МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
Інститут комп'ютерних наук та інформаційних технологій
Кафедра інформаційних систем та мереж

Лабораторна робота №5
з дисципліни: «Спеціалізовані мови програмування»
на тему: «Розробка ASCII ART генератора для візуалізації 3D-фігур»

Виконала:
студентка групи ІТ-32
Моляща Ю.А.
Прийняв:
Щербак С.С.

Львів - 2023

Лабораторна робота №5

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

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

Завдання на лабораторну роботу.

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.

Хід роботи.

main.py

```
# Include the parent directory in the system's import path
import sys
import os

current_dir = os.path.dirname(os.path.abspath(__file__))
parent_dir = os.path.abspath(os.path.join(current_dir, '..'))
sys.path.append(parent_dir)

from lab5.three_d_shape import ThreeDShape

SETTINGS_FILE_PATH = 'source/lab5/settings.json'

def main():
    shape_obj = ThreeDShape()
    shape_obj.set_settings_file_path(SETTINGS_FILE_PATH)
    shape_obj.load_settings()
    # shape_obj.render_3d_shape()

    while True:
        print('\n')
        print('Options:')
        print('1. Render 3D shape rotating')
        print('2. Render 3D shape static')
        print('3. Settings')
        print('4. Exit')
        user_input = input('Enter option number: ')
```

```
        if user_input == '1':
            shape_obj.render_3d_shape_static()
        elif user_input == '2':
            shape_obj.render_3d_shape_rotating()
        elif user_input == '3':
            shape_obj.settings()
        elif user_input == '4':
            break


if __name__ == "__main__":
    main()
```

settings.json

```
{"cube_width": 20, "scale": 40, "color": "\u001b[34m", "A": 0.6, "B": 0.8, "C": 0, "width": 167, "height": 44, "horizontal_offset": 0, "distance_from_viewer": 100, "increment_speed": 1}
```

