МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
Кафедра ЕОМ

ЗВІТ

ЛАБОРАТОРНА РОБОТА № 3

З дисципліни «Автоматизоване проектування комп`ютерних систем»

Виконав: ст. гр. КІ-401
Ларіонов А.О
Перевірив:
Федак П.Р.

Львів 2024

ЗМІСТ

РОЗДІЛ 1. МЕТА РОБОТИ.

Розробка логіки гри

РОЗДІЛ 2. ТЕОРЕТИЧНІ ВІДОМОСТІ.

server.ino – виступає кодом серверної частини. Arduino працює як центральний вузол, який чекає на дані від клієнта через послідовний порт (Serial.available() > 0). Вхідні повідомлення інтерпретуються як команди, що вказують на певний режим гри або дії.

Реакція на запити - код виконує обчислення чи дії залежно від отриманого запиту:

Генерація відповіді - сервер відповідає через Serial.println(), відправляючи клієнту результат гри або хід AI.

Збереження стану - код підтримує стан гри, наприклад, історію ходів гравця (масив player1History) для стратегічного вибору AI. Це дозволяє серверу динамічно адаптувати відповіді залежно від попередніх дій клієнта.

Основна логіка гри (determineResult, generateServerChoice, generateStrategicChoice) виконується саме на Arduino. Клієнт лише відправляє команди та отримує результати, без необхідності виконувати обчислення.

menu.cpp - Цей файл містить код для меню гри та перевірку правильності введених даних користувачем.

game.cpp – файль реалізує логіку гри, яка дозволяє взаємодіяти з сервером, отримувати дані про вибори гравців і результат гри, а також зберігати та завантажувати стан гри та статистику.

game_logic.cpp - цей код реалізує основну логіку вибору та подальшу взаємодію різних режимів гри, а саме

man vs man

man vs ai

ai vs ai

serial.cpp - Цей файл відповідає за ініціалізацію та керування серійним портом для комунікації між комп'ютером і пристроєм через серійний порт.

РОЗДІЛ 3. ЗАВДАННЯ.

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/** if need); b. run tests; c. create artifacts with binaries and test reports; 2. Required steps.

РОЗДІЛ 4. ХІД РОБОТИ.

4.1. Розвиток гри

Створив комунікацію між сервером та клієнтом для подальшого розвитку та формування запитів з клієнта.

```
 * @throws exit(1) If the port cannot be opened.
 */
void initSerialPort(const std::string& portName) {
    // Convert std::string to std::wstring
    std::wstring widePortName = stringToWString(portName);

    // Use CreateFileW for wide strings
    hSerial = CreateFileW(widePortName.c_str(),
        GENERIC_READ | GENERIC_WRITE,
        0,
        0,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        0);

    if (hSerial == INVALID_HANDLE_VALUE) {
        cerr << "Failed to open port!" << endl;
        exit(1);  ///< Exit the program if the port cannot be opened.
    }

    // Configure the serial port
```

Serial.cpp

```
 */
int main() {
    cout << "[info] Starting program..." << endl;
    initSerialPort("\\\\.\\COM3");
    cout << "[info] Serial port initialized." << endl;
```

Main.cpp

Створив меню для вибору режимів гри

```
 */
void menu() {
    cout << "\n============= Game Menu =============" << endl;
    cout << "1. Man vs AI" << endl;
    cout << "2. Man vs Man" << endl;
    cout << "3. AI vs AI (Random Move)" << endl;
    cout << "4. AI vs AI (Win Strategy)" << endl;
    cout << "5. Exit" << endl;
    cout << "=====================================" << endl;
}
```

Menu.cpp

```
if (startChoice == 1) {
    cout << "[info] Starting a new game." << endl;
    menu();
```

main.cpp

Подальша логіка запитів клієнтської частини гри реалізована в файлах game_logi.cpp, game.cpp

Визначенян результату гри відбуваєтсья на сервері

```
String determineResult(String player1, String player2) {
  if (player1 == player2) return "draw";

  if ((player1 == "rock" && player2 == "scissors") ||
      (player1 == "scissors" && player2 == "paper") ||
      (player1 == "paper" && player2 == "rock")) {
    return "Player 1 wins";
  }

  return "Player 2 wins";
}
```

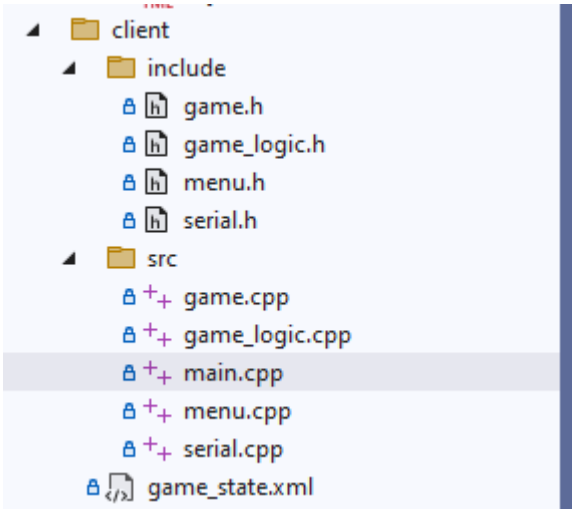
Також тут формуються ходи в AI рижим гри

```
String generateServerChoice() {
  int choice = random(3);
  if (choice == 0) return "rock";
  if (choice == 1) return "paper";
  return "scissors";
}

// Function that analyzes the history of player 1 and chooses the winning move
String generateStrategicChoice() {
  int rockCount = 0;
  int paperCount = 0;
  int scissorsCount = 0;

  // Analysis of the history of the last 3 moves of player 1
  for (int i = 0; i < 3; i++) {
    if (player1History[i] == "rock") rockCount++;
    else if (player1History[i] == "paper") paperCount++;
    else if (player1History[i] == "scissors") scissorsCount++;
  }

  // Choose a strategy for player 2 based on player 1's most common move
  if (rockCount >= paperCount && rockCount >= scissorsCount) {
    return "paper";
  }
  else if (paperCount >= rockCount && paperCount >= scissorsCount) {
    return "scissors";
  }
  else {
    return "rock";
  }
}
```

