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
Гербей О. М.

Прийняв: Федак П. Р.

2024

Опис теми

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

Виконання завдання

1. Розробив серверну та клієнтську частину гри:

Main.cpp

```
#include <SFML/Graphics.hpp>
#include <iostream>
#include <windows.h> // Для роботи з серійним портом на Windows
#include "D:/simpleini-master/simpleini-master/SimpleIni.h"

const int SIZE_BOARD = 3; // Розмір дошки 3x3
const int TILE_SIZE = 100; // Розмір однієї клітини
char board[SIZE_BOARD][SIZE_BOARD] = { {' ', ' ', ' '}, {' ', ' ', ' '}, {'
', ' ', ' '} }; // Ігрова дошка

HANDLE hSerial;
DCB dcbSerialParams = { 0 };
COMMTIMEOUTS timeouts = { 0 };


bool openSerialPort(const char* portName) {
    hSerial = CreateFileA(portName, GENERIC_READ | GENERIC_WRITE, 0, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hSerial == INVALID_HANDLE_VALUE) {
        return false;
    }

    dcbSerialParams.DCBlength = sizeof(dcbSerialParams);
    if (!GetCommState(hSerial, &dcbSerialParams)) {
        return false;
    }

    dcbSerialParams.BaudRate = CBR_4800;
    dcbSerialParams.ByteSize = 8;
    dcbSerialParams.StopBits = ONESTOPBIT;
    dcbSerialParams.Parity = NOPARITY;
    if (!SetCommState(hSerial, &dcbSerialParams)) {
        return false;
```

```
    }

    timeouts.ReadIntervalTimeout = 50;
    timeouts.ReadTotalTimeoutConstant = 50;
    timeouts.ReadTotalTimeoutMultiplier = 10;
    timeouts.WriteTotalTimeoutConstant = 50;
    timeouts.WriteTotalTimeoutMultiplier = 10;
    if (!SetCommTimeouts(hSerial, &timeouts)) {
        return false;
    }

    return true;
}

void updateBoardFromSerial(const std::string& response) {
    if (response.length() < SIZE_BOARD * SIZE_BOARD) {
        return; // Вийти з функції, якщо недостатньо даних
    }

    int index = 0;
    for (int i = 0; i < SIZE_BOARD; ++i) {
        for (int j = 0; j < SIZE_BOARD; ++j) {
            board[i][j] = response[index++];
        }
    }
}

struct Stats {
    // PvP
    int pvpGames = 0;
    int winsX = 0;
    int lossesX = 0;
    int drawsX = 0;
    int winsO = 0;
    int lossesO = 0;
    int drawsO = 0;

    // AI Player First
    int Games = 0;
    int Wins = 0;
    int Draws = 0;
    int Losses = 0;
    int Winrate = 0;

};
```

```
Stats stats;

void saveStatsToExistingINI(const std::string& filename) {
    CSimpleIniA ini;
    ini.SetUnicode();

    // Завантажуємо існуючий INI-файл
    if (ini.LoadFile(filename.c_str()) != SI_OK) {
        std::cout << "Failed to load INI file." << std::endl;
        return;
    }

    // Зберігаємо статистику PvP
    ini.SetLongValue("Stats_PvP", "Games", stats.pvpGames);
    ini.SetLongValue("Stats_PvP", "WinsX", stats.winsX);
    ini.SetLongValue("Stats_PvP", "LossesX", stats.lossesX);
    ini.SetLongValue("Stats_PvP", "DrawsX", stats.drawsX);
    ini.SetLongValue("Stats_PvP", "WinsO", stats.winsO);
    ini.SetLongValue("Stats_PvP", "LossesO", stats.lossesO);
    ini.SetLongValue("Stats_PvP", "DrawsO", stats.drawsO);

    // Зберігаємо статистику для AI-гри (гравець перший)
    ini.SetLongValue("Stats ", "Games", stats.Games);
    ini.SetLongValue("Stats ", "Wins", stats.Wins);
    ini.SetLongValue("Stats ", "Draws", stats.Draws);
    ini.SetLongValue("Stats ", "Losses", stats.Losses);
    ini.SetLongValue("Stats ", "Winrate", stats.Winrate);


    // Зберігаємо зміни в INI-файл
    if (ini.SaveFile(filename.c_str()) != SI_OK) {
        std::cout << "Failed to save INI file." << std::endl;
    }
    else {
        std::cout << "Stats saved to existing INI file successfully." <<
std::endl;
    }
}

void loadStatsFromExistingINI(const std::string& filename) {
    CSimpleIniA ini;
    ini.SetUnicode();

    // Завантажуємо існуючий INI-файл
    if (ini.LoadFile(filename.c_str()) != SI_OK) {
        std::cout << "Failed to load INI file." << std::endl;
```

```
        return;
    }

    // Завантажуємо статистику PvP
    stats.pvpGames = ini.GetLongValue("Stats_PvP", "Games", 0);
    stats.winsX = ini.GetLongValue("Stats_PvP", "WinsX", 0);
    stats.lossesX = ini.GetLongValue("Stats_PvP", "LossesX", 0);
    stats.drawsX = ini.GetLongValue("Stats_PvP", "DrawsX", 0);
    stats.winsO = ini.GetLongValue("Stats_PvP", "WinsO", 0);
    stats.lossesO = ini.GetLongValue("Stats_PvP", "LossesO", 0);
    stats.drawsO = ini.GetLongValue("Stats_PvP", "DrawsO", 0);

    // Завантажуємо статистику для AI-гри (гравець перший)
    stats.Games = ini.GetLongValue("Stats ", "Games", 0);
