```
 1
 2  /* ================================================================
    ================== 
 3   * Copyright (c) 2014 Alpha Cephei Inc.  All rights reserved.
 4   *
 5   * Redistribution and use in source and binary forms, with or without
 6   * modification, are permitted provided that the following conditions
 7   * are met:
 8   *
 9   * 1. Redistributions of source code must retain the above copyright
10   *    notice, this list of conditions and the following disclaimer.
11   *
12   * 2. Redistributions in binary form must reproduce the above copyright
13   *    notice, this list of conditions and the following disclaimer in
14   *    the documentation and/or other materials provided with the
15   *    distribution.
16   *
17   * THIS SOFTWARE IS PROVIDED BY ALPHA CEPHEI INC. ``AS IS'' AND
18   * ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
    LIMITED TO,
19   * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
    PARTICULAR
20   * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL CARNEGIE MELLON
    UNIVERSITY
21   * NOR ITS EMPLOYEES BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL
    ,
22   * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
    NOT
23   * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
     OF USE,
24   * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
    AND ON ANY
25   * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
    TORT
26   * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF
    THE USE
27   * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
    DAMAGE.
28   *
29
     * ================================================================
    ================== 
30   */
31
32  package course.examples.intelligentrobot.activities;
33
34  import android.content.Context;
35  import android.net.nsd.NsdManager;
36  import android.net.nsd.NsdServiceInfo;
37  import android.os.AsyncTask;
38  import android.os.Bundle;
39  import android.os.Handler;
40  import android.util.Log;
41  import android.view.View;
42  import android.widget.Button;
43  import android.widget.TextView;
44
45  import java.io.BufferedReader;
```

```
46  import java.io.File;
47  import java.io.FileOutputStream;
48  import java.io.IOException;
49  import java.io.InputStream;
50  import java.io.InputStreamReader;
51  import java.io.OutputStream;
52  import java.io.PrintWriter;
53  import java.net.InetAddress;
54  import java.net.Socket;
55
56  import course.examples.intelligentrobot.R;
57  import course.examples.intelligentrobot.helperClasses.decisionMaker;
58  import edu.cmu.pocketsphinx.Assets;
59  import edu.cmu.pocketsphinx.Hypothesis;
60  import edu.cmu.pocketsphinx.RecognitionListener;
61  import edu.cmu.pocketsphinx.SpeechRecognizer;
62
63  import static edu.cmu.pocketsphinx.SpeechRecognizerSetup.defaultSetup;
64
65  public class SpeechRecognitionRemote extends LifecycleLoggingActivity
    implements
66          RecognitionListener {
67      private String TAG = SpeechRecognitionRemote.class.getSimpleName();
68
69      //Networking client fields
70      private NsdManager mNsdManager = null;
71      private NsdManager.DiscoveryListener mDiscoveryListener;
72      private NsdManager.ResolveListener mResolveListener = null;
73      private NsdServiceInfo mService;
74      private Socket mSocket = null;
75      private String mServiceName = "SmartBot";
76      private PrintWriter mOutput = null;
77      private android.os.Handler mConnectionHandler =  new android.os.Handler();
78      private final int CONNECTION_INTERVAL = 250; // msec
79      private Runnable mConnectionCheck = null;
80      private Object mConnectionLock = new Object();
81      private boolean mConnectionDone = false;
82
83      private Thread clientCommsThread = null;
84      private boolean serverForcedClose = false;
85
86      //General fields
87      private Button recordButton;
88      private Button button_refresh;
89      private TextView speechTextView;
90      private static final String grammarSearch = "digits";
91      private static final String navSearch = "nav";
92      private decisionMaker responseCalculator;
93      private TextView t_ConnectionStatus;
94      private TextView t_leftEnc_Text;
95      private TextView t_rightEnc_Text;
96      private android.os.Handler m_StatusTextHandler = new android.os.Handler();
97      private android.os.Handler m_LeftEncTextHandler = new android.os.Handler();
98      private android.os.Handler m_RightEncTextHandler = new android.os.Handler();
99      private boolean networkReady = false;
100     private boolean speechRecognizerReady = false;
101     private android.os.Handler UIHandler = new Handler();
102     private FileOutputStream outputStream = null;
103     private String mode = "talk";
```

```
104
105
106
107     //Sphinx fields
108     private SpeechRecognizer recognizer;
109
110     @Override
111     public void onCreate(Bundle state) {
112        super.onCreate(state);
113
114        setupUI();
115        setupDecisionMaker();
116        setupSphinx();
117
118     }
119
120     @Override
121     public void onDestroy() {
122        super.onDestroy();
123        if(recognizer!= null)
124        {
125           recognizer.cancel();
126           recognizer.shutdown();
127
128        }
129     }
130
131     /**
132      * In partial result we get quick updates about current hypothesis. In
133      * keyword spotting mode we can react here, in other modes we need to wait
134      * for final result in onResult.
