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
Лесько М.И.
студент группы ПО11

Проверил
А. А. Крощенко ст.
преп. кафедры ИИТ,
02.05.2025 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!

Вариант 12

12) Задать составление строки из символов, появляющихся из разных углов формы и выстраивающихся друг за другом. Процесс должен циклически повторяться.

Код программы:

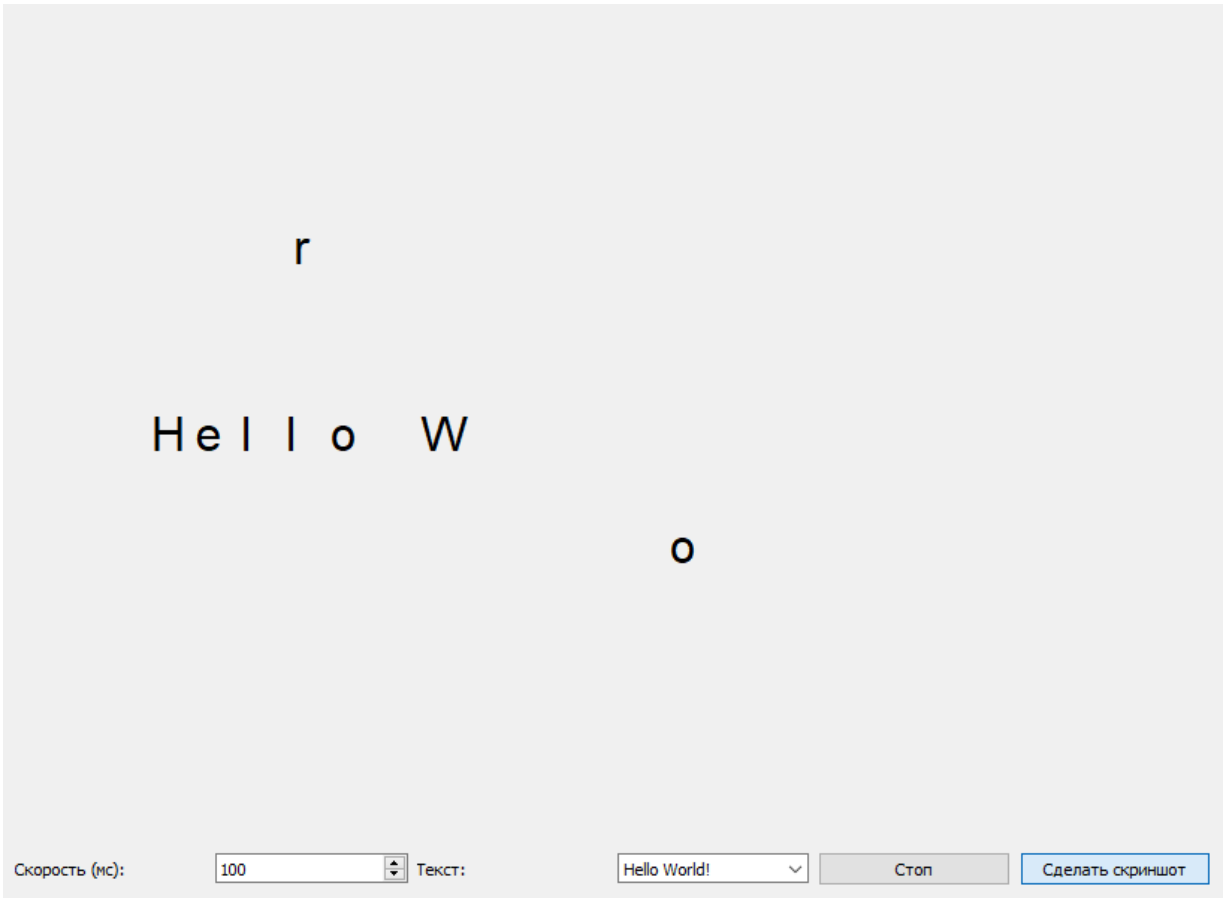
SPP7-1.py:

```
import sys
import os
from datetime import datetime
from PyQt5.QtWidgets import (QApplication, QMainWindow, QWidget, QVBoxLayout,
                             QHBoxLayout, QPushButton, QLabel, QSpinBox,
                             QComboBox, QFileDialog)
from PyQt5.QtCore import Qt, QTimer, QPoint
from PyQt5.QtGui import QPainter, QColor, QFont, QPixmap

class CharacterAnimation(QWidget):
    def __init__(self):
        super().__init__()
        self.characters = []
        self.current_char_index = 0
        self.speed = 100  # миллисекунды
        self.is_running = False
        self.target_text = "Hello World!"
