МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра інформаційних систем та мереж

ЗВІТ

про виконання лабораторної роботи № 4

«Розробка ASCII ART генератора для візуалізації 2D-фігур»

з дисципліни "Спеціалізовані мови програмування”

Виконала:

ст. гр. ІТ-32,

Троцько О. М.

Прийняв:

Щербак С. С.

ЛЬВІВ – 2023

Мета: створення Генератора ASCII-арту без використання зовнішніх бібліотек.

План роботи

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 програми:

```
# data_from_console.py
"""
Console reader Module

This module provides functions for handling user input from the console.
"""
import curses
import termcolor
from shared.input_handler import InputHandler

def get_console_width():
    """
    Get the width of the console.

    Returns:
        int: The width of the console.
    """
    stdscr = curses.initscr()
    rows, columns = stdscr.getmaxyx()
    curses.endwin()
    return columns



def get_text_from_console():
    """
    Get text input from the console.

    Returns:
        str: The text entered by the user.
    """
    text = InputHandler().get_str_input("Enter text")
    return text

def get_width_and_height_from_console():
    """
    Get width and height input from the console.

    Returns:
        tuple: A tuple containing the width and height entered by the user.
    """
    width = InputHandler().get_int_input("Enter width (min 5)")
    height = InputHandler().get_int_input("Enter height(min 5)")
    width, height = check_width_and_height(width, height)
    return width, height

def check_width_and_height(width, height):
    """
    Check if the width and height are valid.

    Args:
        width (int): The width entered by the user.
        height (int): The height entered by the user.

    Returns:
        tuple: A tuple containing the valid width and height.
    """
    if width > get_console_width():
        print("Width is too big (min 5)!\n")
        return get_width_and_height_from_console()
    if width < 5 or height < 5:
        print("Width and height should be at least 5!\n")
        return get_width_and_height_from_console()
    return width, height

def get_symbol_from_console():
    """
    Get a symbol input from the console.
    Returns:
        str: The symbol entered by the user.
    """
     symbol = InputHandler().get_one_char_input("Enter one symbol (e.g. '@', '#',
'*')")
    return symbol

def get_color_from_console():
    """
    Get a color input from the console.

    Returns:
        str: The color entered by the user.
    """
     color = InputHandler().get_one_of_list_input("Enter color name [ red, green,
blue, yellow, white ]", termcolor.COLORS)
    return color

def get_justify_from_console():
    """
    Get a justify input from the console.

    Returns:
        str: The justify entered by the user.
    """
    justify_list = ["left", "center", "right"]
     justify = InputHandler().get_one_of_list_input("Enter justify (left, center,
right)" , justify_list)
    return justify

def get_size_from_console():
    """
    Get a size input from the console.

    Returns:
        int: The size entered by the user.
    """
    size = InputHandler().get_int_input("Enter size (min 6)")
    size = check_size(size)
    return size
```

```
def check_size(size):
    """
    Check if the size is valid.

    Args:
        size (int): The size entered by the user.

    Returns:
        int: The valid size.
    """
    if size > get_console_width():
        print("Size is too big (min 6)!\n")
        return get_size_from_console()
    if size < 6:
        print("Size should be at least 6!\n")
        return get_size_from_console()
    return size

# art_size.py
"""
ArtSize module

This module defines the ArtSize class, which represents the size of an art.
"""
import numpy as np
from shared.settings import get_lab_settings

settings = get_lab_settings("lab4")
DEFAULT_ART_SETTINGS = settings["default_art_settings"]
DEFAULT_WIDTH = DEFAULT_ART_SETTINGS["width"]
DEFAULT_HEIGHT = DEFAULT_ART_SETTINGS["height"]

class ArtSize:
    """
    A class representing the size of an art.

    Attributes:
        _width (int): The width of the art.
        _height (int): The height of the art.
        _chars (list): The characters representing the art.
    """

    def __init__(self, width=DEFAULT_WIDTH, height=DEFAULT_HEIGHT):
        """
        Initializes the ArtSize object.

        Args:
            width (int): The width of the art. Defaults to DEFAULT_WIDTH.
