TP 2 : IA qui joue à Puissance 4

Contexte :

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).

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

Implémentation

Votre projet sera constitué de 2 principales classes :

1. La classe **Noeud** qui correspond à un état donné du jeu.
2. La classe **Puissance3** qui implémente une IA basée sur le principe alpha-beta qui joue contre un adversaire

Détails de l'implémentation

La classe Noeud

- **Attributs :**
  - *grille* : représente la grille d'une situation donnée du jeu.
  - $isMax$: 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 :**

- $\_\_init\_\_(self, grille, isMax = True)$: le constructeur permet d'initialiser les attributs grâce aux valeurs passées en paramètres.
- $\_\_str\_\_$: affiche la grille du noeud et son évaluation.
- $troisPionsAlignesLignes(self, pion)$: évalue la possibilité d'aligner 3 pions sur la même ligne. La méthode analyse chaque 3 cases consécutives et cumule un score égal à ;
  - 10000 si 3 pions sont alignés en ligne
  - 200 à chaque fois qu'il y a 2 pions de $typeJoueur$ sur la même ligne accolés à une case vide
  - 30 si un pion est accolé à 2 cases vides.
- $troisPionsAlignesColonnes(self, pion)$: même principe que $troisPionsAlignesLignes$ sauf qu'elle analyse les alignements en colonne.
- La méthode $evaluer()$ permet d'évaluer une situation donnée avec la formule suivante :

```
poids = -1
self.h += self.troisPionsAlignesLignes(PION_MAX)
self.h += poids * self.troisPionsAlignesLignes(PION_MIN)
self.h +=  self.troisPionsAlignesColonnes(PION_MAX)
self.h +=  poids *  self.troisPionsAlignesColonnes(PION_MIN)
```

- $finJeu(self)$: la méthode retourne $true$ si la partie du jeu s'est terminée (grille remplie ou bien victoire).
- Testez vos méthodes sur la situation intermédiaire du jeu. 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|
------------

evaluation = -140
```

La classe Puissance3

— **Attributs :**

- $grilleJeu$: la grille 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 :**
  — $\_\_init\_\_$: le constructeur doit instancier la grille du jeu (grille vide).
  — $\_\_str\_\_$: affiche la grille du jeu
  — $jouerColonne(self, pion, c, grille)$: cette méthode permet de jouer le coup d'un joueur (humain ou IA) à la colonne c sur la grille passée en paramètre. La méthode retourne $false$ si le coup n'est pas faisable sinon elle retourne $true$.
  — $alpha\_beta(self, noeud, alpha, beta, profondeur)$: implémente l'algorithme Alpha-Beta et retourne le meilleur coup possible pour l'IA.

Algorithm 1: Algorithme Alpha-Beta

Data: $n : Noeud$, $\alpha : int$, $\beta : int$, $profondeur : int$
Result: $meilleurCoup$
if $(profondeur == 0)$ *ou* $n.finJeu()$ **then**
    n.evaluer() ;
    return (n.h, -1);
if $n.isMax$ **then**
    foreach *colonne* c **do**
        m =jouer la colonne c sur une copie de la matrice du noeud n ;
        $successeur$ = créer un noeud successeur de n avec la matrice m;
        $eval, c$ = Alpha-Beta($successeur$, α , β, $profondeur - 1$) ;
        if $eval > \alpha$ **then**
            $\alpha = eval$;
            $bestc = c$;
        if $\alpha >= \beta$ **then**
            return coup($\alpha, bestc$) ;
    return $(\alpha, bestc)$;
else
    A compléter : n est un noeud MIN

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.