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**

Виконав:

ст. гр. ІТ-21сп

**Олександр БІЗЯЄВ**

Прийняв:

доц. каф. ІСМ

**Сергій ЩЕРБАК**

Львів ‒ 2023

**Лабораторна робота №5**

**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.Файл **parallelepiped.py**

class Parallelepiped:  
 def \_\_init\_\_(self, width=None, height=None, length=None):  
 self.width = width if width is not None else 10  
 self.height = height if height is not None else 10  
 self.save\_height = self.height  
 self.length = length if length is not None else 15  
 self.oblique = '\\'  
 self.top = '+'  
 self.column = '|'  
 self.transform = '\n'  
 self.space\_count = 0  
 self.increment\_count = 1  
  
 self.default\_colors = {  
 'color1': 'yellow',  
 'color2': 'blue',  
 'color3': 'green',  
 'color4': 'white'  
 }  
 self.gap = ' '  
 self.line = '—'  
  
 def custom\_background\_color(self, text, color):  
 colors = {  
 'black': '\033[40m',  
 'red': '\033[41m',  
 'green': '\033[42m',  
 'yellow': '\033[43m',  
 'blue': '\033[44m',  
 'magenta': '\033[45m',  
 'cyan': '\033[46m',  
 'white': '\033[47m',  
 }  
  
 reset\_color = '\033[0m'  
 background\_color = colors.get(color.lower(), '\033[47m')  
 return f"{background\_color}{text}{reset\_color}"  
  
 def display\_parallelepiped(self, number, colors=None):  
 colors = colors if colors else self.default\_colors  
 if number == 1:  
 art = self.generate\_layer(  
 self.custom\_background\_color(' ', colors['color1']),  
 self.custom\_background\_color(' ', colors['color2']),  
 self.custom\_background\_color(' ', colors['color3']),  
 self.custom\_background\_color(' ', colors['color4']),  
 gap=self.gap,  
 line=self.line  
 )  
 else:  
 art = self.generate\_layer(  
 self.custom\_background\_color(' ', colors['color1']),  
 self.custom\_background\_color(' ', colors['color2']),  
 self.custom\_background\_color(' ', colors['color3']),  
 self.custom\_background\_color(' ', colors['color4']),  
 gap=' ',  
 line='——'  
 )  
 return art  
  
 def generate\_layer(self, color1, color2, color3, color4, gap, line):  
 shadow = color4 \* self.save\_height  
 art = self.save\_height \* gap + self.top + line \* (self.width - 2) + self.top + self.transform  
 for i in range(self.length - 4, 0, -1):  
 if self.height > 2:  
 art = self.\_generate\_multi\_height\_layer(color1, color2, gap, art)  
 if self.height == 2:  
 art = self.\_generate\_two\_height\_layer(color1, color2, shadow, art)  
 if self.height == 1:  
 art = self.\_generate\_one\_height\_layer(color1, color2, gap, shadow, art)  
 if i == 1:  
 art = self.\_generate\_final\_layer(color1, color3, color4, gap, line, shadow, art)  
 return art  
  
 def \_generate\_multi\_height\_layer(self, color1, color2, gap, art):  
 art += self.save\_height \* gap + self.column + color1 \* self.space\_count + self.oblique + (  
 color2 \* (self.width - 2)) + self.oblique + self.transform  
 self.reduction\_height\_and\_magnification\_count()  
 return art  
  
 def \_generate\_two\_height\_layer(self, color1, color2, shadow, art):  
 art += shadow + self.top + self.space\_count \* color1 + self.oblique + (  
 color2 \* (self.width - 2)) + self.oblique + self.transform  
 self.reduction\_height\_and\_magnification\_count()  
 return art  
  
 def \_generate\_one\_height\_layer(self, color1, color2, gap, shadow, art):  
 art += gap \* self.increment\_count + shadow + self.oblique + (self.space\_count - 1) \* color1 + self.oblique + (  
 color2 \* (self.width - 2)) + self.oblique + self.transform  
 self.update\_increment()  
 return art  
  
 def \_generate\_final\_layer(self, color1, color3, color4, gap, line, shadow, art):  
 art += gap \* self.increment\_count + shadow + self.oblique + (self.space\_count - 1) \* color1 + self.top + (  
 line \* (self.width - 2)) + self.top + self.transform  
 self.update\_increment()  
 for i in range(self.space\_count, 0, -1):  
 art += (gap \* self.increment\_count + shadow + self.oblique + color1 \* (self.space\_count - 2) +  
 self.column + color3 \* (self.width - 2) + self.column + self.transform)  
 self.reduction\_count\_and\_\_height()  
 self.update\_increment()  
 if self.space\_count == 1:  
 art += (gap \* self.increment\_count + color4 \* self.save\_height + self.top + line \*  
 (self.width - 2) + self.top + self.transform)  
 # self.height -= 1  
 break  
 return art  
  
 def reduction\_height\_and\_magnification\_count(self):  
 self.height -= 1  
 self.space\_count += 1  
  
 def update\_increment(self):  
 self.increment\_count += 1  
  
 def reduction\_count\_and\_\_height(self):  
 self.height -= 1  
 self.space\_count -= 1

