Team SAWM Project Report

Department of Computer Science
Wichita State University
Wichita, Ks 67260

Abstract

The report details the process of development and design implementation for the Droid Chess project coded by team SAWM. The end result of the project is an android application which allows the users to play a chess game.

1. Inception Phase

1.1 Requirements

-The user should be able to play a chess game

-The available moves for any chess piece should be highlighted

-The user should be able to view a detailed set of instructions on how to move each piece

-The user should be notified when either king is in check

-The user should be able to specify whether or not certain rules variants should be allows

1.2 User Stories

Chess Game

The user will want to play a full chess game

Splash Menu

The user should be presented with a list of available actions (online play, offline play, instruction menu) upon loading the application

Opposing Move Notification

The user should be notified through system notifications when their opponent over an online game moves

Points

The users should be allowed to play a game with the standard chess points system

Local Game

The user should have the option of playing a local game on a single device

Specific Pairing

The user should be able to select a specific opponent when pairing for a new game

Random Pairing

The user should have the option to randomly pair with another user

On Screen Tutorial

The user should be able to view a list of specific rules about chess

Rule Variants

The user should be allowed to select which rule variants they want to allow in the game.

Game Timer

The user should have the option to play the game with the standard chess timer.

1.3 Iteration Plan

Iteration 1

- Complete UML Design
- Code board layout and chess rules
- Code Local Game
- Code available Moves
- Code Checkmate notification
- Build Howto

Iteration 2

-Implement Online Play

-Implement Random Pairing

-Implement Specific Pairing

-Implement Online Move Notification

Table 1. Time Estimation

User Story	Individual Estimate	Team Estimate	Priority
Chess	12	10	High
Splash Menu	1	.5	High
Opposing Move Notification	.5	.5	High
Random Pairing	6	5	Low
Checkmate Notify	.5	1	Middle
Points	.5	1	Low
Local Game	3	1	High
Specific Pairing	4	5	Low
Available Moves	2	.5	High
Game Timer	.25	.5	Low
On Screen Tutorial	1	2	Middle
Rule Variants	.25	.5	Middle
	31	27.50	

2. Use of Program

2.1 Git

All of the code for our project is available through Git. Our Git repository can be found at: https://github.com/sawm/droid-chess.

2.2 Android

To run the code a device running Android is required. The code is optimized for Android 4.2 (API level 17), but it will run on any device 3.2 (API level 13) or higher. To run the application on your Android device you will simply have to install the .apk file.

3. Development

The development was initially split into two iterations. The first iteration was essentially just to get the game of chess up and running. The second iteration was to add in the online play. Unfortunately, due to time constraints only the first iteration was successfully accomplished.

The biggest factor in the length of the first iteration was a lack of knowledge on the technologies used. None of the members of the group had had any previous experience working for mobile devices, let alone the Android API. It took the team quite a while before we were completely comfortable working in the Android environment. We encountered a lot of problems initially since each of us had Android devices running on different API levels. This caused issues with deprecated function calls, and other features supported on one device but not on the others.

At one point in the first iteration it was deemed necessary to refactor nearly the whole project. The code was a mess, and did not follow any well structured object oriented design principals. We had been using Android's ImageView class for each of our game pieces. This was the only way to implement images in Android. We had been using that class and utilizing its description variable to handle all the specific data on each piece. The data was all stored in a string that had to be parsed. This quickly made our code very difficult to work with. We refactored the entire project. We made an abstract class called Piece which extended the ImageView class. This class has an abstract function called getMoves() so each extension of Piece could implement its own logic in determining available moves. This cleaned up the code a tremendous amount. Before we had to parse the description on the ImageView to get the game piece type. Then it went to a giant switch statement that determined the logic in getting the pieces available moves. After the refactoring, the on click event for getting moves was cut down from several hundred lines to less than ten. Our programming went much, much smoother after the refactoring process. Perhaps if we would have properly decided how to structure our program sooner we could have gotten closer to meeting our goals in the second iteration.

We extended the ImageView class for another important aspect of our project. The chessboard is made up of an array of Square objects. The Square class extends the

ImageView class. The purpose of the Square class is to return all information about the position on the board such as whether or not a game piece is currently residing on it and if the position is a current available move. The getMoves() method in the Piece class requires the array of Squares as an input. This is the only sort of coupling between the Piece class and the Square class. As long as the Square class outputs the same data, no changes in the Square class will affect the performance of the Piece class.

All of the other classes utilized were from the Android API. The entire structure of the application is based on the core Android classes. So, there was not much work in creating the basics for getting the application running as Android had taken care of all of it for us.

We utilized the Eclipse IDE because of its support for Android development. With the Android Development Kit for Eclipse we were able to create a new completely runnable Android project by the click of a button.

Even if we did not have the early delays in learning Android and the delays in refactoring, this project still seems pretty ambitious for two people to accomplish in the given time period. If SAWM were a group of four people instead of two we probably could have gotten some sort of online play implemented.