    stats.Wins = ini.GetLongValue("Stats ", "Wins", 0);
    stats.Draws = ini.GetLongValue("Stats ", "Draws", 0);
    stats.Losses = ini.GetLongValue("Stats ", "Losses", 0);
    stats.Winrate = ini.GetLongValue("Stats ", "Winrate", 0);


    std::cout << "Stats loaded from existing INI file successfully." <<
std::endl;
}

void loadConfig(const std::string& filename, bool& blueLedState, bool&
yellowLedState) {
    CSimpleIniA ini;
    ini.SetUnicode();
    if (ini.LoadFile(filename.c_str()) != SI_OK) {
        std::cout << "Failed to load INI file." << std::endl;
        return;
    }

    // Завантажуємо стан діодів
    blueLedState = ini.GetBoolValue("LEDs", "Blue", false);
    yellowLedState = ini.GetBoolValue("LEDs", "Yellow", false);

    // Лог станів
    std::cout << "Blue LED: " << (blueLedState ? "ON" : "OFF") <<
std::endl;
    std::cout << "Yellow LED: " << (yellowLedState ? "ON" : "OFF") <<
std::endl;
}

void saveConfig(const std::string& filename, bool& blueLedState, bool&
yellowLedState) {
```

```
    CSimpleIniA ini;
    ini.SetUnicode();
    if (ini.LoadFile(filename.c_str()) != SI_OK) {
        std::cout << "Failed to load INI file." << std::endl; // Замінено
Serial.println
        return;
    }

    // Записуємо стан діодів
    ini.SetBoolValue("LEDs", "Blue", blueLedState);
    ini.SetBoolValue("LEDs", "Yellow", yellowLedState);

    // Зберігаємо зміни
    if (ini.SaveFile(filename.c_str()) != SI_OK) {
        std::cout << "Failed to save INI file." << std::endl; // Замінено
Serial.println
    }
    else {
        std::cout << "Configuration saved successfully." << std::endl; //
Лог успіху
    }
}

void clearSerialBuffer() {
    char buffer[256];
    DWORD bytes_read;
    while (ReadFile(hSerial, buffer, sizeof(buffer), &bytes_read, NULL) &&
bytes_read > 0) {
        // Просто зчитуємо всі дані в буфер, нічого не роблячи
    }
}

void writeSerialPort(const std::string& data) {

    DWORD bytes_written;
    if (WriteFile(hSerial, data.c_str(), data.size(), &bytes_written,
NULL)) {
        std::cout << "[Frontend] Sent to Arduino: " << data << std::endl;
// Лог даних, які відправляються
    }
    else {
        std::cout << "[Frontend] Error sending to Arduino!" << std::endl;
// Лог помилки
    }

}
```

```
std::string readSerialPort() {
    char buffer[256] = { 0 }; // Ініціалізуйте буфер нулями
    DWORD bytes_read;
    if (ReadFile(hSerial, buffer, sizeof(buffer) - 1, &bytes_read, NULL)) {
        buffer[bytes_read] = '\0'; // Додайте термінальний нуль
        std::cout << "[Backend] Received from Arduino: " << buffer <<
std::endl; // Лог отриманих даних
        return std::string(buffer);

    }
    std::cout << "[Frontend] Error reading from Arduino!" << std::endl; //
Лог помилки
    return ""; // Повертаємо пустий рядок у разі невдачі
}
void drawBoard(sf::RenderWindow& window) {
    for (int i = 0; i <= SIZE_BOARD; ++i) {
        // Горизонтальні лінії
        sf::RectangleShape horizontalLine(sf::Vector2f(TILE_SIZE *
SIZE_BOARD, 5));
        horizontalLine.setPosition(0, i * TILE_SIZE);
        horizontalLine.setFillColor(sf::Color::Black);
        window.draw(horizontalLine);

        // Вертикальні лінії
        sf::RectangleShape verticalLine(sf::Vector2f(5, TILE_SIZE *
SIZE_BOARD));
        verticalLine.setPosition(i * TILE_SIZE - 4, 0); // Додаємо поправку
        verticalLine.setFillColor(sf::Color::Black);
        window.draw(verticalLine);
    }
}

void drawMarks(sf::RenderWindow& window, sf::Font& font) {
    for (int i = 0; i < SIZE_BOARD; ++i) {
        for (int j = 0; j < SIZE_BOARD; ++j) {
            if (board[i][j] != ' ') {
                sf::Text text;
                text.setFont(font);
                text.setString(board[i][j]);
                text.setCharacterSize(100);
                text.setPosition(j * TILE_SIZE + 15, i * TILE_SIZE - 20);
                text.setFillColor(sf::Color::Black);
                window.draw(text);
            }
        }
```

```
    }
}

// Функція скидання дошки на клієнтській стороні
void resetBoard() {
    for (int i = 0; i < SIZE_BOARD; ++i) {
        for (int j = 0; j < SIZE_BOARD; ++j) {
            board[i][j] = ' '; // Очищуємо дошку
        }
    }
}

//функція для малювання ігрових елементів
void drawGame(sf::RenderWindow& window, sf::Font& font, sf::RectangleShape
playerFirstButton, sf::Text playerFirstText, sf::RectangleShape
aiFirstButton, sf::Text aiFirstText, sf::RectangleShape restartButton,
sf::Text restartText, sf::RectangleShape pvpButton, sf::Text pvpText,
