Enhanced Chess Game for Beginners

1. Project Overview

This project is a chess game developed using PyGame that follows official chess rules. But this game will have additional features, such as move hints, highlighting potential captures, and displaying legal move paths— to make chess more fun and easier to learn.

2. Project Review

Normal chess game offer only the basic board and rules, but this project will provide:

- **Easier Learning:** Visual cue that indicate vulnerable pieces and available moves.
- **Interactive Assistance:** Real-time suggestions that help players better understand game dynamics.

3. Programming Development

3.1 Game Concept

The game simulates a standard chess match between two players (or a human against an AI). The primary objective is to checkmate the opponent's king. The game integrates traditional gameplay with extra visual aids to simplify learning and improve strategic play.

3.2 Object-Oriented Programming Implementation

The project is structured around key classes:

- **Game**

  Role: Manages the overall game flow, including board state and user input.

  Key Attributes:

  - **state:** Current status of the game (ongoing, check, checkmate).
  - **active_player:** Indicates whose turn it is.
  - **move_count**: Total number of moves made in the game.
  - **move_times**: A list tracking the time taken for each move.
  - **game_duration**: Total duration of the game.

  Key Methods: start_game(), update_game(), process_input(), check_end_conditions().

- **Board**

  Role: Handles the chessboard layout and piece placement.

  Key Attributes:

  - **grid:** A 2D array representing the board squares.
  - **pieces:** A list of all active chess pieces.
  - **captured_count:** Count of captured pieces during the game.

  Key Methods: render_board(), update_board(), get_valid_moves(position).

- **Piece (Base Class)**

  Role: Provides common attributes and methods for all chess pieces (excluding pawn-specific behavior).

  Key Attributes:

  - **type:** Type of the chess piece (e.g., knight, bishop).
  - **color:** Color of the piece.
  - **position:** Current location on the board.

  Key Methods: move(), validate_move(), capture().

- **Pawn (Piece)**

  Role: Implements pawn-specific movement rules.

  Key Attributes: Inherits attributes from Piece and may include flags (e.g., first move, en passant eligibility).

  Key Methods: possible_moves(), special_moves() (handling initial two-step move and en passant).

- **AI_Player**

  Role: Implements logic for the computer-controlled opponent.

  Key Attributes:

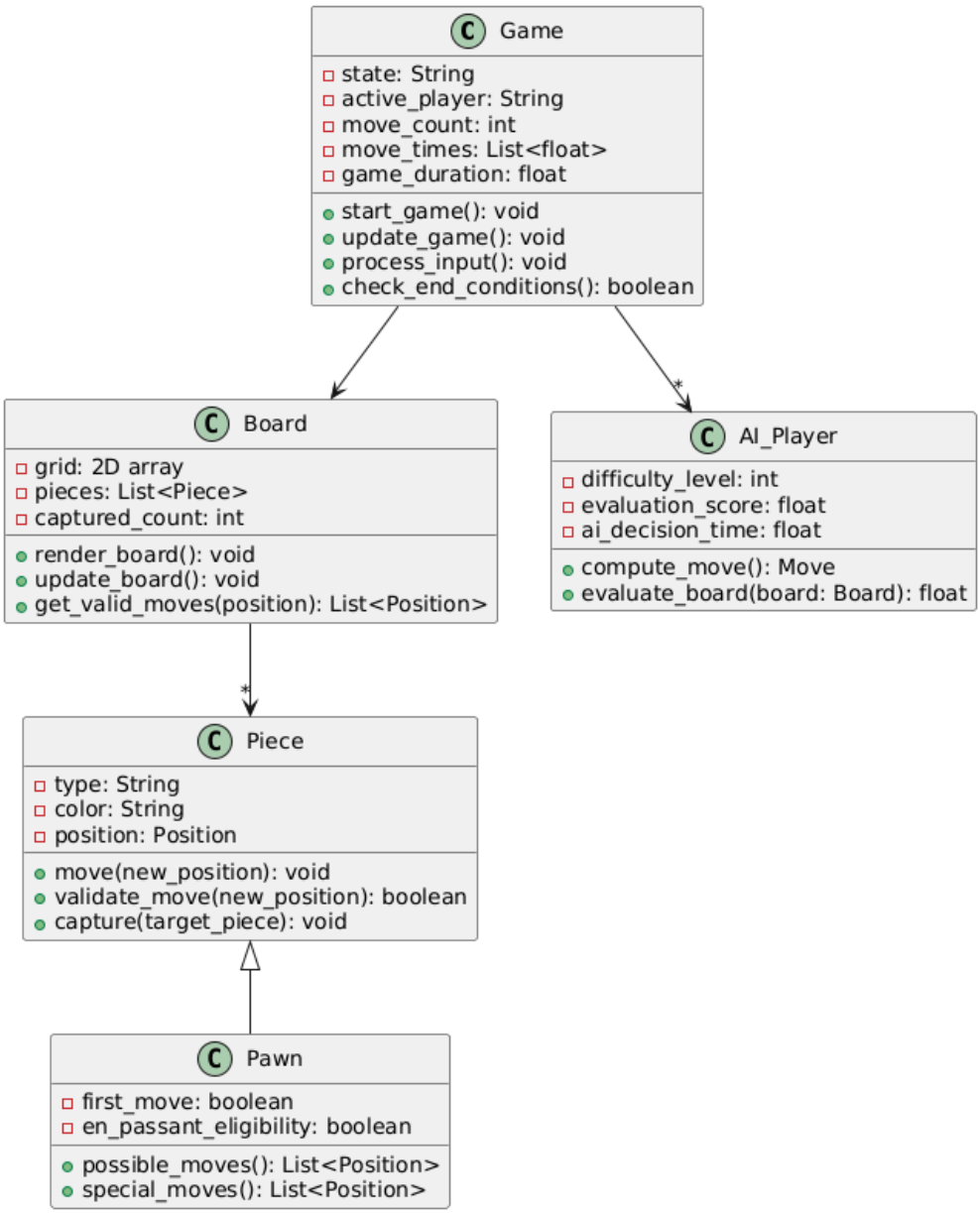
  - **difficulty_level:** Determines the AI's challenge level.
  - **evaluation_score:** for ai to compare each position(represent value of position as score) and use it to choose board positions.
  - **ai_decision_time:** Time taken by the AI to decide on a move.

