МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра інформаційних систем та мереж

Лабораторна робота №5

з дисципліни

СПЕЦІАЛІЗОВАНІ МОВИ ПРОГРАМУВАННЯ

на тему

Розробка ASCII ART генератора для візуалізації 3D-фігур

Виконав:

ст. гр. РІ-21сп

Костецький І.С.

Прийняв:

Шербак С.С.

Львів-2024

**Мета лабораторної 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.

**Виконання роботи**

**Папка Classes/Shapes:**

**Вміст файлу Shape3D.py:**

import math

class Shape3D:

"""Base class for 3D shapes."""

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

self.size = size

self.angle\_x = 0

self.angle\_y = 0

self.angle\_z = 0

def rotate(self, angle\_x=0, angle\_y=0, angle\_z=0):

"""Sets rotation angles."""

self.angle\_x += math.radians(angle\_x)

self.angle\_y += math.radians(angle\_y)

self.angle\_z += math.radians(angle\_z)

def apply\_rotation(self, x, y, z):

"""Applies rotation to a point (x, y, z) based on set angles."""

# Rotate around X-axis

y, z = (

y \* math.cos(self.angle\_x) - z \* math.sin(self.angle\_x),

y \* math.sin(self.angle\_x) + z \* math.cos(self.angle\_x)

)

# Rotate around Y-axis

x, z = (

x \* math.cos(self.angle\_y) + z \* math.sin(self.angle\_y),

-x \* math.sin(self.angle\_y) + z \* math.cos(self.angle\_y)

)

# Rotate around Z-axis

x, y = (

x \* math.cos(self.angle\_z) - y \* math.sin(self.angle\_z),

x \* math.sin(self.angle\_z) + y \* math.cos(self.angle\_z)

)

return int(x), int(y), int(z)

def draw\_line(self, x1, y1, x2, y2, canvas, symbol):

"""Draws a line between two points on the canvas, skipping only for horizontal lines."""

dx = abs(x2 - x1)

dy = abs(y2 - y1)

sx = 1 if x1 < x2 else -1

sy = 1 if y1 < y2 else -1

err = dx - dy

step\_count = 0

while True:

# Skip symbols only for horizontal lines (y1 == y2)

if not (y1 == y2 and step\_count % 2 != 0): # draw continuously for vertical and diagonal lines

if 0 <= y1 < len(canvas) and 0 <= x1 < len(canvas[0]):

canvas[int(y1)][int(x1)] = symbol

step\_count += 1

if x1 == x2 and y1 == y2:

break

e2 = err \* 2

if e2 > -dy:

err -= dy

x1 += sx

if e2 < dx:

err += dx

y1 += sy

def project\_to\_2d(self):

"""Abstract method for projecting the shape from 3D to 2D."""

raise NotImplementedError("The project\_to\_2d method must be implemented in a subclass.")

**Вміст файлу Pyramid.py:**

from Classes.Shapes.Shape3D import Shape3D

class Pyramid(Shape3D):

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

super().\_\_init\_\_(size)

def project\_to\_2d(self):

projection\_height = self.size \* 2

projection\_width = self.size \* 2

projection = [[" " for \_ in range(int(projection\_width))] for \_ in range(int(projection\_height))]

half = self.size // 2

vertices = [

(0, -half, half), (half, half, half),

(-half, half, half), (0, 0, -half)

]

# Обертання для кожної вершини

rotated\_vertices = [self.apply\_rotation(x, y, z) for x, y, z in vertices]

edges = [

(0, 1), (1, 2), (2, 0), # Основа піраміди

(0, 3), (1, 3), (2, 3) # Ребро до вершини

]

center\_x = projection\_width // 2 # Центрування по ширині

min\_y = min(rotated\_vertices, key=lambda v: v[1])[1]

max\_y = max(rotated\_vertices, key=lambda v: v[1])[1]

vertical\_offset = (projection\_height // 2) - ((max\_y + min\_y) // 2) # Центрування по висоті

# Рисувати ребра піраміди

for start, end in edges:

x1, y1, z1 = rotated\_vertices[start]

x2, y2, z2 = rotated\_vertices[end]

x1 += center\_x

y1 += vertical\_offset

x2 += center\_x

y2 += vertical\_offset

symbol = "." if z1 > 0 and z2 > 0 else "#"

self.draw\_line(int(x1), int(y1), int(x2), int(y2), projection, symbol)

return projection

**Вміст файлу Cube.py:**  
from Classes.Shapes.Shape3D import Shape3D

class Cube(Shape3D):

"""Class for a cube."""

