МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчёт по лабораторной работе №7

Специальность ПО11

Выполнил
С. С. Жватель
студент группы ПО11

Проверил
А. А. Крощенко
ст. преп. кафедры ИИТ,
12.04.2025 г.

Брест 2025

Цель работы: освоить возможности языка программирования Python в разработке оконных приложений.

Задание 1: Построение графических примитивов и надписей

Требования к выполнению

• Реализовать соответствующие классы, указанные в задании;

• Организовать ввод параметров для создания объектов (использовать экранные компоненты);

• Осуществить визуализацию графических примитивов

Важное замечание: должна быть предусмотрена возможность приостановки выполнения визуализации, изменения параметров «на лету» и снятия скриншотов с сохранением в текущую активную директорию.

Для всех динамических сцен необходимо задавать параметр скорости!

Создать классы Point и Line. Объявить список из n объектов класса Point. Для объекта класса Line определить, какие из объектов Point лежат на одной стороне от прямой линии и какие на другой. Реализовать ввод данных для объекта Line и случайное задание данных для объекта Point.

Выполнение:

Код программы:

```
"""Module for visualizing points relative to a line using Matplotlib and Tkinter."""

import random
import tkinter as tk
from datetime import datetime
from tkinter import messagebox, ttk

import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg


class Point:
    """Represents a 2D point for point-line visualization."""

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def get_x(self):
        """Return the x-coordinate of the point."""
        return self.x

    def get_y(self):
        """Return the y-coordinate of the point."""
        return self.y

    def is_within_bounds(self):
        """Check if the point is within the visualization range [-10, 10]."""
        return -10 <= self.x <= 10 and -10 <= self.y <= 10

    def __str__(self):
        """Return a string representation of the point."""
        return f"VisPoint({self.x}, {self.y})"


class Line:
```

