

Lesson 3 Multifunctional Reminder

In our daily lives, we are no strangers to sound. Beautiful and melodious music can be enchanting, while rhythmic whistle sounds can alert us. However, when the volume reaches a certain level and interferes with people's normal rest, study, and work, sound becomes noise.

At the same time, alarm clocks play a crucial role in many people's lives, as they can remind us of the time in a timely manner.

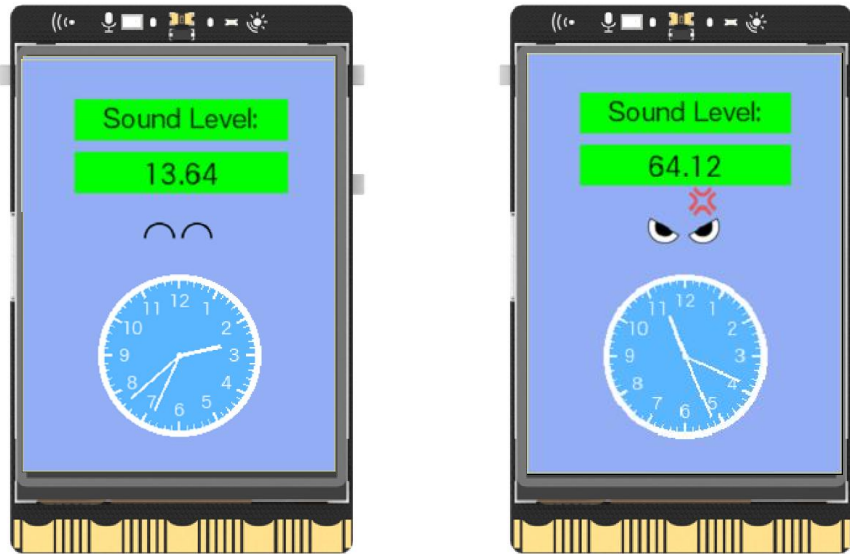
In this lesson, let's create a multifunctional reminder together. On the one hand, it monitors the ambient sound level and issues a warning when the volume is too high. On the other hand, it displays the time in real-time and reminds us when the specified time is reached.



Task Objectives

Function 1: Display the ambient noise level detected through the microphone on the screen, and set a noise threshold to trigger a warning from the buzzer when the volume exceeds the threshold.

Function 2: Display a clock on the screen, and provide a reminder when a set time is reached.





Knowledge points

1. Acquire knowledge of sound and related concepts.
2. Familiarize oneself with the microphone and buzzer.
3. Learn how to make the buzzer produce sound using the Pinpong library.
4. Learn how to detect ambient noise levels using the Unihiker library.
5. Learn how to display filled rectangles, filled clocks, and start threads using the Unihiker library.

Material List

Hardware List:

	
<p>UNIHAKER x1</p>	<p>Type-C & Micro Dual-Use USB Cable x1</p>

Software Preparation: Mind+ Programming Software x1

Knowledge background

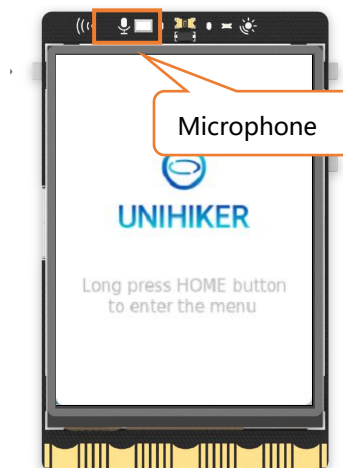
1. Sound and Related Concepts

In physics, sound is produced by the vibration of an object, and the object producing the sound is called the sound source. The frequency, measured in Hertz, is the number of vibrations per second of an object. Loudness, also known as volume, is the subjective perception of the sound's intensity. Loudness is determined by the amplitude of the vibration and the distance between the sound source and the listener. The greater the amplitude, the louder the sound, and the closer the listener is to the sound source, the louder the sound. Pitch refers to the highness or lowness of a sound and is determined by the frequency. The higher the frequency, the higher the pitch.

In this lesson, we will learn how to set the pitch by controlling the frequency.

