

**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
Демчук Д. П.

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/game_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 receive_multiple_messages(ser, count):
    messages = []
    for _ in range(count):
        message = receive_message(ser)
        if message:
            messages.append(message)
    return messages

def user_input_thread(ser):
    global can_input
    while True:
        if can_input:
            user_message = input()
            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": 0,
        "player1Symbol": 'X',
        "player2Symbol": 'O'
    }

    try:
        params = message.split()
        if len(params) == 2 and params[1] in ['0', '1', '2']:
            config["gameMode"] = int(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", 0)
                player1_symbol = config.get("player1Symbol", 'X')
                player2_symbol = config.get("player2Symbol", 'O')
    
```

```

        print(f"Game Mode: {game_mode}")
        print(f"Player 1 Symbol: {player1_symbol}")
        print(f"Player 2 Symbol: {player2_symbol}")

        json_message = {
            "gameMode": game_mode,
            "player1Symbol": player1_symbol,
            "player2Symbol": player2_symbol
        }

        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>

char board[3][3];
bool gameActive = false;
String player1Symbol = "X";
String player2Symbol = "0";
String currentPlayer = "X";
int gameMode = 0;

struct GameConfig {
    int gameMode;
    String player1Symbol;
    String player2Symbol;
    String currentPlayer;
};

void saveConfig(const GameConfig &config) {
    StaticJsonDocument<200> doc;
    doc["gameMode"] = config.gameMode;
    doc["player1Symbol"] = config.player1Symbol;
    doc["player2Symbol"] = config.player2Symbol;
    doc["currentPlayer"] = config.currentPlayer;

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

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

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

    if (doc.containsKey("gameMode")) {
        gameMode = doc["gameMode"].as<int>();
    } else {
        Serial.println("gameMode not found");
        return;
    }
}
```

```

    if (doc.containsKey("player1Symbol") && doc["player1Symbol"].is<String>()) {
        player1Symbol = doc["player1Symbol"].as<String>();
    } else {
        Serial.println("player1Symbol not found or invalid");
        return;
    }

    if (doc.containsKey("player2Symbol") && doc["player2Symbol"].is<String>()) {
        player2Symbol = doc["player2Symbol"].as<String>();
    } else {
        Serial.println("player2Symbol not found or invalid");
        return;
    }

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

void initializeBoard() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            board[i][j] = ' ';
        }
    }
}

void printBoard() {
    String boardState = "Board state:\n";

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == 'X' || board[i][j] == 'O') {
                boardState += board[i][j];
            } else {
                boardState += ' ';
            }
            if (j < 2) boardState += "|";
        }
        if (i < 2) boardState += "\n---\n";
        else boardState += "\n";
    }

    Serial.println(boardState);
}

bool checkWin(char player) {

```



```

    for (int i = 0; i < 3; i++) {
        if ((board[i][0] == player && board[i][1] == player && board[i][2] ==
player) ||
            (board[0][i] == player && board[1][i] == player && board[2][i] ==
player)) {
            return true;
        }
    }

    if ((board[0][0] == player && board[1][1] == player && board[2][2] == player)
||
        (board[0][2] == player && board[1][1] == player && board[2][0] == player))
    {
        return true;
    }

    return false;
}

bool isBoardFull() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == ' ') {
                return false;
            }
        }
    }
    return true;
}

void aiMove(char aiSymbol) {
    if (blockOpponentMove(aiSymbol == 'X' ? 'O' : 'X')) {
        return;
    }

    int startX = random(3);
    int startY = random(3);

    if (random(2) == 0) {
        startX = 0;
        startY = 0;
    }

    for (int i = startX; i < 3; i++) {
        for (int j = startY; j < 3; j++) {
            if (board[i][j] == ' ') {
                board[i][j] = aiSymbol;
            }
        }
    }
}

```

```

        Serial.println("AI played at: " + String(i + 1) + " " + String(j + 1));
        return;
    }
}

for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        if (board[i][j] == ' ') {
            board[i][j] = aiSymbol;
            Serial.println("AI played randomly at: " + String(i + 1) + " " +
String(j + 1));
            return;
        }
    }
}

bool blockOpponentMove(char opponent) {
    for (int i = 0; i < 3; i++) {
        // Горизонтальні та вертикальні лінії
        if (canBlock(i, 0, i, 1, i, 2, opponent)) {
            return true;
        }
        if (canBlock(0, i, 1, i, 2, i, opponent)) {
            return true;
        }
    }

    if (canBlock(0, 0, 1, 1, 2, 2, opponent)) {
        return true;
    }
    if (canBlock(0, 2, 1, 1, 2, 0, opponent)) {
        return true;
    }

    return false;
}

bool canBlock(int x1, int y1, int x2, int y2, int x3, int y3, char opponent) {
    if (board[x1][y1] == opponent && board[x2][y2] == opponent && board[x3][y3] ==
' ') {
        board[x3][y3] = '0';
        Serial.println("AI blocked opponent's winning move at: " + String(x3 + 1) +
" " + String(y3 + 1));
        return true;
    }
}

```

