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
К. А. Головач,
студент группы ПО11

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

Брест 2025

Вариант 6

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

Задание 1. Построение графических примитивов и надписей.

6) Задать движение окружности по форме так, чтобы при касании границы окружность отражалась от нее.

Код программы:

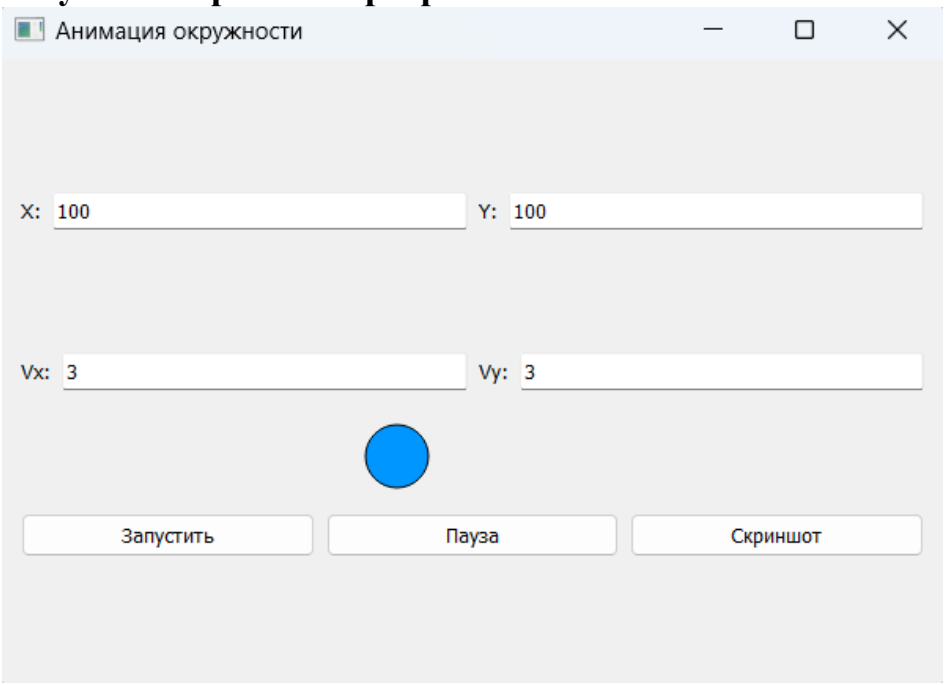
circle_animation.py:

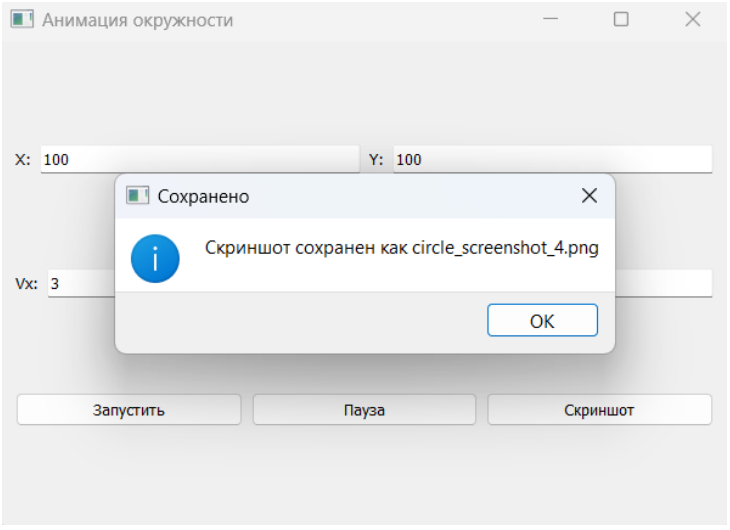
```
import sys
import os
from PyQt5.QtWidgets import (
    QApplication, QWidget, QLabel, QLineEdit, QPushButton,
    QVBoxLayout, QHBoxLayout, QMessageBox
)
from PyQt5.QtGui import QPainter, QColor, QPixmap
from PyQt5.QtCore import QTimer, Qt


class CircleAnimation(QWidget):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("Анимация окружности")
        self.setGeometry(100, 100, 600, 400)

        # Параметры окружности
        self.circle_radius = 20
        self.x = 100
        self.y = 100
        self.vx = 3
        self.vy = 3

        # Скорость таймера
        self.timer_delay = 20  # мс
        self.timer = QTimer()
        self.timer.timeout.connect(self.update_position)

        # Интерфейс
        self.init_ui()

    def init_ui(self):
        layout = QVBoxLayout()

        # Ввод координат
        coord_layout = QHBoxLayout()
        self.x_input = QLineEdit(str(self.x))
        self.y_input = QLineEdit(str(self.y))
        coord_layout.addWidget(QLabel("X:"))
        coord_layout.addWidget(self.x_input)
        coord_layout.addWidget(QLabel("Y:"))
        coord_layout.addWidget(self.y_input)
        layout.addLayout(coord_layout)
```

