Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Лабораторна робота № 3

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

на тему: «Камінь-ножниці-папір на клієнтській та серверній частинах.»

Виконав:ст. гр. КІ-410

Ковальчук А.О.

Прийняв:

к.т.н. асит. каф. СКС

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 8:

Student number	**Game**	**config format**
8	rock paper scissors	JSON

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.

Камінь ножиці папір — це гра для двох осіб, де в кожного гравця є три вибори:

1-Камінь 2-Ножиці 3-Папір

Камінь б’є ножиці та програє паперу

Ножиці б’ють папір та програють каменю

Папір б’є камінь та програє ножицям

Хід роботи

1. Написав код для клієнтської частини.

```
/**
@file Rock_paper_scissors_client.cpp
*/
#include <windows.h>
#include <iostream>
#include <string>
#include <vector>
#include <cstdlib> // For system()
#include <fstream>
#include "nlohmann/json.hpp"
using json = nlohmann::json;

using namespace std;

/**
 * @brief Saves the current game state to a JSON file.
 *
 * @param gameState The current game state represented as a vector of integers.
 * @param resultText The result text of the game.
 * @param gameInfo Additional information about the game.
 * @param filename The filename for saving the game state as JSON (default is "game_config.json").
 */
void saveGameState(const vector<int>& gameState, const string& resultText, const string& gameInfo, const string&
filename = "game_config.json") {
    json j;
    j["gameState"] = gameState;
    j["resultText"] = resultText;
    j["gameInfo"] = gameInfo;

    std::ofstream file(filename);
    if (file.is_open()) {
        file << j.dump(4);  // Saves JSON with indentation for readability
        file.close();
        cout << "Game state saved to " << filename << endl;
    }
    else {
        cerr << "Failed to open file for saving game state" << endl;
    }
}

/**
 * @brief Loads the game state from a JSON file.
 *
```

```
 * @param gameState Reference to a vector that will hold the loaded game state.
 * @param resultText Reference to a string that will hold the loaded result text.
 * @param gameInfo Reference to a string that will hold the loaded game information.
 * @param filename The filename to load the game state from (default is "game_config.json").
 * @return True if loading was successful; otherwise, false.
 */
bool loadGameState(vector<int>& gameState, string& resultText, string& gameInfo, const string& filename =
"game_config.json") {
    std::ifstream file(filename);
    if (file.is_open()) {
        json j;
        file >> j;

        if (j.contains("gameState") && j.contains("resultText") && j.contains("gameInfo")) {
            gameState = j["gameState"].get<vector<int>>();
            resultText = j["resultText"].get<string>();
            gameInfo = j["gameInfo"].get<string>();
            file.close();
            cout << "Game state loaded from " << filename << endl;
            return true;
        }
        else {
            cerr << "Invalid data in config file" << endl;
        }
    }
    else {
        cerr << "Config file not found" << endl;
    }
    return false;
}

/**
 * @brief Creates a JSON string from an integer array.
 *
 * @param array The array of integers to be converted into JSON.
 * @return A JSON string representation of the integer array.
 */
string createJSON(const vector<int>& array) {
    string json = "{\"array\":[";
    for (size_t i = 0; i < array.size(); ++i) {
        json += to_string(array[i]);
        if (i < array.size() - 1) json += ",";
    }
    json += "]}";
    return json;
}

/**
 * @brief Parses a JSON string to extract an integer array, result text, and game information.
 *
 * @param json The JSON string to parse.
 * @param array Reference to a vector to store the extracted integer array.
 * @param resultText Reference to a string to store the extracted result text.
 * @param gameInfo Reference to a string to store the extracted game information.
 * @return True if parsing was successful; otherwise, false.
 */
bool parseJSON(const string& json, vector<int>& array, string& resultText, string& gameInfo) {
    size_t arrayStart = json.find("\"array\":[");
```

```
    size_t resultStart = json.find("\"result\":\"");
    size_t infoStart = json.find("\"info\":\"");

    if (arrayStart != string::npos && resultStart != string::npos && infoStart != string::npos) {
        array.clear();
        size_t startPos = arrayStart + 9;
        size_t endPos = json.find("]", startPos);

        while (startPos < endPos) {
            size_t commaPos = json.find(",", startPos);
            if (commaPos == string::npos || commaPos > endPos) commaPos = endPos;
            array.push_back(stoi(json.substr(startPos, commaPos - startPos)));
            startPos = commaPos + 1;
