Vidvayakdai A documentation

Created by

Nachaphon Maleewong 6632055621
Weeraphat Lerdcharoenvorakul 6632209021

2110215 Programming Methodology

Semester 1 Year 2024

Chulalongkorn University

Chess Game V2

Introduction

Welcome to Chess v2, an enhanced chess experience offering both the Classic and Gamble modes. Whether you're a seasoned chess enthusiast or a newcomer, Chess v2 provides engaging gameplay with unique twists to entertain you.

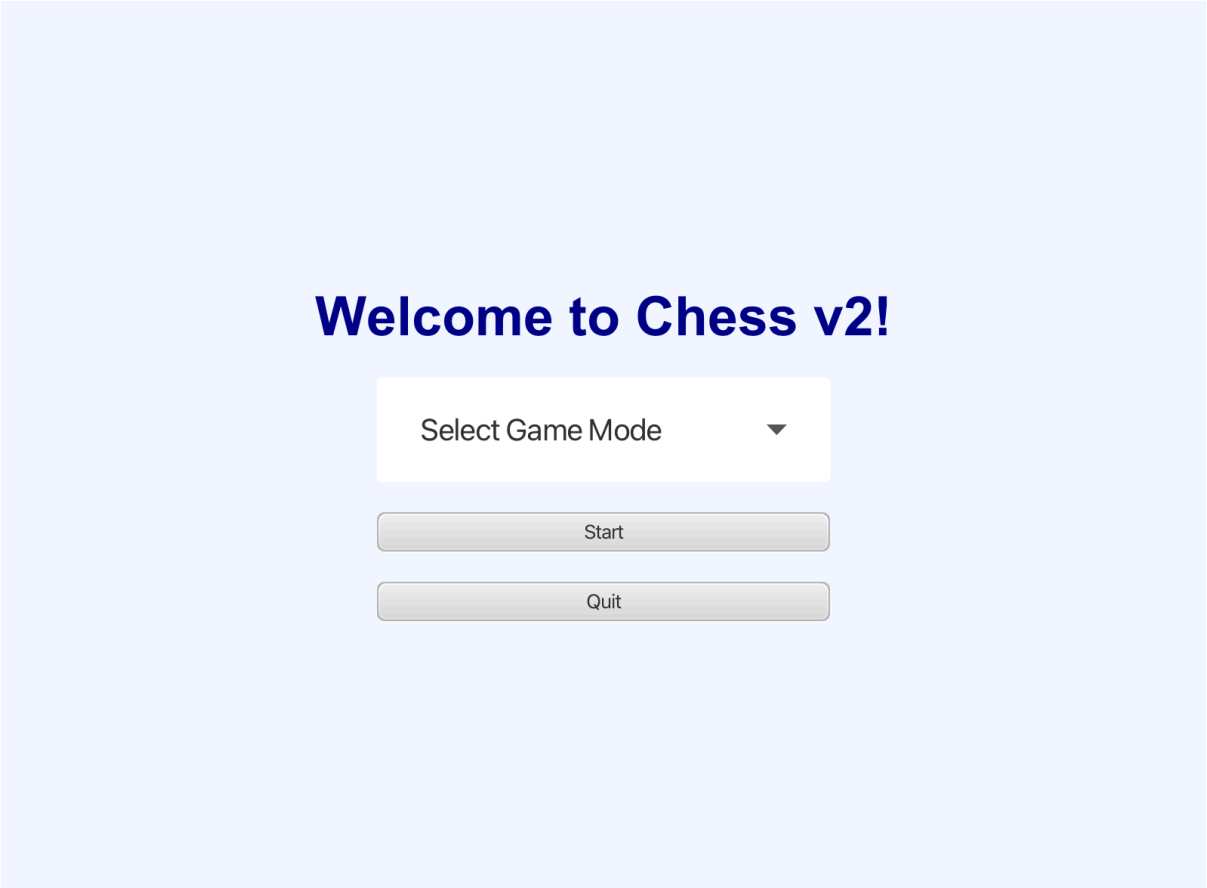
Game Mode

Classic Mode offers the traditional chess experience with the standard starting positions and rules. This mode is ideal for players who wish to engage in a familiar and strategic chess game.

