Ce mémo regroupe mes notes de lecture d’articles de Wikipédia qui m’aident à préparer l’implémentation Python pour les IA de types Minimax pour Jersi et Mikjersi.

Table des matières

[1. Minimax 1](#_Toc93339689)

[2. Negamax 2](#_Toc93339690)

# Minimax

Le jeu est représenté par un arbre composé de nœuds. Un nœud est noté .

Utilisons le prédicat pour savoir si un nœud est terminal : gagné, perdu ou nul.

Le joueur démarre son analyse sur le nœud racine .

Si le nœud racine n’est pas terminal alors le joueur doit choisir un nœud enfant de . Ce nœud choisi , s’il n’est pas terminal, sera ensuite joué par son opposant .

Le joueur suppose que son opposant évalue les nœuds comme lui jusqu’à une profondeur maximale et en utilisant une fonction d’évaluation pour tout nœud terminal ou ayant atteint la profondeur maximale d’analyse.

La fonction exprime toujours la favorabilité pour le joueur . Par convention, si alors :

* si et seulement le nœud est gagnant pour le joueur si .
* si et seulement le nœud est nul pour le joueur si .
* si et seulement le nœud est perdant pour le joueur si .
* pour tout nœud qui n’est ni gagné, ni nul, ni perdu pour le joueur .

La fonction est définie récursivement comme suit :

* Si ou si alors
* Sinon
  + Si alors
  + Si alors

La fonction est appliquée au nœud racine et renvoie une valeur :

Le joueur doit choisir un des nœuds dont la valeur est .

Pour quelques cas de profondeur, explicitons l’analyse d’une racine non-terminale :

* :
* :
* :

# Negamax

On adapte minimax en exploitant la propriété .

La fonction est définie récursivement comme suit :

* Si ou si alors
* Sinon

Comme pour minimax, la fonction exprime toujours la favorabilité pour le joueur .

La fonction est appliquée au nœud racine et renvoie une valeur :

Le joueur doit choisir un des nœuds dont la valeur est .

Pour quelques cas de profondeur, explicitons l’analyse d’une racine non-terminale :

* :
* :
* :