Software Engineering Group 18 Project

Design Specification

|  |  |
| --- | --- |
| Authors: | Jasper Crabb [jac127], Muntazir Rashid [mrm19], Archie Malvern [arm36], Jason Smith [jas160], William Parry [wip24] |
| Config Ref: | DSGroup18 |
| Date: | 18th March 2023 |
| Version: | 0.7 |
| Status: | Draft |

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Copyright © Aberystwyth University 2021

CONTENTS

CONTENTS 2

1. Introduction 3

1.1 Purpose of this Document 3

1.2 Scope 3

1.3 Objectives 3

2. Decomposition description 3

2.1 Programs in System 3

2.2 Significant Classes 3

2.3 Table Mapping Requirements onto Classes 3

3. Dependency description 3

3.1 Component Diagrams 3

4. Interface Description 3

5. DEtailed Design 4

5.1 Sequence Diagrams 4

5.2 Significant Algorithms 4

5.3 UML Class Diagrams 4

5.4 Significant Data Structures 4

REFERENCES 4

DOCUMENT HISTORY 5

# Introduction

## Purpose of this Document

The purpose of this document is to describe how we have designed our chess tutor. This document will do this by showing relationships between the requirements, classes, methods and algorithms and how they combine into our final product

## Scope

This document specifies each aspect of our program and how they work in our final product.

This follows the standards laid down by the Design Specification Standards [1]

This document should be read by all project members. Before reading this document, the reader should familiarise themselves with the UI Specification [2]

## Objectives

The objective of this document is to show how each component of our program interacts to create the final product.

# Decomposition description

## Programs in System

Our system will only have one program in it. This program is structured into classes which handle graphics (using JavaFx) and the logic of the chess game. The JavaFx classes use event handlers to manage user input and we use JavaFx files for the layout.

## Significant Classes

Our program has many classes but there are some significant classes that are detailed below:

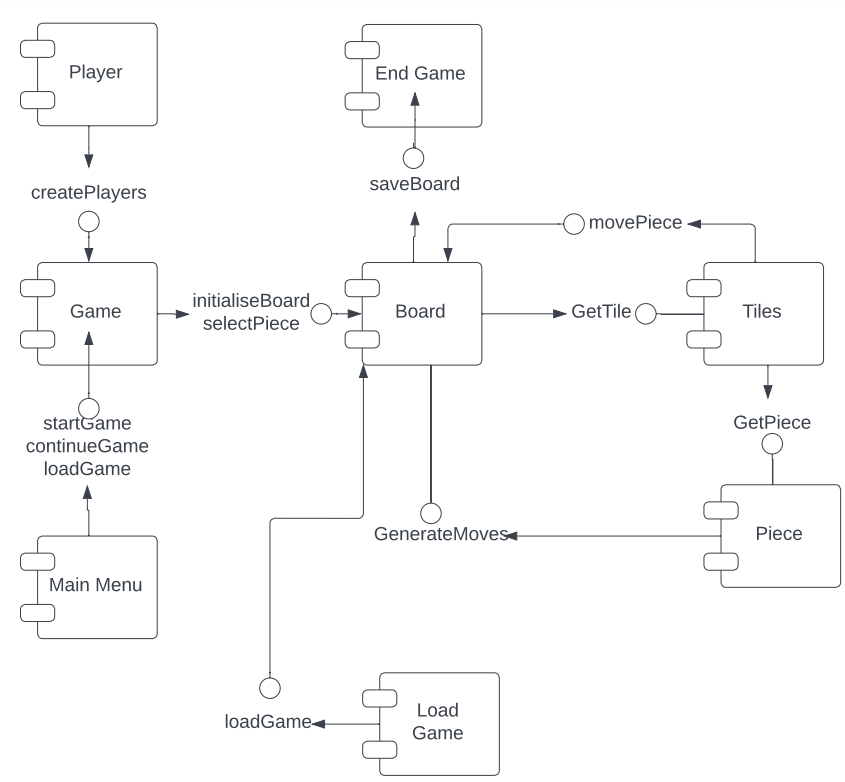
* Main:
  + This class is the entry point to the application. It creates a menu object for the main menu and from here a ChessBoard object is created, which means that we can display and play the game
* ChessBoard:
  + This class contains a 2d array of tiles and acts as a platform that ties the logic aspects of the game together. It also initialises the pieces on the chess board, whether that be a saved layout or the standard starting position
* Tiles:
  + The tiles class is used to store the chess pieces alongside their positions. It allows for us to flip the coordinates of the board without modifying the piece array. This is needed so that we can apply the piece move algorithms to both sides of the board.
* Piece abstract class:
  + This class acts as a template for all the other piece classes, which are for specific pieces. It has methods for valid move generation and a way to store each valid move in an arrayList

