UNIVERSITE PARIS-EST CRETEIL

UFR de Sciences et technologies

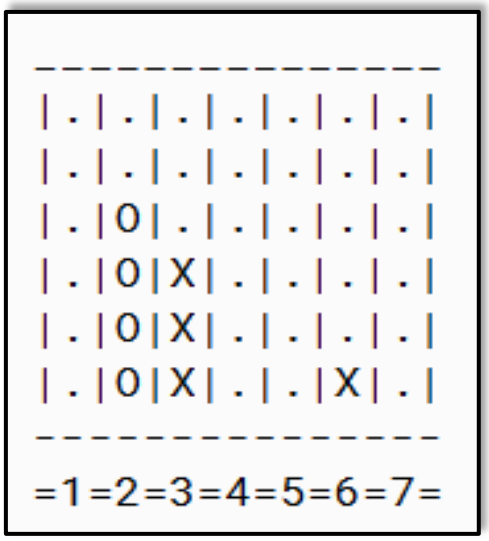
VAL DE MARNE

Compte Rendu Du Projet

Projet programmation L2

2020/2021

Lamia Benamara



Licence d'Informatique
UV Projet
2020/2021

Remerciement :

- Notre équipe remercie Madame Lamia Benamara ainsi que les professeurs de l'université de Créteil, qui ont contribué à notre formation avec les supports adéquats dans cette crise inédite.

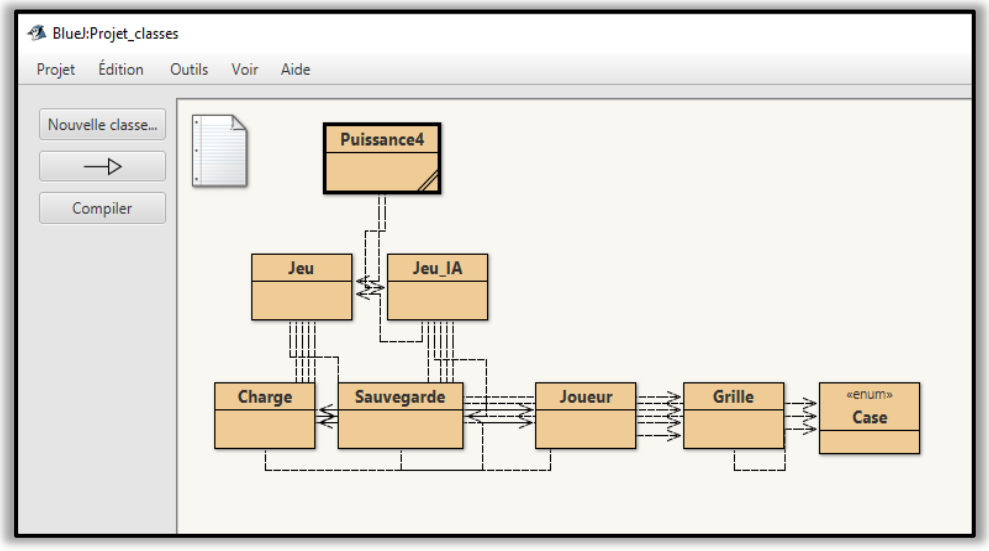
Membres du Projet :

Numéro d'étudiant	NOM/Prénom
U21993992	SALAH Hamza
U31902121	KOCEILA Kemiche

But du projet :

- L'objectif principal de ce projet est de développer nos compétences en programmation et de mettre en pratique les différentes notions théoriques acquises en programmation orienté objet ainsi qu'en algorithmique et structures de données.

  Le projet consiste à implémenter différentes versions du jeu de puissance 4 : la version basique permettant d'affronter deux joueurs humains et la version permettant à un joueur humain d'affronter un ordinateur (IA). La qualité de l'IA est déterminée en fonction de la stratégie (algorithme) adoptée. Le cahier des charges est rédigé par niveau de difficulté. Vous devez suivre l'ordre des questions afin de progresser efficacement.



Règles du jeu :

- Le but du jeu est d'aligner une suite de 4 pions de même couleur sur une grille comptant 6 rangées et 7 colonnes. Chaque joueur dispose de 21 pions d'une couleur. Tour à tour les deux joueurs placent un pion dans la colonne de leur choix, le pion coulisse alors jusqu'à la position la plus basse possible dans la colonne à la suite de quoi c'est à l'adversaire de jouer. Le vainqueur est le joueur qui réalise le premier un alignement (horizontal, vertical ou diagonal) consécutif d'au moins quatre pions de sa couleur. Si, alors que toutes les cases de la grille de jeu sont remplies, aucun des deux joueurs n'a réalisé un tel alignement, la partie est déclarée nulle.

