Oriane CALVAR IND
Emeric ARHANT

Rapport projet final DIA : Puissance 4

1) Présentation du cahier des charges :

Pour ce projet nous devions réaliser un jeu de Puissance 4 en python, où 2 joueurs pouvaient s'affronter entre eux ou bien 1 joueur pouvait affronter une IA utilisant l'algorithme min max avec élagage alpha beta.
Nous devions de plus avoir la possibilité de choisir qui de l'IA ou du joueur commence la partie sur un plateau de 12 colonnes.
Pour finir, il fallait trouver le bon compromis entre temps d'exécution et efficacité concernant l'algorithme de notre IA. La rapidité d'exécution de l'IA étant un facteur important pour la notation, notre programme doit de ce fait pouvoir mesurer le temps que mets l'IA à placer son coup.

2) Architecture du code :

Pour réaliser ce projet nous avons fait le choix d'utiliser le système de classes présent dans le langage python. Nous avons ainsi fait le choix de séparer notre programme en 3 classes :

```
###Classe representant un jeton caractérisé par sa position en x et y ainsi que sa couleur. Un
jeton de joueur sera jaune ou rouge, un jeton blanc est la valeur par défaut
class Jeton:

    def __init__(self, ligne: int= None, colonne: int= None, couleur: str= "blanc")
    def __str__(self)

###Classe representant le plateau de jeu, par défaut un plateau est rempli de jeton blanc
(vide), il est possible de créer un plateau à partir d'un historique (list) de jeton
class Plateau:
    def __init__(self, historique: list = [], nbLignes: int= 6, nbColonnes: int= 12)

    #méthode permettant l'affichage du plateau
    def __str__(self)

    #méthode permettant d'ajouter un jeton au plateau dans la colonne souhaitée si c'est
possible
    def jouerJeton(self, colonne: int, couleur: str)

    #méthode retirant le dernier jeton joué présent dans l'historique
    def annulerCoup(self)

    #méthode pour obtenir le nombre de jeton restant à jouer
    def nombreJetons(self)

    #méthode donnant l'état de la partie: 2-> plus de jeton disponible, 1-> partie gagné par un
joueur, 0->partie continue, -1-> erreur
    def etat(self, jeton: Jeton)

#classe décrivant un joueur, caractérisé par le fait qu'il soit IA ou non et s'il commence ou
non, par défaut le joueur est humain et ne commence pas (jeton jaune)
class Player:
    def __init__(self,start=False, ia=False)

    #méthode faisant le processus d'un tour du jeu pour un joueur humain ou IA
    def jouer(self, plateau: Plateau)
```

Nous avons aussi créé un fichier comportant des fonctions relatives à l'IA :

```
#Algo min max avec elagage alpha beta
def minimax_alpha_beta(plateau: Plateau, couleur: str, couleurIA: str, depth: int,
maximizingPlayer: bool, alpha: float, beta: float,depthMax = 5)

#methode retournant une liste des colonnes où il est possible de jouer
def coupsPossibles(plateau: Plateau)

#methode retournant la couleur des jetons de l'adversaire
def getCouleurAdverse(couleur: str)

#methode calculant l'heuristique en utilisant les fonctions generatrice de points ci dessous
def heuristicScore(plateau: Plateau, couleurIA: str)

#generateur de points :
#on compte sur tout le plateau le nombre de groupe de 2 ou 3 jetons alignés, une fois ces
nombres trouvés on applique un coef multiplicateur (1 pour les groupes de 2 et 5 pour les grp de
3
def ptsVertical(plateau: Plateau, couleurIA: str, multiplicateur: int)
def ptsHorizontal(plateau: Plateau, couleurIA: str, multiplicateur: int)
def ptsDiagonale(plateau: Plateau, couleurIA: str, multiplicateur: int)

#méthode executant le processus de jeu de l'IA
def iaJouerJeton(plateau, couleur: str)

#décorateur calculant le temps d'execution d'une fonction
def calculTempsExec(fonction)
```

