Software Engineering Group Project

Design Specification

|  |  |
| --- | --- |
| Author: | George Cooper [gwcl], Kieran Foy [kif11], James Owen [jco3], Tate Moore [tam41] |
| Config Ref: | SE\_gp17\_Design-Specification |
| Date: | 13th March 2023 |
| Version: | 0.8 |
| 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. Decomposition Description 3

2.1 Programs in system 3

2.2 Significant classes in each program 4

2.3 Mapping from requirements to classes 4

3. DEPENDENCY DESCRIPTION 5

3.1 Component Diagram 5

4. INTERFACE DESCRIPTION 5

4.1 Game 5

4.2 Setup 6

4.3 Piece 6

4.4 Bishop 6

4.5 King 6

4.6 Knight 6

4.7 Pawn 6

4.8 Queen 6

4.9 Rook 7

4.10 Replay 7

4.11 Quit 7

5. DETAILED DESIGN 7

5.1 Sequence diagrams 7

5.2 Significant algorithms 7

5.3 Significant data structures 7

REFERENCES 7

# Introduction.

## Purpose of this Document

The purpose of this document is to outline the design specifications for the Chess Tutor Application, to be presented along with the final draft of the design presentation in week beginning 20th March 2023.

## Scope

This document displays the design specification created by Group 17, covering all sections named in SE.QA.05 [1] and following the layout provided in SE.QA.02 [2]. These two documents should be read by all project members working on this document before they begin working on it and should refer to said documents if needed.

This document should be read by all project members before the day of the design final draft presentation (week beginning 30th March 2023.

## Objectives

The objectives of this document are:

* To justify and describe the breakdown of the system into classes inside a singular program and its purpose, as well as that of each significant class’. 2
* Show that the functional requirements have been met by relating them to the classes. 2
* Using a component diagram, create an understanding of the relationships and dependencies between modules, or in other words how parts of the system fit in together. 3
* Give an outline specification for each class for a developer to understand them. 4
* Show the relationships between classes using UML class diagrams and how they work together using UML sequence diagrams in addition to how they satisfy the use cases [3]. 5
* Explain the planning of difficult or significant algorithms using pseudocode. 5

# Decomposition Description

## Programs in system

The system is made up of a singular program. The program will allow 2 users to play a game of chess on a single computer.

When the program loads up, they will face a main menu scene with 3 JavaFX Buttons: ‘Quit’. ‘Load old game’ and ‘Start new game’.

Left clicking button...

* ‘Quit’ will exit the program.
* ‘Load old game’ will take you to a new load game scene.
  + Use JavaFX FileChooser to pick an old game file to load in and you will be able to either continue an old unfinished game or replay a finished one.
  + Left click three JavaFX buttons to either confirm or unconfirm a chosen file or to go back to the main menu.
* ‘Start new game’ will take you to a new game scene
  + Use two JavaFX TextFields for inputting player names, one for black and the other white, which is which will be shown with JavaFX labels. This allows players to remember if they are black or white.
  + Left click two JavaFX buttons to either confirm the chosen names or to go back to the main menu.

Confirming and successfully loading an old game or starting a new one will take the players to a new scene where it will display the current chessboard state to the users.

The program will keep track of game statistics including:

* Whose turn it is.
* The colour and name of each player.
* The position of the pieces on the chess board.
* The history of all past moves.
* The taken pieces for each colour.

For an unfinished game, at the start of each new turn the player – whose turn it is – will be prompted to select a piece using right click. Once selected, the program will display all legal moves. The player will then choose a legal move by left clicking on its square or can deselect the piece by left clicking anywhere else.

If a user’s king is in…

* Check – the program will alert the payers.
* Checkmate – the program will alert the players and display the winner. The program will give the users the option to save the game.

The program will allow the users to save and quit the game at any time. Saved games will be automatically saved in case of a crash.

For a finished game, the user will be allowed to go forwards and backwards through the game.

