Project: A java programme that recognizes single animal name(English) and produces matching animal sound.

Aim of this programme is to build up a visible system that is able to recognize some English words, and produce sound which represents the recognized word. Further, the number of English words is limited to only 36 in order to increase accuracy.

Approach:

1. Launch the programme by pressing the “start” button on screen

2. After loading is completed, wait approximately 10 seconds to let the computer starts
   microphone, then people can speak to the microphone

3. Sphinx4 detects and recognizes the words that people speak

4. Sphinx4 outputs the recognized words (on the console)

5. The words is compared to a list of animal names

6. If the word exists, it will play the sound of the animal

7. If the word is not exist, it will not play any sound

8. Wait for another word and repeat step 3

About models
The work of speech recognition in this programme is done with the help of

CMU Sphinx. sphinx-core.jar and sphinx4-data.jar files are downloaded from: https://oss.sonatype.org/#nexus-search

There are basically three models required for speech recognition in Sphinx4:

1. Acoustic model

2. Phonetic dictionary

3. Language model

The acoustic model in this programme comes from the sphinx4-data,jar. However, the programme uses particular phonetic dictionary and language model due to the limited words that are used. The specific phonetic dictionary(5206.dic) and language model(5206.lm) are made by Sphinx Online Base Generator(http://www.speech.cs.cmu.edu/tools/lmtool-new.html).

Importing referenced files
The downloaded jar files have been added to the project file 'javaProject', and path has been built to both of them. The generated files(5206.dic and 5206.lm) have been added under the project file 'javaProject'.

Coding
The main package 'voiceLauncher' is built under the project 'javaProject'. There are totally three classes in this package(Figure.1), among which the class named 'VoiceMain' holds the main() method, and the class named 'window' and 'playSound' have all the attributes and methods which will be used in main() method. The following description will start from class 'window', then get into the class 'playSound', lastly end with describing the class 'VoiceMain'.

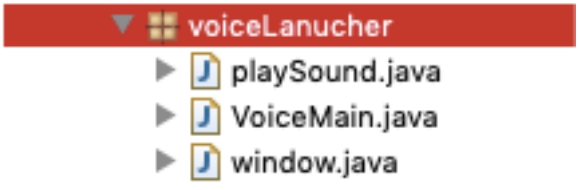

Figure.1

1. The class named ‘window’

Aim of this class is to create a visible window which can interface with and instruct people after launching the programme. Except for the codes to import necessary java packages, the whole class has been divided into several sub-methods.

The first method is called mainContent()(Figure.2). It builds up a visible window on screen. From codes in line 39 to line 42, a window is labeled as “How Animal Talks”, and its size is 590 * 300. Line 40 indicates the way of exiting the programme, namely, if the window is closed, then the execution ends. From codes in line 45 to line 57, a functional button is added to ‘panel1’. To be more specific, first of all, an Actionlistener is built in order to catch status of the button. Then, layout and size of the button are set. Lastly, the button entitled with Actionlistener

is added to ‘panel1’. Code in line 60 intents to give ‘panel1’ a background. Finally, the last step is to add ‘panel1' to this window and set the window as visible.

```
37      public void mainContent() throws IOException {
38          //window
39          f = new JFrame("How Animal Talks");
40          f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
41          f.setSize(590,300);
42          f.setLocation(500,500);
43
44          //button(p1 -> p2)
45          ActionListener Action = new ActionListener() {
46              public void actionPerformed(ActionEvent e) {
47                  changePanel1();
48                  if (e.getSource() == bt){
49                      test = "pressed";
50                  }
51              }
52          };
53
54          panel1.setLayout(null);
55          bt.setBounds(250,120,90,60);
56          panel1.add(bt);
57          bt.addActionListener(Action);
58
59          //p1
60          panel1.setBackground(Color.white);
61
62          //show panel1
63          f.getContentPane().add(panel1);
64          f.setVisible(true);
65      }
```