sf::RectangleShape settingsButton, sf::Text settingsText) {
     window.clear(sf::Color::White);   // Очищуємо вікно
     drawBoard(window);                // Малюємо дошку
     drawMarks(window, font);          // Малюємо мітки (хрестики і
нулики)
    window.draw(playerFirstButton);   // Малюємо кнопку вибору черговості
    window.draw(playerFirstText);     // Текст на кнопці "Player First"
    window.draw(aiFirstButton);       // Малюємо кнопку вибору черговості
    window.draw(aiFirstText);         // Текст на кнопці "AI First"
    window.draw(restartButton);       // Малюємо кнопку рестарту
    window.draw(restartText);         // Малюємо текст на кнопці рестарту
    window.draw(pvpButton);           // Малюємо кнопку PvP
    window.draw(pvpText);             // Малюємо текст на кнопці PvP
    window.draw(settingsButton);       // Малюємо кнопку налаштувань
    window.draw(settingsText);         // Малюємо текст на кнопці
налаштувань
    window.display();                 // Відображаємо все це у вікні

}

void drawSettingsMenu(sf::RenderWindow& settingsWindow, sf::Font& font,
sf::RectangleShape& blueLedButton, sf::Text& blueLedText,
sf::RectangleShape& yellowLedButton, sf::Text& yellowLedText) {
    settingsWindow.clear(sf::Color::White);

    // Кнопка для керування синім діодом
    settingsWindow.draw(blueLedButton);
    settingsWindow.draw(blueLedText);
```

```
    // Кнопка для керування жовтим діодом
    settingsWindow.draw(yellowLedButton);
    settingsWindow.draw(yellowLedText);

    settingsWindow.display();
}

void openSettingsMenu(sf::Font& font, bool& blueLedState, bool&
yellowLedState) {
    sf::RenderWindow settingsWindow(sf::VideoMode(400, 300), "Settings");

    // Кнопка для керування синім діодом
    sf::RectangleShape blueLedButton(sf::Vector2f(200, 50));
    blueLedButton.setPosition(100, 50);
    blueLedButton.setFillColor(sf::Color::Blue);

    sf::Text blueLedText;
    blueLedText.setFont(font);
    blueLedText.setCharacterSize(20);
    blueLedText.setFillColor(sf::Color::White);

    // Кнопка для керування жовтим діодом
    sf::RectangleShape yellowLedButton(sf::Vector2f(200, 50));
    yellowLedButton.setPosition(100, 150);
    yellowLedButton.setFillColor(sf::Color::Yellow);

    sf::Text yellowLedText;
    yellowLedText.setFont(font);
    yellowLedText.setCharacterSize(20);
    yellowLedText.setFillColor(sf::Color::Black);

    while (settingsWindow.isOpen()) {
        sf::Event event;
        while (settingsWindow.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                settingsWindow.close();
            }

            if (event.type == sf::Event::MouseButtonPressed) {
                int mouseX = event.mouseButton.x;
                int mouseY = event.mouseButton.y;

                if (blueLedButton.getGlobalBounds().contains(mouseX,
mouseY)) {
                    blueLedState = !blueLedState; // Змінюємо стан
```

```
saveConfig("D:/scad/csad2425Ki401HerbeiOleksandr03/config/config.ini",
blueLedState, yellowLedState);
                    writeSerialPort("BLed\n");
                }
                else if (yellowLedButton.getGlobalBounds().contains(mouseX,
mouseY)) {
                    yellowLedState = !yellowLedState; // Змінюємо стан

saveConfig("D:/scad/csad2425Ki401HerbeiOleksandr03/config/config.ini",
blueLedState, yellowLedState);
                    writeSerialPort("Yled\n");
                }
            }
        }

        // Оновлюємо текст кнопок залежно від стану діодів
        blueLedText.setString(blueLedState ? "Blue LED: ON" : "Blue LED:
OFF");
        blueLedText.setPosition(blueLedButton.getPosition().x + 20,
blueLedButton.getPosition().y + 10);

        yellowLedText.setString(yellowLedState ? "Yellow LED: ON" : "Yellow
LED: OFF");
        yellowLedText.setPosition(yellowLedButton.getPosition().x + 20,
yellowLedButton.getPosition().y + 10);

        drawSettingsMenu(settingsWindow, font, blueLedButton, blueLedText,
yellowLedButton, yellowLedText);
    }
}



int main() {
    sf::RenderWindow window(sf::VideoMode(TILE_SIZE * SIZE_BOARD, TILE_SIZE
* SIZE_BOARD + 200), "Tic-Tac-Toe with Arduino Backend");
    sf::Font font;
    if (!font.loadFromFile("C:/Windows/Fonts/Arial.ttf")) {
        return 1;
    }

    if (!openSerialPort("COM7")) {
        return 1;
    }

    bool blueLedState;
```

```
    bool yellowLedState;

    bool gameOver = false;
    bool resetRequested = false; // Додаємо змінну для фіксації запиту на
скидання
    loadConfig("D:/scad/csad2425Ki401HerbeiOleksandr03/config/config.ini",
