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-32

Ольга ЧИГИРИК

Прийняв

доцент каф. ІСМ:

Сергій ЩЕРБАК

|  |  |
| --- | --- |
| **Балів** | **Дата** |
|  |  |

Львів-2023

**Мета**: Cтворення 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.

**Main.py**

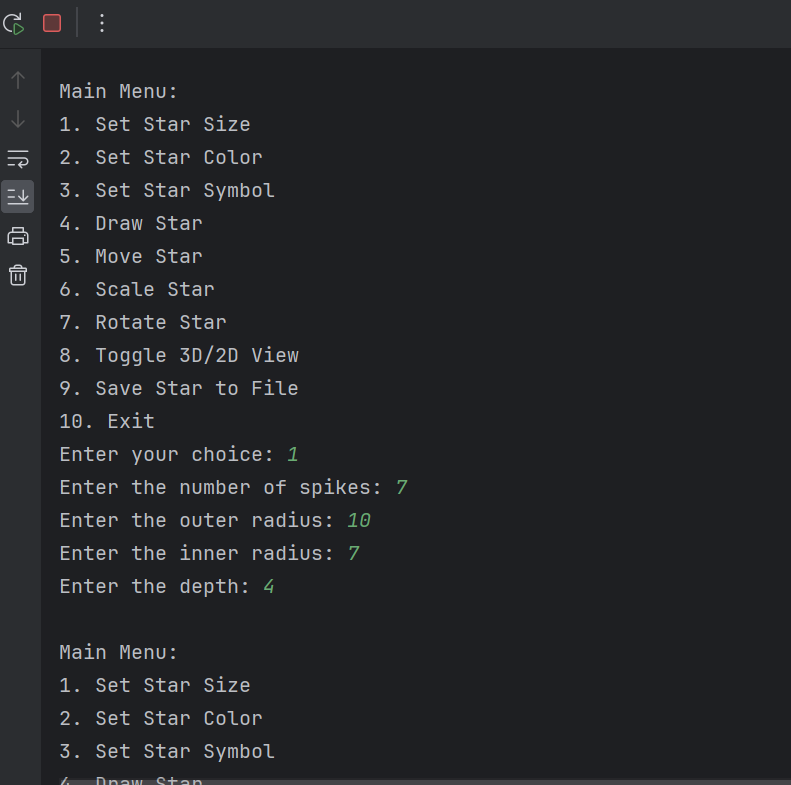
from shapes5 import Star3D  
import sys  
import io  
from Scripts.all\_python\_labs.shared\_lib.runnable import Runnable  
  
def run(self):  
 *"""  
 Run the main application loop.  
  
 Returns:  
 - None  
 """* # Run the main application event loop  
 self.root.mainloop()  
  
def main():  
 # Create a star object with default parameters  
 star = Star3D(spikes=5, outer\_radius=1, inner\_radius=1, depth=1, symbol='\*')  
  
 def save\_star\_to\_file(star, filename="star\_output.txt"):  
 # Save the original stdout  
 original\_stdout = sys.stdout  
  
 # Create a string buffer to capture the output  
 buffer = io.StringIO()  
  
 # Redirect stdout to the buffer  
 sys.stdout = buffer  
  
 # Draw the star in its current mode (2D or 3D)  
 star.draw\_star()  
  
 # Reset stdout to its original value  
 sys.stdout = original\_stdout  
  
 # Get the content from the buffer  
 star\_output = buffer.getvalue()  
  
 # Write the captured output to a file  
 with open(filename, "w") as f:  
 f.write(star\_output)  
  
 # Close the buffer  
 buffer.close()  
  
  
 while True:  
 print("\nMain Menu:")  
 print("1. Set Star Size")  
 print("2. Set Star Color")  
 print("3. Set Star Symbol")  
 print("4. Draw Star")  
 print("5. Move Star")  
 print("6. Scale Star")  
 print("7. Rotate Star")  
 print("8. Toggle 3D/2D View")  
 print("9. Save Star to File")  
 print("10. Exit")  
  
 choice = input("Enter your choice: ")  
  
