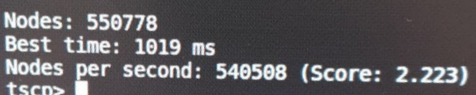
1. Performances de départ
2. Modifications apportées
3. Problèmes rencontrés
4. Performances finales
5. Comparaison des performances
6. Annexes des commits
7. **Performances de départ**



1. **Modifications apportées**

Nous avons apporté l'optimisation 2 en suivant la guideline.

Pour cela nous avons d'abord ajouté la structure du tableau can\_attack[6][64][64] (**commit 6cebf38**) dont l’utilisation définira les déplacements possibles d’une pièce et qui permettra ensuite d’éliminer les boucles inutiles dans la méthode attack().

Une possibilité d'optimisation encore meilleure par l'utilisation de bitboards et de masques était indiquée mais nous ne l'avons pas effectué, préférant d'abord finaliser l’optimisation 2 avant de l'améliorer.

Nous avons modifié la méthode attack() (**commit 7fe7ff1**), en y intégrant les tableaux can\_attack. Cela permet de parcourir, hors case où il y a les pions, les cases où des déplacements de autres pièces sont possibles.

Nous avons ajouté la méthode initAttackTables() (**commit b48b797**). On a pu vérifier grâce à la méthode memset() à l’intérieur de celle-ci qu’il n’y avait pas de bug jusqu’ici. On modifie ensuite initAttackTables() en retirant memset() et pour prégénérer les tableaux can\_attack pour chaque pièce différentes des pions.

1. **Problèmes rencontrés**

Après un temps de compréhension conséquent du programme TSCP -assez complexe aux premiers abords pour des non-initiés- , nous nous sommes penchés sur l'optimisation 1.

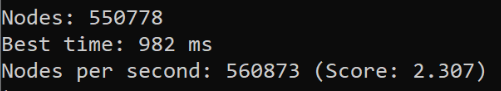
Nous avons amorcé la première étape en ajoutant les nouvelles structures posPieceB et posPieceW dans board.cpp (qui aurait été remis plus tard dans data.h/data.cpp) (**commit 415c569**) mais après trop de confusion et de perte de temps sur la suite des étapes nous avons choisis de poursuivre uniquement l'optimisation 2 qui nous paraissait plus simple avec moins de code pré-existant à changer.

Au cours de notre avancement, après avoir codé initAttackTables() et effectué la commande "bench" dans la console, nous avions des bugs puisque le nombre de node parcourus avait diminué par rapport au nombre de node parcourus par le programme initial.

Ce problème était lié à une inversion de boucle for dans la méthode qui fût ensuite corrigé dans le dernier commit (**commit bf9454c**).

Nous n'avons pas utilisé de tests asserts -ce qui aurait été plus efficace, c'est vrai- et n'avons pas non plus fait assez de tests ou de commits intermédiaires. Cela a eu pour conséquences de nous faire perdre pas mal de temps et, sans déterminer vraiment où étaient nos erreurs, a nous faire reprendre entièrement les fichiers originaux pour les retravailler ensuite.

