Introduction into Object-Oriented Programming

Coursework 1

Mustafa Bozkurt

Chess Game

Johannes Niklas Hartmann
Queen Marry University London
Software Engineering MSc
170904935

Introduction

The application is a console based two player chess game.

How to compile

First make sure you are in the root folder of this game. To compile the code just run the following command. It deletes eventually existing old compiled code and compile the project.

- make clean && make

The rules how to compile the code are defined in the Makefile.

Note: Make sure you have the latest gcc compiler version installed to compile the code

How to execute

To start the game, simply execute the following command in your terminal at the project root folder.

- ./build/main

How to play

To play the game, the player will be asked to enter a figures origin and target. To select a figure just type in its coordinates. For example, to make an initial move type in:

- Please select a figure: E2
- Please select a target: E4

Note: The input is case insensitive

If you type in an invalid input, the player will be asked to retype in your move. The game accepts only valid moves corresponding to the standard chess rules. After a successful move the current board will be printed. Now the second player can type in its move.

There are also bunch of extra commands to type in at the beginning of a move:

Undo the last move:

- :u

Save the game to a file:

- :s

Load a previous saved game:

- :l

Quit the game:

- :q

Note: It is only possible to save one save game. The file location is at the root folder: SaveFile.txt. If the player wants to protect the save game from overriding, he should move it to another location

Programming approach

The application is implemented using the Model View Controller approach.

Therefore, the model holds all the game logic and implements the chess rules. The class “Board” represents the chessboard. It has a 2D chessman pointer array as member to represent the chessboard, and moves will be made with the applyMove method. Chessman is an abstract class that is used as interface for each individual chessman. Each individual chessman must implement the getPossibleMoves method should contain the individual game logic of the individual chessman.

The “View” class implements all needed user interaction over stdin and stdout as well as file interaction to load and save the game.

The “Controller” class combines view with the model and implements the game cycle. Therefore, it continuously asks the user for new moves until one player is checkmate/draw or the game is ended by a user command.

Documentation

The project is documented using Java-Doc style code comments. The detailed documentation can be found at:

https://incrediblehannes.github.io/QMUL-ChessGame/html/index.html

This documentation contains a brief explanation of each method. The documentation was generated by doxygen and its hosted by GitHub pages. There is also an offline version in the source code folder.

Issues during the Implementation

Because auf the use of a pointer array, pointing to each chessman, I had big issues with memory leaks and segmentation faults. But in the end, I was able to solve this memory leaks.

In general, the usage of that many pointes caused many issues, which could have been avoided using smartpointers.

The rest of the Implementation was straight forward, even if some special moves were not that well supported by the design. For example, the design of the “move” class supports only to move one figure from an origin to a target position. To support moves like castling, which involves movements of two figures, the “applyMove” method must check if a move is a castling and react properly. This issue could have been avoided if the move would implement the game logic of this special moves, and the board simply applies a move without extra checks.

Limitation of the Application

Unfortunately, the draw of a chess game is only implemented very basically because of the limited time of this coursework. It only supports a draw when there is no possible anymore move for the current player. Also, the En *passant* rule is not implemented.

Also, there is no AI implemented. The application is a two player only game.

The undo function may make some problems in some special cases. For example, it is not possible to undo a castling and do it again, because the undo function does not reset the moved flag inside the chessman. It would be a bigger refactor to fix this bug, because my architecture does not support an easy fix.

Testing of the Application

To make sure that the application is free of memory leaks, valgrind was used as a tool to analyse and debug the code. Also, the load from file functionality was very helpful to apply many moves at once and check if every rule is applied correctly. Unfortunately, I was unable to implement proper Tests because the limited time and the complexity of the chess game.