# Software Engineering Group Project
# Maintenance Manual

| | |
|---|---|
| Author: | Kacper Dziedzic [ktd1], Jasper Crabb [jac127] |
| Config Ref: | MMGroup18 |
| Date: | 11th May 2023 |
| Version: | 1.0 |
| Status: | Release |

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

## CONTENTS

# 1. INTRODUCTION

### 1.1 Purpose of this Document

This document will be used the client’s developers as a guide on how to maintain the program.

### 1.2 Scope

This document is reliant on the reader knowing the structure and inner workings of the program. Therefore it is recommended that the reader familiarises themselves with the design specification [1].

### 1.3 Objectives

The objective of this document is to guide the maintainer through our program and highlight things that should be taken into consideration when maintaining the program.

# 2. THE PROGRAM

### 2.1 Program Description

This program allows the user to play a standard game of chess locally with one other player on the same computer. The program will save the progress allowing the user to carry on games mid progress or replay games.

### 2.2 Program Structure

Flow of Control: Refer to Design Specification – 5.1 Sequence Diagrams.

Purpose of Modules: Refer to Design Specification – 2.2 Decomposition Description.

List of Methods: Refer to Design Specification – 4 Interface Description.

### 2.3 Algorithms

Refer to Design Specification – 5.2 Significant Algorithms.

### 2.4 Data Areas

Refer to Design Specification – 5.4 Significant Data Structures.

### 2.5 Files

The program uses a necessary “resources” folder within the project folder which stores all the necessary image files for the program to access and use. It also stores the javafx package locally meaning we don’t need to download and import it every time and automatically have the package linked with the program.
The “resources” folder holds a folder within called “saveGames” which is the file location for all the games saved produced by playing chess on the program, these files are individually named using the player names then the year, month, day, hour and minute to produce a unique file name which is easy for the user to find again in the future.

When loading a game, the user will be placed within the saveGames directory by default. However, if the player uses an invalid file or a file name that doesn't exist, the program will default to using the standard board as well as a newly generated file name to save the new game to.
Deleting a file can be done manually with no issues but can also be done manually using the “erase” button on the load game menu, if the directory entered or the file selected matches a file in the “saveGames” folder it will be permanently removed.

### 2.6 Interfaces

Our program does not contain any interfaces.

For more information: Refer to Design Specification – 4 Interface Description.

# 3. ALTERING THE PROGRAM

## 3.1 Suggestions for Improvements

### 3.1.1 Desirable Changes Which Could not be Made

Redo the order of operations on how valid moves are obtained.
Currently all the valid moves are generated for each individual piece, then filtered by cloning the board, making the move on that board then removing any of the invalid moves for each piece. This can be made simpler by generating the moves then immediately filtering them based on whether they place your king in check rather than having a separate method for filtering all the moves on the board.
Improve how check/checkmate is detected:
Currently we require all the valid moves generated and filtered and only after that is all done if a player has 0 valid moves it results in checkmate and if any of the valid moves go over the opposing king it results in check for the other player. Instead of updating the moves for every piece, update the moves based on the piece moved, store a grid of danger levels for every tile, if a piece moves and reveals the king to a high enough danger level tile then disallow that move.
Highlight when king is in check on replay:
Due to time constraints, we didn’t have the time to add the visual display showing a piece currently in check. This is a simple addition as a method in the Game class can be altered to find out if a king is in check without completing the rest of the operations during a normal turn.

### 3.1.2 Desirable Changes Which Could not be Implemented

Change how some of the classes interact with each other:
Some classes are insignificant and don't need to be separate such as the Check class as well as the “replayBoard” class. The “replayBoard” all the same methods from the chessboard class but with a few minor changes by using a couple methods to initialise either a chessboard or a replay board within the chessboard class the entire “replayBoard” class can be removed. The check class is one small method which does not require its own class to be functional and clear.
Change how castling moves are validated:
A method was added to the king class which replicates a method within the piece class which tests a move on a clone board to see if making this move places your king in check. A replica method was added to test if castling would place your own king in check which isn't necessary if the order of operations was changed to getting all normal moves, then getting all the special moves (En passant and castling)

### 3.1.3 Known Bugs

Change the load all fen strings method:
There is currently a bug present which loads the Fen strings incorrectly for the replay and loading. This has not been tested much but should be a simple fix.

