CCP2201 Project

Trimester 2310

By HireSphere

Team Leader:

Ow Ka Sheng, 0183737174, 1211108820@student.mmu.edu.my

Team Members:

Yoong Tzer Shih, 01163311659, 1211108421@student.mmu.edu.my

Harold Goh, 01155052057, 1211108866@student.mmu.edu.my

Lau Zi Herng, 0143058704, 1211108444@student.mmu.edu.my

Table of Contents

Compile & Run Instructions

1. Open terminal and navigate to the directory. Replace `/path/to/your/files` with the actual path where your Java source files are located:
   a. `cd /path/to/your/java/files`

2. Compile the source code:
   a. You can compile specific files with the following command:
      i. `javac Biz.java BlueState.java ChessBoard.java ChessController.java ChessModel.java PlayerState.java ChessView.java IconPathProvider.java KwazamChess.java Piece.java PlayerState.java Position.java Ram.java RedState.java Sau.java Tor.java Xor.java`
   b. You can compile all `.java` files in the folder with the following command:
      i. `javac *.java`

3. Execute the compiled program using the `java` commands:
   a. `java KwazamChess`

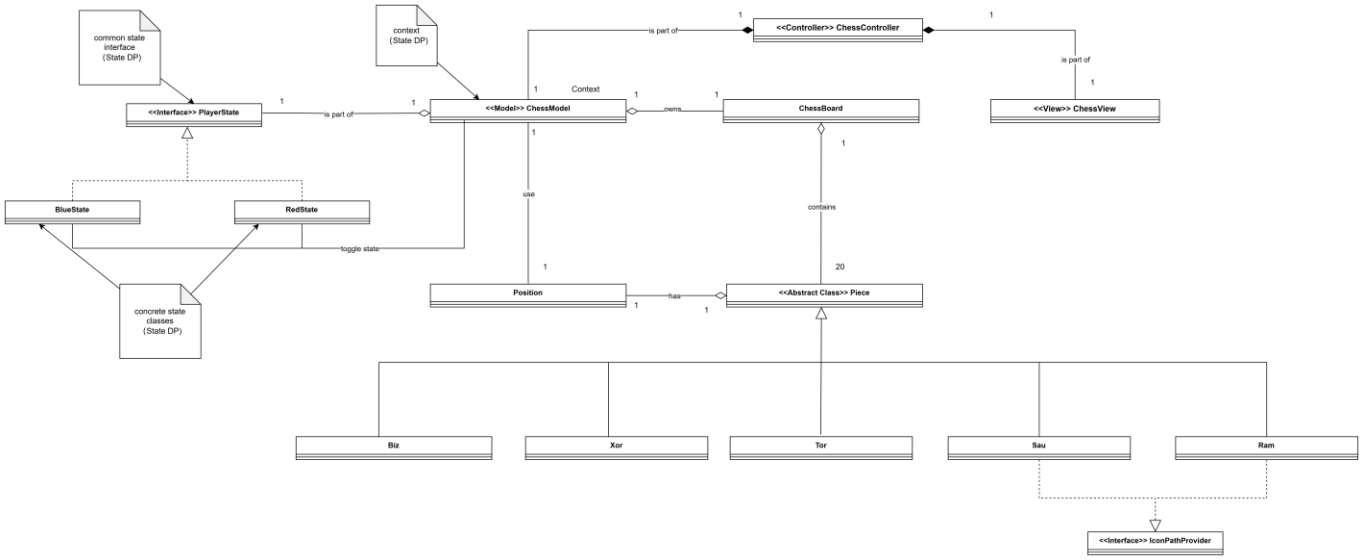
4. Ensure the **Kwazam Chess** folder includes `KwazamChess.java` as this is the file that contains the main method and is the entry point for the program:
   a.
```
public static void main(String[] args) {

        new ChessController(new ChessModel());
    }
```

Notes

- All commands above are **case-sensitive**.
- Ensure you have the **Java Development Kit (JDK)** installed and properly configured.
- If you encounter issues with the `javac` or `java` commands, verify that the `JAVA_HOME` environment variable is set, and the `bin` directory is included in the `PATH`.

UML Class Diagram

https://drive.google.com/file/d/1tW65EbV44tWpEB8-nYj6ryYwvGleNT1V/view?usp=sharing



Ram Class

Ram
- nextPosOperator: String - flipIconPath: String - initialIconPath: String
+ setCurrentValidMoves(Piece[][]): void {override} + move(int): Position {override} - updateIconDirection(int): void - updateDirection(int): void - isWithinBound(int,int): boolean + setNextOperator(String): void + setFlipIconPath(String): void + setInitialIconPath(String): void + getNextOperator(): String + getFlipIconPath(): String + getInitialIconPath(): String

Sau Class

Sau
- flipIconPath: String - initialIconPath: String
+ setCurrentValidMoves(Piece[][]): void {override} - isWithBoard(int,int): boolean - isEmpty(int,int,Piece[][]): boolean + getFlippedIcon(): String {override} + getInitialIcon(): String {override}

Biz Class

Biz
+ setCurrentValidMoves(Piece[][]): void {override} - isWithBoard(int,int): boolean - isEmpty(int,int,Piece[][]): boolean

Xor Class

Xor
+ setCurrentValidMoves(Piece[][]): void {override}

Tor Class

Tor
+ setCurrentValidMoves(Piece[][]): void {override}

Piece Class

<<Abstract Class>> Piece
position: Position # team: String # iconPath: String # possibleNextMove: ArrayList<int[]> possibleNextMove
+ getValidMoves(): ArrayList<int[]> + getTeam(): String + getIconPath(): String + getPosition(): Position + setCurrentValidMoves(Piece[][]): void {abstract} + move(int): Position

ChessBoard Class

ChessBoard
+ COLUMNS: int {static,readOnly} = 5 + ROWS: int {static,readOnly} = 8 - board: Piece[][] - pieceCount: int - selectedPiece: Piece
- initialize(int,int,int,String): void + createPiece(String,Position,String): Piece + moveSelectedPiece(int): void + updateBoard(int): void + flipBoard(String): void + getBoard(): Piece[][] + getPieceCount(): int + getSelectedPiece(): Piece + setBoard(Piece[][]): void + setPieceCount(int): void + setSelectedPiece(Piece): void

Position Class

Position
- row: int - column: int
+ convertPositionToChessNotation(int): String {static} + convertRowColumnToPosition(int,int): int {static} + convertPositionToRowColumn(int): int[] {static} + getRow(): int + getColumn(): int

PlayerState Class

<<Interface>> PlayerState
+ playMove(ChessModel,int): void + getTeam(): String

RedState Class

RedState
+ playMove(ChessModel,int): void {override} + getTeam(): String {override}