135      */
136     @Override
137     public void onPartialResult(Hypothesis hypothesis) {
138        if (hypothesis == null)
139           return;
140
141        String text = hypothesis.getHypstr();
142        speechTextView.setText(text);
143
144        /*
145        //TODO ???
146        if (text.equals(KEYPHRASE))
147           switchSearch(MENU_SEARCH);
148        else if (text.equals(DIGITS_SEARCH))
149           switchSearch(DIGITS_SEARCH);
150        else if (text.equals(PHONE_SEARCH))
151           switchSearch(PHONE_SEARCH);
152        else if (text.equals(FORECAST_SEARCH))
153           switchSearch(FORECAST_SEARCH);
154        //else
155          // ((TextView) findViewById(R.id.result_text)).setText(text);
156           */
157     }
158
159     /**
160      * This callback is called when we stop the recognizer.
161      */
162     @Override
```

```
163     public void onResult(Hypothesis hypothesis) {
164
165         String speakString = "null";
166         if (hypothesis != null) {
167             String text = hypothesis.getHypstr();
168             speechTextView.setText(text);
169
170             text = text.toLowerCase();
171             Log.d(TAG, "onResult(): " + text);
172
173             if(!mode.equals("talk")) {
174                 //TODO make right/left commands reality...
175                 Log.d(TAG, "onResult() inside navigation: " + text);
176
177                     if(text.contains("right"))
178                     {
179                         send_R(-400);
180                         send_L(400);
181                         Log.d(TAG, "onResult() commanded right");
182                     }
183                     else if(text.contains("left")) {
184                         send_R(400);
185                         send_L(-400);
186                         Log.d(TAG, "onResult() commanded left");
187                     }
188                     else if(text.contains("forward"))
189                     {
190                         send_R(400);
191                         send_L(400);
192                     }
193                     else if(text.contains("back")) {
194
195                         send_R(-400);
196                         send_L(-400);
197                     }
198                     else if(text.contains("picture")) {
199
200                         //todo
201                     }
202                     else if(text.contains("stop")) {
203                         send_R(0);
204                         send_L(0);
205                     }
206                 switchSearch(navSearch);
207             }
208             else
209             {
210                 speakString = responseCalculator.getStringTTS(text);
211                 sendTTS(speakString);
212             }
213         }
214
215         //speechTextView.setText("Response: " + speakString);
216
217     }
218
219     @Override
220     public void onBeginningOfSpeech() {
221     }
```

```
222
223     /**
224      * We stop recognizer here to get a final result
225      */
226     @Override
227     public void onEndOfSpeech() {
228         Log.d(TAG, "onEndOfSpeech");
229         recognizer.stop();
230         if(mode.equals("talk"))
231         {
232             recordButton.setEnabled(true);
233         }
234         //speechTextView.setText();
235     }
236
237     private void switchSearch(String searchName) {
238
239         recognizer.stop();
240         recognizer.startListening(searchName);
241
242         // If we are not spotting, start listening with timeout (10000 ms or 10 seconds).
