МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ЛЬВІВСЬКА ПОЛІТЕХНІКА

АВТОМАТИЗОВАНЕ ПРОЕКТУВАННЯ КОМП'ЮТЕРНИХ СИСТЕМ

Task2 create SW<>HW UART communication

Виконав :

Студент групи КІ-401

Онисько М.М.

Прийняв:

Федак П.Р

2024

Завдання:

1. Розробити прототип заданої гри
2. Розробити просту комунікацію між програмою та платою за допомогою UART
3. Плата повинна повертати надіслане повідомлення зі змінами
4. Створити YML файл для GitHub actions
5. Оновити Readme.md
6. Додати тег про нову версію
7. Злити створену гілку до develop

Теоретичні відомості:

UART

UART (англ. universal asynchronous receiver/transmitter — універсальний асинхронний приймач/передавач) — тип асинхронного приймача-передавача, компонентів комп'ютерів та периферійних пристроїв, що передає дані між паралельною та послідовною формами. UART звичайно використовується спільно з іншими комунікаційними стандартами, такими як EIA RS-232.

UART – це зазвичай окрема мікросхема чи частина мікросхеми, що використовується для з'єднання через комп'ютерний чи периферійний послідовний порт. UART нині загалом включені в мікроконтролери. Здвоєний UART (Dual UART або DUART) об'єднує двоє UART в одній мікросхемі. Багато сучасних мікросхем сьогодні випускаються з можливістю комунікації в синхронному режимі, такі прилади називають USART.

Послідовні передача та приймання

Біти даних передаються з одного місця в інше через дроти або інші носії. Якщо мова йде про великі відстані, вартість дротів стає великою. Щоб зменшити вартість довгих комунікацій, що переносять кілька біт паралельно, біти даних передають послідовно один за одним, і використовують UART для перетворення паралельної форми на послідовну на кожному кінці лінії зв'язку. Кожен UART має зсувний регістр, який є фундаментальним методом для перетворення між паралельними та послідовними формами.

Зазвичай UART не отримує і не генерує зовнішні сигнали, які подорожують між різними частинами обладнання. Як правило, для перетворення логічного рівня UART в та з зовнішнього рівня сигналів використовується окремий інтерфейсний блок.

Зовнішній сигнал може мати багато різних форм. Прикладами стандартизованих напруг сигналу можуть служити RS-232, RS-422 чи RS-485 від EIA. Історично присутність або відсутність струму (в електричному колі) використовувалася в телеграфних схемах. Деякі ж сигнальні схеми не використовують електричних дротів. Як приклад можна навести оптоволокно,

інфрачервоний зв'язок чи Bluetooth в своєму Serial Port Profile (SPP). Прикладами модуляції є аудіо сигнал телефонних модемів, РЧ модуляція даних, або DC-LIN для комунікацій по силових дротах.

Зв'язок може бути «дуплексним» (можливість одночасного приймання та передачі) або «напівдуплексним» (пристрої переключаються між режимами приймання та передачі).

UART широко використовується в інтерфейсі RS-232 для вбудованих систем комунікацій. Він використовується для зв'язку між мікроконтролерами і комп'ютером. Багато чипів забезпечують функціональність UART, та існують дешеві мікросхеми для конвертації логічного рівня сигналу (типу TTL) в сигнал рівня RS-232.

