МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе No7

Специальность ПО11(о)

Выполнил
И. А. Головач,
студент группы ПО11

Проверил
А. А. Крощенко,
ст. преп. кафедры ИИТ,
«10» май 2025 г.

Брест 2025

Вариант 5

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

Задание 1. Построение графических примитивов и надписей. Изобразить в окне приложения отрезок, вращающийся в плоскости экрана вокруг одной из своих концевых точек. Цвет прямой должен изменяться при переходе от одного положения к другому.

Код программы:

segment.py:

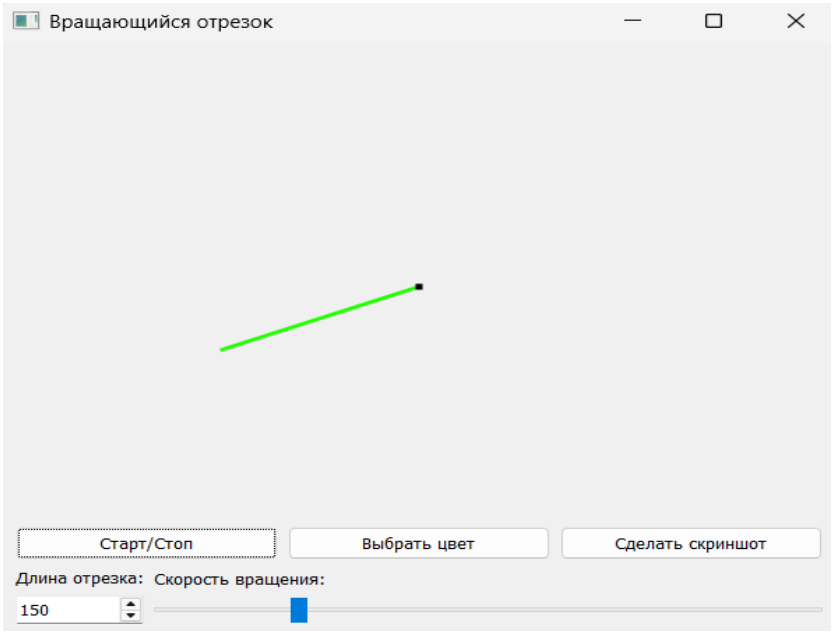
```
import sys
import math
from PyQt5.QtWidgets import (QApplication, QMainWindow, QWidget, QVBoxLayout,
                             QHBoxLayout, QLabel, QSlider, QPushButton, QSpinBox,
                             QColorDialog, QFileDialog)
from PyQt5.QtCore import Qt, QTimer
from PyQt5.QtGui import QPainter, QColor, QPen, QImage


class RotatingLineWidget(QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.angle = 0
        self.line_length = 150
        self.rotation_speed = 1
        self.base_point = None
        self.color = QColor(255, 0, 0)
        self.is_rotating = False
        self.timer = QTimer(self)
        self.timer.timeout.connect(self.update_rotation)

    def set_line_length(self, length):
        self.line_length = length
        self.update()

    def set_rotation_speed(self, speed):
        self.rotation_speed = speed

    def set_color(self, color):
        self.color = color
        self.update()

    def start_rotation(self):
        self.is_rotating = True
        self.timer.start(20)

    def stop_rotation(self):
        self.is_rotating = False
        self.timer.stop()

    def toggle_rotation(self):
        if self.is_rotating:
```