BlueState Class

BlueState
+ playMove(ChessModel,int): void {override} + getTeam(): String {override}

IconPathProvider Class

<<Interface>> IconPathProvider
+ getFlipIconPath(): String + getInitialIconPath(): String

ChessModel Class

<<Model>> ChessModel
- chessBoard: ChessBoard - state: PlayerState - round: int - moveHistory: ArrayList<String>
+ restartChessGame(): void + trackRound(): void + switchTorXor(): Map<String,ArrayList<Piece>> + saveGame(String): void + loadGame(String): boolean + determineWinner(int): String + addMoveToHistory(int): void + clearMoveHistory(): void + setState(PlayerState): void +getMoveHistory(): ArrayList<String> + getChessBoard(): ChessBoard + getCurrentTurnTeam(): String + getPlayerState(): PlayerState + getRound(): int

ChessView Class

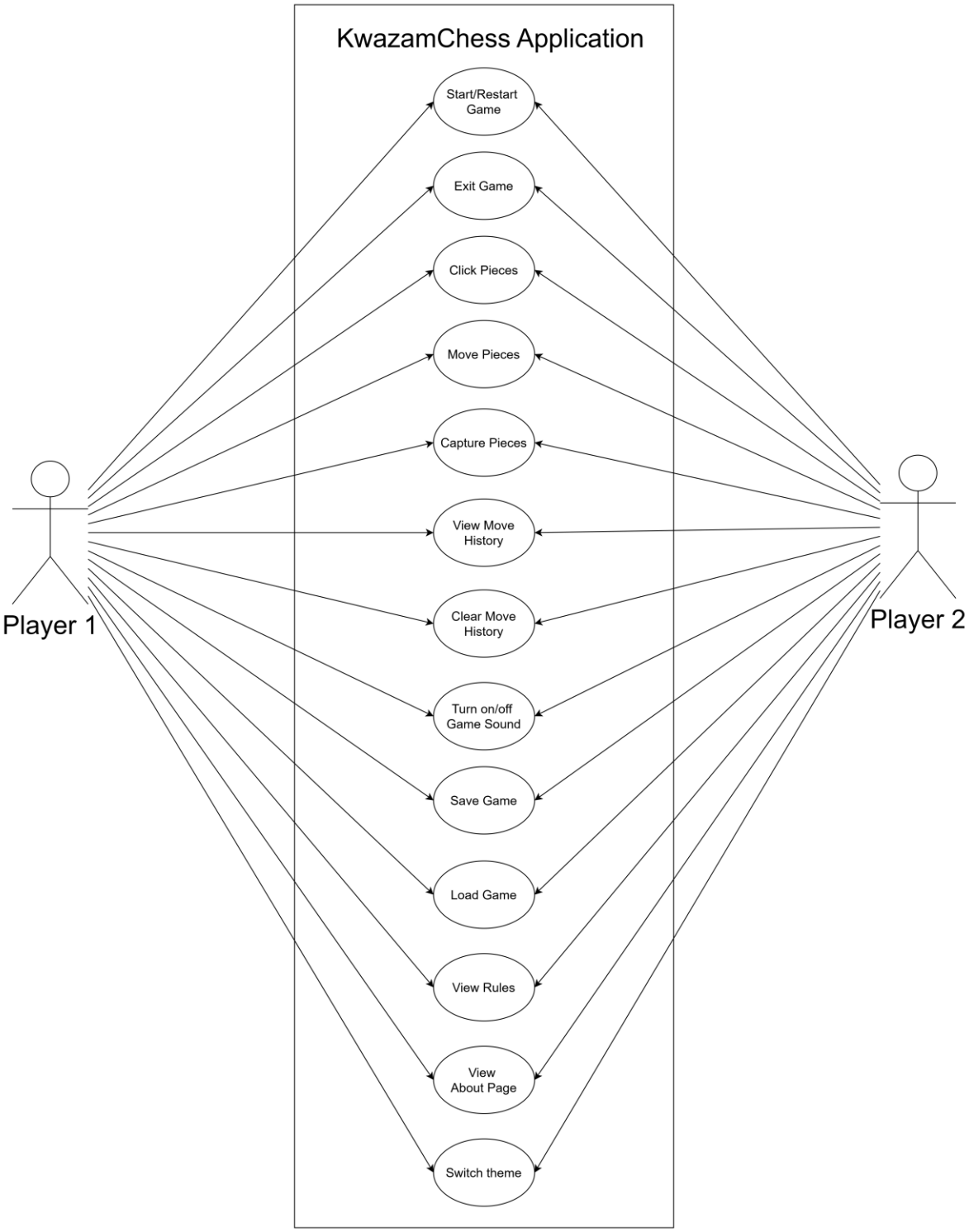
https://drive.google.com/file/d/1tW65EbV44tWpEB8-nYj6ryYwvGleNT1V/view?usp=sharing

```
ChessView
-boardColor_1: Color
-boardColor_2: Color
-taskbarColor: Color
-taskbarItemColor: Color
-rowColor: Color
-colColor: Color
-movesColor: Color
-hintColor: Color
-turnLabelColor: Color
-border: Border
-chessPanel: JPanel
-chessPieces: JButton[][]
-taskBar: JMenuBar
-moves: JPanel
-movesContent: JPanel
-clear: JButton
-boardRows: JPanel
-boardColumns: JPanel
-rowLabel: JLabel
-colLabel: JLabel
-turnIndicator: JLabel
-lastHoveredPiece: JButton
-lastOriginalColor: Color
-lastOrignalBorder: Border
-game: JMenu
-move: JMenu
-settings: JMenu
-sound: JMenu
-theme: JMenu
-help: JMenu
-loadGameItem: JMenuItem
-saveGameItem: JMenuItem
-newGameItem: JMenuItem
-restartGameItem: JMenuItem
-exitItem: JMenuItem
-rulesItem: JMenuItem
-aboutItem: JMenuItem
-onSoundItem: JMenuItem
-offSoundItem: JMenuItem
-lightThemeItem: JMenuItem
-darkThemeItem: JMenuItem
-emptyBox: JPanel
-isAudioEnabled: boolean
-isFlipped: boolean
-layeredPane: JLayeredPane
+getChessPieces(): JButton[][]
+getLoadGameItem(): JMenuItem
+getSaveGameItem(): JMenuItem
+getNewGameItem(): JMenuItem
+getRestartGameItem(): JMenuItem
+getExitItem(): JMenuItem
+getOnSoundItem(): JMenuItem
+getOffSoundItem(): JMenuItem
+getLightThemeItem(): JMenuItem
+getDarkThemeItem(): JMenuItem
+getRulesItem(): JMenuItem
+getAboutItem(): JMenuItem
+getLayeredPane(): JLayeredPane
+setAudioEnabled(isAudioEnabled: boolean): void
+setIsFlipped(isFlipped: boolean): void
+setIcon(btn: JButton, iconPath: String, width: int, height: int): void
+displayRules(): void
+displayAbout(): void
+changeTheme(isLightTheme: boolean): void
+refreshMoveHistory(moveHistory: ArrayList<String>): void
+toggleBoardLabel(team: String): void
+resizeWindow(board: Piece[][]): void
+pieceSelectSound(): void
+pieceMoveSound(): void
+pieceTakenOutSound(): void
+togglePlayerPieceState(color: String): void
+setHighlightPath(possiblePath: ArrayList<int[]>, enableButton: boolean): void
+updateBoardView(prevRow: int, prevCol: int, newRow: int, newCol: int): void
+flipBoardView(board: Piece[][]): void
+updateTurnIndicator(team: String): void
+clearMoves(): void
+switchTorXorView(xorTorPieces: Map<String, ArrayList<Piece>>): void
+displayWinner(winner: String): void
+displayRestartSuccessfulDialog(): void
+displaySaveSuccessfulDialog(): void
+displayLoadDialog(status: String): void
-setUpChessBoard(board: Piece[][]): void
-taskbarSetUp(): void
-movesPanelSetUp(): void
-layeredPaneSetUp(): void
-setMenuStyle(menu: JMenu): void
-resizeButtonIcon(piece: Piece, button: JButton, newSize: Dimension): void
-getHoverColor(baseColor: Color): Color
-applyLightTheme(): void
-applyDarkTheme(): void
-updateBoardColor(color1: Color, color2: Color): void
-updateBoardLabelColors(rowColor: Color, colColor: Color): void
-updateTaskbarColors(): void
-updateMovesPanel(): void
-updateMenuForegroundColors(color: Color): void
-updateMenuColors(menu: JMenu, color: Color): void
-showHoverEffect(cell: JButton): void
-hideHoverEffect(): void
-createWinnerPanel(backgroundColor: Color, shadowColor: Color): JPanel
-swapTorXorIcon(pieces: ArrayList<Piece>): void
+getClearButton(): JButton
```