243       /*  if (searchName.equals(KWS_SEARCH))
244             recognizer.startListening(searchName);
245         else
246             recognizer.startListening(searchName, 10000);
247             */
248
249         //String caption = getResources().getString(captions.get(searchName));
250         //((TextView) findViewById(R.id.caption_text)).setText(caption);
251     }
252
253     private void setupRecognizer(File assetsDir) throws IOException {
254         // The recognizer can be configured to perform multiple searches
255         // of different kind and switch between them
256
257         recognizer = defaultSetup()
258                 .setAcousticModel(new File(assetsDir, "en-us-ptm"))
259                 .setDictionary(new File(assetsDir, "robotV3.dic"))
260
261                         // To disable logging of raw audio comment out this call (takes a lot of
    space on the device)
262                 .setRawLogDir(assetsDir)
263
264                         // Threshold to tune for keyphrase to balance between false alarms and
     misses
265                 .setKeywordThreshold(1e-45f)
266
267                         // Use context-independent phonetic search, context-dependent is too
    slow for mobile
268                 .setBoolean("-allphone_ci", true)
269
270                 .getRecognizer();
271         recognizer.addListener(this);
272
273         /** In your application you might not need to add all those searches.
274          * They are added here for demonstration. You can leave just one.
275          */
276
277
```

```
278         // Create grammar-based search for selection between demos
279         File languageModel = new File(assetsDir, "robotV4.dmp");
280         recognizer.addNgramSearch(grammarSearch, languageModel);
281 
282         // Create grammar-based search for selection between demos
283         File navigationModel = new File(assetsDir, "nav.dmp");
284         recognizer.addNgramSearch(navSearch, navigationModel);
285 
286     }
287 
288     @Override
289     public void onError(Exception error) {
290         //((TextView) findViewById(R.id.caption_text)).setText(error.getMessage());
291     }
292 
293     @Override
294     public void onTimeout() {
295         //switchSearch(KWS_SEARCH);
296     }
297 
298     /*
299         GUI record pushed
300      */
301     public void recordPushed(View v)
302     {
303         if(mode.equals("nav"))
304         {
305             recordButton.setText("Stop navigation mode");
306             mode = "nav1";
307             switchSearch(navSearch);
308         }
309         else if(mode.equals("nav1"))
310         {
311             send_L(0);
312             send_R(0);
313             recordButton.setEnabled(false);
314             recordButton.setText("Record");
315             mode = "talk";
316             sendTTS(responseCalculator.getStringTTS(null));
317             UIHandler.postDelayed(new Runnable(){
318 
319                 @Override
320                 public void run()
321                 {
322                     recordButton.setEnabled(true);
323                 }
324             },500);
325 
326         }
327         else
328         {
329             recordButton.setEnabled(false);
330             switchSearch(grammarSearch);
331         }
332     }
333 
334     public void sendTTS(String speakString)
335     {
336         //send message w/ $ as first character
```

```
337
338         if(speakString.charAt(0) == '1')
339         {
340             //TODO SWTICH TO NAV STATE
341             recordButton.setText("Start navigation mode");
342             mode = "nav";
343             speakString = speakString.substring(1);
344         }
345
346         if(mOutput!=null){
347             mOutput.println("$"+speakString);
348             mOutput.flush();
349         }
350     }
351
352
353
354     private void setupSphinx()
355     {
356         // Recognizer initialization is a time-consuming and it involves IO,
357         // so we execute it in async task
358         new AsyncTask<Void, Void, Exception>() {
359             @Override
360             protected Exception doInBackground(Void... params) {
361                 try {
362                     Assets assets = new Assets(SpeechRecognitionRemote.this);
363                     File assetDir = assets.syncAssets();
364                     setupRecognizer(assetDir);
365                 } catch (IOException e) {
366                     return e;
367                 }
368                 return null;
369             }
370
371             @Override
372             protected void onPostExecute(Exception result) {
373                 if (result != null) {
