Projet IA - jeu Puissance 4
GRAU Quentin
GORKE Océane
HATOUM Michelle
JACQUIER Adrien

RAPPORT PROJET FINAL
Les Pythoniens

- Explication des librairies

```
1 import time
2 import math
3 import random
4 import numpy as np
```

Voici les librairies importées afin de réaliser notre code.
La librairie time a permis de mesurer le temps d'exécution de l'algorithme AlphaBeta afin que l'IA puisse placer son pion de manière la plus optimale possible dans la grille.
La librairie math a permis d'obtenir les valeurs +infini et -infini.
La librairie random, mentionnée dans l'explication de la fonction AlphaBeta
La librairie numpy a permis d'initialiser la grille de jeu à une grille vide remplie de 0.

- Explication des méthodes

» *Variables utilisées*

```
 7 ia = 2
 8 humain = 1
 9 tour_ia=0
10 tour_humain=1
11 NB_COLONNES = 12
12 NB_LIGNES = 6
13 TAILLE_FENETRE = 4
14 VIDE = 0
15 CRED = '\33[31m'
16 CEND = '\033[0m'
17 CBLUE   = '\33[34m'
```

Nous avons décidé de stocker ces variables en dehors de toutes les fonctions car elles permettent de les utiliser plus correctement lors de l'écriture du code.

» *Fonction **est_tour_gagnant(s, joueur)***

```python
def est_tour_gagnant(s, joueur):
    # Horizontales
    for c in range(NB_COLONNES-3):
        for r in range(NB_LIGNES):
            if s[r][c] == joueur and s[r][c+1] == joueur and s[r][c+2] == joueur and s[r][c+3] == joueur:
                return True

    # Verticales
    for c in range(NB_COLONNES):
        for r in range(NB_LIGNES-3):
            if s[r][c] == joueur and s[r+1][c] == joueur and s[r+2][c] == joueur and s[r+3][c] == joueur:
                return True

    # Diagonales positives
    for c in range(NB_COLONNES-3):
        for r in range(NB_LIGNES-3):
            if s[r][c] == joueur and s[r+1][c+1] == joueur and s[r+2][c+2] == joueur and s[r+3][c+3] == joueur:
                return True

    # Diagonales négatives
    for c in range(NB_COLONNES-3):
        for r in range(3, NB_LIGNES):
            if s[r][c] == joueur and s[r-1][c+1] == joueur and s[r-2][c+2] == joueur and s[r-3][c+3] == joueur:
                return True
```

La fonction **est_tour_gagnant(s, joueur)** prend en argument la grille ainsi que le joueur dont c'est le tour. Elle retourne True si le joueur en question gagne à ce tour là et False sinon.
Elle va parcourir toute la grille et va vérifier si le joueur a aligné 4 pions de manière verticale, horizontale ou diagonale.

» *Fonction **est_noeud_terminal(s)***

```python
def est_noeud_terminal(s):
    return est_tour_gagnant(s, humain) or est_tour_gagnant(s, ia) or cases_libres(s) == 30
```

La fonction **est_noeud_terminal(s)** prend en argument la grille. Elle renvoie True si la partie est terminé à ce tour et non sinon. Elle va tester si le tour est gagnant pour le joueur 1, pour le joueur 2 ou s'il y a égalité.

» *Fonction **score_position(s, joueur)***

```
74
75 def score_position(s, joueur):
76     score = 0
77
78     #on modifie le score selon les pions dans la colonne centrale
79     centre_array = [int(i) for i in list(s[:, NB_COLONNES//2])]
80     centre_count = centre_array.count(joueur)
81     score += centre_count * 3
82
83     #on modifie le score selon les pions dans chaque ligne
84     for r in range(NB_LIGNES):
85         ligne_array = [int(i) for i in list(s[r,:])]
86         for c in range(NB_COLONNES-3):
87             fenetre = ligne_array[c:c+TAILLE_FENETRE]
88             score += evaluer_fenetre(fenetre, joueur)
89
90     #on modifie le score selon les pions dans chaque colonne
91     for c in range(NB_COLONNES):
92         col_array = [int(i) for i in list(s[:,c])]
93         for r in range(NB_LIGNES-3):
94             fenetre = col_array[r:r+TAILLE_FENETRE]
95             score += evaluer_fenetre(fenetre, joueur)
96
97     # on modifie le score selon les pions dans chaque diagonale
98     for r in range(NB_LIGNES-3):
99         for c in range(NB_COLONNES-3):
100            fenetre = [s[r+i][c+i] for i in range(TAILLE_FENETRE)]
101            score += evaluer_fenetre(fenetre, joueur)
102
103    for r in range(NB_LIGNES-3):
104        for c in range(NB_COLONNES-3):
105            fenetre = [s[r+3-i][c+i] for i in range(TAILLE_FENETRE)]
106            score += evaluer_fenetre(fenetre, joueur)
107
108    return score
```