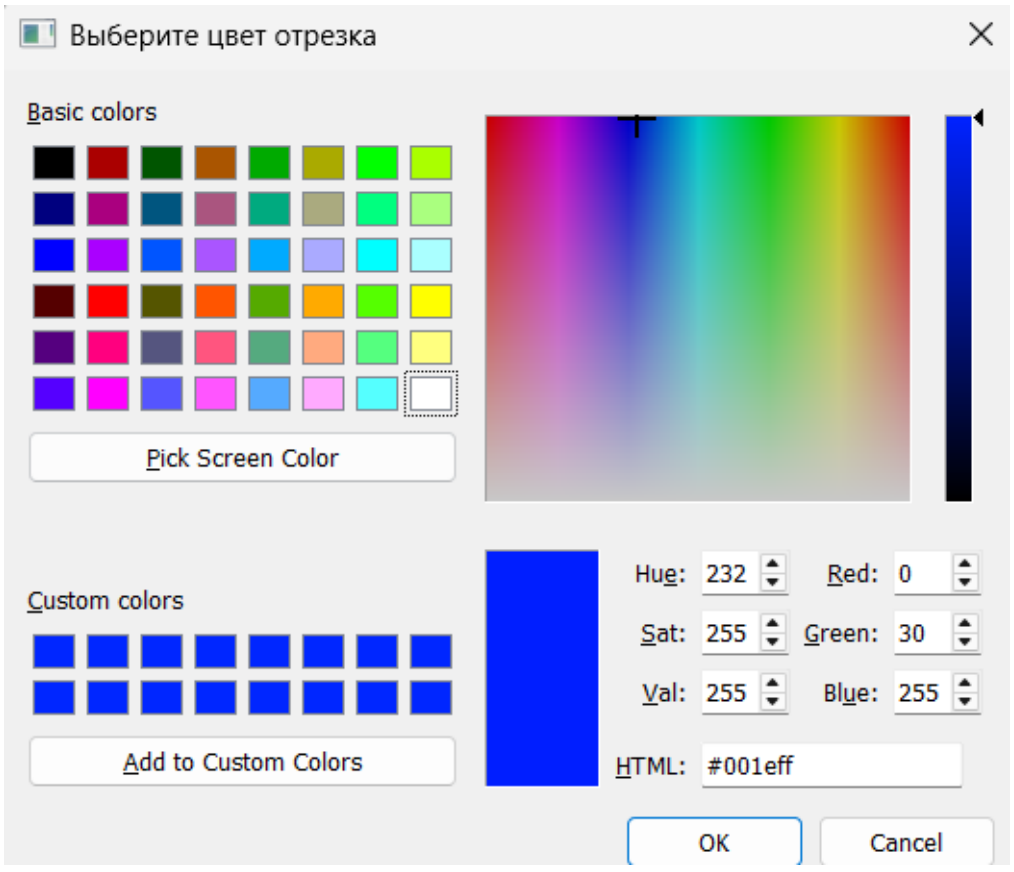
```
            self.stop_rotation()
        else:
            self.start_rotation()

    def update_rotation(self):
        self.angle = (self.angle + self.rotation_speed) % 360
        hue = int((self.angle / 360) * 255)
        self.color.setHsv(hue, 255, 255)
        self.update()

    def paintEvent(self, _):
        painter = QPainter(self)
        painter.setRenderHint(QPainter.Antialiasing)

        width = self.width()
        height = self.height()
        self.base_point = (width // 2, height // 2)

        end_x = self.base_point[0] + self.line_length *
math.cos(math.radians(self.angle))
        end_y = self.base_point[1] + self.line_length *
math.sin(math.radians(self.angle))

        pen = QPen(self.color, 3)
        painter.setPen(pen)
        painter.drawLine(self.base_point[0], self.base_point[1], int(end_x),
int(end_y))

        painter.setPen(QPen(Qt.black, 5))
        painter.drawPoint(self.base_point[0], self.base_point[1])


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Вращающийся отрезок")
        self.setGeometry(100, 100, 600, 500)

        # Initialize UI components
        self.start_button = None
        self.color_button = None
        self.screenshot_button = None
        self.length_spin = None
        self.speed_slider = None
        self.line_widget = None

        self.init_ui()

    def init_ui(self):
        central_widget = QWidget()
        self.setCentralWidget(central_widget)
        main_layout = QVBoxLayout(central_widget)

        self.line_widget = RotatingLineWidget()
        main_layout.addWidget(self.line_widget, 1)

        self.create_control_panel(main_layout)
        self.create_parameters_panel(main_layout)
```

