Chess game

Project Overview

This project is a digital chess game that incorporates data analytics to enhance player experience and game analysis. The game will track various metrics, such as time per move, material balance, and checkmate frequency, to provide insights into player performance and strategic improvements. The implementation will use object-oriented programming (OOP) principles, ensuring modularity and scalability.

Project Review

A review of existing chess applications, such as Lichess and Chess.com, highlights areas for improvement. While these platforms provide analytics, this project aims to enhance data tracking by incorporating statistical analysis and visualization to help players understand their gameplay better.

Key improvements:

- More detailed in-game statistics
- Real-time tracking of game events
- Improved visual representation of player performance

Programming Development

chess game will simulate the traditional chess match with additional features for analysis

Game Concept

Mechanics: Turn-based play with strict rule enforcement; support for local (two players).

Objectives: Achieve checkmate while optimizing play based on real-time statistics.

Key Selling Points:

- Detailed tracking of gameplay data.
- A post-game analysis report with graphs and statistics to help players improve.

Object-Oriented Programming Implementation

The project will employ several classes to encapsulate game components. Below are five core classes:

Main

- **Role:**
  Manages the game loop and initializes Pygame.

- **Key Attributes:**
  - `screen`: The Pygame display surface.
  - `Game instance`: An instance of the `Game` class managing game state.

- **Key Methods:**
  - `mainloop()`: Runs the main event loop for the game.

Game

- **Role:**
  Acts as the central controller of the game state (handling the board, pieces, and turn management).
- **Key Attributes:**
  - `board`: The game board that stores squares and pieces.
  - `next_player`: Indicates which player's turn is next (e.g., `'white'` or `'black'`).
  - `dragger`: Manages the dragging of pieces on the board.

- **Key Methods:**
  - `show_back_ground()`: Displays the game's background.
  - `show_piece()`: Renders the chess pieces on the board.
  - `next_turn()`: Switches the turn between players.
  - `reset()`: Resets the game to its initial state.

Board

- **Role:**
  Stores the board's squares and pieces, and handles the application of moves.

- **Key Attributes:**
  - `2D array of squares`: Represents the layout of the chess board.
  - `references to pieces`: Contains the current pieces placed on the board.
- **Key Methods:**

  - `_create_board()`: Initializes the board layout.
  - `apply_move()`: Applies a given move to update the board state.
  - `is_in_check()`: Determines if a player's king is in check.
  - `move_piece()`: Moves a piece from one square to another.

Piece (Abstract Base Class)

- **Role:**
  Serves as the base class for all chess pieces (such as Pawn, Rook, Knight, etc.).
- **Key Attributes:**
  - `name`: The name/type of the piece (e.g., "pawn", "knight").
  - `color`: The color of the piece (`'white'` or `'black'`).
  - `value`: A numerical value representing the piece's relative strength.
  - `moves`: A list of possible moves available for the piece.

- **Key Methods:**
  - `get_moves()`: An abstract method to generate possible moves for the piece (must be implemented in subclasses).
  - `add_move()`: Adds a possible move to the list of moves.
  - `clear_moves()`: Clears the list of moves.

Move

- **Role:**
  Represents a single move in the game, including details of source and destination squares, and any special move information.
- **Key Attributes:**
  - `player`: The player making the move.
  - `from_square`: The starting square of the move.
  - `to_square`: The destination square of the move.
  - `move_history`: A record of previous moves in the game.
  - Additional attributes for special moves such as `captured_piece`, `promotion`, `en_passant`, `castling`, `check`, `checkmate`, and `stalemate`.

- **Key Methods:**
    - `add_move()`: Adds a move to the move history.
    - `clear_moves()`: Clears the move history.
    - `add_move_to_data()`: (Placeholder) Intended for integrating move data with a larger dataset.

Square

- **Role:**
  Represents an individual square on the chess board.
- **Key Attributes:**
    - Board dimensions and square size (e.g., `WIDTH`, `HEIGHT`, `ROWS`, `COLS`, `SQSIZE`).
    - Mappings for board coordinates (e.g., `ALPHACOLS` and `ALPHAROWS`).
- **Key Methods:**
    - `has_piece()`: Determines if the square contains a piece.
    - `isempty()`: Checks if the square is empty.
    - `has_team_piece(color)`: Checks if the square contains a piece of the given team/color.
    - `has_enemy_piece(color)`: Checks if the square contains an enemy piece.
    - `in_range()`: Static method to verify if provided coordinates are within board limits.

DragHandler *(or Dragger)*

- **Role:**
  Manages the dragging functionality of a chess piece on the board.
- **Key Attributes:**
    - `piece`: The current piece being dragged.
    - `dragging`: Boolean flag indicating if a drag is in progress.
    - `mouseX`, `mouseY`: The current mouse coordinates.
    - `initial_row`, `initial_col`: Starting position of the piece before dragging begins.