Construction du code dans le cadre d'une partie humain vs IA

1) Création d'un objet Plateau() vide (rempli de jeton blanc)
2) Choix du mode de jeu (2 : IA vs humain) puis du joueur qui commence (1 : humain / 2 : IA)
3) Création de deux objets Player(), un pour l'IA un pour l'humain puis entrée dans la boucle de jeu :

```
    while True: # Boucle de jeu
        joueur1.jouer(plateau)
        if plateau.etat(plateau.historique[-1]) == 2:
            print("Match nul !")
            break
        elif plateau.etat(plateau.historique[-1]) == 1:
            print("Le joueur {} a gagné !".format(joueur1.couleur.upper()))
            break

        joueur2.jouer(plateau)
        if plateau.etat(plateau.historique[-1]) == 1:
            print("Le joueur {} a gagné !".format(joueur2.couleur.upper()))
            break
        elif plateau.etat(plateau.historique[-1]) == 2:
            print("Match nul !")
            break
```

Oriane CALVAR IND
Emeric ARHANT

Chaque joueur joue à tour de role, après chaque coup on verifie l'etat du plateau. La boucle de jeu ne se termine qu'une fois que la methode plateau.etat(plateau) renvoie 1 ou 2.

Explication de méthodes clés :

```
joueur.jouer(plateau)
```

➔ Cette méthode permet le placement d'un jeton dans le plateau mit en argument. Elle s'applique aussi bien sur un joueur de type humain qu'AI. Seulement en fonction du type de joueur elle ne fera pas la meme chose :

- Si c'est à un humain de jouer : on demande au joueur d'entrer un numero de colonne valide pour placer un jeton, on ajoute ce jeton dans le plateau en changeant la valeur du jeton par défaut blanc à la position souhaitée par un jeton de la couleur du joueur (on ajoute le jeton dans l'historique aussi), puis on affiche le nouveau plateau.

- Si c'est l'IA de jouer : on execute la methode iaJouerJeton(plateau, couleur), qui appelle la methode minmax_alpha_beta() permettant de calculer le meilleur coup possible. Une fois ce coup trouvé on appelle la methode plateau.jouerJeton() et on place le jeton dans la colonne trouvée puis on affiche le nouveau plateau.

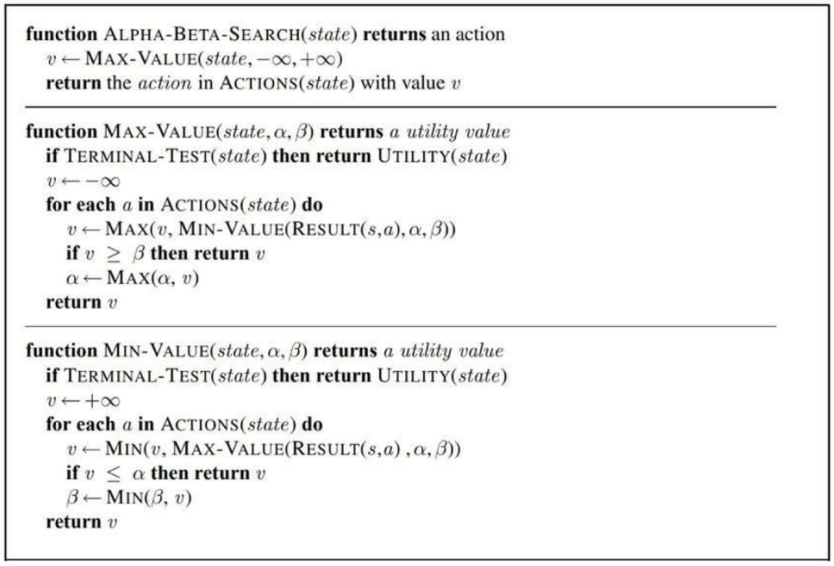
```
plateau.etat(plateau.historique[-1])
```