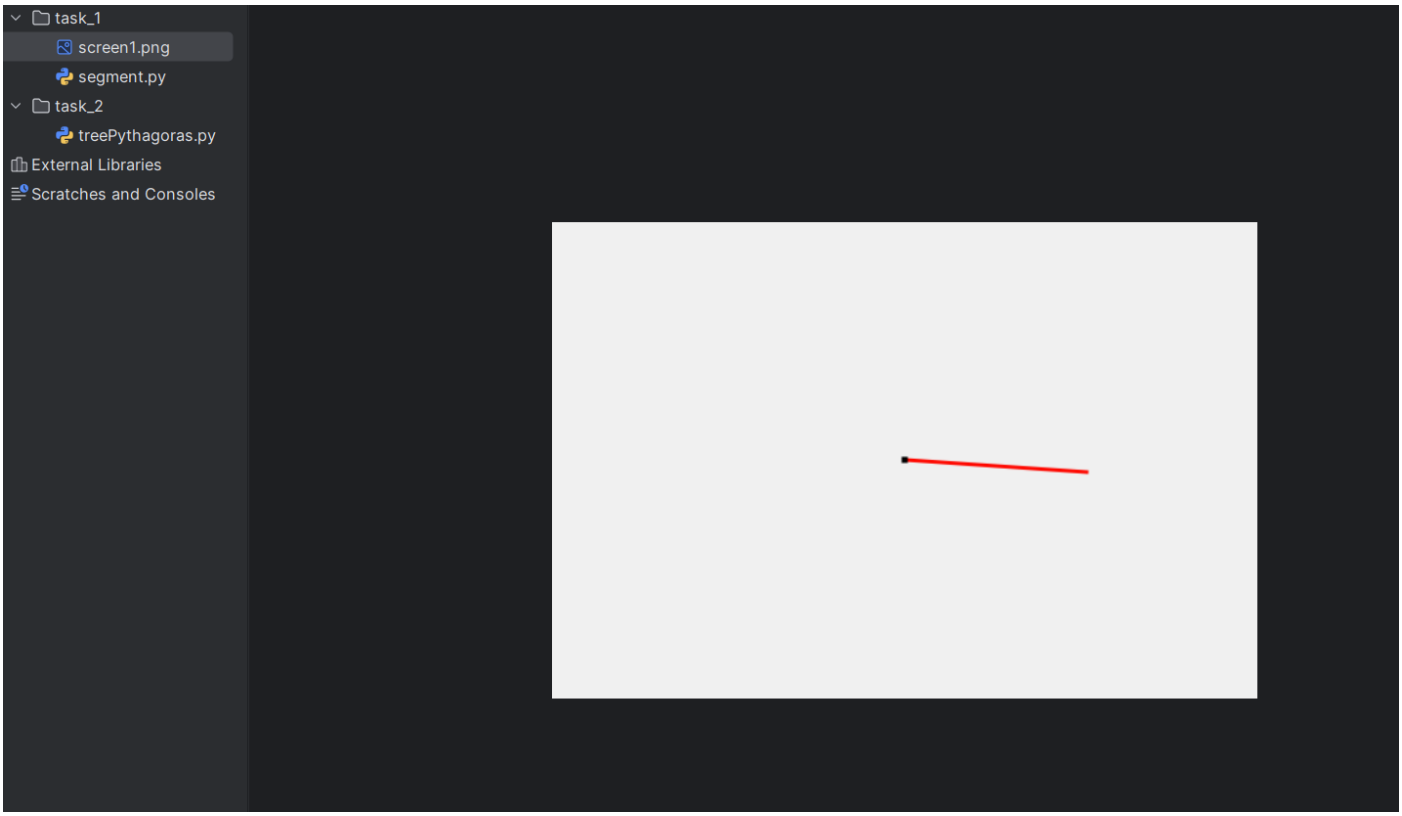
```
    def create_control_panel(self, main_layout):
        control_layout = QHBoxLayout()

        self.start_button = QPushButton("Старт/Стоп")
        self.start_button.clicked.connect(self.line_widget.toggle_rotation)
        control_layout.addWidget(self.start_button)

        self.color_button = QPushButton("Выбрать цвет")
        self.color_button.clicked.connect(self.choose_color)
        control_layout.addWidget(self.color_button)

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

        main_layout.addLayout(control_layout)

    def create_parameters_panel(self, main_layout):
        params_layout = QHBoxLayout()

        self.create_length_control(params_layout)
        self.create_speed_control(params_layout)

        main_layout.addLayout(params_layout)

    def create_length_control(self, layout):
        length_layout = QVBoxLayout()
        length_layout.addWidget(QLabel("Длина отрезка:"))
        self.length_spin = QSpinBox()
        self.length_spin.setRange(50, 300)
        self.length_spin.setValue(150)
        self.length_spin.valueChanged.connect(self.line_widget.set_line_length)
        length_layout.addWidget(self.length_spin)
        layout.addLayout(length_layout)

    def create_speed_control(self, layout):
        speed_layout = QVBoxLayout()
        speed_layout.addWidget(QLabel("Скорость вращения:"))
        self.speed_slider = QSlider(Qt.Horizontal)
        self.speed_slider.setRange(1, 20)
        self.speed_slider.setValue(5)
        self.speed_slider.valueChanged.connect(self.line_widget.set_rotation_speed)
        speed_layout.addWidget(self.speed_slider)
        layout.addLayout(speed_layout)

    def choose_color(self):
        color = QColorDialog.getColor(self.line_widget.color, self, "Выберите цвет
отрезка")
        if color.isValid():
            self.line_widget.set_color(color)

    def take_screenshot(self):
        image = QImage(self.line_widget.size(), QImage.Format_ARGB32)
        painter = QPainter(image)
        self.line_widget.render(painter)
        painter.end()

        file_name, _ = QFileDialog.getSaveFileName(self, "Сохранить скриншот", "",
                                                   "PNG Images (*.png);;JPEG Images
```

```
(*.jpg *.jpeg)")
        if file_name:
            image.save(file_name)


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

Результаты работы программы:







Задание 2. Реализовать построение заданного типа фрактала по варианту: Дерево Пифагора

Код программы:

treePythagoras.py:

