TP 2 : Algorithme Alpha-Beta

Contexte : Jeu Puissance 3

Le but de ce TP est de programmer une intelligence artificielle permettant à un joueur d'affronter une machine dans un jeu simplifié basé sur Puissance 4. On nommera ce jeu Puissance 3.

Tour à tour, les joueurs placent un pion dans la colonne de leur choix. Le but est d'aligner 3 pions en horizontal ou en vertical sur une grille de taille $5 * 5$. (pour simplifier, on ne considère pas l'alignement en diagonal).

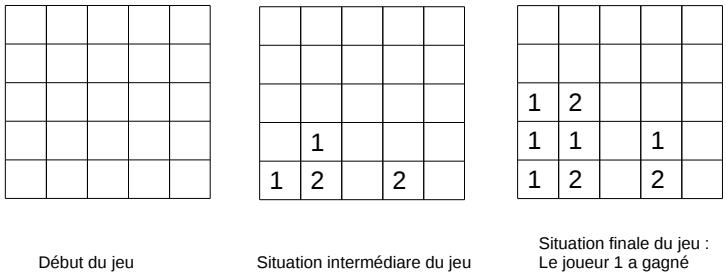


FIGURE 1 – Exemple de jeu Puissance 3

L'algorithme implémenté devra être basé sur le principe d'Alpha-Beta vu en cours.

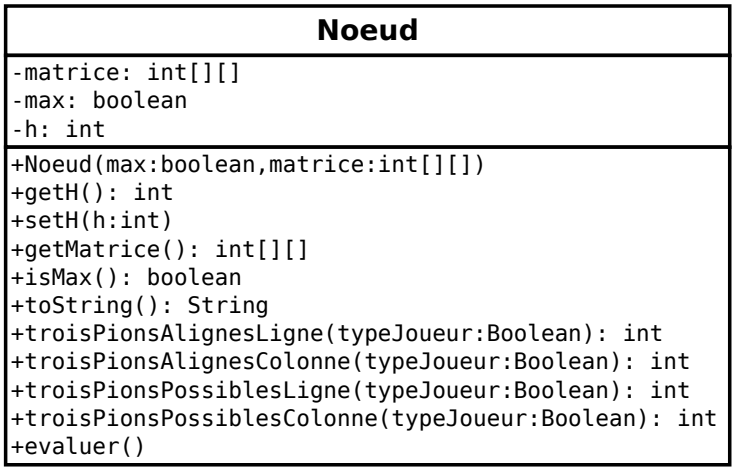
Implémentation

Votre projet sera constitué de 3 principales classes :

1. La classe **Noeud** qui correspond à un état donné du jeu.
2. La classe **Puissance3** implémente l'algorithme alpha-beta.
3. La classe **Main** pour simuler une partie du jeu "humain Vs machine".

Détails de l'implémentation

La classe Noeud



— **Attributs :**
  — *matrice* : représente la grille d'une situation donnée du jeu.
  — *max* : un booléen qui est égal à *true* si le noeud est de type max et à *false* si le noeud est de type min.
  — h : c'est l'évaluation du noeud. Un score positif représente une situation du jeu favorable pour le joueur *max* et un score négatif représente une situation favorable pour le joueur *min*.
— **Méthodes :**
  — $Noeud(max, matrice)$: le constructeur permet d'initialiser les attributs grâce aux valeurs passées en paramètres.
  — Les méthodes $getH()$, $setH()$, $getMatrice()$, $isMax$ sont de simples assesseurs en lecture/écriture. La méthode *toString* devrait être implémentée pour afficher les attributs du noeud.
  — $troisPionsAlignesLigne(typeJoueur)$: retourne un score égal à 1000 si 3 pions sont alignés en ligne pour $typeJoueur$ sinon elle retourne 0.

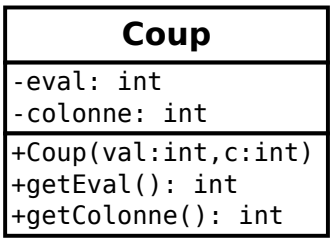
— *troisPionsAlignesColonne*(*typeJoueur*) : même principe que *troisPionsAlignesLigne* sauf qu'elle analyse les alignements en colonne.
— *troisPionsPossiblesLigne*(*typeJoueur*) : Cette méthode évalue la possibilité d'aligner 3 pions pour le joueur *typeJoueur*. Une façon de faire cette évaluation est de cumuler un score égal à 200 à chaque fois qu'il y a 2 pions de *typeJoueur* sur la même ligne accolés à une case vide et un score égal à 30 si 1 pion de *typeJoueur* est accolé à deux cases vides.
— *troisPionsPossiblesColonne*(*typeJoueur*) : même principe que *troisPionsPossiblesLigne* sauf qu'elle analyse les alignements en colonne.
— La méthode *evaluer*() permet d'évaluer une situation donnée avec la formule suivante :

```
h = -1*troisPionsAlignesLigne(false) + troisPionsAlignesLigne(true)
-1*troisPionsAlignesColonne(false) + troisPionsAlignesColonne(true)
-1*troisPionsPossiblesLigne(false) + troisPionsPossiblesLigne(true)
-1*troisPionsPossiblesColonne(false) + troisPionsPossiblesColonne(true)
```

