**МІНІСТЕРСТВО ОСВІТИ І НАУКИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМ.ІГОРЯ СІКОРСЬКОГО»**

**НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ**

Проєктна робота

«Стандартизована ітераційна генерація правильних багатокутників, кривих і фракталів та стандартизоване подання даної генерації у точковому, растровому та векторному вигляді»

Виконали:  
Студенти 2 курсу  
Групи ФІ-21:  
Голуб Михайло,  
Кияшко Дарина,  
Климентьєв Максим

Перевірив:  
Хайдуров В. В.

Київ 2024

ЗМІСТ

[1. ЗАВДАННЯ 3](#_Toc169512607)

[2. ХІД РОБОТИ 4](#_Toc169512608)

[2.1. РОЗРОБКА УНІФІКОВАНОЇ СИСТЕМИ СТВОРЕННЯ ФІГУР 4](#_Toc169512609)

# ЗАВДАННЯ

Розробити уніфіковану систему створення фігур, таких як фрактали, криві та правильні багатокутники, за заданими параметрами. Розробити програмне забезпечення для побудови та виведення на екран фігур.

Презентувати роботу, сформувати звіт з проєктної роботи.

# ХІД РОБОТИ

### РОЗРОБКА УНІФІКОВАНОЇ СИСТЕМИ СТВОРЕННЯ ФІГУР

Було вирішено перед початком розробки програмного забезпечення чітко визначити структуру класів, їх методи та змінні.

Було вирішено розділити код на дві частини: генеративну та візуальну. Генеративна частина буде створювати масив точок з отриманих параметрів, а візуальна буде відображати масив на екрані

Створено клас-лекало Figure в підкаталозі Figures:

class Figure:  
 def \_\_init\_\_(self, \*args):  
 pass  
  
 def check\_args(self, \*args):  
 pass  
  
 def generate\_points(self, iteration):  
 pass

Визначено призначення та аргументи методів:

* \_\_init\_\_(self, \*args) приймає усі константи потрібні для побудови фігури;
* check\_args(self, \*args) перевіряє отримані константи під час ініціалізації класу;
* generate\_points(self, iteration) повертає масив точок фігури згідно вказаної ітерації: для фракталів iteration застосовуються за прямим призначенням, у інших фігурах iteration теж змінює результат, але іншим чином.

Створено клас-лекало FigureBuilder в кореневому каталозі для швидкої ініціалізації та використання Figure:

class FigureBuilder:  
 def build(self, figure: Figure, \*args):  
 return figure.generate\_points(\*args)

Створено клас Window в кореневому каталозі для відображення отриманого від Figure.generate\_points масиву точок:

import matplotlib.animation as animation  
  
  
class Window:  
 def draw(self, input\_value, \*\*kwargs):  
 multiplayer = kwargs.get("multiplayer", 1) \* kwargs.get("multi", 1)  
 interval = kwargs.get("interval", 30)  
 markersize = kwargs.get("markersize", 0.6)  
 figsize = kwargs.get("figsize", (5, 5))  
 fps = kwargs.get("fps", 15)  
  
 cmap = kwargs.get("cmap", 'gray') *# 'inferno'* animation\_need = kwargs.get("animation\_need", False)  
 animation\_save = kwargs.get("animation\_save", False)  
  
 is\_edge = kwargs.get("is\_edge", False)  
 is\_fixed\_size = kwargs.get("is\_fixed\_size", False)  
  
 has\_axes = kwargs.get("has\_axes", True)  
 has\_background = kwargs.get("has\_background", True)  
  
 if animation\_save:  
 if not os.path.isdir("images"):  
 os.mkdir("images")  
 add\_to\_name = len(os.listdir("./images"))  
 filename = kwargs.get('filename', 'figure' + str(add\_to\_name))  
  
 if is\_edge:  
 linestyle = '-'  
 else:  
 linestyle = ''  
  
 try:  
 x, y = input\_value  
 is\_matrix = False  
 except ValueError:  
 is\_matrix = True  
  
 if is\_matrix:  
 if not isinstance(input\_value, list):  
 plt.imshow(input\_value, cmap=cmap)  
 elif not animation\_need:  
 plt.imshow(input\_value[-1], cmap=cmap)  
 else:  
 fig, ax = plt.subplots(figsize=figsize, constrained\_layout=(not has\_background))  
  