Асинхронні передача та приймання

Під час асинхроної передачі UART телетайпного типу посилає стартовий біт, потім від п'яти до восьми бітів даних, перший — найменш значимий, потім опціональний біт парності, і потім один, півтора чи два стопових біти. Стартовий біт надсилається в зворотній полярності до звичайного незайнятого стану ліній зв'язку. Стоповий біт відповідає незайнятому стану лінії і забезпечує паузу перед наступною порцією даних. Це зветься асинхронною старт-стоповою передачею. В механічних телетайпах стоповий біт часто був розтягнутим вдвічі, щоб дати можливість механізму надрукувати символ. Розтягнутий стоповий біт також допомагав при ресинхронізації. Біт парності перевіряє кількість одиниць між стартовим і стоповим бітами або парним та непарним, або ж цей біт може бути відсутнім. Непарна перевірка надійніша, бо вона може засвідчити, що принаймні одна одиниця передалася, а це дозволяє багатьом UART пересинхронізуватися. В синхронній передачі частота тактового генератора відновлюється окремо з потоку даних і старт-стопові біти не використовуються. Це покращує ефективність каналу зв'язку для надійних ліній, також надсилається більше корисних даних. Асинхронна передача не посилає нічого, коли нема що передавати. Натомість синхронний інтерфейс має завжди посилати якісь дані, щоб підтримувати синхронізацію між передавачем і приймачем. Як заповнювач порожнечі часто використовується ASCII-символ «SYN», це робиться автоматично передавачем.

Node MCU на базі ESP 8266

Рис1. – Вигляд плати Node MCU

NodeMCU * (Lolin) являє собою плату розробника на базі чіпа ESP8266 (версія ESP12E), який являти собою UART-WiFi модуль з ультра низьким споживанням. Сам чіп проєктувався для пристроїв зі світу інтернет речей, а дана плата дозволяє спростити розробку, тому що на ній вже реалізовано підключення по USB, регулятор живлення і всі виводи чіпа розведені на гребінки зі стандартним кроком 2.54 мм, що дозволяє вставити його в макетну плату і створити прототип навіть не включаючи паяльник. Крім цього плата поставляється з прошивкою NodeMCU, що дозволяє програмувати її за допомогою мови Lua або за допомогою Arduino IDE.

Характеристики:

- WiFi 802.11 b / g / n
- підтримка STA / AP / STA + AP режимів
- вбудований стек протоколів TCP / IP з підтримкою множинних клієнтських підключень (до 5)
- D0 ~ D8, SD1 ~ SD3: можуть бути використані як GPIO, PWM, IIC, тощо.
- ток на виведення: 15 мА
- AD0: 1 виведення АЦП
- живлення: 4.5 - 9В (10В максимум), живлення від USB з наданням отладочного інтерфейсу
- споживання: обмін даними: ~ 70 мА (200 мА максимум), очікування: <200 мкА
- швидкість передачі: 110-460800 б/сек
- підтримка UART / GPIO інтерфейсів передачі даних
- перепрошивка з хмари або через USB
- відстань між контактними пинами: 28 мм
- діапазон робочих температур: -40 ~ +125 ° C
- вага: 18 г

Виконання

Прототип гри Tic-Tac-Toe на Python Tkinter

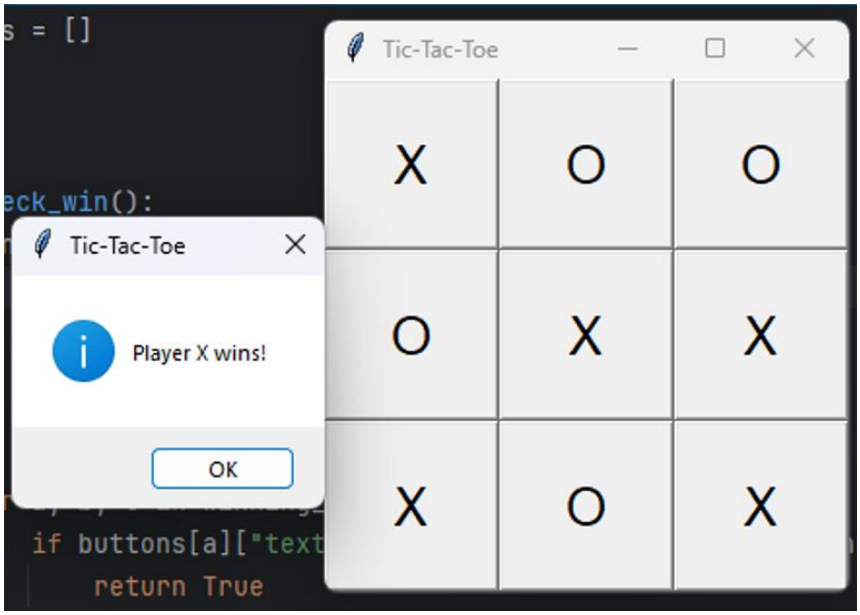

Рис.2 – Інтерфейс гри

Прототип гри написаний за допомогою tkinter що дозволяє запустити її без встановлення допоміжних графічних модулів.

