CS2310 Project Report
Matthew O'Brien

My project is an extension to the SIS system which, taking input from a microphone sensor, calculates the approximate vocal range of the user. It uses three components on top of the SIS server, MicInput, NoteProc, and a GUI component. The components communicate with each other through messages sent through the SIS system and each perform a task or tasks within the system.

MicInput is the component which handles, processes, and propagates microphone data through the system. In order to process the data, a 3rd party Java library (TarsosDSP) was used (with permission from the professor). In Java, sound data is represented as a waveform (like with the .wav file format), presented as a byte array, with a byte at index i corresponding to the i^{th} sample of the waveform. From this waveform, the pitch can be extracted by performing a fast Fourier transform, as pitch corresponds directly to waveform frequency. TarsosDSP has built in functions which allow for a FFT with additional features (filtering high or low frequencies, smoothing results to maximize probability that the frequency returned is the primary frequency of the sample and not some background noise, and filtering results based on probability of accuracy (which allows the system to have meaningful results, even with lower quality sensors)). TarsosDSP is used to capture the audio and, whenever a testable sample is collected, calculate the pitch. Given a calculated pitch (and provided that the measurement was taken with a probability above a certain threshold), the component converts this to Scientific Pitch Notation, which is an integer representation of note and octave, with 0 corresponding to middle C. The system then sends a message identifying the note measured to NoteProc.

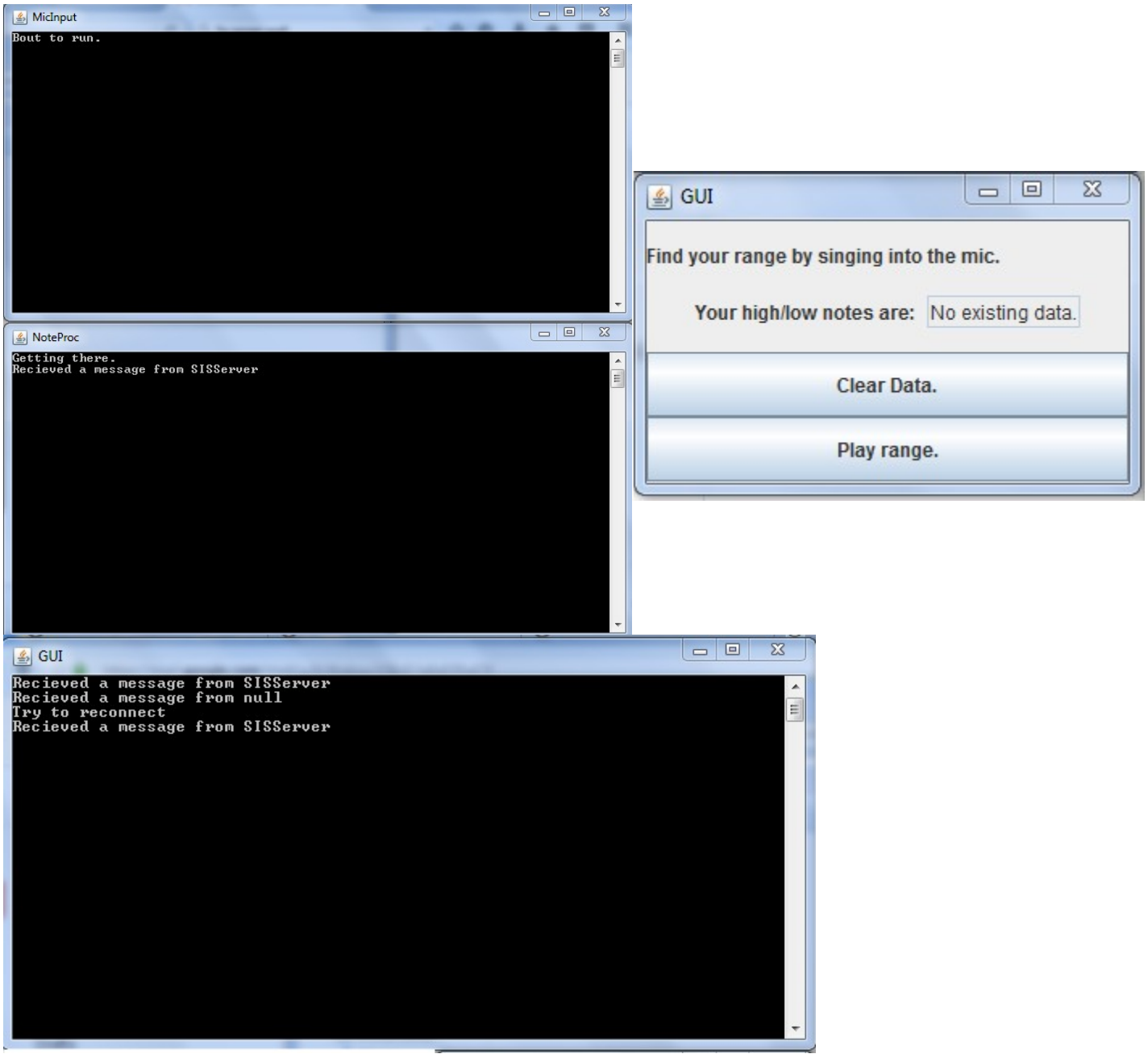
NoteProc maintains a histogram of notes and uses this to give a reasonable estimate of range to the user. Starting with the highest and lowest observed notes on the histogram, NoteProc checks the frequency of each, and if the frequency is lower than ½ of the average frequency it moves on the the next highest/lowest note and performs the same test. Once a range is finalized, it send a message containing the range in String format (##/##) to the GUI. In order to prevent early miscalculations of the range, the component only sends a message to the GUI once it has received a certain number of samples from the MicInput component. The histogram can be reset by the user through the GUI.