        self.corners = ['top_left', 'top_right', 'bottom_left', 'bottom_right']
        self.current_corner = 0

        self.init_ui()

    def init_ui(self):
        self.setMinimumSize(800, 600)
        self.setWindowTitle('Анимация символов')

        # Создаем главный layout
        main_layout = QVBoxLayout()

        # Создаем область для анимации
        self.animation_area = QWidget()
        self.animation_area.setMinimumSize(800, 500)

        # Создаем layout для элементов управления
        controls_layout = QHBoxLayout()

        # Контроль скорости
        speed_label = QLabel('Скорость (мс):')
        self.speed_spin = QSpinBox()
        self.speed_spin.setRange(50, 1000)
        self.speed_spin.setValue(self.speed)
        self.speed_spin.valueChanged.connect(self.update_speed)

        # Ввод текста
        text_label = QLabel('Текст:')
        self.text_combo = QComboBox()
```

```
        self.text_combo.setEditable(True)
        self.text_combo.addItems(['Hello World!', 'Python', 'Animation'])
        self.text_combo.currentTextChanged.connect(self.update_text)

        # Кнопки управления
        self.start_button = QPushButton('Старт')
        self.start_button.clicked.connect(self.toggle_animation)

        self.screenshot_button = QPushButton('Сделать скриншот')
        self.screenshot_button.clicked.connect(self.take_screenshot)

        # Добавляем элементы управления в layout
        controls_layout.addWidget(speed_label)
        controls_layout.addWidget(self.speed_spin)
        controls_layout.addWidget(text_label)
        controls_layout.addWidget(self.text_combo)
        controls_layout.addWidget(self.start_button)
        controls_layout.addWidget(self.screenshot_button)

        # Добавляем виджеты в главный layout
        main_layout.addWidget(self.animation_area)
        main_layout.addLayout(controls_layout)

        self.setLayout(main_layout)

        # Настраиваем таймер
        self.timer = QTimer()
        self.timer.timeout.connect(self.update_animation)

    def update_speed(self, value):
        self.speed = value
        if self.is_running:
            self.timer.setInterval(self.speed)

    def update_text(self, text):
        self.target_text = text
        self.reset_animation()

    def toggle_animation(self):
        if self.is_running:
            self.timer.stop()
            self.start_button.setText('Старт')
        else:
            self.timer.start(self.speed)
            self.start_button.setText('Стоп')
        self.is_running = not self.is_running

    def reset_animation(self):
        self.characters = []
        self.current_char_index = 0
        self.current_corner = 0
        self.update()

    def get_corner_position(self, corner):
        margin = 50
        if corner == 'top_left':
            return QPoint(margin, margin)
        elif corner == 'top_right':
            return QPoint(self.width() - margin, margin)
        elif corner == 'bottom_left':
            return QPoint(margin, self.height() - margin)
        else:  # bottom_right
            return QPoint(self.width() - margin, self.height() - margin)