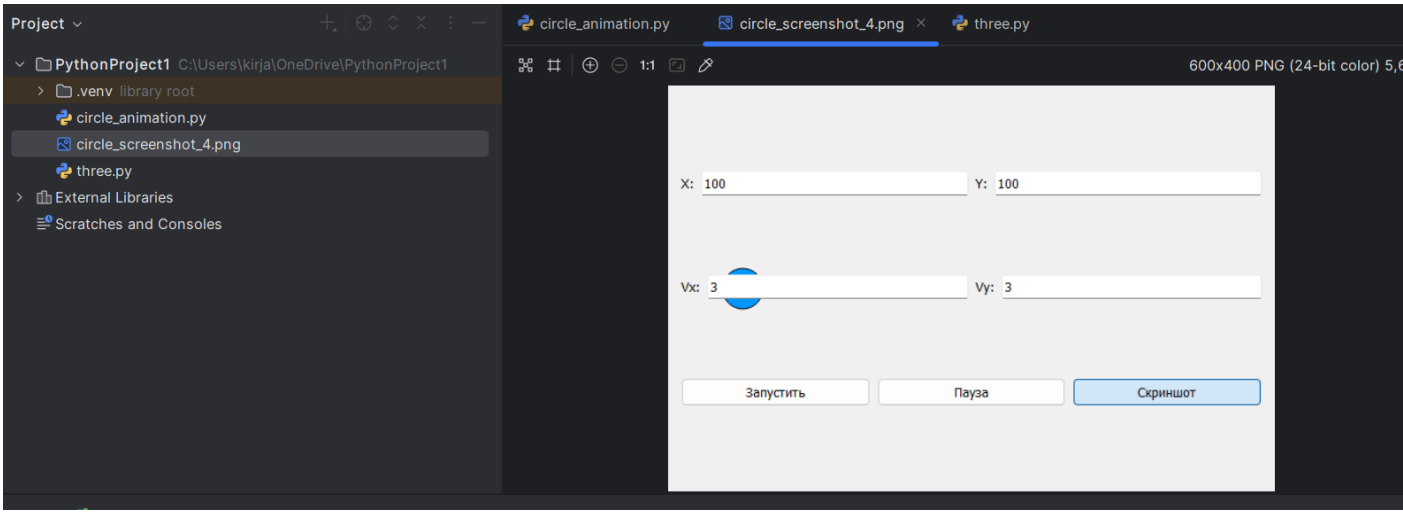
```
        # Ввод скорости
        speed_layout = QHBoxLayout()
        self.vx_input = QLineEdit(str(self.vx))
        self.vy_input = QLineEdit(str(self.vy))
        speed_layout.addWidget(QLabel("Vx:"))
        speed_layout.addWidget(self.vx_input)
        speed_layout.addWidget(QLabel("Vy:"))
        speed_layout.addWidget(self.vy_input)
        layout.addLayout(speed_layout)

        # Кнопки управления
        btn_layout = QHBoxLayout()
        self.start_btn = QPushButton("Запустить")
        self.pause_btn = QPushButton("Пауза")
        self.screenshot_btn = QPushButton("Скриншот")

        self.start_btn.clicked.connect(self.start_animation)
        self.pause_btn.clicked.connect(self.pause_animation)
        self.screenshot_btn.clicked.connect(self.take_screenshot)

        btn_layout.addWidget(self.start_btn)
        btn_layout.addWidget(self.pause_btn)
        btn_layout.addWidget(self.screenshot_btn)
        layout.addLayout(btn_layout)

        self.setLayout(layout)

    def start_animation(self):
        try:
            self.x = int(self.x_input.text())
            self.y = int(self.y_input.text())
            self.vx = int(self.vx_input.text())
            self.vy = int(self.vy_input.text())
        except ValueError:
            QMessageBox.warning(self, "Ошибка", "Введите корректные числовые значения!")
            return

        if not self.timer.isActive():
            self.timer.start(self.timer_delay)

    def pause_animation(self):
        self.timer.stop()

    def take_screenshot(self):
        pixmap = self.grab()
        filename = f"circle_screenshot_{len(os.listdir('.'))}.png"
        pixmap.save(filename)
        QMessageBox.information(self, "Сохранено", f"Скриншот сохранен как {filename}")

    def update_position(self):
        width = self.width() - self.circle_radius * 2