Запис відбувається у айлі game.cpp

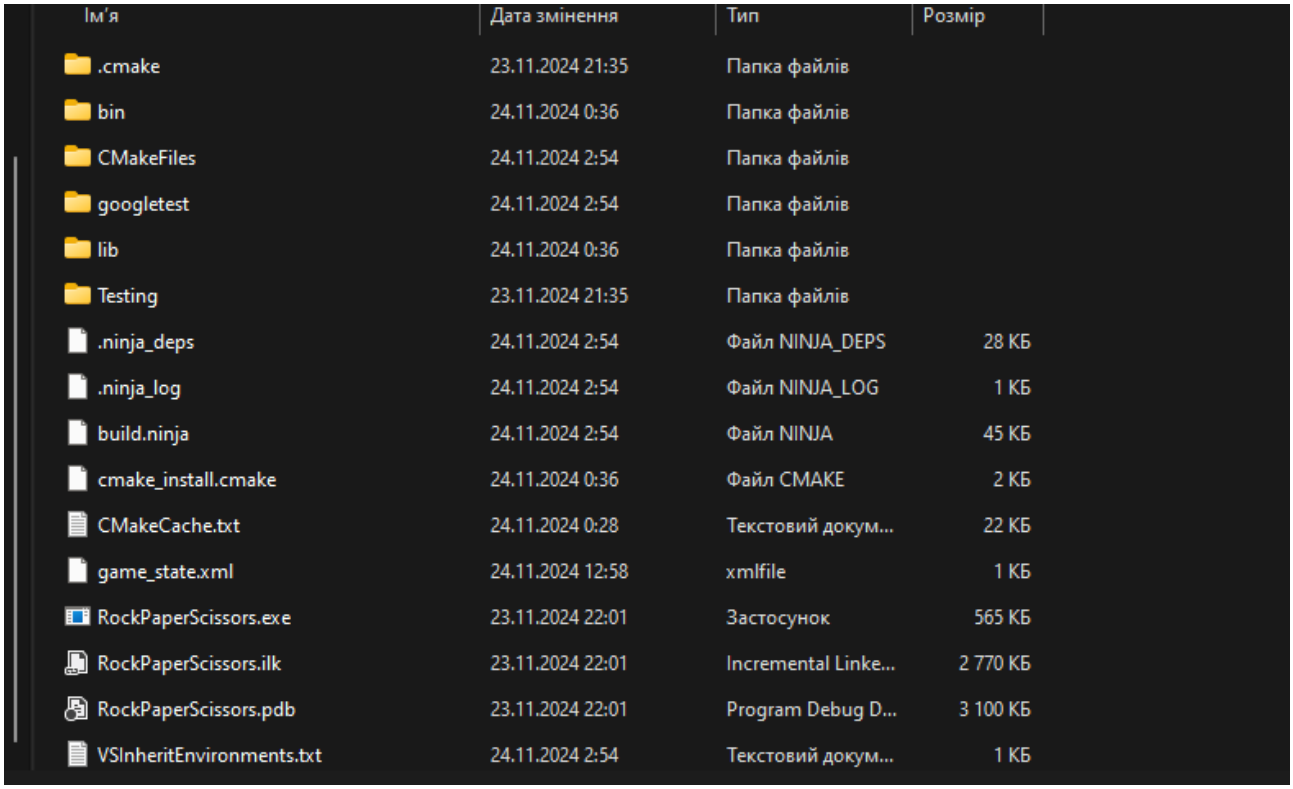
```
*/
void saveGameState(const std::string& player1, const std::string& player2, const std::string& result, int currentGameMode, bool resetStats = true) {
    if (resetStats && currentGameMode != lastGameMode) {
        resetStatistics();
        totalGames = 0;  ///< Reset the total number of games when changing the mode
        lastGameMode = currentGameMode;
    }

    if (!player1.empty() && !player2.empty()) {  // Increment totalGames only after the game is complete
        totalGames++;
        if (result == "Player 1 wins") {
            player1Wins++;
        }
        else if (result == "Player 2 wins") {
            player2Wins++;
        }
        else if (result == "draw") {
            draws++;
        }
    }

    // Write to file
    std::ofstream file("game_state.xml");
    if (!file.is_open()) {
        std::cerr << "Error opening file for writing!" << std::endl;
        return;
    }

    file << "<?xml version=\"1.0\"?>\n";
    file << "<GameState>\n";
    file << "    <GameMode>" << currentGameMode << "</GameMode>\n";
    file << "    <Game>\n";
    file << "        <Player1>" << player1 << "</Player1>\n";
    file << "        <Player2>" << player2 << "</Player2>\n";
```

Вміст проекту має вигляд



Відредагував CMakeLists.txt для нової збірки проекту

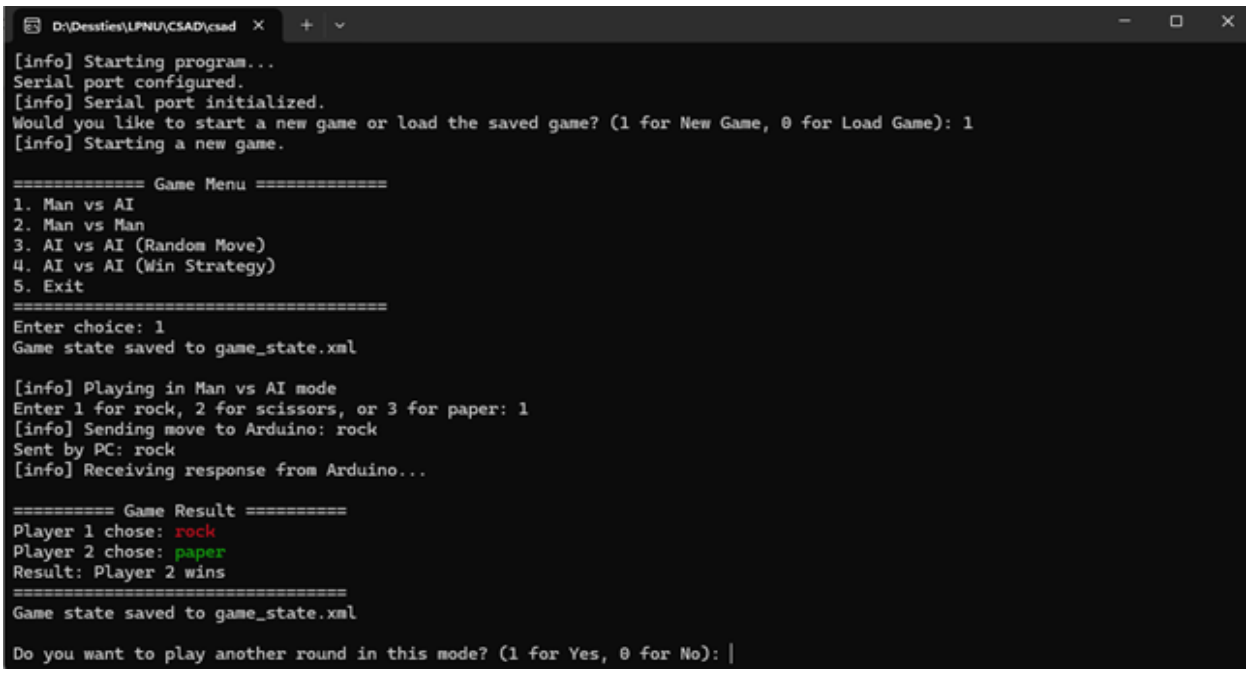
```
# Add an executable file for the client with all the necessary files
add_executable(RockPaperScissors
    ${CMAKE_SOURCE_DIR}/client/src/main.cpp
    ${CMAKE_SOURCE_DIR}/client/src/game.cpp
    ${CMAKE_SOURCE_DIR}/client/src/game_logic.cpp
    ${CMAKE_SOURCE_DIR}/client/src/menu.cpp
    ${CMAKE_SOURCE_DIR}/client/src/serial.cpp
)
```



