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: 3 March 2023

Version: 0.2

Status: Draft

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

CONTENTS

1. INTRODUCTION

Clear, consistently followed, document standards are essential in software engineering.

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, 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

Some text.

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 object classes and the graphics are handled by JavaFX classes alongside CSS stylesheets.

2.2 Significant Classes In Each Program

There are many significant classes in our system. Here they are listed in no particular order.

2.2.1 Board Class

The board class is responsible for logically representing the current state of the board. Using a 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.

2.2.2 Piece Class

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

2.2.3 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 frontend.

2.2.4 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 surrender, and stepping through the history of the game.

2.2.5 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.

2.2.6 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.

2.2.7 moveCalculator

This class is responsible for taking a board class and then calculating the legal moves for all the pieces of a particular color, it is also responsible for checking for check, checkmate, and stalemate.

2.3 Mapping From Requirements To Classes

Requirement	Classes
FR1	Interface, StartScreen, PlayerNameScreen, PlayScreen
FR2	PlayScreen, board
FR3	Chessboard, board
FR4	Chessboard
FR5	Chessboard, moveCalculator, board
FR6	Chessboard, moveCalculator
FR7	PlayScreen, move Calculator
FR8	PlayScreen
FR9	PlayScreen
FR10	StartScreen, LoadScreen, PlayScreen, Chessboard, board
FR11	StartScreen, LoadScreen, PlayScreen, Chessboard, board

3. DEPENDENCY DESCRIPTION

Some text.

3.1 Component Diagrams

Good luck.

4. INTERFACE DESCRIPTION

Some text.

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()
-toNewChessboard(String whiteName, String blackName)
-loadFGames()
-loadUFGames()

4.2 LoadScreen Interface Specification

-Type: Public
-Extends: Nothing
-Public Methods:
-LoadScreen(Interace anInterface): Constructor. Constructs Screen.
-populateSaveBar()
-requestSave(int num)
-Scene getScene()
-setSaveType()

4.3 PlayerNameScreen Interface Specification

-Type: Public
-Extends: Nothing
-Public Methods:
-PlayerNameScreen(Interface anInterface)
-Scene getScene()
-backToMenu()
-forwardsToNewGame(String whiteName, String blackName)

4.4 PlayScreen Interface Specification

-Type: Public
-Extends: Nothing
-Public Methods:
- -PlayScreen(Interface anInterface): Constructor
- -chessBoard()
- -alertPressedTile()
- -setWhitePlayerName(String name)
- -setBlackPlayerName(String name)
- -updatePlayScreen(String boardNotation)
- -offerDraw()
- -resign()
- -incrementThroughLog()
- -decrementThroughLog()
- -areYouSure()
- -Scene getScene

4.5 StartScreen Interface Specification

-Type: Public
-Extends: Nothing
-Public Methods:
- -StartScreen(): Constructor
- -getStartScreen()
- -requestNewGame()
- -requestToViewFinishedGame()
- -requestToFinishGame()

4.6 ChessBoard Interface Specification

-Type: Not Done
-Extends: Not much at the moments
-Public Methods:
- -Chessboard(): Constructor
- -click(int column, int row):
- -updateBoard(String boardNotation):
- -getChessBoard():
- -highlightTiles(): Takes a series of coordinates and changes their colour

4.7 piece Interface Specification

-Type: public
-Extends: Nothing
-Public Methods:
    -piece(char color, vector2 position, char type):
    -getColor():
    -getPosition()
    -setPosition():
    -getPossibleMoves():
    -addMove():
    -getType():
    -hasMoved():
    -setHasMoved():

4.8 vector2 Interface Specification

-Type: public
-Extends: Nothing
-Public Methods:
    -vector2():
    -vector2(int x, int y):
    -getVector2AsBoardNotation():

4.9 moveCalculator Interface Specification

-Type: public
-Extends: Nothing
-Public Methods:
    -moveCalculator(char player, board board):
    -findLegalMovesForPlayer(boolean opponantPlayer):
    -isPlayerInCheck():
    -isPlayerInCheckMate():
    -isPlayerInStaleMate():

4.10 board Interface Specification

-Type: public
-Extends: Nothing
-Public Methods:
    -board(String initializingBoardState):
    -movePiece(piece selectedPiece, vector2 move):
    -getForsythEdwardsBoardNotation():
    -getForsythEdwardsBoardNotationArrayIndex(int Index):
    -getPiece(vector2 coordinate)
    -setPiece(vector2 coordinate)
    -getWhiteKingPosition():
    -getBlackKingPosition():

4.11 Game Interface Specification

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

4.11 Tile Interface Specification

-Type: public
-Extends: Nothing
-Public Methods:
    -Tile(Int row,Int column,boolean white,Chessboard chessboard): Constructor
    -setGraphic(ImageView iv):
    -clearTile():
    -setStyleClass(String styleClass):

