

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"
Кафедра Програмного Забезпечення



Звіт про виконання лабораторних робіт
№ 1-4
З курсу: *“Комп’ютерна графіка”*
Варіант І2

Виконав:
ст. гр. ПЗ-31
Бабіля О.О.

Перевірено:
Доц. каф. ПЗ
Левус Є. В.

« ____ » _____ 2023 р.
Σ= _____

Львів – 2023

1. Зміст	2
2. Формулювання завдання	3
3. Теоретичні відомості	5
3.1. Опис функцій програми	5
3.2. Алгоритми фракталів згідно варіанту	6
3.3. Коротка анотація колірних моделей	8
3.4. Оптимальний матричний вираз	9
3.5. Реалізація графічного режиму	10
4. Результати	13
4.1 Wireflow	13
4.2 Wireframe	14
4.3 Зображення фракталів	18
4.4 Вигляд програми з колірними моделями	19
4.5 Вигляд програми з афінними перетвореннями	19
4.6 Навчальна компонента додатку	20
4.7 Реагування помилки на неправильне введення даних	23
4.8 Текст програми з коментарями	25
5. Висновки	31
6. Список використаних джерел	32

2. Формулювання завдання

2.1. Лабораторна робота №1

Базуючись на wireframes та технологіях, які ви обрали, створіть UI-дизайн для шести виглядів системи (початковий екран, вікно для Л2, вікно для Л3, вікно для Л4, два вікна навчальних матеріалів).

2.2. Лабораторна робота №2

Необхідно побудувати на екрані фрактальні зображення згідно варіанту та забезпечити їх збереження у файлах. Реалізувати введення користувачем усіх зазначених параметрів, що впливають на вигляд фракталів.

Варіант І2:

Побудувати фрактальні зображення:

а) Фрактал Мандельброта $f(z)=z*z+c$. Можливість згенерувати різні зображення, а саме для:

- різних кольорових схем,
- різного масштабування .

б) Броунівський рух.

2.3 Лабораторна робота №3

Необхідно реалізувати:

- 1) перерахування графічного зображення з одного простору інший
- 2) відображення координат точок в кожному просторі кольорів;
- 3) опрацювання атрибуту кольору згідно варіанту з обов'язковим використання перцепційних моделей;
- 4) збереження змін у графічному файлі.

Варіант І2:

Колірні моделі: RGB і HSV. Змінити насиченість по зеленому кольору.

2.4 Лабораторна робота №4

Необхідно реалізувати:

- 1) відображення координатної площини і всіма відповідними позначеннями та підписами;

- 2) зручне введення координат фігури (чи її конструктивних елементів) користувачем згідно варіанту;
- 3) синтез рухомого зображення на основі афінних перетворень згідно варіанту з обов'язковим використанням матричних перетворень для реалізації комбінації афінних перетворень (рух – це зміна положення фігури в часі);
- 4) динамічну зміну одиничного відрізка координатної площини;
- 5) збереження початкового зображення у файл.

Варіант I2:

Реалізувати рух для паралелограма, введеного за його вершинами, на основі повороту за годинниковою стрілкою відносно вибраної вершини з одночасним його зменшення у A разів.

3. Теоретичні відомості

3.1. Опис функцій програми

3.1.1. Діаграма прецедентів

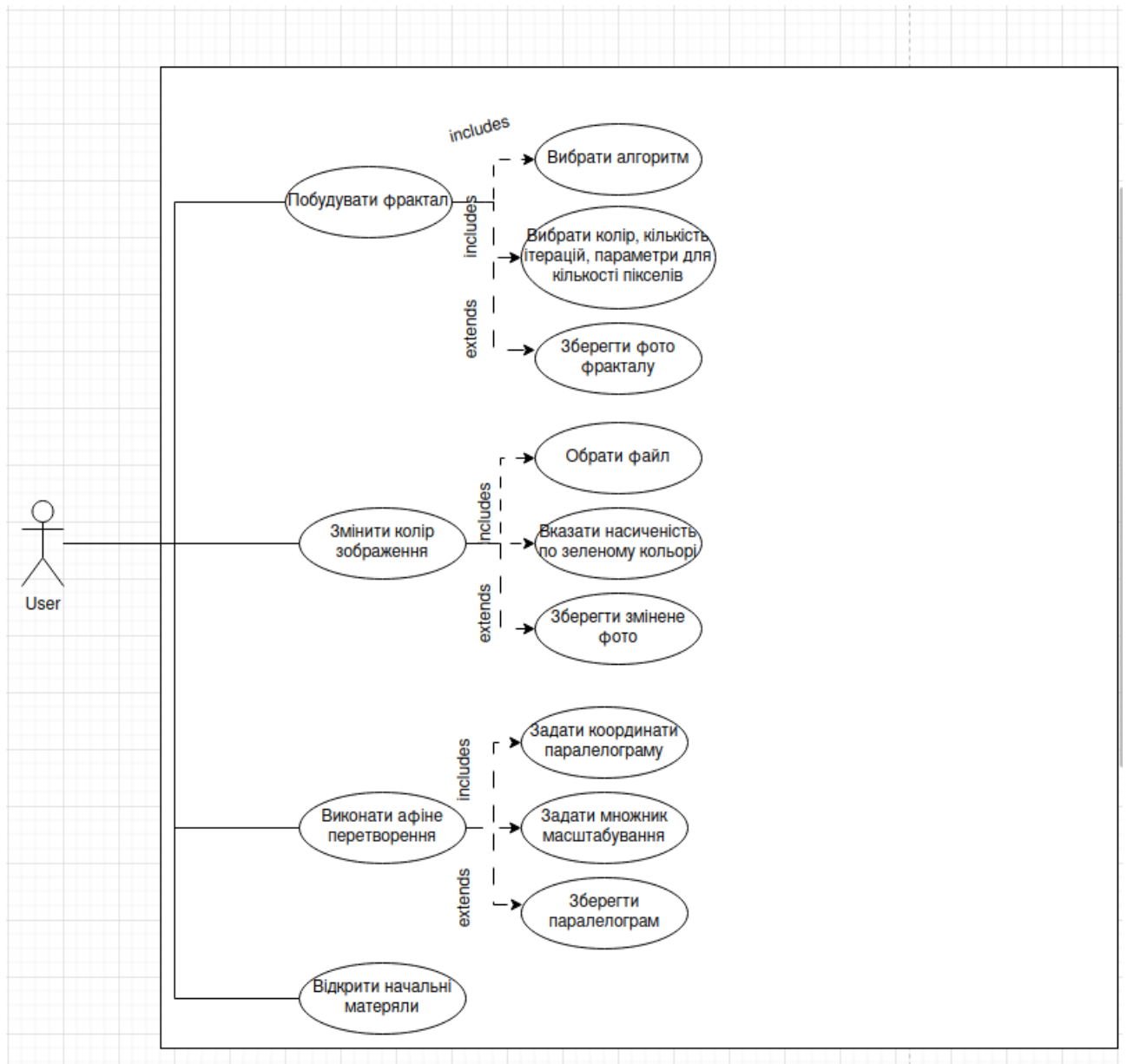


Рис. 1. Діаграма сценаріїв використання

3.1.2. Текстова специфікація функцій програми

Програма виконує наступні функції:

Побудова фракталів, а саме Мандельброта та Броунівського руху.

Перетворення кольорових схем, а саме з RGB в HSV, та навпаки.

Зміна насиченості зображення на зеленому кольорі.

Рух паралелограма за годинниковою стрілкою, з одночасним зменшенням.

Також, програма є цілком навчальною, з можливістю експериментування та вивчення предметної області, за допомогою навчальних матеріалів. Легка для використання з захистом від помилок.

3.2. Алгоритми побудови фракталів:

3.2.1. Алгоритм побудови фракталу Мандельброта

MF1: Введення параметрів:

а) Максимальна кількість ітерацій (`max_iter`), що визначає границю рекурсивного обчислення фракталу.

в) Розміри зображення (`width` і `height`).

б) Колірна палітра (`color`), яка використовується для визначення кольорів у фракті.

MF2: Визначення області у комплексній площині, в якій буде створюватися фрактал. В даному випадку, це прямокутник з верхньою лівою точкою (`xmin`, `ymax`) та нижньою правою точкою (`xmax`, `ymin`).

MF3: Для кожного пікселя (`x`, `y`) зображення визначається відповідне комплексне число `c` через лінійну інтерполяцію між (`xmin`, `xmax`) та (`ymin`, `ymax`).

MF4: Запускається ітеративний процес, де кожна ітерація оновлює значення `z` за правилом $z = z^2 + c$.

MF5: Якщо після `max_iter` ітерацій значення `z` залишається обмеженим ($\text{abs}(z) \leq 2$), то піксель отримує колір, інакше йому призначається чорний колір.

MF6: Створюється двовимірний масив, представляючий зображення, де кожен піксель кодується трьома значеннями кольору.

MF7: Для кожного пікселя використовуються параметри функції Мандельброта для визначення його кольору.

MF8: Зображення генерується за допомогою вказаної колірної палітри.

3.2.2. Алгоритм побудови фракталу Броунівський рух

BMF1: Введення параметрів: а) `max_iter`: Максимальна кількість ітерацій, що визначає границю рекурсивного обчислення фракталу. б) `color`: Колірна палітра для визначення кольорів у фракталі. в) `diffusion coefficient`: коефіцієнт розсіювання. г) `dt` розмір часового кроку.

BMF2: Вибір випадкових кроків: Для кожного кроку частинка робить рух у випадковому напрямку. Величину кроку визначає користувач.

BMF3: Збереження координат: Додати координати кожного кроку до списку для збереження шляху частинки.

BMF4: Повторення кроків: Повторюйте кроки BMF2-BMF3 багато разів для отримання більш складеного шляху частинки.

BMF5: Візуалізація результатів: Візуалізувати траєкторію частинки на площині. Кожна позиція, яка була збережена, з'єднується лініями для утворення траєкторії.

