МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

АВТОМАТИЗОВАНЕ ПРОЕКТУВАННЯ КОМП'ЮТЕРНИХ СИСТЕМ

Лабораторна робота №5

Виконав: ст. гр. КІ-405
Легкобит Н.В.

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

Пророблена робота

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

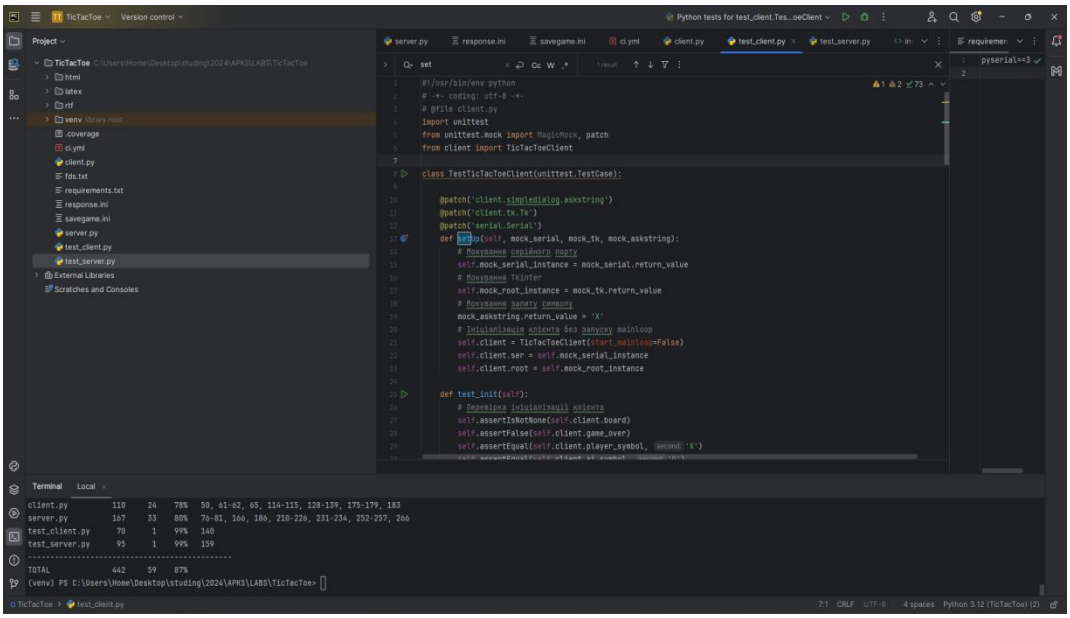

Рис. 1. Частина коду з використанням мокування

