Software Engineering Group Project - Design Specification

Author(s):	Gwion Hughes [gwh18@aber.ac.uk] Shaun Royle [shr27@aber.ac.uk] Tyler Lewis [tyw1@aber.ac.uk]
Config Ref:	Design-Spec-GP9
Date:	11 May 2023
Version:	2.0
Status:	Release

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

CONTENTS

1. INTRODUCTION

1.1 Purpose of this document

The purpose of this document is to educate on the design of our system. This includes the relationships between components and classes in our system and how they achieve the functional requirements set out in the project brief.

1.2 Scope

This document contains the details of all our classes, the relationships between them, and how they come together to form a program that achieves the goals of the functional requirements.

This document should be read by all project members. It is recommended that the reader is familiar with the following:

- UI-Spec-Docu-GP9 - UI specification [1]
- SE.QA.05 - Design Specification Standards [2]
- SE.QA.RS-CS22120 - Requirements Specification [3]

1.3 Objectives

The objective of the document is to inform about how the program works by detailing the functionalities and relationships of the classes.

2. DECOMPOSITION DESCRIPTION

2.1 Programs In System

Our system is designed with only one program, and runs as a single JVM process. The system contains classes that handle the logic of the game and control the graphics display. The logic is handled by Java classes and the graphics are handled by JavaFX classes alongside CSS stylesheets.

2.2 Significant Classes In Each Program

Board

The board class is responsible for logically representing the current state of the board. Using an 8 by 8, 2-dimensional array of piece objects it can represent any board state. It is also responsible for the internal manipulation of the pieces. Furthermore, the class is also responsible for generating and updating its own variation of Forsyth Edwards Notation which is a method of representing the state of a board as a string; this string can be used to load and save board states.

Piece

The piece class is responsible for logically representing the pieces stored in the board class. It stores the piece type, colour, location, legal moves, and whether the piece has moved during the course of the game.

Game

The game class is the main class which is responsible for allowing the UI to communicate with the backend classes, obtaining the fen string and returning it to the interface class, asking the moveCalculator if the board is in check or not and returning it to the Interface class.

GraphicsHandler

This class contains the primarystage for the scenes. This is the application window. It controls the logic of switching between different scenes and messaging ui input to the backend and instruction forwards to the front-end.

PlayScreen

This JavaFX class contains the Screen for the players to interact with the chessboard. It visually displays the turn of the player and provides options for quitting, resigning, offering a draw, and stepping through the history of the game.

LoadScreen

This JavaFX class contains the Screen for the players to choose a save. It displays a selection of games finished, or unfinished, and passes to the PlayScreen if a particular game is selected.

Chessboard

This JavaFX class contains the logic for graphically representing the state of the board. For example, it can take a Forsyth Edwards notation of the board and display the pieces in the correct board positions, as well as visually indicate valid moves for a selected piece.

MoveCalculator

This class is responsible for taking a Board object and calculating the legal moves for all the pieces of a particular colour, by first calculating the possible next moves of the pieces of the opposite colour and then building a ArrayList of threatened squares using these moves, using this arraylist it is able to check if the player is in check and will use this in order to make sure that no moves that will place the player in check will be added to a piece's legal, moves the MoveCalculator class calculates the piece's legal moves as arrayLists of Coordinate objects and then sets the piece's legal moves to this ArrayList. Using this it is able to check if the player is in checkmate by checking if there are no legal moves of any of the player's pieces.

Log

This class is responsible for tracking the board state after each turn. It is also responsible for saving, loading and managing the text files that hold the record of each game.

When a new game is made then a new log object is also made. When a new log object is made then a new text file is also made in the project. After each turn, the log object writes the FEN string that represents the current board state to a new line in the text file. When the game is saved and exited, the file is left in the unfinishedGames folder. When the game is exited without save, the file is deleted from the unfinishedGames folder.

When a game is finished it is moved to the finished games folder. This folder provides the files for the view finished games functionality. When viewing a finished game, if the user chooses to exit without saving then the game will be deleted from the finished games folder.

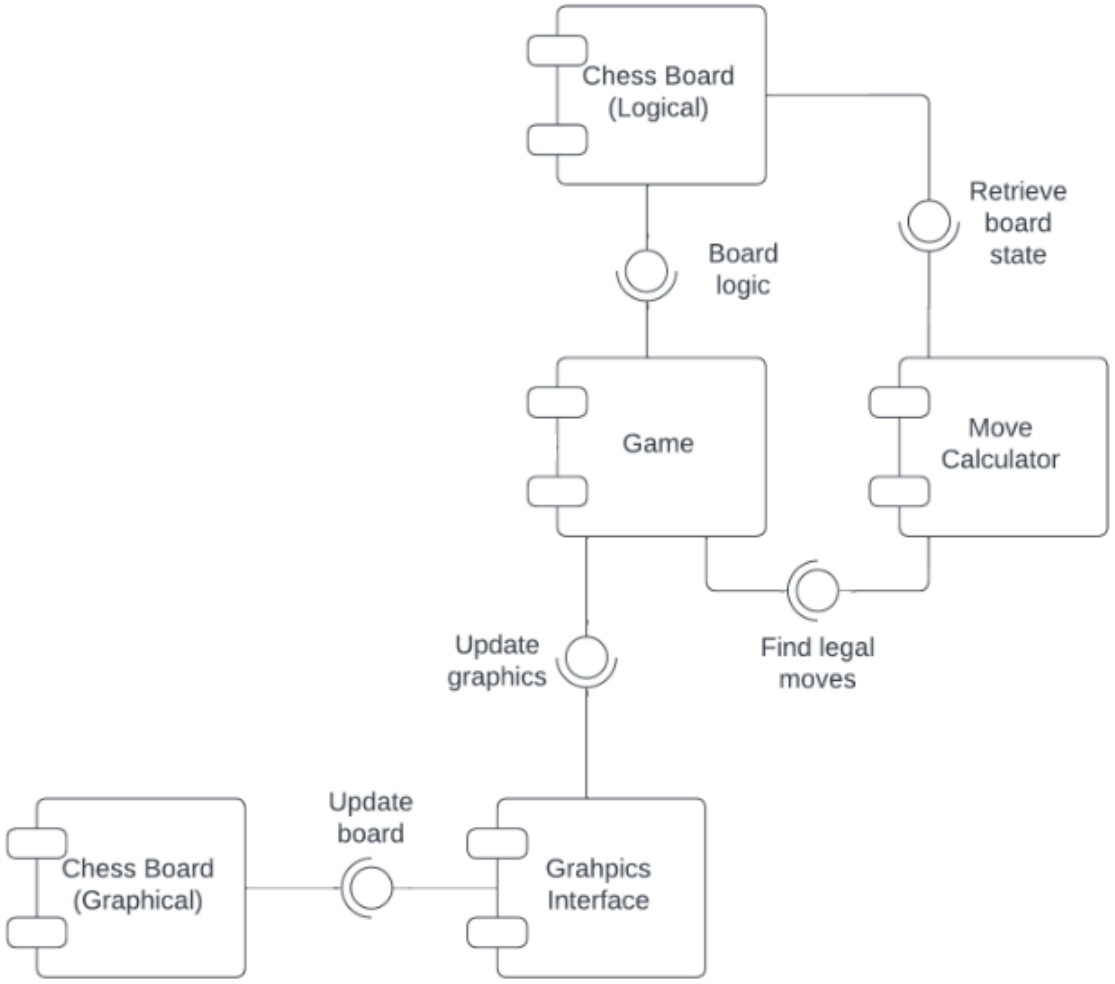
If the program is loading a previous game, then a new text file will not be created when the new log object is made. Instead, it will look for the text file in the finished game folder.

