МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ЛЬВІВСЬКА ПОЛІТЕХНІКА

Автоматизоване проектування комп'ютерних систем

Task 3. Implement Server (HW) and Client (SW) parts of game (FEF)

Виконав:

ст. гр КІ - 401

Сторожков О. В.

Прийняв:

Федак П. Р.

2024

Опис теми

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

Теоретичні відомості

PlatformIO — це інструмент для розробки вбудованого програмного забезпечення з відкритим кодом. Він підтримує різні платформи мікроконтролерів та фреймворки, інтегрується з популярними середовищами розробки (наприклад, Visual Studio Code) і має вбудовану систему управління бібліотеками. PlatformIO спрощує розробку, компіляцію та завантаження програм на мікроконтролери.

JSON (JavaScript Object Notation) — це легкий формат обміну даними, зрозумілий для людини та легко оброблюваний машинами. Він використовує структуру з пар "ключ: значення" і підтримує вкладені об'єкти та масиви. JSON широко використовується для передачі даних у веб-додатках завдяки простоті синтаксису.

Виконання завдання

1. Розробив серверну та клієнтську частину гри:

main,py

```
import serial
import time
import threading
import json
import os

CONFIG_FILE = 'config/game_config.json'


def setup_serial_port():
    try:
        port = input("Enter the serial port (e.g., /dev/ttyUSB0 or COM3): ")
        return serial.Serial(port, 9600, timeout=1)
    except serial.SerialException as e:
        print(f"Error: {e}")
        exit(1)


def send_message(message, ser):
    try:
        ser.write((message + '\n').encode())
    except serial.SerialException as e:
        print(f"Error sending message: {e}")


def receive_message(ser):
    try:
        received = ser.readline().decode('utf-8', errors='ignore').strip()
        if received:
            print(received)
        return received
    except serial.SerialException as e:
        print(f"Error receiving message: {e}")
        return None


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


def user_input_thread(ser):
    global can_input
    while True:
        if can_input:
            user_message = input()
            if user_message.lower() == 'exit':
                print("Exiting...")
                global exit_program
                exit_program = True
                break
            elif user_message.lower().startswith('save'):
                save_game_config(user_message)
            elif user_message.lower().startswith('load'):
                file_path = input("Enter the path to the configuration file:
")
                load_game_config(file_path, ser)
            send_message(user_message, ser)
            can_input = False


def monitor_incoming_messages(ser):
    global can_input
    global last_received_time
    while not exit_program:
        received = receive_message(ser)
        if received:
            last_received_time = time.time()
            if not can_input:
                can_input = True


def save_game_config(message):
    config = {
        "gameMode": 0,
        "player1Symbol": 'X',
        "player2Symbol": 'O'
    }

    try:
        params = message.split()
        if len(params) == 2 and params[1] in ['0', '1', '2']:
            config["gameMode"] = int(params[1])

        with open(CONFIG_FILE, 'w') as f:
            json.dump(config, f)
        print(f"Configuration saved to {CONFIG_FILE}")
    except Exception as e:
        print(f"Error saving configuration: {e}")


def load_game_config(file_path, ser):
    try:
        if os.path.exists(file_path):
            with open(file_path, 'r') as f:
                config = json.load(f)
                game_mode = config.get("gameMode", 0)
                player1_symbol = config.get("player1Symbol", 'X')
                player2_symbol = config.get("player2Symbol", 'O')

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

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

                json_str = json.dumps(json_message)
                print(json_str)

                send_message(json_str, ser)
        else:
            print("Configuration file not found. Please provide a valid
path.")
    except Exception as e:
        print(f"Error loading configuration: {e}")


if __name__ == "__main__":
    ser = setup_serial_port()
    can_input = True
    exit_program = False
    last_received_time = time.time()

    threading.Thread(target=monitor_incoming_messages, args=(ser,),
daemon=True).start()
    threading.Thread(target=user_input_thread, args=(ser,),
daemon=True).start()

    try:
        while not exit_program:
            if time.time() - last_received_time >= 1 and can_input:
                pass
            else:
                time.sleep(0.1)
    except KeyboardInterrupt:
        print("Exit!")
    finally:
        if ser.is_open:
            print("Closing serial port...")
            ser.close()
```

