UNIVERSITÉ PARIS EST-CRÉTEIL

3EME SEMESTRE

Rapport projet

RÉALISÉ PAR

Haddad Amel

Boudjaoui Zayneb

2022/2023

SOMMAIRE

Introduction

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

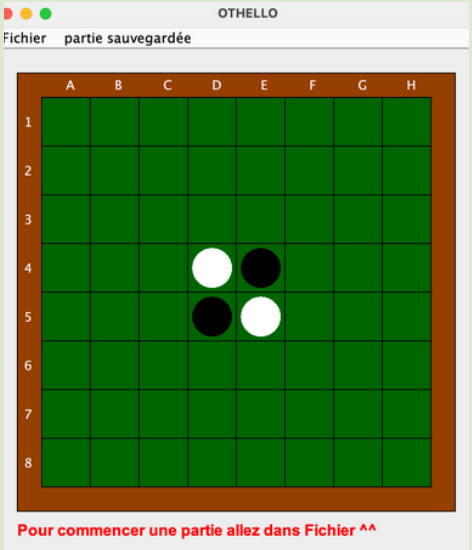
Dans un premier temps nous présenterons le jeu d'Othello, les règles et les stratégies basiques qu'il est nécessaire de connaître pour un bon départ.

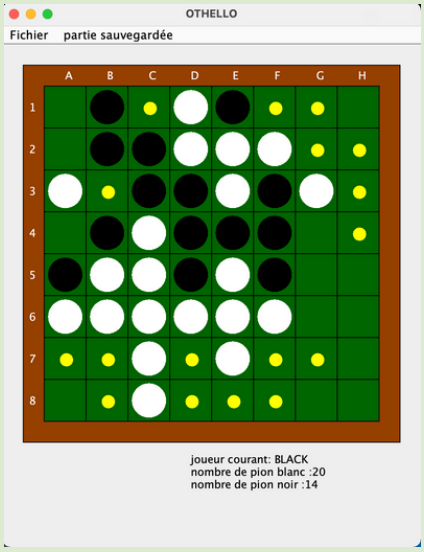
Nous représenterons nos choix de programmation, et nous détaillerons les différentes structures des données qu'on a choisi.
Ensuite nous allons representer quelques fonctionnalité pertinente dans notre code en l'expliquant.
Et dans une deuxième partie nous allons expliquer l'algorithme Min max responsable sur le déroulement de l'intelligence artificielle.

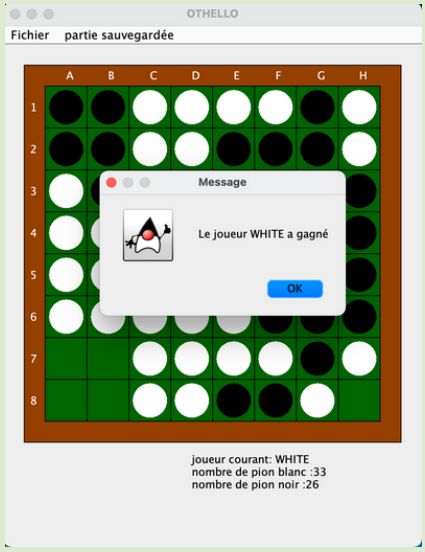
Nous allons finir par citer les différentes difficultés que nous avons eu durant le projet nos différents sources d'inspiration et une conclusion.

Représentation du jeu

Othello (aussi connu sous le nom Reversi) est un jeu de société combinatoire abstraitopposant 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.

