PYTHON PROJECT – V0 Report
AI Chess BOT
GROUP 13

Members –
Ayush Tankha
Jatin Singh
Peter Barnabas Keszthelyi

Motivation
All three of us being avid chess fans from a really young age have always been intrigued by the ever-changing chess scenario in the digital world. With the first engine “Deep Blue” that beat Gary Kasparov in a famous engine vs human chess game made the breakthrough for AI in chess. Since then, AI in chess has been dominant and human intelligence is not even close when it comes to evaluating complex chess positions. With chess engines like Leela and Alpha Zero from Deep Mind, with an expected Elo of 4000, we are advancing in the game of Kings at a rapid pace.

To understand the logic and concepts behind a chess engine and to tickle our common interests we decided to take up a project to create and implement a chess AI bot using decision trees.

We are going to name the AI bot as **SPEED_CHESS**

Important Note

For this chess engine we'll use python and the chess library, which is really useful, for things like legal moves the number of legal moves the ai will be entirely built from scratch.

Conceptualization and Theory behind V0, V1 and V2

First starting from a certain position our engine will create a decision tree each node in this decision tree corresponds to a legal move of this chess position reached by its parent afterwards each terminal node will be assigned a value this value is the corresponding terminal node's chess position evaluation function output but what is the evaluation function what does it do well as you might have guessed the evaluation function takes a certain chess position and it will return its value from the engine's perspective

Chess Position Evaluation using points

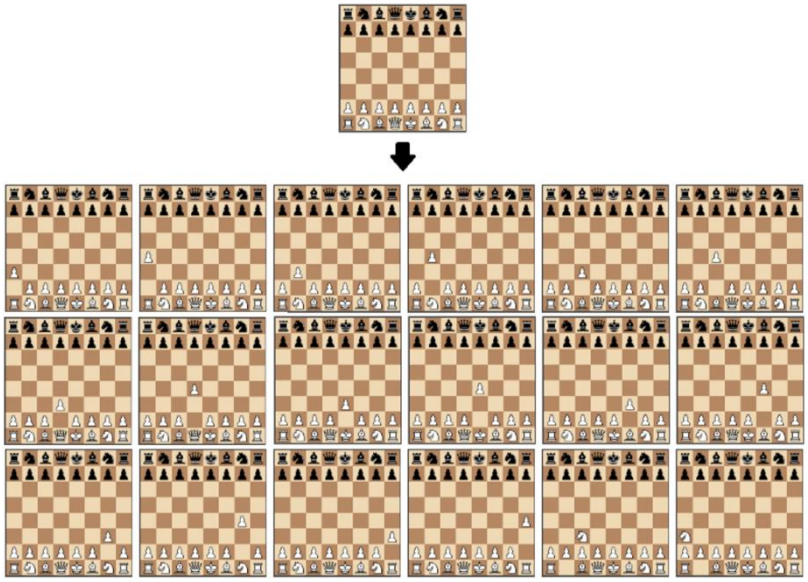
We want to emphasize this point it always returns its value from the engine's perspective now the engine is going toback propagate the values of the terminal load

if it is the engine's turn it will back propagate the highest value of the child's to their parent if it's the human's player turn player's turn it will pr back propagate the lowest value of the child's to their parents this is because the min max algorithm assumes that the opponent will pick the move which would put the engine at the biggest disadvantage so if you are maximizing it means that it's the engine's turn and would and it will back propagate the the highest value and if it's minimizing it will backpop.

It implies that it's the human player's turn and it will back propagate the lowest value now there is a certain modification that we can bring to this algorithm to make it more efficient, and it's called alpha beta pruning

We are going to show two cases of alpha beta pruning so for case one supposes we have already computed this group of node and now we're trying to maximize this set of node and we already have this candidate when we're going to compute this other group of nodes if we reach something that is lower than 5 it means that the node since this is minimizing will be less than or equal to four but since here we're maximizingwe know uh we already know that the this branch is not going to be picked here so we can just skip the rest of the computation of this group of nodes because we know they're not going to be useful

It is to be noted that alpha beta pruning does not change the result of the min max algorithm at all it will output the same answer, but it will do so in less time in fact significantly less time.

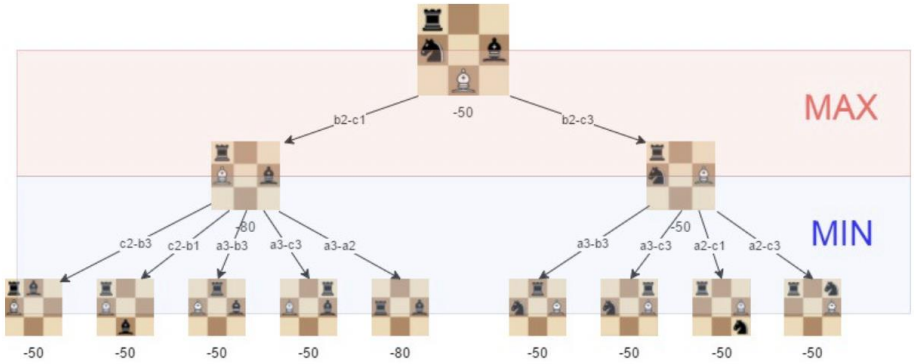


A visualization of the move generation function. The starting position is used as input and the output is all the possible moves from that position.

Implementation of V0

Now let's implement the code, so first thing first we're going to create a new file called chess engine.pi we import module and it's the chess library. We create a class named Engine and then we're gonna create constructor. It has these perimeters -board, max depth of search or max step for the decision tree and our engine's color. We create a big recursion function that we're just going to call engine and this recursive function will includehe min max algorithm with alpha beta pruning

We're gonna say the initial parameters are like our base case for the recursion so if the depth reaches itself point max depth or self.board.legal_moves.count()
moves equals zero so this is simply a tool provided by the the chess library that gives us the amount of legal moves in a certain position and if it's zero then we can't go further in the decision decision tree so we must stop and we return self.evaluation
and this evaluation thing of this evaluation function we will create later and it's basically going to take our engine's iteration board and return its value



A visualization of the minimax algorithm in an artificial position. The best move for white is **b2-c3**, because we can guarantee that we can get to a position where the evaluation is **-50**

So next if we haven't reached maximum depth or if there's still legal moves we're gonna get a list of the legal moves of the current position so move list equals list self.board.legal_moves we're gonna initialize the candidate

The new candidate will be used as a recursive parameter and it's gonna give the information of where what our best candidate is while we're maximizing or minimizing is for the child nodes.

Next Step

For V1 we will further create functions within Engine class for evaluation, reset parameters , mating opportunities while using the decision tree and depth of the engine.