

Henry Kwan, Elias Strizower, Alex Godziela
Chess AI
User Manual

We created an intelligent chess AI that can optimize its moves real-time in order to win a chess game against a human player. There are 3 chess agents that we wrote: 1-ply random which makes random moves each turn, 1- ply best move which takes the move with the highest returned point value, and 2-ply minimax which looks two moves ahead in order to maximize its own advantage while minimizing its opponent's advantage. The motivation for our Chess AI was exploring machine intelligence, which can be found below.

One of our goals for this class was to explore intelligent machines and because of the wide array of logic that is used in Chess, we can actively determine the intelligence of our AI by analyzing the moves it is making. Chess is a mathematical game that heavily employs strategy. What this means is that a player needs to find a comfortable balance of attacking the opponent with the ultimate goal of checkmate as well as defending its own pieces to prevent one's own king from being cornered. In order to get to checkmate, each move needs to be well calculated. Since each chess piece has a given point value, we can easily differentiate good moves from those that are bad and it paves the way for our intelligence analysis.

In able to accomplish the goals of our project, we believe that basic heuristics in combination with the minimax algorithm will successfully implement an intelligent chess agent. The role of the heuristics is to assign a numerical value to each of the chess pieces and subsequently each chess move has an associated point value. The way the minimax algorithm

works is that we know each of the point values increases from pawn to queen, and the minimax algorithm maximizes the point value of the AI while minimizing the point value of the opponent based on the AI's previous move.

We chose this approach because it fits best with the manner in which chess is played. Specifically, having the minimax algorithm evaluate the best move at each turn should work well with the game of chess given the dynamic nature of the game. Although other approaches were considered, such as deep learning and reinforcement learning, we believed that it would be less effective to train a model on the best move since chess changes drastically each turn. This is why we believed that it was best to implement a model that could take any board state and return the best move solely based on that.

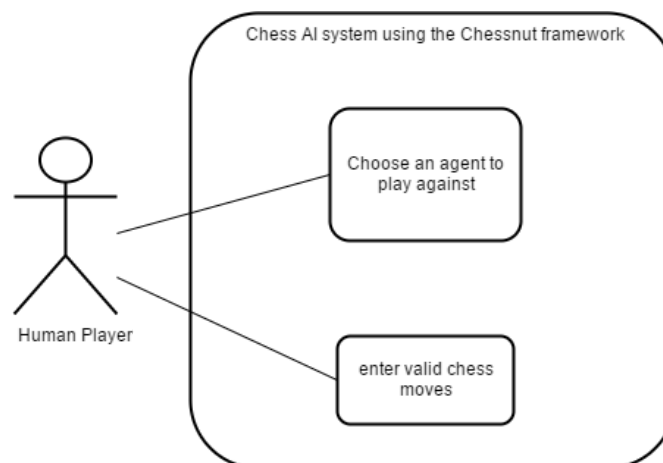


Figure 1: UML Use Case Diagram

Instructions

Player - white (capital letters on bottom of board) AI - black (lowercase letters on top of board)

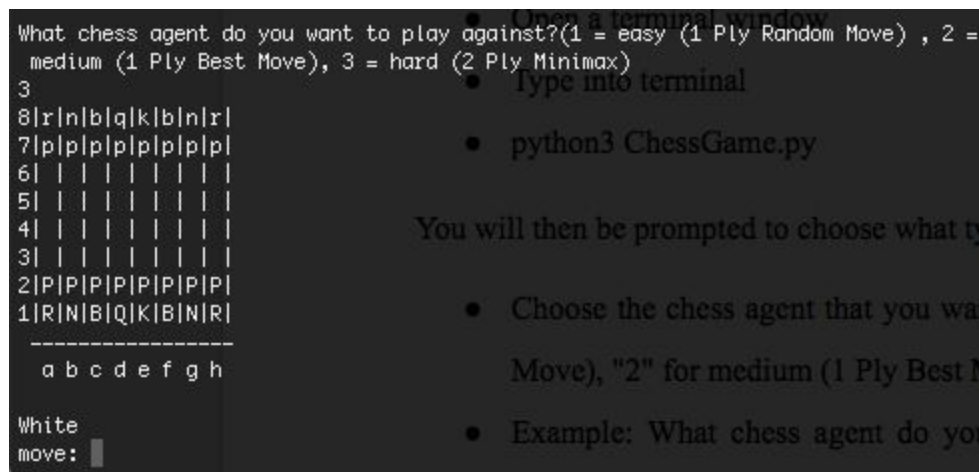
- Open a terminal window
- Type into terminal: `python3 ChessGame.py`

You will then be prompted to choose what type of agent you want to play against:

- Choose the chess agent that you want to play against: Enter "1" for easy (1 Ply Random Move), "2" for medium (1 Ply Best Move), and "3" for hard (2 Ply Minimax)
- Example: What chess agent do you want to play against?(1 = easy (1 Ply Random Move) , 2 = medium (1 Ply Best Move), 3 = hard (2 Ply Minimax)

User input: 3

After the agent is chosen, the GUI representing the board will appear in the terminal window:



```
What chess agent do you want to play against?(1 = easy (1 Ply Random Move) , 2 =
medium (1 Ply Best Move), 3 = hard (2 Ply Minimax)
3
8|r|n|b|q|k|b|n|r|
7|p|p|p|p|p|p|p|
6| | | | | | | |
5| | | | | | | |
4| | | | | | | |
3| | | | | | | |
2|P|P|P|P|P|P|P|
1|R|N|B|Q|K|B|N|R|
-----
 a b c d e f g h
White
move: 
```

Figure 2: Chess game GUI in terminal window

The GUI represents the state of the board and updates itself as the game is played. The pieces are symbolized by the first letter of their name (ie. pawn = p, rooke = r, etc.). In addition, the color of the piece is designated by the case of the letter. Uppercase letters are the white piece's, human player's pieces, and the lowercase letters are the black pieces, the AI's pieces.

The next step is to enter move as current position + next position

- Example: e2e4 -> piece moves from e2 to e4

Further rules:

When move requires pawn promotion (pawn reaches end of board), enter move as current position + next position + piece pawn is promoted to

- Example: a7a8b -> piece moves from a7 to a8 and turns into bishop