        height = self.height() - self.circle_radius * 2

        self.x += self.vx
        self.y += self.vy

        if self.x <= 0 or self.x >= width:
            self.vx *= -1
```

```
            self.x = max(0, min(self.x, width))

        if self.y <= 0 or self.y >= height:
            self.vy *= -1
            self.y = max(0, min(self.y, height))

        self.update()

    def paintEvent(self, event):
        super().paintEvent(event)
        painter = QPainter(self)
        painter.setRenderHint(QPainter.Antialiasing)
        painter.setBrush(QColor(0, 150, 255))
        painter.drawEllipse(self.x, self.y, self.circle_radius * 2, self.circle_radius * 2)


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

Результаты работы программы:







Задание 2. Реализовать построение заданного типа фрактала по варианту:

6) Склоненное дерево Пифагора (обдуваемое ветром)

Код программы: three.py:

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


@dataclass
class Point:
    x: int
    y: int

    def copy(self):
        return Point(self.x, self.y)


@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 = 8
        self.angle = math.radians(45)
        self.wind_angle = math.radians(15)  # угол наклона от ветра
        self.ratio = 0.7
        self.wind_direction = 1  # 1 - ветер слева, -1 - справа
        self.color1 = QColor(139, 69, 19)
        self.color2 = QColor(34, 139, 34)
        self.bg_color = QColor(240, 248, 255)

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

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

    def set_wind_angle(self, wind_angle_degrees):
        self.wind_angle = math.radians(wind_angle_degrees)
        self.update()

    def set_wind_direction(self, direction_index):
        self.wind_direction = 1 if direction_index == 0 else -1
        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

        # Плавный изгиб
```

