МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

КАФЕДРА ЕЛЕКТРОННОЇ ТЕХНІКИ

Практична робота №2

з дисципліни «Методи інженерії комп’ютерних систем»

на тему «Породжувальні патерни»

Варіант 9

Виконав:

студент гр. ІПЗ-20

Софія БОНДАРЧУК

Перевірив:

асистент кафедри ЕТ

Артем ЛЮБИМОВ

Луцьк – 2023

Завдання

В роботі потрібно реалізувати два породжувальні патерни, кожен – за власним завданням. N = 9.



Патерн 1. Генерація абстрактної картини

Розробити систему фігур (лінія, прямокутник, коло). Шляхом реалізації *фабричного методу* реалізувати випадковий вибір класу, за допомогою якого на екрані буде відмальовано випадкову кількість об'єктів із випадковими параметрами (розташування, розміри, колір)

Патерн 2. Реєстрація користувача

Ви розробляєте гру-симулятор футбольного матчу. Для наповнення футбольної команди вам необхідні голкіпер, захисники, півзахисники і нападники. За допомогою патерна «Прототип» реалізуйте процедуру наповнення двох футбольних команд для матчу, причому у кожної команди може бути своя тактична схема (4-3-3, 4-4-2, 3-5-2 тощо).

Хід роботи

Патерн 1

Створимо абстрактний клас Shape, потім створимо конкретні класи Line, Rectangle і Circle, які успадковуються від Shape і реалізують метод create\_random\_shape. У кожному з цих конкретних класів ми задаємо конкретну логіку створення об'єкта фігури цього типу. Фабричний метод для випадкової генерації фігур. Цей метод використовує фабричний підхід для створення випадкових фігур. Вибирається випадковий тип фігури, після чого генеруються випадкові параметри для цієї фігури.

Файл класу Shape

import random

import turtle

import colorsys

class Shape:

    def draw(self):

        # Placeholder method for drawing a shape. To be implemented by subclasses.

        pass

def random\_color():

    # Generate a random RGB color using the HSV color model

    r, g, b = [int(x \* 255) for x in colorsys.hsv\_to\_rgb(random.random(), 1, 1)]

    # Format the RGB values into a hexadecimal color code

    return "#{:02X}{:02X}{:02X}".format(r, g, b)

Файл класу Line

from shape import Shape, random\_color

import turtle

class Line(Shape):

    def \_\_init\_\_(self, x1, y1, x2, y2):

        # Constructor for Line class.

        self.x1 = x1

        self.y1 = y1

        self.x2 = x2

        self.y2 = y2

    def draw(self):

        turtle.penup()

        turtle.goto(self.x1, self.y1)

        turtle.pendown()

        turtle.pencolor(random\_color())

        turtle.goto(self.x2, self.y2)

Файл класу Circle

from shape import Shape, random\_color

import turtle

# Define a Circle class that inherits from the Shape base class

class Circle(Shape):

    def \_\_init\_\_(self, x, y, radius):

        # Constructor to initialize the Circle object with position and radius

        self.x = x

        self.y = y

        self.radius = radius

    def draw(self):

        turtle.penup()

        turtle.goto(self.x, self.y - self.radius)

        turtle.pendown()

        turtle.pencolor(random\_color())

        turtle.circle(self.radius)

Файл класу Rectangle

from shape import Shape, random\_color

import turtle

class Rectangle(Shape):

    def \_\_init\_\_(self, x, y, width, height):

         # Constructor for Rectangle class.

        self.x = x

        self.y = y

        self.width = width

        self.height = height

    def draw(self):

        turtle.penup()

        turtle.goto(self.x, self.y)

        turtle.pendown()

        turtle.pencolor(random\_color())

        for \_ in range(2):

            turtle.forward(self.width)

            turtle.left(90)

            turtle.forward(self.height)

            turtle.left(90)

Файл функції create\_random\_shape()

import random

import turtle

import colorsys

# Generate a random color in hexadecimal format

def random\_color():

    r, g, b = [int(x \* 255) for x in colorsys.hsv\_to\_rgb(random.random(), 1, 1)]

    return "#{:02X}{:02X}{:02X}".format(r, g, b)

def create\_random\_shape():

    pass

Файл main

import turtle

import random

from line import Line

from rectangle import Rectangle

from circle import Circle

# Function to create a random shape (Line, Rectangle, or Circle)

def create\_random\_shape():

    shape\_types = [Line, Rectangle, Circle]

    random\_shape\_type = random.choice(shape\_types)

    if random\_shape\_type == Line:

        # Generate random coordinates for Line

        x1, y1, x2, y2 = random.randint(-200, 200), random.randint(-200, 200), random.randint(-200, 200), random.randint(-200, 200)

        return Line(x1, y1, x2, y2)

    elif random\_shape\_type == Rectangle:

        # Generate random parameters for Rectangle

        x, y, width, height = random.randint(-200, 200), random.randint(-200, 200), random.randint(10, 100), random.randint(10, 100)

        return Rectangle(x, y, width, height)

    else:

        # Generate random parameters for Circle

        x, y, radius = random.randint(-200, 200), random.randint(-200, 200), random.randint(10, 100)

        return Circle(x, y, radius)

# Set turtle speed

turtle.speed(1)

# Create and draw a random number of shapes

for \_ in range(random.randint(1, 10)):

    shape = create\_random\_shape()

    shape.draw()

# Exit on click

turtle.exitonclick()

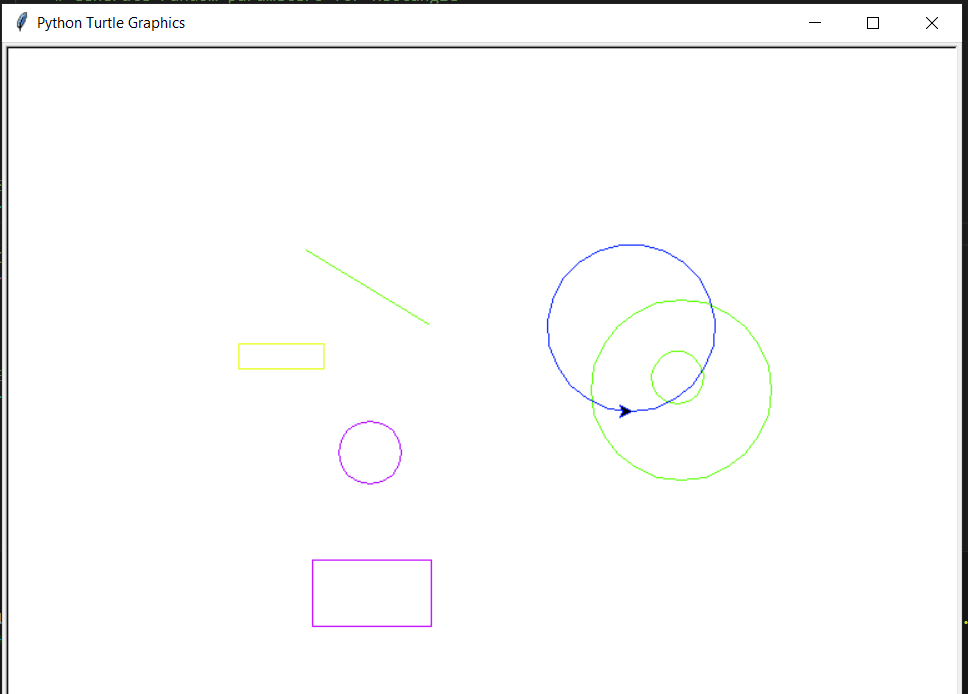


Рисунок 1 – Генерація абстрактної картини

Патерн 2

У коді застосовується паттерн "Прототип" для створення копій футбольних команд та гравців. Клас Player визначає базовий тип для гравців і містить метод clone, який повинен бути реалізований у його підкласах - Goalkeeper, Defender, Midfielder та Forward.

Кожен з цих підкласів представляє гравця на конкретній позиції та реалізує метод clone, який дозволяє створювати копії гравців. Клас TeamFormation визначає формацію футбольної команди та також містить метод clone.

Клас FootballTeam представляє саму команду і має метод add\_player, який додає гравця до команди. Також він має метод clone, який створює копію команди, включаючи копії формації та гравців.

