- 11** Conclusion

Introduction

Dans ce rapport, nous représentons toutes les étapes que nous avons effectuées afin de créer le jeu. Le programme sera réalisé en Java.

Dans un premier temps nous présenterons le jeu d'Othello, les règles et les stratégies basiques qu'il est nécessaire de connaître pour un bon départ.

Nous présenterons nos choix de programmation, et nous détaillerons les différentes structures des données qu'on a choisi.

Ensuite nous allons representer quelques fonctionnalité pertinente dans notre code en l'expliquant.

Et dans une deuxième partie nous allons expliquer l'algorithme Min max responsable sur le déroulement de l'intelligence artificielle.

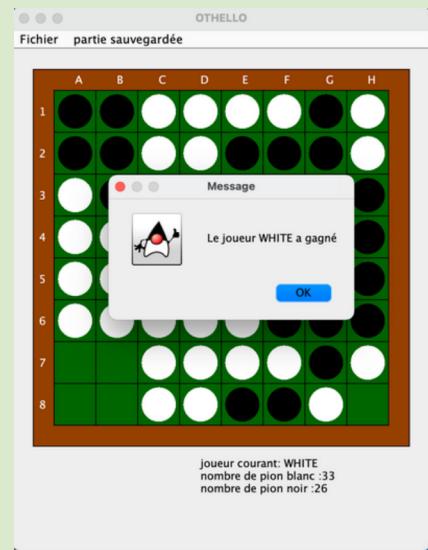
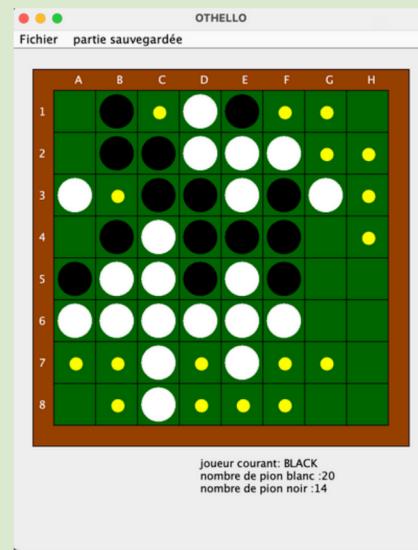
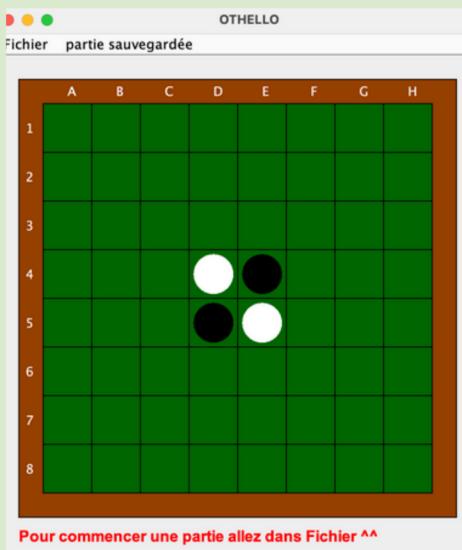
Nous allons finir par citer les différentes difficultés que nous avons eu durant le projet nos différents sources d'inspiration et une conclusion.



Representation du jeu

Othello (aussi connu sous le nom Reversi) est un jeu de société combinatoire abstrait opposant deux joueurs. Il se joue sur un tablier unicolore de 64 cases, 8 sur 8, appelé othellier. Les joueurs disposent de 64 pions bicolores, noirs d'un côté et blancs de l'autre. En début de partie, quatre pions sont déjà placés au centre de l'othellier : deux noirs, en e4 et d5, et deux blancs, en d4 et e5. Chaque joueur, noir et blanc, pose l'un après l'autre un pion de sa couleur sur l'othellier selon des règles précises. Le jeu s'arrête quand les deux joueurs ne peuvent plus poser de pion. On compte alors le nombre de pions. Le joueur ayant le plus grand nombre de pions de sa couleur sur l'othellier a gagné.

(Source: Wikipédia)



Règles du jeu

Nombre de joueurs : 2

Matériel :

- 1 Othellier (plateau unicolore de 64 cases)
- 64 pions bicolores (noirs d'une face, blancs de l'autre).

