Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра інформаційних систем та мереж

ЗВІТ

про виконання лабораторної роботи № 5

**“ Розробка ASCII ART генератора для візуалізації 3D-фігур ”**

з дисципліни **“** **Спеціалізовані мови програмування”**

Виконав:

студент групи ІТ-32

ОЛЕНЮК О. М.

Прийняв:

ЩЕРБАК С. С.

**Львів – 2023**

**Мета**: Cтворення 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.

**Код програми:**

rectangle\_art.py:

from colorama import Fore, init  
  
init(autoreset=True)  
  
colors = {  
 1: 'WHITE',  
 2: 'RED',  
 3: 'BLUE',  
 4: 'YELLOW',  
 5: 'GREEN',  
 6: 'MAGENTA'  
}  
  
  
class RectangleArt:  
 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.")  
 else:  
 self.width = width  
 self.height = height  
 self.outer\_rectangle\_color = outer\_color if outer\_color in colors.values() else 'BLUE'  
 self.middle\_rectangle\_color = middle\_color if middle\_color in colors.values() else 'MAGENTA'  
 self.inner\_rectangle\_color = inner\_color if inner\_color in colors.values() 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\_outer\_rectangle\_color(self, color):  
 self.outer\_rectangle\_color = color  
  
 def set\_middle\_rectangle\_color(self, color):  
 self.middle\_rectangle\_color = color  
  
 def set\_inner\_rectangle\_color(self, color):  
 self.inner\_rectangle\_color = color  
  
 def generate\_outer\_rectangle(self):  
 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 j == 0 or i == 0:  
 rectangle[i][j] = getattr(Fore, self.outer\_rectangle\_color) + '\*'  
 return rectangle  
  
 def generate\_middle\_rectangles(self):  
 middle\_rectangles = []  
 if self.width > 2 and self.height > 2:  
 offset = 1  
 for \_ in range(self.height // 2 - 1):  
 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 i == 0 and (j == 0 or j == self.width - 1):  
 rectangle[i][j] = getattr(Fore, self.middle\_rectangle\_color) + '\*'  
 elif i == self.height - 1 and j == 0:  
 rectangle[i][j] = getattr(Fore, self.middle\_rectangle\_color) + '\*'  
 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) + '\*'  
 if i >= offset\_down and i < self.height - offset\_down and j >= offset\_right and 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\_combined\_rectangles(self):  
 combined\_matrix = self.combine\_rectangles()  
 for row in combined\_matrix:  
 print(''.join(row))  
  
 def draw\_inner\_rectangle(self):  
 for row in self.inner\_rectangle:  
 print(''.join(row))  
  
 def draw\_middle\_rectangles(self):  
 for middle\_rectangle, \_ in self.middle\_rectangles:  
 for row in middle\_rectangle:  
 print(''.join(row))  
  
 def draw\_outer\_rectangle(self):  
 for row in self.outer\_rectangle:  
 print(''.join(row))  
  
 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':  
 for row in combined\_matrix:  
 print(f"{(''.join(row)).center(console\_length)}")  
 elif alignment == 'right':  
 for row in combined\_matrix:  
 print(f"{(''.join(row)).rjust(console\_length)}")  
 elif alignment == 'left':  
 for row in combined\_matrix:  
 print(f"{(''.join(row)).ljust(console\_length)}")  
  
 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 rectangle\_art import \*  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 console\_length = 360  
  