3.3. Анотація кольорових моделей RGB та HSV:

3.3.1 RGB (Red, Green, Blue):

Опис:

- Кольоровий простір: RGB – це адитивна кольорова модель, що базується на суміші трьох основних кольорів: червоного (Red), зеленого (Green) і синього (Blue).
- Представлення кольору: Кожен колір представлений комбінацією значень для червоного, зеленого і синього, де кожне значення може бути в діапазоні від 0 до 255.

Практичне застосування:

- Відображення кольорів на екранах моніторів, телевізорів і дисплеях.
- Обробка та редагування зображень та відео в графічних програмах.

Переваги:

- Простота в реалізації та розумінні.
- Ефективне для відображення кольорів на екранах.

Недоліки:

- Не завжди інтуїтивно зрозуміло для представлення інших кольорових характеристик, таких як насиченість та яскравість.
- Може бути менш ефективним для представлення натуральних кольорів, таких як в друкарстві.

3.3.2 HSV (Hue, Saturation, Value):

Опис:

- Кольоровий простір: HSV – це кольорова модель, яка представляє кожен колір за допомогою трьох параметрів: відтінку (Hue), насиченість (Saturation) і яскравість (Value).
- Представлення кольору: Відтінок вимірюється у градусах (від 0 до 360), а насиченість та яскравість вимірюються від 0 до 100 або від 0 до 1.

Практичне застосування:

- Кольорова корекція та фільтри в графічних редакторах.
- Визначення кольору в області дизайну та візуалізації.

Переваги:

- Інтуїтивно зрозуміла представлення кольорів, особливо для користувачів.
- Легкість у роботі зі змінами кольору, такими як зміна яскравості або насиченості.

Недоліки:

- Складніше для розуміння та реалізації для програмного коду, порівняно із RGB.
- Може викликати зміну відтінку при змінах яскравості або насиченості, що не завжди бажано.

3.4. Оптимальний матричний вираз:

$$\begin{pmatrix} a & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -m & -n & 1 \end{pmatrix} \begin{pmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{pmatrix} \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ 1 & 1 & 1 & 1 \end{pmatrix} =$$
$$\begin{pmatrix} a(\cos(\phi)) & a(\sin(\phi)) & 0 \\ -d(\sin(\phi)) & d(\cos(\phi)) & 0 \\ -m(\cos(\phi)) + n(\sin(\phi)) & -n(\cos(\phi) - m(\sin(\phi))) & 1 \end{pmatrix} \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

3.5. Реалізація графічного режиму

Теорія та аргументація вибору технологій для графічного режиму в FastAPI:

3.5.1. FastAPI:

Обґрунтування:

- 1) Швидкість та продуктивність: FastAPI є однією з найшвидших для створення API на мові програмування Python. Це дозволяє ефективно обробляти запити, зокрема, отримані від клієнта для графічних даних.
- 2) Документація та автодокументування: FastAPI надає автоматичну генерацію документації API. Це полегшує розробку, тестування та розуміння API для реалізації графічного режиму.
- 3) Відмінна підтримка асинхронності: FastAPI підтримує асинхронне програмування, що є важливим аспектом при роботі з асинхронними запитами та оновленнями.

3.5.2. Matplotlib:

Обґрунтування:

- 1) Широкий функціонал графіки: Matplotlib надає величезний спектр можливостей для створення графіків та діаграм. Це дозволяє реалізувати різноманітні графічні елементи для відображення даних.
- 2) Легкість використання: Бібліотека проста у використанні та має зрозумілий API, що робить її ідеальним інструментом для швидкого створення графіків у веб-сервері.
- 3) Широка спільнота та підтримка: Matplotlib активно підтримується і використовується у великій кількості проектів, що гарантує надійність та наявність допомоги від спільноти.

3.5.3. htmx:

Обґрунтування:

- 1) Асинхронність та динаміка: htmx дозволяє асинхронно оновлювати вміст веб-сторінки, що є важливим для реалізації динамічного графічного інтерфейсу без повного перезавантаження сторінки.
- 2) Легкість інтеграції: htmx може легко інтегруватися з FastAPI, використовуючи атрибути HTML та створюючи взаємодію між клієнтом та сервером без зайвого коду.

3.5.4. HTML + CSS:

Обґрунтування:

- 1) Стандартні технології для веб-розробки: HTML та CSS є основними технологіями для створення веб-сторінок. Використання їх є стандартним і дозволяє ефективно розгортати та підтримувати веб-інтерфейс.
- 2) Легкість інтеграції: HTML + CSS легко інтегруються з FastAPI, дозволяючи розширювати можливості фреймворку за допомогою стандартних інструментів для веб-розробки.
- 3) Можливість створення динамічних інтерфейсів: За допомогою HTML та CSS можна легко створювати динамічні інтерфейси та взаємодію з користувачем без повного перезавантаження сторінки.

3.5.5 Загальний підхід:

Застосування FastAPI для обробки HTTP-запитів та відповідей, Matplotlib для генерації графічних даних та htmx для асинхронного оновлення вмісту сторінки створює ефективний та динамічний інтерфейс для взаємодії з графікою на веб-сторінці побудовану на html та css.

3.5.6. Опис використаних графічних примітивів:

Для побудови Броунівського руху було використано графічні примітиви у вигляді ліній, кожна з яких з'єднує дві точки, що описують випадковий рух частинок у часі.

Для побудови фракталу Мандельброта та для роботи над кольоровими схемами примітивів не було використано, оскільки робота виконувалась над конкретними пікселями.

Для відображення руху паралелограма в часі, було використано відповідний примітив — паралелограм.

4. Результати

Під час виконання даних лабораторних робіт, я розробив Wireframe та Wireflow, додаток з фронтендом згідно до них, сервер для виконання обчислень та запитів.

4.1. Wireflow

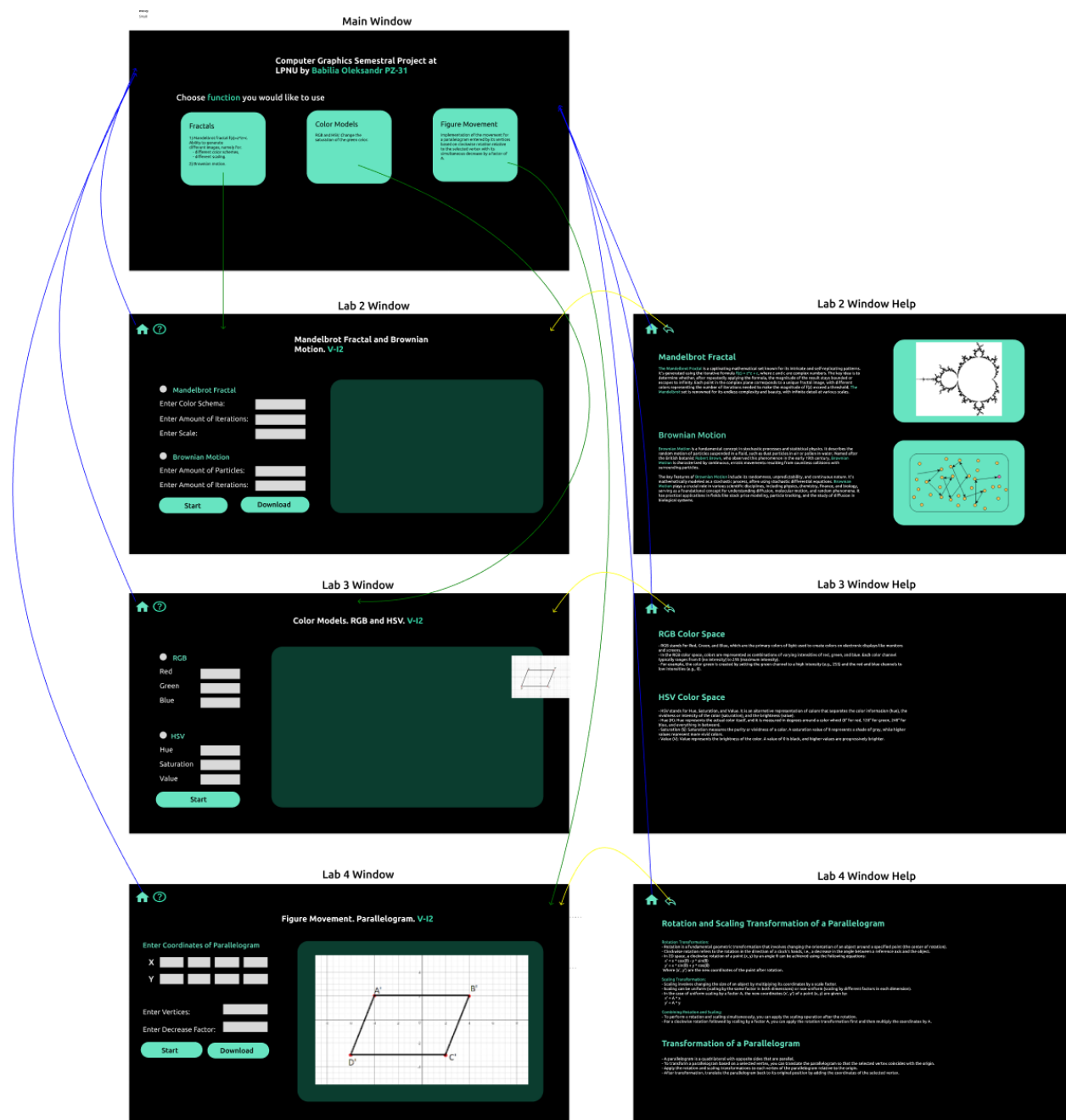


Рис. 2. Wireflow головного меню програми

Після виконання роботи Wireflow залишився не змінним.

