LESSON 18

FINAL PROJECT – ALARM

OBJECTIVE

In this lesson you will create a project that incorporates the camera, audio amplifier, speaker, and other components from previous levels, to create an alarm system capable of capturing images and playing an audio file when triggered.

MATERIALS

- Raspberry Pi connected to a monitor, keyboard, and mouse
- Circuit from Lesson 17, Activity #1 containing the RGB LED, OLED Display, and the Pushbutton Switch
- 3 x Long Male-to-Male Jumper Wires
- 8 x Short Male-to-Male Jumper Wires
- 1 x Audio Amplifier
- 1 x Speaker
- 2 x 1K-Ohm Resistors
- 1 x Ultrasonic Range Sensor
- USB Audio Adapter

REVIEW CONCEPTS

If you do not feel comfortable with the following concepts, please review them before proceeding.

- Global and Local Variables (Lesson B-2)
- Multithreaded Python Operations (Lesson B-12)
- Using the Ultrasonic Range Sensor (Lesson B-14)
- Working with the OLED Display (Lesson B-16)
- Creating GUI Windows using Tkinter (Lesson C-3)
- Controlling the Pi Camera (Lesson C-12)

LESSON

In this lesson you will build an alarm system that can be used to monitor an area while you're away. The circuit and program for this system will combine parts and skills from Levels A, B, and C of this program.

Due to the complexity of the circuit build and program required for this project, the teaching section of this lesson will be fairly limited. Instead, that section will be used to give a thorough explanation of the functionality that will be built into the project, how the components will interact, and how program elements will be used to control these components

PROJECT OVERVIEW

This system will detect the presence of someone near the Pi, which will trigger pre-programmed audio and video events. Here are the components in the circuit and how each will be used:

- Ultrasonic Range Sensor – Used for detecting someone near the system.
- Audio Amplifier and Speaker – Used for playing sounds when an object is detected.
- Camera – Used for capturing and storing image when someone is detected. When captured, the image will be displayed in the GUI window.
- OLED Display – Used for displaying armed/disarmed status and detection event details.
- RGB LED – Used as visual indicator of armed/disarmed status.

This may look like a lot of things to be working with in one project, but the circuits and software that will be used with these components will be borrowed from previous lessons. The Ultrasonic sensor, for example, will be using the same software module that you used for the final project in Level B. The audio portion will use the same wiring and commands as Lesson C-6. The camera capture and display software will just be a slightly modified version of the Tkinter program that you created in Lesson C-12. The OLED display and RGB LED software will also be very similar to programs you've created to interface with those parts in the past.

USER INTERFACE OVERVIEW

When the program is first launched, the user will see a Tkinter window containing a DISARMED image, an **`Arm`** button, a **`Disarm`** button, and a **`Quit`** button. The RGB LED will be green to reflect that the system is disarmed, and a **`DISARMED`** message on the OLED screen will reflect this as well. At this point the Ultrasonic sensor is not active.

The user clicking on the **`Arm`** button will cause the Ultrasonic sensor to begin taking distance readings. The RGB LED will change to red and the OLED display will show the message ARMED. The image displayed in the Tkinter window will be switched to one indicating the system is now armed and that there are no captured images to display.

If the system is armed, the alarm will be triggered any time an object comes within 50 centimeters of the Ultrasonic sensor. Once triggered, these events will occur:

- Display ALARM along with the current timestamp on the OLED display
- Play a pre-recorded sound file through the speaker
- Turn on the camera and capture an image
- Save the image in the /home/pi/Pictures/captures folder
- Display the captured image in the GUI window

After the trigger event is complete, the system will remain armed, and can once again be triggered by an object being within range of the Ultrasonic sensor.

The value of 50 centimeters for the trigger range was arbitrarily selected as a reasonable value for triggering the alarm. This distance can easily be adjusted to your liking once the program is built.

PROGRAM OPERATIONS

The GUI window of this program will run just like the camera image capture program you build in Lesson C-12. One big difference between that program and this one is that functions that were launched did not loop. When you pressed the button, a function would run that would capture the image, display it in the window, and return control back to the window.

That works great when the function you called doesn't loop but imagine what happens when you call a function that checks the range of the Ultrasonic sensor, over and over. The program will call that function, start checking the range, and never stop. This means that the GUI window is now unresponsive as the program is now only looping through the range checking function.

This can be solved by launching the function as a new thread like you did with the status LED of the RFID reader program in Lesson B-12. The LED flashing thread was launched and ran in parallel with the main program, so the LED flash timing did not delay the rest of the program. The same concept will be used in this program with the Ultrasonic sensor. The Ultrasonic sensor range checking function will be launched as a new thread whenever the system is armed. A **`while`** condition in this function will check the current status of a variable named **`armed`** each time the function loops. If the system is disarmed, the value of **`armed`** will no longer satisfy the **`while`** condition in the thread, and the range checking thread will end.

REQUIRED FILES AND MODULES

This program will rely on some additional files to accomplish all of the tasks above:

The **`Adafruit_Python_SSD1306`** module for driving the OLED display. This was installed onto your Pi in Lesson B-16.

The **`ultrasonic.py`** file from Lesson B-18 will be used as an imported module to obtain ranges from the Ultrasonic range sensor.

The ARMED and DISARMED placeholder images for the GUI window will be downloaded from the 42 Electronics Level C GitHub repository during the Activities section of this lesson.

A sound file that will be played when an alarm event is triggered. You will record a file to use for this purpose during the Activities section of this lesson.

A new folder will be created at **`/home/pi/Pictures/captures`** that will store the captured images. Another new folder named **`/home/pi/alarm`** will be created that will hold the ARMED and DISARMED images for the GUI window, the sound file, the ultrasonic.py module file, and the main program. These folders will be created later in the Activities section.

ACTIVITIES

In the following activities you will make changes to the circuit from Lesson 17, Activity #1 to add components required for a simple alarm system. You will also create folders and download image files from the 42 Electronics GitHub repository, as well as create the main alarm program.

