МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

ІНСТИТУТ КОМП'ЮТЕРНИХ НАУК ТА ІНОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗВІТ

для лабораторної роботи № 5 з

дисципліни

«Спеціалізовані мови програмування»

Виконав:

студент гр. ІТ-32

Паньків Б. В.

Прийняв:

доц. каф. ІСМ

Щербак С.С.

Львів-2023

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

Хід виконання:

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:

functions.py

```
from colorama import Fore, init



init(autoreset = True)



class RectangleArt:

    VALID_COLORS = {'WHITE', 'RED', 'BLUE', 'YELLOW', 'GREEN', 'MAGENTA'}
```

```
    def __init__(self, width, height, outer_color='BLUE', middle_color='MAGENTA', inner_color='RED', symbol_count=1, symbol_color='*'):
        if width < 1 or height < 1:
            print("Error: Rectangle dimensions are less than 1.")
            return

        self.width = width
        self.height = height
        self.outer_rectangle_color = outer_color if outer_color in self.VALID_COLORS else 'BLUE'
        self.middle_rectangle_color = middle_color if middle_color in self.VALID_COLORS else 'MAGENTA'
        self.inner_rectangle_color = inner_color if inner_color in self.VALID_COLORS else 'RED'
        self.symbol_count = symbol_count
        self.symbol_color = symbol_color

        self.outer_rectangle = self.generate_outer_rectangle()
        self.middle_rectangles = self.generate_middle_rectangles()
        self.inner_rectangle = self.generate_inner_rectangle()

    def set_rectangle_color(self, attribute, color):
        if color in self.VALID_COLORS:
            setattr(self, f"{attribute}_rectangle_color", color)

    def generate_rectangle(self, color, condition_func):
        rectangle = [[Fore.WHITE + ' ' for _ in range(self.width)] for _ in range(self.height)]
```

```
        for i in range(self.height):
            for j in range(self.width):
                if condition_func(i, j):
                    rectangle[i][j] = getattr(Fore, color) + self.symbol_color
        return rectangle

    def generate_outer_rectangle(self):
        condition_func = lambda i, j: j == 0 or i == 0
        return self.generate_rectangle(self.outer_rectangle_color, condition_func)

    def generate_middle_rectangles(self):
        middle_rectangles = []

        if self.width > 2 and self.height > 2:
            offset = 1
            condition_func = lambda i, j: (i == 0 and (j == 0 or j == self.width - 1)) or (i == self.height - 1 and j == 0)

            for _ in range(self.height // 2 - 1):
                rectangle = self.generate_rectangle(self.middle_rectangle_color, condition_func)
                middle_rectangles.append((rectangle, offset))
                offset += 1

        return middle_rectangles

    def generate_inner_rectangle(self):
```

```
        rectangle = [[Fore.WHITE + ' ' for _ in range(self.width)] for _ in range(self.height)]

        if self.width > 2 and self.height > 2:
            offset_right = (self.width // 2) + 3
            offset_down = self.height // 2

            for i in range(self.height):
                for j in range(self.width):
                    if i == 0 or i == self.height - 1 or j == 0 or j == self.width - 1:
                        rectangle[i][j] = getattr(Fore, self.inner_rectangle_color) + self.symbol_color
                    if offset_down <= i < self.height - offset_down and offset_right <= j < self.width - offset_right:
                        rectangle[i][j] = ' '

        return rectangle

    def resize_matrix(self, matrix):
        for row in matrix:
            row.extend([Fore.WHITE + ' '] * (self.width * 3 - 1))

    def combine_rectangles(self):
        combined_width = int((((self.width + self.height) / 2) + self.width) * 3)