➔ Cette méthode permet d'obtenir l'etat du plateau (partie remportée par un joueur, match nul car le plateau est rempli ou encore partie en cours). Cependant afin d'optimiser ce processsus qui peut etre gourmand en ressources, nous n'explorons pas tout le plateau à la recherche de combinaison gagnante. Etant donné que l'état est vérifié apres chaque coup, il est plus judicieux de ne verifier la présence de combinaison gagnante seulement sur la ligne, colonne, diagonales du dernier jeton joué.

```
def minimax_alpha_beta(plateau: Plateau, couleur: str, couleurIA: str, depth: int,
maximizingPlayer: bool, alpha: float, beta: float,depthMax = 5):
```

➔ Cette méthode utilise le principe de l'algorithme minmax avec elagage alpha beta :

Cet algorithme est utilisé dans les jeux à deux joueurs pour trouver le meilleur coup à jouer. Il explore l'arbre des possibilités tout en éliminant les branches non prometteuses. Les valeurs alpha beta servent de bornes pour couper les branches inutiles. Cela permet d'accélérer la rechercher du meilleur coup en réduisant le nombre de nœuds explorés.



function ALPHA-BETA-SEARCH($state$) **returns** an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the $action$ in ACTIONS($state$) with value v

---

function MAX-VALUE($state, \alpha, \beta$) **returns** a $utility$ $value$
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE($state, \alpha, \beta$) **returns** a $utility$ $value$
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

Oriane CALVAR IND
Emeric ARHANT

3) Notre heuristique :

Le rôle de l'heuristique est de calculer un score basé sur des conditions sensées caractérisées le potentiel de victoire d'un coup. Ainsi pour chaque coup potentiel un score sera calculé et le coup ayant le score le plus haut sera retenu. Nous avons fait le choix de faire une heuristique simple mais efficace dans le cadre de notre puissance 4 :

Elle compte le nombre de groupes de 2 ou 3 jetons alignés dans les directions verticales, horizontales et diagonales.

L'idée principale est d'attribuer un score plus élevé aux positions comportant des groupes de 3 jetons alignés, car cela indique une plus grande probabilité de gagner. Les groupes de 2 jetons alignés sont également pris en compte, mais avec un poids multiplicateur plus faible.

L'heuristique parcourt le plateau et compte le nombre de groupes de jetons alignés dans chaque direction. Elle maintient une variable "inARow" qui est augmentée chaque fois qu'un jeton de la couleur de l'IA est trouvé et réinitialisée lorsqu'un jeton d'une autre couleur est rencontré.
Si "inARow" atteint la valeur 3, cela signifie qu'un groupe de 3 jetons alignés a été trouvé, donc on incrémente la variable "trois" et on diminue la variable "deux" (car un groupe de 3 implique également un groupe de 2). Si "inARow" atteint 2, on incrémente la variable "deux".

Finalement, la fonction renvoie le score total en multipliant le nombre de groupes de 2 par le multiplicateur correspondant (ici 1) et le nombre de groupes de 3 par le multiplicateur correspondant (ici 5).

4) Nos améliorations

- Pour chaque entrée de texte de la part d'un utilisateur, la saisie est sécurisée. On redemande à l'utilisateur de saisir une valeur tant que celle-ci ne correspond aux critères demandés. Cela permet d'éviter les erreurs de type ou d'array.

Exemple : Dans le choix de la colonne, l'utilisateur doit entrer un nombre (int) entre 1 et 12, tant que l'input n'est pas de type int ou entre 1 et 12, le choix de l'utilisateur ne sera pas validé.

- Pour une lecture plus aisée du plateau de jeu, des couleurs ont été utilisées pour illustrer les jetons
- Il est possible de terminer l'exécution du programme de façon simple à tout moment de la partie en tapant « 555 » dans une entrée de texte
- Durant nos tests nous avions remarqué que l'IA ne faisait pas en sorte de terminer la partie rapidement. En effet à partir du moment ou elle trouvait un coup gagnant ou perdant, la valeur « inf » (ou -inf) était attribué comme score peu importe le nombre de coup qu'il aura été nécessaire. Un coup gagnant au tour suivant n'était donc pas prioritaire sur un coup gagnant en 5 coups par exemple. De ce fait nous avons décidé de remplacer inf par « 1000 / (depthMax+1-depth) » ce qui a pour conséquence d'obtenir un score plus élevé pour un coup gagant plus rapide.