МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Кафедра інформаційних систем та мереж

Лабораторна робота №5

з дисципліни

«Спеціалізовані мови програмування»

на тему

«Розробка ASCII ART генератора для візуалізації 3D-фігур»

Виконав:

ст. гр. РІ-32

Олег ВОЛОДЬКО

Прийняв:

доц. каф. ІСМ

Сергій ЩЕРБАК

Львів - 2024

**Мета**

Cтворення додатка для малювання 3D-фігур у ASCII-арті на основі об’єктно - орієнтованого підходу та мови Python

**Хід виконання роботи**

**Завдання 1: Проектування класів**

Створено структуру класів для вашого генератора 3D ASCII-арту. Визначено основні компоненти, атрибути та методи, необхідні для програми.

**Завдання 2: Введення користувача**

Створено методи у межах класу для введення користувача та вказання 3D-фігури, яку вони хочуть намалювати, та її параметрів (наприклад, розмір, кольори).

**Завдання 3: Представлення фігури**

Визначено структури даних у межах класу для представлення 3D-фігури. Для зберігання даних про фігуру використовуються об’єкти класів.

**Завдання 4: Проектування з 3D в 2D**

Реалізовано метод, який перетворює 3D-представлення фігури у 2D-представлення, придатне для ASCII-арту.

**Завдання 5: Відображення ASCII-арту**

Створено у межах класу для відображення 2D-представлення 3D-фігури як ASCII-арту. Це включає відображення кольорів і форми за допомогою символів ASCII.

**Завдання 6: Інтерфейс, зрозумілий для користувача**

Створено зручний для користувача командний рядок або графічний інтерфейс користувача (GUI) за допомогою об'єктно-орієнтованих принципів, щоб дозволити користувачам спілкуватися з програмою.

**Завдання 7: Маніпуляція фігурою**

Створено методи для маніпулювання 3D-фігурою, такі масштабування або зміщення, щоб надавати користувачам контроль над її виглядом.

**Завдання 8: Варіанти кольорів**

Користувачі можуть вибирати варіанти кольорів для їхніх 3D ASCII-арт-фігур. Реалізовано методи для призначення кольорів різним частинам фігури.

**Завдання 9: Збереження та експорт**

Додано функціональність для зберігання згенерованого 3D ASCII-арту у текстовий файл.

**Завдання 10: Розширені функції**

Додано ефекти перспективи, для підвищення реалізму 3D ASCII-арту.

**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())

from OpenGL.GL import \*

from OpenGL.GLUT import \*

from OpenGL.GLU import \*

from PIL import Image

import numpy as np

class AsciiRenderer:

"""

AsciiRenderer:

A class that converts graphical scenes rendered using OpenGL into ASCII art.

\_\_init\_\_(self, width=None, height=None, ascii\_height=None, ascii\_chars=None, color\_palette=None)

Initialize the AsciiRenderer with desired attributes such as width, height, ASCII height, characters for ASCII representation, and an optional color palette.

set\_ascii\_height(self, ascii\_height)

Set the height (number of rows) for the ASCII representation.

map\_pixel\_to\_ascii(self, gray\_value, color\_value)

Map a pixel's grayscale value to a corresponding ASCII character and apply color using ANSI escape codes.

render\_to\_ascii(self)

Capture the current OpenGL framebuffer, process the image to scale and convert it to grayscale, then generate an ASCII art representation of the image with color data.

display(self, scene\_draw\_callback)

Clear the OpenGL buffers, draw the scene using the provided callback function, convert the output to ASCII art, and update the display.

reshape(self, width, height, fov, aspect, z\_near, z\_far)

Adjust the OpenGL viewport and projection matrix to handle window resizing and ensure a consistent field of view.

"""

def \_\_init\_\_(self, width=None, height=None, ascii\_height=None, ascii\_chars=None,

color\_palette=None):

self.width = width

self.height = height

self.ascii\_height = ascii\_height

self.ascii\_chars = ascii\_chars

self.color\_palette = color\_palette

def set\_ascii\_height(self, ascii\_height):

self.ascii\_height = ascii\_height

def map\_pixel\_to\_ascii(self, gray\_value, color\_value):

scale = gray\_value / 255

char = self.ascii\_chars[int(scale \* (len(self.ascii\_chars) - 1))]

r, g, b = color\_value

return f"\033[38;2;{r};{g};{b}m{char}\033[0m"

def render\_to\_ascii(self):

glReadBuffer(GL\_FRONT)

pixels = glReadPixels(0, 0, self.width, self.height, GL\_RGB, GL\_UNSIGNED\_BYTE)

image = Image.frombytes("RGB", (self.width, self.height), pixels)

height\_to\_width\_ratio = 3

scaled\_height = self.ascii\_height if self.ascii\_height else int(self.height \* 0.2)

scaled\_width = int(scaled\_height \* (self.width/self.height)\*height\_to\_width\_ratio) if self.height and self.width else int(

scaled\_height \* 3)

image = image.resize((scaled\_width, scaled\_height), Image.NEAREST)

grayscale\_image = image.convert('L')

color\_image = np.array(image)

pixel\_data = np.array(grayscale\_image)

ascii\_image = []

for row, color\_row in zip(pixel\_data[::-1], color\_image[::-1]):

ascii\_row = ''.join(map(self.map\_pixel\_to\_ascii, row, color\_row))

ascii\_image.append(ascii\_row)

return '\n'.join(ascii\_image)

def display(self, scene\_draw\_callback):

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)

glLoadIdentity()

scene\_draw\_callback()

ascii\_art = self.render\_to\_ascii()

glutSwapBuffers()

return ascii\_art

def reshape(self, width, height, fov, aspect, z\_near, z\_far):

if height < 70:

height = 70

glutReshapeWindow(width, height)

self.width = width

self.height = height

glViewport(0, 0, self.width, self.height)

glMatrixMode(GL\_PROJECTION)

glLoadIdentity()

gluPerspective(fov, aspect, z\_near, z\_far)

glMatrixMode(GL\_MODELVIEW)

**Висновки**

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