4.12 TileGraphicLoader Interface Specification

-Type: public
-Extends: Nothing
-Public Methods:
    -TileGraphicsLoader(): Constructor
    -fetchTileGraphicPiece(char Symbol):

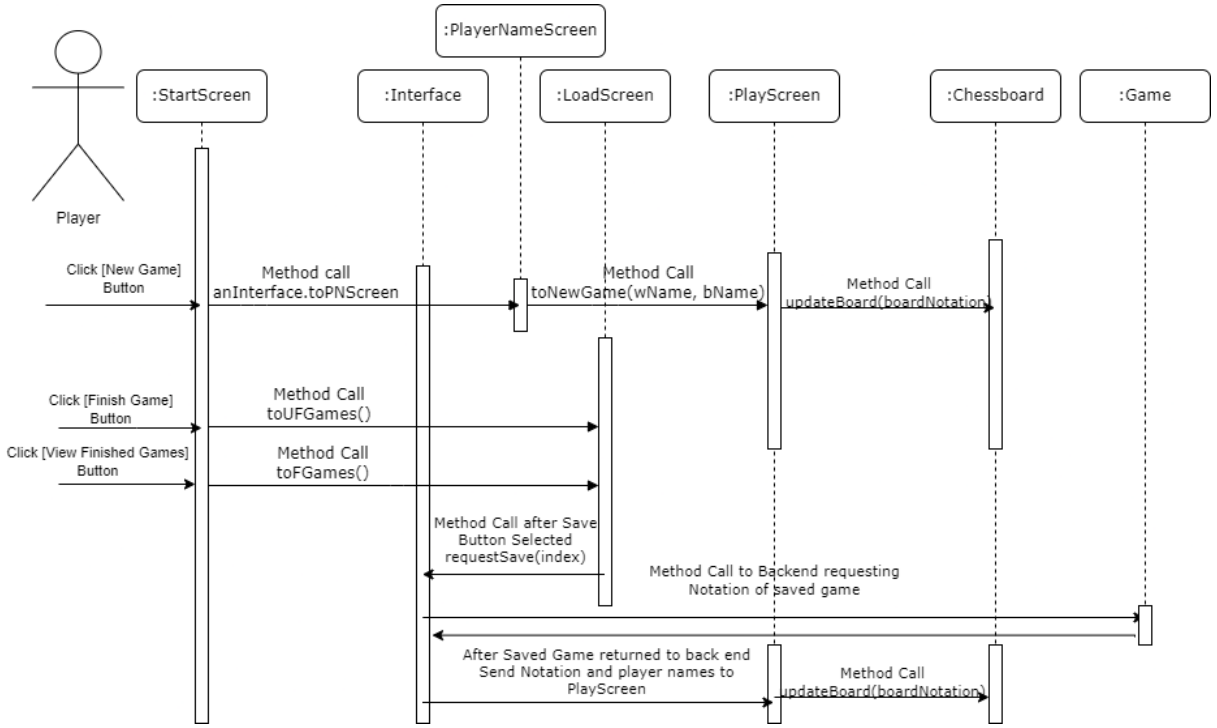
4.13 LoadGameButton

-Type: public
-Extends: Nothing
-Public Methods:
    -LoadGameButton(): constructor
    -selected():

