Міністерство освіти і науки України

Національний університет «Львівська політехніка»



**ЗВІТ**

про виконання лабораторної роботи № 5

з курсу:

«Спеціалізовані мови програмування»

**Виконала:**

студентка гр. ІТ-31

Ірина ПЕРХУН

**Прийняв:**

Сергій ЩЕРБАК

Львів 2023

**Тема:** Розробка ASCII ART генератора для візуалізації 3D-фігур

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

**Хід 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.*

**cube.py**

from classes.lab5.figures.figure3d import Figure3D, Colors

"""

Module Docstring: A brief description of the module.

"""

class Cube(Figure3D):

    """

    Class Docstring: A brief description of the Cube class.

    """

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

        """

        Initialize a Cube object.

        Parameters:

        - length (int): Length of the cube.

        - character (str): Character representing the cube.

        - color (Colors): Color of the cube.

        """

        if length <= 0:

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

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

        self.\_\_length = length

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

        self.\_\_horizontal\_offset = 0

        self.\_\_vertical\_offset = 0

    def change\_offsets(self, horizontal\_offset: int, vertical\_offset: int) -> None:

        """

        Change the horizontal and vertical offsets.

        Parameters:

        - horizontal\_offset (int): Horizontal offset value.

        - vertical\_offset (int): Vertical offset value.

        """

        self.\_\_horizontal\_offset = horizontal\_offset

        self.\_\_vertical\_offset = vertical\_offset

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

        """

        Get the 2D representation of the cube.

        Returns:

        - str: 2D representation of the cube.

        """

        result = ""

        for \_ in range(self.\_\_vertical\_offset):

            result += "\n"

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

            for \_ in range(self.\_\_horizontal\_offset):

                result += "   "

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

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

            result += "\n"

        return f"{self.\_color.value}\n{result}\033[0m"

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

        """

        Get the 3D representation of the cube.

        Parameters:

        - scale (float): Scaling factor.

        Returns:

        - str: 3D representation of the cube.

        """

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

        modified\_offset = int((self.\_\_length + modified\_length) / 4)

        result = ""

        for \_ in range(self.\_\_vertical\_offset):

            result += "\n"

        # Top view

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

            for \_ in range(self.\_\_horizontal\_offset):

                result += "   "

            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"

        # Side view

        for row in range(modified\_length):

            for \_ in range(self.\_\_horizontal\_offset):

                result += "   "

            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 f"{self.\_color.value}\n{result}\033[0m"

**figure3d.py**

from abc import ABC, abstractmethod

from colorama import Fore

from enum import Enum

from tkinter import Tk, Frame, Label, Button, StringVar, Entry, OptionMenu, filedialog, messagebox, simpledialog

"""

Module Docstring: A brief description of the module.

"""

class Colors(Enum):

    RED = Fore.RED

    GREEN = Fore.GREEN

    BLUE = Fore.BLUE

class Figure3D(ABC):

    """

    Class Docstring: A brief description of the Figure3D class.

    """

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

        """

        Initialize a Figure3D object.

        Parameters:

        - character (str): Character representing the figure.

        - color (Colors): Color of the figure.

        """

        if not isinstance(color, Colors) or len(character) != 1:

            raise ValueError("Invalid input for character or color")

        self.\_character = character

        self.\_color = color

    @abstractmethod

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

        """

        Abstract method to get the 2D representation of the figure.

        Returns:

        - list: List of strings representing the 2D representation.

        """

        pass

    @abstractmethod

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

        """

        Abstract method to get the 3D representation of the figure.

        Parameters:

        - scale (float): Scaling factor for the figure (default is 1.0).

        Returns:

        - str: String representing the 3D representation.

        """

        pass

    @staticmethod

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

        """

        Check if the provided character is appropriate for a figure.

        Parameters:

        - character (str): The character to check.

        Returns:

        - bool: True if the character is appropriate, False otherwise.

        """

        return len(character) == 1

    def change\_color(self, color: Colors) -> None:

        """

        Change the color of the figure.

        Parameters:

        - color (Colors): New color for the figure.

        """

        self.\_color = color

    def change\_character(self, character: str) -> None:

        """

        Change the character used to represent the figure.

        Parameters:

        - character (str): New character for the figure.

        Raises:

        - ValueError: If the character is not a single character.

        """

        if self.is\_appropriate\_character(character):

            self.\_character = character

        else:

            raise ValueError("Invalid character. It should be a single character.")

**file\_operations.py**

from tkinter import filedialog, messagebox, Tk

"""

Module Docstring: A brief description of the module.

"""

class FileOperations:

    """

    Class Docstring: A brief description of the FileOperations class.

    """

    @staticmethod

    def save\_representation(representation: str, representation\_type: str) -> None:

        """

        Save the representation to a file.

        Parameters:

        - representation (str): The representation to be saved.

        - representation\_type (str): Type of the representation.

