Brain Recorder

by
Anthony M. Garcia

Computer Engineering Department
College of Engineering
California Polytechnic State University
2012

Date Submitted:________________________

Advisor:________________________

ABSTRACT

Electroencephalography (EEG) is the recording of electrical activity along the scalp. Our mind regulates its activities by means of electric waves which are registered in the brain, emitting tiny electrochemical impulses of varied frequencies, which can be registered by an electroencephalogram. These "brainwave" frequencies are split into different bands. There is ongoing research that attempts to correlate the frequencies in the different bands to different emotional states and levels of concentration. With the increase in availability of commercial EEG devices, such studies can be sped up by making the collection of data easier, faster, and mobile. Creating a mobile application that facilitates the gathering of brainwave data using a commercially available EEG device, can potentially help make brain controlled interfaces feasible. A larger pool of data would be available to help perfect algorithms that would associate certain brainwave patterns with a computer action.

TABLE OF CONTENTS

Statement of the Problem

In 2012 BCI(Brain Computer Interfaces) are still not ready for the mass market. One of the main reasons for this is the difficulty of mapping computer tasks to patterns in EEG. Although there has been projects that successfully let users for example drive a wheelchair, such systems cannot be scaled to more complicated interfaces.

Most attempts at creating a BCI involve rely on changes in motor cortex areas in the brain during imagined movement. For example, a person imagining they are moving their right arm causes changes in the motor cortex that can be differentiated to the signal created when the subject imagines moving their left arm. This limits the amount of things that can be done with BCI and can be difficult to control. More recent research in the EEG field attempts to analyze different patterns in EEG signals without relying on these known changes in the motor cortex, which could increase the number of mental tasks that can be mapped to EEG patterns.

Another hurdle that BCI has to overcome is the cost of EEG hardware. However, this has started to change with the advent of different companies who now sale commercial EEG headsets to the average user. The lack of software for this devices has kept the user base low and has caused some of these companies to go out of business. For BCI technology to become widespread it must become easier to use and setup.

The goal of this project is to combine the emerging EEG hardware with the now familiar interface of smartphones. The end product is an application that sets up a platform for the mobile collection and analysis of EEG data. The Brain Recorder application is not intended as a product for entertainment or to provide a BCI functionality; rather is is a research tool to facilitate the development of the above mentioned EEG pattern recognition techniques with the help of the average user.

Related Work

Being able to record brain activity is the beginning of the development of future brain computer interaction software. For example in one UC Berkeley's labs, scientists recorded blood flow in the brain in reaction to watching different movies[1]. Because of these recordings, they were able to recreate images of what the person was looking at afterwards. In some media articles, this breakthrough is presented as the beginnings of a machine capable of recording what humans see in their dreams. Although this example used a far more sophisticated way of collecting brain data, the idea is similar in that there needs to be a way to collect brain data and associate it with something before decoding algorithms can be implemented.

1 Reconstructing Visual Experiences from Brain Activity Evoked by Natural Movies
Shinji Nishimoto, An T. Vu, Thomas Naselaris, Yuval Benjamini, Bin Yu, Jack L. Gallant
Current Biology - 11 October 2011 (Vol. 21, Issue 19, pp. 1641-1646)

With the advent of commercial EEG, devices there have been many studies that are devoted to verifying their usefulness[2]. There have even been some systems that have been developed already that attempt to use neural signals from a commercial EEG headset to control an iphone.[3]

Background Information

Neurosky and the Mindset

NeuroSky, Inc. was founded in 2004 in the Silicon Valley with the stated mission to "make BCI technology available to any industry". Neurosky produces a chip known as the ThinkGear ASIC Module, an embedded solution that claims to address many of the issues that had previously prevented EEG devices to enter the consumer market. One of the benefits this chip offers is the filtering of ambient noise created by muscle movement or nearby electrical devices.

Neurosky has partnered with large toy companies that utilize the Thinkgear AM. Some known products that use this chip are:

- Mattel's Mindflex toy.
- Uncle Milton's Star Wars Force Trainer.
- Neurowear's Necomimi.

Although Neurosky focuses on developing technologies that is used by other companies, they do have some consumer products. The one used in this project is the Neurosky Mindset.
The Neurosky Mindset is a headset that provides raw EEG signals over bluetooth. It uses dry contact sensors meaning that conducting gel is not necessary for the electrodes to work. One of the sensors is placed over the forehead while three sensors are placed over the ear. The ear sensors are important because they provide a reference signal that contains the noise but not the EEG signals as well as to serve as an electrical ground. This makes the filtering of noise more simpler and more precise.

Android

The android os was the mobile operating system of choice for this project because of several factors. One of them was because of the availability of android devices to test the application. Another reason was because of the large developer community android has for support. The major reason that android was chosen over IOS was because of its

2 A review of the commercial brain-computer interface technology from perspective of industrial robotics. Biao Zhang. Corp. Res. Center, ABB Inc., Windsor, CT, USA Automation and Logistics (ICAL), 2010 IEEE International Conference. 16-20 Aug. 2010. p(379 - 384)

3 Andrew Campbell, Tanzeem Choudhury, Shaohan Hu, Hong Lu, Matthew K. Mukerjee, Mashfiqui Rabbi, and Rajeev D.S. Raizada. 2010. NeuroPhone: brain-mobile phone interface using a wireless EEG headset. In*Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds* (MobiHeld '10). ACM, New York, NY, USA, 3-8. DOI=10.1145/1851322.1851326 http://doi.acm.org/10.1145/1851322.1851326

open bluetooth stack. Communication over bluetooth was desired, making an android device and the Mindset good candidates for this project.

