Software Developer's Toolkit *ECS 198F, Winter 2025*

Final Project: Chess Game

Important: This project **must be completed** in groups of **3 to 4 members**.

To begin, one group member should fork the template repository provided on the course website. Then, they should add all other group members as collaborators to grant them access.

Section 1: Project Explanation

For this project, your group will collaborate to develop a Chess Game. The display and graphical user interface (GUI) have already been implemented using Pygame and are provided in the template code.

Your primary task is to implement the core logic and mechanics of the Chess Game. A class named `ChessLogic` has been pre-defined for this purpose and is located in the file `pychess/logic/chess_logic.py`.

There are two main components of the `ChessLogic` class that you need to understand.

1. **External Fields:**
   The `ChessLogic` class contains two essential fields, `board` and `result`, which are critical for the functionality of the game. It is imperative to follow the guidelines for these fields precisely, as the GUI and Autograder depend on their correct implementation.

   (a) **The `board` Field:**
   The `board` field is a two-dimensional list of strings representing the current state of the chessboard. This list must adhere to the following specifications:

   - The dimensions must be `8x8`, corresponding to the 8 rows and 8 columns of a standard chessboard.
   - Each value in the list must be one of the following:
     - `''` (Empty string): Represents an empty square.
     - `P, R, N, B, Q, K`: Represents white pieces (must be uppercase).
       - `P`: White Pawn
       - `R`: White Rook
       - `N`: White Knight
       - `B`: White Bishop
       - `Q`: White Queen
       - `K`: White King
     - `p, r, n, b, q, k`: Represents black pieces (must be lowercase).
       - `p`: Black Pawn
       - `r`: Black Rook
       - `n`: Black Knight
       - `b`: Black Bishop
       - `q`: Black Queen
       - `k`: Black King

   (b) **The `result` Field:**
   The `result` field is a string that represents the current game outcome. It must be set to one of the following values:

   - `''` (Empty string): Indicates that the game is still in progress.
   - `'w'`: Indicates that White has won.
   - `'b'`: Indicates that Black has won.
   - `'d'`: Indicates a draw.

2. `play_game` **Function and Constructor:**
   The second key component of the `ChessLogic` class consists of the constructor and the `play_move` function. These are crucial for setting up and managing the game state and moves.

   (a) **Constructor:**
   The constructor is invoked when the board is first set up. This is where you should include all your initialization logic, such as setting up the initial board state, the game result, and any other necessary fields.

   (b) `play_move` **Function:**
   This is the primary function that you need to implement. It is called every time a move is proposed on the board. The function receives one argument, `move`, which is a string representing the move in the following format: {`starting_pos`}{`ending_pos`}.

   For example, if the user wants to move a pawn from E2 to E4, they will first click the pawn on E2, then click the E4 square. The `move` string passed to the function will be `"e2e4"`, indicating that the pawn should be moved from E2 to E4.

   - **Parsing the Move:** You must parse the `move` string and determine what move the user is attempting to make.
   - **Handling Invalid Moves:** If the move is invalid, you should return an empty string (`""`).
   - **Updating the Board:** If the move is valid, update the `board` field with the new position of the piece.
   - **Returning Extended Chess Notation:** Additionally, compute the extended chess notation for the move and return it as a string. For example, moving the pawn from E2 to E4 would update the board and return `"e2e4"`, which is the extended chess notation for this move.

   Extended chess notation rules and edge cases in parsing moves will be explained separately.

Section 2: Edge Cases When implementing the `play_move` function, you must account for a variety of edge cases that may arise during gameplay. Below are some common edge cases that you should handle appropriately to ensure the game functions smoothly:

1. **Moving to Occupied Square with Same Color Piece:**
   A player cannot move a piece to a square already occupied by another piece of the same color. For example, if a white pawn is on `e4` and another white piece is already on `e5`, moving the pawn from `e4` to `e5` should be rejected.

2. **Illegal Move for Specific Piece Type:**
   Each piece has specific movement rules. For example, pawns can only move forward one square (except on the first move where they can move two squares), knights move in an "L" shape, and bishops move diagonally. If a piece attempts to make a move that is not legal according to its type, return an empty string. For example, a knight trying to move two squares forward would be invalid. You can view all the movement rules for each chess piece here: `https://www.chess.com/terms/chess-pieces`.

3. **Check and Checkmate Conditions:**
   You should check whether the move places the opposing king in check or checkmate. A player cannot make a move that would place their own king in check. If the move results in a check, you should handle it appropriately. For example, if the white king is on `e1` and the black queen is threatening `e1`, moving the white king to `e1` should be rejected.

4. **Pawn Promotion:**
   When a pawn reaches the opposite end of the board (the 8th rank for white and the 1st rank for black), it should be promoted. The function should handle pawn promotion and update the board accordingly. For example, if a white pawn moves from `e7` to `e8`, it should be promoted to a queen.

5. **En Passant Capture:**
   If a pawn moves two squares forward from its starting position and lands next to an opponent's pawn, the opponent may capture the pawn "en passant" as if it had only moved one square forward. This special rule should be handled if applicable.

6. **Castling:**
   Castling is a special move involving the king and a rook. For simplicity, castling is treated as one move, but all the usual conditions for castling are still enforced:

   - The king has not previously moved.
   - The rook involved has not previously moved.
   - There are no pieces between the king and the rook.
   - The king cannot pass through or land on a square that is under attack.

   If the White King is on `e1` and the White Rook is on `h1` with no pieces between them, a move like `"e1g1"` should be interpreted as castling. Similarly, if the Black King is on `e8` and the Black Rook is on `h8`, `"e8g8"` would also be interpreted as castling.

   - The return value of the `play_move` function should be `"O-O"` for King-side castling (i.e., castling on the right side of the board).
   - Similarly, for Queen-side castling (when the king moves two squares toward the rook on the left), the return value should be `"O-O-O"`.

   If the castling conditions are not met, the move should be rejected, and an empty string should be returned.

7. **Move that Would Leave King in Check:**
   A move that places or leaves the player's own king in check should be invalid. For instance, if a player attempts a move that allows their king to be captured, the move should be rejected, and an appropriate message should be returned.