Rue Chess

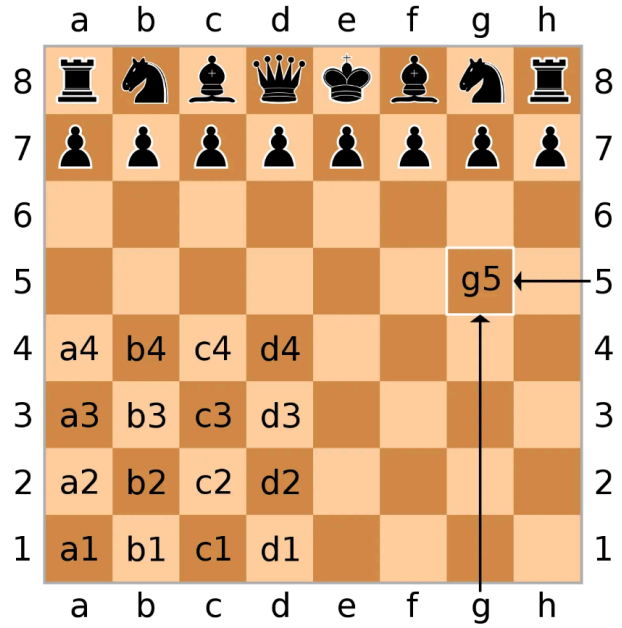
---

1. Introduction

This documentation serves as a comprehensive guide to creating a chess application in Python using the Pygame library. The objective is to provide developers with a detailed understanding of the implementation process, including the algorithms utilized, data visualization techniques employed for analyzing chess engine evaluations, and the integration of various Python libraries such as Pandas, NumPy, Seaborn, and Matplotlib to create heatmaps for evaluating the best move.
The core algorithm used in the chess engine of this application is the minimax algorithm with alpha-beta pruning. The minimax algorithm is a recursive decision-making algorithm that explores the game tree to determine the optimal move for the current player while considering the opponent's possible responses. Alpha-beta pruning enhances the efficiency of the minimax algorithm by pruning branches of the game tree that are guaranteed to be suboptimal.
To visualize the evaluation function of the chess engine and facilitate decision-making, heatmaps are employed. Heatmaps provide a graphical representation of data, where values are depicted using a color scale. In the context of the chess engine evaluation function, heatmaps display the numbers of moves played on a particular square based on a database of chess matches, allowing players to identify the most favorable moves.

1.1 Chess Board:

- A chessboard consists of 64 squares where each square is labeled by an alphabet horizontally (A to H) and a number vertically (1 to 8). For example, if we are to locate g5 on a chessboard, we will move 5 squares vertically and ‘g’ squares horizontally from to left to right



1.2 Approach

Setting up Pygame:

- Begin by installing Pygame, a popular library for building games in Python. Once installed, import it into your project.

Creating the Board:

- Use Pygame's drawing functions to create an 8x8 grid representing the chessboard. Each square on the board is typically of equal size and alternates between light and dark colors to represent the squares' positions.

Defining Squares and Pieces:

- Each square on the chessboard can be represented by its coordinates (e.g., (0,0) for the bottom-left corner). Define a data structure to represent the chess pieces, including their type (pawn, rook, knight, etc.), color (black or white), and position on the board.

Rendering the Pieces:

- Use Pygame's drawing functions and load images to represent the chess pieces on the board. Each piece should be positioned correctly based on its coordinates.

Implementing Drag-and-Drop:

- Utilize Pygame's event handling to detect mouse clicks and movements. When a player clicks on a chess piece, record its initial position. As the player moves the mouse, update the position of the piece accordingly. When the player releases the mouse button, check if the move is valid and update the game state accordingly.

Implementing Game Logic:

- Develop the logic to handle player turns, switching between players after each move.

1.3 User Interface:

- Pygame doesn't inherently support multiple screens, you can simulate this functionality by managing different "states" within your game loop. Each state represents a different screen (e.g., main menu state, options state, game state), and you switch between them by updating the display to render the corresponding elements.
- By organizing your code effectively and structuring it around these different screens, you can create a seamless user experience within your chess application, even though Pygame doesn't provide built-in support for multiple screens.
- This chess application have 3 screen :
    - Main menu , Controls , game screen

- Main Menu Screen: This screen serves as the entry point for the users. It contains options such as starting a new game, accessing options, playing music, and exiting the application. You can implement this by creating a function that handles the rendering of the main menu elements, including buttons and text. When a user clicks on a button, you can change the background to the relevant screen (e.g., the game board or controls screen).

- Controls Screen: This screen provide instructions to the users with clear guidance on how to interact with the options screen and utilize its functionalities effectively.

    - Change Mode (key: “M”):
        - Press the "M" key to toggle between different game modes: Player vs. Player (PvP) and Player vs. AI (PvAI). This feature enables players to switch between playing against another human opponent or challenging the computer AI, providing diverse gaming experiences.

    - Reset Game (Key: "R"):
        - Press the "R" key to reset the current game. This action will revert the game board to its initial state, allowing players to start a new game from the beginning.

    - Change Theme (Key: "T"):
        - Utilize the "T" key to change the theme of the game board. Cycling through available themes, this functionality allows players to customize the visual appearance of the chessboard according to their preferences.

    - Change AI depth (Key: 1,2,,3,4):
        - Press any key among 1,2,3,4 to toggle between different AI depth. This feature enables players to adjust the difficulty level or behavior of the computer opponent, enhancing the gaming experience and providing varied challenges.

