МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ЛЬВІВСЬКА ПОЛІТЕХНІКА

Автоматизоване проектування комп’ютерних систем

Task 4. Create doxygen documentation

Виконав:

ст. гр КІ - 401

Савченко В. О.

Прийняв:

Федак П. Р.

2024

Опис теми

Для виконання завдання №4 потрібно виконати наступні задачі:

1. Додати doxygen коментарі для всіх публічних функцій, класів, властивостей, полів...
2. Створити документації на основі коментарів doxygen

Теоретичні відомості

Doxygen — це інструмент для автоматичної генерації документації з вихідного коду програмного забезпечення. Він підтримує різні мови програмування, такі як C++, C, Java, Python, та інші. Doxygen аналізує коментарі в коді та генерує структуровану документацію у вигляді HTML, PDF, чи інших форматів, що спрощує розуміння і підтримку проєкту.

Doxyfile — це конфігураційний файл, який використовується Doxygen для визначення параметрів генерації документації. Він містить налаштування, такі як формат вихідного документа, директорії для сканування, фільтри, опції форматування та інші параметри, які керують процесом створення документації.

Виконання завдання

1. Додав doxygen коментарі для файлів серверної та клієнтської частин, тестів.

main.py

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

CONFIG_FILE = 'config/rps_config.json'


def setup_serial_port():
```

```
    """
    @brief Sets up the serial port for communication.
    @details Prompts the user to enter the serial port (e.g., /dev/ttyUSB0 or
COM3) and
             returns a serial connection object.
    @return Serial connection object.
    @throws serial.SerialException if the serial port cannot be opened.
    """
    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):
    """
    @brief Sends a message over the serial connection.
    @details Encodes the message and sends it via the given serial connection.
    @param message The message to send.
    @param ser The serial connection object.
    @throws serial.SerialException if sending the message fails.
    """
    try:
        ser.write((message + '\n').encode())
    except serial.SerialException as e:
        print(f"Error sending message: {e}")


def receive_message(ser):
    """
    @brief Receives a message from the serial connection.
    @details Reads a line from the serial connection, decodes it, and strips
it of any
             unnecessary whitespace or errors.
    @param ser The serial connection object.
    @return The received message or None if an error occurs.
    @throws serial.SerialException if receiving the message fails.
    """
    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 user_input_thread(ser):
    """
    @brief Handles user input in a separate thread.
    @details Continuously listens for user input. Depending on the input, the
user can send messages
             or save/load game configurations. The thread will exit if the
user types 'exit'.
    @param ser The serial connection object.
    """
    global can_input
    while True:
        if can_input:
            user_message = input("Enter your choice (rock, paper, scissors) or
'exit' to quit: ")
            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):
    """
    @brief Monitors incoming messages on the serial connection in a separate
thread.
    @details Continuously checks for messages from the serial connection and
updates the can_input
             flag when new data is received.
    @param ser The serial connection object.
    """
    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):
    """
    @brief Saves the game configuration to a JSON file.
    @details Saves the game mode configuration to the `rps_config.json` file.
    @param message The message containing the configuration details.
    @throws Exception if saving the configuration fails.
    """
    config = {
        "gameMode": "classic",  # classic or extended
    }

    try:
        params = message.split()
        if len(params) == 2 and params[1] in ['classic', 'extended']:
            config["gameMode"] = 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):
    """
    @brief Loads the game configuration from a JSON file.
    @details Reads the configuration from a file and sends it to the serial
device. If the file is not
             found, prompts the user to provide a valid path.
    @param file_path The path to the configuration file.
    @param ser The serial connection object.
    @throws Exception if loading the configuration fails.
    """
    try:
        if os.path.exists(file_path):
            with open(file_path, 'r') as f:
                config = json.load(f)
                game_mode = config.get("gameMode", "classic")

                print(f"Game Mode: {game_mode}")

                json_message = {
                    "gameMode": game_mode
```

```
                }

                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__":
    """
    @brief Main entry point of the program.
    @details Sets up the serial port and starts two threads: one for
monitoring incoming messages
             and one for handling user input. The program will keep running
until the exit flag is set.
    """
    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__),
'..')))
from main import send_message, receive_message, save_game_config,
load_game_config