## Table Mapping Requirements onto Classes

|  |  |
| --- | --- |
| Requirement | Classes providing requirement |
| FR1 | Main, PlayerMenu, FENconverter |
| FR2 | PlayerMenu, ChessBoard, Tile |
| FR3 | ChessBoard, Tile |
| FR4 | ChessBoard, Tile |
| FR5 | ChessBoard, Tile, Piece, Bishop, Pawn, King, Queen, Knight, Rook, Check |
| FR6 | ChessBoard, Tile, Check |
| FR7 | ChessBoard, Tile, Check |
| FR8 | ChessBoard, FENconverter |
| FR9 | ChessBoard, FENconverter |
| FR10 | Main, FENconverter, ChessBoard |
| FR11 | Main, FENconverter, ChessBoard |

# Dependency description

## Component Diagrams



# Interface Description

4.1 Piece interface specification

Type: public abstract

Extends: nothing

Public methods:

* Piece(boolean colour): The constructor initializes the colour and moves ArrayList for the piece.
* Piece(): An empty constructor.
* ArrayList<Tile> getAllMoves(): Returns all moves available for the piece.
* void generateMoves(Board board, Tile piece): Generates all moves available for the piece on the board.
* int checkForPiece(Tile tile): Checks the tile for a piece and returns a value representing whether it is empty, occupied by an enemy, occupied by a friendly piece, or out of bounds.
* int checkWhichPiece(Tile tile): Returns a value representing which piece is located on the tile.
* ArrayList<Tile> generateStraightMoves(Board board, Tile piece): Generates all straight moves available for the piece on the board.
* ArrayList<Tile> generateDiagonalMoves(Board board, Tile piece): Generates all diagonal moves available for the piece on the board.
* boolean getColour(): Returns the colour of the piece.
* void setColour(boolean colour): Sets the colour of the piece.
* char getPieceChar(): Returns the character representation of the piece.

4.2 Bishop interface specification

Type: public

Extends: Piece

Public methods:

* Bishop(boolean colour): The constructor initializes the bishop piece with its corresponding color.
* void generateMoves(Board board, Tile piece): Generates all moves available for the bishop on the board.

4.3 King interface specification

Type: public

Extends: Piece

Public methods:

* King(boolean colour): The constructor initializes the king piece with its corresponding colour.
* void generateMoves(Board board, Tile piece): Generates all moves available for the king on the board.
* ArrayList<Tile> castlingMoves(Board board, Tile piece): Returns valid castling moves for the king on the board.

4.4 Pawn interface specification

Type: public

Extends: Piece

Public methods:

* Pawn(boolean colour): The constructor initializes the pawn piece with its corresponding colour and move status.
* Pawn(boolean colour, int moveStatus): Constructor used for testing purposes.
* void updateMoveStatus(): Updates the move status of the pawn.
* void doubleMoveUpdate(): Updates the move status of the pawn when it performs a double move.
* boolean doubleMoveLast(): Returns true if the pawn performed a double move in its last turn.
* void generateMoves(Board board, Tile piece): Generates all moves available for the pawn on the board.

4.6 Queen interface specification

Type: public

Extends: Piece

Public methods:

* Queen(boolean colour): The constructor initializes the queen piece with its corresponding colour.
* void generateMoves(Board board, Tile piece): Generates all moves available for the queen on the board.

4.7 Rook interface specification

Type: public

Extends: Piece

Public methods:

* Rook(boolean colour): The constructor initializes the rook piece with its corresponding colour.
* void generateMoves(Board board, Tile piece): Generates all moves available for the rook on the board.
* boolean hasMoved(): Returns true if the rook has moved.
* void setHasMoved(boolean hasMoved): Sets the hasMoved variable of the rook.

4.8 Knight interface specification

Type: public

Extends: Piece

Public methods:

* Knight(boolean colour): The constructor initializes the rook piece with its corresponding colour.
* void generateMoves(Board board, Tile piece): Generates all moves available for the Knight on the board.

4.9 EmptyPiece interface specification

Type: public

Extends: Piece

Public methods:

* EmptyPiece(): The constructor initializes an empty piece.
* void generateMoves(Board board, Tile piece): Generates no moves for an empty piece.

4.9 Tile interface specification

Type: public

Extends: nothing

Public methods:

* Tile(int x, int y, Piece piece): The constructor initializes the tile with its coordinates and the piece on it.
* Tile(int x, int y): An empty constructor that initializes the tile with its coordinates and an empty piece.
* Piece getPiece(): Returns the piece on the tile.
* void setPiece(Piece piece): Sets the piece on the tile.
* int getX(): Returns the x-coordinate of the tile.
* void setX(int x): Sets the x-coordinate of the tile.
* int getY(): Returns the y-coordinate of the tile.
* void setY(int y): Sets the y-coordinate of the tile.
* void flipCoords(): Flips the coordinates of the tile.
* boolean isEmpty(): Returns true if the tile is empty, false otherwise.
* void setEmpty(boolean empty): Sets the tile to be empty or not.

