МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ЛЬВІВСЬКА ПОЛІТЕХНІКА

Автоматизоване проектування комп’ютерних систем

Task 3. Implement Server (HW) and Client (SW) parts of game (FEF)

Виконав:

ст. гр КІ - 401

Савченко В. О.

Прийняв:

Федак П. Р.

2024

Опис теми

Для виконання завдання №3 потрібно Реалізувати серверну (HW) і клієнтську (SW) частини гри (FEF).

Теоретичні відомості

PlatformIO — це інструмент для розробки вбудованого програмного забезпечення з відкритим кодом. Він підтримує різні платформи мікроконтролерів та фреймворки, інтегрується з популярними середовищами розробки (наприклад, Visual Studio Code) і має вбудовану систему управління бібліотеками. PlatformIO спрощує розробку, компіляцію та завантаження програм на мікроконтролери.

JSON (JavaScript Object Notation) — це легкий формат обміну даними, зрозумілий для людини та легко оброблюваний машинами. Він використовує структуру з пар "ключ: значення" і підтримує вкладені об'єкти та масиви. JSON широко використовується для передачі даних у веб-додатках завдяки простоті синтаксису.

Виконання завдання

1. Розробив серверну та клієнтську частину гри:

main,py

```
import serial
import time
import threading
import json
import os

CONFIG_FILE = 'config/rps_config.json'

def setup_serial_port():
    try:
        port = input("Enter the serial port (e.g., /dev/ttyUSB0 or COM3): ")
        return serial.Serial(port, 9600, timeout=1)
    except serial.SerialException as e:
        print(f"Error: {e}")
        exit(1)
```

```
def send_message(message, ser):
    try:
        ser.write((message + '\n').encode())
    except serial.SerialException as e:
        print(f"Error sending message: {e}")

def receive_message(ser):
    try:
        received = ser.readline().decode('utf-8', errors='ignore').strip()
        if received:
            print(received)
        return received
    except serial.SerialException as e:
        print(f"Error receiving message: {e}")
        return None

def user_input_thread(ser):
    global can_input
    while True:
        if can_input:
            user_message = input("Enter your choice (rock, paper, scissors) or
'exit' to quit: ")
            if user_message.lower() == 'exit':
                print("Exiting...")
                global exit_program
                exit_program = True
                break
            elif user_message.lower().startswith('save'):
                save_game_config(user_message)
            elif user_message.lower().startswith('load'):
                file_path = input("Enter the path to the configuration file:
")
                load_game_config(file_path, ser)
            send_message(user_message, ser)
            can_input = False

def monitor_incoming_messages(ser):
    global can_input
    global last_received_time
    while not exit_program:
        received = receive_message(ser)
        if received:
            last_received_time = time.time()
            if not can_input:
                can_input = True
```

```
def save_game_config(message):
    config = {
        "gameMode": "classic",  # classic or extended
    }

    try:
        params = message.split()
        if len(params) == 2 and params[1] in ['classic', 'extended']:
            config["gameMode"] = params[1]

        with open(CONFIG_FILE, 'w') as f:
            json.dump(config, f)
        print(f"Configuration saved to {CONFIG_FILE}")
    except Exception as e:
        print(f"Error saving configuration: {e}")

def load_game_config(file_path, ser):
    try:
        if os.path.exists(file_path):
            with open(file_path, 'r') as f:
                config = json.load(f)
                game_mode = config.get("gameMode", "classic")

                print(f"Game Mode: {game_mode}")

                json_message = {
                    "gameMode": game_mode
                }

                json_str = json.dumps(json_message)
                print(json_str)

                send_message(json_str, ser)
        else:
            print("Configuration file not found. Please provide a valid
path.")
    except Exception as e:
        print(f"Error loading configuration: {e}")

if __name__ == "__main__":
    ser = setup_serial_port()
    can_input = True
    exit_program = False
    last_received_time = time.time()

    threading.Thread(target=monitor_incoming_messages, args=(ser,),
daemon=True).start()
```

```
    threading.Thread(target=user_input_thread, args=(ser,),
daemon=True).start()

    try:
        while not exit_program:
            if time.time() - last_received_time >= 1 and can_input:
                pass
            else:
                time.sleep(0.1)
    except KeyboardInterrupt:
        print("Exit!")
    finally:
        if ser.is_open:
            print("Closing serial port...")
            ser.close()
```