Код game.py

```
import tkinter as tk
from tkinter import messagebox

root = tk.Tk()
root.title("Tic-Tac-Toe")

current_player = "X"
buttons = []


def check_win():
    winning_combinations = [
        (0, 1, 2), (3, 4, 5), (6, 7, 8),  # Rows
        (0, 3, 6), (1, 4, 7), (2, 5, 8),  # Columns
        (0, 4, 8), (2, 4, 6)  # Diagonals
    ]

    for a, b, c in winning_combinations:
        if buttons[a]["text"] == buttons[b]["text"] == buttons[c]["text"] != " ":
            return True
    return False


def check_draw():
    for button in buttons:
        if button["text"] == " ":
            return False
    return True
```

```
def on_button_click(index):
    global current_player

    if buttons[index]["text"] == " ":
        buttons[index]["text"] = current_player
        if check_win():
            messagebox.showinfo("Tic-Tac-Toe", f"Player {current_player} wins!")
            reset_game()
        elif check_draw():
            messagebox.showinfo("Tic-Tac-Toe", "It's a draw!")
            reset_game()
        else:
            current_player = "O" if current_player == "X" else "X"
    else:
        messagebox.showwarning("Tic-Tac-Toe", "Invalid move! Try again.")


def reset_game():
    global current_player
    current_player = "X"
    for button in buttons:
        button["text"] = " "


for i in range(9):
    button = tk.Button(root, text=" ", font=("Arial", 20), width=5, height=2,
                       command=lambda i=i: on_button_click(i))
    button.grid(row=i // 3, column=i % 3)
    buttons.append(button)

root.mainloop()
```

Клієнтська частина UART комунікації

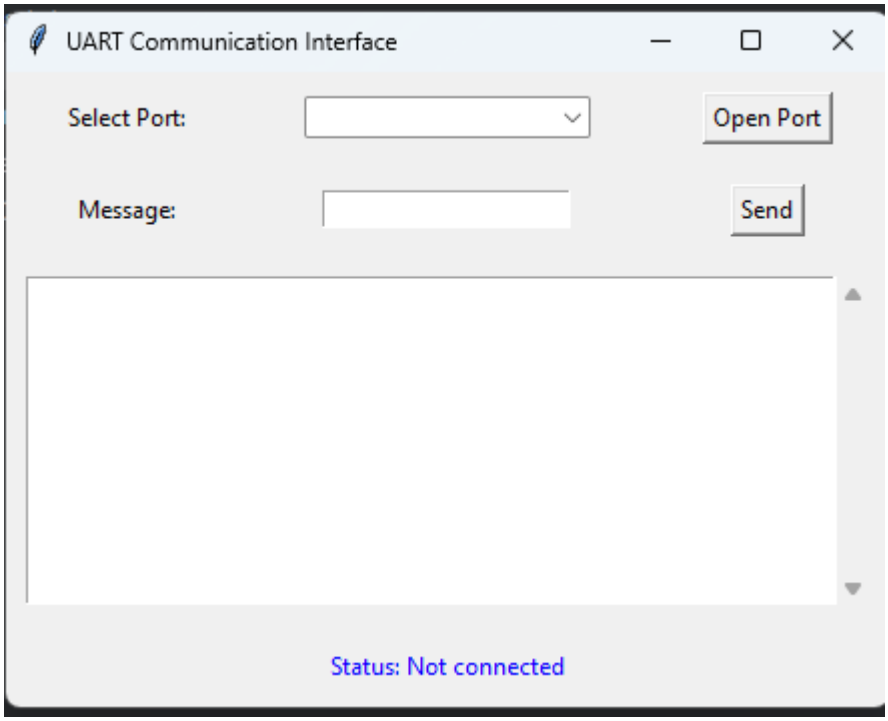

Рис.3 – Інтерфейс клієнтської частини

Клієнтська частина написана за допомогою графічного модуля tkinter та бібліотеки комунікації serial.

