CHESS GAME

alexandrubrabete@yahoo.com

TEHNICAL UNIVERSITY OF CLUJ- NAPOCA, SOFTWARE ENGINEERING DEPARTMENT

Chess Game Java Project — Comprehensive Documentation

This project is a full-featured chess game developed in Java with a graphical user interface implemented using the Swing library. The application supports three distinct gameplay modes—**Learn**, **1v1**, and **Versus AI**—each designed to cater to different user needs: from experimentation and education to competitive and solo play. The project emphasizes object-oriented design, custom graphics rendering, user interaction handling, and basic artificial intelligence integration.

Project Overview

The chess game simulates a traditional 8x8 chessboard and includes logic for all six standard chess pieces: pawn, knight, bishop, rook, queen, and king. Each piece obeys the real-world movement rules of chess, and the board state updates dynamically based on user actions. The game provides a clean, interactive environment that is easy to use, visually intuitive, and adaptable for players at different skill levels.

Gameplay Modes

1. Learn Mode

In Learn mode, users have complete control over the configuration of the board. This mode is designed as an educational tool or sandbox, where users can experiment freely. Key features of Learn mode include:

- **Piece placement**: Users can select any piece type (pawn, rook, etc.) and color (white or black) and place it on the board by clicking.
- **Free movement**: Once placed, pieces can be selected to view their valid moves and repositioned as desired.
- **No turn restrictions**: There are no enforced rules around player turns or game outcomes, making this mode perfect for analyzing custom situations or studying move mechanics.
- **Setup flexibility**: Users can simulate historic games, create puzzle setups, or test specific openings/endgames.

2. 1v1 Mode

This is a classic two-player mode where each user takes alternating turns, one controlling the white pieces and the other the black. Important elements of 1v1 mode:

- **Automatic setup**: All pieces are placed in their standard initial positions at the start.
- **Turn-based logic**: The game enforces correct turn-taking between the players.

- **Legal move enforcement**: Players may only move pieces that belong to their color and only to valid positions.
- **Player identification**: Color-based differentiation ensures clear distinction of each player's pieces (typically blue vs. red or black vs. white).

This mode is ideal for friendly matches between two users on the same device, or for manual tournament-style matches.

3. Versus AI Mode

In this mode, the user (as white) competes against an AI that plays black. Features include:

- **Standard setup**: The game starts with pieces in their classic positions.
- **Human turn logic**: The user plays by selecting a piece and clicking on a valid destination.
- **Basic AI**: After each human move, the AI selects and executes one of its legal moves at random.
- **Alternating turns**: Control shifts seamlessly between the human and AI players, mimicking a real-world chess flow.
- **Future potential**: While the current AI is basic, this architecture opens the door for enhancements such as minimax algorithm, heuristic evaluation, and depth-based tree searches.

Technical Architecture

The application is built using clean and scalable object-oriented principles:

- **Interfaces & Polymorphism**: All chess pieces implement a shared interface, ChessPiece, which defines core methods such as getSymbol, getName, and getMoves. This ensures consistency across different types of pieces and allows for generic manipulation of piece objects.
- **Encapsulation of Logic**: Each piece class (e.g., Pawn, Bishop, Knight) encapsulates its movement logic. For instance, the bishop calculates diagonal paths until obstructed, and the knight considers L-shaped jumps.
- **Graphics**: The game board is rendered using Java2D within a JPanel. Each square is painted in a checkered pattern, and pieces are visualized as colored circles with symbols.

- **Mouse Interaction**: The user interacts with the game through mouse clicks. Click detection translates screen coordinates to board positions, allowing for piece selection, move validation, and board updates.
- **Game State Management**: A hash map (Map<Position, ChessPiece>) maintains the real-time board state. This allows for quick updates and retrieval of pieces during gameplay, as well as efficient detection of collisions and legal move destinations.

Error Handling & Edge Cases

While this version does not yet include complex features like:

- **Check or checkmate detection**
- **Castling and en passant**
- **Pawn promotion**
- **Move undo or history tracking**

…it lays a solid foundation for these to be added in future versions, given the clarity and modularity of the codebase.

Skills Demonstrated

This project has enabled the practical development and application of a broad set of technical and soft skills:

1. Object-Oriented Programming (OOP)

- Designing with interfaces and polymorphism
- Using encapsulation to manage movement logic for each piece
- Applying inheritance and abstraction principles

2. GUI Programming with Swing

- Event-driven programming (mouse listeners)
- Java2D graphics rendering for custom visuals
- Layout management and component handling (e.g., JFrame, JPanel, JComboBox)

3. Game Logic Implementation

- Turn-based gameplay management

- Movement rules enforcement
- Real-time state updating
- User interaction flows and visual feedback

4. Problem Solving & Debugging

- Addressing edge cases in movement logic
- Synchronizing UI with internal logic
- Handling dynamic board setups and rule variations

5. AI & Logic Reasoning

- Implementing a basic rule-based AI
- Handling move generation for a computer-controlled player

6. Software Design and Architecture

- Maintaining clean, readable, modular code
- Designing extensible systems that can accommodate new features
- Applying separation of concerns across game modes, graphics, logic, and interaction

Final Thoughts

The Chess Game project combines logic, design, and interactivity to create a multi-mode playable experience that is both functional and educational. It successfully demonstrates an understanding of complex rules and mechanics in a simplified yet structured Java-based application. With room for future improvements such as enhanced AI, multiplayer networking, and advanced rule enforcement, this project serves as both a showcase and a launchpad for deeper software development in game design.