Kinect Chapter 14. Speech Recognition

When OpenNI is compared to Microsoft's SDK for the Kinect, two 'drawbacks' are often mentioned – a lack of control over the Kinect's tilt motor, and being unable to access the four channel microphone array. I tackled the first issue in chapter 6 when I implemented a simple Java class and driver for the motor, accelerometer, and LED. This chapter and the next are about two different approaches to adding speech recognition and microphone capture to OpenNI.

This chapter focuses on speech recognition, using CloudGarden's TalkingJava SDK (http://www.cloudgarden.com/JSAPI/), recorded with the PC's microphone rather than the Kinect's. TalkingJava offers a full implementation of Sun's Java Speech API (JSAPI) for recognition and synthesis, and utilizes Microsoft's Speech API (SAPI) speech engines beneath Java. SAPI is a standard feature on Windows, and comes with a range of simple configuration and testing tools. In particular, I can improve speech recognition accuracy by training the engine to deal specifically with my voice and microphone setup. SAPI is also at the heart of Microsoft's SDK for the Kinect.

Chapter 15 shows how to directly record from the Kinect's microphone array with Java's Sound API. That chapter's 'trick' is to install audio support from Microsoft's SDK, which *can* co-exist with OpenNI. The SDK's audio driver lets Windows 7 treat the array as a multichannel recording device, making it accessible to Java.

1. Before Programming

Although JSAPI and TalkingJava have features for listing the capabilities of the PC's sound card, for selecting between audio lines and mixers, and for setting parameters such as a microphone's volume, it's generally easier to configure and test audio equipment using OS-level tools. For example, on my Windows 7 test machine, I couldn't record from one of the audio ports with Java, and was able to discover that the hardware was faulty via OS controls.

Windows 7 (and XP) offers three audio Control Panel controls: a "Sound" control, a "Speech Recognition" control, and a control specific to the sound card ("SigmaTel Audio" on my Windows 7 machine, "Realtek HS Sound Effect Manager" on my XP device). If you're not sure what sound card you're using, then you can find details under the "Sound, video, and game controllers" heading of the "Device Manager" control.

The sound card should be checked first to ensure that the microphone ports are working, switched on, and their volume isn't muted. Figure 1 shows part of the SigmaTel control panel.

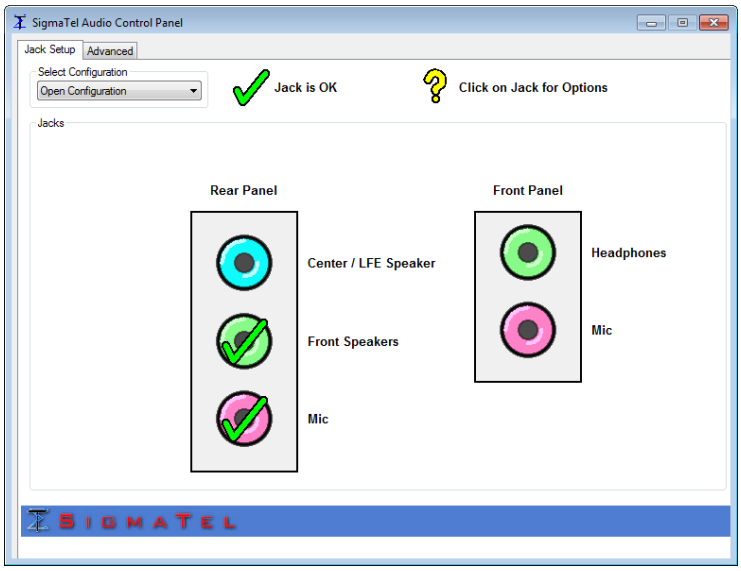

Figure 1. The SigmaTel Control Panel.

It was here that I discovered two problems with my sound card which Java can't detect: firstly that the front panel microphone wasn't working, and secondly that the rear panel microphone port would change into a speaker line if the microphone cable was removed!.

The "Sound" control on Windows 7 has a Recording tab (see Figure 2) and audio level bars for each input device. These might include different microphones, stereo "line in"s, headsets, or TV tuners.

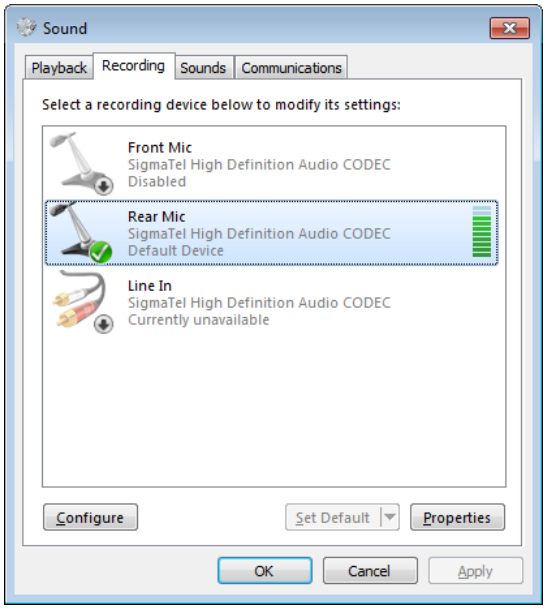

Figure 2. The Sound Control Panel on Windows 7.

The dark green level bars on the right of the rear microphone (Rear Mic) row in Figure 2 shows that it is picking up sound. If more than one microphone is active, then the ones you don't want should be disabled (by right clicking on them, and using

the menu that appears). Alternatively, make sure that the microphone you want is the default OS choice.

The Properties button in Figure 2 leads to a Properties window (see Figure 3) with tabs for setting the volume and boost, noise controls, and format settings (e.g. sample rate and number of channels on the Advanced tab). Make a note of these formats since they might be useful in your Java code.

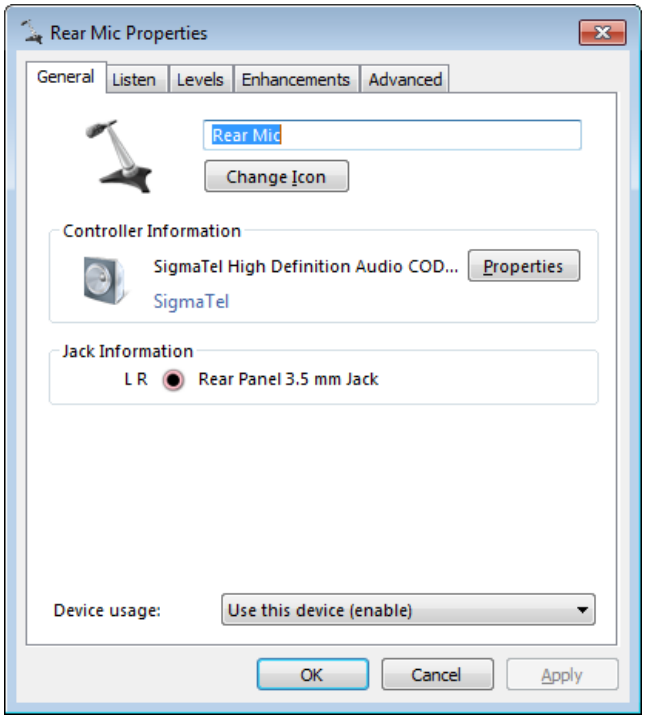

Figure 3. Properties Window for my Microphone.