Les joueurs possèdent un même nombre de pions devant eux, par commodité. En effet, ces pions n'appartiennent à personne. Si l'un des adversaire n'a plus de pions devant lui, il peut piocher dans la réserve de l'autre joueur.

But du jeu :

Posséder plus de pions de sa couleur en fin de partie.

Fin du jeu :

Aucun coup légal n'est possible de la part des deux joueurs. Cela intervient généralement lorsque les 64 cases sont occupées.

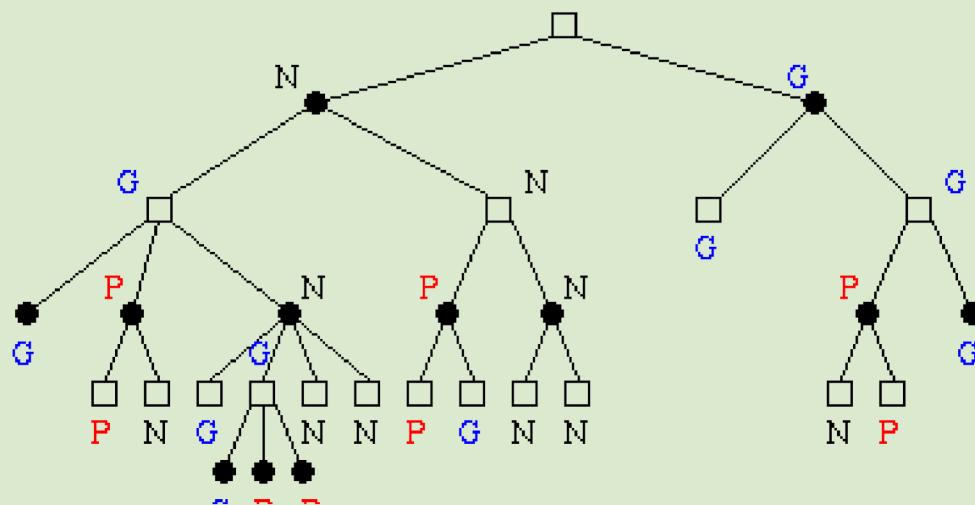
Fonctionnalités

L'intelligence artificielle

Lors de l'élaboration d'un jeu de plateau informatique, la question qui se pose est : comment faire jouer l'ordinateur ? Effectivement, parmi une liste de coups possibles, comment savoir quel coup choisir ? On pourrait choisir le coup qui rapporte le plus de points mais ce coup pourrait nous en faire perdre beaucoup par la suite. Il faut donc un algorithme qui choisit un coup de façon à ce qu'il soit profitable pour la suite de la partie, comme un investissement à long terme.

1. L'algorithme MinMax :

Le but de l'algorithme MinMax est de trouver le meilleur coup à jouer à partir d'un état donné du jeu. Pour chaque coup que l'ordinateur peut jouer, l'algorithme va chercher les répliques que pourrait jouer le joueur adverse ; et pour chaque réplique, l'ordinateur va lister les coups qu'il peut jouer après ce coup adverse. Ainsi de suite jusqu'à atteindre la fin de la partie, qui finira par une victoire, une défaite ou une égalité. L'ordinateur choisira donc le coup qui le mènera vers la plus grande probabilité de victoire. Dans la situation de départ, l'ordinateur cherche à maximiser ses gains, il va donc choisir le coup qui l'amène vers la situation la plus profitable. Le joueur adverse fait de même et cherche à maximiser ses gains (donc minimiser ceux de l'ordinateur). Voilà pourquoi l'algorithme s'appelle MinMax (ou encore MiniMax).



Représentation des simulations de coups par l'ordinateur sous forme d'arbre

Dans cette image, chaque carré représente une situation de la partie où c'est à l'ordinateur de jouer, et chaque rond lorsque c'est au joueur adverse de jouer. Chaque branche représente un coup possible. Ici, l'ordinateur va choisir la branche de droite qui le mène vers une situation gagnante, à contrario du coup représenté par la branche de gauche qui l'amène vers une situation d'égalité. Le joueur adverse qui a deux coups possibles devrait choisir une position qui met en situation de défaite l'ordinateur mais étant donné le choix précédent d'aller vers la branche de droite, l'adversaire peut amener la partie uniquement vers des situations gagnantes pour l'ordinateur. Grâce à MinMax, ce dernier a trouvé le coup qui l'a mené à la victoire. Cependant, on remarque que l'exécution de cet algorithme met beaucoup de temps si on l'implémente sur un ordinateur. Effectivement, si on cherche dès le début de la partie toutes les situations de jeux possibles pour tous les coups possibles, le nombre de coups devient vite très grand même pour un ordinateur (ce n'est pas vrai pour des jeux très limités tel que le morpion, mais c'est le cas pour Othello).

