Maxwell: The Face
Dr. Roppel
ELEC 5530
12/02/19

This report will be structured around the python code written for the digital face designed for Maxwell, a safety administrator robotic animatronic figure. As a robot that will depend on communication through voice and movement, it was important for us to develop a face that would accurately depict motion, specifically in the lips, according to a sound file. This report will be broken into 8 parts, each representing a fraction of the python code. The separate parts will be described in an order that does not correspond with that of the actual code, but of an order that is most understandable to a new reader.

updateface()

First, it has been defined that the outputted face would be determined by a user defined function called ‘updateface’ as seen below:

```
def updateface(color,lb,rb,ud,lr,m):
    pixel(BL,1,1,32,32)
    if ud==-1 or m==4:
        lb=1
        rb=1
```

The parameters for ‘updateface’ are simply listed, from left to right, as the color of the outputted pixels, the position of the left brow (lb), right brow (rb), vertical eye position (up-down or ud), horizontal eye position (left-right or lr), and mouth (m). Now as for the expected values to be passed for each parameter, the results are as follows: The eyebrow parameters expect an inputted ‘0’ for normal, ‘1’ for raised. The eye position expects a ‘-1’ for both up or left, depending on which parameter is at hand (ud or lr), while ‘0’ would neutral, and ‘1’ would be down/right. The mouth expects values 0-3.

```
#0 for normal, 1 for raised
    lbrw(color,lb)
    rbrw(color,rb)
    # ud/lr (-1 is up/left)
    leye(color,ud,lr)
    reye(color,ud,lr)
    #mout expects 0-3
    mout(color,m)
    p.display.update()
```

lbrw()

Honing in on the defined function 'lbrw()', this function virtually draws the rectangular shape that is the left eyebrow. The snippet of code can be seen below:

```
def lbrw(color,upvalue):#-------------------left eyebrow
    pixel(color, 8, 6-upvalue, 7, 1)
```

Simply put, all that is required for this function is a color and an 'upvalue'. As stated before, the eyebrows expect either an inputted '0' or '1'. What follows in the code is a function called 'pixel()'. This function creates the actual rectangular shape, and will be explained in further detail in the next part of this report. Take note as to how the 'upvalue' acts as a changing variable within one of the parameters for 'pixel()'.

pixel()

As previously mentioned, the 'pixel()' function is a user defined function that takes advantage of python's "pygame" package. This function creates rectangular shapes based off of simple parameters as seen below:

```
def pixel(color,x,y,w,h):
    p.draw.rect(screen, color, (x*20-19,y*20-19,w*20,h*20),0)
```

The parameters required from 'pixel()' are color, x-position, y-position, width of the rectangle, and the height of the rectangle. Passed on to one of the default "pygame" functions 'p.draw.rect()', the inputted parameters were scaled to our convenience. This function works off of the size of literal pixels. Since we are using a 32x32 LED array screen, it was important for us to enlarge that simulated area over a size that would be greater than just 32x32 pixels. The scale can be seen within the x, y, w, and h parameters. The single pixels were scaled up by x20, making each real life LED equivalent to 20 on screen pixels. You may have noticed that the x and y parameters have 19 subtracted from them. This is due to the fact that we needed to define the "beginning" of a 20x20 scaled pixel on the top left point of that square. By subtracting 19, you shift the x and y's point of reference to that top-left location.

rbrw()

Now that we understand how and why the 'pixel()' function works, we can further understand the defining code that creates each part of Maxwell's face.

```
def rbrw(color,upvalue):#-------------------right eyebrow
    pixel(color,19, 6-upvalue, 7, 1)
```

Identical to the left eyebrow, the right eyebrow a color and an 'upvalue'. The 'upvalue' will be pushed to the 'pixel()' function, where it dictates the vertical position of the eyebrow. As for the other parameters, the right eyebrow will have constant values for its horizontal position 'x', width

‘w’, and height ‘h’. By observing the values in the code above, we can see that the right eyebrow is a 7x1 rectangle, shifted 19 pixels to the right, and ‘6-upvalue’ downwards.

leye() and reye()