The GUI component is moderately simple. It has a test field which displays the range to the user (or a message approximating how many remaining samples are required) and two buttons. One button send a message to the NoteProc component which instructs it to reset its internal state. The other button takes the note range (provided there is one) and uses the MIDI interface in Java to play two notes corresponding to the high and low note of the user's range.

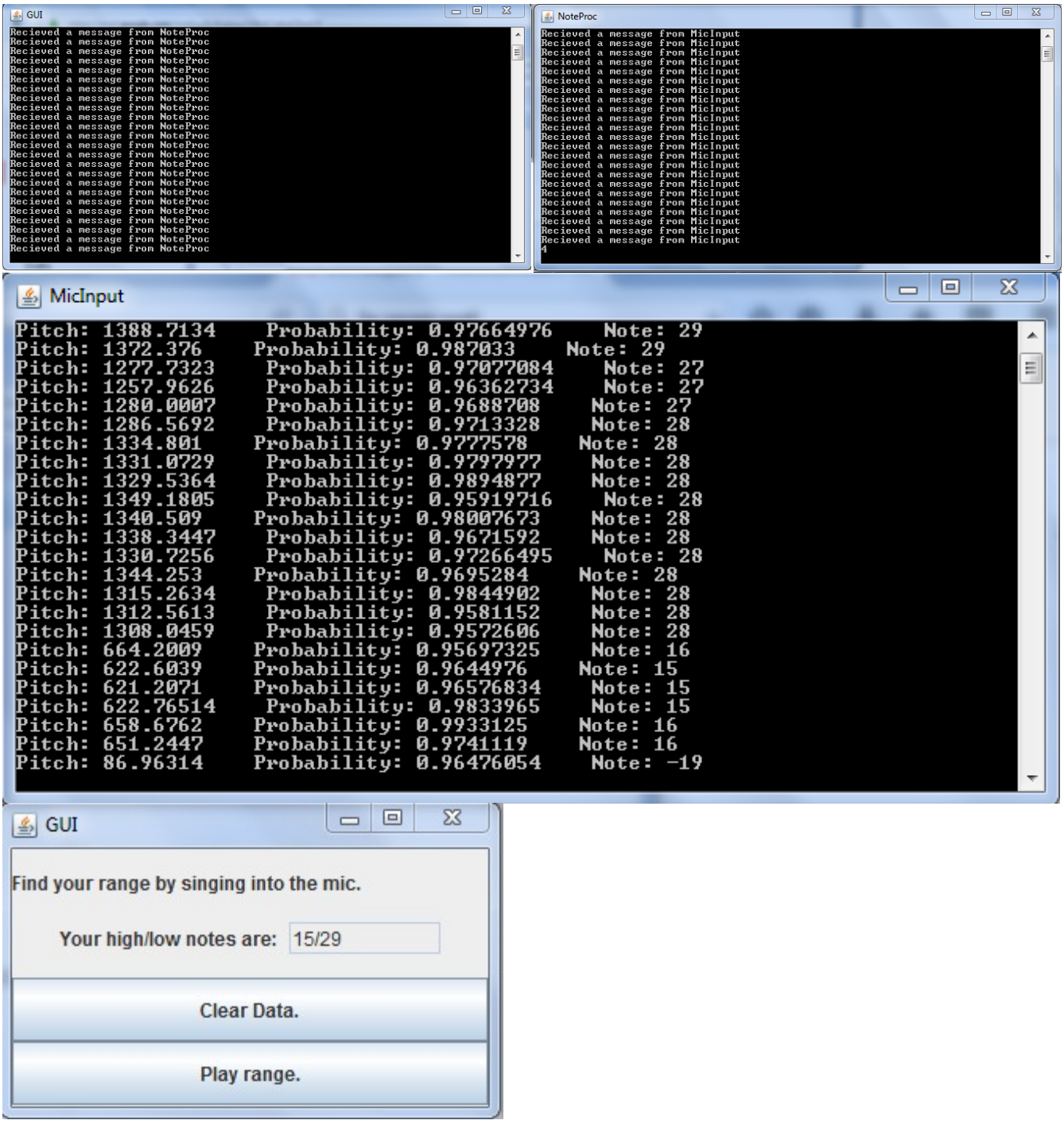
The system was tested and tuned to the Zalman ZM-MIC 1 High Sensitivity Microphone. The microphone has a habit of picking up background noise within a fairly large radius, so for display purposes the system has been tuned to almost exclusively pick up whistling, as the microphone introduces too much noise into the samples for anything else to be registered without also registering large amounts of false samples corresponding to background noise. With a higher quality sensor, the system could be properly configured to work well on human voice, (this was tested briefly and found to be the case with another, higher quality sensor which ceased functioning before any large amount of testing could be done with it).

