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\_01 |
| Date: | 7th March 2023 |
| Version: | 0.6 |
| Status: | Ready for Review |

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 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 CS22120 Software Engineering Group Project 2022/23, 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 give a detailed description of the Chess Tutor program design and how Group 17 went about designing the classes, interfaces and dependencies.~~
* ~~To detail our plan of how to implement the functional and non-functional requirements according to the clients’ specifications.~~
* ~~To describe how each requirement is met through use-case and sequence diagrams, as well as annotated images of the several user interfaces that have been/will be implemented.~~
* To justify and describe how the system is broken down into classes in a single program.
* Deliver a description of what the singular program shall do.
* Briefly describe the purpose of each of the most important classes.
* Using a component diagram, create an understanding of the relationships and dependencies between modules, or in other words how parts of the system described in the section 2 fit in together.
* To indicate how the classes work together to accomplish the tasks of the use cases [] through UML sequence diagrams.
* Explain the planning of difficult or significant algorithms using pseudocode.
* Display the links between classes with UML class diagrams.
* Express in object diagrams how static relationships …

# 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. The users will play chess using one program on one computer screen.

The program will display a setup menu that will allow users to start a new game or restore a previous game.

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.

After setting up a game, the program will display the current board state to the users. At the start of each turn the appropriate player will be prompted to select a piece. Once selected, the program will display all legal moves. The player will only be allowed to move if the move is legal.

If a user’s king is in check, the program will alert the payers.

If a user’s king is in checkmate, the program will alert the players and display the winner. The program will prompt the user 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.

In addition, the program will allow the users to load any previously saved game. The program will display the board state and allow the users to move forward or backwards through the game.

## Significant classes in each program

|  |  |
| --- | --- |
| 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. |
| Main | Main will call the program. |
| 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 knight 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** | **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*

# 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

## Significant algorithms

## Significant data structures

REFERENCES

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

2.3 Release

[2] Software Engineering Group Projects: General Documentation Standards. C.W. Loftus. SE.QA.02. 2.6 Release 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 |