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

Виконала:

ст. гр. ІТ-21сп

Борисов П.Р.

Прийняв:

Щербак С.С.

Львів-2023

**Мета роботи:** Cтворення 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**

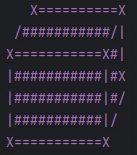
Розгляньте можливість додавання розширених функцій, таких як тінь, освітлення та ефекти перспективи, для підвищення реалізму 3D ASCII-арту.

**Код програми:**  
Клас Cube.py  
class Cube:  
 """  
 The Cube class generates a 3D or 2D representation of a cube.  
 """  
  
 def \_\_init\_\_(self, size=1, color='WHITE', symbol='#', remove\_shades=False):  
 """  
 Initializes a Cube object with a size, color, symbol, and a flag to remove shades.  
 """  
 self.size = size  
 self.color = color  
 self.symbol = symbol  
 self.remove\_shades = remove\_shades  
 self.cube\_3D = None  
 self.cube\_2D = None  
  
 def remove\_color\_codes(self, string):  
 """  
 Removes color codes from the string.  
 """  
 ansi\_escape = re.compile(r'\x1B\[[0-?]\*[ -/]\*[@-~]')  
 return ansi\_escape.sub('', string)  
  
 def draw\_3D(self):  
 """  
 Draws a 3D representation of the cube.  
 """  
 init()  
  
 colors = {  
 'RED': Fore.RED,  
 'GREEN': Fore.GREEN,  
 'YELLOW': Fore.YELLOW,  
 'BLUE': Fore.BLUE,  
 'MAGENTA': Fore.MAGENTA,  
 'CYAN': Fore.CYAN,  
 'WHITE': Fore.WHITE  
 }  
  
 self.cube\_3D = [  
 ["\*" \* (3 if self.size == 1 else 3) + colors[self.color] + "X" + "=" \* self.size \* 10 + "X"],  
 ["\*" \* (1 if self.size == 1 else 1) + colors[self.color] + "/" + self.symbol \* self.size \* 11 + "/" + "|"],  
 [colors[self.color] + "X" + "=" \* self.size \* 11 + "X" + self.symbol \* self.size + "|"],  
 [colors[self.color] + "|" + self.symbol \* self.size \* 11 + "|" + self.symbol \* self.size + "X"],  
 [colors[self.color] + "|" + self.symbol \* self.size \* 11 + "|" + self.symbol \* self.size + "/"],  
 [colors[self.color] + "|" + self.symbol \* self.size \* 11 + "|" + "/"],  
 [colors[self.color] + "X" + "=" \* self.size \* 11 + "X" + Fore.RESET]  
 ]  
  
 if self.remove\_shades:  
 self.cube\_3D[0][0] = self.cube\_3D[0][0].replace("\*", " ", 3 if self.size == 1 else 3)  
 self.cube\_3D[1][0] = self.cube\_3D[1][0].replace("\*", " ", 1)  
  
 if self.size == 2 and self.remove\_shades:  
 self.cube\_3D[0][0] = " " + self.cube\_3D[0][0]  
 self.cube\_3D[1][0] = " " + self.cube\_3D[1][0]  
  
 for row in self.cube\_3D:  
 print(''.join(row))  
  
 def draw\_2D(self):  
 """  
 Draws a 2D representation of the cube.  
 """  
 init()  
  
 colors = {  
 'RED': Fore.RED,  
 'GREEN': Fore.GREEN,  
 'YELLOW': Fore.YELLOW,  
 'BLUE': Fore.BLUE,  
 'MAGENTA': Fore.MAGENTA,  
 'CYAN': Fore.CYAN,  
 'WHITE': Fore.WHITE  
 }  
  
 self.cube\_2D = [  
 [colors[self.color] + "X" + "=" \* self.size \* 11 + "X"],  
 [colors[self.color] + "|" + self.symbol \* self.size \* 11 + "|"],  
 [colors[self.color] + "|" + self.symbol \* self.size \* 11 + "|"],  
 [colors[self.color] + "|" + self.symbol \* self.size \* 11 + "|"],  
 [colors[self.color] + "X" + "=" \* self.size \* 11 + "X" + Fore.RESET]  
 ]  
  