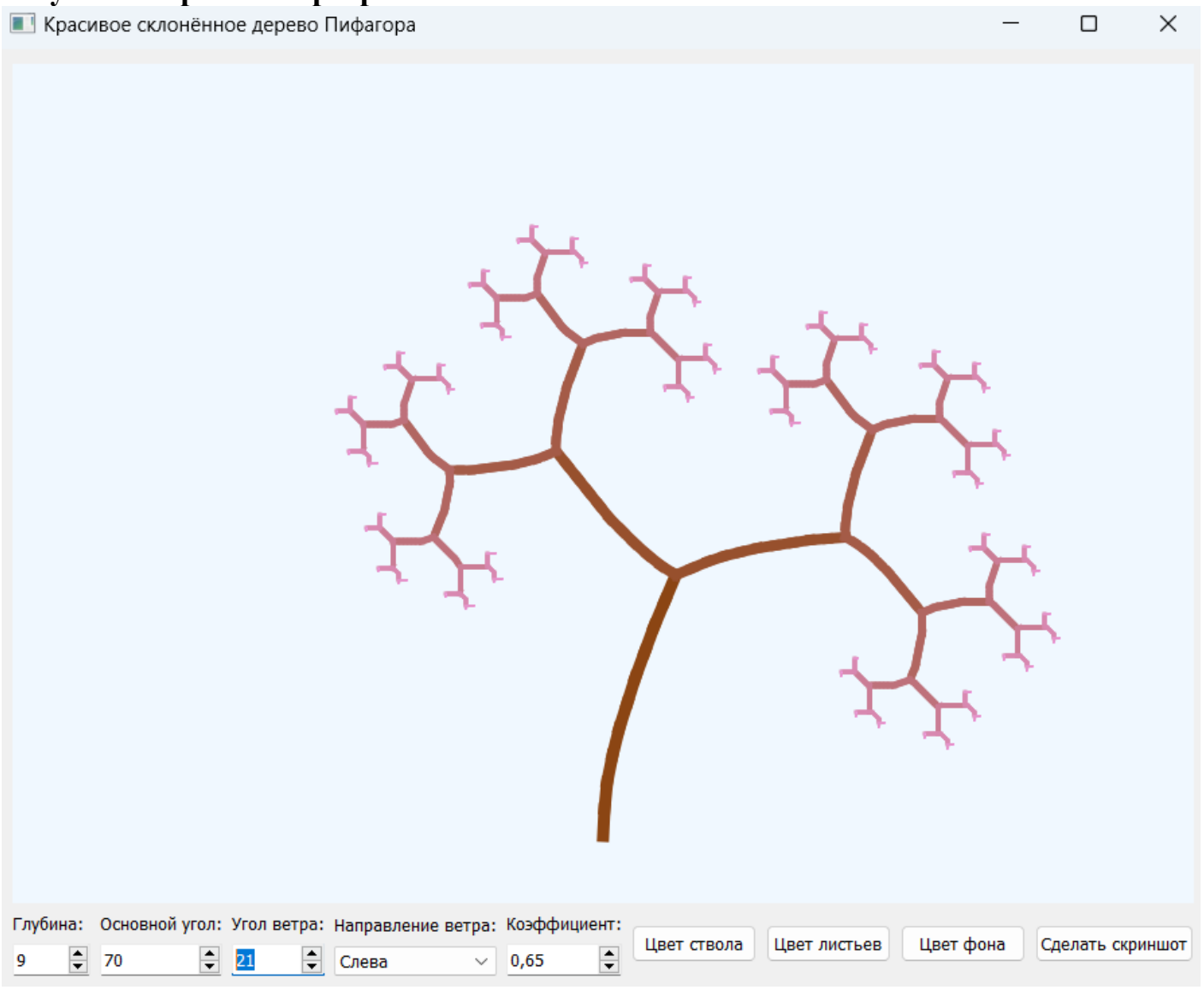
```
        steps = 10
        points = []
        current_point = start_point.copy()
        current_angle = params.angle
        segment_length = params.length / steps

        for i in range(steps + 1):
            points.append(current_point.copy())
            current_angle += self.wind_direction * self.wind_angle * (steps - i) / (steps * 5)
            current_point.x += int(math.cos(current_angle) * segment_length)
            current_point.y += int(math.sin(current_angle) * segment_length)

        # Рисуем плавную ветку
        color = self.get_color_for_depth(params.depth)
        pen = QPen(color)
        pen.setWidth(max(1, params.depth))
        painter.setPen(pen)

        for i in range(len(points) - 1):
            painter.drawLine(points[i].x, points[i].y, points[i + 1].x, points[i + 1].y)

        end_point = points[-1]

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

            left_angle = params.angle - self.angle + self.wind_direction * self.wind_angle * 0.5
            right_angle = params.angle + self.angle - self.wind_direction * self.wind_angle * 0.5

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

            # Правая ветка
            right_params = TreeParams(
                length=new_length,
                angle=right_angle,
                depth=params.depth - 1
            )
            self.draw_tree(painter=painter, start_point=end_point, params=right_params)

    def get_color_for_depth(self, depth):
        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)

    def take_screenshot(self):
        pixmap = self.grab()
        filename = f"pythagoras_tree_{len(os.listdir('.'))}.png"
        pixmap.save(filename)
        QMessageBox.information(self, "Сохранено", f"Скриншот сохранен как {filename}")
```