        }

        size_t resultEnd = json.find("\"", resultStart + 10);
        resultText = json.substr(resultStart + 10, resultEnd - (resultStart + 10));

        size_t infoEnd = json.find("\"", infoStart + 8);
        gameInfo = json.substr(infoStart + 8, infoEnd - (infoStart + 8));

        return true;
    }

    return false;
}

/**
 * @brief The main function initializes serial communication with an Arduino, handles game modes, and processes game
data.
 *
 * @return Exit code of the program (0 for success, 1 for failure).
 */
int main() {
    int portNumber;
    cout << "Enter COM port number (e.g., 6 for COM6): ";
    cin >> portNumber;

    string portName_s = "COM" + to_string(portNumber);
    const char* portName = portName_s.c_str();

    HANDLE hSerial = CreateFileA(portName,
        GENERIC_READ | GENERIC_WRITE,
        0,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL);

    if (hSerial == INVALID_HANDLE_VALUE) {
        cerr << "Error opening port" << endl;
        system("pause");
        return 1;
    }

    DCB dcbSerialParams = { 0 };
    dcbSerialParams.DCBlength = sizeof(dcbSerialParams);
```

```
if (!GetCommState(hSerial, &dcbSerialParams)) {
    cerr << "Error getting port state" << endl;
    system("pause");
    return 1;
}

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

if (!SetCommState(hSerial, &dcbSerialParams)) {
    cerr << "Error setting port parameters" << endl;
    system("pause");
    return 1;
}

COMMTIMEOUTS timeouts = { 0 };
timeouts.ReadIntervalTimeout = 50;
timeouts.ReadTotalTimeoutConstant = 50;
timeouts.ReadTotalTimeoutMultiplier = 10;

if (!SetCommTimeouts(hSerial, &timeouts)) {
    cerr << "Error setting timeouts!" << endl;
    system("pause");
    return 1;
}

Sleep(2000);
vector<int> resultArray(5, 0);
string resultText, gameInfo;

while (true) {
    vector<int> resultArray(5, 0);
    cout << "Gamemodes:\n1-Man vs AI\n2-Man vs Man\n3-AI vs AI\n4-Load previous game\n";
    cout << "Enter Game mode: ";
    int gameMode;
    cin >> gameMode;

    if (gameMode == 4) {
        if (loadGameState(resultArray, resultText, gameInfo)) {
            cout << "Loaded Game Mode: " << resultArray[0] << endl;
            cout << "Current Score: " << resultArray[3] << "/3 : " << resultArray[4] << "/3" << endl;
            cout << "Result: " << resultText << endl;
            cout << "Info: " << gameInfo << endl;

            if (resultArray[3] == 3 || resultArray[4] == 3) {
                cout << "Game already completed. Restart to play again." << endl;
                continue;
            }
        }
        else {
            cout << "No previous game data found. Starting a new game." << endl;
        }
    }
    else {
        resultArray[0] = gameMode;
    }
```

```
while (true) {
    if (resultArray[0] == 1) {
        cout << "\n1 - Rock\n2 - Paper\n3 - Scissors\nEnter:";
        cin >> resultArray[1];
    }
    else if (resultArray[0] == 2) {
        cout << "\n1 - Rock\n2 - Paper\n3 - Scissors\nEnter1:";
        cin >> resultArray[1];
        system("cls");
        cout << "1 - Rock\n2 - Paper\n3 - Scissors\nEnter2:";
        cin >> resultArray[2];
    }
    else {
        cout << "\nAI vs AI\n:";
    }

    string messageJSON = createJSON(resultArray);
    DWORD bytesWritten;

    if (!WriteFile(hSerial, messageJSON.c_str(), messageJSON.size(), &bytesWritten, NULL)) {
        cerr << "Error writing to port" << endl;
        break;
    }
    cout << "\nSent " << messageJSON << endl;

    string receivedMessage;
    while (true) {
        char buffer[256];
        DWORD bytesRead;

        if (ReadFile(hSerial, buffer, sizeof(buffer) - 1, &bytesRead, NULL) && bytesRead > 0) {
            buffer[bytesRead] = '\0';
            receivedMessage += buffer;

            if (receivedMessage.find('\n') != string::npos) {
                receivedMessage.erase(receivedMessage.find('\n'));
                break;
            }
        }
    }

    vector<int> array;
    if (parseJSON(receivedMessage, array, resultText, gameInfo)) {
        cout << "Received:    [";
        for (size_t i = 0; i < array.size(); ++i) {
            resultArray[i] = array[i];
            cout << array[i];
            if (i < array.size() - 1) cout << ",";
        }
        cout << "]" << endl;
        cout << gameInfo << endl;
        cout << "Score: " << resultArray[3] << "/3  :  " << resultArray[4] << "/3\n";
        cout << "Result: " << resultText << endl;
        saveGameState(resultArray, resultText, gameInfo);
        if (resultArray[3] == 3 || resultArray[4] == 3) {
            cout << "\n";
            break;
```

```
            }
        }
        else {
            cerr << "Error parsing JSON from Arduino" << endl;
        }

        Sleep(1000);
    }
  }
  CloseHandle(hSerial);
  return 0;
}
```