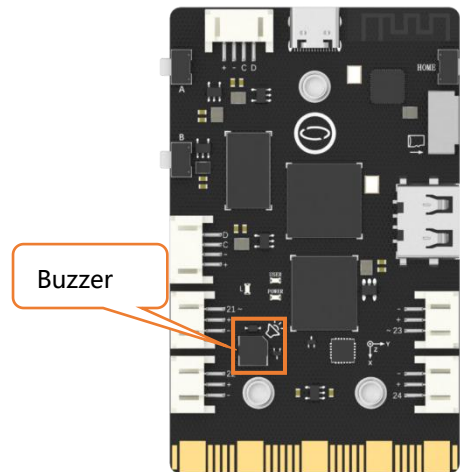
2. Microphone

A microphone, also known as a transducer, is a device that can convert sound signals into electrical signals and is used to detect speech and sound intensity. In this lesson, we will use the built-in microphone on the board to detect environmental volume, which returns a percentage ranging from 0% to 100%. The higher the volume, the higher the value.

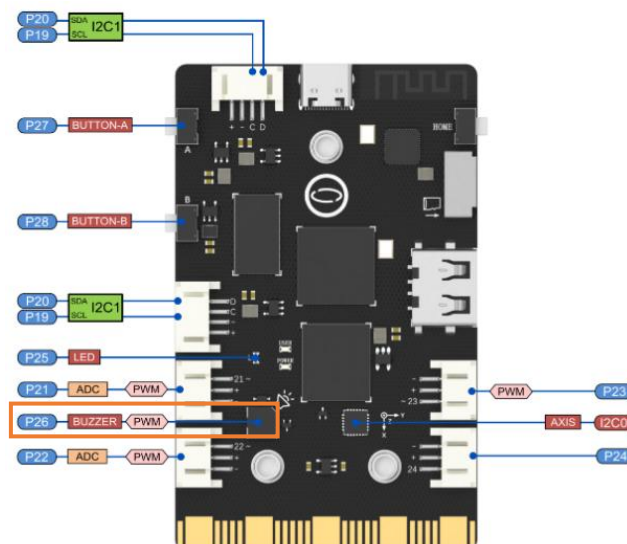


3. Buzzer

A buzzer is a device that can emit a sound similar to a buzzing noise. The buzzer on the UNIHAKER is built into the microcontroller system.



All components on the microcontroller system have their own functional pins, and the pin for the buzzer is P26, as shown in the figure below.



4. Common Methods in the Tone Class of the Pinpong Library

The Tone class in the Pinpong library can control the buzzer to produce sound. Before using it, we need to import the Pinpong library, initialize the board, and instantiate the Tone class to create an object.

```
from pinpong.board import Board, Pin, Tone # Importing the Board, Pin, and Tone modules from the pinpong.board package

Board().begin() # Initialize the board, choose the board type and port number (auto-detection if not specified)

tone = Tone(Pin(Pin.P26)) # Create a Tone object with Pin.P26 for analog output
```

Pin, Tone, and Board are all modules under the pinpong.board package. "P26" refers to the pin where the buzzer is located on the board. "Pin(Pin.P26)" creates a pin object, which is then passed into the Tone class for instantiation, resulting in a tone object.

There are many methods in the Tone class, but we only use some of them. When programming, we implement the desired functionality using the "object.method()" format.

- (1) The freq() method sets the frequency of the sound produced by the buzzer

```
tone.freq(200) # Set the frequency to 200 for the tone playback
```

Here, "200" represents the specific frequency value that is being set.

- (2) The on() method activates the buzzer

```
tone.on() # Turn on the tone output
```

- (3) The off() method turns off the buzzer

```
tone.off() # Turn off the tone output
```

5. Common methods in the Unihiker library's GUI class

There are many methods in the GUI class, but we only use a portion of them. When programming, we implement the functionality through the format 'object.methodName()'.

- (1) The fill_rect() method displays a filled rectangle

The fill_rect() method in the GUI class can display a filled rectangle on the UNIHIKER screen.

```
gui.fill_rect(x=40, y=30, w=160, h=30, color="#00ff00") # Display the first filled rectangle  
gui.fill_rect(x=40, y=70, w=160, h=30, color="#00ff00") # Display the second filled rectangle
```

Here, the parameters x and y represent the horizontal and vertical coordinates of the filled rectangle, respectively. Parameters w and h represent the width and height of the filled rectangle, and color corresponds to the color to be filled.

- (2) The fill_clock() method displays a filled clock on the UNIHIKER screen

The method fill_clock() in the GUI class can display a filled clock on the UNIHIKER screen.

```
# Display filled clock  
  
clock = gui.fill_clock(x=120, y=230, r=60, h=3, m=4, s=5, color=(255, 255, 255), fill="#57b5ff")  
)
```

The parameters x and y represent the horizontal and vertical coordinates of the filled clock, respectively. The parameter r represents the radius of the filled clock. The parameters h, m, and s represent the initial values of the hour hand, minute hand, and second hand of the clock, respectively. The parameter color represents the border color, while the parameter fill represents the fill color, which can be set in three different ways, just like the color parameter.