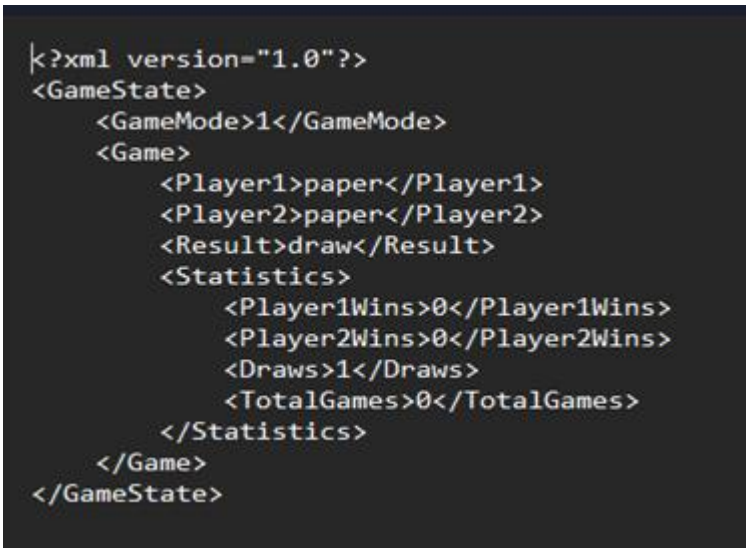
Ім'я	Дата змінення	Тип	Розмір
.cmake	23.11.2024 21:35	Папка файлів	
bin	24.11.2024 0:36	Папка файлів	
CMakeFiles	24.11.2024 2:54	Папка файлів	
googletest	24.11.2024 2:54	Папка файлів	
lib	24.11.2024 0:36	Папка файлів	
Testing	23.11.2024 21:35	Папка файлів	
.ninja_deps	24.11.2024 2:54	Файл NINJA_DEPS	28 КБ
.ninja_log	24.11.2024 2:54	Файл NINJA_LOG	1 КБ
build.ninja	24.11.2024 2:54	Файл NINJA	45 КБ
cmake_install.cmake	24.11.2024 0:36	Файл CMAKE	2 КБ
CMakeCache.txt	24.11.2024 0:28	Текстовий докум...	22 КБ
game_state.xml	24.11.2024 12:58	xmlfile	1 КБ
RockPaperScissors.exe	23.11.2024 22:01	Застосунок	565 КБ
RockPaperScissors.ilk	23.11.2024 22:01	Incremental Linke...	2 770 КБ
RockPaperScissors.pdb	23.11.2024 22:01	Program Debug D...	3 100 КБ
VSInheritEnvironments.txt	24.11.2024 2:54	Текстовий докум...	1 КБ

4.2. Тестування

Підключив Arduino UNO та завантажив серверну частину на плату. Робота логіки гри продемонстрована на скріншоті.

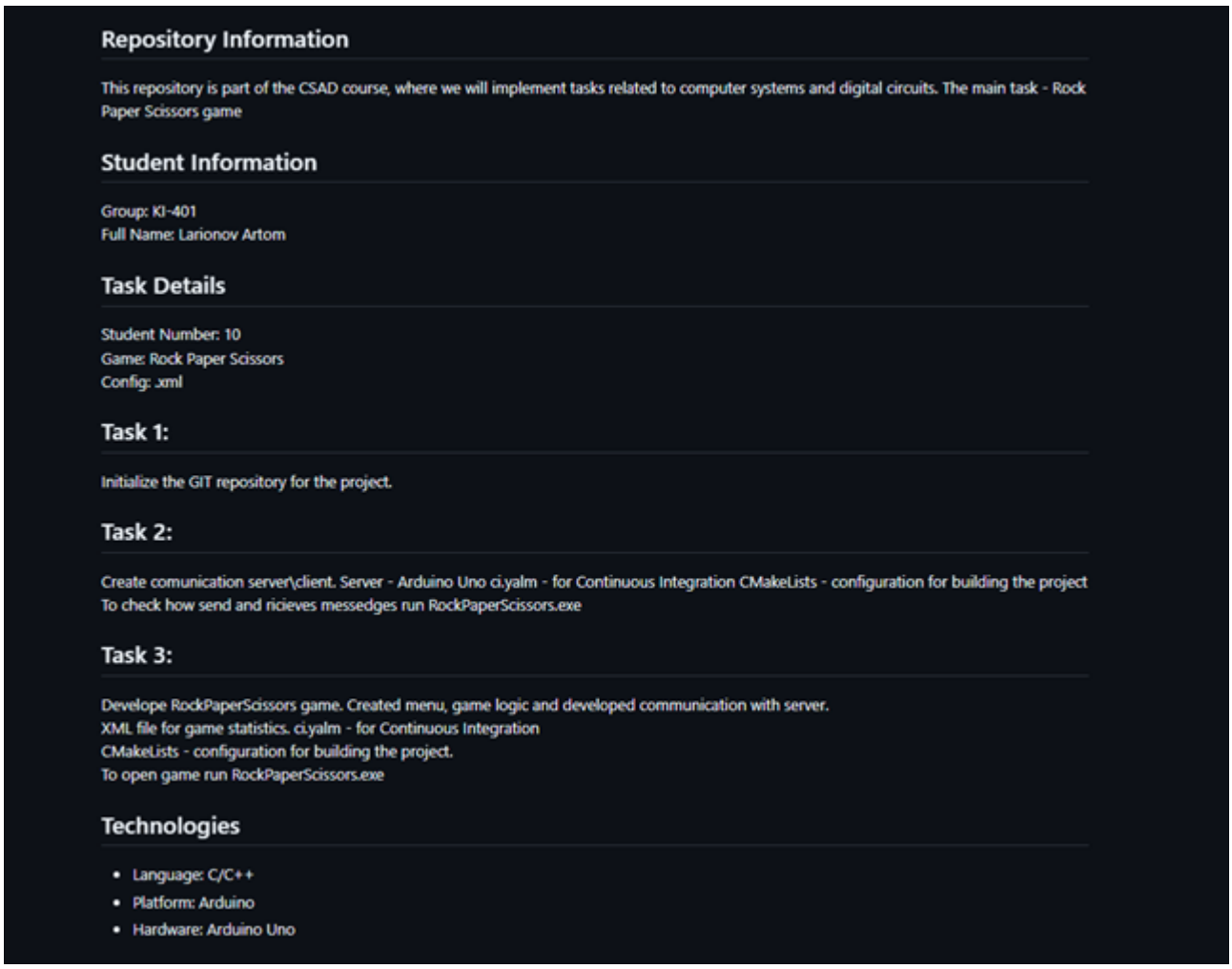


Усі результати записуються в game_state.xml



4.3. Взаємодія з репозиторієм

Відредагував README.MD



Repository Information

This repository is part of the CSAD course, where we will implement tasks related to computer systems and digital circuits. The main task - Rock Paper Scissors game

Student Information

Group: KI-401
Full Name: Larionov Artom

Task Details

Student Number: 10
Game: Rock Paper Scissors
Config: .xml

Task 1:

Initialize the GIT repository for the project.