As a more complicated shape, the left and right eye will include more ‘pixel()’ functions for each detail of the eye. Below is the code for the left eye:

```
def leye(color,ud,lr):#--------------------left eye
    pixel(color,10+lr, 8+ud, 4, 1)
    pixel(color, 9+lr, 9+ud, 1, 1)
    pixel(color,14+lr, 9+ud, 1, 1)
    pixel(color,8+lr, 10+ud, 1, 4)
    pixel(color,15+lr,10+ud, 1, 4)
    pixel(color,11+lr,11+ud, 2, 2)
    pixel(color, 9+lr,14+ud, 6, 1)
```

Using the same ‘pixel()’ function as before, the variable parameters would be applied to the ‘x’ and ‘y’ coordinates. Specifically, variable ‘lr’ (left/right) would be added on to the ‘x’ parameter, while variable ‘ud’ (up/down) would be added to the ‘y’ parameter. As a reminder, this function expects values -1 through 1. Any combination of those values would result in Maxwell’s eyes appearing to look around.

mouth()

The shape and function of the mouth was intuitively designed in order to efficiently save us from writing many lines of code. It was our end goal to draw 4 separate mouth shapes, with the purpose of displaying a specific one according to its corresponding case. Instead of coding 4 separate, predefined drawing of mouths 1-4, we took advantage of python’s ability to include if/else statements within function parameters.

```
def mout(color,state):#--------------------mouth
    pixel(color, 9,23,16, 1)
    pixel(color, 9+state,23+state, 1, 1)
    pixel(color,24-state,23+state, 1, 1)
    pixel(color, 9+(0 if (state-1) < 0 else (state-1)),23+(0 if (state-1)
< 0 else (state-1)), 1, 1)
    pixel(color,24-(0 if (state-1) < 0 else (state-1)),23+(0 if (state-1)
< 0 else (state-1)), 1, 1)
    pixel(color, 9+(0 if (state-2) < 0 else (state-2)),23+(0 if (state-2)
< 0 else (state-2)), 1, 1)
    pixel(color,24-(0 if (state-2) < 0 else (state-2)),23+(0 if (state-2)
< 0 else (state-2)), 1, 1)
    pixel(color,10+state,24+state, 14-(2*state), 1)
```

First, the upper lip of the mouth was made, as seen by the first ‘pixel()’ function. This upper lip would stay constant since all of the motion would be done by the lower lip. By using extra conditional statements, we were able to have the lower lip lower and compress to a smaller size. At the same time, diagonal pixels would shift diagonally when called to. The single pixels responsible for creating the diagonal line “hide” behind the upper lip since python allows there to be overlapping drawings.

Quantization and animation of the .wav file

Now that have all of the possible displays for the face, and the ability to update the face with our function ‘updateface()’, we are able to work towards creating a talking head.

```
updateface(B,0,0,0,0,0)
rate, data = scipy.io.wavfile.read('wavs/mariah.wav')
sin_data = np.sin(data)
i=0
amp=0
while i<len(data):
    insound = (data[i][0]) if (i<9000) else ((data[i][0]
+data[i-1000][0]+data[i-2000][0]

+data[i-3000][0]+data[i-4000][0]+data[i-5000][0]

+data[i-6000][0]+data[i-7000][0]+data[i-8000][0]) / 9)
    if   abs(insound) >=0     and abs(insound) < 50 :
        m=0
    elif abs(insound) >=50  and abs(insound) < 200:
        m=1
    elif abs(insound) >=200 and abs(insound) < 500:
        m=2
    else:
        m=3
    if i%40000==0:
        ud=random.randint(-1,2)
        lr=random.randint(-1,2)
    if i%3000==0:
        updateface(B,0,0,ud,lr,m)
    i=i+1000
    sleep(.01515)
updateface(B,0,0,0,0,0)
sleep(5)
p.display.quit()
```

Simply put, the data of a .wav file was inputted and quantized into the 4 different levels that would determine which mouth drawing would display. As seen at the beginning of the ‘while’ statement, the inputted data was smoothed by averaging data points together. This smoothing process resulted in a better animation of the mouth. Below the quantized steps is the code that randomises the other facial features. The while loop is set to run as long as there's still data points left within the .wav file. Once all of the data is exhausted, the code will reset Maxwell’s face, and after 5 seconds, quit the program.