        Returns:

        - None

        """

        try:

            # Create a Tkinter root window (invisible)

            root = Tk()

            root.withdraw()

            # Prompt the user to select a file for saving

            file\_path = filedialog.asksaveasfilename(

                defaultextension=".txt",

                filetypes=[("Text files", "\*.txt"), ("All files", "\*.\*")],

                title=f"Save {representation\_type} representation as"

            )

            # Check if the user canceled the file selection

            if not file\_path:

                return

            # Write the representation to the selected file

            with open(file\_path, "w") as file:

                file.write(representation)

            # Display a success message

            messagebox.showinfo("Success", f"{representation\_type} representation saved successfully to {file\_path}")

        except (PermissionError, FileNotFoundError) as e:

            # Display an error message if an exception occurs during file saving

            messagebox.showerror("Error", str(e))

        finally:

            # Destroy the Tkinter root window to free resources

            root.destroy()

**figure\_app.py**

from tkinter import Tk, Frame, Label, Button, StringVar, Entry, OptionMenu, filedialog, messagebox, simpledialog

from .figures.cube import Cube, Figure3D, Colors

from .operations.file\_operations import FileOperations

class FigureApp:

    def \_\_init\_\_(self):

        #Initialize the FigureApp

        self.root = Tk()

        self.root.title("Figure App")

        self.figure = None

        self.is\_2d\_representation\_available = False

        self.is\_3d\_representation\_available = False

        self.create\_widgets()

    #Create and layout the Tkinter widgets

    def create\_widgets(self):

       # Frame for figure creation

        self.creation\_frame = Frame(self.root)

        self.creation\_frame.pack(pady=10)

        # Label and Entry for Character input

        Label(self.creation\_frame, text="Character:").grid(row=0, column=0, padx=10)

        self.character\_entry = Entry(self.creation\_frame)

        self.character\_entry.grid(row=0, column=1, padx=10)

        # Label and OptionMenu for Color selection

        Label(self.creation\_frame, text="Color:").grid(row=0, column=2, padx=10)

        self.color\_var = StringVar(self.root)

        self.color\_var.set(Colors.RED.name)  # Default color

        self.color\_dropdown = OptionMenu(self.creation\_frame, self.color\_var, \*Colors.\_\_members\_\_.keys())

        self.color\_dropdown.grid(row=0, column=3, padx=10)

        # Label and Entry for Length input

        Label(self.creation\_frame, text="Length:").grid(row=0, column=4, padx=10)

        self.length\_entry = Entry(self.creation\_frame)

        self.length\_entry.grid(row=0, column=5, padx=10)

        # Button to create the figure based on the provided inputs

        Button(self.creation\_frame, text="Create Figure", command=self.create\_figure).grid(row=0, column=6, padx=10)

        # Frame for figure display and saving

        self.display\_frame = Frame(self.root)

        self.display\_frame.pack(pady=10)

        # Buttons to trigger 2D and 3D display and saving

        Button(self.display\_frame, text="Display 2D", command=self.display\_2d).grid(row=0, column=0, padx=10)

        Button(self.display\_frame, text="Display 3D", command=self.display\_3d).grid(row=0, column=1, padx=10)

        Button(self.display\_frame, text="Save 2D", command=self.save\_2d).grid(row=0, column=2, padx=10)

        Button(self.display\_frame, text="Save 3D", command=self.save\_3d).grid(row=0, column=3, padx=10)

        # Frame for figure modification

        self.modification\_frame = Frame(self.root)

        self.modification\_frame.pack(pady=10)

        # Buttons to trigger color, character, and offset modifications

        Button(self.modification\_frame, text="Change Color", command=self.change\_color).grid(row=0, column=0, padx=10)

        Button(self.modification\_frame, text="Change Character", command=self.change\_character).grid(row=0, column=1, padx=10)

        Button(self.modification\_frame, text="Change Offsets", command=self.change\_offsets).grid(row=0, column=2, padx=10)

    def create\_figure(self):

        #Create a Cube figure based on user input

        character = self.character\_entry.get()

        color = Colors[self.color\_var.get()]

        # Check if the length input is a valid integer

        length\_entry\_value = self.length\_entry.get()

        try:

            length = int(length\_entry\_value)

        except ValueError:

            messagebox.showerror("Error", "Invalid value for length. Please enter a valid integer.")

            return

        try:

            self.figure = Cube(length, character, color)

            messagebox.showinfo("Success", "Figure created successfully!")

        except ValueError as e:

            messagebox.showerror("Error", str(e))

    def display\_2d(self):

        #Display the 2D representation of the figure

        if self.figure:

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

            print(representation\_2d)

            self.is\_2d\_representation\_available = True

        else:

            messagebox.showerror("Error", "There is no figure available!")

    def display\_3d(self):