blueLedState, yellowLedState);

loadStatsFromExistingINI("D:/scad/csad2425Ki401HerbeiOleksandr03/config/con
fig.ini");
    // Створення кнопок для вибору черговості
    sf::RectangleShape playerFirstButton(sf::Vector2f(150, 50)); // Кнопка
вибору черговості гравця
    playerFirstButton.setPosition((TILE_SIZE * SIZE_BOARD - 300) / 2,
TILE_SIZE * SIZE_BOARD + 20);
    playerFirstButton.setFillColor(sf::Color::Blue);
    sf::Text playerFirstText;
    playerFirstText.setFont(font);
    playerFirstText.setString("Player First");
    playerFirstText.setCharacterSize(24);
    playerFirstText.setFillColor(sf::Color::White);
    playerFirstText.setPosition(playerFirstButton.getPosition().x + 10,
playerFirstButton.getPosition().y + 10);

    sf::RectangleShape aiFirstButton(sf::Vector2f(150, 50)); // Кнопка
вибору черговості AI
    aiFirstButton.setPosition((TILE_SIZE * SIZE_BOARD + 50) / 2, TILE_SIZE
* SIZE_BOARD + 20);
    aiFirstButton.setFillColor(sf::Color::Red);
    sf::Text aiFirstText;
    aiFirstText.setFont(font);
    aiFirstText.setString("AI First");
    aiFirstText.setCharacterSize(24);
    aiFirstText.setFillColor(sf::Color::White);
    aiFirstText.setPosition(aiFirstButton.getPosition().x + 10,
aiFirstButton.getPosition().y + 10);

    sf::RectangleShape restartButton(sf::Vector2f(150, 50)); // Кнопка
перезавантаження
    restartButton.setPosition(aiFirstButton.getPosition().x,
aiFirstButton.getPosition().y + 60); // Позиція праворуч під AI
    restartButton.setFillColor(sf::Color::Green);
    sf::Text restartText;
    restartText.setFont(font);
    restartText.setString("Restart");
    restartText.setCharacterSize(24);
```

```
    restartText.setFillColor(sf::Color::White);
    restartText.setPosition(restartButton.getPosition().x + 10,
restartButton.getPosition().y + 10);

    sf::RectangleShape pvpButton(sf::Vector2f(150, 50)); // Кнопка PvP
    pvpButton.setPosition(playerFirstButton.getPosition().x,
playerFirstButton.getPosition().y + 60); // Позиція під Player First
    pvpButton.setFillColor(sf::Color::Yellow);
    sf::Text pvpText;
    pvpText.setFont(font);
    pvpText.setString("PvP");
    pvpText.setCharacterSize(24);
    pvpText.setFillColor(sf::Color::Black);
    pvpText.setPosition(pvpButton.getPosition().x + 10,
pvpButton.getPosition().y + 10);

    // кнопка для налаштування
    sf::RectangleShape settingsButton(sf::Vector2f(150, 50)); // Кнопка
налаштувань
    settingsButton.setPosition(pvpButton.getPosition().x,
pvpButton.getPosition().y + 60); // Позиція під PvP
    settingsButton.setFillColor(sf::Color::Magenta);
    sf::Text settingsText;
    settingsText.setFont(font);
    settingsText.setString("Settings");
    settingsText.setCharacterSize(24);
    settingsText.setFillColor(sf::Color::Black);
    settingsText.setPosition(settingsButton.getPosition().x + 10,
settingsButton.getPosition().y + 10);


    // Початкове відображення елементів у вікні після відкриття
    drawGame(window,font, playerFirstButton, playerFirstText,
aiFirstButton, aiFirstText, restartButton, restartText, pvpButton, pvpText,
settingsButton, settingsText);



    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                window.close();
            }

            if (event.type == sf::Event::MouseButtonPressed) {
```

```
                int mouseX = event.mouseButton.x;
                int mouseY = event.mouseButton.y;

                if (restartButton.getGlobalBounds().contains(mouseX,
mouseY)) {
                    writeSerialPort("reset\n");
                    resetRequested = true; // Запит на скидання без
очищення дошки
                }
                else if
(playerFirstButton.getGlobalBounds().contains(mouseX, mouseY)) {
                    writeSerialPort("player\n");
                    resetBoard(); // Очистити дошку
                    resetRequested = true; // Запит на скидання з очищенням
дошки
                }
                else if (aiFirstButton.getGlobalBounds().contains(mouseX,
mouseY)) {
                    writeSerialPort("ai\n");
                    resetBoard(); // Очистити дошку
                    resetRequested = true; // Запит на скидання з очищенням
дошки
                }
                else if (settingsButton.getGlobalBounds().contains(mouseX,
mouseY)) {
                    openSettingsMenu(font, yellowLedState, blueLedState);
                }
                else if (pvpButton.getGlobalBounds().contains(mouseX,
mouseY)) {
                    writeSerialPort("pvp\n");
