Лабораторна #3: Tetris (Test doubles)

Організаційні кроки:

1. Створіть свій репозиторій для завдання Lab 3 (invite link).
2. Налаштуйте GitHub Actions так, щоб тести запускалися на кожен push та pull request.
3. Продивіться відео з прикладом виконання і репозиторій з прикладом
    a. Підказочка: відео можна дивитись на швидкості 1.5-2x
4. Виконайте алгоритмічне завдання. На що слід звернути увагу при написанні:
    a. Потрібно чітко розділити програму на шари
        i. “логічний” - тут реалізована логіка гри де ви оперуєте з абстракціями, які ви ж і створили (ігрове поле, код пов’язаний з правилами гри, тощо)
        ii. “введення/виведення” - тут реалізується комунікація вашої програми з зовнішнім середовищем, де ви працюєте з наданими абстракціями (стандартний виводом, файловою системою)
        iii. “комунікаційний” - шар який забезпечує комунікацію між “логікою” і “введенням/виведенням”. У цьому шарі повинен бути реалізований шаблон “DI (dependency injection)” який дозволяє підміняти шар введення/виведення у тестовому оточенні.
    b. Покрийте тестами усі шари. При тестуванні “комунікаційного” шару використовуйте mock’и. Будь ласка, не використовуйте бібліотеки для сворення mock’ів - створіть їх самостійно.
5. Додайте файл README.md з інструкцією для налаштування середовища і запуску тестів. За цією інструкцією, людина, що перевіряє повинна бути здатна однією командою налаштувати середовище/встановити залежності та іншою - запустити тести.

Критерії оцінювання робіт

Оцінюється якість тестів, а не код, що вирішує задачу. Це означає, що вирішена задача без тестів оцінюється в 0 балів.

За виконання роботи на одній з наступних мов нараховуються додаткові бали: Kotlin, TypeScript, C#.

За виконання роботи на одній з наступних мов нараховуються подвійні додаткові бали: Rust, Haskell, Erlang, F#, OCaml, Scheme (or any other lisp).

Списана робота оцінюється в 0 балів.

Алгоритмічне завдання:

Необхідно реалізувати логіку гри, що схожа на Тетріс. На вхід подається початковий стан екрану на якому є одна “підвішена” фігура і “ландшафт” - залишки попередніх фігур. Необхідно вивести стан екрану в грі, коли фігура “впала” (тобто гравець не давав жодних команд, а просто чекав коли фігура зіткнеться з “ландшафтом”).

Формат вводу / виводу

Вхідний файл містить розміри екрану та знімок початковому стану гри. У ньому:
. - порожній піксель
p - піксель “підвішеної” фігури ("p" від слова "piece")
- піксель “ландшафта” (залишків фігур, які впали раніше)

Фінальний екран гри потрібно вивести на екран

Приклади вводу/виводу

Виклик фінальної програми
```
> ./tetis input.txt
```

Input	Output
7 8 ..p..... .ppp.... ..p..... ....#.... ...#...# #..#####	 ....p..... .ppp.... ..p#.... ...#...# #..#####
5 6 ..p... ##p.## ##pp## ##..## ##..##	 ..... ##..## ##p.## ##p.## ##pp##

Якщо вхідний файл містить некоректні дані, потрібно вивести повідомлення про помилку (без подробиць про помилку). Фігура завжди тільки одна.

Приклад плану розв'язання

1. Зчитати розміри екрану і зберегти їх.
2. Зчитати початковий стан екрану і створити дві колекції: одну з точок “фігури”, іншу - з точок “ландшафту”. Точка це структура даних, що містить координати одного пікселя.
3. Зібрати розміри екрану, “фігуру” та “ландшафт” у одну структуру даних (наприклад, клас), яку далі будемо називати “поле”
4. Створити функцію, яка приймає на вхід “поле”, зміщує кожен піксель “фігури” на одну позицію вниз і перевіряє чи нова координата не накладається на один з

пікселів “ландшафту” або виходить за межі поля. Якщо накладається або виходить - повертає незмінний стан “поля”, якщо ні - повертає новий стан “поля” (зі зміщеною “фігурою”).
5. Створити функцію, яка циклічно викликає функцію з минулого пункту до того моменту, як “поле” перестає змінюватись.
6. Створити функцію, яка віддає “поле” у текстовому вигляді.
7. Вивести результат на екран (у standard output).