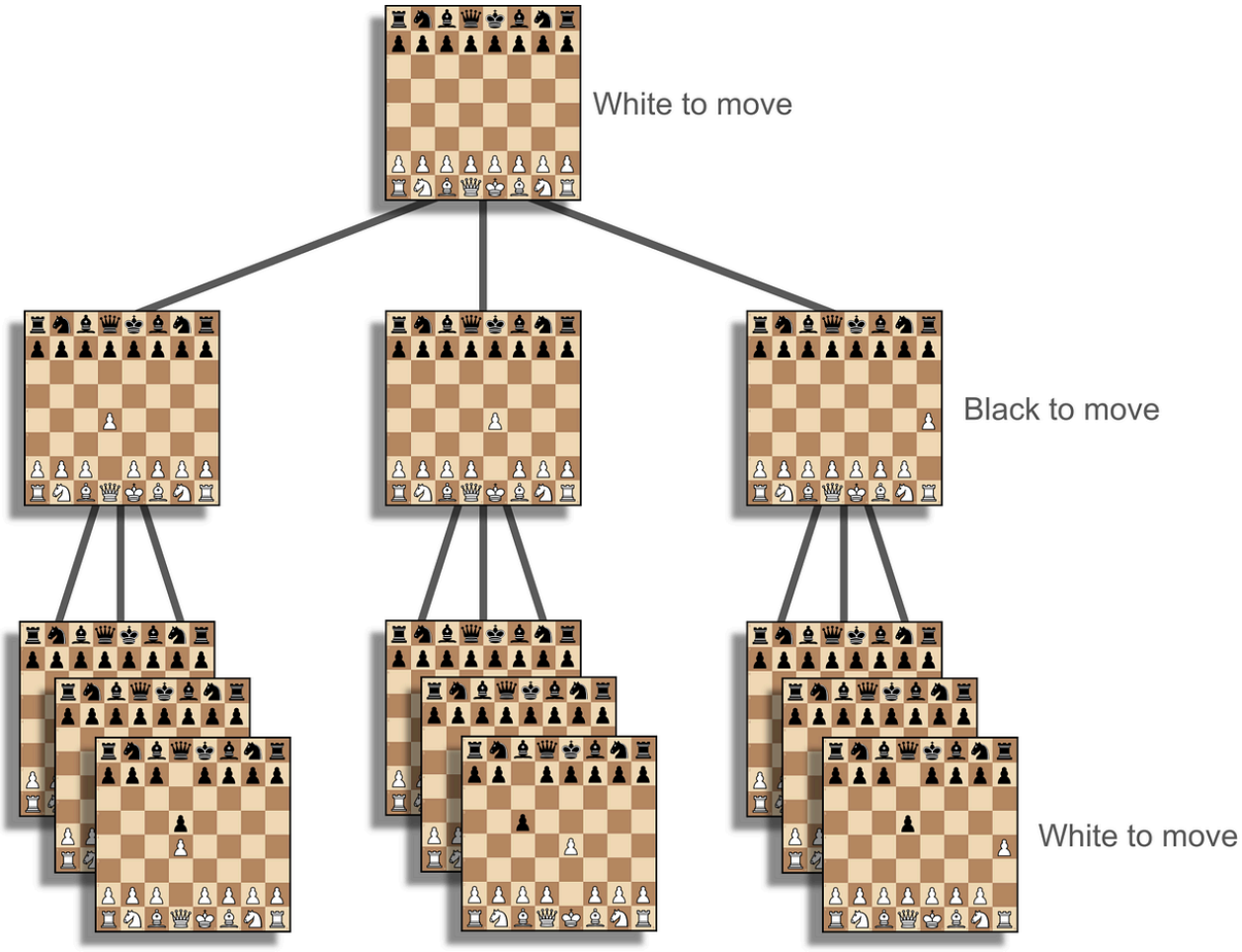
- Chess Game Board Screen: This is where the actual chess game takes place. You can represent the chessboard and pieces as described earlier. When transitioning from the main menu or options screen to the game board, change the background to display the chessboard in the main loop of pygame.

