МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Кафедра інформаційних систем та мереж

Лабораторна робота №4

з дисципліни

«Спеціалізовані мови програмування»

на тему

«Розробка ASCII ART генератора для візуалізації 2D-фігур»

Виконав:

ст. гр. РІ-32

Коготюк В.О

Прийняв:

доц. каф. ІСМ

Сергій ЩЕРБАК

Львів - 2024

**Мета**

Створення Генератора ASCII-арту без використання зовнішніх бібліотек

**Хід виконання роботи**

**Завдання 1: Введення користувача**

Створено програму Python, яка отримує введення користувача щодо слова або фрази, яку вони хочуть перетворити в ASCII-арт.

**Завдання 2: Набір символів**

Визначено набір символів (наприклад, '@', '#', '\*', тощо), які будуть використовуватися для створення ASCII-арту. Ці символи будуть відображати різні відтінки.

**Завдання 3: Розміри Art-у**

Реалізовано функціонал запиту у користувача розміри (ширина і висота) ASCII-арту, який вони хочуть створити. Розміри в межах керованого діапазону.

**Завдання 4: Функція генерації Art-у**

Є функція, яка генерує ASCII-арт на основі введення користувача, набору символів та розмірів. Використовується введення користувача, щоб визначити, які символи використовувати для кожної позиції в Art-у.

**Завдання 5: Вирівнювання тексту**

Реалізовано опції вирівнювання тексту (ліво, центр, право), щоб користувачі могли вибирати, як їх ASCII-арт розміщується на екрані.

**Завдання 6: Відображення мистецтва**

Створений ASCII-арт відображається на екрані за допомогою стандартних функцій друку Python.

**Завдання 7: Збереження у файл**

Додано можливість зберігати створений ASCII-арт у текстовий файл, щоб користувачі могли легко завантажувати та обмінюватися своїми творіннями.

**Завдання 8: Варіанти кольорів**

Користувачі можуть вибирати опції кольорів (чорно-білий, відтінки сірого) для свого ASCII-арту.

**Завдання 9: Функція попереднього перегляду**

Реалізовано функцію попереднього перегляду, яка показує користувачам попередній перегляд їх ASCII-арту перед остаточним збереженням

**Завдання 10: Інтерфейс, зрозумілий для користувача**

Створено інтерфейс для користувача у командному рядку, щоб зробити програму легкою та інтуїтивно зрозумілою для використання.

**Код програмного продукту**

from shared.classes.AsciiGenerator import AsciiGenerator

from labs.lab4.bll.base\_font import base\_font

class CustomGenerator(AsciiGenerator):

"""

Class representing a custom ASCII generator.

class CustomGenerator(AsciiGenerator):

Attributes:

name\_font (dict): A copy of the base font with some modifications.

fonts (dict): A dictionary of fonts, including the base font and the modified name font.

Methods:

generate(cls, data, font='cap\_font', width=80):

Generates ASCII art from the given data using the specified font.

string\_replace(cls, string, replace\_string):

Replaces all non-space and non-newline characters in the string with characters from the replace\_string.

is\_font\_break\_lines(cls, font):

Determines if the specified font breaks lines.

get\_fonts(cls):

Returns a list of available font names.

get\_font(cls, font\_name):

Retrieves the font dictionary for the specified font name. Raises a ValueError if the font is not found.

get\_font\_char\_height(cls, font\_name):

Retrieves the character height of the specified font.

get\_font\_char\_width(cls, font\_name):

Retrieves the character width of the specified font.

"""

name\_font = base\_font.copy()

name\_font["replace\_string"] = "oleh"

fonts = {"cap": base\_font, "oleh\_cap": name\_font}

@classmethod

def generate(cls, data, font="cap\_font", width=80):

char\_width = cls.get\_font\_char\_width(font)

char\_height = cls.get\_font\_char\_height(font)

chars\_in\_line = width // char\_width

line\_result = []

font\_data = cls.get\_font(font)

for i in range(0, len(data), chars\_in\_line):

line = data[i : i + chars\_in\_line].lower()

for row in range(char\_height):

rows\_result = []

for char in line:

if char in font\_data["symbols"]:

ascii\_char = font\_data["symbols"][char]

rows\_result.append(ascii\_char[row])

else:

rows\_result.append(" " \* char\_width)

line\_result.append("".join(rows\_result))

result = "\n".join(line\_result)

if "replace\_string" in font\_data:

replace\_string = font\_data["replace\_string"]

result = cls.string\_replace(result, replace\_string)

return result

@classmethod

def string\_replace(cls, string, replace\_string):

result = []

replace\_len = len(replace\_string)

replace\_index = 0

for char in string:

if char == " " or char == "\n":

result.append(char)

else:

if replace\_index >= replace\_len:

replace\_index = 0

replace\_char = replace\_string[replace\_index]

replace\_index += 1

result.append(replace\_char)

return "".join(result)

@classmethod

def is\_font\_break\_lines(cls, font):

font = cls.get\_font(font)

return font.get("is\_break\_lines", False)

@classmethod

def get\_fonts(cls):

return list(cls.fonts.keys())

@classmethod

def get\_font(cls, font\_name):

if font\_name not in cls.fonts:

raise ValueError(f"Font '{font\_name}' not found.")

return cls.fonts[font\_name]

@classmethod

def get\_font\_char\_height(cls, font\_name):

font = cls.get\_font(font\_name)

return font["height"]

@classmethod

def get\_font\_char\_width(cls, font\_name):

font = cls.get\_font(font\_name)

return font["width"]

from shared.interfaces.PaintTextInterface import PaintTextInterface

class CustomPainter(PaintTextInterface):

"""

CustomPainter class inheriting from PaintTextInterface

This class provides methods to paint text with different colors and to retrieve the list of supported colors.

color\_map:

A dictionary mapping color names to their respective ANSI escape codes.

Methods:

paint(cls, text, color):

Paints the given text with the specified color using ANSI escape codes.

Parameters:

text (str): The text to be painted.

color (str): The color in which to paint the text. Must be one of the keys in the color\_map dictionary.

Returns:

str: The painted text with ANSI escape codes.

Raises:

ValueError: If the specified color is not supported.

get\_colors(cls):

Retrieves the list of supported colors.

Returns:

list: A list of color names supported by the CustomPainter class.

"""

color\_map = {

"red": "\x1b[31m",

"green": "\x1b[32m",

"yellow": "\x1b[33m",

"blue": "\x1b[34m",

"magenta": "\x1b[35m",

"cyan": "\x1b[36m",

"white": "\x1b[97m",

"light\_gray": "\x1b[37m",

"dark\_gray": "\x1b[90m",

"black": "\x1b[30m",

"default": "\x1b[0m",

}

@classmethod

def paint(cls, text, color):

if color in cls.color\_map:

start\_color = cls.color\_map[color]

end\_color = cls.color\_map["default"]

return f"{start\_color}{text}{end\_color}"

else:

raise ValueError(f"Color '{color}' is not supported.")

@classmethod

def get\_colors(cls):

return list(cls.color\_map.keys())

**Висновки**

Я створив генератор ASCII-арту з нуля, надав можливість налаштовувати символи, розміри, вирівнювання та кольори. Я глибше розібрався, як створюється ASCII-арт