It's a good idea to check the sound quality of your microphone. One way is via the Listen tab in the Properties window, but I prefer to record a piece of speech, and look at its visualization inside a audio editing tool. Two good free editors that I've tried are Audacity (http://audacity.sourceforge.net/) and Wavosaur (http://www.wavosaur.com/). Figure 4 shows a short recording from the rear microphone using Audacity.

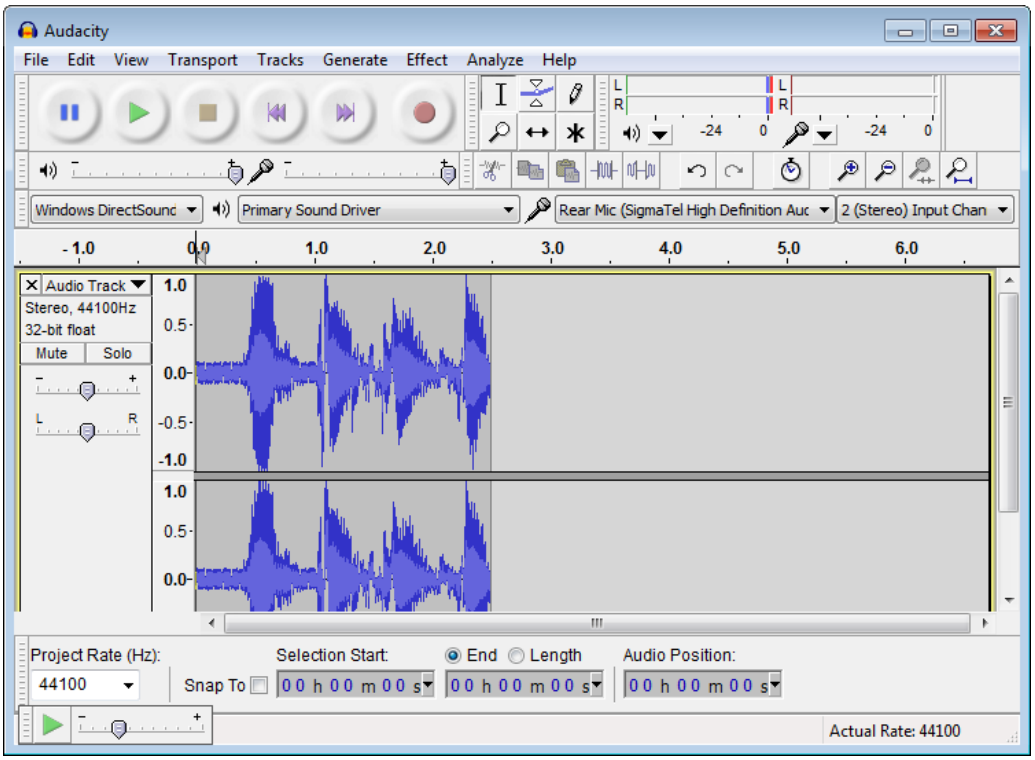

Figure 4. An Audacity Recording.

Sound capture in Audacity is controlled by selecting a microphone input (from the list next to the microphone icon), and then pressing record (the red circle button).

When the track in Figure 4 is played back, there's a slight hiss from the microphone, and a background hum from the room's air conditioner, both of which will affect the quality of the speech recognition.

Windows' "Speech Recognition" control is a great resource for making sure that the microphone works with Microsoft's Speech API (SAPI) recognizer engine, and also for training the engine (see Figure 5).

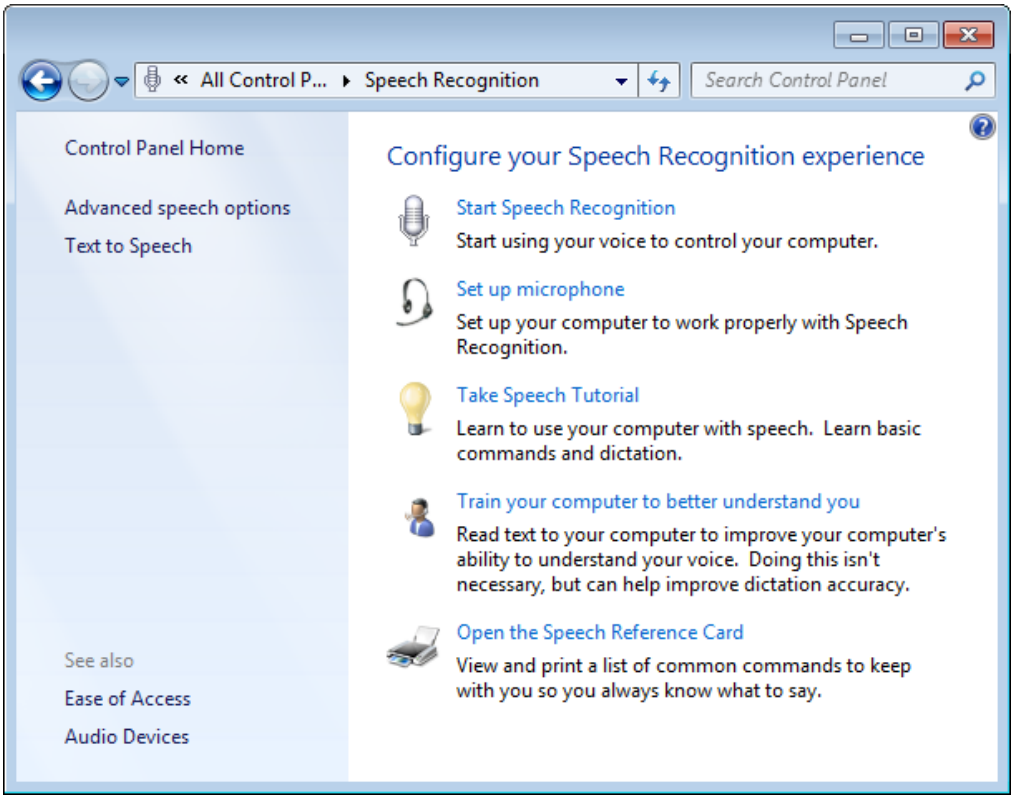

Figure 5. The Speech Recognition Control Panel in Windows 7.

Good quality speech recognition requires that the engine be trained, not only to recognize your voice, but also to deal with background and microphone noise. The Windows 7 training session takes about 10 minutes, but pays dividends later in better recognition accuracy.

Audio Tools on Windows XP

I found another kind of speech recognition problem when checking the audio setup on my Windows XP test machine. In XP, speech recognition is handled by the "Speech" control panel (equivalent to the "Speech Recognition" control in Windows 7), as shown in Figure 6.