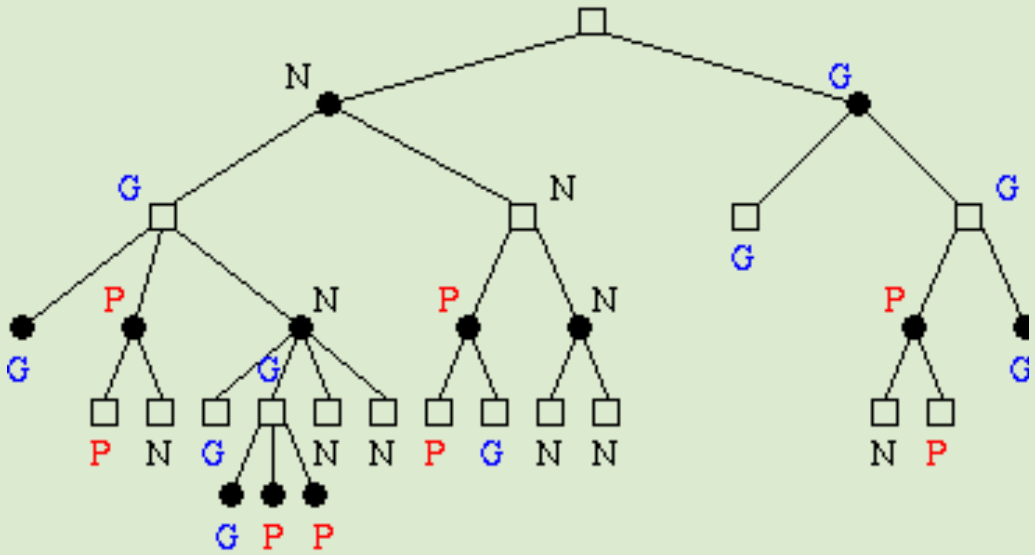
Fonctionnalités

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 le 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 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, a 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 précédant 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'implé- mente 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 :

```
Int fonction Minimax (int depth , Node n){
        if (enfant_noeud = null OU depth = 4){
                return score_noeud();
        }
        else {
                if (Type_noeud == MAX){
                        int max = Integer.MIN_VALUE;
                        for each noeud_fils des enfants
                                int val = minmax(fils, depth+1)
                                afficher les coordonnées du coup
                                afficher le plateau dans le terminale
                                if (val > max ){
                                        max = val
                                }
                        }
                        return max
                }
                else {
                        int min = Integer.MIN_VALUE
                        for each noeud_fils des enfants {
                                int val = minmax(fils, depth+1);
                                afficher les coordonnées du coup
                                afficher le plateau dans le terminale
                                if (val < min ){
                                        min = val
                                }
                        }
                        return min
                }
 }
}
```

2. Fonction d'evaluation

Pour l'evaluation nous avons retenu comme critère :

- Les gains qu'une position apporte au joueur.
- Le nombre de pion placé dans les coins.

En effet plus un joueur a de pions situés dans les coins du plateau et plus celui ci est susceptible de remporter la partie . Un pion situé dans un coin n'a aucune chance d'être retourné par le joueur adverse. De même plus un joueur a de pion sur le plateau et plus celui-ci est en bonne position pour gagner. De plus on peut aussi ajouter a cette fonction le critère de mobilité d'un joueur qui se definit par son nombre plus ou moins élevé de coup possible Plus un joueur a de coups possibles plus il st suceptible de pouvoir renverser la situation s'il est en mauvaise position ou augmenter ses chances de victoire s'il est deja en bonne position .

Voici la fonction d'evaluation:

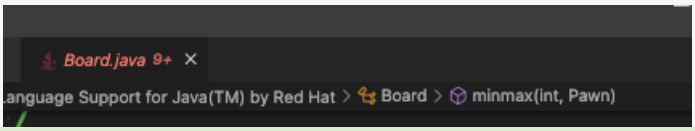
```
Int fonction eval (){
        int eval = 5*nombre_pion_noir_sur_plateau + 10*nombre_pion_noir_dans_les_coins ;
        if( Joueur = Joueur_noir){
                eval += nombre_coups_possible_sur_plateau ;
         }
         return eval
}
```

