МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА

ІНСТИТУТ КОМП’ЮТЕРНИХ НАУК ТА ІНОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра ІСМ



ЗВІТ

Про виконання лабораторної роботи №3

«Розробка ASCII ART генератора для візуалізації текстових даних»

З дисципліни

«Спеціалізовані мови програмування»

Студентки групи РІ – 31

Нащецької Яни Вікторівни

Прийняв викладач

Щербак С.С.

**Лабораторна робота № 3. Розробка ASCII ART генератора для візуалізації текстових даних**

**Мета:** створення додатка Генератора ASCII-арту.

**План роботи**

Завдання 1: Введення користувача

Створіть Python-програму, яка приймає введення користувача для слова або фрази, яку треба перетворити в ASCII-арт.

Завдання 2: Бібліотека ASCII-арту

Інтегруйте бібліотеку ASCII-арту (наприклад, pyfiglet або art) у вашу програму для генерації ASCII-арту з введення користувача

Завдання 3: Вибір шрифту

Дозвольте користувачам вибирати різні стилі шрифтів для свого ASCII-арту. Надайте список доступних шрифтів та дозвольте їм вибрати один.

Завдання 4: Колір тексту

Реалізуйте опцію вибору користувачем кольору тексту для їхнього ASCII-арту. Підтримуйте основний вибір кольорів (наприклад, червоний, синій, зелений).

Завдання 5: Форматування виводу

Переконайтеся, що створений ASCII-арт правильно відформатований та вирівнюється на екрані для зручності читання.

Завдання 6: Збереження у файл

Додайте функціональність для збереження створеного ASCII-арту у текстовому файлі, щоб користувачі могли легко завантажувати та обмінюватися своїми творіннями.

Завдання 7: Розмір ARTу

Дозвольте користувачам вказувати розмір (ширина і висота) ASCII-арту, який вони хочуть створити. Масштабуйте текст відповідно.

Завдання 8: Вибір символів

Дозвольте користувачам вибирати символи, які вони хочуть використовувати для створення ASCII-арту (наприклад, '@', '#', '\*', тощо).

Завдання 9: Функція попереднього перегляду

Реалізуйте функцію попереднього перегляду, яка показує користувачам попередній перегляд їхнього ASCII-арту перед остаточним збереженням.

Завдання 10: Інтерфейс, зрозумілий для користувача

Створіть зручний для користувача інтерфейс командного рядку для додатка, щоб зробити його інтуїтивно зрозумілим та легким у використанні.

**Основний код програми:**

from shared.classes.art\_generator.console\_reader import ConsoleReader

from shared.classes.art\_generator.console\_writer import ConsoleWriter

from shared.classes.art\_generator.ascii\_art\_generator import ArtGenerator

from shared.functions import manage\_app\_settings

from shared.functions.ui\_functions.menu\_functions import print\_menu

from shared.constants.global\_variables import art\_menu

class ArtConsole:

    def \_\_init\_\_(self):

        self.console\_writer = ConsoleWriter()

        self.console\_reader = ConsoleReader()

        self.art\_generator = ArtGenerator()

    def run\_console(self, art\_type):

        while True:

            print\_menu(manage\_app\_settings.console\_color, art\_menu)

            answer = self.console\_reader.input\_data("Оберіть варіант меню")

# from shared.libs.font\_library import get\_ascii\_font

class ArtGenerator:

    def \_\_init\_\_(self, art\_type, symbol="\*", width=40, height=10, alignment="center", color\_mode="bw"):

        self.type = art\_type

        self.symbol = symbol

        self.width = width

        self.height = height

        self.alignment = alignment

        self.color\_mode = color\_mode

    def generate(self):

        raise NotImplementedError("Subclasses must implement the generate method.")

import math

from shared.classes.art\_generator.ascii\_art\_generator import ArtGenerator

class Shape3D:

    def \_\_init\_\_(self, size=5):

        self.size = size

    def draw(self):

        raise NotImplementedError("Subclasses should implement this!")

