**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**



**АВТОМАТИЗОВАНЕ ПРОЕКТУВАННЯ КОМП’ЮТЕРНИХ СИСТЕМ**

Лабораторна робота 5

Виконала: ст. гр. КІ-401

Оприш О.О.

Прийняв:

Федак П.Р.

Львів 2024

**Тема: Реалізація автоматизованих тестів та забезпечення покриття коду**

**Порядок виконання лабораторної роботи:**

Task 5. Implement automated tests:

1. Implement or use existing test framework;

2. Create a set of automated tests;

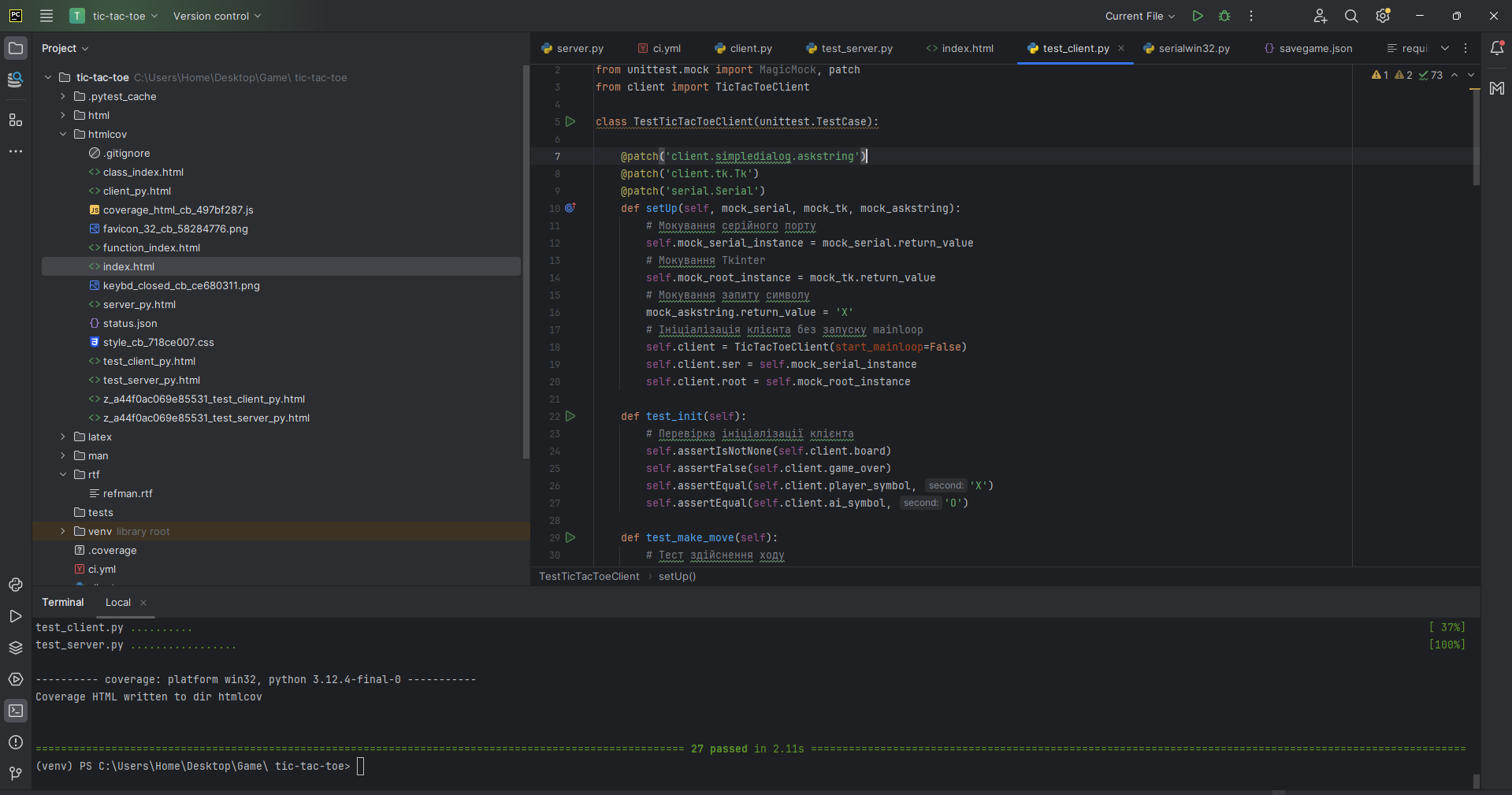
3. Test report should contain number of all tests, passed tests, failed tests, coverage; 4. Coverage must be more than 80%