                    resetBoard(); // Очистити дошку
                    resetRequested = true; // Запит на скидання з очищенням
дошки
                }
                else if (!gameOver && !resetRequested) { // Не обробляємо
хід, якщо очікуємо на скидання
                    int row = mouseY / TILE_SIZE;
                    int col = mouseX / TILE_SIZE;

                    if (row < SIZE_BOARD && col < SIZE_BOARD &&
board[row][col] == ' ') {
                        std::string move = std::to_string(row) + "," +
std::to_string(col) + "\n";
                        writeSerialPort(move);

                        // Дочекайтеся відповіді від Arduino
```

```
                        std::string response = readSerialPort();
                        updateBoardFromSerial(response);

                        // Обробка результату гри
                        if (response.find("X win!") != std::string::npos) {
                            stats.winsX++;
                            stats.lossesO++;
                            stats.pvpGames++;

                            gameOver = true;
                        }
                        else if (response.find("O win!") !=
std::string::npos) {
                            stats.winsO++;
                            stats.lossesX++;
                            stats.pvpGames++;
                            gameOver = true;
                        }
                        else if (response.find("AI win!") !=
std::string::npos) {
                            stats.Losses++;
                            stats.Games++;
                            stats.Winrate = (stats.Wins / stats.Games) *
100;
                            gameOver = true;
                        }
                        else if (response.find("You win!") !=
std::string::npos) {
                            stats.Wins++;
                            stats.Games++;
                            stats.Winrate = (stats.Wins / stats.Games) *
100;
                            gameOver = true;
                        }
                        else if (response.find("Draw!") !=
std::string::npos) {
                            stats.drawsX++;
                            stats.drawsO++;
                            stats.Draws++;
                            stats.Games++;
                            gameOver = true;
                        }

                        // Зберігаємо статистику після завершення гри
                        if (gameOver) {
```

```
saveStatsToExistingINI("D:/scad/csad2425Ki401HerbeiOleksandr03/config/confi
g.ini");
                        }

                        // Оновити відображення дошки після кожного ходу
                        drawGame(window, font, playerFirstButton,
playerFirstText, aiFirstButton, aiFirstText, restartButton, restartText,
pvpButton, pvpText, settingsButton, settingsText);
                    }
                }
            }
        }

        // Перевірка на відповідь від Arduino після запиту скидання
        if (resetRequested) {
            std::string response = readSerialPort();
            if (!response.empty()) {
                updateBoardFromSerial(response);
                resetRequested = false; // Завершили скидання
                gameOver = false;       // Гра триває
                drawGame(window, font, playerFirstButton, playerFirstText,
aiFirstButton, aiFirstText, restartButton, restartText, pvpButton, pvpText,
settingsButton, settingsText);
            }
        }
    }


    CloseHandle(hSerial); // Закрити серійний порт після виходу
    return 0;
}
```

2. Розробив серверну частину

```
#include <Arduino.h>
#include <EEPROM.h> // Для збереження стану діодів між перезавантаженнями

struct Pair {
    int first;
    int second;
};
```

```
const int BlueledPin = 8; // Pin for Led
const int YellowledPin = 9;
char receivedData[10];
int dataIndex = 0;
char board[3][3] = {{' ', ' ', ' '}, {' ', ' ', ' '}, {' ', ' ', ' '}}; //
gameBoard
bool gameOver = false;
int moveCount = 0;
int blinkCount = 0;
bool ledState = LOW;
char ai = 'O';
char player = 'X';
bool waitingForPlayerMove = false;
bool isPlayerOneTurn = true; // Змінна для відстеження черги ходу гравців
bool pvpmode = false;
bool playerTurn = true; // Хто ходить: true - гравець X, false - гравець O
bool blueLedState = false;
bool yellowLedState = false;

void setup(){
    Serial.begin(4800);
    pinMode(BlueledPin, OUTPUT);
    pinMode(YellowledPin, OUTPUT);
    loadLedStateFromEEPROM(); // Завантаження стану діодів
    while (Serial.available() > 0) {