 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  
 if choice in ["1", "2"]:  
 if choice == "1":  
 while True:  
 length = input("Enter the length of the cube (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:  
 break  
 except ValueError:  
 print("Error: Please enter a valid number.")  
 if choice == "2":  
 while True:  
 length = input("Enter the length of the parallelepiped (must be a number greater than or equal to 4): ")  
 width = input("Enter the width of the parallelepiped (must be a number greater than or equal to 4): ")  
 try:  
 length = int(length)  
 width = int(width)  
 if length < 4 or width < 4:  
 print("error: Length and width must be greater than or equal to 4.")  
 else:  
 break  
 except ValueError:  
 print("Error: Please enter a valid number.")  
 print()  
 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.")  
 continue  
 else:  
 break  
 if color\_option == "1":  
 color\_choice = input(f"Choose a color for the shape from the list {list(colors.values())}: ")  
 if color\_choice not in colors.values():  
 print("Error: Incorrect color selection. Please choose the correct option.")  
 continue  
 if choice == "1":  
 rectangle\_art = RectangleArt(length, length, outer\_color=color\_choice, middle\_color=color\_choice, inner\_color=color\_choice)  
 elif choice == "2":  
 rectangle\_art = RectangleArt(length, width, outer\_color=color\_choice, middle\_color=color\_choice, inner\_color=color\_choice)  
 elif color\_option == "3":  
 if choice == "1":  
 rectangle\_art = RectangleArt(length, length, outer\_color='BLUE', middle\_color='MAGENTA', inner\_color='RED')  
 elif choice == "2":  
 rectangle\_art = RectangleArt(length, width, outer\_color='BLUE', middle\_color='MAGENTA', inner\_color='RED')  
 print()  
 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:  
 if alignment == 'left':  
 print("Left alignment:")  
 rectangle\_art.align\_art(alignment, console\_length)  
 elif alignment == 'center':  
 print("Center alignment:")  
 rectangle\_art.align\_art(alignment, console\_length)  
 elif alignment == 'right':  
 print("Right alignment:")  
 rectangle\_art.align\_art(alignment, console\_length)  
 break  
 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 == "1":  
 scale\_factor = float(input("Enter the scale factor: "))  
 rectangle\_art.scale\_figure(scale\_factor)  
 rectangle\_art.draw\_combined\_rectangles()  
 elif manipulation\_type == "2":  
 scale\_factor = float(input("Enter the scale factor: "))  
 rectangle\_art.reverse\_scale\_figure(scale\_factor)  
 rectangle\_art.draw\_combined\_rectangles()  
 else:  
 print("Error: Invalid change type.")  
 else:  
 break  
 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

**Результат програми:**

На рис.1, 2, 3, 4, 5 зображений результат виконання роботи у програмі.