ChessController Class

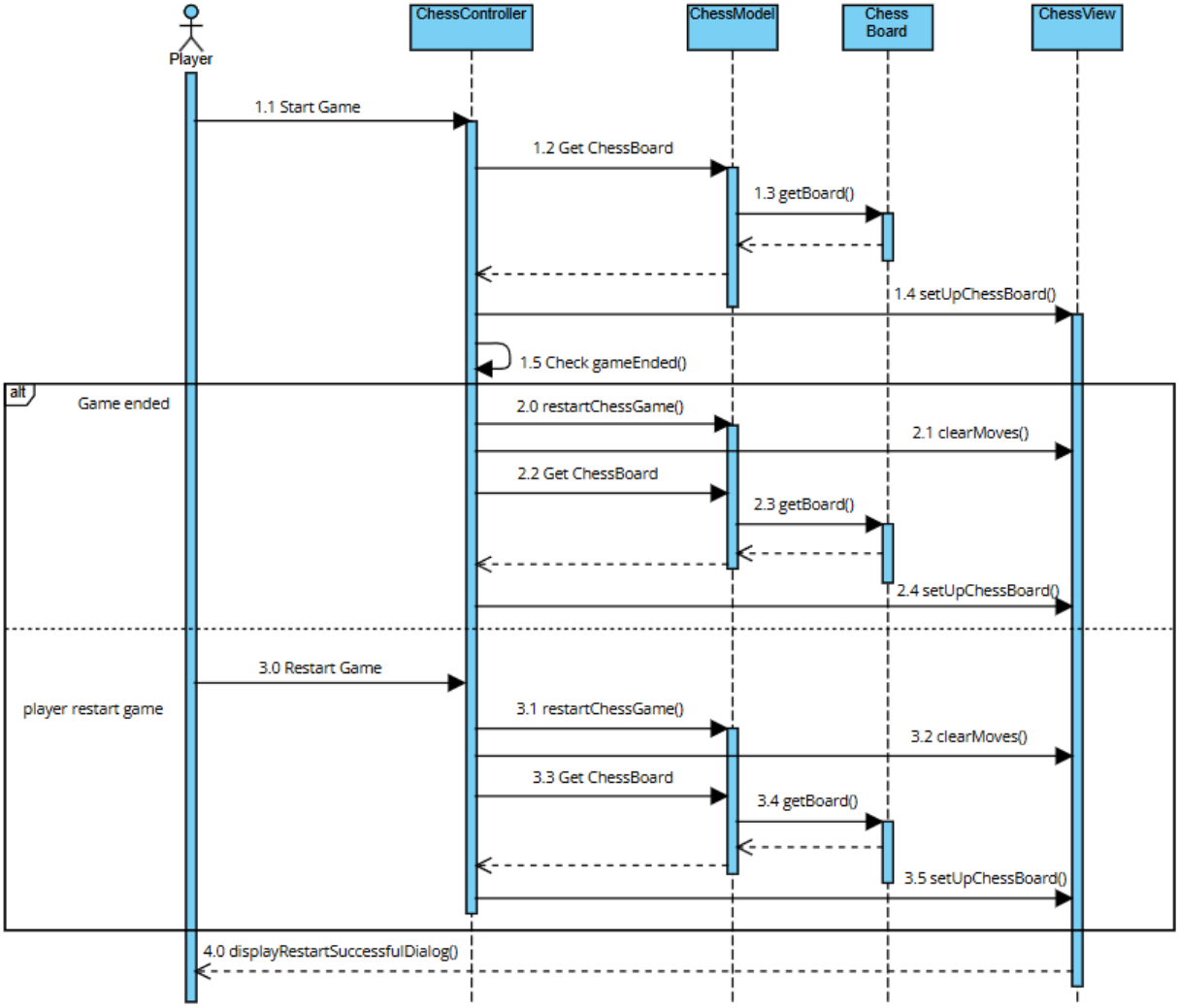
<<Controller>> ChessController
- model: ChessModel - view: ChessView
+ setUpCellActionListener(): void + handlePieceAction(e: ActionEvent): void + updateGame(e: ActionEvent): void + checkGameEnded(): boolean + restartGame(): void