- (3) The method start_thread() starts a thread.

The start_thread() method in the GUI class can initiate a new thread.

```
clock_thread = gui.start_thread(clock_update)# Start thread
```

Here, "clock_thread" refers to the thread object, and "clock_update" refers to the thread function to be started.

6. The Unihiker library's Audio class sound_level() method is used to detect the ambient sound level

The Audio class in the Unihiker library provides the sound_level() method to detect the ambient sound level of the board. Prior to using this method, we need to import this module from the library and create an object by instantiating the class. Then, we can implement the functionality by calling the method using "object.method_name()" syntax.

```
from unihiker import GUI, Audio # Importing the Audio modules from the unihiker library

audio = Audio() # Instantiate the Audio class

Sound = audio.sound_level() # Detect the sound level and store the value in the variable Sound
```

The variable "Sound" is created to store the ambient sound level detected by the method.

7. Common functions in the Datetime library include

The Datetime library is a built-in Python library used for processing date and time. The "now()" function in the datetime module can be used to obtain the current local system time. Before using it, the module needs to be imported.

```
from datetime import datetime # Import the datetime module from the datetime library

now_time = datetime.now().strftime('%H:%M') # Get the current time in HH:MM format
```

The "strftime('%H:%M')" function can be used to convert a time format to a string, where "H" and "M" represent hours and minutes, respectively.

8. Common Functions in the Time Library

The localtime() function in the Time library can format a timestamp to the local time. Before using it, this library needs to be imported.

```
import time # Import the time module

t = time.localtime() # Get the current local time

clock.config(h=time.strftime("%H", t), m=time.strftime("%M", t), s=time.strftime("%S", t)) # Update the clock display
```

Here, "time.strftime("%H", t)" returns the hour of the current time as a string, "time.strftime("%M", t)" returns the minute of the current time, and "time.strftime("%S", t)" returns the second of the current time.

Tips: Both the Datetime and Time libraries have functions to obtain the current time, but they present the information in different formats. We will not discuss the differences in detail here.

Hands-on practice

Task Description 1: Detecting Environmental Sound Levels

The environmental sound level detected by the microphone will be displayed on the screen.

1. Hardware setup

STEP 1: Connect the UNIHAKER to the computer via a USB cable.



2. program coding

STEP1: Create and Save Project File

Launch Mind+, save the project as "003 Multifunctional Reminder".

STEP2: Create and Save Python File

Create a Python program file named "main1.py" and double-click to open it.

STEP3: Importing Images

To import the required PNG images into the project folder.

STEP4: Programming

(1) Import necessary libraries

In this task, we need to use the GUI module from the Unihiker library to display text on the screen, and the Audio module to detect the environmental volume. Therefore, we need to import them first, as shown in the code below. Additionally, since we need to set a certain delay during the program's execution, we also need to import the time library.

```
# Detecting Environmental Sound Level (Percentage)
from unihiker import GUI, Audio # Importing the GUI and Audio modules from the unihiker l
```

```
library
import time # Importing the time module
```

(2) Instantiate the GUI class

Before using the GUI module and Audio module from the Unihiker library, we need to instantiate the GUI class and Audio class to create an object in order to use the various methods within the class.

```
gui = GUI() # Instantiate the GUI class and create a gui object
audio = Audio() # Instantiate the Audio class
```

(3) Display a background image

After creating the GUI object, we can display a background image on the UNIHAKER screen.

```
# Display background image
img = gui.draw_image(x=0, y=0, w=240, h=320, image='background.png')
```

(4) Displaying Filled Rectangles

Next, we can display two rectangles with filled colors on the UNIHAKER screen.

```
# Display filled rectangles
gui.fill_rect(x=40, y=30, w=160, h=30, color="#00ff00") # Display the first filled rectangle
gui.fill_rect(x=40, y=70, w=160, h=30, color="#00ff00") # Display the second filled rectangle
```

(5) Displaying the environment volume value

Afterwards, we can write text on the rectangle to display the initial environment volume.

```
# Display text
text1 = gui.draw_text(x=120, y=30, text='Sound Level:', origin='top') # Display the text "Sound Level:"
text_sound = gui.draw_text(x=120, y=70, text="", font_size=15, origin='top') # Display the initial sound level
```

