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
Н.А. Антонюк
студент группы ПО11

Проверил
А. А. Крощенко
ст. преп. кафедры ИИТ,
02.03.2025 г.

Брест 2025

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

Задание 1. Построение графических примитивов и надписей: Определить класс Rectangle и класс Point. Объявить список из n объектов класса Point. Написать функцию, определяющую, какая из точек лежит снаружи, а какая – внутри прямоугольника.

Код программы:

main.py

```
import sys
import random
from PyQt5.QtWidgets import (QApplication, QMainWindow, QWidget, QVBoxLayout,
                             QHBoxLayout, QPushButton, QLabel, QSpinBox,
                             QDoubleSpinBox, QGroupBox)
from PyQt5.QtCore import Qt, QTimer
from PyQt5.QtGui import QPainter, QColor, QPen
from geometry import Point, Rectangle

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Точки и прямоугольник")
        self.setGeometry(100, 100, 800, 600)

        # Основной виджет и компоновка
        main_widget = QWidget()
        self.setCentralWidget(main_widget)
        layout = QHBoxLayout(main_widget)

        # Панель управления
        control_panel = QGroupBox("Управление")
        control_layout = QVBoxLayout()

        # Параметры прямоугольника
        rect_group = QGroupBox("Прямоугольник")
        rect_layout = QVBoxLayout()

        self.rect_x = QDoubleSpinBox()
        self.rect_x.setRange(-100, 100)
        self.rect_x.setValue(0)
        rect_layout.addWidget(QLabel("X:"))
        rect_layout.addWidget(self.rect_x)

        self.rect_y = QDoubleSpinBox()
        self.rect_y.setRange(-100, 100)
        self.rect_y.setValue(0)
        rect_layout.addWidget(QLabel("Y:"))
        rect_layout.addWidget(self.rect_y)

        self.rect_width = QDoubleSpinBox()
        self.rect_width.setRange(1, 200)
        self.rect_width.setValue(100)
        rect_layout.addWidget(QLabel("Ширина:"))
        rect_layout.addWidget(self.rect_width)

        self.rect_height = QDoubleSpinBox()
```

```
        self.rect_height.setRange(1, 200)
        self.rect_height.setValue(100)
        rect_layout.addWidget(QLabel("Высота:"))
        rect_layout.addWidget(self.rect_height)

        rect_group.setLayout(rect_layout)
        control_layout.addWidget(rect_group)

        # Количество точек
        self.points_count = QSpinBox()
        self.points_count.setRange(1, 100)
        self.points_count.setValue(10)
        control_layout.addWidget(QLabel("Количество точек:"))
        control_layout.addWidget(self.points_count)

        # Кнопки управления
        self.generate_btn = QPushButton("Сгенерировать точки")
        self.generate_btn.clicked.connect(self.generate_points)
        control_layout.addWidget(self.generate_btn)

        self.screenshot_btn = QPushButton("Сделать скриншот")
        self.screenshot_btn.clicked.connect(self.take_screenshot)
        control_layout.addWidget(self.screenshot_btn)

        control_panel.setLayout(control_layout)
        layout.addWidget(control_panel)

        # Область рисования
        self.canvas = Canvas()
        layout.addWidget(self.canvas)

        # Инициализация данных
        self.rectangle = Rectangle(0, 0, 100, 100)
        self.points = []
        self.generate_points()

        # Таймер для обновления
        self.timer = QTimer()
        self.timer.timeout.connect(self.update)
        self.timer.start(16)  # ~60 FPS

    def generate_points(self):
        self.points = []
        for _ in range(self.points_count.value()):
            x = random.uniform(-200, 200)
            y = random.uniform(-200, 200)
            self.points.append(Point(x, y))
        self.canvas.update()

    def take_screenshot(self):
        screenshot = self.canvas.grab()
        screenshot.save("screenshot.png")

    def update(self):
        self.rectangle = Rectangle(
            self.rect_x.value(),
            self.rect_y.value(),
            self.rect_width.value(),
            self.rect_height.value()
```