2.3 Mapping From Requirements To Classes

Requirement	Classes
FR1	GraphicsHandler, StartScreen, PlayerNameScreen, PlayScreen
FR2	Board
FR3	Chessboard, board, Tile, TileGraphicsLoader, PlayScreen, Game
FR4	Chessboard, Tile, Game
FR5	Chessboard, MoveCalculator, board, Tile, Game
FR6	Chessboard, MoveCalculator, Tile, Game
FR7	PlayScreen, MoveCalculator, ChessBoard, Tile, Game
FR8	PlayScreen, Game
FR9	PlayScreen, Game
FR10	StartScreen, LoadScreen, PlayScreen, Chessboard, Board, Log, Game
FR11	StartScreen, LoadScreen, PlayScreen, Chessboard, Board, Log

3. DEPENDENCY DESCRIPTION

3.1 Component Diagram



4. INTERFACE DESCRIPTION

This section is a list of all the classes that make up our system and the publicly listed methods that are called to perform interactions with our chess tutor.

4.1 GraphicsHandler Interface Specification

- Type: Public, standard class
- Extends: Application // This is because it contains the main function
- Public Methods:
    - void:start(Stage stage)
    - void:main(String[] args)
    - void:click(int column, int row):Notifies backend of the position selected by the user.
    - Int getTurnNumber():Returns the turn number of the game.
    - String getPreviousFEN(int turn):Returns the FEN string of the previous turn.
    - void:toMenu():Sets the stage's primary screen to be the menu screen.
    - void:toPNScreen():Sets the stage's primary screen to the player name screen.
    - void:toNewChessboard(String whiteName, String blackName):Switches to playscreen with a new game.
    - void:loadFGames():Display the loadscreen with a selection of finished games
    - void:loadUFGames(): Display the loadscreen with a selection of unfinished games
    - void setGameFromSave(String filename, boolean isFinished):Passes the filename of the saveGame to set up the unfinished game
    - void requestPromotion(int n):displays the piece promotion menu and requests a piece promotion from the back end
    - void updateGameOver(char c): Lets the back end know if a resignation/draw has occurred so the log can be updated accordingly.
    - void deleteGame(): Lets the back end know the user has requested not to save the game so it can be deleted.

4.2 LoadScreen Interface Specification

- Type: Public, standard class
- Extends: Nothing
- Public Methods:
    - Void LoadScreen(Interface anInterface): Constructor. Constructs Screen.
    - Void setLabel(String newLabel):Shows whether the saves show are from finished or unfinished games
    - Void populateSaveBar(ArrayList<String>, boolean): Display a list of saves as buttons
    - Void requestSave(int num): Requests to view/play a selected game
    - Scene getScene(): returns the LoadScreen Scene

4.3 PlayerNameScreen Interface Specification

- Type: Public, standard class
- Extends: Nothing
- Public Methods:
    - PlayerNameScreen(Interface anInterface): constructor
    - Scene:getScene(): getter for Scene
    - backToMenu(): return to main menu
    - forwardsToNewGame(String whiteName, String blackName): start a new game with entered names as players

4.4 PlayScreen Interface Specification

- Type: Public, standard class
- Extends: Nothing
- Public Methods:
    - PlayScreen(Interface anInterface): Constructor
    - Scene getScene():Returns the scene
    - Void constructPlayScreen():Called in the constructor to create the screen
    - StackPane createDashboard():Creates dashboard to display game information and buttons for resignation and offering a draw.
    - Void updatePlayerDashboard(char Player):Updates the dashboard to display the active player.
    - Void alertPressedTile(): Message to backend the tile that was pressed
    - Void setWhitePlayerName(String name): setter for white player's name
    - Void setBlackPlayerName(String name): setter for black player's name
    - Void updatePlayScreen(String boardNotation): Send board notation to the chessboard to update
    - offerDraw(): Creates a container to offer a draw to the next player
    - resign(): Current player resigns and ends the game
    - Void offerPromotion(): Displays the promotion menu in the dashboard
    - Void incrementThroughLog(): go forwards through the game history
    - Void decrementThroughLog(): go backwards through the game history
    - Void areYouSure(): double check the player would like to quit
    - Void gameOverOverlay(char c):Disables chessboard and updates dashboard to show victor.
    - Void highlightTiles(ArrayList<Coordinate> vTiles, ArrayList<Coordinate> checkTiles):Highlights the valid tiles for a piece move
    - Void setLogDisabled():Disables or Enables the buttons to view next and previous move
    - Void setGameFinished: Setter for gameFinished field.

4.5 StartScreen Interface Specification

- Type: Public, standard class
- Extends: Nothing
- Public Methods:

    - StartScreen(): Constructor
    - Scene getStartScreen(): getter for the scene

4.6 Chessboard Interface Specification

- Type: Public, standard class
- Extends: Nothing
- Public Methods:
    - Void Chessboard(PlayScreen playScreen): Constructor
    - Void click(int column, int row): message button press to the backend and highlight pressed tile
    - Void updateBoard(String boardNotation): use FEN to update graphical piece positions
    - GridPane getChessBoard(): getter for the chessboard gridpane
    - Void highlightTiles(ArrayList<Coordinate> validT, ArrayList<Coordinate> checkT): Takes a series of coordinates and changes their colour
    - Void disableChessBoard(boolean b):Disables the chessboard

4.7 Piece Interface Specification

- Type: public, standard class
- Extends: Nothing
- Public Methods:
    - piece(char colour, vector2 position, char type): Constructor
    - Char: getColor(): returns the pieces' colour
    - Coordinate: getPosition() returns the pieces' position as a vector2
    - Void: setPosition(): sets the piece's position to a vector
    - ArrayList<Coordinate>: getPossibleMoves(): returns the ArrayList of Vector2 that represent the piece's moves
    - Void: addMove(): add's a vector2 to the arrayList of possible moves
    - Char: getType(): returns the type of the piece as a char
    - Boolean: hasMoved(): returns a boolean that indicates if the piece has moved or not
    - Void: setHasMoved(): sets whether the piece has moved or not
    - Void: clearMoves(): Clears the piece's internally tracked valid tiles to move to.