Development Environment
The Brain Recorder application was developed using Eclipse IDE to facilitate the use of existing android libraries.

To set up the environment the following steps were taken to include all the necessary libraries and tools:
1. Install JDK- Needed for java development.
2. Install Eclipse IDE- The java developer version was used.
3. Download and install the android SDK - Needed for access to android development libraries.
4. Install the Android Development Tools – Helps in developing for android.
5. Add development platform 2.3 – Android version of the phone that this application was tested on.
6. Create the Brain Recorder project.
7. Import ThinkGear library – This is a library provided by Neurosky for communicating between their devices and an android device. At the time this project was developed, this API was in version beta 9.
8. Import aichart library – This is a free open source library that provides android projects with graphing and charting capabilities.

Initial Design

The initial design of this application was done before taking into account the usability offered by the beta version of the ThinkGear android API offered by Neurosky.

The design was centered around providing the following functionalities:
A. To be able to see in real time raw EEG data.
B. To be able to store and review said data.
C. The ability to use the application as different users.
D. The ability to customize the data collection and appearance based on the user.

The Tables
In order to organize and store the data table skeletons where created. Android stores information in content providers or sqlite databases. This project was designed to use sqlite databases which can be accessed through queries, much like a regular sql database.

One of the tables was designed to store user information
This table was designed to store the user information to customize the look of the application. The other table was designed to store the raw data collected from the headset.

Each user would have a table created for them, the primary key being a combination of a timestamp and the “thought” that was recorded. This assured that more than entry could be recorded with the same “thought” keyword. Having separate tables also simplifies lookup time and lets tables remain smaller.

The GUI
The initial GUI design for this project was created with the idea of having a separate screen for each function of the application. Following is a figure with all the different screens and the explanation for each of them.

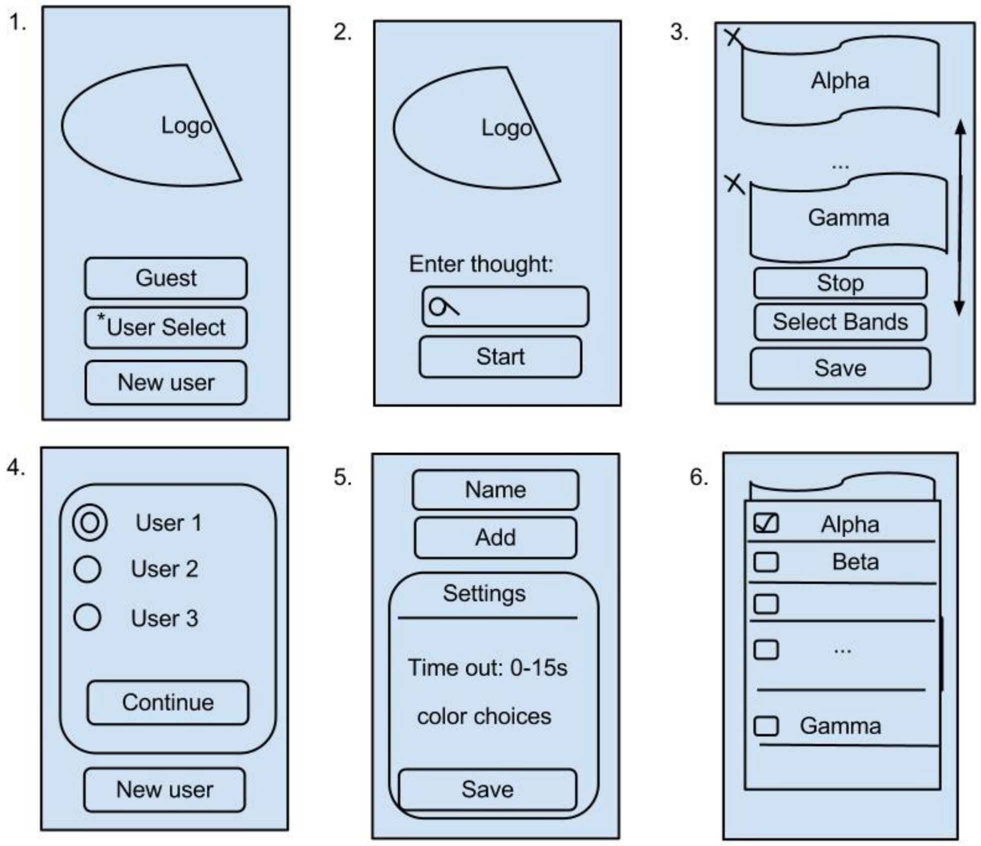


Figure 1. Initial Brain Recorder GUI design.

1. Welcome screen - The initial screen lets the user choose between creating a user, logging in as a user, or logging in as as a guest.

2. Thought screen - If the user chooses to log in as a guest, a screen is displayed where the user inputs a thought they will concentrate on. This screen will also be displayed if an existing user is chosen. If the thought already exists in the database the user can choose to display a previous recording of it.

3. Display Screen - Once a thought is chosen, the recording begins and the brainwave frequencies are displayed. Each band is shown in it own graph, allowing the user to select which bands are displayed on screen. If the save button is pressed the user will be taken back to the second screen.

4. User Select Screen - If the user initially chose to log in as an existing user he will be presented with a list to choose from. He can also create a new user.