4. Design, Implementation, and Testing

Initially in our design we only knew that the pieces would have to be of the same type to ensure proper cohesion. Unfortunately, we took this too far, but trying to make each piece an ImageView. In small scale, this worked fairly well, but when we started to work with this on the large scale of a full chess game we found this would not be an easy implementation.

After the initial revelation about the scope of this project we refactored the code so that each piece was its own class, descended from the ImageView class. It was this class that allowed to use to easily create UML diagrams for the code, which were a tremendous help in implementing future code. With this modular design, one each piece could be changed independently of the others, allows us to more easily debug and code. By doing this, we could easily add or change any piece, or even define custom pieces (programmatically of course).

The automated testing was a fairly simple matter. It consisted of a secondary thread to the program. This thread randomly picked a piece that could move, moved it to a random square, and then did the same thing for the other side. Assuming no exceptions occur, It continues doing this until either side loses the game, it which point it stops the program. Using this we were able to uncover a few errors (mostly to do with pieces that were already taken that we couldn't select manually).

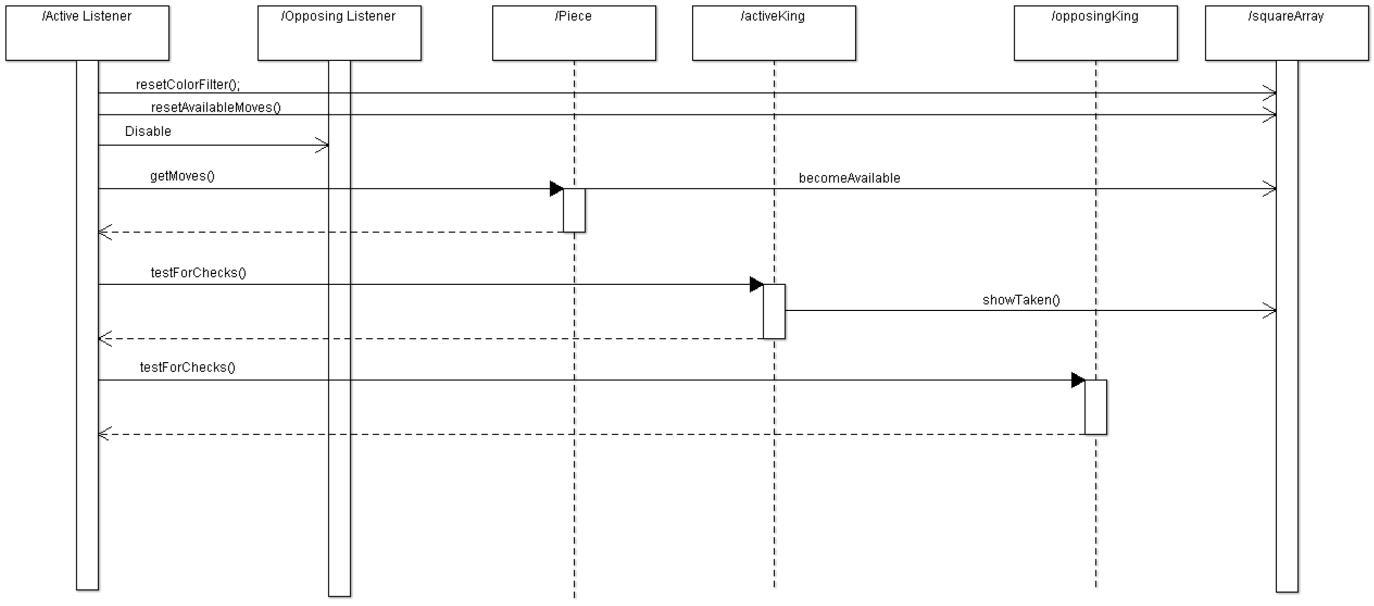
The end result was three main listeners to see where the user clicked, and two main classes to handle the clicks. The code makes extensive use of inheritance to allow for modularity.