Use Case Diagram

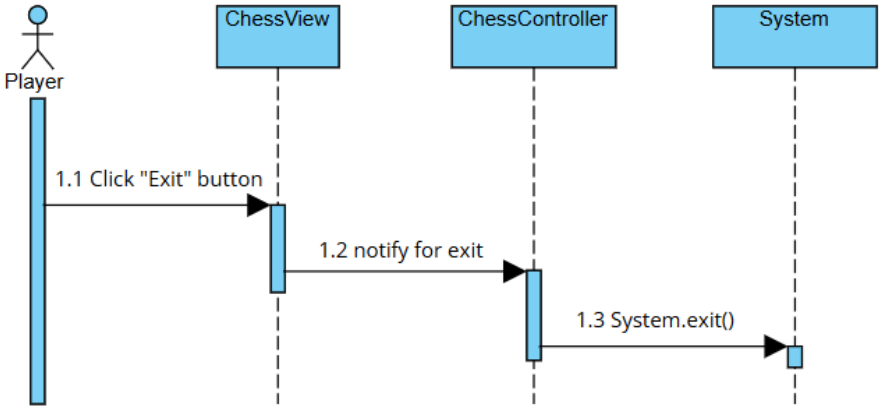


Sequence Diagram

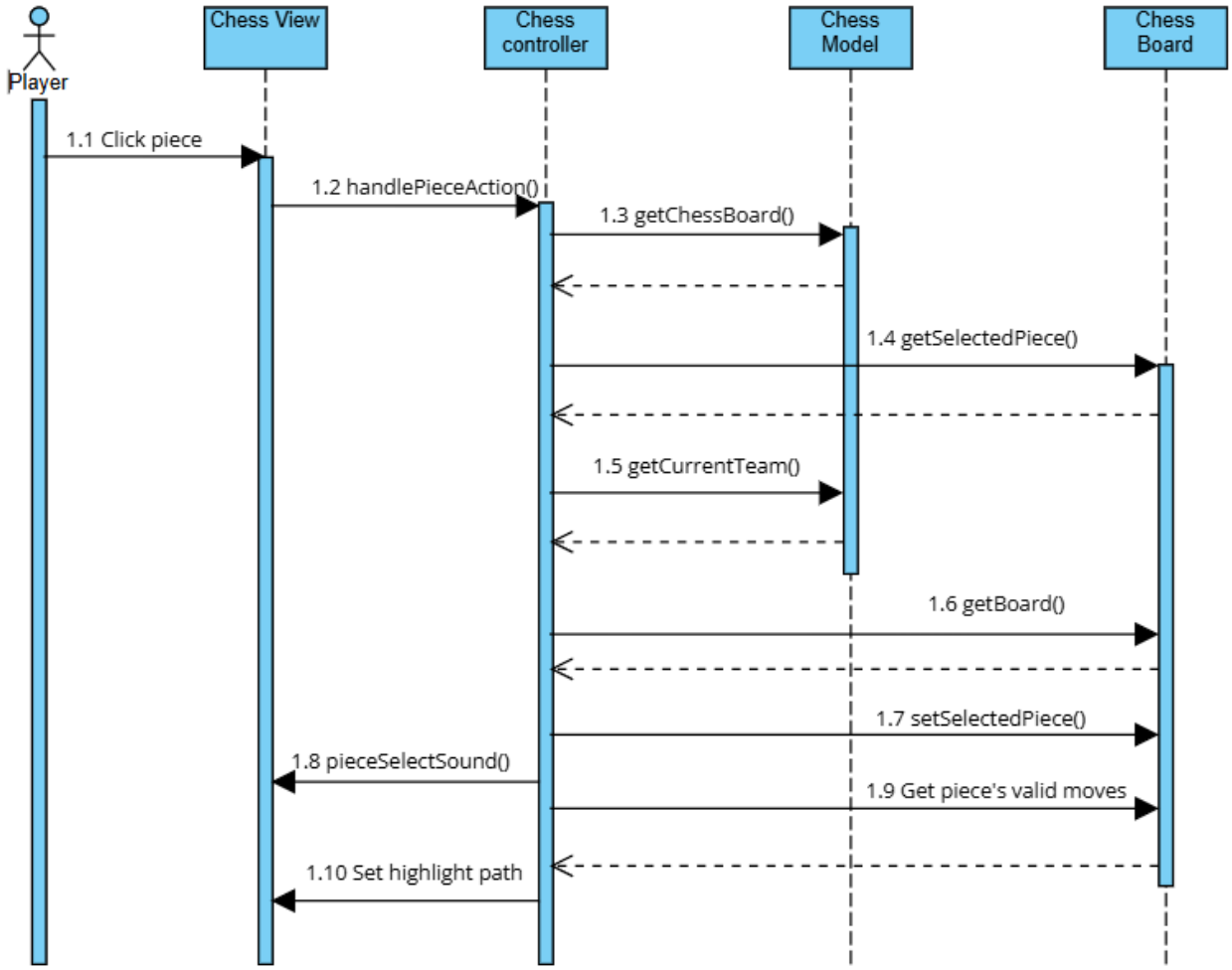
Start & Restart Game



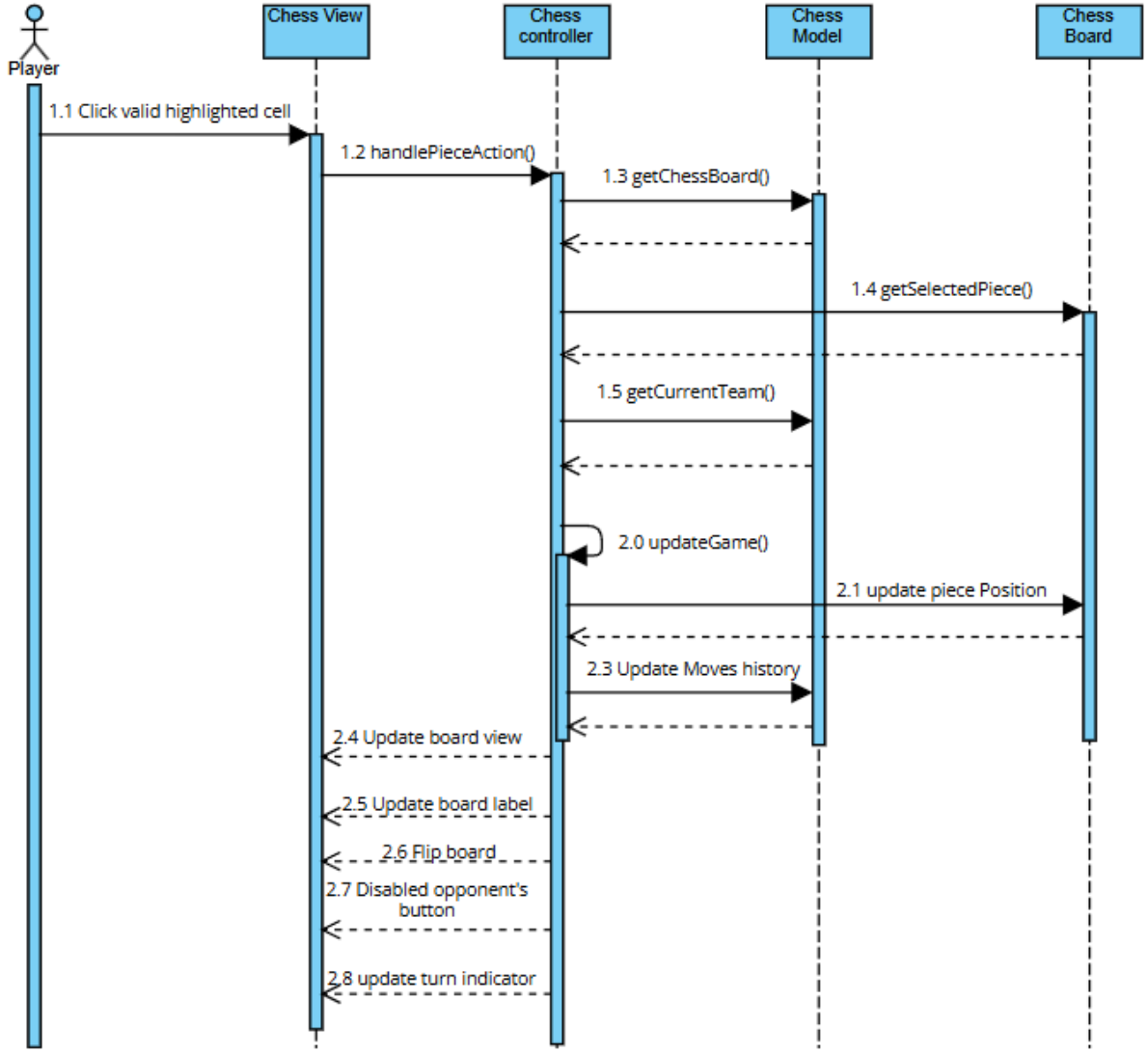
Exit Game


Player
ChessView
ChessController
System
1.1 Click "Exit" button
1.2 notify for exit
1.3 System.exit()

