

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"



**АВТОМАТИЗОВАНЕ ПРОЕКТУВАННЯ КОМП'ЮТЕРНИХ
СИСТЕМ**

Лабораторна робота 3

Виконав:
студент групи КІ-401
Шаклеїн О.Т.
Прийняв:
Шпіцер А.С

Львів 2024

Тема: Імплементування Ігри до клієнт серверу

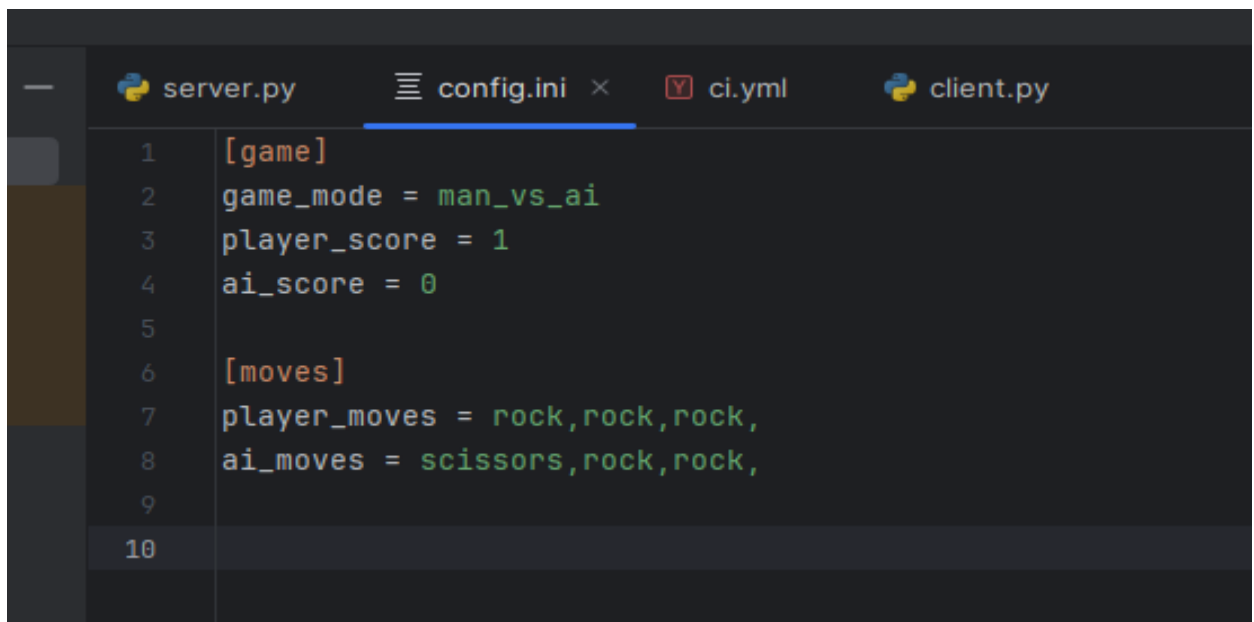
Порядок виконання лабораторної роботи:

1. Develop Server and Client.
2. Required steps.

Виконання роботи

Для реалізації гри "Камінь-Ножиці-Папір" було створено серверну частину, яка виконується на симульованому апаратному середовищі, та клієнтську частину, що працює на програмному рівні.

1. Відповідно до варіанту завдання, для збереження конфігурації гри та її поточного стану використовувався формат INI. Це дозволило зберігати поточний режим гри, кількість балів гравця та AI, а також історію ходів. Для роботи з INI-файлами було застосовано бібліотеку Python `configparser`, що забезпечило зручний доступ до даних і їх збереження у файл `config.ini`.



```
1 [game]
2 game_mode = man_vs_ai
3 player_score = 1
4 ai_score = 0
5
6 [moves]
7 player_moves = rock,rock,rock,
8 ai_moves = scissors,rock,rock,
9
10
```

Рис. 1. config.ini

2. **Розробка серверної частини:** Серверна частина, що виконується на порту COM11, відповідає за обробку запитів від клієнта, таких як "новий хід", "збереження" та "завантаження". Сервер отримує запити від клієнта,

обробляє їх та генерує відповідь, яка включає інформацію про вибір серверу (ход AI), результат гри та оновлені рахунки. Залежно від отриманих команд, сервер:

- Створює нову гру та обнуляє рахунки.
- Зберігає поточний стан гри в INI-файл.
- Завантажує стан гри з INI-файлу.
- Виконує гру в режимі "гравець проти AI", де AI вибирає випадковий хід. Зокрема, реалізовано логіку обчислення переможця на основі вибору гравця та AI, де визначається результат ("гравець перемагає", "сервер перемагає" або "нічия").