        Serial.read(); // Чистимо серійний буфер
    }
    resetBoard();
}

void loop() {
    // Читання серійної команди
    if (Serial.available() > 0) {
        char receivedChar = Serial.read();
        if (receivedChar == '\n') {
            receivedData[dataIndex] = '\0'; // Завершуємо рядок
            processCommand(); // Обробка команди
            dataIndex = 0;    // Скидання індексу після обробки
        } else if (dataIndex < sizeof(receivedData) - 1) {
            receivedData[dataIndex++] = receivedChar;
        } else {
            dataIndex = 0; // Скидання буфера при переповненні
            memset(receivedData, 0, sizeof(receivedData));
        }
    }
```

```
}

void processCommand() {
    if (strlen(receivedData) > 9) {
        Serial.println("Error: Command too long!");
        memset(receivedData, 0, sizeof(receivedData)); // Очищення буфера
        return;
    }

    if (strcmp(receivedData, "My move:") == 0) {
        Serial.println("Nice but");
    }

    if (strcmp(receivedData, "BLed") == 0) {
        BlueblinkLED();
        blueLedState = !blueLedState; // Змінюємо стан синього діода
        saveLedStateToEEPROM(); // Зберігаємо стан у EEPROM

    }
    if (strcmp(receivedData, "Yled") == 0) {
        YellowblinkLED();
        yellowLedState = !yellowLedState; // Змінюємо стан жовтого діода
        saveLedStateToEEPROM(); // Зберігаємо стан у EEPROM

    }
    // Якщо отримана команда - reset
    if (strcmp(receivedData, "reset") == 0) {
        DrawblinkLED();
        resetBoard();
        waitingForPlayerMove = false;
        pvpmode = false; // Вихід із PvP
        playerTurn = true; // Починаємо з першого гравця
        return;
    }

    // Якщо гра ще не завершена
    if (!gameOver) {
        // Обробка режимів гри
        if (strcmp(receivedData, "player") == 0) {
            sendCurrentBoardState();
            waitingForPlayerMove = true; // Очікуємо хід гравця
            pvpmode = false; // Вимикаємо PvP
            memset(receivedData, 0, sizeof(receivedData));
            return;
        } else if (strcmp(receivedData, "ai") == 0) {
            makeAIMove();
```

```
            sendCurrentBoardState();
            waitingForPlayerMove = true; // Після ходу AI чекаємо на хід
гравця
            pvpmode = false;
            memset(receivedData, 0, sizeof(receivedData));
            return;
        } else if (strcmp(receivedData, "pvp") == 0) {
            pvpmode = true;
            waitingForPlayerMove = false; // Вимикаємо інші режими
            playerTurn = true; // Перший гравець X
            sendCurrentBoardState();
            memset(receivedData, 0, sizeof(receivedData));
            return;
        }

        // Обробка ходу (один метод для всіх режимів)
        if (receivedData[1] == ',' && receivedData[0] >= '0' &&
            receivedData[0] <= '2' && receivedData[2] >= '0' &&
            receivedData[2] <= '2') {

            int row = receivedData[0] - '0';
            int col = receivedData[2] - '0';

            if (board[row][col] == ' ') { // Якщо клітинка порожня
                // У PvP змінюємо черговість гравців
                if (pvpmode) {
                    board[row][col] = playerTurn ? 'X' : 'O';
                    playerTurn = !playerTurn; // Змінюємо гравця
                } else {
                    board[row][col] = 'X'; // Хід гравця X
                    makeAIMove(); // Хід AI

                }
                moveCount++;
                sendCurrentBoardState();

                // Перевірка результату гри
                if (checkWinner()) {
                    gameOver = true;
                    if (pvpmode) {
                        // PvP режим
                        if (playerTurn) {
                            YellowblinkLED(); // Жовтий діод для гравця O
                            Serial.println("O win!");
                        } else {
                            BlueblinkLED(); // Синій діод для гравця X
```

```
                        Serial.println("X win!");
                    }
                } else {
                    // AI режим
                    if (isAIMoveWinning()) {
                        YellowblinkLED(); // Жовтий діод для AI
                        Serial.println("AI win!");
                    } else {
                        BlueblinkLED(); // Синій діод для гравця
                        Serial.println("You win!");
                    }
                }
            } else if (moveCount >= 9) {
                gameOver = true;