        #Display the 3D representation of the figure

        if self.figure:

            scale = 1.0  # You can add an Entry for scale and get it here

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

            print(representation\_3d)

            self.is\_3d\_representation\_available = True

        else:

            messagebox.showerror("Error", "There is no figure available!")

    def save\_2d(self):

        #Save the 2D representation of the figure to a file

        if self.is\_2d\_representation\_available:

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

            FileOperations.save\_representation(representation\_2d, "2D")

        else:

            messagebox.showerror("Error", "There is no 2D representation available!")

    def save\_3d(self):

        #Save the 3D representation of the figure to a file

        if self.is\_3d\_representation\_available:

            scale = 1.0  # Default scale

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

            FileOperations.save\_representation(representation\_3d, "3D")

        else:

            messagebox.showerror("Error", "There is no 3D representation available!")

    def change\_color(self):

        #Change the color of the figure based on user input

        if self.figure:

            color\_names = [color.name for color in Colors]

            color\_name = simpledialog.askstring("Input", f"Enter a name of color or choose from {', '.join(color\_names)}:")

            if color\_name is not None:

                try:

                    color = Colors[color\_name.upper()]

                    self.figure.change\_color(color)

                except KeyError:

                    messagebox.showerror("Error", "Invalid color name!")

        else:

            messagebox.showerror("Error", "There is no figure available!")

    def change\_character(self):

        #Change the character representation of the figure based on user input

        if self.figure:

            character = simpledialog.askstring("Input", "Enter a character to represent in the shape:")

            if character is not None:

                if self.figure.is\_appropriate\_character(character):

                    self.figure.change\_character(character)

                else:

                    messagebox.showerror("Error", "Invalid character. It should be a single character.")

        else:

            messagebox.showerror("Error", "There is no figure available!")

    def change\_offsets(self):

        #Change the horizontal and vertical offsets of the figure based on user input

        if self.figure:

            horizontal\_offset = simpledialog.askinteger("Input", "Enter horizontal offset:")

            if horizontal\_offset is not None:

                vertical\_offset = simpledialog.askinteger("Input", "Enter vertical offset:")

                if vertical\_offset is not None:

                    self.figure.change\_offsets(horizontal\_offset, vertical\_offset)

        else:

            messagebox.showerror("Error", "There is no figure available!")

    def run(self):

        #Start the Tkinter main loop

        self.root.mainloop()

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

    from lab\_work\_5.main import main

    main()

**main.py**

from .figure\_app import FigureApp

from shared.runnable import Runnable

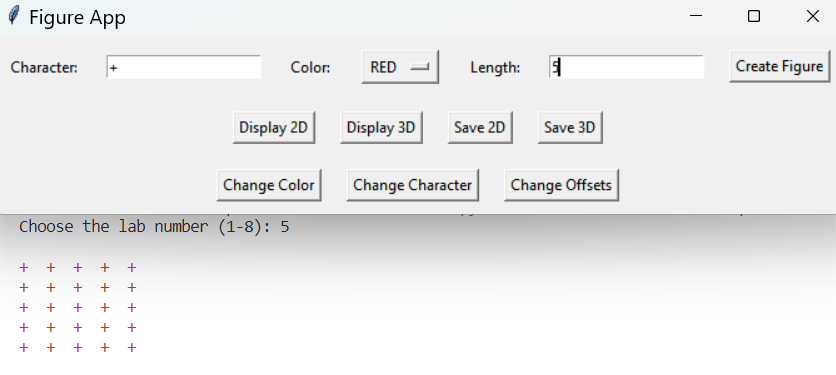
def main():

    app = FigureApp(Runnable)

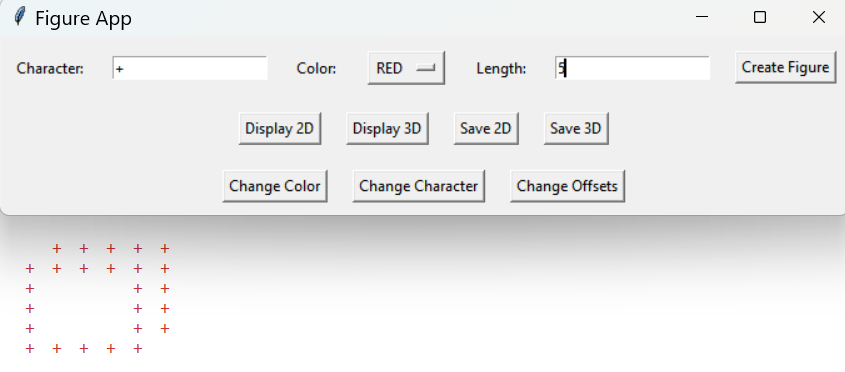
    app.run()

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

    main()



*Рис. 1 – Результат створення Display2D*

**

*Рис. 2 – Результат створення Display3D*

**Висновок:** на даній лабораторній роботі я створила додаток для малювання 3D-фігур у ASCII-арті на основі об’єктно - орієнтованого підходу та мови Python.