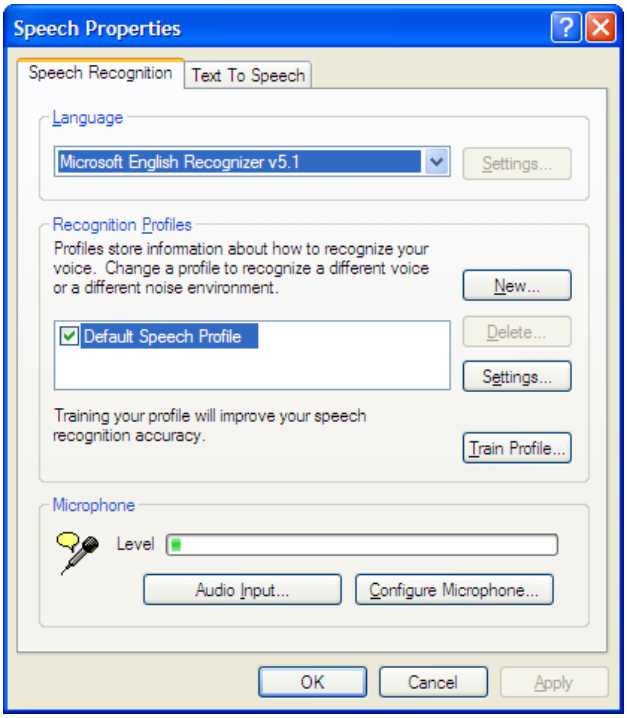

Figure 6. The Speech Control Panel in Windows XP.

Recognizer training is started via the "Train Profile" button, with the training details stored as the default speech profile.

The "Language" pop-down list shows three recognizer engines:

- Microsoft English Recognizer v5.1 (seen in Figure 6)
- Microsoft English (U.S.) v6.1 Recognizer
- SAPI Developer Sample Engine

The "Sample Engine" is a demo, and can't be used for real speech recognition tasks. The version number in "Microsoft English (U.S.) v6.1 Recognizer" refers to the recognizer engine, not the SAPI version (which is v5.1 for Windows XP). There appears to be a problem with its installation, since its "Train Profile" button is grayed out (disabled) when that engine is selected, and pressing on its Settings button throws up an error dialog box stating that the engine cannot be started.

In other words, although my XP machine appears to have three speech recognizers, only "Microsoft English Recognizer v5.1" is actually working.

2. SAPI, JSAPI, and Talking Java

Figure 7 shows the main elements in a speech recognition application.

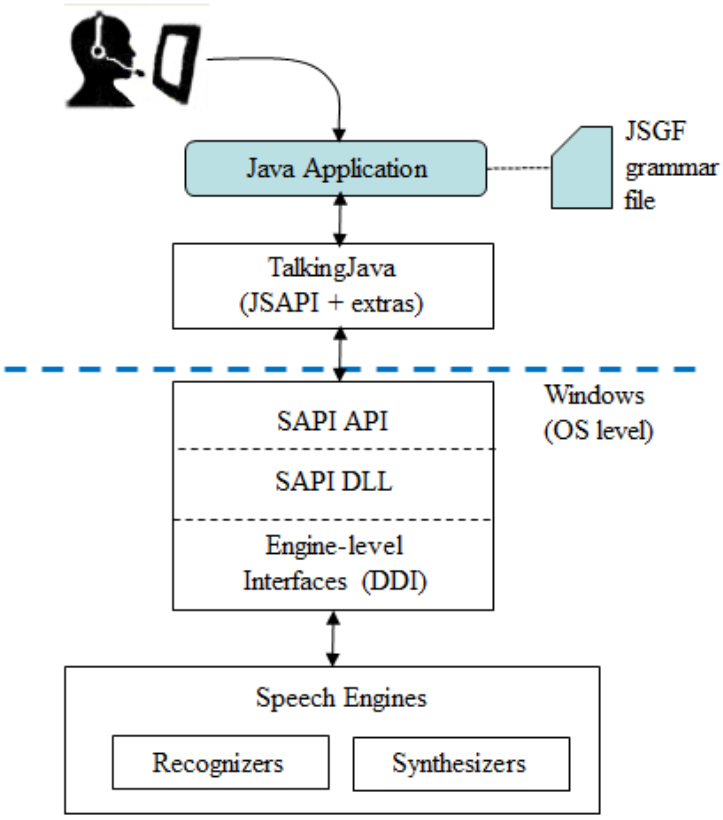

Figure 7. Components in a Speech Recognition Application.

SAPI is MS Windows middleware that provides an API for accessing speech engines (both recognizers and synthesizers). It's installed as part of Windows 7 (and Vista), but may not be present on XP, where is comes as part of MS Office 2003. If you don't have Office, SAPI 5.1 is available online as SpeechSDK51MSM.exe at http://www.microsoft.com/download/en/details.aspx?id=10121. Vista comes with SAPI 5.3, and Windows 7 with SAPI 5.4, but TalkingJava can communicate with any of these versions. A good overview of all the SAPI versions can be found at http://en.wikipedia.org/wiki/Microsoft_Speech_API/. SAPI is also used by Microsoft's SDK for the Kinect, via a .NET managed System.Speech namespace.

SAPI's speech recognition engines currently support eight languages: U.S. English, U.K. English, traditional Chinese, simplified Chinese, Japanese, German, French, and Spanish, with the default determined by your current OS language. To use a different recognizer language, you must install the appropriate language pack, which may be available online from Microsoft, depending on the version of Windows you're using.

Microsoft has a lot of online information about Window's speech recognition capabilities, starting at http://msdn.microsoft.com/en-us/library/bb756992.aspx.

Cloud Garden TalkingJava (http://www.cloudgarden.com/JSAPI/) acts as a link between Java and SAPI. It works with any SAPI-compliant speech engine, and implements the Java Speech API (JSAPI) specification (http://java.sun.com/products/java-media/speech/). TalkingJava also offers useful additional features, such as redirection of audio data to/from files, and GUI tools for selecting engines (e.g. see Figure 8). Two drawbacks of TalkingJava are that it's only

free for non-commercial use, and requires a SAPI-compatible speech engine. Actually, I found this latter condition to be an advantage because of the user-friendly SAPI training tools in Windows.

Oracle (Sun Microsystems, formerly) doesn't offer a JSAPI implementation, but hosts some excellent documentation, including a programmer guide at http://java.sun.com/products/java-media/speech/forDevelopers/jsapi-guide/ and API documents at http://java.sun.com/products/java-media/speech/reference/api/. An omission are code examples, but TalkingJava comes with a large collection.

Speech recognition engines generally support two modes: dictation and grammar-based. Dictation mode allows a user to use unconstrained, free-form speech, which the engine's built-in grammar attempts to recognize. Not surprisingly, the accuracy can be rather poor. Grammar mode offers better success rates because the user's speech is constrained to fit the vocabulary and rules of a particular grammar. A JSAPI grammar is defined using the Java Speech Grammar Format (JSGF), which is documented at http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/. I'll describe some simple examples later.

An alternative to TalkingJava is Sphinx4, an open source Java-based speech recognition engine developed at CMU (http://cmusphinx.sourceforge.net/sphinx4/). Sphinx4 employs Hidden Markov Models to recognize speech, so the system doesn't require any training. Some users have reported excellent results with Sphinx4, but I found the system's accuracy to be poor (perhaps because of my non-North American accent and the noisy recording environment). Creating a user-specific acoustic model in Sphinx4 is non-trivial, although the steps are documented online (http://cmusphinx.sourceforge.net/sphinx4/doc/UsingSphinxTrainModels.html). Sphinx4, unlike SAPI, is available on Linux and Mac OS X.