def test_send_message():
    """!
    @brief Tests the send_message function.
    @details This test verifies that the send_message function correctly calls
the serial
             port's write method with the expected message in the correct
format (encoded as bytes).
    """
    mock_serial = MagicMock(spec=serial.Serial)
    send_message("Hello", mock_serial)
    mock_serial.write.assert_called_with(b"Hello\n")


def test_receive_message():
    """!
    @brief Tests the receive_message function.
    @details This test simulates receiving a message from the serial
connection and checks
             that the function returns the correct decoded string.
    """
    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():
    """!
    @brief Tests the receive_message function with an empty message.
    @details This test simulates receiving an empty message (just a newline)
and ensures that
             the function returns an empty string.
    """
    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):
    """!
    @brief Tests serial port setup.
    @details This test simulates user input for selecting the serial port and
verifies that
             the serial port configuration is correctly set to the mocked
input value.
    """
    mock_serial = MagicMock(spec=serial.Serial)
    mock_serial.portstr = 'COM3'
    port = 'COM3'
    ser = mock_serial
    assert ser.portstr == port
```

task3.ino

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

CONFIG_FILE = 'config/rps_config.json'


def setup_serial_port():
    """
    @brief Sets up the serial port for communication.
    @details Prompts the user to enter the serial port (e.g., /dev/ttyUSB0 or
COM3) and
             returns a serial connection object.
    @return Serial connection object.
    @throws serial.SerialException if the serial port cannot be opened.
    """
    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):
    """
    @brief Sends a message over the serial connection.
    @details Encodes the message and sends it via the given serial connection.
    @param message The message to send.
    @param ser The serial connection object.
```

```
    @throws serial.SerialException if sending the message fails.
    """
    try:
        ser.write((message + '\n').encode())
    except serial.SerialException as e:
        print(f"Error sending message: {e}")


def receive_message(ser):
    """
    @brief Receives a message from the serial connection.
    @details Reads a line from the serial connection, decodes it, and strips
it of any
             unnecessary whitespace or errors.
    @param ser The serial connection object.
    @return The received message or None if an error occurs.
    @throws serial.SerialException if receiving the message fails.
    """
    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 user_input_thread(ser):
    """
    @brief Handles user input in a separate thread.
    @details Continuously listens for user input. Depending on the input, the
user can send messages
             or save/load game configurations. The thread will exit if the
user types 'exit'.
    @param ser The serial connection object.
    """
    global can_input
    while True:
        if can_input:
            user_message = input("Enter your choice (rock, paper, scissors) or
'exit' to quit: ")
            if user_message.lower() == 'exit':
                print("Exiting...")
                global exit_program
                exit_program = True
                break
```

