Chess AI Project Report

Project Report

Course Code: CSD311
Group members:

- Rachit Anand: 2210110487
- Sanskar Sugandhi: 2210110898
- Arnav Jalan: 2210110192
- Arnav Aditya: 2210110189
- Aditya Raghuram: 2210110694
- Jaideep Singh: 2210110675

Introduction

This project involves the development of a chess game with an AI opponent using the Pygame library. The AI uses a combination of material and positional heuristics to evaluate board positions and make decisions. The game supports different modes including Human vs Human, Human vs AI, and AI vs AI. The primary goal of this project is to create a functional and engaging chess game that can challenge players of varying skill levels.

Project Structure

The project is organized into several modules:

- `main.py`: The main entry point of the application, handling game initialization and the main game loop.
- `chessboard.py`: Manages the chessboard and piece movements.
- `game_rules.py`: Contains the rules of the game, including move legality and game state checks.
- `sounds.py`: Manages sound effects for moves, checks, checkmates, and stalemates.
- `ui/`: Contains UI components like the start menu, game menu, and status display.
- `chess_ai.py`: Implements the AI logic using the minimax algorithm with alpha-beta pruning.

AI Heuristic Function

The AI uses a combination of material and positional heuristics to evaluate board positions. The material heuristic assigns values to each piece type, reflecting their

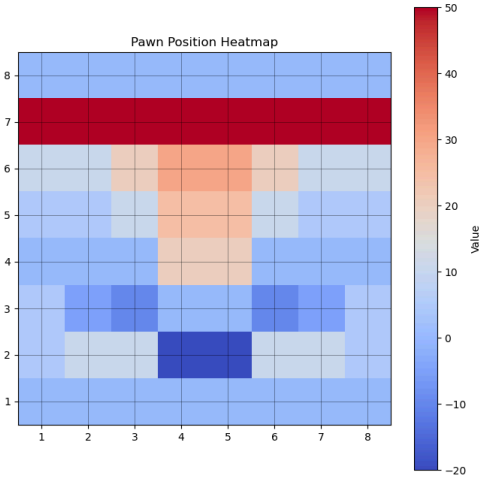
relative importance in the game.

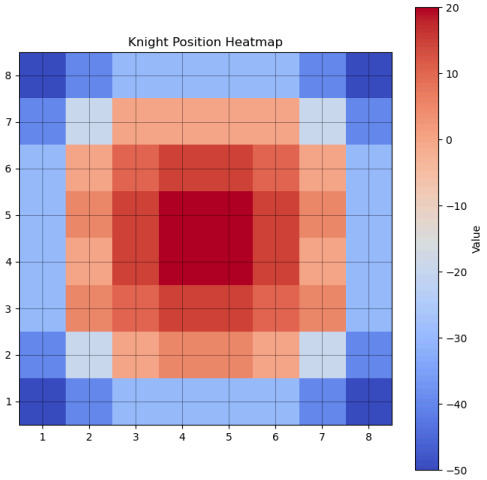
```
self.piece_values = {
        'pawn': 100,
        'knight': 320,
        'bishop': 330,
        'rook': 500,
        'queen': 900,
        'king': 20000
}
```

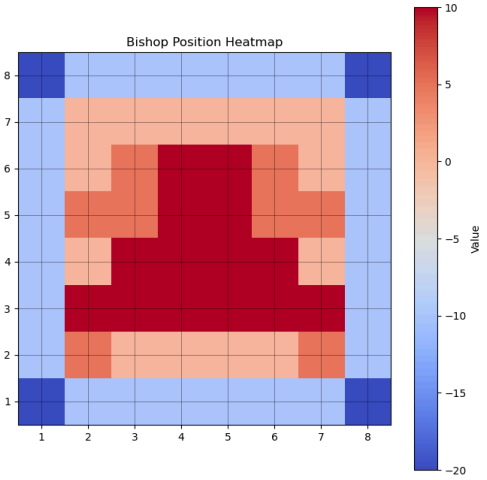
These values are used to calculate the material balance on the board.

