Software Development Progress Report

Part 1 - Project Summary

Specification

Our application allows two users to connect onto a game from separate android devices to play a chess game. Users will be able to login to their account with their saved details, or make a new account, simply by entering their username to load their data. Upon conclusion of a game, players can choose to either rematch, or exit, and their record will be saved.

CRC model

Entities: User, Piece (Pawn, Queen, etc.)

Use Cases: Game Manager, ChangeIcon

Controller: Move, Login, SetPreference, Matchmaker, AI

Interface: GUI, Database

Scenario Walkthrough

Our typical walkthrough has the user decide between starting a singleplayer or multiplayer game. Upon selecting either option, they need to give their username to the program to load their data, and will be assigned either white or black. If they want to play single-player, they will start a chess game against a CPU (AI), or if they want to play multiplayer, they will be matched with someone else. The players will then take turns playing chess until the game concludes. You then have the option of rematching or exiting to the main menu.

Skeleton Program

Our Skeleton Program will be able to do the following

- Display start-menu
- Start a game with a chosen game type
- Display the board

Part 2 - Work Distribution

What each member has been working on and future plans

Bryan Guo

- *Specification*
- *CRC Cards*
- *Walkthrough*
- *Progress Report*

Muhammad Ibrahim

- *Specifications*
- *CRC cards*
- *Progress Report*

Vala Jalalvandi

- *CRC cards*
- *Walkthrough*

Juan Martin

- *Skeleton program*
- *CRC cards*

Jaren Worme

- *CRC cards*
- *Walkthrough*

Part 3 - Things That Have Worked Well With Our Design

While we haven't yet directly dealt with the consequences of our project architecture (we're still *in* the architecting phase), we've certainly planned for numerous areas of expansion. The structure of our program as defined with the CRC codes allows for easy expansion and addition of new features, since the different layers of design architecture are well defined. With this design, the layers of clean architecture only interact with the layers that are adjacent to them.
Another thing that has worked well with our design is the ability to define simple and easy to understand entities that store crucial information about the program without overlapping. This means that each entity has only a single responsibility, which makes any changes in the future easy to implement. An example for this is the piece class, which has subclasses for all the pieces on the board. Each of these subclasses store specific information for the pieces, such as their move-sets and icons which will be loaded to the GUI.`

Part 4 - Current Struggles And Open Questions

One of the struggles with coding our chess game is determining algorithms for getting possible moves. This is especially difficult with our array-based representation of the board, and we may rework this part at a later phase.

A recurring issue we have is a lack of clear implementation of our classes. There appear to be many ways to implement our code, but we need to firmly establish and build upon it so as to avoid

The largest struggle with our group is coordination. At times we are not all on the same page, and as a result our implementation of the skeleton program is spotty. In the nest phases, we need to commit to a model with minimal changes and coordinate a division of labor for greater productivity. In our current state, we are disorganized.