Task 2:

Create comunication server\client. Server - Arduino Uno ci.yalm - for Continuous Integration CMakeLists - configuration for building the project
To check how send and ricieves messedges run RockPaperScissors.exe

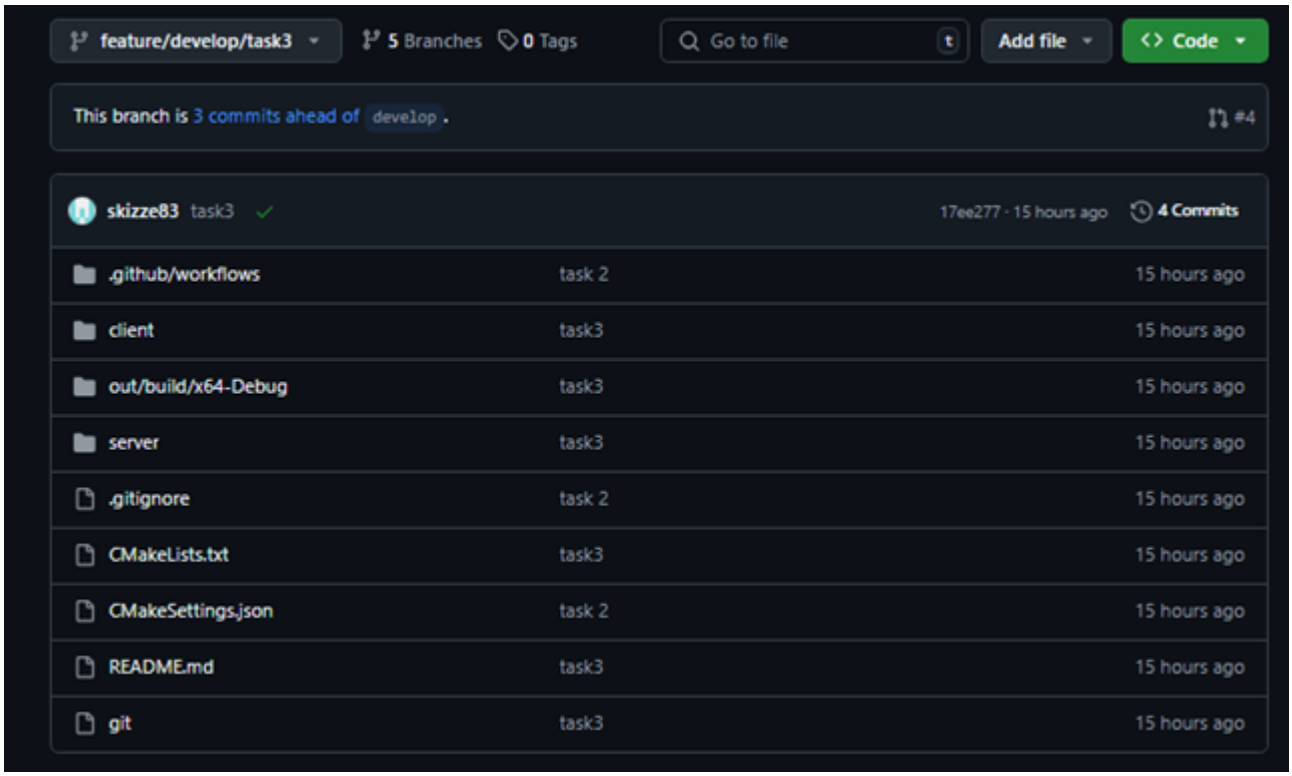
Task 3:

Develope RockPaperScissors game. Created menu, game logic and developed communication with server.
XML file for game statistics. ci.yalm - for Continuous Integration
CMakeLists - configuration for building the project.
To open game run RockPaperScissors.exe

Technologies

- Language: C/C++
- Platform: Arduino
- Hardware: Arduino Uno

Залив на гітхаб на гілку feature/develop/task3



feature/develop/task3 | 5 Branches | 0 Tags | Go to file t | Add file | Code

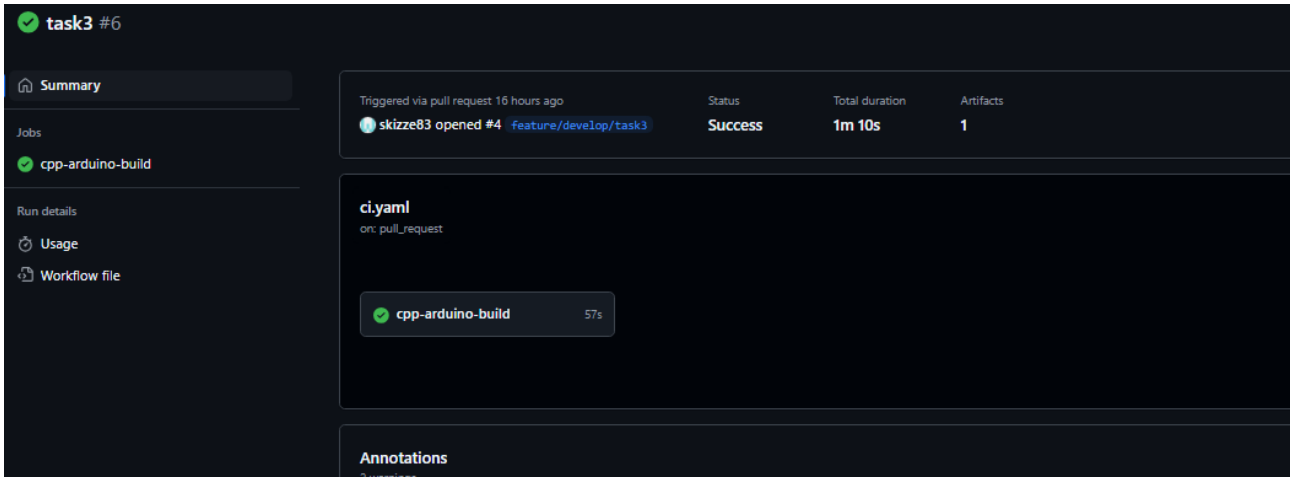
This branch is 3 commits ahead of develop. #4

skizze83 task3 17ee277 · 15 hours ago 4 Commits

.github/workflows	task 2	15 hours ago
client	task3	15 hours ago
out/build/x64-Debug	task3	15 hours ago
server	task3	15 hours ago
.gitignore	task 2	15 hours ago
CMakeLists.txt	task3	15 hours ago
CMakeSettings.json	task 2	15 hours ago
README.md	task3	15 hours ago
git	task3	15 hours ago

Проект згенерувався правильно за заданою конфігурацією ci.yalm на

GitHub Action



ВИСНОВКИ

На лабораторній роботі розробив гру камінь ножиці папір. Розвинув комунікації між сервером та клієнтом різними запитами різної складності. Усю обробку гри заклав у серверну частину, а клієнтська тільки формує запити.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *GitHub Docs - https://docs.github.com/en*
2. *CSAD Instructions for practical tasks and coursework from "Computer systems automated design"*
3. *https://github.com/skizze83/csad2425ki401larionovao10*
4. *CMake - https://cmake.org/download/*

