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:	23 March 2023
Version:	1.0
Status:	Draft

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 UI specification [1].

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. 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 Class

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 Class

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.

Interface

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. It is also responsible for checking for check, checkmate, and stalemate.

Log

This class is responsible for tracking the board state after each turn. When a new game is made 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.

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 use the text file that already exists for that game.

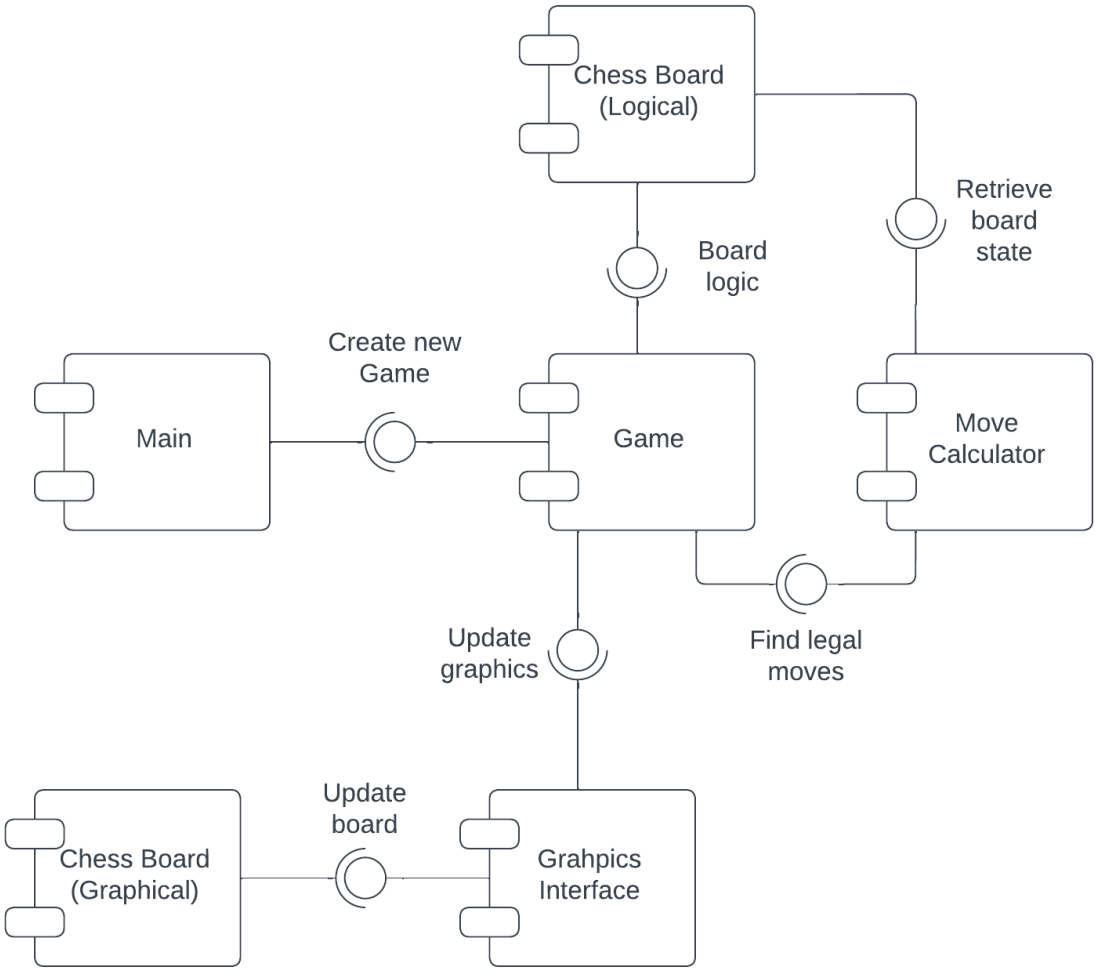
The Log class also has a method for reading a specific line in the text file. This is achieved by passing the line number as a parameter to the method and a FEN string is returned.

