Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Лабораторна робота № 3

з дисципліни: «Автоматизоване проектування комп'ютерних систем»,

на тему: «Хрестики-нулики на клієнтській та серверній частинах.»

Виконав:

ст. гр. КІ-410

Іванюк О.О.

Прийняв:

Кіцера А. О.

Львів – 2024

Task 3.

Implement Server (HW) and Client (SW) parts of game (FEF)	**Create SW build system(EEF)**
1. Develop Server and Client. 2. Required steps.	1. Create **YML file** with next features: a. build all binaries (create scripts in folder **ci/** ifneed); b. run tests; c. create artifacts with binaries and test reports; 2. Required steps.

Варіант 7:

7	tik-tac-toe 3x3	XML

Теоретичні відомості

UART (Universal Asynchronous Receiver/Transmitter) представляє собою стандартний протокол для обміну даними між пристроями через послідовний інтерфейс. Цей інтерфейс дозволяє пристроям передавати та приймати інформацію біт за бітом і знаходить широке застосування в мікроконтролерах, мікропроцесорах, сенсорах, модемах та інших пристроях.

Основні характеристики UART включають асинхронний режим, де дані передаються через два незалежні сигнали - TX (передача) та RX (прийом), структуру кадру, яка включає стартовий біт, біти даних, біти парності (опціонально) та стоповий біт. Ця структура дозволяє правильно ідентифікувати початок та кінець кожного байту.

Швидкість передачі (Baud Rate) грає ключову роль у визначенні того, скільки бітів передається за одну секунду, і ця швидкість повинна бути налаштована на обох пристроях для успішного обміну даними.

Парність (Parity) є опціональною функцією, яка дозволяє визначити четність чи непарність бітів даних для виявлення помилок передачі. Керівництво лінією (Flow Control) може використовуватися для управління потоком даних, особливо при великій швидкості передачі або різних швидкостях пристроїв.

Також іноді використовується ізоляція гальванічна для електричної ізоляції між пристроями та запобігання електричним помилкам.

Дистанція передачі за допомогою UART обмежена, хоча цей інтерфейс дозволяє передавати дані на значні відстані. Зазвичай це кілька метрів без спеціальних заходів.

Хрестики-нулики — це гра для двох осіб, яка грається на квадратному полі розміром 3 на 3 або більшим. Один гравець грає хрестиками, а інший - нуликами, по черзі вони розташовують свої символи на вільних клітинках поля.

Мета гравця — створити лінію із трьох своїх символів горизонтально, вертикально або діагонально.

Хід роботи

1. Написав код для клієнтської частини.

