Software Engineering GP02 Project

Design Specification

|  |  |
| --- | --- |
| Authors: | Jack Book [jab153], Thomas Roethenbaugh [tpr3], Will Abbott [wia14] |
| Config Ref: | SE.G02.DesignSpec |
| Date: | 27th February 2023 |
| Version: | 0.5.2 |
| Status: | Draft |

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Copyright © Aberystwyth University 2023

CONTENTS

CONTENTS 2

1. Introduction 3

1.1 Purpose of this Document 3

1.2 Scope 3

1.3 Objectives 3

2. Decompisition description 3

2.1 Programs in System 3

2.2 Significant Classes 3

2.3 Modules shared between programs 5

2.4 Table mapping requirements onto classes 5

3. dependency description 6

4. interface description 6

4.1 Game Package 6

4.2 GUI Package 7

4.3 Moves Package 8

4.4 Pieces Package 9

4.5 Tests Package 11

4.6 Util Package 15

5. detailed design 16

5.1 UML sequence diagrams 16

5.2 Significant algorithms 16

5.3 Significant data structures 17

REFERENCES 17

DOCUMENT HISTORY 18

# Introduction

## Purpose of this Document

The purpose of this document is to give insight into how the game will be designed. It will describe the decomposition of the programs and the modules that fall within them, including any significant classes and how the classes overlap with the functional requirements. It will also describe the relationships and dependencies the modules have between each other. As well as, providing a description of the interfaces used and how any programmers or testers will use the facilities provided by a module. Finally, it will lay out the internal details of any modules that are not obvious to any reader.

## Scope

The document should be read by all members of the project and any readers should have a good understanding of the User Interface Specification document.

## Objectives

The objective of this document is to allow the reader to understand the inner mechanisms of the classes used in the project and go into further detail of how more complicated classes work.

# DecompOsition description

## Programs in System

The system as designed only contains one program. This singular program handles both the logic and graphics on the game in several packages and classes. The backend logic being handled by the game and moves package classes using Java objects. The frontend by the GUI package classes using JavaFX, testing by JUnit in the testing package and finally, saving and loading handled by the TO BE ADDED package.

## Significant Classes

The program is broken down into separate packages, each package containing the relevant classes and material needed to run the program. Each of the sections below will describe the packages and briefly explain the classes used.

### Game

The game package contains four classes – Board, Game, Player, and Square, these are used to build the game.

**Board** - creates the chessboard and for testing purposes is used to print the array to the console.

**Game -** collects the relevant information from other classes and holds the settings for the current game in progress.

**Player** - collects the information regarding each player, e.g., name, color, captured pieces.

**Square** - is responsible for looking after each cell on the game board and what those cells are doing.

### Pieces

The pieces package has seven classes – Bishop, King, Knight, Pawn, Piece, Queen, and Rook. These classes contain the material for each specific piece, they all work in similar ways apart from the valid moves that determine what each valid move for the pieces are and allows the movement to take place.

### Moves

The moves package contains four classes – Castle, enPassant, CheckChecker, and makeMove. Both the Castle and enPassant classes are considered ‘special moves’ that are not called as often as the other classes.

**Castle** – This class is for checking if either the black king or white king can castle.

**EnPassant** – This class holds the method to get the valid moves for enPassant.

**CheckChecker** – This class is responsible for checking whether either of the kings is currently in check.

**makeMove –** This class holds the methods for moving a piece on the chessboard. It contains methods that are specifically used for making the move enPassant and Castling.

### Tests

The tests package is split into 3 different sections – game tests, moves tests and pieces tests.

**Game Tests:**

**TestBoard –** This class includes methods that test that each piece on the board is in the correct position at the start of the game. It also checks to see if the black and white pieces are in the correct places, as well as checking if there are no extra pieces on the board

**TestGame –** This class includes methods that check the set name functionality, check that white goes first, checks that the start round is 1, checks the nextRound method is working correctly, and the setTurn method is working correctly

**TestPlayer -**

**TestSquare**

**Moves Tests:**

**TestCastle**

**TestEnPassant**

**TestMakeMove**

**Pieces Tests:**

**TestBishop**

**TestKing**

**TestKnight**

**TestPawn**

**TestQueen**

**TestRook**

### Util

The classes within this package simply define the colors and pieces available within the chess game.

