Chess Game with Minimax AI

Purpose:

The code is an implementation of a simple chess engine using the minimax algorithm. It allows a user to play against the engine, which selects its moves based on the minimax algorithm with a fixed depth search.

Imports:

- **chess**: This is a Python library for working with chess games. It provides functionalities for board representation, move generation, and legality checking.
- **random**: Python's built-in library for generating random numbers and making random selections. Here, it's used to introduce randomness in the AI's move selection.
- **sys**: A standard library providing access to some variables used or maintained by the Python interpreter and to functions that interact strongly with the interpreter. It's imported here but not used in the code snippet.
- **IPython.display**: A module used for displaying SVG images in Jupyter notebooks. It's used here to display the chess board.

Piece Values:

- The **piece_values** dictionary assigns numerical values to each piece type. These values are used to evaluate the board's position. The values are based on general guidelines in chess, such as pawns being worth 1 point and queens being worth 9 points.

Minimax Algorithm:

- **minimaxRoot Function**:
    - This function is the entry point for the minimax algorithm. It determines the best move for the current player.
    - It takes into account whether it's maximizing or minimizing player's turn.
    - For each possible move, it calls the minimax function to evaluate the resulting board state.
    - It returns the best move found.
- **minimax Function**:
    - This is the core of the minimax algorithm, which recursively evaluates board positions.
    - It explores possible moves up to a certain depth and evaluates each resulting position using the **evaluate_board** function.
    - It alternates between maximizing and minimizing player's turns.
    - It returns the best score found.

Board Evaluation:

- **evaluate_board Function**:
    - This function evaluates the board's position by summing up the values of all pieces on the board.
    - It uses the **piece_values** dictionary to assign values to each piece.
    - The evaluation score is a simple sum of the values of all pieces on the board.

Main Function:

- Initializes a new chess board and displays it.
- Enters a loop where it alternates between user and AI moves until the game is over.
- For the user move, it takes input in algebraic notation, validates it, and makes the move if it's legal.
- For the AI move, it calls **minimaxRoot** function to find the best move with a fixed depth search, and then introduces randomness by choosing one of the best moves randomly.
- After each move, it displays the updated board.
- Finally, it determines the game result and prints it.

Possible Improvements:

- Implementing alpha-beta pruning for better efficiency in the minimax algorithm.
- Enhancing the evaluation function to consider more factors like piece mobility, pawn structure, king safety, etc.
- Adding more sophisticated move ordering techniques to improve search efficiency.
- Improving the user interface with features like highlighting legal moves, showing possible captures, or providing hints.

Conclusion:

This code provides a foundational implementation of a chess engine using the minimax algorithm. It can be improved in various ways to make it stronger and more user-friendly, such as optimizing the algorithm, refining the evaluation function, and enhancing the user interface.