Код uart_communicate.py

```
import serial
import serial.tools.list_ports
import tkinter as tk
from tkinter import ttk, scrolledtext


class UARTCommunication:
    def __init__(self):
        self.ser = None

    def list_ports(self):
        return [port.device for port in serial.tools.list_ports.comports()]

    def open_port(self, port, baud_rate=9600):
        try:
            self.ser = serial.Serial(port, baud_rate, timeout=1)
            return f"Connected to {port}"
        except Exception as e:
            self.ser = None
            return f"Error: {e}"

    def send_message(self, message):
        if self.ser and self.ser.is_open:
            self.ser.write((message + "\n").encode())
            return f"Sent: {message}"
        return "Port not opened"

    def receive_message(self):
        if self.ser and self.ser.is_open:
            try:
                response = self.ser.readline().decode().strip()
                if response:
                    return response
            except Exception as e:
                return f"Error: {e}"
        return "Port not opened"


def auto_receive(uart, output_text, status_label, root):
    response = uart.receive_message()
    if response and response != "Port not opened":
        output_text.insert(tk.END, f"Received: {response}\n")
        output_text.see(tk.END)
    root.after(100, lambda: auto_receive(uart, output_text, status_label, root))


def start_gui():
    uart = UARTCommunication()

    root = tk.Tk()
    root.title("UART Communication Interface")

    port_label = tk.Label(root, text="Select Port:")
    port_label.grid(row=0, column=0, padx=10, pady=10)
```

```
    port_var = tk.StringVar()
    port_combobox = ttk.Combobox(root, textvariable=port_var, values=uart.list_ports(), state="readonly")
    port_combobox.grid(row=0, column=1, padx=10, pady=10)
    #port_combobox.current(0)

    def open_port_callback():
        status = uart.open_port(port_var.get())
        status_label.config(text=status)
        if "Connected" in status:
            auto_receive(uart, output_text, status_label, root)

    open_button = tk.Button(root, text="Open Port", command=open_port_callback)
    open_button.grid(row=0, column=2, padx=10, pady=10)

    message_label = tk.Label(root, text="Message:")
    message_label.grid(row=1, column=0, padx=10, pady=10)
    message_entry = tk.Entry(root)
    message_entry.grid(row=1, column=1, padx=10, pady=10)

    def send_message_callback():
        status = uart.send_message(message_entry.get())
        status_label.config(text=status)

    send_button = tk.Button(root, text="Send", command=send_message_callback)
    send_button.grid(row=1, column=2, padx=10, pady=10)

    output_text = scrolledtext.ScrolledText(root, width=50, height=10, wrap=tk.WORD)
    output_text.grid(row=2, column=0, columnspan=3, padx=10, pady=10)

    status_label = tk.Label(root, text="Status: Not connected", fg="blue")
    status_label.grid(row=3, column=0, columnspan=3, padx=10, pady=10)

    root.mainloop()


if __name__ == "__main__":
    start_gui()
```

Частина обладнання UART комунікації

Код для hardware частини написаний за допомогою ArduinoIDE для плати NodeMCU на базі ESP8266 що дозволяє, гнучко та швидко налаштовувати плату за допомогою C\C++ -подібної мови.