 if choice == "1":  
 spikes = int(input("Enter the number of spikes: "))  
 outer\_radius = int(input("Enter the outer radius: "))  
 inner\_radius = int(input("Enter the inner radius: "))  
 depth = int(input("Enter the depth: "))  
 star.set\_size(spikes, outer\_radius, inner\_radius, depth)  
 elif choice == "2":  
 star.choose\_color()  
 elif choice == "3":  
 symbol = input("Enter the symbol to use for drawing the star: ")  
 star.set\_symbol(symbol)  
 elif choice == "4":  
 star.draw\_star()  
 elif choice == "5":  
 direction = input("Enter direction (left, right, up, down): ")  
 step = int(input("Enter steps to move: "))  
 if direction == "left":  
 star.move\_left(step)  
 elif direction == "right":  
 star.move\_right(step)  
 elif direction == "up":  
 star.move\_up(step)  
 elif direction == "down":  
 star.move\_down(step)  
 else:  
 print("Invalid direction.")  
  
 elif choice == "6":  
 star.center\_star()  
  
  
 elif choice == "7":  
  
 x\_angle = float(input("Enter rotation angle in degrees for X-axis: "))  
 y\_angle = float(input("Enter rotation angle in degrees for Y-axis: "))  
 z\_angle = float(input("Enter rotation angle in degrees for Z-axis: "))  
 star.rotate\_3d(x\_angle, y\_angle, z\_angle)  
  
 elif choice == "8":  
  
 star.toggle\_projection() # Assuming you have a method to toggle between 3D and 2D  
  
  
 elif choice == "9":  
 save\_star\_to\_file(star) # Save the current state of the star  
 # Save 2D projection  
 star.toggle\_projection() # Ensure it's in 2D mode  
 save\_star\_to\_file(star, "star\_2d\_output.txt")  
  
 # Save 3D projection  
 star.toggle\_projection() # Switch to 3D mode  
 save\_star\_to\_file(star, "star\_3d\_output.txt")  
 print(f"Star state saved to file.")  
  
 elif choice == "10":  
  
 print("Goodbye!")  
  
 break  
 else:  
  
 print("Invalid choice. Please try again.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Shapes.py**

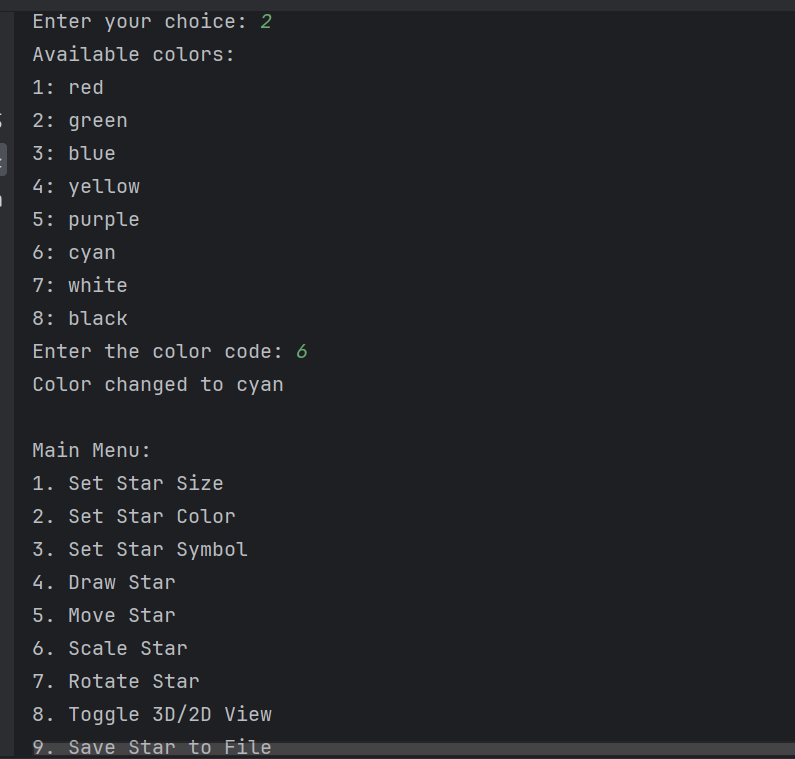
import math  
  
class Star3D:  
 ANSI\_COLOR\_CODES = {  
 'red': '\033[31m',  
 'green': '\033[32m',  
 'blue': '\033[34m',  
 'yellow': '\033[33m',  
 'purple': '\033[35m',  
 'cyan': '\033[36m',  
 'white': '\033[37m',  
 'black': '\033[30m',  
 'reset': '\033[0m' # Resets the color to default  
 }  
  