```

    if (board[x1][y1] == opponent && board[x2][y2] == ' ' && board[x3][y3] ==
opponent) {
        board[x2][y2] = '0';
        Serial.println("AI blocked opponent's winning move at: " + String(x2 + 1) +
" " + String(y2 + 1));
        return true;
    }
    if (board[x1][y1] == ' ' && board[x2][y2] == opponent && board[x3][y3] ==
opponent) {
        board[x1][y1] = '0';
        Serial.println("AI blocked opponent's winning move at: " + String(x1 + 1) +
" " + String(y1 + 1));
        return true;
    }

    return false;
}

void processMove(String input) {
    int row = input[0] - '1';
    int col = input[2] - '1';

    if (row ≥ 0 && row < 3 && col ≥ 0 && col < 3 && board[row][col] == ' ') {
        if (gameMode == 1) {
            board[row][col] = (currentPlayer == "X") ? 'X' : '0';
        } else {
            board[row][col] = 'X';
        }
        printBoard();

        if (checkWin('X')) {
            Serial.println("Player X wins!");
            gameActive = false;
            return;
        }

        if (checkWin('0')) {
            Serial.println("Player 0 wins!");
            gameActive = false;
            return;
        }

        if (isBoardFull()) {
            Serial.println("It's a draw!");
            gameActive = false;
            return;
        }
    }
}

```

```

    if (gameMode == 2) {
        aiMove(player1Symbol[0]);
        if (checkWin(player1Symbol[0])) {
            Serial.println("Player 1 (AI) wins!");
            gameActive = false;
            return;
        }
        aiMove(player2Symbol[0]);
        if (checkWin(player2Symbol[0])) {
            Serial.println("Player 2 (AI) wins!");
            gameActive = false;
            return;
        }
    }

    currentPlayer = (currentPlayer == "X") ? 'O' : 'X';
} else {
    Serial.println("Invalid move, try again.");
}
}

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

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

        if (receivedMessage == "new") {
            initializeBoard();
            gameActive = true;

            if (gameMode == 1) {
                Serial.println("Player 1, choose your symbol: X or O");
                currentPlayer = (random(2) == 0) ? 'X' : 'O';
                player1Symbol = currentPlayer;
                player2Symbol = (currentPlayer == "X") ? 'O' : 'X';
                Serial.println("Player 1 is " + String(player1Symbol));
                Serial.println("Player 2 is " + String(player2Symbol));
            } else {
                currentPlayer = 'X';
            }
        }
    }
}

```

```

        Serial.println("New game started! " + String(currentPlayer) + " goes
first.");
        printBoard();

        if (gameMode == 0) {
            while (gameActive) {
                if (currentPlayer == "X") {
                    Serial.println("Your move, player (enter row and column):");
                    while (Serial.available() == 0) {
                        ;
                    }
                    String userMove = Serial.readStringUntil('\n');
                    processMove(userMove);
                    printBoard();

                    if (checkWin('X')) {
                        Serial.println("Player X wins!");
                        gameActive = false;
                        break;
                    }
                    if (isBoardFull()) {
                        Serial.println("It's a draw!");
                        gameActive = false;
                        break;
                    }

                    currentPlayer = 'O';
                } else {
                    aiMove('O');
                    printBoard();
                    if (checkWin('O')) {
                        Serial.println("AI O wins!");
                        gameActive = false;
                        break;
                    }
                    if (isBoardFull()) {
                        Serial.println("It's a draw!");
                        gameActive = false;
                        break;
                    }

                    currentPlayer = 'X';
                }
            }
        }
        else if (gameMode == 2) {
            while (gameActive) {
                aiMove('X');
            }
        }
    }
}

```

```

        printBoard();
        if (checkWin('X')) {
            Serial.println("AI X wins!");
            gameActive = false;
            break;
        }
        if (isBoardFull()) {
            Serial.println("It's a draw!");
            gameActive = false;
            break;
        }

        aiMove('O');
        printBoard();
        if (checkWin('O')) {
            Serial.println("AI O wins!");
            gameActive = false;
            break;
        }
        if (isBoardFull()) {
            Serial.println("It's a draw!");
            gameActive = false;
            break;
        }
    }
}

} else if (receivedMessage.startsWith("save")) {
    GameConfig config = { gameMode, player1Symbol, player2Symbol,
currentPlayer };
    saveConfig(config);
} else if (receivedMessage.startsWith("{")) {
    if (receivedMessage.length() > 0) {
        loadConfig(receivedMessage);
    } else {
        Serial.println("No message received");
    }
} else if (receivedMessage.startsWith("modes")) {
    if (receivedMessage == "modes 0") {
        gameMode = 0;
        Serial.println("Game mode: Man vs AI");
    } else if (receivedMessage == "modes 1") {
        gameMode = 1;
        Serial.println("Game mode: Man vs Man");
    } else if (receivedMessage == "modes 2") {
        gameMode = 2;
        Serial.println("Game mode: AI vs AI");
    }
}

```

```
} else if (gameActive) {  
    processMove(receivedMessage);  
} else {  
    Serial.println("No active game. Type 'new' to start.");  
}  
}  
}
```

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>.