4.2. Wireframe

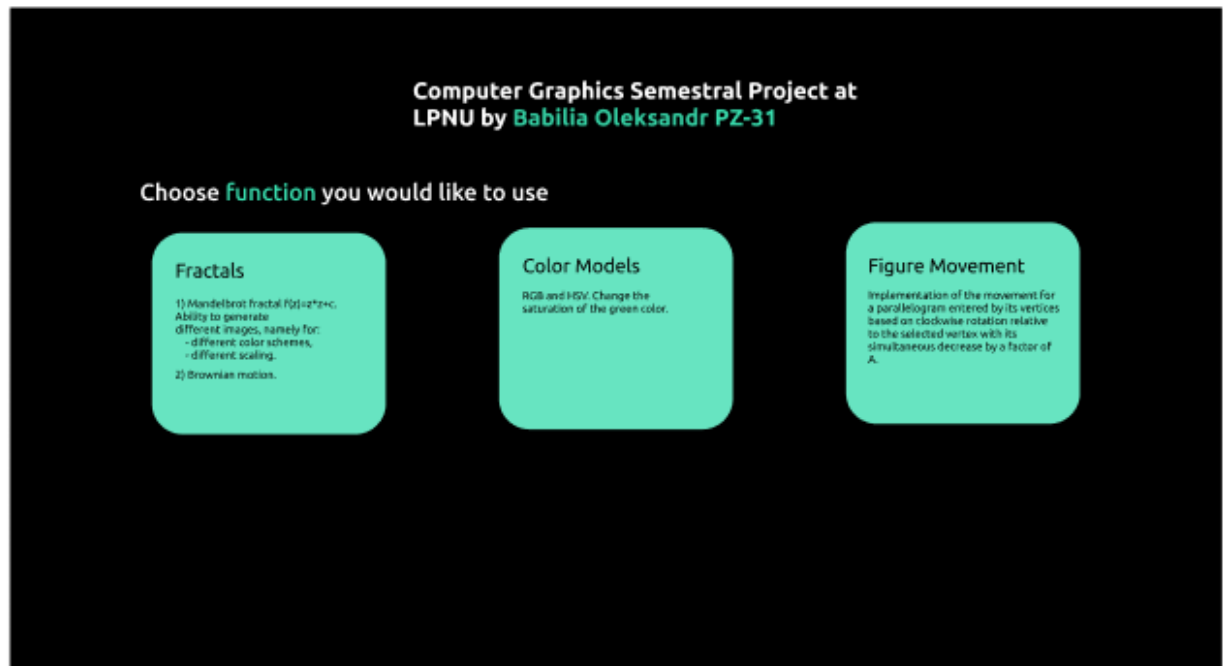


Рис. 3. Wireframe головного меню програми

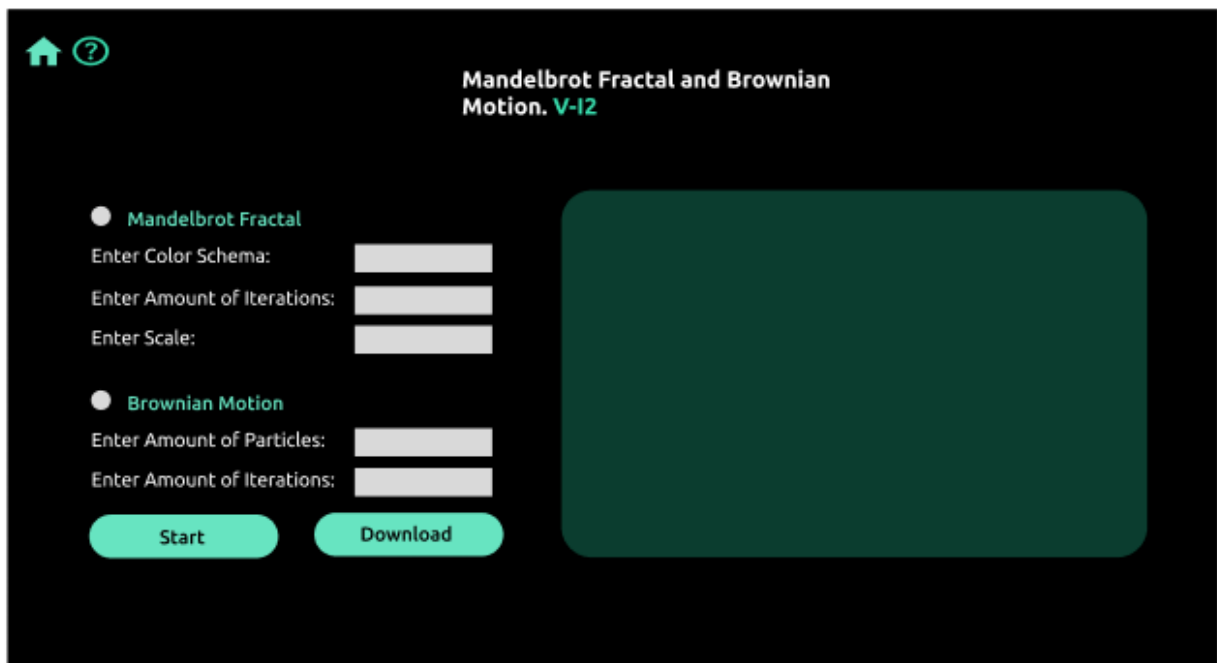


Рис. 4. Wireframe вікна з фракталами

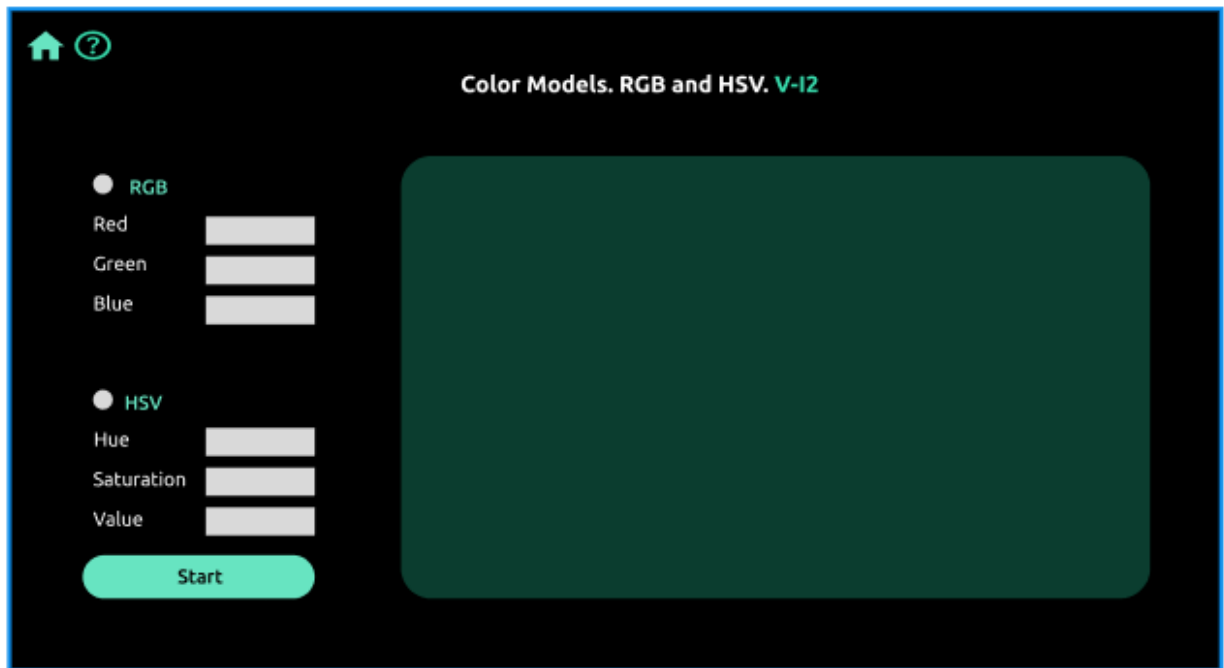


Рис. 5. Wireframe вікна з колірними моделями

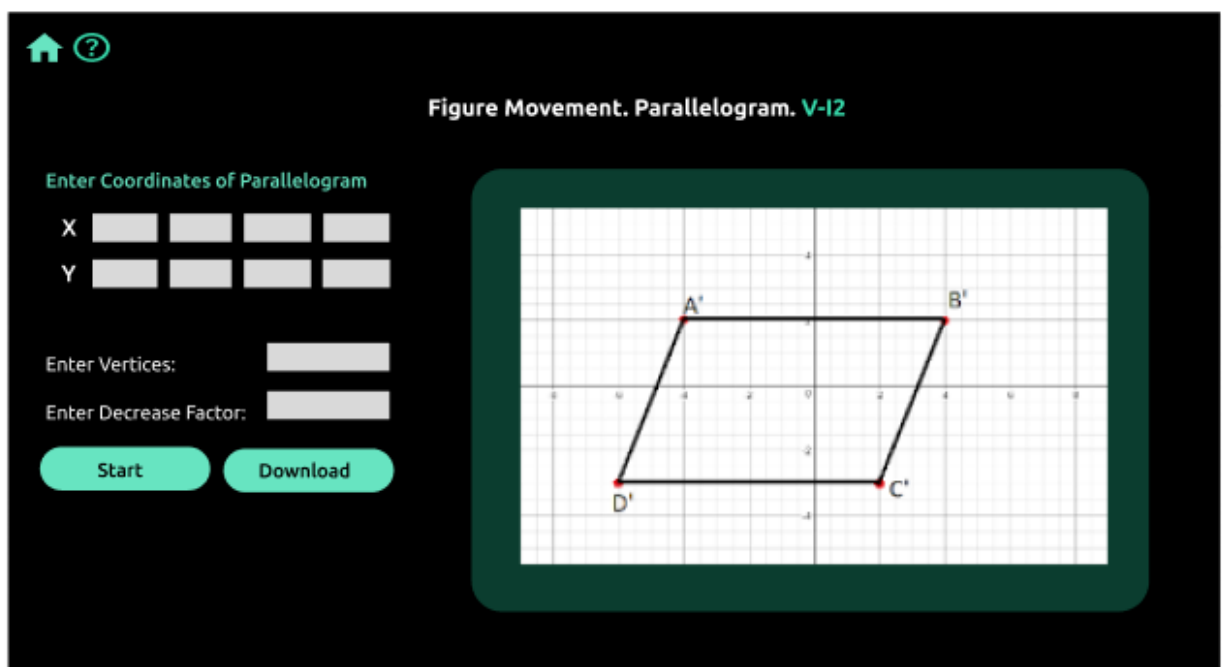


Рис. 6. Wireframe вікна з афінними перетвореннями

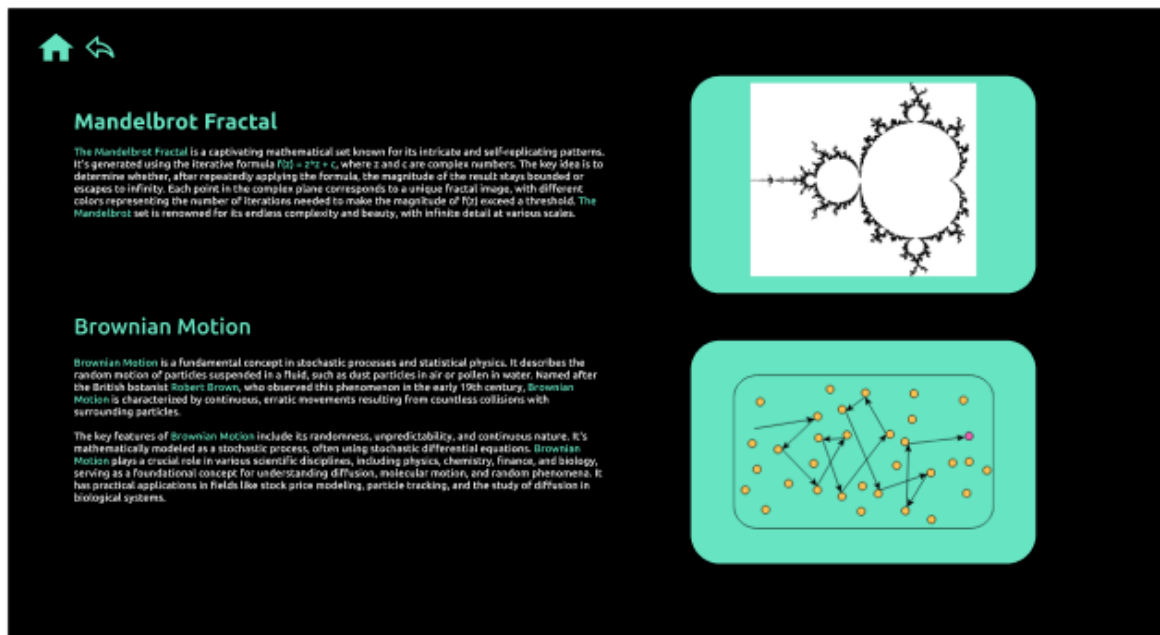


Рис. 7. Wireframe допоміжного вікна з фракталами