Файл **variables.py**

choose\_exit = "Do you want to make another art? (Yes/No): "  
choice\_yes = "yes"  
choice\_y = "y"  
choose\_zoom = "Do you want to zoom the art? (Yes/No): "  
  
colors = "\nAvailable colors: 'black', 'red', 'green', 'yellow', 'blue', 'magenta', 'cyan', 'white'"  
color\_left = "Enter a color for the left edge (default yellow): "  
color\_top = "Enter a color for the top border (default blue): "  
color\_front = "Enter a color for the leading edge (default green): "  
color\_shadow = "Enter a color for the shadow (default white): "  
  
user\_width = "Enter the width (more than 8):"  
user\_height = "Enter the height (more than 8):"  
user\_length = "Enter the length (more than the height by 4):"  
  
error\_width\_and\_height = "Error: The number must be an int!"  
error\_length = "Error: The number must be an int and more than the height by 4!"  
  
ask\_save\_art = "Do you want to save the art? (Yes/No): "  
file\_name = "Please enter file name: "

Файл **errors.py**

class Error:  
 def check\_empty(self, value):  
 return bool(value)  
  
 def check\_width\_and\_height(self, parameter):  
 try:  
 parameter = int(parameter)  
 if parameter < 8:  
 raise ValueError  
 except ValueError:  
 return ValueError()  
  
 def check\_length(self, height, length):  
 try:  
 height = int(height)  
 length = int(length)  
 if length - height < 4:  
 raise ValueError  
 except ValueError:  
 return ValueError()  
  
 def check\_color(self, number):  
 try:  
 str(number)  
 except ValueError:  
 return ValueError()

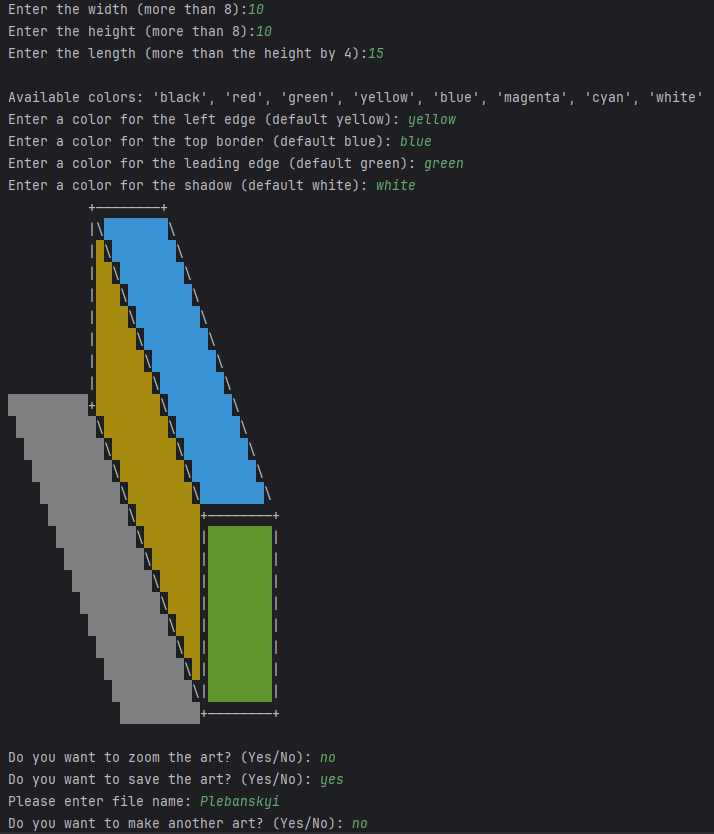
Файл **interface.py**

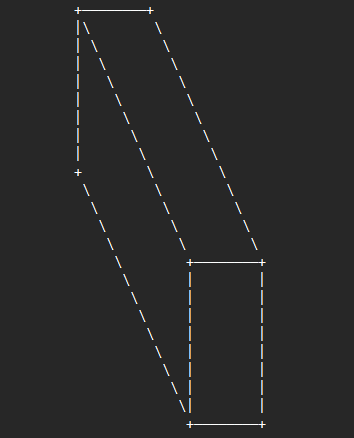
import re  
  
from colorama import Fore, Style  
  
from labwork\_5 import variables  
from labwork\_5.errors import Error  
from labwork\_5.parallelepiped import Parallelepiped  
  
  
class Interface:  
 def \_\_init\_\_(self):  
 self.error = Error()  
  
