Software Engineering Group Project

Maintenance Manual

Authors:	Jack Book [jab153], Will Abbott [wia14], Bilal Yousufzai [biy1], Micah Barendse [mib60], Michael Stamp [mjs36], Abdullah Durrani [abd15]
Config Ref:	SE.G02.MaintenceManual
Date:	10th May 2023
Version:	1.2
Status:	Release

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

1. INTRODUCTION

1.1 Purpose of this Document

This document is the maintenance manual for the program, it is designed to allow a program maintainer to be guided to the specific part of the program that might need changing due to updates or bug fixing. It gives an overview of how the program works and where to find more detailed information related to particular areas of the program contained in other documents.

1.2 Scope

This document should be understood by all principal programmers and any future program maintainers.

1.3 Objectives

The objective of this document is to give a clear outline to future program maintainers.

2. PROGRAM DESCRIPTION

This program allows a user to learn how to play chess, whilst taking turns against an opponent who can play and understands the rules. It allows the user to understand how pieces move, as well as the specific scenarios that can lead to special moves being played. The User Interface is also intuitive enough to help the user to understand how and where each piece can move [2].

3. PROGRAM STRUCTURE

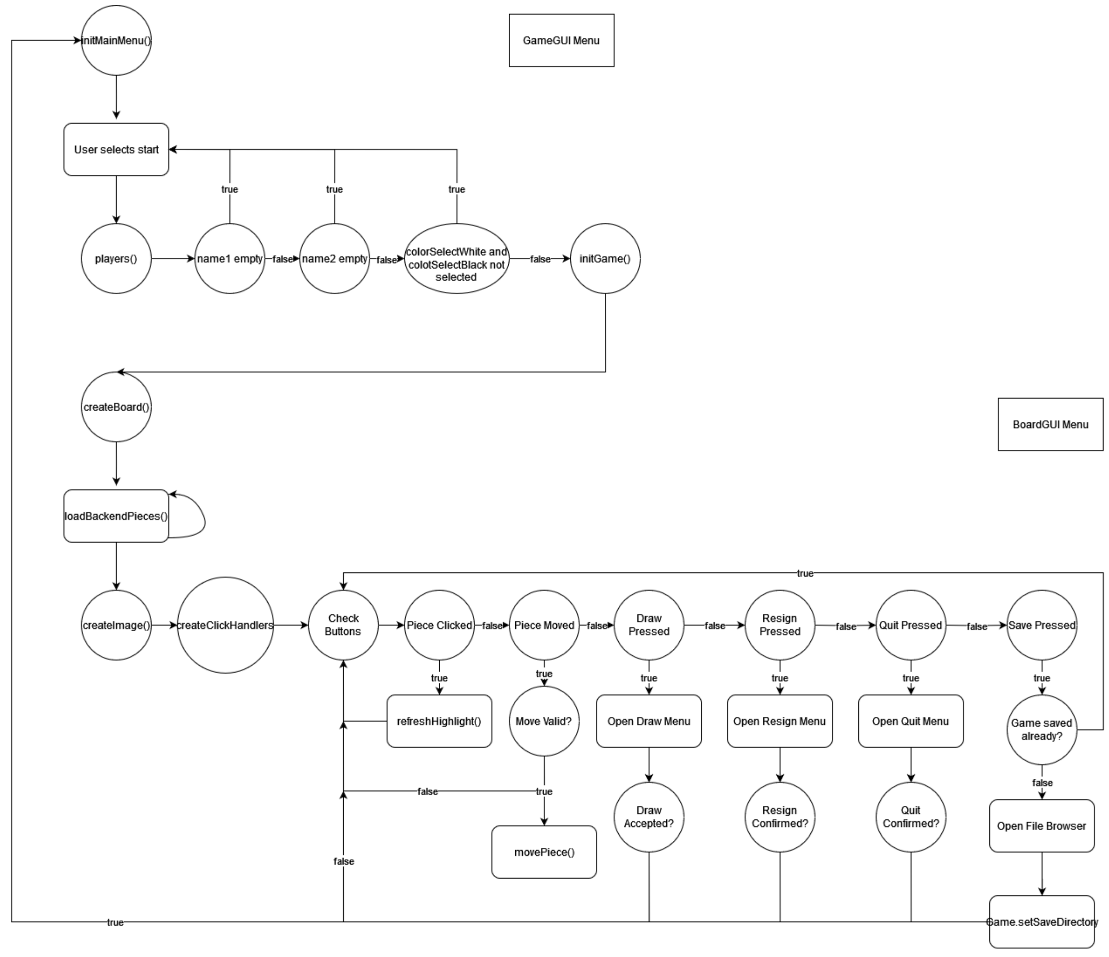

Figure 1 - Flow of control Diagram

Methods listed in Section 4 of Design Specification Document [1].

4. ALGORITHMS

Significant algorithms include, move a piece, castling, enPassant, Update GUI, Test game, Check Checker, and Checkmate Checker. Details of these algorithms can be seen in section 5.3 of Design Specification Document [1].