Analyse du problème, Cahier des charges :

1. PARTIE1 : Créer le jeu de base (humain vs humain).
2. Choisir la hauteur et la taille de la grille.
3. Sauvegarder et charger une partie en cours dans un fichier texte.
4. **PARTIE2 A**: IA naïve,
5. **PARTIE2 B** : IA basée sur Minmax et Alpha-Beta.

Consignes importantes :

- Ce projet doit être réalisé en binôme.
- Langage de programmation : JAVA.
- Visionnage sous GIT.

Répartitions des taches :

- Les répartitions des taches ont était égale entre les deux partie du binôme et ont était respecter par les deux partie : <en plus de l'entre aide mutuelle dans les taches>

Etudiant :	**Taches effectué :**
SALAH Hamza	Joueur vs Joueur, IA_simple, Contribution Minmax, Réalisation et contribution Alphabeta, Résolution de bugs, Java_Doc et commentaires. Rédaction du rapport.
KOCEILA Kemiche	Joueur vs Joueur, Réalisation du code MinMax et du code Alphabeta, Résolution de bugs, Java_Doc et commentaires. Rédaction du rapport.

Source du Projet :

Puissance 4 : https://fr.wikipedia.org/wiki/Puissance_4

Arbre du jeu : https://fabien-torre.fr/Enseignement/Cours/Intelligence-Artificielle/jeux.php

Minmax :

- https://fr.wikipedia.org/wiki/Algorithme_minimax.
- https://www.youtube.com/watch?v=da1uzaj549A&t=180s&ab_channel=SylvainSaurel
- https://forums.futura-sciences.com/programmation-langages-algorithmique/442444-aide-intelligence-artificielle-jeu-puissance4-matlab.html

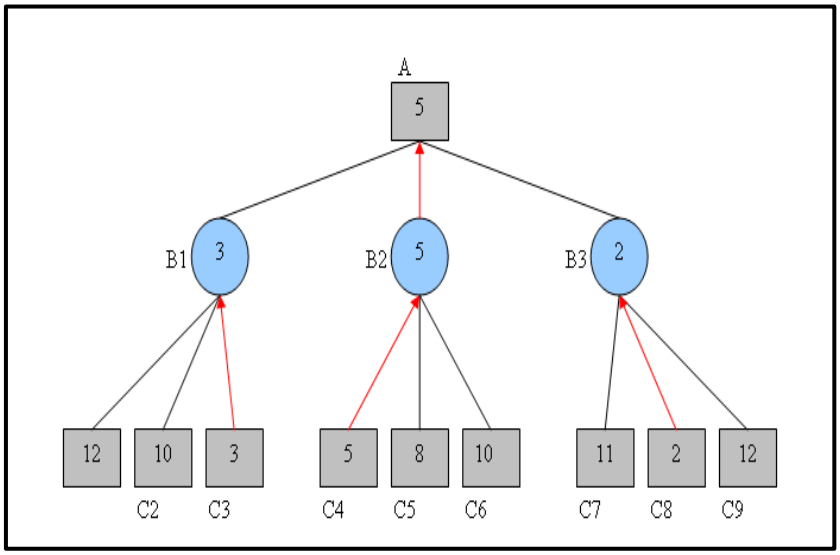


Illustration Minmax

Alpha Beta :

- https://fr.wikipedia.org/wiki/%C3%89lagage_alpha-b%C3%AAta
- https://www.youtube.com/watch?v=j4bFdwkAll0&ab_channel=HugoLarochelle

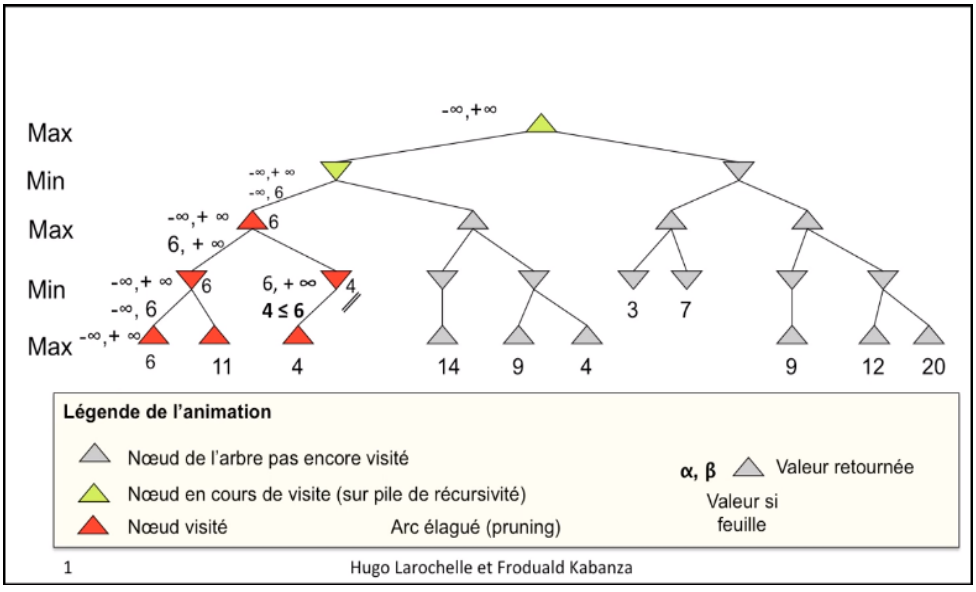
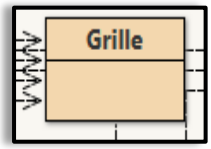