ACTIVITY #1 – CIRCUIT MODIFICATIONS

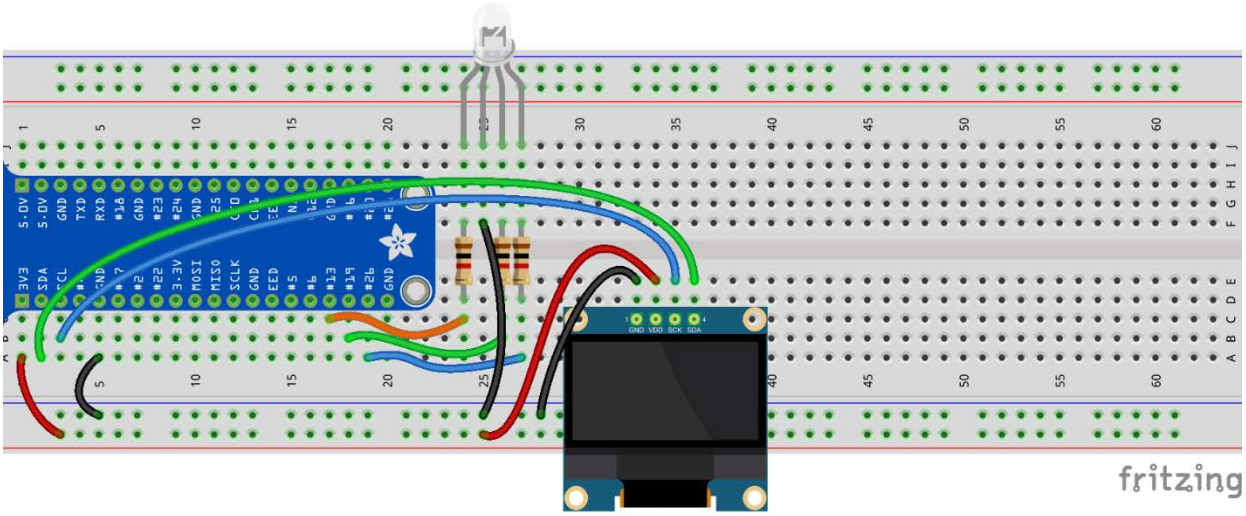
In this activity you will modify the breadboard circuit from Lesson 17, Activity #1 by removing the pushbutton switch, and adding both the audio amplifier with speaker, and the Ultrasonic sensor.

STEP #1

First, ensure the Raspberry Pi is safely powered off. Once the Pi is off, remove the following from the circuit:

- Pushbutton switch
- 10K-Ohm resistor
- Three jumper wires attached to the switch

Once these modifications are made, your breadboard will still contain the RGB LED, the OLED display, and associated their associated components:

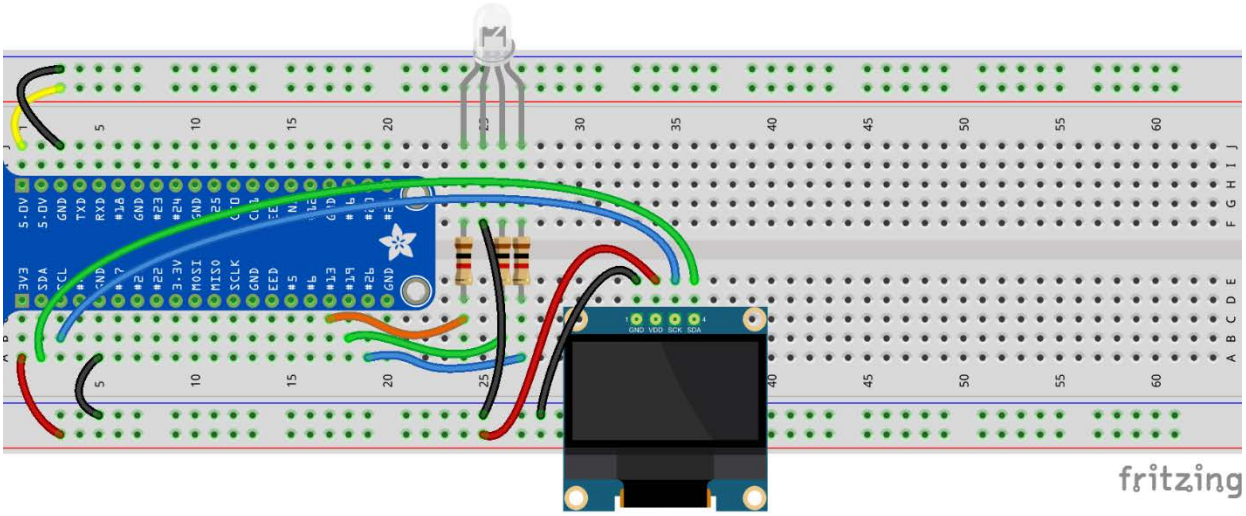


STEP #2

The Ultrasonic sensor and the audio amplifier will both need access to 5V power and ground, so let's make those available on the P2 and N2 power rails. Insert two short jumper wires between the breadboard locations below:

5V – Short Male-to-Male Jumper Wire – between J1 and P2-3

GND - Short Male-to-Male Jumper Wire – between J3 and N2-3



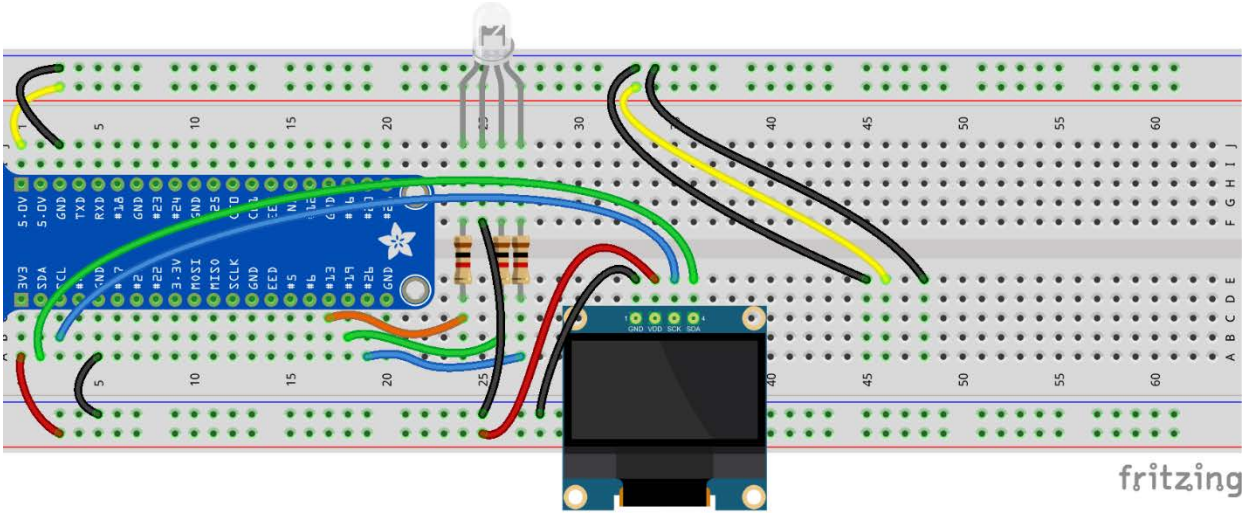
STEP #3

Next, you will install the power wiring for the audio amplifier which requires a single 5V connection and two ground connections: Insert three short jumper wires into the following locations:

5V - Short Male-to-Male Jumper Wire – between P2-33 and E46

GND - Short Male-to-Male Jumper Wire – between N2-33 and E45

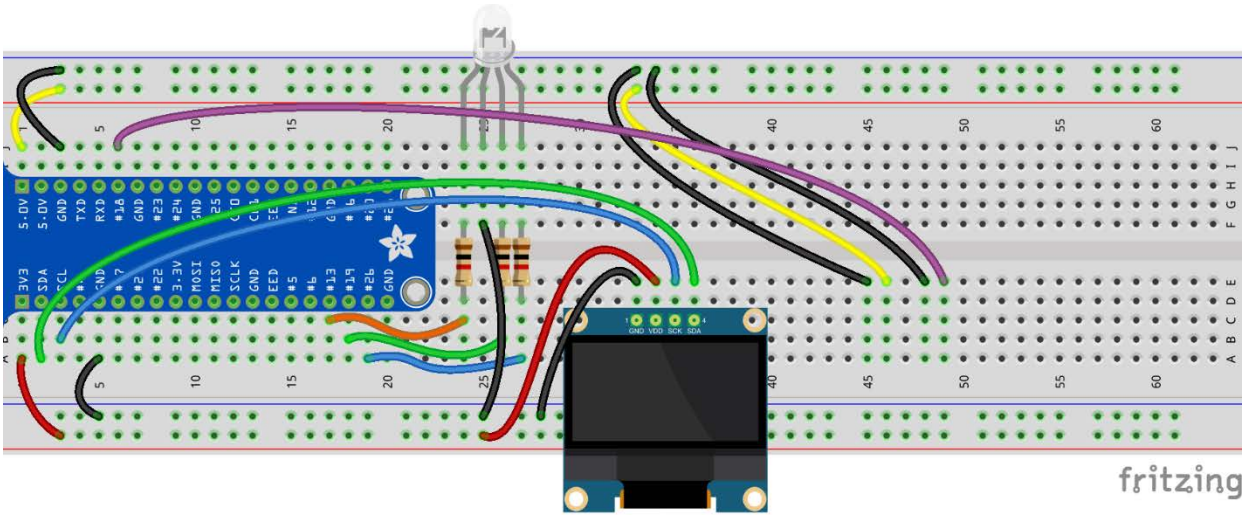
GND - Short Male-to-Male Jumper Wire – between N2-34 and E48



STEP #4

The audio amplifier needs one more connection and that's an audio input source from the Pi. Just like in Lesson 6, you will be using GPIO18 for sending audio to the amplifier. Use one long jumper wire to connect the following points:

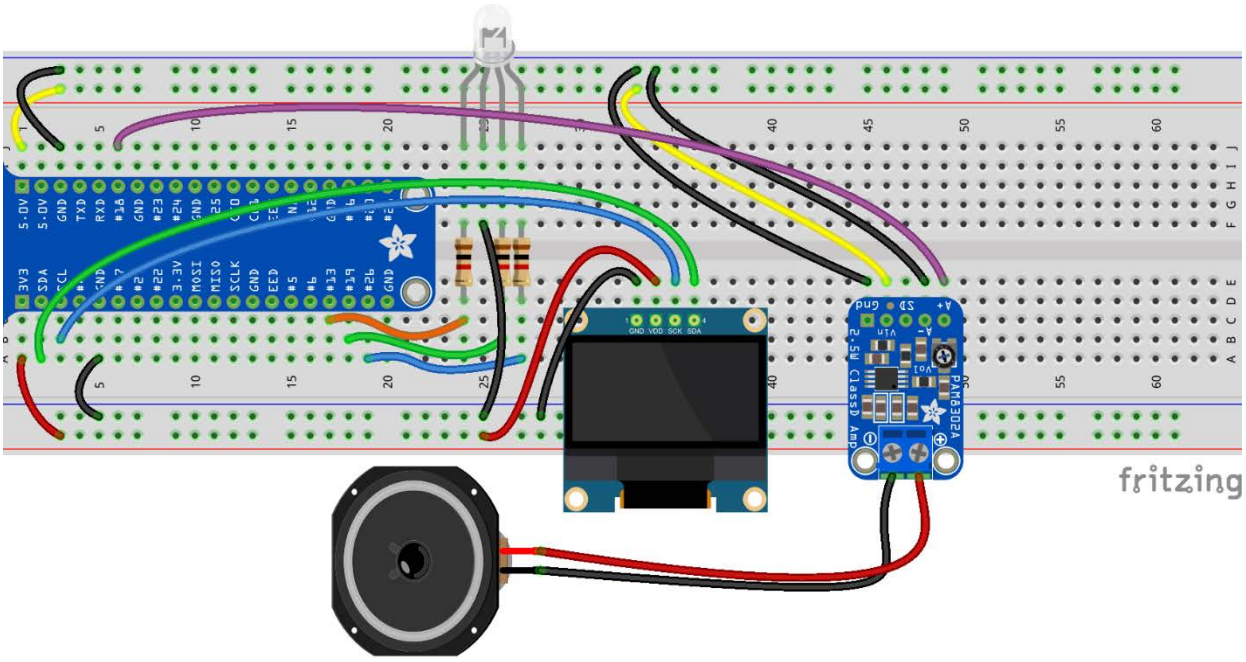
Audio signal - Long Male-to-Male Jumper Wire – between J6 and E49



STEP #5

The final step for the audio components will be to add the audio amplifier and speaker. Add the amplifier to the breadboard in the locations shown below:

Audio Amplifier – between pins C45 through C49, with Ground in C45 and A+ in C49



If the speaker is no longer attached to the amplifier, use the small, flat screwdriver to reconnect the speaker to the terminal block of the amplifier. For more information on this procedure you can reference Lesson 6 where the speaker and amplifier were first connected.

