Contents

Category preference

Search problems used in Game Development.

Domain selection and Familiarization

The project falls under the Entertainment domain. Just creating engaging games to keep minds busy. To be specific, the project is the implementation of a Chess AI bot using search algorithms.

Disclaimer : The Chess bot does not use any form of Deep learning. Its primary engine is made up of search algorithms that are designed to reduce damage to self and inflict damage to opponent.

It uses the mini-max algorithm and Alpha-beta search

Problem Identification

Human Problem –

- It is hard to play chess while evaluating many possible future moves. Human brain power is limited. It is better to have a machine do that work.
- Chess trainers are expensive and hard to come by - A chess bot makes a better chess trainer than many humans.
- It is not easy to find human play-mates. People are busy. A chess bot solves the problem of finding a play-partner .It is available throughout.

Objectives

1. To build a system which evaluates possible moves and executes the best alternative depending on the difficulty level that the human player has set.
2. To show the performance of the search algorithms on the user interface (on the main page).

Project justification

To chess lovers :

1. Top players today use chess engines extensively to analyze positions and generate ideas.
2. The Chess AI acts as an available playmate – A chess bot solves the problem of finding a play-partner. You can play with it anytime or anywhere.
3. The chess AI acts as a cheap ,accessible , skilled trainer.

Related Work

There are many other Chess bots. However they can be roughly categorized into :-

- Bots that are made using search algorithms only.
- Bots that are made with a neural network as the main engine.
- Bot made by combining both neural networks

Concepts from the Related works.

Bots using search Algorithms only :

Chess engines are complex. However, in simplest terms, they do two important things:

1. **Evaluate.**

Chess engines look at individual positions and evaluate which position is better. Almost all chess engines display an evaluation number, or “eval,” based on the same scoring that most chess players use (a pawn being worth one point, a minor piece three, etc). Each chess engine does this differently, but most engines look at things like material on each side, all the threats on the board, the king safety, and pawn structure.

The cumulative score of the best evaluation in the future is summed up to one number. Traditional engines evaluate similarly to humans because they were designed by humans. Neural net engines evaluate differently.

Evaluation is based on : -

- The weight of the chess piece (a queen has more weight than a pawn).
- The strategic positioning of the chess pieces (for example, a knight at the center of a board has more strategic advantage than a knight in a dense corner.

2. **Search**

Like good chess players, engines try to look deeply into the position. The further ahead they can see, the better the move they can make now, as they can evaluate positions that will result after the best possible moves in the future.

Each individual chess move is called a “ply” (a layer), and the depth is explained in how many ply deep. At 20 ply (10 white moves, and 10 black moves), most engines are already evaluating far deeper and stronger than humans. Depending on the time allowed and the complexity of the position, engines can look more than 50 ply deep.

From the current position, an engine starts to look at all of the possible moves and replies. And then all of the possible replies to that. And then all of the possible replies to that! Imagine there are 32 possible moves in any position. After four moves, there are already more than one million positions to evaluate. After just four more moves, that would be more than one trillion position. That becomes extremely unpractical.

So instead, engines try to use smart “pruning” to look deeply at just the most promising lines, and ignore the obviously bad ones. The engine keeps a running “principal variation” (PV) of the most promising moves in every position.

Traditional chess engines use complex evaluation functions and intelligent search algorithms to find the best possible move. Their power is also related to how much CPU processing power the phone, computer, or server has. The more powerful and plentiful the CPUs, the stronger the engine becomes.

Popular examples of Traditional Search Engines – include : Stockfish, Komodo , Houdini, Fire

Popular examples of Neural network Engines – include : Antifish, Lc0, Leelenstein.

Architecture

Here is the proposed architecture.