4.8 Coordinate Interface Specification

- Type: public, standard class
- Extends: Nothing
- Public Methods:
    - Coordinate(): constructor
    - Coordinate(int x, int y): constructor
    - String getCoordinateAsBoardNotation(): returns the coordinate as a chess board position notation

4.9 MoveCalculator Interface Specification

- Type: public, standard class
- Extends: Nothing
- Public Methods:
    - moveCalculator(char player, board board): constructor
    - Void findLegalMovesForPlayer(boolean opponentPlayer): finds the legal moves for the player or opponent player
    - Boolean isPlayerInCheck(): returns true if the player is in check
    - Boolean isPlayerInCheckMate(): returns true if the player is in checkmate
    - Void printCheckMap(): prints the list of every unsafe square for the current player's king, used for debugging.

4.10 Game Interface Specification

- Type: public, standard class
- Extends: Nothing
- Public Methods:
    - game(String boardState, String fileName, boolean load): constructor
    - game(): constructor
    - Void createGame(): creates a new logical gameboard with a new log file for that game.
    - Void move(): the general game loop of the game responsible for checking if the selected piece is valid and responsible for communicating to the backend which piece to move to where and whether the move is legal.
    - Void updateBoard(): called after a move is made updates which player's turn it is updates the log file to reflect on the new move made, calculates the new legal moves for the new board position, checks if the game is over by checkmate and updates the fenstring if so.
    - Boolean isMoveMade(): returns whether or not a move has been made.
    - Void calculateMoves: uses the move calculator in order to calculate the moves for a particular board state, passing the game class's board after each move is made.
    - Boolean isGameOverByCheckMate(): returns whether or not the moveCalculator class' isPlayerInCheckMate Method to the front end in order for the front end to reflect on that fact.
    - String gameNotation(): returns the board's Forsyth Edwards Notation String to the front end so that the front end is able to construct the visual board.
    - ArrayList<Coordinate> validTiles(): creates an arrayList which contains the valid tiles of the selected piece, so that the front end can highlight valid tiles, and to check if tiles that are selected for the piece to move to are contained within this array.
    - ArrayList<Coordinate> checkedKing(): returns to the front end the coordinate of the checked king.
    - Void promote(int n): promotes a piece on the board.
    - Boolean isPromotionAvailable() returns whether or not a pawn can be promoted.
    - Void endGame(): method which checks if the game is over by checkmate and updates the log to reflect.

    - Board getGameBoard(): a method created for the testing team to be able to access the board through the game object.

4.11 Board Interface Specification

- Type: public, standard class
- Extends: Nothing
- Public Methods:
    - board(String initializingBoardState): constructor
    - movePiece(piece selectedPiece, vector2 move): move a piece to a position
    - getForsythEdwardsBoardNotation(): return the FEN string
    - getForsythEdwardsBoardNotationArrayIndex(int Index): returns a particular section of the FEN string
    - getPiece(vector2 coordinate) gets a piece at a coordinate on the board
    - getWhiteKingPosition(): returns the white king's position as a vector2
    - getBlackKingPosition(): returns the black king's position as a vector2
    - Coordinate getAvailablePromotion(): returns the position of a piece that is valid for promotion
    - Coordinate setAvailablePromotion(Coordinate position): sets the field availableForPromotion to the position parameter.
    - Boolean canBlackPromote(): returns whether or not black can promote a piece.
    - Boolean canWhitePromote() returns whether or not white can promote a piece.
    - Void promotePawn(int n): promotes the pawn that is valid for promotion to a piece indicated by the UI n = 0 results in a queen, n = 1 results in a rook, n = 2 results in a bishop, n = 3 results in a knight.
    - Void printBoardStateToConsole(): prints an ascii representation of the board to the console.
    - Void clearMoves(): clears the moves for each piece on the board.
    - Void updateFENStringWhenCheckMate(String winningPlayer): updates the final part of the FEN string to indicate which player has won the game, so that you are able to tell which games have been won and which are still going.
    - Int getTurnNumber(): returns the full move number.

4.12 Tile Interface Specification

- Type: public, standard class
- Extends: Nothing
- Public Methods:
    - Tile(Int row,Int column,boolean isWhite,Chessboard chessboard): Constructor
    - Public Button getButton:Returns the button
    - Void setGraphic(ImageView iv): Set the button to display the passed imageview
    - Void clearTile(): clear the tile of any graphics or highlights
    - Void setStyleClass(String styleClass): change the button's styleclass to a new one.
    - Void switchButton(boolean b):Disables/Enables the button

4.13 TileGraphicLoader Interface Specification

- Type: public, standard class
- Extends: Nothing
- Public Methods:
    - TileGraphicsLoader(): Constructor
    - ImageView fetchTileGraphicPiece(char Symbol): return the graphical image representing the chess character

4.14 LoadGameButton Interface Specification

- Type: public, standard class
- Extends: Nothing
- Public Methods:
    - LoadGameButton(String saveName, int saveNumber, LoadScreen loadScreen, boolean isFinished): constructor
    - Button getLoadButton: getter for loadButton

4.15 Log Interface Specification

- Type: public, standard class
- Extends: Nothing
- Public Methods:
    - Log(String fileName): constructor for new games
    - Log(): constructor for load games
    - Void setFinishedGames(boolean finishedGame): Sets the nameOfFolder used for finished or unfinished games
    - Void setFileName(String fileName): Sets the fileName
    - Void updateLog(String FEN): Appends the FEN String to the text file
    - String readLog(int lineNumber): Returns the FEN String at the specified line of the text file
    - Int getNumberOfLines(): gets the number of lines in the text file
    - ArrayList<String> displayExistingGameFiles(): returns a list of files in the existing games folder
    - Void moveFileToFinishedGamesDir(): moves the text file from the unfinished to finished games folder
    - Void replaceLine(int lineNumber, String replacementLine): replaces a line in the text file
    - Void deleteFile(): deletes the file from either the finished or unfinished game folder