2.3 Mapping From Requirements To Classes

Requirement	Classes
FR1	Interface, StartScreen, PlayerNameScreen, PlayScreen
FR2	board
FR3	Chessboard, board, Tile, TileGraphicsLoader, PlayScreen
FR4	Chessboard, Tile
FR5	Chessboard, moveCalculator, board, Tile
FR6	Chessboard, moveCalculator, Tile
FR7	PlayScreen, moveCalculator, ChessBoard, Tile
FR8	PlayScreen
FR9	PlayScreen
FR10	StartScreen, LoadScreen, PlayScreen, Chessboard, board, Log
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 Interface Interface Specification

- Type: Public
- Extends: Application // This is because it contains the main function
- Public Methods:
    - start(Stage stage)
    - main(String[] args)
    - click(int column, int row):Notifies backend of the position selected by the user.
    - toMenu():Sets the stage's primary screen to be the menu screen.
    - toPNScreen():Sets the stage's primary screen to the player name screen.
    - toChessboard():Switches to Playscreen
    - toNewChessboard(String whiteName, String blackName):Switches to playscreen with a new game.
    - loadFGames():Display the loadscreen with a selection of finished games
    - loadUFGames(): Display the loadscreen with a selection of unfinished games

4.2 LoadScreen Interface Specification

- Type: Public
- Extends: Nothing
- Public Methods:
    - LoadScreen(Interface anInterface): Constructor. Constructs Screen.
    - populateSaveBar(): Display a list of saves as buttons
    - requestSave(int num): Requests to view/play a selected game
    - Scene getScene(): returns the LoadScreen Scene
    - setSaveType(): Sets the label to indicate finished or unfinished games

4.3 PlayerNameScreen Interface Specification

- Type: Public
- 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
- Extends: Nothing
- Public Methods:
    - PlayScreen(Interface anInterface): Constructor
    - alertPressedTile(): Message to backend the tile that was pressed
    - setWhitePlayerName(String name): setter for white player's name
    - setBlackPlayerName(String name): setter for black player's name
    - 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
    - incrementThroughLog(): go forwards through the game history
    - decrementThroughLog(): go backwards through the game history
    - areYouSure(): double check the player would like to quit
    - Scene getScene: getter for the scene

4.5 StartScreen Interface Specification

- Type: Public
- Extends: Nothing
- Public Methods:
    - StartScreen(): Constructor
    - getStartScreen(): getter for the scene
    - requestNewGame(): move to the player name input screen to start a new game
    - requestToViewFinishedGame(): move to loadscreen populated with finish games to view
    - requestToFinishGame(): move to loadscreen populated with unfinished games to continue

4.6 ChessBoard Interface Specification

- Type: Not Done
- Extends: Not much at the moments
- Public Methods:
    - Chessboard(): Constructor
    - click(int column, int row): message button press to the backend and highlight pressed tile
    - updateBoard(String boardNotation): use FEN to update graphical piece positions
    - getChessBoard(): getter for the chessboard gridpane
    - highlightTiles(): Takes a series of coordinates and changes their colour

4.7 piece Interface Specification

- Type: public
- Extends: Nothing
- Public Methods:
    - piece(char colour, vector2 position, char type):
    - getColor(): returns the pieces' colour
    - getPosition() returns the pieces' position as a vector2
    - setPosition(): sets the piece's position to a vector
    - getPossibleMoves(): returns the ArrayList of Vector2 that represent the piece's moves
    - addMove(): add's a vector2 to the arrayList of possible moves
    - getType(): returns the type of the piece as a char
    - hasMoved(): returns a boolean that indicates if the piece has moved or not
    - setHasMoved(): sets whether the piece has moved or not

4.8 vector2 Interface Specification

- Type: public
- Extends: Nothing
- Public Methods:
    - vector2(): constructor
    - vector2(int x, int y): constructor
    - getVector2AsBoardNotation(): returns the vector2 as a chess board position notation

4.9 moveCalculator Interface Specification

- Type: public
- Extends: Nothing
- Public Methods:
    - moveCalculator(char player, board board): constructor
    - findLegalMovesForPlayer(boolean opponentPlayer): finds the legal moves for the player or opponent player
    - isPlayerInCheck(): returns true if the player is in check
    - isPlayerInCheckMate(): returns true if the player is in checkmat
    - isPlayerInStaleMate(): returns true if the player is in stalemate

4.10 Game Interface Specification

- Type: public
- Extends: Nothing
- Public Methods:
    - game(boardState): constructor
    - move():

4.11 board Interface Specification

- Type: public
- 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
    - setPiece(vector2 coordinate) sets a coordinate on the board to a piece
    - getWhiteKingPosition(): returns the white king's position as a vector2
    - getBlackKingPosition(): returns the black king's position as a vector2

4.12 Tile Interface Specification

- Type: public
- Extends: Nothing
- Public Methods:
    - Tile(Int row,Int column,boolean white,Chessboard chessboard): Constructor
    - setGraphic(ImageView iv): Set the button to display the passed imageview
    - clearTile(): clear the tile of any graphics or highlights
    - setStyleClass(String styleClass): change the button's styleclass to a new one.

