import asyncio

import threading

import tkinter as tk

import tkinter.font

from gpiozero import LED, Button

from bleak import BleakScanner

from RPLCD.i2c import CharLCD

import RPi.GPIO as GPIO

import time

from queue import Queue, Empty

# Initialize LEDs and Buttons

ledGREEN = LED(20) # LED for green light

ledBLUE = LED(21) # LED for blue light (not used in traffic light logic here)

ledRED = LED(6) # LED for red light

ledYELLOW = LED(16) # LED for yellow light

btnGREEN = Button(13) # Button for manual green light control

btnBLUE = Button(19) # Button for automatic/manual mode toggle

btnRED = Button(12) # Button for manual red light control

btnYELLOW = Button(26) # Button for blinking yellow light

# Initialize LCD for status messages

lcd = CharLCD(i2c\_expander='PCF8574', address=0x27, port=1, cols=16, rows=2, dotsize=8)

# Global variables

counter = 5 # Start with yellow blinking (state counter)

stateAmount = 5 # Number of different states

automatic = False # Flag to indicate if the system is in automatic mode

thread = None # Thread reference for automatic mode

stop\_event = threading.Event() # Event to signal the thread to stop

device\_name = "Yosef khamaiseh" # Name of the Bluetooth device to scan for

scan\_interval = 5 # Interval in seconds for scanning

device\_discovered = False # Flag to indicate if the device has been discovered

device\_in\_range = False # Flag to track device connection status

actions\_performed = False # Flag to check if actions have been performed after device discovery

# Queue to communicate between threads

queue = Queue()

# Function definitions for traffic light control

def turn\_all\_off():

"""Turn off all traffic lights."""

ledGREEN.off()

ledYELLOW.off()

ledRED.off()

def set\_still\_red():

"""Set the traffic light to show only red light."""

turn\_all\_off()

ledRED.on()

def set\_still\_red\_yellow():

"""Set the traffic light to show red and yellow lights."""

turn\_all\_off()

ledRED.on()

ledYELLOW.on()

def set\_still\_green():

"""Set the traffic light to show only green light."""

turn\_all\_off()

ledGREEN.on()

def set\_blinking\_green():

"""Set the traffic light to blink the green light."""

turn\_all\_off()

ledGREEN.blink(0.2, 0.2) # Blink with 0.2s on and 0.2s off

def set\_still\_yellow():

"""Set the traffic light to show only yellow light."""

turn\_all\_off()

ledYELLOW.on()

def set\_blinking\_yellow():

"""Set the traffic light to blink the yellow light."""

turn\_all\_off()

ledYELLOW.blink(0.2, 0.2) # Blink with 0.2s on and 0.2s off

def start\_with\_blinking\_yellow():

"""Start the traffic light with a blinking yellow light and prepare the LCD display."""

global automatic, counter

counter = 0 # Reset the state counter

turn\_all\_off()

ledYELLOW.blink(0.2, 0.2) # Blink yellow light

update\_lcd("BlUe Automatic GRN FWD Red BACK")

if automatic:

automatic = False

stop\_event.set() # Signal the thread to stop

if thread and thread.is\_alive():

thread.join() # Wait for the thread to finish

stop\_event.clear() # Reset the event flag

def endless\_cycle():

"""Cycle through traffic light states in automatic mode."""

global automatic

while automatic and not stop\_event.is\_set():

switch\_state\_forward() # Move to the next state

time.sleep(2) # Wait 2 seconds before changing state

stop\_event.clear() # Clear the event flag

def automatic\_mode():

"""Toggle between automatic and manual mode."""

global automatic, thread, stop\_event

if automatic:

# Turn off automatic mode

automatic = False

stop\_event.set() # Signal the thread to stop

if thread and thread.is\_alive():

thread.join() # Wait for the thread to finish

stop\_event.clear() # Reset the event flag

else:

# Turn on automatic mode

automatic = True

stop\_event.clear() # Reset the event before starting

thread = threading.Thread(target=endless\_cycle) # Create a new thread for automatic mode

thread.start() # Start the thread

def manually\_switch\_state\_forward():