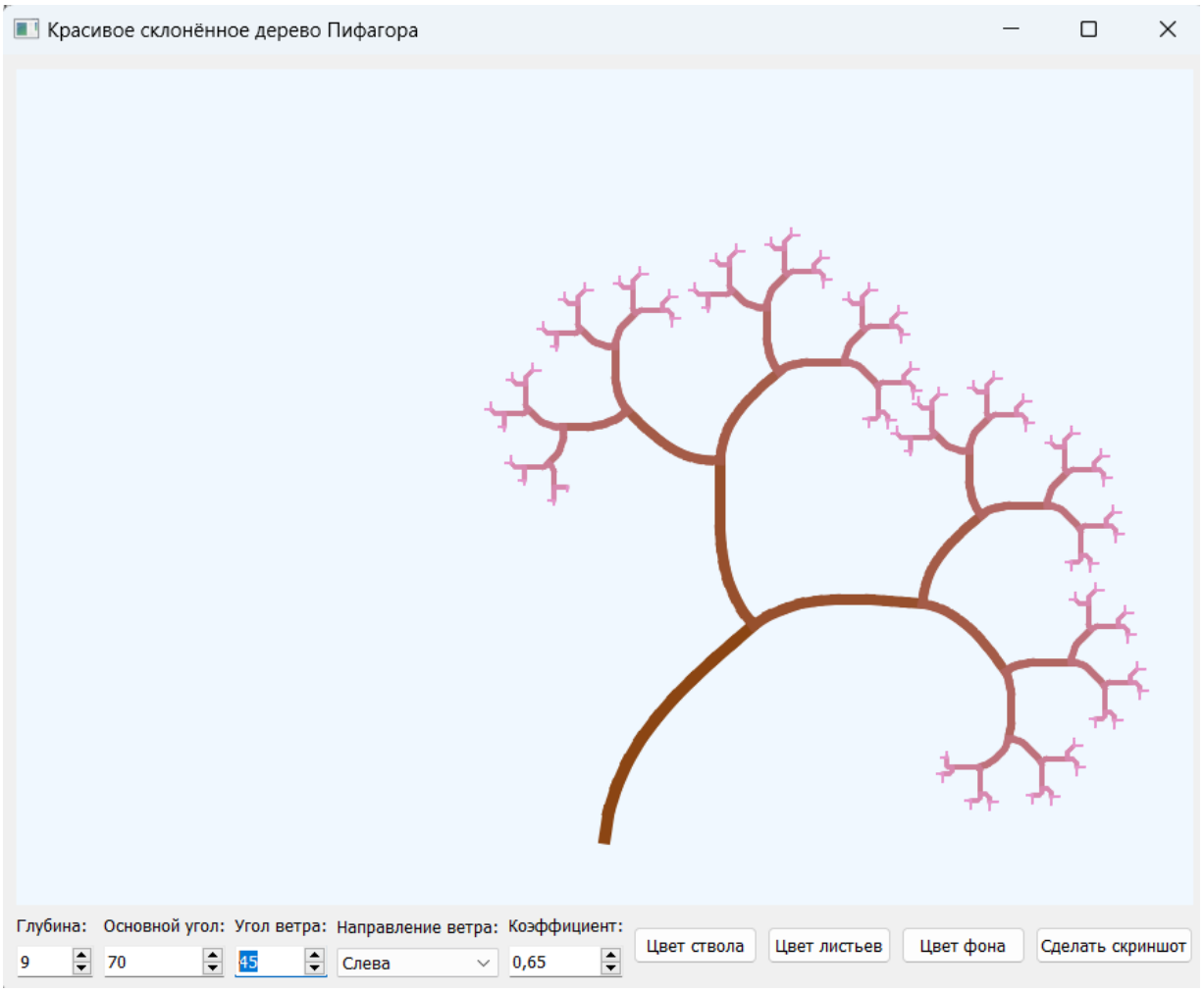
```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Красивое склонённое дерево Пифагора")
        self.setGeometry(100, 100, 900, 700)

        self.tree_widget = None
        self.depth_spin = None
        self.angle_spin = None
        self.wind_angle_spin = None
        self.wind_dir_combo = None
        self.ratio_spin = None
        self.color1_button = None
        self.color2_button = None
        self.bg_color_button = None
        self.screenshot_btn = 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_wind_angle_control(control_layout)
        self.create_wind_dir_combo(control_layout)
        self.create_ratio_control(control_layout)
        self.create_color_buttons(control_layout)
        self.create_screenshot_button(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(8)
        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("Основной угол:"))
        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)
```