test_serial_communication.py

```
import pytest
from unittest.mock import patch, MagicMock
import serial
import sys
import os
sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__),
'..')))
```

```python
from main import send_message, receive_message, save_game_config,
load_game_config


def test_send_message():
    mock_serial = MagicMock(spec=serial.Serial)
    send_message("Hello", mock_serial)
    mock_serial.write.assert_called_with(b"Hello\n")

def test_receive_message():
    mock_serial = MagicMock(spec=serial.Serial)
    mock_serial.readline.return_value = b"Test Message\n"
    result = receive_message(mock_serial)
    assert result == "Test Message"

def test_receive_empty_message():
    mock_serial = MagicMock(spec=serial.Serial)
    mock_serial.readline.return_value = b"\n"
    result = receive_message(mock_serial)
    assert result == ""

@patch('builtins.input', return_value='COM3')
def test_serial_port(mock_input):
    mock_serial = MagicMock(spec=serial.Serial)
    mock_serial.portstr = 'COM3'
    port = 'COM3'
    ser = mock_serial
    assert ser.portstr == port
```

task3.ino

```cpp
#include <Arduino.h>
#include <ArduinoJson.h>

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

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

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

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

bool loadStringConfig(JsonDocument& doc, const char* key, String& value) {
  if (doc.containsKey(key) && doc[key].is<String>()) {
    value = doc[key].as<String>();
    return true;
  }
  Serial.println(String(key) + " not found or invalid");
  return false;
}

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

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

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

  if (!loadStringConfig(doc, "player1Symbol", player1Symbol)) return;
  if (!loadStringConfig(doc, "player2Symbol", player2Symbol)) return;

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



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

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

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

  Serial.println(boardState);
}

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

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

  return false;
}

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

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

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

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

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

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

bool blockOpponentMove(char opponent) {
  int row[3] = {0, 1, 2};
  int col[3] = {0, 1, 2};

  // Check rows and columns
  for (int i = 0; i < 3; i++) {
    int rowCoords[3][2] = {{row[i], 0}, {row[i], 1}, {row[i], 2}};
    int colCoords[3][2] = {{0, col[i]}, {1, col[i]}, {2, col[i]}};

    if (canBlock(rowCoords[0], rowCoords[1], rowCoords[2], opponent)) {
      return true;
    }
    if (canBlock(colCoords[0], colCoords[1], colCoords[2], opponent)) {
      return true;
    }
  }

  // Check diagonals
  int diag1[3][2] = {{0, 0}, {1, 1}, {2, 2}};
  int diag2[3][2] = {{0, 2}, {1, 1}, {2, 0}};

  if (canBlock(diag1[0], diag1[1], diag1[2], opponent)) {
    return true;
  }
  if (canBlock(diag2[0], diag2[1], diag2[2], opponent)) {
    return true;
  }

  return false;
}

bool isWinningMove(int coords1[2], int coords2[2], int coords3[2], char
opponent) {
  return (board[coords1[0]][coords1[1]] == opponent &&
board[coords2[0]][coords2[1]] == opponent && board[coords3[0]][coords3[1]] ==
' ') ||
         (board[coords1[0]][coords1[1]] == opponent &&
board[coords2[0]][coords2[1]] == ' ' && board[coords3[0]][coords3[1]] ==
opponent) ||
         (board[coords1[0]][coords1[1]] == ' ' &&
board[coords2[0]][coords2[1]] == opponent && board[coords3[0]][coords3[1]] ==
opponent);
}

void placeMove(int coords[2], char symbol) {
  board[coords[0]][coords[1]] = symbol;
  Serial.println("AI blocked opponent's winning move at: " + String(coords[0]
+ 1) + " " + String(coords[1] + 1));
}
```

```
bool canBlock(int coords1[2], int coords2[2], int coords3[2], char opponent) {
  if (isWinningMove(coords1, coords2, coords3, opponent)) {
    if (board[coords1[0]][coords1[1]] == ' ') {
      placeMove(coords1, 'O');
    } else if (board[coords2[0]][coords2[1]] == ' ') {
      placeMove(coords2, 'O');
    } else {
      placeMove(coords3, 'O');
    }
    return true;
  }

  return false;
}

void handlePlayerMove(int row, int col) {
  board[row][col] = (currentPlayer == "X") ? 'X' : 'O';
  printBoard();

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

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