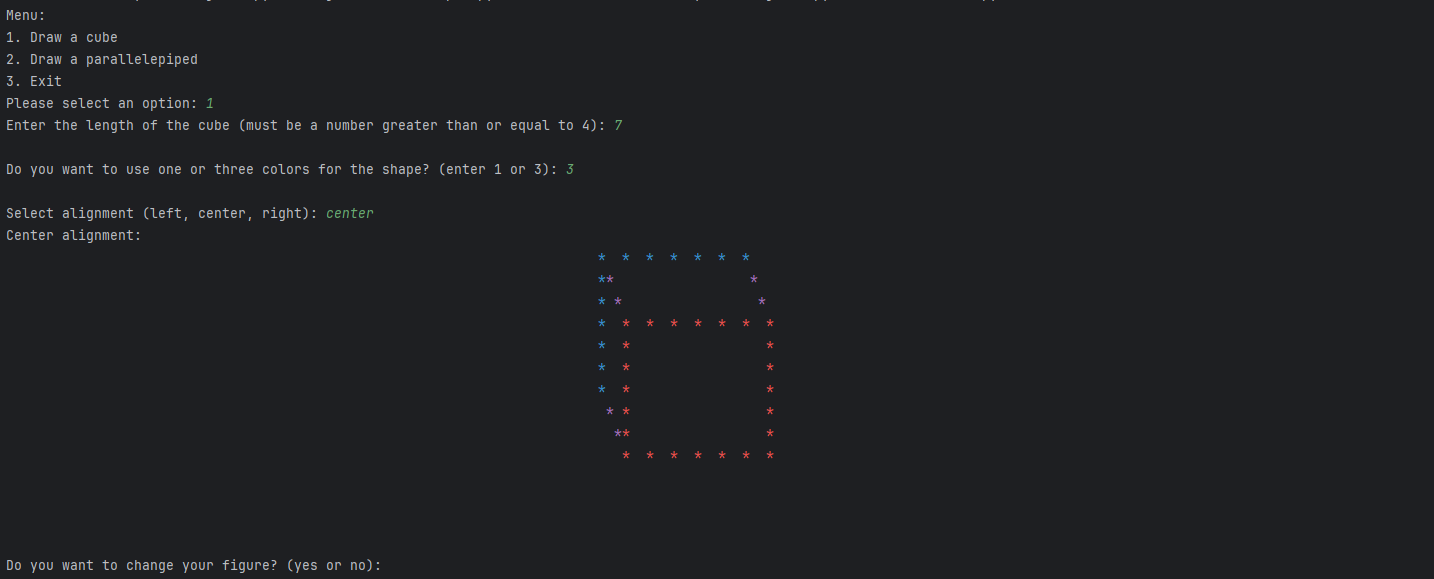


Рис.1

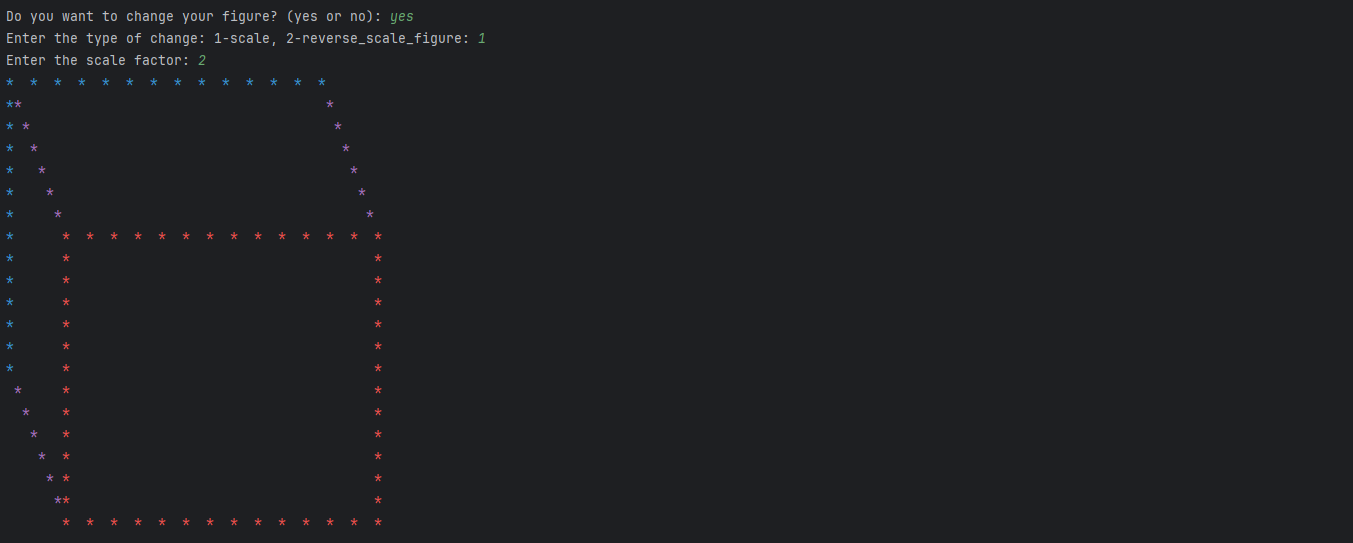


Рис.2

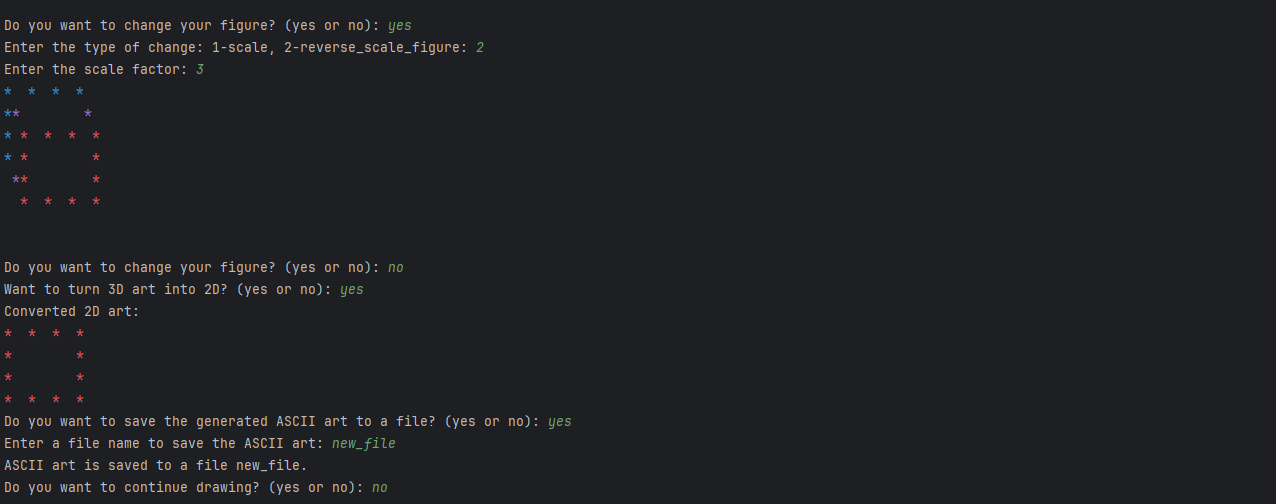


Рис.3

