**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**


# АВТОМАТИЗОВАНЕ ПРОЕКТУВАННЯ КОМП'ЮТЕРНИХ СИСТЕМ

Лабораторна робота 5

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

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**

## Проробленa робота

Розробка тестових сценаріїв: Було створено тестові сценарії для клієнтської частини у файлі tests/test_client.py, що включали:

- Тестування вибору пунктів меню, таких як "Нова гра", "Грати хід" та "Вихід".
- Використання unittest.mock для імітації введення користувача за допомогою декоратора @patch та імітації серійного зв’язку між клієнтом та сервером.
- Тестування обробки введення користувача, включаючи обробку некоректного вибору та взаємодії з сервером через серійний інтерфейс.

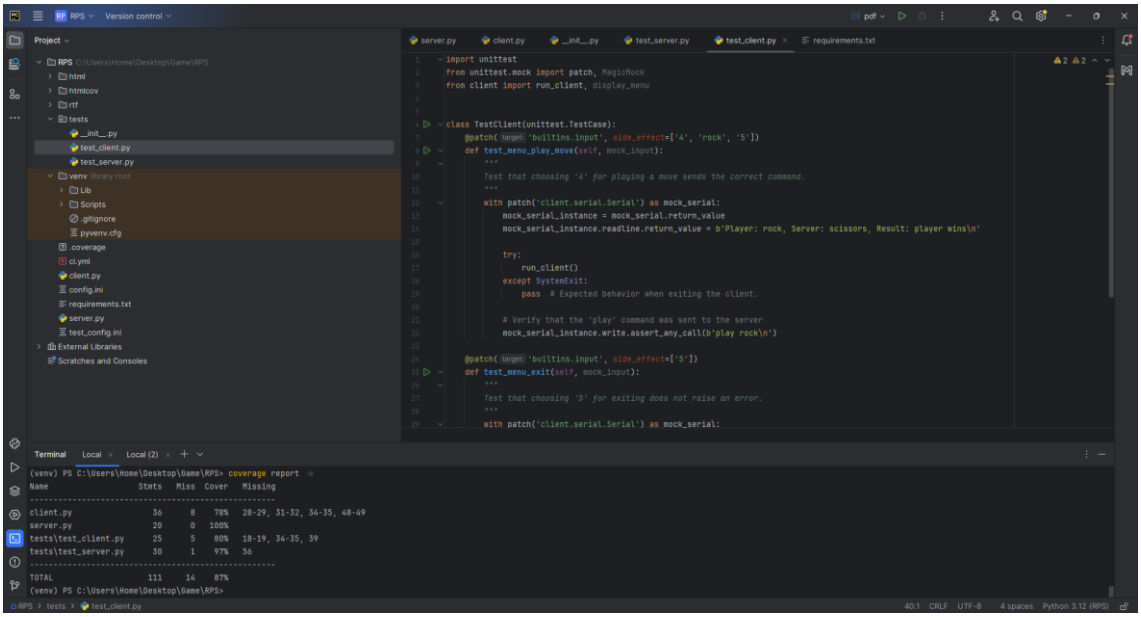


**Рис. 1.** Частина коду з використанням @patch для імітації введення користувача та серійного з'єднання.

**Виконання тестів та генерація звіту покриття**: Використано `coverage.py` для запуску тестів та оцінки покриття коду:


**Рис. 2.** Покриття

У результаті було досягнуто загальне покриття коду на рівні **87%**, що значно перевищило мінімально необхідний поріг у 80%.

