Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра ІСМ



Звіт

про виконання лабораторної роботи № 5

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

з дисципліни

«Спеціалізовані мови програмування»

Виконала:

Студентка групи ІТ-31,

Василик Ю. М.

Прийняв:

Щербак С.С

Львів

2024

**Мета роботи:** створення додатка для малювання 3D-фігур у ASCII-арті на основі об’єктно - орієнтованого підходу та мови Python

**План роботи:**

**Завдання 1: Проектування класів**

**Розробіть структуру класів для вашого генератора 3D ASCII-арту. Визначте основні компоненти, атрибути та методи, необхідні для програми.**

**Завдання 2: Введення користувача**

**Створіть 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**

**Визначте структури даних у межах класу для представлення 3D-фігури. Це може включати використання списків, матриць або інших структур даних для зберігання форми фігури та її властивостей.**

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

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

**Завдання 5: Відображення ASCII-арту**

**Напишіть метод у межах класу для відображення 2D-представлення 3D-фігури як ASCII-арту. Це може включати відображення кольорів і форми за допомогою символів ASCII.**

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

**Створіть 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**

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

**Завдання 8: Варіанти кольорів**

**Дозвольте користувачам вибирати варіанти кольорів для їхніх 3D ASCII-арт-фігур. Реалізуйте методи для призначення кольорів різним частинам фігури.**

**Завдання 9: Збереження та експорт**

**Додайте функціональність для зберігання згенерованого 3D ASCII-арту у текстовий файл**

**Завдання 10: Розширені функції**

**Розгляньте можливість додавання розширених функцій, таких як тінь, освітлення та ефекти перспективи, для підвищення реалізму 3D ASCII-арту.**

**Результат виконання:**

**Figure.py**

from abc import (

    ABC,

    abstractmethod,

)

import colorama

from colorama import (

    Fore,

)

colorama.init(autoreset=True)

# Create a dictionary to map color names to their corresponding codes

colors = dict(enumerate(sorted(Fore.\_\_dict\_\_.keys())))

# Define a function to display available colors

def display\_colors() -> None:

    for i in colors:

        print(str(i) + ". " + colors[i])

# Create an abstract base class 'Figure3D' that defines the common interface for 3D figures

class Figure3D(ABC):

    def \_\_init\_\_(self, character: str, color\_position: int):

        # Check if the specified color\_position is valid

        if not colors.\_\_contains\_\_(color\_position):

            raise ValueError("Color position must be in the range of available colors")

        # Check if the character is appropriate (only one character is allowed)

        elif self.is\_appropriate\_character(character) is False:

            raise ValueError(

                "Only one character is allowed as the figure's representation"

            )

        self.\_character = character

        self.\_color\_position = color\_position

    @abstractmethod

    def get\_2d\_representation(self) -> list:

        pass

    @abstractmethod

    def get\_3d\_representation(self, scale: float = 1.0) -> str:

        pass

    @staticmethod

    def is\_appropriate\_character(character: str) -> bool:

        return len(character) == 1

# Create a concrete class 'Cube' that inherits from 'Figure3D'

class Cube(Figure3D):

    def \_\_init\_\_(self, length: int, character: str, color\_position: int):

        # Check if the specified length is valid

        if length <= 0:

            raise ValueError("Length must be greater than 0")

        super().\_\_init\_\_(character, color\_position)

        self.\_\_length = length

        self.\_\_offset = int(length / 2 + 1)

    def get\_2d\_representation(self) -> list:

        result = ""

        # Create a 2D representation of a cube

        for row in range(self.\_\_length):

            for col in range(self.\_\_length):

                if row == 0 or row == self.\_\_length - 1:

                    result += f"{self.\_character}  "

                elif col == 0 or col == self.\_\_length - 1:

                    result += f"{self.\_character}  "

                else:

                    result += "   "

            result += "\n"

        return [

            (Fore.\_\_getattribute\_\_(colors[self.\_color\_position]) + "\n" + result)

            for \_ in range(

                6

            )  # Repeat the same 2D representation 6 times for the 3D effect

        ]

    def get\_3d\_representation(self, scale: float = 1.0) -> str:

        # Calculate modified dimensions based on the scale factor

        modified\_length = (

            int(self.\_\_length \* scale) if self.\_\_length \* scale >= 2 else self.\_\_length

        )

        modified\_offset = int(modified\_length / 2 + 1)

        result = ""

        # Create a 3D representation of a cube

        for row in range(modified\_offset - 1):

            for col in range(modified\_length + modified\_offset - 1):

                if (row + col == modified\_offset - 1) or (

                    row == 0 and col > modified\_offset - 1

                ):

                    result += f"{self.\_character}" + (

                        ""