                DrawblinkLED(); // Обидва діоди блимають
                Serial.println("Draw!");
            }
        }
        memset(receivedData, 0, sizeof(receivedData));
        dataIndex = 0;
    }
}

// Функція для збереження стану діодів у EEPROM
void saveLedStateToEEPROM() {
    if (EEPROM.read(0) != blueLedState) {
        EEPROM.write(0, blueLedState);

    }
    if (EEPROM.read(1) != yellowLedState) {
        EEPROM.write(1, yellowLedState);

    }
}

// Функція для завантаження стану діодів із EEPROM
void loadLedStateFromEEPROM() {
    int blueState = EEPROM.read(0);
    int yellowState = EEPROM.read(1);

    // Перевірка, чи значення є допустимим
    blueLedState = (blueState == 1);
    yellowLedState = (yellowState == 1);

    digitalWrite(BlueledPin, blueLedState ? HIGH : LOW);
```

```
    digitalWrite(YellowledPin, yellowLedState ? HIGH : LOW);
}

bool isAIMoveWinning() {
    return evaluate(board) == 1; // 1 означає виграш AI
}

Pair makeAIMove() {
    Pair bestMove = findBestMove(board); // Знаходимо найкращий хід
    board[bestMove.first][bestMove.second] = 'O'; // AI робить хід за O
    moveCount++; // Збільшуємо лічильник ходів
    return bestMove;
}

void resetBoard() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            board[i][j] = ' ';
        }
    }
    moveCount = 0;
    gameOver = false;
    waitingForPlayerMove = false;
    pvpmode = false;
    playerTurn = true;
    isPlayerOneTurn = true; // Додано
    memset(receivedData, 0, sizeof(receivedData));
    sendCurrentBoardState();
}


void sendCurrentBoardState() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            Serial.print(board[i][j]); // Відправка символу
        }
    }
    Serial.println(); // Перехід на новий рядок
}

bool checkWinner() {
    // Перевірка горизонтальних та вертикальних ліній
    for (int i = 0; i < 3; i++) {
        if (board[i][0] == board[i][1] && board[i][1] == board[i][2] &&
board[i][0] != ' ') return true;
        if (board[0][i] == board[1][i] && board[1][i] == board[2][i] &&
```

```
board[0][i] != ' ') return true;
    }
    // Перевірка діагоналей
    if (board[0][0] == board[1][1] && board[1][1] == board[2][2] &&
board[0][0] != ' ') return true;
    if (board[0][2] == board[1][1] && board[1][1] == board[2][0] &&
board[0][2] != ' ') return true;
    return false;
}

void BlueblinkLED() {
    if (blueLedState) { // Виконуємо тільки якщо діод увімкнений у
налаштуваннях
        digitalWrite(BlueledPin, HIGH);
        delay(300);
        digitalWrite(BlueledPin, LOW);
    }
}

void YellowblinkLED() {
    if (yellowLedState) { // Виконуємо тільки якщо діод увімкнений у
налаштуваннях
        digitalWrite(YellowledPin, HIGH);
        delay(300);
        digitalWrite(YellowledPin, LOW);
    }
}

void DrawblinkLED() {
    if (!blueLedState && !yellowLedState) return;
    if (blueLedState && yellowLedState){
      digitalWrite(BlueledPin, HIGH);
      digitalWrite(YellowledPin, HIGH);
      delay(300);
      digitalWrite(BlueledPin, LOW);
      digitalWrite(YellowledPin, LOW);
    } else if (blueLedState && !yellowLedState) {
        digitalWrite(BlueledPin, HIGH);
        delay(300);
        digitalWrite(BlueledPin, LOW);
    } else if (yellowLedState && !blueLedState){
        digitalWrite(YellowledPin, HIGH);
        delay(300);
        digitalWrite(YellowledPin, LOW);
    }
}
```

```
//
**** AI LOGIC ****

int evaluate(char board[3][3]) {
    // Перевірка рядків і стовпців
    for (int i = 0; i < 3; i++) {
        if (board[i][0] != ' ' && board[i][0] == board[i][1] && board[i][1]
== board[i][2])
            return (board[i][0] == ai) ? 1 : -1;
        if (board[0][i] != ' ' && board[0][i] == board[1][i] && board[1][i]
== board[2][i])
            return (board[0][i] == ai) ? 1 : -1;
    }
    // Перевірка діагоналей
    if (board[0][0] != ' ' && board[0][0] == board[1][1] && board[1][1] ==
board[2][2])
        return (board[0][0] == ai) ? 1 : -1;
    if (board[0][2] != ' ' && board[0][2] == board[1][1] && board[1][1] ==
board[2][0])
        return (board[0][2] == ai) ? 1 : -1;

    return 0; // Нічия
}

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

bool canCreateFork(char board[3][3], char playerSymbol) {
    int winningMoves = 0;

    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            if (board[i][j] == ' ') {
                board[i][j] = playerSymbol;
                if (evaluate(board) == ((playerSymbol == ai) ? 1 : -1)) {
                    winningMoves++;