4.16 PlayScreenGraphicsLoader Interface Specification

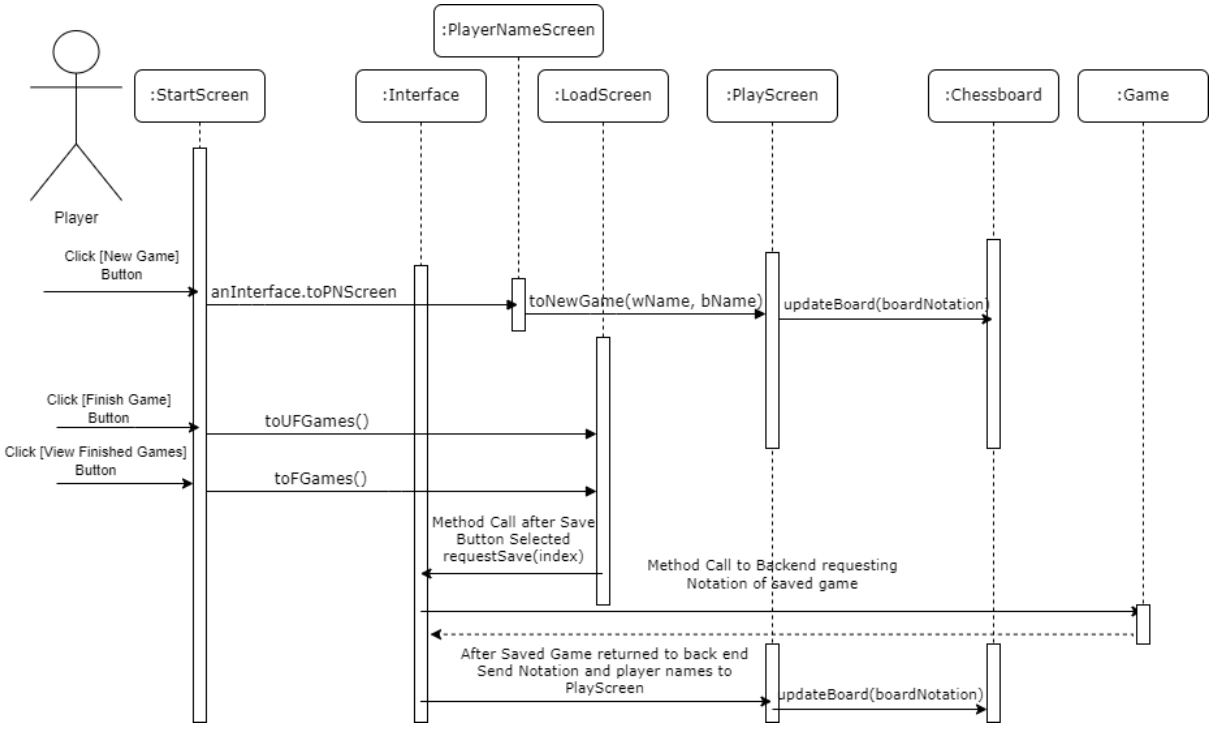
- Type: public, standard class
- Extends: Nothing
- Public Methods:
    - PlayScreenGraphicsLoader: constructor

- Image getImage(char): Return an Image loaded in from files corresponding to the char.