5. New User/ User settings Screen - When creating a new user, there will be setting the person gets to choose such as the colors the graphs will be displayed in and how many data points to collect for each thought.

6. Select Band screen - When displaying the graphs, the select bands button will pop up a screen that lets the user easily choose which graphs are displayed.

Problems Encountered

The Long Setup
One of the initial problems encountered was setting up the development environment. Getting all the necessary components was harder than I had initially thought and required several attempts with a lot of time spent reading up on android development forums. As a first time android developer, a lot of learning took place before a complete development environment was set up.

The environment had been originally set up to work on the android 4.0 Ice Cream Sandwich operating system for increased future compatibility. The testing would be done through an emulator that comes with the android development tools package. However, this proved to be unsuccessful since this emulator does not support bluetooth capabilities or any of the sensors without several complicated workarounds. The solution was to instead develop for the Android 2.3. platform since that was the only android device physically available for testing.

The Over Complicated GUI
I decided to implement the GUI first since that would lay the groundwork for any background processes. After implementing a couple of the screens it became clear that the initial design over complicated the mental picture of the application. Some of the screens seemed too empty containing a single button. This was revised down to three screens.

The Database
A database handler was written for this project, but there was a problem that could not be tracked down due to lack of experience debugging android. The phone the application was being tested on also did not have enough memory available which prevented the testing of large persistent databases.

The Data
The original plan for this project included collecting raw data from the headset as well as frequency bands values calculated on the chip. Unfortunately at the time the thinkgear-android9beta API seemed not to work well with the phone that was being used for testing. When the headset was connected in "raw mode" the bluetooth connection would stop working after a few points were collected.
There are times when the data being received is not entirely accurate. This is because of improper adjustment of the the headset; more specifically the ear grounding sensors are difficult to adjust correctly.

The Bluetooth Connection
One of main initial problems encountered was the unreliability of the bluetooth connection. When using the API provided by Neurosky, it assumes that the headset is already paired. Each time a new set of data is collected, the bluetooth connection is established, and it is closed when the data collection is finished. There are times when the bluetooth connection cannot be established after this point without first restarting the headset. This might be because the headset can only establish one connection at a time and when the second connection tries to be established, the first one still has not been cleared.

The Graphing
There is currently no native android graphing library, which third party libraries to fulfill the graphing capabilities of the Brain Recorder application. Many different open source libraries were researched and tested for compatibility with the code from the Brain Recorder application. One of the initial ideas was to do real time graphing of the data being collected, but this was not offered by most android charting libraries. Those that did offer it and offered more professional looking graphs were not free.

Final Application

The communication between the Mindset and the Brain Recorder application was done using message classes set up by the Thinkgear-android9beta API. The message classes that are supported by the library's TGDevice object include messages about connection status, signal quality, Neurosky's esense meters, and EEG power bands. This communication can be seen in figure 2, which is a simple system architecture diagram. It is based on these values that the information displayed in the application is updated.