Файл **main.cpp:**

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.setWindowTitle("TicTacToe");
    QIcon icon(":/icons/ttt_icon.ico");
    w.setWindowIcon(icon);
    w.show();
    return a.exec();
}
```

Файл **mainwindow.cpp:**

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QFile>
#include <QTextStream>
#include <windows.h>

HANDLE hSerial;
QString portArduino;

QString connect_arduino, game_started, game_mode, ai_strategy, message, next_turn;
QString board[3][3];

void resetValues() {
    // connect_arduino = "0";
    game_started = "0";
    // game_mode = "mva";
    // ai_strategy = "rand";
    message = "";

    // Заповнюємо масив board значенням "-"
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            board[i][j] = "-";
        }
    }
}

QString buildXML() {
    QString output = "<g>\n";
    output += "<con>" + connect_arduino + "</con>\n";
    output += "<gs>" + game_started + "</gs>\n";
    output += "<gm>" + game_mode + "</gm>\n";
    output += "<ais>" + ai_strategy + "</ais>\n";
    output += "<msg>" + message + "</msg>\n";
    output += "<nt>" + next_turn + "</nt>\n";
    output += "<brd>\n";
    output += "<c11>" + board[0][0] + "</c11>\n";
    output += "<c12>" + board[0][1] + "</c12>\n";
    output += "<c13>" + board[0][2] + "</c13>\n";
    output += "<c21>" + board[1][0] + "</c21>\n";
    output += "<c22>" + board[1][1] + "</c22>\n";
    output += "<c23>" + board[1][2] + "</c23>\n";
```

```
    output += "<c31>" + board[2][0] + "</c31>\n";
    output += "<c32>" + board[2][1] + "</c32>\n";
    output += "<c33>" + board[2][2] + "</c33>\n";
    output += "</brd>\n";
    output += "</g>\n";
    return output;
}


QString getTagValue(const QString& response, const QString& tagName) {
    // Формуємо строки для відкриваючого та закриваючого тегів
    QString openTag = "<" + tagName + ">";
    QString closeTag = "</" + tagName + ">";

    // Знаходимо позиції відкриваючого та закриваючого тегів
    int startIndex = response.indexOf(openTag);
    int endIndex = response.indexOf(closeTag, startIndex);

    // Якщо тег не знайдено, повертаємо пустий рядок
    if (startIndex == -1 || endIndex == -1) {
        return QString();
    }

    // Вираховуємо позицію, з якої починається текст між тегами
    startIndex += openTag.length();

    // Витягуємо текст між тегами
    return response.mid(startIndex, endIndex - startIndex);
}


void MainWindow::parseXML(QString input)
{
    connect_arduino = getTagValue(input, "con");
    game_started = getTagValue(input, "gs");
    game_mode = getTagValue(input, "gm");
    ai_strategy = getTagValue(input, "ais");

    message = getTagValue(input, "msg");
    ui->label_arduino_msg->setText(QString(message));

    next_turn  = getTagValue(input, "nt");

    board[0][0] = getTagValue(input, "c11");
    board[0][1] = getTagValue(input, "c12");
    board[0][2] = getTagValue(input, "c13");
    board[1][0] = getTagValue(input, "c21");
    board[1][1] = getTagValue(input, "c22");
    board[1][2] = getTagValue(input, "c23");
    board[2][0] = getTagValue(input, "c31");
    board[2][1] = getTagValue(input, "c32");
    board[2][2] = getTagValue(input, "c33");
}


void MainWindow::saveGameState(const QString& filePath) {
    QFile file(filePath);

    if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {
        ui->label_pc_msg->setText("Can't open file for write");
        return;
    }

    QTextStream out(&file);
    QString xmlData = buildXML();  // Викликаємо функцію, яка формує XML
    out << xmlData;

    file.close();

    ui->label_pc_msg->setText("Game Saved");
}
```

```cpp
void MainWindow::loadGameState(const QString& filePath) {
    QFile file(filePath);

    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        ui->label_pc_msg->setText("Can't open file for read");
        return;
    }

    QTextStream in(&file);
    QString xmlData = in.readAll();  // Читаємо весь файл у строку
    file.close();

    parseXML(xmlData);  // Парсимо XML та відновлюємо стан гри
    ui->label_pc_msg->setText("Load Complete");
}


MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    loadComPorts();

    connect(ui->comboBoxPorts,
            SIGNAL(currentTextChanged(const QString &)),
            this,
            SLOT(onComboBoxPortChanged(const QString &)));

    ui->frame_main_buttons->setEnabled(false);
}

MainWindow::~MainWindow()
{
    delete ui;
}


void MainWindow::updateGameBoard(){
    updateButtonIcon(ui->button_11, board[0][0]);
    updateButtonIcon(ui->button_12, board[0][1]);
    updateButtonIcon(ui->button_13, board[0][2]);
    updateButtonIcon(ui->button_21, board[1][0]);
    updateButtonIcon(ui->button_22, board[1][1]);