Voici notre algorithme implémentant MinMax :

```

public static int minmax(Node n, int depth){

    if(n.getChildren().isEmpty() || depth == 4){
        return n.getScore();
    }
    else{
        if(n.getType() == Type.MAX){
            int max = Integer.MIN_VALUE;
            for(Node fils : n.getChildren()){
                int val = minmax(fils, depth +1);
                System.out.println(fils.getCoup().getX() + " " + fils.getCoup().getY() + "
" + depth );
                n.getBoard().showterminal();
                if(val > max){
                    max = val;
                }
            }
            return max;
        }
        else{
            int min = Integer.MAX_VALUE;
            for(Node fils : n.getChildren()){
                int val = minmax(fils , depth +1);
                System.out.println(fils.getCoup().getX() + " " + fils.getCoup().getY() + "
max " + depth );
                n.getBoard().showterminal();
                if(val < min){
                    min = val;
                }
            }
            return min;
        }
    }
}

```

2.Fonction d'evaluation

Comme vu précédemment, la fonction MinMax permet de simuler des coups dans le but d'en déduire le meilleur coup jouable. Cependant, comment noter chaque coup que l'on simule de façon à différencier les bons coups des mauvais ? C'est pour répondre à cette problématique qu'intervient la fonction d'évaluation. Cette fonction E évalue un coup et son influence (positive ou négative) sur la partie à partir de plusieurs critères.

Pour l'évaluation, nous avons retenu à un critère :les gains qu'une positionnement apportepour le joueur

Critère de gains : la fonction d'évaluation permet de calculer le nombre des pions d'un joueur apres chaque coup possible. Et le coups qui apporte le plus des points sera joué.

Voilà la fonction d'évaluation:

```
public int eval(){
    int eval = 5*this.plateau.nbPlayer(Player.BLACK) //nombre de pion noir sur le
    plateau
        +10*this.plateau.nbPlayerCoin(Player.BLACK); // nombre de pion noir dans les
    coins du plateau
    if(this.plateau.getPlayer()==Player.BLACK){
        eval+= this.plateau.nbcoupPossible();
    }
    return eval;
}
```

3. Fonction de Coups Possibles

Cette fonction permet de savoir si un coup est jouable ou pas selon si la case où on veut jouer est vide, correspondant à un ennemi ou pas. Pour cela, la fonction vérifie toute les directions vers les quels un pions peut se déplacer. il s'agit donc des directions suivantes :

NORD , NORDOUEST, OUEST, SUD, SUDEST, EST, NORDEST

Cette fonction vérifie toutes les directions precedente et renvoie true si un coup est possible false sinon.

Voici la fonction qui vérifie les coups possibles :

```
public boolean coupPossible(int x, int y) {
    if (isEmpty(x, y)) {
        for (Direction d : Direction.values()) {
            if (this.isEnnemi(x, y, d)) { //si il y a un ennemi a coté selon la direction d
                if (this.coupPossible_aux(x, y, d)) { //on verifie que le coup est possible
                    selon cette direction
                    return true;
                }
            }
        }
    }
    return false;
}
```

Les choix de programmation

```
/**  
 * Ce constructeur produit une grille de jeu.  
 * en parametre  
 */  
public Board() {  
}  
  
/**  
 * Ce constructeur produit une grille de jeu.  
 * un element : EMPTY, BLACK ou WHITE  
 * @param taille la taille de la grille  
 */  
public Board(int taille){  
    this.board= new Pawn[taille][taille];  
    for(int i=0; i<taille;i++)  
        for(int j =0 ; j<taille;j++)  
            this.board[i][j] = EMPTY;  
    }  
    board[3][4] = board[4][3] = BLACK;  
    board[3][3] = board[4][4] = WHITE;  
    this.length=taille;  
}
```

Au niveau de la programmation, comme demandé nous avons utilisé la langage de programmation Java et la programmation orienté objet.