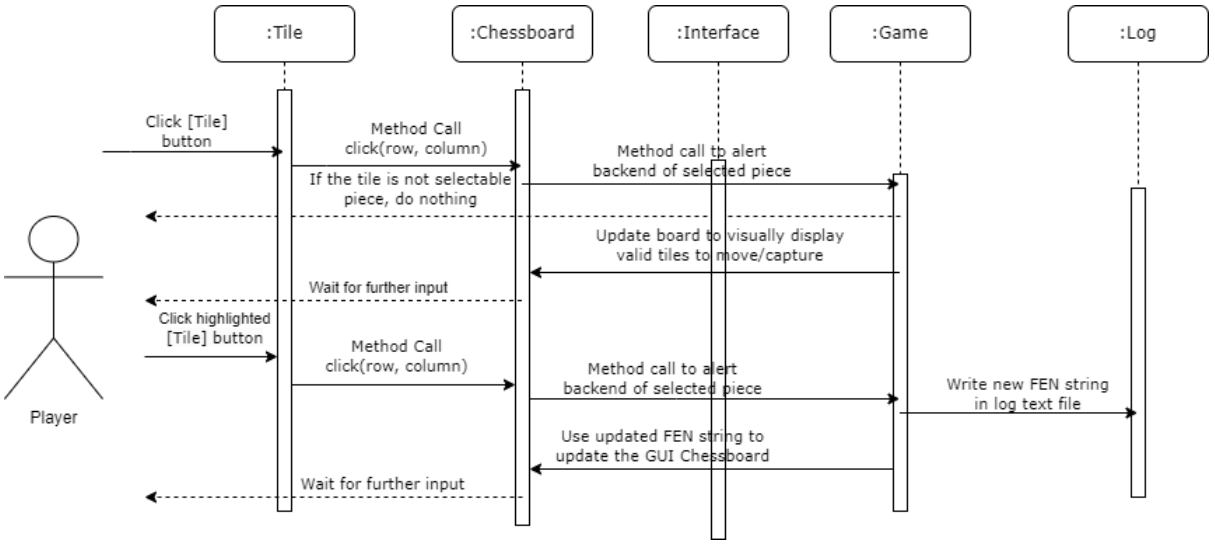
5. DETAILED DESIGN

5.1 Sequence Diagrams

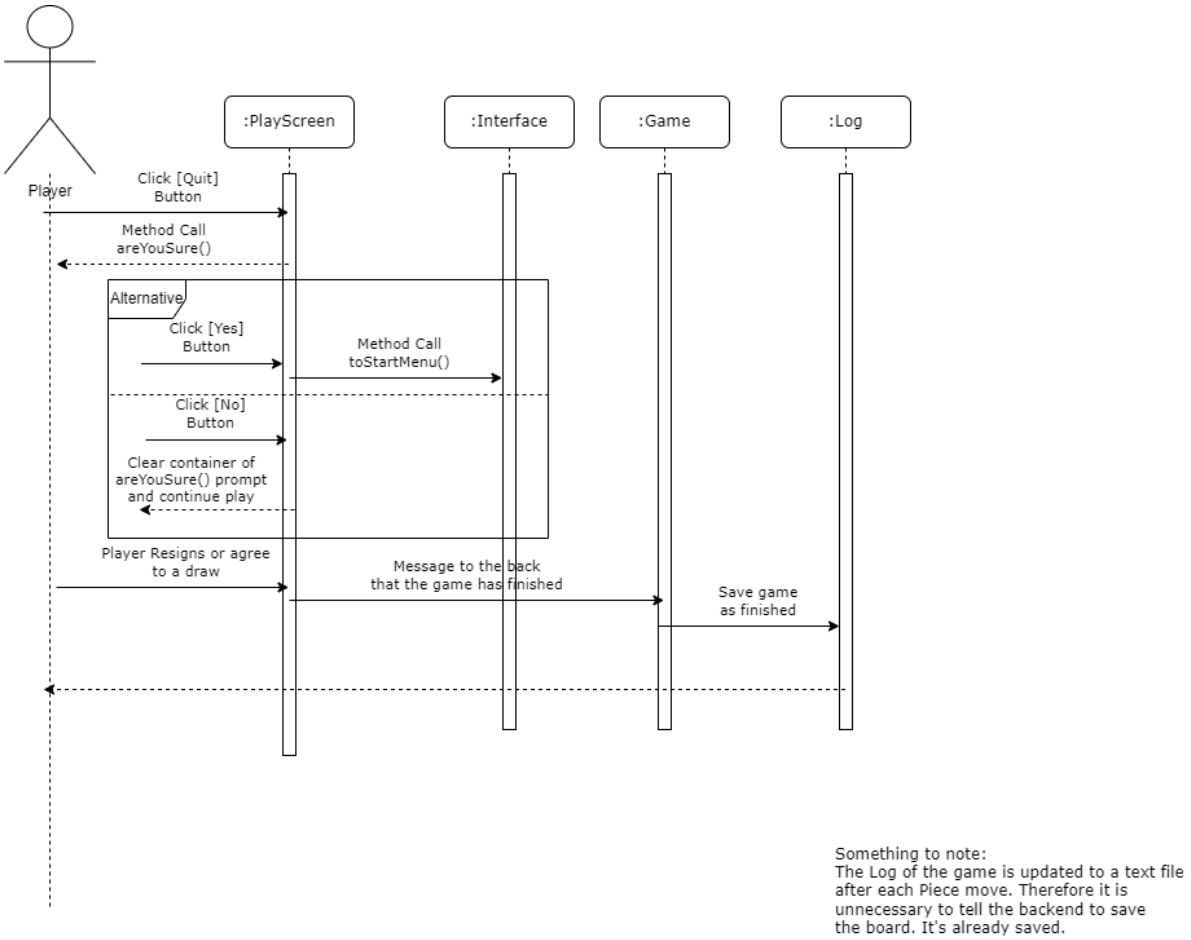
5.1.1 UC-0.0 Launching Menu



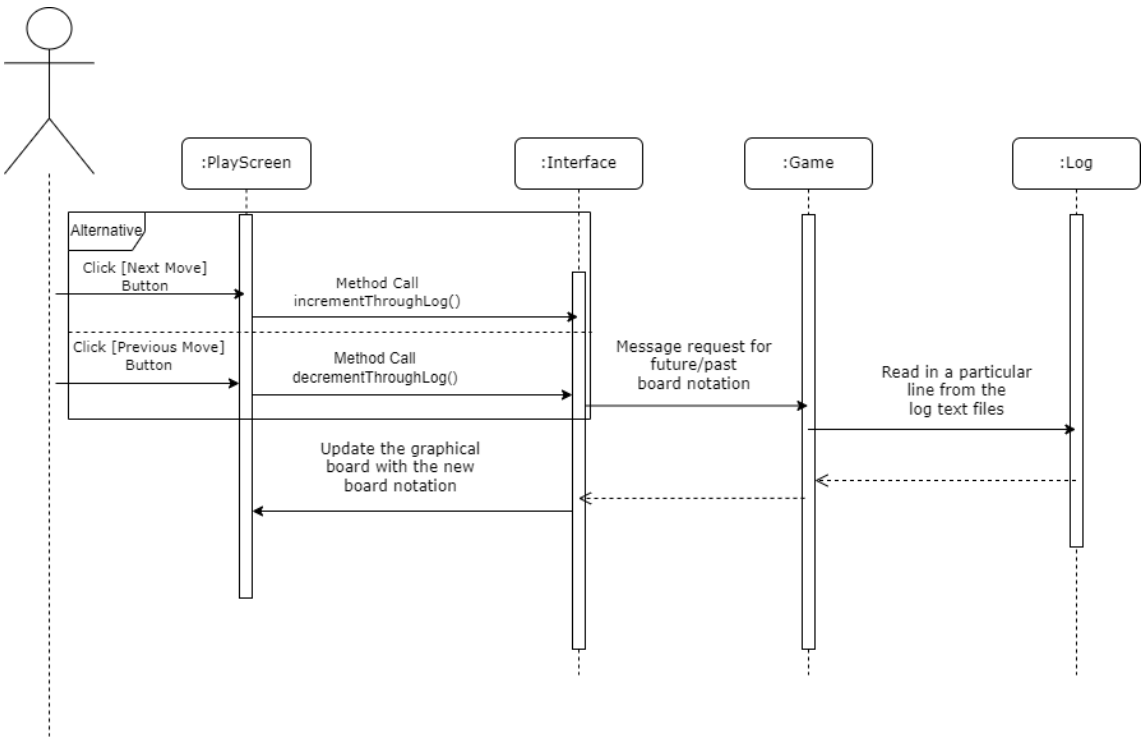
5.1.2 UC-1.0 Selecting a Piece and moving a Piece



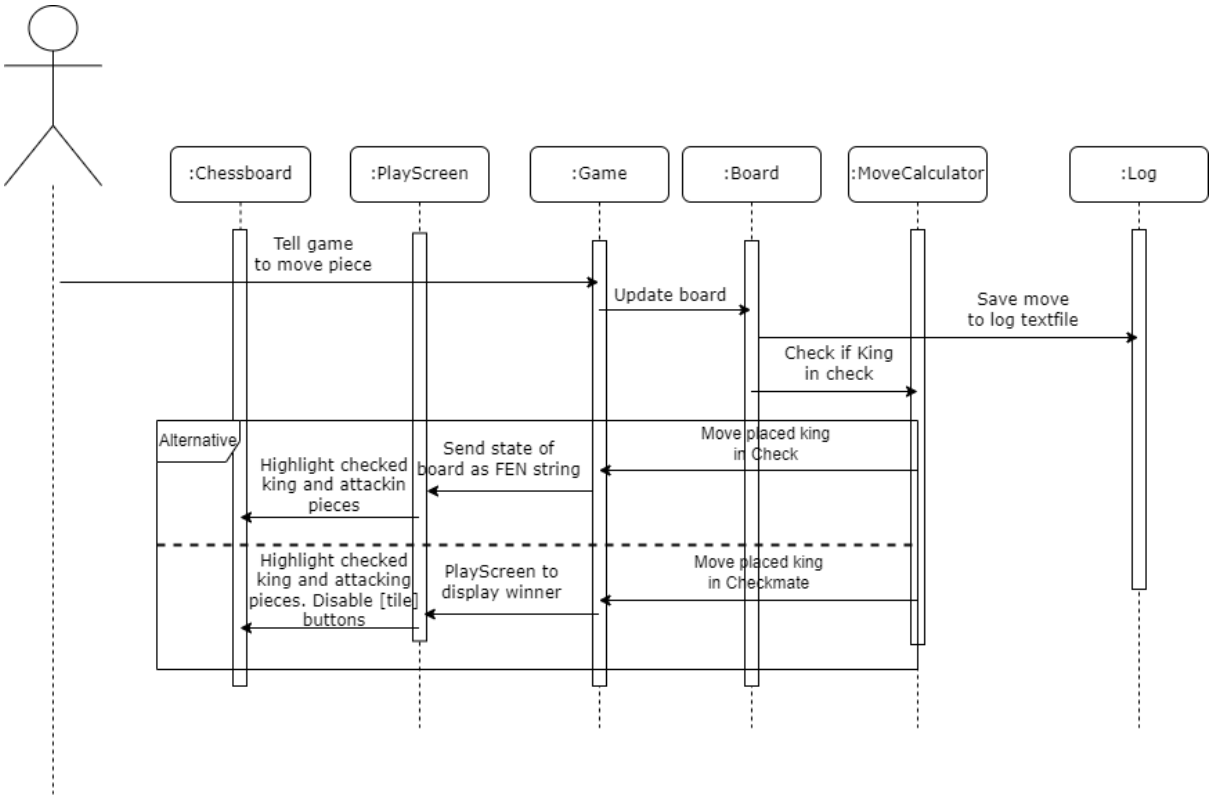
5.1.3 UC-2.0 Quitting/Resigning/Offering Draw



5.1.4 UC-3.0 Reviewing a Previous Game



5.1.5 UC-4.0 End of Game



5.2 Significant Algorithms

5.2.1 Setting Up the Board Using Forsyth Edwards Notation

Our Program is using Forsyth Edwards notation[4] in order to set up board states, Forsyth Edwards notation is a String of text used to represent a board state in a chess game here is an example:

“rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1 -”

This FEN string is the representation for the start state of a game of chess.