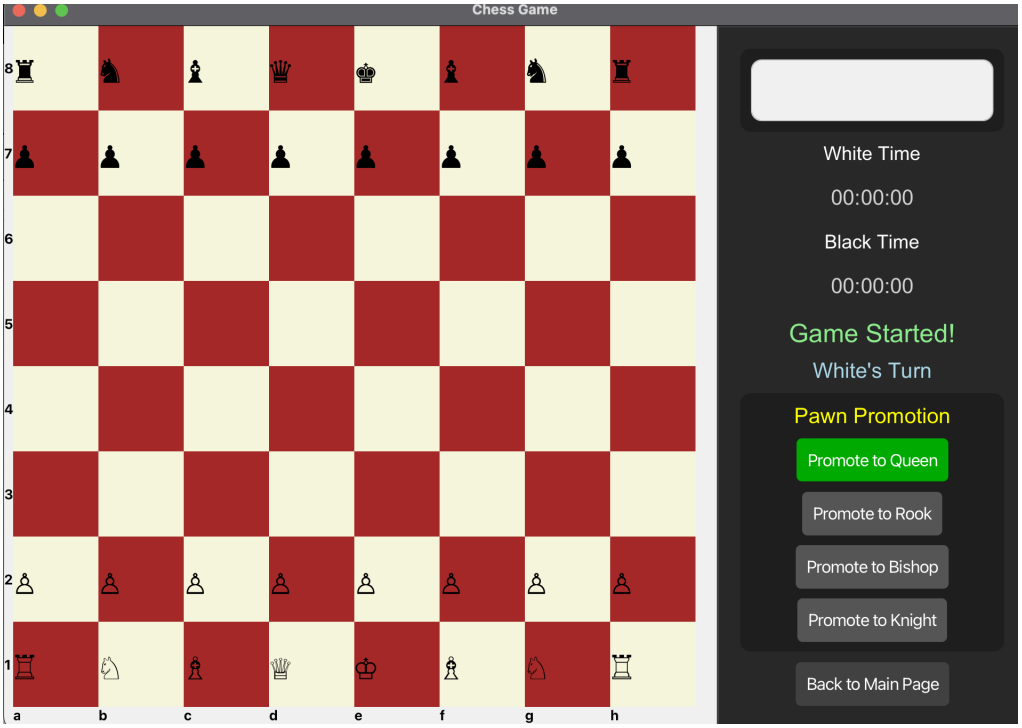
Gamble Mode introduces a fresh and unpredictable twist to chess. Instead of the traditional setup, pieces on the back rows (rows 0 and 7) are randomly shuffled each game, adding an element of surprise and requiring players to adapt their strategies dynamically.

Example

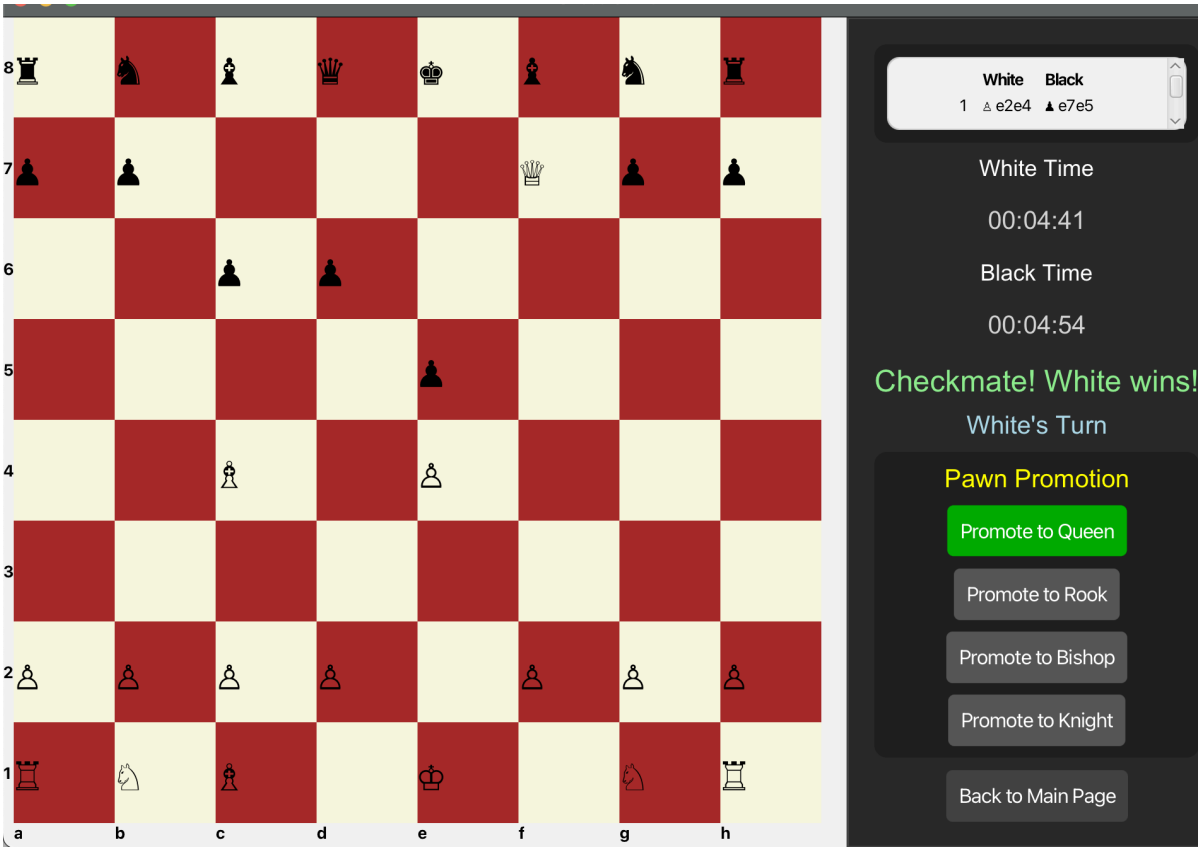
When enter the game you have choose Mode (Classic and Gamble)