  Key Methods: compute_move(), evaluate_board().

3.3 Algorithms Involved

Key algorithms include:

- **Move Validation:** Ensuring all moves follow to chess rules.
- **Pathfinding:** Determining valid move paths for pieces.
- **Minimax with Alpha-Beta Pruning:** algorithms for AI moving decision.
- **Event Handling:** Managing real-time user inputs and game state updates.



4. Statistical Data (Prop Stats)

4.1 Data Features

The game will record essential metrics, such as:

- Number of moves per game.
- Time taken for each move.
- Captured pieces count.
- Overall game duration.
- AI decision time.

4.2 Data Recording and Analysis

Data will be stored in CSV format, ensuring ease of access and manipulation for further analysis.

4.3 Data Analysis Report

The recorded data will be analyzed using basic statistical measures such as mean, median, mode, and standard deviation. The analysis will be presented via bar charts.

5. Project Timeline

Week	Task
1 (10 March)	Proposal submission / Project initiation
2 (17 March)	Full proposal submission, design UML diagram
3 (24 March)	Begin development
4 (31 March)	Implement core game mechanics
5 (7 April)	Implement an AI part and additional features
6 (14 April)	Submission week (Draft)

6. Document Version

- **Version:** 2.0
- **Date:** 31 March 2025

Feature	Why is it good to have this data? What can it be used for?	How will you obtain 50 values of this feature data?	Which variable (and which class) will you collect this from?	How will you display this feature data?
1. Number of moves	Helps measure how long or complex a match is. Can indicate if players are quick to finish or prolong games.	Each completed game produces a final move count. After 50 games, you'll have 50 values.	`move_count` from the **Game** class (increments every time a move is made).	Bar chart or line chart showing the distribution of move counts per game.
2. Time taken per move	Useful for analyzing player speed and decision-making. Identifies whether players are moving slowly or quickly.	Every time a move is made, After 50 moves,we will have 50 entries.	`move_time` stored in the **Game** or **Board** class (tracked with timestamps before and after a move).	A bar chart or histogram showing the frequency of different move durations.
3. Captured pieces count	Shows how aggressive or defensive a game is. High captures may indicate a more attacking style.	After each capture event, increment the count. By the end of 50 capturing, we will have 50 data.	`captured_count` in the **Board** class or maintained by each **Piece** when it's captured.	A simple tally or bar chart comparing how many pieces each side captures.
4. Game duration	Reflects how quickly players reach checkmate or stalemate. Useful for analyze pacing of the game.	Record the start and end time for each game. After 50 games, you'll have 50 total durations.	`game_duration` in the **Game** class (difference between start time and end time).	A table of average duration, or a bar chart showing durations of each game.
5. AI decision time	Measures the efficiency of the AI. If the AI is taking too long, you can optimize its algorithms or difficulty.	Each time the AI makes a move, record how long it took to calculate. After 50 AI moves, we'll have 50 data points.	`ai_decision_time` in the **AI_Player** class (tracked from start of evaluation until move selection).	A line chart or bar chart illustrating the variation in AI calculation time.