2. Chess AI Engine

2.1 Minimax Algorithm

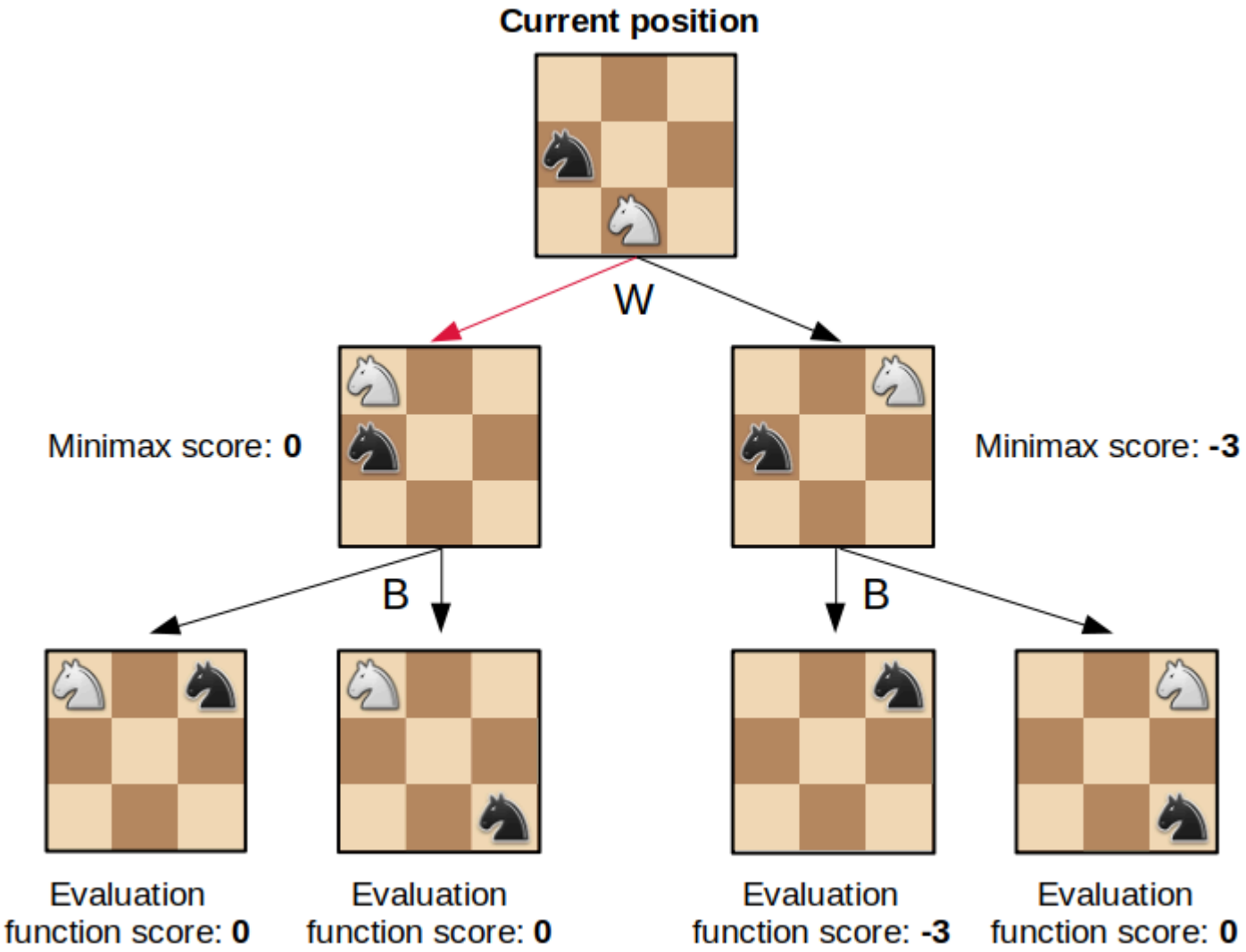
- Min Max is a search algorithm that can be used to determine the best move in a two-player game, such as chess. It works by considering all possible moves by both players (given a certain “depth of engine”) and choosing the move that maximizes the value for the current player while minimizing the value for the opponent.

- The minimax algorithm is a decision rule used in artificial intelligence and game theory to minimize potential losses.
- In our chess engine, it helps in selecting the best move by recursively simulating future game states.



2.2 Evaluation Function:

- We incorporated an evaluation function to assess the current state of the chessboard.
- This function assigns numerical values to positions based on factors like material advantage, piece mobility, and pawn structure.
- The evaluation function in a chess engine aims to assign a numerical value to each position on the board. This value represents how advantageous or disadvantageous the position is for the side to move. The ultimate goal is to guide the engine towards making moves that lead to positions that are favorable according to the evaluation function.
- The evaluation function takes into account various aspects of the chess position, including:
    - Material Advantage
    - Pawn Structure
    - Piece Activity
    - Control of the Center



Chess piece relative value

2.3 Depth-Limited Search:

- Depth-limited search is a technique used in artificial intelligence and game playing algorithms, including chess engines. Its goal is to limit the depth of the search tree explored during the search process.
- To manage computational resources, we implemented depth limitation.
- The engine only explores a certain number of moves ahead rather than searching all possible moves to the end of the game.
- Depth-limited search is a valuable technique for exploring game trees in artificial intelligence and game playing algorithms. By limiting the depth of the search, it strikes a balance between computational complexity and search accuracy, making it an essential component of many intelligent systems, including chess engines.
- Our chess engine have depth upto level 4 . User can adjust the engine depth.

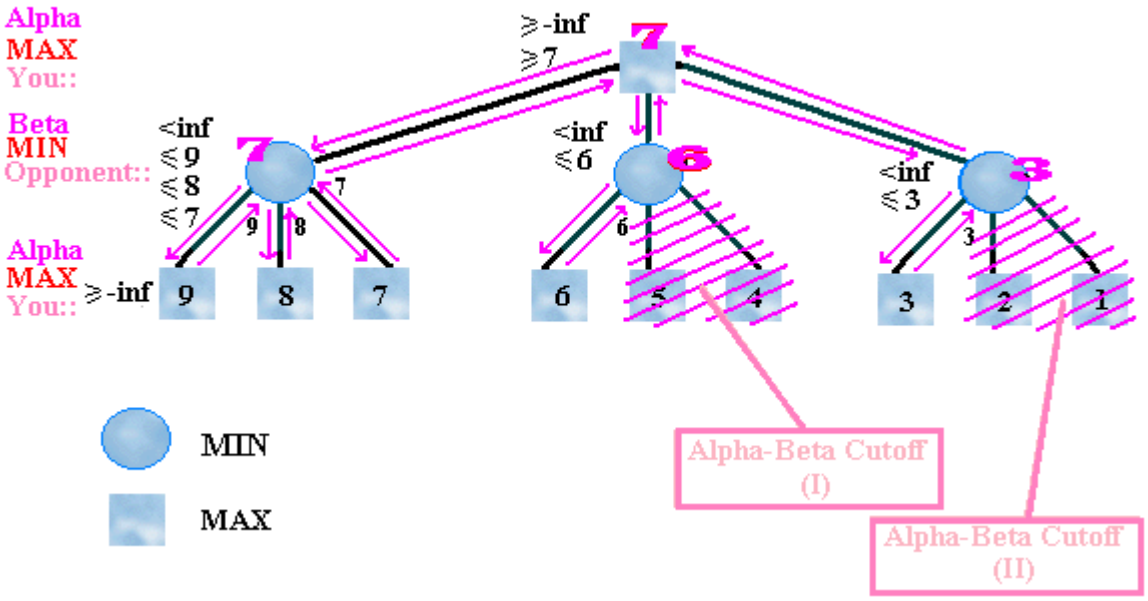
2.4 Alpha-Beta Pruning:

- Alpha-beta pruning is a search algorithm optimization technique used in game-playing algorithms, such as those used in chess engines. Its goal is to reduce the number of nodes evaluated in the search tree, thereby improving the efficiency of the search process.
- Alpha-beta pruning works by eliminating or "pruning" branches of the search tree that are guaranteed to be irrelevant to the final decision. By intelligently discarding these branches, the algorithm can reduce the search effort without affecting the final result.
- During the search process, the algorithm maintains two values, alpha and beta, which represent bounds on the possible outcomes of the current node.
    - Alpha: The best value found so far for the maximizing player (e.g., the player trying to maximize their score).
    - Beta: The best value found so far for the minimizing player (e.g., the opponent trying to minimize the score).

- At each node in the search tree, the algorithm compares the current node's evaluation with the alpha and beta bounds. If the evaluation falls outside the bounds, it means that the current node cannot affect the final decision and can be pruned. This is because the maximizing or minimizing player can already achieve a better outcome through other branches of the search tree.
- When alpha becomes greater than or equal to beta at a minimizing node (indicating that the opponent has a better alternative), or when beta becomes less than or equal to alpha at a maximizing node (indicating that the current player has a better alternative), the search at

that node can be terminated early. This saves computational resources by avoiding further exploration of that branch.

- Alpha-beta pruning is typically implemented recursively, with each recursive call updating the alpha and beta values as the search progresses.



3. Heatmap

- The heatmap will show which squares in chess game have been the most crucial ones. Intense the color of a square, more crucial it has been in the game.
- By leveraging Python libraries such as Pandas, NumPy, Seaborn, and Matplotlib, we can analyze chess matches data and generate informative heatmaps for all 64 squares on the board. These heatmaps provide valuable visualizations of evaluation values, enhancing strategic insights and facilitating better decision-making in chess gameplay.

3.1 Approach:

Data Collection:

- Access a chess matches database containing game positions and evaluation values for each move.
- In this analysis, we will utilize a database containing around 50,000 chess games.

Data Preparation:

- Use Pandas to load and preprocess the data, organizing it into a structured format suitable for analysis.
- The database includes various information such as player names, game outcomes, and more. However, for our analysis, we will focus solely on the match moves data.

Evaluation Value Extraction:

- Extract evaluation values for each chess piece's position from the dataset.

Data Transformation:

- Utilize NumPy to reshape the evaluation values into a 2D array representing the chessboard's 8x8 grid.

Heatmap Generation:

- Use Seaborn and Matplotlib to create heatmaps for all 64 squares on the board.
- Assign the evaluation values to each square and visualize them using a color gradient, where higher values correspond to warmer colors and lower values to cooler colors.

Visualization Enhancement:

- Enhance the heatmaps with additional visual elements like axis labels, grid lines, and annotations to improve readability and interpretability.