                }
                board[i][j] = ' ';
            }
        }
    }
    return winningMoves >= 2;
```

```
}

int minimax(char board[3][3], int depth, bool isMaximizing, int alpha, int
beta) {
    if (depth > 9) return 0; // Ліміт на глибину
    int score = evaluate(board);

    if (score == 1 || score == -1) return score; // Якщо є перемога
    if (!isMovesLeft(board)) return 0; // Якщо нічия

    if (isMaximizing) {
        int best = -1000;
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (board[i][j] == ' ') {
                    board[i][j] = ai; // Пробуємо хід AI
                    best = max(best, minimax(board, depth + 1, false,
alpha, beta));
                    board[i][j] = ' '; // Відміняємо хід
                    alpha = max(alpha, best);
                    if (beta <= alpha) break; // Обрізання
                }
            }
        }
        return best;
    } else {
        int best = 1000;
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (board[i][j] == ' ') {
                    board[i][j] = player; // Пробуємо хід гравця
                    best = min(best, minimax(board, depth + 1, true, alpha,
beta));
                    board[i][j] = ' '; // Відміняємо хід
                    beta = min(beta, best);
                    if (beta <= alpha) break; // Обрізання
                }
            }
        }
        return best;
    }
}

Pair findBestMove(char board[3][3]) {
    Pair bestMove = {-1, -1};
    int bestVal = -1000;
```

```
    int positionPriority[3][3] = {
        {3, 2, 3}, // Пріоритетність позицій
        {2, 4, 2},
        {3, 2, 3}
    };
    Pair winningMove = findWinningMove(board, ai);
    if (winningMove.first != -1) {
        return winningMove; // Виконуємо виграшний хід
    }
    // Спершу перевіряємо, чи є хід, який блокує перемогу гравця
    Pair blockingMove = findBlockingMove(board, player);
    if (blockingMove.first != -1) {
        return blockingMove; // Повертаємо блокувальний хід
    }
    Pair forkMove = findForkMove(board, ai);
    if (forkMove.first != -1) {
        return forkMove; // Повертаємо хід, який створює форк
    }


    // MiniMax для визначення найкращого ходу
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == ' ') { // Якщо клітинка порожня
                board[i][j] = ai; // Пробуємо хід AI
                int moveVal = minimax(board, 0, false, -1000, 1000);
                board[i][j] = ' '; // Відміняємо хід

                // Додаємо оцінку пріоритету позиції
                moveVal += positionPriority[i][j];

                if (moveVal > bestVal) {
                    bestMove = {i, j};
                    bestVal = moveVal;
                }
            }
        }
    }
    return bestMove;
}
Pair findWinningMove(char board[3][3], char playerSymbol) {
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            if (board[i][j] == ' ') { // Якщо клітинка порожня
                board[i][j] = playerSymbol; // Симулюємо хід
                if (evaluate(board) == 1) { // Перевіряємо, чи це виграш
```

```
                    board[i][j] = ' '; // Скидаємо хід
                    return {i, j}; // Повертаємо виграшний хід
                }
                board[i][j] = ' '; // Скидаємо хід
            }
        }
    }
    return {-1, -1}; // Немає виграшних ходів
}
Pair findBlockingMove(char board[3][3], char opponent) {
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            if (board[i][j] == ' ') { // Якщо клітинка порожня
                board[i][j] = opponent; // Симулюємо хід опонента
                if (evaluate(board) == -1) { // Якщо це виграшний хід
                    board[i][j] = ' '; // Відміняємо хід
                    return {i, j}; // Повертаємо блокувальний хід
                }
                board[i][j] = ' '; // Відміняємо хід
            }
        }
    }
    return {-1, -1}; // Немає загроз
}
Pair findForkMove(char board[3][3], char playerSymbol) {
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            if (board[i][j] == ' ') { // Якщо клітинка порожня
                board[i][j] = playerSymbol; // Симулюємо хід
                if (canCreateFork(board, playerSymbol)) {
                    board[i][j] = ' '; // Скидаємо хід
                    return {i, j}; // Повертаємо хід, який створює форк
                }
                board[i][j] = ' '; // Скидаємо хід
            }
        }
    }
    return {-1, -1}; // Форк неможливий
}
```

Висновок

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