4.10 Main interface specification

Type: public

Extends: Application

* This is required by JavaFX

Public methods:

* void main(String[] args): The entry point for the application. Launches the application with the specified arguments.
* void start(Stage stage): The main method that initializes and starts the application. Creates a menu screen with buttons for starting, loading, replaying, and quitting a game. The start button launches the ChessBoard screen. The load and replay buttons print a message to the console, and the quit button closes the application.

4.11 ChessBoard class interface specification

Type: public

Extends: Application

* This is required by JavaFX

Public methods:

* ChessBoard(): The constructor creates a ChessBoard object with a default size of 8x8. It creates and adds the tiles and pieces to the board.
* Tile getTile(int row, int col): Returns the Tile located at the specified coordinates on the board. Returns null if the coordinates are out of bounds.
* void start(Stage stage): Sets up the scene and shows the board on the stage.
* void flipBoard(): Flips the gameBoard and all Tile coordinates to allow for viewing the board from the opposite perspective.

Private methods:

* void drawPiece(String imageName, int row, int col): Draws the specified image on the board at the given row and column, and sets the event handler for clicking on the piece.

4.12 Check interface specification

Type: public

Extends: nothing

Public methods:

* void checkCheck(Tile tile, Board board): Checks if the specified Tile puts the opposing King in check. If the opposing King is in check, it checks if the King is in checkmate by iterating through all possible moves and checking if any of them will get the King out of check. If the King is in checkmate, the method prints a message indicating that the game has been won. If the King is in check but not in checkmate, the method prints a message indicating that the King is in check.

4.13 PlayerMenu class interface specification

Type: public

Extends: Application

* This is required by JavaFX

Public methods:

* PlayerMenu(String name1, String name2): The constructor creates a PlayerMenu object with two given names.
* void start(Stage stage): Sets up the player menu.
* String getPlayerOneName(): Returns the name of the first player.
* String getPlayerTwoName(): Returns the name of the second player.
* void chooseColor(): Prompts player one to choose between playing as white or black.
* String getChosenColor(): Returns the colour that player one has chosen to play as.

4.14 FenConverter class interface specification

Type: public

Extends: nothing

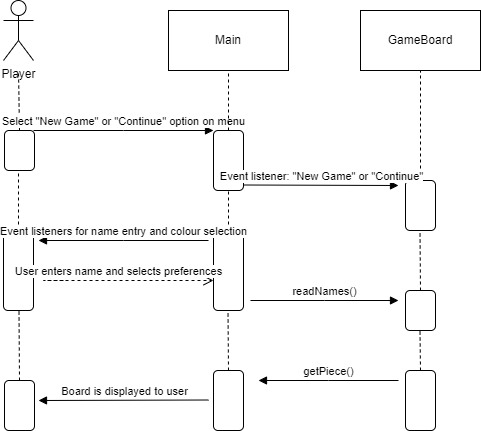
Public methods:

* static void writeFENToFile(String filePath, String stringToWrite): Writes the specified string to the file at the given file path in append mode.
* static Piece[][] fenToPieceArray(String fen): Converts the specified FEN string into a 2D array of Piece objects representing the board layout.
* static void convertBoard(Piece[][] pieces, Board board): Converts the 2D array of Piece objects representing the board layout to a Board object.
* static String pieceArrayToFen(Piece[][] pieces): Converts the 2D array of Piece objects representing the board layout into a FEN string.