Sphinx4 doesn't support synthesis, but a good speech synthesizer choice is FreeTTS (http://freetts.sourceforge.net/). I use it in the "Prof. Bob is Ready to Answer Your Questions" chapter online at http://fivedots.coe.psu.ac.th/~ad/jg/ch105/.

3. Listing the Speech Engines

A simple way of testing TalkingJava and JSAPI is to list all the SAPI synthesizer and recognizer engines on your PC. The following is the output for my Windows 7 machine:

```
> java -cp "D:\TalkingJavaSDK-170\cgjsapi.jar;."
       -Djava.library.path=D:\TalkingJavaSDK-170\ ListEngines

CloudGarden's JSAPI1.0 implementation
Version 1.7.0
Implementation contained in files cgjsapi.jar and cgjsapi170.dll
>> Initializing Cloudgarden's JSAPI 1.0, version 1.7.0
>> Free for personal use only.
>> Any form of commercial, corporate or institutional use requires
purchase of a license.
>> Please visit http://www.cloudgarden.com for details.

Synthesizer 1: com.cloudgarden.speech.CGSynthesizerModeDesc@1774b9b [
engineName=Microsoft, modeName=Microsoft SAPI5, Locale=en_US,
```

```
running=null ]
   -->com.cloudgarden.speech.CGVoice@3fa5ac [ name="Microsoft Anna,
SAPI5, Microsoft", AGE_MIDDLE_ADULT, GENDER_FEMALE, style=Microsoft
Anna ]

    Voice: "Microsoft Anna, SAPI5, Microsoft"

Recognizer 1: com.cloudgarden.speech.CGRecognizerModeDesc@6e1dec[
engineName="Microsoft Speech Recognizer 8.0 for Windows (English -
US), SAPI5, Microsoft", modeName=Microsoft Speech Recognizer 8.0 for
Windows (English - US), Locale=en_US running=null,
dictationSupported=true ]

    Profile: "Default Speech Profile"

Recognizer 2: com.cloudgarden.speech.CGRecognizerModeDesc@1264eab[
engineName="Microsoft Speech Recognizer 8.0 for Windows (English -
UK), SAPI5, Microsoft", modeName=Microsoft Speech Recognizer 8.0 for
Windows (English - UK), Locale=en_GB running=null,
dictationSupported=true ]

    Profile: "Default Speech Profile"

>> Shutting down Cloudgarden's JSAPI 1.0 version 1.7.0
```

The output lists one synthesizer with one voice, and two recognizers with the same speech profile. The first recognizer is for US English, the other for UK English. Both have access to SAPI's default profile, which I trained using the "Speech Recognizer" control.

One of the TalkingJava extensions are GUI dialogs to list the available engines, profiles, and voices, illustrated in Figure 8.

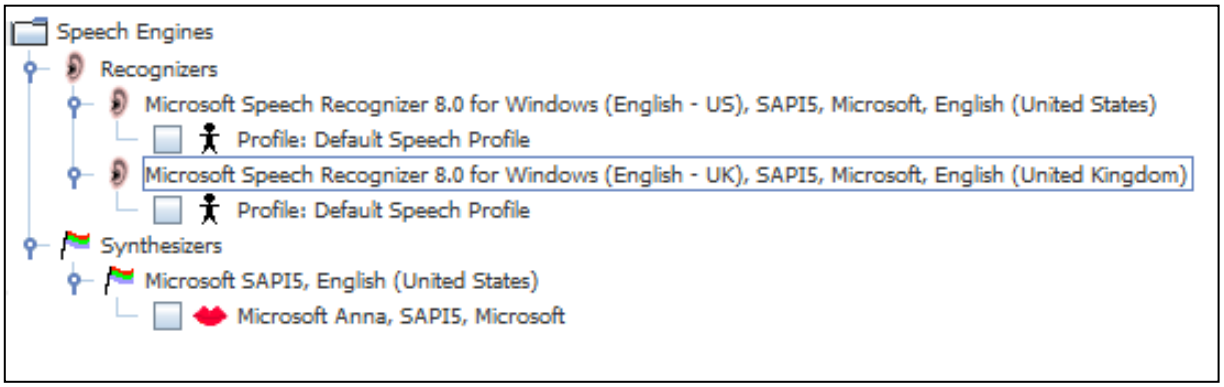

Figure 8. TalkingJava GUI View of SAPI Engines.

I won't be using TalkingJava extensions in this chapter, restricting my examples to JSAPI. However, in the next chapter, I will employ TalkingJava's ability to redirect audio data streams.

The recognizer information shows me that I needn't bother selecting a speaker profile at runtime, as there's only one available, but I could choose between the US and UK English recognizers. If I don't make an explicit choice, then the first recognizer, the US version, will be utilized.

The top-level of my text-based ListEngines application uses the JSAPI Central class to access the speech recognizers and synthesizers.

```
public static void main(String[] args)
{
  // report on all the synthesizers
  EngineList list = Central.availableSynthesizers(null);
  for(int i = 0; i < list.size(); i++) {
    System.out.println("\nSynthesizer " + (i+1) + ":");
    System.out.println("  " + list.elementAt(i));
    listVoices((SynthesizerModeDesc)list.elementAt(i));
  }

  // report on all the recognizers
  list = Central.availableRecognizers(null);
  for(int i = 0; i < list.size(); i++) {
    System.out.println("\nEnglish Recognizer " + (i+1) + ":");
    System.out.println("  " + list.elementAt(i));
    listProfiles((RecognizerModeDesc)list.elementAt(i));
  }
  System.exit(0);
}  // end of main()
```

listVoices() prints the name of the voices associated with a synthesizer. It's necessary to start a synthesizer before reading its properties.

```
private static void listVoices(SynthesizerModeDesc desc)
{
  try {
    Synthesizer synth = Central.createSynthesizer(desc);
    synth.allocate();
    synth.resume();
    synth.waitEngineState(Synthesizer.ALLOCATED);
    SynthesizerProperties props = synth.getSynthesizerProperties();

    desc = (SynthesizerModeDesc)synth.getEngineModeDesc();
    Voice[] vs = desc.getVoices();
    for(int i=0; i < vs.length; i++)
      System.out.println("    Voice: \"" + vs[i].getName() + "\"");

    synth.deallocate();
    synth.waitEngineState(Synthesizer.DEALLOCATED);
  }
  catch(Exception e) {
    e.printStackTrace();
  }
}  // end of listVoices()
```

JSAPI engines (both synthesizers and recognizers) are state machines which must be moved between different execution states by the programmer. The call to Engine.allocate() ensures that the engine loads the necessary resources. Once the engine reaches its ALLOCATED state, the resume() call makes sure that the engine is ready to receive input.

listProfiles() is structured in a similar way to listVoices(), but deals with recognizer properties.

```
private static void listProfiles(RecognizerModeDesc desc)
{
  try {
    Recognizer recg = Central.createRecognizer(desc);
    recg.allocate();
    recg.resume();
    recg.waitEngineState(Synthesizer.ALLOCATED);
    RecognizerProperties props = recg.getRecognizerProperties();

    desc = (RecognizerModeDesc)recg.getEngineModeDesc();
    SpeakerProfile[] profs = desc.getSpeakerProfiles();
    for(int i=0; i < profs.length; i++)
      System.out.println("    Profile: \"" +
                                profs[i].getName() + "\"");

    recg.deallocate();
    recg.waitEngineState(Synthesizer.DEALLOCATED);
  }
  catch(Exception e) {
    e.printStackTrace();
  }
}  // end of listProfiles()
```

A speech profile is built from the training information collected for a particular user. It's possible for recognizers to have multiple profiles, one for each user.

