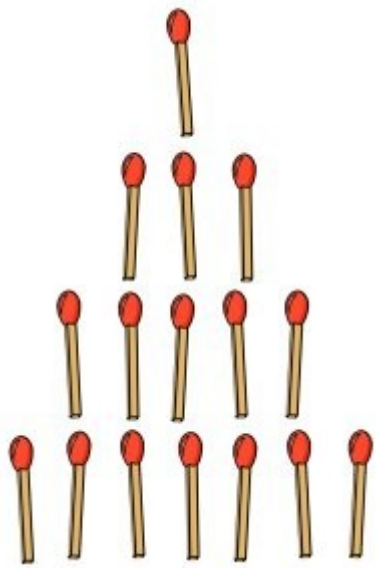
18 Septembre 2024



Rapport jeu de Nim

Killian Schied et Dylan Landrin

Table des matières

Description du code

Notre code à été créer en deux fichiers distincts. Le premier à permis de faire un premier jet du minimax, puis le second à permis de l'optimiser afin de faire le nombre minimal d'appel à exploreMin et exploreMax.

game.py

Ce fichier contient la première version de notre travail.il permet de savoir si le joueur 1 à un coup gagnant pour un nombre d'allumette de départ donné.

MiniMax :

Cette methode permet de lancer l'algorythme miniMax, elle renvoie pour un nombre d'allummette données :

- -1 si le joueur 1 n'a pas de coup gagnant
- 0 en cas d'égalité (impossible dans le jeu de Nim)
- 1 si le joueur 1 à un coup gagnant

Si c'est le tour du joueur 1, elle appelle exploreMax, et si c'est au tour du joueur 2, exploreMin.

ExploreMax :

Cette méthode permet de vérifier si le joueur 1 a la possibilité de gagner la partie.

La méthode va essayer les trois coups possibles, et pour chaque coup, il va vérifier s'il est perdant en faisant appele à exploreMin.

Il va retourner :

- -1 si le joueur 1 n'a pas de coup gagnant
- 0 en cas d'égalité (impossible dans le jeu de Nim)
- 1 si le joueur 1 à un coup gagnant

ExploreMin :

Cette méthode permet de vérifier si le joueur 1 a la possibilité de perdre la partie.

La méthode va essayer les trois coups possibles, et pour chaque coup, il va vérifier si avec ce coup le joueur serait gagnant en faisant appele à exploreMax.

Il va retourner :

- -1 si le joueur 1 n'a pas de coup gagnant
- 0 en cas d'égalité (impossible dans le jeu de Nim)
- 1 si le joueur 1 à un coup gagnant

__Str__

Nous avons également redéfinis str afin de print les statistiques de la parties lancé.

Le résumé affiche :

- Le nombre d'allumette de départ
- Le numéro du joueur dont c'est le tour
- Le nombre d'appel à exploreMax
- Le nombre d'appel à exploreMin
- La victoire revient elle au joueur 1

game_optimized.py

ExploreMax :

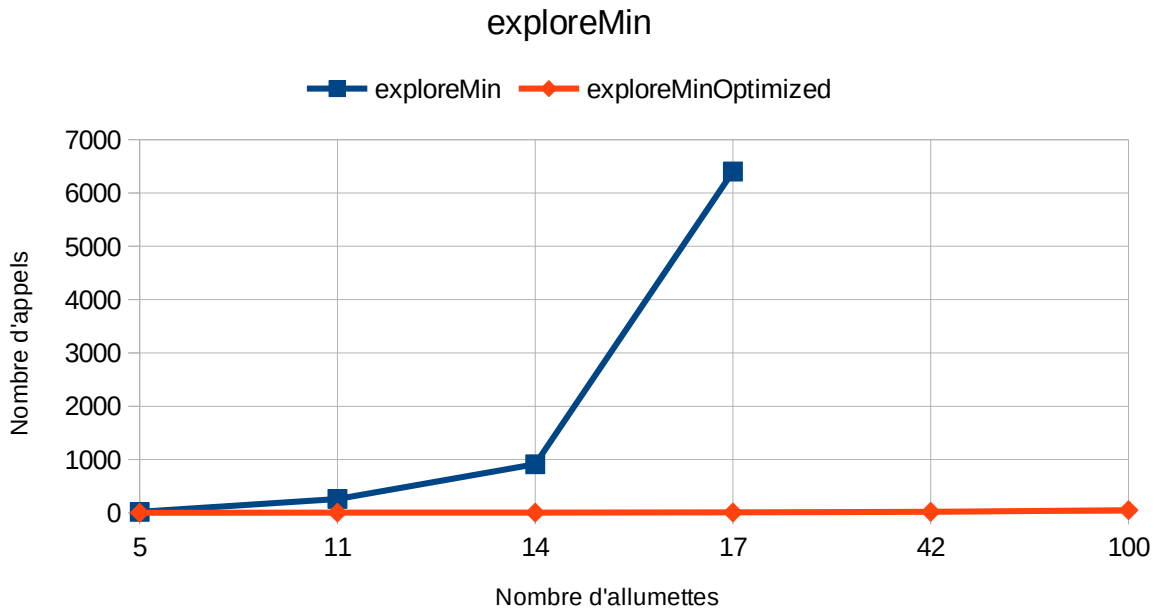
Afin de diminuer le nombre d'appel et de réduire le temps d'execution, nous avons ajouté un bloceur a la méthode, afin qu'elle s'arrete elle même si elle trouve une solution gagnante. Par conséquent, si la première allumette retiré donne un coup gagnant, on ne testera pas la deuxième ni la troisieme.

