Project Report

Project Title: Chess960 (Fischer Random Chess) with AI Opponent
Submitted By: Mustafa Ahmed (22K-5056), Abdul Aziz (22K-4933), and Ramzan Asif (22K-5096)
Course: AI
Instructor: Ms. Alishba Subhani
Submission Date: [May 11, 2025]

1. Executive Summary

- **Project Overview**:
  This project implements Chess960, a chess variant with 960 possible starting positions, featuring a graphical user interface (GUI) built with Pygame, an AI opponent using the Negamax algorithm with Alpha-Beta pruning, and a data analysis module to evaluate AI performance. The main objectives were to adapt standard chess rules for Chess960's random starting positions, develop a competitive AI, and analyze game outcomes to assess AI effectiveness. Modifications include flexible castling rules, randomized back-rank piece placement, and a customizable board theme system. The project showcases skills in Python programming, AI algorithm design, GUI development, and data analysis using Pandas.

2. Introduction

- **Background**:
  Chess960, also known as Fischer Random Chess, is a variant of chess invented by Bobby Fischer to reduce reliance on memorized openings. In standard chess, pieces have fixed starting positions, but Chess960 randomizes the back-rank pieces (bishops on opposite-colored squares, king between rooks), creating 960 possible setups. This project was selected to explore AI strategies in a dynamic chess environment, where traditional opening databases are less effective. New elements include a Pygame-based GUI with sound effects, a Tkinter menu for game settings, and a data analysis script to track AI performance metrics like win rate and decision time.

- **Objectives of the Project**:

  - Develop a fully functional Chess960 game adhering to official rules, including random starting positions and flexible castling.
  - Implement an AI opponent using Negamax with Alpha-Beta pruning, optimized for Chess960 with custom evaluation heuristics.
  - Create an intuitive GUI with move animations, sound effects, and move history tracking.
  - Analyze game data to compute AI win rate, average decision time, and outcome distribution.
  - Test the AI against human players and save game data for performance evaluation.

3. Game Description

- **Original Game Rules**:
  In standard chess, two players (White and Black) move pieces on an 8x8 board with fixed starting positions (rooks on a1/h1, knights on b1/g1, etc.). Pieces have specific movement

rules: pawns move forward (capturing diagonally), rooks move horizontally/vertically, bishops diagonally, knights in an L-shape, queens in any direction, and kings one square in any direction. The objective is to checkmate the opponent's king. Chess960 retains these rules but randomizes the back-rank piece placement.

- **Innovations and Modifications**:
  - **Random Starting Positions**: The back-rank pieces (rook, knight, bishop, queen, king) are randomized, ensuring bishops are on opposite-colored squares and the king is between rooks, yielding 960 unique setups.
  - **GUI Features**: A Pygame-based interface with five board themes (blue, black-and-white, green, wood, purple), move animations, sound effects (move, capture, check, game-end), and a sidebar for move history, undo/redo, and quitting.
  - **Game Data Tracking**: Saves game outcomes, move counts, AI decision times, and starting positions to `game_data.csv`.
  - **Analysis Script**: Processes `game_data.csv` to generate a report (`chess960_ai_report.txt`) with AI win rate, average decision time, and outcome distribution.

4. AI Approach and Methodology

- **AI Techniques Used**:
  The AI employs the Negamax algorithm with Alpha-Beta pruning to select optimal moves. Negamax simplifies the Minimax algorithm by using a single evaluation function, negating scores for the opponent's perspective. Alpha-Beta pruning reduces the search space by eliminating branches that cannot improve the outcome, making the AI efficient for a depth-3 search. The AI is tailored for Chess960 with custom heuristics to account for randomized starting positions.

- **Algorithm and Heuristic Design**:
  - **Negamax Implementation**: The `negamaxAlphaBeta` function in `chess_ai.py` recursively explores game states up to a depth of 3, evaluating moves based on a board evaluation function. It uses Alpha-Beta pruning to skip unpromising branches, improving performance.
  - **Evaluation Function**: The `evaluateBoard` function assigns material values (King: 20000, Queen: 9, Rook: 5, Bishop: 3.25, Knight: 3, Pawn: 1) and applies Chess960-specific bonuses:
    - **Bishop Pair Bonus**: +0.5 for controlling two bishops, encouraging their retention.
    - **Rook Development Penalty**: -0.3 for undeveloped rooks after move 10, promoting early rook activity.
  - **Checkmate and Stalemate**: Returns +20001 for checkmate (favoring the AI) or 0 for stalemate.
  - **Move Randomization**: Shuffles valid moves to make AI behavior less predictable.
  - **Decision Time Tracking**: Records per-move decision times for analysis, stored in `ai_decision_times`.

- **AI Performance Evaluation**:
  Performance is evaluated by analyzing `game_data.csv` using `analyze_game_data.py`. Metrics include:
  - **Win Rate**: Percentage of completed games won by the AI (Computer vs. Human).

- **Average Decision Time**: Mean time (in seconds) for the AI to select a move, calculated from `ai_decision_times`.
- **Outcome Distribution**: Frequency and percentage of checkmate and stalemate outcomes.
  The analysis script outputs results to the console and `chess960_ai_report.txt`, enabling assessment of AI competitiveness and efficiency.