class Cube(Shape3D):

    def \_\_init\_\_(self, size=5, symbol="#"):

        super().\_\_init\_\_(size)

        self.symbol = symbol

    def draw(self):

        width = self.size \* 2

        height = self.size + 5

        return draw\_cube(width, height, self.symbol)

def draw\_cube(width, height, symbol):

    cube = [[" "] \* width for \_ in range(height)]

    vertices = {

        "tc": (width // 2, 0),

        "tl": (0, int(0.25 \* height)),

        "tr": (width - 1, int(0.25 \* height)),

        "cc": (width // 2, int(0.5 \* height)),

        "bl": (0, int(0.75 \* height)),

        "br": (width - 1, int(0.75 \* height)),

        "bc": (width // 2, height - 1),

    }

    edges = (

        ("tc", "tl"),

        ("tc", "tr"),

        ("tl", "cc"),

        ("tr", "cc"),

        ("bl", "bc"),

        ("br", "bc"),

        ("cc", "bc"),

    )

    for edge in edges:

        v1 = vertices[edge[0]]

        v2 = vertices[edge[1]]

        x1, y1 = v1

        x2, y2 = v2

        if x1 > x2:

            x1, x2 = x2, x1

            y1, y2 = y2, y1

        try:

            m = (y2 - y1) / (x2 - x1)

        except ZeroDivisionError:

            for yy in range(min(y1, y2), max(y1, y2) + 1):

                cube[yy][x1] = symbol

        else:

            yy = y1

            for xx in range(x1, x2 + 1):

                cube[int(round(yy))][xx] = symbol

                yy += m

    x1, y1 = vertices["tl"]

    x2, y2 = vertices["bl"]

    for yy in range(min(y1, y2), max(y1, y2) + 1):

        cube[yy][x1] = "|"

    x1, y1 = vertices["tr"]

    x2, y2 = vertices["br"]

    for yy in range(min(y1, y2), max(y1, y2) + 1):

        cube[yy][x1] = "|"

    for xx in range(vertices["tl"][0], vertices["tr"][0] + 1):

        cube[vertices["tc"][1]][xx] = symbol

        cube[vertices["bc"][1]][xx] = symbol

    cube\_str = "\n".join("".join(row) for row in cube)

    return cube\_str

class Sphere(Shape3D):

    def \_\_init\_\_(self, size, symbol="\*", shadow\_symbol=".", shadow\_offset=2):

        super().\_\_init\_\_(size)

        self.symbol = symbol

        self.shadow\_symbol = shadow\_symbol

        self.shadow\_offset = shadow\_offset

    def draw(self):

        output = []

        for i in range(-self.size, self.size + 1):

            row = []

            for j in range(-self.size, self.size + 1):

                distance = math.sqrt(i\*\*2 + j\*\*2)

                if distance <= self.size:

                    shadow\_intensity = int((distance / self.size) \* 10)

                    if shadow\_intensity < 4:

                        row.append(self.symbol)

                    else:

                        if j > 0:

                            row.append(self.shadow\_symbol)

                        else:

                            row.append(self.shadow\_symbol)

                else:

                    row.append(" ")

            output.append("".join(row).rstrip())

        return "\n".join(output)

class Art3DGenerator(ArtGenerator):

    def \_\_init\_\_(

        self, shape: Shape3D, symbol="\*", width=40, height=10, color\_mode="bw"

    ):

        super().\_\_init\_\_("", symbol, width, height, "center", color\_mode)

        self.shape = shape

    def apply\_color(self, text):

        if self.color\_mode == "gray":

            return "\033[90m" + text + "\033[0m"

        return text

    def generate\_3d\_art(self):

        ascii\_art = self.apply\_color(ascii\_art)

        ascii\_art = self.shape.draw()

        self.text = ascii\_art

        return ascii\_art

**Висновок:** Виконуючи ці завдання, я створила універсальний Генератор ASCII-арту, який дозволяє користувачам налаштовувати свої творіння з різними шрифтами, кольорами, розмірами та символами. Проект надає практичний досвід роботи з введенням користувача, зовнішніми бібліотеками, роботою з файлами та дизайном інтерфейсу користувача в Python.