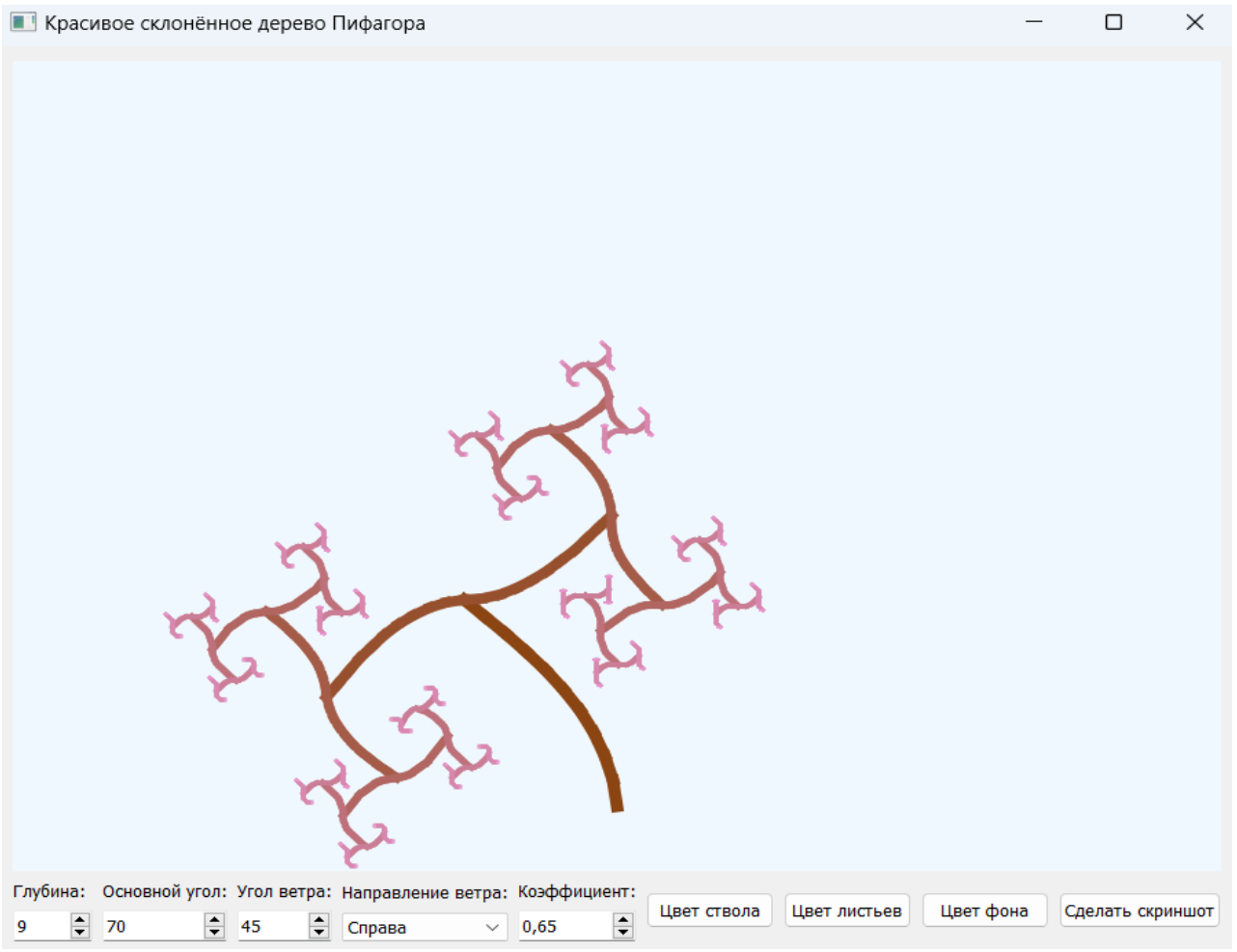
```
    def create_wind_angle_control(self, layout):
        wind_layout = QVBoxLayout()
        wind_layout.addWidget(QLabel("Угол ветра:"))
        self.wind_angle_spin = QSpinBox()
        self.wind_angle_spin.setRange(0, 45)
        self.wind_angle_spin.setValue(15)
        self.wind_angle_spin.valueChanged.connect(self.tree_widget.set_wind_angle)
        wind_layout.addWidget(self.wind_angle_spin)
        layout.addLayout(wind_layout)

    def create_wind_dir_combo(self, layout):
        dir_layout = QVBoxLayout()
        dir_layout.addWidget(QLabel("Направление ветра:"))
        self.wind_dir_combo = QComboBox()
        self.wind_dir_combo.addItems(["Слева", "Справа"])
        self.wind_dir_combo.currentIndexChanged.connect(self.tree_widget.set_wind_direction)
        dir_layout.addWidget(self.wind_dir_combo)
        layout.addLayout(dir_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)

    def create_screenshot_button(self, layout):
        self.screenshot_btn = QPushButton("Сделать скриншот")
        self.screenshot_btn.clicked.connect(self.tree_widget.take_screenshot)
        layout.addWidget(self.screenshot_btn)

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


if __name__ == "__main__":
    import os
    from PyQt5.QtWidgets import QMessageBox
```