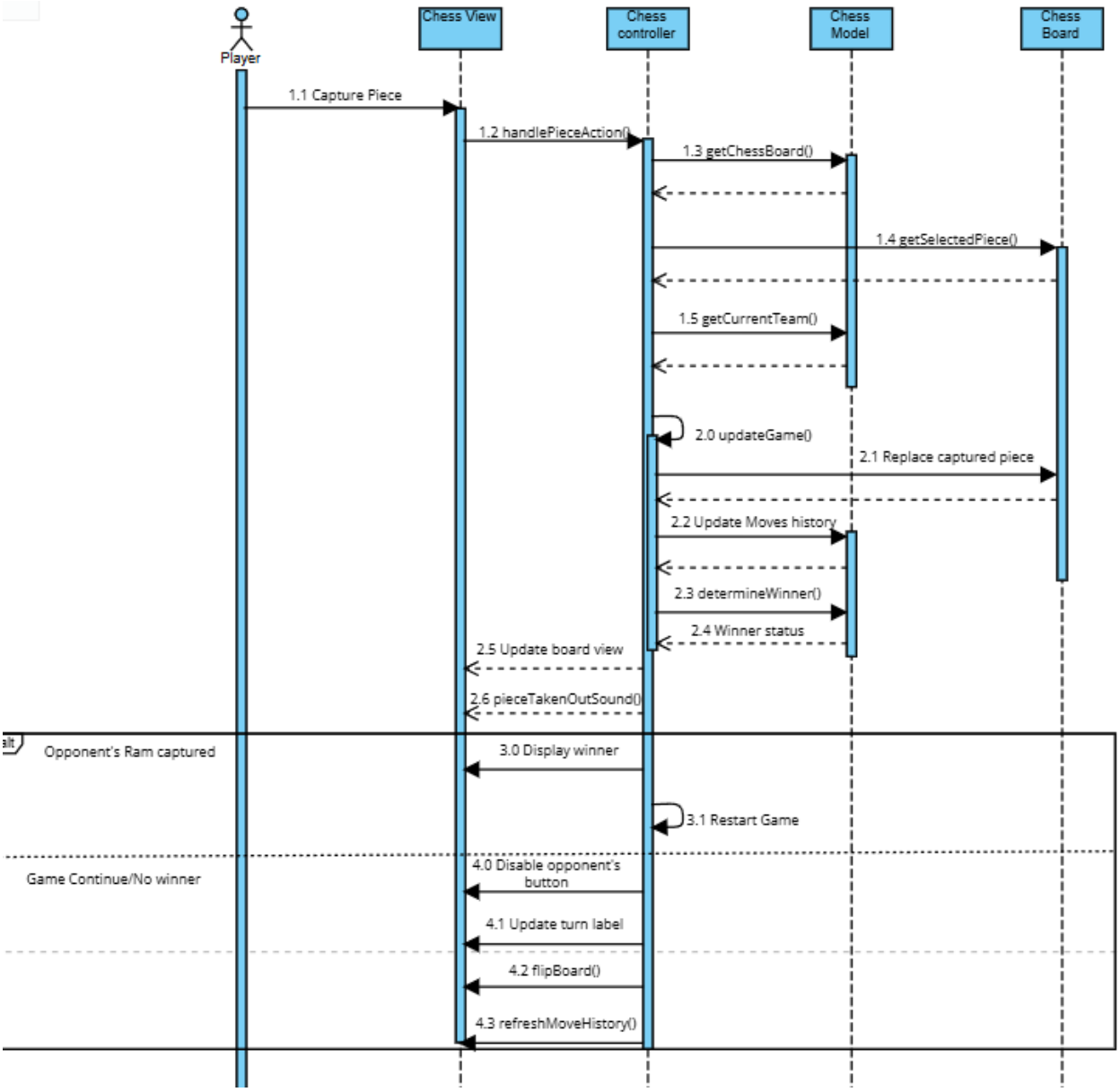
Click piece


Player
Chess View
Chess controller
Chess Model
Chess Board
1.1 Click piece
1.2 handlePieceAction()
1.3 getChessBoard()
1.4 getSelectedPiece()
1.5 getCurrentTeam()
1.6 getBoard()
1.7 setSelectedPiece()
1.8 pieceSelectSound()
1.9 Get piece's valid moves
1.10 Set highlight path