Screenshot Demo:

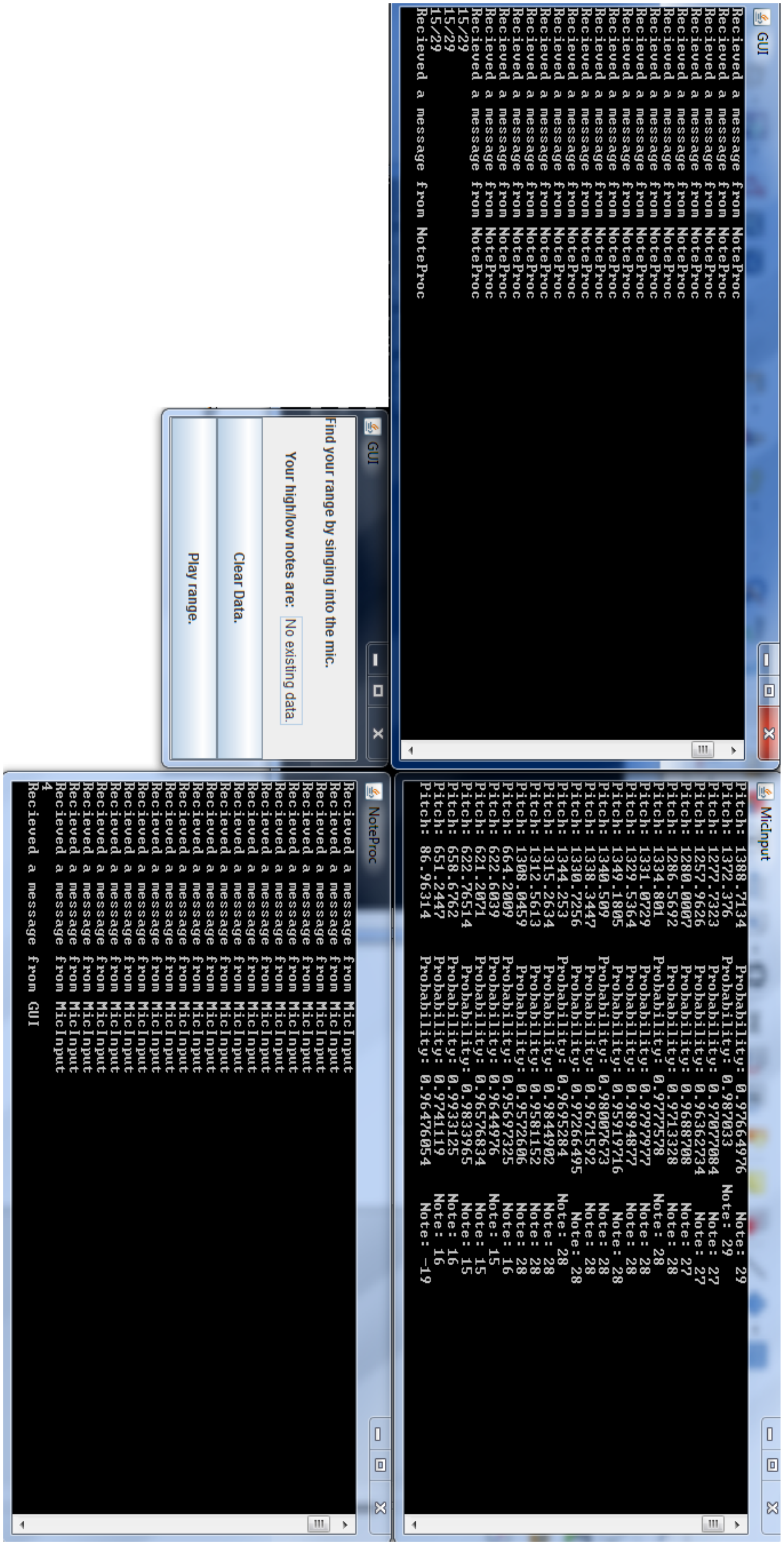
Begin of Run:



After Data Collection:



After Reset:


GUI
Recieved a message from NoteProc
GUI
Find your range by singing into the mic.
Your high/low notes are:
No existing data.
Clear Data.
Play range.
MicInput
NoteProc
Recieved a message from MicInput
Recieved a message from GUI

Appendix:

CreateMicInput.java:

```
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.SourceDataLine;
import javax.sound.sampled.TargetDataLine;


import be.tarsos.dsp.*;
import be.tarsos.dsp.beatroot.*;
import be.tarsos.dsp.effects.*;
import be.tarsos.dsp.filters.*;
import be.tarsos.dsp.io.*;
import be.tarsos.dsp.io.jvm.*;
import be.tarsos.dsp.mfcc.*;
import be.tarsos.dsp.onsets.*;
import be.tarsos.dsp.pitch.*;
import be.tarsos.dsp.resample.*;
import be.tarsos.dsp.synthesis.*;
import be.tarsos.dsp.ui.*;
import be.tarsos.dsp.ui.layers.*;
import be.tarsos.dsp.ui.layers.pch.*;
import be.tarsos.dsp.util.*;
import be.tarsos.dsp.util.fft.*;
import be.tarsos.dsp.wavelet.*;
import be.tarsos.dsp.wavelet.lift.*;
import be.tarsos.dsp.writer.*;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.StringTokenizer;
import java.util.*;
import java.io.*;
import java.net.*;
import java.lang.*;
import javax.mail.Authenticator;
import javax.mail.Message;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

public class CreateMicInput {

	// socket for connection to SISServer
	private static Socket universal;
	private static int port = 53217;
	// message writer
	private static MsgEncoder encoder;
	// message reader
	private static MsgDecoder decoder;

	// scope of this component
	private static final String SCOPE = "SIS.Scope1";
	// name of this component
	private static final String NAME = "MicInput";
	// messages types that can be handled by this component
	private static final List<String> TYPES = new ArrayList<String>(
				Arrays.asList(new String[] { "Setting", "Alert", "Confirm" }));

	// summary for all incoming / outgoing messages
```

```
  private static final String incomingMessages = "IN\t Confirm|Setting";
  private static final String outgoingMessages = "OUT\t Connect|Reading";

  // shared by all kinds of emergencies that can be generated by this component
  private static KeyValueList emergency = new KeyValueList();

  //static String alert_msg = "Patient has abnormal bloodPressure because of uncomfortable temperature";
  //static String doctorEmail = "sisfortest@outlook.com"; // default email
  static ArrayList<Double> tempRecord = new ArrayList<Double>();

  /*
   * Main program
   */
  public static void main(String[] args) {
                while (true) {
                                try {
                                                // try to establish a connection to SISServer
                                                universal = connect();

                                                // bind the message reader to inputstream of the socket
                                                decoder = new MsgDecoder(universal.getInputStream());
                                                // bind the message writer to outputstream of the socket
                                                encoder = new MsgEncoder(universal.getOutputStream());

                                                /*
                                                 * construct a Connect message to establish the connection
                                                 */
                                                KeyValueList conn = new KeyValueList();
                                                conn.putPair("Scope", SCOPE);
                                                conn.putPair("MessageType", "Connect");
                                                conn.putPair("IncomingMessages", incomingMessages);
conn.putPair("OutgoingMessages", outgoingMessages);
                                                conn.putPair("Role", "Basic");
                                                conn.putPair("Name", NAME);
                                                encoder.sendMsg(conn);

                                                // KeyValueList for inward messages, see KeyValueList for
                                                // details
                                                KeyValueList kvList;

                                                while (true) {
                                                                //this component doesn't receive messages, we just send them out.
    //Messages are readings where the important field is <Note>
    //There will be a decible test to make sure low tier background nopise doesn't get sent out.
    Boolean test = false;
    if(test)
    {
        //we're only gonna see in practice -20 to 24, but higher pitches exist
        System.out.print("Input test note (Scientific Pitch Notation: -18 to 28): ");
        String note = System.console().readLine();
        KeyValueList nMess = new KeyValueList();
        nMess.putPair("Scope", SCOPE);
        nMess.putPair("MessageType", "Alert");
        nMess.putPair("Sender", NAME);
        nMess.putPair("Receiver", "NoteProc");
        nMess.putPair("Note", note);
        encoder.sendMsg(nMess);
    }
    else
    {
        //actual code goes in here.
        AudioDispatcher d = AudioDispatcherFactory.fromDefaultMicrophone(1024,0);
