МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

ІНСТИТУТ КОМП’ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ



Лабораторна робота №5

з дисципліни “Спеціалізовані мови програмування”

Виконала:

студентка групи ІТ-31

Анастасія КАМІНОВСЬККА

Прийняв:

доцент кафедри ІСМ

Сергій ЩЕРБАК

Львів-2023

**Тема роботи:** 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.

Код:  
Main.py:

""" Main file"""

from .classes.cube import Cube

def main() -> None:

    """ Main method"""

    scale = int(input("Input a number of scaling: "))

    cube = Cube(center=(0, 0, 0), size=2, scale=scale)

    # Setting colors for each face

    for face in ['front', 'back', 'left', 'right', 'top', 'bottom']:

        char = input(f"Enter character for {face} face: ")

        cube.set\_color(face, char)

    ascii\_art\_cube = cube.draw()

    print(ascii\_art\_cube)

    ascii\_art\_cube\_2d = cube.draw\_2d()

    print(ascii\_art\_cube\_2d)

Cube.py:

""" Cube file"""

import logging

from .figure import Figure

class Cube(Figure):

    """Class cube for arts which inherits class Figure """

    def \_\_init\_\_(self, center, size,scale):

        super().\_\_init\_\_(center)

        self.size = size

        self.scale = scale

        self.colors = {

            'front': '\*',

            'back': ' ',

            'left': ' ',

            'right': ' ',

            'top': ' ',

            'bottom': ' ',

        }

        self.\_construct\_cube()

    def apply\_shadow(self):

        """Apply shadow onto cube """

        # Припустимо, що тінь падає на праву та нижню грань куба.

        shadow\_chars = {

            'right': ':',  # Темніший символ для правої грані

            'bottom': '.',  # Темніший символ для нижньої грані

        }

        self.colors.update(shadow\_chars)

        logging.info('Shadow are applied')

    def \_determine\_shadow(self, edge):

        """ Find shadows"""

        # Припустимо, що тінь падає з лівої верхньої сторони,

        # тому права і нижня ребра будуть темніші

        shadow\_edges = [(2, 6), (3, 2), (2, 0)]

        logging.info('Shadow are determined')

        return edge in shadow\_edges

    def \_construct\_cube(self):

        """ Generate cube"""

        x, y, z = self.center

        s = self.size

        self.vertices = [

            [x - s, y - s, z - s], [x - s, y - s, z + s],

            [x - s, y + s, z - s], [x - s, y + s, z + s],

            [x + s, y - s, z - s], [x + s, y - s, z + s],

            [x + s, y + s, z - s], [x + s, y + s, z + s],

        ]

        self.edges = [

            (0, 1), (1, 3), (3, 2), (2, 0),

            (4, 5), (5, 7), (7, 6), (6, 4),

            (0, 4), (1, 5), (2, 6), (3, 7),

        ]

    def set\_color(self, face, char):

        """Sets the character for a specific face of the cube."""

        valid\_faces = ['front', 'back', 'left', 'right', 'top', 'bottom']

        if face not in valid\_faces:

            logging.warning('Input value is not correct %s',face)

            raise ValueError(f"{face} is not a valid face for cube. Valid faces are: {valid\_faces}")

        if not isinstance(char, str) or len(char) != 1:

            raise ValueError("Color character must be a single character string.")

        self.colors[face] = char

    def isometric\_projection(self):

        """ Isometrical"""

    # Ізометрична проекція зі збільшенням масштабу для 3D координат

        projected\_2d = []

        for vertex in self.vertices:

            x, y, z = vertex

            # Проекція з використанням ізометричних кутів

            iso\_x = (x - z) \* self.scale

            iso\_y = ((x + 2 \* y + z) / 2) \* self.scale

            projected\_2d.append((iso\_x, iso\_y))

        return projected\_2d

    def draw\_line(self, canvas, start, end, char='\*'):

        """ Draw lines"""

        # Bresenham's Line Algorithm для малювання лінії між двома точками на canvas

        x0, y0 = start

        x1, y1 = end

        dx = abs(x1 - x0)

        dy = -abs(y1 - y0)

        sx = 1 if x0 < x1 else -1

        sy = 1 if y0 < y1 else -1

        err = dx + dy  # Помилка

        while True:

            canvas[y0][x0] = char

            if x0 == x1 and y0 == y1:

                break

            e2 = 2 \* err

            if e2 >= dy:

                err += dy

                x0 += sx

            if e2 <= dx:

                err += dx

                y0 += sy

    def draw(self):

        """ Draw"""

        # Створюємо полотно для малювання куба

        projection = self.isometric\_projection()  # Задайте бажаний масштаб тут

        min\_x = min(x for x, y in projection)

        max\_x = max(x for x, y in projection)

        min\_y = min(y for x, y in projection)

        max\_y = max(y for x, y in projection)

        width = int(max\_x - min\_x) + 1

        height = int(max\_y - min\_y) + 1

        canvas = [[' ' for \_ in range(width)] for \_ in range(height)]

        # Малюємо кожне ребро куба

        for edge in self.edges:

            start = projection[edge[0]]

            end = projection[edge[1]]

            # Determine the face this edge belongs to (simplified example logic)

            face = self.\_determine\_face(edge)

            if self.\_determine\_shadow(edge):

                char = ':'  # Темніший символ для тіні

            else:

                char = self.colors[face]

            self.draw\_line(canvas, (int(start[0] - min\_x), int(start[1] - min\_y)),

                           (int(end[0] - min\_x), int(end[1] - min\_y)), char)

        # Перетворюємо полотно у рядок для відображення

        return "\n".join("".join(row) for row in canvas)

    def \_determine\_face(self, edge):