## Significant classes

|  |  |
| --- | --- |
| Class Name | Description |
| Setup | Setup will handle all methods required to setup the game for the user. The class will handle creating a new game or restoring a previous game. The class will build the board using JavaFX, handle usernames and colour choice. The class will also handle restoring a game if the program crashed. |
| Game | Game will handle all systems required to run the chess match. These include piece movement, detecting checks and piece removal. |
| Save | Save will handle all functions relating to saving. The class will handle storing all moves in the game to allow for the user to restore and step through the game in its entirety. |
| Replay | Replay will handle loading and displaying previously played games. It will hand systems required to allow the players to step through previously played games. |
| Piece | Piece will act as a superclass to all specific piece classes. It will include attribute shared by all chess pieces. |
| Pawn | Pawn is a subclass of Piece. It will act as a blueprint for all pawn pieces in the game. The class will store pawn specific attributes such as legal moves, model and point worth. |
| Knight | Knight is a subclass of Piece. It will act as a blueprint for all knight pieces in the game. The class will store knight specific attributes such as legal moves, model and point worth. |
| Bishop | Bishop is a subclass of Piece. It will act as a blueprint for all bishop pieces in the game. The class will store bishop specific attributes such as legal moves, model and point worth. |
| Rook | Rook is a subclass of Piece. It will act as a blueprint for all rook pieces in the game. The class will store rook specific attributes such as legal moves, model and point worth. |
| Queen | Queen is a subclass of Piece. It will act as a blueprint for all queen pieces in the game. The class will store queen specific attributes such as legal moves, model and point worth. |
| King | King is a subclass of Piece. It will act as a blueprint for all king pieces in the game. The class will store king specific attributes such as legal moves, model and point worth. |

## Mapping from requirements to classes

|  |  |
| --- | --- |
| **Requirement [4]** | **Classes providing requirement** |
| FR1 | Setup, SetupController, NewGame, NewGameController |
| FR2 | Game, Save |
| FR3 | Game, GameController |
| FR4 | Game |
| FR5 | Game, GameController, Pawn, Knight, Bishop, Rook, Queen, King |
| FR6 | Game, GameController, King |
| FR7 | Game, King |
| FR8 | Game, EndScreenController, Quit |
| FR9 | Quit |
| FR10 | Replay |
| FR11 | Replay |

# DEPENDENCY DESCRIPTION

## Component Diagram

*Figure 1: Component Diagram of the Chess Tutor Game System to show the relationships between modules.*

# INTERFACE DESCRIPTION

## Game

* Type: public
* Extends: nothing
* Public methods:
  + checkOrCheckmate() – detects if the King of the opposite colour of the player who just had their turn is in check or checkmate
  + selectPiece(int xPos, int yPos) – highlights a piece on the board and displays its current moves.
  + deselectPiece(int xPos, int yPos) – removes highlighting of a selected piece and stops displaying its legal possible moves.
  + movePiece(int xPos, int yPos) – changes the position of a piece to a given new legal position.
  + removePiece(int xPos, int yPos) – removes a piece from the board when another piece legally captures/takes it.

## Setup

* Type: public
* Extends: nothing
* Public methods:
  + whiteName() – gets user input for the name of the white player.
  + blackName() – gets user input for the name of the black player.
  + boardSetup() – sets up the chessboard array that the game will be played on. Called everytime a new game is started.
  + addPiecesToBoard() – adds all pieces to their starting positions on the chessboard array.

## Piece

* Type: public
* Extends: nothing
* Public methods:
  + Piece(String colour, int xPos, int yPos, int score) – constructor.

## Bishop

* Type: public
* Extends: Piece
* Public methods:
  + Piece(String colour, int xPos, int yPos) – constructor.

## King

* Type: public
* Extends: Piece
* Public methods:
  + King(String colour, int xPos, int yPos) – constructor.

## Knight

* Type: public
* Extends: Piece
* Public methods:
  + Knight(String colour, int xPos, int yPos) – constructor.

## Pawn

* Type: public
* Extends: Piece
* Public methods:
  + Pawn(String colour, int xPos, int yPos) – constructor.

## Queen

* Type: public
* Extends: Piece
* Public methods:
  + Queen(String colour, int xPos, int yPos) – constructor.

## Rook

* Type: public
* Extends: Piece
* Public methods:
  + Rook(String colour, int xPos, int yPos) – constructor.

## Replay

* Type: public
* Extends: nothing
* Public methods:
  + saveGame(String fileName) – saves the game data to a given save file (fileName) including the
    - Type, colour and position of the pieces on the board.
    - Type and colour of taken pieces.
    - History of piece moves.
  + loadGame(String fileName)) – loads the game data from a given save file (fileName)
  + addMove(?) – adds a move to the history of past moves.