    updateButtonIcon(ui->button_23, board[1][2]);
    updateButtonIcon(ui->button_31, board[2][0]);
    updateButtonIcon(ui->button_32, board[2][1]);
    updateButtonIcon(ui->button_33, board[2][2]);
}

void MainWindow::updateButtonIcon(QPushButton *button, const QString &value){
    if (value == "x") {
        button->setIcon(QIcon(":/resources/assets/x2c.png"));  // Іконка хрестика
    } else if (value == "o") {
        button->setIcon(QIcon(":/resources/assets/o2c.png"));  // Іконка круга
    } else {
        button->setIcon(QIcon());  // Очищаємо іконку, якщо кнопка порожня
    }
}

void MainWindow::on_newButton_clicked()
{
    resetValues();
    game_started = "1";

    if (game_mode == "ava")
    {
        ui->frame_main_buttons->setEnabled(false);
        ui->frame_game_modes->setEnabled(false);
        ui->frame_ai_modes->setEnabled(false);
```

```
        ui->label_pc_msg->setText("AI vs AI mode started");

        while (game_started == "1")
        {
            // QString receivedXml = sendArduino();
            // parseXML(receivedXml);
            parseXML(sendArduino());
            updateGameBoard();

            QCoreApplication::processEvents(); // Дозволяємо Qt оновити інтерфейс

            //Sleep(150);
        }

        ui->frame_main_buttons->setEnabled(true);
        ui->frame_game_modes->setEnabled(true);
        ui->frame_ai_modes->setEnabled(true);

        ui->label_pc_msg->setText("Game is over");
    }
    else
    {
        // QString receivedXml = sendArduino();
        // parseXML(receivedXml);
        parseXML(sendArduino());

        if (game_started == "1"){
            ui->label_pc_msg->setText("Game started");
        }

        updateGameBoard();
    }
}
void MainWindow::on_loadButton_clicked()
{
    loadGameState("game_state.xml");
    updateGameBoard();

    if (game_mode == "mva") ui->radioButton_mai->setChecked(true);
    if (game_mode == "mvm") ui->radioButton_mvm->setChecked(true);
    if (game_mode == "ava") ui->radioButton_ava->setChecked(true);

    if (ai_strategy == "rand") ui->radioButton_rm->setChecked(true);
    if (ai_strategy == "win") ui->radioButton_ws->setChecked(true);

}
void MainWindow::on_saveButton_clicked(){
    saveGameState("game_state.xml");
}

void MainWindow::loadComPorts()
{
    ui->comboBoxPorts->clear();

    // Список для збереження імен портів
    QStringList comPortList;

    // Буфер для збереження імен пристроїв
    char lpTargetPath[5000]; // Буфер для збереження шляху до пристрою
    DWORD result;            // Результат виклику функції

    for (int i = 1; i <= 255; ++i) {
        // Формуємо ім'я порту: COM1, COM2, ..., COM255
        QString portName = QString("COM%1").arg(i);

        // Отримуємо інформацію про порт
        result = QueryDosDeviceA(portName.toStdString().c_str(), lpTargetPath, 5000);

        // Якщо порт існує, додаємо його до списку
        if (result != 0) {
            comPortList.append(portName);
        }
```

```
    }

    // Додаємо отримані порти в comboBox
    ui->comboBoxPorts->addItems(comPortList);
}

// Слот, який викликається при зміні вибраного порту
void MainWindow::onComboBoxPortChanged(const QString &port)
{
    ui->labelPort->setText(QString("Trying to connect via %1").arg(port));
    ui->labelPort->setStyleSheet("QLabel { color : lightblue; }");
    QCoreApplication::processEvents(); // Дозволяємо Qt оновити інтерфейс

    if (connectArduino(port)) {
        // Якщо відповідь від Arduino коректна
        ui->labelPort->setText("Arduino is connected");
        ui->labelPort->setStyleSheet("QLabel { color : lightgreen; }");

        // Активуємо кнопки
        ui->frame_main_buttons->setEnabled(true);

        portArduino = port;
    }
    else {
        if (port.length() > 0) {
            // Якщо помилка при зв'язку з Arduino
            ui->labelPort->setText(QString("Failed to connect via %1").arg(port));
            ui->labelPort->setStyleSheet("QLabel { color : red; }");
        }
        else {
            ui->labelPort->setText(QString("Select COM Port with Arduino"));
            ui->labelPort->setStyleSheet("QLabel { color : lightblue; }");
        }

        // Деактивуємо кнопки
        ui->frame_main_buttons->setEnabled(false);
    }

    updateGameBoard();

}

void MainWindow::on_refreshButton_clicked()
{
    loadComPorts();
}

QString MainWindow::sendArduino() {

    QString xmlData = buildXML();

    QString xmlData_trim = xmlData.remove(QChar(' '));
    xmlData_trim.remove(QChar('\n'));
    xmlData_trim.remove(QChar('\t'));

    std::string xmlString = xmlData_trim.toStdString();
    const char *dataToSend = xmlString.c_str();

