Лабораторна робота №1

1. Тема, мета, короткий теоретичний вступ

Тема: Практичне застосування системи контролю версій Git та юніт-тестування у процесі розробки програмного забезпечення.

Формування повного робочого циклу з GitHub — від створення репозиторію до Pull Request.

Мета: Отримати практичні навички використання сучасних інструментів розробки ПЗ, а саме:

• роботи з системою контролю версій Git та сервісом GitHub;

• побудови історії комітів і гілок проєкту;

• написання юніт-тестів до існуючого коду;

• формування правильного процесу створення Pull Request та командної взаємодії у GitHub.

Короткий теоретичний вступ:

У сучасному процесі розробки програмного забезпечення важливу роль відіграють інструменти, що забезпечують ефективну командну роботу, контроль версій коду та підтримку його якості. Одним із найпоширеніших рішень для цього є система контролю версій Git у поєднанні з хостингом репозиторіїв GitHub. Використання Git дозволяє відстежувати зміни у проєкті, створювати окремі гілки для нових функцій, об’єднувати результати роботи кількох розробників і зберігати повну історію змін коду.

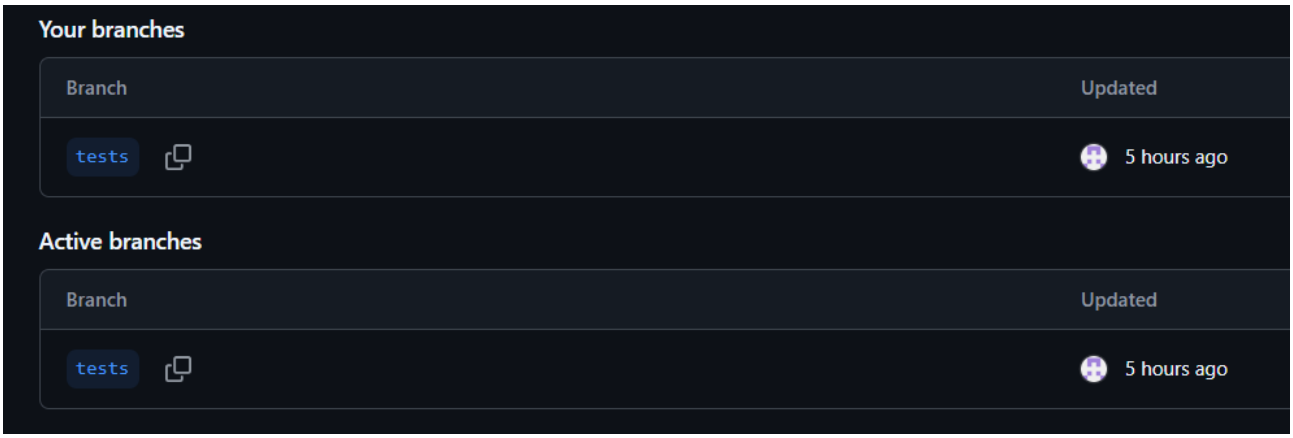
Не менш важливою складовою професійної розробки є юніт-тестування, яке забезпечує автоматичну перевірку правильності роботи окремих модулів програми. Юніт-тести допомагають своєчасно виявляти помилки, полегшують супровід і розвиток коду, а також сприяють підвищенню його надійності.

2. Посилання на репозиторій GitHub.

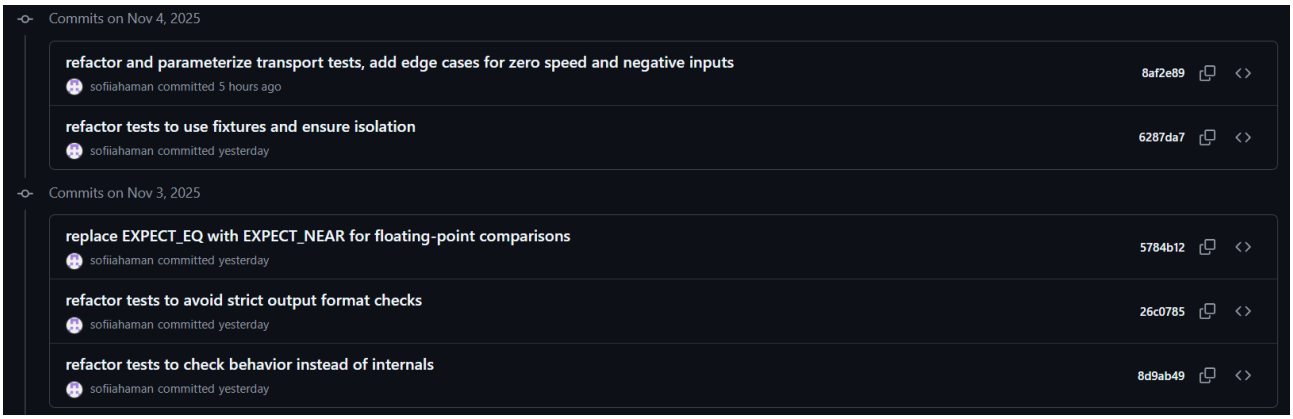
https://github.com/sofiiahaman/lab1-unit-tests.git

