Project: Creating a fully working game of chess

Barry Denby

November 22, 2017

1 Introduction

NOTE: read the whole assignment brief first before implementing it contains very important information

In this project you will be required to make teams of 2 to develop a fully working chess game. The chess game will implement most of the basic rules of chess (you are not required to implement en passent or castling). I will also ask you to implement a few extra features to your game. The rules you will be required to implement will be listed below.

Everyone is required to be in a team. All teams must be declared in class or a lab with both persons present. If email is favoured I require an email from both members of a team. Groups of three or more will not be accepted.

Please read through the rest of this assignment before you start coding straight away. In the words of Scott Meyers "Weeks of coding saves hours of planning"

2 Rules of Chess to Implement

The game of chess is a centuries old game that is still popular today. It requires much strategy and thinking ahead in order to win. The objective is to checkmate the opponents king, while preventing your own king from being checkmated.

The game is played on an 8x8 board. There are two players black and white. Each player has 16 pieces divided as follows: 8 pawns, 2 rooks, 2 knights, 2 bishops, a queen, and a king. Each piece has rules governing how they move on the board. If you are not familiar with the rules on how pieces move or capture other pieces I suggest you ask questions or research online. There are many sites out there that will introduce you to the rules and general play of chess. I will cover the more complex rules here. You are not required to implement the en passent or castling rules for this project.

The general movement of pieces and capture of pieces should not pose an issue. Where things get really interesting are the game states of check, stalemate, and checkmate. All of these game states involve simple line of sight rules.

Check occurs when one opposing piece is in a position to attack the king directly but it is possible for the king to evade that attack on the next move or for another piece to move in the way of the attacking piece or capture the attacking piece. The only exception to this is the knight who is not obstructed by other pieces so can only be evaded or removed. To clarify with an example if the black king is in check by a white piece there are three methods of escape.

- The king moves to a square that is unchecked by white pieces. i.e. out of line of sight of all white pieces
- The attacking white piece is captured by a black piece as long as the movement of the black piece does not expose the king to check.
- The attacking white piece has its line of sight blocked by a black piece moving in the way (does not work for a knight). This also assumes like the previous case that moving the black piece does not expose the king to check.

Checkmate occurs when a king that has been put in check cannot by any single move escape check. The player who is then checkmated loses.

There are times however, when it will not be possible for either player to force a checkmate state. In this case neither player can win the game. This is called stalemate and is declared a draw. There are two causes for stalemate

- A player's king is currently not in check but any move he could make would result in check (assumes the player only has a king left)
- Neither player has the necessary minimum required material to force checkmate

In terms of minimum required material there are six minimum sets of pieces that can force checkmate. They are listed below (all sets have the king present):

- A queen
- A rook
- A bishop, and a knight
- Two knights
- One bishop on dark squares, and a bishop on light squares
- One or more pawns left (can change to other pieces if they get to other side of the board)

2.1 Advice

There are a few pieces of advice I would give to you on this

- Use a source code manager (Git, SVN, CVS, etc) to develop, share, and update code.
- Do not use a publicly visable repository e.g. GitHub, this will lead to plagarism issues
- Try to work on independant parts if possible. Working in parallel will cut down the time needed for development.
- If possible use the JUnit test framework embedded with your project for test driven development. This will further enhance the ability to work independantly.
- Design the project first before implementing. Agree on datastructures and class layout in particular. This will save lots of headaches later.

3 Notes

You are required to submit this assignment by 2017-12-17 (Sunday 17th of December) by 23:55. You are required to submit two separate components to the Moodle

- An archive containing your complete Android Studio project. The accepted archive formats are: zip, rar, 7z, tar.gz, tar.bz2, tar.xz. The use of any other archive format will incur a 10% penalty before grading.
- A PDF containing documentation of your code. **If you do not provide documentation your code will not be marked.** Copying and pasting code into a PDF does not count as documentation.

There are also a few penalties you should be aware of

- Code that fails to compile will incur a 30% penalty before grading. At this stage you have zero excuse to produce non compiling code. I should be able to open your project and be able to compile and run without having to fix syntax errors.
- The use of libraries outside the SDK will incur a 20% penalty before grading. You have all you need in the standard SDK. I shouldn't have to figure out how to install and use an external library to get your app to work
- The standard late penalties will also apply

Very Important: Take note of the milestones listed below. These are meant to be completed in order. If you skip a milestone or trigger one of the failing conditions the following milestones will not be considered for marking. You should be well capable of producing strong and generally robust software by now. For example if there are six milestones and you fail the third one, then the fourth, fifth, and sixth milestones will not be marked. Documentation milestones will be treated separately from Coding milestones.

You should also be aware that I will remove marks for the presence of bugs anywhere in the code and this will incur a deduction of between 1% and 15% depending on the severity. If you have enough of these bugs it is entirely possible that you may not score very many marks overall. I want robust bug free code that also validates all user input to make sure it is sensible in nature.

Also note that the percentage listed after the milestone is the maximum mark you can obtain if you complete that many milestones without error.

4 Coding Milestones (70%)

1. Generate a custom view that is forced to be square in size. It should display a full 8x8 chess board with the initial state of the game. it should accept user input for making moves (either two touches, or dragging) (10%)
2. Add in highlighting to your custom view to show what piece was selected and where it can move to (15%)
3. Implement fully working movement of pieces including capturing (30%)
4. Implement stalemate detection. If stalemate occurs end the game immediately (40%)
5. Implement check and checkmate detection. If checkmate occurs then the game should end immediately (50%)
6. Build a UI around your custom view that shows the current state of the game and provides options for resetting the game. It should also indicate the current turn and notify if check, checkmate or stalemate has occured. (55%)
7. Implement two additional features of your own choosing. (70%)

5 Documentation Milestones (30%)

8. Document why you designed the UI the way you did. This should detail your choices in widget layout and position and how they make user interaction easier. (15%)
9. Give a high level description of all methods in your Java code, the data-structures used and why. Also discuss the two extra features you added. (30%)