When a JSAPI recognizer is created, the profile can be specified, or the default profile will be used.

Listing the Engines on Windows XP

The ListEngines application produces different output on my Windows XP machine:

```
Synthesizer 1: com.cloudgarden.speech.CGSynthesizerModeDesc@dbe178 [
engineName=Microsoft, modeName=Microsoft SAPI5, Locale=en_US,
running=null ]
   -->com.cloudgarden.speech.CGVoice@1af9e22 [ name="Microsoft Mary,
SAPI5, Microsoft", AGE_MIDDLE_ADULT, GENDER_FEMALE, style=Microsoft
Mary ]
   -->com.cloudgarden.speech.CGVoice@b6ece5 [ name="Microsoft Mike,
SAPI5, Microsoft", AGE_MIDDLE_ADULT, GENDER_MALE, style=Microsoft
Mike ]
   -->com.cloudgarden.speech.CGVoice@17ace8d [ name="Microsoft Sam,
SAPI5, Microsoft", AGE_MIDDLE_ADULT, GENDER_MALE, style=Microsoft Sam
]
   -->com.cloudgarden.speech.CGVoice@18eb9e6 [ name="SampleTTSVoice,
SAPI5, Microsoft", AGE_MIDDLE_ADULT, GENDER_MALE,
style=SampleTTSVoice ]

    Voice: "Microsoft Mary, SAPI5, Microsoft"
    Voice: "Microsoft Mike, SAPI5, Microsoft"
    Voice: "Microsoft Sam, SAPI5, Microsoft"
    Voice: "SampleTTSVoice, SAPI5, Microsoft"

Synthesizer 2: com.cloudgarden.speech.CGSynthesizerModeDesc@110d81b [
engineName=N/A, modeName=N/A SAPI5, Locale=en_US, running=null ]
   -->com.cloudgarden.speech.CGVoice@1050e1f [ name="LH Michael,
```

```
SAPI5, N/A", AGE_NEUTRAL, GENDER_MALE, style=LH Michael ]
   -->com.cloudgarden.speech.CGVoice@e24e2a [ name="LH Michelle,
SAPI5, N/A", AGE_NEUTRAL, GENDER_FEMALE, style=LH Michelle ]

    Voice: "LH Michael, SAPI5, N/A"
    Voice: "LH Michelle, SAPI5, N/A"

Recognizer 1: com.cloudgarden.speech.CGRecognizerModeDesc@128e20a[
engineName="Microsoft English Recognizer v5.1, SAPI5, Microsoft",
modeName=Microsoft English Recognizer v5.1, Locale=en_US
running=null, dictationSupported=true ]

    Profile: "Default Speech Profile"

Recognizer 2: com.cloudgarden.speech.CGRecognizerModeDesc@1bf52a5[
engineName="Microsoft English (U.S.) v6.1 Recognizer, SAPI5,
Microsoft", modeName=Microsoft English (U.S.) v6.1 Recognizer,
Locale=en_US running=null, dictationSupported=true ]

SetRecognizer failed for inproc rec, pos = 1
ERROR=8004503a
Created shared rec, pos = 1
SetRecognizer failed for shared rec, pos = 1
ERROR=8004503a

    Profile: "Default Speech Profile"

Recognizer 3: com.cloudgarden.speech.CGRecognizerModeDesc@10b9d04[
engineName="SAPI Developer Sample Engine, SAPI5, Microsoft",
modeName=SAPI Developer Sample Engine, Locale=en_US running=null,
dictationSupported=true ]

    Profile: "Default Speech Profile"
```

There are two synthesizers, offering a total of six voices, and three recognizers with a single profile. A 8004503a error is reported when ListEngine attempts to start the "Microsoft English (U.S.) v6.1 Recognizer". The corresponding error string is SPERR_NOT_FOUND, which indicates that some requested data item was not found. The problem is most likely an installation issue, since "Microsoft English Recognizer v5.1" was probably installed as part of the SAPI 5.1 Speech SDK (available at http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=10121), while "Microsoft English (U.S.) v6.1 Recognizer" will have been added separately, perhaps during an installation of MS Office 2003.

4. Recognizing Speech Using Grammars

The following Recog.java program shows how to process spoken input that matches a Java Speech Grammar Format (JSGF) grammar.

I won't be explaining JSGF in much detail, assuming the reader is familiar with grammars from other branches of computing. JGSF allows the definition of regular grammars, which essentially limit the use of recursion to the end of a grammar rule. Regular grammars are equivalent in expressive power to regular expressions, as used in tools such as grep.

A good source of more information is the JSGF Specification at http://java.sun.com/products/java-media/speech/reference/docs/

The main steps in using a grammar are always the same:

- create a recognizer, moving it to its ALLOCATED state;
- load the grammar from a file;
- add recognition result listeners that respond to RESULT_ACCEPTED or RESULT_REJECTED events;
- commit the grammar, which assigns the grammar to the recognizer after checking that it is legal;
- request the recognizer's focus and start processing user input.

The top-level of Recog.java follows those steps:

```
// global
private Recognizer rec;


public Recog(String fnm)
{
  try {
    // create a recognizer that supports English
    rec = Central.createRecognizer(
                    new EngineModeDesc(Locale.ENGLISH));
    rec.allocate();

    // load the grammar
    RuleGrammar gram = rec.loadJSGF( new FileReader(fnm) );
    gram.setEnabled(true);
    System.out.println("Loaded grammar from " + fnm);
    printGrammar(gram);

    // add the listener to report results
    rec.addResultListener(this);

    // commit the grammar
    rec.commitChanges();

    // request focus and start listening
    System.out.println("\nSay something please...");
    rec.requestFocus();
    rec.resume();
  }
  catch (Exception e) {
    e.printStackTrace();
    System.exit(1);
  }
}  // end of Recog()
```

Recog is defined as a subclass of ResultAdapter, which contains empty method implementations for the ResultListener interface. Recog redefines the ResultListener methods resultAccepted() and resultRejected(): resultAccepted() is called when a user's utterance matches the grammar, and resultRejected() when the utterance cannot be matched.

resultAccepted() prints out the tokens in the result, and triggers application termination if the first token is the word "exit":

```
public void resultAccepted(ResultEvent e)
{
  Result r = (Result)(e.getSource());
  ResultToken tokens[] = r.getBestTokens();
  if (tokens == null)
    return;

  // print recognized text
  for (int i = 0; i < tokens.length; i++)
    System.out.print(tokens[i].getSpokenText() + " ");
  System.out.println();

  if (tokens[0].getSpokenText().equals("exit"))
    endRecognizer();
  else
    System.out.println("\nSay something please...");
}  // end of resultAccepted()
```

A call to resultAccepted() doesn't mean that the tokens in the Result object are correct, only that the user's utterance corresponds to grammar tokens with a specified confidence level. (This level can be adjusted in the recognizer at creation time via the RecognizerProperties class.) The call to Result.getBestTokens() returns the tokens that span the user's utterance with the highest confidence. It is possible that there may be no suitable tokens, in which case the tokens array will be null.