Код TicTacToe.ino

```
String modifyMessage(String receivedMessage) {
  return receivedMessage + " - Modified";
}

void setup() {
  Serial.begin(9600);
  Serial.println("Ready to receive messages");
}
```

```
void loop() {
  // Check if there is incoming data
  if (Serial.available() > 0) {
    String receivedMessage = Serial.readStringUntil('\n');

    String modifiedMessage = modifyMessage(receivedMessage);
    Serial.println(modifiedMessage);
  }
}
```

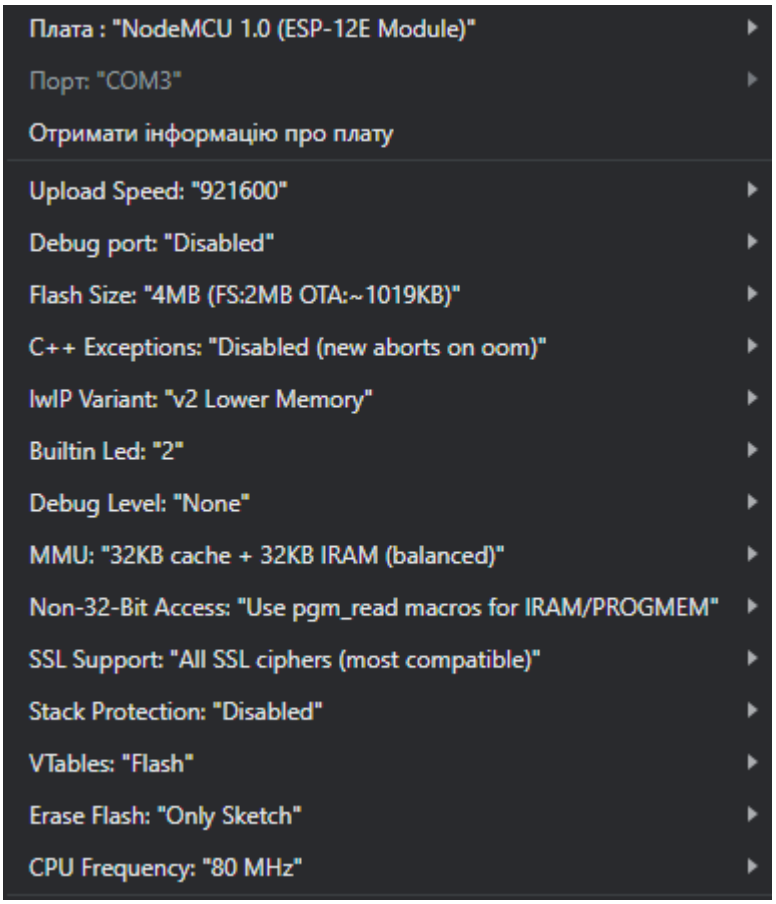

Рис.4 – Налаштування «завантажувача» плати

Інсталяція та запуск

Для початку роботи потрібно завантажити та інсталювати ArduinoIDE, це можна зробити на сайті https://www.arduino.cc/, та після додати ESP8266 у список виконавчих плат в середовищі

Наступним кроком потрібно завантажити та інсталювати драйвер плати CH340G

Завантажуємо скрипт в плату

Далі потрібно інсталювати залежності проекту інтерфейсу на Python за допомогою pip install -r requirements.txt, та запускаємо інтерфейс за допомогою команди python uart_communocate.py