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

```
/**
@file sketch_nov6a.ino
*/

#include <ArduinoJson.h>

/// @brief The target score needed to win the game.
const int targetScore = 3;

void setup() {
  /**
   * @brief Initializes the Serial communication and the random number generator.
   *
   * Sets the baud rate to 9600 for Serial communication and seeds the random
   * number generator using an analog read on pin 0 to ensure randomness.
   */
  Serial.begin(9600);
  while (!Serial) { }
  randomSeed(analogRead(0));
}

void loop() {
  /// @brief Game mode labels for display in the game info.
  String modes[] = {"Player vs Computer ", "Player vs Player ", "Computer vs
Computer "};

  /// @brief Labels for Player 1 based on the selected game mode.
  String player1[] = {" Player ", " Player 1 ", " Computer1 "};

  /// @brief Labels for Player 2 based on the selected game mode.
  String player2[] = {"Computer ", " Player 2 ", " Computer2 "};

  /// @brief Choices available for the game: Rock, Paper, Scissors.
  String choices[] = {"Rock ", " Paper ", " Scissors"};
```

```
  // Check if there is incoming data on the Serial.
  if (Serial.available() > 0) {
    // Read the incoming JSON string until a newline character.
    String input = Serial.readStringUntil('\n');

    /**
     * @brief JSON document to parse the input data.
     *
     * A static JSON document of size 200 is used for parsing the input data
     * to prevent dynamic memory allocation on the microcontroller.
     */
    StaticJsonDocument<200> doc;
    DeserializationError error = deserializeJson(doc, input);

    // Check for JSON parsing errors.
    if (error) {
      Serial.println("{\"error\":\"Invalid JSON\"}");
      return;
    }

    // Retrieve the array from the JSON document.
    JsonArray array = doc["array"];
    if (!array.isNull() && array.size() == 5) {
      int mode = array[0]; ///< @brief Game mode selected by the player.

      // Generate random choices for the computer based on the game mode.
      if (mode == 1) {
        array[2] = random(1, 4);  // Player vs Computer mode
      } else if (mode == 3) {
        array[1] = random(1, 4);  // Computer vs Computer mode, both players are AI
        array[2] = random(1, 4);
      }

      int player1Choice = array[1]; ///< @brief Player 1's choice.
      int player2Choice = array[2]; ///< @brief Player 2's choice.
      int player1Wins = array[3];   ///< @brief Current win count for Player 1.
      int player2Wins = array[4];   ///< @brief Current win count for Player 2.

      // Determine the result of the round.
      String resultText;  ///< @brief Text describing the outcome of the round.
      String gameInfo;    ///< @brief Additional information about the current game
state.

      // Logic for determining the winner of the round.
      if (player1Choice == player2Choice) {
        resultText = "Draw";
```

```
      } else if ((player1Choice == 1 && player2Choice == 3) ||
                 (player1Choice == 2 && player2Choice == 1) ||
                 (player1Choice == 3 && player2Choice == 2)) {
        player1Wins++;
        resultText = player1[mode - 1] + " wins this round";
      } else {
        player2Wins++;
        resultText = player2[mode - 1] + " wins this round";
      }

      // Update the array with the current scores.
      array[3] = player1Wins;
      array[4] = player2Wins;

      // Check if either player has reached the target score.
      if (player1Wins == targetScore) {
        resultText = player1[mode - 1] + " wins the game";
      } else if (player2Wins == targetScore) {
        resultText = player2[mode - 1] + " wins the game";
      }

      /**
       * @brief JSON document to store the response data.
       *
       * This document is used to create the JSON response, including the updated
       * game state array, the result text, and additional game information.
       */
      StaticJsonDocument<200> responseDoc;
      JsonArray responseArray = responseDoc.createNestedArray("array");
      for (int i = 0; i < 5; i++) {
        responseArray.add(array[i]);
      }

      // Add the result and game information to the JSON response.
      responseDoc["result"] = resultText;
      responseDoc["info"] = modes[mode - 1] + player1[mode - 1] + " choice: " +
choices[player1Choice - 1] + ";  " +
                            player2[mode - 1] + " choice: " + choices[player2Choice
- 1];

      // Serialize the JSON response and send it over Serial.
      serializeJson(responseDoc, Serial);
      Serial.println();  // End the message with a newline character.
    }
  }

  delay(100);  // Delay to avoid overloading the Serial communication.
```

```
}
```