5. Game Mechanics and Rules

- **Modified Game Rules**:

  - **Starting Position**: Randomized back-rank placement (e.g., RNBQKBBNR), ensuring bishops are on opposite-colored squares and king between rooks.
  - **Game End**: Checkmate or stalemate (including 50-move rule draw after 100 half-moves without capture or pawn move).

- **Turn-based Mechanics**:

  - Players alternate turns, with White moving first.
  - Human players click a piece and a valid destination (highlighted in green for moves, red for captures).
  - The AI selects moves automatically when it's the Computer's turn, with a 200ms delay for visual clarity.
  - Undo/redo is available via sidebar buttons or arrow keys (left for undo, right for redo).
  - The sidebar displays move history in algebraic notation, with scrolling for long games.

- **Winning Conditions**:

  - **Checkmate**: The opponent's king is in check and cannot escape, ending the game with a win for the checking player.
  - **Stalemate**: No legal moves are available, and the king is not in check, resulting in a draw.
  - **50-Move Rule**: Draw after 100 half-moves without a capture or pawn move.

6. Implementation and Development

- **Development Process**:
  The project was developed iteratively using Python, following these steps:

  1. **Game Logic**: Implemented piece movements, board setup, and Chess960 rules in `chess_pieces.py`, `chess_board.py`, and `chess_engine.py`.
  2. **AI Development**: Coded the Negamax algorithm with Alpha-Beta pruning in `chess_ai.py`, tuning heuristics for Chess960.
  3. **GUI Implementation**: Built the Pygame-based GUI in `chess_main.py`, adding animations, sound effects, and a Tkinter menu (`chess_menu.py`).
  4. **Data Analysis**: Created `analyze_game_data.py` to process game data and generate performance reports.
  5. **Testing**: Validated move generation, castling, and AI behavior through human vs. AI games, debugging issues like invalid move detection.

- **Programming Languages and Tools**:

- **Programming Language**: Python 3.8+
- **Libraries**:
  - `pygame==2.6.1`: For GUI rendering, animations, and sound effects.
  - `numpy>=2.2.5`: For array-based board representation in `chess_board.py`.
  - `pandas>=2.2.3`: For CSV data analysis in `analyze_game_data.py`.
  - `tkinter`: Standard library module for the game settings menu.
- **Tools**:
  - GitHub for version control (assumed for project management).
  - Visual Studio Code (or similar IDE) for coding and debugging.
  - Standard library modules: `random`, `time`, `os`, `sys`, `csv`, `datetime`, `copy`.

- **Challenges Encountered**:
  - **AI Efficiency**: Initial Negamax searches were slow due to the large branching factor. Alpha-Beta pruning and a depth limit of 3 improved performance, balancing strength and speed.
  - **GUI Responsiveness**: Synchronizing animations with game state updates caused occasional lag. Optimized by rendering only changed elements and using a separate sidebar surface.
  - **Data Analysis**: Handling incomplete CSV data (e.g., no completed games) required robust error handling in `analyze_game_data.py`, implemented with try-except blocks and file checks.

7. Team Contributions

- **Team Members and Responsibilities**:
  - **Mustafa Ahmed**: Responsible for AI algorithm development (Negamax, Alpha-Beta Pruning).
  - **Abdul Aziz**: Handled game rule modifications and board design.
  - **Ramzan Asif**: Focused on implementing the user interface and integrating AI with gameplay.

8. Results and Discussion

- **AI Performance**:
  The AI's performance was evaluated over 12 completed Chess960 games, with results analyzed using `analyze_game_data.py` and reported in `chess960_ai_report.txt`. Key metrics include:

  - **Win Rate**: The AI achieved a 50.00% win rate, winning 6 out of 12 games against human players. This indicates a balanced performance, competitive with human opponents in the dynamic Chess960 environment.
  - **Average Decision Time**: The AI's average decision time was 0.9833 seconds per move, demonstrating efficient computation with the Negamax algorithm and Alpha-Beta pruning at a search depth of 3. This speed ensures a smooth gameplay experience without noticeable delays.
  - **Outcome Distribution**: Of the 12 games, 8 (66.67%) ended in checkmate, and 4 (33.33%) ended in stalemate. The higher frequency of checkmates suggests decisive gameplay, while stalemates reflect the AI's ability to navigate complex positions where neither player can force a win.

  The 50% win rate highlights the AI's competitiveness, particularly given Chess960's randomized starting positions, which reduce the advantage of precomputed openings. The

Negamax algorithm, enhanced by Alpha-Beta pruning, effectively evaluates game states, with custom heuristics (bishop pair bonus and rook development penalty) contributing to strategic play. The average decision time of under 1 second reflects successful optimization, making the AI responsive in real-time gameplay. However, the moderate win rate suggests room for improvement, such as increasing the search depth or refining heuristics to better handle late-game scenarios. The outcome distribution indicates robust game logic, as both checkmate and stalemate conditions are correctly detected and recorded.

9. References

- Fischer, Bobby. "Chess960 Rules." ChessVariants.org, https://www.chessvariants.com/diffsetup.dir/fischer.html.
- Norvig, Peter, and Stuart Russell. *Artificial Intelligence: A Modern Approach*, 3rd Edition, Prentice Hall, 2010. (For Negamax and Alpha-Beta pruning concepts)
- Pygame Documentation, https://www.pygame.org/docs/.
- Pandas Documentation, https://pandas.pydata.org/docs/.
- NumPy Documentation, https://numpy.org/doc/.
- Tkinter Documentation, https://docs.python.org/3/library/tkinter.html.