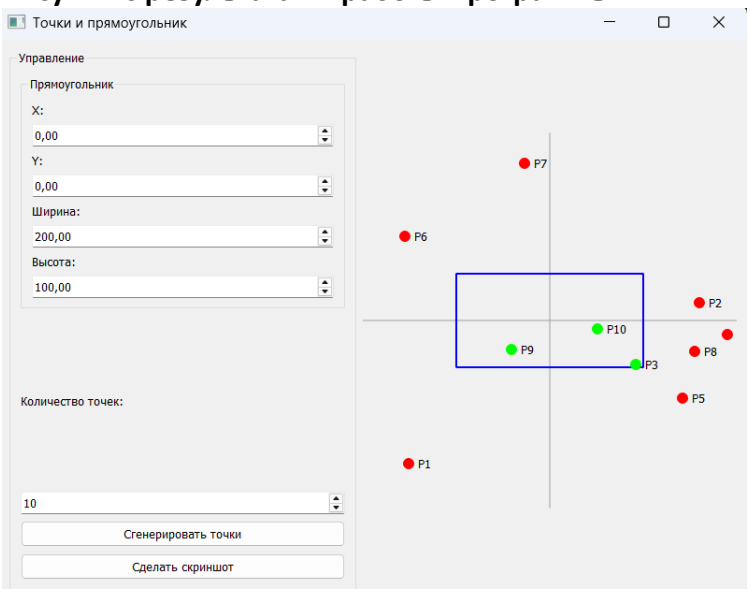
```
        )
        self.canvas.set_data(self.rectangle, self.points)
        self.canvas.update()

class Canvas(QWidget):
    def __init__(self):
        super().__init__()
        self.rectangle = None
        self.points = []
        self.setMinimumSize(400, 400)

    def set_data(self, rectangle, points):
        self.rectangle = rectangle
        self.points = points

    def paintEvent(self, event):
        if not self.rectangle or not self.points:
            return

        painter = QPainter(self)
        painter.setRenderHint(QPainter.Antialiasing)

        # Центрирование и масштабирование
        width = self.width()
        height = self.height()
        scale = min(width, height) / 400
        painter.translate(width/2, height/2)
        painter.scale(scale, scale)

        # Оси координат
        painter.setPen(QPen(Qt.gray, 1))
        painter.drawLine(-200, 0, 200, 0)
        painter.drawLine(0, -200, 0, 200)

        # Прямоугольник
        painter.setPen(QPen(Qt.blue, 2))
        painter.drawRect(
            int(self.rectangle.x - self.rectangle.width/2),
            int(self.rectangle.y - self.rectangle.height/2),
            int(self.rectangle.width),
            int(self.rectangle.height)
        )

        # Точки
        for i, point in enumerate(self.points):
            if point.is_inside_rectangle(self.rectangle):
                painter.setPen(QPen(Qt.green, 2))
                painter.setBrush(Qt.green)
            else:
                painter.setPen(QPen(Qt.red, 2))
                painter.setBrush(Qt.red)

            # Рисуем точку как круг
            painter.drawEllipse(int(point.x) - 5, int(point.y) - 5, 10, 10)

            # Добавляем подпись с номером точки
            painter.setPen(QPen(Qt.black, 1))
            painter.drawText(int(point.x) + 10, int(point.y) + 5, f"P{i+1}")
```

```
if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())
```

geometry.py

```
from dataclasses import dataclass
from typing import List

@dataclass
class Point:
    x: float
    y: float

    def is_inside_rectangle(self, rect: 'Rectangle') -> bool:
        # Проверяем, находится ли точка внутри прямоугольника, учитывая его центр
        left = rect.x - rect.width/2
        right = rect.x + rect.width/2
        top = rect.y - rect.height/2
        bottom = rect.y + rect.height/2

        return (left <= self.x <= right and
                top <= self.y <= bottom)

@dataclass
class Rectangle:
    x: float
    y: float
    width: float
    height: float

    def contains_point(self, point: Point) -> bool:
        return point.is_inside_rectangle(self)
```