"""Move to the next traffic light state manually."""

if not automatic:

switch\_state\_forward()

def switch\_state\_forward():

"""Move to the next traffic light state."""

global counter

counter += 1 # Increment the state counter

counter %= stateAmount # Wrap around if necessary

switch\_state() # Update the traffic light state

def manually\_switch\_state\_backward():

"""Move to the previous traffic light state manually."""

if not automatic:

switch\_state\_backward()

def switch\_state\_backward():

"""Move to the previous traffic light state."""

global counter

counter -= 1 # Decrement the state counter

if counter < 1:

counter = stateAmount # Wrap around if necessary

switch\_state() # Update the traffic light state

def switch\_state():

"""Update the traffic light state based on the counter."""

global counter

match counter:

case 1:

set\_still\_red() # Show red light

case 2:

set\_still\_red\_yellow() # Show red and yellow lights

case 3:

set\_still\_green() # Show green light

case 4:

set\_blinking\_green() # Blink green light

case 5:

set\_still\_yellow() # Show yellow light

case \_:

set\_still\_yellow() # Default to yellow light

update\_lcd(get\_lcd\_message()) # Update LCD display

# LCD Functions

def update\_lcd(message):

"""Update the LCD with a new message."""

lcd.clear()

lcd.write\_string(message)

def get\_lcd\_message():

"""Get the message to be displayed on the LCD based on the state."""

if not automatic:

return "BlU AUT YEL BLNKGRN FWD Red BACK" if counter == 5 else "BlU AUT YEL BLNKGRN FWD Red BACK"

else:

return "BlUe Manual YEL YellowBLNKng"

# Button Event Handlers

def handle\_btnGREEN\_press():

"""Handle the press event of the green button."""

if device\_in\_range:

print("Green Button pressed")

manually\_switch\_state\_forward()

def handle\_btnRED\_press():

"""Handle the press event of the red button."""

if device\_in\_range:

print("Red Button pressed")

manually\_switch\_state\_backward()

def handle\_btnYELLOW\_press():

"""Handle the press event of the yellow button."""

if device\_in\_range:

print("Yellow Button pressed")

start\_with\_blinking\_yellow()

def handle\_btnBLUE\_press():

"""Handle the press event of the blue button."""

if device\_in\_range:

print("Blue Button pressed")

automatic\_mode()

# Assign button press handlers to physical buttons

btnGREEN.when\_pressed = handle\_btnGREEN\_press

btnRED.when\_pressed = handle\_btnRED\_press

btnYELLOW.when\_pressed = handle\_btnYELLOW\_press

btnBLUE.when\_pressed = handle\_btnBLUE\_press

# Close function to cleanup GPIO

def close():

"""Cleanup GPIO and close the GUI window."""

GPIO.cleanup()

win.destroy()

# Function to perform actions after finding the device

def perform\_actions\_after\_device\_found():

"""Perform actions after the device is discovered."""

global actions\_performed

if not actions\_performed:

turn\_all\_off()

update\_lcd("Ready")

start\_with\_blinking\_yellow() # Start with yellow blinking

actions\_performed = True # Prevent future executions

create\_gui()

# Bluetooth Scanning Function

async def scan\_for\_device(device\_name):

"""Scan for the Bluetooth device with the given name."""

global device\_discovered, device\_in\_range

while True:

print("Scanning for device...")

devices = await BleakScanner.discover() # Discover nearby Bluetooth devices

found = False

for device in devices:

if device\_name in device.name:

print(f"Device '{device.name}' found with address {device.address}")

device\_discovered = True

device\_in\_range = True # Device is in range

queue.put("device\_found") # Notify that the device is found

found = True

break

if not found:

if device\_in\_range:

print("Device is out of range.")

device\_in\_range = False # Device is out of range

queue.put("device\_lost") # Notify that the device is lost

await asyncio.sleep(scan\_interval) # Wait before the next scan

# Function to run the asyncio event loop in a separate thread

def asyncio\_thread\_func():