374                     speechTextView.setText("Failed to init recognizer " + result);
375                 } else {
376                     speechRecognizerReady = true;
377                     if(speechRecognizerReady && networkReady)
378                     {
379                         UIHandler.post(new Runnable() {
380                             @Override
381                             public void run()
382                             {
383                                 sendTTS(responseCalculator.getStringTTS(null));
384                                 recordButton.setEnabled(true);
385                                 speechRecognizerReady = false;
386                             }
387                         });
388                     }
389                 }
390             }
391         }.execute();
392     }
393
394     private void setupUI()
395     {
```

```
396         setContentView(R.layout.activity_speech_recognition);
397         recordButton = (Button) findViewById(R.id.button);
398         speechTextView = (TextView) findViewById(R.id.textView);
399         button_refresh = (Button) findViewById(R.id.refresh);
400         button_refresh.setOnClickListener(new View.OnClickListener() {
401             public void onClick(View v) {
402                 send_EOS();
403                 tearDown();
404                 initialiseClient();
405             }
406         });
407 
408         t_ConnectionStatus = (TextView) findViewById(R.id.textView_status);
409         t_leftEnc_Text = (TextView) findViewById(R.id.textView_leftEnc);
410         t_rightEnc_Text = (TextView) findViewById(R.id.textView_rightEnc);
411     }
412 
413     @Override
414     protected void onStart(){
415         initialiseClient();
416         super.onStart();
417     }
418 
419     @Override
420     protected void onStop() {
421         tearDown();
422         super.onStop();
423     }
424 
425     private void initialiseClient(){
426         Log.d(TAG, "initialiseClient()");
427         initializeConnectionCheck();
428         mNsdManager = (NsdManager) this.getSystemService(Context.
    NSD_SERVICE);
429         initializeDiscoveryListener();
430         mNsdManager.discoverServices(
431                 "_http._udp.", NsdManager.PROTOCOL_DNS_SD,
    mDiscoveryListener);
432     }
433 
434     private void tearDown(){
435         Log.d(TAG, "tearDown()");
436         cancelConnectionCheck();
437 
438         if(clientCommsThread!=null){
439             if(clientCommsThread.isInterrupted()){
440                 // server called End of stream
441             }
442             else{
443                 this.clientCommsThread.interrupt();
444             }
445             clientCommsThread = null;
446         }
447         if (mResolveListener != null) {
448             mResolveListener = null;
449         }
450         if (mDiscoveryListener != null){
451             mNsdManager.stopServiceDiscovery(mDiscoveryListener);
452             mDiscoveryListener = null;
```

```
453         }
454     }
455
456     public void initializeDiscoveryListener() {
457         Log.d(TAG, "initialiseDiscoveryListener()");
458         // Instantiate a new DiscoveryListener
459         mDiscoveryListener = new NsdManager.DiscoveryListener() {
460
461             //  Called as soon as service discovery begins.
462             @Override
463             public void onDiscoveryStarted(String regType) {
464                 Log.d(TAG, "Service discovery started");
465             }
466
467             @Override
468             public void onServiceFound(NsdServiceInfo service) {
469                 // A service was found!  Do something with it.
470                 Log.d(TAG, "Service discovery success" + service);
471                 if (!service.getServiceType().equals("_http._udp.")) {
472                     // Service type is the string containing the protocol and
473                     // transport layer for this service.
474                     Log.d(TAG, "Unknown Service Type: " + service.getServiceType());
475                     setStatusText("Unknown Service Type: " + service.getServiceType());
476                 } else if (service.getServiceName().contains(mServiceName)){
477                     initializeResolveListener();
478                     mNsdManager.resolveService(service, mResolveListener);
479                     setStatusText("Resolving service: " + service.getServiceType());
480                 }
481             }
482
483             @Override
484             public void onServiceLost(NsdServiceInfo service) {
485                 // When the network service is no longer available.
486                 // Internal bookkeeping code goes here.