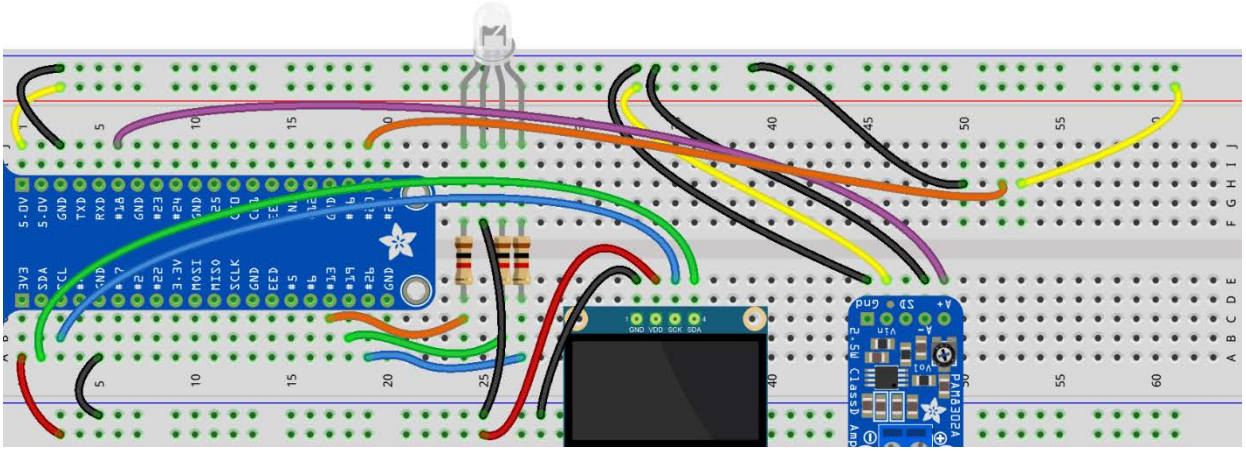
STEP #6

The last component that will be added to the circuit is the Ultrasonic sensor. In this step you will add input power, ground, and the trigger connection for the Ultrasonic sensor. Make the following connections below on your breadboard:

5V - Short Male-to-Male Jumper Wire – between P2-61 and H53

GND - Short Male-to-Male Jumper Wire – between N2-39 and H50

GPIO20 - Long Male-to-Male Jumper Wire – between J19 and H52



STEP #7

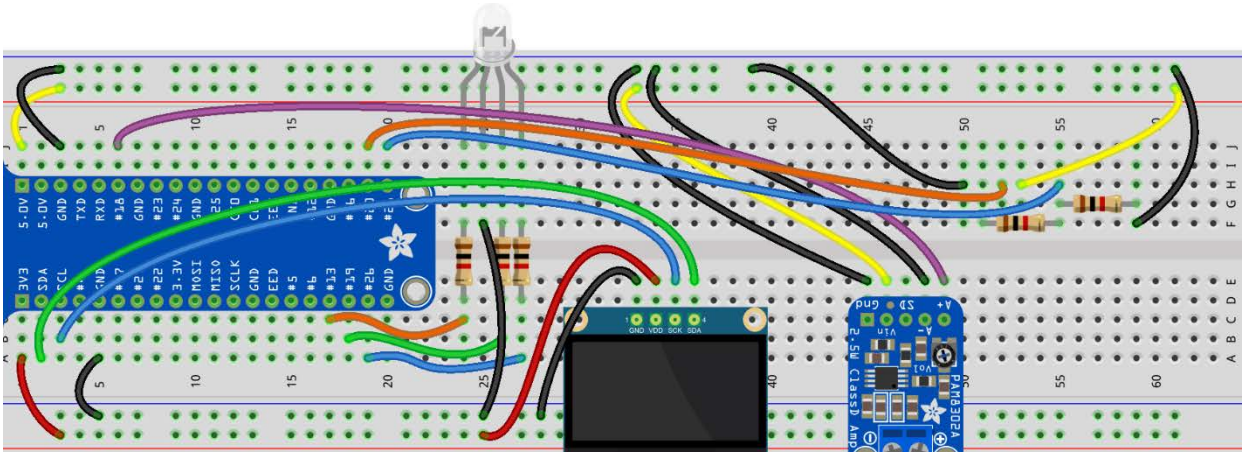
The Echo output of the Ultrasonic sensor is not safe to connect directly to a GPIO pin, so you will run the output through a voltage divider made up of two 1K-Ohm resistors that will cut the output voltage in half. Add two jumper wires and two 1K-Ohm resistors to the locations specified below:

GPIO21 - Long Male-to-Male Jumper Wire – between J20 and H55

GND - Short Male-to-Male Jumper Wire – between N2-61 and F59

1K-Ohm Resistor - between G55 and G59

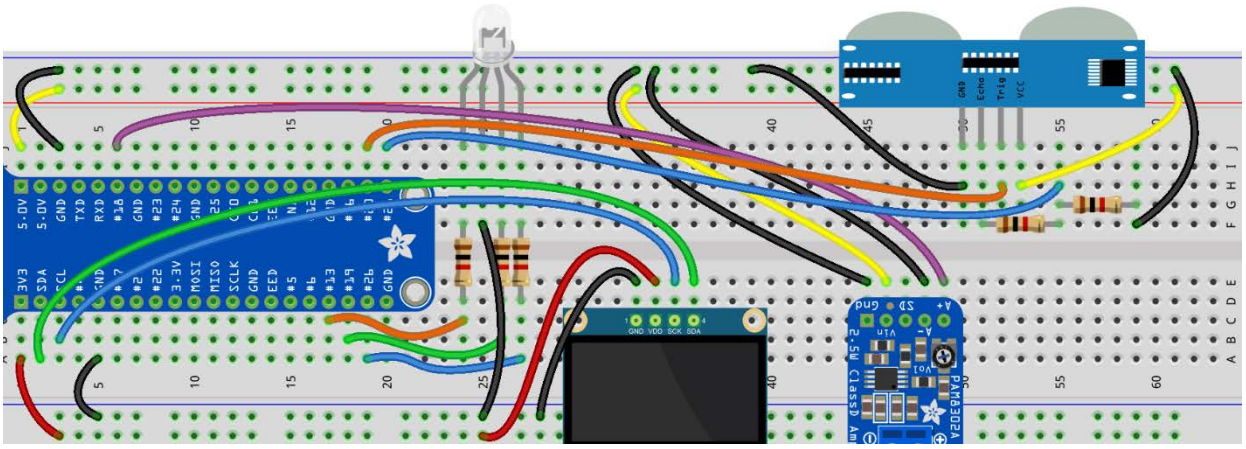
1K-Ohm Resistor - between F51 and F55



STEP #8

The last circuit modification will be to add the Ultrasonic sensor. Insert the Ultrasonic sensor into the breadboard holes below:

Ultrasonic Sensor – between J50 and J53, with the emitter/receiver portion of the sensor pointing away from the breadboard.