ДОДАТОК А. MAIN.CPP

```
 /**
 * @mainpage
 *
 * # RockPaperScissors
 *
 * RockPaperScissors is a game that allows playing in different modes:
 * - Man vs Man
 * - Man vs AI
 * - AI vs AI
 *
 * ## Features:
 * - Choose game modes from the main menu.
 * - AI configuration options.
 * - Different strategies for gameplay.
 *
 * ## How to Run:
 * 1. Download the project.
 * 2. Set up the serial port.
 * 3. Run the program RockPaperScissors.exe.
 *

/**
 * @file main.cpp
 * @brief Main program for the game with serial communication and game modes.
 */

#include <iostream>
#include <string>
#include "menu.h"
#include "game_logic.h"
#include "game.h"
#include "serial.h"

using namespace std;

int currentGameMode = -1;

/**
 * @brief Initializes a new game with the given game mode.
 *
 * This function resets statistics and initializes the game state
 * with the chosen game mode.
 *
 * @param gameMode The game mode to initialize.
 */
void initializeNewGame(int gameMode) {
    currentGameMode = gameMode;
    resetStatistics();
    totalGames = 0;
    saveGameState("", "", "", currentGameMode, true); // Keeps the initial state with
reset statistics
}

/**
 * @brief Main function to start the game program.
 *
 * Handles the user input for selecting a new game or loading a saved game,
 * initializing the serial communication, and playing the selected game mode.
 *
 * @return int Returns 0 upon successful execution.
 */
int main() {
    cout << "[info] Starting program..." << endl;
```

```
    initSerialPort("\\\\.\\COM3");
    cout << "[info] Serial port initialized." << endl;

    bool isLoadedGame = false;
    string player1, player2, result;
    int startChoice;

    /**
     * @brief Ask the user if they want to start a new game or load a saved game.
     *
     * Ensures that the input is either 1 for New Game or 0 for Load Game.
     */
    cout << "Would you like to start a new game or load the saved game? (1 for New
Game, 0 for Load Game): ";
    while (!(cin >> startChoice) || (startChoice != 1 && startChoice != 0)) {
        cout << "[error] Invalid input. Please enter 1 (New Game) or 0 (Load Game): ";
        cin.clear();
        cin.ignore(10000, '\n');
    }
    cin.ignore();

    if (startChoice == 1) {
        cout << "[info] Starting a new game." << endl;
        menu();
        int newGameMode = getValidatedChoice(1, 5);
        if (newGameMode == 5) {
            cout << "[info] Exiting program..." << endl;
            closeSerialPort();
            return 0;
        }
        initializeNewGame(newGameMode);
        isLoadedGame = false;
    }
    else if (startChoice == 0) {
        cout << "[info] Loading saved game state" << endl;
        bool loadSuccess = loadGameState(player1, player2, result, currentGameMode);
        if (loadSuccess) {
            cout << "Game state loaded from game_state.xml" << endl;
            displayResult(player1, player2, result);
            isLoadedGame = true;
        }
        else {
            cout << "[info] No saved game found or error loading game state. Starting
new game." << endl;
            menu();
            int newGameMode = getValidatedChoice(1, 5);
            if (newGameMode == 5) {
                cout << "[info] Exiting program..." << endl;
                closeSerialPort();
                return 0;
            }
            initializeNewGame(newGameMode);
            isLoadedGame = false;
        }
    }

    while (true) {
        /**
         * @brief Main game loop based on the selected game mode.
         *
         * The loop allows the user to play in the selected mode and choose to
         * play additional rounds or exit.
         */
        while (true) {
            switch (currentGameMode) {
```

```
            case 1:
                cout << "\n[info] Playing in Man vs AI mode" << endl;
                playManVsAI(player1, player2, result);
                break;
            case 2:
                cout << "\n[info] Playing in Man vs Man mode" << endl;
                playManVsMan(player1, player2, result);
                break;
            case 3:
                cout << "\n[info] Playing in AI vs AI (Random Move) mode" << endl;
                playAIVsAIRandom(player1, player2, result);
                break;
            case 4:
                cout << "\n[info] Playing in AI vs AI (Win Strategy) mode" << endl;
                playAIVsAIWinStrategy(player1, player2, result);
                break;
            }

            displayResult(player1, player2, result);
            saveGameState(player1, player2, result, currentGameMode, false);

            cout << "\nDo you want to play another round in this mode? (1 for Yes, 0
for No): ";
            int continueChoice;
            while (!(cin >> continueChoice) || (continueChoice != 0 && continueChoice
!= 1)) {
                cout << "[error] Invalid input. Please enter 1 (Yes) or 0 (No): ";
                cin.clear();
                cin.ignore(10000, '\n');
            }
            cin.ignore();

            if (continueChoice == 0) {
                break;
            }
        }

        /**
         * @brief Asks the user if they want to select a different game mode or exit.
         *
         * The user can choose to select a new mode or exit the program.
         */
        cout << "\nDo you want to select a different game mode or exit? (1 for Select
Mode, 0 for Exit): ";
        int nextAction;
        while (!(cin >> nextAction) || (nextAction != 1 && nextAction != 0)) {
            cout << "[error] Invalid input. Please enter 1 (Select Mode) or 0 (Exit):
";
            cin.clear();
            cin.ignore(10000, '\n');
        }
        cin.ignore();

        if (nextAction == 0) {
            cout << "[info] Exiting program..." << endl;
            break;
        }
        else if (nextAction == 1) {
            menu();
            int newGameMode = getValidatedChoice(1, 5);
            if (newGameMode == 5) {
                cout << "[info] Exiting program..." << endl;
                break;
            }
            initializeNewGame(newGameMode);
```

```
        }
    }

    closeSerialPort();
    cout << "[info] Program finished." << endl;
    return 0;
}
```