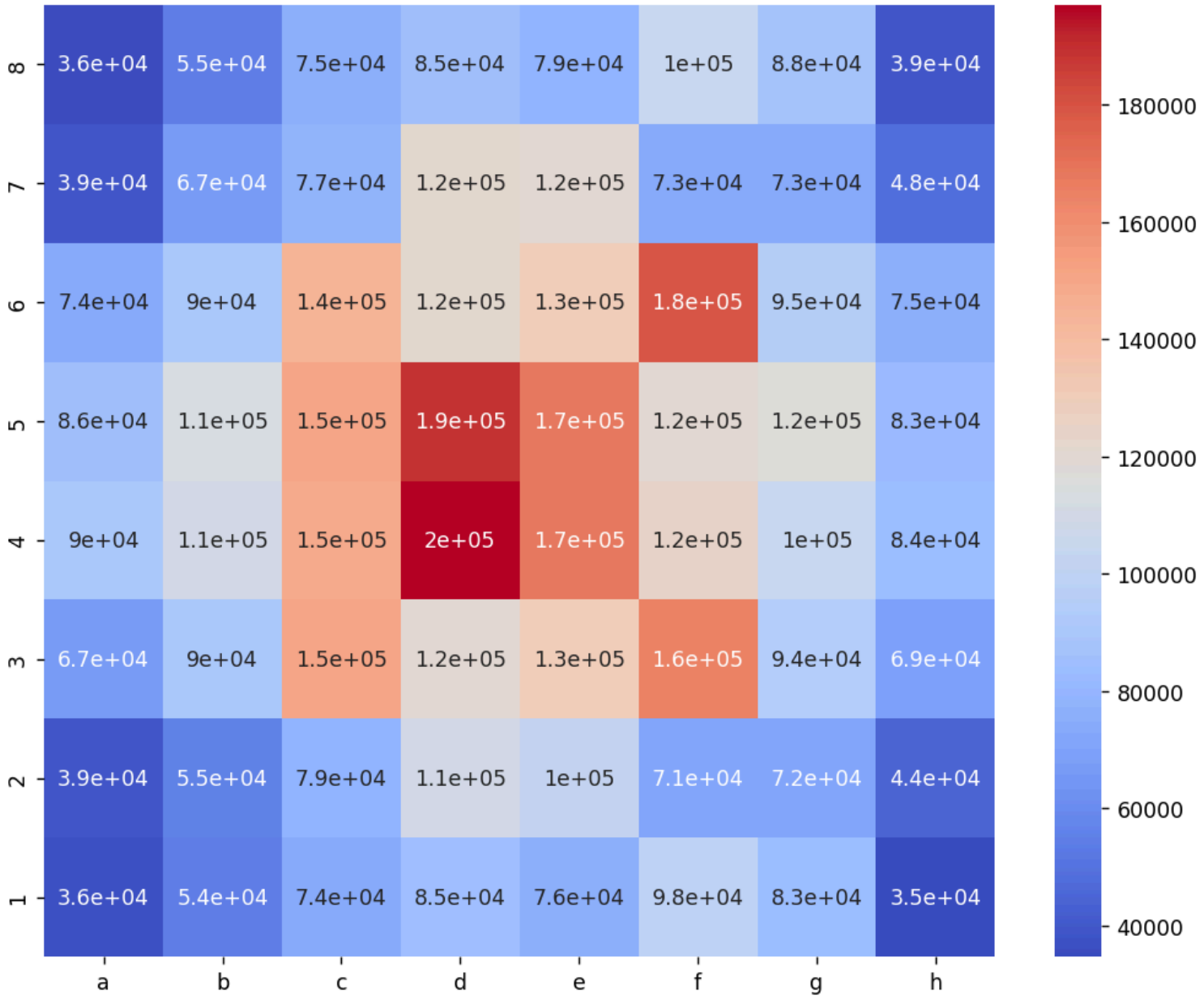
Strategic Insights:

- Analyze the generated heatmaps to gain strategic insights into the strengths and weaknesses of piece positions.

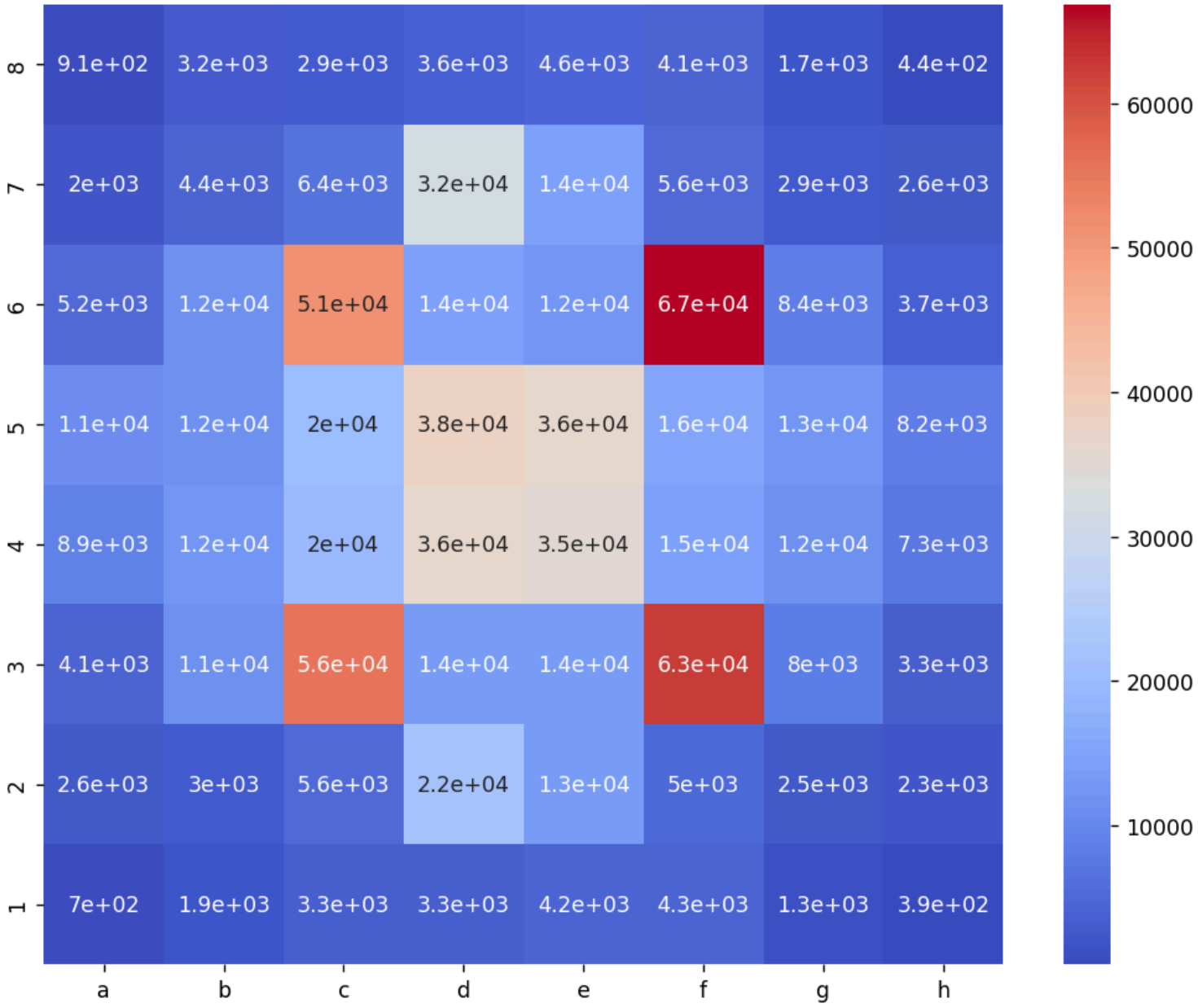
- Identify patterns and trends in evaluation values across the board to inform decision-making in gameplay.