**Color –** This class simply defines the two available colors within chess: Black and White.

**Type –** This class defines the pieces available within chess. These pieces are King, Queen, Rook, Bishop, Knight and Pawn.

### GUI

This package includes the game Gui class as well as the image used in the background of the menu, and all the images for each piece on the board

**GameGUI –** This is the class that runs the main graphical user interface for the game. This class will show the game in progress and display the game in the correct manner, for the game to run correctly, in accordance with the users’ inputs. It includes 2 methods. The first initializes the main menu, displaying the text boxes for the users’ names, the selection box for assigning the user colors, and any buttons that are needed in accordance to the User Interface Design Specification[2]

**/\*\* These sections will be removed, changed the layout but still using the info in them \*\*/**

### Board

This class implements the game board and the graphical section of the program, it builds the individual squares that the pieces are placed on. It also gives the location and the places that pieces in their starting location.

### Pieces

There are different classes for each individual pieces on the board, these classes lay out the valid moves the pieces can take and determines how these pieces move throughout the board.

### Player

The player class holds the information connected to the two individual players and what colour they are playing with.

### Main

The main class connects all the other classes and launches the GUI and starts the game.

### Square

The square class is responsible for checking what piece or if no pieces are in the square that is called.

## Modules shared between programs.

## Table mapping requirements onto classes

|  |  |
| --- | --- |
| Functional Requirements | Classes providing requirement |
| FR1 | Board, Game, Player, Square |
| FR2 | Player, Game |
| FR3 | Board |
| FR4 | makeMove, Bishop, King, Knight, Pawn, Piece, Queen, Rook |
| FR5 | makeMove, Bishop, King, Knight, Pawn, Piece, Queen, Rook, Castle, enPassant, CheckChecker |
| FR6 | CheckChecker |
| FR7 | CheckChecker |
| FR8 | empty |
| FR9 | empty |
| FR10 | empty |
| FR11 | empty |

# dependency description

# interface description

## Game Package

### Board

* Type: Public
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | Board() | Constructor. |
| Array | getBoardArray() |  |
|  | getEnPassantPiece() |  |
|  | setEnPassantPiece(int[] enPassantPiece) |  |
|  | boardReset() |  |
|  | clearBoard() |  |
|  | printBoard() | Used for debugging, prints the array to the console. |

### Game

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | Game() | Constructor. |
|  | getBoard() |  |
|  | setWhitePlayer(String name) |  |
|  | setBlackPlayer(String name) |  |
|  | getWhitePlayer() |  |
|  | getBlackPlayer() |  |
|  | getTurn() |  |
|  | setTurn(Color turn) |  |
|  | getRoundCount() |  |
|  | nextRound() |  |

### Player

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | Player(Color color, String name) | Constructor. |
|  | getColor() |  |
|  | getName() |  |
|  | getTakenPieces() |  |
|  | addTakenPieces(Piece piece) |  |

### Square

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | Square() |  |
|  | Square(Piece piece) |  |
|  | setPiece(Piece piece) |  |
|  | isHasPiece() |  |
|  | setHasPiece(boolean hasPiece) |  |

## GUI Package

### Empty

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | GameGui(Stage stage) | Constructor. |
|  | initMainMenu(Stage primaryStage) |  |

## Moves Package

### Check Checker

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| Array List | isInCheck(Board board, int x, int y) | Returns a list of coordinates of pieces that are attacking the king |

### Make Move

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | movePiece(Board board, Player player, int pieceX, int pieceY, int targetX, int targetY) | A method to move a piece on the chessboard. |
|  | movePiece(Board board, Player player, int pieceX, int pieceY, int[] targetX, int[] targetY) | A method to move a piece with en passant on the chessboard. |
|  | movePiece(Board board, Player player, int[] pieceX, int[] pieceY, int[] targetX, int[] targetY) | A method to move a piece with castling on the chessboard. |