    // qDebug() << "Sended  :" << dataToSend;

    DWORD bytesWritten;
    if (!WriteFile(hSerial, dataToSend, strlen(dataToSend), &bytesWritten, nullptr)) {
        CloseHandle(hSerial);
        return "";  // Помилка при відправці даних
    }

    // Чекаємо на відповідь
    char incomingData[256] = {0};
    DWORD bytesRead;
    if (!ReadFile(hSerial, incomingData, sizeof(incomingData), &bytesRead, nullptr)) {
        CloseHandle(hSerial);
        return "";  // Помилка при отриманні даних
```

```
    }

    // Закриваємо порт
    // CloseHandle(hSerial);

    // Перевіряємо, чи отримали правильну відповідь
    QString response(incomingData);
    // qDebug() << "Response:" << response;

    return response;
}

// Функція для тестування зв'язку з Arduino
bool MainWindow::connectArduino(const QString &portName)
{
    CloseHandle(hSerial);

    // Відкриваємо COM-порт
    hSerial = CreateFileA(portName.toStdString().c_str(),
                          GENERIC_READ | GENERIC_WRITE,
                          0, nullptr, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, nullptr);

    if (hSerial == INVALID_HANDLE_VALUE) {
        return false;  // Не вдалося відкрити порт
    }

    // Налаштування порту
    DCB dcbSerialParams = {};
    dcbSerialParams.DCBlength = sizeof(dcbSerialParams);
    if (!GetCommState(hSerial, &dcbSerialParams)) {
        CloseHandle(hSerial);
        return false;  // Помилка отримання стану порту
    }

    dcbSerialParams.BaudRate = CBR_9600;  // Налаштування швидкості
    dcbSerialParams.ByteSize = 8;
    dcbSerialParams.StopBits = ONESTOPBIT;
    dcbSerialParams.Parity = NOPARITY;

    if (!SetCommState(hSerial, &dcbSerialParams)) {
        CloseHandle(hSerial);
        return false;  // Помилка налаштування порту
    }

    // Налаштування тайм-аутів
    COMMTIMEOUTS timeouts = {};
    timeouts.ReadIntervalTimeout = 50;
    timeouts.ReadTotalTimeoutConstant = 50;
    timeouts.ReadTotalTimeoutMultiplier = 10;
    timeouts.WriteTotalTimeoutConstant = 50;
    timeouts.WriteTotalTimeoutMultiplier = 10;

    if (!SetCommTimeouts(hSerial, &timeouts)) {
        CloseHandle(hSerial);
        return false;  // Помилка налаштування тайм-аутів
    }

    Sleep(2000);

    resetValues();
    connect_arduino = "0";
    QString receivedXml = sendArduino();

    // QString conValue = getTagValue(receivedXml, "con");
    // qDebug() << "conValue:" << conValue;

    parseXML(receivedXml);

    return connect_arduino.trimmed() == "1";  // Порівнюємо з відповіддю Arduino
}

void MainWindow::add_player_turn(int row, int col)
```

```
{
    // connect_arduino, game_started, game_mode, ai_strategy, message, next_turn;
    if (connect_arduino != "1" || game_started != "1" || game_mode == "ava") {
        ui->label_pc_msg->setText("Game not started");
        return;
    }

    if (board[row][col] != "x" && board[row][col] != "o")
    {
        if (next_turn == "x") { board[row][col] = "x"; }
        else { board[row][col] = "o"; }

        ui->label_pc_msg->setText("");
    }
    else {
        ui->label_pc_msg->setText("Select empty cell");
        return;
    }

    updateGameBoard();
    QCoreApplication::processEvents(); // Дозволяємо Qt оновити інтерфейс

    QString receivedXml = sendArduino();
    parseXML(receivedXml);

    updateGameBoard();
}

void MainWindow::on_button_11_clicked() { add_player_turn(0, 0); }
void MainWindow::on_button_12_clicked() { add_player_turn(0, 1); }
void MainWindow::on_button_13_clicked() { add_player_turn(0, 2); }
void MainWindow::on_button_21_clicked() { add_player_turn(1, 0); }
void MainWindow::on_button_22_clicked() { add_player_turn(1, 1); }
void MainWindow::on_button_23_clicked() { add_player_turn(1, 2); }
void MainWindow::on_button_31_clicked() { add_player_turn(2, 0); }
void MainWindow::on_button_32_clicked() { add_player_turn(2, 1); }
void MainWindow::on_button_33_clicked() { add_player_turn(2, 2); }


void MainWindow::on_radioButton_mai_clicked(){
    game_mode = "mva";
    // ui->frame_ai_modes->setEnabled(true);
    // qDebug() << "game_mode:" << game_mode;
}
void MainWindow::on_radioButton_mvm_clicked(){
    game_mode = "mvm";
    // ui->frame_ai_modes->setEnabled(false);
    // qDebug() << "game_mode:" << game_mode;
}
void MainWindow::on_radioButton_ava_clicked(){
    game_mode = "ava";
    // ui->frame_ai_modes->setEnabled(true);
    // qDebug() << "game_mode:" << game_mode;
}

void MainWindow::on_radioButton_rm_clicked(){
    ai_strategy = "rand";
    // qDebug() << "ai_strategy:" << ai_strategy;
}
void MainWindow::on_radioButton_ws_clicked(){
    ai_strategy = "win";
    // qDebug() << "ai_strategy:" << ai_strategy;
}
```