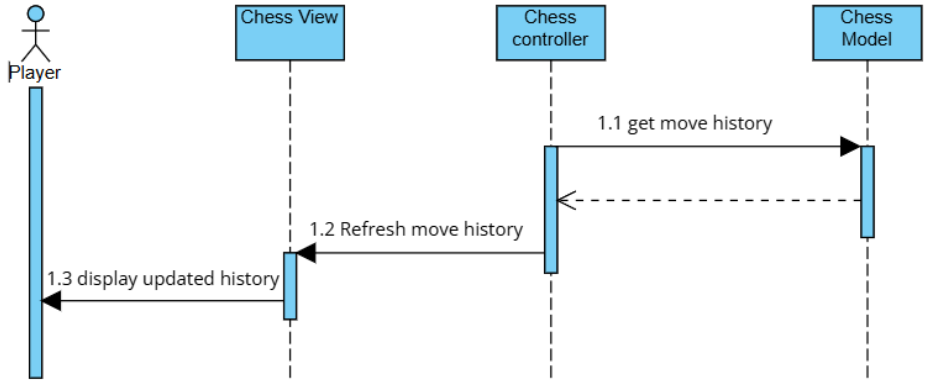
Move Piece


Player
Chess View
Chess controller
Chess Model
Chess Board
1.1 Click valid highlighted cell
1.2 handlePieceAction()
1.3 getChessBoard()
1.4 getSelectedPiece()
1.5 getCurrentTeam()
2.0 updateGame()
2.1 update piece Position
2.3 Update Moves history
2.4 Update board view
2.5 Update board label
2.6 Flip board
2.7 Disabled opponent's button
2.8 update turn indicator