Рисунки с результатами работы программы:



Задание 2. Реализовать построение заданного типа фрактала по варианту

2) H-фрактал

Код программы:

main.py

```
import sys
from PyQt5.QtWidgets import (QApplication, QMainWindow, QWidget, QVBoxLayout,
                             QHBoxLayout, QPushButton, QLabel, QSpinBox,
                             QDoubleSpinBox, QGroupBox, QColorDialog)
from PyQt5.QtCore import Qt
from PyQt5.QtGui import QPainter, QPen, QColor
from fractal import Point, HFractal

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("H-фрактал")
        self.setGeometry(100, 100, 800, 600)

        # Основной виджет и компоновка
        main_widget = QWidget()
        self.setCentralWidget(main_widget)
        layout = QHBoxLayout(main_widget)

        # Панель управления
        control_panel = QGroupBox("Управление")
        control_layout = QVBoxLayout()

        # Параметры фрактала
        self.size = QDoubleSpinBox()
        self.size.setRange(10, 1000)
        self.size.setValue(400)
        self.size.valueChanged.connect(self.update_fractal)
        control_layout.addWidget(QLabel("Размер:"))
        control_layout.addWidget(self.size)

        self.depth = QSpinBox()
        self.depth.setRange(1, 10)
        self.depth.setValue(4)
        self.depth.valueChanged.connect(self.update_fractal)
        control_layout.addWidget(QLabel("Глубина:"))
        control_layout.addWidget(self.depth)

        # Толщина линий
        self.line_width = QSpinBox()
        self.line_width.setRange(1, 10)
        self.line_width.setValue(1)
        self.line_width.valueChanged.connect(self.update_fractal)
        control_layout.addWidget(QLabel("Толщина линий:"))
        control_layout.addWidget(self.line_width)

        # Цвет линий
        self.line_color = QColor(Qt.black)
        self.color_btn = QPushButton("Выбрать цвет")
        self.color_btn.clicked.connect(self.choose_color)
        control_layout.addWidget(QLabel("Цвет линий:"))
        control_layout.addWidget(self.color_btn)

        # Кнопка скриншота
        self.screenshot_btn = QPushButton("Сделать скриншот")
        self.screenshot_btn.clicked.connect(self.take_screenshot)
        control_layout.addWidget(self.screenshot_btn)
```

```
        control_panel.setLayout(control_layout)
        layout.addWidget(control_panel)

        # Область рисования
        self.canvas = Canvas()
        layout.addWidget(self.canvas)

        # Инициализация фрактала
        self.update_fractal()

    def choose_color(self):
        color = QColorDialog.getColor(self.line_color, self, "Выберите цвет линий")
        if color.isValid():
            self.line_color = color
            self.update_fractal()

    def update_fractal(self):
        center = Point(0, 0)
        self.fractal = HFractal(center, self.size.value(), self.depth.value())
        self.canvas.set_fractal(self.fractal, self.line_color, self.line_width.value())
        self.canvas.update()

    def take_screenshot(self):
        screenshot = self.canvas.grab()
        screenshot.save("fractal_screenshot.png")

class Canvas(QWidget):
    def __init__(self):
        super().__init__()
        self.fractal = None
        self.line_color = Qt.black
        self.line_width = 1
        self.setMinimumSize(400, 400)

    def set_fractal(self, fractal, color, width):
        self.fractal = fractal
        self.line_color = color
        self.line_width = width

    def paintEvent(self, event):
        if not self.fractal:
            return

        painter = QPainter(self)
        painter.setRenderHint(QPainter.Antialiasing)

        # Центрирование и масштабирование
        width = self.width()
        height = self.height()
        scale = min(width, height) / (self.fractal.size * 1.2)
        painter.translate(width/2, height/2)
        painter.scale(scale, scale)

        # Рисование линий фрактала
        painter.setPen(QPen(self.line_color, self.line_width))
        for line in self.fractal.lines:
            painter.drawLine(
                int(line[0].x), int(line[0].y),
                int(line[1].x), int(line[1].y)
            )
```