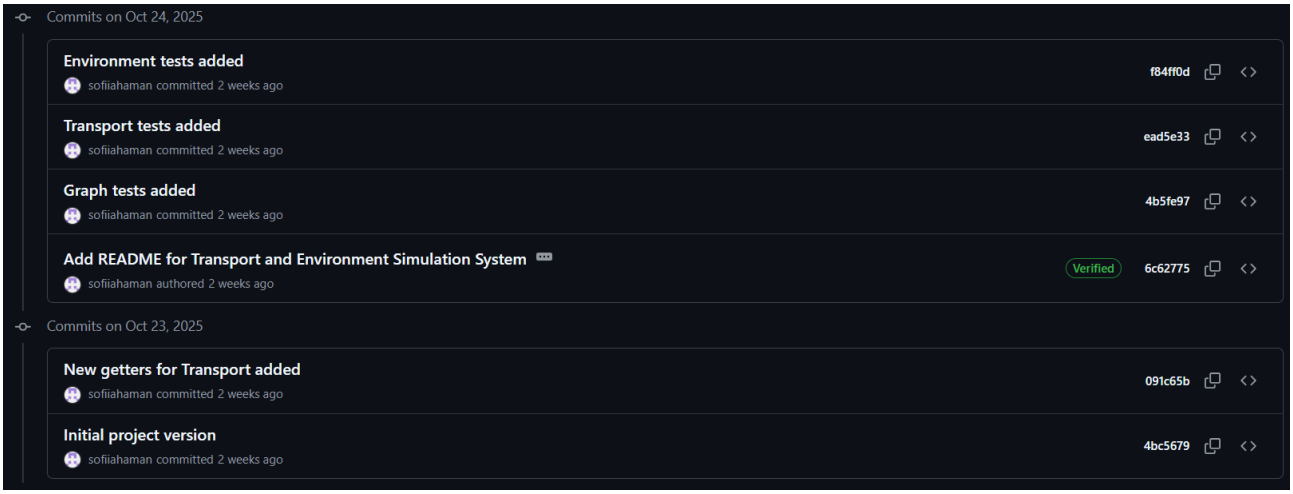
3. Скріншоти основних етапів роботи

Створення гілки



Коміти





Pull Request

https://github.com/sofiiahaman/lab1-unit-tests/pull/1

4. Опис процесу написання юніт-тестів

Процес написання юніт-тестів був спрямований на перевірку коректності реалізації основних класів програми – *Graph*, *Transport*, та *Environment*. Для тестування використовувався фреймворк Google Test (gtest).

Тестування класу Graph

Було реалізовано серію тестів для перевірки базових операцій та алгоритмів роботи з графами:

- Операції з вершинами та ребрами: додавання, видалення та коректність збереження ваги ребер.
- Алгоритми пошуку мінімального кістякового дерева (МКД): реалізації Прима, Крускала та Борувки. Перевірялася правильність кількості ребер у результаті та сумарної ваги дерева.
- Алгоритм Дейкстри: перевірка знаходження найкоротшого шляху між вершинами та поведінки у випадку відсутності шляху.
- Граничні ситуації: тестування порожніх графів, обробка петель (loop-edges) та випадків, коли побудова МКД неможлива через напрямленість графа.

Тестування класів Transport та його нащадків

Було створено параметризовані тести для класів Transport, LandTransport, WaterTransport, AirTransport, а також похідних класів Car, Train, Yacht, Helicopter.

Основні перевірки включали:

- Переміщення транспорту: тестування зміни координат при наявності пального.

- Динаміка швидкості: робота методів accelerate() та brake(); перевірка, що швидкість не стає від’ємною.
- Робота при відсутності пального: транспорт не змінює положення при нульовому рівні палива.
- Коректність поведінки при нетипових параметрах: виклики з від’ємними значеннями швидкості чи прискорення не призводять до помилок виконання.

Використання механізму параметризації тестів (TEST_P) дозволило уникнути дублювання коду та перевірити однакову логіку для різних типів транспортних засобів.

Тестування класу Environment

У межах цього модуля було протестовано функціональність середовища, яка відповідає за взаємодію між транспортом, маршрутами та перешкодами. Зокрема:

- Додавання маршрутів та перешкод: перевірка коректності збереження та доступу до даних.
- Виведення стану середовища: перевірка коректності текстового виводу інформації про маршрути та перешкоди.
- Пошук оптимального маршруту: тестування алгоритму, який визначає найкоротший шлях між точками за допомогою графа.
- Обробка виключень: перевірка реакції системи на відсутність вузлів або маршрутів у графі.
- Функція moveTransport(): перевірка, що при русі транспорту відображається інформація про його маршрут.

Перевірка граничних випадків

Особлива увага приділялася тестуванню крайових ситуацій:

- порожні графи;
- вершини, що відсутні в структурі;

- транспорти без палива;
- маршрути, для яких не існує зв’язку між точками.

Такі перевірки забезпечують підвищену надійність системи та захищають її від неочікуваних помилок під час реального використання.

Підсумки тестування

Усі юніт-тести було успішно пройдено, що підтверджує:

- коректність реалізації алгоритмів у класі Graph;
- правильну поведінку об’єктів транспорту;
- стабільну роботу середовища Environment у різних умовах.

5. Висновки про набуті навички

У результаті виконання роботи були набуті практичні навички використання сучасних інструментів розробки програмного забезпечення, зокрема системи контролю версій Git та сервісу GitHub. Було засвоєно принципи створення та налаштування віддалених репозиторіїв, формування комітів, роботи з гілками проєкту, а також виконання операцій злиття змін через механізм Pull Request.

Під час виконання практичної частини було сформовано розуміння повного циклу командної роботи над програмним продуктом – від ініціалізації репозиторію до інтеграції перевіреного коду у головну гілку.

Окрім цього, отримано досвід написання та виконання юніт-тестів за допомогою фреймворку Google Test, що дозволило перевірити коректність роботи основних класів програми. Практичне застосування юніт-тестування дало змогу оцінити важливість автоматизованої перевірки якості коду та її роль у забезпеченні надійності програмних систем.

Таким чином, у ході роботи було закріплено знання щодо принципів командної розробки, тестування та контролю версій, що є невід’ємними складовими сучасного процесу створення програмного забезпечення.