You can store multiple of these FEN strings in order to log a whole chess game for example:

rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1 -
rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq e3 1 1 -
rnbqkb1r/pppppppp/5n2/8/4P3/8/PPPP1PPP/RNBQKBNR w KQkq - 2 2 -
rnbqkb1r/pppppppp/5n2/8/4P3/2N5/PPPP1PPP/R1BQKBNR b KQkq - 3 2 -
r1bqkb1r/pppppppp/2n2n2/8/4P3/2N5/PPPP1PPP/R1BQKBNR w KQkq - 4 3 -
r1bqkb1r/pppppppp/2n2n2/8/2B1P3/2N5/PPPP1PPP/R1BQK1NR b KQkq - 5 3 -

We are using a modified variation of Forsyth Edwards notation with the one extra character at the very end of the string to determine if a game has finished or not “-” represents that the game has not finished, “w” representing that the game has been won by white, “b” representing if the game has been won by black, and “d” which represents that the game resulted in a draw.

The board is given the Forsyth Edwards board notation as a string, then the String is split on the delimiter‘ ’ (a space character) into an array of size 6:

forsythEdwardsBoardNotation.split(‘ ’, 6);

In order to set the board array to an accurate representation of that in the Forsyth Edwards board notation string, it only needs the first part of the string so we can access the first index in the array in order to obtain this:

forsythEdwardsBoardNotationArray[0];

Using a for loop it then iterates over each character in the string keeping track of the board file and rank.

It checks if the character it has reached is a ‘/’ character, if so then we decrement the rank value, set the file value to 0, then look at the next character.

If the character it has reached is a digit then we must add the digit to the file, and then look at the next character.

Otherwise, we know that we have reached a character that is neither a ‘/’, nor a digit, this means that this character is a character that represents a piece that we must add to the board. First, we check if the character is uppercase or lowercase to distinguish between whether the piece is white, or black. Then we add the piece to the board using the value of the file and rank that we are keeping track of while iterating through the string. Furthermore, if the piece happens to be a king piece we must note the position of the white, or black king, depending on whether or not the character is an uppercase or lowercase character, to this new piece’s position.

After we have added a piece to the board we must increment the file value.

Pseudocode for reading Forsyth Edwards Board Notation:

```
int file = 0; rank = 7;

For each Char in the Forsyth Edwards Notation String {
    if current Char == / {
        reset file to zero;
        decrement rank;
    }

    if current Char is a digit {
        file += current Char;
        look at next Char;
    }

    if current Char is uppercase {
        Piece piece = new Piece( current Char, white );
        add the new piece to the board;

        if current Char == “K” {
            update white king tracker variable;
        }
    } else {
        Piece piece = new Piece( current Char, black );
        add the new piece to the board;

        if current Char == “k” {
            update black king tracker variable;
        }
    }
    file++;
}
```

5.2.2 Calculating The Legal Moves

First we must determine which player is the attacking player, we can do this by asking the board to return the second index in the Forsyth Edwards board notation array:

```
determineCurrentPlayer() {
        Attacking player =
        gameBoard.getForsythEdwardsBoardNotationArray(1).toCharArray()[0];
}
```

In order to calculate the legal moves that the player can make when it is their turn, first we must calculate the squares that are under attack by the opponent in order to determine if we are in check.

The game creates a new move calculator object, giving it the current player and the game board:

```
moveCalcualor movecalculator = new moveCalculator(attacking player, game board);
```

Then it calls the findLegalMoves method with the parameter true, which indicates that we are looking for the opponent's attacking moves. This method will loop through all of the opponent's pieces and then calculate those pieces' legal moves and adds those legal moves to the "check map" this is a list of all attacked squares:

```
moveCalculator.findLegalMoves(true);
```

Now that it knows the attacked squares, it can begin calculating the legal moves for all the attacking players pieces, then for every move it will play that move on a copy of the current board, and then check if the player is in check on that board:

```
isPlayerInCheck() {
        If player = white {
                For every Coordinate in checkMap {
                        If whiteKingPosition == Coordinate in checkMap {
                                Return true
                        }
                }
        } Else If player = black {
                For every Coordinate in checkMap {
                        If blackKingPosition == Coordinate in checkMap {
                                Return true
                        }
                }
        }
        Return false
}
```

```
isMoveSafe(piece, move) {
        New Board = copyOfBoard;
        copyOfBoard.movePiece(move);

        MoveCalculator moveCalculator = new MoveCalculator(piece color, copyOfBoard);
        moveCalculator.findLegalMovesForPlayer(true);
        Return moveCalculator.isPlayerInCheck();
}
```

if they are we know that the move would put, or leave the player in check therefore we know that we cannot add that move as a legal move.

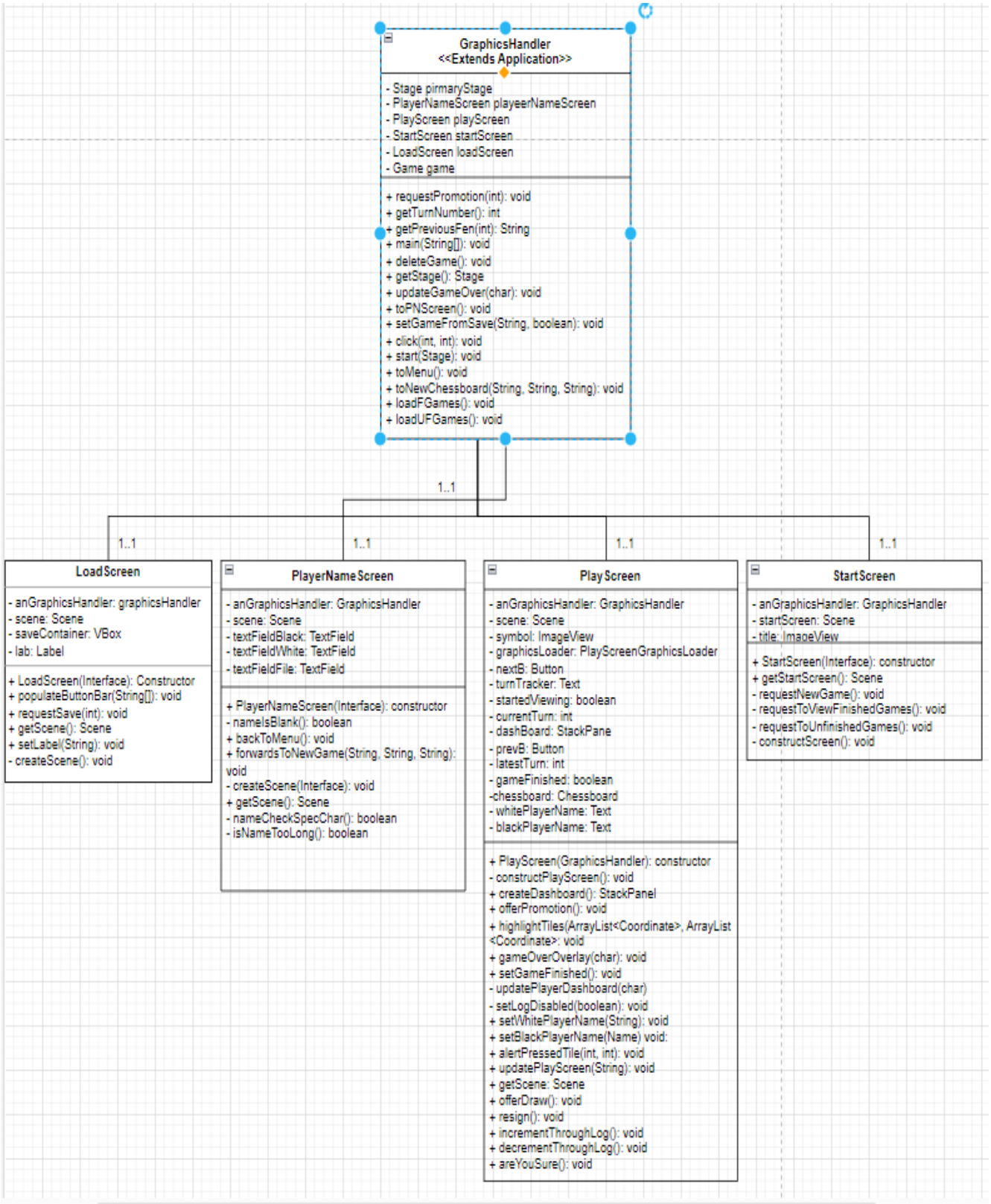
```
moveCalculator.findLegalMoves(true);
```