Cette fonction prend en argument la grille de jeu et le joueur dont c'est le tour. Elle va attribuer un score à la grille après un potentiel tour de ce joueur et va le retourner.
La fonction va tout d'abord calculer le nombre de pions du joueur en question dans la colonne centrale, et attribuer un score en fonction du nombre de pions. Ceci va avoir pour effet de préférer la colonne centrale aux autres.
Elle va ensuite parcourir la grille et va définir des fenêtres. Une fenêtre est une ligne de 4 cases, qu'elle soit verticale, horizontale ou diagonale. Elle va évaluer chaque fenêtre de la grille grâce à la fonction **evaluer_fenetre(fenetre, joueur)** et lui attribuer un score qu'elle ajoutera au score total.

» *Fonction **Actions(s)***

```
110
111
112 def Actions(s):
113     actions = []
114     for col in range(NB_COLONNES):
115         if s[NB_LIGNES-1][col] == VIDE:
116             actions.append(col)
117     return actions
```

La fonction **Actions(s)** prend en argument la grille. Elle renvoie la liste des colonnes où il est possible de jouer son pion en testant simplement si la case du haut est vide. Si celle-ci est vide,

il y a au moins une case de libre dans la colonne donc l'indice de la colonne est ajouté à la liste.

» *Fonction* ***Affichage(grillle)***

```
127
128 def Affichage(grille):
129     for i in reversed(range(NB_LIGNES)):
130         print(" | ",end=' ')
131         for j in range(NB_COLONNES):
132             if(grille[i][j]==1):
133                 print(CBLUE+'0'+CEND,end=' ')
134             elif grille[i][j]==2:
135                 print(CRED+'0'+CEND,end=' ')
136             else:
137                 print(" ",end=' ')
138             print(" | ",end=' ')
139         print()
140     print(" | ",end=' ')
141     for i in range(NB_COLONNES):
142         print("_",end=" ")
143         print(" | ",end=' ')
144     print()
145     print(" | ",end=' ')
146     for i in range(1,NB_COLONNES+1):
147         print(i,end=" ")
148         if i>9:
149             print("| ",end=' ')
150         else:
151             print(" | ",end=' ')
152     print()
153
```

Cette fonction permet d'afficher la grille de jeu dans la console. Pour rappel, le joueur 'humain' est représenté par un 1 dans le jeu et le joueur 'ia' par un 2. Néanmoins, des '0' colorés sont affichés dans la grille grâce aux lignes de code 133 et 135. Une case vide est représentée par " " (ligne 136/137).
Afin de pouvoir remplir la grille du haut vers le bas, i doit varier de 6 à 0 et non de 0 à 6 d'où la ligne 129.
Les print(" | ", end=' ') délimitent les colonnes dans l'affichage et le paramètre 'end' permet de remplacer le saut de ligne ('\n') par défaut de la fonction print par un espace.
Les lignes 141 à 143 permettent de faire une séparation entre la numérotation des colonnes et la grille de jeu.
La numérotation des colonnes est notamment possible grâce aux lignes 146 à 151.

» *Fonction* ***prochaine_ligne_vide(s, col),***
» *Fonction* ***grille_apres_jeu(s, row, col, joueur),***
» *Fonction* ***cases_libres(s),***

```
154
155 def prochaine_ligne_vide(s, col):
156     for r in range(NB_LIGNES):
157         if s[r][col] == VIDE:
158             return r
159
160 def grille_apres_jeu(s, row, col, joueur):
161     s[row][col] = joueur
162
163 def cases_libres(s):
164     result = 0
165     for i in range(NB_LIGNES):
166         for j in range(NB_COLONNES):
167             if s[i][j]==0:
168                 result+=1
169     return(result)
170
```

La fonction **prochaine_ligne_vide(s, col)** prend en argument la grille ainsi qu'une colonne. Elle va retourner l'indice de la première ligne vide, donc de l'emplacement où le pion va tomber quand un joueur va jouer dans la colonne en question.