STEP #9

Double-check all the wiring with the image in the last step to ensure there are no connection or placement errors in your circuit. If everything looks correct, plug the USB audio device and power on the Raspberry Pi. The USB audio device will be used to record a sound file in the next activity.

ACTIVITY #2 – FILE AND FOLDER PREPARATION

In this activity you will create two folders and populate them with the files needed for the final program to function. The folder **/home/pi/Pictures/captures** will be used to hold all images that are captured by the alarm program. The folder **/home/pi/alarm** will used to hold program files.

STEP #1

The first step will be to create the captures directory inside the **/home/pi/Pictures** folder. Open a Terminal window and enter the following command:

```
cd Pictures
```

Since you're already located in **/home/pi**, this command will move you into the **Pictures** folder. Use the following command to create the new **captures** directory:

```
mkdir captures
```

The **captures** directory will be created inside the **Pictures** folder.

STEP #2

The next step will be to create a folder named alarm inside the **/home/pi** directory. Change your location back to the **/home/pi** directory with the following command:

```
cd ~
```

Now that you're back in the **/home/pi** directory, create the **alarm** folder with the following command:

```
mkdir alarm
```

Change into the new **alarm** folder by using this command:

```
cd alarm
```

STEP #3

Now that the alarm folder has been created, and you're in that location in the CLI, it's time to start getting the files you need into that folder.

First, download the two image files named **armed.png** and **disarmed.png** from the 42 Electronics Level C GitHub repository.

Please note that the next two commands won't fit on a single line, but each one should be entered as one continuous command in Terminal:

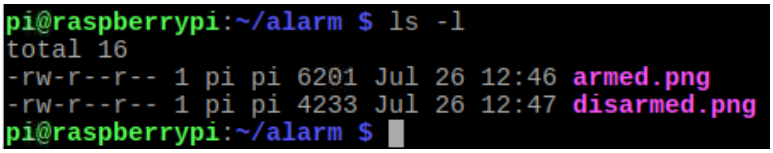
Download **armed.png** with the following command:

```
curl https://raw.githubusercontent.com/42electronics/level_c/master/lesson_18/armed.png > armed.png
```

Download **disarmed.png** with the following command:

```
curl https://raw.githubusercontent.com/42electronics/level_c/master/lesson_18/disarmed.png > disarmed.png
```

Run the **ls -l** command to make sure the file sizes are **6201** for **armed.png** and **4233** for **disarmed.png**:



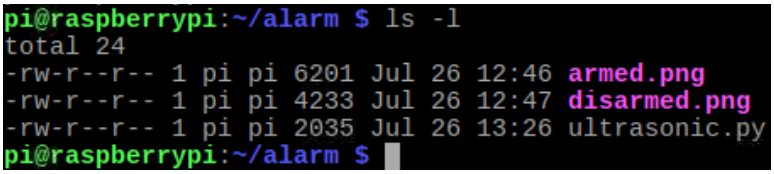
If the file sizes are below 20, there was most likely a typo in your curl command and the file was created but is empty. If this is the case, rerun the command for the incorrect file, ensuring that the command matches those above exactly, and the good file will overwrite the bad version. Use **ls -l** again to confirm your file sizes match those above before proceeding.

STEP #4

In addition to the image files you just downloaded, you will also need the ultrasonic.py that you used for the final project in Level B. This file may still be on your Desktop, but since that was many lessons ago, there is a copy hosted in the same GitHub folder. Use the command below in your existing Terminal window to download the file:

```
curl https://raw.githubusercontent.com/42electronics/level_c/master/lesson_18/ultrasonic.py > ultrasonic.py
```

Confirm your command downloaded the file properly by running another **`ls -l`** command to confirm the downloaded file size is **`2035`**:



If the file size does not match, run the command again, double-checking for any typing errors. Once you've confirmed the file size is **`2035`**, proceed to the next step.

STEP #5

The last piece you need is the audio file that will be played when a detection event occurs. While you could use your recorded voice file from Lesson 6, it might be more interesting to record a file that relates more directly to an alarm event. This sound will be played whenever the alarm is triggered, so a sound like "Warning" or “Step Away" would be more applicable, but the choice is up to you.

You will be using the same Terminal commands from Lesson 6 to record a new file. Record a new file called **alarm.wav** by using the following command. Remember to use CTRL-C to stop the recording:

```
arecord --device=hw:1,0 --format S16_LE --rate 48000 -c1 alarm.wav
```

Make sure you're happy with the recording by playing back the file with the following commands. First, configure GPIO18 to output audio:

```
gpio -g mode 18 ALT5")
```

Next, play the new recording with this command:

```
aplay alarm.wav
```

If you're not happy with the recording, use the **arecord** command again to record over the existing **alarm.wav** file, and use the **aplay** command to listen to your file. Once you're happy with the recording, use the following command to return GPIO18 to an input state:

```
gpio -g mode 18 in
```

You now have all the files that will be needed by the program.

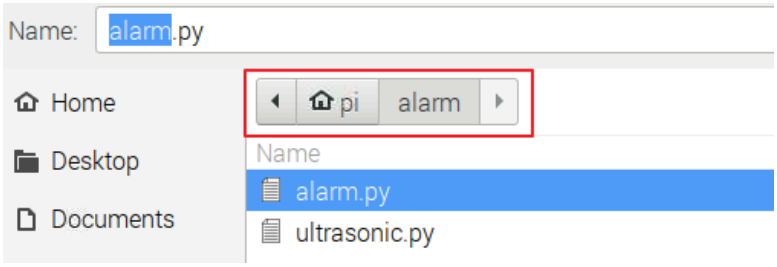
ACTIVITY #3 – CREATING THE ALARM PROGRAM

In this activity you will create the program for the alarm project. This program will be made using blocks of code from many previous lessons and programs. Since the functionality of each of these blocks were covered in previous lessons, this lesson won't go through every piece of code in each block, Instead, some steps will direct you to the lesson that contains more information about that block of code, which you can use for further reference if needed.