"""Run the asyncio event loop in a separate thread."""

loop = asyncio.new\_event\_loop()

asyncio.set\_event\_loop(loop)

loop.run\_until\_complete(scan\_for\_device(device\_name))

# GUI setup

def create\_gui():

"""Create the GUI window and set up the traffic light display and control buttons."""

global win, canvas, red\_light, yellow\_light, green\_light

win = tk.Tk()

win.title("Traffic Light Control and LED Toggler")

myFont = tkinter.font.Font(family='Helvetica', size=12, weight="bold")

canvas = tk.Canvas(win, width=300, height=300)

canvas.pack()

# Create traffic light display

traffic\_light = canvas.create\_rectangle(50, 50, 100, 230, fill="grey")

red\_light = canvas.create\_oval(60, 60, 90, 90, fill="black")

yellow\_light = canvas.create\_oval(60, 120, 90, 150, fill="black")

green\_light = canvas.create\_oval(60, 180, 90, 210, fill="black")

def update\_traffic\_light():

"""Update the traffic light display on the GUI based on the current state."""

canvas.itemconfig(red\_light, fill="black")

canvas.itemconfig(yellow\_light, fill="black")

canvas.itemconfig(green\_light, fill="black")

# Set initial state based on the counter

match counter:

case 1:

canvas.itemconfig(red\_light, fill="red")

case 2:

canvas.itemconfig(red\_light, fill="red")

canvas.itemconfig(yellow\_light, fill="yellow")

case 3:

canvas.itemconfig(green\_light, fill="green")

case 4:

if ledGREEN.is\_active:

canvas.itemconfig(green\_light, fill="green")

case 5:

if ledYELLOW.is\_active:

canvas.itemconfig(yellow\_light, fill="yellow")

case \_:

if ledYELLOW.is\_active:

canvas.itemconfig(yellow\_light, fill="yellow")

# Schedule the update\_traffic\_light function again after 200ms

win.after(200, update\_traffic\_light)

update\_traffic\_light()

# Create buttons on canvas

btn\_red = tk.Button(canvas, text="Red Button (Backward)", command=handle\_btnRED\_press, bg='red', height=1, width=20)

btn\_red.place(x=110, y=50)

btn\_green = tk.Button(canvas, text="Green Button (Forward)", command=handle\_btnGREEN\_press, bg='green', height=1, width=20)

btn\_green.place(x=110, y=100)

btn\_yellow = tk.Button(canvas, text="Yellow Button (Blink)", command=handle\_btnYELLOW\_press, bg='yellow', height=1, width=20)

btn\_yellow.place(x=110, y=150)

btn\_blue = tk.Button(canvas, text="Blue Button (Auto/Manual)", command=handle\_btnBLUE\_press, bg='blue', height=1, width=20)

btn\_blue.place(x=110, y=200)

exitButton = tk.Button(win, text='Exit', font=myFont, command=close, bg='red', height=1, width=6)

exitButton.pack(pady=5)

win.protocol("WM\_DELETE\_WINDOW", close) # Cleanup GPIO when user closes window

# Main loop to process messages from the queue

def process\_queue():

"""Process messages from the queue and update the system state."""

try:

message = queue.get\_nowait()

if message == "device\_found":

perform\_actions\_after\_device\_found()

elif message == "device\_lost":

# Perform actions when device is lost (e.g., stop all operations)

global actions\_performed

actions\_performed = False # Allow actions to be performed again when device is found

turn\_all\_off()

update\_lcd("Device lost")

except Empty:

pass

win.after(100, process\_queue) # Check the queue every 100ms

win.after(100, process\_queue) # Start processing the queue

win.mainloop()

# Start the Bluetooth scanning in a separate thread

asyncio\_thread = threading.Thread(target=asyncio\_thread\_func, daemon=True)

asyncio\_thread.start()

# The GUI will run in the main thread

create\_gui()