test_serial_communication.py

```
import pytest
from unittest.mock import patch, MagicMock
import serial
import sys
import os
sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__),
'..')))
from main import send_message, receive_message, save_game_config,
load_game_config

def test_send_message():
    mock_serial = MagicMock(spec=serial.Serial)
    send_message("Hello", mock_serial)
    mock_serial.write.assert_called_with(b"Hello\n")

def test_receive_message():
    mock_serial = MagicMock(spec=serial.Serial)
    mock_serial.readline.return_value = b"Test Message\n"
    result = receive_message(mock_serial)
    assert result == "Test Message"

def test_receive_empty_message():
    mock_serial = MagicMock(spec=serial.Serial)
    mock_serial.readline.return_value = b"\n"
    result = receive_message(mock_serial)
    assert result == ""

@patch('builtins.input', return_value='COM3')
def test_serial_port(mock_input):
```

```
    mock_serial = MagicMock(spec=serial.Serial)
    mock_serial.portstr = 'COM3'
    port = 'COM3'
    ser = mock_serial
    assert ser.portstr == port
```

task3.ino

```
#include <Arduino.h>
#include <ArduinoJson.h>

String player1Choice = "";
String player2Choice = "";
bool gameActive = false;
String gameMode = "classic";

struct GameConfig {
  String gameMode;
};

void saveConfig(const GameConfig &config) {
  StaticJsonDocument<200> doc;
  doc["gameMode"] = config.gameMode;

  String output;
  serializeJson(doc, output);
  Serial.println(output);
}

bool loadStringConfig(JsonDocument& doc, const char* key, String& value) {
  if (doc.containsKey(key) && doc[key].is<String>()) {
    value = doc[key].as<String>();
    return true;
  }
  Serial.println(String(key) + " not found or invalid");
  return false;
}

void loadConfig(String jsonConfig) {
  StaticJsonDocument<200> doc;
  DeserializationError error = deserializeJson(doc, jsonConfig);

  if (error) {
    Serial.println("Failed to load configuration");
    return;
  }
```

```
  if (!loadStringConfig(doc, "gameMode", gameMode)) return;

  Serial.println("Configuration loaded!");
}

void processReceivedMessage(String receivedMessage) {
  if (receivedMessage == "new") {
    initializeGame();
  } else if (receivedMessage.startsWith("save")) {
    GameConfig config = { gameMode };
    saveConfig(config);
  } else if (receivedMessage.startsWith("{")) {
    if (receivedMessage.length() > 0) {
        loadConfig(receivedMessage);
    } else {
        Serial.println("No message received");
    }
  } else if (receivedMessage.startsWith("modes")) {
    handleGameMode(receivedMessage);
  } else if (gameActive) {
    processMove(receivedMessage);
  } else {
    Serial.println("No active game. Type 'new' to start.");
  }
}

void initializeGame() {
  gameActive = true;
  Serial.println("New game started! Choose: rock, paper, or scissors.");
}

void handleGameMode(String receivedMessage) {
  if (receivedMessage == "modes classic") {
    gameMode = "classic";
    Serial.println("Game mode: Classic");
  } else if (receivedMessage == "modes extended") {
    gameMode = "extended";
    Serial.println("Game mode: Extended");
  }
}

void processMove(String move) {
  if (player1Choice == "") {
    player1Choice = move;
    Serial.println("Player 1 chose: " + player1Choice);
  } else if (player2Choice == "") {
```

```
    player2Choice = move;
    Serial.println("Player 2 chose: " + player2Choice);
    determineWinner();
  }
}

void determineWinner() {
  if (player1Choice == player2Choice) {
    Serial.println("It's a draw!");
  } else if ((player1Choice == "rock" && player2Choice == "scissors") ||
             (player1Choice == "scissors" && player2Choice == "paper") ||
             (player1Choice == "paper" && player2Choice == "rock")) {
    Serial.println("Player 1 wins!");
  } else {
    Serial.println("Player 2 wins!");
  }
  gameActive = false;
  player1Choice = "";
  player2Choice = "";
}

void setup() {
  Serial.begin(9600);
}

void loop() {
  if (Serial.available() > 0) {
    String receivedMessage = Serial.readStringUntil('\n');
    receivedMessage.trim();
    processReceivedMessage(receivedMessage);
  }
}
```

2. Реалізував збереження конфігурації в форматі JSON:

```
{"gameMode": 0, "player1Symbol": "X", "player2Symbol": "O"}
```

Висновок

Під час виконання завдання №3 було розроблено серверну та клієнтську частини гри, а також реалізовано збереження конфігурації в форматі JSON.

Список використаних джерел

1. PlatformIO Documentation. "What is PlatformIO?". https://docs.platformio.org/en.
2. JSON Official Website. "Introducing JSON". https://www.json.org.