Conscious Sound - Final Report

David Breneisen

CSCI 4237 - Fall 2011

Introduction

The goal of Conscious Sound was to create an android application that would allow for musicians to develop their musical sense through ear training. Conscious Sound aims to remove the "natural talent" that is commonly deemed necessary for one to possess in order to become a successful musician, by developing exactly this talent.

Features

Conscious sound features three games: Identification, Memory Match, and N-Back. Identification is the most basic game, it plays a musical relation or pitch and the user must identify it. Memory Match requires the user to match two cards, one a sound, the other the matching note. This introduces the need for the user to develop some sense of memory along with the hearing. N-Back then pushes this sense of memory to the limit as the user must remember the note played "n" times ago. If the user does well, n is increased, making sure that the user is always challenged playing this game.

Development

The first thing to do was to develop a system for playing sound files. This was done using Android's Services. By using a strict naming system for the sound, it made it possible to play random sound files, while knowing what sound files were played. This was a necessary feature for all of the games in the application.

For demonstration purposes all the levels were instantly made available to the user. In the actual app the user would need to progressively unlock the more difficult levels.

-Identification Game

The identification game begins by letting the user play a random pitch. The user then must try

to match one of the available pitches from that level to the pitch that was played.

The RandomNote class was used for abstracting the functionality of selecting a random note/octave, and retrieving the associated sound resource.

A LevelStatistics class was used to abstract the tracking of user statistics, and the displaying of those statistics. The updateSession method withing LevelStatistics causes the layout that displays the statistics to be updated.

-Memory Cards Game

When the memory cards game is first loaded, all of the memory cards (ImageButtons) must be created. A random card is chosen and assigned a note/sound, and then another random card is chosen which matches the note. This is repeated until all the cards have been assigned either a note/sound or a match to a note/sound. The button is then assigned the corresponding xml selector which controls the functionality of flipping over. A checker determines whether the last two cards flipped over match, and if they do the images are turned to green and can no longer be pressed, but if they don't match they are flipped back over to the initial state.

N-Back Game

The N-Back game starts out with n=2. It displays 5 images of notes, and the cursor starts out hanging over the second note from the right. If the gets enough correct in a row, the cursor moves to the left. If the user gets past n=5, then the interface displays 10 images of notes, starting with the cursor on the 6th note from the right.

A thread is used in order to keep the game playing random notes at a designated time interval. A simple Queue data structure is used for storing the last n notes that were played, which is needed when validating the user's response. Every time a new note is played, an animation is run which makes it appear as though the series of notes at the top is shifting to the left, a new note coming in from the right, and one going out to the left. The animation simply uses a series of images to give it this effect.

Difficulties

One of the largest difficulties of this project was determining what parts of the user interface to create programmatically and what parts to do in xml layouts. Some difficulties with trying to abstract the layouts led me to doing them in a more manual manner. Overall, layout development and animation was much more time consuming and difficult than actually implementing the functionality of the application.

Design decisions for how the games should play were also very difficult to determine. In hindsight, I think that allowing the levels to increase within each game, rather than having the user choose the level may be a way to simplify the user interface. This would make it so there are only 4 different levels/games to choose from, and each one simply increases in difficulty as the user improves.

Another problem is that the apk file size becomes very large with all of the sound files that are needed for the game. A way to reduce the apk size might be to try and synthesize the sounds rather than play them from file. Another way to reduce size might be to allow other instruments sounds to be used by downloading another package, rather than trying to include all the instruments in one apk.