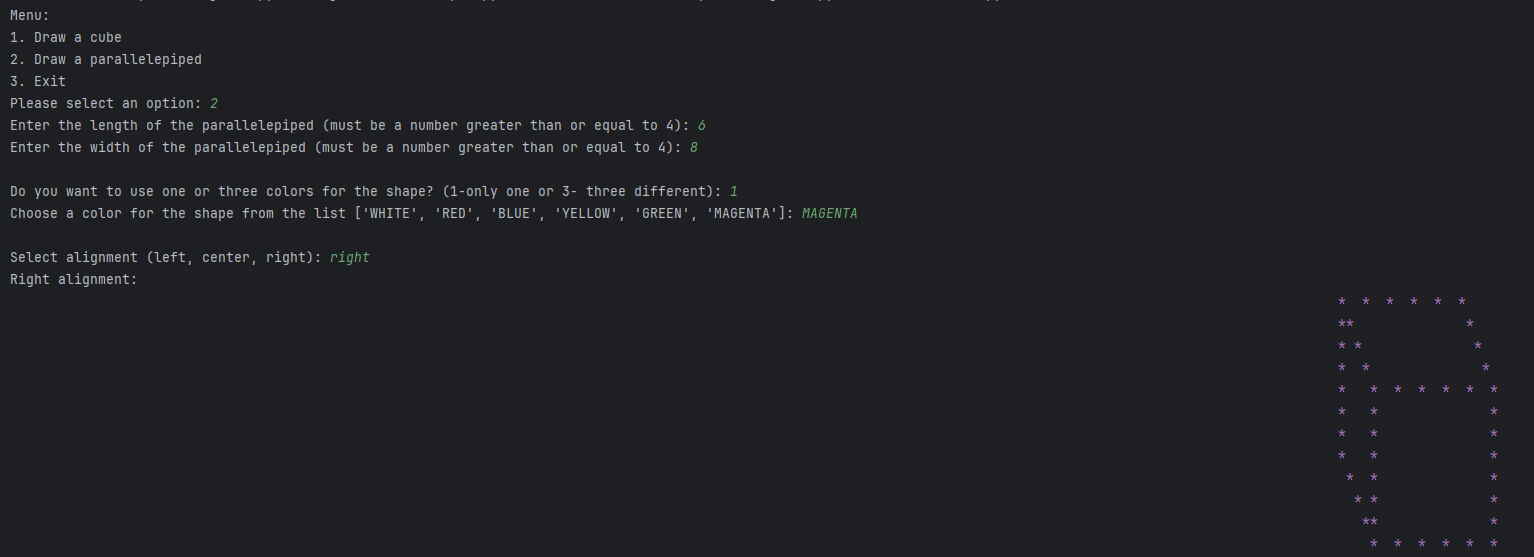
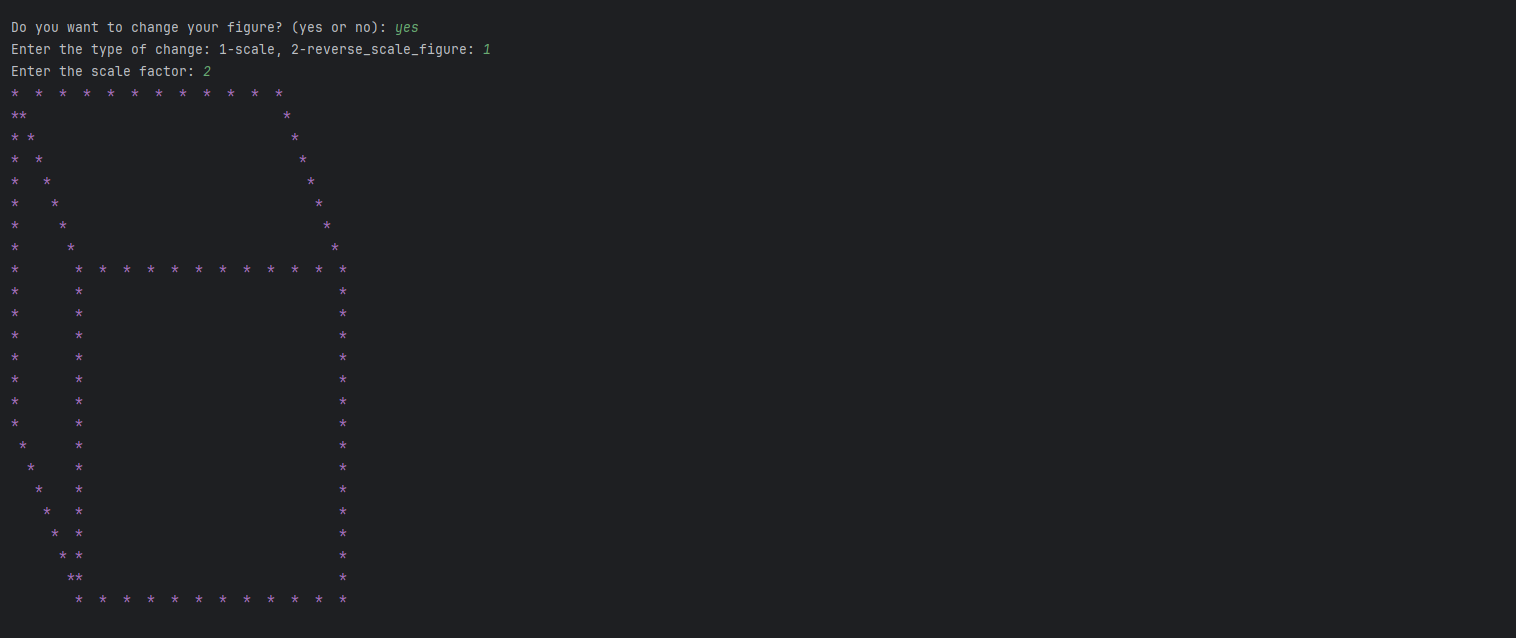


Рис.4

 Рис.5

**Посилання на репозиторій**: <https://github.com/Oleksandr2004Oleniuk/SMP.git>

**Висновок**. у цій лабораторній роботі я створив 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, сприяв творчому підходу до створення ASCII-арту.