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

super().\_\_init\_\_(size)

def project\_to\_2d(self):

projection\_height = self.size \* 2

projection\_width = self.size \* 2

projection = [[" " for \_ in range(int(projection\_width \* 2))] for \_ in range(int(projection\_height))]

half = self.size // 2

vertices = [

(-half, -half, -half), (half, -half, -half),

(half, half, -half), (-half, half, -half),

(-half, -half, half), (half, -half, half),

(half, half, half), (-half, half, half),

]

rotated\_vertices = [self.apply\_rotation(x, y, z) for x, y, z in vertices]

edges = [

(0, 1), (1, 2), (2, 3), (3, 0), # Нижня грань

(4, 5), (5, 6), (6, 7), (7, 4), # Верхня грань

(0, 4), (1, 5), (2, 6), (3, 7) # Вертикальні ребра

]

center\_x = projection\_width // 2 # Центрування по ширині

min\_y = min(v[1] for v in rotated\_vertices)

max\_y = max(v[1] for v in rotated\_vertices)

vertical\_offset = (projection\_height // 2) - ((max\_y + min\_y) // 2) # Центрування по висоті

# Рисувати ребра куба

for start, end in edges:

x1, y1, z1 = rotated\_vertices[start]

x2, y2, z2 = rotated\_vertices[end]

x1 += center\_x

y1 += vertical\_offset

x2 += center\_x

y2 += vertical\_offset

symbol = "." if z1 > 0 and z2 > 0 else "#"

self.draw\_line(x1 \* 2, y1, x2 \* 2, y2, projection, symbol)

return projection

**Папка Classes:**

**Вміст файлу ascii\_art\_colorizer.py:**

import json

import os

class Colorizer:

def \_\_init\_\_(self, config\_path=os.path.join(os.path.dirname(\_\_file\_\_), "../Config/colors.json")):

print(config\_path)

self.\_color = None

with open(config\_path, 'r') as config\_file:

self.\_config = json.load(config\_file)

self.\_colors = self.\_config["colors"]

def colorize(self, shape, color):

if color in self.\_colors:

return f"{self.\_colors[color]}{shape}{self.\_colors['reset']}"

return shape

def update\_color(self):

self.\_color = input("Введіть колір: ")

def get\_color(self):

return self.\_color

def list\_colors(self, num\_columns=4):

color\_names = [name for name in self.\_colors if name != "reset"]

max\_name\_length = max(len(name) for name in color\_names) + 2

for i in range(0, len(color\_names), num\_columns):

row = color\_names[i:i + num\_columns]

formatted\_row = [

self.colorize(name.ljust(max\_name\_length), name) for name in row

]

print("".join(formatted\_row))

**Вміст файлу ascii\_art\_displayer.py:**

from Classes.ascii\_art\_colorizer import Colorizer

class ArtDisplayer:

def \_\_init\_\_(self):

self.colorizer = Colorizer()

def display\_ascii\_art(self, shape, color):

projection = shape.project\_to\_2d()

for row in projection:

projection\_row = "".join(row)

if color:

projection\_row = self.colorizer.colorize(projection\_row, color)

print(projection\_row)

**Вміст файлу ascii\_generator.py:**

from Classes.Shapes.Cube import Cube

from Classes.Shapes.Pyramid import Pyramid

class ASCIIGenerator:

def \_\_init\_\_(self):

self.\_shape = None

self.\_shape\_type = "cube"

self.\_size = 20

self.\_angle\_x = 21

self.\_angle\_y = 10

self.\_angle\_z = 0

self.set\_shape(self.\_shape\_type, self.\_size)

self.rotate\_shape(self.\_angle\_x, self.\_angle\_y, self.\_angle\_z)

def set\_shape\_type(self, shape\_type):

self.\_shape\_type = shape\_type

def get\_shape\_type(self):