Illustration Alpha Beta

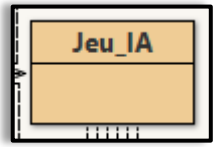
Problèmes rencontrées :

1. Refus de push/pull sur Gitlab
2. Compréhension de Minmax et d'alpha beta
3. Bug contre IA-MinMax choix de la meme colonne a l'infinie quand elle est pleine <Résolue>

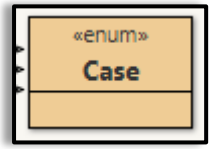
Les Classes du jeu :



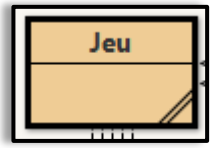
Cette classe représente la grille. Elle contient toutes les méthodes nécessaires.



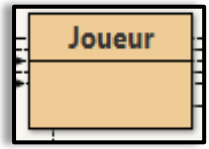
Cette classe représente IA de notre jeu.



: Cette classe représente une case de la grille. Soit elle est Vide ou pleine

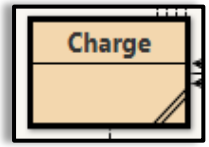


Cette classe représente la partie « J v J » de notre jeu.

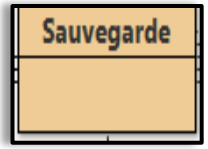


Cette classe représente le joueur. Elle contient le nom du joueur et son type de pion.

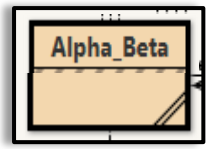
Cette classe le lancement de notre jeu.



Cette classe représente le chargement d'une partie.

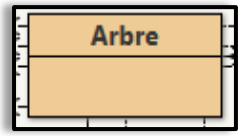


Cette classe représente la sauvegarde d'une partie.

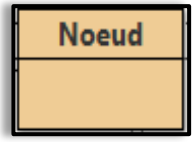


Cette classe représente l'algorithme d'élagage Alpha beta.

Cette classe représente l'algorithme MinMax et son code de lancement.



Cette classe représente l'algorithme d'élagage Alpha beta.