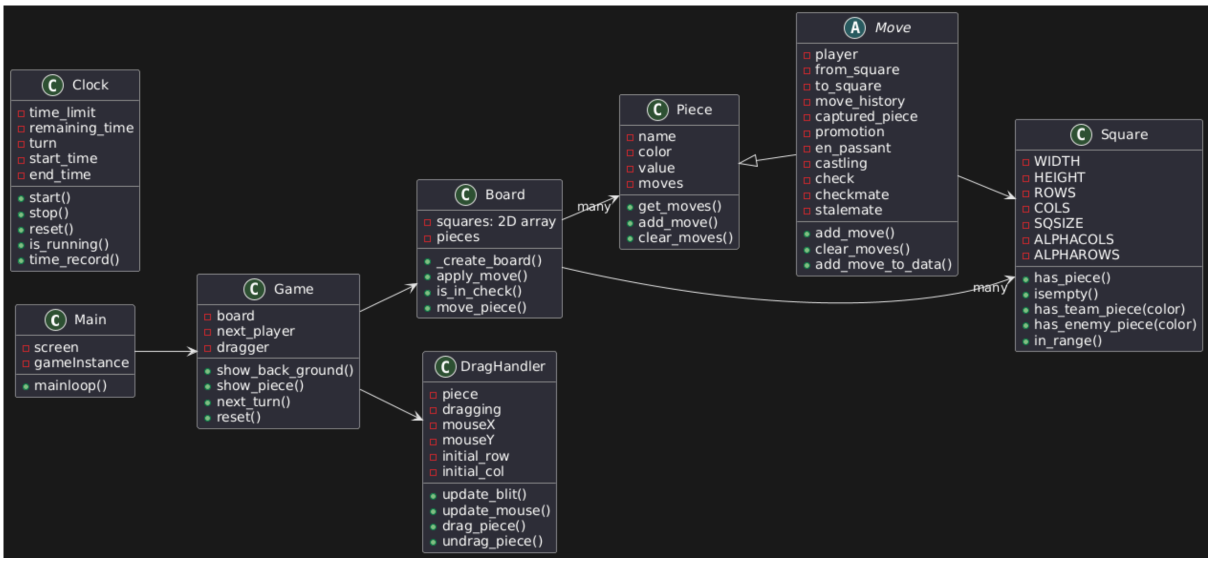
- **Key Methods:**
    - `update_blit()`: Updates the display for the dragged piece.
    - `update_mouse()`: Updates mouse coordinates during dragging.
    - `drag_piece()`: Initiates the dragging process for a given piece.
    - `undrag_piece()`: Stops the dragging process and resets relevant attributes.

Clock

- **Role:**
  Manages timing for game events, such as move timing and overall game duration.
- **Key Attributes:**
    - `time_limit`: The overall time limit set for a game or move.
    - `remaining_time`: The current remaining time.
    - `turn`: Indicates which player's time is being tracked.
    - `start_time`, `end_time`: Timestamps marking the beginning and end of a timed period.
- **Key Methods:**
    - `start()`: Begins the timing process.
    - `stop()`: Stops the timing process.
    - `reset()`: Resets the clock to its initial state.
    - `is_running()`: Checks if the clock is currently active.
    - `time_record()`: Records the elapsed time for game data analysis.

<u>Algorithms Involved</u>

- **Move Validation Algorithm**: Ensures only legal moves are allowed
- **Check and Checkmate Detection**: Identifies game-ending scenarios
- **Material Balance Calculation**: Tracks piece value differences



Statistical Data (Prop Stats)

Data Features

The following gameplay metrics will be tracked:

Feature	Purpose	Data Collection Method	Display Format
Time per move (s)	Analyze player speed and decision-making	collect from clock class	Histogram
Win/Draw/Loss Rate	Track player success rate	collect from game class	Pie chart
Material Balance	Evaluate positional advantage	collect from move class	Line Graph
Checks & Checkmates	Identify attacking strategies	collect from board (check_count method)	Summary Statistics/Table
Game Duration / Moves	Measure overall game pacing	collect from clock class	Graph

Data Recording Method

Data will be stored in CSV format for easy analysis and visualization.

Data Analysis Report

- **Time per move (s):**
  Displayed as a Histogram.
  Analysis: **The histogram will illustrate the distribution of move times, providing insights into player speed and decision-making patterns.**

- **Win/Draw/Loss Rate:**
  Displayed as a Pie Chart.
  Analysis: **The pie chart will show the proportion of wins, draws, and losses, effectively tracking the overall player success rate.**

- **Material Balance:**
  Displayed as a Line Graph.
  Analysis: **The line graph will visualize changes in material balance over time, allowing for evaluation of positional advantage throughout the game.**

- **Checks & Checkmates:**
  Presented as Summary Statistics/Table.
  ***Analysis:* A table will summarize the occurrences of checks and checkmates per game, highlighting attacking strategies and defensive effectiveness.**

- **Game Duration / Moves:**
  Displayed as a Scatter Plot.
  ***Analysis:* The scatter plot will depict the relationship between game duration and the number of moves, providing insights into overall game pacing.**

Graphs Specification

We will create three graphs using the following features. The table below summarizes the planned visualizations:

Feature	Graph Type	Purpose
Time per move (s)	Histogram	To illustrate the distribution of moved times, indicating decision speed.
Material Balance	Line Graph	To visualize changes in material advantage over time.
Game Duration / Moves	Scatter Plot	To analyze the relationship between game duration and number of moves.

Statistical Values for Checks & Checkmates:
For this feature, we will compute and present in the summary table:

- **Total Count:** The total number of check events (including checkmates) per game.
- **Average per Game:** The mean number of checks per game.
- **Maximum:** The highest number of checks recorded in any game.
- **Standard Deviation (SD):** The variation in the number of checks across games.

These values will help highlight attacking strategies and defensive effectiveness.

Project Timeline

Week	Task
1 (10 March)	Proposal submission / Project initiation
2 (17 March)	Full proposal submission/ start
3 (24 March)	Working on board/piece
4 (31 March)	Working on player
5 (7 April)	Working on clock
6 (14 April)	Submission week (Draft)

Document version
Version: *4.0*
Date: *31 March 2025*