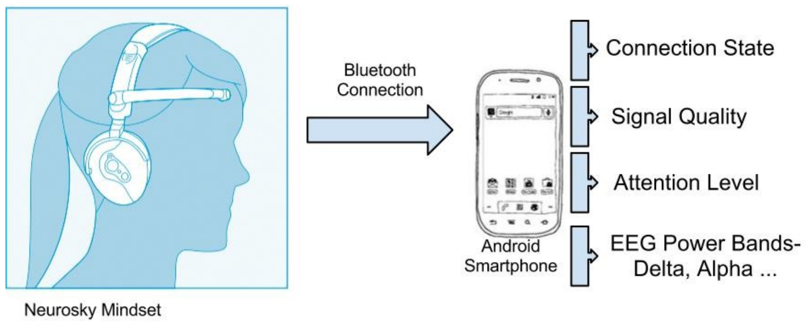


Figure 2. System Architecture Diagram

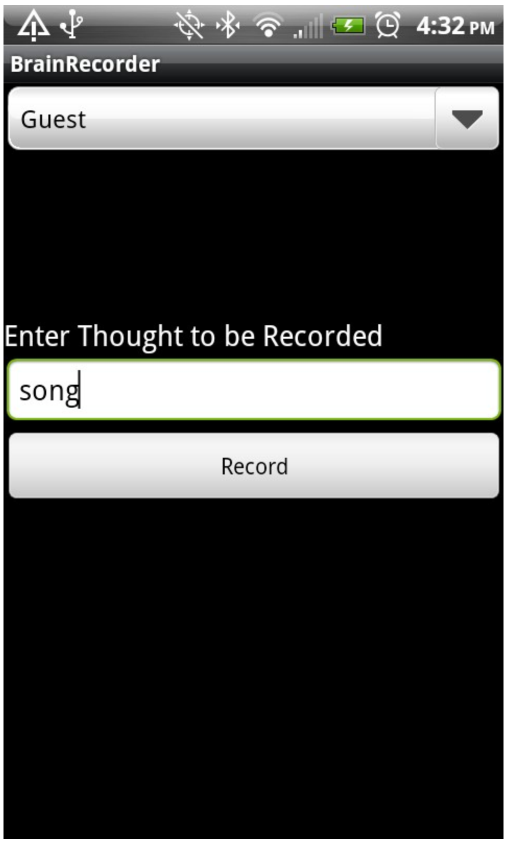


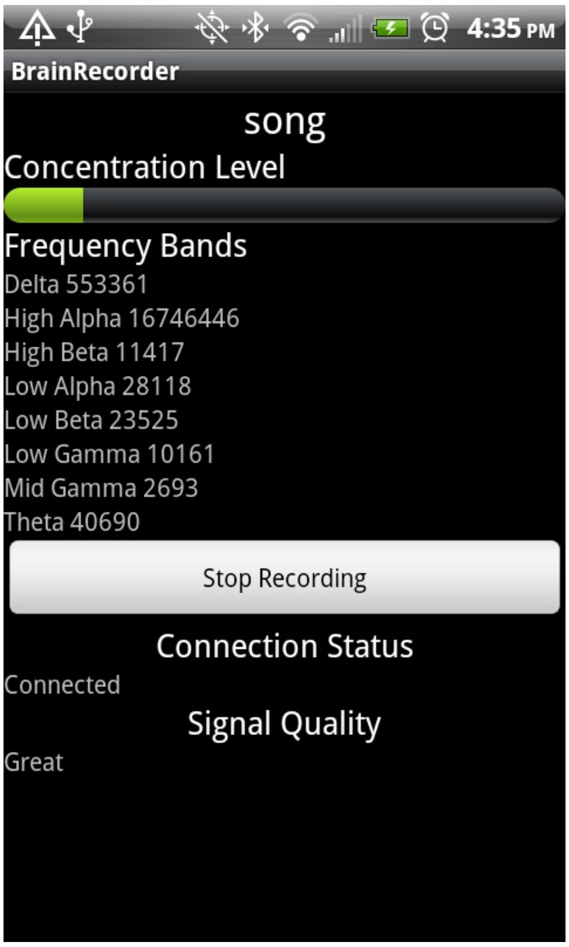
Figure 3. Brain Recorder welcome screen.

Screen 1
The initial screen where the user enters a word that will be used as a keyword in the database to store the EEG data. The user will concentrate on this word or "thought" during the recording phase.

This screen solves some of the problems described in the previous section. This welcome screen simplifies the log in and initial screens into a single one. At the top there is a "spinner" that lets you log in as a different user. The ability to add other users was not implemented.

When the Record button is pressed, whatever is entered in the prompt will be sent to the next activity and the bluetooth connection will be started.

The code for this screen can be found in appendix A under the WelcomeBrainRecorder.java title. The layout design for this screen is found also in appendix B in the main.xml file.



BrainRecorder

song

Concentration Level

Frequency Bands

Delta 553361

High Alpha 16746446

High Beta 11417

Low Alpha 28118

Low Beta 23525

Low Gamma 10161

Mid Gamma 2693

Theta 40690

Stop Recording

Connection Status

Connected

Signal Quality

Great

Figure 4. Brain Recorder recording screen.

Screen 2
This screen is shown when the app is in "recording" mode.

The thought that was entered in the previous screen is displayed at the top. Below that is a bar that changes according to the concentration level. The concentration level is based on a value calculated on board the chip that is labeled by neurosky as an "attention esense meter."

"*Attention" and "Meditation" eSense meters output by NeuroSky's MindSet are comprised of a complex combination of artifact rejection and data classification methods*.

Below the bar, the incoming data is displayed so the user will know that the connection is working. This data is followed by a button that will stop the collection of data and display a graph with the data collected so far.

At the bottom of the screen there are two system status messages. The issues with the bluetooth connection mentioned in the problems encountered section are still present, so the connection status message helps with that. It will let the user know when they need to reset the headset.

The signal quality message lets the user know if the the sensors are making proper contact or if there needs to be some readjustment. These two status messages are extremely helpful when setting up. The idea for these was gotten from some usability testing and a heuristic evaluation on the pc software that comes with the mindset. This evaluation can be found in appendix C.

The code for this screen is in the appendix under appendix A, brainRecorderActivity.java.
The xml layout is under appendix B, recording.xml.

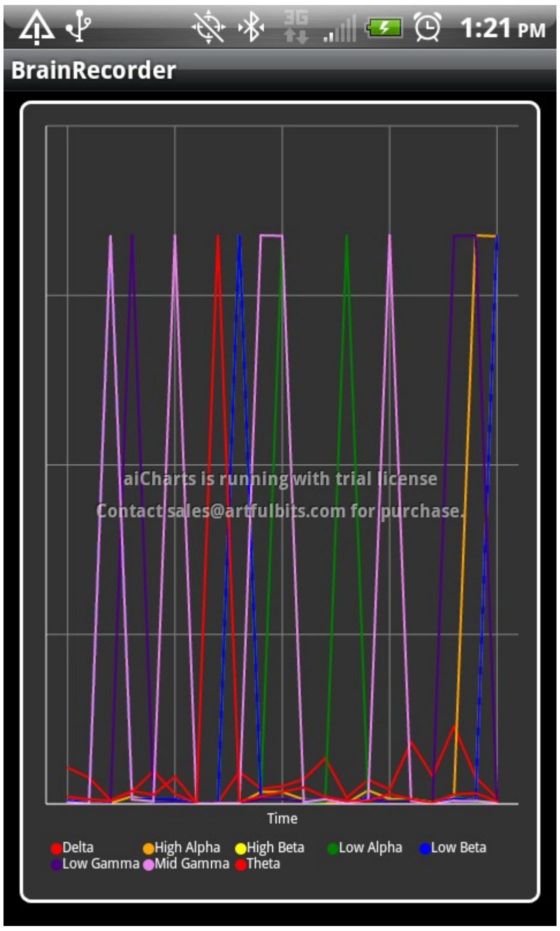


Figure 5. Brain Recorder graph screen.