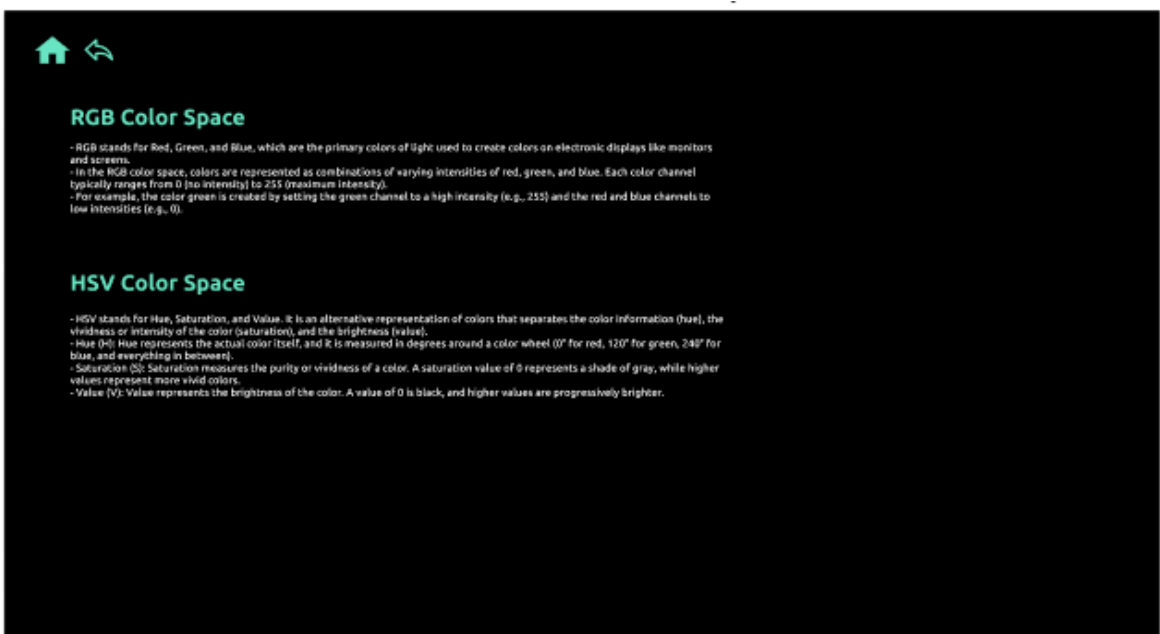


Рис. 8. Wireframe допоміжного вікна з колірними моделями

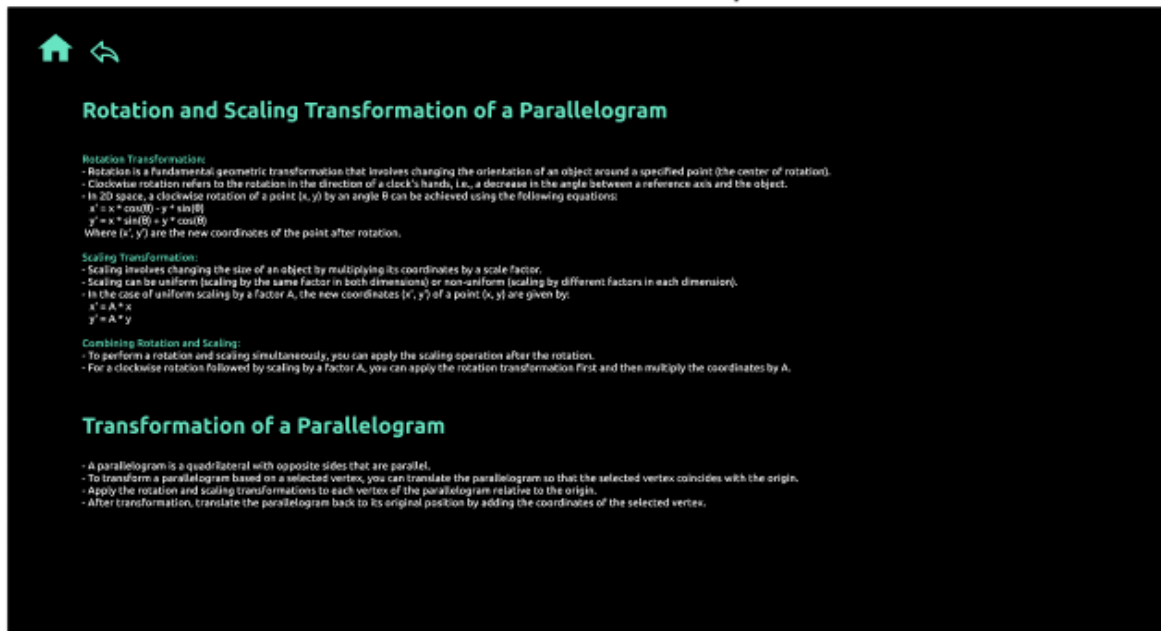


Рис. 9. Wireframe допоміжного вікна з афіними перетвореннями

При виконанні лабораторних робіт, змінились форми для введення даних на всіх головних вікнах, оскільки при створенні вайрфрейму була допущена помилка при зборі інформації по предметній галузі і не коректно обрані поля для введення даних у формі