### Castle

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | canCastle(Board board, Piece piece, int x, int y) | Checks whether a king can castle. |

### En Passant

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | getMoves(Board board, Piece piece, int x, int y) | A method to return a List of valid move coordinates |

## Pieces Package

### Piece

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | Bishop(Color color) | Constructor. |
|  | getPieceColor() | Gets the piece color. |
|  | getEnemyPieceColor() | Gets the color of the opponent’s piece. |
|  | getPieceType() | Gets the type of piece. |
|  | isHasMoved() | Gets the hasMoved field of the piece. |
|  | setHasMoved() | Sets the hasMoved field of the piece to true. |
| Array List | getValidMoves(Square[][] boardArray, Piece piece, int x, int y) | A method to return a List of valid move coordinates. |
| Array List | addValidMove(Square[][] boardArray, Piece piece, int x, int y) | A method to add a move to a List if it is valid. |

### Bishop

* Type: Public
* Extends: Piece
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | Bishop(Color color) | Constructor. |
| Array List | getValidMoves(Square[][] boardArray, Piece piece, int x, int y) | A method to return a List of valid move coordinates |

### King

* Type: Public
* Extends: Piece
* Public Methods:

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | King(Color color) | Constructor. |
| Array List | getValidMoves(Square[][] boardArray, Piece piece, int x, int y) | A method to return a List of valid move coordinates |

### Knight

* Type: Public
* Extends: Piece
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | Knight(Color color) | Constructor. |
| Array List | getValidMoves(Square[][] boardArray, Piece piece, int x, int y) | A method to return a List of valid move coordinates |

### Pawn

* Type: Public
* Extends: Piece
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | Pawn(Color color) | Constructor. |
| Array List | getValidMoves(Square[][] boardArray, Piece piece, int x, int y) | A method to return a List of valid move coordinates |

### Queen

* Type: Public
* Extends: Piece
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | Queen(Color color) | Constructor. |
| Array List | getValidMoves(Square[][] boardArray, Piece piece, int x, int y) | A method to return a List of valid move coordinates |

### Rook

* Type: Public
* Extends: Piece
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | Rook(Color color) | Constructor. |
| Array List | getValidMoves(Square[][] boardArray, Piece piece, int x, int y) | A method to return a List of valid move coordinates |

## Tests Package

### Test

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | checkTest() | Tests for Check. |

### Test Board

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | rookLocationsAllCorrect() |  |
|  | knightLocationsAllCorrect() |  |
|  | bishopLocationsAllCorrect() |  |
|  | queenLocationsAllCorrect() |  |
|  | kingLocationsAllCorrect() |  |
|  | pawnLocationsAllCorrect() |  |
|  | whiteLocationsAllCorrect() |  |
|  | blackLocationsAllCorrect() |  |
|  | noExtraPiecesOnBoard() |  |

### Test Game

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | whitesNameIsJohn() |  |
|  | blacksNameIsSmith() |  |
|  | initialTurnIsWhite() |  |
|  | startRoundIsOne() |  |
|  | testNextRoundMethod() |  |
|  | testSetTurnMethod() |  |

### Test Player

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | testPlayerObject() |  |
|  | pawnPieceTaken() |  |
|  | rookPieceTaken() |  |
|  | knightPieceTaken() |  |
|  | bishopPieceTaken() |  |
|  | queenPieceTaken() |  |
|  | kingPieceTaken() |  |
|  | testAllPossibleCaptures() |  |

### Test Square

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | constructSquareNoPiece() |  |
|  | constructSquareWithPiece() |  |
|  | setPieceOnSquare() |  |

### Test Make Move

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | movePieceToEmptySpace() |  |
|  | movePieceOffBoard() |  |
|  | movePieceToVacatedSpace() |  |

### Test Castle

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | castleFunctional() |  |
|  | tryToCastleButKingMoved() |  |
|  | tryToCastleButCastleMoved() |  |
|  | ryToCastleButKingChecked() |  |