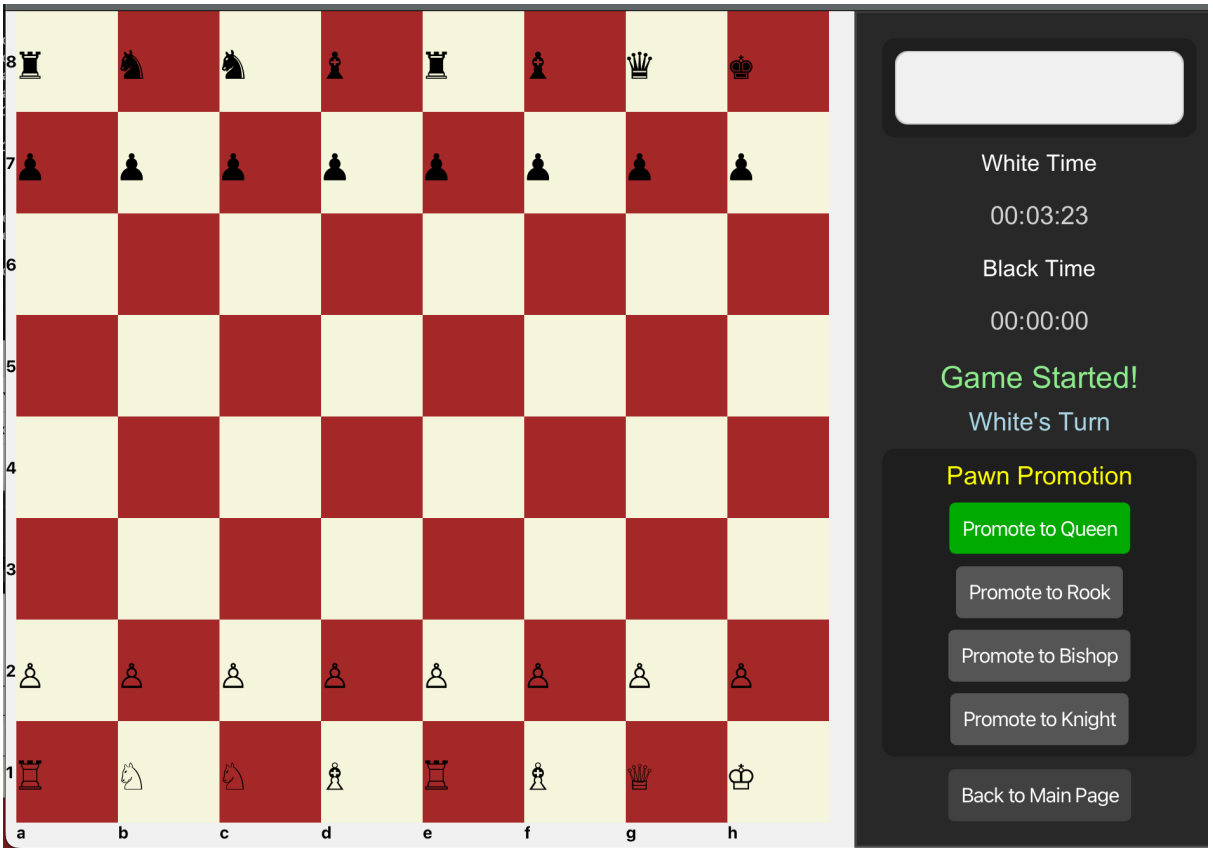
Classic Game (Like a normal chess game)