        combined_height = int(((self.height + self.width) / 2) + self.height)

        combined_matrix = [[Fore.WHITE + ' ' for _ in range(combined_width)] for _ in range(combined_height)]
```

```
    for i in range(self.height):
        for j in range(self.width):
            combined_matrix[i][int(j * 3)] = self.outer_rectangle[i][j]

    middle_offset = 0
    for middle_rectangle, offset in self.middle_rectangles:
        for i in range(self.height):
            for j in range(self.width):
                combined_matrix[i + offset][int(j * 3) + offset] = middle_rectangle[i][j]
            middle_offset = offset

    inner_offset = middle_offset + 1
    for i in range(self.height):
        for j in range(self.width):
            combined_matrix[i + inner_offset][int(j * 3) + inner_offset] = self.inner_rectangle[i][j]

    return combined_matrix

def draw_rectangles(self, rectangles):
    for rectangle in rectangles:
        for row in rectangle:
            print(''.join(row))

def draw_combined_rectangles(self):
```

```
    combined_matrix = self.combine_rectangles()
    self.draw_rectangles([combined_matrix])

def draw_inner_rectangle(self):
    self.draw_rectangles([self.inner_rectangle])

def draw_middle_rectangles(self):
    rectangles = [middle_rectangle for middle_rectangle, _ in self.middle_rectangles]
    self.draw_rectangles(rectangles)

def draw_outer_rectangle(self):
    self.draw_rectangles([self.outer_rectangle])

def convert_to_2d(self):
    print("Converting 3D art to 2D...")
    for row in self.inner_rectangle:
        print('  '.join(row))

def scale_figure(self, scale_factor):
    if scale_factor <= 0:
        print("Error: Scale factor should be a positive number.")
        return

    self.width = int(self.width * scale_factor)
    self.height = int(self.height * scale_factor)
```

```
    self.outer_rectangle = self.generate_outer_rectangle()
    self.middle_rectangles = self.generate_middle_rectangles()
    self.inner_rectangle = self.generate_inner_rectangle()

def reverse_scale_figure(self, scale_factor):
    if scale_factor <= 0:
        print("Error: Scale factor should be a positive number.")
        return

    new_width = int(self.width / scale_factor)
    new_height = int(self.height / scale_factor)

    if new_width < 3 or new_height < 3:
        print("It is not possible to reduce the figure to dimensions smaller than 3x3.")
        return

    self.width = new_width
    self.height = new_height

    self.outer_rectangle = self.generate_outer_rectangle()
    self.middle_rectangles = self.generate_middle_rectangles()
    self.inner_rectangle = self.generate_inner_rectangle()

def align_art(self, alignment, console_length):
```

```
        combined_matrix = self.combine_rectangles()
        max_length = max(len("".join(row)) for row in combined_matrix)

        if alignment == 'center':
            print("\n".join(row.center(console_length) for row in map("".join, combined_matrix)))
        elif alignment == 'right':
            print("\n".join(row.rjust(console_length) for row in map("".join, combined_matrix)))
        elif alignment == 'left':
            print("\n".join(row.ljust(console_length) for row in map("".join, combined_matrix)))

    def save_to_file(self, file_name):
        combined_matrix = self.combine_rectangles()
        with open(file_name, 'w') as f:
            for row in combined_matrix:
                row = ''.join(
                    map(lambda x: x.replace(Fore.WHITE, '').replace(Fore.RED, '').replace(Fore.BLUE, '').replace(
                        Fore.YELLOW, '').replace(Fore.GREEN, '').replace(Fore.MAGENTA, ''), row))
                f.write(row + '\n')
        print(f"ASCII art is saved to a file {file_name}.")
```