4.3. Зображення фракталів

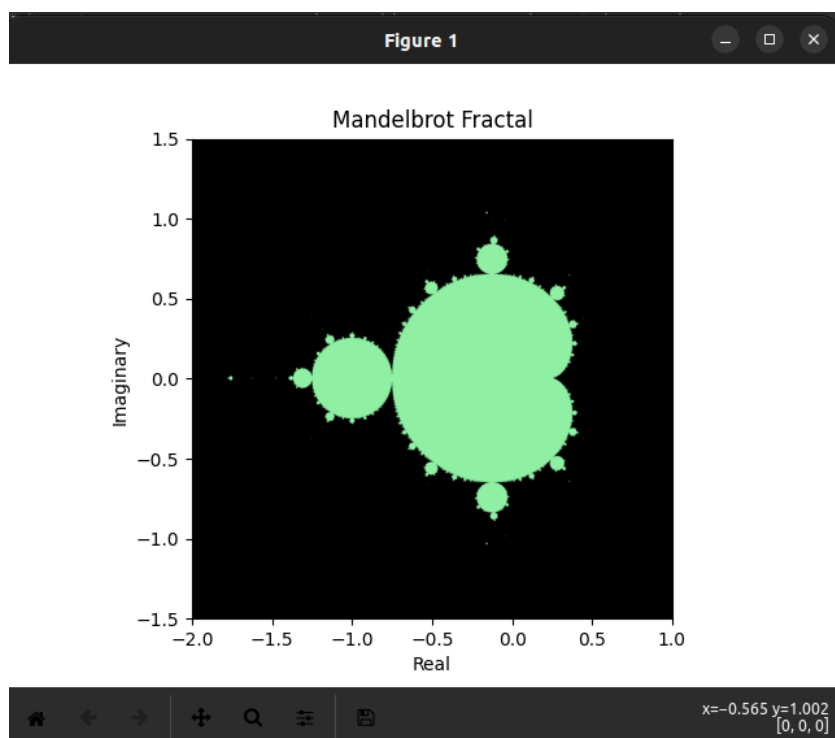


Рис. 10. Фрактал Мандельброта

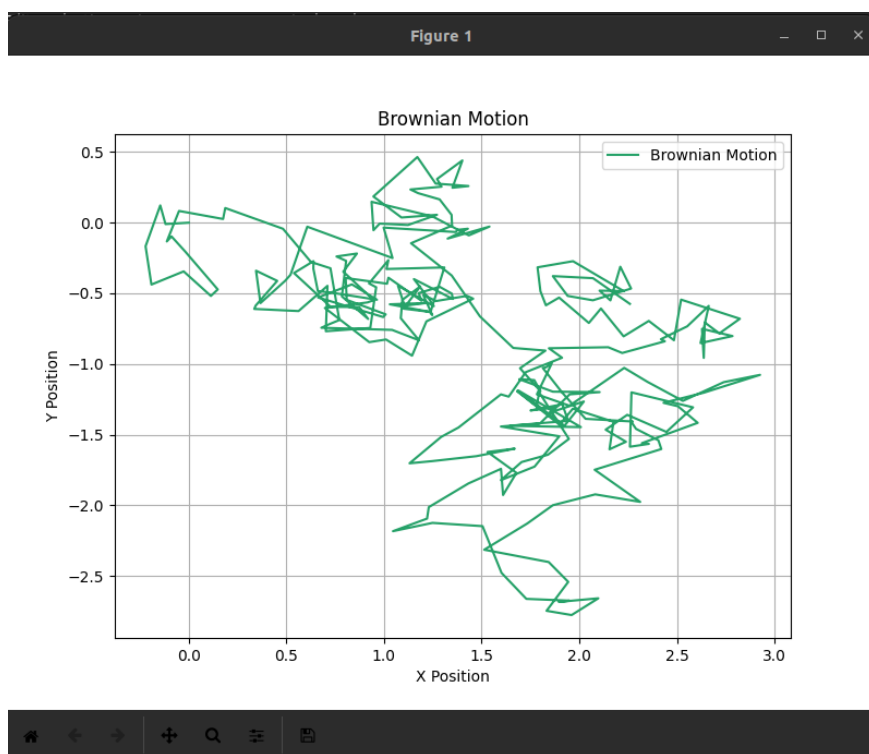


Рис. 11. Фрактал «Броунівський рух»

4.4. Вигляд програми при роботі з колірними моделями

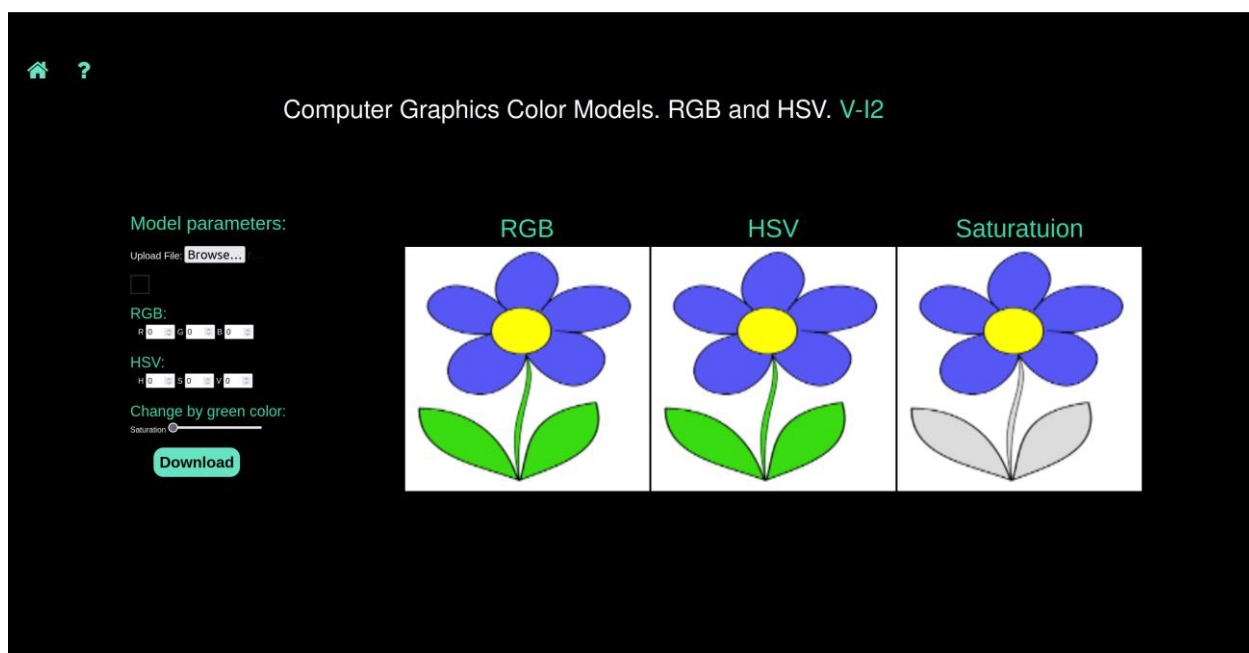


Рис. 12. Вигляд програми роботи з колірними моделями та модифікація зображення

4.5. Вигляд програми при роботі з афінними перетвореннями

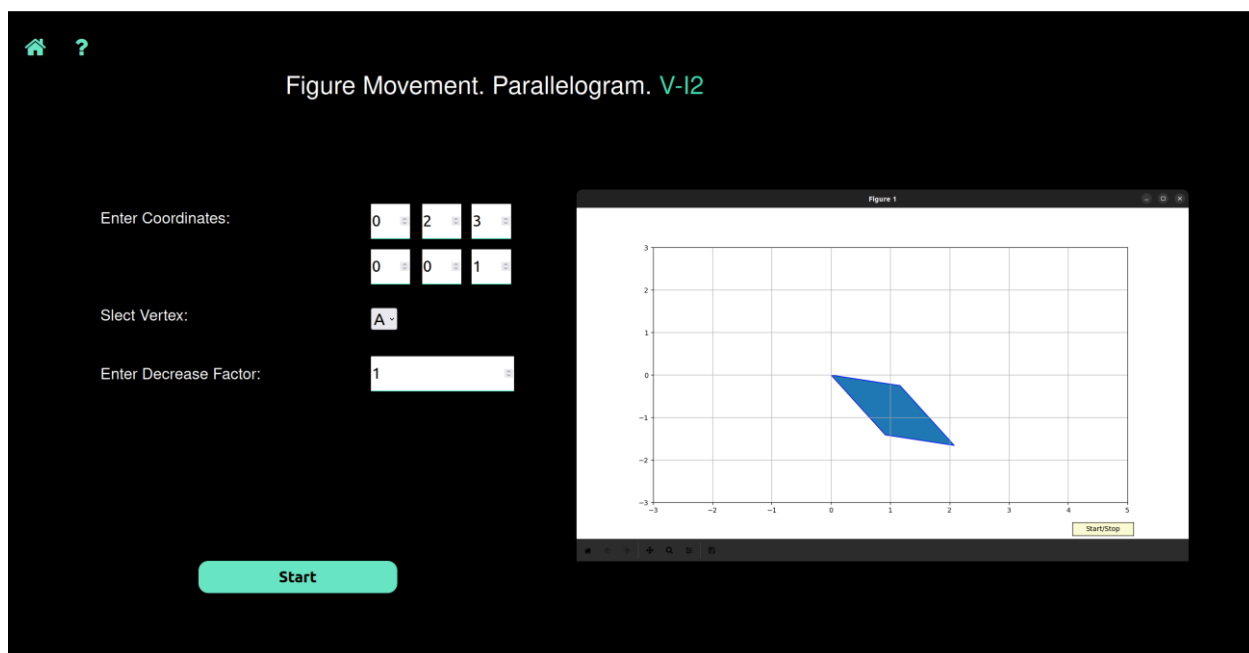


Рис. 13. Вигляд програми роботи з афінними перетвореннями

4.6. Навчальна компонента додатку

На головному вікні в паровму верхньому кутку є іконка, яка дає змогу пройти тестування по предметній області:

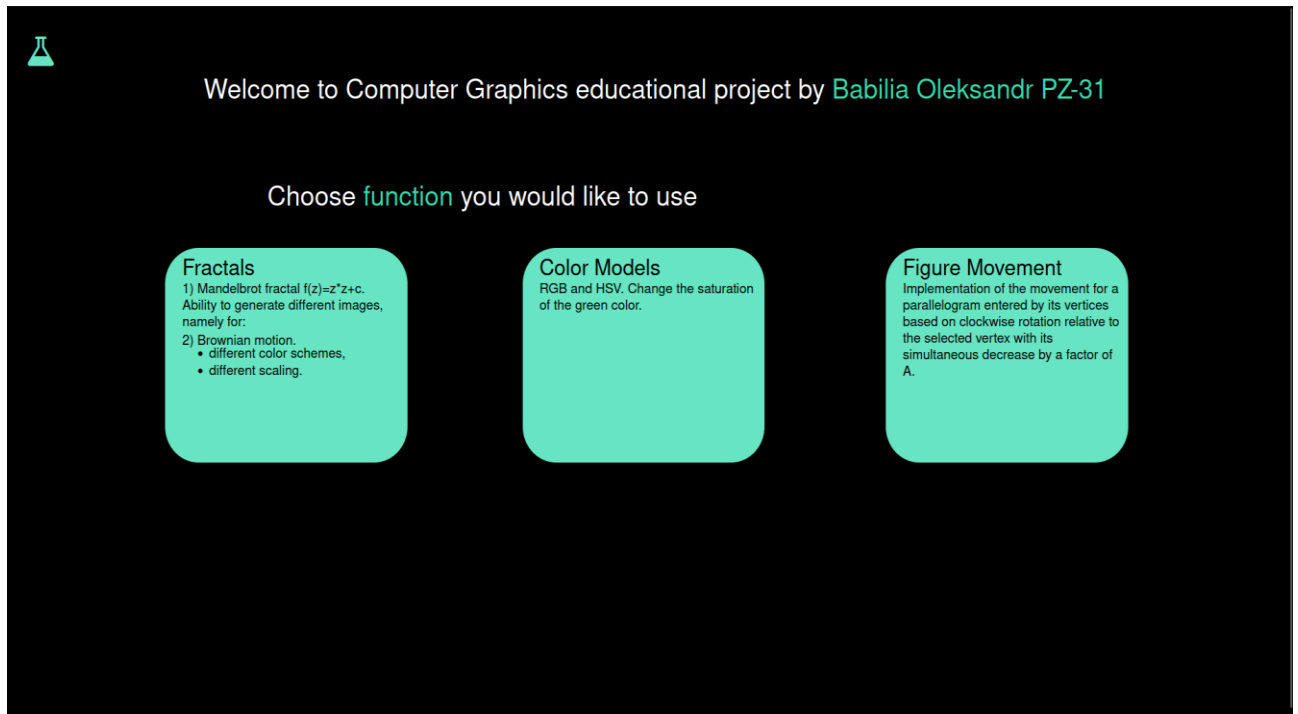


Рис. 14. Вигляд головного вікна

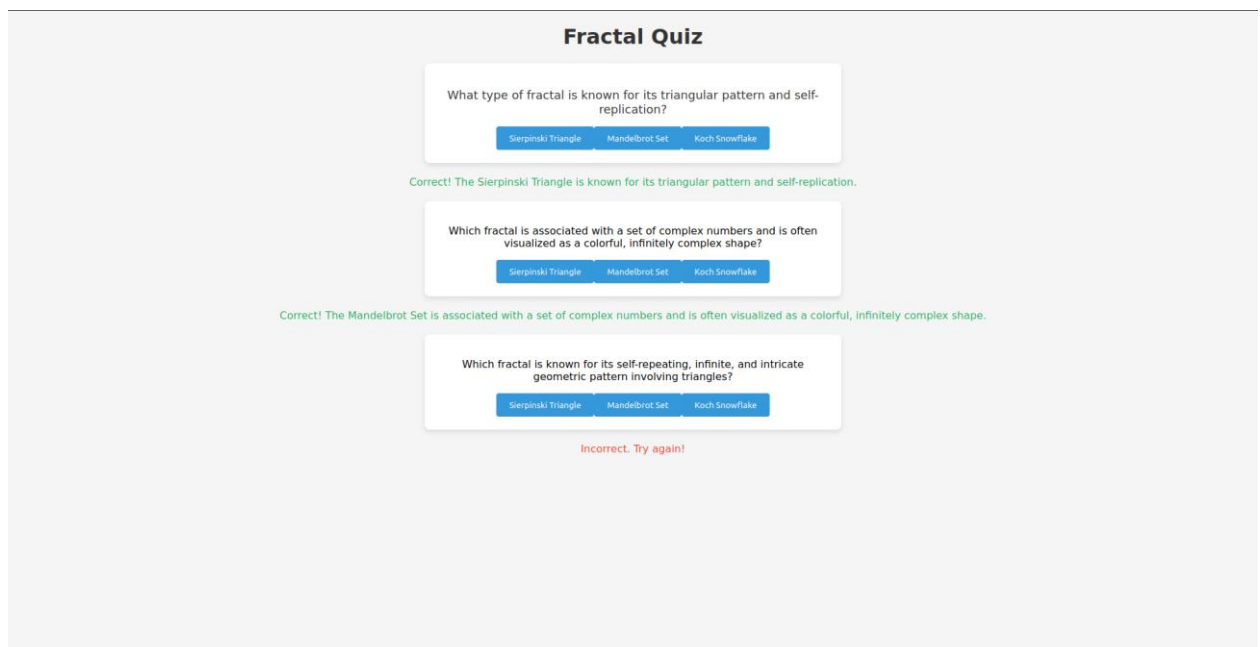


Рис. 15. Тестування

На кожному вікні з функцією є відповідна іконка, яка дає змогу перейти до навчальних матеріалів та переглянути коротке відео по темі.