1. **Performances finales**

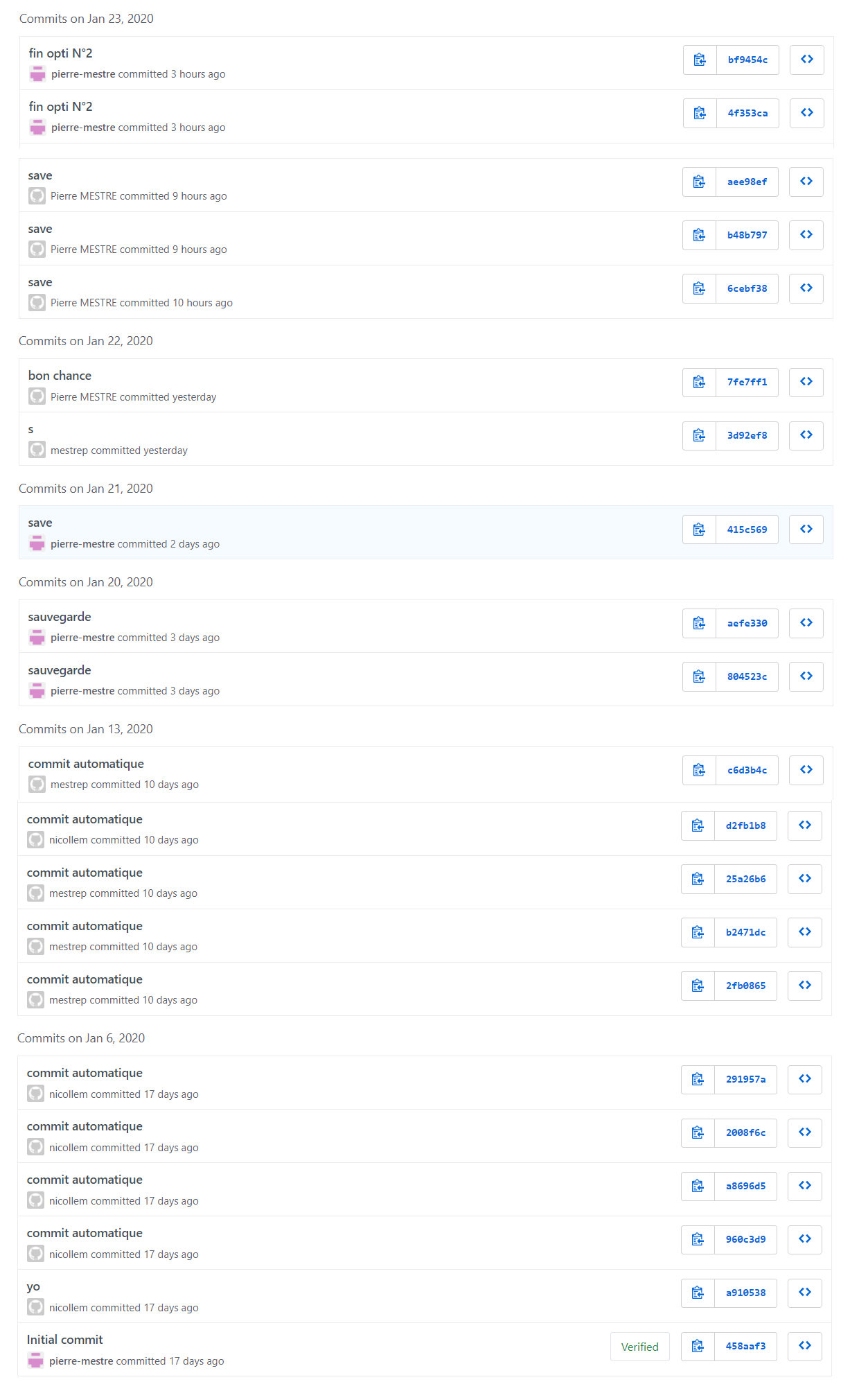
****

1. **Comparaison des performances**

|  |  |
| --- | --- |
| **TSCP** | **TSCP modifié** |
|  |  |

Le nombre de node analysés au final est le même qu’initialement, donc il n’y a pas eu de perte à ce niveau là, ni de bugs. Le temps d’exécution à été diminué de 100 ms puisque le programme analyse maintenant environ 20.000 nodes de plus par seconde. Le programme a donc bien été optimisé.

1. **Annexe des commits**



Configuration Winboard + sauvegarde «Ordi VS Nous»

Màj Sublime Text + **(1) Ajout tab pospieceW, pospieceB, piece\_dead**

1. **Ajout de des «useful squares» (defs.h)**
2. **Correction des «useful squares»**

**(4) Modif tab pospieceW, pospieceB + ajout posIniPieceW et posIniPieceB**

**(5) Modif PIECE\_DEAD + parcours des tab pospieceW et pospieceB dans init\_board()**

**(6) Modif init\_board() + modif attack() (copie ORIGINAL + changement for et PAWN pour 3 <= i <= 7)**

Configuration Sublime Text **+ (7) Ajout initAttackTables() pour test + ajout canAttack[6][64][64] data.h + Modif des des «useful squares»**

**(8) Modif initAttackTables() + petites modif tab can\_attack**

**(9) Modif des «useful squares» + ajout tests assert**

**(10) Retire commit (9)**

**(11) Correction initAttackTables()**