МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Кафедра інформаційних систем та мереж

Лабораторна робота №5

з курсу

СПЕЦІАЛІЗОВАНІ МОВИ ПРОГРАМУВАННЯ

Виконав студент

групи ІТ-21сп

**Гузар А.І.**

Прийняв

**Щербак С.С.**

Львів - 2023

**Мета:** створення 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 виконаних завдань представлено нижче.

Клас Rectangle:

# from Data.lab\_5.Figure import Figure

from Data.lab\_5.Figure import Figure

class Rectangle(Figure):

@staticmethod

def build\_up\_2d\_line(self, color\_1, color\_2, symbol\_1, symbol\_2):

line = ''

for size\_x in range(self.size\_x + 1):

if size\_x == 0 or size\_x == self.size\_x:

line += self.color\_line(self, color\_1, symbol\_1)

else:

line += self.color\_line(self, color\_2, symbol\_2)

return line

@staticmethod

def build\_up\_2d(self, lines, symbol, space\_color):

for index in range(len(lines)):

if index == 0:

color\_1 = self.\_symbol\_color

color\_2 = self.\_symbol\_color

symbol\_1 = self.\_SYMBOLS.get('corner')

symbol\_2 = self.\_SYMBOLS.get('horizontal')

else:

color\_1 = self.\_symbol\_color

color\_2 = space\_color

symbol\_1 = symbol

symbol\_2 = self.\_SYMBOLS.get('space')

line = self.build\_up\_2d\_line(self, color\_1, color\_2, symbol\_1, symbol\_2)

lines[index] = lines[index] + line

return lines

@staticmethod

def build\_3d(self, lines):

inclined\_count = 1

for index in range(len(lines)):

line = ''

mx = (len(lines[0]))

shadow\_mx = mx + self.size\_x

shadow = ''

if index != 0 and index != len(lines) - 1:

count = mx - len(lines[index]) - 1

if index < self.size\_y:

symbol = self.\_SYMBOLS.get('vertical')

elif index == self.size\_y:

symbol = self.\_SYMBOLS.get('corner')

else:

count -= inclined\_count

shadow\_count = shadow\_mx - len(lines[index]) - 1

shadow = self.build\_symbol(self.\_SYMBOLS.get('space'), shadow\_count, self.\_shadow\_color)

symbol = self.\_SYMBOLS.get('inclined')

inclined\_count += 1

line = self.build\_symbol(self.\_SYMBOLS.get('space'), count, self.\_color\_3)

line += self.color\_line(self, self.\_symbol\_color, symbol)

line += self.color\_line(self, self.\_symbol\_color, shadow)

lines[index] += line

return lines

def build(self):

self.set\_zoom()

shadow\_up\_lines = self.build\_2d\_shadow(self, self.size\_z, self.\_SYMBOLS.get('space'), self.size\_z)

shadow\_down\_lines = self.build\_2d\_shadow(self, self.size\_y, self.\_SYMBOLS.get('nothing'), self.size\_z)

up\_2d = self.build\_up\_2d(self, shadow\_up\_lines, self.\_SYMBOLS.get('inclined'), self.\_color\_1)

down\_2d = self.build\_up\_2d(self, shadow\_down\_lines, self.\_SYMBOLS.get('vertical'), self.\_color\_2)

down\_2d += [down\_2d[0]]

lines = up\_2d + down\_2d

result = self.build\_3d(self, lines)

self.\_result = self.convert(result)

Клас Pyramid:

from Data.lab\_5.Figure import Figure

class Pyramid(Figure):

def set\_length\_x(self):

self.length\_x = self.size\_x + self.size\_y

def set\_length\_y(self):

self.length\_y = self.size\_z

@staticmethod

def build\_up\_2d\_line(self, color\_1, color\_2, symbol\_1, symbol\_2, count):

line = ''

for size\_x in range(count):

if size\_x == 0:

line += self.color\_line(self, color\_1, symbol\_1)

else:

line += self.color\_line(self, color\_2, symbol\_2)

return line

@staticmethod

def build\_up\_2d(self, lines, symbol, space\_color):

mx = len(lines[0])

for index in range(len(lines)):

if index == 0:

color\_1 = self.\_symbol\_color

color\_2 = ''

symbol\_1 = self.\_SYMBOLS.get('corner')

symbol\_2 = ''

count = 1

else:

color\_1 = self.\_symbol\_color

color\_2 = space\_color

symbol\_1 = symbol

symbol\_2 = self.\_SYMBOLS.get('space')

count = mx - len(lines[index])

line = self.build\_up\_2d\_line(self, color\_1, color\_2, symbol\_1, symbol\_2, count)

lines[index] = lines[index] + line

return lines

@staticmethod

def build\_3d(self, lines):

