Software Engineering Group Project
Maintenance Manual

Author:	Kieran Foy [kif11], Dustin Baker [Dub4], Adrian Enache [Ade12]
Config Ref:	SE_GP17_MaintenanceManual
Date:	11th May 2023
Version:	1.0
Status:	Release

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

INTRODUCTION

1.1 Purpose of this Document

This document is to help assist the program maintenance of the second year Software Engineering Group Project, created by Group 17.

1.2 Scope

This document provides a detailed description of the project structure and its contents, as well as an analysis of what could potentially cause issues or be improved upon.

This should be read by everyone responsible for updating or maintaining this project post version 1.0 release.

1.3 Objectives

The aim of this document is to help the efforts of program maintenance by including:

- A description of the program and its structure
- A list of algorithms, data areas, files and interfaces
- A list of potential improvements for code that weren't implemented due to time constraints
- Potential problems and their possible sources
- The physical limitations that hinder the program
- What to do when rebuilding the program

2. THE PROGRAM

2.1 Program Description

The application is a form of chess application designed to assist in helping new players learn how to play. This is accomplished via a series of useful visual aids such as, highlighting piece moves and differentiating their purposes. It allows for two players to play a chess match locally on a PC. Each user takes turns moving pieces attempting to beat the other via standard rules of chess. The users may save both finished and unfinished games to either watch a replay of them or continue where they left off another time.

The program is built in an object-oriented style in java, utilising classes to create objects in order to mimic a real-world chess board. The application itself also uses a number of external libraries including: “Javafx” (GUI) and “Gson” (file handling). The main components for the design of the system can be viewed via section 2.1 of the Design Specification [1]

See s

2.2 Program Structure

This program utilises 4 significant classes, Setup, Game, Replay and Piece as well as numerous sub-classes and graphics controller classes.

Setup handles all the methods to set up the game, restore an old game, handle usernames and colours, and restoring crashed games.

The Game class handles all methods used to run the chess match, including movement, and detecting check and checkmate.

Replay handles loading and displaying previously saved games, undoing and redoing moves in replay mode, and saving each turn.

Piece is a superclass of each of the piece sub-classes e.g. Pawn, King, Bishop. All subclasses store their specific attributes such as piece type and legal moves.

Further details for each class can be seen in sections 2.2, 3.1, 4 and 5.1 of the Design Specification [1]

2.3 Algorithms

This program makes use of many significant algorithms to implement the most important features of chess, as stated by the functional requirements listed in Chess Tutor Requirements Specification [2]. These include movement, piece-taking, check, and checkmate.

Details for each set of algorithms can be found in section 5.2 of the Design Specification [1]

2.4 Data Areas

As the program was programmed in the object-oriented paradigm, it makes heavy use of objects using chess pieces. Each piece is an object of either Pawn, Knight, Rook, Bishop, Queen, or King type, and each of those classes is a subclass of the Piece superclass. These objects are created at the start of every game and placed on the board (See Below) in the position according to its object’s x and y co-ordinate values, type, and colour.

The state of the board is stored in a two-dimensional array utilising co-ordinate. The x value relates to the file the piece is placed at and the y value relating to the rank. E.G. (3,0) would be the initial location of the black queen.

Further details can be found in sections 5.4 of the Design Specification [1]

2.5 Files

The program uses JSON files to store the information for each game. Utilising the key-value pairs that JSON formats offer, we can store different parts of information in the file for easy access and manipulation. A JSON file is created for each game instance, storing the data once the user opts to save a game. Upon creation, a history of the game is saved into it saving details such as each player's name, what colour they were playing as and all the moves they made and their necessary information. This includes whether the king is eligible for castling, or a pawn is eligible for en passant. The user is responsible for managing their saved games I.E, deleting unwanted games etc. The JSON files are stored in the "resources" directory under "Saved Games". See section "4.1.2 Source Code".

Further details can be found in section 5.4 of the Design Specification [1]

2.6 Interfaces

The application does not utilise any interfaces. This is because there is no interaction with the real world.

3. ALTERING THE PROGRAM

3.1 Improvement Suggestions

Upon the final development of the program, some essential as well as desirable improvements can still be made. The following sections outline the priority in which they should be tackled via the maintainers. A further section also outlines some minor improvements to improve the quality of the program for user experience.