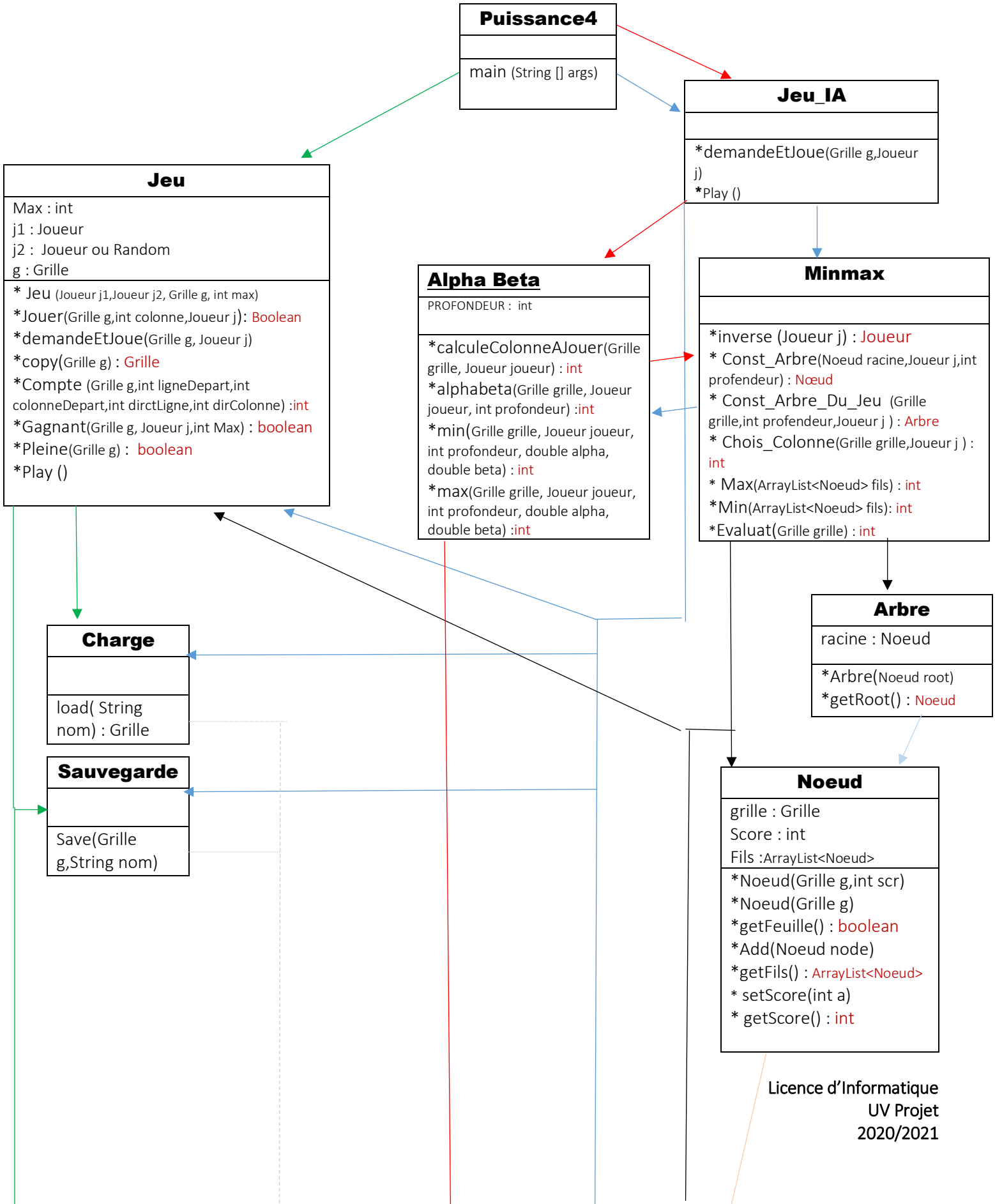


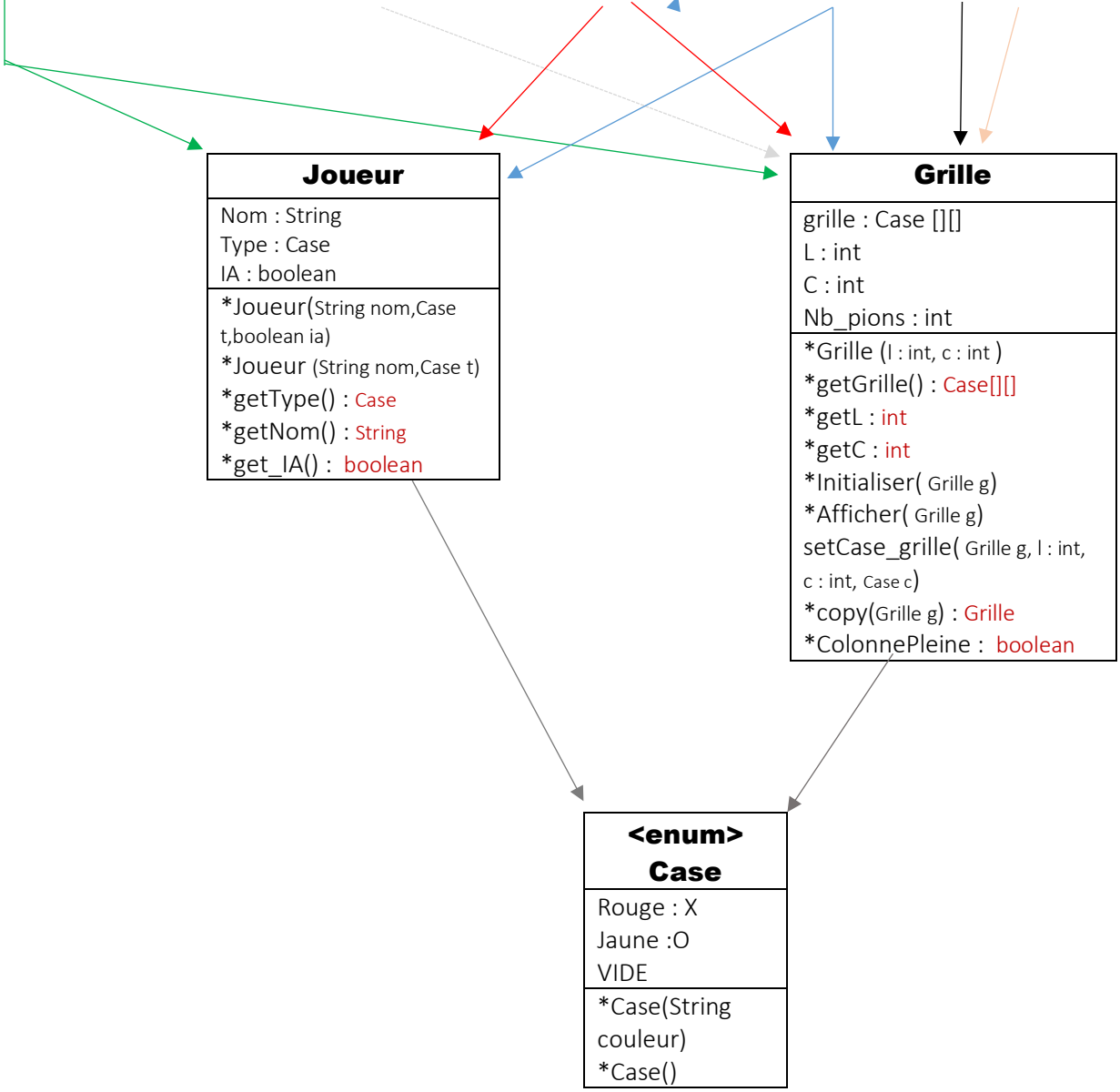
Cette classe représente l'algorithme d'élagage Alpha beta.