 def \_\_init\_\_(self, spikes, outer\_radius, inner\_radius, depth, symbol):  
 self.spikes = spikes  
 self.outer\_radius = outer\_radius  
 self.inner\_radius = inner\_radius  
 self.depth = depth  
 self.symbol = symbol  
 self.color\_code = 7 # Default to white  
 self.center\_x = self.outer\_radius + 1  
 self.center\_y = self.outer\_radius + 1  
 self.rotation\_angle = 0 # Initialize rotation\_angle before calculate\_vertices  
 self.vertices = self.calculate\_vertices()  
 self.is\_3d\_mode = False # Start in 2D mode  
  
 color\_table = {  
 1: 'red',  
 2: 'green',  
 3: 'blue',  
 4: 'yellow',  
 5: 'purple',  
 6: 'cyan',  
 7: 'white',  
 8: 'black'  
 }  
  
 def \_\_str\_\_(self):  
 # Construct a string representation of the star's current state  
 star\_info = [  
 f"Spikes: {self.spikes}",  
 f"Outer Radius: {self.outer\_radius}",  
 f"Inner Radius: {self.inner\_radius}",  
 f"Depth: {self.depth}",  
 f"Symbol: '{self.symbol}'",  
 f"Color: {self.color\_table[self.color\_code]}",  
 f"Center: ({self.center\_x}, {self.center\_y})",  
 f"Rotation Angle: {self.rotation\_angle}",  
 f"3D Mode: {'Enabled' if self.is\_3d\_mode else 'Disabled'}"  
 ]  
 return "Star3D(" + ', '.join(star\_info) + ")"  
  
 def calculate\_vertices(self):  
 *"""Calculate the vertices of the star in 2D."""* vertices = []  
 start\_angle = math.radians(self.rotation\_angle)  
 for i in range(self.spikes \* 2):  
 angle = start\_angle + i \* (math.pi / self.spikes)  
 radius = self.outer\_radius if i % 2 == 0 else self.inner\_radius  
 x = radius \* math.cos(angle)  
 y = radius \* math.sin(angle)  
 vertices.append((x, y, 0)) # 2D representation  
 return vertices  
  
 def set\_size(self, spikes, outer\_radius, inner\_radius, depth):  
 self.spikes = spikes  
 self.outer\_radius = outer\_radius  
 self.inner\_radius = inner\_radius  
 self.depth = depth  
 self.vertices = self.calculate\_vertices()  
  
 def choose\_color(self):  
 print("Available colors:")  
 for number, color in self.color\_table.items():  
 print(f"{number}: {color}")  
 color\_code = int(input("Enter the color code: "))  
 if color\_code in self.color\_table:  
 self.color\_code = color\_code  
 print(f"Color changed to {self.color\_table[color\_code]}")  
 else:  
 print("Invalid color code.")  
  
 def set\_symbol(self, symbol):  
 self.symbol = symbol  
  
  
 def scale(self, scale\_factor):  
 *"""Scales the star's size by a given factor."""* self.outer\_radius \*= scale\_factor  
 self.inner\_radius \*= scale\_factor  
 self.vertices = self.calculate\_vertices()  
  
 def toggle\_projection(self):  
 *"""Toggle between 3D and 2D projections."""* self.is\_3d\_mode = not self.is\_3d\_mode  
 print("Toggled to", "3D" if self.is\_3d\_mode else "2D", "mode.")  
  
 def draw\_star(self):  
 *"""Draws the star based on its vertices."""* # Initialize the size of the grid  
 size = int(self.outer\_radius \* 2 + 2)  
 grid = [[' ' for \_ in range(size)] for \_ in range(size)]  
  
 # Choose vertices based on the current mode  
 vertices = self.vertices if self.is\_3d\_mode else self.project\_to\_2d()  
  
 # Draw lines between the vertices  
 for i in range(len(vertices)):  
 start\_vertex = vertices[i]  
 end\_vertex = vertices[(i + 1) % len(vertices)]  
 self.draw\_line(grid, start\_vertex[0], start\_vertex[1], end\_vertex[0], end\_vertex[1])  
  