return self.\_shape\_type

def set\_shape(self, shape\_type, size):

self.\_shape\_type = shape\_type

self.\_size = size

if shape\_type == "cube":

self.\_shape = Cube(size)

elif shape\_type == "pyramid":

self.\_shape = Pyramid(size)

def get\_shape(self):

return self.\_shape

def set\_size(self, size):

self.\_size = size

def get\_size(self):

return self.\_size

def rotate\_shape(self, angle\_x, angle\_y, angle\_z):

if self.\_shape:

self.\_shape.rotate(angle\_x, angle\_y, angle\_z)

def set\_x(self, x):

self.\_angle\_x = x

def set\_y(self, y):

self.\_angle\_y = y

def set\_z(self, z):

self.\_angle\_z = z

def get\_x(self):

return self.\_angle\_x

def get\_y(self):

return self.\_angle\_y

def get\_z(self):

return self.\_angle\_z

**Вміст файлу FileManager.py:**

import os

class FileManager:

def \_\_init\_\_(self, data\_directory="Data/"):

self.\_data\_directory = data\_directory

if not os.path.exists(self.\_data\_directory):

os.makedirs(self.\_data\_directory)

def save\_to\_file(self, projection, filename):

file\_path = os.path.join(self.\_data\_directory, filename)

with open(file\_path, 'w') as f:

for row in projection:

f.write("".join(row) + "\n")

print(f"ASCII art збережено у {file\_path}")

**Вміст файлу user\_input\_handler.py:**

from Classes.ascii\_generator import ASCIIGenerator

class UserInputManager:

def \_\_init\_\_(self):

self.ascii\_art\_generator = ASCIIGenerator()

def update\_shape(self):

try:

shape\_type = input("Виберіть фігуру (cube/pyramid): ").strip().lower()

if shape\_type not in ['cube', 'pyramid']:

raise ValueError("Невідома фігура.")

self.ascii\_art\_generator.set\_shape\_type(shape\_type)

except ValueError as e:

print(e)

def update\_size(self):

try:

size = int(input(f"Введіть розмір фігури (7-50, поточний: {self.ascii\_art\_generator.get\_size()} ): ") or self.ascii\_art\_generator.get\_size())

if 7 <= size <= 50:

self.ascii\_art\_generator.set\_size(size)

else:

print("Розмір має бути від 7 до 50.")

except ValueError:

print("Будь ласка, введіть ціле число.")

def update\_x(self):

try:

angle\_x = float(input(f"Введіть обертання по X° (-180° до 180°, поточний: {self.ascii\_art\_generator.get\_x()}): ") or self.ascii\_art\_generator.get\_x())

if -180 <= angle\_x <= 180:

self.ascii\_art\_generator.set\_x(angle\_x)

else:

print("Кут обертання має бути від -180° до 180°.")

except ValueError:

print("Будь ласка, введіть число.")

def update\_y(self):

try:

angle\_y = float(input(f"Введіть обертання по Y° (-180° до 180°, поточний: {self.ascii\_art\_generator.get\_y()} ): ") or self.ascii\_art\_generator.get\_y())

if -180 <= angle\_y <= 180:

self.ascii\_art\_generator.set\_y(angle\_y)

else:

print("Кут обертання має бути від -180° до 180°.")

except ValueError:

print("Будь ласка, введіть число.")

def update\_z(self):

try:

angle\_z = float(input(f"Введіть обертання по Z° (-180° до 180°, поточний: {self.ascii\_art\_generator.get\_z()}): ") or self.ascii\_art\_generator.get\_z())

if -180 <= angle\_z <= 180:

self.ascii\_art\_generator.set\_z(angle\_z)

else:

print("Кут обертання має бути від -180° до 180°.")

except ValueError:

print("Будь ласка, введіть число.")