3. Fonction de Coups Possibles

Cette fonction permet de savoir si un coup est jouable. On verifie dabord si la case ou l'on veut jouer est vide , puis l'on regarde si les cases voisines contiennent des pions adverses si oui on cherche un pion de la meme couleur que celui qu'on joue au dela de ces cases voisines

Voici la fonction qui vérifie les coups possibles :

```
boolean fonction coupPossible (int x, int y){
      if (case_x_y = VIDE) {
            for each direction des directions
                  if (sur_direction = ennemi) {
                        if(Coup_possible_direction ) {
                              return true;
                          }
                   }
            }
      }
      return false;
}
```



Les choix de programmation

```
/**
 * Ce constructeur produit
 * en parametre
 */
public Board() {
}

/**
 * Ce constructeur produit
 * un element : EMPTY, BLAC
 * @param taille la taille
 */
public Board(int taille){
    this.board= new Pawn[ta
    for(int i=0; i<taille;i
        for(int j =0 ; j<ta
            this.board[i][j
        }
    }
    board[3][4] = board[4][
    board[3][3] = board[4][
    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'entre 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 énumérations afin de rendre le code plus court et plus compréhensible : Direction, Ennemi ..

Structures des données

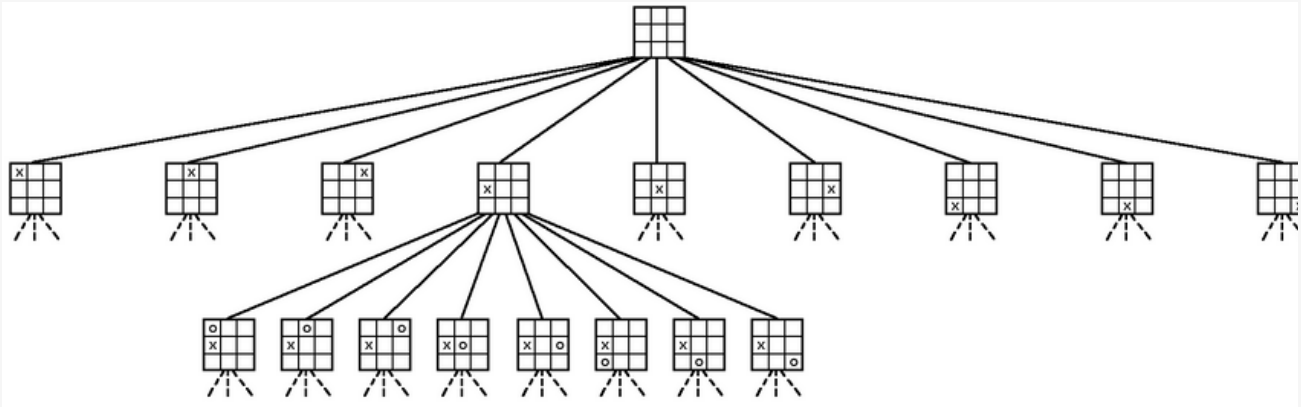
Pour cette partie nous allons représentés toutes les structures des données que nous avons utilisé dans notre code.