3.1.1 High Priority Improvements

Saved games always open in replay mode, despite not being finished when the user attempts to load them. This is a high priority issue since it fails functional requirement 11, see System Requirements [2]. This issue is also the cause of a system blocker; The user is unable to access the pause screen when viewing a replay until the game ends. Given un-finished games do not end, the user cannot return to the main menu screen and is required to close the application to proceed.
[*Gitlab Issue #89*]

Game files are not saved if the application is closed by force/crashing during an active game. This is a high priority issue since it fails functional requirements 9 and 11, see System Requirements [2].
[*Gitlab Issue # 85]*

3.1.2 Medium Priority Improvements

The application can overwrite save files if certain criteria is met when saving a game: Black/White Player names match that of another save file and the game is saved on the same date. This is a medium priority issue since it could greatly affect the experience of the user.
[*Gitlab Issue #84*]

Player names are not updated when loading a saved game when viewing the active game. The current names display placeholder text for “Player 1” and “Player 2”. This is a medium priority issue since placeholder text is displayed and the lack of player names limits information to the user.
[*Gitlab Issue #91*]

3.1.3 Low Priority Improvements

Users can select the “Load Old Game” button when no save files are available in the Main menu. This is a low priority issue since it doesn’t affect the application, however it does fail SE-F-010 in the Test Specification [3].
[*Gitlab Issue #67*]

Red highlights are not displayed on the king during replay mode. This is a low priority bug since it doesn’t not affect the functionality of the replay mode, however it does fail SE-F-069 in the Test Specification [3].
[*Gitlab Issue #90*]

Error messages are not displayed if the user attempts to open a save file that is formatted incorrectly or contains the incorrect file extension. This is a low priority issue since the application still functions correctly and the file instead does not open. It also fails SE-F-076 / 077 in the Test Specification [3].
[*Gitlab Issue #92*]

Error messages are not displayed in the user attempts to start a game with invalid player names. This is a low priority issue since the application only allows the user to continue with valid names, however it does fail SE-F-005 / 006 /007 in the Test Specification [3].
[*Gitlab Issue #65*]

Unrelated screen flash up when selecting “Main Menu” during a game. This is a low priority issues since it doesn’t not affect the functionality of the program, however it could confuse the user as to where they are navigating to.

3.1.4 General Improvements

The highlights to demonstrate a pawn taking another pawn via en passant is currently highlight in grey, matching other “moving to empty square” highlights. This however could be changed to match the yellow

highlights of "taking a piece" since en passant is designed for the intension of capturing a piece rather than moving to a square. Details on highlights can be found in section 3.1, UC.2.2.2/2.2.5 in the UI Specification [4]. [*Gitlab Issue #88*]

Some piece assets appear off centre to the tiles they are positioned on. This is most clear with the furthest queen side Pawn asset. Having the assets centred would allow for a nicer feel to the program.
[*Gitlab Issue #78]*

The new game setup screen has a few improvements that could be made to it. They are as follows:

- Title for the screen can be changed from "Login" since this incorrectly indicates the purpose of the screen.
- The white player could be positioned before the black player since they move first.
- The font colours could be switched since "Black" uses white text and "White" uses black text.
- The positioning for the labels could be made more consistent (Alignment)
- Button names could be made more consistent (Capitalisation)

When selecting a piece, and then select a friendly piece, the program will write an error message in the terminal stating that this is an illegal move. Only once that has appeared can you then select another piece.

Introducing the ability to drag and drop pieces with the mouse, instead of just selecting the piece and the position the player wants it to move to. This would be difficult to implement, as the JavaFX for the chess pieces would need to be overhauled to allow them to move freely without being confined to the centre of the grid panes.

3.2 Things to Watch Out for

As the project has been developed in an object-oriented approach each class has a set of properties and the application utilises inheritance for specific classes, namely the varying pieces: Pawn, Knight, Bishop, Rook, Queen and King. However, these classes do not currently follow standard inheritance rules; Attributes stored in the super-class, Piece, utilise private attributes rather than protected. This means some methods belonging to the sub-classes utilise passing arguments, such as x/y positions, rather than read-only/read-write property methods (e.g., Getters/Setters).