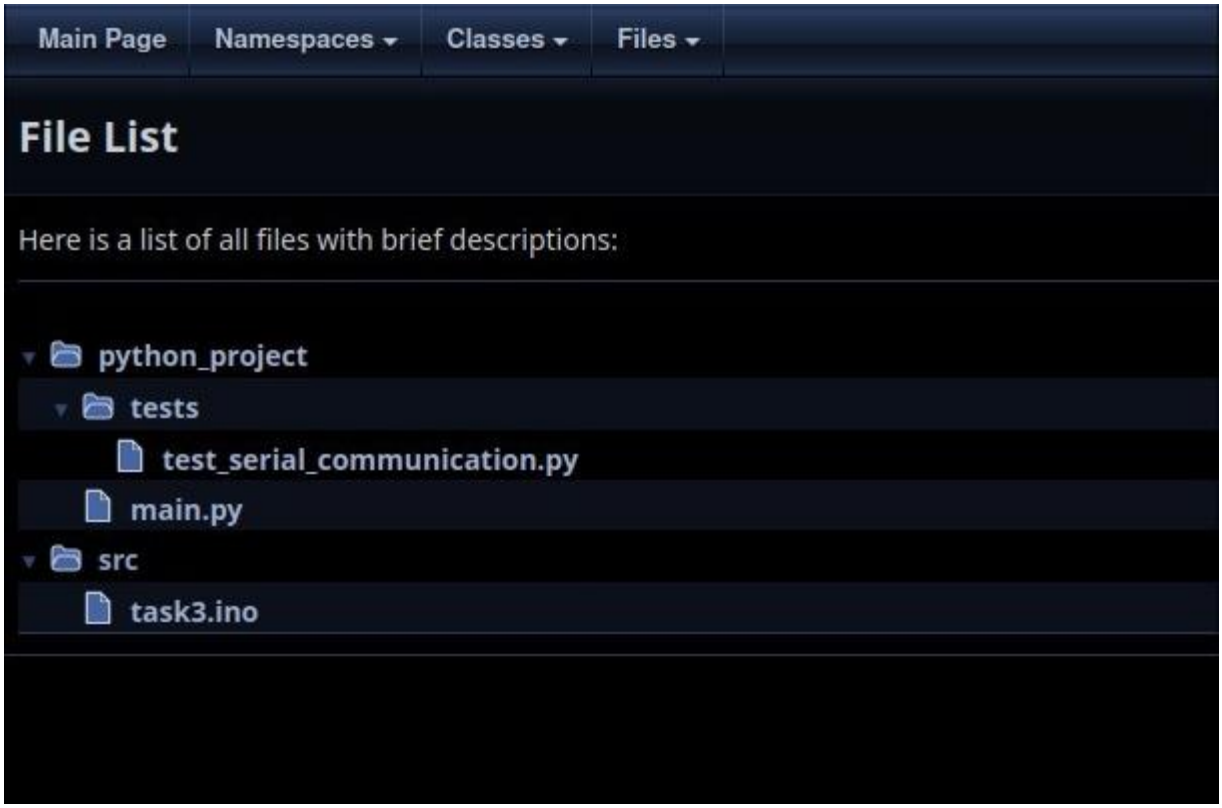
```
            elif user_message.lower().startswith('save'):
                save_game_config(user_message)
            elif user_message.lower().startswith('load'):
                file_path = input("Enter the path to the configuration file:
")
                load_game_config(file_path, ser)
            send_message(user_message, ser)
            can_input = False


def monitor_incoming_messages(ser):
    """
    @brief Monitors incoming messages on the serial connection in a separate
thread.
    @details Continuously checks for messages from the serial connection and
updates the can_input
             flag when new data is received.
    @param ser The serial connection object.
    """
    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):
    """
    @brief Saves the game configuration to a JSON file.
    @details Saves the game mode configuration to the `rps_config.json` file.
    @param message The message containing the configuration details.
    @throws Exception if saving the configuration fails.
    """
    config = {
        "gameMode": "classic",  # classic or extended
    }

    try:
        params = message.split()
        if len(params) == 2 and params[1] in ['classic', 'extended']:
            config["gameMode"] = 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):
    """
    @brief Loads the game configuration from a JSON file.
    @details Reads the configuration from a file and sends it to the serial
device. If the file is not
             found, prompts the user to provide a valid path.
    @param file_path The path to the configuration file.
    @param ser The serial connection object.
    @throws Exception if loading the configuration fails.
    """
    try:
        if os.path.exists(file_path):
            with open(file_path, 'r') as f:
                config = json.load(f)
                game_mode = config.get("gameMode", "classic")

                print(f"Game Mode: {game_mode}")

                json_message = {
                    "gameMode": game_mode
                }

                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}")


def determine_winner(player1_choice, player2_choice):
    """
    @brief Determines the winner of the Rock, Paper, Scissors game.
    @details Compares the choices of player1 and player2 to determine the
winner.
    @param player1_choice The choice of player 1 ('rock', 'paper', or
'scissors').
    @param player2_choice The choice of player 2 ('rock', 'paper', or
'scissors').
```

```
    @return A string indicating the result ('Player 1 wins', 'Player 2 wins',
or 'It's a draw').
    """
    if player1_choice == player2_choice:
        return "It's a draw!"
    elif (player1_choice == 'rock' and player2_choice == 'scissors') or \
         (player1_choice == 'scissors' and player2_choice == 'paper') or \
         (player1_choice == 'paper' and player2_choice == 'rock'):
        return "Player 1 wins!"
    else:
        return "Player 2 wins!"


if __name__ == "__main__":
    """
    @brief Main entry point of the program.
    @details Sets up the serial port and starts two threads: one for
monitoring incoming messages
             and one for handling user input. The program will keep running
until the exit flag is set.
    """
    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()
```

2. Згенерувати конфігураційни файл **Doxyfile** та вніс необхідні параметри.

3. Відкрив **index.html**:



Висновок

Під час виконання завдання №4 було згенеровано doxygen документацію.

Список використаних джерел

1. Doxygen Manual. "Introduction to Doxygen". https://doxygen.nl/manual/index.html.
2. Doxygen Manual. "Configuration (Doxyfile)". https://doxygen.nl/manual/config.html.