One way of ending a recognition session is to look for a specific word, which is "exit" in my grammars. If the first token is "exit" then the application is closed down by endRecognizer().

```
private void endRecognizer()
// deallocate the recognizer and exit
{
  if (rec.testEngineState(Recognizer.ALLOCATED) &&
      !rec.testEngineState(Recognizer.DEALLOCATING_RESOURCES)) {
    try {
      System.out.println("Finishing...\n");
      rec.forceFinalize(true);
      rec.deallocate();
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
    System.exit(0);
  }
}  // end of endRecognizer()
```

Recognizer.forceFinalize() instructs the recognizer to complete any user processing before deallocating resources and closing down. The if-test ensures that deallocation is only carried out if the recognizer is in the ALLOCATED state and not already closing down.

Although endRecognizer() implements the preferred termination approach, it can take several seconds to complete. During that time, the application will stop responding,

which may worry the user. A faster, but less orderly, way to finish is to leave out the call to Recognizer.forceFinalize() and ignore any exception that is raised by immediately calling Recognizer.deallocate().

The other ResultListener method implemented by Recog is resultRejected(), which is called when a user utterance fails to match against the grammar. The method asks the user to repeat what they just said:

```
public void resultRejected(ResultEvent e)
{ System.out.println("Sorry, I didn't understand that.
                              Please try again...");  }
```

4.1. Defining Grammars

A JSGF grammar is typically read in from a text file (as in Recog.java), but it is possible to create one within the program, and to modify a loaded grammar at run time .

Each grammar starts with header information, followed by a series of rules. The top-level rule is preceded by a "public" keyword. For example, the file "hello.gram" contains:

```
#JSGF V1.0;
grammar hello;

public <sentence> = hello world | good morning | hi |
                    hello computer | exit ;
```

The grammar's name is "hello", and consists of a single sentence rule which can match five utterances: "hello world", "good morning", "hi", "hello", or "exit". The "|" grammar operator means "or" (as in regular expressions).

The following output shows Recog.java being run with "hello.gram" as its grammar:

```
> java -cp "D:\TalkingJavaSDK-170\cgjsapi.jar;."
      -Djava.library.path=D:\TalkingJavaSDK-170\  Recog hello.gram

CloudGarden's JSAPI1.0 implementation
Version 1.7.0
Implementation contained in files cgjsapi.jar and cgjsapi170.dll
>> Initializing Cloudgarden's JSAPI 1.0, version 1.7.0
>> Free for personal use only.
>> Any form of commercial, corporate or institutional use requires
purchase of a license.
>> Please visit http://www.cloudgarden.com for details.
Loaded grammar from hello.gram
Grammar Name: hello
  sentence = ( ( hello  world) | ( good  morning) |  hi | ( hello
computer) | exit )

Say something please...
hello world

Say something please...
hi

Say something please...
```

```
exit
Finishing...

>> Shutting down Cloudgarden's JSAPI 1.0 version 1.7.0
>
```

The user said "hello world", "hi", and "exit", which were correctly parsed, and the "exit" input caused the application to end.

One part of Recog.java which hasn't been explained is printGrammar(), which prints out the loaded grammar:

```
private void printGrammar(RuleGrammar gram)
{
  System.out.println("Grammar Name: " + gram.getName());

  String[] rules = gram.listRuleNames();
  for(int i=0; i < rules.length; i++)
    System.out.println("  " + rules[i] + " = " +
                       gram.getRule(rules[i]) + "\n");
}  // end of printGrammar()
```

All the rule names are retrieved (the words to the left of "=" in rules) and stored in rules[]. The code iterates through the names to access their rule bodies with RuleGrammar.getRule().

4.2. A Numbers Grammar

The following "numbers" grammar uses additional grammar features, namely optional grouping ([…]) and nonterminals in the rule body (i.e. calls to other rules).

```
#JSGF V1.0;

grammar numbers;

public <number> = <hundreds> | <tens> | <teens> | <units> |
          (<hundreds> [and] <tens>) |
          (<hundreds> [and] <teens>) |
          (<hundreds> [and] <units>) |
          (<tens> [and] <units>) |
          (<hundreds> [and] <tens> [and] <units>) |
          exit ;

<units> = one | two | three | four | five | six | seven |
           eight | nine;

<teens> = eleven | twelve | thirteen | fourteen | fifteen |
           sixteen | seventeen | eighteen | nineteen;

<tens> = ten | twenty | thirty | forty | fifty | sixty | seventy |
          eighty | ninety;

<hundreds> = <units> hundred;
```

This grammar can recognize spoken numbers involving hundreds, tens, and units, such as "five hundred and twenty seven", or "thirty two".

5. Spoken Breakout

After a somewhat lengthy preamble, it's time to add speech recognition to a Kinect application, and I've modified the Breakout game of chapter 9 (online at http://fivedots.coe.psu.ac.th/~ad/jg/nui166/). Figure 9 shows the game executing, and it's just possible to see the user holding a microphone in his left hand.

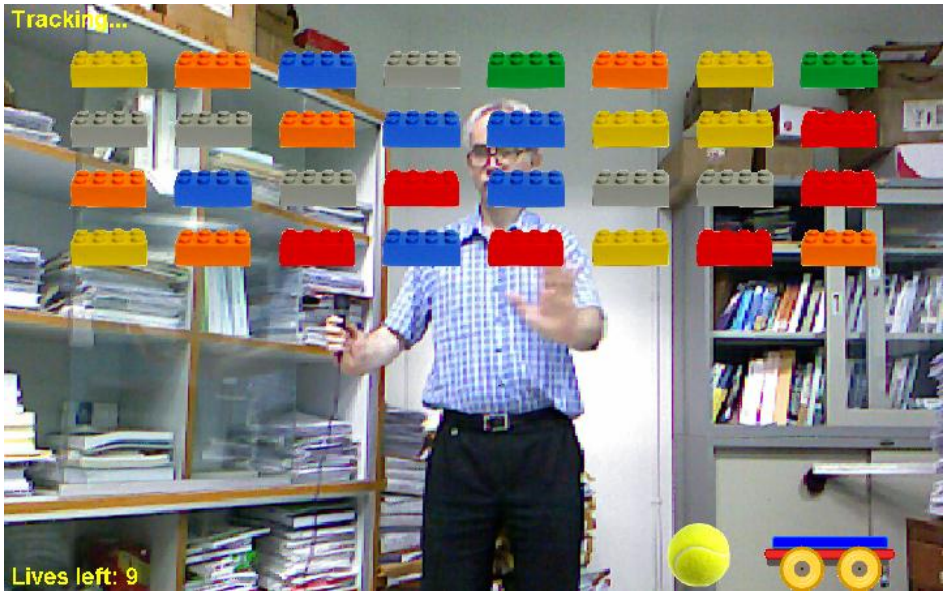

Figure 9. Playing Spoken Breakout.

Figure 10 is a close-up of the microphone plugged into a five meter extension cable. The cable is needed so the user can stand far enough away from the Kinect while still having the microphone lead reach the audio port of the PC.

Figure 10. The Microphone Used for the Game.