3.2 Chess Pieces Heatmaps :

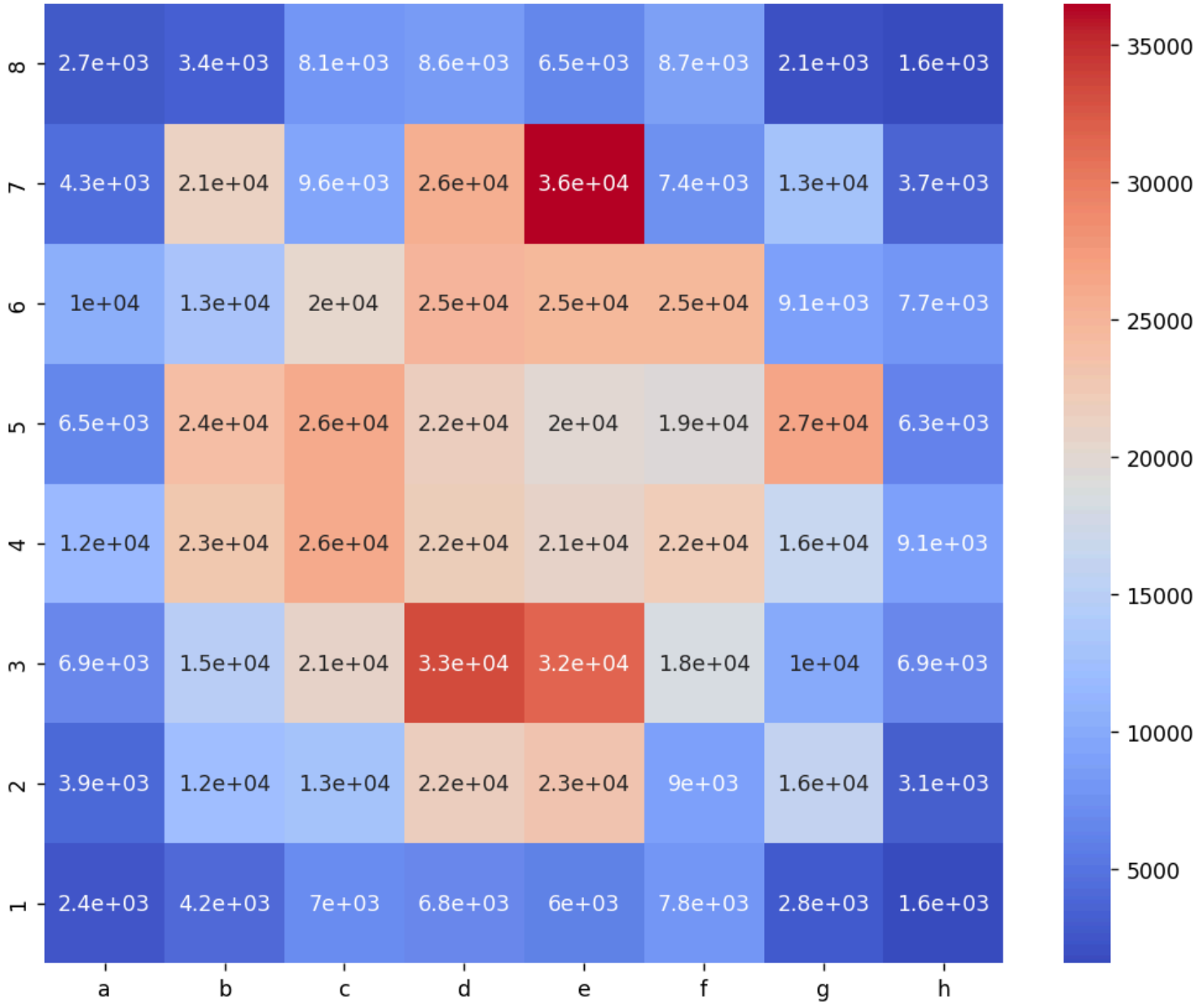
3.2.1 Overall Data :