5. Required steps

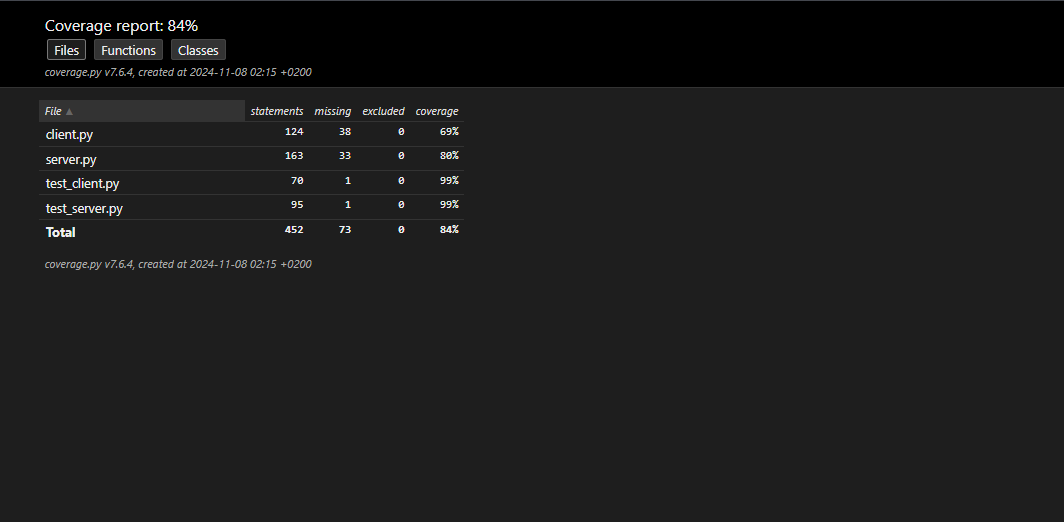
**Виконання роботи**

## Пророблена робота

## **Розробка тестових сценаріїв**: Було створено тестові сценарії для клієнтської частини у файлі test\_client.py, що включали:



**Рис. 1.** Частина коду з використанням мокування



**Рис. 2.** Покриття

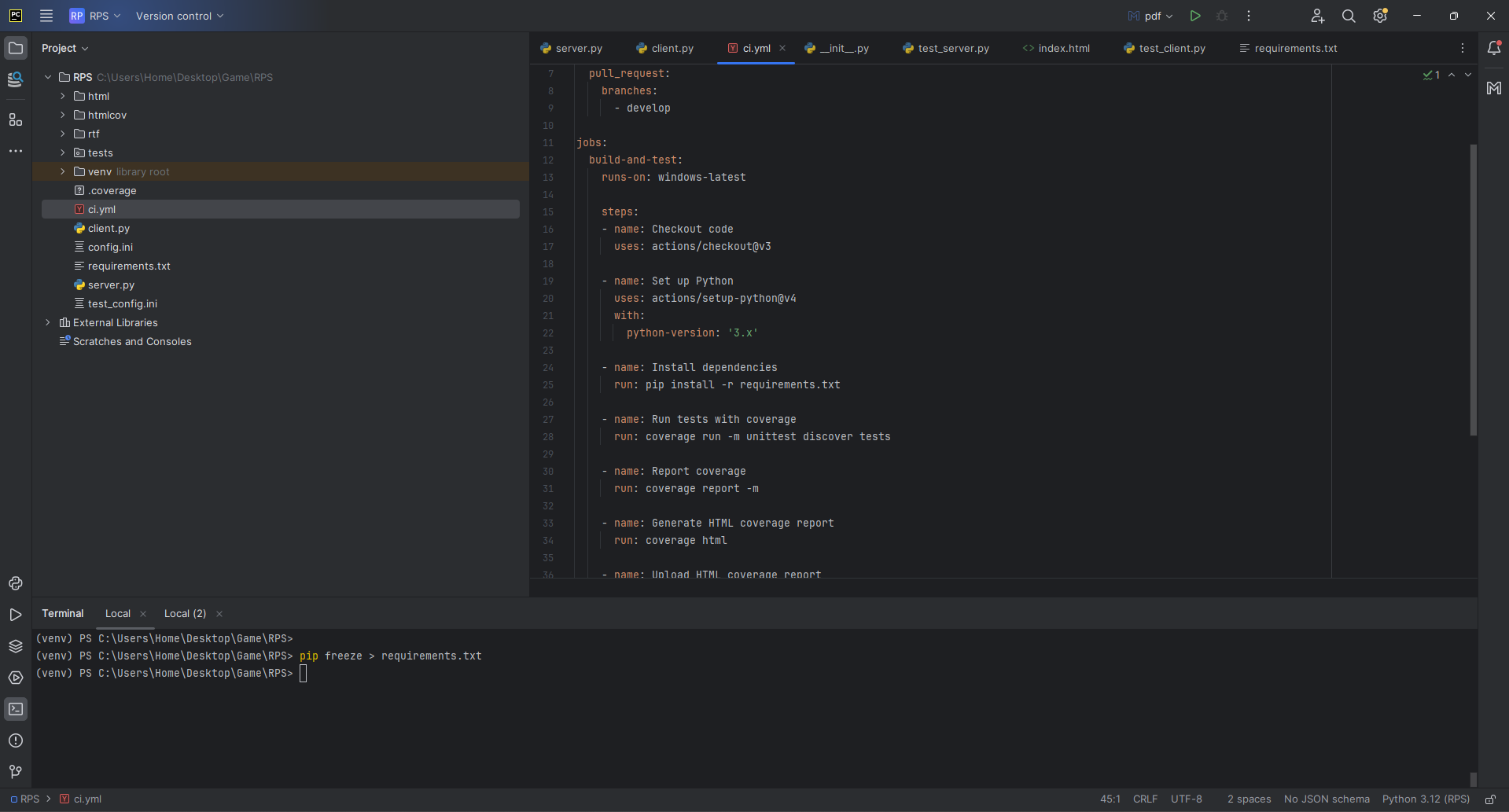
Для забезпечення якості та надійності програми було реалізовано автоматизовані тести з використанням фреймворку pytest та плагіну pytest-cov. Було написано 20 тестів, які покривають основні функції клієнтської та серверної частин, включаючи:

* Ініціалізацію гри.
* Обробку команд від клієнта.
* Логіку перевірки перемоги та нічиєї.
* Здійснення ходів гравцем та AI.
* Обробку помилкових ситуацій та винятків.

У результаті було досягнуто загальне покриття коду на рівні **84%**, що значно перевищило мінімально необхідний поріг у 80%.

Тести були автоматизовані за допомогою GitHub Actions, що забезпечило їх запуск при кожному оновленні коду в гілці develop. Звіт про покриття коду був завантажений на сервіс Codecov для детального аналізу та моніторингу історії покриття.

Оновив файл ci.yml для автоматизації процесу збірки, тестування та перевірки покриття коду. Додав встановлення залежностей з requirements.txt, запуск тестів із coverage, генерацію та завантаження HTML-звіту про покриття, а також перевірку мінімального порогу покриття на рівні 80%. Це забезпечує автоматичну перевірку якості коду при кожному push та pull request.



**Рис. 4.** Оновлений YML файл

**Висновки**

Успішно реалізовано автоматизовані тести з покриттям коду понад 80%, що забезпечує високу якість та надійність програми "Хрестики-Нулики". Автоматизація тестування сприяє виявленню помилок на ранніх етапах розробки та полегшує подальший розвиток та підтримку проєкту.