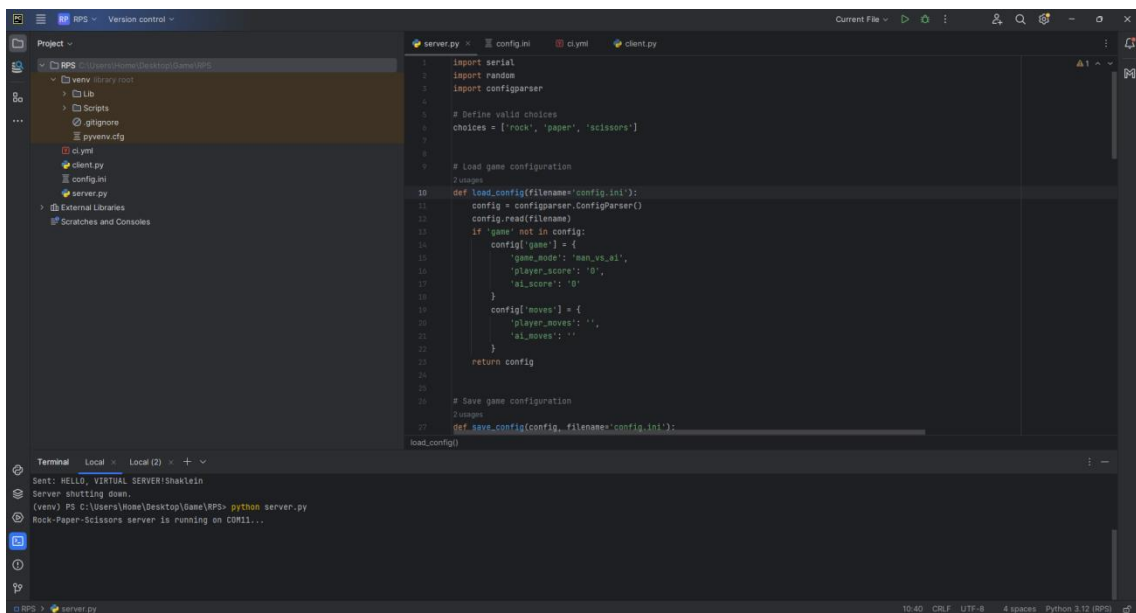


Рис. 2. Серверна частина

3. Розробка клієнтської частини: Клієнтська частина, що виконується на порту COM12, забезпечує текстовий інтерфейс для взаємодії з користувачем. Клієнт надає користувачеві меню, в якому можна обрати наступні дії:

- Розпочати нову гру.
- Зберегти поточний стан гри.
- Завантажити збережений стан гри.
- Зробити хід у грі (вибір "камінь", "ножиці" або "папір"). Після

вибору користувачем однієї з дій, клієнт надсилає відповідну команду на сервер і отримує відповідь, яка відображається на екрані.