 # Print the star grid  
 for row in grid:  
 print(''.join(row))  
  
 # Convert 3D vertices to 2D if necessary  
 vertices\_2d = self.project\_to\_2d()  
  
 # Draw lines between the vertices  
 for i in range(len(vertices\_2d)):  
 start\_vertex = vertices\_2d[i]  
 end\_vertex = vertices\_2d[(i + 1) % len(vertices\_2d)]  
 self.draw\_line(grid, \*start\_vertex, \*end\_vertex)  
  
 # Print the star grid  
 for row in grid:  
 print(''.join(row))  
  
 def draw\_line(self, grid, x0, y0, x1, y1):  
 *"""Draws a line on the grid from (x0, y0) to (x1, y1) using Bresenham's algorithm."""* # Convert coordinates to integers  
 x0, y0, x1, y1 = map(int, [x0, y0, x1, y1])  
  
 dx = abs(x1 - x0)  
 dy = abs(y1 - y0)  
 sx = 1 if x0 < x1 else -1  
 sy = 1 if y0 < y1 else -1  
 err = dx - dy  
  
 # Get the ANSI color code  
 color\_code = self.ANSI\_COLOR\_CODES[self.color\_table[self.color\_code]]  
  
 while True:  
 if 0 <= x0 < len(grid) and 0 <= y0 < len(grid[0]):  
 grid[y0][x0] = color\_code + self.symbol + self.ANSI\_COLOR\_CODES['reset']  
  
 if x0 == x1 and y0 == y1:  
 break  
 e2 = 2 \* err  
 if e2 > -dy:  
 err -= dy  
 x0 += sx  
 if e2 < dx:  
 err += dx  
 y0 += sy  
  
 def rotate\_3d(self, x\_angle, y\_angle, z\_angle):  
 *"""Rotates the star in 3D space."""* # Convert angles to radians  
 x\_angle = math.radians(x\_angle)  
 y\_angle = math.radians(y\_angle)  
 z\_angle = math.radians(z\_angle)  
  
 rotated\_vertices = []  
 for x, y, z in self.vertices:  
 # Rotation around the X-axis  
 y\_rotated = y \* math.cos(x\_angle) - z \* math.sin(x\_angle)  
 z\_rotated = y \* math.sin(x\_angle) + z \* math.cos(x\_angle)  
  
 # Rotation around the Y-axis  
 x\_rotated = x \* math.cos(y\_angle) + z\_rotated \* math.sin(y\_angle)  
 z\_rotated = -x \* math.sin(y\_angle) + z\_rotated \* math.cos(y\_angle)  
  
 # Rotation around the Z-axis  
 x\_rotated = x\_rotated \* math.cos(z\_angle) - y\_rotated \* math.sin(z\_angle)  
 y\_rotated = x\_rotated \* math.sin(z\_angle) + y\_rotated \* math.cos(z\_angle)  
  
 rotated\_vertices.append((x\_rotated, y\_rotated, z\_rotated))  
  
 self.vertices = rotated\_vertices  
  
 def project\_to\_2d(self):  
 *"""Projects the 3D star to 2D."""* return [(x, y) for x, y, z in self.vertices]  
  
 def calculate\_vertices(self):  
 *"""Calculate the vertices of the star."""* vertices = []  
 start\_angle = math.radians(self.rotation\_angle) # Convert the rotation angle to radians  
 for i in range(self.spikes \* 2):  
 angle = start\_angle + i \* (math.pi / self.spikes)  
 radius = self.outer\_radius if i % 2 == 0 else self.inner\_radius  
 x = self.center\_x + radius \* math.cos(angle)  
 y = self.center\_y + radius \* math.sin(angle)  
 vertices.append((x, y, 0)) # Assuming a z-coordinate of 0 for 2D representation  
 return vertices  
  
 def prompt\_for\_rotation(self):  
 *"""Prompts the user to enter rotation angles and applies the rotation."""* while True:  
 try:  
 x\_angle = float(input("Enter rotation angle in degrees for X-axis: "))  
 y\_angle = float(input("Enter rotation angle in degrees for Y-axis: "))  
 z\_angle = float(input("Enter rotation angle in degrees for Z-axis: "))  
  
 self.rotate\_3d(x\_angle, y\_angle, z\_angle)  
 break # Correctly placed inside the loop  
 except ValueError:  
 print("Invalid input. Please enter valid numbers for X, Y, and Z angles.")  
  