```
import sys
import math
from dataclasses import dataclass
from PyQt5.QtWidgets import (QApplication, QMainWindow, QWidget, QVBoxLayout,
                             QHBoxLayout, QLabel, QPushButton, QSpinBox,
                             QColorDialog, QDoubleSpinBox)
from PyQt5.QtGui import QPainter, QColor, QPen


@dataclass
class Point:
    x: int
    y: int


@dataclass
class Line:
    start: Point
    end: Point


@dataclass
class TreeParams:
```

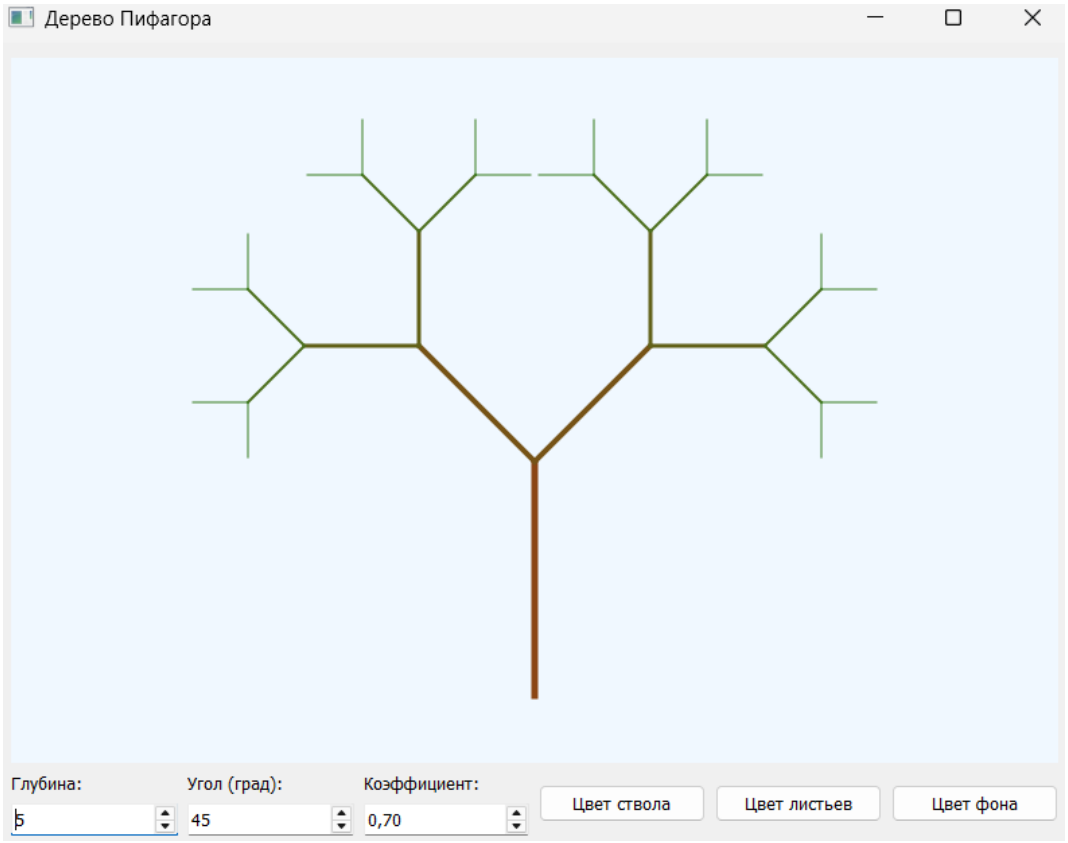
```
    length: float
    angle: float
    depth: int