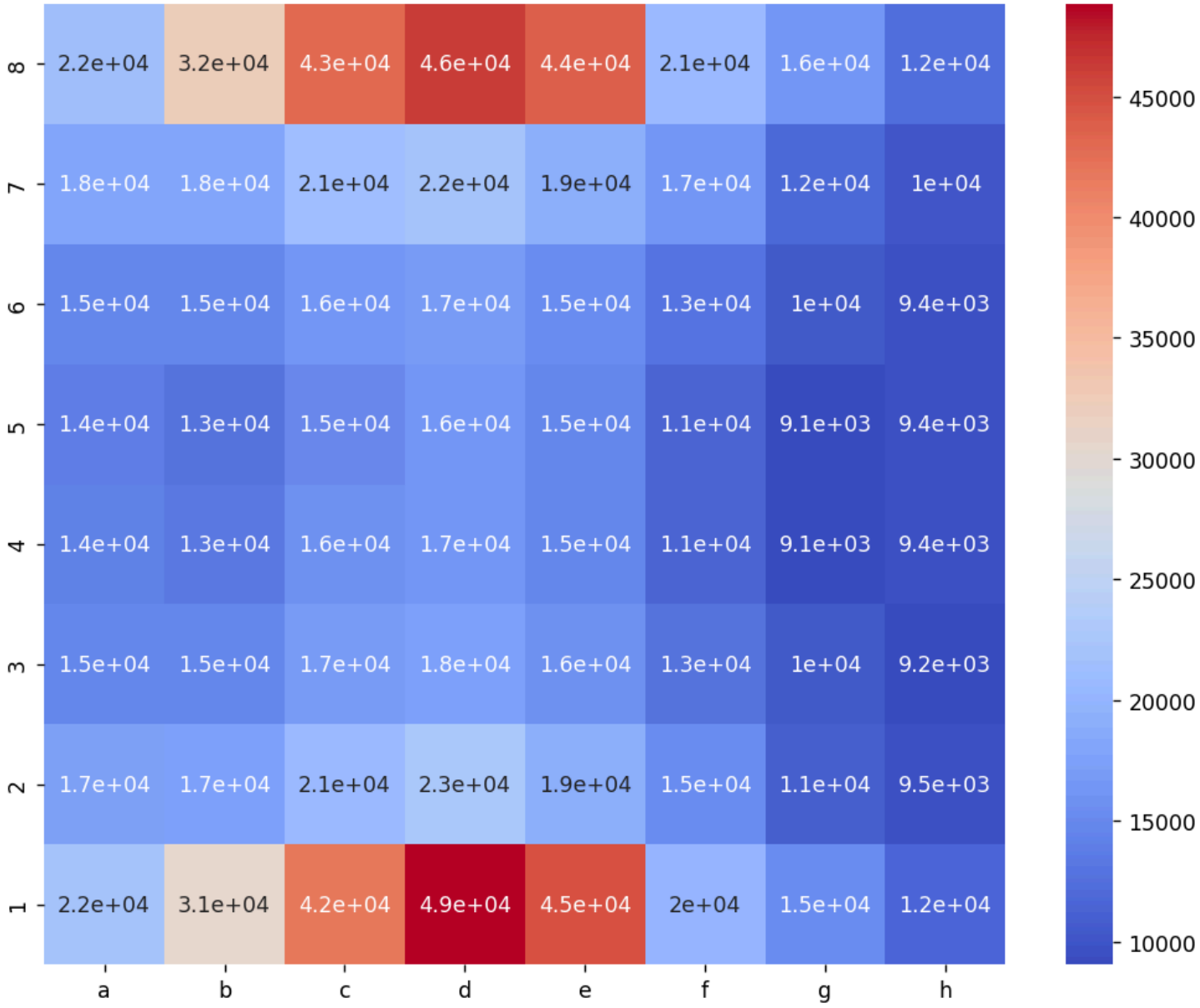
3.2.2 Knight Heatmap :



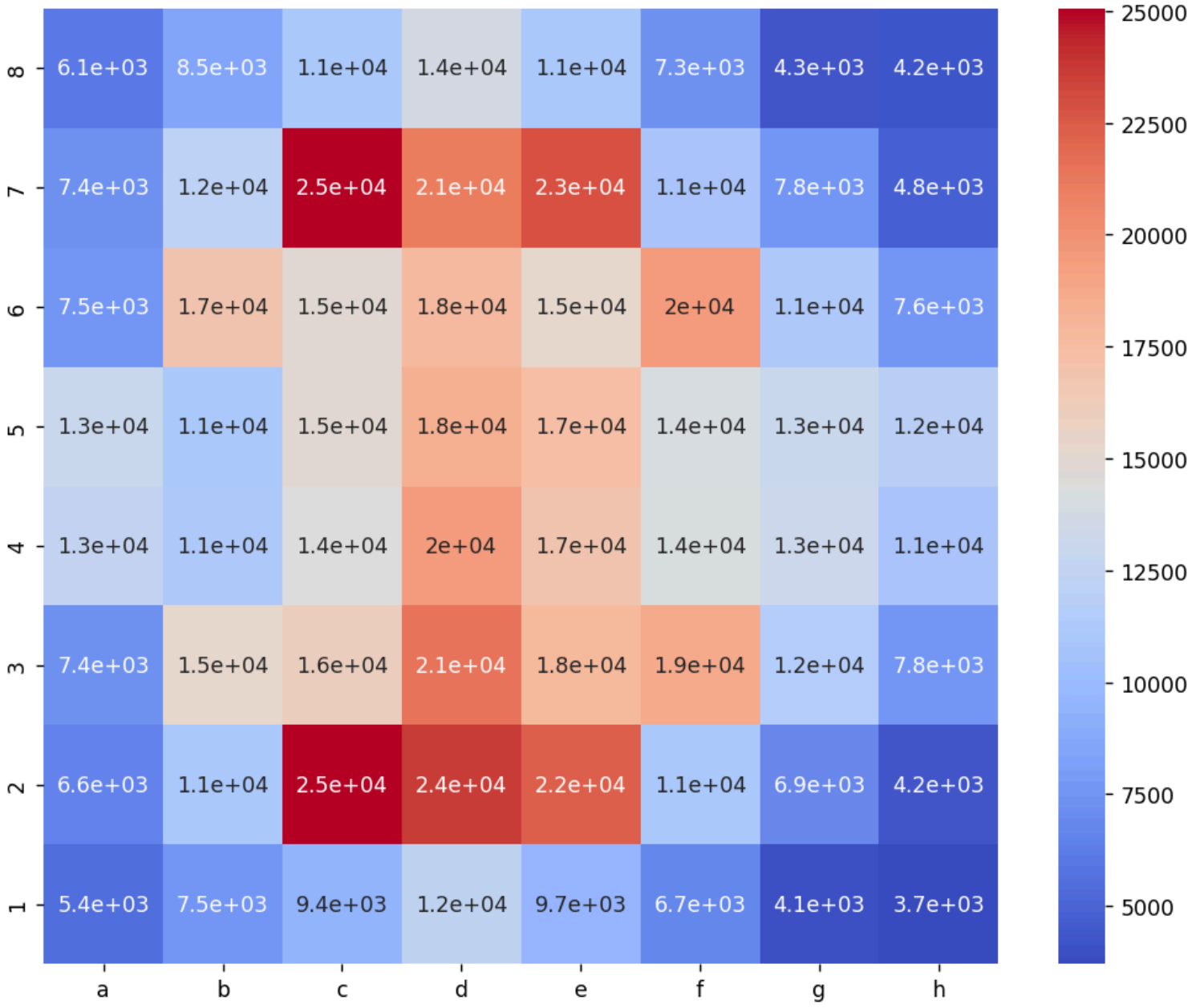
3.2.3 Bishop Heatmap :