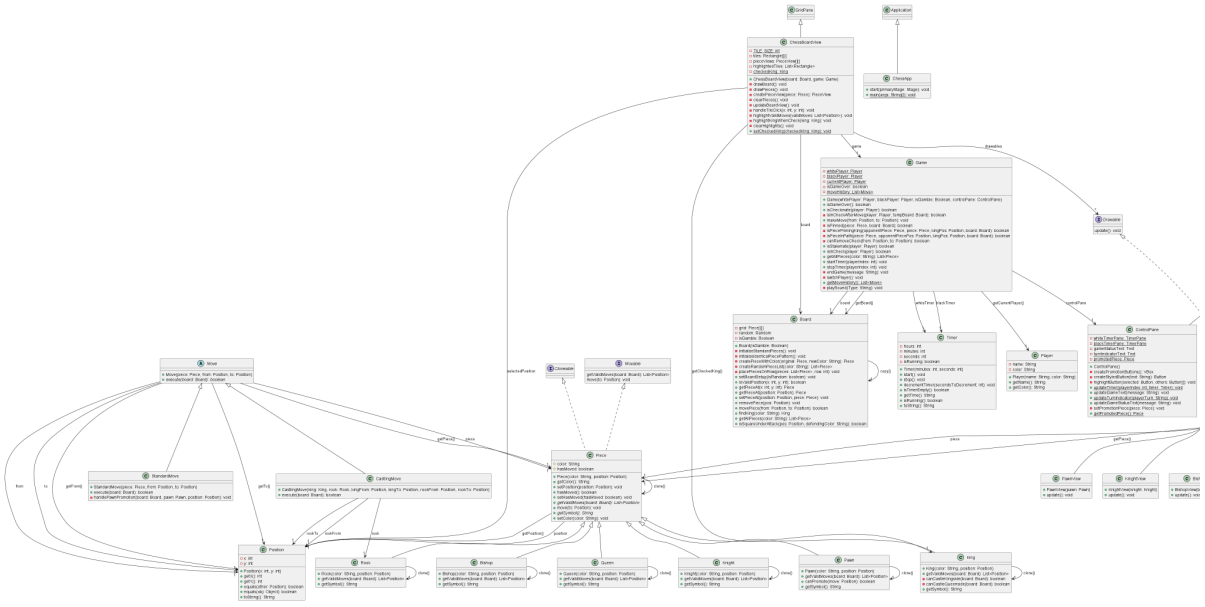
You can see history pane, white timer, black time, Game status, Pawn promotion, and the back button will appear on the control pane.



Gamble Mode



Class Diagram



1. Package interfaces

1.1 Interface Drawable

1.1.1 Methods

Method	Description
+ void update();	Updates the visual representation or state of the object. Useful for refreshing the display to reflect changes in properties.

1.2 Interface Movable

1.2.1 Methods

Method	Description
+ List<Position> getValidMoves(Board board);	Retrieves a list of all valid positions for the object based on the current game board state. Consider movement rules and constraints.
+ void move(Position to);	Moves the object to the specified Position and updates its internal state accordingly.

2. Package Pieces

2.1 Abstract Class Piece

2.1.1 Fields

Field	Description
# String color;	The color of the piece (white or black).
# Position position;	The current position of the piece

	on the board.
# boolean hasMoved;	Indicates whether the piece has moved from its initial position.

2.1.2 Constructor

Constructor	Description
+ Piece(String color, Position position);	Initializes the piece with a specific color and position.