Файл класу Player

class Player:

    def \_\_init\_\_(self, position):

        self.position = position

    def clone(self):

        pass

Файл класу Midfielder

from player import Player

class Midfielder(Player):

    def clone(self):

        return Midfielder(self.position)

Файл класу Goalkeeper

from player import Player

class Goalkeeper(Player):

    def clone(self):

        return Goalkeeper(self.position)

Файл класу Forward

from player import Player

class Forward(Player):

    def clone(self):

        return Forward(self.position)

Файл класу Defender

from player import Player

class Defender(Player):

    def clone(self):

        return Defender(self.position)

Файл класу FootballTeam

from team\_formation import TeamFormation

class FootballTeam:

    def \_\_init\_\_(self, name, formation):

        # Constructor for FootballTeam class.

        self.name = name

        self.formation = formation

        self.players = []

    def add\_player(self, player):

        # Add a player to the football team.

        self.players.append(player)

    def clone(self):

        # Clone the football team.

        team\_copy = FootballTeam(self.name, self.formation.clone())

        for player in self.players:

            team\_copy.add\_player(player.clone())

        return team\_copy

Файл класу TeamFormation

class TeamFormation:

    def \_\_init\_\_(self, formation):

        self.formation = formation

    def clone(self):

        pass

Файл main

import tkinter as tk

from goalkeeper import Goalkeeper

from midfielder import Midfielder

from defender import Defender

from forward import Forward

from player import Player

from football\_team import TeamFormation, FootballTeam

# Function to display information about a team in a pop-up window

def show\_team\_info(team):

    team\_info = f"{team.name} ({team.formation.formation}):\n"

    for player in team.players:

        team\_info += f"{player.position}\n"

    info\_window = tk.Toplevel(root)

    info\_window.title("Team Info")

    label = tk.Label(info\_window, text=team\_info)

    label.pack()

# Example of creating players and teams based on prototypes:

def main():

    goalkeeper\_prototype = Goalkeeper("Goalkeeper")

    defender\_prototype = Defender("Defender")

    midfielder\_prototype = Midfielder("Midfielder")

    forward\_prototype = Forward("Forward")

    formation\_4\_3\_3 = TeamFormation("4-3-3")

    formation\_4\_4\_2 = TeamFormation("4-4-2")

    team\_a = FootballTeam("Team A", formation\_4\_3\_3)

    team\_b = FootballTeam("Team B", formation\_4\_4\_2)

    team\_a.add\_player(goalkeeper\_prototype.clone())

    team\_a.add\_player(defender\_prototype.clone())

    team\_a.add\_player(midfielder\_prototype.clone())

    team\_a.add\_player(forward\_prototype.clone())

    team\_a.add\_player(midfielder\_prototype.clone())

    team\_a.add\_player(defender\_prototype.clone())

    team\_a.add\_player(midfielder\_prototype.clone())

    team\_a.add\_player(forward\_prototype.clone())

    team\_a.add\_player(forward\_prototype.clone())

    team\_b.add\_player(goalkeeper\_prototype.clone())

    team\_b.add\_player(defender\_prototype.clone())

    team\_b.add\_player(defender\_prototype.clone())

    team\_b.add\_player(midfielder\_prototype.clone())

    team\_b.add\_player(midfielder\_prototype.clone())

    team\_b.add\_player(midfielder\_prototype.clone())

    team\_b.add\_player(midfielder\_prototype.clone())

    team\_b.add\_player(forward\_prototype.clone())

    team\_b.add\_player(forward\_prototype.clone())

    # Display information about teams in pop-up windows

    show\_team\_info(team\_a)

    show\_team\_info(team\_b)

if \_\_name\_\_ == "\_\_main\_\_":

    root = tk.Tk()

    main()

    root.mainloop()

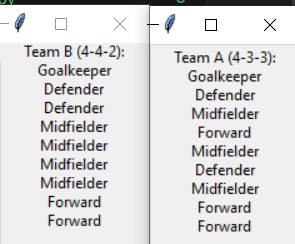


Рисунок 2 – Створення команд

Начало формы

Висновок: у ході виконання завдань, пов'язаних з фабричним методом та патерном "Прототип", було придбано навички створювати об'єкти зі складними параметрами та забезпечувати гнучкість та розширюваність коду, а також структурувати код для покращення його читабельності і обслуговуваності.