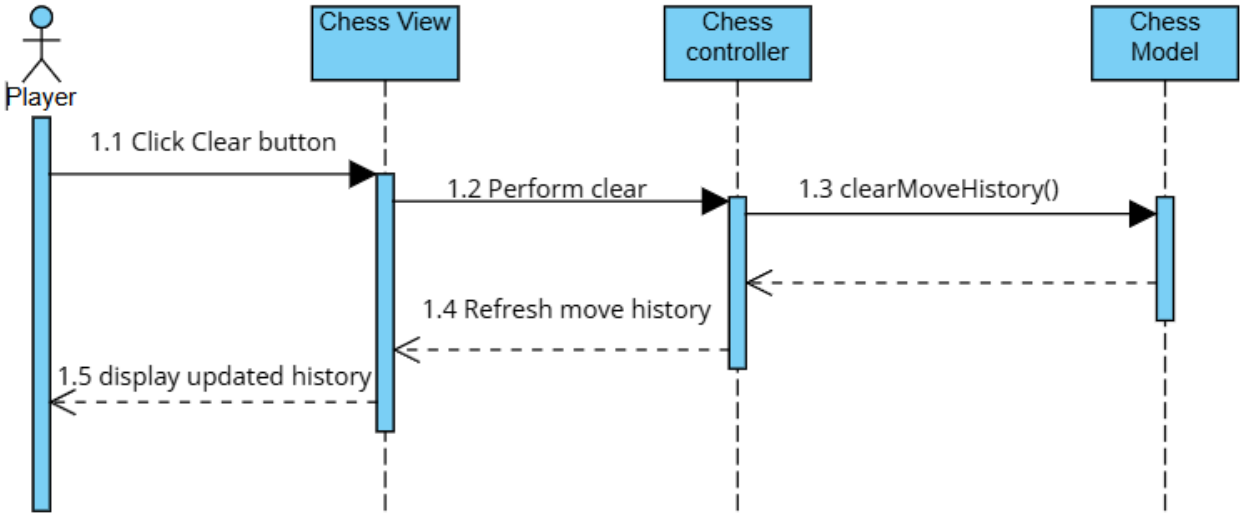
Capture Piece



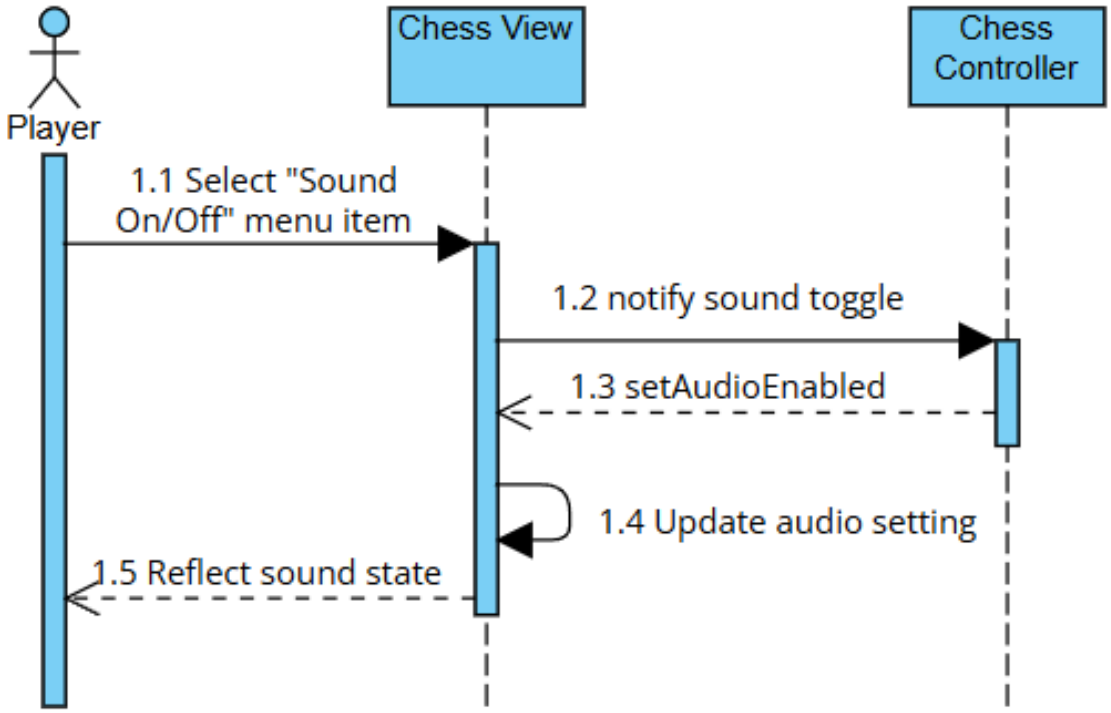
View Move History



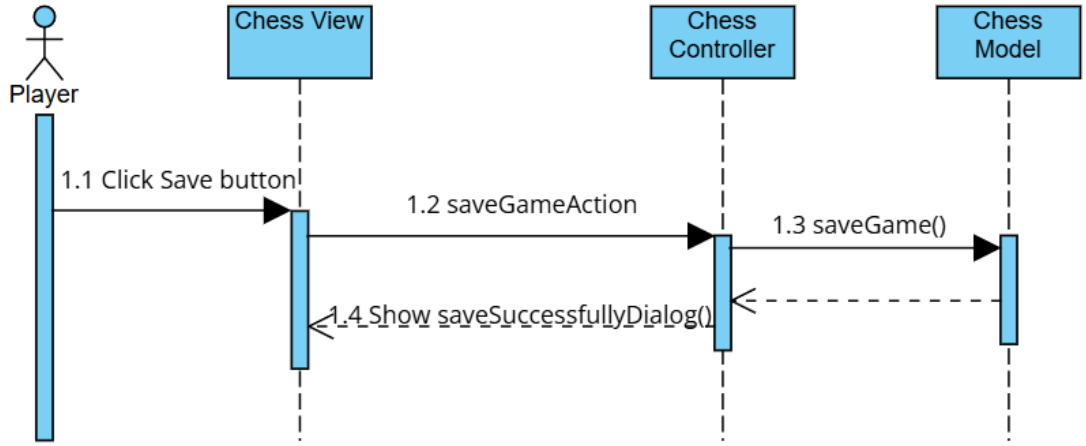
Clear Move History



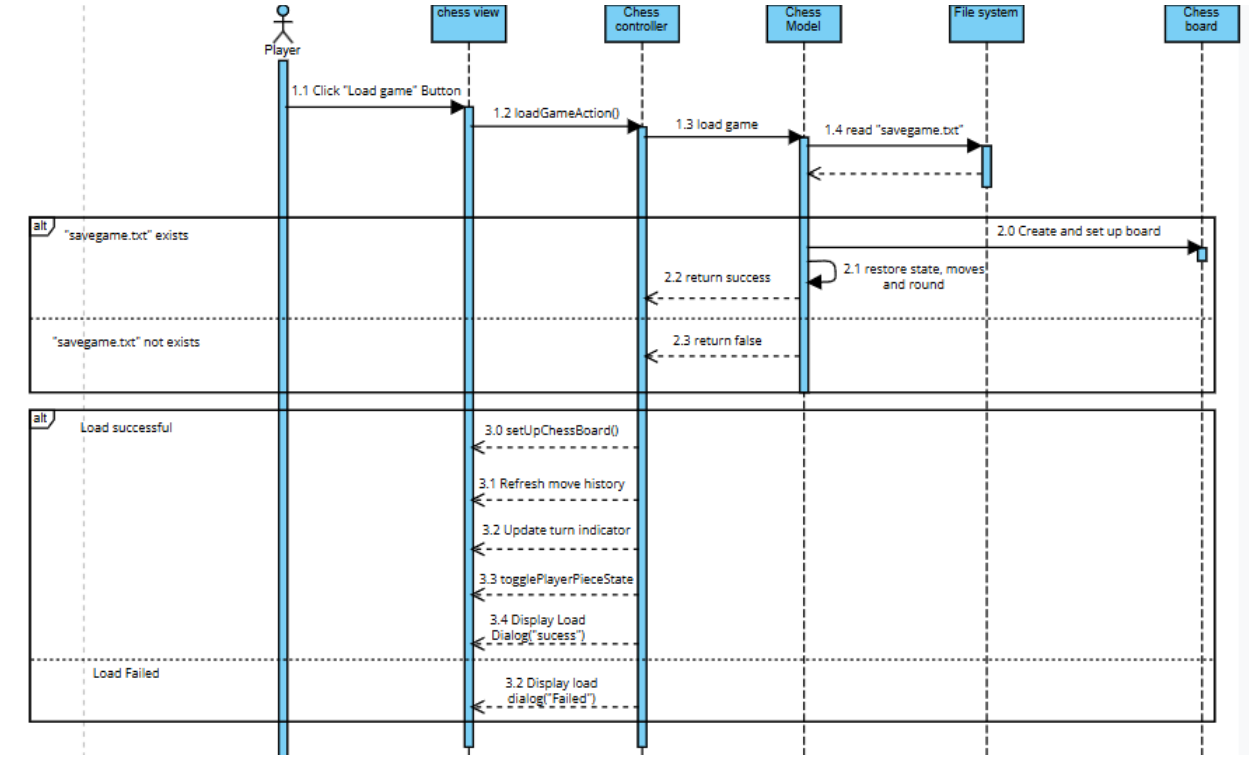
Toggle Audio On/Off



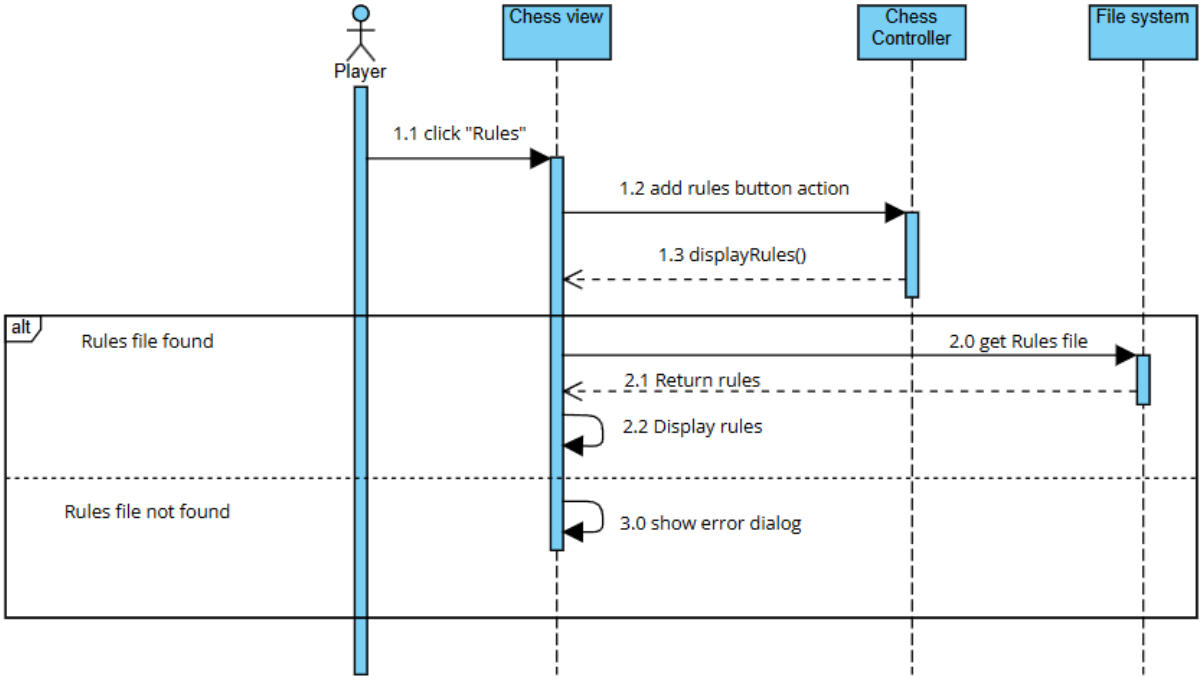
Save Game



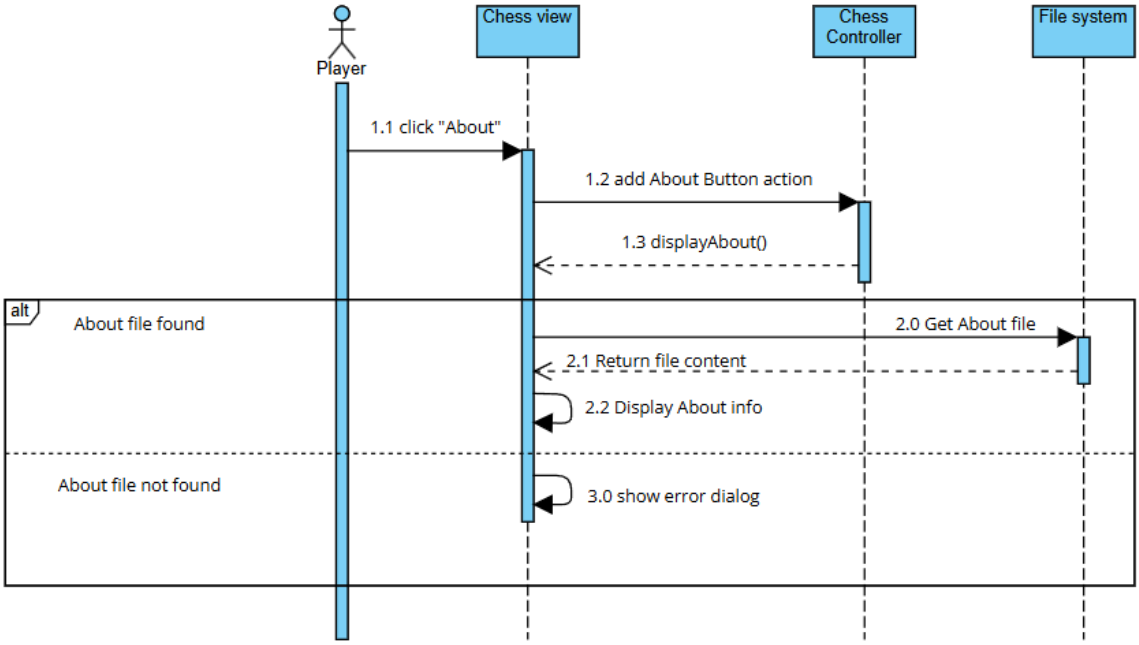
Load Game



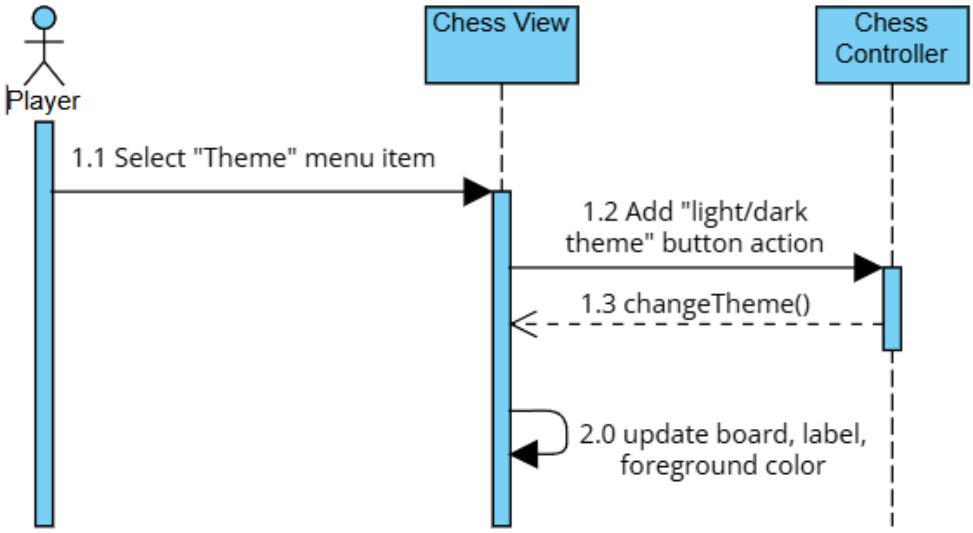
View Rules



View About Page



Switch Theme



User Documentation

Introduction

Welcome to **Kwazam Chess**! Kwazam Chess is a unique PVP chess game, built using Java, that offers a user-friendly interface for players to fully immerse themselves in the strategic and captivating challenge of the game.

Getting Started

1. Running the Game

1. **Download the game folder** to your local machine.
2. Open you terminal and navigate to the downloaded folder.
3. Follow the instructions in "**Compile & Run Instruction**" section to compile and start the game.

2. Taskbar Options

- Game:
    - Start a new game
    - Restart the game
    - Exit the game

- Move:
    - Save the current game
    - Load saved game

- Setting:
    - Switch between light and dark themes
    - Toggle audio on or off

- Help:
    - Access rules and additional game information

How To Play

1. Objective

- Capture your opponent's **Sau** to win the game!

2. Basic Controls

- **Select a Piece**: Click on a piece to select it.
- **Move a Piece**: Click the highlighted square to move the selected piece to that position.