```
}

void switchPlayer() {
  currentPlayer = (currentPlayer == "X") ? 'O' : 'X';
}

bool isValidMove(int row, int col) {
  return (row >= 0 && row < 3 && col >= 0 && col < 3 && board[row][col] == '
');
}

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

  if (isValidMove(row, col)) {
    handlePlayerMove(row, col);

    if (!gameActive) return;

    handleAIMove();

    if (gameActive) {
      switchPlayer();
    }
  } else {
    Serial.println("Invalid move, try again.");
  }
}


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

void initializeGame() {
  initializeBoard();
  gameActive = true;

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

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

bool checkAndPrintWinner(char symbol) {
  if (checkWin(symbol)) {
    Serial.println(String(symbol) + " wins!");
    gameActive = false;
    return true;
  }
  return false;
}

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

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

      if (checkAndPrintWinner('X') || checkAndPrintDraw()) {
        break;
      }

      currentPlayer = 'O';
    } else {
      aiMove('O');
      printBoard();
      if (checkAndPrintWinner('O') || checkAndPrintDraw()) {
        break;
      }
      currentPlayer = 'X';
    }
  }
}

void processAIvsAI() {
  while (gameActive) {
    aiMove('X');
    printBoard();
    if (checkAndPrintWinner('X') || checkAndPrintDraw()) {
      break;
    }

    aiMove('O');
    printBoard();
    if (checkAndPrintWinner('O') || checkAndPrintDraw()) {
      break;
    }
  }
}

void processReceivedMessage(String receivedMessage) {
  if (receivedMessage == "new") {
    initializeGame();

    if (gameMode == 0) {
      processHumanVsAI();
    } else if (gameMode == 2) {
      processAIvsAI();
    }
  } else if (receivedMessage.startsWith("save")) {
    GameConfig config = { gameMode, player1Symbol, player2Symbol,
currentPlayer };
    saveConfig(config);
  } else if (receivedMessage.startsWith("{")) {
    if (receivedMessage.length() > 0) {
        loadConfig(receivedMessage);
    } else {
        Serial.println("No message received");
    }
  } else if (receivedMessage.startsWith("modes")) {
    handleGameMode(receivedMessage);
  } else if (gameActive) {
    processMove(receivedMessage);
  } else {
    Serial.println("No active game. Type 'new' to start.");
  }
```

```
}

void handleGameMode(String receivedMessage) {
  if (receivedMessage == "modes 0") {
    gameMode = 0;
    Serial.println("Game mode: Man vs AI");
  } else if (receivedMessage == "modes 1") {
    gameMode = 1;
    Serial.println("Game mode: Man vs Man");
  } else if (receivedMessage == "modes 2") {
    gameMode = 2;
    Serial.println("Game mode: AI vs AI");
  }
}

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

2. Реалізував збереження конфігурації в форматі JSON:

```
{"gameMode": 0, "player1Symbol": "X", "player2Symbol": "O"}
```

Висновок

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

Список використаних джерел

1. PlatformIO Documentation. "What is PlatformIO?". https://docs.platformio.org/en.
2. JSON Official Website. "Introducing JSON". https://www.json.org.