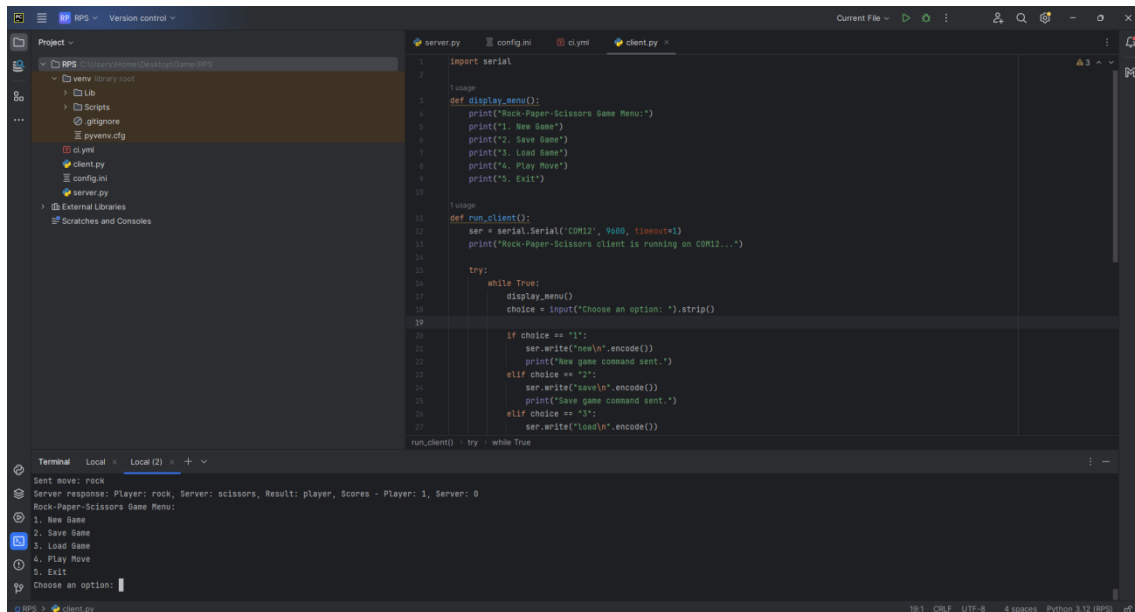


Рис. 3. Клієнтська частина

4. **Збереження та завантаження стану гри:** Використовуючи INI-файл config.ini, сервер зберігає та відновлює конфігурацію гри. Це дозволяє користувачеві зберігати прогрес гри та продовжувати її з того місця, де він зупинився. Наприклад, якщо гравець вийде з гри, він може завантажити попередній стан і продовжити гру з попередніми рахунками та історією ходів.
5. **Тестування гри:** Проведено тестування роботи клієнт-серверної взаємодії, включаючи перевірку команд "новий хід", "збереження" та "завантаження". Клієнт успішно надсилає запити на сервер, який обробляє їх та повертає правильні відповіді. Переконався, що збереження та завантаження стану гри відбувається коректно, а сервер правильно обробляє результати гри та оновлює рахунки.
6. **Інтерфейс AI vs AI:** Додатково було реалізовано режим гри "AI проти AI", в якому сервер і клієнт автоматично роблять випадкові ходи. Це дозволило додатково перевірити логіку гри без втручання користувача.

Було проведено оновлення файлу конфігурації YAML для автоматизації процесу тестування гри "Камінь-Ножиці-Папір" у середовищі **GitHub Actions**.