Screen 3
This graph displays the data collected for the different bands. The x axis is time while the y axis does not have any units. According to the Neurosky knowledge base:

"*The ASIC_EEG_POWER_INT values are indications of relative amplitudes of the individual EEG bands.*
Typically, power spectrum band powers would be reported in units such as Volts-squared per Hz (V^2/Hz), but since our values have undergone a number of complicated transforms and rescale operations from the original voltage measurements, there is no longer a simple linear correlation to units of Volts. Hence, we do not try to label them with any conventional unit. They are useful as an indication of whether each particular band is increasing or decreasing over time, and how strong each band is relative to the other bands."

This graph was implemented using a trial version of the aicharts library created by Artfulbits[4], hence the watermark. This library does support zooming capabilities which would be useful, but that was not implemented.
The library works by using xml charts, the one used for this project is found in the appendix under chart.xml.

Troubleshooting
Note: Before using the Brain Recorder Application for the first time make sure the mindset is paired to the phone.

For a connection problem:
- Make sure bluetooth is enabled on your phone.
- The Mindset it turned on and fully charged.
- Mindset is paired.

In case of persisting connection problems disconnect power off then on the Mindset and attempt to connect again.

4 http://www.artfulbits.com/

For best data collection look at the signal quality. It must say “Great”, otherwise adjust the contacts that go over the ear.

Future work

One of the main future improvements to this project would be the implementation of a server application that would be receiving the data from the Brain Recorder android application. A portal could be made so users could log on to the server and see how their brainwave patterns compares to other people.

This server program would be better suited for performing comparison computations on large data sets and would serve the purpose of centralizing all data to be analyzed. This also sets up the necessary components to make a multiplayer game using a Neurosky headset.Giving the Brain Recorder Application the ability to export databases to a server through a network connection would also help keep the consumption of memory for this application low.

Some other improvements to the application would be with aesthetic components such as making the graph look nicer, addings zooming capabilities, and giving the user the option to choose and change the look of the application.

Conclusions

With the completion of this project users can now add analyzing their brainwaves to the list of things they can do on their smartphone. Brain Recorder adequately fulfils the initial goal to make an android application that the collection of EEG data portable and relatively easy to use.
Brain Recorder also sets up a platform that future students can build from. Neurosky has already released a cheaper and more reliable version of the Mindset headset called the Mindwave Mobile which could also be used with the Brain Recorder application. This indicates a future for BCI interfaces that are affordable to everyone and could lead to future Cal Poly students working on projects similar to this one.

After the completion of this project I have gained a deeper understanding of EEG data and the true capabilities that the commercial EEG headsets of today.

APPENDIX A

Source Code

WelcomeBrainRecorder.java

```
package cpslo.brainrecorder.activity;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;
import com.neurosky.thinkgear.*;

public class WelcomeBrainRecorder extends Activity {
        public String thought;

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      Spinner spinner = (Spinner) findViewById(R.id.userSelector);
      ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
            this, R.array.user_array, android.R.layout.simple_spinner_item);

adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
      spinner.setAdapter(adapter);
   }

   @Override
   public void onDestroy() {
      super.onDestroy();
   }
```

```
    public void record(View view) {
          final TextView input = (TextView) findViewById(R.id.thoughtBar);
       thought = input.getText().toString();
          Intent intent = new Intent(this, BrainRecorderActivity.class);
          intent.putExtra("thought", thought);
          startActivity(intent);
    }

    public class MyOnItemSelectedListener implements OnItemSelectedListener {

       public void onItemSelected(AdapterView<?> parent,
           View view, int pos, long id) {
         Toast.makeText(parent.getContext(), "Selected User " +
             parent.getItemAtPosition(pos).toString(), Toast.LENGTH_LONG).show();
       }

       public void onNothingSelected(AdapterView parent) {
         // Do nothing.
       }
    }
}
```