@dataclass
class DrawingContext:
    painter: QPainter
    color: QColor
    depth: int


class PythagorasTreeWidget(QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.depth = 5
        self.angle = math.pi / 4  # 45 degrees
        self.ratio = 0.7
        self.color1 = QColor(139, 69, 19)  # Brown
        self.color2 = QColor(34, 139, 34)  # Green
        self.bg_color = QColor(240, 248, 255)  # AliceBlue

    def set_depth(self, depth):
        self.depth = depth
        self.update()

    def set_angle(self, angle):
        self.angle = math.radians(angle)
        self.update()

    def set_ratio(self, ratio):
        self.ratio = ratio
        self.update()

    def set_color1(self, color):
        self.color1 = color
        self.update()

    def set_color2(self, color):
        self.color2 = color
        self.update()

    def set_bg_color(self, color):
        self.bg_color = color
        self.update()

    def paintEvent(self, _):
        painter = QPainter(self)
        painter.setRenderHint(QPainter.Antialiasing)
        painter.fillRect(self.rect(), self.bg_color)

        width = self.width()
        height = self.height()
        start_point = Point(width // 2, height - 50)
        length = height // 3

        params = TreeParams(length=length, angle=-math.pi / 2, depth=self.depth)
        self.draw_tree(painter=painter, start_point=start_point, params=params)
```

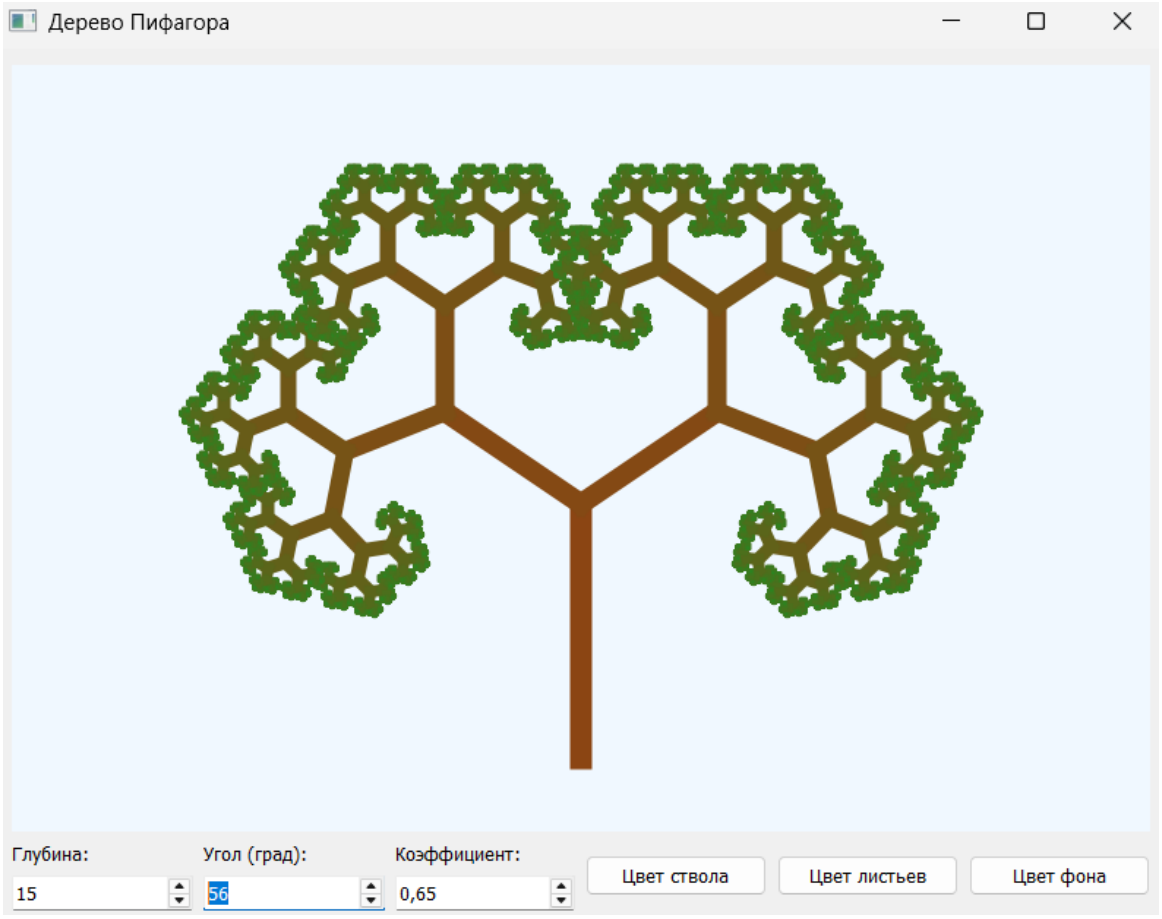
```
    def draw_tree(self, painter, start_point, params):
        if params.depth == 0:
            return