 for row in self.cube\_2D:  
 print(''.join(row))  
  
 def saveCube\_3D(self, filename):  
 """  
 Saves the 3D representation of the cube to a file.  
 """  
 if self.cube\_3D is None:  
 print("Error: Draw the 3D cube first.")  
 return  
  
 cube\_3D\_no\_color = [''.join([self.remove\_color\_codes(cell) for cell in row]) for row in self.cube\_3D]  
  
 self.output\_directory = os.path.join('..', 'Data', 'Lab5')  
  
 os.makedirs(self.output\_directory, exist\_ok=True)  
  
 file\_path = os.path.join(self.output\_directory, filename)  
  
 with open(file\_path, 'w') as f:  
 for row in cube\_3D\_no\_color:  
 f.write(row + '\n')

Клас Pyramid.py:  
class Pyramid:  
 """  
 The Pyramid class generates a 3D or 2D representation of a pyramid.  
 """  
  
 def \_\_init\_\_(self, size=1, color='WHITE', symbol='#', remove\_shadesPyramid=False):  
 """  
 Initializes a Pyramid object with a size, color, symbol, and a flag to remove shades.  
 """  
 self.size = size  
 self.color = color  
 self.symbol = symbol  
 self.remove\_shadesPyramid = remove\_shadesPyramid  
 self.pyramid\_3D = None  
 self.pyramid\_2D = None  
  
 def remove\_color\_codes(self, string):  
 """  
 Removes color codes from the string.  
 """  
 ansi\_escape = re.compile(r'\x1B\[[0-?]\*[ -/]\*[@-~]')  
 return ansi\_escape.sub('', string)  
  
 def remove\_shades(self):  
 """  
 Removes shades from the pyramid.  
 """  
 if self.pyramid\_3D is not None:  
 self.pyramid\_3D = [[cell.replace("=", "") for cell in row] for row in self.pyramid\_3D]  
 if self.pyramid\_2D is not None:  
 self.pyramid\_2D = [[cell.replace("=", "") for cell in row] for row in self.pyramid\_2D]  
  
 def draw\_3D(self):  
 """  
 Draws a 3D representation of the pyramid.  
 """  
 init()  
  
 colors = {  
 'RED': Fore.RED,  
 'GREEN': Fore.GREEN,  
 'YELLOW': Fore.YELLOW,  
 'BLUE': Fore.BLUE,  
 'MAGENTA': Fore.MAGENTA,  
 'CYAN': Fore.CYAN,  
 'WHITE': Fore.WHITE  
 }  
  
 self.pyramid\_3D = [  
 [colors[self.color] + " " \* self.size \* 11 + "/\\" + "\*" \* self.size + "."],  
 [colors[self.color] + " " \* self.size \* 10 + "/" + self.symbol \* self.size \* 2 + "\\" + "\*" \* self.size \* 2 + "."],  
 [colors[self.color] + " " \* self.size \* 9 + "/" + self.symbol \* self.size \* 4 + "\\" + "\*" \* self.size \* 3 + "."],  
 [colors[self.color] + " " \* self.size \* 8 + "/" + self.symbol \* self.size \* 6 + "\\" + "\*" \* self.size \* 4 + "."],  
 [colors[self.color] + " " \* self.size \* 7 + "/" + self.symbol \* self.size \* 8 + "\\" + "\*" \* self.size \* 4 + "|"],  
 [colors[self.color] + " " \* self.size \* 6 + "/" + self.symbol \* self.size \* 10 + "\\" + "\*" \* self.size \* 3 + "|"],  
 [colors[self.color] + " " \* self.size \* 5 + "/" + self.symbol \* self.size \* 12 + "\\" + "\*" \* self.size \* 2 + "|"],  
 [colors[self.color] + " " \* self.size \* 4 + "/" + self.symbol \* self.size \* 14 + "\\" + "\*" \* self.size + "|"],  
 [colors[self.color] + " " \* self.size \* 3 + "/" + "\_" \* self.size \* 16 + "\\" + "|" + Fore.RESET],  
 [colors[self.color] + " " \* self.size \* 2 + "=" \* self.size \* 15],  
 [colors[self.color] + " " \* self.size \* 2 + "=" \* self.size \* 12],  
 [colors[self.color] + " " \* self.size \* 2 + "=" \* self.size \* 9],  
 [colors[self.color] + " " \* self.size \* 2 + "=" \* self.size \* 6],  
 [colors[self.color] + " " \* self.size \* 2 + "=" \* self.size \* 3]  
 ]  
  