Nous avons crée plusieurs classes chacune d'ente elles s'occupent d'une partie du jeu.
situons quelques-unes:

public class Board {} est la classe qui permet de manipuler la grille du jeu, constitué d'un nom, d'un pion d'un joueur et d'une taille.

public class Partie{} est la classe qui s'occupe de chaque partie du jeu déclenchée.

public class Window{} est la classe qui permet d'afficher la fenêtre du jeu.

Nous avons également utilisé des Enums afin de rendre le code plus court et plus compréhensible : Direction, Ennemi ..

Structures des données

Pour cette partie nous allons représenter toutes les structures des données que nous avons utilisé dans notre code.

Arbre n-aire: Nous avons représenté notre jeu comme un arbre n-aire des possibilités. L'arbre va nous aider pour faire l'IA.

LinkedList : Nous avons représenté les noeuds de l'arbre de notre jeu qui sont les coups possibles comme une liste chaînée des possibilités. Afin de stocker toutes les coups possibles dans cette liste

ArrayList : Cette structure des données est utilisée dans notre code afin de stocker les coups possibles de l'IA débutante.

Exemple: Arbre n-aire jeu morion

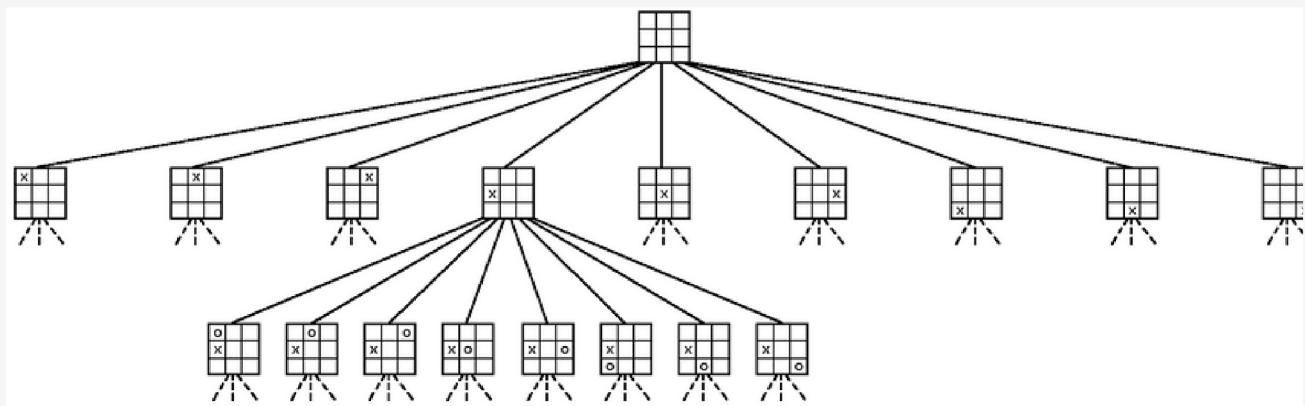
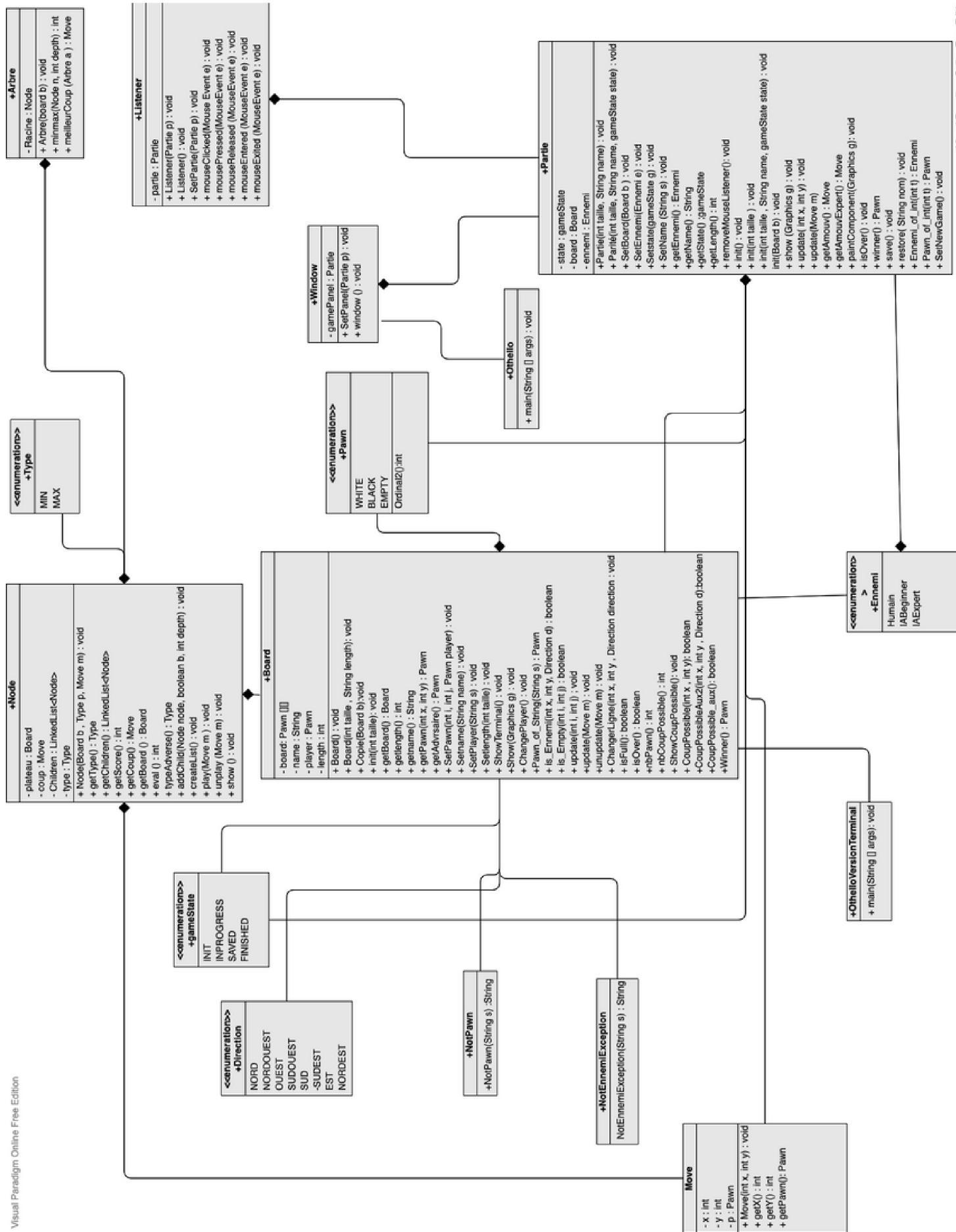