(6) Displaying an emoji

Additionally, we can add an emoji below the environment volume value.

```
# Display emoji
emj1 = gui.draw_emoji(x=82, y=90, w=100, h=100, emoji="Smile", duration=0.1) # Display the initial emoji
```

(7) Continuously monitoring the environment volume

Lastly, we will continuously monitor and update the environment volume value every 0.1 seconds.


```
while True:
```

```
    Sound = audio.sound_level() # Detect the sound level and store it in the variable Sound  
    text_sound.config(text=Sound) # Update the display of the sound level on the screen  
    time.sleep(0.1) # Delay for 0.1 seconds
```

Tips: The complete example program is as follows:

```
# Detecting Environmental Sound Level (Percentage)
```

```
from unihiker import GUI, Audio # Importing the GUI and Audio modules from the unihiker library
```

```
import time # Importing the time module
```

```
gui = GUI() # Instantiate the GUI class and create a gui object
```

```
audio = Audio() # Instantiate the Audio class
```

```
print("Sound Level Detection") # Print a message indicating the start of sound level detection
```

```
# Display background image
```

```
img = gui.draw_image(x=0, y=0, w=240, h=320, image='background.png')
```

```
# Display filled rectangles
```

```
gui.fill_rect(x=40, y=30, w=160, h=30, color="#00ff00") # Display the first filled rectangle
```

```
gui.fill_rect(x=40, y=70, w=160, h=30, color="#00ff00") # Display the second filled rectangle
```

```
# Display text
```

```
text1 = gui.draw_text(x=120, y=30, text='Sound Level:', origin='top') # Display the text "Sound Level:"
```

```
text_sound = gui.draw_text(x=120, y=70, text="", font_size=15, origin='top') # Display the initial sound level
```

```
# Display emoji
```

```
emj1 = gui.draw_emoji(x=82, y=90, w=100, h=100, emoji="Smile", duration=0.1) # Display the initial emoji
```

```
while True:
```

```
    Sound = audio.sound_level() # Detect the sound level and store it in the variable Sound  
    text_sound.config(text=Sound) # Update the display of the sound level on the screen  
    time.sleep(0.1) # Delay for 0.1 seconds
```

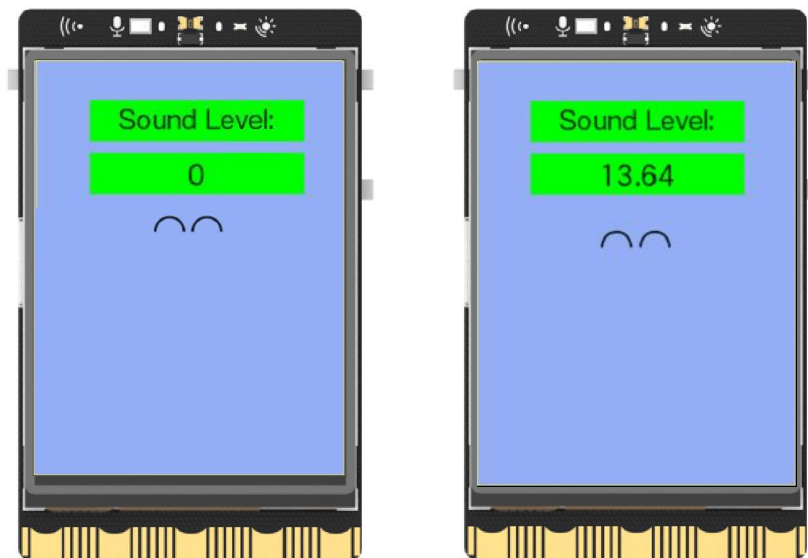
3. Running the Program

STEP 1: Remote connection to the UNIIKER.

STEP2: Click on the "Run" button located in the top right corner.

STEP3: Observe the results.

Observing the UNIIKER, we can see that the initial environment volume value is 0 and there is a smiling face displayed on the screen. If we blow into the microphone of the blank board (creating artificial noise), we can see that the volume value instantly increases.



Task Description 2: Adding feedback reminders

After implementing real-time volume detection, we will add a feedback reminder function. We will set a noise threshold, and when the volume exceeds this value, the buzzer will sound to remind us, and the expression on the screen will change to angry.

1. program coding

STEP 1: Create and Save Project Files

Create a Python program file named "main2.py" and double-click to open it.