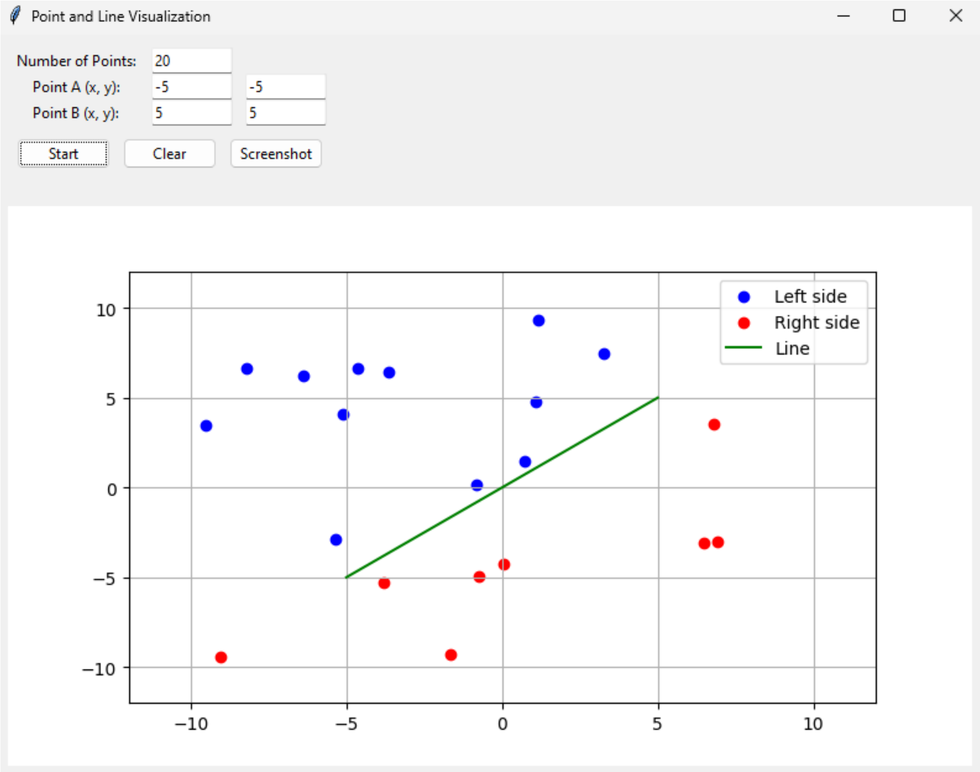
```
    """Represents a line segment between two points."""

    def __init__(self, point_a, point_b):
        self.point_a = point_a
        self.point_b = point_b

    def side(self, point):
        """Determine which side of the line a point lies on using cross product.

        Args:
            point (Point): The point to check.

        Returns:
            float: Positive if left, negative if right, zero if on the line.
        """
        ax = self.point_b.x - self.point_a.x
        ay = self.point_b.y - self.point_a.y
        bx = point.x - self.point_a.x
        by = point.y - self.point_a.y
        return ax * by - ay * bx

    def get_point_a(self):
        """Return the first point of the line."""
        return self.point_a

    def get_point_b(self):
        """Return the second point of the line."""
        return self.point_b


class VisualizationApp:
    """Application for visualizing points and a line segment."""

    def __init__(self, root):
        """Initialize the visualization application.

        Args:
            root (tk.Tk): The Tkinter root window.
        """
        self.root = root
        self.root.title("Point and Line Visualization")
        self.points = []
        self.line = None
        self.plot_config = {
            "fig": plt.figure(),
            "ax": None,
            "canvas": None,
            "plot_frame": None,
        }
        self.entries = {
            "num_points": None,
            "ax": None,
            "ay": None,
```

```
            "bx": None,
            "by": None,
        }

        self.create_gui()
        self.plot_config["ax"] = self.plot_config["fig"].add_subplot(111)
        self.plot_config["canvas"] = FigureCanvasTkAgg(
            self.plot_config["fig"],
            master=self.plot_config["plot_frame"],
        )
        self.plot_config["canvas"].get_tk_widget().pack(fill=tk.BOTH, expand=True)

    def create_gui(self):
        """Create the GUI elements for the application."""
        control_frame = ttk.Frame(self.root)
        control_frame.pack(pady=10, padx=10, fill=tk.X)

        ttk.Label(control_frame, text="Number of Points:").grid(row=0, column=0, padx=5)
        self.entries["num_points"] = ttk.Entry(control_frame, width=10)
        self.entries["num_points"].grid(row=0, column=1, padx=5)
        self.entries["num_points"].insert(0, "20")

        ttk.Label(control_frame, text="Point A (x, y):").grid(row=1, column=0, padx=5)
        self.entries["ax"] = ttk.Entry(control_frame, width=10)
        self.entries["ax"].grid(row=1, column=1, padx=5)
        self.entries["ax"].insert(0, "-5")
        self.entries["ay"] = ttk.Entry(control_frame, width=10)
        self.entries["ay"].grid(row=1, column=2, padx=5)
        self.entries["ay"].insert(0, "-5")

        ttk.Label(control_frame, text="Point B (x, y):").grid(row=2, column=0, padx=5)
        self.entries["bx"] = ttk.Entry(control_frame, width=10)
        self.entries["bx"].grid(row=2, column=1, padx=5)
        self.entries["bx"].insert(0, "5")
        self.entries["by"] = ttk.Entry(control_frame, width=10)
        self.entries["by"].grid(row=2, column=2, padx=5)
        self.entries["by"].insert(0, "5")

        button_frame = ttk.Frame(control_frame)
        button_frame.grid(row=3, column=0, columnspan=3, pady=10)

        ttk.Button(button_frame, text="Start", command=self.start_visualization).pack(
            side=tk.LEFT,
            padx=5,
        )
        ttk.Button(button_frame, text="Clear", command=self.clear_visualization).pack(
            side=tk.LEFT,
            padx=5,
        )
        ttk.Button(button_frame, text="Screenshot", command=self.take_screenshot).pack(
            side=tk.LEFT,
            padx=5,
        )
```

```
        self.plot_config["plot_frame"] = ttk.Frame(self.root)
        self.plot_config["plot_frame"].pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