 ax.imshow(input\_value[0], cmap=cmap)  
 if not has\_axes:  
 ax.axis('off')  
  
 def update(frame):  
 ax.clear()  
 ax.imshow(input\_value[frame], cmap=cmap)  
 if not has\_axes:  
 ax.axis('off')  
  
 ani = animation.FuncAnimation(fig=fig, func=update, frames=len(input\_value), interval=interval)  
 else:  
 if is\_fixed\_size:  
 x\_limit\_left = min(x) - abs(min(x)) / 2  
 x\_limit\_right = max(x) + abs(max(x)) / 2  
 y\_limit\_bottom = min(y) - abs(min(y)) / 2  
 y\_limit\_top = max(y) + abs(max(y)) / 2  
 if not animation\_need:  
 plt.plot(x, y, marker='o', linestyle=linestyle, markersize=markersize)  
 else:  
 fig, ax = plt.subplots(figsize=figsize, constrained\_layout=(not has\_background))  
  
 if is\_fixed\_size:  
 ax.set\_xlim(x\_limit\_left, x\_limit\_right)  
 ax.set\_ylim(y\_limit\_bottom, y\_limit\_top)  
 if not has\_axes:  
 ax.axis('off')  
 ax.plot(x, y, marker='o', linestyle=linestyle, markersize=markersize)  
  
 def update(frame):  
 frame = frame \* multiplayer  
 ax.clear()  
 if is\_fixed\_size:  
 ax.set\_xlim(x\_limit\_left, x\_limit\_right)  
 ax.set\_ylim(y\_limit\_bottom, y\_limit\_top)  
 if not has\_axes:  
 ax.axis('off')  
 ax.plot(x[:frame], y[:frame], marker='o', linestyle=linestyle, markersize=markersize)  
  
 ani = animation.FuncAnimation(fig=fig, func=update, frames=len(x)//multiplayer+1, interval=interval)  
  
 if animation\_need and animation\_save:  
 ani.save("./images/" + filename + '.gif', writer=animation.PillowWriter(fps=fps))  
  
 plt.show()

Цей клас містить єдиний метод draw, що приймає масив точок в input\_value та інші опціональні аргументи, такі як:

* multiplayer – кількість точок, що виводяться кожного кадру анімації;
* interval – затримка між кадрами анімації побудови, що виводиться на екран;
* markersize – розмір крапок, які зображають точки;
* figsize – розмір вікна в умовних одиницях;
* fps – частота кадрів анімації побудови, що зберігається у файл .gif;
* cmap – палітра кольорів;
* animation\_need – вказує на те, чи потрібна анімація;
* animation\_save – вказує на те, чи потрібно зберігати анімацію;
* is\_edge – вказує на те, чи потрібно малювати ребра;
* is\_fixed\_size – вказує на те, чи потрібно зафіксувати розмір вікна під час анімації;
* has\_axes – вказує на те, чи потрібно відображати вісі;
* has\_background – вказує на те, чи потрібно відображати фон довкола побудованої фігури.

Створено клас-лекало Direcctor в кореневому каталозі для швидкої побудови і відображення побудованих об’єктів, використовуючи класи Window та FigureBuilder:

class FigureDirector:  
 def build(self, figure: Figure, \*\*kwargs):  
 if "it" in kwargs:  
 Window().draw(FigureBuilder().build(figure, kwargs['it']), \*\*kwargs)  
 elif "iterations" in kwargs:  
 Window().draw(FigureBuilder().build(figure, kwargs['iterations']), \*\*kwargs)  
 else:  
 Window().draw(FigureBuilder().build(figure), \*\*kwargs)

Цей клас під час ініціалізації отримує клас-лекало фігури, яку потрібно створити та усі необхідні аргументи для її створення.

Створено клас App для швидкої ініціалізації екземплярів класу Director:

class App:  
 def \_\_init\_\_(self):  
 self.figures = {  
 *# “name” : Figure.Figure*   
 }  
  
 def create\_figure(self, name, \*args, \*\*kwargs):  
 if name in self.figures:  
 FigureDirector().build(self.figures[name](\*args), \*\*kwargs)  
 else:  
 raise ValueError("Wrong fractal name")

Цей клас міститиме назви усіх класів-лекал фігур та посилання на них у змінній self.figures. Через метод create\_figure можна швидко створити та відобразити потрібну фігуру з потрібними параметрами.