4.13 TileGraphicLoader Interface Specification

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

4.14 LoadGameButton Interface Specification

- Type: public
- Extends: Nothing
- Public Methods:
    - LoadGameButton(): constructor
    - selected(): message forward the index of the save selected.

4.15 Log Interface Specification

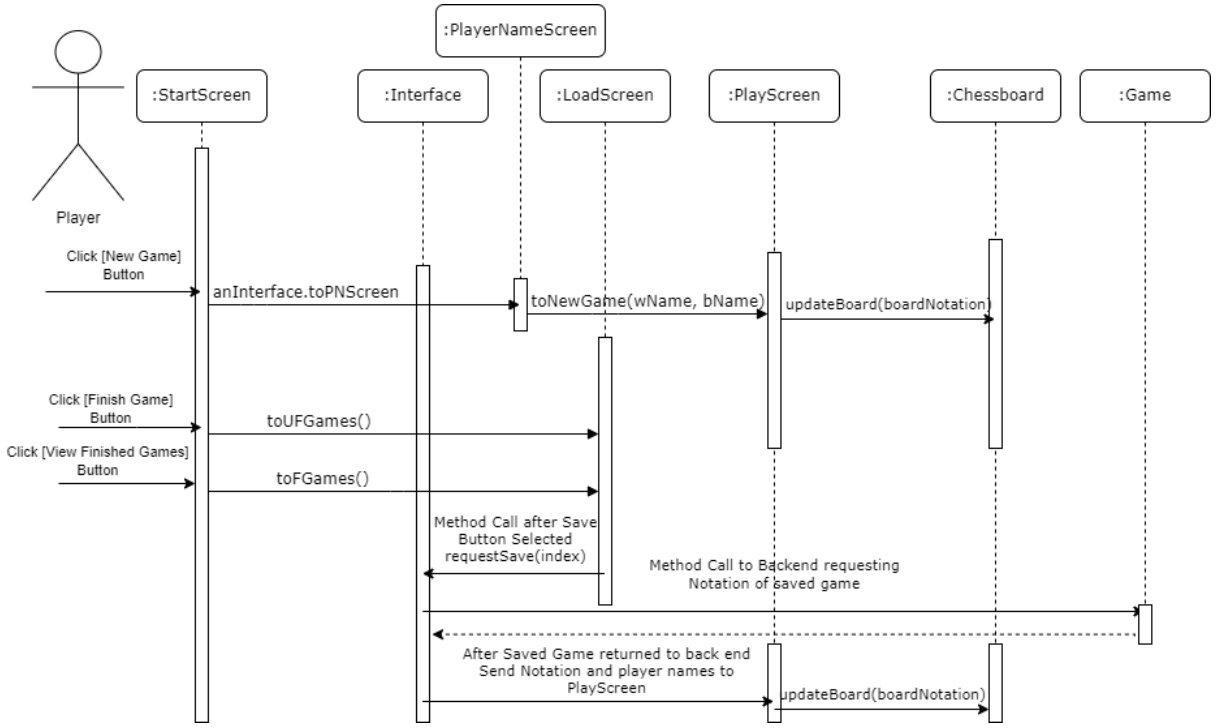
- Type: public
- Extends: Nothing
- Public Methods:
    - Log(): constructor

- updateLog(String FEN): Appends the FEN String to the text file
- readLog(int lineNumber): Returns the FEN String at the specified line of the text file.