2.1.3 Methods

Method	Description
+ String setColor();	Setter of string color.
+ String getColor();	Returns the color of the piece.
+ Position getPosition();	Returns the current position of the piece.
+ void setPosition(Position position);	Sets a new position for the piece.
+ boolean hasMoved();	Check if the piece has moved.
+ void setHasMoved(boolean hasMoved);	Sets the hasMoved flag.
+ List<Position> getValidMoves(Board board);	Abstract method to retrieve valid moves.
+ void move(Position to);	Moves the piece to the specified position.
+ String getSymbol();	Abstract method to get the piece's symbol.
+ Piece clone();	Abstract method for cloning the piece.

2.2 Class Bishop extends Piece

Moves diagonally any number of squares.

2.2.1 Constructor & Methods

Constructor & Method	Description
+ Bishop(String color, Position position);	Initializes the piece with a color and position.
+ getValidMoves(Board board);	Returns a list of valid positions the piece can move to.
+ clone();	Creates a deep copy of the piece.
+ getSymbol();	Returns the Unicode symbol for the piece, depending on its color.

2.3 Class King extends Piece

Moves one square in any direction. Cannot move into check.

2.3.1 Constructor & Methods

Constructor & Method	Description
+ Knight(String color, Position position);	Initializes the piece with a color and position.
+ getValidMoves(Board board);	Returns a list of valid positions the piece can move to.
- boolean canCastleKingside(board Board);	Checks if the King can perform castling on the kingside.
- boolean canCastleQueenside(board Board);	Checks if the King can perform castling on the queenside.
+ clone();	Creates a deep copy of the piece.
+ getSymbol();	Returns the Unicode symbol for the piece, depending on its color.

2.4 Class Knight extends Piece

Moves in an 'L' shape: two squares in one direction, then one square perpendicular.

2.4.1 Constructor & Methods

Constructor & Method	Description
+ Knight(String color, Position position);	Initializes the piece with a color and position.
+ getValidMoves(Board board);	Returns a list of valid positions the piece can move to.
+ clone();	Creates a deep copy of the piece.
+ getSymbol();	Returns the Unicode symbol for the piece, depending on its color.

2.5 Class Pawn extends Pieces

Moves forward one square. Can move two squares on its first move. Captures diagonally.

2.5.1 Constructor & Methods

Constructor & Method	Description
+ Pawn(String color, Position position);	Initializes the piece with a color and position.
+ getValidMoves(Board board);	Returns a list of valid positions the piece can move to.
+ clone();	Creates a deep copy of the piece.
+ getSymbol();	Returns the Unicode symbol for the piece, depending on its color.

2.6 Class Queen extends Pieces

Combines the movement of a Rook and a Bishop.

2.6.1 Constructor & Methods

Constructor & Method	Description
+ Queen(String color, Position position);	Initializes the piece with a color and position.
+ getValidMoves(Board board);	Returns a list of valid positions the piece can move to.
+ clone();	Creates a deep copy of the piece.
+ getSymbol();	Returns the Unicode symbol for the piece, depending on its color.

2.7 Class Rook extends Pieces

Moves horizontally or vertically any number of squares.

2.7.1 Constructor & Methods

Constructor & Method	Description
+ Rook(String color, Position position);	Initializes the piece with a color and position.
+ getValidMoves(Board board);;	Returns a list of valid positions the piece can move to.
+ clone();	Creates a deep copy of the piece.
+ getSymbol();	Returns the Unicode symbol for the piece, depending on its color.

3. Package Model

3.1 Class Player

Represents a chess player with a name and color.

3.1.1 Fields

Fields	Description

- String name;	The name of player
- String color;	The color of player

3.1.2 Constructor

Constructor	Description
+ Player(String name, String color, Game game);	Initializes the player with a name, color, and game association.

3.1.3 Methods

Methods	Description
+ String getName();	Returns the name of the player.
+ String getColor();	Returns the color of the player.

3.2 Class Position

Represents a position on the chessboard.

3.2.1 Fields

Fields	Description
- int x;	The position in horizontal
- int y;	The position in vertical

3.2.2 Constructor

Constructor	Description
+ Position(int x, int y);	Initializes the position with x and y coordinates.

3.2.3 Methods

Methods	Description
+ int getX();	Returns the x-coordinate of the position.
+ int getY();	Returns the y-coordinate of the position.
+ boolean equals(Position other);	Checks if two positions are equal based on their coordinates.
+ String toString();	Returns the string representation of the position, e.g., (x,y).