        """ Find face"""

        # Simplified logic to determine the face based on edge indices

        if edge in [(4, 5), (5, 7), (6, 7), (4, 6)]:

            logging.info('Face is determined to front( %s )' ,edge)

            return 'front'

        # Assign 'top' to top edges

        elif edge in [(0, 1), (1, 3), (2, 3), (0, 2)]:

            logging.info('Face is determined to top ( %s )' ,edge)

            return 'top'

        # Assign 'left' to vertical edges on the left side of the cube

        elif edge in [(0, 4), (2, 6)]:

            logging.info('Face is determined to left ( %s )' ,edge)

            return 'left'

        # Default to 'right' for other edges, assuming right side is not visible

        else:

            logging.info('Face is determined to right ( %s )' ,edge)

            return 'right'

    def draw\_2d(self):

        """Draw 2d cube """

        size = self.size \* 2

        square = [[' ' for \_ in range(size)] for \_ in range(size)]

        # Draw the top border of the square

        for i in range(size):

            square[0][i] = self.colors['top']

        # Draw the bottom border of the square

        for i in range(size):

            square[-1][i] = self.colors['bottom']

        # Draw the left border of the square

        for i in range(1, size - 1):

            square[i][0] = self.colors['left']

        # Draw the right border of the square

        for i in range(1, size - 1):

            square[i][-1] = self.colors['right']

        # Convert the 2D array into a string for display

        return "\n".join(" ".join(row) for row in square)

    def project\_to\_2d(self):

        # Ізометрична проекція зі збільшенням масштабу для 3D координат

        projected\_2d = []

        for vertex in self.vertices:

            x, y, z = vertex

            # Проекція з використанням ізометричних кутів

            iso\_x = (x - z) \* self.scale

            iso\_y = ((x + 2 \* y + z) / 2) \* self.scale

            projected\_2d.append((iso\_x, iso\_y))

        return projected\_2d

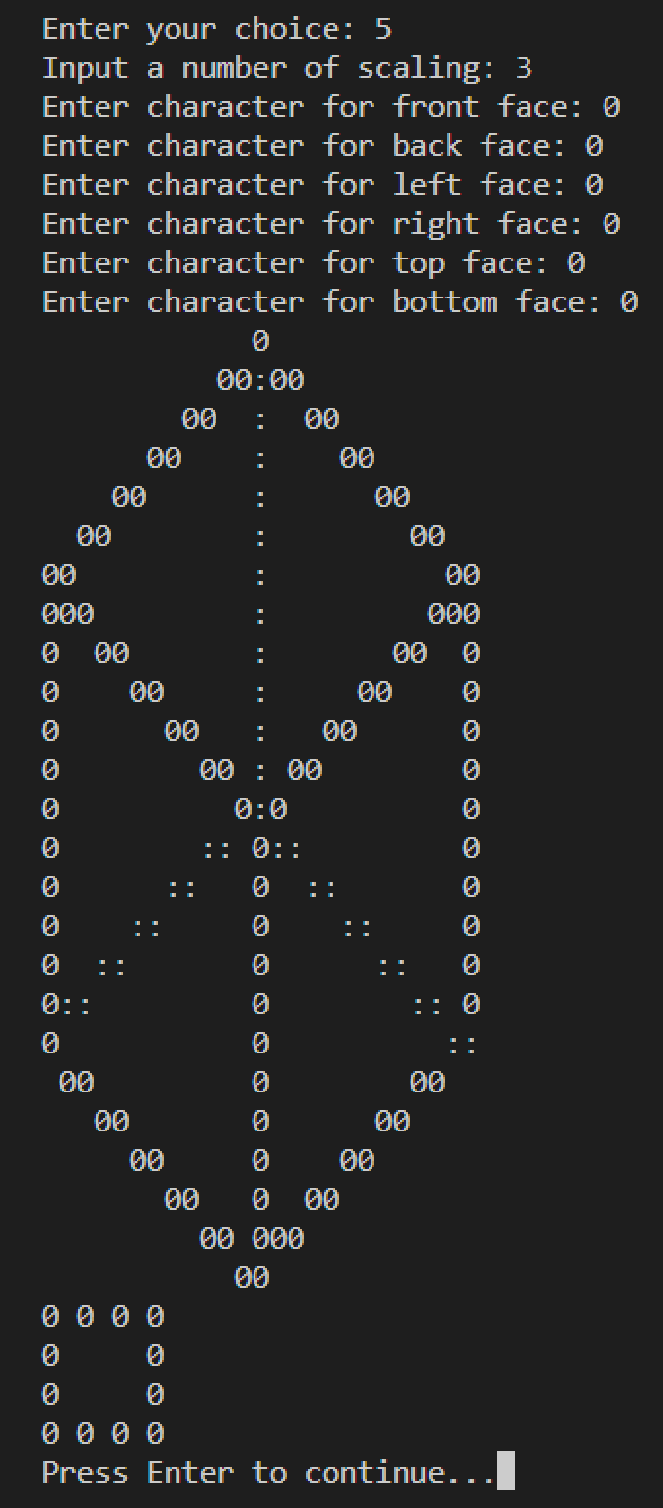


Рис.1 Результат виконання роботи

**Висновок**: При виконанні лабораторної роботи було створено високорівневий об'єктно-орієнтований генератор 3D ASCII-арту, який дозволить користувачам проектувати, відображати та маніпулювати 3D-фігурами в ASCII-арті. Цей проект надав мені глибоке розуміння об'єктно-орієнтованого програмування і алгоритмів графіки, сприятиме творчому підходу до створення ASCII-арту.