

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра інформаційних систем та мереж

Звіт

до лабораторної роботи №5

Розробка ASCII ART генератора для візуалізації 3D-фігур

Виконала:

ст. гр. РІ-32
Довгошия А.А

Балів	Дата

Прийняв:

асис. каф. ІСМ

Щербак С.С

Львів — 2024

Мета: Створення 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.

Art_ui.py

"""

This module defines the ArtUI class, which provides an interface for

generating and saving ASCII art based on user preferences.

Users can select text, font, color, justification, and width for the ASCII art.

```
"""
```

```
from src.classes.lab3.generate_art import AsciiArtGenerator
from src.const.const import (
    FONT_CHOICE_PROMPT, COLOR_CHOICE_PROMPT,
    JUSTIFY_CHOICE_PROMPT,
    WIDTH_CHOICE_PROMPT, SAVE_CHOICE_PROMPT,
    FILE_NAME_PROMPT,
    USER_ASCII_TEXT_PROMPT
)
```

```
from lib import FileOperations, UserInput
```

```
class ArtUI:
```

```
    """
```

```
    Interface class for managing user input, generating ASCII art,
    and providing options to save the artwork.
```

```
    """
```

```
    def __init__(self, config):
```

```
        """Initialize ArtUI with a configuration and ASCII art generator."""
```

```
        self.config = config
```

```
        self.art_generator = AsciiArtGenerator(self.config)
```

```
    def get_text_input(self):
```

```

        """Prompt the user to input text for ASCII art generation."""
        return input(USER_ASCII_TEXT_PROMPT)

def get_font_choice(self):
    """Prompt the user to choose a font from available options."""
    return UserInput.get_input(FONT_CHOICE_PROMPT, self.config.fonts)

def get_color_choice(self):
    """Prompt the user to choose a color from available options."""
    return UserInput.get_input(COLOR_CHOICE_PROMPT,
list(self.config.colors.keys()))

def get_justification_choice(self):
    """Prompt the user to select text justification."""
    return UserInput.get_input(JUSTIFY_CHOICE_PROMPT,
self.config.justify)

def get_width_input(self):
    """Prompt the user to input a width for the ASCII art."""
    return UserInput.get_integer_input(WIDTH_CHOICE_PROMPT, 10, 175)

def generate_ascii_art(self):
    """Generate ASCII art based on user preferences for text, font, color, and
alignment."""
    text = self.get_text_input()
    font = self.get_font_choice()
    color = self.get_color_choice()
    justify = self.get_justification_choice()

```

```

width = self.get_width_input()

ascii_art = self.art_generator.generate_art(text, color, font, width, justify)

print(f"{self.config.colors.get(color)}{ascii_art}{self.config.colors.get('reset')}")

self.prompt_to_save_ascii_art(ascii_art)

def prompt_to_save_ascii_art(self, ascii_art):
    """Prompt the user to decide if the ASCII art should be saved."""
    save_choice = input(SAVE_CHOICE_PROMPT).strip().lower()
    if save_choice in ('yes', 'y'):
        self.save_ascii_art(ascii_art)

def save_ascii_art(self, ascii_art):
    """Save ASCII art to a specified file."""
    filename = input(FILE_NAME_PROMPT).strip()
    if not filename.endswith('.txt'):
        filename += '.txt'

    FileOperations.save_art(ascii_art, self.config.file_path, filename)

def start_ui(self):
    """Initiate the ASCII art generation process."""
    self.generate_ascii_art()

```

cube.py

```

from src.classes.lab5.bresenham import Bresenham

```

```
class Cube:
```

```
    def __init__(self, size):
```

```
        self.x = 5
```

```
        self.y = 5
```

```
        self.z = 5
```

```
        self.size = size
```

```
    def draw(self):
```

```
        vertices = [
```

```
            (self.x, self.y, self.z),
```

```
            (self.x + self.size, self.y, self.z),
```

```
            (self.x + self.size, self.y + self.size, self.z),
```

```
            (self.x, self.y + self.size, self.z),
```

```
            (self.x, self.y, self.z + self.size),
```

```
            (self.x + self.size, self.y, self.z + self.size),
```

```
            (self.x + self.size, self.y + self.size, self.z + self.size),
```

```
            (self.x, self.y + self.size, self.z + self.size)
```

```
        ]
```

```
        edges = [
```