STEP 2: Programming

(1) Setting up the Buzzer Functionality with the Pinpong Library

Here, we will be using the onboard buzzer, so when importing the library, we need to include the Board, Pin, and Tone modules from the Pinpong library and initialize the UNIIKER. Additionally, we need to instantiate the Tone class and set the frequency at which the buzzer will emit sound.

```

from pinpong.board import Board, Pin, Tone # Importing the Board, Pin, and Tone modules f
rom the pinpong.board package

Board().begin() # Initialize the board, choose the board type and port number (auto-detectio
n if not specified)

tone = Tone(Pin(Pin.P26)) # Create a Tone object with Pin.P26 for analog output

tone.freq(200) # Set the frequency to 200 for the tone playback

```

(2) Adding Feedback Alert

Next, we will add a conditional statement based on the environment volume detection every 0.1 seconds. When the volume exceeds a threshold of 50, the screen expression will switch to anger, and the buzzer will emit sound for 1.5 seconds.

```

while True:

    Sound = audio.sound_level() # Detect the sound level and store the value in the variable S
ound

    text_sound.config(text=Sound) # Update the display of the sound level on the screen

    time.sleep(0.1) # Delay for 0.1 seconds

    if Sound > 50: # If the sound level is greater than 50

        emj1.config(emoji="Angry") # Change the emoji to "Angry"

        tone.on() # Turn on the tone output

        time.sleep(1.5) # Delay for 1.5 seconds

        tone.off() # Turn off the tone output

        time.sleep(0.5) # Delay for 0.5 seconds

        emj1.config(emoji="Smile") # Change the emoji back to "Smile"

```

Tips: The complete example program is as follows:

```

from unihiker import GUI, Audio # Importing the GUI and Audio modules from the unihiker li
brary

import time # Importing the time module

from pinpong.board import Board, Pin, Tone # Importing the Board, Pin, and Tone modules f
rom the pinpong.board package

Board().begin() # Initialize the board, choose the board type and port number (auto-detectio
n if not specified)

tone = Tone(Pin(Pin.P26)) # Create a Tone object with Pin.P26 for analog output

tone.freq(200) # Set the frequency to 200 for the tone playback

gui = GUI() # Instantiate the GUI class and create a gui object

audio = Audio() # Instantiate the Audio class for sound operations

print("Sound Level Detection") # Print a message indicating the start of sound level detectio

```

```

n

# Display the background image
img = gui.draw_image(x=0, y=0, w=240, h=320, image='background.png')

# Display filled rectangles
gui.fill_rect(x=40, y=30, w=160, h=30, color="#00ff00") # Display the first filled rectangle
gui.fill_rect(x=40, y=70, w=160, h=30, color="#00ff00") # Display the second filled rectangle

# Display text
text1 = gui.draw_text(x=120, y=30, text='Sound Level:', origin='top')
text_sound = gui.draw_text(x=120, y=70, text="", font_size=15, origin='top')

# Display emoji
emj1 = gui.draw_emoji(x=82, y=90, w=100, h=100, emoji="Smile", duration=0.1)

while True:
    Sound = audio.sound_level() # Detect the sound level and store the value in the variable Sound
    text_sound.config(text=Sound) # Update the display of the sound level on the screen
    time.sleep(0.1) # Delay for 0.1 seconds

    if Sound > 50: # If the sound level is greater than 50
        emj1.config(emoji="Angry") # Change the emoji to "Angry"
        tone.on() # Turn on the tone output
        time.sleep(1.5) # Delay for 1.5 seconds
        tone.off() # Turn off the tone output
        time.sleep(0.5) # Delay for 0.5 seconds
        emj1.config(emoji="Smile") # Change the emoji back to "Smile"