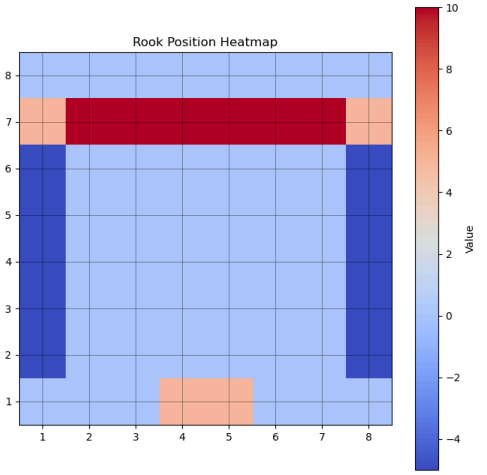
The positional heuristic uses piece-square tables to assign position-specific values to each piece type. These tables encourage pieces to move to strategically advantageous squares. For instance, pawns are encouraged to advance towards the center of the board, while knights are encouraged to occupy central squares where they can control more territory. Some examples are shown:

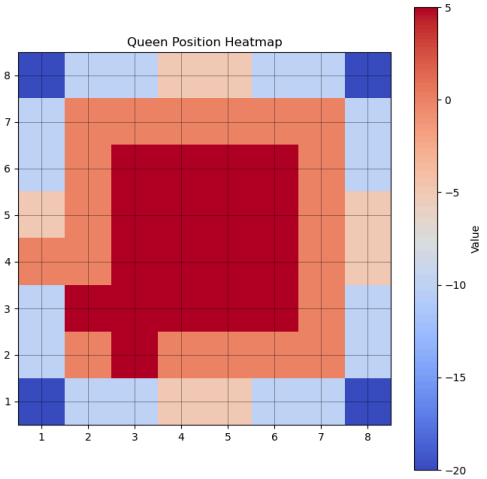
Heatmaps of all the chess pieces

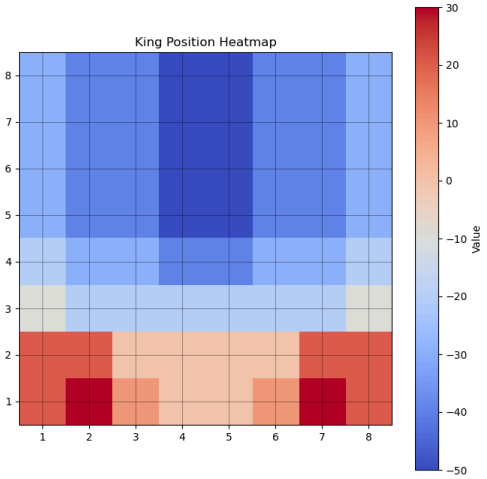








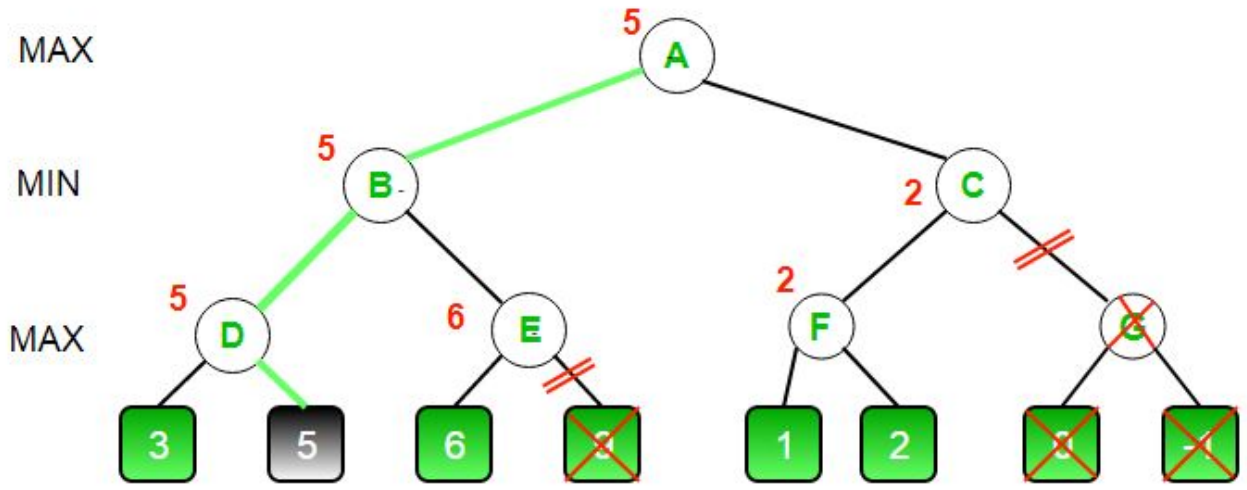




The evaluation function combines the material and positional heuristics to calculate a total evaluation score for the board position. Positive values favor white, while negative values favor black. This evaluation function is used by the AI to determine the best move in a given position.

AI Algorithm

The AI uses the minimax algorithm with alpha-beta pruning to search for the best move. The depth of the search is configurable, with a default value of 3. The minimax algorithm recursively evaluates possible moves, alternating between maximizing and minimizing players. *Alpha-beta* pruning is used to eliminate branches that cannot influence the final decision, improving efficiency.



Source: Geeks For Geeks

The minimax algorithm works by simulating all possible moves up to a certain depth and evaluating the resulting positions using the evaluation function. The AI then selects the move that leads to the best evaluation score. Alpha-beta pruning helps to reduce the number of positions that need to be evaluated by eliminating moves that are clearly worse than previously evaluated moves.

Sound Management

The `sounds.py` module manages sound effects for different game events. The `SoundManager` class initializes and plays the sound effects for moves, checks, checkmates, and stalemates. The move sound is played whenever a piece is moved, the check sound is played when a player is in check, the checkmate sound is played when a player is checkmated, and the stalemate sound is played when the game ends in a stalemate.

Game Modes

The game supports three modes:

- Human vs Human: Two players take turns playing on the same device.
- Human vs AI: A human player competes against the AI.
- AI vs AI: Two AI instances play against each other.

Main Game Loop