        float sr = 44100;
        AudioProcessor highPass = new HighPass(110,sr);
        d.addAudioProcessor(highPass);
        PitchDetectionHandler printPitch = new PitchDetectionHandler()
        {
            @Override
            public void handlePitch(PitchDetectionResult pitch, AudioEvent a)
            {
                float p = pitch.getPitch();
                float prob = pitch.getProbability();
                int spn = -25;
```

```
                    if(p>0 && prob >= .95)
                    {
                       //get the note in here
                       System.out.print("Pitch: "+p+"    Probability: "+prob);
                       spn = getSPN(p);
                       System.out.println("    Note: "+spn);
                    }
                    if(spn != -25)
                    {
                       //send the message in here.
                       try
                       {
                          KeyValueList nMess = new KeyValueList();
                          nMess.putPair("Scope", SCOPE);
                          nMess.putPair("MessageType", "Alert");
                          nMess.putPair("Sender", NAME);
                          nMess.putPair("Receiver", "NoteProc");
                          nMess.putPair("Note", Integer.toString(spn));
                          encoder.sendMsg(nMess);
                       }
                       catch (Exception e)
                       {
                          System.out.println("Message send failed. :(");
                       }
                    }
                 }
              };
              AudioProcessor pitchEstimator = new PitchProcessor(PitchProcessor.PitchEstimationAlgorithm.FFT_YIN,sr,1024,printPitch);
              d.addAudioProcessor(pitchEstimator);
              //d.addAudioProcessor(new AudioPlayer(new AudioFormat(sr,16,1,true,true)));
              System.out.println("Bout to run.");
              d.run();
              System.out.println("It \"ends\" here.");
           }
                                        }

                                    } catch (Exception e) {
                                        // if anything goes wrong, try to re-establish the connection
                                        try {
                                            // wait for 1 second to retry
                                            Thread.sleep(1000);
                                        } catch (InterruptedException e2) {
                                        }
                                        System.out.println("Try to reconnect");
                                        try {
                                            universal = connect();
                                        } catch (IOException e1) {
                                        }
                                    }
                                }
                            }
                        }

    public static int getSPN(float p)
    {
        double[] freqs2 = {65.406,69.296,73.416,77.782,82.407,87.307,92.499,97.999,103.83,110,116.54,123.47};//every other one is a multiple of these it
doubles every octave
        double[] freqs3 = new double[12];
        double[] freqs4 = new double[12];
        double[] freqs5 = new double[12];
        double[] freqs6 = new double[12];
        double[] freqs7 = new double[12];
        for(int i = 0; i < 12; i++)
        {
           freqs3[i] = freqs2[i]*2;
           freqs4[i] = freqs2[i]*4;
           freqs5[i] = freqs2[i]*8;
           freqs6[i] = freqs2[i]*16;
           freqs7[i] = freqs2[i]*32;
        }
        double[][] freqs = {freqs2,freqs3,freqs4,freqs5,freqs6,freqs7};

        //for(int i = 0; i < 12; i++)
        //{
```

```
//    System.out.println(freqs2[i]+" : "+freqs3[i]+" : "+freqs4[i]+" : "+freqs5[i]+" : "+freqs6[i]+" : "+freqs7[i]);
//}

double distance = Math.abs(p - freqs[0][0]);
int note = -24;
for(int i = 0; i < 12; i++)
{
   for(int j = 0; j < 12; j++)
   {
      if(Math.abs(p-freqs[i][j]) <= distance)
      {
         distance = Math.abs(p-freqs[i][j]);
         note = -24 + i*12 + j;
      }
      else
      {
         return note;
      }
   }
}
return note;

}

        /*
         * used for connect(reconnect) to SISServer
         */

        static Socket connect() throws IOException {
                Socket socket = new Socket("127.0.0.1", port);
                return socket;
        }

        /*
         * process a certain message, execute corresponding actions
         */
        static void ProcessMsg(KeyValueList kvList) throws IOException {
   System.out.println("Recieved a message from " + kvList.getValue("Sender"));
        }

}
```

CreateNoteProc.java:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.StringTokenizer;
import java.util.*;
import java.io.*;
import java.net.*;
import java.lang.*;
import javax.mail.Authenticator;
import javax.mail.Message;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

public class CreateNoteProc {

        // socket for connection to SISServer
        private static Socket universal;
        private static int port = 53217;
        // message writer
        private static MsgEncoder encoder;
        // message reader
```

