Idea inspired from A-B:

As mentioned before, [Alpha-beta pruning](https://en.wikipedia.org/wiki/Alpha-beta_pruning) speeds the [minimax algorithm](https://en.wikipedia.org/wiki/Minimax_algorithm) by identifying cutoffs, points in the [game tree](https://en.wikipedia.org/wiki/Game_tree) where the current position is so good for the side to move that best play by the other side would have avoided it. Since such positions could not have resulted from best play, they and all branches of the game tree stemming from them can be ignored. The faster the program produces cutoffs, the faster the search runs. The null-move heuristic is designed to guess cutoffs with less effort than would otherwise be required, whilst retaining a reasonable level of accuracy.

The null-move heuristic is based on the fact that most reasonable chess moves improve the position for the side that played them. So, if the player whose turn it is to move can forfeit the right to move (or make a [null move](https://en.wikipedia.org/wiki/Null_move) - an illegal action in [chess](https://en.wikipedia.org/wiki/Chess)) and still have a position strong enough to produce a cutoff, then the current position would almost certainly produce a cutoff if the current player actually moved. (basically that you are very far ahead to your opponent)

**Workload decrease**

|  |  |
| --- | --- |
| **Number of plies (half-moves)** | **Number of possible games** |
| 1 | 20 |
| 2 | 400 |
| 3 | 8,902 |
| 4 | 197,281 |
| 5 | 4,865,609 |
| 6 | 119,060,324 |
| 7 | 3,195,901,860 |
| 8 | 84,998,978,956 |
| 9 | 2,439,530,234,167 |

Implement

Swap the side whose turn it is to move, then performs an alpha-beta search on the resulting position to a shallower depth than it would have searched the current position had it not used the null move heuristic. If this shallow search produces a cutoff, it assumes the full-depth search in the absence of a forfeited turn would also have produced a cutoff.

1. Appropriate evaluation

Logarith

Graphical user interface, chart

Description automatically generated

Chart, line chart

Description automatically generated

Linear

Chart, line chart, histogram

Description automatically generated

Chart, line chart, histogram

Description automatically generated

These graph just to show that it’s really faster, nothing to say

1. Difficulty/solution

As I mentioned above, null-move have to make no move without any change in chess board, just switch the side to move – which is an illegal move in python Chess, however we have to do some evaluate in the new chess position and then undo all of it, so we have to make a null-move that can be push or pop like a legal move, which the provided function chess.move.null in python chess can’t do.

Not like alpha-beta which reduce workload without reduce accuracy compared to minimax, null-move pruning may return to a worse move (it still doing the maximize and minimize job as it based on alpha-beta, but with reduced accuracy due to reduced depth)

So the main difficulty here is that we have to choose R (level of depth reduced) to balance the workload reduce and accuracy reduce.

We haved tested some value of R (with depth 8) and the result is that R=2 is the best thing we got (in fact we use the research result over years from Heinz, E.A.: Adaptive null-move pruning. ICCA Journal 22(3), 123–132 (1999), <https://www.researchgate.net/publication/220962442_Extended_Null-Move_Reductions>) -> R=1 is too small to have workload significantly reduce, but R=3 is too large, making the null-move too likely to miss tactics.

1. Improvement

Nothing yet