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 :

* :
* :
* :

# Alpha-beta

Alpha-beta est une optimisation de « minimax » qui élague une partie de l’arbre des nœuds du jeu qui ne contribue pas à la valeur finale de la racine.

La fonction est remplacée par la fonction qui tient à jour deux paramètres supplémentaires :

* est le score minimum dont est assuré le joueur qui maximise.
* est le score maximum dont est assuré le joueur qui minimise.

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

Le joueur doit choisir un des nœuds dont la valeur est . On verra qu’il y a une subtilité sur les ex-aequo.

On présente ici la version dite « fail-hard », par opposition à la version dite « fail-soft ».

La fonction est définie récursivement comme par l’algorithme suivant ; les changements par rapport à sont surlignés en jaune :

|  |
| --- |
| 1. then    * + 1. (\* cutoff \*) 2. then    * + 1. (\* cutoff \*) |

L'algorithme maintient deux valeurs, alpha et bêta, qui représentent respectivement :

* : le score minimum dont est assuré le joueur qui maximise ;
* : le score maximum dont est assuré le joueur qui minimise.

Initialement, alpha est l'infini négatif et bêta est l'infini positif, c'est-à-dire que les deux joueurs commencent avec leur pire score possible.

Chaque fois que le score maximum dont le joueur qui minimise (c'est-à-dire le joueur "bêta") est assuré devient inférieur au score minimum dont le joueur qui maximise (c'est-à-dire le joueur "alpha") est assuré (c'est-à-dire bêta < alpha), le joueur n'a pas besoin de considérer d'autres descendants de ce nœud, car ils ne seront jamais atteints dans le jeu réel.