ДОДАТОК Б. GAME.CPP

```
/**
 * @file game.cpp
 * @brief Contains functions for managing game state, processing player choices, and
handling game statistics.
 */

#include <iostream>
#include <string>
#include <fstream>
#include <regex>
#include "game.h"
#include "serial.h"

using namespace std;

// Global variables for statistics
int player1Wins = 0, player2Wins = 0, draws = 0, totalGames = 0;
int lastGameMode = -1;  ///< Initial value to determine the mode change

/**
 * @brief Converts a number to the corresponding string representation of a player's
choice.
 *
 * @param number The number representing a choice (1 for rock, 2 for scissors, 3 for
paper).
 * @return A string representing the choice ("rock", "scissors", or "paper").
 */
std::string getChoiceFromNumber(int number) {
    if (number == 1) return "rock";
    else if (number == 2) return "scissors";
    else if (number == 3) return "paper";
    return "";
}

/**
 * @brief Prompts the user to enter their choice and returns the corresponding string
representation.
 *
 * This function continues to ask for input until the user enters a valid choice (1,
2, or 3).
 *
 * @return A string representing the user's choice ("rock", "scissors", or "paper").
 */
std::string getUserChoice() {
    int choice;
    while (true) {
        std::cout << "Enter 1 for rock, 2 for scissors, or 3 for paper: ";
        if (std::cin >> choice && (choice >= 1 && choice <= 3)) {
            if (choice == 1) return "rock";
            if (choice == 2) return "scissors";
            return "paper";
        }
        else {
            std::cout << "[error] Invalid choice. Please enter 1, 2, or 3." <<
std::endl;
            std::cin.clear();
            std::cin.ignore(10000, '\n');
        }
    }
}

/**
 * @brief Parses the server response and extracts player choices and game result.
```

```
 *
 * The function uses regular expressions to extract the information from the server
response string.
 *
 * @param response The server response in JSON format.
 * @param player1 The string to store Player 1's choice.
 * @param player2 The string to store Player 2's choice.
 * @param result The string to store the game result ("Player 1 wins", "Player 2
wins", or "draw").
 */
void parseResponse(const std::string& response, std::string& player1, std::string&
player2, std::string& result) {
    std::regex player1Regex("\"Player1\": \"(.*?)\"");
    std::regex player2Regex("\"Player2\": \"(.*?)\"");
    std::regex serverRegex("\"Server\": \"(.*?)\"");
    std::regex resultRegex("\"Result\": \"(.*?)\"");
    std::smatch match;

    if (std::regex_search(response, match, player1Regex) && match.size() > 1) {
        player1 = match.str(1);
    }

    if (std::regex_search(response, match, player2Regex) && match.size() > 1) {
        player2 = match.str(1);
    }
    else if (std::regex_search(response, match, serverRegex) && match.size() > 1) {
        player2 = match.str(1);
    }

    if (std::regex_search(response, match, resultRegex) && match.size() > 1) {
        result = match.str(1);
    }
}

/**
 * @brief Resets the game statistics (wins, draws, and total games).
 *
 * This function is called to reset the statistics at the start of a new game mode or
to clear accumulated statistics.
 */
void resetStatistics() {
    player1Wins = 0;
    player2Wins = 0;
    draws = 0;
    totalGames = 0;
}

/**
 * @brief Saves the current game state to an XML file.
 *
 * This function writes the current game state, including the players' choices, the
result, and statistics, to the "game_state.xml" file.
 * It also resets statistics if a new game mode is selected.
 *
 * @param player1 The choice made by Player 1.
 * @param player2 The choice made by Player 2.
 * @param result The result of the game ("Player 1 wins", "Player 2 wins", or "draw").
 * @param currentGameMode The current game mode.
 * @param resetStats Whether to reset the statistics (default is true).
 */
void saveGameState(const std::string& player1, const std::string& player2, const
std::string& result, int currentGameMode, bool resetStats = true) {
    if (resetStats && currentGameMode != lastGameMode) {
        resetStatistics();
        totalGames = 0;  ///< Reset the total number of games when changing the mode
```

```
        lastGameMode = currentGameMode;
    }

    if (!player1.empty() && !player2.empty()) {  // Increment totalGames only after
the game is complete
        totalGames++;
        if (result == "Player 1 wins") {
            player1Wins++;
        }
        else if (result == "Player 2 wins") {
            player2Wins++;
        }
        else if (result == "draw") {
            draws++;
        }
    }

    // Write to file
    std::ofstream file("game_state.xml");
    if (!file.is_open()) {
        std::cerr << "Error opening file for writing!" << std::endl;
        return;
    }

    file << "<?xml version=\"1.0\"?>\n";
    file << "<GameState>\n";
    file << "    <GameMode>" << currentGameMode << "</GameMode>\n";
    file << "    <Game>\n";
    file << "        <Player1>" << player1 << "</Player1>\n";
    file << "        <Player2>" << player2 << "</Player2>\n";
    file << "        <Result>" << result << "</Result>\n";
    file << "        <Statistics>\n";
    file << "            <Player1Wins>" << player1Wins << "</Player1Wins>\n";
    file << "            <Player2Wins>" << player2Wins << "</Player2Wins>\n";
    file << "            <Draws>" << draws << "</Draws>\n";
    file << "            <TotalGames>" << totalGames << "</TotalGames>\n";
    file << "        </Statistics>\n";
    file << "    </Game>\n";
    file << "</GameState>\n";

    file.close();
    std::cout << "Game state saved to game_state.xml" << std::endl;
}

/**
 * @brief Loads the saved game state from the "game_state.xml" file.
 *
 * This function reads the saved game state from the XML file, including the players'
choices, the result, and statistics.
 *
 * @param player1 The string to store Player 1's choice.
 * @param player2 The string to store Player 2's choice.
 * @param result The string to store the game result ("Player 1 wins", "Player 2
wins", or "draw").
 * @param currentGameMode The integer to store the current game mode.
 * @return true if the game state was successfully loaded; false otherwise.
 */
bool loadGameState(std::string& player1, std::string& player2, std::string& result,
int& currentGameMode) {
    std::ifstream file("game_state.xml");
    std::string line;
    bool hasData = false;

    if (!file.is_open()) {
        std::cerr << "No saved game found." << std::endl;
```

```
        return false;
    }

    std::regex modeRegex("<GameMode>(.*?)</GameMode>");
    std::regex player1Regex("<Player1>(.*?)</Player1>");
    std::regex player2Regex("<Player2>(.*?)</Player2>");
    std::regex resultRegex("<Result>(.*?)</Result>");
    std::regex player1WinsRegex("<Player1Wins>(\\d+)</Player1Wins>");
    std::regex player2WinsRegex("<Player2Wins>(\\d+)</Player2Wins>");
    std::regex drawsRegex("<Draws>(\\d+)</Draws>");
    std::regex totalGamesRegex("<TotalGames>(\\d+)</TotalGames>");
    std::smatch match;

    while (std::getline(file, line)) {
        if (std::regex_search(line, match, modeRegex) && match.size() > 1) {
            currentGameMode = std::stoi(match.str(1));
            hasData = true;
        }
        if (std::regex_search(line, match, player1Regex) && match.size() > 1) {
            player1 = match.str(1);
        }
        if (std::regex_search(line, match, player2Regex) && match.size() > 1) {
            player2 = match.str(1);
        }
        if (std::regex_search(line, match, resultRegex) && match.size() > 1) {
            result = match.str(1);
        }
        if (std::regex_search(line, match, player1WinsRegex) && match.size() > 1) {
            player1Wins = std::stoi(match.str(1));
        }
        if (std::regex_search(line, match, player2WinsRegex) && match.size() > 1) {
            player2Wins = std::stoi(match.str(1));
        }
        if (std::regex_search(line, match, drawsRegex) && match.size() > 1) {
            draws = std::stoi(match.str(1));
        }
        if (std::regex_search(line, match, totalGamesRegex) && match.size() > 1) {
            totalGames = std::stoi(match.str(1));
        }
    }

    file.close();
    return hasData;
}
```