    def update_animation(self):
        if self.current_char_index < len(self.target_text):
            corner = self.corners[self.current_corner]
            start_pos = self.get_corner_position(corner)
            target_x = 100 + len(self.characters) * 30  # Располагаем символы горизонтально
```

```
            target_y = self.height() // 2

            self.characters.append({
                'char': self.target_text[self.current_char_index],
                'pos': start_pos,
                'target': QPoint(target_x, target_y),
                'progress': 0
            })

            self.current_char_index += 1
            self.current_corner = (self.current_corner + 1) % len(self.corners)
        else:
            self.reset_animation()

        self.update()

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

        # Рисуем фон
        painter.fillRect(self.rect(), QColor(240, 240, 240))

        # Рисуем символы
        font = QFont('Arial', 24)
        painter.setFont(font)

        for char_data in self.characters:
            progress = char_data['progress']
            current_x = char_data['pos'].x() + (char_data['target'].x() - char_data['pos'].x()) * progress
            current_y = char_data['pos'].y() + (char_data['target'].y() - char_data['pos'].y()) * progress

            painter.drawText(int(current_x), int(current_y), char_data['char'])
            char_data['progress'] = min(1.0, char_data['progress'] + 0.1)

    def take_screenshot(self):
        screenshot = QPixmap(self.size())
        self.render(screenshot)

        # Создаем имя файла с временной меткой
        timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
        filename = f'screenshot_{timestamp}.png'

        # Сохраняем скриншот
        screenshot.save(filename)

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.animation = CharacterAnimation()
        self.setCentralWidget(self.animation)
        self.setWindowTitle('Анимация символов')
        self.show()

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

Рисунок с результатом работы программы:



Задание 2 Реализовать построение заданного типа фрактала по варианту

Везде, где это необходимо, предусмотреть ввод параметров, влияющих на внешний вид фрактала

Вариант 12

12) Кривая Гильберта

Код программы:

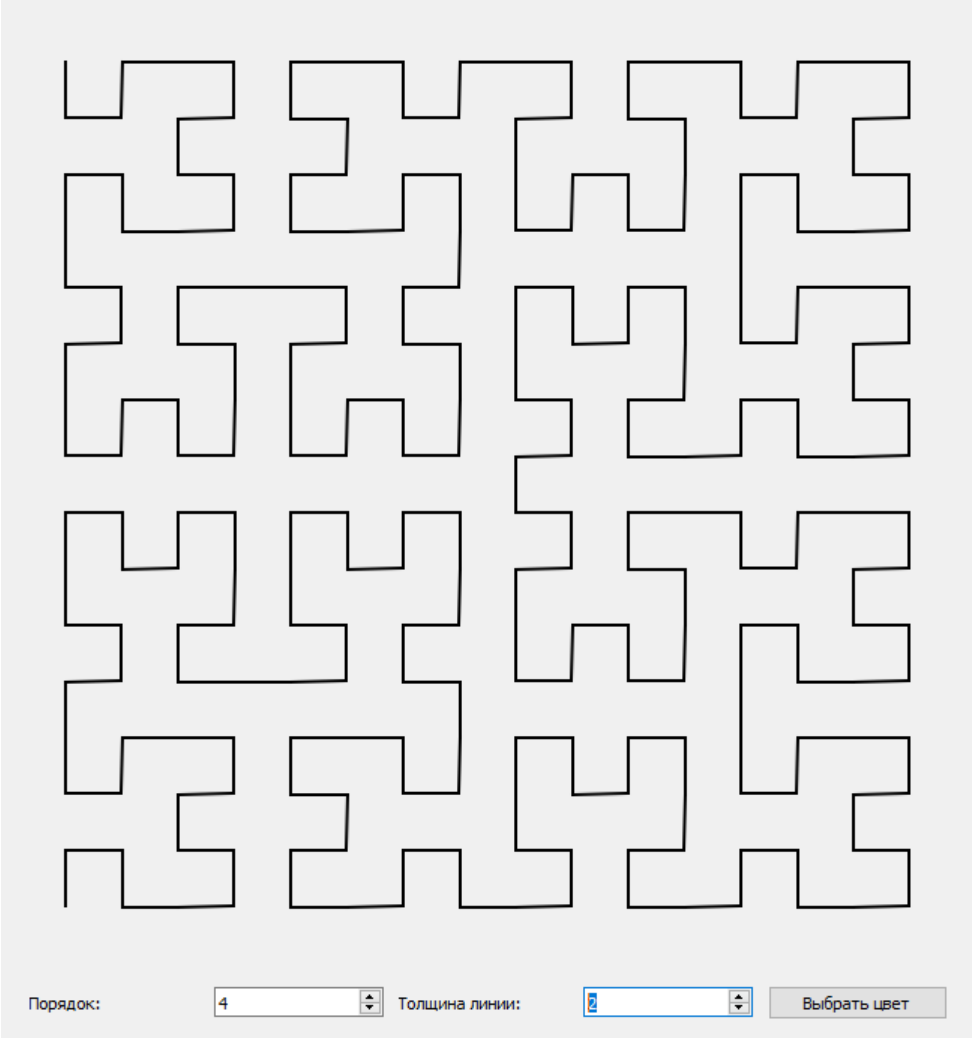
SPP7-2.py:

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

class HilbertCurve(QWidget):
    def __init__(self):
        super().__init__()
        self.order = 3  # Порядок кривой
        self.size = 600  # Размер области рисования
        self.line_color = QColor(0, 0, 0)  # Синий цвет по умолчанию
        self.line_width = 2
        self.init_ui()

    def init_ui(self):
        self.setMinimumSize(self.size + 50, self.size + 100)
        self.setWindowTitle('Кривая Гильберта')

        # Создаем главный layout
        main_layout = QVBoxLayout()
        main_layout.setSpacing(10)  # Добавляем отступы между элементами

        # Создаем область для рисования
        self.drawing_area = QWidget()
        self.drawing_area.setMinimumSize(self.size, self.size)