### Test En Passant

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | enPassantFunctional() |  |
|  | enPassantMissedChance() |  |

### Test Bishop

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | testNoValidMoves() |  |
|  | testCapture() |  |
|  | testMove() |  |
|  | testCheck() |  |

### Test King

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | testNoValidMoves() |  |
|  | testCapture() |  |
|  | testMove() |  |
|  | testCheck() |  |

### Test Knight

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | testNoValidMoves() |  |
|  | testCapture() |  |
|  | testMove() |  |
|  | testCheck() |  |

### Test Bishop

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | testNoValidMoves() |  |
|  | testPromotion() |  |
|  | testCapture() |  |
|  | testMove() |  |
|  | testCheck() |  |

### Test Queen

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | testNoValidMoves() |  |
|  | testCapture() |  |
|  | testMove() |  |
|  | testCheck() |  |

### Test Rook

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | testNoValidMoves() |  |
|  | testCapture() |  |
|  | testMove() |  |
|  | testCheck() |  |

## Util Package

### Color

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | Color |  |

### Type

* Type: Public
* Extends: Nothing
* Public Methods

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
|  | Type |  |

# detailed design

## UML sequence diagrams

### Main menu UML (UC01 to UC05)

Diagram

Description automatically generated

### Game UML (UC06 to UC19)

## Significant algorithms

### makeMove

The make move algorithm handles the movement of chess pieces across the boardArray, requiring inputs for both the starting and targeted row and column coordinates. Depending on the rules of the piece selected, as well as whether or not the move is ‘special’ (like Castling or enPassant) it will then

### Castling

For castling this has been put in a special moves set. This class will return a list of integer arrays with each integer array containing the position the king must move to in order to castle, the position of the rook it will be castling with, and the position the rook will be moved to, in that order.

### enPassant

This particular move requires some unique rules that make it difficult to keep its move set in the Pawn class. The method of determining if a pawn can complete this move will kept in a separate class that will first return a board array to check if the move is possible. If enPassant is possible, the method will return both a move position and a remove piece position on the board. Allowing those two squares to be used for both moving and removing pieces on the board should the player choose this move.

### Update GUI

For each turn in the game the GUI for the chess board needs to be updated. In order to avoid slowdown or stuttering…

### Test game

In order to properly unit test the program, a simulated chess game will required to show that all of the features implemented in the program are working. The creation and simulation of a new chess game will be required in order to demonstrate how effectively each aspect of the program will work.

## Significant data structures

### boardArray

The board array data structure is a 2D array (matrix) of Square objects. Each square acts as a container for each part of the board which can contain any type of game piece or be empty. The board array does not handle any of the logic of the chess game, only acting as a ‘flat surface’ from which the chess game will be played on.

### Valid moves

Each piece on the board will contain their own arrayLists of the valid moves they can take, which can then be used by the makeMove class to help its calculations for where the piece can be moved to.

### Saving & loading

For saving and loading games, both for continuing running games as well as replaying previous game each step in the sequence of moves will be stored in an XML file…

REFERENCES

[1] Software Engineering Group Projects: General Documentation Standards. C.W. Loftus. SE.QA.05. 2.3 For Release

[2] Software Engineering Design Document (correct formatting needed here)

DOCUMENT HISTORY

| *Version* | *Issue No.* | *Date* | *Changes made to document* | *Changed by* |
| --- | --- | --- | --- | --- |
| 0.1 | N/A | 27-02-2023 | N/A - original version | JAB153 |
| 0.2 | N/A | 07-03-2023 | Added interface descriptions | JAB153 |
| 0.3 | N/A | 08-03-2023 | Added UML sequence diagram and started significant algorithms | TPR3 |
| 0.4 | N/A | 09-03-2023 | Added interface descriptions | JAB153 |
| 0.5.1 | #43 | 11-03-2023 | Finished first UML sequence diagram as well added more descriptions to significant algorithms | TPR3 |
| 0.5.2 | N/A | 15-03-2023 | Added more significant classes | WIA14 |