```

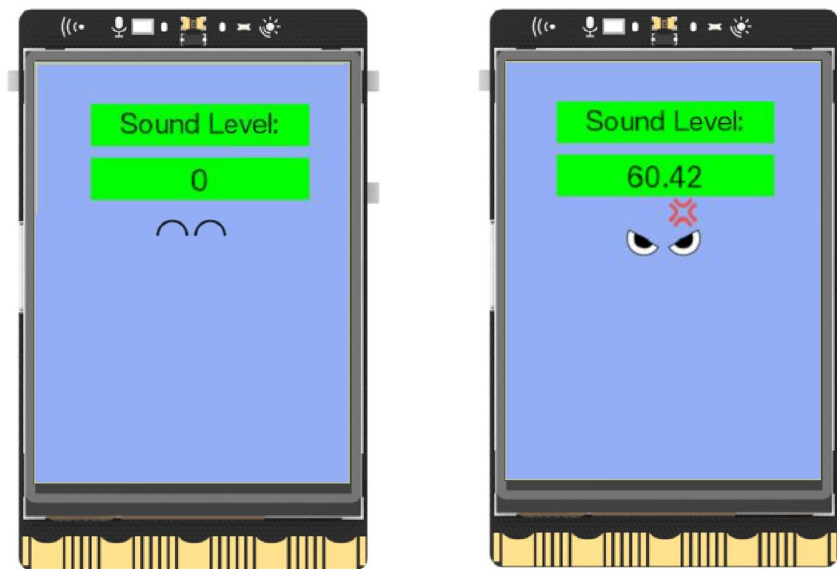
2. Running the Program

STEP 1: Remote connection to the UNIHAKER

STEP2: Running the Program and Observing the Results

After clicking run, observe the UNIHAKER. Initially, the environment volume value is 0, and the expression on the screen is "Smile". Blow air towards the microphone on the board (creating artificial noise), and you will notice that the volume value instantly increases. When the volume exceeds the set threshold of 50, the expression on the UNIHAKER screen will instantly switch to

anger, and the buzzer will sound for 1.5 seconds.



Task Description 3: Adding Alarm Functionality

Next, we will add the alarm functionality based on the noise alert. We will display the clock on the screen and set a specific time to trigger the buzzer for the alarm.

1. program coding

STEP1: Create and Save Python File

Create a Python program file named "main1.py" and double-click to open it.

STEP2: Programming

(1) Import necessary libraries

As we need to check the system time, we need to import the datetime library for later use.

```
import datetime # Import the datetime library
```

(2) Displaying the Filled-in Clock

We will add a clock display below the initial expression on the screen.

```
# Display filled clock  
clock = gui.fill_clock(x=120, y=230, r=60, h=3, m=4, s=5, color=(255, 255, 255), fill="#57b5ff")
```

(3) Defining the Function to Update the Clock

In order to keep the clock display synchronized with the real time, we need to define a function to update the clock display, and set the buzzer to sound an alert when the specified time is reached.

Tips: The time here can be changed as needed.

```
# Define clock update function
def clock_update():
    print("thread1 start")
    while True:
        t = time.localtime() # Get the current local time
        clock.config(h=time.strftime("%H", t), m=time.strftime("%M", t), s=time.strftime("%S", t))
    # Update the clock display
    time.sleep(1)
    now_time = datetime.now().strftime('%H:%M') # Get the current time in HH:MM format
    print(now_time)
    if now_time == '18:00': # If the current time is 18:00 (can be changed)
        tone.on() # Turn on the tone output
        time.sleep(1.5) # Delay for 1.5 seconds
        tone.off() # Turn off the tone output
        time.sleep(0.5) # Delay for 0.5 seconds
    print("thread1 end")
```

(4) Starting the Thread to Update the Clock

Since both the environment volume and time need to be updated in real time, we will start the function to update the clock as a new thread separately.

```
# Start thread
clock_thread = gui.start_thread(clock_update)
```

Tips: The complete example program is as follows:

```
from unihiker import GUI, Audio # Import the GUI and Audio modules from the unihiker library
import time # Import the time module
from datetime import datetime # Import the datetime module from the datetime library
from pinpong.board import Board, Pin, Tone # Import the Board, Pin, and Tone modules from the pinpong.board package

Board().begin() # Initialize the board, choose the board type and port number (auto-detection if not specified)
```

```

tone = Tone(Pin(Pin.P26)) # Create a Tone object with Pin.P26 for analog output
tone.freq(200) # Set the frequency to 200 for the tone playback

gui = GUI() # Instantiate the GUI class and create a gui object
audio = Audio() # Instantiate the Audio class for sound operations
print("Sound Level Detection") # Print a message indicating the start of sound level detection

# Display the background image
img = gui.draw_image(x=0, y=0, w=240, h=320, image='background.png')

# Display filled rectangles
gui.fill_rect(x=40, y=30, w=160, h=30, color="#00ff00") # Display the first filled rectangle
gui.fill_rect(x=40, y=70, w=160, h=30, color="#00ff00") # Display the second filled rectangle

# Display text
text1 = gui.draw_text(x=120, y=30, text='Sound Level:', origin='top')
text_sound = gui.draw_text(x=120, y=70, text='', font_size=15, origin='top')

# Display emoji
emj1 = gui.draw_emoji(x=82, y=90, w=100, h=100, emoji="Smile", duration=0.1)

# Display filled clock
clock = gui.fill_clock(x=120, y=230, r=60, h=3, m=4, s=5, color=(255, 255, 255), fill="#57b5ff")

# Define clock update function
def clock_update():
    print("thread1 start")
    while True:
        t = time.localtime() # Get the current local time
        clock.config(h=time.strftime("%H", t), m=time.strftime("%M", t), s=time.strftime("%S", t))
        # Update the clock display
        time.sleep(1)
        now_time = datetime.now().strftime('%H:%M') # Get the current time in HH:MM format
        print(now_time)
        if now_time == '18:00': # If the current time is 18:00 (can be changed)
            tone.on() # Turn on the tone output
            time.sleep(1.5) # Delay for 1.5 seconds