3. Провів перевірку на працездатність.

```
C:\Users\Andrew\Desktop\apks\lab3_apks\Rock_paper_scissors_client\x64\Debug\Rock_paper_scissors_client.exe
Enter COM port number (e.g., 6 for COM6): 7
Gamemodes:
1-Man vs AI
2-Man vs Man
3-AI vs AI
4-Load previous game
Enter Game mode:
```

Рис. 1. Стартове вікно програми з вибором режиму гри

```
Enter Game mode: 1

1 - Rock
2 - Paper
3 - Scissors
Enter:1

Sent {"array":[1,1,0,0,0]}
Received:     [1,1,3,1,0]
Player vs Computer  Player  choice: Rock ;  Computer  choice:  Scissors
Score: 1/3  :  0/3
Result:  Player  wins this round
Game state saved to game_config.json
```

Рис. 2. Режим Man vs AI

```
Enter Game mode: 2

1 - Rock
2 - Paper
3 - Scissors
Enter1:1
```

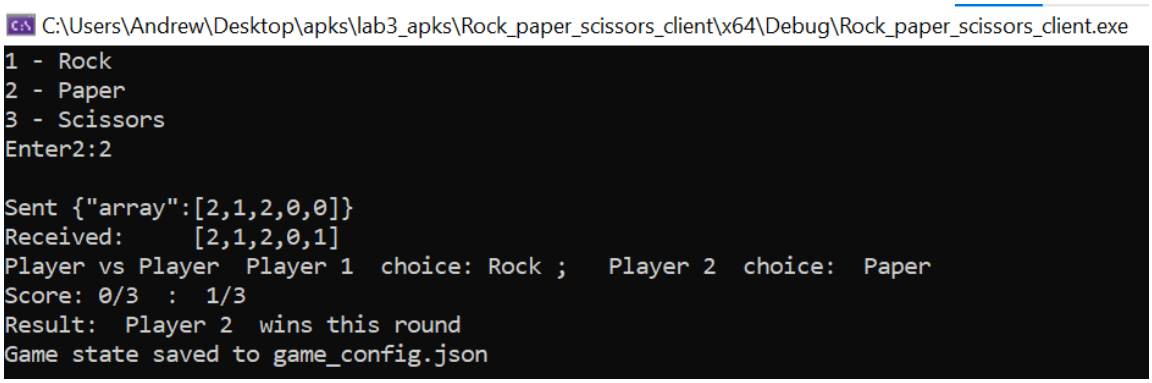


Рис. 3. Режим Man vs Man

```
Enter Game mode: 3

AI vs AI
:
Sent {"array":[3,0,0,0,0]}
Received:     [3,3,1,0,1]
Computer vs Computer  Computer1  choice:  Scissors;   Computer2  choice: Rock
Score: 0/3  :  1/3
Result:  Computer2  wins this round
Game state saved to game_config.json

AI vs AI
:
Sent {"array":[3,3,1,0,1]}
Received:     [3,3,1,0,2]
Computer vs Computer  Computer1  choice:  Scissors;   Computer2  choice: Rock
Score: 0/3  :  2/3
Result:  Computer2  wins this round
Game state saved to game_config.json

AI vs AI
:
Sent {"array":[3,3,1,0,2]}
Received:     [3,2,2,0,2]
Computer vs Computer  Computer1  choice:  Paper ;   Computer2  choice:  Paper
Score: 0/3  :  2/3
Result: Draw
Game state saved to game_config.json
```

Рис. 4. Режим AI vs AI

```
Sent {"array":[1,1,2,0,2]}
Received:     [1,1,2,0,3]
Player vs Computer  Player  choice: Rock ;  Computer  choice:  Paper
Score: 0/3  :  3/3
Result: Computer  wins the game
Game state saved to game_config.json

Gamemodes:
1-Man vs AI
2-Man vs Man
3-AI vs AI
4-Load previous game
Enter Game mode:
```