Файл **mainwindow.ui:**

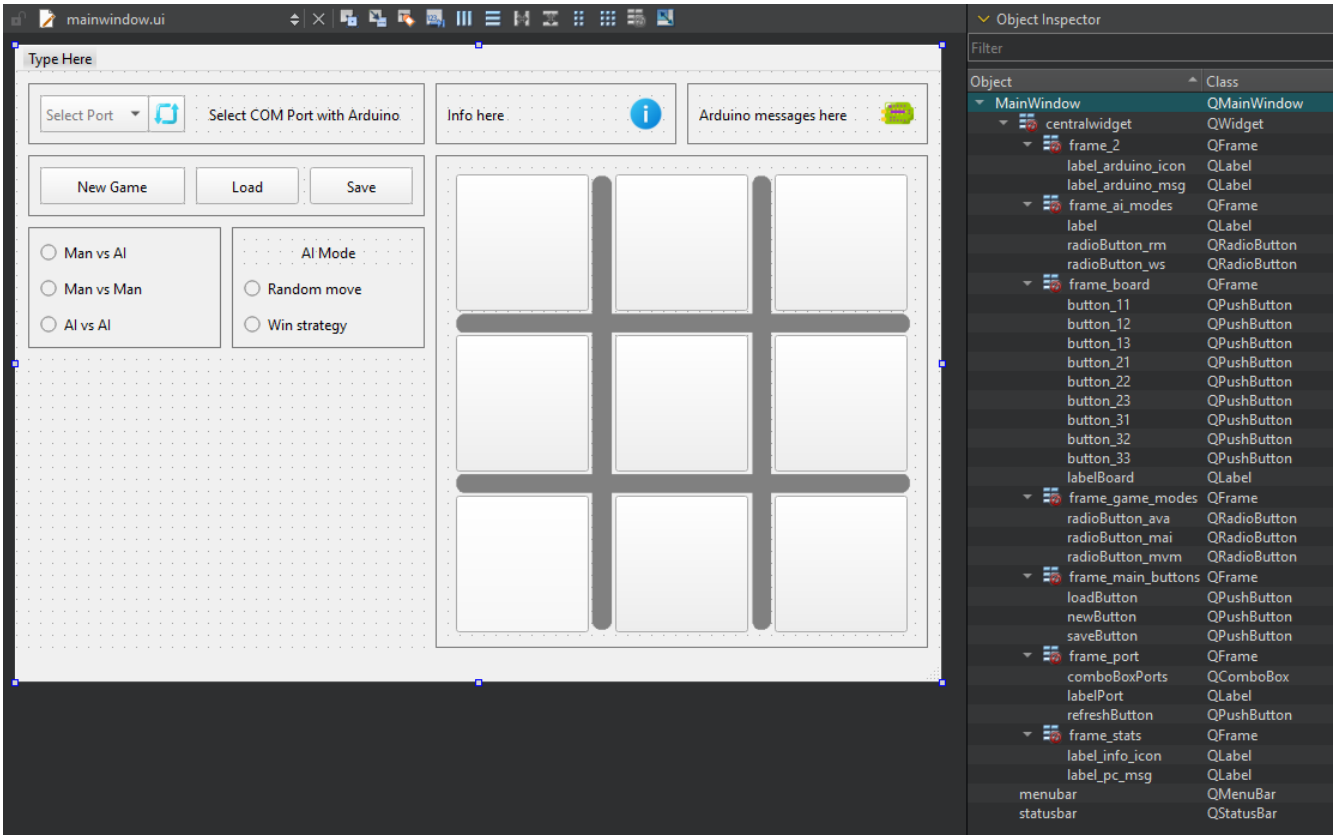

Рис. 1. mainwindow.ui в середовищі qt creator

2. Модифікував код для серверної частини з попередньої лабораторної роботи.

```
String connect_arduino, game_started, game_mode, ai_strategy, message, next_turn;
String board[3][3];

void setup() {
Serial.begin(9600);  // Налаштування серійного зв'язку зі швидкістю 9600 біт/с
randomSeed(analogRead(0));  // Ініціалізація генератора випадкових чисел

}

void loop() {
if (Serial.available() > 0) {

String receivedMessage = Serial.readString();  // Читаємо дані з серійного порту
parseXML(receivedMessage);

if (connect_arduino == "0") {
connect_arduino = "1";
message = "Successful connection";
}
else if (game_started == "1" && game_mode == "mva" && ai_strategy == "rand") {
mva_rand_move();
}
else if (game_started == "1" && game_mode == "mva" && ai_strategy == "win") {
win_move("o", true);
}
else if (game_started == "1" && game_mode == "ava" && ai_strategy == "rand") {
ava_rand_move();
```

```
}
else if (game_started == "1" && game_mode == "ava" && ai_strategy == "win") {
if (next_turn == "o") { win_move("o", false); }
else { win_move("x", false); }
}
else if (game_started == "1" && game_mode == "mvm") {
mvm_move_check();
}
else {
message = "Invalid input";
game_started = "0";
}

String output = buildXML();
output.replace("\n", ""); // Заміна /n на пустий рядок
Serial.print(output);
}
}


bool checkBoardWin(String board[3][3], String player) {
// Перевірка рядків (горизонталі)
for (int i = 0; i < 3; i++) {
if (board[i][0] == player && board[i][1] == player && board[i][2] == player) {
return true;
}
}

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

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

return false;
}

bool allFieldsOccupied(String board[3][3]) {
for (int i = 0; i < 3; i++) {
for (int j = 0; j < 3; j++) {
if (board[i][j] != "x" && board[i][j] != "o") {
return false; // Знайшли пусте поле, повертаємо false
}
}
}
return true; // Якщо всі поля заповнені
}

bool checkForWinner() {
if (checkBoardWin(board, "x")) {
message = "Winner - X";
game_started = "0";
return true;
}
else if (checkBoardWin(board, "o")) {
message = "Winner - O";
game_started = "0";
return true;
}
else if (allFieldsOccupied(board)) {
message = "Draw";
game_started = "0";
return true;
```

```
}
else {
return false;
}
}



void ava_rand_move() {
int emptyCells[9];
int emptyCount = 0;

if (checkForWinner()) { return; }

// Проходимо через поле та знаходимо всі порожні клітинки
for (int row = 0; row < 3; row++) {
for (int col = 0; col < 3; col++) {

if (board[row][col] != "x" && board[row][col] != "o") {
emptyCells[emptyCount] = row * 3 + col;  // Зберігаємо індекс клітинки у вигляді одного числа
emptyCount++;
}
}
}

// Якщо є порожні клітинки, вибираємо випадкову
if (emptyCount > 0) {
int randomIndex = random(0, emptyCount);  // Випадковий індекс від 0 до emptyCount-1
int rand_cell = emptyCells[randomIndex];  // Індекс випадкової клітинки

// Перетворюємо індекс назад у координати (row, col)
int row = rand_cell / 3;
int col = rand_cell % 3;

if(next_turn == "o"){
board[row][col] = "o";
next_turn = "x";
message = "Player X turn";
}
else {
board[row][col] = "x";
next_turn = "o";
message = "Player O turn";
}

if (checkForWinner()) { return; }

}
else {
message = "Draw";
game_started = "0";
}
}



void mvm_move_check() {
int emptyCount = 0;

if (checkForWinner()) { return; }

// Проходимо через поле та знаходимо всі порожні клітинки
for (int row = 0; row < 3; row++) {
for (int col = 0; col < 3; col++) {

if (board[row][col] != "x" && board[row][col] != "o") {
emptyCount++;
}
}
}