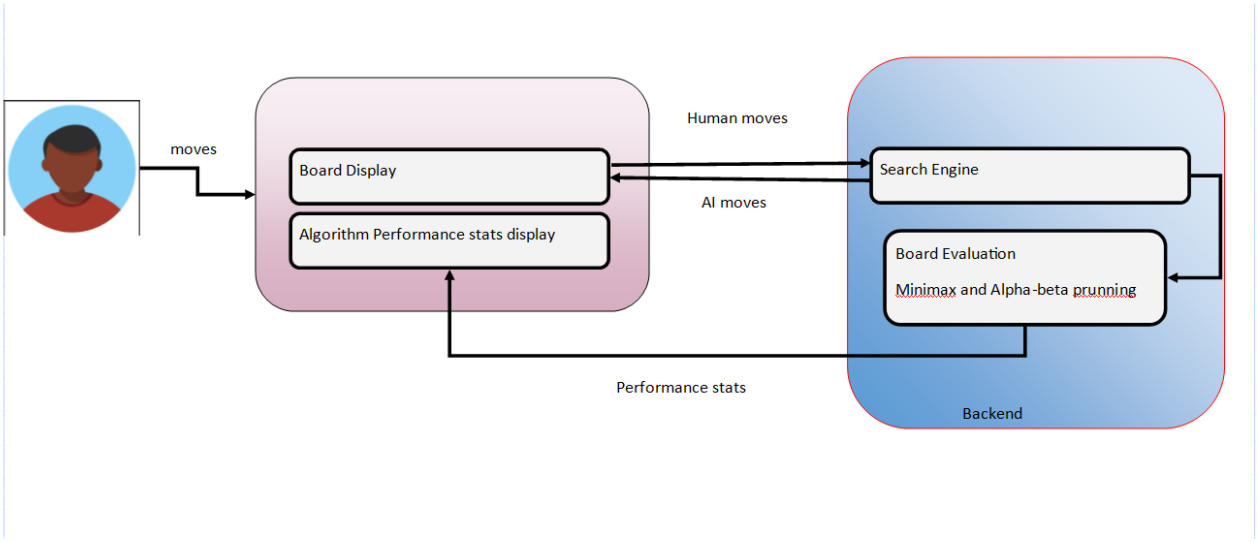

Figure 1 General architecture

Brief Description

The user moves the chess pieces in accordance to the chess rules.

The Board display displays how the chess pieces are move (both by Human and AI).

The Algorithm performance stats display the total moves evaluated and time taken to make a decision.

The mini-max algorithm tries to find the most best move . It is limited to the depth that the user has chosen.

The Alpha-beta algorithm minimizes the steps needed to execute the mini-max algorithm.

Requirements

Functional Requirements

1. The user is able to drag and drop valid pieces.
2. The user is able to undo moves up to the games beginning.
3. The user is able to redo moves.

Methodology

The methodology used is the Waterfall Method of development. This is because the project is fairly simple and straight forward. All the steps in making it are well defined and it does not need constant improvement to suit consumers. It is a project that gets produced once and very little changes happen from there on.

I could not use methods like Spiral and Rapid development ; The main aim is not to understand consumer's needs(they are already known).Also, the project does not need the creation of many prototypes.