 def move\_left(self, step=1):  
 *"""Move the star to the left by a specified number of steps."""* self.center\_x = max(1, self.center\_x - step)  
 self.vertices = self.calculate\_vertices()  
  
 def move\_right(self, step=1):  
 *"""Move the star to the right by a specified number of steps."""* self.center\_x += step  
 self.vertices = self.calculate\_vertices()  
  
 def move\_up(self, step=1):  
 *"""Move the star up by a specified number of steps."""* self.center\_y = max(1, self.center\_y - step)  
 self.vertices = self.calculate\_vertices()  
  
 def move\_down(self, step=1):  
 *"""Move the star down by a specified number of steps."""* self.center\_y += step  
 self.vertices = self.calculate\_vertices()  
  
 def center\_star(self):  
 *"""Center the star on the grid."""* self.center\_x = self.outer\_radius + 1  
 self.center\_y = self.outer\_radius + 1  
 self.vertices = self.calculate\_vertices()

**Art\_functions.py**

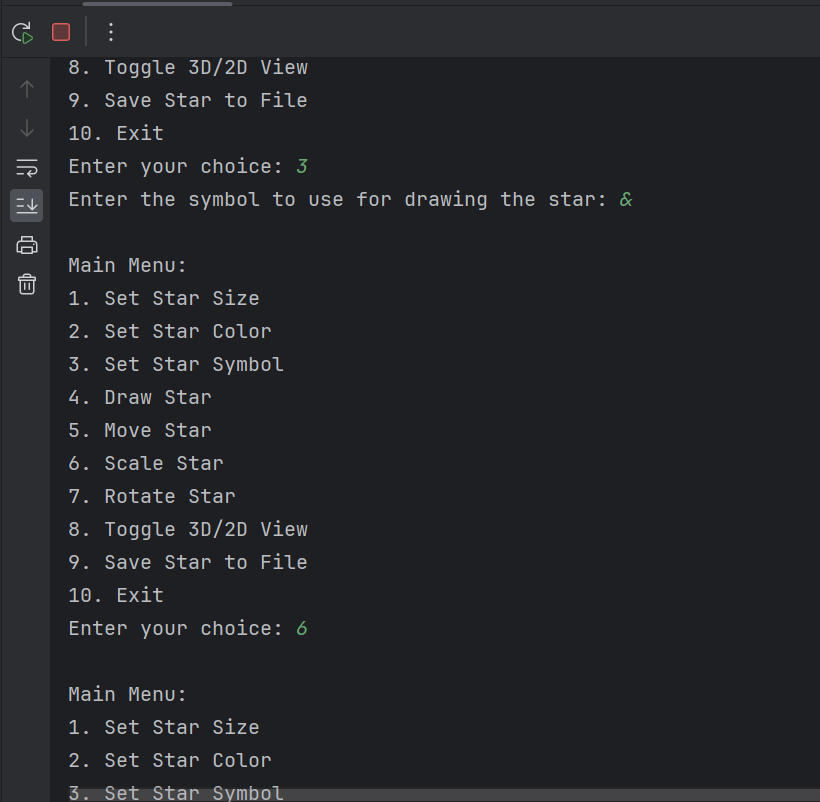
import math  
  
class Star3D:  
 ANSI\_COLOR\_CODES = {  
 'red': '\033[31m',  
 'green': '\033[32m',  
 'blue': '\033[34m',  
 'yellow': '\033[33m',  
 'purple': '\033[35m',  
 'cyan': '\033[36m',  
 'white': '\033[37m',  
 'black': '\033[30m',  
 'reset': '\033[0m' # Resets the color to default  
 }  
  