    def generate_points(self, n):
        """Generate n random points within the range [-10, 10].

        Args:
            n (int): Number of points to generate.
        """
        self.points = [Point(random.uniform(-10, 10), random.uniform(-10, 10)) for _ in range(n)]

    def start_visualization(self):
        """Start the visualization based on user input."""
        try:
            n = int(self.entries["num_points"].get())
            ax = float(self.entries["ax"].get())
            ay = float(self.entries["ay"].get())
            bx = float(self.entries["bx"].get())
            by = float(self.entries["by"].get())

            if n <= 0:
                messagebox.showerror("Error", "Number of points must be positive")
                return

            if ax == bx and ay == by:
                messagebox.showerror("Error", "Points A and B must be different")
                return

            point_a = Point(ax, ay)
            point_b = Point(bx, by)
            if not (point_a.is_within_bounds() and point_b.is_within_bounds()):
                messagebox.showerror(
                    "Error",
                    "Coordinates of points A and B must be between -10 and 10",
                )
                return

            self.line = Line(point_a, point_b)
            self.generate_points(n)
            self.draw_visualization()

        except ValueError:
            messagebox.showerror("Error", "Invalid input values")

    def clear_visualization(self):
        """Clear the current visualization."""
        self.plot_config["ax"].clear()
        self.plot_config["ax"].set_xlim(-12, 12)
        self.plot_config["ax"].set_ylim(-12, 12)
        self.plot_config["ax"].grid(True)
        self.plot_config["canvas"].draw()
```

```python
    def draw_visualization(self):
        """Draw the points and line segment on the plot."""
        self.plot_config["ax"].clear()

        left_points = []
        right_points = []

        for point in self.points:
            side_value = self.line.side(point)
            if side_value > 0:
                left_points.append(point)
            elif side_value < 0:
                right_points.append(point)

        if left_points:
            x, y = zip(*[(p.x, p.y) for p in left_points])
            self.plot_config["ax"].scatter(x, y, color="blue", label="Left side")

        if right_points:
            x, y = zip(*[(p.x, p.y) for p in right_points])
            self.plot_config["ax"].scatter(x, y, color="red", label="Right side")

        self.plot_config["ax"].plot(
            [self.line.point_a.x, self.line.point_b.x],
            [self.line.point_a.y, self.line.point_b.y],
            "g-",
            label="Line",
        )

        self.plot_config["ax"].set_xlim(-12, 12)
        self.plot_config["ax"].set_ylim(-12, 12)
        self.plot_config["ax"].legend()
        self.plot_config["ax"].grid(True)
        self.plot_config["canvas"].draw()

    def take_screenshot(self):
        """Save the current plot as a screenshot."""
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        filename = f"screenshot_{timestamp}.png"
        self.plot_config["fig"].savefig(filename)
        messagebox.showinfo("Success", f"Screenshot saved as {filename}")


def main():
    """Run the visualization application."""
    root = tk.Tk()
    VisualizationApp(root)
    root.geometry("800x600")
    root.mainloop()


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

Рисунки с результатами работы программы:



Задание 2. Реализовать построение заданного типа фрактала по варианту

Везде, где это необходимо, предусмотреть ввод параметров, влияющих на внешний вид фрактала(Остров Минковского).

Выполнение:

Код программы:

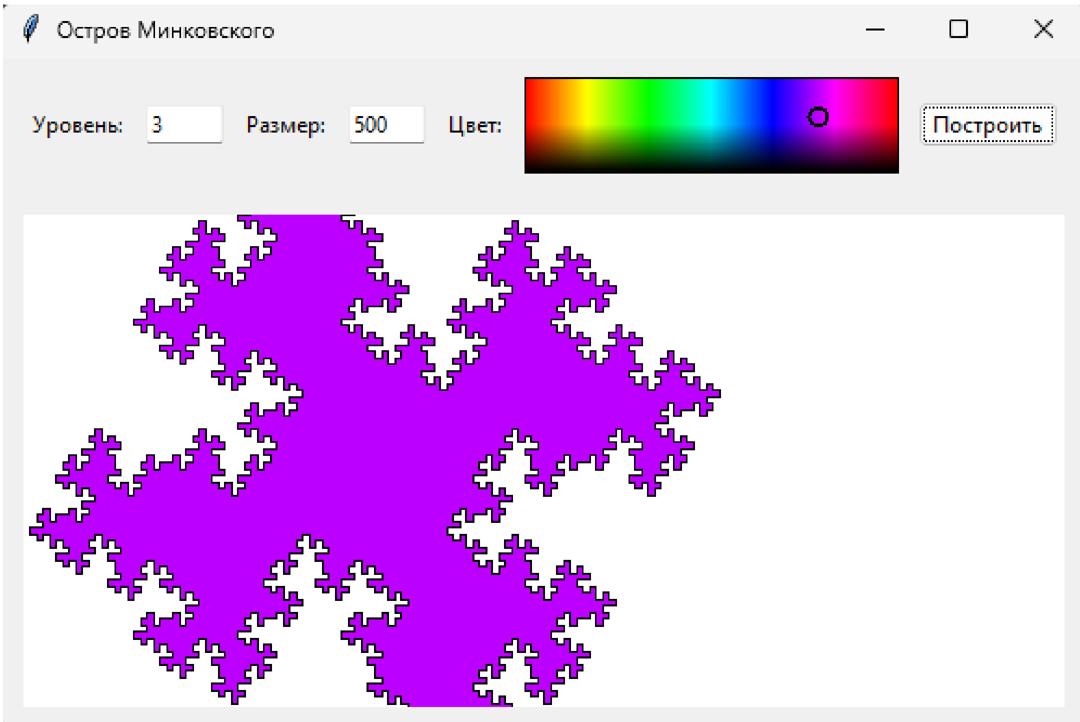
```
"""Module for generating and visualizing Minkowski Island fractals using Tkinter and Turtle."""