runner.py

```
from functions import RectangleArt

def get_length_input(shape_name):
    while True:
        length = input(f"Enter the {shape_name} length (must be a number greater than or equal to 4): ")
```

```
        try:
            length = int(length)
            if length < 4:
                print("Error: Length must be greater than or equal to 4.")
            else:
                return length
        except ValueError:
            print("Error: Please enter a valid number.")


def get_color_input():
    while True:
        color_option = input("Do you want to use one or three colors for the shape? (1-only one or 3-three different): ")
        if color_option not in ["1", "3"]:
            print("Error: Incorrect selection. Please choose the correct option.")
        else:
            break
    if color_option == "1":
        color_choice = input(f"Choose a color for the shape from the list {list(RectangleArt.VALID_COLORS)}: ")
        if color_choice not in RectangleArt.VALID_COLORS:
            print("Error: Incorrect color selection. Please choose the correct option.")
        return [color_choice]
    elif color_option == "3":
        return ['BLUE', 'MAGENTA', 'RED']
```

```
def get_alignment_input():
    while True:
        alignment = input("Select alignment (left, center, right): ")
        if alignment not in ["left", "center", "right"]:
            print("Error: Incorrect alignment. Please choose the correct option.")
        else:
            return alignment


def get_manipulation_input(rectangle_art):
    while True:
        manipulate_choice = input("Do you want to change your figure? (yes or no): ").lower()
        if manipulate_choice == "yes":
            manipulation_type = input("Enter the type of change: 1-scale, 2-reverse_scale_figure: ")
            if manipulation_type in ["1", "2"]:
                scale_factor = float(input("Enter the scale factor: "))
                if manipulation_type == "1":
                    rectangle_art.scale_figure(scale_factor)
                elif manipulation_type == "2":
                    rectangle_art.reverse_scale_figure(scale_factor)
                rectangle_art.draw_combined_rectangles()
            else:
                print("Error: Invalid change type.")
        else:
            break
```

```
def main():
    console_length = 400

    while True:
        print("Menu:")
        print("1. Draw a cube")
        print("2. Draw a parallelepiped")
        print("3. Exit")

        choice = input("Please select an option: ")

        if choice == "3":
            break
        if choice not in ["1", "2"]:
            print("Error: Incorrect selection. Please choose the correct option.")
            continue

        length = get_length_input("shape")

        if choice == "2":
            width = get_length_input("parallelepiped width")

        colors = get_color_input()

        if choice == "1":
```

```
            rectangle_art = RectangleArt(length, length, *colors)
        elif choice == "2":
            rectangle_art = RectangleArt(length, width, *colors)

        alignment = get_alignment_input()
        print(f"{alignment.capitalize()} alignment:")
        rectangle_art.align_art(alignment, console_length)

        get_manipulation_input(rectangle_art)

        convert_2D = input("Want to turn 3D art into 2D? (yes or no): ").lower()
        if convert_2D == "yes":
            rectangle_art.convert_to_2d()

        save_choice = input("Do you want to save the generated ASCII art to a file? (yes or no): ").lower()
        if save_choice == "yes":
            file_name = input("Enter a file name to save the ASCII art: ")
            rectangle_art.save_to_file(file_name)

        continue_choice = input("Do you want to continue drawing? (yes or no): ").lower()
        if continue_choice != "yes":
            break

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

На рис. 1 зображено знімок екрану виконання програми.

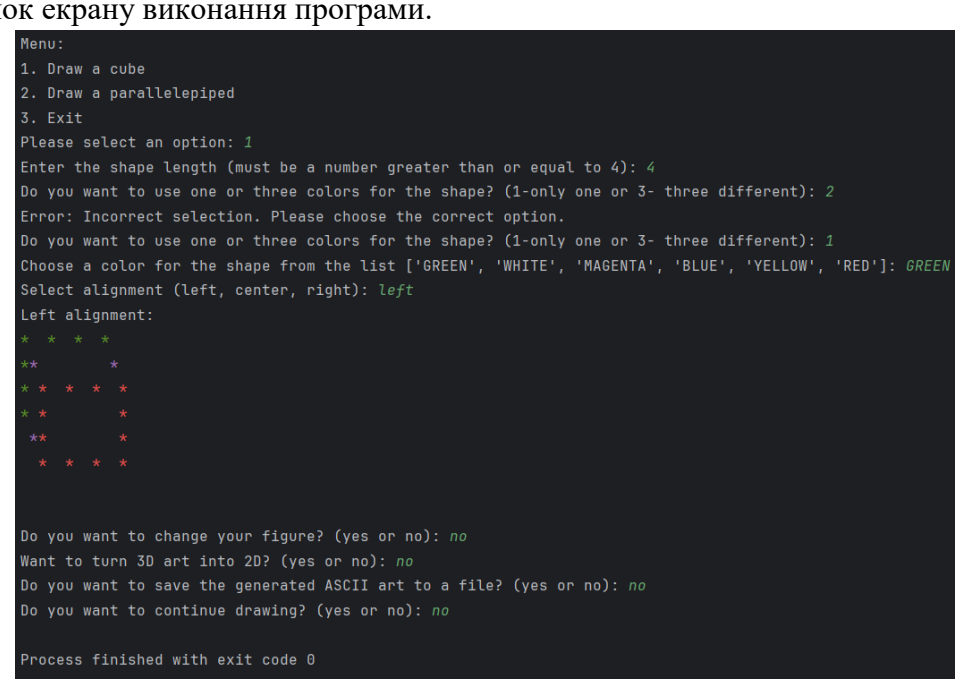

Рис. 1 Виконання програми

Посилання на GitHub-репозиторій із кодом: https://github.com/BOHDAN1329/SMP

Висновки: Виконуючи ці завдання, я створив високорівневий об'єктно-орієнтований генератор 3D ASCII-арту, який дозволяє користувачам проектувати, відображати та маніпулювати 3Dфігурами в ASCII-арті. Цей проект надав мені глибоке розуміння об'єктно-орієнтованого програмування і алгоритмів графіки.