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 representé les noeuds de l'arbre de notre jeu qui sont les coups possibles comme une liste chainée des possibilités. Afin de stocker tous les coups possibles dans cette liste.

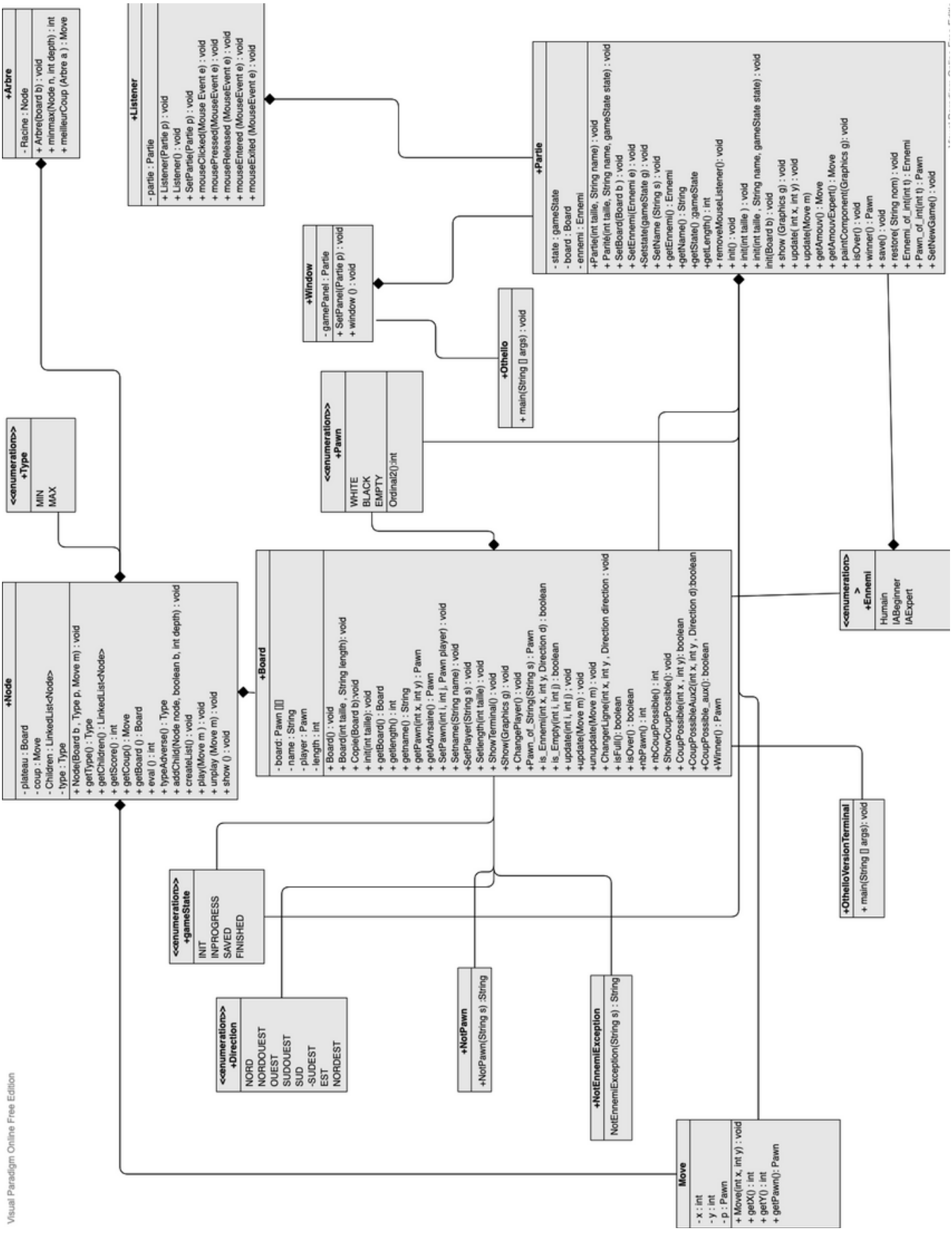
ArrayList : Cette structure des données est utilisée dans notre code afin de stocker les coups possibles de l'IA débutante.

Exemple: Arbre n-aire jeu morion



Source:http://formations.telecom-bretagne.eu/pyrat/?p=912

Diagramme des classes



Visual Paradigm Online Free Edition

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

Difficultés rencontrées

Nous allons représentes dans cette parties 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 toutes les classes ensemble (exemple : Jframe, JPanel, Listener)
- Difficultés à gérer les bugs qui apparaissent au fur et à mesure d'avancement du projet.
- Comprendre les fonctionnements des classes non vus en cours.
- Afficher le plateau coloré dans le terminal.

Niveau personnel:

- Difficulté pour trouver un créneau ou on était disponible toutes les deux.
- Le stress.

Solutions :

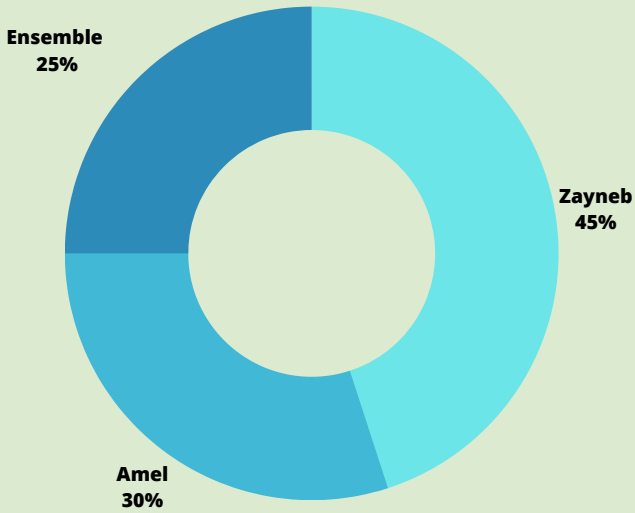
Afin de garder un rythme de travail,nous avons décidé de mettre en pratique un système de travail à distance (Discord) pendant une courte durée.
De plus, nous avons inclus des affichages dans le terminal afin de trouver les sources des différents bugs. Enfin , nous nous sommes encouragées a faire de notre mieux.