**Вміст файлу user\_input\_handler.py:**

from Classes.FileManager import FileManager

from Classes.ascii\_art\_displayer import ArtDisplayer

from Classes.user\_input\_handler import UserInputManager

class AsciiArtService:

def \_\_init\_\_(self):

self.file\_manager = FileManager()

self.art\_displayer = ArtDisplayer()

self.user\_input\_manager = UserInputManager()

self.ascii\_generator = self.user\_input\_manager.ascii\_art\_generator

def update\_shape(self):

return self.user\_input\_manager.update\_shape()

def update\_size(self):

return self.user\_input\_manager.update\_size()

def update\_color(self):

self.art\_displayer.colorizer.list\_colors()

self.art\_displayer.colorizer.update\_color()

def display\_ascii\_art(self):

self.ascii\_generator.set\_shape(self.ascii\_generator.get\_shape\_type(), self.ascii\_generator.get\_size())

self.ascii\_generator.rotate\_shape(self.ascii\_generator.get\_x(), self.ascii\_generator.get\_y(),

self.ascii\_generator.get\_z())

return self.art\_displayer.display\_ascii\_art(self.ascii\_generator.get\_shape(),

self.art\_displayer.colorizer.get\_color())

def save\_to\_file(self):

filename = input("Введіть назву файла у який зберегти ASCII art (наприклад, art.txt): ")

self.file\_manager.save\_to\_file(self.ascii\_generator.get\_shape().project\_to\_2d(), filename)

def update\_x(self):

return self.user\_input\_manager.update\_x()

def update\_y(self):

return self.user\_input\_manager.update\_y()

def update\_z(self):

return self.user\_input\_manager.update\_z()

**Вміст файлу Config/colors.json:**

{

"colors": {

"black": "\u001b[30m",

"red": "\u001b[31m",

"green": "\u001b[32m",

"yellow": "\u001b[33m",

"blue": "\u001b[34m",

"magenta": "\u001b[35m",

"cyan": "\u001b[36m",

"grey": "\u001b[37m",

"reset": "\u001b[0m",

"bright\_black": "\u001b[90m",

"bright\_red": "\u001b[91m",

"bright\_green": "\u001b[92m",

"bright\_yellow": "\u001b[93m",

"bright\_blue": "\u001b[94m",

"bright\_magenta": "\u001b[95m",

"bright\_cyan": "\u001b[96m",

"white": "\u001b[97m",

"bg\_black": "\u001b[40m",

"bg\_red": "\u001b[41m",

"bg\_green": "\u001b[42m",

"bg\_yellow": "\u001b[43m",

"bg\_blue": "\u001b[44m",

"bg\_magenta": "\u001b[45m",

"bg\_cyan": "\u001b[46m",

"bg\_grey": "\u001b[47m",

"bg\_bright\_black": "\u001b[100m",

"bg\_bright\_red": "\u001b[101m",

"bg\_bright\_green": "\u001b[102m",

"bg\_bright\_yellow": "\u001b[103m",

"bg\_bright\_blue": "\u001b[104m",

"bg\_bright\_magenta": "\u001b[105m",

"bg\_bright\_cyan": "\u001b[106m",

"bg\_white": "\u001b[107m"

}

}

**Вміст файлу UI/cli.py:**

import sys

import os

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "..")))

from Classes.ascii\_art\_service import AsciiArtService

def user\_input\_interface():

ascii\_art\_service = AsciiArtService()

while True:

ascii\_art\_service.display\_ascii\_art()

print("\nМеню:")

print("1. Вибрати фігуру (cube/pyramid)")

print("2. Встановити розмір")

print("3. Обновити колір")

print("4. Кут X ")

print("5. Кут Y")

print("6. Кут Z")

print("7. Зберегти у файл")

print("9. Вийти")

choice = input("Виберіть опцію: ").strip()

match choice:

case '1':

ascii\_art\_service.update\_shape()

case '2':

ascii\_art\_service.update\_size()

case '3':

ascii\_art\_service.update\_color()

case '4':

ascii\_art\_service.update\_x()

case '5':

ascii\_art\_service.update\_y()

case '6':

ascii\_art\_service.update\_z()

case '7':

ascii\_art\_service.save\_to\_file()

case '9':

print("Вихід з програми...")

break

case \_:

print("Неправильний вибір.")

**В корені лабораторної є файл run.py:**

from UI import cli

if \_\_name\_\_ == '\_\_main\_\_':

cli.user\_input\_interface()

Результататом виконання є меню з можливістю міняти кут чи розмір фігури (куб або піраміда) та надана можливість вибору кольору і збереження у файл.