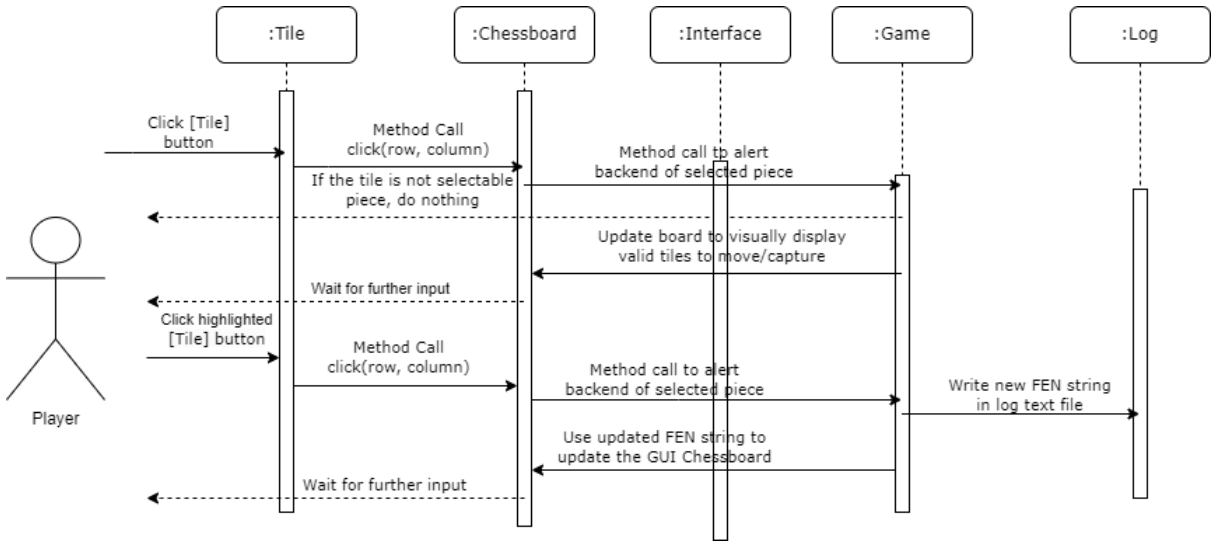
5. DETAILED DESIGN

5.1 Sequence Diagrams

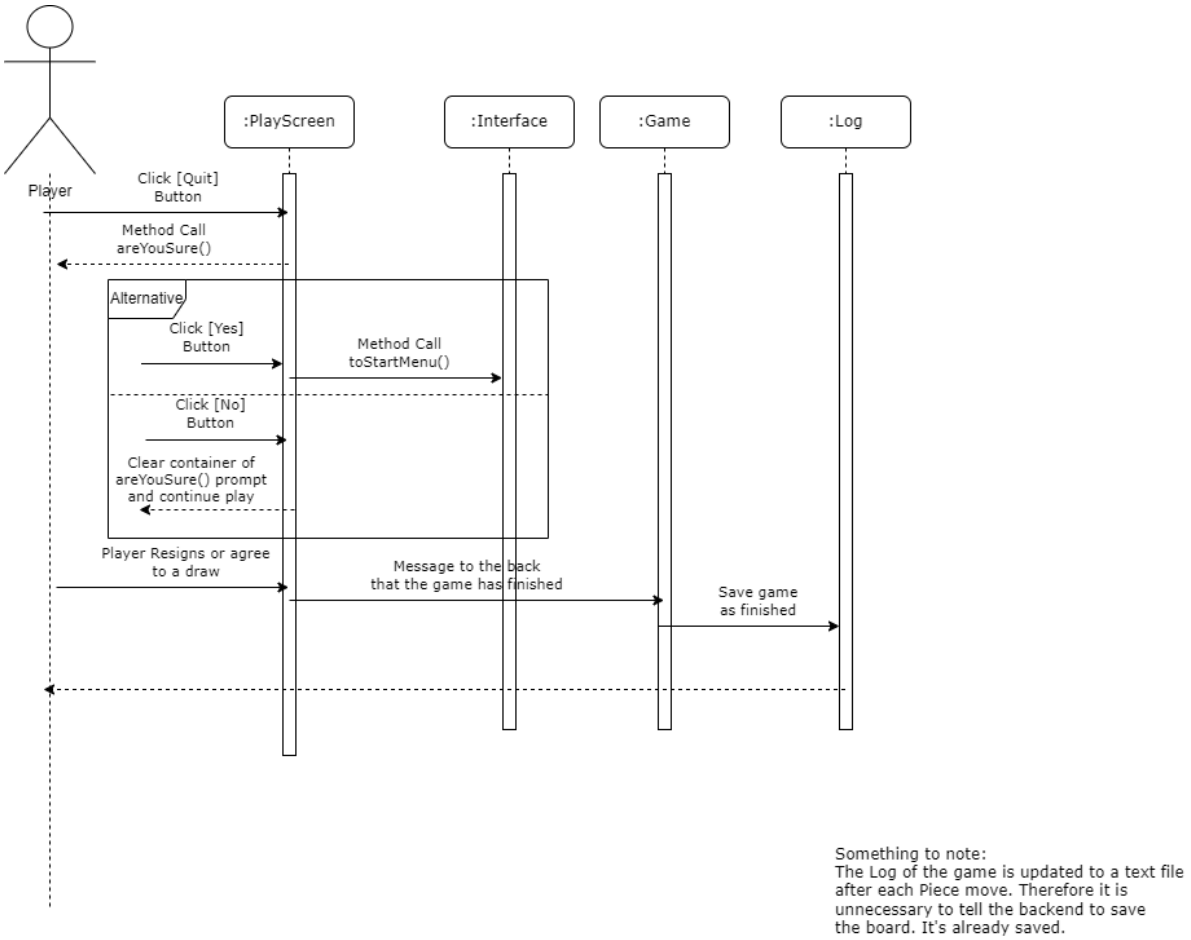