ДОДАТОК В. GAME_LOGIC.CPP

```cpp
/**
* @file game_logic.cpp
* @brief Implements the game logic for different game modes.
*/

#include <iostream>
#include <string>
#include <regex>
#include "game_logic.h"
#include "game.h"
#include "serial.h"

using namespace std;

/**
 * @brief Plays the game between the user (Player 1) and the AI.
 *
 * This function gets the player's choice, sends it to the Arduino, and then receives
and parses the AI's response.
 *
 * @param player1 The string to store Player 1's choice.
 * @param player2 The string to store Player 2's choice (AI).
 * @param result The string to store the game result ("Player 1 wins", "Player 2
wins", or "draw").
 */
void playManVsAI(string& player1, string& player2, string& result) {
    player1 = getUserChoice();
    cout << "[info] Sending move to Arduino: " << player1 << endl;
    sendMessage(player1.c_str());
    Sleep(1000);

    cout << "[info] Receiving response from Arduino..." << endl;
    string response = receiveMessage();
    parseResponse(response, player1, player2, result);
}

/**
 * @brief Plays the game between two human players.
 *
 * This function allows Player 1 and Player 2 to input their choices, sends them to
the Arduino, and then
 * receives and parses the game result.
 *
 * @param player1 The string to store Player 1's choice.
 * @param player2 The string to store Player 2's choice.
 * @param result The string to store the game result ("Player 1 wins", "Player 2
wins", or "draw").
 */
void playManVsMan(string& player1, string& player2, string& result) {
    player1 = getUserChoice();
    sendMessage(("Player1:" + player1).c_str());
    Sleep(1000);

    system("CLS");

    player2 = getUserChoice();
    sendMessage(("Player2:" + player2).c_str());
    Sleep(1000);

    string response = receiveMessage();
    parseResponse(response, player1, player2, result);
}
```

```
/**
 * @brief Plays an AI vs AI game with random strategies.
 *
 * This function sends the "AI_vs_AI_random" command to the Arduino, then receives and
parses the result of
 * the game between the two AIs.
 *
 * @param player1 The string to store Player 1's choice (AI).
 * @param player2 The string to store Player 2's choice (AI).
 * @param result The string to store the game result ("Player 1 wins", "Player 2
wins", or "draw").
 */
void playAIVsAIRandom(string& player1, string& player2, string& result) {
    sendMessage("AI_vs_AI_random");
    Sleep(1000);

    cout << "[info] Receiving response from Arduino..." << endl;
    string response = receiveMessage();
    parseResponse(response, player1, player2, result);
}

/**
 * @brief Plays an AI vs AI game with a win-strategy.
 *
 * This function sends the "AI_vs_AI_win_strategy" command to the Arduino, then
receives and parses the result
 * of the game between the two AIs using the win strategy.
 *
 * @param player1 The string to store Player 1's choice (AI).
 * @param player2 The string to store Player 2's choice (AI).
 * @param result The string to store the game result ("Player 1 wins", "Player 2
wins", or "draw").
 */
void playAIVsAIWinStrategy(string& player1, string& player2, string& result) {
    sendMessage("AI_vs_AI_win_strategy");
    Sleep(1000);

    cout << "[info] Receiving response from Arduino..." << endl;
    string response = receiveMessage();
    parseResponse(response, player1, player2, result);
}

/**
 * @brief Displays the result of the game, including players' choices and the winner.
 *
 * This function prints the choices made by Player 1 and Player 2, and highlights the
winner in green or red.
 * The result (win, draw) is also displayed.
 *
 * @param player1 The choice made by Player 1.
 * @param player2 The choice made by Player 2.
 * @param result The result of the game ("Player 1 wins", "Player 2 wins", or "draw").
 */
void displayResult(const string& player1, const string& player2, const string& result)
{
    const string GREEN = "\033[32m";  ///< ANSI escape code for green text
    const string RED = "\033[31m";    ///< ANSI escape code for red text
    const string RESET = "\033[0m";   ///< ANSI escape code to reset text color

    cout << "\n========== Game Result ==========" << endl;
    if (result == "Player 1 wins") {
        cout << "Player 1 chose: " << GREEN << player1 << RESET << endl;
        cout << "Player 2 chose: " << RED << player2 << RESET << endl;
    }
    else if (result == "Player 2 wins") {
```

```
        cout << "Player 1 chose: " << RED << player1 << RESET << endl;
        cout << "Player 2 chose: " << GREEN << player2 << RESET << endl;
    }
    else {
        cout << "Player 1 chose: " << player1 << endl;
        cout << "Player 2 chose: " << player2 << endl;
    }
    cout << "Result: " << result << endl;
    cout << "================================" << endl;
}
```

ДОДАТОК Г. SERIAL.CPP

```
 /**
 * @file serial.cpp
 * @brief Contains functions for initializing and managing serial port communication.
 */

#pragma once
#include <iostream>
#include <windows.h>
#include <string>
#include "serial.h"

using namespace std;

HANDLE hSerial;  ///< Handle to the serial port.
DCB dcbSerialParams = { 0 };  ///< Serial port parameters (DCB structure).
COMMTIMEOUTS timeouts = { 0 };  ///< Timeout settings for serial port communication.

/**
 * @brief Converts a std::string to a std::wstring.
 *
 * This helper function converts a narrow character string (std::string) to a wide
character
 * string (std::wstring), which is needed for working with the Windows API.
 *
 * @param str The std::string to be converted.
 *
 * @return std::wstring The converted wide character string.
 */
std::wstring stringToWString(const std::string& str) {
    return std::wstring(str.begin(), str.end());
}

/**
 * @brief Initializes the serial port with specified settings.
 *
 * This function configures the serial port with the standard settings: 9600 baud
rate, 8 data bits,
 * 1 stop bit, and no parity. It also sets timeouts for read and write operations.
 *
 * @param portName The name of the serial port (e.g., "COM3").
 *
 * @throws exit(1) If the port cannot be opened.
 */
void initSerialPort(const std::string& portName) {
    // Convert std::string to std::wstring
    std::wstring widePortName = stringToWString(portName);

    // Use CreateFileW for wide strings
    hSerial = CreateFileW(widePortName.c_str(),
        GENERIC_READ | GENERIC_WRITE,
        0,
        0,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        0);

    if (hSerial == INVALID_HANDLE_VALUE) {
        cerr << "Failed to open port!" << endl;
        exit(1);  ///< Exit the program if the port cannot be opened.
    }

    // Configure the serial port
    dcbSerialParams.DCBlength = sizeof(dcbSerialParams);
```

```
    GetCommState(hSerial, &dcbSerialParams);
    dcbSerialParams.BaudRate = CBR_9600;
    dcbSerialParams.ByteSize = 8;
    dcbSerialParams.StopBits = ONESTOPBIT;
    dcbSerialParams.Parity = NOPARITY;
    SetCommState(hSerial, &dcbSerialParams);

    // Set timeouts
    timeouts.ReadIntervalTimeout = 50;
    timeouts.ReadTotalTimeoutConstant = 50;
    timeouts.ReadTotalTimeoutMultiplier = 10;
    timeouts.WriteTotalTimeoutConstant = 50;
    timeouts.WriteTotalTimeoutMultiplier = 10;
    SetCommTimeouts(hSerial, &timeouts);

    cout << "Serial port configured." << endl;
}

/**
 * @brief Sends a message to the serial port.
 *
 * This function sends a null-terminated string (message) to the serial port.
 *
 * @param message The message to send.
 */
void sendMessage(const char* message) {
    DWORD bytesWritten;
    WriteFile(hSerial, message, strlen(message), &bytesWritten, NULL);
    cout << "Sent by PC: " << message << endl;
}

/**
 * @brief Receives a message from the serial port.
 *
 * This function reads a message from the serial port into a buffer. It returns
 * the received message as a std::string. If no message is received, an empty string
is returned.
 *
 * @return std::string The message received from the serial port.
 */
std::string receiveMessage() {
    char buffer[1024] = { 0 };
    DWORD bytesRead;
    bool readSuccess = ReadFile(hSerial, buffer, sizeof(buffer), &bytesRead, NULL);

    if (readSuccess && bytesRead > 0) {
        return string(buffer);
    }
    else {
        return "";
    }
}

/**
 * @brief Closes the serial port.
 *
 * This function closes the serial port and prints a confirmation message.
 */
void closeSerialPort() {
    CloseHandle(hSerial);
    cout << "Serial port closed." << endl;
}
```