Diagramme de Classes:

Source de conception :

- https://eprel-v2.u-pec.fr/pluginfile.php/54135/mod_resource/content/1/POO_java_bases.pdf
- https://fr.wikipedia.org/wiki/Diagramme_de_classes#/media/Fichier:LOM_base_schema.svg





	Appel de la Classe IA
	Appel de la Classe jeu IA MinMAx
	Appel de la Classe jeu
	Appel de la Classe jeu IA simple
	Appel de la Classe Charge et Sauvegarde
	Appel de la Classe IA alpha_beta

Code pertinent :

La méthode Jouer :

```
public static boolean  Jouer(Grille g,int colonne,Joueur j){
    // on commence par verifeir que le num de colonne est valide
    if (colonne >= g.getGrille()[0].length || colonne <0){ return false ;}
    //si la colonne est pleine ,le coup n'est pas valide :
    //on verfie que la premiere case de la ligne" celle qui se trouve en plus haut "
    //n'est pas vide
    if (g.getGrille() [0][colonne]!= Case.VIDE){
        return false;

    }
    //on parcour la colonne en partant du bas (ligne.lenght-1)
    //  jusq'a trouver une case VIDE
    int ligne = g.getGrille().length-1;
    while(g.getGrille()[ligne ][colonne]!=Case.VIDE){
        ligne--;

    }
    // on remplie la premier case vide
    g.setCase_grille(g,ligne,colonne,j.getType());

    return true;
}
```

La méthode Compte :

```
public static int Compte(Grille g,int ligneDepart,int colonneDepart,int dirctLigne,int dirColonne){
    int compteur =0;
    int ligne=ligneDepart;
    int colonne=colonneDepart;
    // On part de la case de depart, et on parcours la grille dans la direction donnée
    //On implémente le compteur tant que on trouve des pions du meme joueur dans la direction donnée
    while(
    ligne >=0 &&
    ligne <g.getGrille().length
    && colonne>=0
    && colonne<g.getGrille()[ligne].length && g.getGrille()[ligne ][colonne] == g.getGrille()[ligneDepart][colonneDepart]){
        //On implémente le compteur et on avance dans la direction donnéé
        compteur++;
        ligne = ligne + dirctLigne;
        colonne= colonne + dirColonne;
        //

    }

    return compteur;
}
```

La méthode Gagnant :

```
public static boolean Gagnant(Grille g, Joueur j,int Max){
    for(int ligne =0;ligne<g.getGrille().length;ligne++){
        for(int colonne =0;colonne < g.getGrille()[ligne].length;colonne++){
            Case courant= g.getGrille()[ligne][colonne];
            if (courant.equals(j.getType())){

                if (
                // Parcour en diagonal vers le haut  droite
                (ligne >= 3 && colonne <= g.getGrille ()[ligne].length -4 && Compte(g,ligne,colonne,-1,1) >= Max ) ||
                    // Parcour horizontal vers la droite
                (colonne <= g.getGrille ()[ligne].length -4 && Compte(g,ligne,colonne,0,1) >= Max )||
                    // Parcour en diagonal vers le bas  droite
                (ligne <= g.getGrille ().length - 4 && colonne <= g.getGrille ()[ligne].length -4 && Compte(g,ligne,colonne
                    // Parcour vertical vers le bas
                (ligne <= g.getGrille ().length - 4 && Compte(g,ligne,colonne,1,0) >= Max )){
                    return true;
                }
            }

        }
    }
    return false;
}
```