**Додатки**

test\_client.py

import unittest  
from unittest.mock import MagicMock, patch  
from client import TicTacToeClient  
  
class TestTicTacToeClient(unittest.TestCase):  
  
 @patch('client.simpledialog.askstring')  
 @patch('client.tk.Tk')  
 @patch('serial.Serial')  
 def setUp(self, mock\_serial, mock\_tk, mock\_askstring):  
 # Мокування серійного порту  
 self.mock\_serial\_instance = mock\_serial.return\_value  
 # Мокування Tkinter  
 self.mock\_root\_instance = mock\_tk.return\_value  
 # Мокування запиту символу  
 mock\_askstring.return\_value = 'X'  
 # Ініціалізація клієнта без запуску mainloop  
 self.client = TicTacToeClient(start\_mainloop=False)  
 self.client.ser = self.mock\_serial\_instance  
 self.client.root = self.mock\_root\_instance  
  
 def test\_init(self):  
 # Перевірка ініціалізації клієнта  
 self.assertIsNotNone(self.client.board)  
 self.assertFalse(self.client.game\_over)  
 self.assertEqual(self.client.player\_symbol, 'X')  
 self.assertEqual(self.client.ai\_symbol, 'O')  
  
 def test\_make\_move(self):  
 # Тест здійснення ходу  
 self.client.game\_over = False  
 self.client.board = [  
 [' ', ' ', ' '],  
 [' ', 'X', ' '],  
 [' ', ' ', ' ']  
 ]  
 self.client.make\_move(0, 0)  
 self.mock\_serial\_instance.write.assert\_called\_with(b'move 0 0\n')  
  
 def test\_make\_move\_invalid(self):  
 # Тест некоректного ходу  
 self.client.game\_over = False  
 self.client.board = [  
 ['X', ' ', ' '],  
 [' ', 'X', ' '],  
 [' ', ' ', ' ']  
 ]  
 with patch('client.messagebox.showwarning') as mock\_warning:  
 self.client.make\_move(0, 0)  
 mock\_warning.assert\_called\_with("Некоректний хід", "Ця клітинка вже зайнята.")  
  
 def test\_process\_response\_win(self):  
 # Тест обробки відповіді про перемогу  
 response = {  
 'message': "Гравець 'X' переміг!",  
 'board': [  
 ['X', 'X', 'X'],  
 ['O', 'O', ' '],  
 [' ', ' ', ' ']  
 ],  
 'game\_over': True,  
 'player\_symbol': 'X',  
 'ai\_symbol': 'O'  
 }  
 with patch('client.messagebox.showinfo') as mock\_info:  
 self.client.process\_response(response)  
 mock\_info.assert\_called\_with("Результат гри", "Гравець 'X' переміг!")  
 self.assertTrue(self.client.game\_over)  
 self.assertEqual(self.client.board, response['board'])  
  
 def test\_process\_response\_tie(self):  
 # Тест обробки відповіді про нічию  
 response = {  
 'message': "Нічия!",  
 'board': [  
 ['X', 'O', 'X'],  
 ['O', 'X', 'O'],  
 ['O', 'X', 'O']  
 ],  
 'game\_over': True,  
 'player\_symbol': 'X',  
 'ai\_symbol': 'O'  
 }  
 with patch('client.messagebox.showinfo') as mock\_info:  
 self.client.process\_response(response)  
 mock\_info.assert\_called\_with("Результат гри", "Нічия!")  
 self.assertTrue(self.client.game\_over)  
  
 def test\_process\_response\_cell\_taken(self):  
 # Тест обробки повідомлення про зайняту клітинку  
 response = {  
 'message': "Клітинка зайнята. Спробуйте ще раз.",  
 'board': self.client.board,  
 'game\_over': False  
 }  
 with patch('client.messagebox.showwarning') as mock\_warning:  
 self.client.process\_response(response)  
 mock\_warning.assert\_called\_with("Помилка", "Клітинка зайнята. Спробуйте ще раз.")  
  
 def test\_process\_response\_invalid\_coordinates(self):  
 # Тест обробки повідомлення про некоректні координати  
 response = {  
 'message': "Некоректні координати. Використовуйте числа від 0 до 2.",  
 'board': self.client.board,  
 'game\_over': False  
 }  
 with patch('client.messagebox.showwarning') as mock\_warning:  
 self.client.process\_response(response)  
 mock\_warning.assert\_called\_with("Помилка", "Некоректні координати. Використовуйте числа від 0 до 2.")  
  