 def \_\_init\_\_(self, spikes, outer\_radius, inner\_radius, depth, symbol):  
 self.spikes = spikes  
 self.outer\_radius = outer\_radius  
 self.inner\_radius = inner\_radius  
 self.depth = depth  
 self.symbol = symbol  
 self.color\_code = 7 # Default to white  
 self.center\_x = self.outer\_radius + 1  
 self.center\_y = self.outer\_radius + 1  
 self.rotation\_angle = 0 # Initialize rotation\_angle before calculate\_vertices  
 self.vertices = self.calculate\_vertices()  
 self.is\_3d\_mode = False # Start in 2D mode  
  
 color\_table = {  
 1: 'red',  
 2: 'green',  
 3: 'blue',  
 4: 'yellow',  
 5: 'purple',  
 6: 'cyan',  
 7: 'white',  
 8: 'black'  
 }  
  
 def \_\_str\_\_(self):  
 # Construct a string representation of the star's current state  
 star\_info = [  
 f"Spikes: {self.spikes}",  
 f"Outer Radius: {self.outer\_radius}",  
 f"Inner Radius: {self.inner\_radius}",  
 f"Depth: {self.depth}",  
 f"Symbol: '{self.symbol}'",  
 f"Color: {self.color\_table[self.color\_code]}",  
 f"Center: ({self.center\_x}, {self.center\_y})",  
 f"Rotation Angle: {self.rotation\_angle}",  
 f"3D Mode: {'Enabled' if self.is\_3d\_mode else 'Disabled'}"  
 ]  
 return "Star3D(" + ', '.join(star\_info) + ")"  
  
 def calculate\_vertices(self):  
 *"""Calculate the vertices of the star in 2D."""* vertices = []  
 start\_angle = math.radians(self.rotation\_angle)  
 for i in range(self.spikes \* 2):  
 angle = start\_angle + i \* (math.pi / self.spikes)  
 radius = self.outer\_radius if i % 2 == 0 else self.inner\_radius  
 x = radius \* math.cos(angle)  
 y = radius \* math.sin(angle)  
 vertices.append((x, y, 0)) # 2D representation  
 return vertices  
  
 def set\_size(self, spikes, outer\_radius, inner\_radius, depth):  
 self.spikes = spikes  
 self.outer\_radius = outer\_radius  
 self.inner\_radius = inner\_radius  
 self.depth = depth  
 self.vertices = self.calculate\_vertices()  
  
 def choose\_color(self):  
 print("Available colors:")  
 for number, color in self.color\_table.items():  
 print(f"{number}: {color}")  
 color\_code = int(input("Enter the color code: "))  
 if color\_code in self.color\_table:  
 self.color\_code = color\_code  
 print(f"Color changed to {self.color\_table[color\_code]}")  
 else:  
 print("Invalid color code.")  
  
 def set\_symbol(self, symbol):  
 self.symbol = symbol  
  
  
 def scale(self, scale\_factor):  
 *"""Scales the star's size by a given factor."""* self.outer\_radius \*= scale\_factor  
 self.inner\_radius \*= scale\_factor  
 self.vertices = self.calculate\_vertices()  
  
 def toggle\_projection(self):  
 *"""Toggle between 3D and 2D projections."""* self.is\_3d\_mode = not self.is\_3d\_mode  
 print("Toggled to", "3D" if self.is\_3d\_mode else "2D", "mode.")  
  
 def draw\_star(self):  
 *"""Draws the star based on its vertices."""* # Initialize the size of the grid  
 size = int(self.outer\_radius \* 2 + 2)  
 grid = [[' ' for \_ in range(size)] for \_ in range(size)]  
  
 # Choose vertices based on the current mode  
 vertices = self.vertices if self.is\_3d\_mode else self.project\_to\_2d()  
  
 # Draw lines between the vertices  
 for i in range(len(vertices)):  
 start\_vertex = vertices[i]  
 end\_vertex = vertices[(i + 1) % len(vertices)]  
 self.draw\_line(grid, start\_vertex[0], start\_vertex[1], end\_vertex[0], end\_vertex[1])  
  
 # Print the star grid  
 for row in grid:  
 print(''.join(row))  
  
 # Convert 3D vertices to 2D if necessary  
 vertices\_2d = self.project\_to\_2d()  
  
 # Draw lines between the vertices  
 for i in range(len(vertices\_2d)):  
 start\_vertex = vertices\_2d[i]  
 end\_vertex = vertices\_2d[(i + 1) % len(vertices\_2d)]  
 self.draw\_line(grid, \*start\_vertex, \*end\_vertex)  
  