— Testez vos méthodes sur la situation intermédiaire du jeu de la Figure 1. Voici un exemple de résultats qu'on pourrait obtenir :

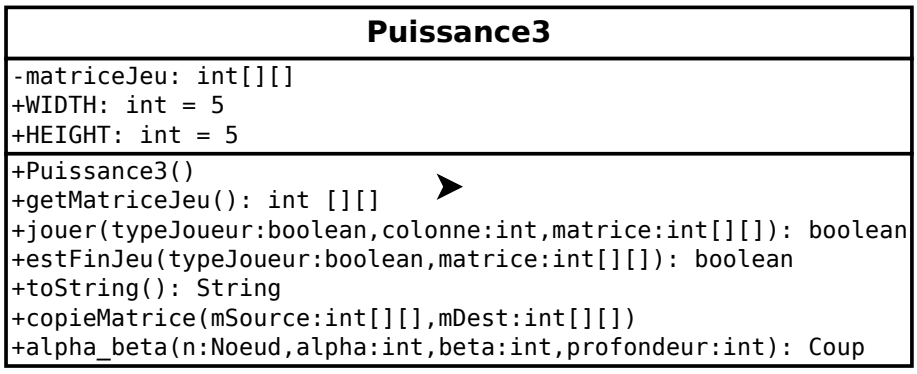
```
0|0|0|0|0|
0|0|0|0|0|
0|0|0|0|0|
0|1|0|0|0|
1|2|0|2|0|
------------

troisPionsPossiblesLigne(true)=60
troisPionsPossiblesLigne(false)=230
troisPionsPossiblesColonne(true)=60
troisPionsPossiblesColonne(false)=30
evaluation = -140
```

La classe Coup



la classe Coup représente les informations sur le coup devant être joué. Un objet de type *Coup* est retourné par l'algorithme Alpha-Beta. Un coup = l'évaluation du noeud + le numéro de la meilleure colonne à jouer.

La classe Puissance3

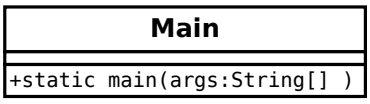


- **Attributs :**
  - $martriceJeu$: la matrice du jeu sur la quelle s'affronte les deux joueurs .
  - $WIDTH$ et $HEIGHT$: les dimensions de la matrice. On fera les premiers tests avec une grille de taille $5 * 5$
- **Méthodes :**
  - $Puissance3()$: le constructeur doit instancier la matrice du jeu correctement.
  - $getMatrice()$: retourne la matrice du jeu.
  - $jouer(typeJoueur, j, matrice)$: cette méthode permet de jouer le coup d'un joueur (humain ou ordinateur) à la colonne j sur la matrice passée en paramètre. La méthode retourne $false$ si le coup n'est pas faisable sinon elle retourne $true$.
  - $estFinJeu(typeJoueur, matrice)$: la méthode retourne $true$ si la partie du jeu s'est terminée (grille remplie ou bien le joueur passé en paramètre a gagné).
  - $alpha\_beta(n, alpha, beta, profondeur)$: implémente l'algorithme Alpha-Beta et retourne le meilleur coup possible pour l'ordinateur. On testera une profondeur de l'arbre égale à 4.

Algorithm 1: Algorithme Alpha-Beta

```
Data: n : Noeud, α : int , β : int, profondeur : int
Result: meilleurCoup
if (profondeur == 0) ou finJeu(!n.isMax()) then
    noeud.evaluer() ;
    meilleurCoup = Coup(noeud.getH(),-1) ;
    return meilleurCoup;
if n.isMax() then
    foreach colonne j do
        m =jouer la colonne j sur une copie de la matrice du noeud n ;
        successeur = créer un noeud successeur de n avec la matrice m;
        coup = Alpha-Beta(successeur, α , β, profondeur − 1) ;
        if coup.getEval() > α then
            α = coup.getEval() ;
            bestj = j ;
        if α >= β then
            return coup(α, bestj) ;
    return coup(α, bestj) ;
else
    A compléter : n est un noeud MIN
```

La classe Main



Simuler dans une classe Main une partie du jeu Humain Vs Programme Alpha_Beta. Testez les deux possibilités : l'humain qui commence en premier, Alpha_Beta qui commence.

Pour aller plus loin

Adaptez le code pour réaliser le jeu du Puissance 4 selon les régles de la page wikipedia : `https://fr.wikipedia.org/wiki/Puissance_4`.

Dorénavant, les diagonales sont à prendre en compte et la grille compte 6 lignes et 7 colonnes. Le plus gros du travail consiste à définir les situations de danger en ligne, colonne

et diagonale. Les situations de dangers sont de plusieurs types : "danger immédiat" si au prochain coup le joueur peut aligner 4 pions ; "danger possible" si au coup suivant, le joueur peut aligner 3 pions. Il est possible d'ajouter plus de niveaux. A chaque niveau de danger correspond un score, l'évaluation d'une situation consiste à cumuler les valeurs des dangers associés.