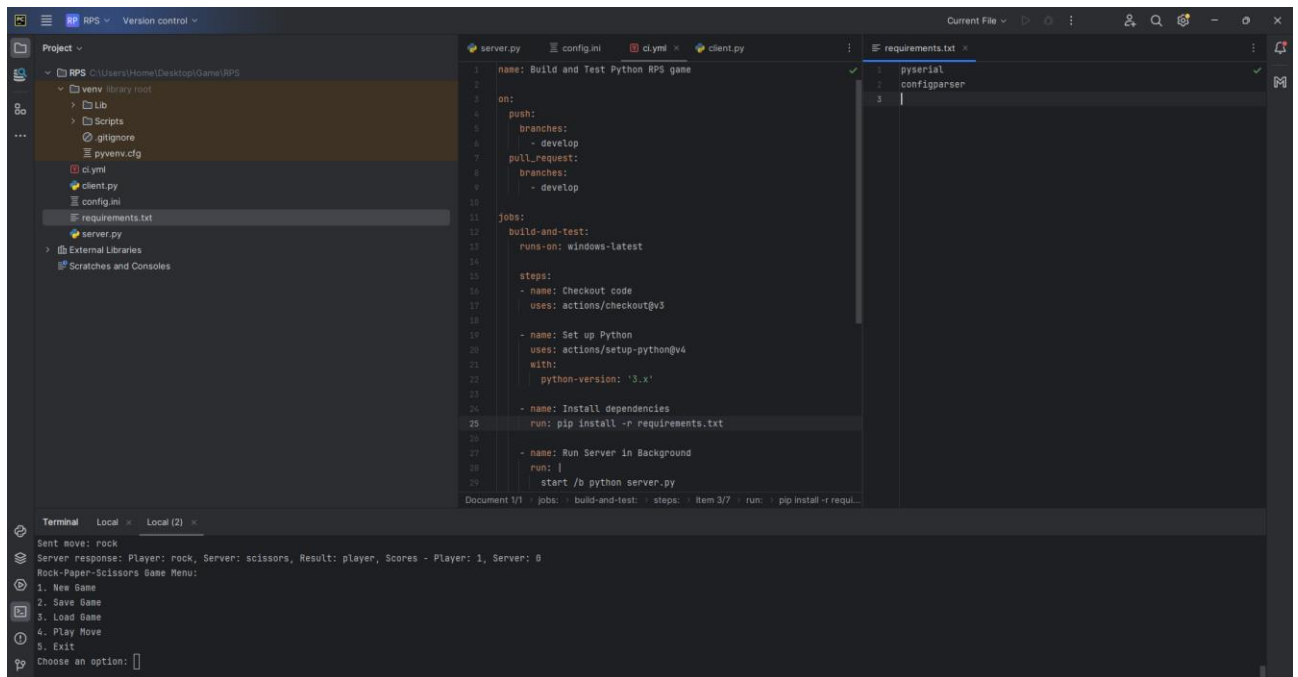


Рис. 4. YAML та requirments.txt

Додатки

server.py

```
import serial
import random
import configparser

# Define valid choices
choices = ['rock', 'paper', 'scissors']

# Load game configuration
def load_config(filename='config.ini'):
    config = configparser.ConfigParser()
    config.read(filename)
    if 'game' not in config:
        config['game'] = {
            'game_mode': 'man_vs_ai',
            'player_score': '0',
            'ai_score': '0'
        }
    config['moves'] = {
        'player_moves': '',
        'ai_moves': ''
    }
    return config

# Save game configuration
def save_config(config, filename='config.ini'):
    with open(filename, 'w') as configfile:
        config.write(configfile)
```

```

def get_winner(player_choice, server_choice):
    if player_choice == server_choice:
        return "draw"
    elif (player_choice == "rock" and server_choice == "scissors") or \
        (player_choice == "scissors" and server_choice == "paper") or \
        (player_choice == "paper" and server_choice == "rock"):
        return "player"
    else:
        return "server"

def run_server():
    ser = serial.Serial('COM11', 9600, timeout=1)
    print("Rock-Paper-Scissors server is running on COM11...")

    config = load_config()

    try:
        while True:
            if ser.in_waiting > 0:
                command = ser.readline().decode().strip().lower()

                if command == "new":
                    config['game'] = {
                        'game_mode': 'man_vs_ai',
                        'player_score': '0',
                        'ai_score': '0'
                    }
                    config['moves'] = {
                        'player_moves': '',
                        'ai_moves': ''
                    }
                    ser.write("New game started.\n".encode())

                elif command == "save":
                    save_config(config)
                    ser.write("Game saved.\n".encode())

                elif command == "load":
                    config = load_config()
                    ser.write("Game loaded.\n".encode())

                elif command.startswith("play"):
                    _, player_choice = command.split()

                    if player_choice not in choices:
                        ser.write("Invalid choice. Choose rock, paper, or scissors.\n".encode())
                        continue

                    # AI makes a choice (random)
                    server_choice = random.choice(choices)

                    # Determine the winner
                    winner = get_winner(player_choice, server_choice)
                    if winner == "player":
                        config['game']['player_score'] = str(int(config['game']['player_score']) + 1)
                    elif winner == "server":
                        config['game']['ai_score'] = str(int(config['game']['ai_score']) + 1)

                    # Update moves in the config
                    player_moves = config['moves'].get('player_moves', '')
                    ai_moves = config['moves'].get('ai_moves', '')
                    config['moves']['player_moves'] = player_moves + player_choice + ','
                    config['moves']['ai_moves'] = ai_moves + server_choice + ','

```

```

        # Prepare the response message
        response = f"Player: {player_choice}, Server: {server_choice}, Result: {winner}, Scores - Player: {config['game']['player_score']}, Server: {config['game']['ai_score']}"
        ser.write(f"{response}\n".encode())

except KeyboardInterrupt:
    print("Server shutting down.")
finally:
    ser.close()
    save_config(config)

if __name__ == "__main__":
    run_server()

```

client.py

```

import serial

def display_menu():
    print("Rock-Paper-Scissors Game Menu:")
    print("1. New Game")
    print("2. Save Game")
    print("3. Load Game")
    print("4. Play Move")
    print("5. Exit")

def run_client():
    ser = serial.Serial('COM12', 9600, timeout=1)
    print("Rock-Paper-Scissors client is running on COM12...")

    try:
        while True:
            display_menu()
            choice = input("Choose an option: ").strip()

            if choice == "1":
                ser.write("new\n".encode())
                print("New game command sent.")
            elif choice == "2":
                ser.write("save\n".encode())
                print("Save game command sent.")
            elif choice == "3":
                ser.write("load\n".encode())
                print("Load game command sent.")
            elif choice == "4":
                player_choice = input("Enter your choice (rock, paper, or scissors): ").strip().lower()
                ser.write(f"play {player_choice}\n".encode())
                print(f"Sent move: {player_choice}")
            elif choice == "5":
                print("Exiting game.")
                break

            # Read the response from the server
            response = ser.readline().decode().strip()
            print(f"Server response: {response}")

    except KeyboardInterrupt:
        print("Client shutting down.")
    finally:
        ser.close()

```

```
if __name__ == "__main__":  
    run_client()
```

ci.yml

name: Build and Test Python RPS game

on:

push:

branches:

- develop

pull_request:

branches:

- develop

jobs:

build-and-test:

runs-on: windows-latest

steps:

- name: Checkout code

uses: actions/checkout@v3

- name: Set up Python

uses: actions/setup-python@v4

with:

python-version: '3.x'

- name: Install dependencies

run: pip install -r requirements.txt

- name: Run Server in Background

run: |

start /b python server.py

shell: cmd

- name: Run Client Test

run: python client.py > test_output.txt

- name: Save Test Output

uses: actions/upload-artifact@v3

with:

name: test-results

path: test_output.txt

- name: Clean up

run: taskkill /IM python.exe /F

shell: cmd