 if self.remove\_shadesPyramid:  
 self.remove\_shades()  
  
 for row in self.pyramid\_3D:  
 print(''.join(row))  
  
 def draw\_2D(self):  
 """  
 Draws a 2D representation of the pyramid.  
 """  
 init()  
  
 colors = {  
 'RED': Fore.RED,  
 'GREEN': Fore.GREEN,  
 'YELLOW': Fore.YELLOW,  
 'BLUE': Fore.BLUE,  
 'MAGENTA': Fore.MAGENTA,  
 'CYAN': Fore.CYAN,  
 'WHITE': Fore.WHITE  
 }  
  
 self.pyramid\_2D = [  
 [colors[self.color] + " " \* self.size \* 11 + "/\\"],  
 [colors[self.color] + " " \* self.size \* 10 + "/" + self.symbol \* self.size \* 2 + "\\"],  
 [colors[self.color] + " " \* self.size \* 9 + "/" + self.symbol \* self.size \* 4 + "\\"],  
 [colors[self.color] + " " \* self.size \* 8 + "/" + self.symbol \* self.size \* 6 + "\\"],  
 [colors[self.color] + " " \* self.size \* 7 + "/" + self.symbol \* self.size \* 8 + "\\"],  
 [colors[self.color] + " " \* self.size \* 6 + "/" + self.symbol \* self.size \* 10 + "\\"],  
 [colors[self.color] + " " \* self.size \* 5 + "/" + self.symbol \* self.size \* 12 + "\\"],  
 [colors[self.color] + " " \* self.size \* 4 + "/" + self.symbol \* self.size \* 14 + "\\"],  
 [colors[self.color] + " " \* self.size \* 3 + "/" + "\_" \* self.size \* 16 + "\\" + Fore.RESET]  
 ]  
  
 if self.remove\_shadesPyramid:  
 self.remove\_shades()  
  
 for row in self.pyramid\_2D:  
 print(''.join(row))  
  
 def set\_size(self, size):  
 """  
 Sets the size of the pyramid.  
 """  
 if size in [1, 2]:  
 self.size = size  
 else:  
 print("Invalid size. Please choose 1 or 2.")  
  
 def savePyramid\_3D(self, filename):  
 """  
 Saves the 3D representation of the pyramid to a file.  
 """  
 if self.pyramid\_3D is None:  
 print("Error: Draw the 3D pyramid first.")  
 return  
  
 pyramid\_3D\_no\_color = [''.join([self.remove\_color\_codes(cell) for cell in row]) for row in self.pyramid\_3D]  
  
 self.output\_directory = os.path.join('..', 'Data', 'Lab5')  
  
 os.makedirs(self.output\_directory, exist\_ok=True)  
  
 file\_path = os.path.join(self.output\_directory, filename)  
  
 with open(file\_path, 'w') as f:  
 for row in pyramid\_3D\_no\_color:  
 f.write(row + '\n')

Клас MenuLab5:  
class MenuLab5:  
 """  
 A class to manage the menu for Lab5.  
 """  
  
 def \_\_init\_\_(self, reader\_writer, validator):  
 """  
 Initializes the MenuLab5 with a reader/writer and a validator.  
  