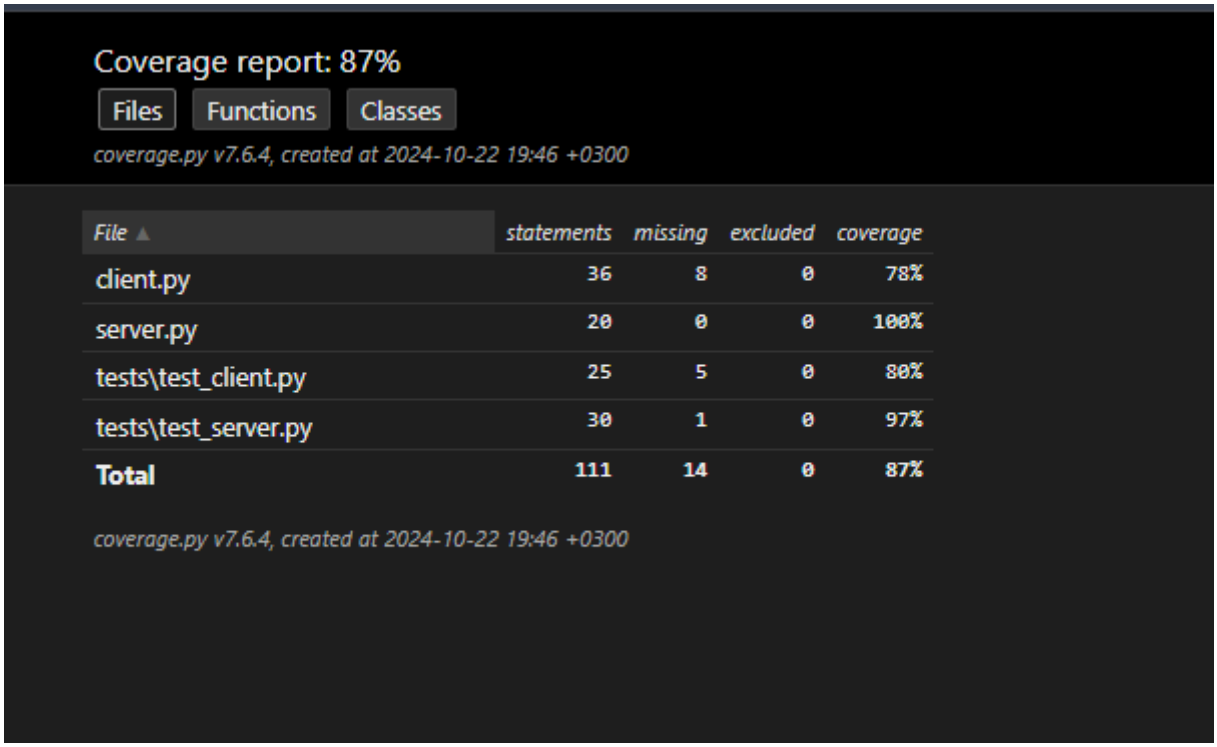


**Рис. 3.** Covarage report

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.