4. Package gui

4.1 Abstract Class PieceView

4.1.1 Fields

Fields	Description
- Piece piece;	Piece type
- Text text;	Symbol of piece

4.1.2 Constructor

Constructor	Description
+ PieceView(Piece piece);	Creates a PieceView object with the specified chess piece.

4.1.3 Methods

Methods	Description

+ Piece getPiece();	Returns the associated Piece object.
+ Text getText();	Returns the Text object used for rendering the piece.
+ void update();	Updates the visual representation of the piece based on its current state.

4.2 Class BishopView extends PieceView

4.2.1 Constructor

Constructor	Description
+ BishopView(String color, Position position);	Create Bishop GUI

4.2.2 Method

Methods	Description
+ update();	Update the bishop view.

4.3 Class KingView extends PieceView

4.3.1 Constructor

Constructor	Description
+ KingView(String color, Position position);	Create King GUI

4.3.2 Method

Methods	Description
+ update();	Update the king view.

4.4 Class KnightView extends PieceView

4.4.1 Constructor

Constructor	Description
+ KnightView(String color, Position position);	Create Knight GUI

4.4.2 Method

Methods	Description
+ update();	Update the knight view.

4.5 Class PawnView extends PieceView

4.5.1 Constructor

Constructor	Description
+ PawnView(String color, Position position);	Create Pawn GUI

4.5.2 Method

Methods	Description
+ update();	Update the pawn view.

4.6 Class QueenView extends PieceView

4.6.1 Constructor

Constructor	Description
+ QueenView(String color, Position position);	Create Queen GUI

4.6.2 Method

Methods	Description

+ update();	Update the queen view.

4.7 Class RookView extends PieceView

4.7.1 Constructor

Constructor	Description
+ RookView(String color, Position position);	Create Rook GUI

4.7.2 Method

Methods	Description
+ update();	Update the rook view.

4.8 Class ChessBoardView

4.8.1 Fields

Fields	Description
- *int TILE_SIZE;*	Tile size in pixels.
- Board board;	The current state of the game board.
- Game game;	The game instance managing the logic.
- Position selectedPosition;	The currently selected position on the board.
- Rectangle[][] tiles;	Grid of tiles that make up the board.
- PieceView[][] pieceViews;	Views of the pieces on the board.
- List<Drawable> drawables;	List of drawable elements on the board.

- List<Rectangle> highlightedTiles;	Tiles that are highlighted.
- King checkedKing;	The king is currently in check, if any.

4.8.2 Constructor

Constructor	Description
+ ChessBoardView(Board board, Game game);	Initializes the chessboard view with the given board and game instances.

4.8.3 Methods

Methods	Description
- void drawBoard();	Draws the chessboard grid and labels.
- void drawPieces();	Place pieces on the board according to the game state.
- PieceView createPieceView(Piece piece);	Creates a view for the given chess piece.
- void clearPieces();	Removes all pieces from the board.
- void updateBoardView();	Updates the view of the board, refreshing pieces and highlights.
- void handleTileClick(int x, int y);	Handles the event of a tile being clicked.
- void highlightValidMoves(List<Position > validMoves);	Highlights the tiles for valid moves of a selected piece.
- void highlightKingWhenCheck(King king);	Highlights the king's tile if it is in check.

- void clearHighlights();	Clears all highlighted tiles.
+ void setCheckedKing(King checkedKing);	Sets the king currently in check.
+ King getCheckedKing();	Gets the king currently in check, if any.

4.9 Class ControlPane

4.9.1 Fields

Fields	Description
- TimerPane whiteTimerPane;	Timer for the white player.
- TimerPane blackTimerPane;	Timer for the black player.
- Text gameStatusText;	Displays the current status of the game.
- Text turnIndicatorText;	Indicates the current player's turn.
- Piece promotedPiece;	Holds the piece chosen for promotion.
- int WHITE_TIMER_INDEX = 0;	set white timer index for the array.
- int BLACK_TIMER_INDEX = 0;	set black timer index for the array.

4.9.2 Constructor

Constructor	Description
+ ControlPane(Stage stage);	Initializes the ControlPane with game status and timer displays.