import colorsys
import math
import tkinter as tk
import turtle
from tkinter import ttk


class Point:
    """Represents a 2D point for Minkowski Island fractals."""

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def to_tuple(self):
        """Return point as (x, y) tuple for Turtle."""
        return (self.x, self.y)

    def scale(self, factor):
        """Return a scaled Point."""
        return Point(self.x * factor, self.y * factor)
```

```
class MinkowskiIslandApp:
    """Application for rendering Minkowski Island fractals."""

    def __init__(self, root):  # pylint: disable=redefined-outer-name
        self.root = root
        self.root.title("Minkowski Island")
        self.entries = {}
        self.turtle = None
        self.turtle_screen = None
        self.selected_color = tk.StringVar(value="gray")
        self.color_indicator = None
        self.setup_gui()
        self.root.bind("<Configure>", self.on_resize)

    def setup_gui(self):
        """Set up GUI elements."""
        control_frame = ttk.Frame(self.root)
        control_frame.pack(side=tk.TOP, fill=tk.X, padx=10, pady=10)

        for label, default in [("Level:", "3"), ("Size:", "500")]:
            ttk.Label(control_frame, text=label).pack(side=tk.LEFT, padx=5)
            entry = ttk.Entry(control_frame, width=6)
            entry.insert(0, default)
            entry.pack(side=tk.LEFT, padx=5)
            self.entries[label] = entry

        ttk.Label(control_frame, text="Color:").pack(side=tk.LEFT, padx=5)
        color_canvas = tk.Canvas(control_frame, width=100, height=20, highlightthickness=1)
        color_canvas.pack(side=tk.LEFT, padx=5)
        self.draw_color_bar(color_canvas)
        color_canvas.bind("<Button-1>", self.pick_color)

        build_button = ttk.Button(control_frame, text="Build", command=self.generate_fractal)
        build_button.pack(side=tk.LEFT, padx=5)

        plot_frame = ttk.Frame(self.root)
        plot_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)
        canvas = tk.Canvas(plot_frame, bg="light gray")
        canvas.pack(fill=tk.BOTH, expand=True)
        self.turtle_screen = turtle.TurtleScreen(canvas)
        self.turtle_screen.tracer(0, 0)
        self.turtle = turtle.RawTurtle(self.turtle_screen)
        self.turtle.speed(0)
        self.turtle.hideturtle()

        self.generate_fractal()

    def draw_color_bar(self, canvas):
        """Draw a simplified color bar with a selection indicator."""
        width, height = 100, 20
        for x in range(width):
            hue = x / width
            rgb = colorsys.hsv_to_rgb(hue, 1.0, 1.0)
```