```
	private static MsgDecoder decoder;

	// scope of this component
	private static final String SCOPE = "SIS.Scope1";
	// name of this component
	private static final String NAME = "NoteProc";
	// messages types that can be handled by this component
	private static final List<String> TYPES = new ArrayList<String>(
				Arrays.asList(new String[] { "Setting", "Alert", "Confirm" }));

	// summary for all incoming / outgoing messages
	private static final String incomingMessages = "IN\t Confirm|Setting|Reading|Alert";
	private static final String outgoingMessages = "OUT\t Connect|Alert";

	// shared by all kinds of emergencies that can be generated by this component
	private static KeyValueList emergency = new KeyValueList();

	//static String alert_msg = "Patient has abnormal bloodPressure because of uncomfortable temperature";
	//static String doctorEmail = "sisfortest@outlook.com"; // default email
	static ArrayList<Double> tempRecord = new ArrayList<Double>();

static HashMap<Integer, Integer> noteFreqs;
static String vocType = "";

	/*
	 * Main program
	 */
	public static void main(String[] args) {
  noteFreqs = new HashMap<Integer, Integer>();
			while (true) {
				try {
					// try to establish a connection to SISServer
					universal = connect();

					// bind the message reader to inputstream of the socket
					decoder = new MsgDecoder(universal.getInputStream());
					// bind the message writer to outputstream of the socket
					encoder = new MsgEncoder(universal.getOutputStream());

					/*
					 * construct a Connect message to establish the connection
					 */
					KeyValueList conn = new KeyValueList();
					conn.putPair("Scope", SCOPE);
					conn.putPair("MessageType", "Connect");
					conn.putPair("IncomingMessages", incomingMessages);
	conn.putPair("OutgoingMessages", outgoingMessages);
					conn.putPair("Role", "Basic");
					conn.putPair("Name", NAME);
					encoder.sendMsg(conn);

					// KeyValueList for inward messages, see KeyValueList for
					// details
					KeyValueList kvList;
	System.out.println("Getting there.");
					while (true) {
						// attempt to read and decode a message, see MsgDecoder for
						// details
						kvList = decoder.getMsg();

						// process that message
						ProcessMsg(kvList); //this is just a stub, it should only connect.
					}

				} catch (Exception e) {
					// if anything goes wrong, try to re-establish the connection
					try {
						// wait for 1 second to retry
						Thread.sleep(1000);
					} catch (InterruptedException e2) {
					}
					System.out.println("Try to reconnect");
					try {
```

```
                                        universal = connect();
                                    } catch (IOException e1) {
                                    }
                                }
                        }
                }

                /*
                 * used for connect(reconnect) to SISServer
                 */
                static Socket connect() throws IOException {
                        Socket socket = new Socket("127.0.0.1", port);
                        return socket;
                }

                /*
                 * process a certain message, execute corresponding actions
                 */
                static void ProcessMsg(KeyValueList kvList) throws IOException {
System.out.println("Recieved a message from " + kvList.getValue("Sender"));

String sender = kvList.getValue("Sender");
                        String receiver = kvList.getValue("Receiver");
                        String purpose = kvList.getValue("Purpose");
String type = kvList.getValue("MessageType");

if(sender.equals("MicInput"))
{
   //MicInput will result in updating local data and possibly sending mesage to GUI
   if(type.equals("Alert"))
   {
      int note = Integer.parseInt(kvList.getValue("Note"));
      if(noteFreqs.containsKey(note))
      {
         int freq = noteFreqs.get(note);
         freq++;
         noteFreqs.put(note, freq);
      }
      else
      {
         noteFreqs.put(note,1);
      }
      int total = 0;
      for(int value : noteFreqs.values())
      {
         total=total+value;
      }
      if(total>24)
      {
         /*int sop = 0;
         int mez = 0;
         int cont = 0;
         int cten = 0;
         int ten = 0;
         int bari = 0;
         int bass = 0;
         for(int key : noteFreqs.keySet())
         {
            int freq = noteFreqs.get(key);
            if(key >=0 && c<= 24) sop = sop+freq; //Soprano
            if(key >=-3 && c<= 21) mez = mez+freq; //Mezzo-soprano
            if(key >=-7 && c<= 17) cont = cont+freq; //Contralto
            if(key >=-8 && c<= 16) cten = cten+freq; //Countertenor
            if(key >=-12 && c<= 12) ten = ten+freq; //Tenor
            if(key >=-17 && c<= 7) bari = bari+freq; //Baritone
            if(key >=-20 && c<= 4) bass = bass+freq; //Bass
         }*/