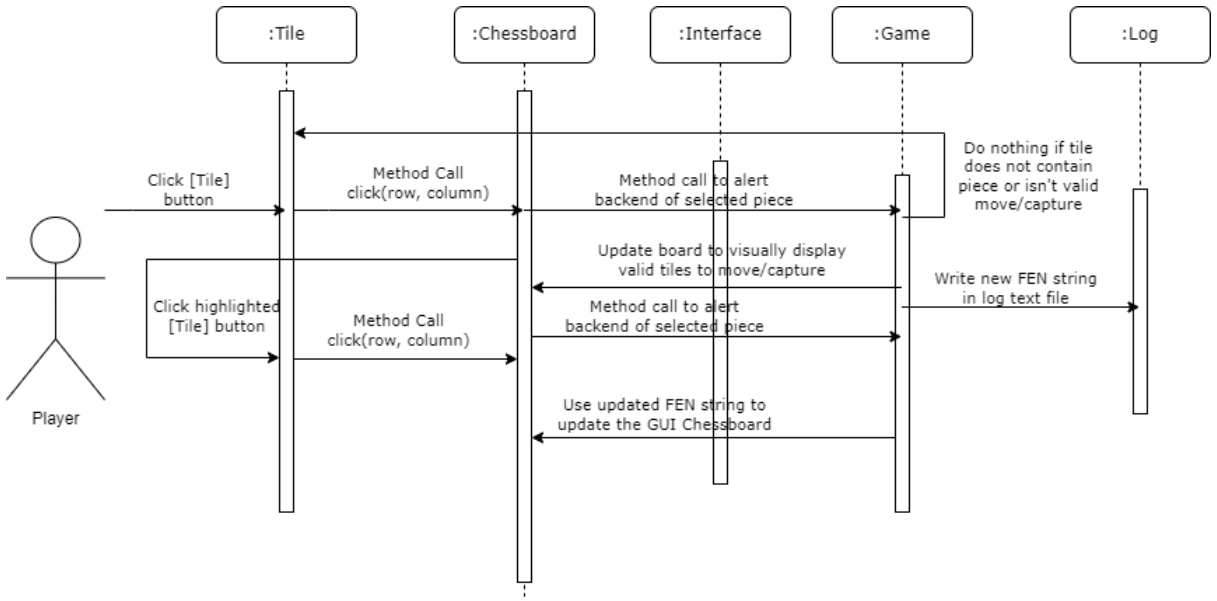
5. Detailed Design

5.1 Sequence Diagrams

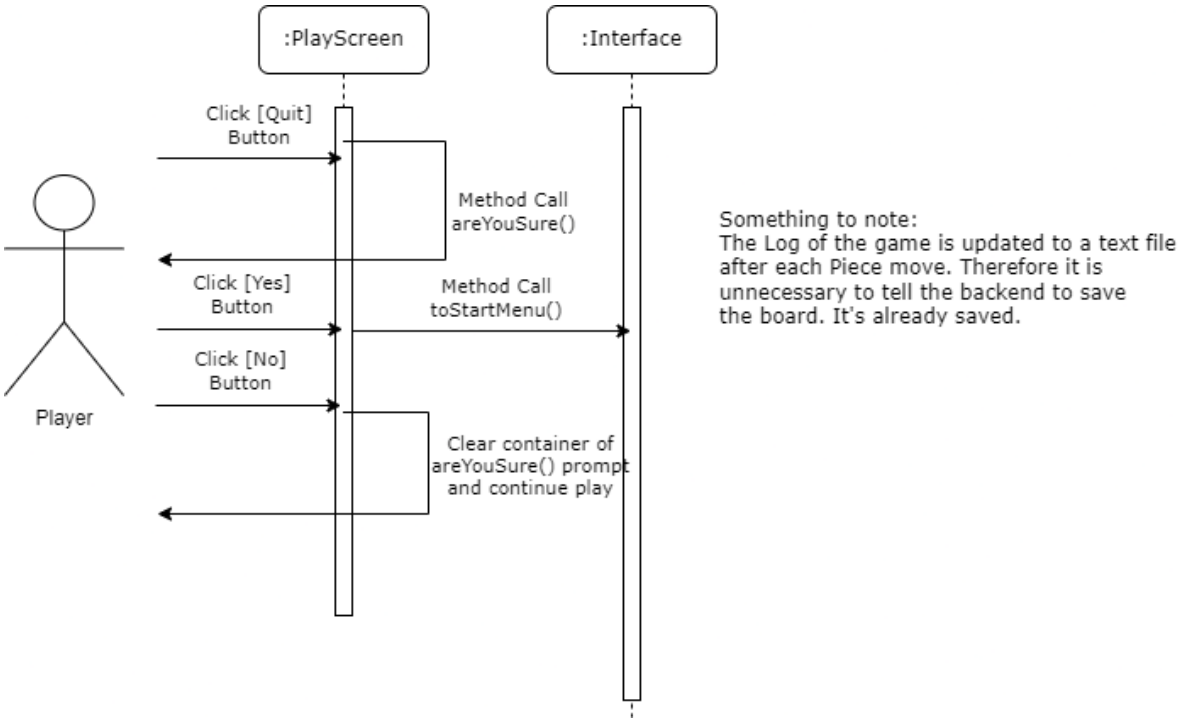
5.1.1 UC-0.0 Launching Menu



5.1.2 UC-1.0 Selecting a Piece



5.1.3 UC-2.0 Quitting a Game



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 Reading Forsyth Edwards Notation

The board is given the Forsyth Edwards board notation as a string, then the String is split on the demileter ‘ ’ (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 itterates 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.

Other wise we know that we have reached a character that is neither a ‘/’, or a digit, this means that this character is a character which represents a piece that we must add to the board. First we check if the character is uppercase or lowercase to distinguish between wether the piece is a white, or black piece. 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 set the position of the white, or black king, depending wether or not the character is a 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 (int posInString = 0; posInString < boardRepresentation.length(); posInString ++)
{
        If (current character = '/') {file = 0; rank–; continue}

        If (current character is digit) {rank + current character; continue}

        If (current character is uppercase) {
                Piece p = new piece(current character, white)
                addPiece(p, file, rank)

                if(current character = 'K') {
                        //update black king position.
                }
        } else {
                Piece p = new piece(current character, white)
                addPiece(p, file, rank)

                if(current character = 'K') {
                        //update black king position.
                }
        }
        rank++
}
```

5.2.2 Calculating The Legal Moves

First we must determine which player's 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 opponant 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

5.4 Significant Data Structures

5.4.1 internal representation of the board

REFERENCES

[1] QA Document SE.QA.01 - Quality Assurance Plan

[2] QA Document SE.QA.02 - General Documentation Standards

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