La méthode Choix Colonne

```
/**
 * Renvoi la Colonne a jouer
 * @param grille La grille initialle
 * @Joueur le joueur a qui est le tour
 * @return retourne la colonne a jouer
 */
public static int Chois_Colonne(Grille grille,Joueur j ){
    //On initialise le score a une valeur qui ne sera jamais atteinte
    int Score=-999999999;
    int colonne_ajouer=1;
    //On construit l'arbre du jeu
    Arbre arb = Const_Arbre_Du_Jeu(grille,5,j);
    //On récupere sa racine
    Noeud racine = arb.getRoot();
    for(int i=0;i<racine.getFils().size();i++){
        //On affiche le score finale de chaque colonne pour verifier que le programme marche bien
        if( racine.getFils().get(i).getScore()>= Score){
            //On choisi le meilleur Score
            Score=racine.getFils().get(i).getScore();
            //On met a jour la colonne a jouer si le score est superiere au précédent

            colonne_ajouer=i;
        }
        // }
```

la méthode Construction de l'arbre après simulation et évaluation :

```
 */
public static Noeud Const_Arbre(Noeud racine,Joueur j,int profendeur){

    // Si on est pas arrivé a la derniere profendeur
    if (profendeur!=1){
        //On parcour les 7 colonne possible
        for(int i=0;i<7;i++){

            //On fait une copie profonde pour ne pas modifier le contenue de la grille
            Grille copy =Grille.copy(racine.getGrille_Noeud());
            //Jouer renvoie true si elle a reussi a jouer
            // elle verifie que la colonne n'est pas pleine et recherche la premiere case vide et joue le coup
            if (Jeu.Jouer(copy,i,j)){
                //On crée un noeud avec la Grille modifié aprés avoir jouer
                Noeud node= new Noeud(copy);
                //Une deuxieme copie pour verifier que l'adversaire ne gagnera pas du premier coup
                Grille copy2 =Grille.copy(racine.getGrille_Noeud());
                //On verifie si un des deux joueur peut gagner dés le premier coup
                //Si c'est le cas on Arrete la création de l'arbre
                //On commence d'abord par verifier La machine ,si elle ne gagne pas on verifei l'Adversaire
                //Profendeur 4 pour dire que c'est le premier coup a jouer
                if(Jeu.Gagnant(copy,j,4)&& profendeur==5){
                    //On met le score a une valeur Maximale pour le choisir
                    if(j.get_IA()==true)node.setScore(101000);
                }
                else {
                    if (Jeu.Jouer(copy2,i,inverse(j))){
                        // On met le score a une valeur Maximale pour le choisir
                        if(Jeu.Gagnant(copy2,inverse(j),4)&& profendeur==5){node.setScore(10000);
                        }
                    }
                }
                // Si aucun des deux joueurs ne peut gagner,On Crée L'arbre avec la reccurence
                if (Jeu.Gagnant(copy2,inverse(j),4)==false && Jeu.Gagnant(copy,j,4)==false ){
                    //on va jusqu'a la profendeur pour appeller la fonction d'evaluation
                    node=Const_Arbre(node,inverse(j),profendeur-1);
                    //On initalise le Score avec le Max de ses fils si c'est la machine
                    //le Score de la profendeur sera remonter
                    if( j.get_IA()==true){
                        node.setScore(Max(node.getFils()));

                    } else  {
                        node.setScore(Min(node.getFils()));

                    }

                }
                // Ajoute le Noeud avec son Score a la racine

            } else {
                //si la colonne est pleine
                if(profendeur==5)
                    racine.Add_Fils(new Noeud (Grille.copy(racine.getGrille_Noeud()),-90000));
                //si la colonne est pleine a partir d'une certaine profendeur
                //donc c'est une feuille;on appelle la fonction d'evaluation
                else {racine.Add_Fils(new Noeud (Grille.copy(racine.getGrille_Noeud()),Evaluat(racine.getGrille_Noeud())));

            }
        }
    }
    else {
        // Arriver a la profendeur
        for(int i=0;i<7;i++){
            Grille copy =Grille.copy(racine.getGrille_Noeud());
            //Jouer renvoie true si elle a reussi a jouer
            // elle verifie que la colonne n'est pas pleine et recherche la premiere case vide et joue le coup
            if (Jeu.Jouer(copy,i,j)){
                // On evalue les 7 possibilités
                //Pour cela on lui passe la fonction d'evaluation qui renverrai son score
                Noeud node= new Noeud(copy, Evaluat(copy));
                //ajoute le noeud evaluer aux fils du parent
                racine.Add_Fils(node);
```