STEP #1

The first step will be to create a new program in Thonny. This program must be located in the /home/pi/alarm folder so it will have access to the files that you've previously prepared.

Open Thonny and create a new program. Use the File > Save As menu to save the file as **`alarm.py`** in the **`/home/pi/alarm`** directory. Make sure to use the navigation bar beneath the file name to save **`alarm.py`** in the proper directory:



STEP #2

The first block of the program will be the imports. There are quite a few since you are working with GPIO pins, the OLED display, Tkinter, os commands, the Ultrasonic sensor, and others.

Add this block to the beginning of your program:

```
import RPi.GPIO as GPIO
import Adafruit_SSD1306
from tkinter import *
from PIL import Image, ImageDraw, ImageFont
import time, os, ultrasonic, _thread
```

STEP #3

Next, you will set up some variables to hold values that will be used throughout the program:

red – This variable will hold the pin number of the red RGB element, or 13.

green – This variable will hold the pin number of the green RGB element, or 19.

sensitivity – This variable will hold the value in centimeters that will trigger an alarm event. This will initially be 50 but can easily be changed later.

armed – This variable will hold a 0 or 1, 0 indicates a disarmed state and 0 indicates an armed state.

Set up these variables by adding the highlighted block below to the end of your program:

```
import time, os, ultrasonic, _thread

red = 13
green = 19
sensitivity = 50
armed = 0
```

STEP #4

Since this program will interact with GPIO pins, the pin numbering mode must be declared, and red and green pins must be configured as outputs. Add the highlighted block of code below to the end of your program:

```
armed = 0

GPIO.setmode(GPIO.BCM)
GPIO.setup(red, GPIO.OUT)
GPIO.setup(green, GPIO.OUT)
```

STEP #5

It's now time for some the setup code required for the OLED display. This block of code is pulled directly from Lesson B-16. The only adjustment that's been made to the code is that the size of the font has been adjusted to **24** because we're displaying less text on the screen, so the characters can be larger.

Add the highlighted block of code below to the end of your program:

```
GPIO.setup(green, GPIO.OUT)

disp = Adafruit_SSD1306.SSD1306_128_64(rst=None)
disp.begin()
width = disp.width
height = disp.height
image = Image.new('1', (width, height))
draw = ImageDraw.Draw(image)
font =
ImageFont.truetype('/usr/share/fonts/truetype/freefont/FreeSans.ttf',24)
```

STEP #6

Since you will be using Tkinter to build a GUI window to display captured images and the arm/disarm buttons, you will need an alias to use for referring to the window. You will use **root** for the Tkinter alias. and you will also use this section to assign a window title of **Alarm System**. Add the following highlighted block of code to the end of your program:

```
font = ImageFont.truetype('/usr/share/fonts/truetype/freefont/FreeSans.ttf',24)

root = Tk()
root.title('Alarm System')
```

For more information on these commands you can refer back to Lessons C-3 and C-12.

STEP #7

The next block will be a function that contains the commands required to play your pre-recorded sound file through the amplifier and speaker connected to GPIO18. This code is very similar to the code used in Lesson C-6, except now it will run inside of a function named **audio_alert()** and it will play a file named **alarm.wav**. Add the highlighted block of code below to the end of your program:

```
root.title('Alarm System')

def audio_alert():
    os.system("gpio -g mode 18 ALT5")
    os.system("aplay /home/pi/alarm/alarm.wav")
    os.system("gpio -g mode 18 in")
```

STEP #8

The next function will update the red and green elements of the RGB LED. This is the same function that you've used in many other programs, except that blue has been removed since it will not be used in this program. Add the highlighted block of code below to the end of your program:

```
    os.system("gpio -g mode 18 in")

def led_update(red_value,green_value):
    GPIO.output(red, red_value)
    GPIO.output(green, green_value)
```

Since blue has been removed from this function, the red and green values are all that need to be specified when calling the function:

- led_update(1,0) red on and green off
- led_update(0,1) red off and green on
- led_update(0,0) red off and green off

STEP #9

The function you will add in this step will update the contents of the OLED display. This version is slightly different than the function you've used in the past, as **line1** and **line2** have been added as attributes. This works just like the **led_update()** function that allows you to specify whether an LED element is on or off by sending a **0** or **1** for that position. The same can be done with the lines on the display. By sending two strings when calling this function, those strings can be displayed on the OLED. The content in **line1** will be displayed at Y position 0 of the screen and **line2** will be displayed at Y position 22.

Add the highlighted block of code below to the end of your program:

```
    GPIO.output(green, green_value)

def display_update(line1,line2):
    draw.rectangle((0,0,width,height), outline=0, fill=0)
    draw.text((0, 0), line1, font=font, fill=255)
    draw.text((0, 22), line2, font=font, fill=255)
    disp.image(image)
    disp.display()
```

For more information on displaying information on the OLED display, please refer to Lesson B-16.

STEP #10

This next function comes directly from Lesson C-12 where photos were captured and displayed in a Tkinter window. The only modification to this function is the folder where captured images will be stored. In this alarm program, the images will now be stored in the **/home/pi/Pictures/captures** folder.

Add the highlighted block of code below to the end of your program:

```
    disp.display()

def update():
    global img
    timestamp = (time.strftime('%Y-%m-%d_%H:%M:%S'))
    img_file = ('/home/pi/Pictures/captures/%s.png' % timestamp)
    os.system('raspistill -o %s -e png -w 640 -h 480 -t 1500' % img_file)
    print('Image saved as %s' % img_file)
    img = PhotoImage(file='%s' % img_file)
    Label(root, image=img).grid(row=0, column=0)
```

For more information on this image capture function, please refer to Lesson C-12.

STEP #11

This next function named **`proximity()`** will keep monitoring the distance from the Ultrasonic sensor as long as the value of **`armed`** equals 1. Once **`armed`** equals 0, looping will stop, and the function will end.

While armed, if the distance falls below the **`sensitivity`** value (50cm), the OLED screen will be updated with an **`ALARM`** message with the current time, the **`audio_alert()`** function will be launched to play the sound, and a function named **`update()`** will be called to capture an image. A **`time.sleep`** of **`0.1`** is added to free up system resources between distance checks.