Diagramme des classes



-: attribut/classe privé
+: attribut/classe publique

Difficultés rencontrées

Nous allons représenter dans cette partie toutes les difficultés que nous avons rencontrées durant la réalisation du projet.

Niveau de codage :

- Listener : faire cliquer sur le plateau correctement
- Save : sauvegarder le contenu du plateau correctement et le recharger.
- Faire marcher toute les classe ensemble (exemple : JFrame, JPanel, Listener)
- Difficultés à gérer les bugs qui apparaissent au fur et à mesure d'avancement du projet.
- Comprendre les fonctionnement des classes non-vus en cours.
- Afficher le plateau coloré dans le terminal.

Niveau personnel:

- Difficulté de trouver un créneau où on était disponible toutes les deux.
- Le stress.

Solutions :

Afin de garder un rythme de travail, nous avons décidé de mettre en pratique une système de travail à distance (Discord) pendant une courtes durées.

De plus, nous avons inclus des affichages dans les terminaux afin de trouver la source du bug. Et nous nous avons encouragé afin de faire de notre mieux.

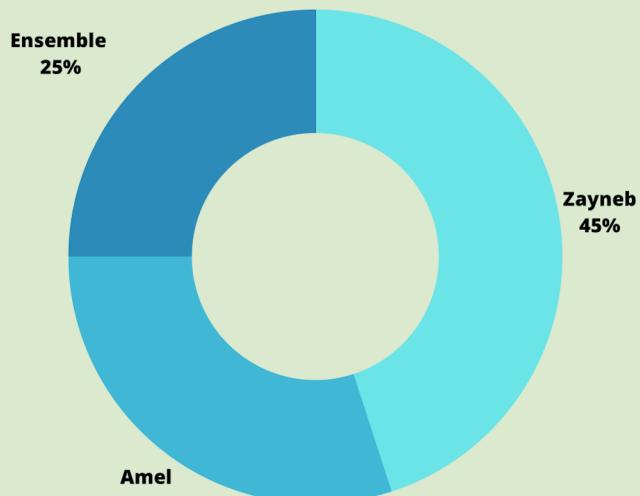
Répartition des tâches

Pour cette partie nous allons présenter comment nous nous avons organisé pour réaliser ce projet.