// Вибір гравця для першого ходу
if (emptyCount == 9) {
```

```
int randomChoice = random(0, 2);  // Випадковий вибір: 0 або 1
if (randomChoice == 0) {
message = "Player X turn";
next_turn = "x";
return;
}
else {
message = "Player O turn";
next_turn = "o";
return;
}
}

if(next_turn == "x") {
message = "Player O turn";
next_turn = "o";
}
else {
message = "Player X turn";
next_turn = "x";
}
}



void mva_rand_move() {
int emptyCells[9];
int emptyCount = 0;

if (checkForWinner()) { return; }

// Проходимо через поле та знаходимо всі порожні клітинки
for (int row = 0; row < 3; row++) {
for (int col = 0; col < 3; col++) {

if (board[row][col] != "x" && board[row][col] != "o") {

emptyCells[emptyCount] = row * 3 + col;  // Зберігаємо індекс клітинки у вигляді одного числа
emptyCount++;
}
}
}

next_turn = "x";
message = "Player X turn";

if (emptyCount == 9) {
int randomChoice = random(0, 2);  // Випадковий вибір: 0 або 1
if (randomChoice == 0) {
return;
}
}

// Якщо є порожні клітинки, вибираємо випадкову
if (emptyCount > 0) {
int randomIndex = random(0, emptyCount);  // Випадковий індекс від 0 до emptyCount-1
int rand_cell = emptyCells[randomIndex];  // Індекс випадкової клітинки

// Перетворюємо індекс назад у координати (row, col)
int row = rand_cell / 3;
int col = rand_cell % 3;

// Записуємо "o" в обрану клітинку
board[row][col] = "o";
}

checkForWinner();
}



bool checkWin(String player, String currentPlayer) {
```

```
for (int i = 0; i < 3; i++) {
// Перевірка рядків і стовпців
if ((board[i][0] == player && board[i][1] == player && board[i][2] == "-") ||
(board[i][0] == player && board[i][2] == player && board[i][1] == "-") ||
(board[i][1] == player && board[i][2] == player && board[i][0] == "-")) {
board[i][0] == "-" ? board[i][0] = currentPlayer :
board[i][1] == "-" ? board[i][1] = currentPlayer : board[i][2] = currentPlayer;
return true;
}

if ((board[0][i] == player && board[1][i] == player && board[2][i] == "-") ||
(board[0][i] == player && board[2][i] == player && board[1][i] == "-") ||
(board[1][i] == player && board[2][i] == player && board[0][i] == "-")) {
board[0][i] == "-" ? board[0][i] = currentPlayer :
board[1][i] == "-" ? board[1][i] = currentPlayer : board[2][i] = currentPlayer;
return true;
}
}

// Перевірка діагоналей
if ((board[0][0] == player && board[1][1] == player && board[2][2] == "-") ||
(board[0][0] == player && board[2][2] == player && board[1][1] == "-") ||
(board[1][1] == player && board[2][2] == player && board[0][0] == "-")) {
board[0][0] == "-" ? board[0][0] = currentPlayer :
board[1][1] == "-" ? board[1][1] = currentPlayer : board[2][2] = currentPlayer;
return true;
}

if ((board[0][2] == player && board[1][1] == player && board[2][0] == "-") ||
(board[0][2] == player && board[2][0] == player && board[1][1] == "-") ||
(board[1][1] == player && board[2][0] == player && board[0][2] == "-")) {
board[0][2] == "-" ? board[0][2] = currentPlayer :
board[1][1] == "-" ? board[1][1] = currentPlayer : board[2][0] = currentPlayer;
return true;
}

return false;
}

void win_move(String currentPlayer, bool randomPlayer) {

if (checkForWinner()) { return; }

bool made_move = false;

String opponent = (currentPlayer == "x") ? "o" : "x";

next_turn = opponent;

String upperOpponent = opponent;
upperOpponent.toUpperCase();
message = "Player "+ upperOpponent +" turn";

if (randomPlayer){
int emptyCount = 0;
for (int row = 0; row < 3; row++) {
for (int col = 0; col < 3; col++) {
if (board[row][col] != "x" && board[row][col] != "0") {
emptyCount++;
}
}
}

if (emptyCount == 9) {
int randomChoice = random(0, 2);  // Випадковий вибір: 0 або 1
if (randomChoice == 0) { return; }
}
