**Ai Mini Project:**

Aroz Imran 21L-6246

Hshim Siddique 21L-6230

Muneeb Bhatti 21L-6257

**Detailed Report of the Chess Game Code**

**1. Introduction:**

* The provided Python script implements a simple chess player game where a user can play against an AI opponent.
* The AI opponent uses the Minimax algorithm with Alpha-Beta pruning to determine its moves.
* The game is played on a standard 8x8 chessboard.

**2. Libraries Used:**

* chess: This library is used for handling the chess board, moves, and game logic.
* numpy: Utilized for numerical computations and handling infinity values.

**3. Piece Values:**

* Basic piece values are defined for evaluation purposes.
* Each type of chess piece (pawn, knight, bishop, rook, queen, king) is assigned a value representing its relative strength.

**4. Evaluation Function:**

* The evaluate\_board(board) function calculates the strength of the chess board based on the total value of pieces for both players.
* It iterates through each piece type and counts the number of pieces on the board for each player, multiplying by the respective piece value.

**5. Minimax Algorithm:**

* The minimax(board, depth, alpha, beta, maximizing) function implements the Minimax algorithm with Alpha-Beta pruning.
* It recursively searches through possible moves up to a specified depth to find the best move for the current player.
* The algorithm alternates between maximizing and minimizing players to evaluate the potential outcomes of each move.

**6. Board Visualization:**

* The print\_board\_with\_boxes(board) function prints the chessboard with colored squares and pieces.
* It uses ASCII characters to represent the chess pieces and colors the squares to enhance visualization.
* Light squares have a cream background, and dark squares have a light brown background.

**7. Playing the Game:**

* The play\_chess() function allows the user to play against the AI opponent.
* It initializes the chessboard and prompts the user to specify the search depth for the AI.
* The game alternates between the user and the AI, allowing each player to make moves until the game is over.
* After each move, the current state of the board is displayed.

**8. User Input and Move Format:**

* Users input their moves in UCI (Universal Chess Interface) format.
* For example, "e2e4" represents moving a pawn from e2 to e4.
* The script validates user moves to ensure they are legal moves on the current board configuration.

**9. Game Outcome:**

Once the game is over, the script determines the result (checkmate or draw) and announces the winner if applicable.

If the game ends in a draw, the script notifies the players accordingly.

**10. Conclusion:**

* The implemented chess game provides a basic but functional platform for users to play against an AI opponent.
* While the AI uses a simple Minimax algorithm, it demonstrates the fundamental principles of chess AI.
* The game could be further enhanced with additional features such as improved evaluation functions, graphical user interface (GUI), and support for more advanced chess engines or algorithms.

Overall, the provided Python script offers a solid foundation for learning and experimenting with chess AI concepts and algorithms. It provides an interactive platform for users to engage in chess gameplay while gaining insights into AI-driven decision-making processes in chess