La fonction **grille_apres_jeu(s, row, col, joueur)** prend en argument la grille, la ligne, la colonne et le joueur dont c'est le tour. Elle va positionner le pion dans l'emplacement joué et donc modifier la grille.

La fonction **cases_libres(s)** prend en argument la grille et va retourner le nombre de cases vides dans la grille.

» *Fonction **AlphaBeta(s, depth, alpha, beta, joueurAMaximiser)***

```python
#%%Minimax
def AlphaBeta(s,depth,alpha,beta,joueurAMaximiser):
    actions = Actions(s)
    est_terminal = est_noeud_terminal(s)
    if depth == 0 or est_terminal:
        if est_terminal:
            if est_tour_gagnant(s, ia):
                return (None, 100000000000000)
            elif est_tour_gagnant(s, humain):
                return (None, -100000000000000)
            else: # GameOver
                return (None, 0)
        else: # à la prof 0
            return (None, score_position(s, ia))
    if joueurAMaximiser:
        value = -math.inf
        colonne = random.choice(actions)
        for col in actions:
            ligne = prochaine_ligne_vide(s, col)
            b_copy = s.copy()
            grille_apres_jeu(b_copy, ligne, col, ia)
            nouveau_score = AlphaBeta(b_copy, depth-1, alpha, beta, False)[1]
            if nouveau_score > value:
                value = nouveau_score
                colonne = col
            alpha = max(alpha, value)
            if alpha >= beta:
                break
        return colonne, value

    else: # tour du joueur à minimiser
        value = math.inf
        colonne = random.choice(actions)
        for col in actions:
            ligne = prochaine_ligne_vide(s, col)
            b_copy = s.copy()
            grille_apres_jeu(b_copy, ligne, col, humain)
            nouveau_score = AlphaBeta(b_copy, depth-1, alpha, beta, True)[1]
            if nouveau_score < value:
                value = nouveau_score
                colonne = col
            beta = min(beta, value)
            if alpha >= beta:
                break
```

La fonction **AlphaBeta(s, depth, alpha, beta, joueurAMaximiser)** prend en argument la grille, la profondeur, un alpha, un beta et un booléen (joueurAMaximiser) qui vaut True si c'est le joueur à maximiser et False sinon. Elle va retourner la meilleure colonne à jouer.
Elle va tester pour chaque profondeur, et pour chaque colonne, le score de la grille. Et va prendre le score maximum pour les profondeurs où le joueur est à maximiser et le score minimum pour les profondeurs où le joueur est à minimiser. Nous avons ajouté à cet algorithme un élagage alpha-beta pour réduire le nombre de nœuds évalués.
L'utilisation de random aux lignes 177 et 193 permet à Alpha-Beta d'initialiser la variable colonne à une valeur aléatoire parmi la liste retournée par Actions(). Cela permet d'être sûr de pouvoir retourner une colonne dans le cas où l'algorithme ne passe pas dans la boucle if.

