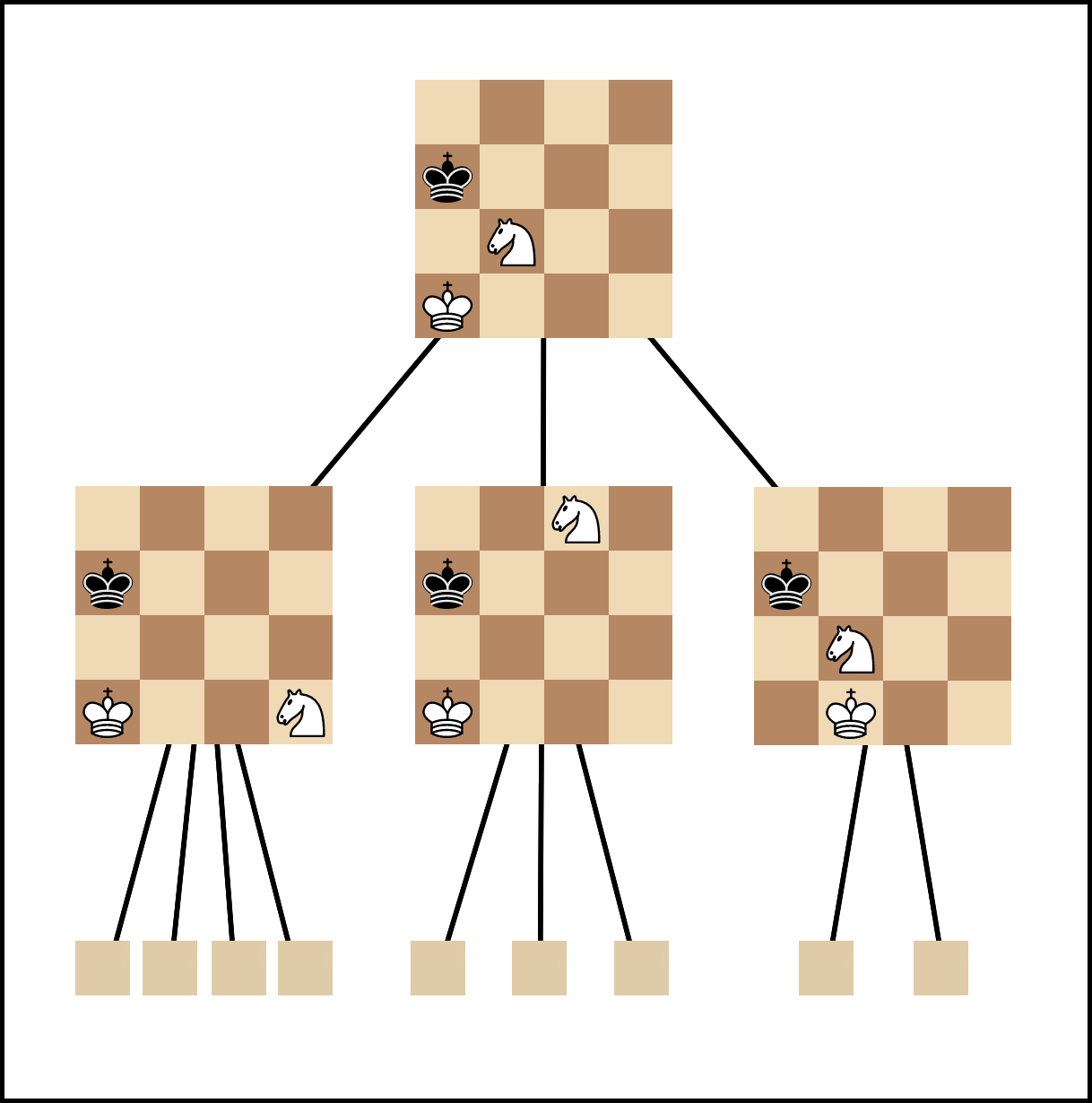
Negamax Report

# The details of the solution/method used to solve the problem

Since Negamax algorithm is simply a modification of Minimax algorithm, the search space of the problem can also be illustrated as a tree:



Each node of the tree is a state of the board, and each branch is one move the player makes. The algorithm is to scan all possibilities of the current board after a fixed number of moves (I’ll call it DEPTH). To put it a clearer way, the current board is the root of our search tree. Each move from the root will lead to a 1st level node, each move from the 1st level boards will lead to a 2nd level node… until we reach our DEPTH (in my code I chose 3). This repetitions is the same with playing DEPTH moves from the current board again and again, each time choose a new move to make, whether it’s a 1st or 2nd move. Among all the possibilities we scanned, we will then choose the best move to make, and to evaluate this there are actually a several ways. In our project we used the same evaluation system for convenient of comparing the algorithms later.

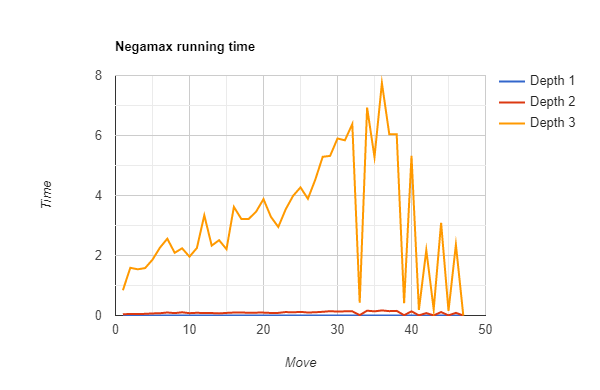
About coding, Negamax algorithm is basically MinMax algorithm but instead of having separate subroutines for finding Min and Max, we combine those two into one and passed one part as negative, one as positive as follows: max(a, b) = -min(-a, -b).

Negamax algorithm simply shorten the MiniMax algorithm to roughly half the lines of MiniMax while maintaining identical performance.

Since Python already provides a Chess library, including all necessities so all we have to do is to come up with the algorithm and apply them, the library does all the “dirty work”.

# The appropriate evaluation of the system’s performance

The algorithm guarantee us of a very promising result with good evaluation system and a big enough DEPTH. Notice that the bigger DEPTH is, the more arccuracy the algorithm.



The graph presents Negamax algorithm running time with a fixed DEPTH equals 3.

# The issues/difficulties occurs in the course project’s implementation, and how you resolved them

The issues of this algorithm lies in the fact that it’s an exhaustive search algorithm. That means with bigger DEPTH (for better arccury), the search time is too huge. With a DEPTH = 5 the game is nearly unplayable because time spent on search is too long.   
For a more search time I incorporated the Alpha Beta Pruning (with this one see my friend’s part) into Negamax. The idea is to cut off the branch which we know for sure will not give us the best move, hence reduce the search time.

Also the running easily bumps into loops or just evaluating a move again and again…

# The discussions/findings/conclusions and proposals for system improvement

This is a simple algorithm, both for understanding and implementing it. Improving and modifying it to be better is the real challenge. It’s not a perfect algorithm but I think it’s a decent way to start with an AI project for myself.