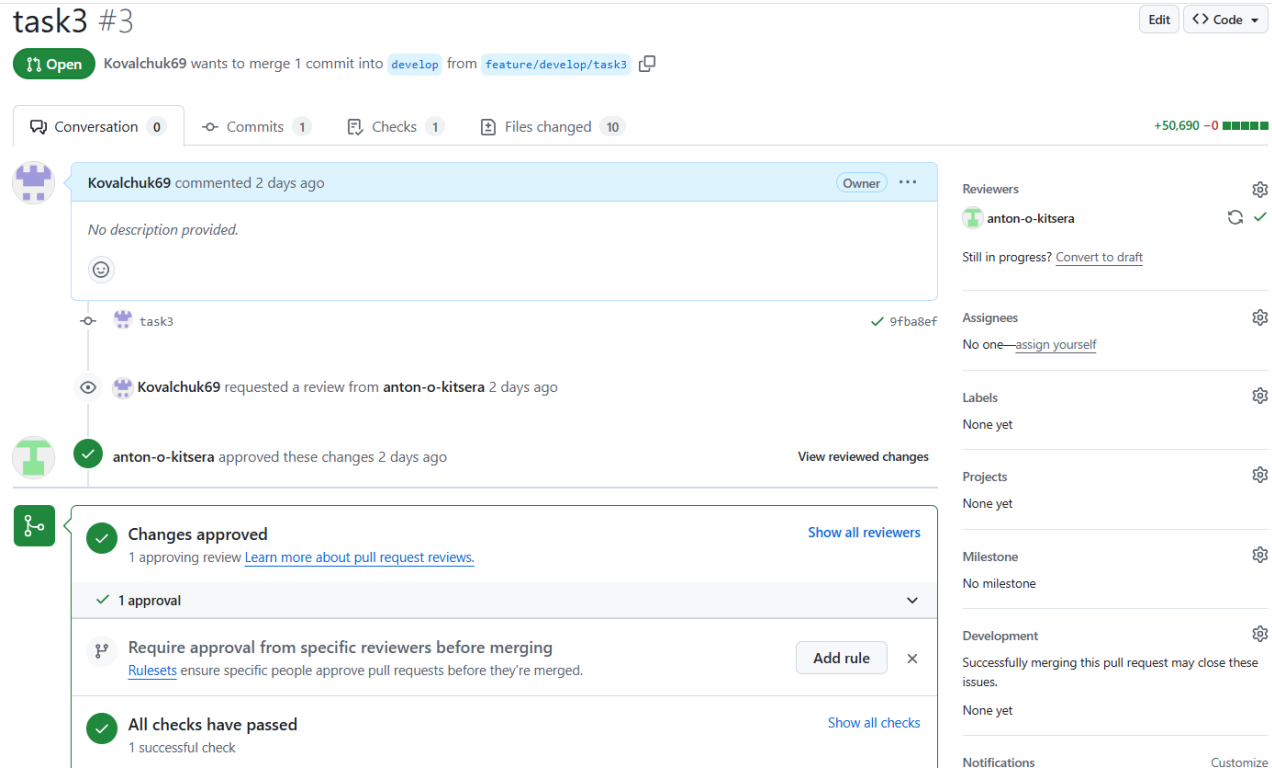
Рис. 5. Перемога одного з гравців

C:\Users\Andrew\Desktop\apks\lab3_apks\Rock_paper_scissors_client\x64\Debug\Rock_paper_scissors_client.exe

```
Enter COM port number (e.g., 6 for COM6): 7
Gamemodes:
1-Man vs AI
2-Man vs Man
3-AI vs AI
4-Load previous game
Enter Game mode: 4
Game state loaded from game_config.json
Loaded Game Mode: 1
Current Score: 0/3 : 1/3
Result: Computer  wins this round
Info: Player vs Computer  Player  choice: Rock ;  Computer  choice:  Paper

1 - Rock
2 - Paper
3 - Scissors
Enter:2

Sent {"array":[1,2,2,0,1]}
Received:     [1,2,2,0,1]
Player vs Computer  Player  choice:  Paper ;  Computer  choice:  Paper
Score: 0/3  :  1/3
Result: Draw
Game state saved to game_config.json

1 - Rock
2 - Paper
3 - Scissors
Enter:
```

Рис. 6. Завантаження збереженої гри

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



Висновок:

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