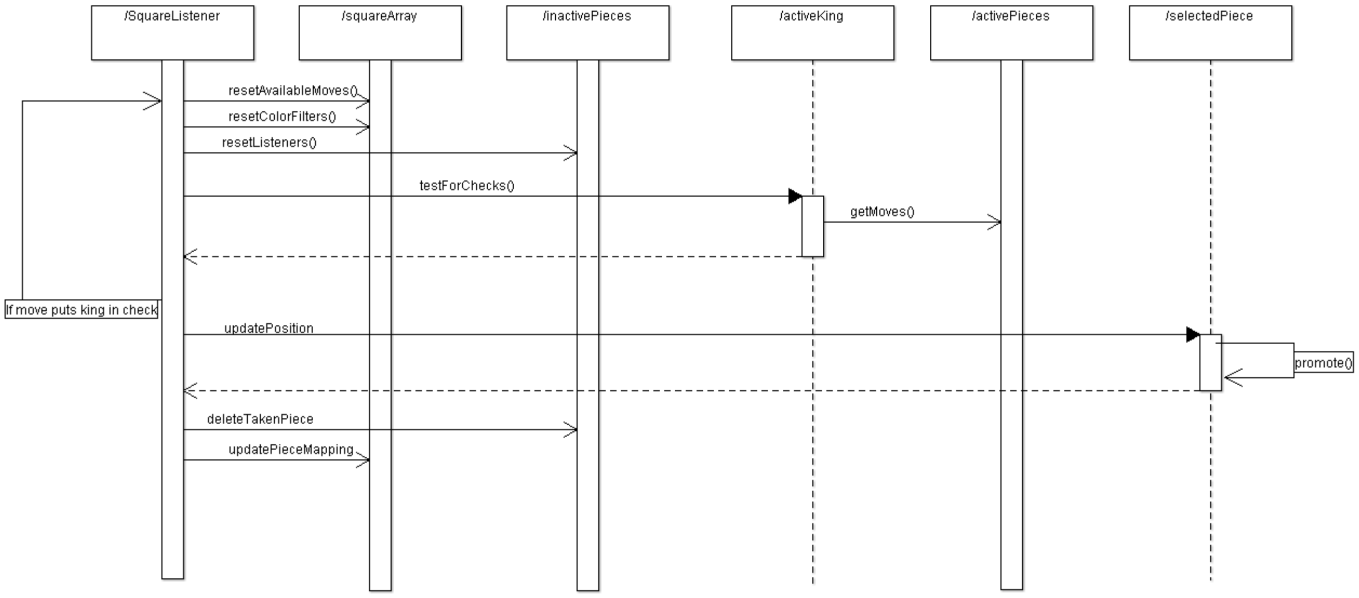
5. Team Structure

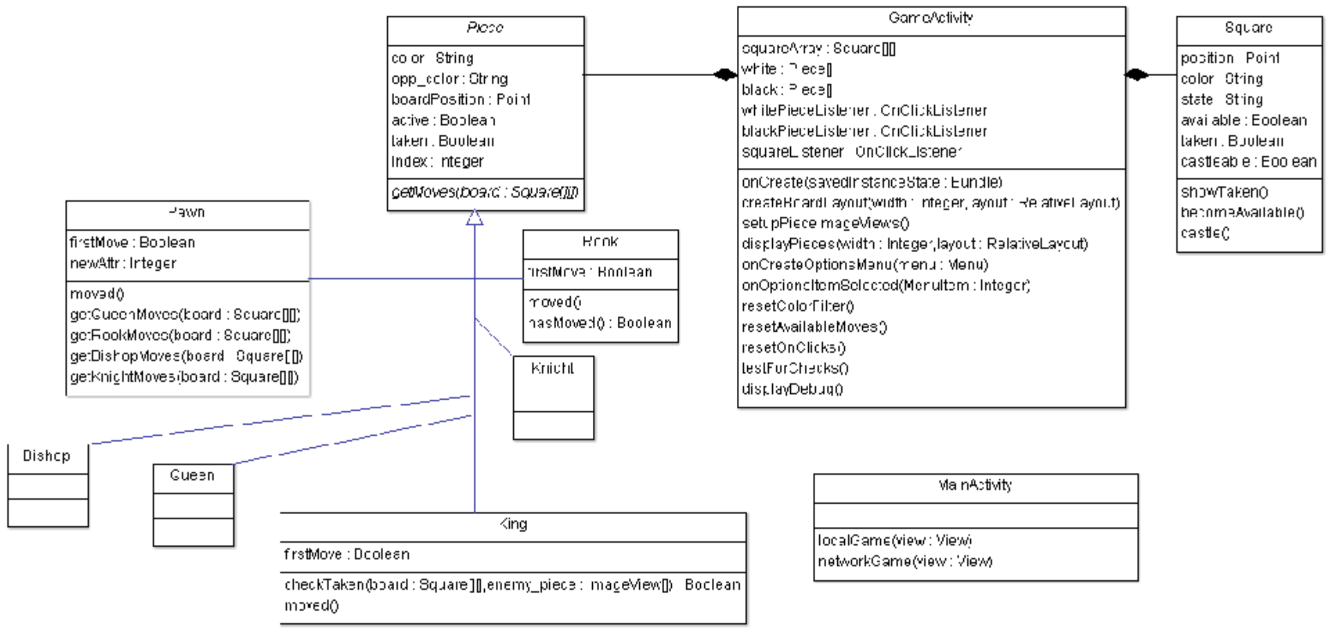
Being a two person team, there was

not much need for extensive team structure. We communicated mainly by text message, however we met once a week and sat down to resolve any issues we had together. We did not use any bug tracking system like BugZilla, because with only two people it was just as easy to alert each other about bugs through text messaging and e-mail. Having a Git repository was very helpful for coordination, as each of us had instantaneous access to new updates or bug fixes.

6. UML Diagrams







7. Contribution Matrix

Task	Will	Sidney
User Stories	50%	50%
Iteration Planning	50%	50%
UML Class Diagrams	50%	50%
Test Cases	0%	100%
UML Revisions	100%	0%
Code	45%	55%
PowerPoint	50%	50%
Report	50%	50%
Final Presentation	50%	50%