Chess Game Design Document

1. Project Overview

This project implements a fully functional chess game with an AI opponent. The AI uses strategic algorithms to evaluate moves and play against a human player dynamically. The game includes graphical interaction using Pygame, allowing players to move pieces on a visual chessboard. The AI is designed to make calculated moves based on a heuristic evaluation of the board state.

---

2. Features

- Graphical User Interface (GUI) using Pygame for interactive play.
- Legal chess moves validated for all pieces.
- AI opponent with:
    - Dynamic behavior, including attacking and defending.
    - Evaluation of board states to maximize advantage.
    - Avoidance of repetitive moves.
- Special chess rules implemented:
    - Castling.
    - Pawn promotion with user selection.

---

3. Implementation Steps

1. **Board Setup**:
    - Created an 8x8 board with standard chess starting positions.
2. **Piece Movement**:
    - Implemented movement logic for all chess pieces.
    - Validated moves based on chess rules.
3. **Graphical Interface**:
    - Displayed the board and pieces using Pygame.
    - Enabled user interaction for selecting and moving pieces.
4. **AI Development**:
    - Developed board evaluation function for strategic scoring.
    - Implemented dynamic AI logic to avoid repetition and play aggressively.

5. **Game Mechanics**:
    - Integrated turn-based gameplay.
    - Added win/loss detection with checkmate and stalemate conditions.

---

4. File Structure

main.py

- **Purpose**: Coordinates the game loop and user interaction.
- **Key Variables**:
    - board: Stores the current state of the chessboard.
    - is_white_turn: Tracks the current turn.
    - last_ai_move: Tracks the last move made by the AI to avoid repetitions.
- **Methods**:
    - play_game: Manages the game loop, alternating between player and AI turns.

graphics.py

- **Purpose**: Handles graphical rendering and user interaction.
- **Key Methods**:
    - initialize_screen: Sets up the game window.
    - load_images: Loads piece images for rendering.
    - draw_board: Draws the chessboard grid.
    - draw_pieces: Renders pieces on the board.
    - get_board_position: Maps mouse clicks to board coordinates.

board.py

- **Purpose**: Initializes and manages the chessboard state.
- **Key Methods**:
    - initialize_board: Sets up the starting position of pieces.
    - move_piece: Updates the board when a piece is moved.

ai.py

- **Purpose**: Implements the AI's move logic.
- **Key Methods**:

- evaluate_board: Calculates a heuristic score for a given board state.
- choose_best_move: Selects the best move for the AI based on evaluation scores.
- get_legal_moves: Generates all valid moves for a side.
- find_targets: Identifies opponent pieces that can be attacked.

pieces.py

- **Purpose**: Defines movement logic for individual chess pieces.
- **Key Methods**:
  - is_valid_pawn_move: Validates pawn movement and captures.
  - is_valid_knight_move: Validates knight movement.
  - is_valid_bishop_move: Validates bishop movement.
  - is_valid_rook_move: Validates rook movement.
  - is_valid_queen_move: Validates queen movement.
  - is_valid_king_move: Validates king movement and castling.

---

5. Example Walkthrough

1. **Game Initialization**:
   - The chessboard is displayed, and the player can interact with it via the mouse.
2. **Player's Turn**:
   - The player clicks on a piece and selects a valid square to move it.
   - The move is validated and executed on the board.
3. **AI's Turn**:
   - The AI evaluates all possible moves and selects one based on its heuristic scoring system.
   - The move is rendered on the board.
4. **Win/Loss Detection**:
   - If the king is captured or there are no valid moves left for one side, the game ends with a checkmate or stalemate.

---

6. Challenges and Solutions

Repetitive AI Moves:

- Initially, the AI repeated moves due to lack of memory.
- **Solution**: Added logic to track the AI's last move and penalize repeated states.

Lack of Aggression:

- Early versions of the AI only focused on defense.
- **Solution**: Enhanced the board evaluation function to prioritize attacking and active piece positions. (still work in progress but AI behavior is much better now)

Complex Move Rules:

- Implementing special moves like castling and pawn promotion was challenging.
- **Solution**: Added specific methods to handle these scenarios and validated them thoroughly.

---

7. Future Enhancements

- Add difficulty levels by varying AI depth and evaluation complexity.
- Include sound effects and animations for a more immersive experience.
- Optimize AI performance for faster decision-making.