Aitalla Mehdi

Ait Lahcen Souhail

Ben Mansour Célina

Agert Petru-Luigi

Rapport : Affrontement Intelligence Artificielle

Le code est disponible sous format google colab :

https://colab.research.google.com/drive/1z72zeVdMoGEcqYdWNpGl2R63Srb115wX?usp=sharing&fbclid=IwAR2FabH-ghQxWyAXBwMQs4SMXi-M9f-3xuCtWmP3o-kGPYxXn7nGGbxT-ek

I. Mise en place du jeu

Création d'une matrice numpy 12x12 qui peut être parcouru en ligne et en colonne.

Affichage d'une grille dont les indexes vont de 1 à 12 en colonne et en ligne

Fonction joueur() permettant de savoir quel est le joueur qui doit jouer.

II. Heuristique

On suppose ici que l'IA joue avec les 'X'.

On évalue chaque position possible, et pour chacune d'elle on regarde dans un carré de 4x4 autour le nombre de 'X' ou 'O' et on attribue un score à cette case en fonction de ce nombre et de la situation.

- Si l'on a 3 'X' alignés on attribue un score très élevé (10^21).

```
if([jeu[i[x],b] for b in range(0,4) if abs(b-m[z])<=3].count('X') == 3):
    val+=100000000000000000000
```

- Même opération en colonne

- Si l'on a 3 'X' sur une même ligne espacés ou non de 3 cases.

- On compte le nombre de 'X' sur la ligne sur une plage de 3 cases en arrière et 3 en avant et le score de la case prend la valeur+=2**(nombre de 'X')

- Même opération en colonne

- On étudie la diagonale de la même manière mais la condition pour évaluer le nombre de ‘X’ n’est pas correcte et ne permet pas son utilisation

- On modifie également le score à la position en fonction de la présence de pions adverses

```
val+=val**[jeu[i[x],b] for b in range(i[x],12) if abs(b-m[z])<=3].count('X')
val+=val**[jeu[b,j[x]] for b in range(j[x],12) if abs(b-m[z])<=3].count('X')
```

La position a alors un score attribué en fonction de tous ces paramètres et classe les coups possibles en fonction du meilleur résultat.

III. Kernels

Pour la réalisation des fonctions gagner et perdre (NouveauGagner(transformation(jeu)) et NouveauGagner(transformation2(jeu))), nous pourrons déterminer qui a gagné ou perd et ce grâce à l'utilisation de kernels : nous pourrons réaliser cela par calcul matriciel. Avant de réaliser ce calcul matriciel, nous devons transformer notre matrice de STR en INT pour cela nous utilisons les deux fonctions transformations (l'une considère que c'est gagné si on a un enchaine successif de 4 'X' et l'autre de 4 'O'). Ensuite, nous réalisons une convolution qui va nous permettre de savoir si, que ce soit en diagonale, en ligne ou en colonne, il y'a un vainqueur.

```
def convo(input, kernels):
    expanded_input = as_strided(
        input,
        shape=(
            input.shape[0] - kernels[0].shape[0] + 1,
            input.shape[1] - kernels[0].shape[1] + 1,
            kernels[0].shape[0],
            kernels[0].shape[1],
        ),
        strides=(
            input.strides[0],
            input.strides[1],
            input.strides[0],
            input.strides[1],
        ),
        writeable=False,
    )
    res = False
    for kern in kernels:
        feature_map = np.einsum(
            'xyij,ij->xy',
            expanded_input,
            kern,
        )
        #res = res or (np.any(feature_map==4)
        res = res or np.any(feature_map==4)
    return res
```

IV. Autres Fonctions

Fonction jouer() permet de demander au joueur où placer son pion. Elle implémente la fonction coupPossible(Grille). Cette autre fonction vérifie que la position est valide (‘.’)

coupPossible(Grille) retourne une liste.

Transformation(jeu) : permet d’attribuer des valeurs numériques à une grille remplie de ‘X’,’O’, ‘.’

Minimax(grille,profondeur, alpha, beta, tour) :

La fonction permet de trouver la valeur maximale à jouer en parcourant les branches de l’arbre des possibilités. La profondeur du parcours doit être renseignée. Nous avons choisi une profondeur 3, le fonctionnement étant plus rapide et plus efficace en jeu. Nous avons incorporé un élagage alpha/beta permettant d’éliminer des solutions de l’arbre qui ne seraient pas judicieuses.

V. Déroulement de la partie

Création de l’algorithme jouerPartie() :

- Initialisation d’une grille vide
- Déterminer le premier joueur
- Début du chronomètre
- Algorithme du minimax : on commence par donner des pénalités très grandes (infini ou -infini)
- Fin du chronomètre
- Donne la position que le minimax a retourné