BrainRecorderActivity.java

```
package cpslo.brainrecorder.activity;

import java.util.ArrayList;
import java.util.List;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.content.Context;
import android.content.Intent;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;

import com.artfulbits.aiCharts.ChartView;
import com.artfulbits.aiCharts.Base.ChartArea;
import com.artfulbits.aiCharts.Base.ChartSeries;
import com.artfulbits.aiCharts.Types.ChartTypes;
import com.neurosky.thinkgear.*;

public class BrainRecorderActivity extends Activity {
        BluetoothAdapter bluetoothAdapter;
        TextView myTextView;
        ChartView chartView;
        TGRawMulti rawData;
        TGDevice tgDevice;
        TGEegPower fbands;
        final boolean rawEnabled = false;
        frequencyTable db;
        EEGPoint current;
        String thought;
        boolean graphing = false;
        List<TGEegPower> points;
        private ProgressBar mProgress;
   /** Called when the activity is first created. */

        @Override
   public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.recording);
        points = new ArrayList<TGEegPower>();

        //db = new frequencyTable(this);
        Bundle extras = getIntent().getExtras();
        thought = extras.getString("thought");
        current = new EEGPoint(thought);

        myTextView = (TextView) findViewById(R.id.thinking);
          myTextView.setText(thought);

        /*start up bluetooth*/
        bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        if(bluetoothAdapter == null) {
          // Alert user that Bluetooth is not available
          Toast.makeText(this, "Bluetooth not available", Toast.LENGTH_LONG).show();
          finish();
          return;
        }else {
          /* create the TGDevice */
          tgDevice = new TGDevice(bluetoothAdapter, handler);
        }

          if(tgDevice.getState() != TGDevice.STATE_CONNECTING &&
tgDevice.getState() != TGDevice.STATE_CONNECTED)
                  tgDevice.connect(rawEnabled);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        tgDevice.close();
    }

    @Override
    public void onResume() {
        super.onResume();

    }

    public void graph(View view) {
          graphing = true;
          setContentView(R.layout.graphing);
          /*chartView = new ChartView(this);
                  ChartSeries series1 = new ChartSeries("s1", ChartTypes.Line);
                  ChartSeries series2 = new ChartSeries("s2", ChartTypes.Line);
                  ChartSeries series3 = new ChartSeries("s3", ChartTypes.Line);
                  ChartSeries series4 = new ChartSeries("s4", ChartTypes.Line);
                  ChartSeries series5 = new ChartSeries("s5", ChartTypes.Line);
                  ChartSeries series6 = new ChartSeries("s6", ChartTypes.Line);
                  ChartSeries series7 = new ChartSeries("s7", ChartTypes.Line);
                  ChartSeries series8 = new ChartSeries("s8", ChartTypes.Line);*/


          chartView = (ChartView) findViewById(R.id.chartView);
                  ChartSeries series1 = chartView.getSeries().get("Delta");
                  ChartSeries series2 = chartView.getSeries().get("High Alpha");
                  ChartSeries series3 = chartView.getSeries().get("High Beta");
                  ChartSeries series4 = chartView.getSeries().get("Low Alpha");
                  ChartSeries series5 = chartView.getSeries().get("Low Beta");
                  ChartSeries series6 = chartView.getSeries().get("Low Gamma");;
                  ChartSeries series7 = chartView.getSeries().get("Mid Gamma");
                  ChartSeries series8 = chartView.getSeries().get("Theta");

          //        ChartArea area1 = chartView.getAreas().get("area");
                  //ChartArea area2 = new ChartArea("area2");

                  //area1.getDefaultYAxis().setTitle("Frequency");
                  //area1.setName("Frequency Bands");

                  for(int i = 0; i < points.size(); i++)
                  {
                            series1.getPoints().addXY(i, points.get(i).delta);
                            series2.getPoints().addXY(i, points.get(i).highAlpha);
                            series3.getPoints().addXY(i, points.get(i).highBeta);
                            series4.getPoints().addXY(i, points.get(i).lowAlpha);
                            series5.getPoints().addXY(i, points.get(i).highBeta);
                            series6.getPoints().addXY(i, points.get(i).lowGamma);
                            series7.getPoints().addXY(i, points.get(i).midGamma);
                            series8.getPoints().addXY(i, points.get(i).theta);
                  }
                  /*
                  series1.setArea("area1");
                  series2.setArea("area1");
                  series3.setArea("area1");
                  series4.setArea("area1");
                  series5.setArea("area1");
                  series6.setArea("area1");
                  series7.setArea("area1");
```

```
                series8.setArea("area1");
*/

                //chartView.getAreas().add(area1);
                //chartView.getAreas().add(area2);
                //chartView.getAreas().add(area3);
                /*
                chartView.getSeries().add(series1);
                chartView.getSeries().add(series2);
                chartView.getSeries().add(series3);
                chartView.getSeries().add(series4);
                chartView.getSeries().add(series5);
                chartView.getSeries().add(series6);
                chartView.getSeries().add(series7);
                chartView.getSeries().add(series8);
                */

    }

    /**
     * Handles messages from TGDevice
     */
    private final Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
          if(!graphing)
          {
          switch (msg.what) {
           case TGDevice.MSG_STATE_CHANGE:

               switch (msg.arg1) {
                       case TGDevice.STATE_IDLE:
                          break;
                       case TGDevice.STATE_CONNECTING:
                           myTextView = (TextView) findViewById(R.id.connectStatus);
                           myTextView.setText("Connecting ...");
                           break;
                       case TGDevice.STATE_CONNECTED:
                           myTextView = (TextView) findViewById(R.id.connectStatus);
                           myTextView.setText("Connected");
                           tgDevice.start();
                          break;
                       case TGDevice.STATE_NOT_FOUND:
                           myTextView = (TextView) findViewById(R.id.connectStatus);
```

```
                        myTextView.setText("Connection Not Found, reset and try 
again.");;
                        break;
                    case TGDevice.STATE_NOT_PAIRED:
                        myTextView = (TextView) findViewById(R.id.connectStatus);
                        myTextView.setText("There is not Mindset paired to this device.");
                        break;
                    case TGDevice.STATE_DISCONNECTED:
                        myTextView = (TextView) findViewById(R.id.connectStatus);
                        myTextView.setText("Disconnected");
            }

            break;
        case TGDevice.MSG_POOR_SIGNAL:
                myTextView = (TextView) findViewById(R.id.signal);
                if(msg.arg1 == 0)
                {
                        myTextView.setText("Great");

                }
                else if(msg.arg1 > 50)
                {
                        myTextView.setText("Poor, please readjust headset.");
                }
                break;
        case TGDevice.MSG_ATTENTION:
                mProgress = (ProgressBar) findViewById(R.id.attentionBar);
                mProgress.setProgress(msg.arg1);
                break;
        case TGDevice.MSG_MEDITATION:
                break;
        case TGDevice.MSG_BLINK:
                        //tv.append("Blink: " + msg.arg1 + "\n");
                break;
        case TGDevice.MSG_EEG_POWER:
                fbands = (TGEegPower)msg.obj;
            points.add(fbands);

                myTextView = (TextView) findViewById(R.id.ch1);
                myTextView.setText("Delta " + fbands.delta);
                myTextView = (TextView) findViewById(R.id.ch2);
                myTextView.setText("High Alpha " + fbands.highAlpha);
                myTextView = (TextView) findViewById(R.id.ch3);
                myTextView.setText("High Beta " + fbands.highBeta);
                myTextView = (TextView) findViewById(R.id.ch4);
```

```
                    myTextView.setText("Low Alpha " + fbands.lowAlpha);
                    myTextView = (TextView) findViewById(R.id.ch5);
                    myTextView.setText("Low Beta " + fbands.lowBeta);
                    myTextView = (TextView) findViewById(R.id.ch6);
                    myTextView.setText("Low Gamma " + fbands.lowGamma);
                    myTextView = (TextView) findViewById(R.id.ch7);
                    myTextView.setText("Mid Gamma " + fbands.midGamma);
                    myTextView = (TextView) findViewById(R.id.ch8);
                    myTextView.setText("Theta " + fbands.theta);
                    break;
               case TGDevice.MSG_LOW_BATTERY:
                    Toast.makeText(getApplicationContext(), "Low battery!",
Toast.LENGTH_SHORT).show();
                    break;

               default:
                    break;
             }
             }
             }
   };
}
```