Figure.2

The second method is called panelContent()(Figure.3). This method builds up all the contents of six panels. These panels will appear in turns according to the progress of programme. On every panel, there are two elements, so it is important to set the layout forehead. Codes in line 70 and line 71 decide that these two elements are overlay with each other. Codes from line 73 to line 75 decide features of ‘label2’ which is added to ‘panel2'. Then, line 78 to line 81 set the picture ‘picLabel’ which is the background of ‘label2’. Lastly, adding ‘label2’ and ‘picLabel' on top of ‘panel2’. Since the background picture will not change no matter what label is on it, only labels with different contents are created from line 85 to line 114.

```
67    public void panelContent() throws IOException {
68
69        //panel2-text & image for all panels
70        LayoutManager overlay2 = new OverlayLayout(panel2);
71        panel2.setLayout(overlay2);
72
73        label2 = new JLabel("Loading...");
74        label2.setFont(new Font("Serif", Font.BOLD, 22));
75        label2.setForeground(Color.white);
76        panel2.add(label2);
77
78        imageIcon = new ImageIcon("/Users/ankiegao/Desktop/pic.jpg");
79        image = imageIcon.getImage();
80        newimg = image.getScaledInstance(590, 300,  java.awt.Image.SCALE_SMOOTH);
81        picLabel = new JLabel(new ImageIcon(newimg));
82        panel2.add(picLabel);
83
84        //panel3-text
85        LayoutManager overlay3 = new OverlayLayout(panel3);
86        panel3.setLayout(overlay3);
87
88        label3 = new JLabel("Speaking...");
89        label3.setFont(new Font("Serif", Font.BOLD, 22));
90        label3.setForeground(Color.white);
91
92        //panel4-text
93        LayoutManager overlay4 = new OverlayLayout(panel4);
94        panel4.setLayout(overlay4);
95
96        label4 = new JLabel("Recognizing...");
97        label4.setFont(new Font("Serif", Font.BOLD, 22));
98        label4.setForeground(Color.white);
99
100       //panel5-text
101       LayoutManager overlay5 = new OverlayLayout(panel5);
102       panel5.setLayout(overlay5);
103
104       label5 = new JLabel("Playing sound...");
105       label5.setFont(new Font("Serif", Font.BOLD, 22));
106       label5.setForeground(Color.white);
107
108       //panel6-text
109       LayoutManager overlay6 = new OverlayLayout(panel6);
110       panel6.setLayout(overlay6);
111
112       label6 = new JLabel("Try again...");
113       label6.setFont(new Font("Serif", Font.BOLD, 22));
114       label6.setForeground(Color.white);
115   }
```

Figure.3

Methods left in this class all aim to change the previous panel to target panel(Figure.4). There are six panels in total, so it needs five methods to connects every two panels. The first one in this part is different from others in that it skips the code that is used to add contents to panel, because contents of ‘panel1’ has already been added by method mainContent().

```
118     public void changePanel1() {
119         f.getContentPane().remove(panel1);
120         f.getContentPane().add(panel2);
121         f.validate();
122     }
123     public void changePanel2() {
124         panel3.add(label3);
125         panel3.add(picLabel);
126         f.getContentPane().remove(panel2);
127         f.getContentPane().add(panel3);
128         f.validate();
129     }
130     public void changePanel3() {
131         panel4.add(label4);
132         panel4.add(picLabel);
133         f.getContentPane().removeAll();
134         f.getContentPane().add(panel4);
135         f.validate();
136     }
137     public void changePanel4() {
138         panel5.add(label5);
139         panel5.add(picLabel);
140         f.getContentPane().removeAll();
141         f.getContentPane().add(panel5);
142         f.validate();
143     }
144     public void changePanel5() {
145         panel6.add(label6);
146         panel6.add(picLabel);
147         f.getContentPane().removeAll();
148         f.getContentPane().add(panel6);
149         f.validate();
150     }
```

Figure.4

2. The class named ‘playSound’(Figure.5)

Aim of this class is to create a method which can play selected sound file. First of all, necessary packages are imported. Then, a method called sound() is created to play the sound of specific path. The playSound method used in sound() is further indicated by codes in line 20 to line 29.

```
1  package voiceLanucher;
2
3  import java.io.File;
4  import javax.sound.sampled.AudioSystem;
5  import javax.sound.sampled.Clip;
6
7
8  public class playSound {
9
10     String path ="/Users/ankiegao/Desktop/audio/";
11     String animal = "";
12     String suffix = ".wav";
13
14     public void sound() {
15
16         File sound = new File(path+animal+suffix);
17         PlaySound(sound);
18     }
19
20     private static void PlaySound(File Sound) {
21         try {
22             Clip clip = AudioSystem.getClip();
23             clip.open(AudioSystem.getAudioInputStream(Sound));
24             clip.start();
25
26             Thread.sleep(clip.getMicrosecondLength()/1000);
27
28         }catch(Exception e) {
29             System.err.println("Problem when playing sound : " + e);
30         }
31     }
32 }
```

Figure.5

3. The class named ‘VoiceMain’

Aim of this main class is to generate a while-loop that can recognize what people speak to microphone repeatedly, show visible window as required and play sound if input speech matches word in words list.