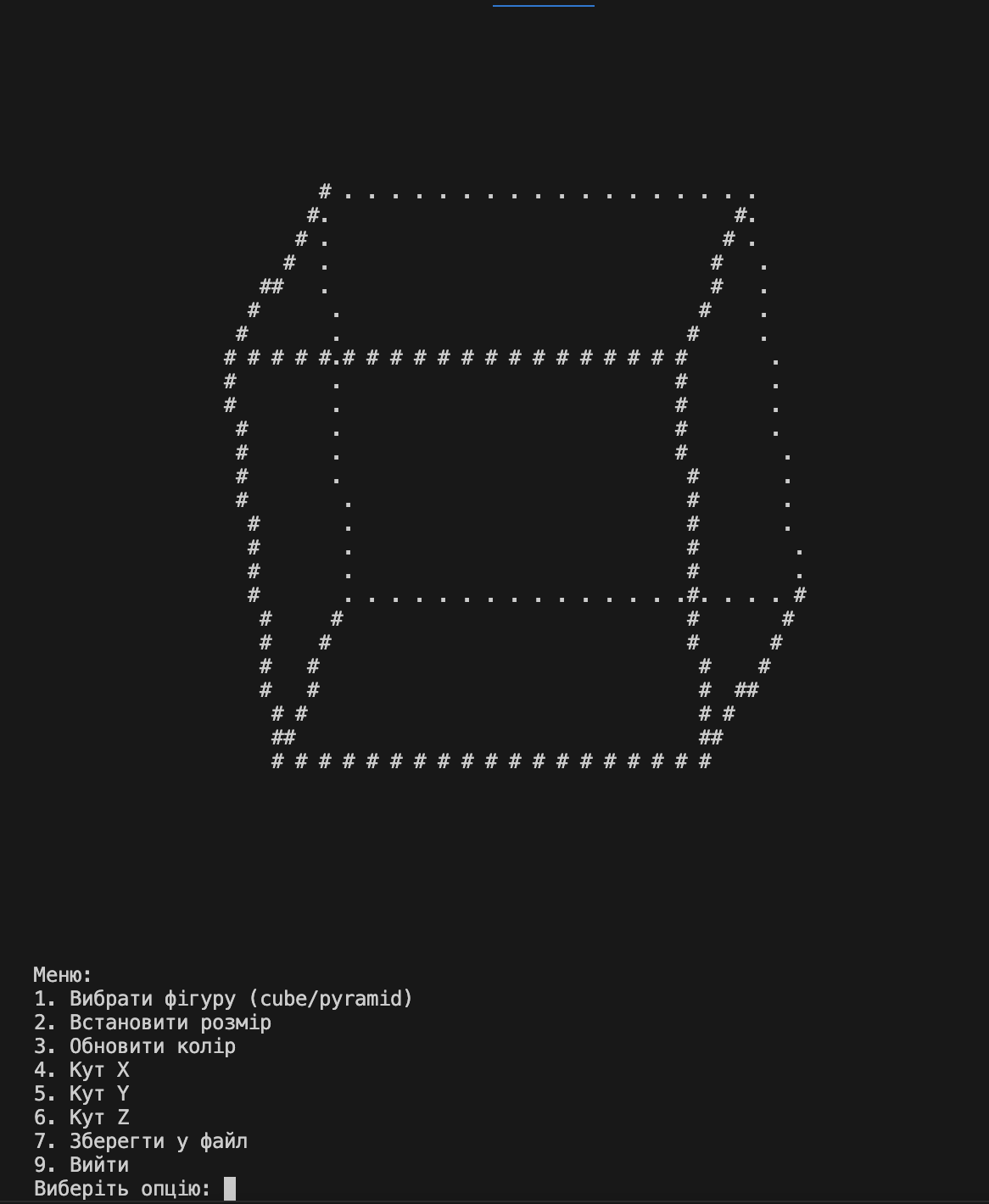


Рисунок 1 – Вивід усіх діаграм

**Висновок:** У ході виконання лабораторної роботи я створив додаток для малювання 3D-фігур у ASCII-арт з використанням ООП. Дана програма генерує ASCII зображення 3D фігури, та надає можливість задавати для неї колір або ж крутити нею у просторі.