        # Создаем layout для элементов управления
```

```
    controls_layout = QHBoxLayout()
    controls_layout.setContentsMargins(10, 10, 10, 10)  # Добавляем отступы вокруг элементов управления

    # Контроль порядка кривой
    order_label = QLabel('Порядок:')
    self.order_spin = QSpinBox()
    self.order_spin.setRange(1, 8)
    self.order_spin.setValue(self.order)
    self.order_spin.valueChanged.connect(self.update_order)

    # Контроль толщины линии
    width_label = QLabel('Толщина линии:')
    self.width_spin = QSpinBox()
    self.width_spin.setRange(1, 10)
    self.width_spin.setValue(self.line_width)
    self.width_spin.valueChanged.connect(self.update_line_width)

    # Кнопка выбора цвета
    self.color_button = QPushButton('Выбрать цвет')
    self.color_button.setMinimumWidth(120)  # Устанавливаем минимальную ширину кнопки
    self.color_button.clicked.connect(self.choose_color)

    # Добавляем элементы управления в layout
    controls_layout.addWidget(order_label)
    controls_layout.addWidget(self.order_spin)
    controls_layout.addWidget(width_label)
    controls_layout.addWidget(self.width_spin)
    controls_layout.addWidget(self.color_button)

    # Добавляем виджеты в главный layout
    main_layout.addWidget(self.drawing_area)
    main_layout.addStretch()  # Добавляем растягивающийся элемент
    main_layout.addLayout(controls_layout)

    self.setLayout(main_layout)

def update_order(self, value):
    self.order = value
    self.update()

def update_line_width(self, value):
    self.line_width = value
    self.update()

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

def hilbert_curve(self, order, x, y, xi, xj, yi, yj, points):
    if order == 0:
        points.append(QPoint(x + (xi + yi) // 2, y + (xj + yj) // 2))
    else:
        self.hilbert_curve(order - 1, x, y, yi // 2, yj // 2, xi // 2, xj // 2, points)
        self.hilbert_curve(order - 1, x + xi // 2, y + xj // 2, xi // 2, xj // 2, yi // 2, yj // 2, points)
        self.hilbert_curve(order - 1, x + xi // 2 + yi // 2, y + xj // 2 + yj // 2, xi // 2, xj // 2, yi // 2, yj // 2, points)
        self.hilbert_curve(order - 1, x + xi // 2 + yi, y + xj // 2 + yj, -yi // 2, -yj // 2, -xi // 2, -xj // 2, points)

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

    # Устанавливаем цвет и толщину линии
    pen = QPen(self.line_color, self.line_width)
    painter.setPen(pen)

    # Вычисляем размер шага
    step = self.size // (2 ** self.order)
```

```
        # Генерируем точки кривой
        points = []
        self.hilbert_curve(self.order, 25, 25, self.size, 0, 0, self.size, points)  # Изменяем начальные координаты

        # Рисуем линии между точками
        for i in range(len(points) - 1):
            painter.drawLine(points[i], points[i + 1])

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.hilbert = HilbertCurve()
        self.setCentralWidget(self.hilbert)
        self.setWindowTitle('Кривая Гильберта')
        self.show()

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

Рисунок с результатом работы программы:



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