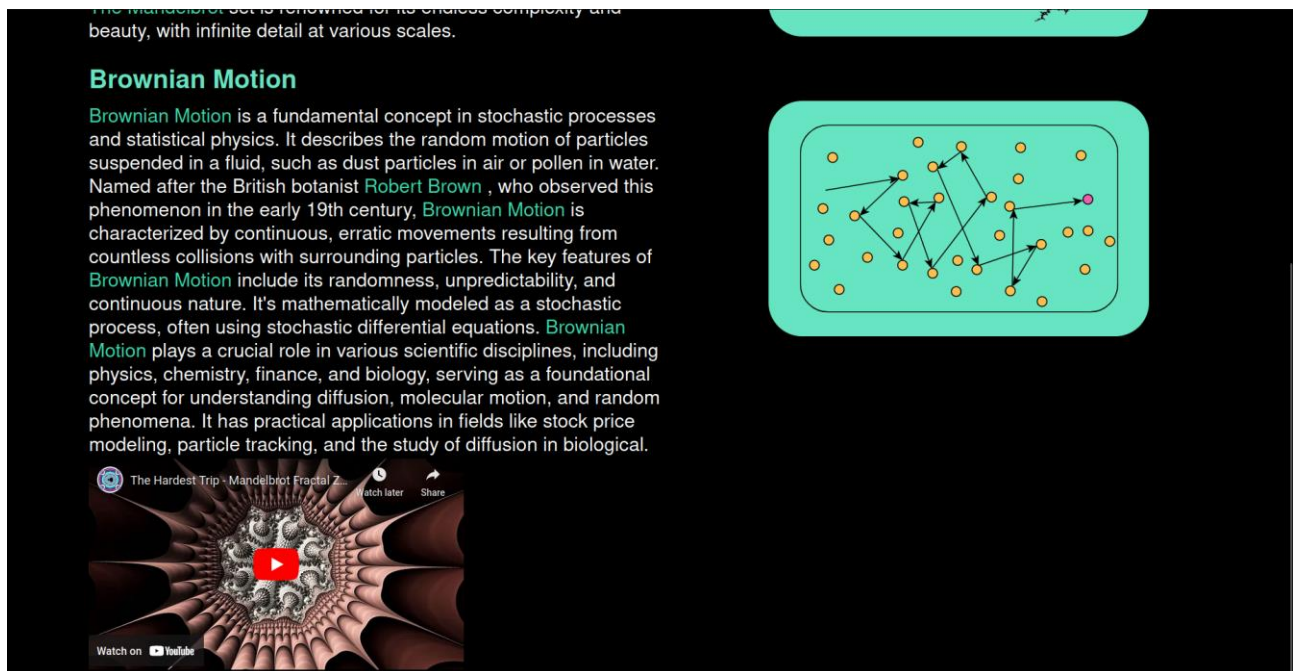


Рис. 16 Навчальне вікно до фракталів

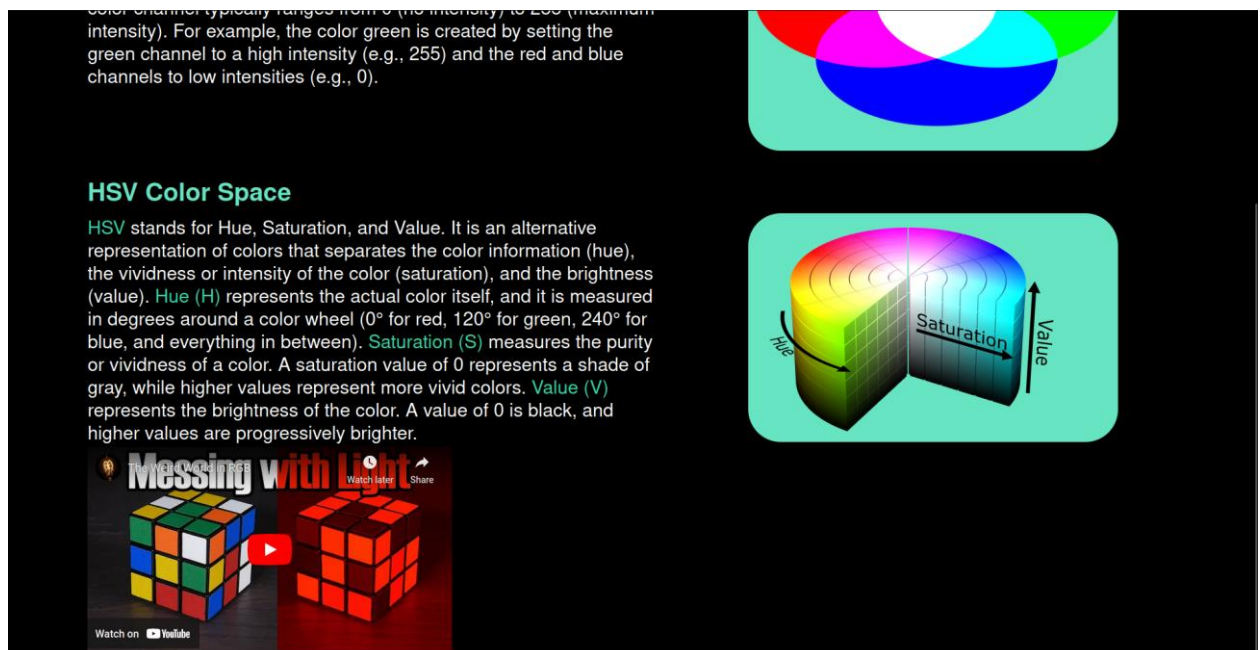


Рис. 17 Навчальне вікно до схем кольорів

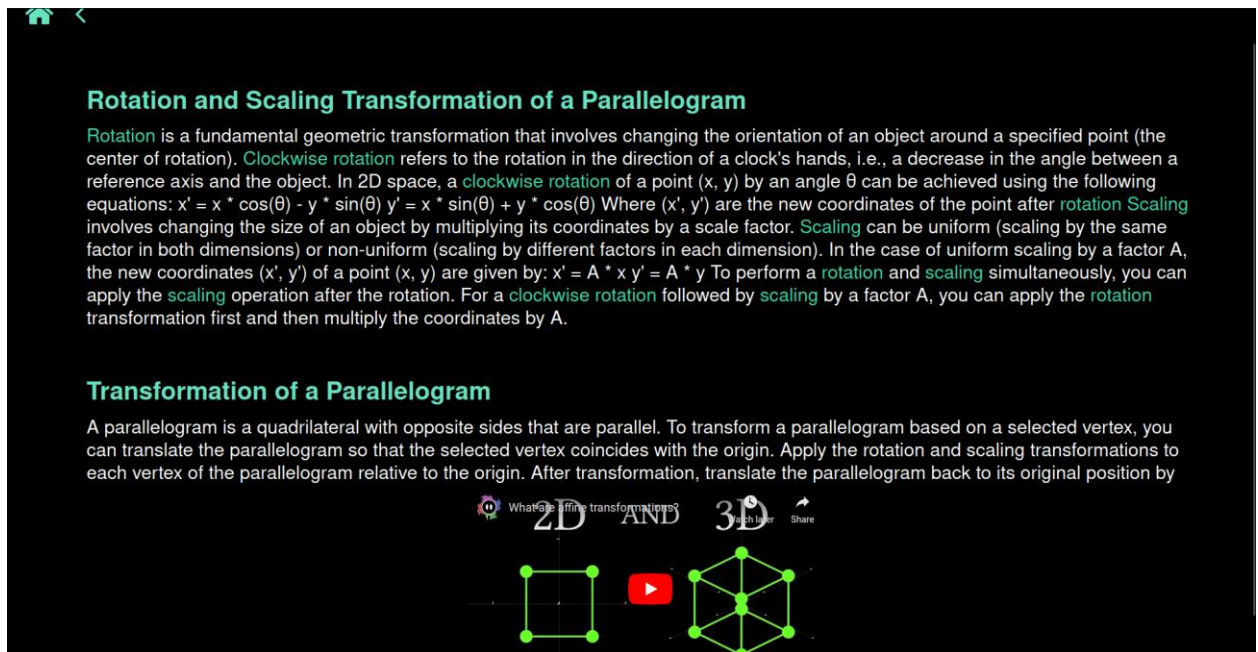


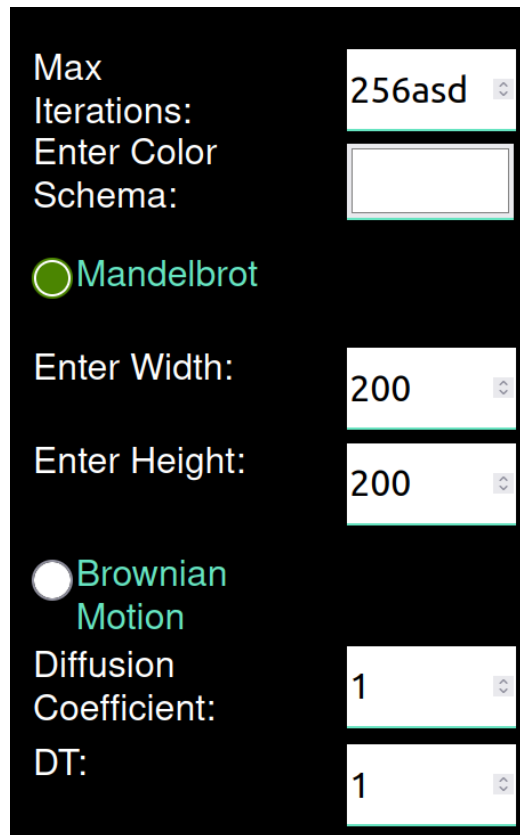
Рис. 18 Навчальне вікно до афінних перетворень

Деталізація навчальних функцій програми:

- 1) На вікні з навчальними матеріалами фракталів вказаний опис фракталу Мандельброта та Броунівського руху. Також до кожного з фракталів є фото додаток для кращого розуміння учнів. Якщо в учня з'явилося бажання поглибитись в темі, то надане відео, яке розширює погляд на фрактали.
- 2) На вікні з навчальними матеріалами кольорових схем вказаний опис моделей, а саме RGB та HSV. Вони дають змогу учневі зрозуміти як працюють колірні схеми, у чому головна різниця та побачити зображення, які характеризують відповідні моделі. Також при бажанні учень може переглянути відео, які більш детально описує RGB, яке демонструє цікаві зміни кольорів в залежності від освітлення.
- 3) На вікні з навчальними матеріалами афінних перетворень, описано як працює ротація та деформація. Також наданий параграф, який описує трансформацію паралелограма. У кінці вікна надане відео для поглибленого вивчення теми афінних перетворень.
- 4) Після вивчення матеріалу на головній сторінці є кнопка, яка дає змогу пройти тест. Це дає змогу учням перевірити наскільки добре вони вивчили матеріал.

4.7. Реагування помилки на неправильне введення даних

Кожному полю для введення даних було надане базове значення, щоб уникнути пустих полів зі сторони користувача.

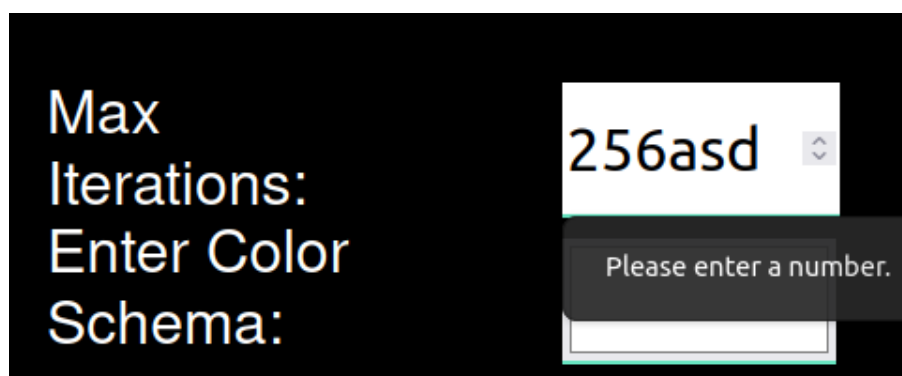


The screenshot shows a web form with the following fields and values:

- Max Iterations: 256asd (text input)
- Enter Color Schema: (empty text input)
- ☒ Mandelbrot (radio button)
- Enter Width: 200 (text input)
- Enter Height: 200 (text input)
- ☐ Brownian Motion (radio button)
- Diffusion Coefficient: 1 (text input)
- DT: 1 (text input)

Рис. 19. Дані за замовчуванням

Кожному полю для введення даних було створення з відповідним типом (number, color, radio button, combobox, і т. д.) щоб уникнути введення даних не правильного типу



The screenshot shows the same web form as in Figure 19, but with an error message displayed below the 'Max Iterations' field:

Please enter a number.

Рис. 20. Захист від неправильного формату даних

Кожному полю для введення даних було надане мінімальне та максимальне значення, щоб уникнути недопустимих значень зі сторони користувача.

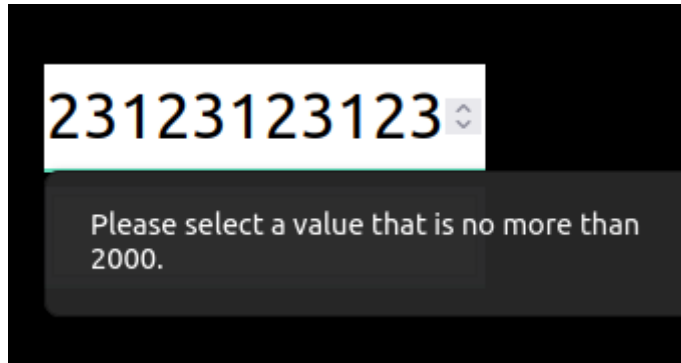


Рис. 21. Захист від недопустимих значень значення (вище максимальної границі)

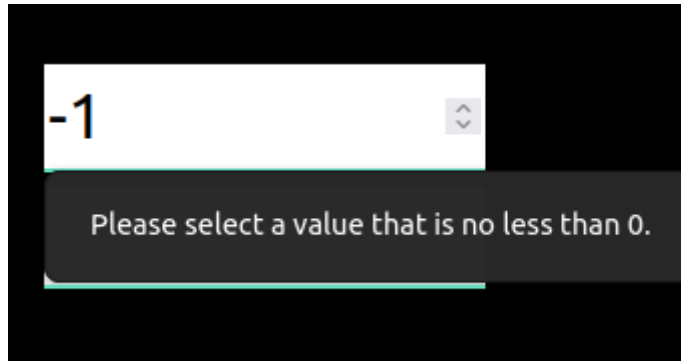


Рис. 22. Захист від недопустимих значень значення (нище мінімальної границі)

4.8. Текст програми з коментарями

Brownian_motion.py

```
import numpy as np

import matplotlib.pyplot as plt

class BrownianMotion():
    def __init__(self, num_steps: int, color: str,
dt: int, diffusion_coefficient: int) -> None:
        self.num_steps = num_steps
        self.dt = dt / 10
        # This parameter represents the time step size.
        # It determines the intervals at which
        # the position of the particle is updated in the simulation.
        self.diffusion_coefficient = diffusion_coefficient / 10
        # This parameter represents the diffusion coefficient,
        # which is a measure of how particles spread out over time.
        self.positions_x = np.zeros(self.num_steps)
        self.positions_y = np.zeros(num_steps)
        self.color = color

    def __brownian_motion(self) -> None:
        for i in range(1, self.num_steps):
            # Calculate the variance of the normal distribution
            displacement_x = np.random.normal(
0, np.sqrt(2 * self.diffusion_coefficient * self.dt))
            displacement_y = np.random.normal(
0, np.sqrt(2 * self.diffusion_coefficient * self.dt))

            self.positions_x[i] = self.positions_x[i - 1] + displacement_x
            self.positions_y[i] = self.positions_y[i - 1] + displacement_y

    def show_graph(self) -> None:
        self.__brownian_motion()
        plt.figure(figsize=(8, 6))
        plt.plot(self.positions_x,
self.positions_y,
label='Brownian Motion',
color=self.color)
        plt.title('Brownian Motion')
        plt.xlabel('X Position')
        plt.ylabel('Y Position')
        plt.legend()
        plt.grid(True)

plt.show()
```

