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
И. А. Гурин
студент группы ПО11

Проверил
А. А. Крощенко
ст. преп. кафедры ИИТ,
26.04.2025 г.

Брест 2025

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

Задание 1. Задать движение по экрану строк (одна за другой) из списка строк. Направление движения по форме и значение каждой строки выбираются случайным образом.

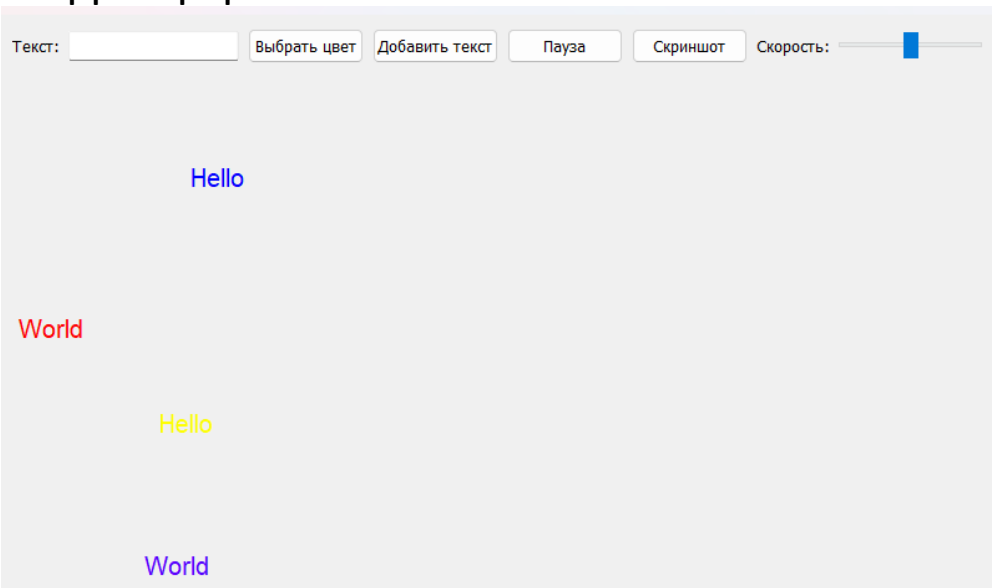
Код программы:

```
import random
import sys
from datetime import datetime

from PyQt5.QtCore import Qt, QTimer
from PyQt5.QtGui import QColor, QFont, QPainter
from PyQt5.QtWidgets import (
    QApplication,
    QColorDialog,
    QHBoxLayout,
    QLabel,
    QLineEdit,
    QMainWindow,
    QPushButton,
    QSlider,
    QVBoxLayout,
    QWidget,
)

class MovingText:
    def __init__(self, text, width, height, color):
        self.text = text
        self.x = random.randint(0, width)
        self.y = random.randint(0, height)
        self.color = color
        self.dx = random.choice([-3, -2, -1, 1, 2, 3])
        self.dy = random.choice([-3, -2, -1, 1, 2, 3])

    def update(self, width, height):
        self.x = (self.x + self.dx) % width
        self.y = (self.y + self.dy) % height

class TextScene(QWidget):
    def __init__(self):
        super().__init__()
        self.texts = []
        self.timer = QTimer()
        self.timer.timeout.connect(self.update_scene)
        self.speed = 30

    def start(self):
        self.timer.start(1000 // self.speed)

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

    def add_text(self, text, color):
        self.texts.append(MovingText(text, self.width(), self.height(), color))

    def update_scene(self):
        for text in self.texts:
            text.update(self.width(), self.height())
        self.update()

    def paintEvent(self, event):
        painter = QPainter(self)
        painter.setFont(QFont("Arial", 12))

        for text in self.texts:
            painter.setPen(text.color)
            painter.drawText(text.x, text.y, text.text)

    def resizeEvent(self, event):
        for text in self.texts:
```