ДОДАТОК Д. SERVER.INO

```
String player1Choice = "";
String player2Choice = "";

// Player 1's move history for a winning strategy
String player1History[3] = {"", "", ""};
int historyIndex = 0;

void setup() {
  Serial.begin(9600);
  Serial.println("Arduino ready");

  randomSeed(analogRead(0)); // Initialize the random number generator
}

void loop() {
  if (Serial.available() > 0) {
    String clientMessage = Serial.readString();
    clientMessage.trim();

    if (clientMessage.startsWith("Player1:")) {
      // Human vs. Human mode — we get player 1's choice
      player1Choice = clientMessage.substring(8);
    }
    else if (clientMessage.startsWith("Player2:")) {
      // Human vs. Human mode — we get player 2's choice and process the result
      player2Choice = clientMessage.substring(8);

      //  result
      String result = determineResult(player1Choice, player2Choice);
      String response = "{ \"Player1\": \"" + player1Choice + "\", \"Player2\": \""
+ player2Choice + "\", \"Result\": \"" + result + "\" }";
      Serial.println(response);

      // Clear the selections for the new game
      player1Choice = "";
      player2Choice = "";
    }
    else if (clientMessage == "get_ai_choice") {
      // For AI vs. AI or Human vs. AI mode
      String aiChoice = generateServerChoice();
      Serial.println(aiChoice); // Send the AI selection to the client
    }
    else if (clientMessage == "AI_vs_AI_random") {
      // AI vs. AI mode with random selection for each player
      player1Choice = generateServerChoice();
      player2Choice = generateServerChoice();

      // result
      String result = determineResult(player1Choice, player2Choice);
```

```
      // Form and send the response
      String response = "{ \"Player1\": \"" + player1Choice + "\", \"Player2\": \""
+ player2Choice + "\", \"Result\": \"" + result + "\" }";
      Serial.println(response);

     // Clear the selections for the new game
      player1Choice = "";
      player2Choice = "";
    }
    else if (clientMessage == "AI_vs_AI_win_strategy") {
      // AI vs. AI mode using a winning strategy for player 2

      // The choice of player 1 is random
      player1Choice = generateServerChoice();

      // We remember the choice of player 1 in history
      player1History[historyIndex] = player1Choice;
      historyIndex = (historyIndex + 1) % 3;

     // Player 2's choice is strategic
      player2Choice = generateStrategicChoice();

      // result
      String result = determineResult(player1Choice, player2Choice);

      // Form and send the response
      String response = "{ \"Player1\": \"" + player1Choice + "\", \"Player2\": \""
+ player2Choice + "\", \"Result\": \"" + result + "\" }";
      Serial.println(response);

     // Clear the selections for the new game
      player1Choice = "";
      player2Choice = "";
    }
    else {
      // For Human vs. AI mode - player submits their choice, server generates AI
choice
      String serverChoice = generateServerChoice();
      String result = determineResult(clientMessage, serverChoice);

      // Send server selection and result
      String response = "{ \"Server\": \"" + serverChoice + "\", \"Result\": \"" +
result + "\" }";
      Serial.println(response);
    }
  }
}

String generateServerChoice() {
  int choice = random(3);
```

```
  if (choice == 0) return "rock";
  if (choice == 1) return "paper";
  return "scissors";
}

// Function that analyzes the history of player 1 and chooses the winning move
String generateStrategicChoice() {
  int rockCount = 0;
  int paperCount = 0;
  int scissorsCount = 0;

  // Analysis of the history of the last 3 moves of player 1
  for (int i = 0; i < 3; i++) {
    if (player1History[i] == "rock") rockCount++;
    else if (player1History[i] == "paper") paperCount++;
    else if (player1History[i] == "scissors") scissorsCount++;
  }

  // Choose a strategy for player 2 based on player 1's most common move
  if (rockCount >= paperCount && rockCount >= scissorsCount) {
    return "paper";
  }
  else if (paperCount >= rockCount && paperCount >= scissorsCount) {
    return "scissors";
  }
  else {
    return "rock";
  }
}

String determineResult(String player1, String player2) {
  if (player1 == player2) return "draw";

  if ((player1 == "rock" && player2 == "scissors") ||
      (player1 == "scissors" && player2 == "paper") ||
      (player1 == "paper" && player2 == "rock")) {
    return "Player 1 wins";
  }

  return "Player 2 wins";
}
```

ДОДАТОК Е. MENU.CPP

```cpp
/**
 * @file menu.cpp
 * @brief Contains functions for displaying the game menu and validating user input.
 */

#include <iostream>
#include "menu.h"

using namespace std;

/**
 * @brief Displays the game menu with options for the player.
 *
 * This function prints a list of game modes to the console, allowing the user to
choose
 * a game mode to start. The options include different modes for playing against AI or
another player.
 */
void menu() {
    cout << "\n============= Game Menu =============" << endl;
    cout << "1. Man vs AI" << endl;
    cout << "2. Man vs Man" << endl;
    cout << "3. AI vs AI (Random Move)" << endl;
    cout << "4. AI vs AI (Win Strategy)" << endl;
    cout << "5. Exit" << endl;
    cout << "=====================================" << endl;
}

/**
 * @brief Prompts the user for a valid choice within a specified range.
 *
 * This function repeatedly asks the user to input a number and validates that the
choice
 * is within the specified range (min to max). If the input is invalid, the user will
be
 * prompted again.
 *
 * @param min The minimum valid value for the choice.
 * @param max The maximum valid value for the choice.
 *
 * @return int The validated choice entered by the user.
 */
int getValidatedChoice(int min, int max) {
    int choice;
    while (true) {
        cout << "Enter choice: ";
        if (cin >> choice && choice >= min && choice <= max) {
            cin.ignore(10000, '\n');  ///< Clears the input buffer.
            return choice;
        }
        else {
            cout << "[error] Invalid input. Please enter a number between " << min <<
" and " << max << "." << endl;
            cin.clear();  ///< Clears the error flag on the input stream.
            cin.ignore(10000, '\n');  ///< Clears the input buffer.
        }
    }
}
```

ДОДАТОК Ж. CMAKELISTS.TXT

```
 # Minimum version of CMake
cmake_minimum_required(VERSION 3.10)

# Project name
project(RockPaperScissors)

# Set the C++ standard
set(CMAKE_CXX_STANDARD 17)

# Specify directories with header files
include_directories(${CMAKE_SOURCE_DIR}/client/include)

# Add an executable file for the client with all the necessary files
add_executable(RockPaperScissors
    ${CMAKE_SOURCE_DIR}/client/src/main.cpp
    ${CMAKE_SOURCE_DIR}/client/src/game.cpp
    ${CMAKE_SOURCE_DIR}/client/src/game_logic.cpp
    ${CMAKE_SOURCE_DIR}/client/src/menu.cpp
    ${CMAKE_SOURCE_DIR}/client/src/serial.cpp
)

# Adding definitions for Unicode support
add_definitions(-DUNICODE -D_UNICODE)

# Specify that the Windows library should be used
if(WIN32)
    target_link_libraries(RockPaperScissors PRIVATE ws2_32)
endif()

# Add Google Test subdirectory
add_subdirectory(googletest)

# Add executable for tests and link with Google Test libraries
add_executable(tests
    ${CMAKE_SOURCE_DIR}/test/tests.cpp
)

# Link tests executable with gtest and gtest_main
target_link_libraries(tests gtest gtest_main)
```

ДОДАТОК 3. CI.YALM

```yaml
name: CI for C++ and Arduino Uno

on:
  push:
    branches:
      - '**'
  pull_request:
    branches:
      - '**'

jobs:
  cpp-arduino-build:
    runs-on: windows-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Download Arduino CLI
        uses: arduino/setup-arduino-cli@v1
        with:
          version: '0.19.2'

      - name: Download CMake
        run: choco install cmake

      - name: Download platform Arduino AVR
        run: arduino-cli core install arduino:avr

      - name: Config CMake
        run: cmake -S . -B build
        shell: cmd

      - name: Build with CMake
        run: cmake --build build
        shell: cmd

      - name: Download build artefacts
        uses: actions/upload-artifact@v3
        with:
          name: build-artifacts
          path: build/*
```