Add the highlighted code below to the end of your program:

```
    Label(root, image=img).grid(row=0, column=0)

def proximity():
    while armed == 1:
        distance = ultrasonic.average()
        if distance < sensitivity:
            timestamp = time.strftime('%H:%M:%S')
            display_update('ALARM at ', timestamp)
            _thread.start_new_thread(audio_alert, ())
            update()
        time.sleep(0.1)
```

The ultrasonic code was used in Lesson B-14, and the **`timestamp`** and **`new_thread`** code were both used in Lesson B-12.

STEP #12

This function will define what action will occur when the system is armed by clicking a button in the Tkinter window. Here is a full list of the actions that will occur in this function:

1. The **armed** and **img** variables are pushed to the global scope.
2. The value of **armed** is updated to **1.**
3. The **led_update** function is called with arguments to turn the LED red.
4. The value of img is updated to the **armed.png** image so it can be displayed in the Tkinter window at grid location row 0, column 0.
5. The OLED display is updated with **ARMED** for **line1**.
6. The **proximity()** function is launched as a new thread and will start monitoring the distances returned from the Ultrasonic sensor.

Add the highlighted code below to the end of your program:

```
        time.sleep(0.1)

def arm():
    global armed
    global img
    armed = 1
    led_update(1,0)
    img = PhotoImage(file='/home/pi/alarm/armed.png')
    Label(root, image=img).grid(row=0, column=0)
    display_update('ARMED','')
    _thread.start_new_thread(proximity, ())
```

STEP #13

This next function will define what action will occur when the system is disarmed by clicking a button in the Tkinter window. Here is a full list of the actions that will occur in this function:

1. The **armed** and **img** variables are pushed to the global scope.
2. The value of **armed** is updated to **0.**
3. The **led_update** function is called with arguments to turn the LED green.
4. The value of img is updated to the **disarmed.png** image so it can be displayed in the Tkinter window at grid location row 0, column 0.
5. The OLED display is updated with **DISARMED** for **line1**.

Add the highlighted code below to the end of your program:

```
    _thread.start_new_thread(proximity, ())

def disarm():
    global armed
    global img
    armed = 0
    led_update(0,1)
    img = PhotoImage(file='/home/pi/alarm/disarmed.png')
    Label(root, image=img).grid(row=0, column=0)
    display_update('DISARMED','')
```

STEP #14

This function will define the action that will occur when the Quit button is pressed in the Tkinter window. This function will raise **`SystemExit`** so the **`except:`** code at the bottom of the program can clean everything up before exiting the program (the **`except:`** block will be added later).

Add the highlighted code below to the end of your program:

```
    display_update('DISARMED','')

def quit():
    raise SystemExit()
```

STEP #15

The functions have all been defined and now it's time for the main program block. Here are the actions that will be happening on the **`try:`** block of this program:

1. Run the disarm() function to get the RGB LED, OLED display, and the Tkinter window image in the proper state.
2. Place the image held by **`img`** into the Tkinter window at grid location row 0, column 0.
3. Place the Arm button into the Tkinter window at grid location row 0, column 1.
4. Place the Disarm button into the Tkinter window at grid location row 0, column 2.
5. Place the Quit button into the Tkinter window at grid location row 0, column 2.
6. Specify that the X close button in the Tkinter window will run the **`quit()`** function when clicked.
7. Start the Tkinter window named **`root`** with the **`mainloop()`** command.

Add the highlighted code below to the end of your program:

```
    raise SystemExit()

try:
    disarm()
    Label(root, image=img).grid(row=0, column=0)
    Button(root, text="Arm", command=arm, width=5).grid(row=1, column=0)
    Button(root, text="Disarm", command=disarm, width=5).grid(row=2, column=0)
    Button(root, text="Quit", command=quit, width=5).grid(row=3, column=0)
    root.protocol("WM_DELETE_WINDOW", quit)
    root.mainloop()
```

Tkinter window operations were covered in Lessons C-3 and C-12. Please refer to those lessons for questions regarding the commands in the block of code above.

STEP #16

The last block of the program will specify the actions that will happen when a **KeyboardInterrupt** is encountered or a **SystemExit** is raised. Here is the list of actions that will take place if one of these exceptions occurs:

1. The **led_update** function is called with arguments to turn both LED elements off.
2. The window named **root** is destroyed.
3. The text being shown on the OLED display is cleared.
4. A **GPIO.cleanup()** will return all GPIO pins back to their default state.

Add the highlighted code below to the end of your program:

```
    root.mainloop()

except (KeyboardInterrupt, SystemExit):
    led_update(0,0)
    root.destroy()
    disp.clear()
    disp.display()
    GPIO.cleanup()
```

Lesson B-16 has additional information about the **disp.clear()** and **disp.display()** lines of code used above.

STEP #17

The program is now complete and ready to run. Run the program in Thonny and the GUI window will display the DISARMED image along with the Arm, Disarm, and Quit buttons.

Click the Arm button and the system will display the ARMED image, the LED will turn red, and the OLED will indicate the system is armed.

Wave your hand within 50 centimeters of the Ultrasonic sensor and the system will be triggered. The OLED display will show time of the alarm event, your pre-recorded sound file will play through the speaker, and the camera will capture an image, store the image in **`/home/pi/Pictures/alarm`**, and display the image inside the Tkinter window.

The system will remain armed after a capture. Placing your hand in front of the Ultrasonic sensor will result in the OLED alert being updated, the audio file playing again, and another image being captured, stored, and displayed.

When you're done capturing images, you can use the Disarm button to disarm the system or use the Quit button to quit the program. You can review any images that were captured by the alarm system using File Manager to browse the contents of **`/home/pi/Pictures/alarm`**.

STEP #18 – TROUBLESHOOTING AS NEEDED

If the program does not work as expected, try to narrow the problem down to one area:

If the Ultrasonic sensor is not reacting to objects in front of it, run the **`ultrasonic.py`** program from Lesson B-14 to determine if it's wired properly. If it gets ranges using that program, then check the contents of the **`proximity()`** function as an error in that code could keep close objects from triggering alarm events.

If a button in the program window is not behaving as expected, check that the button is calling the right function, and that the function it's calling contains the correct code.