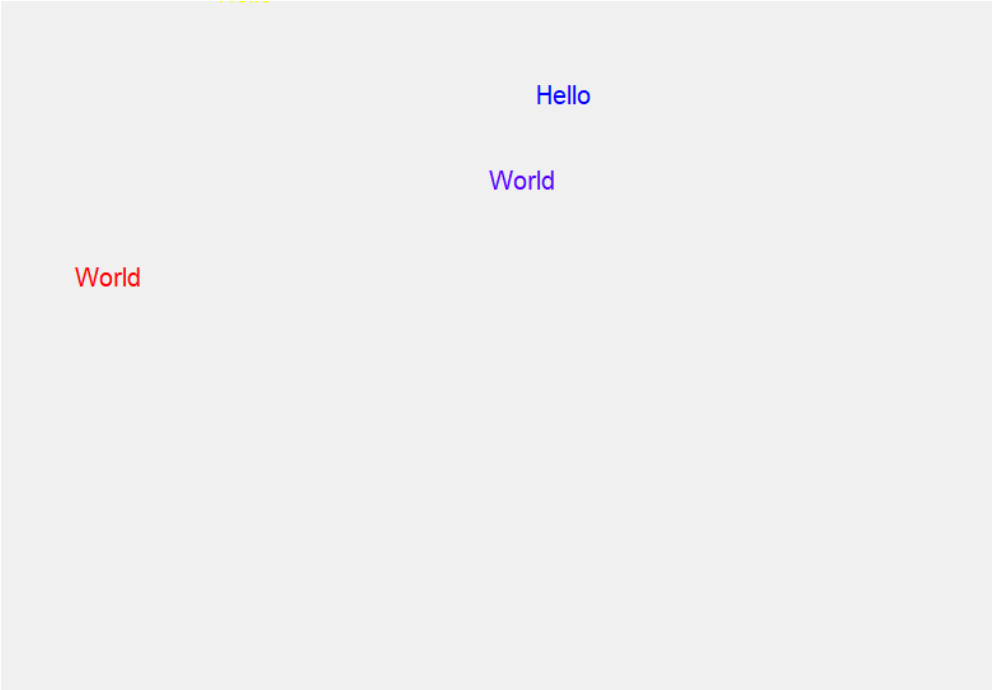
```
            text.x = text.x % self.width()
            text.y = text.y % self.height()

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.scene = TextScene()
        self.text_input = QLineEdit()
        self.color_btn = QPushButton("Выбрать цвет")
        self.add_btn = QPushButton("Добавить текст")
        self.pause_btn = QPushButton("Пауза")
        self.screenshot_btn = QPushButton("Скриншот")
        self.speed_slider = QSlider(Qt.Horizontal)

        self.init_ui()

        self.add_btn.clicked.connect(self.add_text)
        self.pause_btn.clicked.connect(self.toggle_pause)
        self.screenshot_btn.clicked.connect(self.save_screenshot)
        self.color_btn.clicked.connect(self.choose_color)
        self.speed_slider.valueChanged.connect(self.change_speed)

        self.current_color = QColor(0, 0, 0)
        self.speed_slider.setRange(1, 60)
        self.speed_slider.setValue(30)

    def init_ui(self):
        central = QWidget()
        layout = QVBoxLayout()

        control_panel = QHBoxLayout()
        control_panel.addWidget(QLabel("Текст:"))
        control_panel.addWidget(self.text_input)
        control_panel.addWidget(self.color_btn)
        control_panel.addWidget(self.add_btn)
        control_panel.addWidget(self.pause_btn)
        control_panel.addWidget(self.screenshot_btn)
        control_panel.addWidget(QLabel("Скорость:"))
        control_panel.addWidget(self.speed_slider)

        layout.addLayout(control_panel)
        layout.addWidget(self.scene)

        central.setLayout(layout)
        self.setCentralWidget(central)
        self.setGeometry(100, 100, 800, 600)

    def add_text(self):
        text = self.text_input.text()
        if text:
            self.scene.add_text(text, self.current_color)
            self.scene.start()
            self.text_input.clear()

    def toggle_pause(self):
        if self.scene.timer.isActive():
            self.scene.stop()
            self.pause_btn.setText("Продолжить")
        else:
            self.scene.start()
            self.pause_btn.setText("Пауза")

    def save_screenshot(self):
        pixmap = self.scene.grab()
        filename = f"screenshot_{datetime.now().strftime('%Y%m%d_%H%M%S')}.png"
        pixmap.save(filename)
        print(f"Скриншот сохранен как {filename}")

    def choose_color(self):
        color = QColorDialog.getColor()
```

```
        if color.isValid():
            self.current_color = color

    def change_speed(self, value):
        self.scene.speed = value
        if self.scene.timer.isActive():
            self.scene.timer.setInterval(1000 // value)

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

Интерфейс программы:



Сохранённый скриншот:



Задание 2. Реализовать построение заданного типа фрактала по варианту (8. кривая дракона). Везде, где это необходимо, предусмотреть ввод параметров, влияющих на внешний вид фрактала

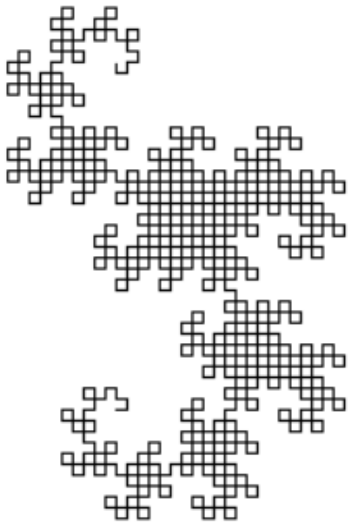
Код программы:

```python
import turtle

def generate_dragon_curve(axiom: str, rules: dict, iterations: int) -> str:
    """
    Генерирует строку для фрактала по L-системе.

    Параметры:
    - axiom: начальная строка.
    - rules: словарь правил переписывания.
    - iterations: число итераций переписывания.

    Возвращает итоговую строку, где записаны команды для рисования.
    """
    sequence = axiom
    for _ in range(iterations):
        next_sequence = ""
        for ch in sequence:
            next_sequence += rules.get(ch, ch)
        sequence = next_sequence
    return sequence

def draw_dragon(sequence: str, step: float, angle: float):
    """
    Рисует кривую дракона по сгенерированной строке.

    Параметры:
    - sequence: строка команд.
    - step: длина шага для команды 'F'.
    - angle: угол поворота (например, 90 градусов).
    """
    for command in sequence:
        if command == "F":
            turtle.forward(step)
        elif command == "+":
            turtle.left(angle)
        elif command == "-":
            turtle.right(angle)

def main():
    try:
        iterations = int(input("Введите число итераций (например, 10): "))
    except ValueError:
        print("Неверный ввод итераций. Используется значение по умолчанию: 10")
        iterations = 10

    try:
        step_length = float(input("Введите длину отрезка (например, 5): "))
    except ValueError:
        print("Неверный ввод длины. Используется значение по умолчанию: 5")
        step_length = 5

    pen_color = input("Введите цвет пера (например, blue): ")
    bg_color = input("Введите цвет фона (например, black): ")

    turtle.setup(width=800, height=600)
    turtle.title("Кривая дракона")
    turtle.bgcolor(bg_color)
    turtle.speed(0)
    turtle.color(pen_color)
    turtle.penup()
    turtle.goto(0, 0)
    turtle.pendown()
    turtle.hideturtle()

    # Определение аксиомы и правил L-системы
    axiom = "FX"
    rules = {"X": "X+YF+", "Y": "-FX-Y"}

    # Генерация строки с командами
    instructions = generate_dragon_curve(axiom, rules, iterations)
```

```
    # Рисование кривой
    draw_dragon(instructions, step_length, 90)

    turtle.hideturtle()
    turtle.done()

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

Результат работы:

```
Введите число итераций (например, 10): 10
Введите длину отрезка (например, 5): 5
Введите цвет пера (например, blue): black
Введите цвет фона (например, black): white
```



Вывод: освоил возможности языка программирования Python в разработке оконных приложений