### РЕАЛІЗАЦІЯ КЛАСІВ-ЛЕКАЛ ДЛЯ ФРАКТАЛІВ, ЩО ВИВЧАЛИСЬ ПРОТЯГОМ СЕМЕСТРУ

Було обрано фрактали, що вивчались протягом семестру, як найпростіші для реалізації, тож було вирішено їх реалізувати найпершими. До фракталів, що вивчались протягом семестру, входять:

* L-системи;
* Афінні;
* Двовимірна (матрична) множина Кантора.

### РЕАЛІЗАЦІЯ КЛАСУ-ЛЕКАЛА ДЛЯ ФРАКТАЛІВ L-СИСТЕМ

Фрактал L-системи це фрактал, що використовує текстовий рядок для ітерацій: символ F робить крок вперед на визначену довжину, +/- повертають напрямок кроку на визначений кут, усі інші символи використовуються для ітерування. Набір правил складається з пар (символ, рядок), під час ітерування кожне правило замінює кожен символ з пари на рядок.

На базі класа Figure було реалізовано клас-лекало для фракталів L-систем:

import numpy as np  
class LsystemFractal:  
 *"""  
 Lsystem implementation of fractals (see compgraph Lab3)  
 Only radians  
 """* def \_\_init\_\_(self, axiom: str, rules: dict, fi: float, dfi: float, \*args):  
 *"""  
 Initiates Lsystem fractal with given parameters but before checks if parameters are correct  
  
 # Parameters:  
 axiom: string (starting L-axiom)  
 rules: dict (rules for how to change each letter (not specific symbol) in iteration)  
 max\_iterations: int (how many iterations)  
 fi: float (starting angular) (now only radians)  
 dfi: float (angular velocity) (now only radians)  
 """* if args != ():  
 raise ValueError(f"Wrong number of arguments, {args} excess")  
 self.list\_to\_check = [str, dict, float, float] *# Change if count of arguments changes* self.check\_args(axiom, rules, fi, dfi)  
  
 self.axiom = axiom  
 self.rules = rules  
 self.fi = fi  
 self.dfi = dfi  
  
 def check\_args(self, \*args):  
 *"""  
 Checks if parameters are correct  
  
 if not - raises ValueError with appropriate message  
 """* for argument\_index in range(len(args)):  
 if type(args[argument\_index]) is not self.list\_to\_check[argument\_index]:  
 raise ValueError(f"Wrong argument {args[argument\_index]}, which is {type(args[argument\_index])} type, expected {self.list\_to\_check[argument\_index]} type")  
  
 def generate\_points(self, iteration):  
 *"""  
 Generates specified iteration of Lsystem fractal  
  
 # Returns:  
 (N+1 shape, N+1 shape) arrays of x and y coordinates  
 """* result = self.axiom  
 for iteration in range(iteration):  
 new\_axiom = ''  
 for word\_place in range(len(result)):  
 if result[word\_place] in self.rules.keys():  
 new\_axiom += self.rules[result[word\_place]]  
 else:  
 new\_axiom += result[word\_place]  
 result = new\_axiom  
  
 N = len(result)  
 L = 2  
 x = np.zeros(N+1)  
 y = np.zeros(N+1)  
 for i in range(N):  
 x[i+1] = x[i]  
 y[i+1] = y[i]  
 if result[i] == 'F':  
 x[i+1] += L\*np.cos(self.fi)  
 y[i+1] += L\*np.sin(self.fi)  
 elif result[i] == '+':  
 self.fi += self.dfi  
 elif result[i] == '-':  
 self.fi -= self.dfi  
 return x, y

Метод \_\_init\_\_ приймає початковий рядок axiom, словник правил rules, початковий напрямок кроку fi та дельту зміни напрямку кроку dfi. Після чого йде перевірка на відсутність зайвих аргументів та правильність типів введених аргументів. Після успішного проходження перевірки, отримані аргументи записуються як змінні класу. У разі неуспішного проходження перевірки, виконання коду зупиняється та викликається відповідна помилка.

Метод generate\_points приймає номер ітерації iteration яку потрібно згенерувати. Метод ітерує рядок iteration разів, після чого обчислює розміщення точок отриманого фракталу на координатній площині та повертає його.