```

```

        tone.off() # Turn off the tone output
        time.sleep(0.5) # Delay for 0.5 seconds
    print("thread1 end")

# Start thread
clock_thread = gui.start_thread(clock_update)

while True:
    Sound = audio.sound_level() # Detect the sound level and store the value in the variable Sound
    text_sound.config(text=Sound) # Update the display of the sound level on the screen
    time.sleep(0.1) # Delay for 0.1 seconds
    if Sound > 50: # If the sound level is greater than 50
        emj1.config(emoji="Angry") # Change the emoji to "Angry"
        tone.on() # Turn on the tone output
        time.sleep(1.5) # Delay for 1.5 seconds
        tone.off() # Turn off the tone output
        time.sleep(0.5) # Delay for 0.5 seconds to avoid triggering the microphone by the tone itself
        emj1.config(emoji="Smile") # Change the emoji to "Smile"

```

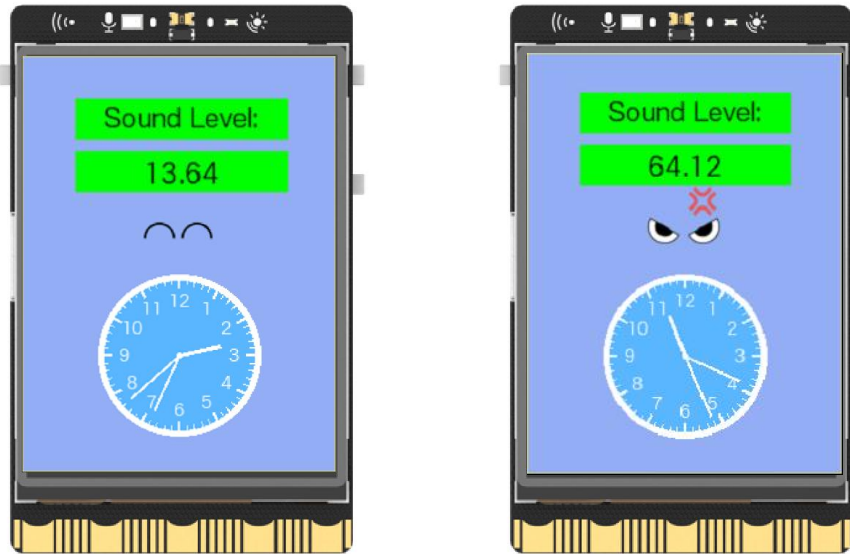
2. Running the Program

STEP 1: Remote connection to the UNIIKER

STEP2: Running the Program and Observing the Results

After clicking run, observe the blank space and you will see that a clock has been added below the expression. When the time reaches "18:00" as set in the program, the buzzer will sound an alarm. At the same time, the ambient sound level is being continuously monitored.

Tips: To ensure the accuracy of the clock display, it is necessary to connect the UNIIKER to the network. For detailed instructions, please refer to the further reading materials.



Challenge Yourself

Why not try to add a suitable background image to our multi-functional reminder to make it more realistic? And let's brainstorm additional features we can add to the reminder, besides noise and time alerts. We can leverage our existing knowledge and give it a try ourselves!