         Set<Integer> keys = noteFreqs.keySet();
         Integer[] keyArray = keys.toArray(new Integer[keys.size()]);
         Arrays.sort(keyArray);
         int high = keyArray[keyArray.length -1];
         int low = keyArray[0];
```

```
            int highfreq = 0;
            int lowfreq = 0;

            Collection<Integer> values = noteFreqs.values();
            int avg = 0;
            for(int i : values)
            {
               avg = avg + i;
            }
            avg = avg/values.size();
            System.out.println(avg);
            for(int i = 0; i <= keyArray.length/2; i++)
            {
               int freq = noteFreqs.get(keyArray[i]);
               if(freq < avg/2)
               {
                  ;
               }
               else
               {
                  low = keyArray[i];
                  lowfreq = freq;
                  break;
               }
            }
            for(int i = keyArray.length-1; i >= keyArray.length/2; i--)
            {
               int freq = noteFreqs.get(keyArray[i]);
               if(freq < avg/2)
               {
                  ;
               }
               else
               {
                  high = keyArray[i];
                  highfreq = freq;
                  break;
               }
            }
            KeyValueList mess = new KeyValueList();
            mess.putPair("Scope", SCOPE);
            mess.putPair("MessageType", "Alert");
            mess.putPair("Sender", NAME);
            mess.putPair("Receiver", "GUI");
            mess.putPair("Text", ""+low+"/"+high);
            encoder.sendMsg(mess);
         }
         else if(total> 12)
         {
            KeyValueList mess = new KeyValueList();
            mess.putPair("Scope", SCOPE);
            mess.putPair("MessageType", "Alert");
            mess.putPair("Sender", NAME);
            mess.putPair("Receiver", "GUI");
            mess.putPair("Text", "About halfway there.");
            encoder.sendMsg(mess);
         }
         else
         {
            KeyValueList mess = new KeyValueList();
            mess.putPair("Scope", SCOPE);
            mess.putPair("MessageType", "Alert");
            mess.putPair("Sender", NAME);
            mess.putPair("Receiver", "GUI");
            mess.putPair("Text", "Not sufficient data points.");
            encoder.sendMsg(mess);
         }

      }
   }
   else if(sender.equals("GUI"))
   {
      //GUI message should only be Setting message to reset data.
```

```
        if(type.equals("Setting"))
        {
           if(purpose.equals("Clear"))
           {
              //we just reset the hashmap
              noteFreqs = new HashMap<Integer, Integer>();
              KeyValueList mess = new KeyValueList();
              mess.putPair("Scope", SCOPE);
              mess.putPair("MessageType", "Alert");
              mess.putPair("Sender", NAME);
              mess.putPair("Receiver", "GUI");
              mess.putPair("Text", "No existing data.");
              encoder.sendMsg(mess);
           }
        }
     }
            }