Change code in game method:
Currently loading is completely broken and this is due to incomplete code within the game constructors and "initializeGame" method which need to be changed to function with the replay board as well as the Fen String board position loader as well as the save file loader.
Not showing king in check when loading a game:
If a game is loaded where a king is initially in check, the chessboard will not visually display the king in check until a mouse click event is detected. This is due to the chessboard class having some missing initial code to fix this issue to adapt to loading.

### 3.2 Things to Watch Out For

- "filterAllMoves" method in the board class:
When changing any of methods to generate any moves within any of the pieces, it will be used within this method, if there are any bugs when generating any moves and it's inconsistent on the board, this method will be the first thing to break the program as it will pick up any generate moves inconsistencies on the board.
- Changes to the board class or the pieces:
When adding or removing any new variables to the board class or piece class and its sub classes, it's crucial that these changes are added to the following methods: loadBoardFromFen, saveBoardToFen, cloneBoard.
If the board changes inbetween turns and the board is missing information, this could cause major bugs within the code especially when generating all moves + filtering all moves due to a lack of consistency when cloning the board.

### 3.3 Physical Limitations

The minimum specifications for the program to run effectively is a single core processor with 1 GHz clock speed. However, this has not been fully tested and the minimum processor requirements are likely to be much lower.
The program also requires a minimum screen resolution of 1280 x 720 otherwise it doesn't fit the entire tab on the screen as resolution adaptability has not been added to the core of the program, 1280 x 720 is small enough resolution to fit on most modern screens which makes this an unlikely outcome.
There are no other hardware requirements.
If the program is built and executed from the respective versions of IntelliJ on each OS, it should have no problems executing but there may be issues with the javafx package not working on operating systems other than Windows.

## 4. REBUILDING AND TESTING

- All of the necessary files for the program are located in the gp18 folder with the necessary 3 folders being the "src" folder holding all of the classes being used for the project, the resources folder holding all of the files the program is loading from and saving including the javafx package and the last folder being the tests folder holding all of the tests we conducted on the program itself.
- We used IntelliJ to develop the software, to rebuild the entire project locally you would first create a "gp18" folder somewhere on your computer. Then using the HTTP GitLab clone URL, using IntelliJ clone the project using the URL with the destination set to the gp18 folder. Once the project is cloned it will automatically open, the next step will be to go into the project settings and set the JDK version to amazon coretto 20. Next, you can navigate to the "src" folder and the Main class within it, once the project has finished indexing, you should be able to click run in the main class, Once you try to run it, immediately enter the runtime configuration settings and enable VM options, then add the following text: "--module-path resources/javafx-sdk-20.0.1/lib --add-modules javafx.controls,javafx.fxml". After adding that, try and run from main again and it should build the project and run successfully. If it does not run from there, try reinstalling javafx for your corresponding OS, linking it then re-trying.
- Testing files can be found within the "tests" folder and use the Junit package to execute tests in an organised fashion. Tests were made for most of the backend code and will confirm whether certain parts of the code are working correctly.

- The tests were written using the Junit package, this means the user can simply run them to confirm whether certain parts of the code are working as intended or to find out if a specific test passes. When the tests are run the results will be immediately highlighted as either a PASS or a FAIL.
- If a new problem is discovered and is causing one or more tests to fail, the best option would be to place a break point before the line thought to be the root cause of the issue and run the test in debugging mode, tracing through the program to find the cause of the bug and what to do to fix it within the main code.

## REFERENCES

[1] Design Specification/1.4(Release)

## DOCUMENT HISTORY

| *Version* | *Issue No.* | *Date* | *Changes made to document* | *Changed by* |
|---|---|---|---|---|
| 0.1 | N/A | 11/05/23 | N/A - original version | Ktd1 |
| 0.2 | N/A | 11/05/23 | Formatting Changes | Jac127 |
| 1.0 | N/A | 11/05/2023 | Release | Jac127 |