```
            color = f"#{int(rgb[0] * 255):02x}{int(rgb[1] * 255):02x}{int(rgb[2] * 255):02x}"
            canvas.create_rectangle(x, 0, x + 1, height, fill=color, outline=color)

        self.color_indicator = canvas.create_oval(5, 5, 15, 15, outline="black", width=2, fill="")

    def pick_color(self, event):
        """Select color from the bar and update indicator position."""
        x = event.x
        width = 100
        if 0 <= x < width:
            hue = x / width
            rgb = colorsys.hsv_to_rgb(hue, 1.0, 1.0)
            color = f"#{int(rgb[0] * 255):02x}{int(rgb[1] * 255):02x}{int(rgb[2] * 255):02x}"
            self.selected_color.set(color)
            canvas = event.widget
            canvas.coords(self.color_indicator, x - 5, 5, x + 5, 15)

    def compute_minkowski_points(self, start, end):
        """Compute points for a Minkowski curve segment."""
        dx, dy = end.x - start.x, end.y - start.y
        dist = math.sqrt(dx**2 + dy**2)
        direction = math.atan2(dy, dx)
        cos_dir, sin_dir = math.cos(direction), math.sin(direction)
        cos_90, sin_90 = math.cos(direction + math.pi / 2), math.sin(direction + math.pi / 2)
        cos_neg90, sin_neg90 = math.cos(direction - math.pi / 2), math.sin(direction - math.pi / 2)

        points = []
        points.append(Point(start.x + dist / 4 * cos_dir, start.y + dist / 4 * sin_dir))
        points.append(Point(points[0].x + dist / 4 * cos_90, points[0].y + dist / 4 * sin_90))
        points.append(Point(points[1].x + dist / 4 * cos_dir, points[1].y + dist / 4 * sin_dir))
        points.append(Point(points[2].x + dist / 4 * cos_neg90, points[2].y + dist / 4 * sin_neg90))
        points.append(Point(points[3].x + dist / 4 * cos_neg90, points[3].y + dist / 4 * sin_neg90))
        points.append(Point(points[4].x + dist / 4 * cos_dir, points[4].y + dist / 4 * sin_dir))
        points.append(Point(start.x + 3 * dist / 4 * cos_dir, start.y + 3 * dist / 4 * sin_dir))

        return points

    def minkowski_curve(self, start, end, n):
        """Draw Minkowski curve recursively."""
        if n == 0:
            self.turtle.goto(*end.to_tuple())
            return
        points = self.compute_minkowski_points(start, end)
        self.minkowski_curve(start, points[0], n - 1)
        for i in range(len(points) - 1):
            self.minkowski_curve(points[i], points[i + 1], n - 1)
        self.minkowski_curve(points[-1], end, n - 1)

    def draw_minkowski_island(self, size, n, fill_color):
        """Draw the Minkowski Island."""
        self.turtle.clear()
        self.turtle.up()
        self.turtle.goto(-size / 2, -size / 2)
```

```
        self.turtle.down()
        self.turtle.fillcolor(fill_color)
        self.turtle.begin_fill()

        corners = [
            Point(-size / 2, -size / 2),
            Point(-size / 2, size / 2),
            Point(size / 2, size / 2),
            Point(size / 2, -size / 2),
        ]
        for i in range(4):
            self.minkowski_curve(corners[i], corners[(i + 1) % 4], n)

        self.turtle.end_fill()
        self.turtle_screen.update()

    def generate_fractal(self):
        """Generate fractal based on user input."""
        try:
            n = int(self.entries["Level:"].get())
            size = float(self.entries["Size:"].get())
            if n < 0 or size <= 0:
                print("Level must be non-negative and size must be positive.")
                return
        except ValueError:
            print("Invalid input parameters.")
            return

        canvas_size = min(self.turtle_screen.canvwidth, self.turtle_screen.canvheight)
        scale = canvas_size / 600
        self.draw_minkowski_island(size * scale, n, self.selected_color.get())

    def on_resize(self, event):
        """Redraw fractal on window resize."""
        if event.widget == self.root and self.turtle_screen.canvwidth > 1:
            self.generate_fractal()


if __name__ == "__main__":
    root = tk.Tk()
    app = MinkowskiIslandApp(root)
    root.mainloop()
```

Рисунки с результатами работы программы:



Вывод: закрепил базовые знания Python, а также научились создавать простые оконные приложения.