            height (int): The height of the art. Defaults to DEFAULT_HEIGHT.
        """
        self._width = width
        self._height = height
        self._chars = None

    def __str__(self):
        """
        Returns a string representation of the ArtSize object.

        Returns:
            str: The string representation of the ArtSize object.
        """
        return f"Width: {self._width} \nHeight: {self._height}"

    def set_width(self, width):
        """
        Sets the width of the art.

        Args:
            width (int): The width of the art.
        """
        self._width = width

    def set_height(self, height):
        """
        Sets the height of the art.

        Args:
            height (int): The height of the art.
        """
        self._height = height

    def set_chars(self, chars):
        """
        Sets the characters representing the art.

        Args:
            chars (list): The characters representing the art.
        """
        self._chars = chars

    def get_width(self):
        """
        Returns the width of the art.

        Returns:
            int: The width of the art.
        """
        return self._width

    def get_height(self):
        """
        Returns the height of the art.

        Returns:
            int: The height of the art.
        """
        return self._height

    def get_chars(self):
        """
        Returns the characters representing the art.

        Returns:
            list: The characters representing the art.
        """
        return self._chars

    def get_resized_chars(self):
        """
        Returns the resized characters representing the art.

        Returns:
            list: The resized characters representing the art.
        """
        resized_chars = []
        if self._chars is None:
            return "No chars to resize"
        for char in self._chars:
            resized_chars.append(self.__change_char_size(char))

        self._chars = resized_chars
        return resized_chars

    def __change_char_size(self, matrix):
        """
        Changes the size of a character matrix.

        Args:
            matrix (list): The character matrix to resize.

        Returns:
            list: The resized character matrix.
        """
        matrix = self.__change_height(matrix)
        matrix = self.__change_width(matrix)
        return matrix

    def __change_width(self, matrix):
        """
        Changes the width of a character matrix.

        Args:
            matrix (list): The character matrix to resize.

        Returns:
            list: The resized character matrix.
        """
        if self._width <= 5:
            return matrix
        columns_to_add = self._width-5
        column_index = round(len(matrix[0])/2)
        column_to_add = [row[column_index] for row in matrix]
        np_matrix = np.array(matrix)

        for i in range(columns_to_add):
            np_matrix = np.insert(np_matrix, column_index, column_to_add, axis=1)

        matrix = np_matrix.tolist()
        return matrix

    def __change_height(self, matrix):
        """
        Changes the height of a character matrix.

        Args:
            matrix (list): The character matrix to resize.

        Returns:
            list: The resized character matrix.
        """
        if self._height <= 5:
            return matrix

        lines_to_add = round((self._height-5)/2)
        top_line = matrix[1]
        bottom_line = matrix[len(matrix) - 2]

        np_matrix = np.array(matrix)

        for i in range(lines_to_add):
            top_row_index = 1
            bottom_row_index = np_matrix.shape[0] - 1
            np_matrix = np.insert(np_matrix, top_row_index, top_line, axis=0)
            np_matrix = np.insert(np_matrix, bottom_row_index, bottom_line, axis=0)

        matrix = np_matrix.tolist()
        return matrix
```

```
# art_settings.py
"""
ArtSettings Module

This module defines the ArtSettings class, which represents the settings for
creating art.
It includes methods for initializing settings, getting and setting individual
settings,
changing settings interactively through the console, and viewing the current
settings.
"""
from UI.menu import Menu
from UI.menu_item import Item
from shared.settings import get_lab_settings
from classes.lab4.art_settings.art_size import ArtSize
from classes.lab4.console_reader.data_from_console import
get_width_and_height_from_console, get_symbol_from_console,
get_justify_from_console, get_color_from_console

settings = get_lab_settings("lab4")