5. MAIN DATA AREAS

Details of main data areas explained in section 5.4 of design specification document [1].

6. FILES

A game is saved in a directory of the user's choice. This directory contains an 'info.xml' file, which dictates whether the game is an ongoing game or a replay. An ongoing game cannot be loaded as a replay and vice versa. The directory also contains the rounds in the game, each round is an xml file containing the board, who's turn it is, the captured pieces and the round count.

When the user starts a new game, the temporary folder is cleared and the game starts saving to it. It continues to save to the temporary folder until the user chooses the 'Save Game' option, after this the game files are moved and start saving to the new folder. If the user wishes to load a saved game, they select their desired game folder, and the game's previous state will be displayed.

If the user wishes to replay a previous game, the game must have been finished. The user selects the game folder they wish to replay, and the game will display the board at the beginning, with arrows at the top, allowing the user to scroll forward and backwards through all the .xml file game states that were made within that game.

Each game state is stored as a singular .xml file, holding information regarding whether or not each cell is occupied, and if so, by what.

7. INTERFACES

Use of digital interface (GUI), buttons are used to initiate games, start games, and show menus. Text boxes are used for users to input names into the program. Radio-buttons for users input choice of player color, also ensures only one color can be chosen. Mouse event handlers are used to interact with the chess board, allowing pieces to be clicked on the board and select a square for it to be moved to.

7.1 Button Rules

Buttons work by clicking on them. However, when certain buttons are clicked (e.g., Player clicks "Resign" and then "Yes"), all other buttons are disabled besides the resignation menu in the middle of the screen. This also applies for "Draw".

7.2 Text boxes Rules

Text boxes allow all types of characters, no matter the format. However, it cannot be left empty, as an alias or name must be present to start a game. If either the player 1 or player 2 name boxes are left empty, a warning is given to the player regarding this issue.

7.3 Radio buttons Rules

Radio buttons allow the user to decide the color for player 1. Selection of color has been restricted for player 1 only, as two inputs for choice would leave one selection void. One radio-button must be selected as the game cannot start without a color being assigned to each player. If no radio-button is selected, a warning will prompt the user to ensure a color has been selected.

7.4 Mouse event handler Rules

Event handlers are added to each square on the chess board as well as the promotion pieces (in instance of pawn promotion). However, no action is taken when the wrong color piece has been clicked when it's not that users turn. This will prompt the user with a warning informing them to select the correct color piece. If a square on the chess board with no piece is clicked, no action should be taken. If a square with a piece is clicked and it is that player's turn, the possible moves for clicked piece are shown on the board. The only places a piece can be moved are on the board. Any clicks that are not valid moves will be disregarded. If an invalid click of an opponent's piece is clicked at this point, a warning will be given reminding the user of whose turn it is.

8. SUGGESTIONS FOR IMPROVEMENTS

8.1 Removed method

The first major suggestion is that, since we used an old version of Java specifically Java JDK corretto 1.8.0_362 which is a version of Java 8 which has Javafx bundled with it to develop the program. We used the deprecated and now deleted method impl_getUrl(). This does not allow our program to run as an executable for anyone who has a version of java newer than that of Java 8 installed. A way to fix this would be to create a separate class called something like FoundImage which extends image and a has a private final string url which is the url of the image and a public method called getUrl() that returns the private url of the image. Everywhere we create an image we create it as a FoundImage and then use the getUrl() method instead of the deprecated method.

8.2 Structure of The Program

For the future versions of the program it would be a priority to use a more Object-oriented Approach to the program this would in doubt require an almost full rewrite of the program. As at the moment the program only masquerades as an Object-Oriented program. It does not fully take advantage of Inheritance correctly in all ways not to mention the Backend and the Frontend could be integrated more thoroughly. At this point almost all of the setup for the GUI is done in the GUI methods although there are ways to lighten the setup by moving some setup to the backend. The way that game logic works also creates inherit warnings in the IDE such as values always being true and or 0 when in fact they are not this is because the program is more procedural then Object-Oriented as most methods are void and static with setters and getters very rarely being seen and some classes not having them at all. This leads to classes that wouldn't exist in an object-oriented version of the Program such as makeMove and EnPassant.

9. THINGS TO WATCH FOR WHILE MAKING CHANGES

While making changes, all the JUnit and system tests should be run and passed for the program to be considered working, new tests should be made by the dev team to test any new features/changes that have been implemented. When running tests, it must be ensured that 'TestGameSaving' is run before 'TestGameLoading'

9.1 Saving/Loading

Saving and loading are performed by the JAXB parser. If the developer wishes to update the java version this will have to be added manually, as it is not bundled with Java past Java 8. If any variables are added to the 'Board', 'Game', 'Player', 'Square', or any of the piece classes that the developer wishes to be saved in the save files, they should be annotated with the '@XMLElement' annotation, this will ensure it is saved in the game files, similarly, if they wish for the variable to not be saved it should be marked with the 'XMLTransient' annotation. It should also be noted that the game save function is called 'nextRound()' in the Game class. If an invalid game is loaded from GameSaveManager it will return null.