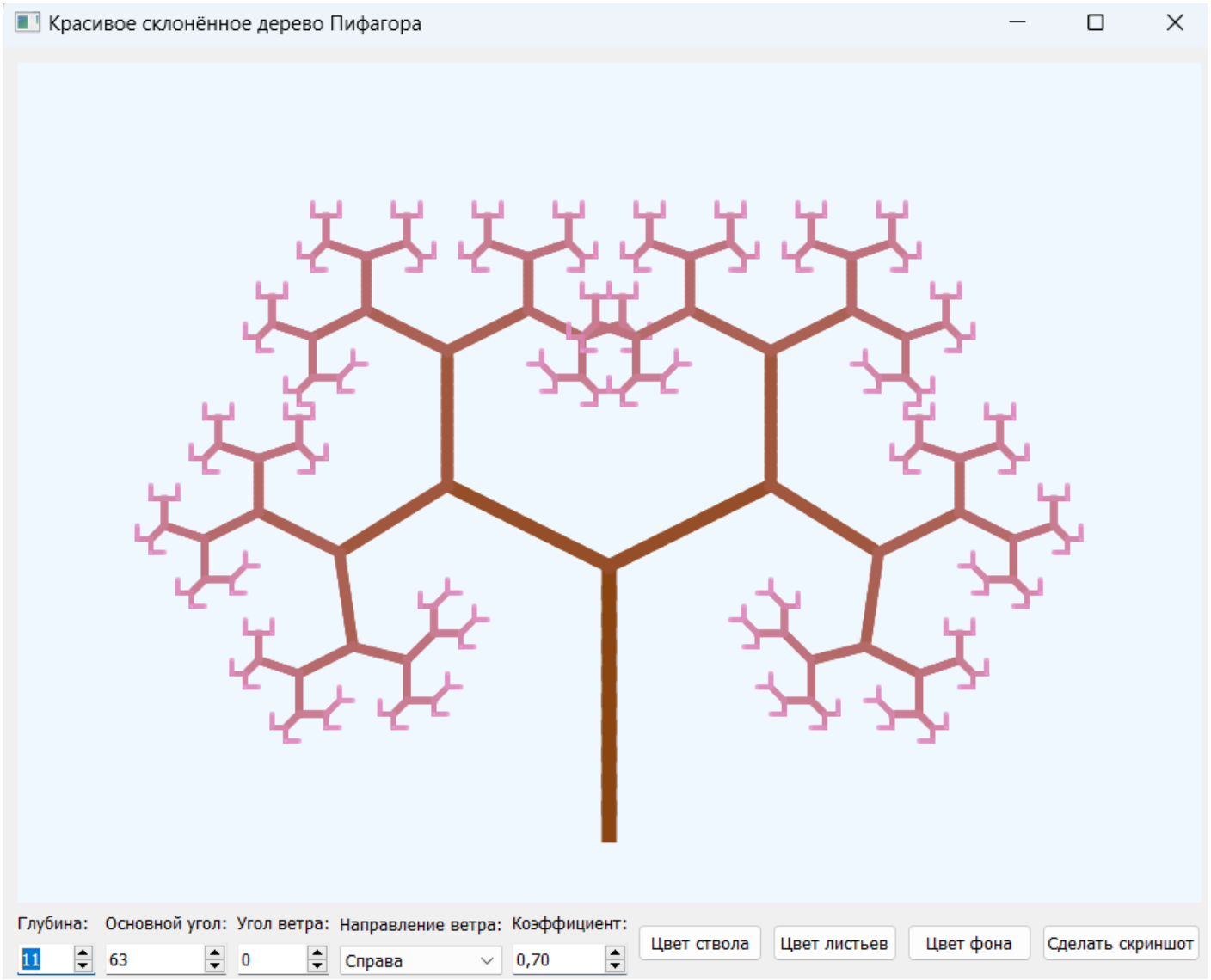
```
app = QApplication(sys.argv)
window = MainWindow()
window.show()
sys.exit(app.exec_())
```

Результаты работы программы:




Красивое склонённое дерево Пифагора
Глубина:
Основной угол:
Угол ветра:
Направление ветра:
Коэффициент:
9
70
45
Слева
0,65
Цвет ствола
Цвет листьев
Цвет фона
Сделать скриншот


Красивое склонённое дерево Пифагора
Глубина:
Основной угол:
Угол ветра:
Направление ветра:
Коэффициент:
9
70
45
Справа
0,65
Цвет ствола
Цвет листьев
Цвет фона
Сделать скриншот



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