» *Fonction **Jeu()***

```
218
219 #%%Jeu
220 def Jeu():
221     grille = np.zeros((NB_LIGNES,NB_COLONNES),dtype=int)
222     Affichage(grille)
223     game_over = False
224     tour = 1
225
226     while not game_over and tour <= 42 :
227         if tour % 2 != 0:#Tour de l'IA
228             #start_time=time.time()
229             col = AlphaBeta(grille, 4, -math.inf, math.inf, True)[0]
230             #print("temps d'exécution : "+str(time.time()-start_time))
231             print("\nL'IA a joué en "+ str(col+1) + "\n" )
232             if grille[NB_LIGNES-1][col] == VIDE:
233                 ligne = prochaine_ligne_vide(grille,col)
234                 grille_apres_jeu(grille,ligne,col,ia)
235                 Affichage(grille)
236                 tour += 1
237             if est_tour_gagnant(grille, ia):
238                 game_over=True
239                 print("Dommage, le robot est meilleur et le sera toujours !")
240         else :
241             print('liste des actions possibles :')
242             print([i+1 for i in Actions(grille)])
243             print('Que voulez vous jouer ?')
244             while True:
245                 col=int(input('numéro de la colonne =')) - 1
246                 if col in Actions(grille):
247                     break
248             #start_time=time.time()
249             #col = AlphaBeta(grille, 4, -math.inf, math.inf, True)[0]
250             #print("temps d'exécution : "+str(time.time()-start_time))
251             ligne = prochaine_ligne_vide(grille,col)
252             grille_apres_jeu(grille,ligne,col,humain)
253             Affichage(grille)
254             tour += 1
255             if est_tour_gagnant(grille, humain):
256                 game_over = True
257                 print("BRAVOOOO TU AS GAGNE L'HUMAIN !!")
258                 break;
259     if tour == 43:
260         print("Bien joué, tu as quand même réussi à égaliser avec la machine !")
261
262 Jeu()
```

La fonction **Jeu()** est la fonction qui permet de changer de joueur une fois la partie lancée.

Au départ, la grille est initialisée à une matrice de 0 de taille 6x12, le booléen game_over à False afin de déterminer s'il y a un gagnant ou non.
A la ligne 226, on vérifie que les deux conditions essentielles au déroulement du jeu.
La condition à la ligne 227 détermine si l'ia joue car on considère que c'est elle qui commence au tour 1. Le choix de l'ia est déterminé par la fonction **AlphaBeta** expliquée plus haut. Les lignes de code 232 à 236 permettent de placer le pion à la colonne col choisie.
Si c'est le tour de l'humain, on lui affiche toutes les possibilités possibles pour qu'il puisse faire son choix et on récupère sa valeur en vérifiant qu'il s'agisse bien d'une action possible (244 à 247). De même que pour l'ia, le pion est placé selon le choix col du joueur humain.
La partie est arbitrairement terminée lorsque 43 pions ont été placés.

- <u>Explication des heuristiques</u>

```
55 def evaluer_fenetre(fenetre, joueur):
56     score = 0
57     adversaire = humain
58     if joueur == humain:
59         adversaire = ia
60 
61     if fenetre.count(joueur) == 4:
62         score += 100
63     if fenetre.count(joueur) == 3 and fenetre.count(VIDE) == 1:
64         score += 5
65     if fenetre.count(joueur) == 2 and fenetre.count(VIDE) == 2:
66         score += 2
67     if fenetre.count(adversaire) == 4:
68         score -= 15
69     if fenetre.count(adversaire) == 3 and fenetre.count(VIDE) == 1:
70         score -= 5
71     if fenetre.count(joueur) == 3 and fenetre.count(adversaire) == 1:
72         score -= 10
73     return score
```

Cette fonction donne le score du joueur qui sera minimisé (humain) ou maximisé (IA) par AlphaBeta.
La fonction ci-dessus **evaluer_fenetre(fenetre, joueur)** retourne le score de la fenêtre *fenetre* rentrée en paramètre. Durant l'exécution de cette fonction, le score varie selon le choix du joueur pour placer son pion.
Elle prend en argument une zone de la grille : fenetre et un joueur (IA ou humain).
Nous avons choisi de mettre des if et non des elif afin que notre IA vérifie un maximum les conditions ci-dessus avant de placer son pion. Cela permettra à la machine d'avoir le meilleur positionnement.
Lignes 61 à 66 : ces lignes de code servent à expliquer à la machine quels placements de pions la mènent vers la victoire, et par conséquent, la poussent à favoriser les placements qui remplissent une fenêtre (+100 points), qui alignent 3 pions avec une case vide (+5 points), et 2 pions avec 2 cases vides (+2 points).
Lignes 67 à 72 : ici, on explique à la machine quelles situations la feraient perdre la partie afin qu'elle puisse bloquer les actions de son adversaire. Si l'adversaire a réussi à aligner 4 pions elle perd 15 points, si l'adversaire a 3 pions et une case vide elle perd 5 points et enfin si l'adversaire place un pion qui empêche la machine de gagner, elle perd 10 points. Cette dernière condition est la plus importante car elle permettra à l'algorithme Alpha-Beta de choisir le chemin le plus optimal.