Checking for checkmate is done in a function which is called at the beginning of each move if it returns true then we know that the game should end and the FEN string should be updated:

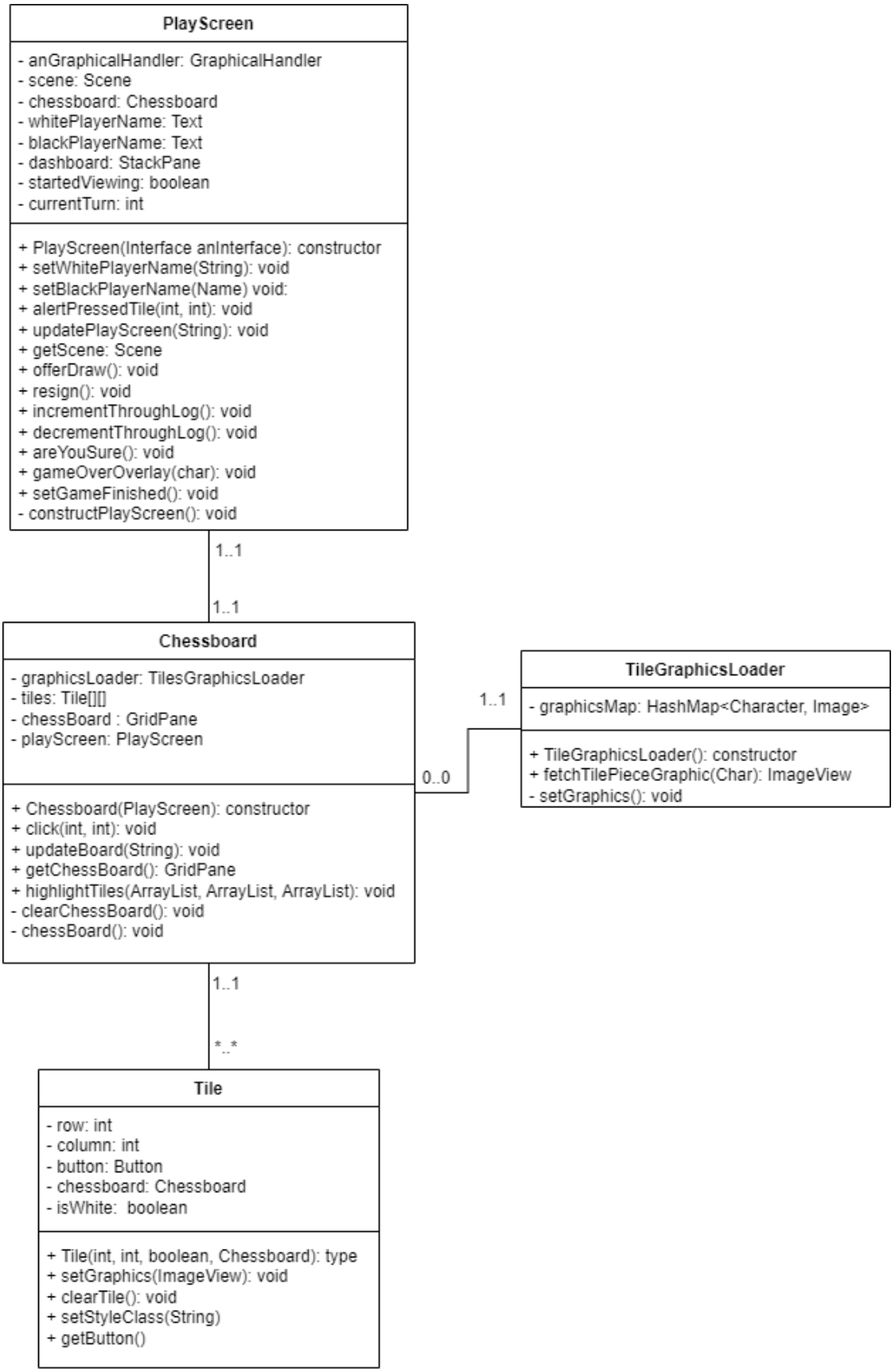
```
isPlayerInCheckMate() {
        Boolean doesPlayerHaveMoves = false;
        For every square on the chess board {
                If the square does contain a piece {
                        If the piece is of correct color && the piece does has moves {
                                doesPlayerHaveMoves = true;
                        }
                }
        }
        Return isPlayerInCheck() && !doesPlayerHaveMoves;
}
```

