Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Кафедра ІСМ

Звіт
про виконання лабораторної роботи № 4
«Розробка ASCII ART генератора для візуалізації 2D-фігур»
з дисципліни
«Спеціалізовані мови програмування»

Виконав:
Студент групи ІТ-32,
Вольвенко І. Р.

Прийняв:
Щербак С.С

Львів 2023

Мета роботи: Створення Генератора ASCII-арту без використання зовнішніх бібліотек.

Завдання:

Завдання 1: Введення користувача

Створіть програму Python, яка отримує введення користувача щодо слова або фрази, яку вони хочуть перетворити в ASCII-арт.

Завдання 2: Набір символів

Визначте набір символів (наприклад, '@', '#', '*', тощо), які будуть використовуватися для створення ASCII-арту. Ці символи будуть відображати різні відтінки.

Завдання 3: Розміри Art-у

Запитайте у користувача розміри (ширина і висота) ASCII-арту, який вони хочуть створити. Переконайтеся, що розміри в межах керованого діапазону

Завдання 4: Функція генерації Art-у

Напишіть функцію, яка генерує ASCII-арт на основі введення користувача, набору символів та розмірів. Використовуйте введення користувача, щоб визначити, які символи використовувати для кожної позиції в Art-у.

Завдання 5: Вирівнювання тексту

Реалізуйте опції вирівнювання тексту (ліво, центр, право), щоб користувачі могли вибирати, як їх ASCII-арт розміщується на екрані.

Завдання 6: Відображення мистецтва

Відобразіть створений ASCII-арт на екрані за допомогою стандартних функцій друку Python.

Завдання 7: Збереження у файл

Додайте можливість зберігати створений ASCII-арт у текстовий файл, щоб користувачі могли легко завантажувати та обмінюватися своїми творіннями.

Завдання 8: Варіанти кольорів

Дозвольте користувачам вибирати опції кольорів (чорно-білий, відтінки сірого) для свого ASCII-арту.

Завдання 9: Функція попереднього перегляду

Реалізуйте функцію попереднього перегляду, яка показує користувачам попередній перегляд їх ASCII-арту перед остаточним збереженням

Завдання 10: Інтерфейс, зрозумілий для користувача

Створіть інтерфейс для користувача у командному рядку, щоб зробити програму легкою та інтуїтивно зрозумілою для використання.

Код:

main.py:

```
from ascii_art_generator import *

def save_to_file(destination_path, content) -> None:
    with open(destination_path, "w") as output_file:
        output_file.write(content)


def load_from_file(source_path) -> str:
    with open(source_path, "r") as input_file:
        return input_file.read()


def main():
    try:
```

```
        user_input_text = str(input("Input text to convert to ASCII art: "))
        AbstractDataHandler.show_color_options()
        chosen_color_index = int(input("Choose a color index: "))
        desired_width = int(input("Set the width for ASCII art: "))
        alphabet_file_path = "ascii_alphabet.txt"
        ascii_processor = TextFileHandler(alphabet_file_path)
        ascii_art = ascii_processor.fetch(user_input_text, chosen_color_index, desired_width)
        print(ascii_art)
        save_to_file("ascii_output.txt", ascii_art)
    except KeyError:
        print("Error: Incorrect color index or missing characters in the ASCII file.")
    except ValueError as ve:
        print(ve)
    except FileNotFoundError:
        print("Error: ASCII character file not found. Please verify the path.")

if __name__ == "__main__":
    main()
```