La méthode Evaluation :

```
public static int Evaluat(Grille grille){
    int fourInLine = 1000;
    int threeInLine = 16;
    int twoInLine =6;
    int Xlines = 0;
    int Olines = 0;

    // Checking rows
    for(int i = 0; i < 6; i++){
        for(int j = 0; j < 3; j++){
            int s = 0;

            for(int count = 0; count < 4; count++){
                if(grille.getGrille()[i][j + count] == grille.getGrille()[i][j + count + 1] && grille.getGrille()[i][j + co
                    s++;
                }
                else if( (grille.getGrille()[i][j + count] !=grille.getGrille()[i][j + count + 1] && grille.getGrille()[i][
                    if(grille.getGrille()[i][j + count] == Case.X){
                        if(s == 1){
                            Xlines = Xlines + twoInLine;
                        }
                        else if(s == 2){
                            Xlines = Xlines + threeInLine;
                        }
                        else if(s == 3){
                            Xlines = Xlines + fourInLine;
                        }
                    }
                    else if(grille.getGrille()[i][j + count] == Case.O){
                        if(s == 1){
                            Olines = Olines + twoInLine;
                        }
                        else if(s == 2){
                            Olines = Olines + threeInLine;
                        }
                        else if(s == 3){
                            Olines = Olines + fourInLine;
                        }

                    }

                    s = 0;
                }
                if(grille.getGrille()[i][j + count] == Case.VIDE){
                    Olines = Olines + twoInLine;
                    }
                    else if(s == 2){
                        Olines = Olines + threeInLine;
                    }
                    else if(s==3 ){
                        Olines = Olines + fourInLine;
                    }

                }
                s = 0;
            }
            if(grille.getGrille()[i - count][j - count] == Case.VIDE){
                s = 0;
            }
        }
    }

    return Olines - Xlines;
}
}
```

La méthode Alphabeta

```
private static int alphabeta(Grille grille, Joueur joueur, int profondeur){
    int  alpha =-99999;
    int  beta=99999;
    return min(grille, joueur, profondeur, alpha, beta);
}
/**
```

La méthode max d'alphabeta :

```
/**
 * Applique la partie max de minmax
 */
private static int  max(Grille grille, Joueur joueur, int profondeur, double alpha, double beta){
    if(profondeur != 0){
        int valeurDeJeu =-99999;
        for(int i=0; i < grille.getC(); i++){
            Grille copy = grille.copy(grille);
            if(Jeu.Jouer(copy,i,joueur)){

                valeurDeJeu = Math.max(valeurDeJeu, min(copy, joueur, profondeur-1, alpha, beta));

                if(valeurDeJeu >= beta){
                    return valeurDeJeu; // Coupure beta
                }
                alpha = Math.max(alpha, valeurDeJeu);

            }
        }
        return valeurDeJeu;
    }else{
        return IA.Evaluat(grille);
    }
```

La méthode min d'alphabeta :

```
 * Applique la partie min, de minmax
 */
private static  int  min(Grille grille, Joueur joueur,  int profondeur, double alpha, double beta){
    if(profondeur != 0){
        int  valeurDeJeu =99999;
        for(int i=0; i <7 ; i++){
            Grille copy = grille.copy(grille);
            if(Jeu.Jouer(copy,i,joueur)){
                valeurDeJeu = Math.min(valeurDeJeu, max(copy, joueur, profondeur-1, alpha, beta));
                if(alpha >= valeurDeJeu){
                    return valeurDeJeu; // Coupure alpha
                }
                beta = Math.min(beta, valeurDeJeu);
            }
        }
        return valeurDeJeu;
    }
    else{
        return IA.Evaluat(grille);
    }
}
```

La méthode Demande et joue :

```
public static void demandeEtJoue(Grille g, Joueur j){
    Scanner s = new Scanner(System.in);
    boolean valide ;
    int colonne ;
    Random r = new Random () ;
    // On commence par d'abord lire le nombre et puis verifier sa validité
    do {
        System.out.println(j.getNom()+ " : Entrez un numéro d'une colonne ");
        if (j.getNom().equals("IA_Alpha")){colonne =AlphaBeta.calculeColonneAJouer(g,j); System.out.println(colonne);}
        else
        {if (j.getNom().equals("IA_Max")){colonne =Minmax.Chois_Colonne(g,j); System.out.println(colonne);}
            else {if (j.getNom().equals("IA")){colonne =r.nextInt(g.getGrille()[0].length)+1;System.out.println(colonne);}e
        }
        // Pour les joueur ,la premiere case de la  grille du jeu commence par 1.
        //Mais les indince en java commence par 0 , donc on soustrait "1" du nombre entré.
        colonne--;
        valide=Jouer(g,colonne,j);
        if (!valide){System.out.println(" le coup n'est valide");}

    }
    while(!valide);
    // Le coup est valide : on affiche la grille!
```