Further reading

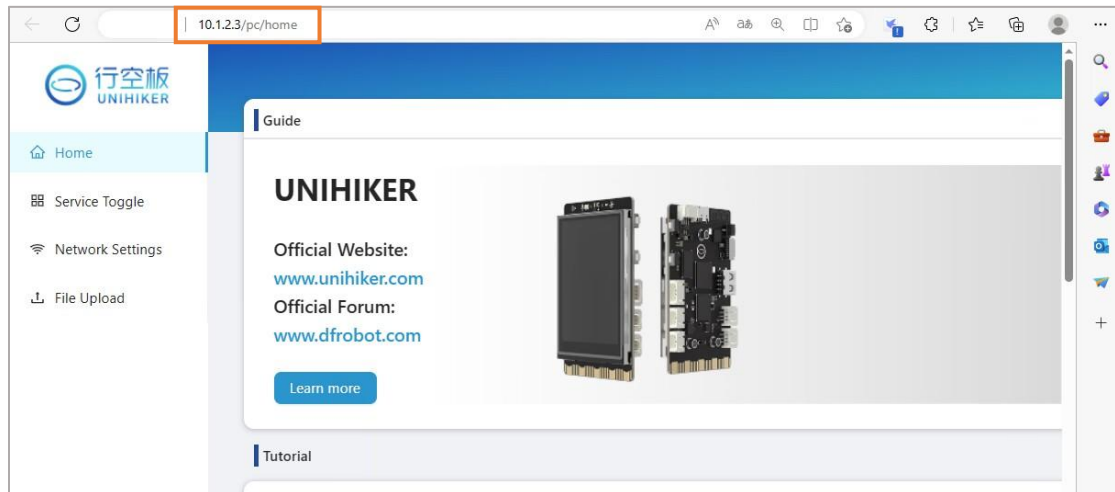
1. The Working Principle and Development of Microphones

At the beginning of the 20th century, microphones developed from the initial resistance-based sound-to-electricity conversion to inductance and capacitance-based conversion. A large number of new microphone technologies gradually emerged. The common working principle of microphones is that a capacitive polarized microphone sensitive to sound is built into the sensor. Sound waves cause the polarized membrane in the microphone to vibrate, resulting in a change in capacitance and generating a corresponding small voltage. This voltage is then converted into a 0-5V voltage, accepted by the data collector after A/D conversion, and transmitted.

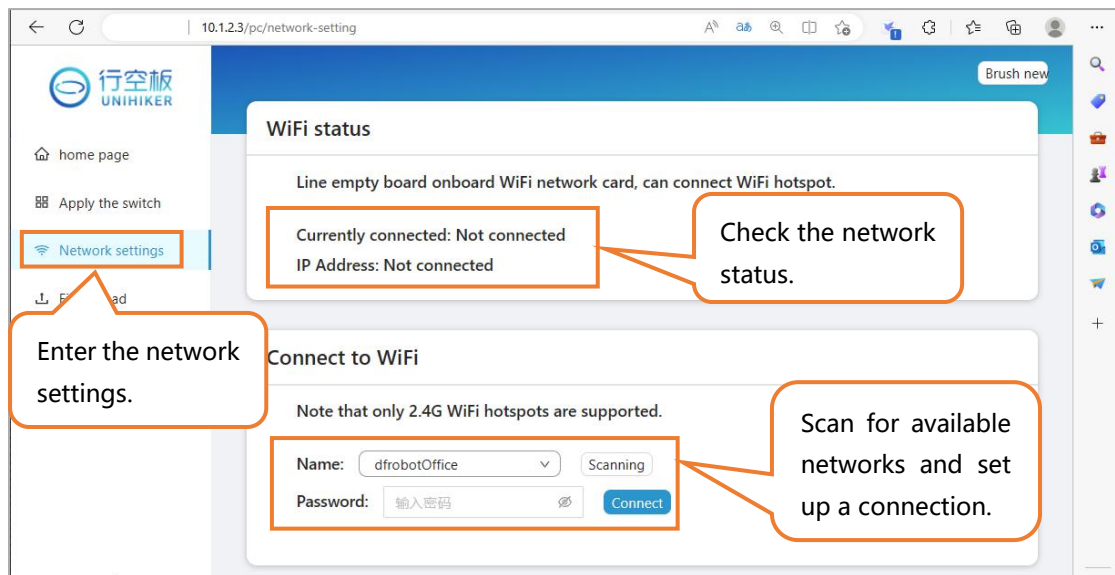
Many sound-controlled devices used in daily life, such as sound-controlled lights and smart TVs, rely on microphones. The application field of microphones is constantly expanding, from robots to aerospace technology. The role of microphones in modern technology is becoming increasingly important.

2. Connecting the UNIHAKER board to the internet method

STEP 1: Connect the Rowan board to your computer using a USB cable. Once connected, open your browser and type "10.1.2.3" to access the Rowan board's web menu.



STEP 2 : Access the network settings, which will automatically scan the hotspots around the board and the current network status. After the scanning is completed, select the hotspot you want to connect to and enter the password to connect to the network.



STEP 3: After successfully connecting to the network, refresh the webpage to view the WiFi connection status.