5.1.1 UC-0.0 Launching Menu



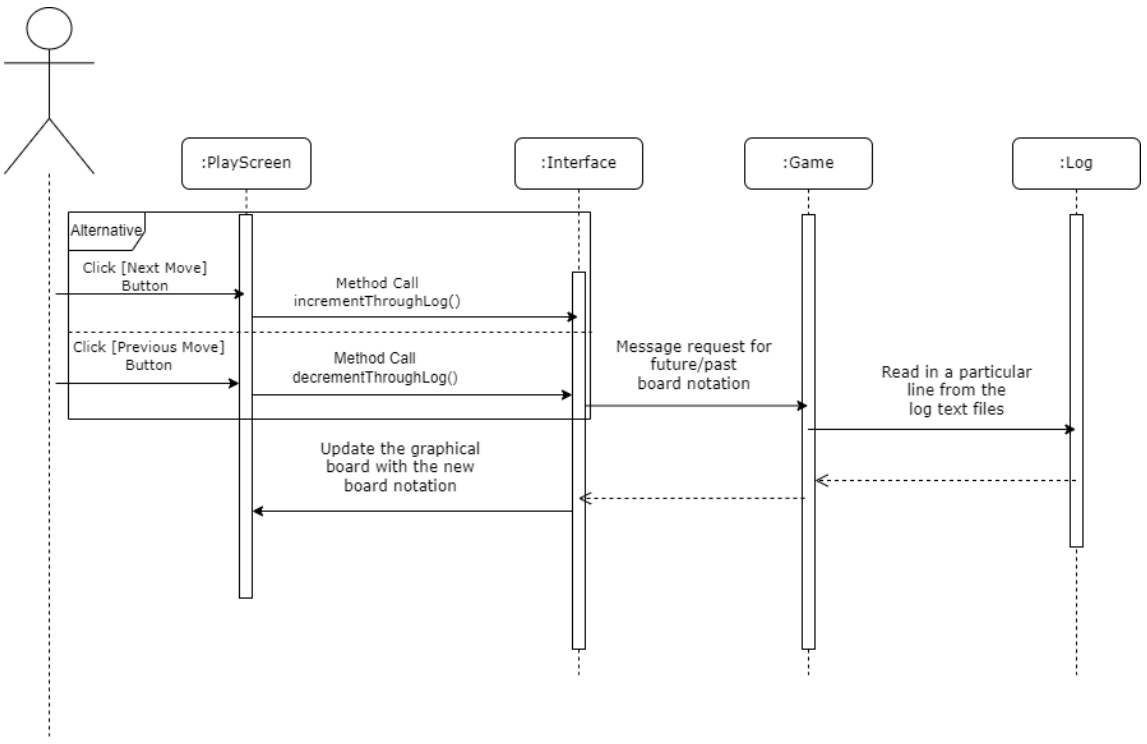
5.1.2 UC-1.0 Selecting a Piece and moving a Piece



