Smart Alarm

Jacob Siepker

Abstract

The goal of this project was to create an alarm clock that I would prefer to use over standard alarm clock packages. The original mockup included a system that could display time, be turned off by button switch, detect when the user woke up via pir motion detector, play white noise throughout the night, and correctly set of the alarm at the given time. These goals were accomplished with the one exception of the pir motion sensor which was not working properly. To replace this feature in the final version, I added a weather interface that routinely asks openweathermap.org for the Chicago weather and displays it on screen.

In creating an alarm that is better than typical, I also spend a lot of time making the interface look visually pleasing. I dove very deep into GUIZero documentation and ended up with a result I am happy with. During operation either the text or background (depending on current mode) are being continually hue shifted.

Over the last few nights, I have used the WIP version of this alarm clock to wake me up and it has been a scientifically better morning experience than my other alarms.

Project Description

I started the project by creating a simple python script that could get the current time and compare it for equality with a hard-coded alarm time to trigger an "alarm" state. The next thing I added was a set of audio files in 3 different folders: music, white noise, and alarms. I created 2 functions, playAudio() and getFile() which work together to play any of the three audio types with one call. For example, playAudio(Music) will find the absolute file location of the running .py file and work relative from their to find the music folder. It will pass the music folder to getFile() which counts the files and returns a random file that was not played last. I had to learn TinyTag for the next part, which finds the length of the audio file from its metadata and sets a timer for when it is done so it knows when to, and when to not play music.

Originally, I implemented a command line argument that opened the current audio file in a hidden VLC window. I had to abandon that method since the lines of code after the VLC command would not continue to run.

I then implemented a button interface, which I had to quickly comment out until the end since I was running to GPIO issues which have since been fixed. After that, I moved to the UI, which I should have started with. The Guizero package runs in an event based model (I learned after reading a good deal of the guizero docs) which means all my while loops in the main code had to be reworked into if statements in functions that the UI could call on a timer, ie every 500 milliseconds. Upon implementing the UI, I essentially had to redo all my code.

Once the UI was in place, I realized I wanted a few more options and added buttons to have no alarm, display output in a 12hr or 24hr mode, a no white noise mode so I quickly implemented those. The next few days were spent bug testing for several different conditions. The app had a lot of stability issues which I managed to get worked out.

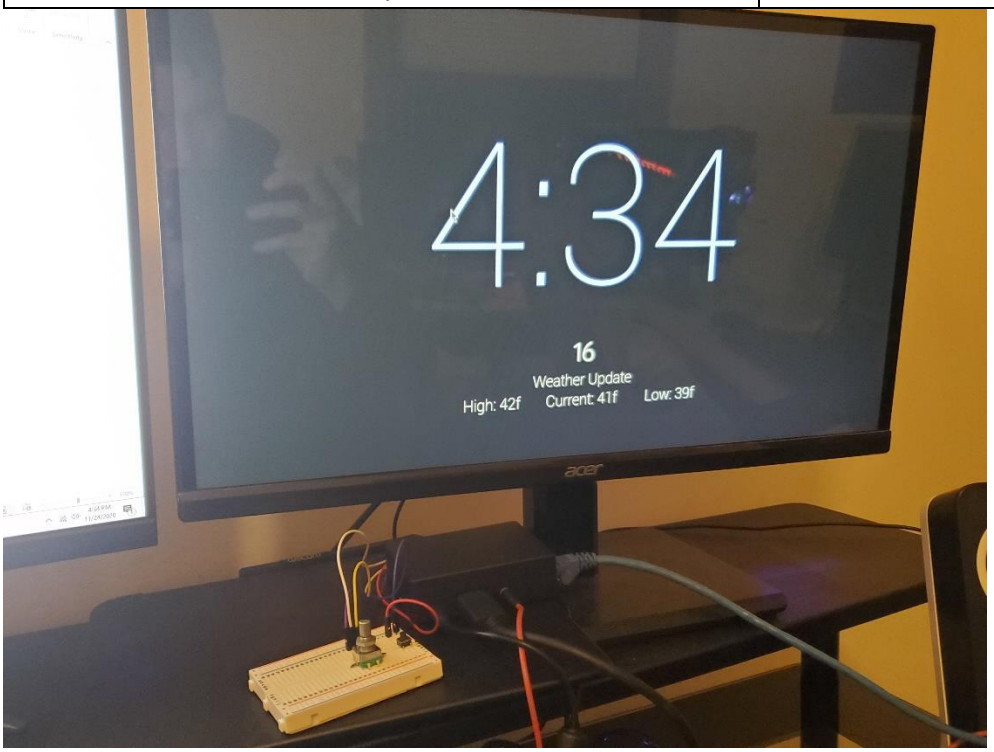
After the app was more stable, I started working on the aesthetics of the UI. I played around a lot of different fonts, text sizes, and color schemes until I found what I liked. I also added a pickColor() function which will slowly shift the hue for each call. It is called inside the UI's event-based system so the frequent calls make a subtle hue transition over time.