ascii_art_generator.py:

```
import os
import re
from abc import ABC, abstractmethod
import colorama
from colorama import Fore
from functools import reduce

colorama.init(autoreset=True)
color_palette = {index: color for index, color in enumerate(sorted(Fore.__dict__.keys()))}


class AbstractDataHandler(ABC):

    def __init__(self, path_to_file):
        if not path_to_file:
```

```
            raise ValueError("Path to file cannot be empty")
        self._file_path = path_to_file

    @abstractmethod
    def _read_and_process_data(self) -> None:
        pass

    @abstractmethod
    def fetch(self, text, color_index, line_width) -> str:
        pass

    def _load_file_content(self) -> str:
        with open(self._file_path, 'r') as file:
            return file.read()

    @staticmethod
    def show_color_options() -> None:
        for index, color_name in color_palette.items():
            print(f"{index}. {color_name}")



class TextFileHandler(AbstractDataHandler):
    _meta_info = {}
    _content = {}

    def __init__(self, path_to_file):
        if not path_to_file.endswith(".txt") or not os.path.exists(path_to_file):
            raise ValueError("File must be a .txt file and exist")
        super().__init__(path_to_file)
        self._read_and_process_data()

    def _read_and_process_data(self) -> None:
        file_content = self._load_file_content().split("\n")
        data_section_found = False
        symbol_representation = ""
```

```
        current_symbol = None

        for line in file_content:
            if data_section_found:
                if re.match("^@symbol::.$", line):
                    if current_symbol:
                        self._content[current_symbol] = symbol_representation
                    current_symbol = line[-1]
                    symbol_representation = ""
                    row_length = 0
                    line_counter = 1
                elif re.match("^\\^.+\\$$", line):
                    if line_counter == 1:
                        row_length = len(line)
                    if len(line) != row_length:
                        raise ValueError("Inconsistent row length for symbol representation")
                    symbol_representation += line[1:-1] + ("\n" if line_counter < self._meta_info["height"] else "")
                    line_counter += 1
                else:
                    raise ValueError("Incorrect format in data section")
            elif line == "@data":
                if "height" not in self._meta_info:
                    raise ValueError("Meta information 'height' is missing")
                data_section_found = True
            elif not data_section_found:
                if re.match("^\\w+::\\d+$", line):
                    key, value = line.split("::")
                    self._meta_info[key] = int(value)
                else:
                    raise ValueError("Invalid meta data format")

    def fetch(self, text, color_index, max_width) -> str:
        symbol_data = {}
        needed_symbols = {}
```

```
line_properties = {}
current_line = 0
current_line_width = 0

for index, char in enumerate(text):
    symbol_lines = self._content[char].split("\n")
    if current_line_width + len(symbol_lines[0]) > max_width:
        if len(symbol_lines[0]) > max_width:
            raise ValueError("Text width too small")
        current_line += 1
        current_line_width = 0
    current_line_width += len(symbol_lines[0])
    line_properties[current_line] = line_properties.get(current_line, 0) + 1
    needed_symbols[index] = "\n".join(symbol_lines)

symbol_index = 0
for line in line_properties:
    for _ in range(line_properties[line]):
        lines = needed_symbols[symbol_index].split("\n")
        symbol_index += 1
        for row_index, row in enumerate(lines):
            line_key = row_index + line * self._meta_info["height"]
            symbol_data[line_key] = symbol_data.get(line_key, "") + row

colored_output = Fore.__getattribute__(color_palette[color_index])
return colored_output + "\n".join(symbol_data.values())
```

Виконання програми

На рис. 1 зображенно виконання програми:

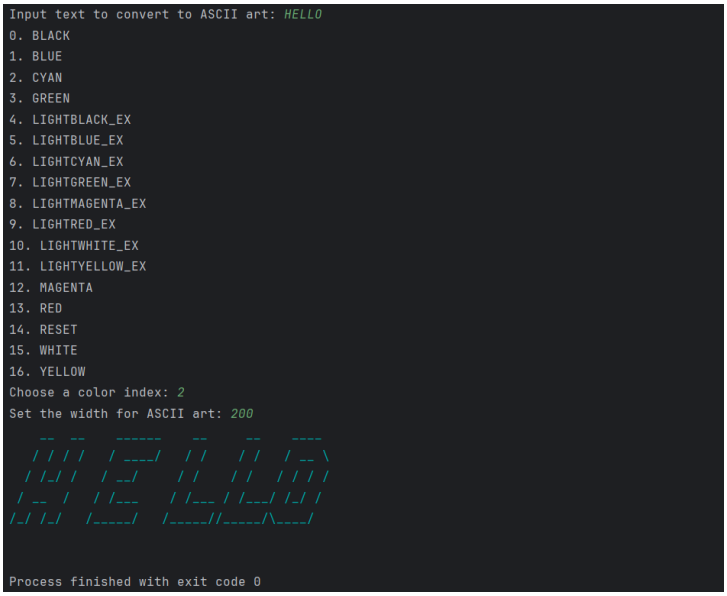

Рис. 1 Виконання програми

Посилання на GitHub репозиторій: https://github.com/Deadmarvald/smp

Висновки: Виконуючи ці завдання, я створив генератор ASCII-арту з нуля, та створив можливість налаштовувати символи, розміри, вирівнювання та кольори, що дозволило мені глибше розібратися як створюється ASCII-арт.