 def \_choice\_exit(self, choice):  
 if choice.lower() not in [variables.choice\_yes, variables.choice\_y]:  
 return True  
  
 def \_choice(self, choice):  
 if choice.lower() in [variables.choice\_yes, variables.choice\_y]:  
 return True  
  
 def user\_input(self):  
 while True:  
 width, height, length = self.input\_parameters()  
 user\_parallelepiped = Parallelepiped(width, height, length)  
 print(variables.colors)  
 colors = self.input\_colors(user\_parallelepiped)  
 user\_parallelepiped = user\_parallelepiped.display\_parallelepiped(number=1, colors=colors)  
 print(user\_parallelepiped)  
 choose\_zoom = input(variables.choose\_zoom)  
 if self.\_choice(choose\_zoom):  
 parallelepiped\_zoom = Parallelepiped(width, height, length)  
 user\_parallelepiped = parallelepiped\_zoom.display\_parallelepiped(number=2, colors=colors)  
 print(user\_parallelepiped)  
 change\_selection = input(variables.ask\_save\_art)  
 if self.\_choice(change\_selection):  
 name\_file = input(variables.file\_name)  
 self.save\_to\_file(name\_file, user\_parallelepiped)  
 choose\_exit = input(variables.choose\_exit)  
 if self.\_choice\_exit(choose\_exit):  
 break  
  
 def input\_parameters(self):  
 width = self.get\_valid\_input(variables.user\_width, self.error.check\_width\_and\_height,  
 variables.error\_width\_and\_height)  
 height = self.get\_valid\_input(variables.user\_height, self.error.check\_width\_and\_height,  
 variables.error\_width\_and\_height)  
 length = self.get\_valid\_length(variables.user\_length, height, self.error.check\_length, variables.error\_length)  
  
 return width, height, length  
  
 def get\_valid\_input(self, prompt, error\_check, error\_message):  
 while True:  
 value = input(prompt).strip()  
 if not self.error.check\_empty(value):  
 return None  
 if error\_check(value):  
 print(Fore.RED + Style.BRIGHT + error\_message + Style.RESET\_ALL)  
 else:  
 return int(value)  
  
 def get\_valid\_length(self, prompt, height, error\_check, error\_message):  
 while True:  
 length = input(prompt).strip()  
 if not self.error.check\_empty(length):  
 return None  
 if error\_check(height, length):  
 print(Fore.RED + Style.BRIGHT + error\_message + Style.RESET\_ALL)  
 else:  
 return int(length)  
  
 def input\_colors(self, parallelepiped):  
 custom\_color1 = self.get\_input\_color(variables.color\_left, self.is\_valid\_color,  
 parallelepiped.default\_colors['color1'])  
 custom\_color2 = self.get\_input\_color(variables.color\_top, self.is\_valid\_color,  
 parallelepiped.default\_colors['color2'])  
 custom\_color3 = self.get\_input\_color(variables.color\_front, self.is\_valid\_color,  
 parallelepiped.default\_colors['color3'])  
 custom\_color4 = self.get\_input\_color(variables.color\_shadow, self.is\_valid\_color,  
 parallelepiped.default\_colors['color4'])  
  
 parallelepiped\_colors = {  
 'color1': custom\_color1,  
 'color2': custom\_color2,  
 'color3': custom\_color3,  
 'color4': custom\_color4  
 }  
 return parallelepiped\_colors  
  
 def is\_valid\_color(self, color):  
 colors = {  
 'black': '\033[40m',  
 'red': '\033[41m',  
 'green': '\033[42m',  
 'yellow': '\033[43m',  
 'blue': '\033[44m',  
 'magenta': '\033[45m',  
 'cyan': '\033[46m',  
 'white': '\033[47m',  
 }  
 return color.lower() in colors  
  
 def get\_input\_color(self, entered\_value, error\_check, default\_value):  
 while True:  
 value = input(entered\_value).strip().lower()  
 if value in {'', ' ', None} or not error\_check(value):  
 return default\_value  
 return value  
  
 def save\_to\_file(self, file\_name, user\_parallelepiped):  
 file\_name = f"{file\_name}.txt"  
 stripped\_text = self.strip\_ansi(user\_parallelepiped)  
  
 with open(file\_name, "w") as file:  
 file.write(stripped\_text)  
  
 def strip\_ansi(self, text):  
 ansi\_escape = re.compile(r'\x1B(?:[@-Z\\-\_]|\[[0-?]\*[ -/]\*[@-~])')  
 return ansi\_escape.sub('', text)

Файл **main.py**

from labwork\_5.interface import Interface  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 interface = Interface()  
 interface.user\_input()

Результат виконання всіх завдань:





**Рис.1** Результат виконання

**Висновок:** Виконавши ці завдання, я розробив додаток малювання 3D-фігур у ASCII-арті на основі об’єктно-орієнтованого підходу та мови Python.