Compte rendu TP2 AI : Nim

Sommaire

[1) Algorithme 2](#_Toc60779729)

[1-2) Trace d'exécution 5](#_Toc60779730)

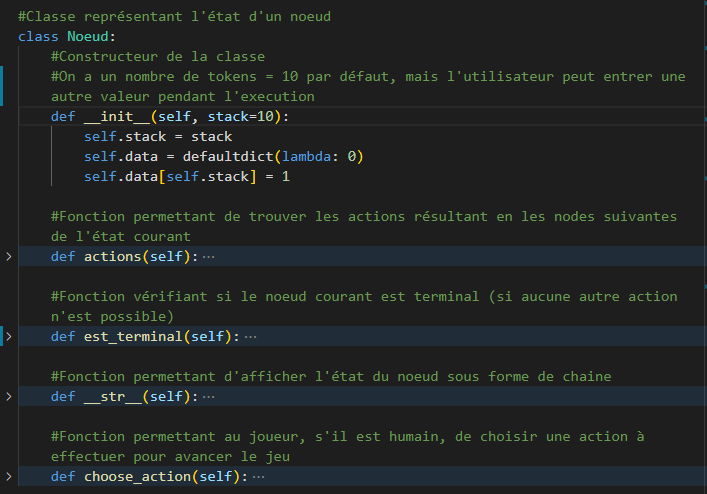
[2) Elagage 6](#_Toc60779731)

[3) Statistiques: 8](#_Toc60779732)

[4) Conclusion: 9](#_Toc60779733)

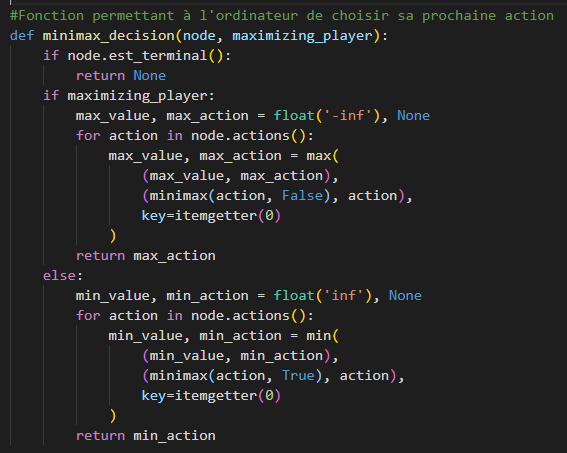
# 1) Algorithme

Nous commençons par la définition d'une classe représentant l'état d'un nœud et contenant différentes fonctions permettant la manipulation des données. La classe est comme suit:

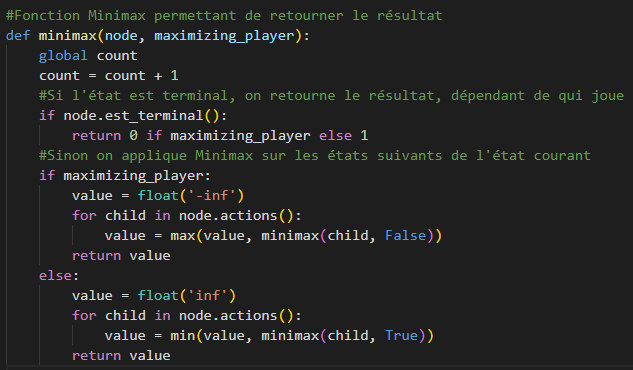


Joint à ce document est un fichier .py qui vous permettra de voir les fonctions avec plus de détail.

Ensuite, nous créons la fonction minimax\_decision qui s'occupera de faire les décisions de la machine:



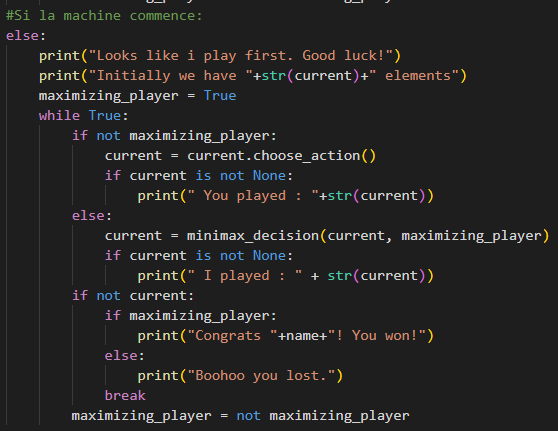
Cette fonction appelle notre fonction minimax:



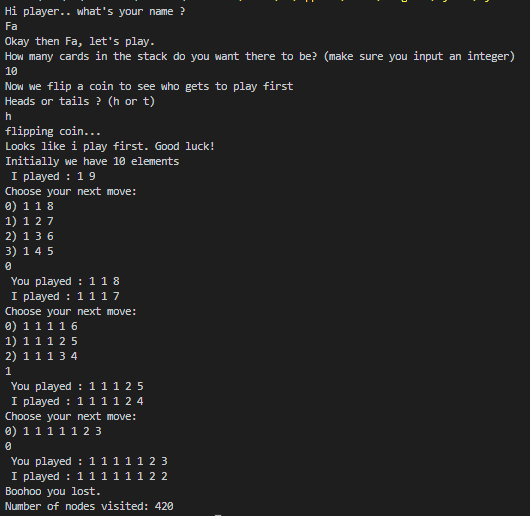
La variable "count" permet d'enregistrer le nombre de nœuds visités.

Puis nous lançons le jeu, nous décidons de qui commence à jouer grâce à un lancer de pile ou face.



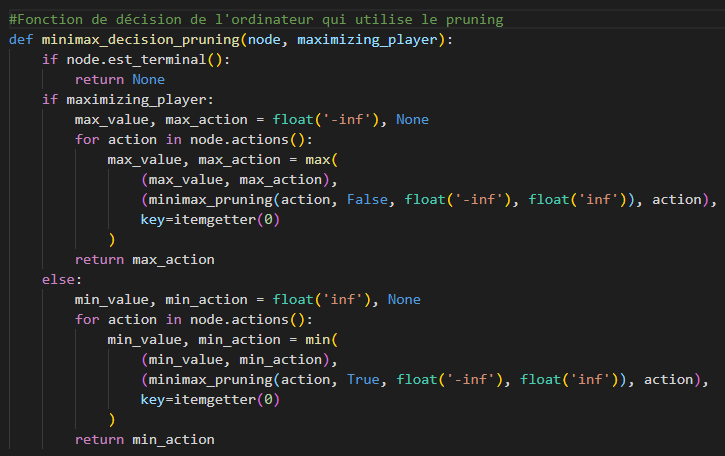


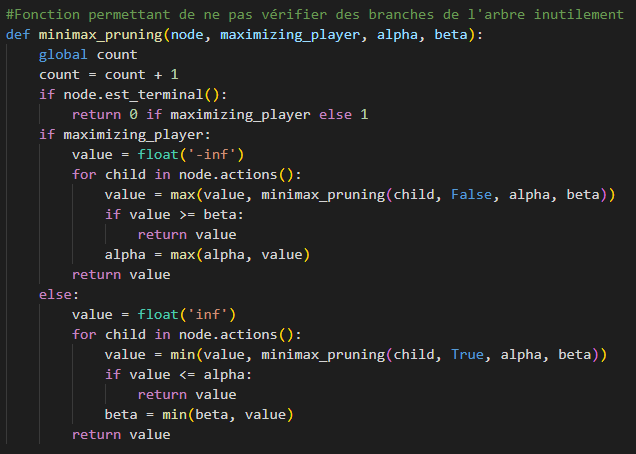
# 1-2) Trace d'exécution



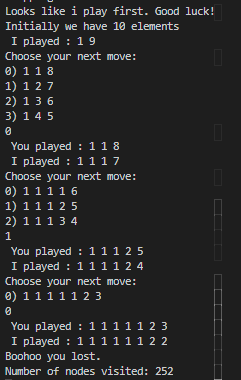
# 2) Elagage

Nous créons les fonctions minmax\_decision\_pruning et minmax\_pruning suivantes:





3) Nous répétons le même exemple mais en utilisant les fonctions de pruning:



On remarque que le nombre de nœuds visités et légèrement plus que la moitié! C'est déjà très bien.

# 3) Statistiques:

|  |  |  |
| --- | --- | --- |
| Nombre Tokens | Nœuds visités sans elagage | Nœuds visités avec elagage |
| 5 |  |  |
| 10 |  |  |
| 13 |  |  |
| 15 |  |  |
| 18 |  |  |
| 20 | Execution trop longue. |  |

# 4) Conclusion:

On remarque que le nombre de nœuds visités augmente exponentiellement par rapport au nombre n de tokens dans le tas. Le temps d'execution est aussi beaucoup plus long, et encore plus quand il n'y a pas d'elagage.

L'elagage permet de gagner du temps à ne pas visiter des nœuds non nécessaires.

Plus le nombre de tokens augmente, plus la nécessité de l'elagage est évidente dans le temps d'execution.

Merci pour votre lecture.