At this point, I was still having issues with the GPIO input so I signed up for office hours the last day. You suggested that I had something a bit more dynamic like an email system. Since I think it best fits my project, I decided instead to try implementing some weather API. I went with openweathermaps.org which can take python calls through the pyowm package. It took me a bit, but I ended up getting that interface working, which I am overall happier with than an email system since this fits cleanly into the alarm display. After the office hours meeting, I also added a rotary switch which controls the volume of the system.

The only feature I had to drop was the PIR motion sensor which was supposed to turn the alarm off when it sensed the user is out of bed. I scrapped this for two reasons, the first is that I was having trouble getting the part to work as it always triggered "motion" no matter if there was motion or not. The second is that I kind of like the alarm better without it. It plays music that I like to wake up to so if the alarm automatically turns off, so does the music I want to listen to.

Objectives

Objective	In Proposal	In Final
Create a clean UI	X	X
Implement a button toggle	X	X
Implement a PIR sensor	X	
Implement a rotary volume dial		X
Use public weather data		X
Play audio from a Python script	X	X
Utilize Time/Date functions	X	X
Create an audio file selection system		X



Recipes

From the textbook, I used 7.2 Formatting Time and Date, 7.7 Writing to a File, 7.12 Random Numbers, 7.14 Command-Line arguments in Python, 7.22 Creating User Interfaces (though most of my helpful material was from the GUIZero documentation), 12.1 Connecting a Push Swith, 12.2 Toggling with a Push Switch, 12.5 Debouncing a Button Press, and 12.7 Using a Rotary Encoder.

I also pulled a lot of information from other sources that helped me get my code working. The book doesn't cover PyGame, TinyTag, PyOWM, or Colorsys, all of which I had to learn on my own and are essential to my project. What I learned out of the book is what I expected to learn going into the project, the things that were very different were outside the scope of things we covered.

Class Experience

Overall, I had a very enjoyable time with this class and the final project. Coming into this class, I had heard a lot about Raspberry Pi's and now I feel confident that I can develop personal or mockup projects with it. It was a weird quarter with COVID-19 and having class in person, but it worked out in my opinion. I did about half of the labs from home and about half in person. I did run into some issues when I was working remotely but they were never anything too serious with the Pi.

For the final project, I am very relieved that it is done now and I have a final project that I can be happy with. I didn't know how ambitious I was being when I decided on this project since most aspects of it are things you don't think about having to worry about when you first think of the project. Now that it is done, I am very pleased with it. I really like the small feature I implemented the most, like the hue shift and second counter. Those two things were not planned to be implemented but once they were done, I knew they were features to stay. Overall, I think my program looks very clean which is exactly what I hoped for. I want something that I can throw up on a TV display that isn't ugly or distracting and this is it.

I wrote far more code than I expected to. The various helper functions really added a lot of lines to the code. In the end, it is about 400 lines of code (with whitespace) making this the longest, functional program I have ever written. I am also happy that even without the toolkit, I can still have nearly complete functionality of my alarm clock since all dials and input can be switched to keyboard commands instead of physical additions through the GPIO ports.