487                 Log.d(TAG, "onServiceLost()");
488                 Log.e(TAG, "Service lost" + service);
489                 setStatusText("Service lost: " + service);
490                 tearDown();
491                 initialiseClient();
492             }
493
494             @Override
495             public void onDiscoveryStopped(String serviceType) {
496                 Log.i(TAG, "Discovery stopped: " + serviceType);
497                 Log.d(TAG, "onDiscoveryStopped()");
498
499             }
500
501             @Override
502             public void onStartDiscoveryFailed(String serviceType, int errorCode) {
503                 Log.e(TAG, "Discovery failed: Error code:" + errorCode);
504                 mNsdManager.stopServiceDiscovery(this);
505                 setStatusText("Discovery failed!");
506             }
507
508             @Override
509             public void onStopDiscoveryFailed(String serviceType, int errorCode) {
510                 Log.e(TAG, "Discovery failed: Error code:" + errorCode);
511                 mNsdManager.stopServiceDiscovery(this);
```

```
512             }
513         };
514     }
515
516     public void initializeResolveListener() {
517         Log.d(TAG, "initializeResolveListener()");
518         mResolveListener = new NsdManager.ResolveListener() {
519
520             @Override
521             public void onResolveFailed(NsdServiceInfo serviceInfo, int errorCode) {
522                 // Called when the resolve fails.  Use the error code to debug.
523                 Log.e(TAG, "Resolve failed" + errorCode);
524                 setStatusText("Resolve failed");
525             }
526
527             @Override
528             public void onServiceResolved(NsdServiceInfo serviceInfo) {
529                 Log.e(TAG, "Resolve Succeeded. " + serviceInfo);
530                 if (serviceInfo.getServiceName().contains(mServiceName)) {
531                     mService = serviceInfo;
532                     int port = mService.getPort();
533                     InetAddress host = mService.getHost();
534                     try {
535                         mSocket = new Socket(host, port);
536                         clientCommsThread = new Thread(new ClientCommsThread(mSocket
    ));
537                         clientCommsThread.start();
538                         setStatusText("Resolved:" + mService.getServiceName() +
539                                 "\n Host:" + host.toString() + " Port:" + Integer.toString(port));
540                         mConnectionCheck.run();
541
542
543                     } catch (IOException e) {
544                         e.printStackTrace();
545                         mSocket = null;
546                     }
547                 }
548                 return;
549             }
550         };
551     }
552
553     class ClientCommsThread implements Runnable {
554         private Socket clientSocket;
555         private BufferedReader input;
556
557         public ClientCommsThread(Socket clientSocket){
558             Log.d(TAG, "ClientCommsThread()");
559             this.clientSocket = clientSocket;
560             try {
561                 this.input = new BufferedReader(new InputStreamReader(this.clientSocket
    .getInputStream()));
562                 OutputStream out = this.clientSocket.getOutputStream();
563                 mOutput = new PrintWriter(out);
564
565                 networkReady = true;
566                 if(speechRecognizerReady && networkReady)
567                 {
568                     //when ready to record...
```

```
569                 UIHandler.post(new Runnable() {
570                     @Override
571                     public void run()
572                     {
573                         sendTTS(responseCalculator.getStringTTS(null));
574                         recordButton.setEnabled(true);
575                         speechRecognizerReady = false;
576                     }
577                 });
578             }
579         } catch (IOException e) {
580             e.printStackTrace();
581         }
582     }
583     @Override
584     public void run(){
585         Log.d(TAG, "ClientCommsThread run()");
586         while (!Thread.currentThread().isInterrupted()) {
587             try {
588                 if(input.ready()) {
589                     String read = input.readLine();
590                     if(read.compareTo("EOS") == 0 || read==null ){
591                         // end of stream
592                         setStatusText("Server disconnected!");
593                         serverForcedClose = true;
594                         Thread.currentThread().interrupt();
595                     }
596                     else if(read != null) {
597                         receivedCommand(read);
598                     }
599                 }
600             }
601             catch (Exception e){
602                 e.printStackTrace();
603             }
604         }
605
606         UIHandler.post(new Runnable() {
607             @Override
608             public void run()
609             {
610                 recordButton.setEnabled(false);
611             }
612         });
613