4.9.3 Methods

Methods	Description

- private Vbox createPromotionButtons();	Creating 4 buttons for pawn promotion.
- private Button createStyledButton(String text);	To make the button beautier.
- void highlightButton(Button selected, Button… others);	To highlight the button which gets clicked.
+ void updateTimer(int playerIndex, Timer timer);	Updates the timer for the specified player.
+ void updateGameText(String message);	Updates the game status text. Message is shown in green or red.
+ void updateTurnIndicator(String playerTurn);	Updates the turn indicator with the current player's turn.
- void setPromotionPiece(Piece piece);	Sets the piece chosen for promotion.
+ Piece getPromotedPiece();	Returns the piece chosen for promotion.

4.10 Class HistoryPane

4.10.1 Fields

Fields	Description
- GridPane historyGrid;	GridPane to hold the move history.
- ScrollPane scrollPane;	ScrollPane to make the GridPane scrollable.

4.10.2 Constructor

Constructor	Description

+ HistoryPane();	Creates the HistoryPane with a scrollable view for displaying move history.

4.10.3 Methods

Methods	Description
+ void updateHistory();	Updates the history grid with the current moves.
- String positionToNotation(Position position);	Converts a Position object to chess notation (e.g., a1, f4).

4.11 Class StartPane

Represents the starting pane of the chess game GUI, where the user can select the game mode and start the game.

4.11.1 Fields

Fields	Description
- VBox root;	The root container for all GUI elements in the StartPane.
+ ComboBox<String> modeComboBox;	Drop-down menu for selecting the game mode.
+ Button startButton;	Button to start the game.

4.11.2 Constructor

Constructor	Description
+ StartPane(Stage primaryStage);	Creates and initializes the StartPane with the primary stage.

4.12 Class TimerPane

4.11.1 Field

Field	Description

- Text timerText;	The text object to display the timer.

4.11.2 Constructor

Constructor	Description
+ TimerPane(int playerIndex);	Creates a TimerPane for a player, initializing the timer text.

4.11.3 Methods

Methods	Description
+ void setTimer(Timer timer);	Updates the timer text to display the given Timer object.

5. Package game

5.1 Class Board

5.1.1 Fields

Field	Description
- Piece[][] grid;	The grid representing the chess board.
- Random random;	Random number generator for shuffling.
- Boolean isGamble;	Flag indicating if gamble mode is enabled.
- int BOARD_SIZE = 8;	set board size.

5.1.2 Constructor

Constructor	Description
+ Board(Boolean isGamble);	Constructor to initialize the board based on game mode.

5.1.3 Methods

Methods	Description

- void initializeStandardPieces();	Set up the board for standard chess.
- void initializeIdenticalPiecePattern();	Set up the board with identical patterns for gamble mode.
- Piece createPieceWithColor(Piece original, String newColor);	Creating new piece with a color.
- List<Piece> createRandomPieceList(String color);	Creating a list to randomize it later.
- void placePiecesOnRow(List<Piece> pieces, int row);	Placing pieces on the row.
+ void setBoardSetup(boolean iseRandon)	to check if it is a Gamble or Classic.
+ boolean isValidPosition(int x, int y);	Return is that row and column is valid.
+ Piece getPieceAt(int x, int y);	Get piece by row and column.
+ Piece getPieceAt(Position position);	Get piece by position.
+ void setPieceAt(Position position, Piece piece);	Set piece at that position.
+ void removePiece(Position pos);	Removing pieces from those positions.
+ boolean movePiece(Position from, Position to);	Move a piece from one position to another.
+ King findKing(String color);	Return king if it has.
+ List<Piece> getAllPieces(String color);	Return all pieces.
+ Board copy();	Copying board.
+ boolean isSquareUnderAttack(Position pos, String defendingColor);	Check if a square is under attack.

5.2 Abstract Class Move

5.2.1 Fields

Field	Description

- Piece piece;	The piece being moved.
- Position from;	The initial position of the piece.
- Position to;	The destination position of the piece.

5.2.2 Constructor

Constructor	Description
+ Move(Piece piece, Position from, Position to);	Constructor for the Move class.

5.2.3 Methods

Methods	Description
+ Piece getPiece();	Return pieces.
+ Position getFrom();	Return position before moving.
+ Position getTo();	Return position after moving.
+ boolean execute(Board board);	Abstract method to execute the move on the board.

5.3 Class CastlingMove extends Move

5.3.1 Fields

Field	Description
- Rook rook;	The rook involved in the castling move.
- Position rookFrom;	The initial position of the rook.
- Position rookTo;	The final position of the rook.