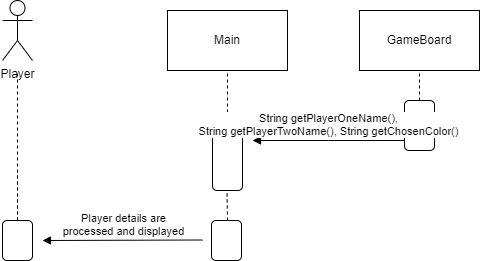
# Detailed Design

## Sequence Diagrams

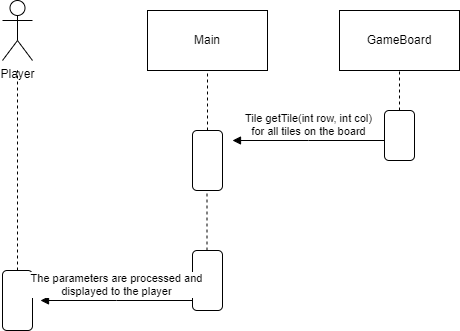
FR1: Player Setup



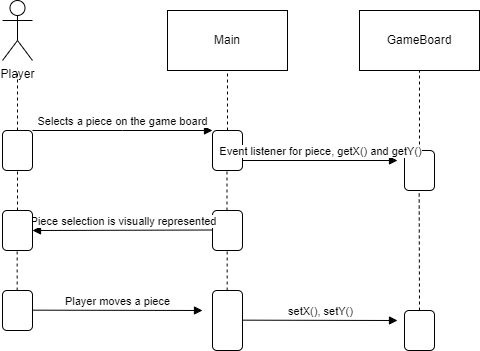
FR2: Player Management



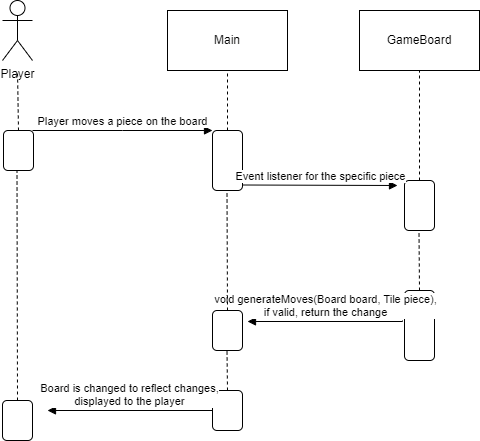
FR3: Board Management



FR4: Piece selection

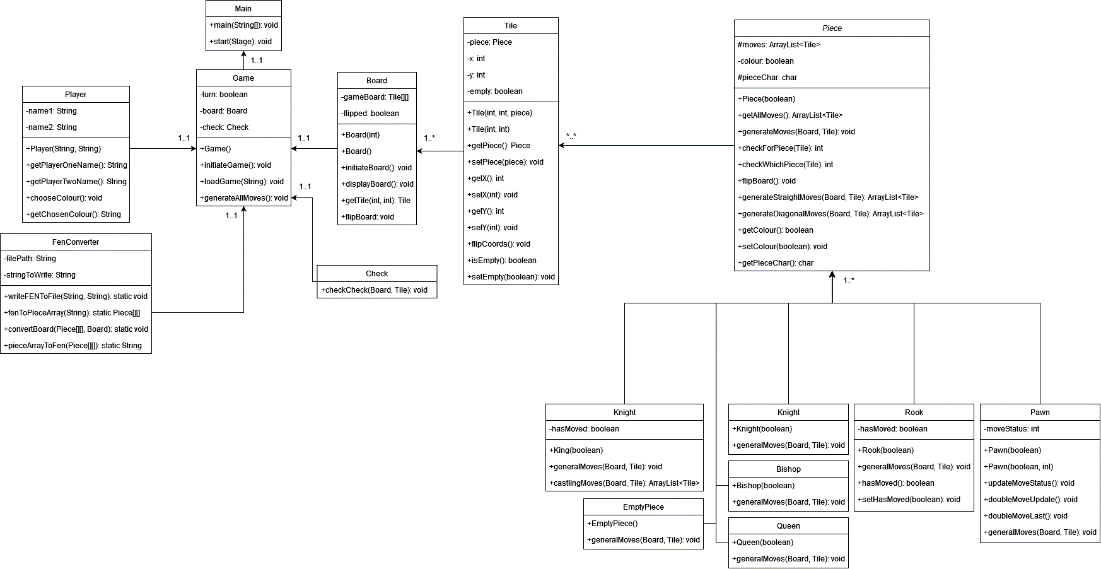


FR5: Movement



## Significant Algorithms

## UML Class Diagrams



## Significant Data Structures

Our program will have a 2d array that is to be created by the ChessBoard class. This will be used to store tiles, which contain all information about pieces. This 2d array is a virtual representation of a chessboard. Our program has several methods to accommodate this array. Such as a method that flips and mirrors the coordinates, allowing chess moves to be calculated for both sides of the board.

Our program will also use text files to store games and their configurations. Each file will have a list of FEN strings which represent the state of the board at each turn. These files will be used for resuming games as well as the replay function.

REFERENCES

[1] Software Engineering Group Projects – Design Specification Standards / 2.3 (Release)

[2] Use Case Document/1.0(Release)

DOCUMENT HISTORY

| *Version* | *Issue No.* | *Date* | *Changes made to document* | *Changed by* |
| --- | --- | --- | --- | --- |
| 0.1 | N/A | 08/03/23 | N/A - original version | jac127 |
| 0.2 | N/A | 13/03/23 | Added some sequence diagrams | arm36 |
| 0.3 | N/A | 14/03/23 | Added interface description | mrm19 |
| 0.4 | N/A | 14/03/23 | Added component diagram | jas160 |
| 0.5 | N/A | 15/03/23 | Added more sequence diagrams and updated existing ones to better match the interface description | arm36 |
| 0.6 | N/A | 15/03/23 | Added UML Class Diagram | wip24 |
| 0.7 | N/A | 16/03/23 | Added class mapping to requirements and significant data structures | jac127 |