Demonstration

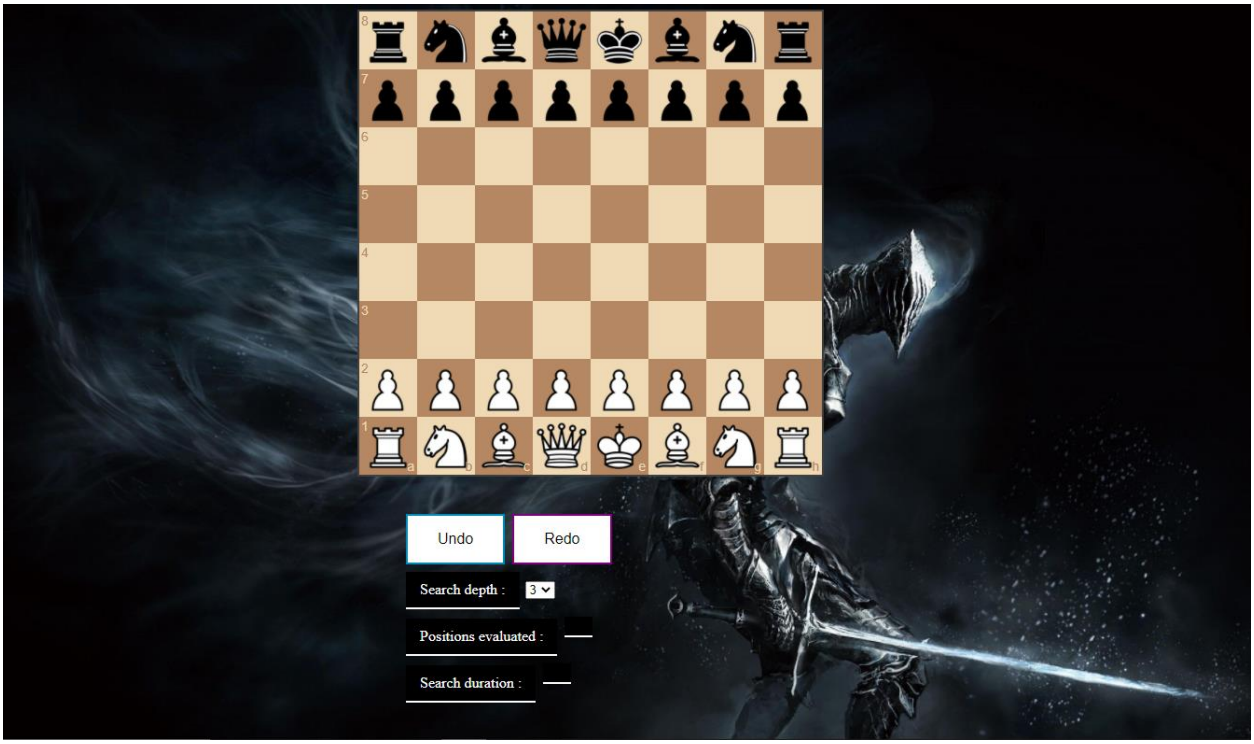

Figure 2How the game starts

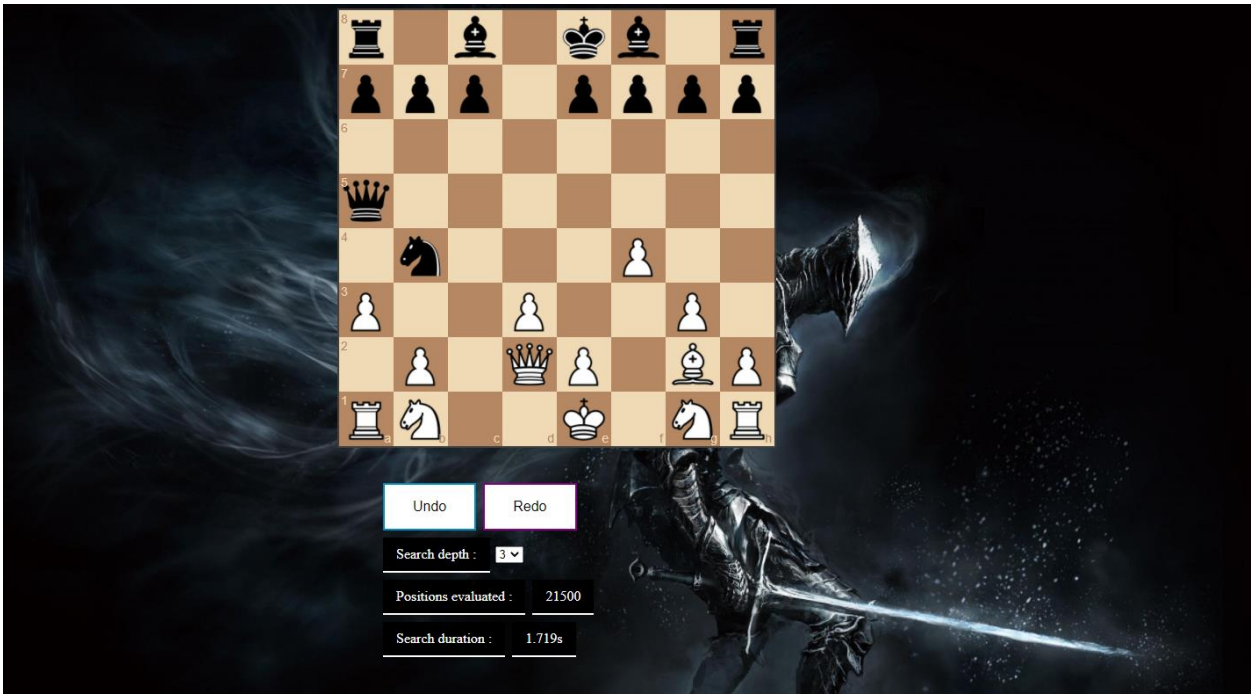

Figure 3Performance stats at the bottom