As first part of this class, new objects are initialised(line 13,14). A name list includes all the 36 animal names is created. Then, the programme starts by calling up the mainContent() method and panelContent() in class ‘window’(Figure.6).

```
1  package voiceLanucher;
2
3  import java.io.IOException;
9
10 public class VoiceMain {
11
12     public static void main(String[] args) throws IOException {
13         window w = new window();
14         playSound p = new playSound();
15
16         String outcome;
17         String[] namelist = {"bird","buffalo","bat","bear","chicken","cat","kitten","cc
18
19         w.mainContent();
20         w.panelContent();
```

Figure.6

The second part is the core part that recognizes speech and performs accordingly(Figure.7). Codes from line 23 to line 26 hold the programme for a while until button of ‘panel1' is pressed. When the button is pressed, configuration object is created and set. To use the microphone as a source of input, a LiveSpeechRecognizer is created to pass all configuration. While loading is completed, code in line 40 changes ‘panel2’ to ‘panel3', guiding people to speak. It normally takes some time before the recognition starts. From line 43 to line 63, a while loop is used to catch all the speech that people speak. In line 44, panelContent() is called up again, otherwise the panel will not return to any panel that has previously appeared. Then, getHypothesis() is used to get the recognized word. The word is transferred into lower case and printed on the console. Continuously, a if-loop here checks if the recognized word is in ‘namelist’(line 51 to line 61). To make scripts more clear, an additional

method checkWord() is created from line 75 to line 81(Figure.8). The underlying logic of the if-loop is that if the recognized word is in the namelist, print out "GOT THE WORD", show panel5, and play a matching sound; if the word is not in namelist, print out "NO SUCH WORD". After running for the first time, panel6 comes out to say "try again", then the core part stars over when new speech is received.


```
28      try {
29          // Configuration Object
30          System.out.println("loading...");
31          Configuration configuration = new Configuration();
32          configuration.setAcousticModelPath("resource:/edu/cmu/sphinx/models/en-us/en-us");
33          configuration.setLanguageModelPath("/Users/ankiegao/eclipse/java-2019-03/javaProject/5206.lm");
34          configuration.setDictionaryPath("/Users/ankiegao/eclipse/java-2019-03/javaProject/5206.dic");
35
36          LiveSpeechRecognizer recognize = new LiveSpeechRecognizer(configuration);
37          recognize.startRecognition(true);
38          System.out.println("loading complete!");
39
40          w.changePanel2(); //panel3=speaking
41          SpeechResult result;
42
43          while ((result = recognize.getResult()) != null) {
44              w.panelContent();
45
46             //Get recognized speech
47              String word = result.getHypothesis();
48              word = word.toLowerCase();
49              System.out.println(word);
50
51              if(checkWord(namelist, word)==true) {
52                  outcome = word;
53                  System.out.println("GOT THE WORD");
54                  w.changePanel4(); //panel5=playing sound
55
56                  p.animal = outcome; //play sound
57                  p.sound();
58
59              }else if (checkWord(namelist, word)==false) {
60                  outcome = null;
61                  System.out.println("NO SUCH WORD");
62
63              w.changePanel5(); //panel6=try again
64
65              }
66          }
```

Figure. 7

```
75      public static boolean checkWord(String[] namelist, String value) {
76          for (int n = 0; n < namelist.length; n++) {
77              if (value.contentEquals(namelist[n])) {
78                  return true;
79              }
80          }
81          return false;
82      }
83
84  }
```

Figure. 8

Discussion

The programme runs and plays matching sounds as expected, one word at a time. However, there are still limitations such as over-recognition, under-recognition, and wrong-recognition. For example, the programme recognizes a word that is not in the namelist as a namelist word. Alternatively, when people speak a word that is actually in the namelist, the programme fails to recognize it. In addition, it also happens that a certain word is wrongly recognized as another word, such as ‘dog’ is frequently recognized as ‘donkey’. There are various methods which can fix these problems to some degree, including optimize dictionary and model, or deeply train the models.