}
```

CreateGUI.java:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.StringTokenizer;
import java.util.*;
import java.io.*;
import java.net.*;
import java.lang.*;
import javax.mail.Authenticator;
import javax.mail.Message;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import javax.sound.midi.*;

public class CreateGUI extends JFrame {

            // socket for connection to SISServer
            private static Socket universal;
            private static int port = 53217;
            // message writer
            private static MsgEncoder encoder;
            // message reader
            private static MsgDecoder decoder;

            // scope of this component
            private static final String SCOPE = "SIS.Scope1";
            // name of this component
            private static final String NAME = "GUI";

            // shared by all kinds of emergencies that can be generated by this component
            private static KeyValueList emergency = new KeyValueList();

   // summary for all incoming / outgoing messages
            private static final String incomingMessages = "IN\t Confirm|Setting|Reading";
            private static final String outgoingMessages = "OUT\t Connect|Reading|Setting";

            //static String alert_msg = "Patient has abnormal bloodPressure because of uncomfortable temperature";
            //static String doctorEmail = "sisfortest@outlook.com"; // default email
            static ArrayList<Double> tempRecord = new ArrayList<Double>();
   private JLabel header;
   private JLabel label1;
```

```
private JTextField status1;
private JButton clear;
private JButton play;
private JPanel controlPanel1;
private static JTextField status;

public CreateGUI()
{
   setTitle("GUI");
   setSize(315,200);
   setLocation(10,200);
   setLayout(new GridLayout(4,1));

   //Window Listeners
   addWindowListener(
      new WindowAdapter()
      {
         public void windowClosing(WindowEvent e)
         {
            System.exit(0);
         }
      }
   );

   //handle componants
   clear = new JButton("Clear Data.");
   clear.addActionListener
   (
      new ActionListener()
      {
         public void actionPerformed(ActionEvent e)
         {
            try {
               buttonSend();
            } catch (Exception j) {
               //
            }

         }
      }
   );
   play = new JButton("Play range.");
   play.addActionListener
   (
      new ActionListener()
      {
         public void actionPerformed(ActionEvent e)
         {
            try
            {
               String notes = status.getText();
               System.out.println(notes);
               if(notes.indexOf("/") == -1)
               {
                  System.out.println("Range not available.");
               }
               else
               {
                  String[] notesarray = notes.split("/");
                  int low = Integer.parseInt(notesarray[0]);
                  int high = Integer.parseInt(notesarray[1]);
                  Synthesizer syn = MidiSystem.getSynthesizer();
                  syn.open();
                  final MidiChannel[] mc = syn.getChannels();
                  Instrument[] instr = syn.getDefaultSoundbank().getInstruments();
                  syn.loadInstrument(instr[90]);
                  mc[0].noteOn(low+48,150); //scientific pitch notation starts at -48, midi starts at 0.
                  Thread.sleep(1500);
                  mc[0].noteOn(high+48,150);
               }
            } catch (Exception j)
            {
               //
```

```
            }

          }
        }
    );
    controlPanel1 = new JPanel();
    controlPanel1.setLayout(new FlowLayout());
    header = new JLabel("Find your range by singing into the mic.", JLabel.LEFT);
    label1 = new JLabel("Your high/low notes are: ", JLabel.LEFT);
    status1 = new JTextField();
    status1.setText("No existing data.");
    status=status1;
    status1.setEditable(false);
    add(header);
    controlPanel1.add(label1);
    controlPanel1.add(status1);
    add(controlPanel1);
    add(clear);
    add(play);

}

public void buttonSend() throws IOException
{
    //this will eventually result in sending a message.
    KeyValueList conn = new KeyValueList();
    conn.putPair("Scope", SCOPE);
    conn.putPair("MessageType", "Setting");
    conn.putPair("Sender", NAME);
    conn.putPair("Receiver", "NoteProc");
    conn.putPair("Purpose", "Clear");
    encoder.sendMsg(conn);
}

          /*
           * Main program
           */
          public static void main(String[] args)
{
    JFrame frame = new CreateGUI();
    frame.show();
                    while (true) {
                            try {
                                    // try to establish a connection to SISServer
                                    universal = connect();

                                    // bind the message reader to inputstream of the socket
                                    decoder = new MsgDecoder(universal.getInputStream());
                                    // bind the message writer to outputstream of the socket
                                    encoder = new MsgEncoder(universal.getOutputStream());

                                    /*
                                     * construct a Connect message to establish the connection
                                     */
                                    KeyValueList conn = new KeyValueList();
                                    conn.putPair("Scope", SCOPE);
                                    conn.putPair("MessageType", "Connect");
                                    conn.putPair("Role", "Monitor");
                                    conn.putPair("Name", NAME);
                                    encoder.sendMsg(conn);

                                    // KeyValueList for inward messages, see KeyValueList for
                                    // details
                                    KeyValueList kvList;

                                    while (true) {
                                            // attempt to read and decode a message, see MsgDecoder for
                                            // details
                                            kvList = decoder.getMsg();

                                            // process that message
                                            ProcessMsg(kvList, frame);
                                    }
```

```
                                        } catch (Exception e) {
                                                // if anything goes wrong, try to re-establish the connection
                                                try {
                                                        // wait for 1 second to retry
                                                        Thread.sleep(1000);
                                                } catch (InterruptedException e2) {
                                                }
                                                System.out.println("Try to reconnect");
                                                try {
                                                        universal = connect();
                                                } catch (IOException e1) {
                                                }
                                        }
                                }
                        }

                        /*
                         * used for connect(reconnect) to SISServer
                         */
                        static Socket connect() throws IOException {
                                Socket socket = new Socket("127.0.0.1", port);
                                return socket;
                        }

                        /*
                         * process a certain message, execute corresponding actions
                         */
                        static void ProcessMsg(KeyValueList kvList, JFrame frame) throws IOException {
                System.out.println("Recieved a message from " + kvList.getValue("Sender"));

                //actually do things with the message from here.
                String sender = kvList.getValue("Sender");
                                String receiver = kvList.getValue("Receiver");
                                String purpose = kvList.getValue("Purpose");
                String type = kvList.getValue("MessageType");
                if(sender.equals("NoteProc"))
                {
                   //NoteProc is the only person whose messagse we care about doing something with.
                   if(type.equals("Alert"))
                   {
                      //we only recieve Readings from NoteProc that we care about.
                      //System.out.println(kvList.getValue("Text"));
                      status.setText(kvList.getValue("Text"));
                   };
                }
                        }

}
```