5.3 UML Class Diagrams

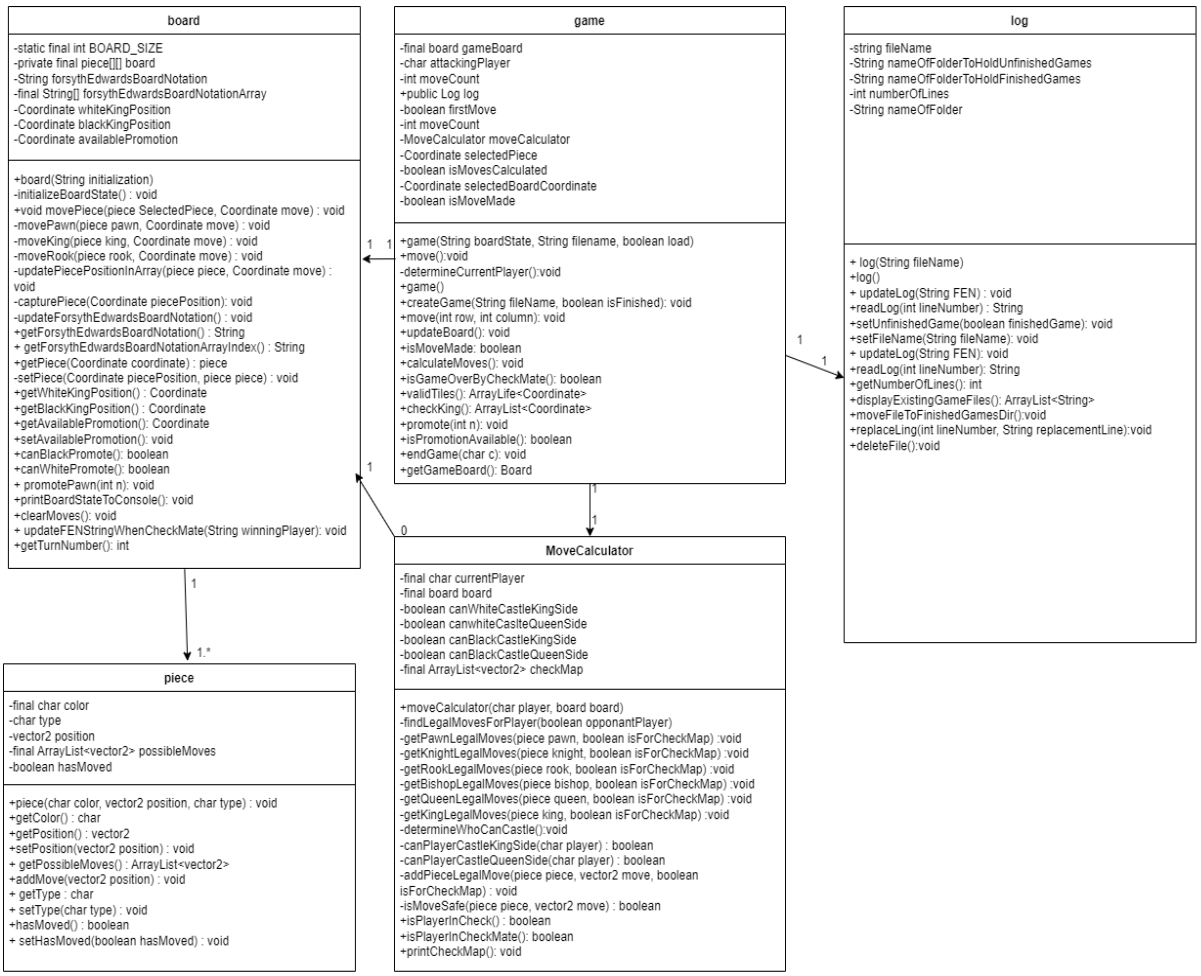
This Class diagram shows the classes that make up the GUI menu and screen navigation



This Class diagram shows the classes that control the graphical representation of a chess tutor game



This UML class diagram shows the logic and functionality associated with creating and maintaining a chessboard.



5.4 Significant Data Structures

5.4.1 internal representation of the board

The board class consists of a 2-dimensional array of piece objects, and methods to manipulate them, such as moving them around the array. It also stores a string representation of the array in the format of Forsyth Edwards notation, when pieces are manipulated the notation is updated to represent the new state of the board.

5.4.2 coordinate class

The coordinate class is a class which stores an X and a Y position which represents the rank and file of the chess board, knowing these axes it can translate them into board notation coordinates.

5.4.3 TileGraphicLoader Class

The class contains a hashmap which maps a char to an Image. The Chessboard has an instance of this class so that it can iterate through a notation of the board, passing the char to the Loader class and returning an image to display on a particular tile. The images are only loaded into the program once this way.

5.4.4 ChessBoard Class

The class contains the Javafx node GridPane which controls how the board is graphically displayed. It also has a 2 dimensional array of Tile Class which contains the actionable button nodes of the chessboard’s gridpane. This decreases lines of code because a gridpane is a trinary tree of nodes and not indexable in a traditional sense. The two dimensional array of tiles is important to keeping the code human friendly.

5.4.5 Log Class

The class is not made of any significant data structures; however it is responsible for the interaction with a text file which holds the history of a game. On each line of the text file is a Forsyth Edwards notation String which represents the state of the board for each turn in the game. The Log class is responsible for updating the text file after each turn and reading from the text file when required.

REFERENCES

[1] UI-Spec-Docu-GP9 - User Interface Specification

[2] SE.QA.05 - Design Specification Standard

[3] SE.QA.RS-CS22120 - Requirements Specification

[4] Forsyth-Edward Notation

DOCUMENT HISTORY

Version	Issue No.	Date	Changes made to document	Changed by
0.1		13/02/23	Document created.	TYW1
0.2		08/03/23	Introduction written, four classes added to decomposition description, mapping classes to requirements added, and interface descriptions begun	GWH18
0.2.1		15/03/23	Extended Interface Description with two new classes, Tile and GraphicsLoader	GWH18
0.3		16/03/23	Added LoadGameButton class and enhanced the Interface Description of TileGraphicLoader, Tile, Chessboard, StartScreen, PlayScreen, PlayerNameScreen, LoadScreen, and Interface. Added three Sequence Diagrams for UC 0.0, 1.0, and 2.0.	GWH18
0.4		18/03/23	Added Important algorithms, Set up the board using Edwards Forsyth Notation with pseudocode, Added Finding Legal move Algorithm description. Added Back end Interface descriptions, Added back-end classes to significant classes	SHR27
0.5		19/03/23	Added Component Diagram	TYW1
0.6		20/03/23	Added two new Sequence Diagrams. Added descriptions to sections and TileGraphicLoader and ChessBoard to significant data structures.	GWH18
0.7		21/03/23	Added two UML diagrams and updated the first three Sequence Diagrams.	GWH18
0.8		22/03/23	Added descriptions to public methods in	GWH18

			frontend class interface descriptions	
0.9		22/03/23	Added UML class diagram for the back-end	SHR27
0.10		23/03/23	Made changes to the pseudocode, corrected typos, changed interface specification to be included in the contents	SHR27
0.11		23/03/23	Formatting changes, adjusted TOC, adjusted headings. Minor content alterations.	TYW1
0.12		23/03/23	Added the Log class to section 2 and 4. Added information about Log class' interaction with a text file in the data structures section.	JAT92
0.13		23/03/23	Updated the sequence diagrams and the UML diagrams correcting small mistakes found during review.	GWH18
1.0		24/03/23	Document Released	TYW1
1.1	#16	02/05/23	Changed status to “Released”	TYW1
*	#17	^	Removed “Contents” from contents table	^
*	#18	^	Added citations to SE.QA.	^
*	#20	^	Included reference to JVM process	^
*	#21	^	Removed title references to "Class"	^
*	#23	^	Renamed moveCalculator to MoveCalculator	^
1.2		04/05/23	Updated content regarding Log class in section 4.	JAT92
1.3		10/05/23	Updated game class interface specification, board class interface specification, updated backend UML diagram	SHR27
1.4		10/05/23	Updated Interface specification for front end, updated GUI menu and screen navigation UML	JAT92
2.0		11/0/5/23	Document re-release	TYW1