Python Code

```
from datetime import datetime
import time
from gpiozero import Button
import os
import pygame
from tinytag import TinyTag
from random import *
from guizero import *
from colorsys import hsv_to_rgb
import pyowm



### Switch, Pick, Update Functions
### Helpers for routine functions
################
def switchOff():
   global alarmStatus
   alarmStatus = not alarmStatus

def switch24H():
   global H24
   H24 = not H24

def setH(time):
   global alarmH
   alarmH = int(time)

def setM(time):
   global alarmM
   alarmM= int(time)

def switchPM():
   global pm
   pm = not pm

def switchMusic():
   global music
   music = not music
```

```
def switchWhiteNoise():
    global whiteNoise
    whiteNoise = not whiteNoise

def updateSecond():
    secondDisplay.value = int(time.time()%60)

#picks a background color, each call shifts hue by one degree
#Call in while loop for gradual color change
def pickColor():
    global hue

    hue = (hue+1)%360

    color = hsv_to_rgb(hue/360, 40/100, 100/100)
    R = int(color[0]*255)
    G = int(color[1]*255)
    B = int(color[2]*255)

    return (R,G,B)



def getTemp():
    global weatherDisplay
    global weather

    temp = weather.get_temperature(unit = 'fahrenheit')
    weatherDisplay.value = ('Weather Update\nHigh: ' + str(int(temp['temp_max'])) + 'f       Current: '+
str(int(temp['temp'])) + 'f       Low: ' + str(int(temp['temp_min'])) + 'f' )
    return

### For User Output, convert time format
def convert24h_12h(time):
    pm = False
    h = time[:2]
    m = time[3:]

    h = int(h)%12
    if (h == 0):
        h = 12
    if (int(h) >= 12):
        pm = True
```

```
    retVal = str(h)+':'+m
 #   if (pm):
 #       retVal += " pm"
 #   else:
#       retVal += " am"
    return retVal


def formatTimeSet(H, M, pm):
    #ensure all values in correct range, carry over if necessary
    H += int(M/60)
    M = M%60
    H = H%24

    if (pm == True and H != 12):
        H+=12

    returned = ''
    if (H == 0):
        H = 12
    if (H < 10):
        returned += '0'+ str(H)
    else:
        returned += str(H)
    returned += ':'
    if (M < 10):
        returned += '0'+ str(M)
    else:
        returned += str(M)

    return returned


#Gets current system time in user requested format
def getTime(H12=False):
    dt = datetime.now()
    output = "{:%H:%M}".format(dt)
    if (H12):
        return convert24h_12h(output)
    return output
```

```
#Counts files in path
#Picks random file from path and returns path/file
#Files MUST follow naming convention 01.mp3, 02.mp3... 99.pm3
def getFile(path):
    global pathLastPlayed
    global music
    global absolutePath

    fileCount = len(next(os.walk(path))[2])
    print('There are ' + str(fileCount) + ' files in path ' + str(path))
    theFile = randint(1, fileCount)
    print('Playing File No. ' + str(theFile) )
    returned = ''

    if (theFile < 10):
        returned += '0'
    returned += str(theFile)

    if (path == absolutePath+'music/' and path+returned == pathLastPlayed):
        theFile = ((theFile+1)%fileCount)+1
        if (theFile < 10):
            returned = '0'
        returned += str(theFile)

    return returned

#Gets folder location based on audioType request
#Sends folder to getFile to pick random (but not repeated) audio file
#Plays the selected audio file
#Audio file runs until it is over or until stopAudio() is called
def playAudio(audioType):
    global pathLastPlayed
    global audioEnd
    global absolutePath

    path = absolutePath

    if (audioType == 'Music'):
        path += 'music/'

    elif (audioType == 'WhiteNoise'):
```

```
        path += 'whiteNoise/'

    elif (audioType == 'Alarm'):
        path += 'alarms/'

    fileID = getFile(path);
    path+= fileID + '.mp3'

    audioEnd = time.time() + TinyTag.get(path).duration

    pygame.mixer.music.load(path)
    pygame.mixer.music.play()

    return


#Gracefully Ends Audio
def stopAudio():
    global audioEnd
    pygame.mixer.fadeout(1000)
    pygame.mixer.stop()
    audioEnd = 0
    print("Audio Stopped")
    return

#Checks for a signal from the button and/or motion sensor
def checkSignal(device = 0):
    #0 = button, 1 = vision sensor, 2 = both
    if (device == 0):
        return button.is_pressed

def get_encoder_turn():
    # return -1, 0, or +1
    global oldA, oldB
    global volUp, volDown
    result = 0
    newA = volUp.is_pressed
    newB = volDown.is_pressed
    if newA != oldA or newB != oldB :
        if oldA == 0 and newA == 1 :
            result = (oldB * 2 - 1)
        elif oldB == 0 and newB == 1 :
```

```
            result = -(oldA * 2 - 1)
    oldA, oldB = newA, newB
    time.sleep(0.001)

    return result

#main loop, checks alarm time with system clock, changes text and background color
#State depenedent on if statements, each call with run new section depending on time
def alarmOn():
    global alarmH
    global alarmM
    global pm
    global music
    global audioEnd
    global timeDisplay
    global whiteNoise
    global alarmTime
    global triggered
    global alarmStatus
    global alarmTime
    global H24

    global lightApp