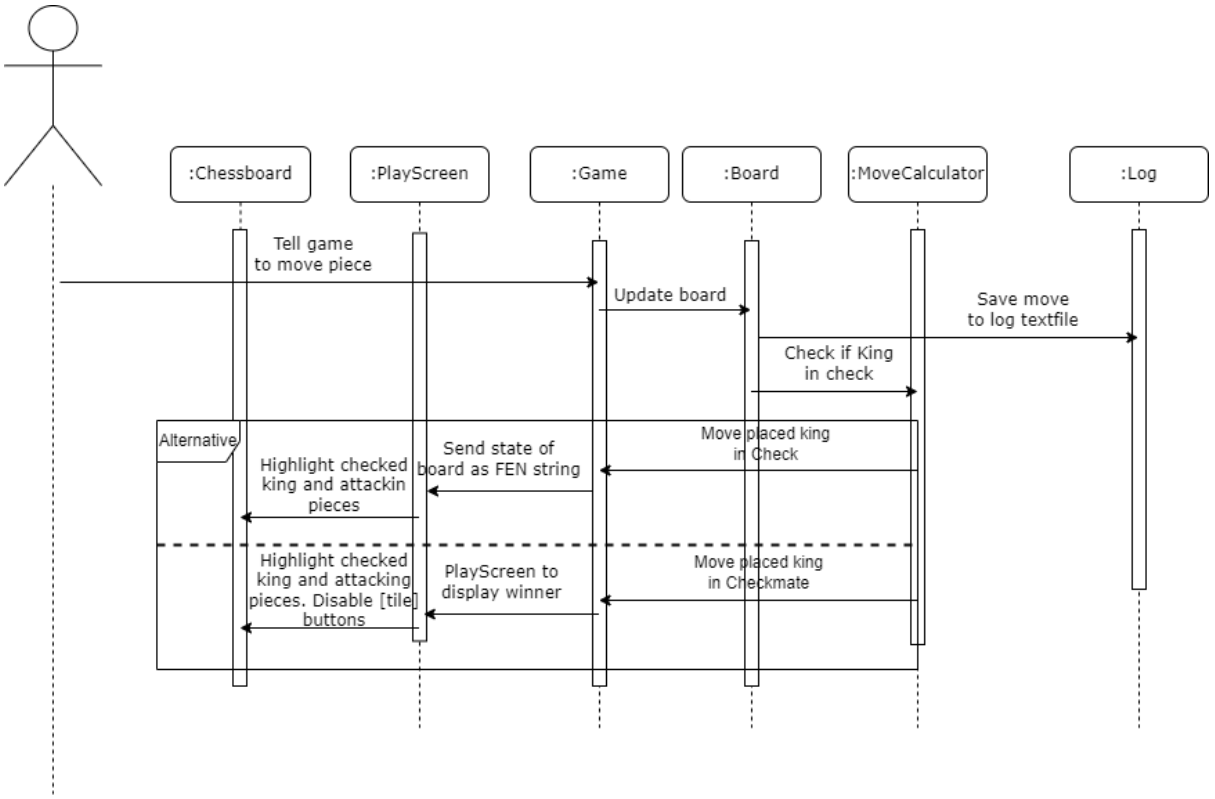
5.1.3 UC-2.0 Quitting/Resigning/Offering Draw



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

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:

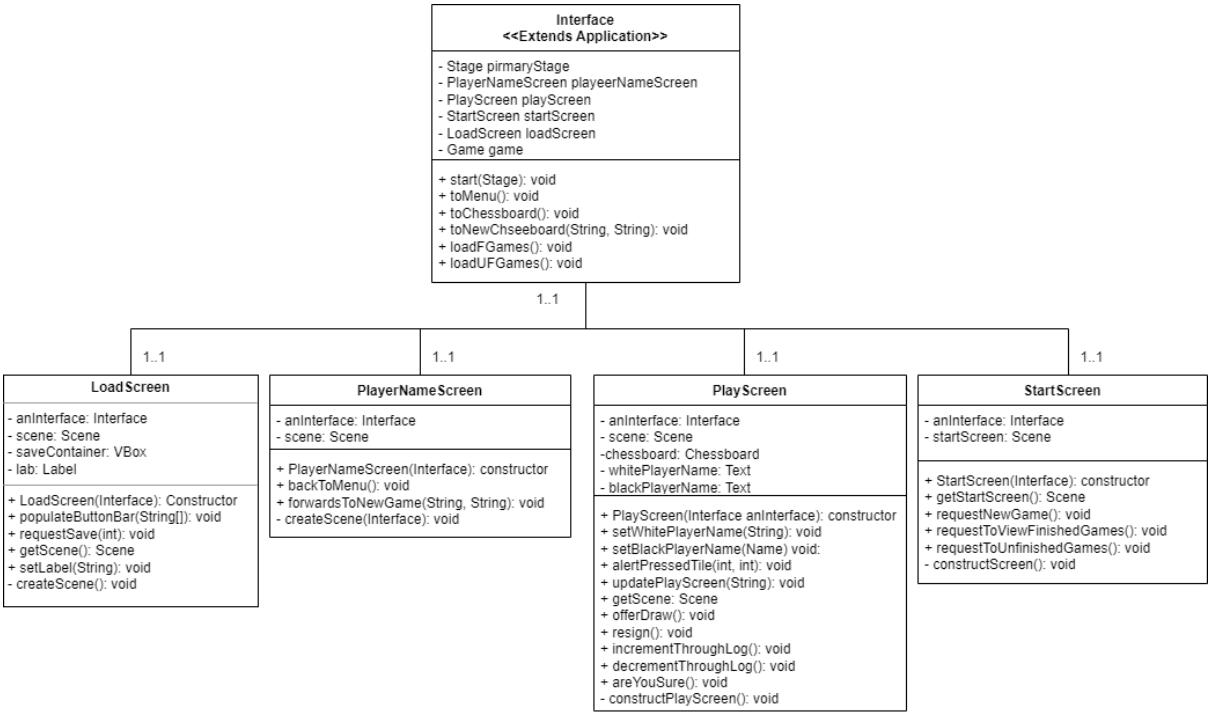
```
moveCalculator.findLegalMoves(true);
```

