AI - BATTLE

Affrontement entre IA pour le jeu de Morpion 4

Nom de l'équipe : Les Fast Tokens

Noms et prénoms des membres de l'équipe :

PETITCOL Alix, RIPAUD Elodie, RAIMBAULT Constance, RAFALIMANANA Glory

Table des matières

L'objectif de ce projet est de créer une IA capable de jouer au Morpion 4 sur un plateau comptant 12 colonnes et 12 lignes. Le gagnant du jeu est celui qui réalise un alignement de 4 pions en premier. Nous visons également une vitesse de calcul inférieure à 10 secondes lorsque c'est au tour de l'IA de jouer tout en garantissant la qualité de son jeu.

Adaptation du Morpion 3x3 pour le Morpion 12x12 :

Dans un premier temps, nous nous basons sur la solution du td 4 sur le Morpion 3x3 :

Après avoir modifié la taille du plateau du Morpion 3x3 en plateau de 12, nous adaptons la fonction Action, elle a pour objectif de donner les coups possibles en regardant les cases vides du plateau à un état de jeu donné.

Nous reprenons également la fonction Evaluate qui annonce quel joueur gagne. Si l'adversaire gagne alors sa valeur d'identification -1 est retournée, si notre IA gagne alors la fonction retourne sa valeur type qui est 1. Dans le cas d'un match nul, la fonction renvoie 0.

```
def Evaluate(state):
    """
    This function evaluates the state
    :param state: the state of the current plate
    :return: +1 if Computer, -1 if Human, 0 if draw
    """
    if Win(state, HUMAN):
        score = -1
    elif Win(state, COMP):
        score = +1
    else:
        score = 0
    return score
```

La fonction GameOver renvoie un booléen, elle nous aide à déterminer la fin du jeu en utilisant la fonction Win expliquée ci-dessous. Si l'un des deux joueurs gagne alors la fin du jeu est annoncée.

Fonction « Win » :

Cette fonction nous permet de savoir, pour un joueur donné, s'il a remporté la partie à un état de jeu du plateau state. Elle renvoie un booléen 'true' si le joueur donné en paramètre a gagné, sinon elle renvoie 'false'.

Pour cela nous allons vérifier s'il y a des alignements de 4 jetons du joueur donné sur le plateau. On commence par vérifier un potentiel alignement sur les colonnes. Nous parcourons la liste de l'index 0 à 9 exclu pour ne pas sortir de la grille car nous allons jusqu'à x+3 lignes et étant donné notre plateau 12x12.

Si aucun alignement sur les colonnes n'est détecté (win='false'), on fait de même pour vérifier l'alignement sur les lignes.

Puis si aucun alignement n'est détecté ni sur les colonnes ni sur les lignes on effectue notre vérification sur les diagonales, tout d'abord les diagonales montantes puis les diagonales descendantes.

Des breaks étant mis dès que le booléen win est égal à 'true', ceci nous permet de ne pas vérifier toutes les dispositions possibles et donc de gagner du temps.

Le problème rencontré avec cette fonction a été décelé en utilisant la fonction GameOver car on a remarqué que lorsqu'un joueur gagnait, le programme ne le décelait pas et donc ne l'affichait pas.

La fonction GameOver utilisant la fonction Win on a rapidement vu que le problème venait de cette dernière.

Le problème que nous avons rencontré est le suivant :

```
for x in range(9):
    for y in range(12):
        if [player, player, player, player] in [state[x][y], state[x + 1][y], state[x + 2][y], state[x + 3][y]]:
            win = True
            break
    if win:
        break
```

En effet contrairement à la version présentée précédemment nous avions un problème de syntaxe. Nous avions mis un « in » à la place d'un « == » dans notre condition principale, ce qui ne réalisait pas ce que nous voulions.

Fonction « main » :

C'est la fonction qui nous permet de lancer la partie. On commence par initialiser le jeu et par déterminer quel joueur joue en premier. Ainsi on commence différemment si l'IA joue en premier.

Tant que le jeu n'est pas terminé (Gameover renvoie False) on copie le plateau pour ne pas qu'il soit modifié avec les tests du AlphaBeta et on compte les tours, si le tour est pair l'humain joue sinon c'est à l'IA de jouer.

Si l'IA joue en premier elle joue par défaut dans la case (6,6) sur le plateau sinon on appelle directement la fonction AlphaBeta . Pour les autres tours l'IA jouera les coups retournés par AlphaBeta . A chaque fin de tour la fonction Result affectera à la nouvelle case choisie par l'IA +1 qui donnera un 'O' à l'affichage (Show). On relève aussi le temps de jeu de l'IA dans la variable « z ».

Pour l'humain on affecte directement « HUMAN » à la case choisie soit -1 pour l'affichage de 'X'.