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

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

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

```
        ]
```

```
        points = []
```

```
        for edge in edges:
```

```

        start, end = vertices[edge[0]], vertices[edge[1]]
        x1, y1 = start[0] - start[2] // 2 + self.size // 2, start[1] - start[2] // 2 +
self.size // 2
        x2, y2 = end[0] - end[2] // 2 + self.size // 2, end[1] - end[2] // 2 + self.size
// 2
        points += Bresenham.draw(x1, y1, x2, y2)

    return points

```

bresenham.py

```
class Bresenham:
```

```
    @staticmethod
```

```
    def draw(x1, y1, x2, y2):
```

```
        points = []
```

```
        dx = x2 - x1
```

```
        dy = y2 - y1
```

```
        sx = 1 if dx > 0 else -1
```

```
        sy = 1 if dy > 0 else -1
```

```
        dx = abs(dx)
```

```
        dy = abs(dy)
```

```
    if dx > dy:
```

```
        err = dx / 2.0
```

```
        while x1 != x2:
```

```
            points.append((x1, y1))
```

```
            err -= dy
```

```
            if err < 0:
```

```
                y1 += sy
```



```

        err += dx
        x1 += sx
        points.append((x2, y2))
    else:
        err = dy / 2.0
        while y1 != y2:
            points.append((x1, y1))
            err -= dx
            if err < 0:
                x1 += sx
                err += dy
            y1 += sy
        points.append((x2, y2))

    return points

```

generate_shape.py

```

from src.classes.lab5.cube import Cube
from src.ui.lab3.art_ui import ArtUI
from src.classes.lab3.generate_art import AsciiArtGenerator
from lib.user_input import UserInput

```

```

class AsciiArtGeneratorLab5:

```

```

    def __init__(self, config):
        self.config = config
        self.art_ui = ArtUI(self.config)
        self.ascii_art_generator = AsciiArtGenerator(self.config)
        self.user_input = UserInput()

```

```

def start_ui(self):
    self.generate_ascii_art()

def generate_ascii_art(self):

    size = self.user_input.get_integer_input("Choose a size: ", 5, 20)
    color = self.art_ui.get_color_choice()
    justify = self.art_ui.get_justification_choice()

    cube = Cube(size)
    cube_points = cube.draw()

    ascii_art = self.generate_ascii_art_from_points(cube_points)

    justified_ascii_art = self.ascii_art_generator.apply_justification(ascii_art,
justify)

    print(f"{self.config.colors.get(color)}{justified_ascii_art}{self.config.colors.get(
'reset')}")

    self.art_ui.prompt_to_save_ascii_art(ascii_art)

def generate_ascii_art_from_points(self, points):
    grid_size = 35
    grid = [['.' for _ in range(grid_size)] for _ in range(grid_size)]

```

```

for x, y in points:
    if 0 <= x < grid_size and 0 <= y < grid_size:
        grid[y][x] = '#'

ascii_art = '\n'.join("".join(row) for row in grid)
return ascii_art

```

```

Choose a size: 10
# choose a color: red
# choose a justification: center

```

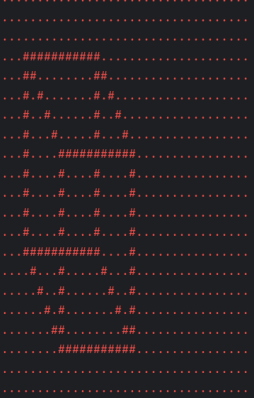


Рис.1. Результат програми.

Висновок: на цій лабораторній роботі я вивчила створення додатка 3-D Генератора ASCII-арту на основі об'єктно - орієнтованого підходу та мови Python