DEFAULT_ART_SETTINGS = settings["default_art_settings"]
DEFAULT_WIDTH = DEFAULT_ART_SETTINGS["width"]
DEFAULT_HEIGHT = DEFAULT_ART_SETTINGS["height"]
DEFAULT_JUSTIFY = DEFAULT_ART_SETTINGS["justify"]
DEFAULT_COLOR = DEFAULT_ART_SETTINGS["color"]
DEFAULT_SYMBOL = DEFAULT_ART_SETTINGS["symbol"]

class ArtSettings:
    """
    Represents the settings for creating art.

    Attributes:
        _symbol (str): The symbol used for the art.
        _size (ArtSize): The size of the art.
        _justify (str): The justification of the art.
        _color (str): The color of the art.
    """

    def __init__(self, symbol=DEFAULT_SYMBOL, width=DEFAULT_WIDTH,
height=DEFAULT_HEIGHT, justify=DEFAULT_JUSTIFY, color=DEFAULT_COLOR):
        """
        Initialize the ArtSettings object.

        Args:
            symbol (str): The symbol used for the art.
            width (int): The width of the art.
            height (int): The height of the art.
            justify (str): The justification of the art.
            color (str): The color of the art.
        """
        self._symbol = symbol
        self._size = ArtSize(width, height)
        self._justify = justify
        self._color = color

    def __str__(self):
        """
        Return a string representation of the ArtSettings object.

        Returns:
            str: The string representation of the ArtSettings object.
        """
        return f"Symbol: {self._symbol} \n{self._size} \nJustify: {self._justify} \
nColor: {self._color}"

    def get_symbol(self):
        """
        Get the symbol used for the art.

        Returns:
            str: The symbol used for the art.
        """
        return self._symbol

    def get_size(self):
        """
        Get the size of the art.

        Returns:
            ArtSize: The size of the art.
        """
        return self._size

    def get_justify(self):
        """
        Get the justification of the art.

        Returns:
            str: The justification of the art.
        """
        return self._justify

    def get_color(self):
        """
        Get the color of the art.

        Returns:
            str: The color of the art.
        """
        return self._color

    def set_symbol(self, symbol):
        """
        Set the symbol used for the art.

        Args:
            symbol (str): The symbol used for the art.
        """
        self._symbol = symbol

    def set_size(self, size):
        """
        Set the size of the art.

        Args:
            size (ArtSize): The size of the art.
        """
        self._size = size

    def set_justify(self, justify):
        """
        Set the justification of the art.

        Args:
            justify (str): The justification of the art.
        """
        self._justify = justify

    def set_color(self, color):
        """
        Set the color of the art.

        Args:
            color (str): The color of the art.
        """
        self._color = color

    def change_size(self):
        """
           Change the size of the art by getting the width and height from the
console.
        """
        width, height = get_width_and_height_from_console()
        self._size.set_width(width)
        self._size.set_height(height)

    def change_symbol(self):
        """
        Change the symbol used for the art by getting it from the console.
        """
        self._symbol = get_symbol_from_console()

    def change_justify(self):
        """
        Change the justification of the art by getting it from the console.
        """
        self._justify = get_justify_from_console()

    def change_color(self):
        """
        Change the color of the art by getting it from the console.
        """
        self._color = get_color_from_console()

    def view_settings(self):
        """
        Print the current settings of the art.
        """
        print(str(self))

    def menu(self):
        """
        Create and run the settings menu.
        """
        settings_menu = Menu("\nSettings Menu")
        settings_menu.add_item(Item('1', 'Change Symbol', self.change_symbol))
        settings_menu.add_item(Item('2', 'Change Size', self.change_size))
        settings_menu.add_item(Item('3', 'Change Justify', self.change_justify))
        settings_menu.add_item(Item('4', 'Change Color', self.change_color))
        settings_menu.add_item(Item('5', 'View Settings', self.view_settings))
        settings_menu.add_item(Item('0', 'Back'))

        settings_menu.run()
```