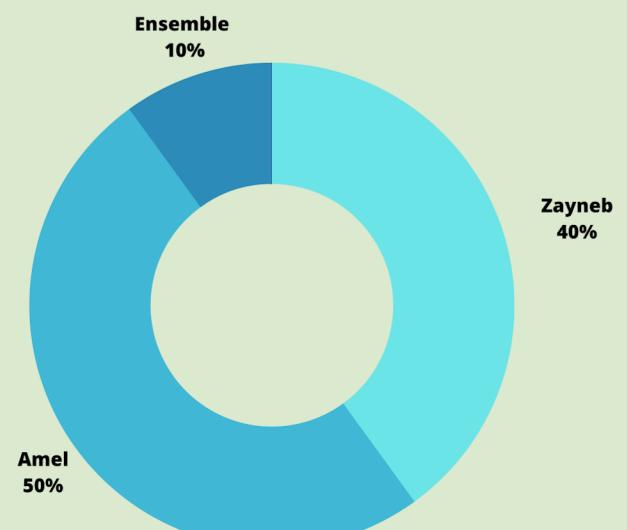
Nous avons décidé de faire toute les parties du projet ensemble afin d'avancer ensemble et rester tout au long du réalisation du projet au courant de toutes modifications.

Il y a bien sur des exceptions, puisque des fois nous avons du mal à s'expliquer un code qu'une d'entre nous a fait. Par conséquent, nous nous trouvons entrain de s'expliquer le code ou de le corriger.

Nous allons représenter le travail de chacune dans une diagramme circulaire afin de le rendre bien claire.



Partie Codage



Partie Rapport

Citations de nos sources

Pour ce projet nous avons bien sur eu besoin de nous inspirer de quelques codes traitant le même sujet sur Internet.

Nous allons citer les sources ci-dessous

JavaDoc: <https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html>

Min max : https://www.youtube.com/watch?v=f30Ry1WOe_Q

Classe MouseListener : https://www.youtube.com/watch?v=jptf1Wd_omw&t=96s

Dessiner sur une jframe : <https://www.youtube.com/watch?v=F0F8A99yEp8>

Classe graphics : <https://docs.oracle.com/javase/7/docs/api/java.awt/Graphics.html>

Compréhension du jeu de l'Othello : <https://www.youtube.com/watch?v=H7GUJSuIJol>
<https://www.youtube.com/watch?v=cNUndcMZcUE>

Dimensionner une jframe: <https://www.codeurjava.com/2015/05/comment-dimensionner-fenetre-selon-ecran.html>

Création des menus sur une jframe : <https://www.youtube.com/watch?v=dm4BAv4xLsM>

Table de code ascii:

<https://www.commentcamarche.net/informatique/technologies/1589-code-ascii/>

Traitement de fichier texte en java : <https://www.youtube.com/watch?v=1JlbzTQu4rs>

Lire un fichier text en java: <https://www.youtube.com/watch?v=DU2VpMIzdFY>

Classe Joption pane: <https://miashs-www.u-ga.fr/prevert/Prog/Java/swing/JOptionPane.html>

<https://waytolearnx.com/2020/05/les-boites-de-dialogue-joptionpane-java-swing.html>

Écrire sur un fichier texte en java: <https://www.youtube.com/watch?v=kjzmaJPoaNc>





Conclusion

Le jeu d'othello est donc un jeu de société rassemble deux joueurs. Chacun prendre un pion en lui associant une couleur Noir ou blanc. Le joueur qui réussi à avoir le nombre le plus grands de ses pions est le gagnant. Notre jeu est réalisé en langage de programmation Java constitué donc de plusieurs classes représentant chaque matériel du jeu (l'othellier ,les pions etc..). En plus, nous avons réussi grâce à notre travail en binômes et aussi la bonne utilisation des sources, à faire l'IA naïve mais aussi l'IA basé sur l'algorithme Min Max.