## Quit

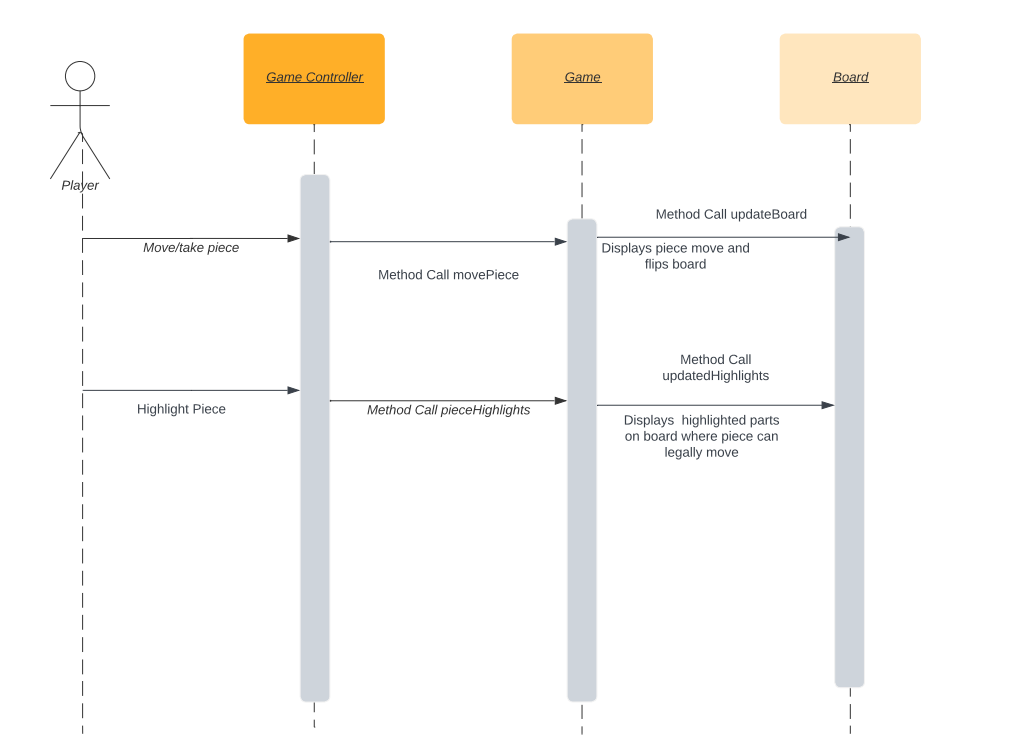
* Type: public
* Extends: nothing
* Public methods:
  + quitGame() – ends the game when players decide to quit.
  + gameFinished() – calls functions for when the game ends in checkmate, a mutual draw occurs or a player resigns.

# DETAILED DESIGN

## Sequence diagrams

Game Menus Sequence diagram:

Piece Sequence Diagram:



## Significant algorithms

## 5.2.1 Move Generation

As stated in FR5 pieces must display all legal moves when highlighted. In chess there are some notable exceptions to the normal piece behaviours we will have to run checks on when displaying these moves.

**Check/Disccovered check –** If the king will get checked it will not display it as a valid move. The program will:

Find the king on the board, Check if any of the opponents pieces can attack the king, do checks to check if any of the opponents pieces can legally check the king using a function for each piece to check if it’s a valid move or not.

Function is\_in\_check:

Find players king on the board

Check if any opponents pieces can attack the king

Funcion is valid move:

Check what piece it is attacking the king

Run through checks seeing if it’s a legal/valid move

**En Passant –** When a pawn is moved two squares a flag will be brought up showing en passant will be legal next turn

## Significant data structures

--The data structure used to store games

--Data structure used to store usernames and passwords

REFERENCES

[1] Loftus C.W., “Software Engineering Group Projects –

Design Specification Standards”, 2.3, Computer Science Department, 6th December 2021

[2] Loftus C.W., “Software Engineering Group Projects –

General Documentation Standards”, 2.6, Computer Science Department, 10th January 2023

[3] Loftus C.W., “Software Engineering Group Projects –

User Interface Specification Standards”, 1.4, Computer Science Department, 10th January 2023

[4] Loftus C.W., “Software Engineering Group Project - Chess Tutor Requirement Specification”, 1.2, Computer Science Department, 28th February 2023

DOCUMENT HISTORY

| *Version* | *Issue No.* | *Date* | *Changes made to document* | *Changed by* |
| --- | --- | --- | --- | --- |
| 0.1 | N/A | 7/02/23 | N/A - original draft version | GWC1 |
| 0.2 | 11 | 14/02/23 | Filled out section 1: Introduction | KIF11 |
| 0.3 | 11 | 14/02/23 | Filled out section 2: Decomposition Description | GWC1 |
| 0.4 | 11 | 04/03/23 | Adjusted section 2: Decomposition Description  Added almost finished diagram (Figure 1) | TAM41 |
| 0.5 | 11 | 05/03/23 | Added to section 4: Interface Description | TAM41 |
| 0.6 | 11 | 07/03/23 | Added in references | TAM41 |
| 0.7 | 11 | 13/03/23 | Added to section 5: Detailed Design | JCO3 |
| 0.8 | 11 | 13/03/23 | Adjusted sections to review meeting feedback | TAM41 |