Many classes use a large quantity of static methods. These methods could be avoided to ensure correct encapsulation of classes. These mainly apply to the controller classes whereby no instances for these are created and instead each class calls the necessary class methods when needed.

3.3 Physical Limitations

Based on minor amounts of testing, it is recommended that the program utilises around 2GB of memory. This value is not very accurate. No further system requirements could be collected.

The program runs well and effectively on the Aberystwyth University IS PCs.

4. REBUILDING AND TESTING

4.1 File Locations

4.1.1 Documentation

All documentation is located under the “docs” directory. These include a separate directory for each piece of documentation and its assets. E.G., docs/User Interface Specification contains the UI Presentation (pptx) and Specification (docx and pdf).

4.1.2 Source Code

The source code contains all the classes and resources necessary to build the project as well as perform unit tests. It does not however, include the resources for external libraries. The structure is as follows:

Main – Contains the main resources/classes

Java –

Com.exmaple.devprojectcode – Contains all the classes for the app

Enums – Contains the enumeration classes

PieceColour.java

PieceName.java

Pieces – Contains the piece sub-classes

Pawn.java

Knight.java

Bishop.java

Rook.java

Queen.java

King.java

UIControllers – Contains the UI Controllers

EndScreenController.java

GameScreenController.java

NewGameScreenController.java

PauseScreenController.java

PawnPromotionScreenController.java

Main.java

Game.java

Setup.java

Replay.java

Resources/com.exmaple.devprojectcode. – Contains all the graphics-based resources

Pieces – Contains all the graphics for each piece.

… - fxml files for screens

Tests/java/com.example.devproject.code – Contains all the test related resources/classes

GameTest.java

ReplayTest.java

SetupTest.java

4.2 Rebuilding The System

The initial development of the project was built using Maven. This version of the built system can be found under “dev/projectcode” and can be utilised as a reference. In order to rebuild the project, the maintainer will need to ensure the necessary pre-Apache Maven 3.9.1).

To begins with, open an IDE which allows for the utilisation of Maven (E.G InteliJIDEA). Proceed to create a new project with maven. Open the “POM” file and add the necessary dependencies and plugins for javafx 20.0.1, and GSon 2.10.1.

Proceed to populate the project with the files found in the “src” directory.

4.3 Unit Tests

All unit tests can be found under “src/Tests” directory. The unit tests are produced using Junit 4.13.2 and will require the necessary maven dependencies to run and build them. There are 3 Test classes which test the main functionality of the program.

The Game Test class is used to test the main aspects of the chess functionality. These include moving pieces, taking piece, performing checks as well as checkmate.

The Setup Test class is used to test how the board is setup.

The Replay Test class is used to test the loading features. These include analysing the file structure, loading specific properties from the file.

When new issues are discovered, unit tests can be created by creating test assertions to test individual features. The maintainer can also make use of functionalities such as “Before Each” to ensure that the tests are kept to the same standards.

REFERENCES

[1] Cooper G., Foy K., Owen J., Moore T., "Software Engineering Group 17 Design Specification", 1.0, SE_GP17_DesignSpecification, 20th March 2023

[2] Loftus C.W., "Software Engineering Group Project - Chess Tutor Requirement Specification", 1.2, Computer Science Department, 28th Feb 2023

[3] Cooper G., Foy K,., Baker D., Enache A., "Software Engineering Group 17 Test Specification", 1.7, SE_GP17_TestSpecification, 2nd May 2023

[4] Moore T., Enache A., "Software Engineering Group 17 User Interface", 1.7, SE_GP17_UISpecification, 19th March 2023

DOCUMENT HISTORY

Version	*Issue No.*	*Date*	*Changes made to document*	*Changed by*
0.1	N/A	01/05/23	N/A – New Document	DUB4
0.2	57	01/05/23	Wrote Introduction section and part of the "Improvement Suggestions" section	KIF11
0.3	57	08/05/23	Completed "Improvement Suggestions" and most of Part 2: The Program	KIF11
0.4	57	09/05/23	Added the information about rebuilding and testing for the future maintainers of the project.	ADE12
0.5	57	11/05/23	Completed all sections – Revised format	DUB4
1.0	N/A	1/05/23	Released Document	DUB4