9.2 The board

Within the back end of the program, the board is stored as an array of 'Square' class. The developers should be aware that the index '0, 0' marks the top left square on the board, and '7, 7' indicates the bottom right corner of the board. In the front-end the board is displayed starting at '1, 0' so that the grid numbers are displayed. Note that if a square does not contain a piece 'getPiece()' will return null, so 'Square.hasPiece()' should be called first. The back end does not check coordinates to see if they are out of bounds, this is done by the GUI. So, the developer should make sure any code in the back end does not violate the bounds of the board array.

9.3 Check/Checkmate

When the king is moved by the user, 'inDangerChecker()' is called to remove any moves that would endanger him. This method is specifically for moving the king, if this for any reason is changed to a call to the function 'removeMovesThatEndangerKing()' a stack overflow error will be caused.

9.4 Styles and Icons

The styles for various buttons and texts can be found in 'src/uk/ac/aber/cs221/gp02/chesstutor/gui/util', these can be changed by the developer if they wish to change the styles and icons. 'styles.css' is commented with what each style corresponds to, and the .png files are named and self-explanatory.

9.5 Game

When a new game is created, the turn is black, and the round is 0. Before the game starts the round is incremented so that it becomes round 1 and turn becomes white, this is so that the initial board state is saved for replays.

Building

Instructions for building an executable jar file for the game can be found in the readme for the main project.

10. PHYSICAL LIMITATIONS OF THE PROGRAM

10.1 Devices limitations

Chess tutor is program limited to the use of computers. No other compatible devices have been tested to see if it will run correctly. Devices such as phones, tablets and other handheld entertainment systems will not be able to run it unless they have the capability to use and run IntelliJ.

10.2 Display limitations

The host computer's display resolution must be at least 900pixels by 500pixels to run the program. This is because the program is not re-sizable. In the case where the users display is less than the required size for the GUI, users may experience limitations in game play as certain interfaces will not be shown on screen.

10.3 Sound limitations

Computers may experience sounds on pop-up waring displays that have been included in the game. However, this is a case-by-case occurrence and is not an actual implemented method. This may be limiting to some user's experience as no sounds have been added to indicate any actions or warnings.

10.4 Mouse limitations

Almost all the inputs for the game are made using the user's mouse or touchpad, thus, in order for the game to run correctly, a mouse or touchpad may be needed. Buttons, pieces, and radio buttons are used within the game, and can only be interacted with using these.

Some exceptions to this rule, although more unlikely that using a mouse, could be either a laptop with a touchscreen, or a laptop 'touchpoint'. A touchscreen will allow for similar inputs to a mouse and thus can be used to interact with any objects mentioned above. A laptop touchpoint can be used to interact with the game also. However, for both of these cases, the user will mostly likely have access to the touchpad, which would be considered a more easily usable and efficient way of interacting with the game.

10.5 Keyboard limitations

For the game to run, the user's device must have a keyboard. This is because the user needs to input the player's names before they're able to start a new game. Users without a keyboard will only be able to replay or continue a previous game as this does not require any text interface use.

11. REBUILDING AND TESTING

All program files, related to the final program code, are located on the standard repository in the *src* folder. This folder contains a JAR file which should be built in IntelliJ to ensure path variables are updated to your local system. All tests can be run via the standard IntelliJ project. All tests are contained in a package called tests, running this package runs all the tests. The results of the tests are shown in the standard IntelliJ console, if a tests passes it is show with a green tick. Future developers should add their tests within this package, or inside a subpackage if it is applicable.

REFERENCES

[1] Software Engineering GP02 Project Design Specification Document

[2] Software Engineering GP02 User Interface Specification Document

DOCUMENT HISTORY

Version	*Issue No.*	*Date*	*Changes made to document*	*Changed by*
0.1	N/A	24-04-2023	N/A – original version	JAB153
0.2	N/A	24-04-2023	Added program description and simple file description	WIA14
0.3	N/A	04-05-2023	Added introductory sections.	JAB153
0.4	N/A	04-05-2023	Updated file description	WIA14
0.5	N/A	04-05-2023	Added interface information	BIY1
0.6	N/A	05-05-2023	Changed wording for Interface sections	WIA14
0.7	N/A	05-05-2023	Added physical limitations	WIA14, BIY1
0.8	N/A	05-05-2023	Added things to watch for when making changes. Added information to the files section	MIB60
0.9	N/A	05-05-2023	Updated Section 11	MJS36
1.0	N/A	08-05-2023	Added Section 8	ABD15
1.1	N/A	10-05-2023	Added document objectives and removed any unneeded sections. As Project Leader, document is approved and set to release.	MJS36
1.2	N/A	10-05-2023	Updated descriptions, references and dates.	JAB153