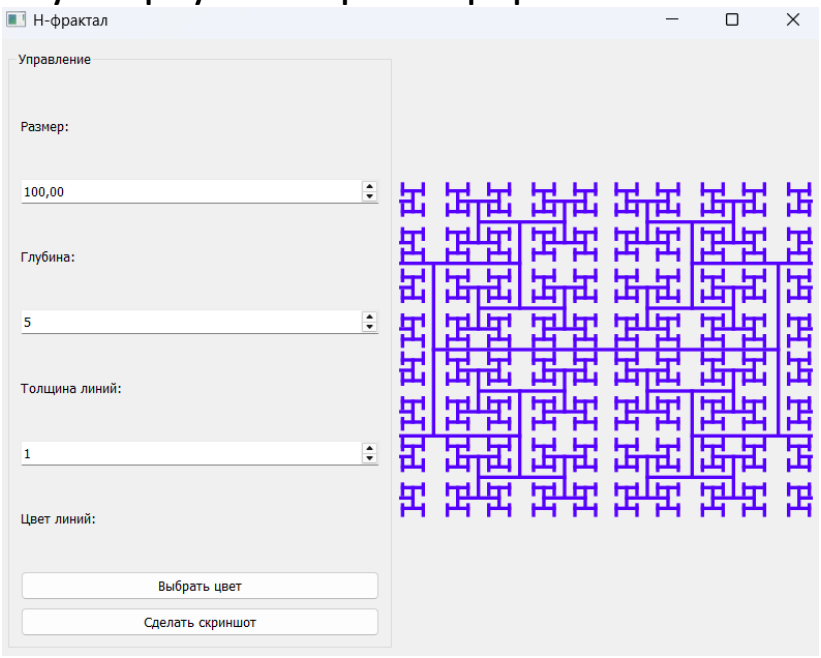
```
if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())
```

fractal.py

```
from dataclasses import dataclass
from typing import List, Tuple
import math

@dataclass
class Point:
    x: float
    y: float

class HFractal:
    def __init__(self, start_point: Point, size: float, depth: int):
        self.start_point = start_point
        self.size = size
        self.depth = depth
        self.lines = []
        self._generate_fractal(start_point, size, depth)

    def _generate_fractal(self, center: Point, size: float, depth: int):
        if depth == 0:
            return

        # Размеры для текущей итерации
        half_size = size / 2
        quarter_size = size / 4

        # Горизонтальные линии
        self.lines.append((
            Point(center.x - half_size, center.y),
            Point(center.x + half_size, center.y)
        ))

        # Вертикальные линии
        self.lines.append((
            Point(center.x - half_size, center.y - quarter_size),
            Point(center.x - half_size, center.y + quarter_size)
        ))
        self.lines.append((
            Point(center.x + half_size, center.y - quarter_size),
            Point(center.x + half_size, center.y + quarter_size)
        ))

        # Рекурсивный вызов для четырех новых центров
        new_size = size / 2
        new_depth = depth - 1

        # Левый верхний
        self._generate_fractal(
            Point(center.x - half_size, center.y - quarter_size),
            new_size,
            new_depth
        )
```

```
        # Правый верхний
        self._generate_fractal(
            Point(center.x + half_size, center.y - quarter_size),
            new_size,
            new_depth
        )

        # Левый нижний
        self._generate_fractal(
            Point(center.x - half_size, center.y + quarter_size),
            new_size,
            new_depth
        )

        # Правый нижний
        self._generate_fractal(
            Point(center.x + half_size, center.y + quarter_size),
            new_size,
            new_depth
        )
```

Рисунки с результатами работы программы:



Вывод: освоил возможности языка программирования Python в разработке оконных приложений.