        end_point = Point(
            start_point.x + int(math.cos(params.angle) * params.length),
            start_point.y + int(math.sin(params.angle) * params.length)
        )

        color = self.get_color_for_depth(depth=params.depth)
        line = Line(start=start_point, end=end_point)
        context = DrawingContext(painter=painter, color=color, depth=params.depth)

        self.draw_branch(line=line, context=context)

        if params.depth > 1:
            new_length = params.length * self.ratio

            # Left branch
            left_params = TreeParams(
                length=new_length,
                angle=params.angle - self.angle,
                depth=params.depth - 1
            )
            self.draw_tree(painter=painter, start_point=end_point, params=left_params)

            # Right branch
            right_params = TreeParams(
                length=new_length,
                angle=params.angle + self.angle,
                depth=params.depth - 1
            )
            self.draw_tree(painter=painter, start_point=end_point, params=right_params)

    def get_color_for_depth(self, depth):
        if depth == self.depth:
            return self.color1

        t = (self.depth - depth) / self.depth
        r = int(self.color1.red() * (1 - t) + self.color2.red() * t)
        g = int(self.color1.green() * (1 - t) + self.color2.green() * t)
        b = int(self.color1.blue() * (1 - t) + self.color2.blue() * t)
        return QColor(r, g, b)

    @staticmethod
    def draw_branch(line: Line, context: DrawingContext):
        pen = QPen(context.color)
        pen.setWidth(max(1, context.depth))
        context.painter.setPen(pen)
        context.painter.drawLine(line.start.x, line.start.y, line.end.x, line.end.y)


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Дерево Пифагора")
        self.setGeometry(100, 100, 800, 600)

        # Initialize UI components
```

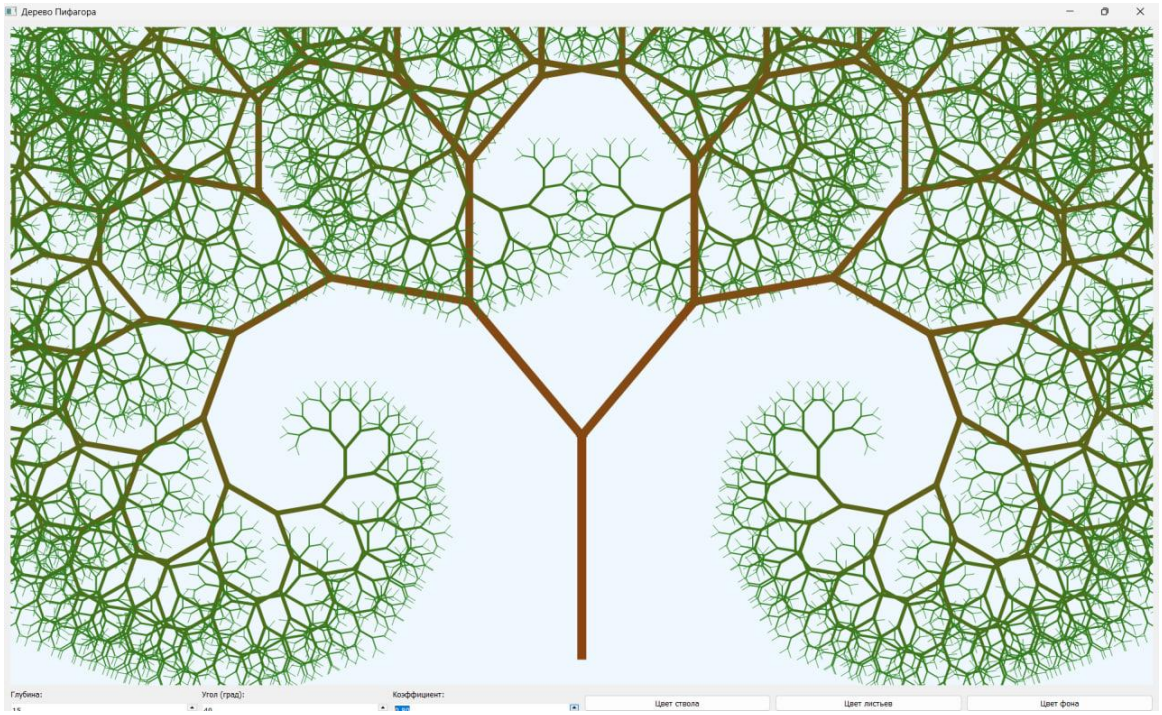
```
        self.tree_widget = None
        self.depth_spin = None
        self.angle_spin = None
        self.ratio_spin = None
        self.color1_button = None
        self.color2_button = None
        self.bg_color_button = None