three_d_shape.py

```
# Include the parent directory in the system's import path
import sys
import os

current_dir = os.path.dirname(os.path.abspath(__file__))
parent_dir = os.path.abspath(os.path.join(current_dir, '..'))
sys.path.append(parent_dir)

# Imports
from math import sin, cos
import time
import json
from colorama import Fore
from colorama import init as colorama_init
from lab3.ascii_art_generator import set_color

# Initialize colorama
# autoreset=True -> Color settings will automatically reset after each print statement
```

```
colorama_init(autoreset=True)

class ThreeDShape:
    def __init__(self):
        self.settings_file_path = None

        self.cube_width = None
        self.scale = None
        self.color = None

        self.A = None
        self.B = None
        self.C = None

        self.width = None
        self.height = None
        self.horizontal_offset = None
        self.distance_from_viewer = None
        self.increment_speed = None

        self.char_array = None
        self.depth_array = None

    def set_settings_file_path(self, settings_file_path):
        self.settings_file_path = settings_file_path

    def set_cube_width(self, cube_width):
        cube_width = input('Enter cube width: ')
        self.cube_width = int(cube_width)
        self.save_settings()

    def set_scale(self, scale):
        scale = input('Enter scale: ')
        self.scale = int(scale)
```

```
        self.save_settings()

    def set_color(self, color):
        color = set_color()
        self.color = color
        self.save_settings()

    def set_A(self, A):
        A = input('Enter angle A: ')
        self.A = float(A)
        self.save_settings()

    def set_B(self, B):
        B = input('Enter angle B: ')
        self.B = float(B)
        self.save_settings()

    def set_C(self, C):
        C = input('Enter angle C: ')
        self.C = float(C)
        self.save_settings()

    def set_width(self, width):
        width = int(input('Enter width: '))

        terminal_columns, terminal_lines = self.get_terminal_size()

        if width < 0:
            raise ValueError('Width must be a positive integer.')
        elif width > terminal_columns:
               raise ValueError(f'Width {width} exceeds the terminal length of
{terminal_columns}')
        else:
            pass
```

```
        self.width = width
        self.save_settings()

    def set_height(self, height):
        height = int(input('Enter height: '))

        terminal_columns, terminal_lines = self.get_terminal_size()

        if height < 0:
            raise ValueError('Height must be a positive integer.')
        elif height > terminal_lines:
              raise ValueError(f'Height {height} exceeds the terminal length of
{terminal_lines}')
        else:
            pass

        self.height = height
        self.save_settings()

    def get_terminal_size(self):
        terminal_columns, terminal_lines = os.get_terminal_size()
        return terminal_columns, terminal_lines

    def set_horizontal_offset(self, horizontal_offset):
        horizontal_offset = input('Enter horizontal offset: ')
        self.horizontal_offset = int(horizontal_offset)
        self.save_settings()

    def set_distance_from_viewer(self, distance_from_viewer):
        distance_from_viewer = input('Enter distance from viewer: ')
        self.distance_from_viewer = int(distance_from_viewer)
        self.save_settings()
```

```
def set_increment_speed(self, increment_speed):
    increment_speed = input('Enter increment speed: ')
    self.increment_speed = int(increment_speed)
    self.save_settings()

# Settings

def default_settings(self):
    self.cube_width = 20
    self.scale = 40
    self.color = '\x1b[39m'

    self.A = 0
    self.B = 0
    self.C = 0

    self.width = self.get_terminal_size()[0] - 1
    self.height = self.get_terminal_size()[1] - 1
    self.horizontal_offset = 0
    self.distance_from_viewer = 100
    self.increment_speed = 1

def show_settings(self):
    print('\n')
    print('Current settings:')
    print('Cube width:', self.cube_width)
    print('Scale:', self.scale)
    print('Color:', self.color.replace('\x1b', '\\x1b'))
    print('A:', self.A)
    print('B:', self.B)
    print('C:', self.C)
    print('Width:', self.width)
    print('Height:', self.height)
    print('Horizontal offset:', self.horizontal_offset)
```

```
    print('Distance from viewer:', self.distance_from_viewer)
    print('Increment speed:', self.increment_speed)

def save_settings(self):
    settings_data = {
        'cube_width': self.cube_width,
        'scale': self.scale,
        'color': self.color,
        'A': self.A,
        'B': self.B,
        'C': self.C,
        'width': self.width,
        'height': self.height,
        'horizontal_offset': self.horizontal_offset,
        'distance_from_viewer': self.distance_from_viewer,
        'increment_speed': self.increment_speed
    }
    with open(self.settings_file_path, 'w') as file:
        json.dump(settings_data, file)

def load_settings(self):
    try:
        with open(self.settings_file_path, 'r') as file:
            settings_data = json.load(file)
            self.cube_width = settings_data['cube_width']
            self.scale = settings_data['scale']
            self.color = settings_data['color']
            self.A = settings_data['A']
            self.B = settings_data['B']
            self.C = settings_data['C']
            self.width = settings_data['width']
            self.height = settings_data['height']
            self.horizontal_offset = settings_data['horizontal_offset']
```

```
                                                    self.distance_from_viewer   =
settings_data['distance_from_viewer']