ExploreMin :

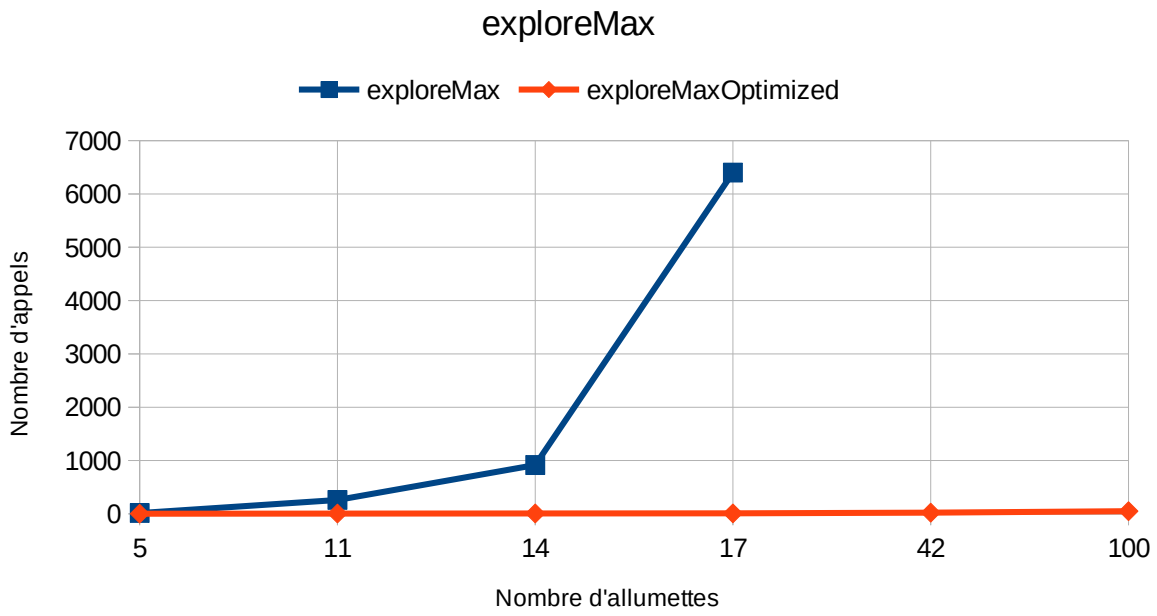
Les amélioration de exploreMin sont les même que exploreMax, a la différence que le bloceur vérifie si un coup est forcément perdant pour arreter les appels.

Comparaison des performances :

ExploreMin



ExploreMax



Temps

Lors de l’execution des deux programmes, on voit clairement que la version non optimisé fait un nombre d’appel aux méthodes bien trop importante. Sur des petites valeur comme 5 ou 11 alumettes on ne voit aucune différence avec la version optimisé. Les 2 on un temps de 0s d’execution.

Cependant, lorsqu’on augmente le nombre d’alumette à 17, on a observer qu’il fallait 4 secondes au programmes pour s’executer, contre 0s pour l’optimisé.

Lorsqu’on essai avec 42 ou 100, le non optimisé ne se termine pas, nous avons arreter manuellement le programme au bout de 15 minutes. Le programme optimisé est quand a lui resté instantanné.

L’optimisation d’une application est donc un point à ne pas négliger lors de la création d’un produit. Un programme qui donne le résultat attendu au bout d’énormement de temps est presque aussi utile qu’un programme qui ne fonctionne pas. Réfléchir au nombre cyclomatique, ou au nombre d’appel d’une fonction est donc indispensable.