        self.init_ui()

    def init_ui(self):
        central_widget = QWidget()
        self.setCentralWidget(central_widget)
        main_layout = QVBoxLayout(central_widget)

        self.tree_widget = PythagorasTreeWidget()
        main_layout.addWidget(self.tree_widget, 1)
        self.create_control_panel(main_layout)

    def create_control_panel(self, main_layout):
        control_layout = QHBoxLayout()
        self.create_depth_control(control_layout)
        self.create_angle_control(control_layout)
        self.create_ratio_control(control_layout)
        self.create_color_buttons(control_layout)
        main_layout.addLayout(control_layout)

    def create_depth_control(self, layout):
        depth_layout = QVBoxLayout()
        depth_layout.addWidget(QLabel("Глубина:"))
        self.depth_spin = QSpinBox()
        self.depth_spin.setRange(1, 15)
        self.depth_spin.setValue(5)
        self.depth_spin.valueChanged.connect(self.tree_widget.set_depth)
        depth_layout.addWidget(self.depth_spin)
        layout.addLayout(depth_layout)

    def create_angle_control(self, layout):
        angle_layout = QVBoxLayout()
        angle_layout.addWidget(QLabel("Угол (град):"))
        self.angle_spin = QSpinBox()
        self.angle_spin.setRange(0, 90)
        self.angle_spin.setValue(45)
        self.angle_spin.valueChanged.connect(self.tree_widget.set_angle)
        angle_layout.addWidget(self.angle_spin)
        layout.addLayout(angle_layout)

    def create_ratio_control(self, layout):
        ratio_layout = QVBoxLayout()
        ratio_layout.addWidget(QLabel("Коэффициент:"))
        self.ratio_spin = QDoubleSpinBox()
        self.ratio_spin.setRange(0.1, 0.9)
        self.ratio_spin.setSingleStep(0.05)
        self.ratio_spin.setValue(0.7)
        self.ratio_spin.valueChanged.connect(self.tree_widget.set_ratio)
        ratio_layout.addWidget(self.ratio_spin)
        layout.addLayout(ratio_layout)

    def create_color_buttons(self, layout):
```

```
        self.color1_button = QPushButton("Цвет ствола")
        self.color1_button.clicked.connect(lambda:
self.choose_color(self.tree_widget.set_color1))
        layout.addWidget(self.color1_button)

        self.color2_button = QPushButton("Цвет листьев")
        self.color2_button.clicked.connect(lambda:
self.choose_color(self.tree_widget.set_color2))
        layout.addWidget(self.color2_button)

        self.bg_color_button = QPushButton("Цвет фона")
        self.bg_color_button.clicked.connect(lambda:
self.choose_color(self.tree_widget.set_bg_color))
        layout.addWidget(self.bg_color_button)

    @staticmethod
    def choose_color(setter):
        color = QColorDialog.getColor()
        if color.isValid():
            setter(color)


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

Результаты работы программы:







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