                self.increment_speed = settings_data['increment_speed']
        except FileNotFoundError:
            # If file does not exist use default settings
            self.default_settings()

    def settings(self):
        while True:
            print('\n')
            print('Settings:')
            print('1. Set cube width')
            print('2. Set scale')
            print('3. Set color')
            print('4. Set angle A')
            print('5. Set angle B')
            print('6. Set angle C')
            print('7. Set width')
            print('8. Set height')
            print('9. Set horizontal offset')
            print('10. Set distance from viewer')
            print('11. Set increment speed')
            print('12. Show settings')
            print('13. Set default settings')
            print('14. Back')

            user_input = input('Enter option number: ')

            if user_input == '1':
                self.set_cube_width(self.cube_width)
            elif user_input == '2':
                self.set_scale(self.scale)
            elif user_input == '3':
                self.set_color(self.color)
```

```
        elif user_input == '4':
            self.set_A(self.A)
        elif user_input == '5':
            self.set_B(self.B)
        elif user_input == '6':
            self.set_C(self.C)
        elif user_input == '7':
            self.set_width(self.width)
        elif user_input == '8':
            self.set_height(self.height)
        elif user_input == '9':
            self.set_horizontal_offset(self.horizontal_offset)
        elif user_input == '10':
            self.set_distance_from_viewer(self.distance_from_viewer)
        elif user_input == '11':
            self.set_increment_speed(self.increment_speed)
        elif user_input == '12':
            self.show_settings()
        elif user_input == '13':
            self.default_settings()
        elif user_input == '14':
            break
        else:
            print('Invalid option.')

# Render

def calculate_x(self, i, j, k):
    """ Calculate x coordinate after applying current rotation angles."""
    return ( j * sin(self.A) * sin(self.B) * cos(self.C) -
             k * cos(self.A) * sin(self.B) * cos(self.C) +
             j * cos(self.A) * sin(self.C) +
             k * sin(self.A) * sin(self.C) +
             i * cos(self.B) * cos(self.C) )
```

```
    def calculate_y(self, i, j, k):
        """ Calculate y coordinate after applying current rotation angles."""
        return ( j * cos(self.A) * cos(self.C) +
                 k * sin(self.A) * cos(self.C) -
                 j * sin(self.A) * sin(self.B) * sin(self.C) +
                 k * cos(self.A) * sin(self.B) * sin(self.C) -
                 i * cos(self.B) * sin(self.C) )

    def calculate_z(self, i, j, k):
        """Calculate z coordinate after applying current rotation angles."""
        return ( k * cos(self.A) * cos(self.B) -
                 j * sin(self.A) * cos(self.B) +
                 i * sin(self.B) )

    def threed_to_twod(self, i, j, k):
        """Convert 3D coordinates to 2D coordinates."""
        x = self.calculate_x(i, j, k)
        y = self.calculate_y(i, j, k)
        z = self.calculate_z(i, j, k) + self.distance_from_viewer

        depth_value = 1 / z
        return x, y, depth_value

    def calculate_surface(self, i, j, k, char):
        """Calculate surface of the cube."""
        x, y, depth_value = self.threed_to_twod(i, j, k)

          screen_x = int(self.width / 2 + self.horizontal_offset + self.scale *
depth_value * x * 2)
        screen_y = int(self.height / 2 + self.scale * depth_value * y)

        idx = screen_x + screen_y * self.width
        if 0 <= idx < self.width * self.height:
```

```
            if depth_value > self.depth_array[idx]:
                self.depth_array[idx] = depth_value
                self.char_array[idx] = char

    def render_cube(self):
                for cube_x in range(-self.cube_width, self.cube_width, max(1,
int(self.increment_speed))):
                    for cube_y in range(-self.cube_width, self.cube_width, max(1,
int(self.increment_speed))):
                self.calculate_surface(cube_x, cube_y, -self.cube_width, '@')
                self.calculate_surface(self.cube_width, cube_y, cube_x, '$')
                self.calculate_surface(-self.cube_width, cube_y, -cube_x, '~')
                self.calculate_surface(-cube_x, cube_y, self.cube_width, '#')
                self.calculate_surface(cube_x, -self.cube_width, -cube_y, ';')
                self.calculate_surface(cube_x, self.cube_width, cube_y, '+')

    def initialize_arrays(self):
        self.char_array = [' '] * (self.width * self.height)
        self.depth_array = [0] * (self.width * self.height)

    def display_frame(self):
        # Move cursor to top-left corner
        print("\033[H", end="")
        for index in range(self.width * self.height):
            # Check if new line is needed
            if index % self.width == 0:
                print()
            print(self.color + self.char_array[index] , end="")

    def update_rotation_angles(self):
        self.A += 0.05
        self.B += 0.05
        self.C += 0.01

    def render_3d_shape_static(self):
```

```
        os.system('clear')
        self.initialize_arrays()
        self.render_cube()
        self.display_frame()

    def render_3d_shape_rotating(self):
        os.system('clear')
        while True:
            self.initialize_arrays()
            self.render_cube()
            self.display_frame()
            self.update_rotation_angles()
            time.sleep(0.0001)
```

Висновок: під час виконання лабораторної роботи було створено дотаток для малювання 3D-фігур у ASCII-арті на основі об’єктно - орієнтованого підходу та мови Python