}

// 1. Спробуємо виграти
if (!made_move && checkWin(currentPlayer, currentPlayer)) {
made_move = true;
```

```
}

// 2. Заблокуємо "x", якщо він може виграти
if (!made_move && checkWin(opponent, currentPlayer)) {
made_move = true;
}

// 3. Займаємо центр, якщо він вільний
if (!made_move && board[1][1] == "-") {
board[1][1] = currentPlayer;
made_move = true;
}

// 4. Займаємо кути, якщо вони вільні
if (!made_move && board[0][0] == "-" && board[2][2] != opponent) {
board[0][0] = currentPlayer;
made_move = true;
}
if (!made_move && board[0][2] == "-" && board[2][0] != opponent) {
board[0][2] = currentPlayer;
made_move = true;
}
if (!made_move && board[2][0] == "-" && board[0][2] != opponent) {
board[2][0] = currentPlayer;
made_move = true;
}
if (!made_move && board[2][2] == "-" && board[0][0] != opponent) {
board[2][2] = currentPlayer;
made_move = true;
}
if (!made_move) {
int emptyCells[9];
int emptyCount = 0;

// Проходимо через поле та знаходимо всі порожні клітинки
for (int row = 0; row < 3; row++) {
for (int col = 0; col < 3; col++) {

if (board[row][col] != "x" && board[row][col] != "o") {
emptyCells[emptyCount] = row * 3 + col;  // Зберігаємо індекс клітинки у вигляді одного числа
emptyCount++;
}
}
}

// Якщо є порожні клітинки, вибираємо випадкову
if (emptyCount > 0) {
int randomIndex = random(0, emptyCount);  // Випадковий індекс від 0 до emptyCount-1
int rand_cell = emptyCells[randomIndex];  // Індекс випадкової клітинки

// Перетворюємо індекс назад у координати (row, col)
int row = rand_cell / 3;
int col = rand_cell % 3;

board[row][col] = currentPlayer;
}
}

// 5. Займаємо будь-яке інше місце
// for (int i = 0; i < 3; i++) {
//   for (int j = 0; j < 3; j++) {

//     if (!made_move && board[i][j] == "-") {
//       board[i][j] = currentPlayer;
//       made_move = true;
//     }
//   }
// }

if (checkForWinner()) { return; }
}
```

```
void parseXML(String input) {
connect_arduino = getTagValue(input, "con");
game_started = getTagValue(input, "gs");
game_mode = getTagValue(input, "gm");
ai_strategy = getTagValue(input, "ais");
message = getTagValue(input, "msg");

board[0][0] = getTagValue(input, "c11");
board[0][1] = getTagValue(input, "c12");
board[0][2] = getTagValue(input, "c13");
board[1][0] = getTagValue(input, "c21");
board[1][1] = getTagValue(input, "c22");
board[1][2] = getTagValue(input, "c23");
board[2][0] = getTagValue(input, "c31");
board[2][1] = getTagValue(input, "c32");
board[2][2] = getTagValue(input, "c33");
}


String getTagValue(String input, String tag) {
String openTag = "<" + tag + ">";
String closeTag = "</" + tag + ">";
int start = input.indexOf(openTag) + openTag.length();
int end = input.indexOf(closeTag);

if (start == -1 || end == -1 || end < start) {
return "-";
}

return input.substring(start, end);
}

// Функція для збирання нового XML рядка
String buildXML() {
String output = "<g>\n";
output += "<con>" + connect_arduino + "</con>\n";
output += "<gs>" + game_started + "</gs>\n";
output += "<gm>" + game_mode + "</gm>\n";
output += "<ais>" + ai_strategy + "</ais>\n";
output += "<msg>" + message + "</msg>\n";
output += "<nt>" + next_turn + "</nt>\n";
output += "<brd>\n";
output += "<c11>" + board[0][0] + "</c11>\n";
output += "<c12>" + board[0][1] + "</c12>\n";
output += "<c13>" + board[0][2] + "</c13>\n";
output += "<c21>" + board[1][0] + "</c21>\n";
output += "<c22>" + board[1][1] + "</c22>\n";
output += "<c23>" + board[1][2] + "</c23>\n";
output += "<c31>" + board[2][0] + "</c31>\n";
output += "<c32>" + board[2][1] + "</c32>\n";
output += "<c33>" + board[2][2] + "</c33>\n";
output += "</brd>\n";
output += "</g>\n";
return output;
}
```

