

Component & Interfaces

Terrence Lewis

Presenting 101 (P101) is run locally on using two separate programs one for visual processing (gestures, relational movements & posture) and the other for audio processing (speech rhythm, disfluency and bad word detection & volume).

Note: much of the UI has not been implemented and all figures depicting the UI are mockups.

When the program starts it opens with the following screen.

Welcome to Presenting 101

What you should know:

The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.

Do nothing to use system defaults

Bad words:

Enter

Exit

Start

The Welcome Screen contains information regarding how to use P101, including optimal speaking distances, what to expect when users start the program, what P101 is checking for and any inputs the bad words input field. The user has the option to have the program to check for their specific “bad words” or they can leave it blank and it will use the system defaults. Making the use of Java’s GUI API dialogue boxes like the above will be shown which would look something like below (a modified Java dialogue class)

```

private String btnString1 = "Enter";
private String btnString2 = "Exit";
private String btnString3 = "Start";

/**
 * Returns null if the typed string was invalid;
 * otherwise, returns the string as the user entered it.
 */
public String getValidatedText() {
    return typedText;
}
/** Creates the reusable dialog. */
public CustomDialog(Frame aFrame, String aWord, DialogDemo parent) {
    super(aFrame, true);
    dd = parent;

    magicWord = aWord.toUpperCase();
    setTitle("Welcome Screen");

    textField = new JTextField(20);
    //Create an array of the text and components to be displayed.

    Object[] array = {msgString1, msgString2, textField};

    //Create an array specifying the number of dialog buttons
    //and their text.
    Object[] options = {btnString1, btnString2, btnString2};

    //Create the JOptionPane.
    optionPane = new JOptionPane(array,
                                JOptionPane.QUESTION_MESSAGE,
                                JOptionPane.YES_NO_OPTION,
                                null,
                                options,
                                options[0]);

    //Make this dialog display it.
    setContentPane(optionPane);
}

```

The bad words input makes use of a dictionary text file to make sure that input is valid. If this were programed this would look something like the following:

```

class badWordCheck {
    public static boolean checkWordd(String w) {
        try {
            BufferedReader in = new BufferedReader(new FileReader(
                "american-english"));
            String str;
            while ((str = in.readLine()) != null) {
                if (str.indexOf(w) != -1) {
                    return true;
                }
            }
            in.close();
        } catch (IOException e) {
        }
        return false;
    }

    public static void main(String[] args) {
        System.out.println(checkWordd("test"));
    }
}

```

If the user clicks the Exit button the program will close and if start is pressed the Welcome screens goes away and the following dialogue appears alerting users to prepare by getting into place.

The numbers will count down (not printing the way it appears above).

```
int countdown = 15000;
timer = new Timer(countdown, this);
JOptionPane.showMessageDialog(frame,
    "Please get in place the program will start in" + timer + "
seconds",
    "Notification Message",
    JOptionPane.PLAIN_MESSAGE);
if timer==0 close dialog box
    nextPart();
```

Once the countdown completes in the notification box the following box runs concurrently with the other two programs.

When the user is able to start presenting, the program is constantly listening to see when it should end through the use of the “Done” button.

```
Object[] options = {"DONE!"};
int n = JOptionPane.showOptionDialog(frame,
    "Press done to finish presenting ",
    "End program?",
    JOptionPane.YES_NO_CANCEL_OPTION,
    JOptionPane.QUESTION_MESSAGE,
    null,
    options,
    options[0]);
if Done==isPressed
    WriteResults();
```

Once the “Done” button is pressed the program stops recording, starts processing data, and it print out to a user error report (UER). The UER has a breakdown of the six things that P101 look for in determining what was wrong with someone’s presentation. This screen contains information from the two programs written to different parts of the dialogue box. When the user is done looking at that their information they can press done reviewing, which will close the program completely.

Algorithms

1. Generate recommendations
 - a. Volume
 - i. Input: Audio data
 - ii. Output: Average audio volume (using decibel level), graph of

progression of volume depicting trend over the course of the speech and recommendations.

- b. Disfluency
 - i. Input: Bad words (optional text), audio data
 - ii. Output: Bad word(s) said, disfluencies made and the number of times for each and recommendations.
- c. Rhythm
 - i. Input: Audio data
 - ii. Output: Speed and awkwardness (if applicable) and recommendation
 - 1. This should be able to analyze words per minute by being able to do this can detect someone's flow of speech.
 - a. Be able to take that data and know if something is "awkward" for a speech
 - b. Should also look for varied speech rhythms and be able to determine the difference between awkwardness and being a good presenter.
- d. Gestures
 - i. Input: Visual data
 - ii. Output: Where gestures went wrong and recommendations
 - 1. Must be able to detect when arms/hands go outside of the "frame".
- e. Relational movements
 - i. Input: Visual data
 - ii. Output: How much (if any) someone walked around and recommendations
 - 1. Must be able to know when a user is actually walking around
- f. Posture
 - i. Input: Visual data
 - ii. Output: If the user was leaning, swaying, etc and recommendations
 - 1. Must be able to know when users are leaning and swaying or just making minor adjustments

External API

- Java JOptionPane for dialogue boxes

User Interface

Below is the flow of screens that a user will see when they use P101. The action performed can be seen with the use of arrows that move from screen to screen.

What you should know:

The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.

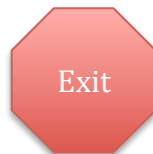
Do nothing to use system defaults

Bad words:

Enter

Exit

Start



Please get in place the
program will start in 15
sec... 14 sec ... 13 sec ...



No user interaction
required



Press done to
finish presentation

Done



U.E.R

Audio

Disfluencies

Bad words said:
_____ x's

Disfluencies:
_____ x's

Recommendation:

Rhythm


Rhythm: *info*

Bad words said:
_____ x's

Recommendation:

Volume

Average Volume: ____ dBs



Recommendation:

Visual

Gestures

When out of frame: _____ times

Recommendation:

Relational

Paced _____ times

Recommendation:

Posture

Action performed *swayed, etc*

Recommendation:

Done Reviewing