Mandelbrot.py

```
import numpy as np

import matplotlib.pyplot as plt

class Mandelbrot:
    def __init__(self, max_iter: int, width: int,
height: int, color: tuple) -> None:
        self.max_iter = max_iter
        self.width, self.height = width, height
        self.xmin, self.xmax = -2.0, 1.0
        self.ymin, self.ymax = -1.5, 1.5
        self.color = color

    def __mandelbrot(self, c: complex) -> int:
        z = 0
        n = 0
        while abs(z) <= 2 and n < self.max_iter:
            z = z * z + c
            n += 1
        return n

    def __generate_graph(self) -> None:
        img = np.zeros((self.height, self.width, 3), dtype=np.uint8)
        for x in range(self.width):
            for y in range(self.height):
                # linear interpolation formula
                real = (self.xmin + (self.xmax - self.xmin)
                    * x / (self.width - 1))
                imag = (self.ymin + (self.ymax - self.ymin)
                    * y / (self.height - 1))
                c = complex(real, imag)
                color = self.__mandelbrot(c)
                img[y, x] = self.color if color == self.max_iter else (0, 0, 0)
        return img

    def show_graph(self) -> None:
        img = self.__generate_graph()
        plt.imshow(img, extent=(self.xmin, self.xmax, self.ymin, self.ymax))
        plt.title('Mandelbrot Fractal')
        plt.xlabel('Real')
        plt.ylabel('Imaginary')
        # manager = plt.get_current_fig_manager()
        # manager.window.wm_geometry("+880+380")
        # manager.resize(975, 550)
        plt.show()
```

rgbToHsv.js

```
function rgbToHsv(r, g, b) {

    // Ділимо значення R, G, B на 255
    r /= 255;
    g /= 255;
    b /= 255;

    // Обчислюємо cmax (максимальне з трьох значень), cmin (мінімальне з трьох значень) та різницю між ними
    var cmax = Math.max(r, g, b);
    var cmin = Math.min(r, g, b);
    var diff = cmax - cmin;
    var h = 0; // Відтінок
    var s = 0; // Насиченість

    // Обчислення відтінку (Hue)
    if (diff === 0) {
        h = 0; // Якщо різниця 0, відтінок буде 0 (відтінок не визначено)
    } else if (cmax === r) {
        h = (60 * ((g - b) / diff) + 360) % 360;
    } else if (cmax === g) {
        h = (60 * ((b - r) / diff) + 120) % 360;
    } else if (cmax === b) {
        h = (60 * ((r - g) / diff) + 240) % 360;
    }

    // Обчислення насиченості (Saturation)
    if (cmax === 0) {
        s = 0; // Якщо cmax 0, то насиченість 0 (абсолютно ненасичений колір)
    } else {
        s = (diff / cmax) * 100;
    }

    // Обчислення значення (Value)
    var v = cmax * 100;
    // Повертаємо результати у форматі HSV
    return [h, s, v];
}
```

HsvToRgb.js

```
function hsvToRgb(h, s, v) {  
  // Перевірка на вхідні значення h, s, v (в діапазоні [0, 360], [0, 100], [0, 100])  
  h = (h % 360 + 360) % 360; // Гарантуємо, що h знаходиться у діапазоні [0, 360]  
  s = Math.min(100, Math.max(0, s)); // Обмежуємо s в діапазоні [0, 100]  
  v = Math.min(100, Math.max(0, v)); // Обмежуємо v в діапазоні [0, 100]  
  
  // Нормалізуємо значення s та v до діапазону [0, 1]  
  s /= 100;  
  v /= 100;  
  
  var c = v * s; // Обчислюємо Chroma (насиченість), яка показує, наскільки сильним є колір  
  var x = c * (1 - Math.abs((h / 60) % 2 - 1)); // Обчислюємо x  
  var m = v - c; // Обчислюємо зсув яскравості (це кількість білого кольору, яку потрібно додати, щоб  
  отримати потрібну яскравість)  
  var r, g, b;  
  
  // Визначаємо, в якому секторі кольорового колеса перебуваємо та обчислюємо значення R, G і B  
  if (h >= 0 && h < 60) {  
    r = c;  
    g = x;  
    b = 0;  
  } else if (h >= 60 && h < 120) {  
    r = x;  
    g = c;  
    b = 0;  
  } else if (h >= 120 && h < 180) {  
    r = 0;  
    g = c;  
    b = x;  
  } else if (h >= 180 && h < 240) {  
    r = 0;  
    g = x;  
    b = c;  
  } else if (h >= 240 && h < 300) {  
    r = x;  
    g = 0;  
    b = c;  
  } else {  
    r = c;  
    g = 0;  
    b = x;  
  }  
  // Перетворюємо значення R, G і B у діапазон [0, 255] та повертаємо їх у форматі RGB  
  r = Math.round((r + m) * 255);  
  g = Math.round((g + m) * 255);  
  b = Math.round((b + m) * 255);  
}
```

```

return [r, g, b];
}

```

affine_transformation.py

```

import numpy as np

import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
from matplotlib.animation import FuncAnimation
from matplotlib.transforms import Affine2D
from matplotlib.widgets import Button

def find_fourth_point(points):
    p1, p2, p3 = points
    vector_B = [p3[0] - p2[0], p3[1] - p2[1]]
    p4 = [p1[0] + vector_B[0], p1[1] + vector_B[1]]
    return p4

def apply_affine_transform(matrix, points):
    transformed_points = []
    for x, y in points:
        point = np.array([x, y, 1]) # Convert to column vector
        transformed_point = matrix @ point
        transformed_points.append((transformed_point[0, 0], transformed_point[1, 0]))
    return transformed_points

def update_animation(frame, poly, points, vertex, scale):
    theta = np.radians(-frame)
    vertices = np.array(points)
    rotation_vertex = vertices[vertex]

    scale_factor = 1 - frame * scale / 360.0

    transform_matrix = Affine2D()
    transform_matrix.rotate_deg_around(rotation_vertex[0], rotation_vertex[1], np.degrees(theta))
    transform_matrix.scale(scale_factor, scale_factor)

    # Apply affine transformation to all vertices
    transformed_vertices = apply_affine_transform(transform_matrix.get_matrix(), vertices)

    # Update the xy data of the existing Polygon
    poly.set_xy(transformed_vertices)

def toggle_animation(event, anim):
    global running
    if running:
        anim.event_source.stop()
    else:
        anim.event_source.start()
    running = not running

```

```

def animate_rotation(points, vertex, scale):
    global running
    running = True
    fig, ax = plt.subplots()
    ax.grid(True)
    points.append(find_fourth_point(points))

    # Create an initial Polygon
    poly = Polygon(points, closed=True, edgecolor='b')
    ax.add_patch(poly)

    anim = FuncAnimation(fig, update_animation, fargs=(poly, points, vertex, scale),
        frames=np.arange(0, 360, 1), interval=50)

    # Add a button to start/stop the animation
    ax_button = plt.axes([0.81, 0.01, 0.1, 0.04])
    button = Button(ax_button, 'Start/Stop', color='lightgoldenrodyellow',
        hovercolor='0.975')
    button.on_clicked(lambda event: toggle_animation(event, anim))

    # Adjust the limits to zoom out
    ax.set_xlim(-3, 5)
    ax.set_ylim(-3, 3)

    plt.show()

```

ВИСНОВКИ

На лабораторних роботах 1-4 я ознайомився з деякими аспектами комп'ютерної графіки.

Під час виконання лабораторної роботи №1 я навчився розробляти Wireflow до ще не розробленого програмного забезпечення, користуючись вимогами до нього. Для побудови Wireframes я використовував програму Figma, яка надає всі необхідні засоби для побудови Wireflow. При створенні UI було дотримано такі евристичні норми як: послідовість і стандарти, допомога у виявленні та виправленні помилок, відповідність між системою та реальним світом, впізнавання, а не вгадування і контроль та свобода дій користувача, естетичний та мінімалістичний дизайн, запобігання помилок.

Під час виконання лабораторної роботи №2 я навчився графічно відображати попередньо побудовані фрактали. Для побудови фракталів я використовував геометричний метод, який базується на простих геометричних операціях.

Під час виконання лабораторної роботи №3 я навчився працювати з кольірними моделями RGB та HSV. Згідно з варіантом, я змінював насиченість зеленого кольору та переводу зображення з RGB в HSV і навпаки.

Під час виконання лабораторної роботи №4 я навчився виконувати афінні перетворення, попередньо побудованих фігур. Основною фігурою був паралелограм.

Для виконання лабораторних робіт 2-4 було використано середовище розробки FastAPI, Matplotlib, numpy, cv2 та мову програмування python, js.

При виконанні лабораторних робіт було чітко відображати результати серверу на фронтенті, оскільки було використано додакове вікно. Не вдалось зробити сервер мультипоточним через специфіку мови програмування та не вдалось зробити сервер асинхронним через нестачу часу.

Лабораторні роботи було вирішино виконувати самому, оскільки при командній роботі ти постійно залежиш від результату іншої людини, і не можеш повністю контролювати процес.

Список використаних джерел

- 1) МЕТОДИЧНІ ІНСТРУКЦІЇ ДЛЯ ВИКОНАННЯ ЛАБОРАТОРНОГО ПРАКТИКУМУ з дисципліни «Комп'ютерна графіка» для студентів 3 курсу спеціальності 121 «Інженерія програмного забезпечення» / 2023 / Левус Є.В.
- 2) Лекції з дисципліни «Комп'ютерна графіка» для студентів 3 курсу спеціальності 121 «Інженерія програмного забезпечення» / Левус Є.В.
- 3) (Не)поведінка ринків: фрактальний погляд на ризик, руйнування та винагороду / 2006 / Бенуа Мандельброт та Річард Л. Гудзон