3. Провів перевірку на працездатність.

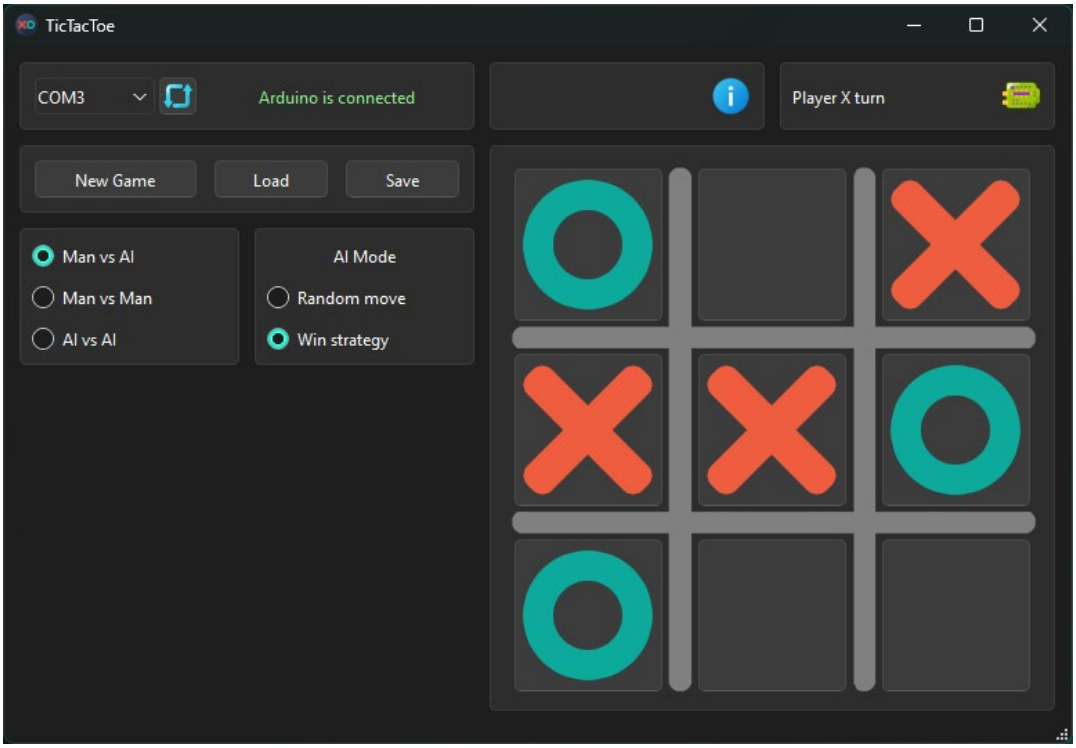

Рис. 2. Вікно програми

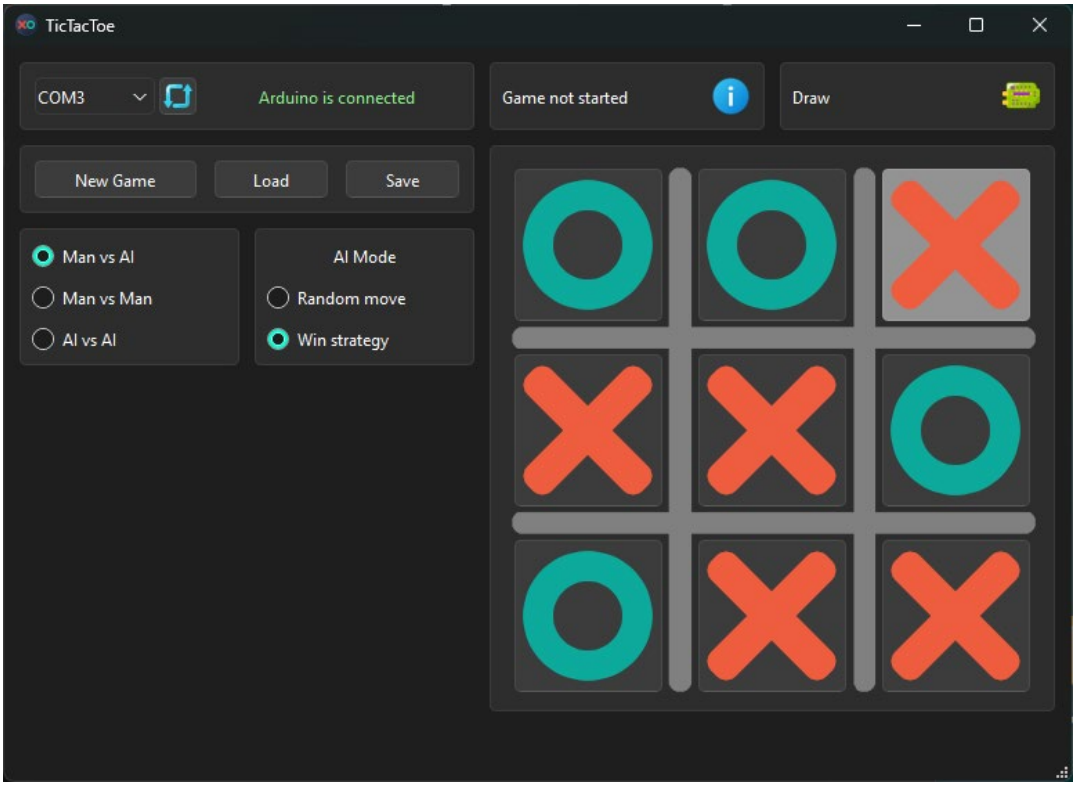

Рис. 3. Гра завершилась нічиєю

4. Створив нову гілку feature/develop/task3. Створив Pull request для підтвердження змін в гілці develop, і надіслав запит на злиття викладачу.

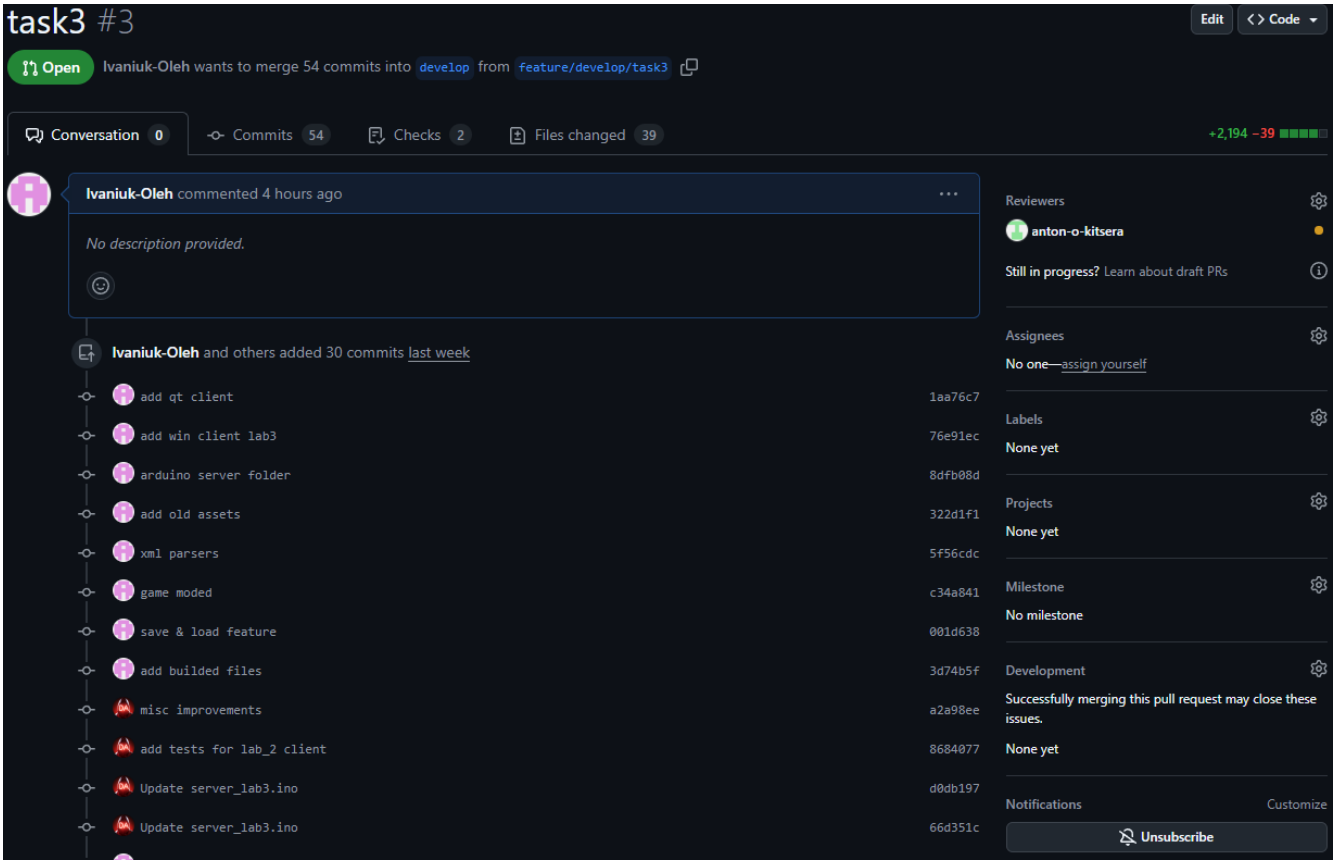

Рис. 4. Pull request для task3

Висновок:

В ході виконання лабораторної роботи було розроблено клієнт-серверну гру "хрестики-нулики", яка використовує комунікацію між клієнтською програмою та апаратним забезпеченням через UART інтерфейс. Цей підхід дозволяє взаємодіяти з грою за допомогою апаратного забезпечення.