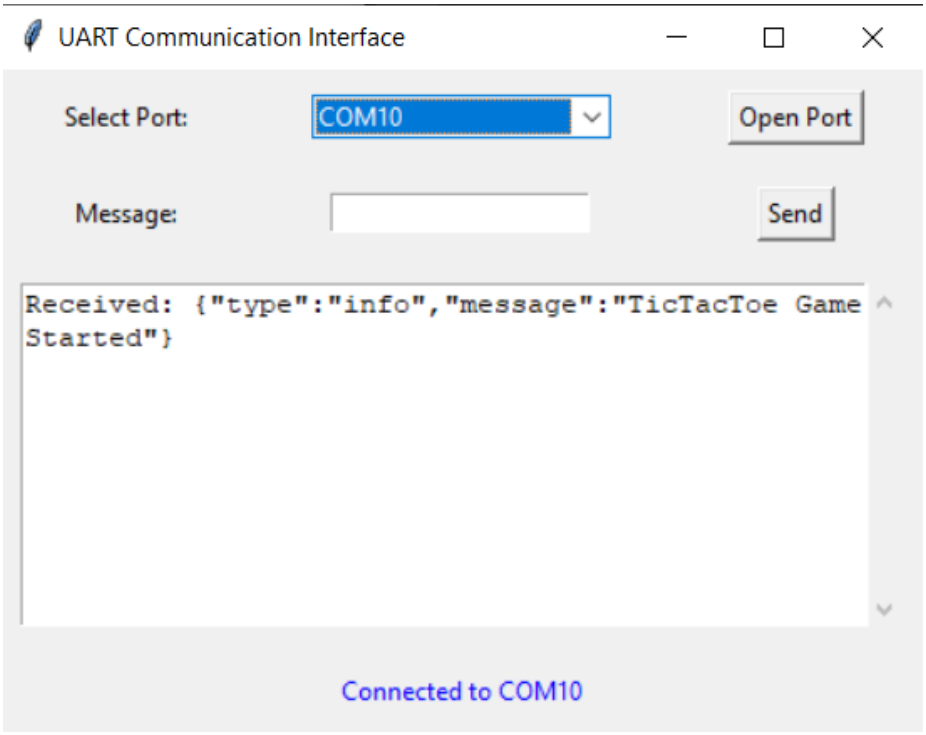

Рис.5 – Обираємо робочий порт

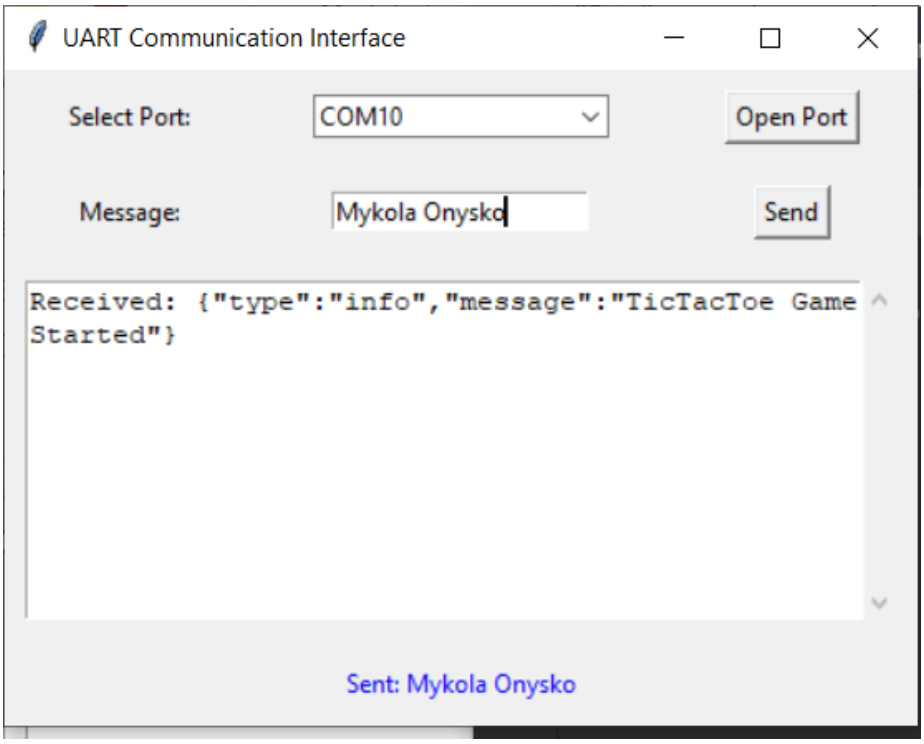

Рис.6 – Надсилаємо та отримуємо повідомлення

Тести

Код test.py

```
import unittest
from unittest.mock import patch
from uart_communicate import UARTCommunication


class TestUARTCommunication(unittest.TestCase):

    @patch('serial.Serial')
    def test_open_port_success(self, mock_serial):
        uart = UARTCommunication()
        mock_serial.return_value.is_open = True
        status = uart.open_port("COM3")
        self.assertEqual(status, "Connected to COM3")
        self.assertIsNotNone(uart.ser)

    @patch('serial.Serial')
    def test_open_port_fail(self, mock_serial):
        uart = UARTCommunication()
        mock_serial.side_effect = Exception("Cannot open port")
        status = uart.open_port("COM3")
        self.assertEqual(status, "Error: Cannot open port")
        self.assertIsNone(uart.ser)

    @patch('serial.Serial')
    def test_send_message_success(self, mock_serial):
        uart = UARTCommunication()
        mock_serial.return_value.is_open = True
        uart.ser = mock_serial()
        status = uart.send_message("Hello")
        self.assertEqual(status, "Sent: Hello")
        uart.ser.write.assert_called_with(b"Hello\n")

    @patch('serial.Serial')
    def test_send_message_fail(self, mock_serial):
        uart = UARTCommunication()
        uart.ser = None
        status = uart.send_message("Hello")
        self.assertEqual(status, "Port not opened")

    @patch('serial.Serial')
    def test_receive_message_success(self, mock_serial):
        uart = UARTCommunication()
        mock_serial.return_value.is_open = True
        uart.ser = mock_serial()
        uart.ser.readline.return_value = b"Received data\n"
        response = uart.receive_message()
        self.assertEqual(response, "Received data")

    @patch('serial.Serial')
    def test_receive_message_fail(self, mock_serial):
        uart = UARTCommunication()
        uart.ser = None
        response = uart.receive_message()
        self.assertEqual(response, "Port not opened")


if __name__ == '__main__':
    unittest.main()
```

Запуск GitHub actions

Це дозволяє компілювати код, та випробовувати тести прямо в GitHub

ci.yml

```
name: CI Workflow

on:
  push:
    branches:
      - develop
      - feature/develop/task2
  pull_request:
    branches:
      - develop

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      # Checkout the repository
      - name: Checkout code
        uses: actions/checkout@v3

      # Set up Python environment
      - name: Set up Python 3.x
        uses: actions/setup-python@v4
        with:
          python-version: '3.x'

      # Install Python dependencies
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r TicTacToeSWPart/requirements.txt
          pip install pytest

      # Run Python tests
      - name: Run tests
        run: |
          python -m pytest --junitxml=test-reports/results.xml TicTacToeSWPart/tests.py

      # Install Arduino CLI
      - name: Set up Arduino CLI
        run: |
          wget https://downloads.arduino.cc/arduino-cli/arduino-cli_latest_Linux_64bit.tar.gz
          tar -xvf arduino-cli_latest_Linux_64bit.tar.gz
          sudo mv arduino-cli /usr/local/bin/
          arduino-cli config init

      - name: Add ESP8266 package URL
        run: |
          arduino-cli config set board_manager.additional_urls http://arduino.esp8266.com/stable/package_esp8266com_index.json


      - name: Install ESP8266 core
```

```
      run: |
        arduino-cli core update-index
        arduino-cli core install esp8266:esp8266  # Install the ESP8266 core

    - name: Compile Arduino Sketch for NodeMCU
      run: |
        mkdir -p build  # Ensure the build directory exists
        arduino-cli compile --fqbn esp8266:esp8266:nodemcuv2 --output-dir build HWPart/TicTacToe/TicTacToe.ino

    # Collect binaries as artifacts
    - name: Upload binaries
      uses: actions/upload-artifact@v3
      with:
        name: compiled-files
        path: build/*.bin

    # Collect test results as artifacts
    - name: Upload test reports
      uses: actions/upload-artifact@v3
      with:
        name: test-reports
        path: test-reports/results.xml
```

Висновки

На лабораторній роботі я налаштував середовище для розробки проекту на Python + NodeMCU. Розробив просту UART комунікацію між платою та інтерфейсом та написав тести для неї. Також розробив прототип гри TicTacToe та скрипт GitGub actions.

Посилання

1. https://www.arduino.cc/
2. https://uk.wikipedia.org/wiki/UART
3. https://arduino.ua/prod1492-wi-fi-modyl-nodemcu-esp8266
4. https://docs.python.org/uk/3/library/tkinter.html
5. https://pyserial.readthedocs.io/en/latest/pyserial.html