Puis à la fin de la partie on affiche le nom du gagnant grâce à la fonction Win décrite précédemment.

```
    else:
        if(tour == 1 and IA_Start == True): # l'IA démarre au milieu du plateau si c'est lui qui commence la partie
            action = (5,5) # à la colonne 6 et ligne 6
        else:
            print("Tour de l'IA")
            start = time.perf_counter()
            action = AlphaBeta(s, tour, IA_Start)
            end = time.perf_counter()
            z = float(end-start)
            print(f"Temps de réponse de l'IA : {z}")
        print(f"L'IA a choisi la COLONNE {action[1]+1} et la LIGNE {action[0]+1}")
        plate = Result(plate, action, COMP)
    tour += 1
    Show(plate)
if Win(plate, HUMAN):
    print('HUMAN Win!')
elif Win(plate, COMP):
    print('AI Win!')
else:
    print('Draw')
```

Explication de l'élagage :

Cependant une difficulté se pose pour le morpion 12x12 car le nombre de coups possibles est plus important. Nous avons décidé de faire un élagage en profondeur en utilisant l'algorithme Alpha-Beta et un élagage en largeur avec la fonction WidthPruning afin de diminuer notre espace de recherche. Cette fonction permet d'appliquer à l'élagage Alpha-Beta les coups les plus avantageux selon les stratégies que nous avons mises en place.

Pour l'élagage en profondeur, nous choisissons une profondeur k=4 car on a besoin que de 4 coups pour gagner. Nous explicitons ce choix dans les fonctions MinValue et MaxValue.

```
    k += 1
    if k == 4 or GameOver(state):
        return Evaluate(state)
```


PETITCOL Alix, RIPAUD Elodie, RAIMBAULT Constance, RAFALIMANANA Glory

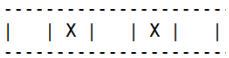
Notre élagage en largeur repose sur l'analyse de l'état du jeu. L'algorithme recherche et classifie, pour chaque plateau de jeu en cours, les stratégies gagnantes offensives et défensives de l'IA. Il les classe en 6 premières catégories : ultimate, classe A, classe B, classe C, classe D et classe E.

Pour le set **"ultimate"** (stratégie offensive) : on vérifie si, sur le plateau de jeu actuel, il y a 3 pions alignés de type ordinateur (IA player = +1). Si c'est le cas et que l'on détecte une case vide qui va permettre d'aligner 4 pions et donc de gagner, on ajoute les coordonnées de cette dernière au set "ultimate". Il s'agit d'une stratégie gagnante avec probabilité de 1 donc prioritaire sur toutes les autres actions possibles.

La **classe A** correspond à une stratégie défensive. On regarde si l'adversaire a déjà aligné 3 pions avec au plus une case vide entre deux de ces pions. On ajoute alors les coordonnées de la case (ou les cases) permettant de contrer l'adversaire dans le set "A_best_cell" qui sera renvoyé à notre Alpha-Beta dans le cas où le set de priorité supérieure (dans notre cas : "ultimate") est vide.

La **classe B** regroupe les coordonnées des cases vides voisines et dans l'alignement de 2 pions alignés de type ordinateur (IA player = +1). On ajoute ces coordonnées au set "B_best_cell". C'est une stratégie offensive qui permet d'aboutir à la formation d'un alignement de 3 pions de type IA.

On retrouve aussi dans cette classe le cas défensif / offensif suivant :



Si dans cet exemple on considère que les croix sont les pions de l'adversaire, l'IA doit impérativement placer son pion sur la case du milieu afin de contrer son adversaire sinon ce dernier gagnera forcément. A l'inverse, si les croix sont les pions de l'IA, il lui faut mettre son pion sur la case du milieu pour gagner. On ajoutera les coordonnées de cette case au set "B_prioritaire".

La **classe C** repose sur la même stratégie que la classe B mais défensive : les 2 pions alignés détectés sont ceux de l'adversaire. On cherche alors à éviter un potentiel alignement de 3 pions adverses. On ajoute les coordonnées des cases qui nous intéressent au set "C_best_cell". Cette classe est retournée à Alpha-beta quand les classes de niveaux supérieures sont vides.

Enfin les **classes D et E** sont les classes utiles en début de partie. Aucun alignement n'est détecté. Pour chaque case occupée par un pion de type ordinateur (classe D), on ajoute les cases vides voisines au set "D_best_cell". De la même façon, pour un pion adverse, on ajoute les cases vides voisines au set "E_best_cell".

A toutes ces classes nous en ajoutons 3 autres qui complètent nos stratégies de jeu :

La classe **D_prioritaire**, **C_prioritaire** et **B_prioritaire** : l'idée est d'adopter en 1 coup une stratégie offensive et défensive.

La classe D_prioritaire contient l'intersection des cases présélectionnées dans les classes D et E. Elle est renvoyée à Alpha-beta en priorité par rapport à la classe D.

Les classes C_prioritaire (intersection entre D_prioritaire et C) et B_prioritaire (intersection de C_prioritaire et B) fonctionnent de la même manière. Finalement le classement par ordre de renvoi à la fonction Alpha Beta est le suivant:

- Ultimate
- Classe A
- Classe B_prioritaire
- Classe B
- Classe C_prioitaire
- Classe C
- Classe D_prioritaire
- Classe D
- Classe E

Ainsi nous limitons drastiquement notre espace de recherche en ne renvoyant à la fonction AlphaBeta uniquement les actions jugées utiles et/ou nécessaires. Notre temps de calcul est alors optimisé.

Conclusion :

Finalement grâce aux différentes fonctions mises en œuvre nous avons un Morpion 12*12 fonctionnel et qui tourne en dessous de la limite des 10 secondes. En effet notre IA met environ 2*10^-3 secondes pour choisir et placer son coup.