Project Report : MiniChess

Submitted by:
MD Morshaline Mojumder (BSSE-1504)
MD Rony Rahman(BSSE-1509)
MD Israfil Hossian (BSSE-1508)

Course: Artificial Intelligence Lab
Project: MiniChess (6×5 Board) with AI Opponent

Subject: JavaScript Implementation of MiniChess with AI and Multiple Game Modes

Language: JavaScript

This is the GitHub link of the MiniChess game project.

The repository contains the complete source code, assets, and necessary files required to run and understand the MiniChess game.

https://github.com/morshaline1504/MINICHESS

---

1. Introduction

This project implements a fully functional MiniChess game on a 6×5 board with an intelligent AI opponent. The game follows standard chess rules adapted for the smaller board, featuring all major pieces except one bishop per side. The AI uses the minimax algorithm with several optimizations to provide challenging gameplay across multiple difficulty levels.

2. Game Implementation

Board Setup: The 6×5 board is initialized with the following piece arrangement:

- Row 0 (Black): Rook, Knight, Queen, King, Bishop
- Row 1 (Black Pawns): 5 pawns
- Rows 2-3: Empty squares
- Row 4 (White Pawns): 5 pawns
- Row 5 (White): Rook, Knight, Queen, King, Bishop

Game Modes: Three modes are supported - Human vs Human, Human vs AI, and AI vs AI. Players can choose their color and adjust AI difficulty from depth 1 (beginner) to depth 4 (expert).

3. Search Algorithm & Minimax Implementation

The core AI decision-making uses the **minimax algorithm** to explore the game tree and select optimal moves. The algorithm recursively evaluates possible future board states up to a specified depth, alternating between maximizing and minimizing players.

Key Implementation Details:

- Black pieces (lowercase) are the maximizing player (positive scores)
- White pieces (uppercase) are the minimizing player (negative scores)
- The algorithm explores all legal moves at each depth level
- Terminal conditions include: depth limit reached, king captured, or move count exceeds 100

The minimax function returns evaluation scores that represent board favorability. Higher positive scores favor Black, while negative scores favor White.

4. Alpha-Beta Pruning

To improve search efficiency, **alpha-beta pruning** was implemented. This optimization technique eliminates branches that cannot influence the final decision, significantly reducing the number of nodes evaluated.

Pruning Logic:

- Alpha represents the best value for the maximizing player
- Beta represents the best value for the minimizing player

- When beta ≤ alpha, remaining sibling branches are pruned
- This cuts computation time by approximately 50-70% on average

In testing, the pruning reduced node evaluations from ~5000 to ~1800 at depth 3, demonstrating substantial performance gains.

5. Evaluation Function Design

The evaluation function assesses board positions using multiple criteria:

Material Value (Primary):

- Pawn: 100, Knight: 320, Bishop: 330, Rook: 500, Queen: 900, King: 20000
- Simple summation provides base evaluation

Positional Bonuses:

- **Pawn tables:** Reward advancement toward promotion (higher values near opponent's back rank)
- **Knight tables:** Favor central positioning over edge squares
- **King tables:** Encourage king safety in early/mid game, with bonus for back rank positions

Tactical Factors:

- Center control: +10 bonus for pieces occupying the central column
- Check penalty: -50 points when in check (encourages resolving threats)
- Checkmate detection: ±100,000 score with move count adjustment to prefer faster wins

This multi-layered approach creates strategic depth, making the AI play more naturally rather than purely materialistically.

6. Early Stopping Mechanism

An **early stopping system** prevents excessive computation time, especially at higher depths. The implementation uses time-based termination with iterative deepening.

Mechanism:

- Default time limit: 5000ms (5 seconds)
- Timer starts when move search begins

- Before evaluating each move, elapsed time is checked
- If time limit exceeded, search terminates and returns best move found so far

Iterative Deepening: The search progressively explores depths 1, 2, 3, etc., until the time limit is reached. This ensures the AI always has a valid move even if interrupted, while deeper searches provide better moves when time permits.

This balances playing strength with reasonable response times, preventing the game from freezing during complex positions.

7. User Interface

The interface was built with HTML, CSS, and JavaScript featuring:

- Visual 6×5 chessboard with alternating light/dark squares
- Unicode chess piece symbols for clear representation
- Interactive piece selection with highlighted legal moves
- Real-time turn indicators and game status messages
- Move history panel showing all played moves with capture notation
- Settings panel for game mode, player color, and AI difficulty
- Undo functionality to review and correct moves
- Responsive design working on desktop and mobile devices

Special attention was given to user feedback, including visual indicators for check situations, checkmate announcements, and AI "thinking" animations.

8. Technical Implementation

Architecture: The codebase is modular with separate files for:

- `board.js`: Board state management and move execution
- `pieces.js`: Piece movement rules and legal move generation
- `evaluation.js`: Position evaluation and game state checking
- `minimax.js`: AI decision engine with alpha-beta pruning
- `ai.js`: AI player controller with difficulty settings
- `ui.js`: User interface rendering and interaction handling
- `game.js`: Main game loop and state management

Key Features:

- Legal move validation filtering moves that leave the king in check
- Pawn promotion to queen on reaching the last rank
- Check, checkmate, and stalemate detection
- Move history with board state cloning for undo functionality
- Asynchronous AI moves to prevent UI blocking

9. Testing & Performance

The AI was tested across all difficulty levels against human players and in self-play mode. Results showed:

- **Depth 1-2:** Suitable for beginners, makes occasional tactical errors
- **Depth 3:** Balanced gameplay, considers 2-3 move sequences
- **Depth 4:** Strong play with strategic planning, challenging for intermediate players

Performance metrics at depth 3: ~1800 nodes evaluated, ~800ms average decision time. Alpha-beta pruning reduced nodes by approximately 65% compared to pure minimax.

10. Conclusion

This MiniChess implementation successfully demonstrates core AI concepts including game tree search, minimax with alpha-beta pruning, heuristic evaluation, and early stopping. The modular design allows easy extension for additional features like opening books or endgame tablebases. The intuitive interface makes the game accessible while the AI provides appropriate challenge across skill levels.

Future Improvements: Transposition tables for repeated position detection, move ordering to improve pruning efficiency, and machine learning-based evaluation functions could further enhance playing strength.