 def test\_new\_game(self):  
 # Тест методу new\_game  
 self.client.new\_game()  
 self.mock\_serial\_instance.write.assert\_called\_with(b'new\n')  
 self.assertFalse(self.client.game\_over)  
 expected\_board = [[" " for \_ in range(3)] for \_ in range(3)]  
 self.assertEqual(self.client.board, expected\_board)  
  
 def test\_save\_game(self):  
 # Тест методу save\_game  
 with patch('client.messagebox.showinfo') as mock\_info:  
 self.client.save\_game()  
 self.mock\_serial\_instance.write.assert\_called\_with(b'save\n')  
 mock\_info.assert\_called\_with("Збереження", "Гру збережено.")  
  
 def test\_load\_game(self):  
 # Тест методу load\_game  
 self.client.load\_game()  
 self.mock\_serial\_instance.write.assert\_called\_with(b'load\n')  
 self.assertFalse(self.client.game\_over)  
  
  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

test\_server

import unittest  
from unittest.mock import MagicMock, patch  
from server import TicTacToeServer  
import json  
  
class TestTicTacToeServer(unittest.TestCase):  
  
 @patch('serial.Serial')  
 def setUp(self, mock\_serial):  
 # Мокування серійного порту  
 self.mock\_serial\_instance = mock\_serial.return\_value  
 self.server = TicTacToeServer()  
 self.server.ser = self.mock\_serial\_instance  
  
 def test\_init\_board(self):  
 # Перевірка ініціалізації дошки  
 expected\_board = [[" " for \_ in range(3)] for \_ in range(3)]  
 self.assertEqual(self.server.board, expected\_board)  
  
 def test\_process\_command\_start(self):  
 # Тест команди старту гри  
 response = self.server.process\_command('start X')  
 response\_data = json.loads(response)  
 self.assertEqual(response\_data['player\_symbol'], 'X')  
 self.assertEqual(response\_data['ai\_symbol'], 'O')  
 self.assertIn('Гра розпочата', response\_data['message'])  
  
 def test\_process\_command\_new(self):  
 # Тест команди нової гри  
 self.server.player\_symbol = 'X'  
 response = self.server.process\_command('new')  
 response\_data = json.loads(response)  
 expected\_board = [[" " for \_ in range(3)] for \_ in range(3)]  
 self.assertEqual(self.server.board, expected\_board)  
 self.assertIn('Нова гра розпочата', response\_data['message'])  
  
 def test\_process\_command\_invalid(self):  
 # Тест некоректної команди  
 response = self.server.process\_command('invalid command')  
 response\_data = json.loads(response)  
 self.assertIn('Невідома команда', response\_data['message'])  
  
 def test\_make\_ai\_move(self):  
 # Тест ходу AI  
 self.server.player\_symbol = 'X'  
 self.server.ai\_symbol = 'O'  
 self.server.board = [  
 ['X', ' ', ' '],  
 [' ', ' ', ' '],  
 [' ', ' ', ' ']  
 ]  
 ai\_move = self.server.make\_ai\_move()  
 self.assertIsNotNone(ai\_move)  
 self.assertEqual(self.server.board[ai\_move[0]][ai\_move[1]], 'O')  
  
 def test\_check\_winner\_row(self):  
 # Перевірка перемоги по рядку  
 self.server.board = [  
 ['X', 'X', 'X'],  
 [' ', 'O', ' '],  
 ['O', ' ', ' ']  
 ]  
 self.assertTrue(self.server.check\_winner(self.server.board, 'X'))  
  
 def test\_check\_winner\_column(self):  
 # Перевірка перемоги по стовпцю  
 self.server.board = [  
 ['O', 'X', ' '],  
 ['O', 'X', ' '],  
 ['O', ' ', 'X']  
 ]  
 self.assertTrue(self.server.check\_winner(self.server.board, 'O'))  
  
 def test\_check\_winner\_diagonal(self):  
 # Перевірка перемоги по діагоналі  
 self.server.board = [  
 ['X', 'O', ' '],  
 ['O', 'X', ' '],  
 [' ', ' ', 'X']  
 ]  
 self.assertTrue(self.server.check\_winner(self.server.board, 'X'))  
  