Répartition des tâches

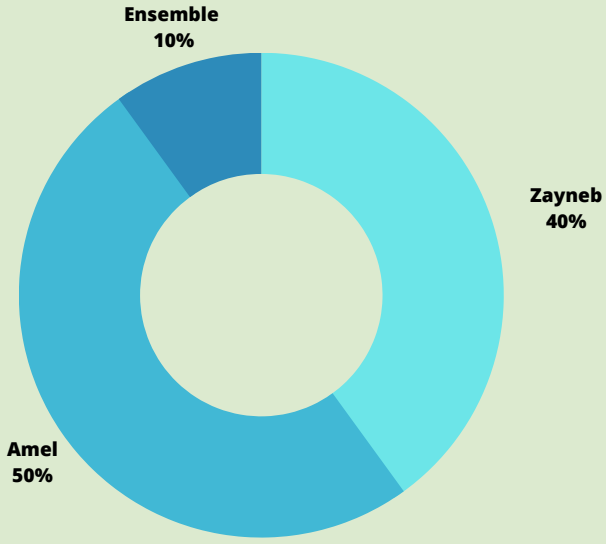
Pour cette partie nous allons présenter comment nous nous sommes organisées pour réaliser ce projet
Nous avons décider de faire toutes les parties du projet ensemble afin d'avancer ensemble et rester tout au long de la realisation du projet au courant de toutes modifications.

Evidemment , il y a eu des exceptions car il nous arrivait d'avoir du mal à nous expliquer entre nous le code que l'une ou l'autre avait fait.

Ci -dessous un diagramme circulaire représentatif du travail fournit par chacune pour ce projet



Partie Codage



Partie Rapport

Citations de nos sources

Pour ce projet les principales ressources utilisées sont toutes issues d'internet. La majorité des liens mènent à des tutoriels sur youtube sur les classes que nous avons utilisées dans le projet

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=H7GUJSulJoI
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=1JIbzTQu4rs
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 de l'othello se joue a deux. La version que nous avons implementé permet de choisir la difficulté du jeu. Nous pouvons jouer à deux (humain vs humain) contre une IA Debutante ou une Ia confirmée. La victoire est à celui qui obtient le plus de pion retourné de sa couleur sur l'othellier. Notre version du jeu permet aussi de faire varier la taille du plateau. En effet celle-ci commence à 4 pour finir à 12(L'othello se joue normalement sur un plateau de taille 8x8).