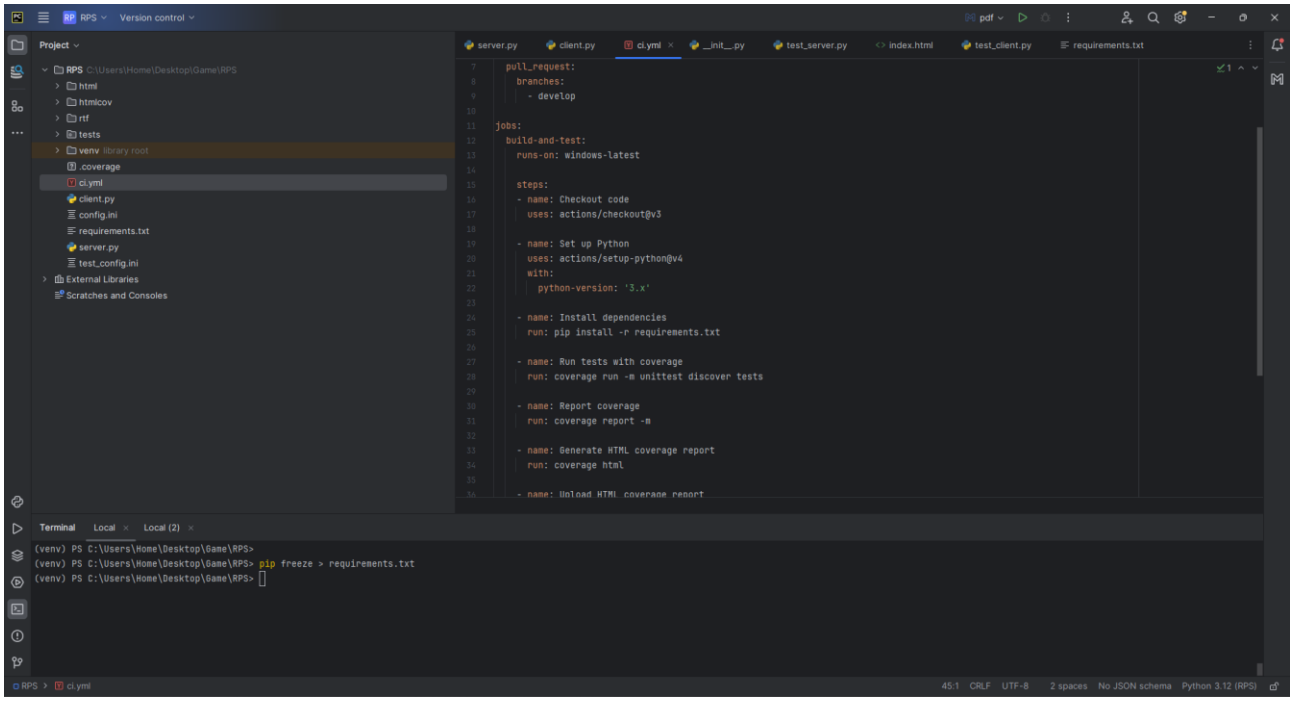


**Рис. 4.** Оновлений YML файл

## Висновки

У ході виконання роботи було реалізовано набір тестів для основної логіки гри та взаємодії з користувачем, що забезпечило рівень покриття коду понад 80%. Використання unittest у поєднанні з coverage.py дозволило забезпечити високий рівень перевірки коду та виявити області, що потребували додаткового тестування.

## Додатки

### *test_client.py*

```
import unittest
from unittest.mock import patch, MagicMock
from client import run_client, display_menu


class TestClient(unittest.TestCase):
    @patch('builtins.input', side_effect=['4', 'rock', '5'])
    def test_menu_play_move(self, mock_input):
        """
        Test that choosing '4' for playing a move sends the correct command.
        """
        with patch('client.serial.Serial') as mock_serial:
            mock_serial_instance = mock_serial.return_value
            mock_serial_instance.readline.return_value = b'Player: rock, Server: scissors, Result: player wins\n'

            try:
                run_client()
            except SystemExit:
                pass  # Expected behavior when exiting the client.

            # Verify that the 'play' command was sent to the server
            mock_serial_instance.write.assert_any_call(b'play rock\n')

    @patch('builtins.input', side_effect=['5'])
    def test_menu_exit(self, mock_input):
        """
        Test that choosing '5' for exiting does not raise an error.
        """
        with patch('client.serial.Serial') as mock_serial:
            mock_serial_instance = mock_serial.return_value
            mock_serial_instance.readline.return_value = b'Exiting game.\n'
            try:
                run_client()
            except SystemExit:
                pass


if __name__ == '__main__':
    unittest.main()
```

### *test_server.py*

```
import unittest
from server import get_winner, load_config, save_config
import configparser

class TestGameLogic(unittest.TestCase):
```

```
    def test_get_winner_draw(self):
        """
        Test that a draw is correctly identified.
        """
        self.assertEqual(get_winner('rock', 'rock'), 'draw')
        self.assertEqual(get_winner('paper', 'paper'), 'draw')
        self.assertEqual(get_winner('scissors', 'scissors'), 'draw')

    def test_get_winner_player_wins(self):
        """
        Test cases where the player wins.
        """
        self.assertEqual(get_winner('rock', 'scissors'), 'player')
        self.assertEqual(get_winner('scissors', 'paper'), 'player')
        self.assertEqual(get_winner('paper', 'rock'), 'player')

    def test_get_winner_server_wins(self):
        """
        Test cases where the server wins.
        """
        self.assertEqual(get_winner('rock', 'paper'), 'server')
        self.assertEqual(get_winner('scissors', 'rock'), 'server')
        self.assertEqual(get_winner('paper', 'scissors'), 'server')

    def test_load_config_default(self):
        """
        Test loading configuration when the file does not exist.
        """
        config = load_config('nonexistent.ini')
        self.assertEqual(config['game']['game_mode'], 'man_vs_ai')
        self.assertEqual(config['game']['player_score'], '0')
        self.assertEqual(config['game']['ai_score'], '0')

    def test_save_and_load_config(self):
        """
        Test saving and then loading configuration.
        """
        config = configparser.ConfigParser()
        config['game'] = {
            'game_mode': 'man_vs_ai',
            'player_score': '3',
            'ai_score': '2'
        }
        save_config(config, 'test_config.ini')

        loaded_config = load_config('test_config.ini')
        self.assertEqual(loaded_config['game']['player_score'], '3')
        self.assertEqual(loaded_config['game']['ai_score'], '2')

if __name__ == '__main__':
    unittest.main()
```