 # Print the star grid  
 for row in grid:  
 print(''.join(row))  
  
 def draw\_line(self, grid, x0, y0, x1, y1):  
 *"""Draws a line on the grid from (x0, y0) to (x1, y1) using Bresenham's algorithm."""* # Convert coordinates to integers  
 x0, y0, x1, y1 = map(int, [x0, y0, x1, y1])  
  
 dx = abs(x1 - x0)  
 dy = abs(y1 - y0)  
 sx = 1 if x0 < x1 else -1  
 sy = 1 if y0 < y1 else -1  
 err = dx - dy  
  
 # Get the ANSI color code  
 color\_code = self.ANSI\_COLOR\_CODES[self.color\_table[self.color\_code]]  
  
 while True:  
 if 0 <= x0 < len(grid) and 0 <= y0 < len(grid[0]):  
 grid[y0][x0] = color\_code + self.symbol + self.ANSI\_COLOR\_CODES['reset']  
  
 if x0 == x1 and y0 == y1:  
 break  
 e2 = 2 \* err  
 if e2 > -dy:  
 err -= dy  
 x0 += sx  
 if e2 < dx:  
 err += dx  
 y0 += sy  
  
 def rotate\_3d(self, x\_angle, y\_angle, z\_angle):  
 *"""Rotates the star in 3D space."""* # Convert angles to radians  
 x\_angle = math.radians(x\_angle)  
 y\_angle = math.radians(y\_angle)  
 z\_angle = math.radians(z\_angle)  
  
 rotated\_vertices = []  
 for x, y, z in self.vertices:  
 # Rotation around the X-axis  
 y\_rotated = y \* math.cos(x\_angle) - z \* math.sin(x\_angle)  
 z\_rotated = y \* math.sin(x\_angle) + z \* math.cos(x\_angle)  
  
 # Rotation around the Y-axis  
 x\_rotated = x \* math.cos(y\_angle) + z\_rotated \* math.sin(y\_angle)  
 z\_rotated = -x \* math.sin(y\_angle) + z\_rotated \* math.cos(y\_angle)  
  
 # Rotation around the Z-axis  
 x\_rotated = x\_rotated \* math.cos(z\_angle) - y\_rotated \* math.sin(z\_angle)  
 y\_rotated = x\_rotated \* math.sin(z\_angle) + y\_rotated \* math.cos(z\_angle)  
  
 rotated\_vertices.append((x\_rotated, y\_rotated, z\_rotated))  
  
 self.vertices = rotated\_vertices  
  
 def project\_to\_2d(self):  
 *"""Projects the 3D star to 2D."""* return [(x, y) for x, y, z in self.vertices]  
  
 def calculate\_vertices(self):  
 *"""Calculate the vertices of the star."""* vertices = []  
 start\_angle = math.radians(self.rotation\_angle) # Convert the rotation angle to radians  
 for i in range(self.spikes \* 2):  
 angle = start\_angle + i \* (math.pi / self.spikes)  
 radius = self.outer\_radius if i % 2 == 0 else self.inner\_radius  
 x = self.center\_x + radius \* math.cos(angle)  
 y = self.center\_y + radius \* math.sin(angle)  
 vertices.append((x, y, 0)) # Assuming a z-coordinate of 0 for 2D representation  
 return vertices  
  
 def prompt\_for\_rotation(self):  
 *"""Prompts the user to enter rotation angles and applies the rotation."""* while True:  
 try:  
 x\_angle = float(input("Enter rotation angle in degrees for X-axis: "))  
 y\_angle = float(input("Enter rotation angle in degrees for Y-axis: "))  
 z\_angle = float(input("Enter rotation angle in degrees for Z-axis: "))  
  
 self.rotate\_3d(x\_angle, y\_angle, z\_angle)  
 break # Correctly placed inside the loop  
 except ValueError:  
 print("Invalid input. Please enter valid numbers for X, Y, and Z angles.")  
  