Rules Overview

1. Each player has **10 pieces**:
    a. 5 Ram
    b. 2 Biz
    c. 1 Xor
    d. 1 Tor
    e. 1 Sau

2. Pieces move in specific patterns:

Pieces	Patterns
The Ram Piece 	• The Ram can **move forward one step at a time**. • When it reaches the end of the board, it turns around and starts heading back the other way. • It **cannot skip** over other pieces.

The Biz Piece 	• The Biz moves in a **3x2 L-shape** in any orientation. • It is the only piece **capable of skipping** over other pieces.
The Tor Piece 	• The Tor **moves orthogonally** (up, down, left, or right), any number of steps. • It **cannot skip** over other pieces. • **Special Rule:** After two turns (one blue move and one red move), the **Tor** transforms into the **Xor** piece.
The Xor Piece 	• The Xor **moves diagonally** (in any of the four diagonal directions), any number of steps. • It **cannot skip** over other pieces. • **Special Rule:** After two turns (one blue move and one red move), the **Xor** transforms into the **Tor** piece.

The Sau Piece	
	• The Sau can **move only one step in any direction** (horizontally, vertically, or diagonally). • The **game ends** when the Sau is captured by the opposing side.

3. The game ends when
    a. **Victory**: Capture your opponent's **Sau**.

Support

Contact:

Ow Ka Sheng - 1211108820@student.mmu.edu.my

Yoong Tzer Shih - 1211108421@student.mmu.edu.my

Harold Goh - 1211108866@student.mmu.edu.my

Lau Zi Herng - 1211108444@student.mmu.edu.my