inclined\_count = 0

mx = 0

count = 0

for index in range(len(lines)):

line = ''

shadow\_mx = mx + self.size\_x

shadow = ''

if index != 0 and index != len(lines) - 1:

if index < self.size\_y:

count = index - 1

symbol = '\\'

elif index == self.size\_y:

mx = count

count = 0

symbol = self.\_SYMBOLS.get('nothing')

else:

count = mx - inclined\_count

shadow\_count = shadow\_mx - len(lines[index]) - 1

shadow = self.build\_symbol(self.\_SYMBOLS.get('space'), shadow\_count, self.\_shadow\_color)

symbol = self.\_SYMBOLS.get('inclined')

inclined\_count += 1

line = self.build\_symbol(self.\_SYMBOLS.get('space'), count, self.\_color\_2)

line += self.color\_line(self, self.\_symbol\_color, symbol)

line += self.color\_line(self, self.\_symbol\_color, shadow)

lines[index] += line

return lines

def build\_footer(self):

line = ''

for index in range(self.size\_y \* 2 + 1):

if index == 0 or index == (self.size\_y \* 2):

symbol = '+'

color = self.\_color\_1

elif index == self.size\_y:

symbol = '|'

color = self.\_color\_1

else:

symbol = ' '

color = self.\_color\_1

line += self.color\_line(self, self.\_symbol\_color, symbol)

return line

def build(self):

self.set\_zoom()

shadow\_up\_lines = self.build\_2d\_shadow(self, self.size\_y, self.\_SYMBOLS.get('space'), self.size\_y)

first\_list = self.build\_up\_2d(self, shadow\_up\_lines, self.\_SYMBOLS.get('inclined'), self.\_color\_1)

for index in range(len(first\_list)):

if index != 0:

line = self.color\_line(self, self.\_default\_color, self.\_SYMBOLS.get('vertical'), )

else:

line = ''

first\_list[index] += line

second\_list = [s.replace('/', '\\') for s in first\_list]

second\_list += [self.build\_footer()]

second\_list = list(reversed(second\_list))

lst = first\_list + second\_list

result = self.build\_3d(self, lst)

self.\_result = self.convert(result)

Клас Figure:

class Figure:

length\_x = 0

length\_y = 0

\_result = ''

\_color\_1 = '\033[44m'

\_color\_2 = '\033[42m'

\_color\_3 = '\033[43m'

\_default\_color = '\033[00m'

\_shadow\_color = '\033[47m'

\_symbol\_color = '\033[36m'

\_space = 0

\_zoom = 1

size\_x = None

size\_y = None

size\_z = None

\_COLORS = {

'RED': '\033[31m',

'GREEN': '\033[32m',

'YELLOW': '\033[33m',

'BLUE': '\033[34m',

'MAGENTA': '\033[35m',

'CYAN': '\033[36m',

'WHITE': '\033[37m',

}

\_SYMBOLS = {

'corner': '+',

'horizontal': '—',

'vertical': '|',

'inclined': '/',

'space': ' ',

'nothing': '',

}

def \_\_init\_\_(self, \*\*kwargs):

if 'size\_y' in kwargs:

self.size\_y = kwargs['size\_y']

if 'size\_x' in kwargs:

self.size\_x = kwargs['size\_x']

if 'size\_z' in kwargs:

self.size\_z = kwargs['size\_z']

if self.size\_x:

if not self.size\_y or not self.size\_z:

self.size\_y = self.size\_z = self.size\_x

self.set\_length\_x()

self.set\_length\_y()

else:

self.size\_y = self.size\_z = self.size\_x = 0

def set\_length\_x(self):

self.length\_x = self.length\_x

def set\_length\_y(self):

self.length\_y = self.length\_y

def set\_zoom(self):

self.size\_x \*= self.\_zoom

self.size\_y \*= self.\_zoom

self.size\_z \*= self.\_zoom

@staticmethod

def color\_line(self, color, line):

return color + line + self.\_default\_color

@staticmethod

def build\_2d\_shadow(self, count, symbol, size):

lines = []

for index in range(count):

line = self.build\_symbol(self.\_SYMBOLS.get('space'), self.\_space, self.\_default\_color)

line += self.color\_line(self, self.\_default\_color, symbol \* (size - index))

lines += [line]

return lines

def build\_symbol(self, symbol, count, color):

line = ''

for i in range(count):

line += self.color\_line(self, color, symbol)

return line

def build(self):

self.\_result = self.\_result

def create(self):

self.set\_zoom()

self.build()

def save(self, filename):

self.create()

result = self.remove\_color\_codes(self.\_result)