 Args:  
 reader\_writer: The reader/writer to use for file operations.  
 validator: The validator to use for input validation.  
 """  
 self.reader\_writer = reader\_writer  
 self.validator = validator  
 self.figure = None  
 log\_file\_path = os.path.join('..', 'Data', 'Lab9', 'app.log')  
 logging.basicConfig(filename=log\_file\_path, filemode='w',  
 format='%(asctime)s - %(name)s - %(levelname)s - %(message)s')  
 logging.getLogger().setLevel(logging.INFO)  
  
 def display\_options(self):  
 """  
 Displays the menu options to the user.  
 """  
 current\_file\_directory = os.path.dirname(os.path.abspath(\_\_file\_\_))  
 relative\_config\_path = os.path.join('..', '..', 'Config', 'init.json')  
 config\_path = os.path.abspath(os.path.join(current\_file\_directory, relative\_config\_path))  
  
 with open(config\_path, 'r') as file:  
 config = json.load(file)  
  
 menu\_config = config['menu5']  
 menu\_builder = MenuBuilder(menu\_config)  
  
 menu\_builder.display()  
  
 def run(self):  
 """  
 Runs the menu, allowing the user to choose options and generate ASCII art.  
 """  
 while True:  
 self.display\_options()  
 choice = self.validator.validate\_input("Choose an option (1/2/3/4/5/6): ", ['1', '2', '3', '4', '5', '6'])  
 logging.info(f'User chose option: {choice}')  
 if choice == '1':  
 figure\_choice = self.validator.validate\_input("Choose a figure (Cube/Pyramid): ", ['Cube', 'Pyramid'])  
 if figure\_choice == 'Cube':  
 self.figure = Cube()  
 else:  
 self.figure = Pyramid()  
 elif choice == '2':  
 if self.figure is None:  
 print("Error: Choose a figure first (option 1).")  
 else:  
 size = self.validator.validate\_input("Enter size for figure (1 or 2): ", ['1', '2'])  
 color = self.validator.validate\_input(  
 "Enter color for figure (RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE): ",  
 ['RED', 'GREEN', 'YELLOW', 'BLUE', 'MAGENTA', 'CYAN', 'WHITE'])  
 symbol = self.reader\_writer.read\_input("Enter symbol for figure: ")  
 remove\_shades = self.validator.validate\_input(  
 "Do you want to remove shades? (yes/no): ", ['yes', 'no'])  
 self.figure = Cube(int(size), color, symbol, remove\_shades.lower() == 'yes') if isinstance(  
 self.figure, Cube) else Pyramid(int(size), color, symbol, remove\_shades.lower() == 'yes')  
 elif choice == '3':  
 if self.figure is None:  
 print("Error: Enter parameters for figure first (option 2).")  
 else:  
 self.figure.draw\_3D()  
 elif choice == '4':  
 if self.figure is None:  
 print("Error: Enter parameters for figure first (option 2).")  
 else:  
 self.figure.draw\_2D()  
 elif choice == '5':  
 if self.figure is None:  
 print("Error: Enter parameters for figure first (option 2).")  
 else:  
 filename = self.reader\_writer.read\_input("Enter a filename to save the 3D figure: ")  
 if isinstance(self.figure, Cube):  
 self.figure.saveCube\_3D(filename)  
 else:  
 self.figure.savePyramid\_3D(filename)  
 print(f"3D figure saved in file {filename}")  
 elif choice == '6':  
 print("Goodbye!")  
 logging.info('User exited.')  
 break  
 else:  
 print("Invalid choice. Try again.")

На рисунку 1 зображено результат у вигляді 3D куба.



**Рис. 1** 3D куб

На рисунку 1 зображено результат у вигляді 3D куба.



**Рис. 2** 3D піраміда

**Рис. 2** Архітектура проєкту

**Висновок:** я створила високорівневий об'єктно-орієнтований генератор 3D ASCII-арту, який дозволяє користувачам проектувати, відображати та маніпулювати 3D-фігурами в ASCII-арті.