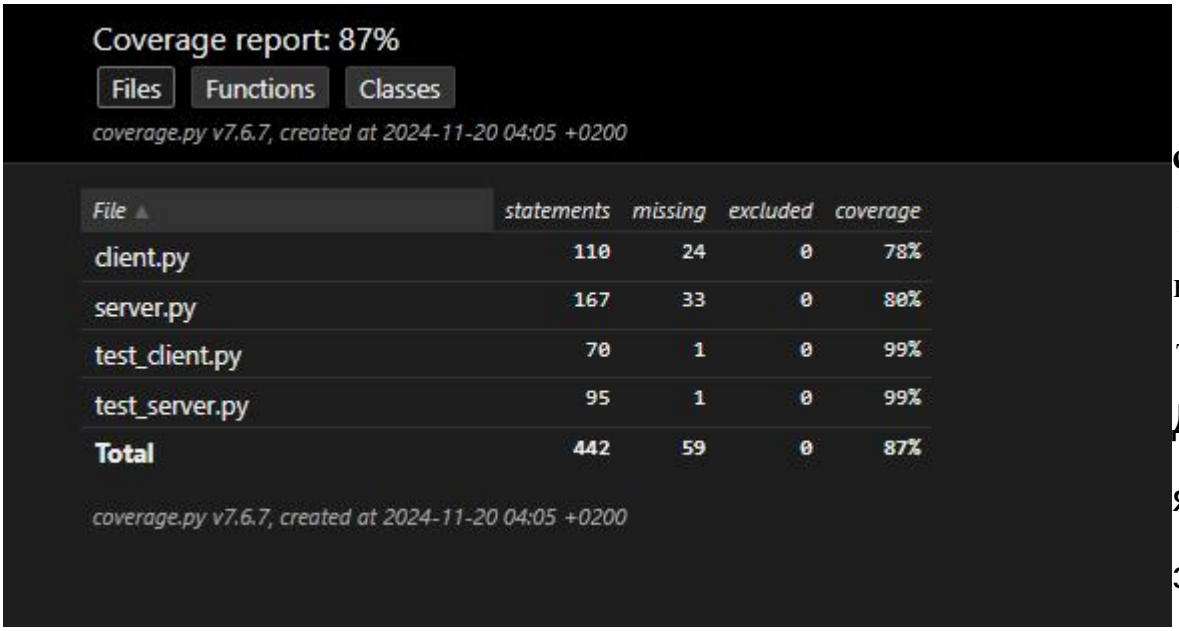

Рис. 2. Покриття

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 **87%**, 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.

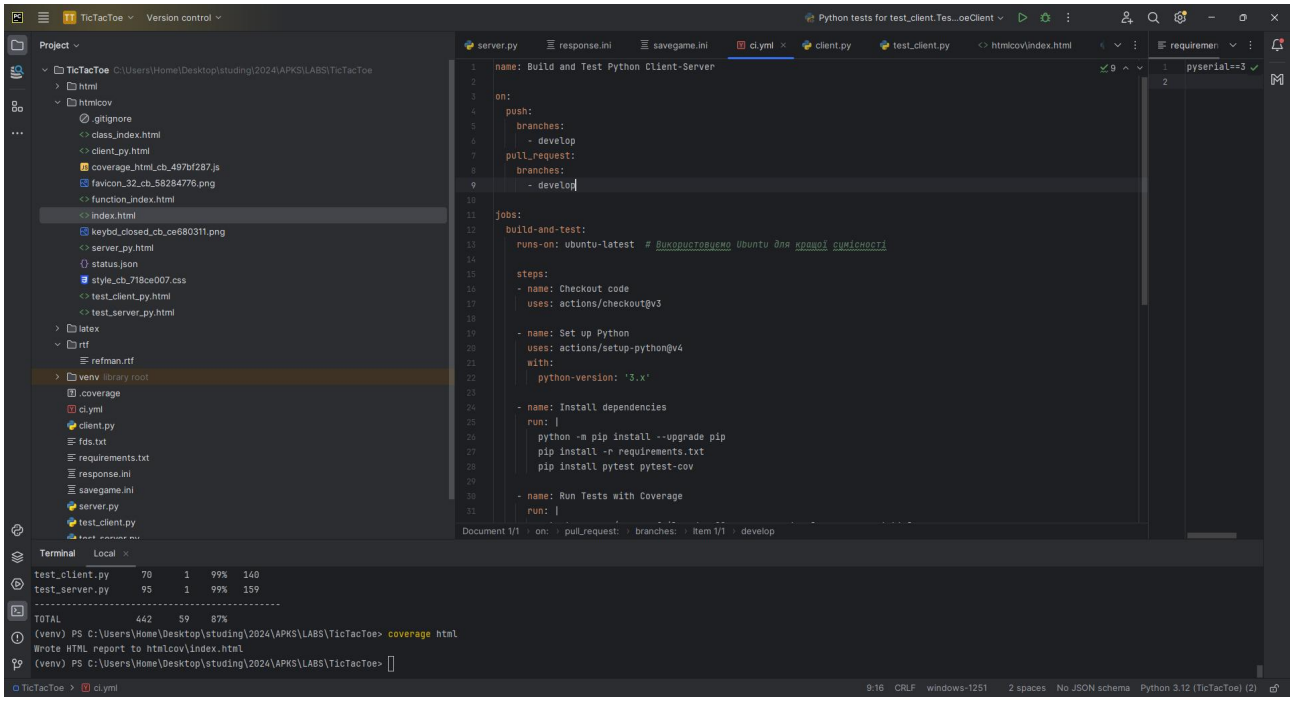

Рис. 4. Оновлений YML файл

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
```

```yaml
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
```