RustChessAi: Final Project Report

Thomas Schmidt

January 3, 2026

Project Overview

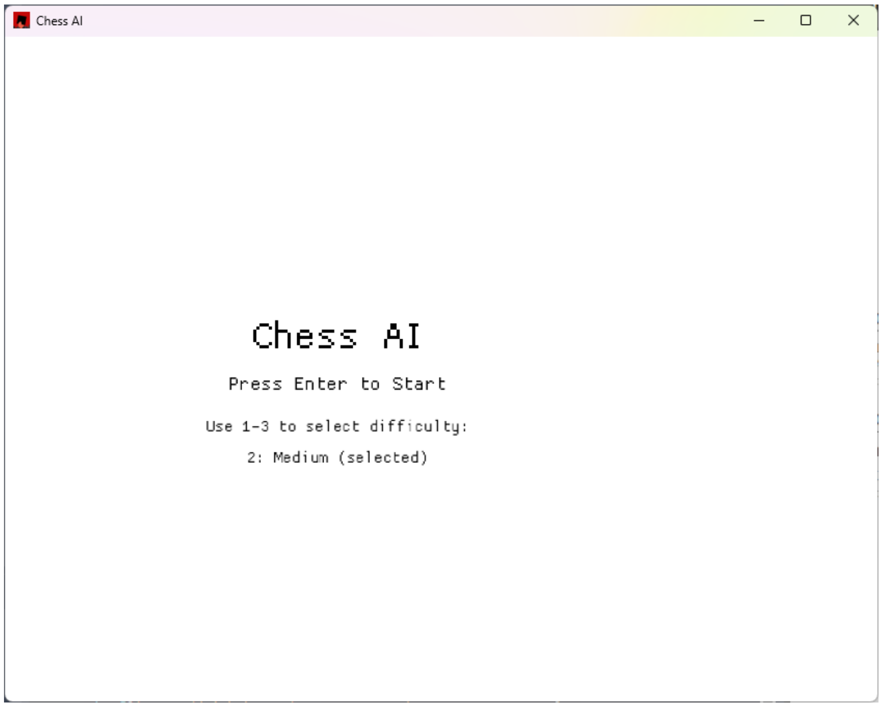
RustChessAi is a fully playable chess game written in Rust, featuring an AI opponent and an interactive graphical user interface (GUI) built with the `Macroquad` library. This final phase focused on polishing the user interface, improving code structure, and preparing the project for portfolio presentation on GitHub.

User Interface (UI) Development

The game now features a complete and user-friendly GUI:

- A functional main menu and pause menu for starting, pausing, restarting, and exiting games.
- Drag-and-scroll support for the moves history panel, with a draggable scrollbar.
- Real-time visual feedback for moves, captures, promotions, and checkmate/stalemate conditions.
- Clean separation between gameplay controls (pause button) and the moves history panel.

Screenshots illustrating the final UI:

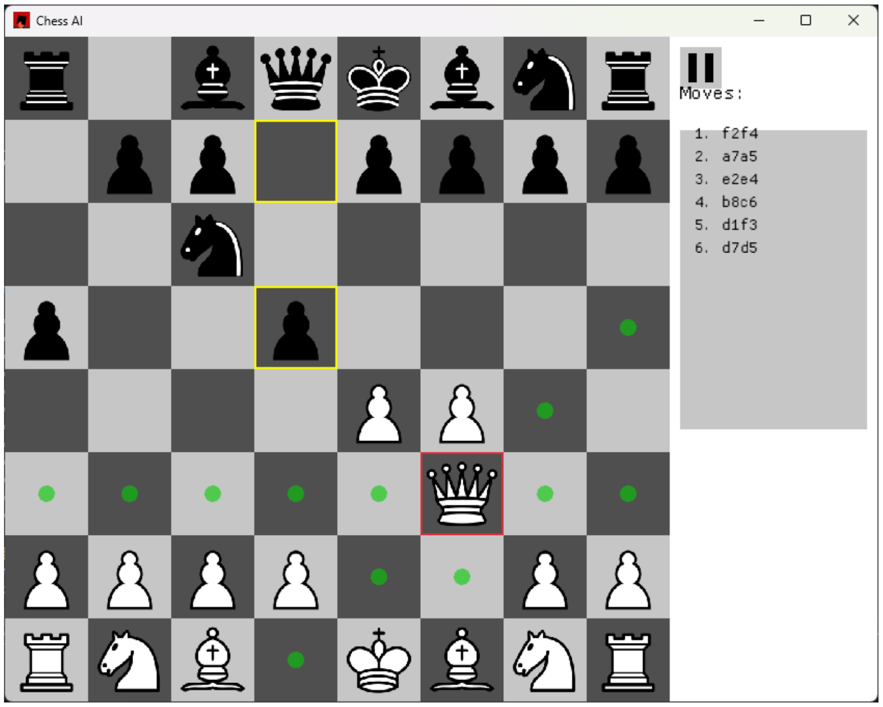


Chess AI

Press Enter to Start

Use 1-3 to select difficulty:

2: Medium (selected)



1. f2f4
2. a7a5
3. e2e4
4. b8c6
5. d1f3
6. d7d5

Code Cleanup and Polish

During this phase, significant improvements were made to the codebase:

- **Removed** the older static evaluation function and implemented a stand-pat-based quiescence search.
- **Improved** AI search efficiency with a complete rewrite of the Negamax with alpha-beta pruning.
- **Refactored** UI drawing routines for modularity and clarity.
- **Fixed** UI bugs related to scrolling, button placement, and artifacting.
- **Enhanced** maintainability through consistent formatting, descriptive comments, and clear structure.
- **Prepared** the GitHub repository for public access with a professional README and commit history.

Challenges Encountered

While polishing the project, I encountered challenges that required extensive debugging and research:

- **Moves Panel Scrolling:** Getting the mouse scroll wheel and draggable scrollbar to behave naturally was unexpectedly difficult. I had to carefully balance scrollbar bounds, clamping, and mouse dragging math. It still requires small refinements to feel completely smooth.
- **Quiescence Search Stability:** Implementing a quiescence search that avoids infinite loops or evaluation crashes at leaf nodes was challenging. Careful handling of checkmate/stalemate cases was necessary.

Despite these issues, the core functionality is complete and the game is fully playable.

Final AI Core Improvements

Updated Negamax with Alpha-Beta Pruning and Quiescence Search:

Listing 1: Negamax with Quiescence Search

```
fn negamax_ab(board: &Board, depth: i32, mut alpha: i32, beta: i32, color:
    ↪  i32) -> i32 {
    if board.status() != BoardStatus::Ongoing {
        return match board.status() {
            BoardStatus::Checkmate => -color * 1_000_000,
            BoardStatus::Stalemate => 0,
            _ => 0,
        };
    }
    if depth == 0 {
        return color * quiescence_search(board, alpha, beta, color);
    }
    let mut best_score = i32::MIN;
    for mv in MoveGen::new_legal(board) {
        let next = board.make_move_new(mv);
        let score = -negamax_ab(&next, depth - 1, -beta, -alpha, -color);
        best_score = best_score.max(score);
        alpha = alpha.max(score);
        if alpha >= beta { break; }
    }
    best_score
}

fn quiescence_search(board: &Board, mut alpha: i32, beta: i32, color: i32)
    ↪  -> i32 {
    if board.status() != BoardStatus::Ongoing {
        return match board.status() {
            BoardStatus::Checkmate => -color * 1_000_000,
            BoardStatus::Stalemate => 0,
            _ => 0,
        };
    }
    let stand_pat = stand_pat(board, color);
    if stand_pat >= beta { return beta; }
    if alpha < stand_pat { alpha = stand_pat; }

    for mv in MoveGen::new_legal(board).filter(|m| board.piece_on(m.
        ↪  get_dest()).is_some()) {
        let next = board.make_move_new(mv);
        let score = -quiescence_search(&next, -beta, -alpha, -color);
        if score >= beta { return beta; }
        alpha = alpha.max(score);
    }
    alpha
}
```

GitHub Repository Preparation

The project is now publicly available on GitHub, ready for portfolio presentation:

- **Repository:** `github.com/TWi5td/RustChessAi`
- **Features:**
  - Full source code with clean modular design.
  - Detailed README with build instructions and gameplay description.
  - Public commit history documenting key project phases.

Conclusion

RustChessAi offers a complete chess experience with a functional GUI and a strong AI opponent. While some user interface polish (especially in scrolling behavior) could still be improved, the project successfully meets all core objectives and is ready for presentation and portfolio use.