Speech recognition accuracy requires a good quality microphone, together with engine training with that mike, in the audio environment where the game will be played. Since I'm using grammar-based recognition, it's possible to survive with a

cheaper microphone because its poor sound will be offset by the easier processing required by SAPI. However, if you want to use dictation-style recognition, or play back the recording through speakers, then spend a bit more on decent equipment.

The spoken extensions to the game involve being able to verbally:

- pause, resume, or terminate the game;
- make the ball move faster or slower.

Commands for these features are defined in a "breakout" grammar:

```
#JSGF V1.0;
grammar breakout;

public <cmds> = ( move <speed> ) | pause | resume | exit;
<speed> = faster | slower;
```

The class diagram for the modified application is shown in Figure 11.

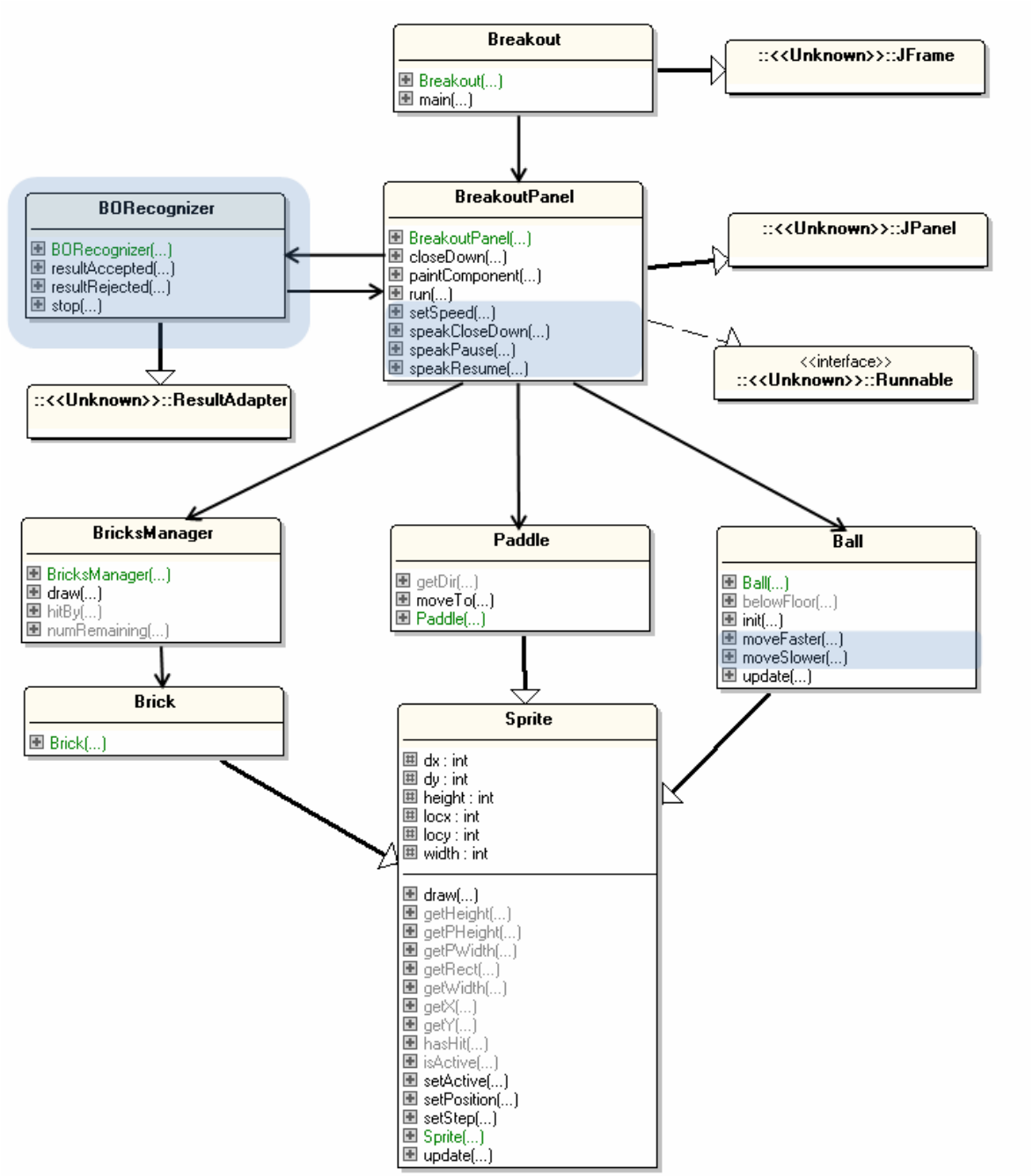

Figure 11. Class Diagrams for the Spoken Breakout Game.

The changes to the code of Chapter 9 are shaded – the BORecognizer class is a JSAPI recognizer which processes the user's utterances according to the grammar above. BORecognizer communicates with the game panel (BreakoutPanel) via new public methods that affect the ball speed and pause/resume/exit the game. Changing the ball's speed is handled by the addition of moveFaster() and moveSlower() methods to the Ball class.

5.1. Processing User Speech

BORecognizer is quite similar to the earlier Recog.java example, and starts in the same way:

- it creates a recognizer, moving it to its ALLOCATED state;
- loads the breakout grammar from a file (bout.gram);
- adds result listeners that respond to RESULT_ACCEPTED and RESULT_REJECTED events;
- commits the grammar, which assigns the grammar to the recognizer after checking that it is legal;
- requests the recognizer's focus and starts processing user input.

The BORecognizer() constructor implements these steps:

```
// globals
private static final String GRAM_FNM = "bout.gram";

private BreakoutPanel boPan;
private Recognizer rec;


public BORecognizer(BreakoutPanel bp)
{
  boPan = bp;

  try {
    // create a recognizer that supports English
    rec = Central.createRecognizer(
                new EngineModeDesc(Locale.ENGLISH));
    rec.allocate();

    // load the grammar
    RuleGrammar gram = rec.loadJSGF( new FileReader(GRAM_FNM) );
    gram.setEnabled(true);
    System.out.println("Loaded grammar from " + GRAM_FNM);
    printGrammar(gram);

    // add the listener to report results
    rec.addResultListener(this);

    // commit the grammar
    rec.commitChanges();

    // request focus and start listening
    System.out.println("\nSay something please...");
    rec.requestFocus();
    rec.resume();
  }
  catch (Exception e) {
    e.printStackTrace();
    System.exit(1);
  }
}  // end of BORecognizer()
```

One new feature is the storage of a reference to the BreakoutPanel object, which will allow its methods to be called when the verbal input is being processed.