# print(result)

with open(filename, 'w') as file:

file.write(result)

def remove\_color\_codes(self, text):

while '\033[' in text:

start = text.find('\033[')

end = text.find('m', start)

if end != -1:

text = text[:start] + text[end+1:]

else:

break

return text

@staticmethod

def convert(lines):

result = ''

for line in lines:

result += line + '\n'

return result

def \_\_str\_\_(self):

return self.\_result

Клас ConsoleRectangle:

from Data.lab\_5.Rectangle import Rectangle

class ConsoleRectangle(Rectangle):

figure = Rectangle

@classmethod

def set\_parameters(cls, dct, input\_message):

for key, value in dct.items():

print(f"{key}: {value}")

value = input(f"Select {input\_message}: ")

current\_value = dct.get(value, None)

if current\_value:

return current\_value

else:

return None

def input\_function(self, dct, input\_message, default):

check = input(f"do you want to set {input\_message}? (1/0): ")

font = None

if check == '1':

font = self.set\_parameters(dct, input\_message)

if font:

return font

else:

return default

def configuration(self):

self.size\_x = int(input("Input size\_x: "))

self.size\_y = int(input("Input size\_y: "))

self.size\_z = int(input("Input size\_z: "))

self.\_color\_1 = self.input\_function(self.\_COLORS, 'color\_1', self.\_color\_1)

self.\_color\_2 = self.input\_function(self.\_COLORS, 'color\_2', self.\_color\_2)

self.\_color\_3 = self.input\_function(self.\_COLORS, 'color\_3', self.\_color\_3)

change\_space = input('do you want to create space? (1/0): ')

if change\_space == '1':

space = int(input("Input space: "))

self.\_space(space)

change\_zoom = input('do you want to create zoom? (1/0): ')

if change\_zoom == '1':

self.\_zoom = int(input("Input zoom: "))

self.set\_zoom()

check = input("do you want to save? (1/0): ")

if check == '1':

save = input("Input filename: ")

self.save(save)

def run(self):

stop = ''

while stop != 'f':

try:

self.configuration()

self.create()

print(self)

except Exception as e:

print(e)

finally:

stop = input("Press 'f' if you want to exit: ")

Клас ConsoleRectangle:

from Data.lab\_5.Pyramid import Pyramid

class ConsolePyramid(Pyramid):

@classmethod

def set\_parameters(cls, dct, input\_message):

for key, value in dct.items():

print(f"{key}: {value}")

value = input(f"Select {input\_message}: ")

current\_value = dct.get(value, None)

if current\_value:

return current\_value

else:

return None

def input\_function(self, dct, input\_message, default):

check = input(f"do you want to set {input\_message}? (1/0): ")

font = None

if check == '1':

font = self.set\_parameters(dct, 'font')

if font:

return font

else:

return default

def configuration(self):

self.size\_x = int(input("Input size\_x: "))

self.size\_y = int(input("Input size\_y: "))

self.size\_z = int(input("Input size\_z: "))

self.\_color\_1 = self.input\_function(self.\_COLORS, 'color\_1', self.\_color\_1)

self.\_color\_2 = self.input\_function(self.\_COLORS, 'color\_2', self.\_color\_2)

self.\_color\_3 = self.input\_function(self.\_COLORS, 'color\_3', self.\_color\_3)

change\_space = input('do you want to create space? (1/0): ')

if change\_space == '1':

space = int(input("Input space: "))

self.\_space(space)

change\_zoom = input('do you want to create zoom? (1/0): ')

if change\_zoom == '1':

zoom = int(input("Input zoom: "))

self.\_zoom(zoom)

check = input("do you want to save? (1/0): ")

if check == '1':

save = input("Input filename: ")

self.save(save)

def run(self):

stop = ''

while stop != 'f':

try:

self.configuration()

self.create()

print(self)

except Exception as e:

print(e)

finally:

stop = input("Press 'f' if you want to exit: ")

from Data.lab\_5.ConsolePyramid import ConsolePyramid

Клас main:

from Data.lab\_5.ConsoleRectangle import ConsoleRectangle

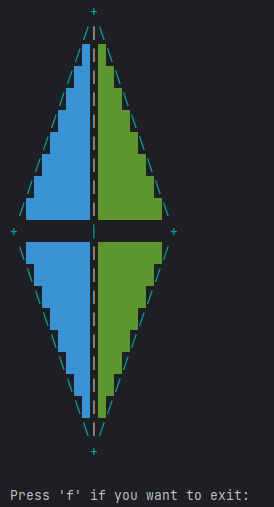
art = ConsolePyramid()

art.run()

На рисунку 1 та 2 зображено результат виконання програми.



*Рис.1 Вивід куба*



*Рис.2 Вивід піраміди*

**Висновок:** під час виконання лабораторної роботи я навчився створювати 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, посприяв творчому підходу до створення ASCII-арту.