    global welcomeMessage
    global timeDisplay
    global weatherDisplay
    global secondDisplay



    volKnob = get_encoder_turn()
    if (volKnob != 0):
        newVol = pygame.mixer.music.get_volume() + volKnob/10
        if (newVol > 1):
            newVol = 1
        elif (newVol < 0):
            newVol = 0
        pygame.mixer.music.set_volume(newVol)

    #Triggers alarm, changes display mode

    if (not triggered and alarmTime == getTime() and alarmStatus):
```

```
    print("Alarm Triggered")
    pygame.mixer.music.set_volume(1)
    triggered = True
    timeDisplay.text_color = 'black'
    secondDisplay.text_color = 'black'
    weatherDisplay.text_color = 'black'
    stopAudio()


#update time display
timeDisplay.value = getTime(not H24)


#White noise, not tiggered state
if (not triggered or not alarmStatus):

    timeDisplay.text_color = pickColor()
    secondDisplay.text_color = 'white'
    weatherDisplay.text_color = 'white'
    welcomeMessage.value = "Good morning, the time is"
    lightApp.bg = 'black'


    if (checkSignal(0)):
        print("Signal Recieved")
        whiteNoise = not whiteNoise
        if (not whiteNoise):
            pygame.mixer.music.set_volume(0)
        else:
            pygame.mixer.music.set_volume(0.2)
        time.sleep(0.2)


    elif (whiteNoise and audioEnd < time.time()):
        playAudio('WhiteNoise')

#Triggered state, play music or alarm, display ne winfo
#(not button.is_pressed) and
if(triggered and alarmStatus):
    lightApp.bg = pickColor() #update background color per cycle, creates gradual hue transition

    if (checkSignal(0)):
```

```
            pygame.mixer.music.set_volume(0)
            currentVolume = 0
            alarmStatus = False
            triggered = False
            print("Signal Recieved")
            whiteNoise = False
            stopAudio()
            time.sleep(0.2)

        elif (music and audioEnd < time.time()):
            playAudio('Music')
        elif (not music and audioEnd < time.time()):
            playAudio('Alarm')

        return False

    if (getTime() > formatTimeSet(alarmH, alarmM+5, pm)):
        exit()

    return True


##########################################################################################
########
#Global Vars, changed by functions and app
print("started")
alarmH = 12
alarmM = 00
alarmStatus = True  #If false, alarm off
pm = False
whiteNoise = False
music = False
H24 = False
absolutePath = os.path.abspath('AppliedComputingFinal.py')[:-24]
triggered = False
hue = 0 #used as global var to change background of time display

button = Button(18)

volUp = Button(23)
volDown = Button(24)
oldA = False
```

```
oldB = False



###First app, set params
app = App(title="Alarm", height = 500, width = 500, bg = '#dbf3ff')
message = Text(app, text='Set Your Alarm',font = "Quicksand", size = 30)
message = Text(app, text = "Hour",font = "Quicksand", size = 20)
sliderH = Slider(app, start=1, end = 12, command = setH, width= 400)
message = Text(app, text = "Minute",font = "Quicksand", size = 20)
sliderM = Slider(app, start=0, end = 59, command = setM, width = 400)
checkboxPM = CheckBox(app, text = "PM", command = switchPM)
checkboxPM.font = 'Quicksand'
checkboxMusic = CheckBox(app, text = "Musical Alarm", command = switchMusic)
checkboxMusic.font = "Quicksand"
checkboxWhiteNoise = CheckBox(app, text = "White Noise", command = switchWhiteNoise)
checkboxWhiteNoise.font = "Quicksand"
checkbox24H = CheckBox(app, text = "24H Time", command = switch24H)
checkbox24H.font = 'Quicksand'
checkboxAlarmOff = CheckBox(app, text = "No Alarm", command = switchOff)
checkboxAlarmOff.font = 'Quicksand'
message = Text(app, text = '')
buttonExit = PushButton(app, text = "All Set", command = app.destroy)
buttonExit.font = 'Quicksand'
buttonExit.bg = "#5ed6a2"

app.display()



#Get Weather info
apiKey = "1a04c39711a9a27878b48f9e65b7c8d8"
weatherObject = pyowm.OWM(apiKey)
obsObj = weatherObject.weather_at_place('Chicago, United States')
weather = obsObj.get_weather()



##Second app, night display and alarm
#########################
```

```
lightApp = App(title="Alarm", height = 1080, width = 1920, bg = (0,0,0))
welcomeMessage = Text(lightApp, text = "\nGood morning, the time is", size = 80, font = "Quicksand")
timeDisplay = Text(lightApp, text = getTime(), size = 350, font = "Piboto Thin", color = pickColor())
secondDisplay = Text(lightApp, text = time.time()%60, size = 50, color = (255,255,255), font =
"Quicksand")
weatherDisplay = Text(lightApp, text = '\n\n' , size = 30, font = "Piboto Thin", color = 'white')

lightApp.set_full_screen()


audioEnd = 0
pathLastPlayed = '' #save the path of the last audio file so nothing plays twice in a row

###initialize audio channel for use
pygame.mixer.init()
pygame.mixer.set_num_channels(1);
pygame.mixer.music.set_volume(0.2)

alarmTime = formatTimeSet(alarmH, alarmM, pm);


timeDisplay.repeat(100, alarmOn)
secondDisplay.repeat(20, updateSecond)
weatherDisplay.repeat(10000, getTemp)
lightApp.display()


#print('Pi thinks the date/time is:', getTime())
#print('The Alarm will go off at:', alarmTime)

stopAudio()
```