The full contents of this program will be posted below, but if you're still having trouble getting your program running, you can download a copy of the program from the 42 Electronics Level C GitHub repository by running the following command in the CLI:

```
curl https://raw.githubusercontent.com/42electronics/level_c/master/lesson_18/alarm.py > /home/pi/alarm/alarm42.py
```

The downloaded file will be saved in your **`/home/pi/alarm`** directory, but the new file will be named **`alarm42.py`**. This way you can view and run the file without overwriting your file in case you still might want to investigate what went wrong with your file.

Here is the full version of the program for you to compare against your program if needed:

```
import RPi.GPIO as GPIO
import Adafruit_SSD1306
from tkinter import *
from PIL import Image, ImageDraw, ImageFont
import time, os, ultrasonic, _thread

red = 13
green = 19
sensitivity = 50
armed = 0

GPIO.setmode(GPIO.BCM)
GPIO.setup(red, GPIO.OUT)
GPIO.setup(green, GPIO.OUT)

disp = Adafruit_SSD1306.SSD1306_128_64(rst=None)
disp.begin()
width = disp.width
height = disp.height
image = Image.new('1', (width, height))
draw = ImageDraw.Draw(image)
font = ImageFont.truetype('/usr/share/fonts/truetype/freefont/FreeSans.ttf',24)

root = Tk()
root.title('Alarm System')

def audio_alert():
    os.system("gpio -g mode 18 ALT5")
    os.system("aplay /home/pi/alarm/alarm.wav")
    os.system("gpio -g mode 18 in")

def led_update(red_value,green_value):
    GPIO.output(red, red_value)
    GPIO.output(green, green_value)

def display_update(line1,line2):
    draw.rectangle((0,0,width,height), outline=0, fill=0)
    draw.text((0, 0), line1, font=font, fill=255)
    draw.text((0, 22), line2, font=font, fill=255)
    disp.image(image)
    disp.display()

def update():
    global img
    timestamp = (time.strftime('%Y-%m-%d_%H:%M:%S'))
    img_file = ('/home/pi/Pictures/captures/%s.png' % timestamp)
    os.system('raspistill -o %s -e png -w 640 -h 480 -t 1500' % img_file)
    print('Image saved as %s' % img_file)
    img = PhotoImage(file='%s' % img_file)
    Label(root, image=img).grid(row=0, column=0)
```

```
def proximity():
    while armed == 1:
        distance = ultrasonic.average()
        if distance < sensitivity:
            timestamp = time.strftime('%H:%M:%S')
            display_update('ALARM at ', timestamp)
            _thread.start_new_thread(audio_alert, ())
            update()
        time.sleep(0.1)

def arm():
    global armed
    global img
    armed = 1
    led_update(1,0)
    img = PhotoImage(file='/home/pi/alarm/armed.png')
    Label(root, image=img).grid(row=0, column=0)
    display_update('ARMED','')
    _thread.start_new_thread(proximity, ())

def disarm():
    global armed
    global img
    armed = 0
    led_update(0,1)
    img = PhotoImage(file='/home/pi/alarm/disarmed.png')
    Label(root, image=img).grid(row=0, column=0)
    display_update('DISARMED','')

def quit():
    raise SystemExit()

try:
    disarm()
    Label(root, image=img).grid(row=0, column=0)
    Button(root, text="Arm", command=arm, width=5).grid(row=1, column=0)
    Button(root, text="Disarm", command=disarm, width=5).grid(row=2, column=0)
    Button(root, text="Quit", command=quit, width=5).grid(row=3, column=0)
    root.protocol("WM_DELETE_WINDOW", quit)
    root.mainloop()

except (KeyboardInterrupt, SystemExit):
    led_update(0,0)
    root.destroy()
    disp.clear()
    disp.display()
    GPIO.cleanup()
```

QUESTIONS FOR UNDERSTANDING

1. Could a second sound file be played after the first?

2. What value should be modified to make the alarm trigger at 100 centimeters instead of 50 centimeters?

3. How could arm and disarm sounds be added to the program?

Answers can be found on the next page.

ANSWERS TO THE QUESTIONS FOR UNDERSTANDING

1. *Could a second sound file be played after the first?*

   ANSWER: Yes. By adding another os.system command right after the first, you could make the program play two sound files each time the alarm is triggered:

```
os.system("aplay /home/pi/alarm/alarm.wav")
os.system("aplay /home/pi/alarm/additional.wav
os.system("gpio -g mode 18 in")
```

2. *What value should be modified to make the alarm trigger at 100 centimeters instead of 50 centimeters?*

   ANSWER: The sensitivity value determines the distance, in centimeters between the Ultrasonic sensor and an object that will trigger an alarm event. This distance could be changed to 100 centimeters by making the following change to line 9 of the program:

   `sensitivity = 50` becomes **`sensitivity = 100`**

3. *How could arm and disarm sounds be added to the program?*

   ANSWER: The simplest way to add arm and disarm sounds to the program would be to add the three lines needed to play a sound file directly to the **`arm()`** and **`disarm()`** functions. Here is an example of playing a sound file named **`disarmed.wav`** to the **`disarm()`** function:

```
def disarm():
    global armed
    global img
    armed = 0
    led_update(0,1)
    img = PhotoImage(file='/home/pi/alarm/disarmed.png')
    Label(root, image=img).grid(row=0, column=0)
    display_update('DISARMED','')
    os.system("gpio -g mode 18 ALT5")
    os.system("aplay /home/pi/alarm/disarmed.wav")
    os.system("gpio -g mode 18 in")
```

Keep in mind that playing a long sound file using this method could cause delays in the program as this sound file is not being launched as a separate thread.

CONCLUSION

In this lesson you learned how to create a fairly complex program capable of using a sensor to trigger many actions in a program. This program and circuit could also be modified in many other ways to create completely new user interactions.

Congratulations! You have completed Level C of this course. You have learned a great number of new skills and you should be proud of yourself!

What's next?

- Order a copy of Level D of this course to learn to work with motor drive boards, advanced project planning and troubleshooting skills, and learning to assemble components and write code for a mobile platform (a significant challenge!) *Please note, the link above will be active once Level D is available (late 2019).*

- The skills you have gained and the components you have amassed working with Levels A-C, have put you in a great position to tackle most Raspberry Pi and Python projects online. You are likely to find that you have worked with much of the code and components that will be called for in the projects, and those you haven't, you very likely now have the skills to figure out.

Intro to Robotics Course of Study:

Level A: Building Circuits and Beginning Programming

Level B: Working with Sensors and Intermediate Programming

Level C: Audio-Visual and Advanced Programming

Level D: Working with Motors and Taking It Mobile