 def move\_left(self, step=1):  
 *"""Move the star to the left by a specified number of steps."""* self.center\_x = max(1, self.center\_x - step)  
 self.vertices = self.calculate\_vertices()  
  
 def move\_right(self, step=1):  
 *"""Move the star to the right by a specified number of steps."""* self.center\_x += step  
 self.vertices = self.calculate\_vertices()  
  
 def move\_up(self, step=1):  
 *"""Move the star up by a specified number of steps."""* self.center\_y = max(1, self.center\_y - step)  
 self.vertices = self.calculate\_vertices()  
  
 def move\_down(self, step=1):  
 *"""Move the star down by a specified number of steps."""* self.center\_y += step  
 self.vertices = self.calculate\_vertices()  
  
 def center\_star(self):  
 *"""Center the star on the grid."""* self.center\_x = self.outer\_radius + 1  
 self.center\_y = self.outer\_radius + 1  
 self.vertices = self.calculate\_vertices()



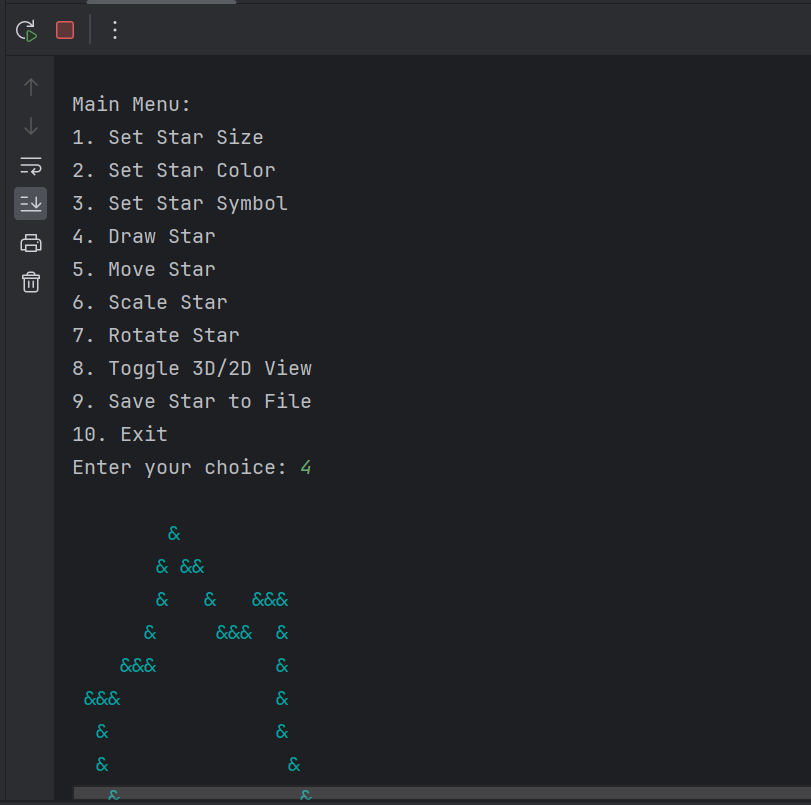
*Рис. 1 Результат виконання завдання*



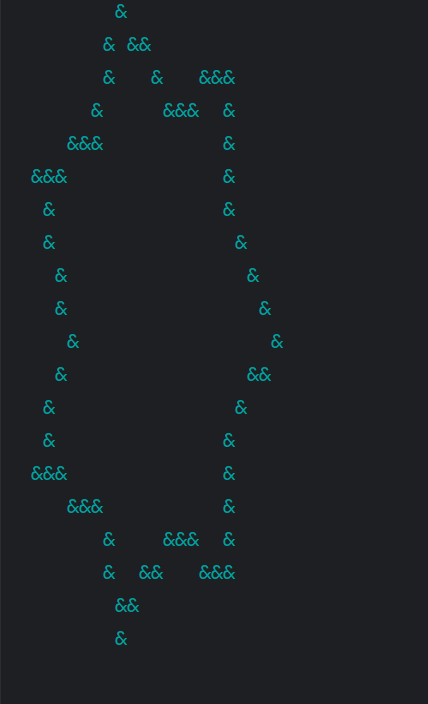
*Рис. 2 Результат виконання завдання*



*Рис. 3 Результат виконання завдання*



*Рис. 4 Результат виконання завдання*



*Рис. 5 Результат виконання завдання*

**Висновок:** Виконуючи ці завдання, я створила 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, сприяв творчому підходу до створення ASCII-арту.