# art_generator.py

```
"""
ArtGenerator Module

This module defines the ArtGenerator class, which generates ASCII
art based on user input and settings. It includes methods for setting text,
changing settings, viewing and saving generated art, and viewing saved art from a
file.
"""

from termcolor import colored
from classes.lab4.art_settings.art_settings import ArtSettings
from  classes.lab4.console_reader.data_from_console  import  get_text_from_console,
get_console_width
from classes.lab4.font.font import font_dict
from UI.menu import Menu
from UI.menu_item import Item
from shared.file_handler import FileHandler
from shared.settings import get_lab_settings

settings = get_lab_settings("lab4")
ART_PATH = settings["art_path"]

class ArtGenerator:
    """
    A class that generates ASCII art based on user input and settings.

    Attributes:
    - __text: The text used to generate the art.
    - __settings: The settings for generating the art.
    - __art: The generated ASCII art.
    """
    def __init__(self):
        self.__text = None
        self.__settings = ArtSettings()
        self.__art = None

    def menu(self):
        """
        This function creates and runs a menu for the Art Generator program.
         It  adds menu items for various actions such as setting art text, changing 
settings, viewing art,
        saving art to a file, viewing saved art, and exiting the program.
        """
        art_menu = Menu("\nArt Menu (Lab 4)")
        art_menu.add_item(Item('1', 'Set Art Text', self.set_text))
        art_menu.add_item(Item('2', 'Change Settings', self.change_settings))
        art_menu.add_item(Item('3', 'View Art', self.view_art))
        art_menu.add_item(Item('4', 'Save Art to File', self.save_art_to_file))
        art_menu.add_item(Item('5', 'View Saved Art', self.view_saved_art))
        art_menu.add_item(Item('0', 'Exit'))

        art_menu.run()

    def set_text(self):
        """
        Sets the text used to generate the art.
        """
        self.__text = get_text_from_console()
        self.__settings.get_size().set_chars(self.__map_chars())

    def get_text(self):
        """
        Returns the text used to generate the art.
        """
        return self.__text

    def set_settings(self, art_settings):
        """
        Sets the settings for generating the art.

        Parameters:
        - settings: The settings object.
        """
        self.__settings = art_settings

    def get_settings(self):
        """
        Returns the settings for generating the art.
        """
        return self.__settings

    def get_art(self):
        """
        Returns the generated ASCII art.
        """
        return self.__art

    def __map_chars(self):
        """
        Maps the characters in the text to the corresponding ASCII art characters.
        """
        text = self.__text.upper()
        chars = []
        for char in text:
            if char in font_dict:
                chars.append(font_dict[char])
            else:
                chars.append(" ")
        return chars

    def view_art(self):
        """
        Displays the generated ASCII art.
        """
        if self.__text is None:
            self.set_text()

        self.set_art_settings()
        print()
        print(colored(self.__art, self.__settings.get_color()))

    def set_art_settings(self):
        """
        Sets the settings for generating the art.
        """
        console_width = get_console_width()
        art_len = self.__settings.get_size().get_width()*(len(self.__text) + 3)

        if art_len > console_width:
            print("Art is too big for console\nResizing...\n")
            self.__settings.get_size().set_width(console_width//(len(self.__text) +
3))
             self.__settings.get_size().set_height(console_width//(len(self.__text)
+ 3))

        chars = self.__settings.get_size().get_chars()
                      if   self.__settings.get_size().get_width()   >   5   or
self.__settings.get_size().get_height() > 5:
            chars = self.__settings.get_size().get_resized_chars()

        self.__art = self.__generate_art(chars)
        padding = self.__get_padding()
        art_lines = self.__art.split('\n')
        aligned_lines = [" " * padding + line for line in art_lines]
        self.__art = '\n'.join(aligned_lines)
```

```python
def __generate_art(self, chars):
    """
    Generates the ASCII art based on the mapped characters.

    Parameters:
    - chars: The mapped characters.

    Returns:
    - The generated ASCII art.
    """
    height = len(chars[0])
    width = len(chars[0][0])
    art = ""
    for row in range(height):
        for char in chars:
            for column in range(width):
                if char[row][column] == 1:
                    art += self.__settings.get_symbol()
                else:
                    art += " "
            art += "   "
        art += "\n"
    return art

def __get_padding(self):
    """
    Calculates the padding for aligning the art.

    Returns:
    - The padding value.
    """
    console_width = get_console_width()
    art_len = round(len(self.__art)/self.__settings.get_size().get_height())
    if self.__settings.get_justify() == "center":
        return (console_width - art_len) // 2
    if self.__settings.get_justify() == "right":
        return console_width - art_len
    return 0
def change_settings(self):
    """
    Allows the user to change the settings for generating the art.
    """
    self.__settings.menu()

def save_art_to_file(self):
    """
    Saves the generated ASCII art to a file.
    """
    if self.__art is None:
        print("No art to save")
        return
    saved_file = FileHandler(ART_PATH)
    saved_file.write_to_file(self.__art)

def view_saved_art(self):
    """
    Displays the saved ASCII art from a file.
    """
    saved_file = FileHandler(ART_PATH)
    saved_file.read_from_file()
```

# runner.py

```
"""
Module: run_art_generator

Module provides a simple script to run the Art Generator for Lab 4.

"""
from classes.lab4.art_generator.art_generator import ArtGenerator

def run():
    """
    Initializes and runs the Art Generator.
    """
    art_generator = ArtGenerator()
    art_generator.menu()
```

GitHub Repository: https://github.com/trolchiha/SPL-labs.git

Висновок: під час виконання лабораторної роботи навчилася створювати генератор ASCII-арту з нуля, з можливістю налаштовувати символи, розміри, вирівнювання та кольори, що дозволить їм глибше розібратися як створюється ASCII-арт.