614         if(!serverForcedClose){
615             //client side close
616             //client sends EOS
617             send_EOS();
618         }
619         // close socket
620         try {
621             this.clientSocket.shutdownInput();
622             this.clientSocket.shutdownOutput();
623             this.clientSocket.close();
624         } catch (IOException e) {
625             e.printStackTrace();
626         }
627
```

```
628             if(serverForcedClose) {
629                 //close
630                 tearDown();
631                 // reopen
632                 initialiseClient();
633                 serverForcedClose = false;
634             }
635         }
636     }
637
638     private void initializeConnectionCheck(){
639         synchronized (mConnectionLock) {
640             mConnectionDone = false;
641         }
642         mConnectionCheck =
643                 new Runnable() {
644                     @Override
645                     public void run() {
646                         send_C();
647                         synchronized (mConnectionLock) {
648                             if(!mConnectionDone)
649                                 mConnectionHandler.postDelayed(this,
    CONNECTION_INTERVAL);
650                         }
651                     }
652                 };
653     }
654
655     private void cancelConnectionCheck(){
656         synchronized (mConnectionLock) {
657             mConnectionDone = true;
658         }
659     }
660
661     public void receivedCommand(String msg){
662         final String cmd = msg;
663         if (msg.contains("R")) {
664             m_RightEncTextHandler.post(new Runnable() {
665                 @Override
666                 public void run() {
667                     t_rightEnc_Text.setText(cmd);
668                     t_rightEnc_Text.invalidate();
669                 }
670             });
671         }
672         if (msg.contains("L")) {
673             m_LeftEncTextHandler.post(new Runnable() {
674                 @Override
675                 public void run() {
676                     t_leftEnc_Text.setText(cmd);
677                     t_leftEnc_Text.invalidate();
678                 }
679             });
680         }
681     }
682
683     public void send_EOS(){
684         if(mOutput!=null){
685             mOutput.println("EOS");
```

```
686             mOutput.flush();
687         }
688     }
689
690     public void send_L(int spd){
691         if(mOutput!=null){
692             String cmd = "L" + String.valueOf(spd);
693             mOutput.println(cmd);
694             mOutput.flush();
695             Log.d(TAG, "send_L() wrote");
696
697         }
698     }
699
700     public void send_R(int spd){
701         if(mOutput!=null){
702             String cmd = "R" + String.valueOf(spd);
703             mOutput.println(cmd);
704             mOutput.flush();
705             Log.d(TAG, "send_R() wrote");
706
707         }
708     }
709
710     public void send_C(){
711         if(mOutput!=null){
712             String cmd = "C";
713             mOutput.println(cmd);
714             mOutput.flush();
715         }
716     }
717
718     public void setStatusText(String txt){
719         final String msg = txt;
720         m_StatusTextHandler.post(new Runnable() {
721             @Override
722             public void run() {
723                 t_ConnectionStatus.setText(msg);
724             }
725         });
726     }
727
728     public void setupDecisionMaker()
729     {
730
731         File direct = getFilesDir();
732         File file = new File(direct, "people3");
733         boolean deleted = file.delete();
734
735         // WRITE: Create/append a file in the Internal Storage
736         String fileName = "people4";
737         String content = "hello world";
738         String dir = getFilesDir().getAbsolutePath();
739         Log.d(TAG, "setupDecisionMaker(), files path: "+dir);
740
741         try {
742             outputStream = openFileOutput(fileName, Context.MODE_APPEND);
743             // outputStream.write(content.getBytes());
744             // outputStream.close();
```

```
745
746             Log.d(TAG, "setupDecisionMaker(), outputStream created");
747         } catch (Exception e) {
748             e.printStackTrace();
749         }
750
751         //READ:
752         BufferedReader bufferedReader = getBufferedReader(fileName);
753
754         //initialize decisionMaker
755         responseCalculator = new decisionMaker(outputStream, bufferedReader,
    fileName, this);
756
757     }
758
759     public BufferedReader getBufferedReader(String filename)
760     {
761         BufferedReader buffer = null;
762         try{
763             InputStream inputStream = openFileInput(filename);
764             InputStreamReader inputStreamReader = new InputStreamReader(inputStream
    );
765             buffer = new BufferedReader(inputStreamReader);
766          //   Log.d(TAG, "setupDecisionMaker(), inputStream created");
767
768             //inputStream.close();
769
770         }catch (Exception e){
771             e.printStackTrace();
772         }
773
774         return buffer;
775     }
776 }
```