7. Statistical Data Revision

7.1 Table Presentation

i will focus on **“Time Taken per Move”** as the feature to present in a **table**.

- **Feature (A):** Time Taken per Move
- **Statistical Values (B):**
    - **Minimum (min)** – The shortest time taken for a single move.
    - **Maximum (max)** – The longest time taken for a single move.
    - **Average (mean)** – The average time taken per move across all moves.
    - **Standard Deviation (SD)** – How much variation there is in move times.

7.2 Graph Presentation

i plan to create **three (3) graphs** using the following features:

1. **Number of Moves**
    - **Graph Objective:** Compare the total moves made in different games.
    - **Proposed Graph Type:** Bar chart (each bar represents one completed game).
2. **Captured Pieces Count**
    - **Graph Objective:** Show how many pieces are captured per game to illustrate aggressiveness of play.
    - **Proposed Graph Type:** Stacked bar chart (split by piece color or piece type).
3. **AI Decision Time**
    - **Graph Objective:** Show how the AI’s move-calculation time changes throughout a game.
    - **Proposed Graph Type:** Line graph (time on the y-axis vs. move number on the x-axis).

should look something like this

Feature Name	Graph Objective	Graph Type	X-axis	Y-axis
Number of Moves	Compare total moves in different games	Bar chart	Game Index	Total Moves
Captured Pieces Count	Visualize how many pieces each side captures to show aggressiveness	Stacked bar chart	Game Index	Count of Captured Pieces
AI Decision Time	Show changes in AI calculation times across moves	Line graph	Move Number	AI Decision Time (seconds)

8. Project Planning

8.1 Weekly Planning

- **26 March – 2 April**
    - Finalize project structure.
    - Set up the basic user interface (window creation, board layout).
    - Start implementing classes: Game, Board, and partial Piece logic.
- **3 April – 9 April**
    - Complete core Piece and Pawn classes (movement, validation).
    - Implement capturing logic and update board state after captures.
    - Begin data collection framework (e.g. number of moves).
- **10 April – 16 April**
    - Implement AI logic with Minimax.
    - Integrate data-logging for AI decision times.
    - Conduct initial testing and debugging of core game mechanics.
- **17 April – 23 April**
    - Refine AI logic (improve evaluation function, difficulty levels).
    - Implement full statistical tracking (CSV logging, storing metrics).
    - Develop partial data-analysis code (calculating min, max, mean, SD).
- **24 April – 11 May**
    - Finalize data analysis features (generate table and graphs).
    - Polish UI (visual hints, highlighting, final improvements).
    - Comprehensive testing, bug fixes, documentation, and final submission.

8.2 50% of Tasks by 16 April

By **16 April**,expect to have:

- All core classes (Game, Board, Piece, Pawn) functional.
- Basic AI logic implemented.
- Data collection for move times, number of moves, captured pieces, and AI decision time in place.
- Basic testing of the main gameplay loop to ensure rules are followed.

8.3 75% of Tasks by 23 April

By **23 April**, expect to have:

- Enhanced AI evaluation function and adjustable difficulty.
- Full integration of data-logging into CSV.
- Initial data analysis methods (calculating min, max, mean, SD) ready.
- Preliminary UI improvements (visual hints, highlighting moves and captures).

8.4 Remaining 25% of Tasks by 11 May

By **11 May**, i plan to:

- Finalize the data presentation (table of time per move stats, plus 3 graphs).
- Complete UI polishing and user experience enhancements.
- Perform final bug fixes and comprehensive testing.
- Submit final documentation and project deliverables.