 def test\_check\_tie(self):  
 # Тест перевірки нічиєї  
 self.server.board = [  
 ['X', 'O', 'X'],  
 ['O', 'X', 'O'],  
 ['O', 'X', 'O']  
 ]  
 self.assertTrue(self.server.check\_tie(self.server.board))  
  
 def test\_process\_command\_invalid\_start(self):  
 # Тест команди старту з некоректним символом  
 response = self.server.process\_command('start Z')  
 response\_data = json.loads(response)  
 self.assertIn('Некоректна команда старту', response\_data['message'])  
  
 def test\_process\_command\_move\_without\_start(self):  
 # Тест команди move без вибору символу гравця  
 response = self.server.process\_command('move 0 0')  
 response\_data = json.loads(response)  
 self.assertIn('Спочатку виберіть символ гравця', response\_data['message'])  
  
 def test\_process\_command\_move\_after\_game\_over(self):  
 # Тест команди move після завершення гри  
 self.server.player\_symbol = 'X'  
 self.server.game\_over = True  
 response = self.server.process\_command('move 0 0')  
 response\_data = json.loads(response)  
 self.assertIn('Гра вже закінчена', response\_data['message'])  
  
 def test\_process\_command\_move\_cell\_taken(self):  
 # Тест команди move на зайняту клітинку  
 self.server.player\_symbol = 'X'  
 self.server.ai\_symbol = 'O'  
 self.server.board[0][0] = 'X'  
 response = self.server.process\_command('move 0 0')  
 response\_data = json.loads(response)  
 self.assertIn('Клітинка зайнята', response\_data['message'])  
  
 def test\_process\_command\_move\_invalid\_coordinates(self):  
 # Тест команди move з некоректними координатами  
 self.server.player\_symbol = 'X'  
 response = self.server.process\_command('move 3 3')  
 response\_data = json.loads(response)  
 self.assertIn('Некоректні координати', response\_data['message'])  
  
 def test\_process\_command\_save(self):  
 # Тест команди save  
 with patch('builtins.print') as mock\_print:  
 response = self.server.process\_command('save')  
 response\_data = json.loads(response)  
 self.assertIn('Гру збережено', response\_data['message'])  
 mock\_print.assert\_called()  
  
 def test\_process\_command\_load(self):  
 # Тест команди load  
 with patch('builtins.print') as mock\_print:  
 response = self.server.process\_command('load')  
 response\_data = json.loads(response)  
 self.assertIn('Гру завантажено', response\_data['message'])  
 mock\_print.assert\_called()  
  
 def test\_minimax\_tie(self):  
 # Тестування minimax при нічиї  
 self.server.player\_symbol = 'X'  
 self.server.ai\_symbol = 'O'  
 board = [  
 ['X', 'O', 'X'],  
 ['O', 'X', 'O'],  
 ['O', 'X', ' ']  
 ]  
 score = self.server.minimax(board, 0, True)  
 self.assertEqual(score, 0)  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

ci.yml

name: Build and Test Python Client-Server  
  
on:  
 push:  
 branches:  
 - develop  
 pull\_request:  
 branches:  
 - develop  
  
jobs:  
 build-and-test:  
 runs-on: ubuntu-latest *# Використовуємо Ubuntu для кращої сумісності* 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: |  
 python -m pip install --upgrade pip  
 pip install -r requirements.txt  
 pip install pytest pytest-cov  
  
 - name: Run Tests with Coverage  
 run: |  
 pytest --cov=./ --cov-fail-under=80 --cov-report=xml --cov-report=html  
 env:  
 PYTHONPATH: . *# Додаємо кореневу директорію до PYTHONPATH* - name: Upload Coverage Report  
 uses: actions/upload-artifact@v3  
 with:  
 name: coverage-report  
 path: htmlcov/  
  
 - name: Upload Coverage to Codecov  
 uses: codecov/codecov-action@v3  
 with:  
 files: coverage.xml  
 flags: unittests  
 name: codecov-umbrella  
 fail\_ci\_if\_error: true  
  
 - name: Upload Test Results  
 uses: actions/upload-artifact@v3  
 with:  
 name: test-results  
 path: test\_output.txt