frequencyTable.java

```
package cpslo.brainrecorder.activity;

import java.util.ArrayList;
import java.util.List;

import com.neurosky.thinkgear.TGEegPower;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class frequencyTable extends SQLiteOpenHelper {

   private static final int DATABASE_VERSION = 2;
   private static final String TABLE_NAME = "fBands";
        private static final String THOUGHT = "thought";
        private static final String DELTA = "delta";
        private static final String HIGH_ALPHA = "halpha";
        private static final String HIGH_BETA = "hbeta";
        private static final String LOW_ALPHA = "lalpha";
        private static final String LOW_BETA = "lbeta";
        private static final String LOW_GAMMA = "lgamma";
        private static final String MID_GAMMA = "mgamma";
        private static final String THETA = "theta";
        private static final String DATE = "date";

        private static final String TABLE_CREATE =
            "CREATE TABLE " + TABLE_NAME + " (" +
            DELTA + " INTEGER," +
            HIGH_ALPHA + " INTEGER," +
            HIGH_BETA + " INTEGER," +
            LOW_ALPHA + " INTEGER," +
            LOW_BETA + " INTEGER," +
            LOW_GAMMA + " INTEGER," +
            MID_GAMMA + " INTEGER," +
            THETA + " INTEGER," +
            DELTA + " INTEGER," +
            ");";
        private static final String DATABASE_NAME = "BrainRecorder";

    frequencyTable(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(TABLE_CREATE);
    }

    // Upgrading database
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Drop older table if existed
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        // Create tables again
        onCreate(db);
    }

    /**
     * All CRUD(Create, Read, Update, Delete) Operations
     */
    void addPoint(TGEegPower fbands) {
        SQLiteDatabase db = this.getWritableDatabase();

        ContentValues values = new ContentValues();
    values.put(DELTA, fbands.delta);
    values.put(HIGH_ALPHA, fbands.highAlpha);
    values.put(HIGH_BETA, fbands.highBeta);
    values.put(LOW_ALPHA, fbands.lowAlpha);
    values.put(LOW_BETA, fbands.lowBeta);
    values.put(LOW_GAMMA, fbands.lowGamma);
    values.put(MID_GAMMA, fbands.midGamma);
    values.put(THETA, fbands.theta);

    // Inserting Row
    db.insert(TABLE_NAME, null, values);
    db.close(); // Closing database connection
}

// Getting single contact
List<TGEegPower> getAllPoints() {
    List<TGEegPower> pointsList = new ArrayList<TGEegPower>();
    // Select All Query
    String selectQuery = "SELECT  * FROM " + TABLE_NAME;

    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, null);

    // looping through all rows and adding to list
    if (cursor.moveToFirst()) {
        do {
                TGEegPower current = new TGEegPower();
                current.delta = Integer.parseInt(cursor.getString(0));
                current.highAlpha = Integer.parseInt(cursor.getString(1));
                current.highBeta = Integer.parseInt(cursor.getString(2));
                current.lowAlpha = Integer.parseInt(cursor.getString(3));
                current.lowBeta = Integer.parseInt(cursor.getString(4));
                current.lowGamma = Integer.parseInt(cursor.getString(5));
                current.midGamma = Integer.parseInt(cursor.getString(6));
                current.theta = Integer.parseInt(cursor.getString(7));
                        // Adding contact to list
            pointsList.add(current);
        } while (cursor.moveToNext());
    }
    db.close();
    // return contact list
    return pointsList;
}
```

```
    // Getting contacts Count
    public int getCount() {
        String countQuery = "SELECT  * FROM " + TABLE_NAME;
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(countQuery, null);
        cursor.close();

        // return count
        return cursor.getCount();
    }
}
```

APPENDIX B

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="cpslo.brainrecorder.activity"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="10" />
          <uses-permission android:name="android.permission.BLUETOOTH" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".WelcomeBrainRecorder"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".BrainRecorderActivity">
        </activity>
        <activity android:name=".frequencyTable">
        </activity>
    </application>