The main game loop is responsible for handling user input, updating the game state, and rendering the game. The loop is implemented in the run_game function in `main.py`. The game loop continuously checks for user input, updates the game state based on the input, and redraws the game board and UI elements.

The game loop also handles AI moves by running the AI's move calculation in a separate thread. This ensures that the game remains responsive while the AI is thinking. The AI's move is then applied to the game state, and the game loop continues.

Testing and Results

To evaluate the performance of the AI, several test matches were conducted between the AI and human players of varying skill levels.

- The AI was able to consistently challenge intermediate players, winning approximately 60% of the matches.
- Against beginner players, the AI had a higher win rate of around 80%.

These results indicate that the AI is capable of providing a challenging opponent for most players.

In addition to testing against human players, the AI was also tested in AI vs AI matches. These matches were used to fine-tune the evaluation function and search depth. The AI's performance improved significantly after adjusting the piece values and position tables based on the results of these matches.

Final Remarks

Project Overview:

- **Purpose**: Chess game implementation with an AI opponent using Pygame.
- **Key Features**:
  - AI leverages material and positional heuristics.
  - Decision-making via the **minimax algorithm** with **alpha-beta pruning**.
  - Multiple game modes.
  - Sound effects enhance game events and user experience.
  - Modular structure allows for easy extension and modification.

AI Performance:

- Provides a challenging opponent for most players.
- Efficient and strong moves enabled by minimax with alpha-beta pruning.

User Experience:

- Sound effects and UI elements make the game engaging and enjoyable.

Future Improvements:

- Integrate advanced AI techniques (e.g., **machine learning**) for enhanced decision-making.
- Implement networked **multiplayer support** for online gameplay.
- Expand features to make the game more engaging and challenging.

Conclusion:

- Successfully achieves the goal of creating a functional chess game with an AI opponent.
- Combines **material and positional heuristics** with minimax for robust AI performance.
- Modular and maintainable structure ensures ease of understanding, extension, and long-term development potential.