5.2. Processing the Results

The resultAccepted() method is called when a utterance is successfully parsed:

```
public void resultAccepted(ResultEvent e)
// Receives RESULT_ACCEPTED event: print tokens, processes them
{
  Result r = (Result)(e.getSource());
  ResultToken tokens[] = r.getBestTokens();

  // print recognized text
  for (int i = 0; i < tokens.length; i++)
    System.out.print(tokens[i].getSpokenText() + " ");
  System.out.println();

  processCmd(tokens);

  System.out.println("\nSay command please...");
}  // end of resultAccepted()
```

The sequence of tokens are passed to processCmd() where the command is acted upon by calling the relevant method in BreakoutPanel.

```
private void processCmd(ResultToken tokens[])
/* pass commands to the BreakoutPanel
     <cmds> = ( move <speed> ) | pause | resume | exit;
     <speed> = faster | slower;
*/
{ String cmd = tokens[0].getSpokenText();
  if (cmd.equals("move")) {
    if (tokens.length == 1)
      System.out.println("move has no direction argument");
    else
      boPan.setSpeed(tokens[1].getSpokenText());
  }
  else if (cmd.equals("pause"))
    boPan.speakPause();
  else if (cmd.equals("resume"))
    boPan.speakResume();
  else if (cmd.equals("exit"))
    boPan.speakCloseDown();
  else
    System.out.println("Unknown command: " + cmd);
}  // end of processCmd()
```

5.3. Pausing and Resuming the Game

The pausing and resumption of the game are managed by the BreakoutPanel methods speakPause() and speakResume():

```
// in BreakoutPanel
```

```
// global
private boolean isPaused;


public void speakPause()
{ isPaused = true;  }

public void speakResume()
{ isPaused = false;  }
```

The global isPaused boolean is utilized by the panel's game loop inside BreakoutPanel.run(). The loop passes through four stages: waiting for the Kinect, updating the game, drawing the game, and sleeping.

```
// globals
private static final int CYCLE_TIME = 25;
                              // time for one game iteration, in ms

private boolean isRunning = false;   // used to stop the game loop
private boolean gameOver = false;    // has user finished the game?

// OpenNI and NITE vars
private Context context;
private SessionManager sessionMan;

private BORecognizer rec;
private JFrame top;


public void run()
{
  long duration;
  isRunning = true;

  while(isRunning) {
    try {    // wait for Kinect input
      context.waitAnyUpdateAll();
      sessionMan.update(context);
    }
    catch(StatusException e)
    {  System.out.println(e);
       System.exit(1);
    }

    long startTime = System.currentTimeMillis();
    updateCameraImage();
    if (!isPaused && !gameOver)
      updateGame();   // update game elements

    duration = System.currentTimeMillis() - startTime;
    repaint();
    if (duration < CYCLE_TIME) {
      try {
        Thread.sleep(CYCLE_TIME-duration);
                // wait until CYCLE_TIME time has passed
      }
      catch (Exception ex) {}
    }
  }
```

```
  // close down
  try {
    context.stopGeneratingAll();
  }
  catch (StatusException e) {}
  context.release();
  top.setState(Frame.ICONIFIED);
      // minimize application while closing down
  rec.stop();    // stop the speech recognizer
  System.exit(0);
}  // end of run()
```

The isPaused boolean is checked before the call to updateGame() which updates the game's state (e.g. updates the ball's position).

```
// in BreakoutPanel.run()
//   :
if (!isPaused && !gameOver)
  updateGame();
```

This means that if the game is paused then its state won't be changed during a game iteration, but the game visuals will still be drawn onto the panel. This allows the background of the game, the Kinect camera's input, to be kept up-to-date.

5.4. Stopping the Game

The game is stopped when BreakoutPanel.run() exits its game loop and closes down the Kinect and the speech recognizer. Looping stops when isRunning is set to false, which can occur when the user says "exit". BORecognizer responds by calling BreakoutPanel.speakCloseDown():

```
// global
private boolean isRunning = false;

public void speakCloseDown()
{  isRunning = false; }
```

The closing-down code at the end of run() has two additions over the original version in Chapter 9, one of which is the termination of the BORecognizer speech recognizer:

```
// globals
private BORecognizer rec;    // the speech recognizer
private JFrame top;   // ref to the top-level of the application

// at the end of BreakoutPanel.run()
//   :
top.setState(Frame.ICONIFIED);
               // minimize application while closing down
rec.stop();    // stop the speech recognizer
```

BORecognizer.stop() makes the recognizer stop processing, release its resources, and terminate.

```
// in BORecognizer
// globals
private Recognizer rec;


private void stop()
// deallocate the recognizer
{
  if (rec.testEngineState(Recognizer.ALLOCATED) &&
     !rec.testEngineState(Recognizer.DEALLOCATING_RESOURCES)) {
    try {
      System.out.println("Closing down the Speech Recognizer;
                                        please wait...\n");
      rec.forceFinalize(true);
      rec.deallocate();
    }
    catch (Exception ex) {}
  }
}  // end of stop()
```

stop() is very similar to the endRecognizer() method in Recog.java.

One drawback of calling Recognizer.forceFinalize() and Recognizer.deallocate() is the possibility of a lengthy wait while recognition is wound up. Sometimes the delay can last for 2-3 seconds, during which time Breakout freezes. This can be quite disconcerting to a player, and so I added a call to Frame.setState() to minimize the full-screen window just prior to calling BORecognizer.stop().

5.5. Changing the Ball's Speed

When the user says "move faster" or "move slower", BORecognizer calls BreakoutPanel.setSpeed() with "faster" or "slower" as its argument. setSpeed() passes a speed request onto the Ball object by calling Ball.moveFaster() or Ball.moveSlower().

```
// in BreakoutPanel
// global
private Ball ball;


public void setSpeed(String speedStr)
{
  if(speedStr.equals("faster"))
    ball.moveFaster();
  else if (speedStr.equals("slower"))
    ball.moveSlower();
  else
    System.out.println("Unknown speed: " + speedStr);
}  // end of setSpeed()
```

The Ball class uses an INCR integer constant to adjust the ball's speed when it bounces off the paddle. I utilize this constant in a similar way in moveFaster() and moveSlower(), applying it to current x- and y- axis steps values in dx and dy.

```
// in Ball
```

```
// globals
private static final int INCR = 10;
         // for speeding up / slowing down the sprite


public void moveFaster()
{
  dx = (dx >= 0) ? (dx + INCR) : (dx - INCR);
  dy = (dy >= 0) ? (dy + INCR) : (dy - INCR);
}  // end of moveFaster()


public void moveSlower()
{
  if (dx > INCR)
    dx -= INCR;
  else if (dx < -INCR)   // is negative
    dx += INCR;

  if (dy > INCR)
    dy -= INCR;
  else if (dy < -INCR)   // is negative
    dy += INCR;
}  // end of moveSlower()
```

moveSlower() has slightly lengthier code to prevent the ball's speed from dropping to 0 along a particular axis.

5.6. Game Play Speech Issues

There are obvious benefits to being able to interact with the game verbally. It's more convenient to say "pause", "resume", or "exit" instead of walking over to the PC's keyboard to press the relevant key. In general, it's preferable not to use the keyboard in Kinect games since the act of moving out of the Kinect camera's viewing area will affect its processing.

A downside of speech recognition is the relative slowness of verbal command execution. I've already mentioned that the closing down of the recognizer can take 2-3 seconds, which I've 'hidden' by minimizing the game's window. More generally, a command seems to take about one second to be processed, which is quite slow for a game.

Another drawback is that recognition accuracy requires the user to train the engine. This isn't particularly difficult, but might be asking too much of a casual game player. Furthermore, the trained speech profile will only be accurate for that user, or someone who sounds a lot like them.

Arguably, asking the user to hold a microphone while playing is a disadvantage. However, it does depend on the game – one involving karaoke-style interactions will feel more realistic with a mike. In any case, I address this concern in the next chapter where I switch over to using the Kinect sensor's own microphones rather than one plugged into the PC.