                        if col == modified\_length + modified\_offset - 2 and row == 0

                        else "  "

                    )

                elif modified\_length + modified\_offset - row == col + 2:

                    result += f"{self.\_character}"

                elif col == modified\_length + modified\_offset - 2:

                    result += f"  {self.\_character}"

                else:

                    result += "   "

            result += "\n"

        for row in range(modified\_length):

            for col in range(modified\_length + modified\_offset):

                if (

                    (row == 0 or row == modified\_length - 1)

                    and col < modified\_length

                    or (col == 0 or col == modified\_length - 1)

                    and row < modified\_length

                    and col < modified\_length

                ):

                    result += f"{self.\_character}" + (

                        ""

                        if row == modified\_length - 1 and col == modified\_length - 1

                        else "  "

                    )

                elif (

                    row + col == (modified\_length - 1) \* 2

                    and col < modified\_length + modified\_offset - 1

                ):

                    result += "   " \* (modified\_length - row - 2) + f"{self.\_character}"

                elif col < modified\_length and row < modified\_length:

                    result += "   "

                elif row < modified\_length - modified\_offset and col > modified\_length:

                    if col == modified\_offset + modified\_length - 1:

                        result += f"{self.\_character}"

                    else:

                        result += "   "

            result += "\n"

        return Fore.\_\_getattribute\_\_(colors[self.\_color\_position]) + "\n" + result

**lab\_5.py**

from figure import \*

# Function to get user input for the character representing the shape

def get\_character\_input():

    while True:

        character = input("Enter a character to represent in the shape: ")

        if Figure3D.is\_appropriate\_character(character) is False:

            print("You should have entered one character!")

        else:

            return character

# Function to get user input for the color position

def get\_color\_position\_input():

    while True:

        try:

            color = int(input("Enter a number of color: "))

            if color not in range(len(colors)):

                print("You should have entered a color option which is available!")

            else:

                return color

        except ValueError:

            print("You should have entered an integer number!")

# Function to get user input for the length of the figure

def get\_length\_input():

    while True:

        try:

            length = int(input("Enter a length: "))

            if length <= 0:

                print("You should have entered a length greater than 0!")

            else:

                return length

        except ValueError:

            print("You should have entered an integer number!")

# Function to get user input for the scale of the figure

def get\_scale\_input():

    while True:

        try:

            scale = float(input("Enter a scale for the figure: "))

            if scale <= 0:

                print("You should have entered a scale greater than 0!")

            else:

                return scale

        except ValueError:

            print("You should have entered a float number!")

# Define a file to save 3D representation

representation\_3d\_file = "cube.txt"

# Main function to interact with the user and create 3D figures

def main():

    is\_figure\_available: bool = False

    is\_3d\_representation\_available = False

    while True:

        print("1 - Create a cube")

        print("2 - Display 2D")

        print("3 - Save 3D")

        print("0 - Exit")

        option = str(input("Enter an option: "))

        match option:

            case "1":

                character = get\_character\_input()

                print("There are such colors available:")

                display\_colors()

                color\_position = get\_color\_position\_input()

                length = get\_length\_input()

                scale = get\_scale\_input()

                try:

                    figure = Cube(length, character, color\_position)

                    is\_figure\_available = True

                    representation\_3d = figure.get\_3d\_representation(scale=scale)

                    print(representation\_3d)

                    is\_3d\_representation\_available = True

                except ValueError as e:

                    print(e)

                    is\_figure\_available = False

            case "2":

                if is\_figure\_available is True:

                    representation\_2d = figure.get\_2d\_representation()

                    [print(item) for item in representation\_2d]

                else:

                    print("There is no figure available!")

            case "3":

                if is\_3d\_representation\_available is True:

                    try:

                        with open(representation\_3d\_file, "w") as file:

                            file.write(representation\_3d)

                    except PermissionError:

                        print("You do not have permission to write to the file!")

                    except FileNotFoundError:

                        print("The file does not exist!")

                else:

                    print("There is no figure available!")

            case "0":

                break

            case \_:

                print("Invalid option!")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Висновки:**

Виконуючи ці завдання, я створила високорівневий об'єктно-орієнтований генератор 3D ASCII-арту, який дозволить користувачам проектувати, відображати та маніпулювати 3D-фігурами в ASCII-арті. Цей проект надасть мені глибоке розуміння об'єктно-орієнтованого програмування і алгоритмів графіки, сприятиме творчому підходу до створення ASCII-арту