</manifest>
```

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:gravity="right"
   android:orientation="vertical" >

   <Spinner
      android:id="@+id/userSelector"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:prompt="@string/user_prompt" />

   <TextView
      android:id="@+id/thoughtPrompt"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center_vertical"
      android:layout_marginTop="100dip"
      android:text="Enter Thought to be Recorded"
      android:textAppearance="?android:attr/textAppearanceMedium" />

      <EditText
         android:id="@+id/thoughtBar"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"

         android:maxLength="40"
         android:singleLine="true" />

   <Button
      android:id="@+id/recordButton"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Record"
        android:onClick="record"/>

</LinearLayout>
```

recording.xml

```
   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:id="@+id/linearLayout1"
      android:layout_width="fill_parent"
```

```
android:layout_height="fill_parent"
android:orientation="vertical" >

<TextView
   android:id="@+id/thinking"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text=""
   android:layout_gravity="center"
   android:textAppearance="?android:attr/textAppearanceLarge" />

<TextView
   android:id="@+id/attentionText"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Concentration Level"
   android:textAppearance="?android:attr/textAppearanceMedium" />

<ProgressBar
   android:id="@+id/attentionBar"
   style="?android:attr/progressBarStyleHorizontal"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content" />

<TextView
   android:id="@+id/channels"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Frequency Bands"
   android:textAppearance="?android:attr/textAppearanceMedium" />

<TextView
   android:id="@+id/ch1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Delta"
   android:textAppearance="?android:attr/textAppearanceSmall" />

<TextView
   android:id="@+id/ch2"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="High Alpha"
   android:textAppearance="?android:attr/textAppearanceSmall" />
```

```
<TextView
    android:id="@+id/ch3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="High Beta"
    android:textAppearance="?android:attr/textAppearanceSmall" />

<TextView
    android:id="@+id/ch4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Low Alpha"
    android:textAppearance="?android:attr/textAppearanceSmall" />

<TextView
    android:id="@+id/ch5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Low Beta"
    android:textAppearance="?android:attr/textAppearanceSmall" />

<TextView
    android:id="@+id/ch6"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Low Gamma"
    android:textAppearance="?android:attr/textAppearanceSmall" />

<TextView
    android:id="@+id/ch7"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Mid Gamma"
    android:textAppearance="?android:attr/textAppearanceSmall" />

<TextView
    android:id="@+id/ch8"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Theta"
    android:textAppearance="?android:attr/textAppearanceSmall" />

<Button
    android:id="@+id/stopButton"
    android:layout_width="fill_parent"
```

```
            android:layout_height="wrap_content"
            android:text="Stop Recording"
            android:onClick="graph"/>/>

        <TextView
            android:id="@+id/connectText"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:text="Connection Status"
            android:textAppearance="?android:attr/textAppearanceMedium" />
        <TextView
            android:id="@+id/connectStatus"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Connecting ..."
            android:textAppearance="?android:attr/textAppearanceSmall" />
        <TextView
            android:id="@+id/textView1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:text="Signal Quality"
            android:textAppearance="?android:attr/textAppearanceMedium" />
        <TextView
            android:id="@+id/signal"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text=""
            android:textAppearance="?android:attr/textAppearanceSmall" />

    </LinearLayout>
```

graphing.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
  <com.artfulbits.aiCharts.ChartView
    android:id="@+id/chartView"
    chart="@xml/chart"
    android:background="@android:drawable/alert_dark_frame"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"/>
```

```
</LinearLayout>
```

chart.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ai:chart xmlns:ai="http://www.artfulbits.com/android/aiCharts">
        <ai:area >
                <ai:area.xaxis labels_visible="false"/>
                <ai:area.yaxis labels_visible="false"/>
                <ai:area.xaxis title="Time"/>
                <ai:area.xaxis scrollbar_enabled="true"/>
        </ai:area>
        <ai:series type="Line" name="Delta"/>
        <ai:series type="Line" name="High Alpha"/>
        <ai:series type="Line" name="High Beta" />
        <ai:series type="Line" name="Low Alpha" />
        <ai:series type="Line" name="Low Beta"/>
        <ai:series type="Line" name="Low Gamma"/>
        <ai:series type="Line" name="Mid Gamma" />
        <ai:series type="Line" name="Theta" />
        <ai:legend/>

</ai:chart>
```

APPENDIX C

Heuristic Evaluation of Neurosky Software.

Visibility of system status
It is not always clear if the headset is connected. At times the graphs display nothing and no system status messages appear.

Match between system and the real world
The software is clear when displaying data on screen. Messages explaining the data appears when you hover over a graph. It would be helpful if more explanation was given about how to control your "brainwaves."

User control and freedom
There are only three different screens presented in the software and there are buttons that can easily switch between them.

Consistency and standards
In the main display, the brainwave data is presented in more than one way. This may be overwhelming and confusing to those who don't know that is the same data presented in different ways.

Error prevention
Seeing as the software is not very interactive, there is not much room for error after it is set up.
Reducing the number of buttons and options, helps prevent errors.

Recognition rather than recall
People using the Mindset software easily navigated through it. The few buttons present state their purpose obviously.

Flexibility and efficiency of use
The Mindset software is very easy to use for all users. Anyone familiar with computers can easily navigate through it, without any extra features for experts.

Aesthetic and minimalist design
A lot of information is presented in the main screen. This may be confusing but it helps reduce the number of screens the user has to switch between.

Help users recognize, diagnose, and recover from errors
Once the headset is connected, there are no error messages displayed. Even if the headset somehow becomes disconnected, there is no error message. This leads to the occasional frustration of not knowing why the graphs are not changing.

Help and documentation
The documentation that comes with headset contains easy to follow instructions on how to connect it using bluetooth.