5.3.2 Constructor

Constructor	Description
+ CastlingMove(King king, Rook rook, Position kingFrom, Position kingTo, Position rookFrom, Position rookTo);	Constructor for the castling move.

5.3.3 Methods

Methods	Description
+ boolean execute(Board board);	Execute the castling move on the board.

5.4 Class StandardMove extends Move

5.4.1 Fields

Field	Description
- Piece piece;	The piece being moved.
- Position from;	The starting position of the piece.
- Position to;	The destination position of the piece.

5.4.2 Constructor

Constructor	Description
+ StandardMove(Piece piece, Position from, Position to);	Creates a move for a piece from one position to another.

5.4.3 Methods

Methods	Description
+ boolean execute(Board board);	Executes the move on the board.
- void handlePawnPromotion(Board board, Pawn pawn, Position position);	Handles the promotion of a pawn.

5.5 Class Timer

5.5.1 Fields

Field	Description
- int hours;	The hour component of the timer.
- int minutes;	The minute component of the timer.
- int seconds;	The second component of the timer.

- boolean isRunning;	Indicates whether the timer is running.

5.5.2 Constructor

Constructor	Description
+ Timer(int minutes, int seconds);	Creates a timer with the specified minutes and seconds.

5.5.3 Methods

Methods	Description
+ void start();	Starts the timer.
+ void stop();	Stops the timer.
+ void decrementTimer(int secondsToDecrement);	Decrements the timer by a specified number of seconds.
+ boolean isTimerEmpty();	Check if the timer has run out.
+ String getTime();	Returns the time in HH:mm:ss format.
+ boolean isRunning();	Check if the timer is currently running.
+ String toString();	Returns the time in string format.

5.6 Class Game

5.6.1 Fields

Field	Description
- Board board;	The chessboard for the game.
- Player whitePlayer;	The white player in the game.
- Player blackPlayer;	The black player in the game.
- Player currentPlayer;	The current player whose turn it is.
- boolean isGameOver;	Flag indicating whether the game is over.
- List<Move> moveHistory;	History of all moves made during the game.

- Timer whiteTimer;	Timer for the white player.
- Timer blackTimer;	Timer for the black player.
- ControlPane controlPane;	GUI component for control actions.

5.6.2 Constructor

Constructor	Description
+ Game(Player whitePlayer, Player blackPlayer, Boolean isGamble, ControlPane controlPane);	Initializes a new chess game with specified players, mode, and GUI control pane.

5.6.3 Methods

Methods	Description
+ Board getBoard();	Returns the game board.
+ Player getCurrentPlayer();	Returns the current player.
+ boolean isGameOver();	Checks if the game is over.
+ void endGame(String message)	set isGameOver = true.
+ boolean isCheckmate(Player player);	Checks if the specified player is in checkmate.
+ boolean isInCheckAfterMove(String playerColor, Board tempBoard);	Check if the king is checked after piece move
+ boolean isStalemate(Player player);	Checks if the specified player is in stalemate.
+ boolean isInCheck(Player player);	Checks if the specified player is in check.
+ void makeMove(Position from, Position to);	Makes a move from the specified position to another.

+ boolean isPinned(Piece piece, Board board,Position targetPos);	Check if the piece is pinned.
+ boolean isAligned(Position pos1, Position pos2);	Check if the piece and king is on the same row,column, or diagonal
+ boolean canRemoveCheck(Position from, Position to);	Check if one of our pieces can capture the checking piece.
+ void startTimer(int playerIndex);	Starts the timer for the specified player.
+ void stopTimer(int playerIndex);	Stop the timer for the specified player.
+ List<Move> getMoveHistory();	Returns the history of moves made during the game.
- void switchPlayer();	Switch the current player.
- void playSound(String Type);	Plays a sound effect based on the type.
+ List<Piece> getAllPieces(String color)	To get a list of all pieces.

6. Package app

6.1 Class ChessApp

6.1.1 Constructor

Constructor	Description
+ ChessApp();	Creates an instance of the ChessApp class.

6.1.2 Methods

Methods	Description

+ void start(Stage primaryStage);	Launches the main window of the chess application with the StartPane as the initial screen.
+ static void main(String[] args);	The entry point for the JavaFX application. Launch the application.