La méthode Play :

```
public static void Play (int choix){

    Joueur j1 = new Joueur("Joueur 1",Case.X);
    Joueur j2 = new Joueur("Joueur 2",Case.O);
    Scanner s = new Scanner(System.in);
    Grille tmp = null ;
    int C=0;
    int L=0;
    int X=0;

    // choix represente le type de partie jouer
    // 1 : Joueur vs Joueur
    // 2 : Joueur vs IA simple
    // 3 : Joueur vs IA Minmax
    // 4 : Joueur vs IA alphabeta

    if (choix == 1){Joueur humain = new Joueur("Joueur 2",Case.O,false);
        j2 = humain ;}
    else {if (choix==2){Joueur ia = new Joueur("IA",Case.O,true);
            j2 = ia ;} }

    boolean gagne ;

    Grille grille = new Grille(6,7);
    grille.Initialiser(grille);
    Jeu partie = new Jeu(j1,j2,grille,4);

    //Une variable temporaire pour alterner les joueurs
    Joueur j=j1; ;

    if (choix == 1){tmp  = Charge.load("Save_Joueur");}
    else{if (choix==2){ tmp  = Charge.load("Save_IA");}
    }

    if (tmp != null){
        if (!(Pleine(tmp) || Gagnant(tmp,j1,4) || Gagnant(tmp,j2,4)) ){

            System.out.println("Voulez vous Continuer La Partie Précédente.");
            System.out.println("1. Continuer");
            System.out.println("2. Nouvelle partie");
            int nbr= s.nextInt();
            while( nbr!=1 && nbr != 2){

                System.out.println("Entrer un nombre valide");
                nbr=s.nextInt();
            }
```

```
        }
        if (nbr==1) {
            grille=tmp;
            int cpt_j1=0;
            int cpt_j2=0;
            System.out.print(tmp.getL()+"*"+tmp.getC()+"  ");
            for (int l = tmp.getGrille().length-1;l>=0; l--){
                for(int c =0 ; c < tmp.getGrille()[l].length;c++ ) {
                    if (tmp.getGrille()[l][c]==Case.X){System.out.print("X");cpt_j1=cpt_j1+1;}
                    else {if (tmp.getGrille()[l][c]==Case.O)
                        {System.out.print("O");cpt_j2=cpt_j2+1;} }
                }
                System.out.print(" | ");
            }
            if (cpt_j1 >cpt_j2) {j=j2;} else {j=j1;}
            grille.Afficher(grille);

        }
        if (nbr == 2){do{
                System.out.println("Entrez le nombre de Ligne") ;
                L = s.nextInt() ;
                System.out.println("Entrez le nombre de Colonne") ;
                C = s.nextInt() ;
                if (L < 4) {System.out.println ("Ligne doit etre superieure a 4");
                    System.out.println("Entrez le nombre de Ligne") ;
                    L = s.nextInt() ;
                }
                if (C < 4) {System.out.println ("Colonne doit etre superieure a 4");
                    System.out.println("Entrez le nombre de Colonne") ;
                    C = s.nextInt() ;}

            }while(L < 4 || C < 4);
        }
    }
}
if (tmp == null ){do{
        System.out.println("Entrez le nombre de Ligne") ;
        L = s.nextInt() ;
        System.out.println("Entrez le nombre de Colonne") ;
        C = s.nextInt() ;
        if (L < 4) {System.out.println ("Ligne doit etre superieure a 4");
            L = s.nextInt() ;
        }
        if (C < 4) {System.out.println ("Colonne doit etre superieure a 4");
            System.out.println("Entrez le nombre de Colonne") ;
            C =s.nextInt() ;}

    }while(L < 4 || C < 4);
}
if (C>= 4 && L>= 4 ){ grille = new Grille(L,C);
    grille.Initialiser(grille);
    partie = new Jeu(j1,j2,grille,4);}
do {

    demandeEtJoue(grille,j);

    grille.Afficher(grille);
    gagne = Gagnant(grille,j,4);

    if (choix == 1){Sauvegarde.Save(grille,"Save_Joueur");}
    else{if (choix==2){Sauvegarde.Save(grille,"Save_IA");}
    }
    //On alterne les joueur
    if(j.equals(j1)){

} while(!gagne && !Pleine(grille));
if (gagne){
    System.out.println () ;
    for (int i = 0; i < 2 ; i ++){for(int c =0 ; c < 15 ; c++){System.out.print ("**") ; } System.out.println () ;
    }
    System.out.println () ;
    //On verifie le joueur gagnant sachant qu'on a fait une alternance aprés la victoire du gagnant
    if(j.equals(j1)){
        System.out.println("      VICTOIRE DE " + j2.getNom()+" ;) ");} else
    {System.out.println("      VICTOIRE DE " + j1.getNom()+" ;)");}
    System.out.println () ;
    for (int i = 0; i < 2 ; i ++){for(int c =0 ; c < 15 ; c++){System.out.print ("**") ; } System.out.println () ;
    }
    System.out.println () ;
    System.out.println ("Voulez vous refaire une partie ?");
    System.out.println ("1- OUIIII");
    System.out.println ("2- NON");
    X = s.nextInt();

    // si la partie est finie on suprime la sauvegarde
    if (choix == 1){Sauvegarde.Save(null,"Save_Joueur");}
    else{if (choix==2){Sauvegarde.Save(null,"Save_IA");}
    }
```