3.2.4 Rook Heatmap :



3.2.5 Queen Heatmap :



4. Book Moves:

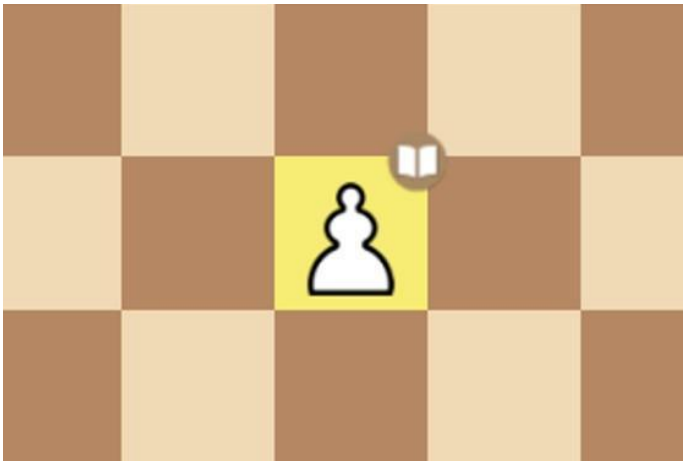
- The opening is the initial stage of a chess game. It usually consists of established theory. The other phases are the middlegame and the endgame.
- Opening moves that are considered standard are referred to as "book moves", or simply "book". When a game begins to deviate from known opening theory, the players are said to be "out of book
- The "Book" class serves as a repository for storing and accessing predefined moves played by either the player or the AI during a chess game. This class facilitates the implementation of a predetermined set of moves, allowing the engine to select moves from the book without engaging in extensive computational analysis.

4.1 Storage of Moves:

- The "Book" class contains an array data structure to store most common moves made by players in the opening sequences. These moves are organized and indexed for efficient retrieval during opening of the gameplay by the A.I engine.
- Utilizing moves from the book reduces the computational overhead of the engine, particularly during the opening phase of the game. Instead of conducting extensive search algorithms, the engine can quickly retrieve and play moves from the book, conserving computational resources and improving performance.

4.2 Dynamic Updating:

- The "Book" class support dynamic updating, allowing new moves to be added to the book . This ensures that the book remains relevant and adaptable to evolving gameplay strategies and variations.
- The "Book" class is integrated into the AI engine's decision-making process, providing an additional strategy for move selection. When the engine encounters a board position represented in the book, it evaluates the available moves and selects one from the predefined options stored in the book.



5.Improving the Chess Application: Bug Fixes and Additional Features :

- The chess application has significant potential for improvement. Addressing existing bugs and adding new features can enhance user experience and functionality.

5.1 Bug Fixes:

- **Checkmate Bug Fix:**
  - Address the checkmate bug, ensuring that the game correctly identifies and handles checkmate situations.
  - Implement thorough testing to verify the fix across various game scenarios.

- **Additional Features:**
  - Add an undo move feature, allowing players to retract their last move if needed.
  - Introduce a board flip option, enabling players to view the board from their opponent's perspective.
  - Implement functionality to store game moves for review and analysis after the game.

5.2 Engine and Book:

- Enhancing the chess engine's evaluation values and heatmaps can significantly improve its performance and strategic decision-making capabilities.

- **Evaluation Values Adjustment:**
  - Modify the evaluation values assigned to squares on the chessboard.
  - Experiment with different evaluation criteria to observe how the engine's decision-making process changes.

- **Heatmap Refinement:**
  - Update the heatmap visualization to reflect the adjusted evaluation values.
  - Visualize the impact of the modified evaluation values on piece positioning and strategic considerations.

- **Evaluation Value Variation:**
  - Introduce variations in evaluation values to analyze the engine's performance across different strategic scenarios.
  - Assess how changes in evaluation values affect the engine's ability to make optimal moves and respond to opponent actions.