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, 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.

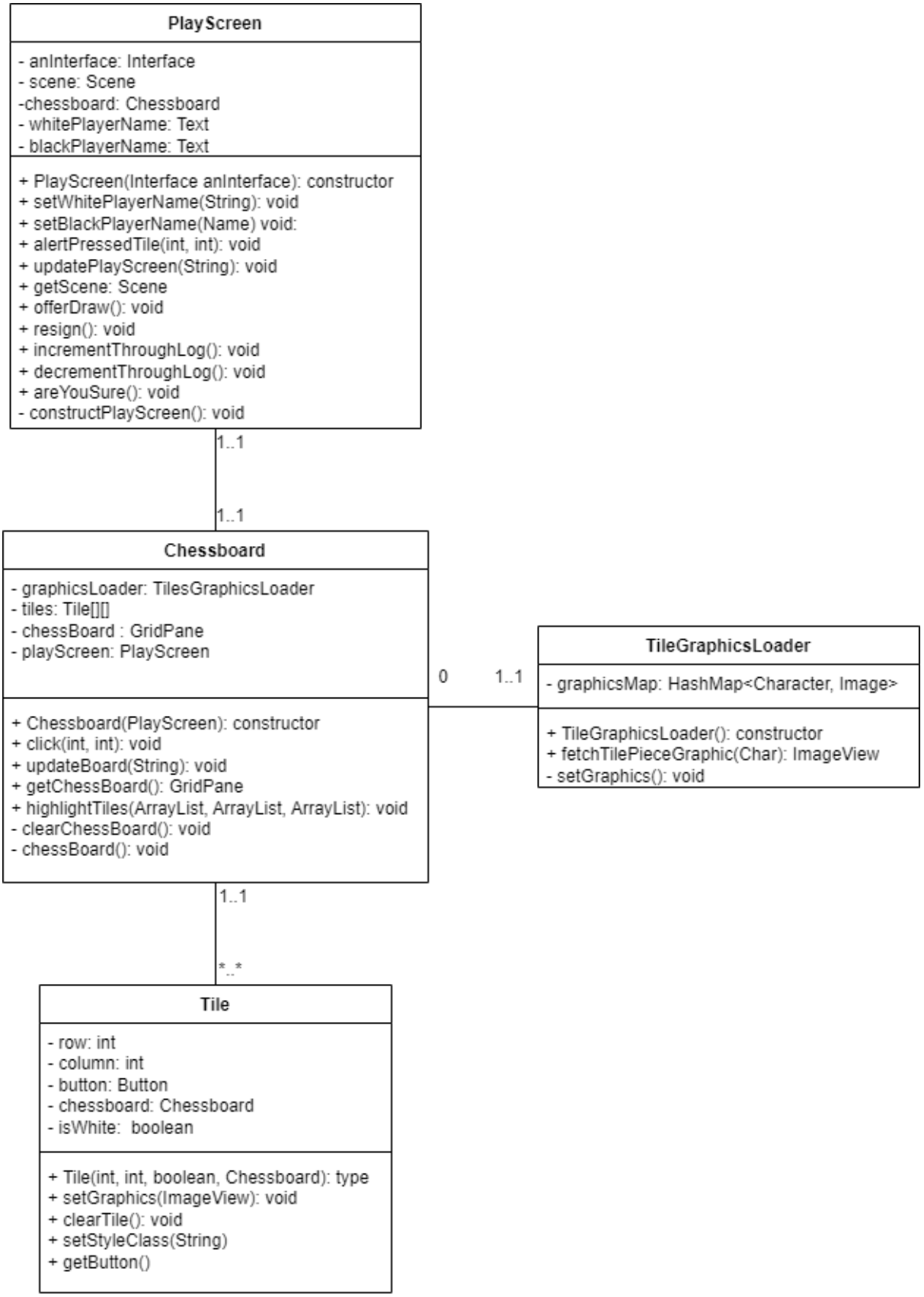
```
moveCalculator.findLegalMoves(true);
```

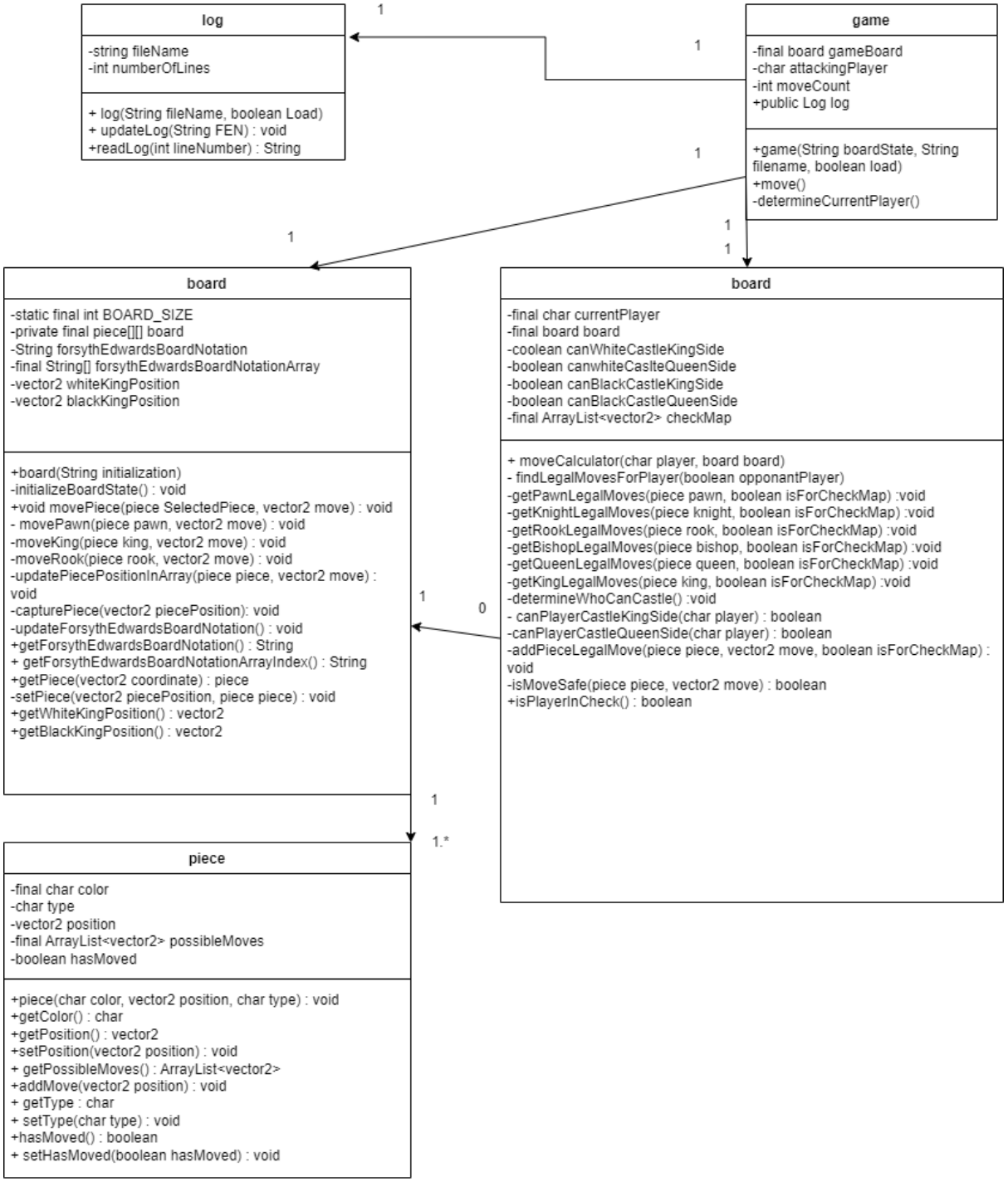
5.3 UML Class Diagrams

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 Vector2 class

The Vector2 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

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 frontend class interface descriptions	GWH18
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	SHR27

			specification to be included in the contents	
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