МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и
автоматизированных систем

Лабораторная работа №13
по дисциплине: Объектно-ориентированное программирование
Тема: Знакомство с библиотеками языка Python. PyQT.

Выполнил: студент группы ПВ-223
Дмитриев А.А.
Проверил:
Черников С.В.

Белгород 2024 г.

Цель работы: приобретение практических навыков создания приложений на языке Python, QT приложения.

Задание:
Для выполнения лабораторной работы требуется установить интерпретатор Python версии 3.6+. Выполнить написание программы-сценария в соответствии с вариантом задания. Провести тестирование. Оформить отчет.

Вариант 2:
QT-судоку

Код (отрисовщик, слушатель событий):

```
from collections import defaultdict
import copy
import sys

from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from PyQt5.QtGui import *

import sudoku_generator


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

        self.setWindowTitle('Sudoku')
        self.setMouseTracking(True)

        self.cell_size = 20
        self.default_size = 300
        self.default_pen_size_1 = 1.0
        self.default_pen_size_2 = 5.0
        self.min_default_pen_size_2 = 2.0

        self.matrix_size = 9
        self.sub_matrix_size = 3
        self.x_highlight_cell = -1
        self.y_highlight_cell = -1

        self.resize(self.default_size, self.default_size)

        self.matrix = None
        self.sudoku_size = None
        self.orig_matrix = None
        self.def_num_matrix = None

        self.invalid_indexes = []

        self.new_sudoku()

    def new_sudoku(self):
        self.invalid_indexes.clear()

        self.matrix, self.sudoku_size = sudoku_generator.gen()
        self.orig_matrix = copy.deepcopy(self.matrix)

        self.def_num_matrix = [
            [bool(i) for i in row]
            for row in self.orig_matrix
        ]

    def keyPressEvent(self, event):
        super().keyPressEvent(event)

        if event.key() == Qt.Key_Space:
            self.new_sudoku()
            self.update()

    def resizeEvent(self, event):
        super().resizeEvent(event)

        self.cell_size = min(event.size().width(), \
                                    event.size().height()) // self.matrix_size

    def mouseMoveEvent(self, event):
        super().mouseMoveEvent(event)
```

```
        pos = event.pos()

        self.x_highlight_cell = pos.x() // self.cell_size
        self.y_highlight_cell = pos.y() // self.cell_size

        self.update()

    def mouseReleaseEvent(self, event):
        super().mouseReleaseEvent(event)

        pos = event.pos()
        x = pos.x() // self.cell_size
        y = pos.y() // self.cell_size

        has_empty_cell = False
        try:
            if not self.def_num_matrix[x][y]:
                if event.button() == Qt.LeftButton:
                    self.matrix[x][y] = self.matrix[x][y] + 1 \
                                                    if self.matrix[x][y] < 9 else 0
                elif event.button() == Qt.RightButton:
                    self.matrix[x][y] = self.matrix[x][y] - 1 \
                                                    if self.matrix[x][y] > 0 else 9
                elif event.button() == Qt.MiddleButton:
                    self.matrix[x][y] = 0

                self.invalid_indexes.clear()

                # В одной плоскости
                for i in range(self.matrix_size):
                    num_by_indexes = defaultdict(list)

                    for j in range(self.matrix_size):
                        num = self.matrix[i][j]
                        if num:
                            num_by_indexes[num].append((i, j))
                        else:
                            has_empty_cell = True

                    for k, v in num_by_indexes.items():
                        if len(v) > 1:
                            self.invalid_indexes += v

                # В другой плоскости
                for i in range(self.matrix_size):
                    num_by_indexes = defaultdict(list)

                    for j in range(self.matrix_size):
                        num = self.matrix[j][i]
                        if num:
                            num_by_indexes[num].append((j, i))
                        else:
                            has_empty_cell = True

                    for k, v in num_by_indexes.items():
                        if len(v) > 1:
                            self.invalid_indexes += v

                # В подквадратах
                for si in range(0, self.matrix_size, self.sub_matrix_size):
                    for sj in range(0, self.matrix_size, self.sub_matrix_size):
                        num_by_indexes = defaultdict(list)
                        sub_indexes = []

                        for i in range(self.sub_matrix_size):
                            for j in range(self.sub_matrix_size):
```

```
                            sub_indexes.append((si + i, sj + j))

                    for i, j in sub_indexes:
                        num = self.matrix[i][j]
                        if num:
                            num_by_indexes[num].append((i, j))

                    for k, v in num_by_indexes.items():
                        if len(v) > 1:
                            self.invalid_indexes += v

            self.update()

            # Проверяем решение судоку
            if not has_empty_cell and not self.invalid_indexes:
                QMessageBox.information(self, 'Победа', 'Судоку решена!')

    except IndexError:
        pass

def _draw_background_cell(self, painter: QPainter):
    painter.save()

    for i in range(self.matrix_size):
        for j in range(self.matrix_size):
            if (i, j) in self.invalid_indexes:
                color = Qt.red
            elif self.def_num_matrix[i][j]:
                color = Qt.yellow
            else:
                color = Qt.white

            painter.setBrush(color)

            x = i * self.cell_size
            y = j * self.cell_size
            w, h = self.cell_size, self.cell_size
            painter.drawRect(x, y, w, h)

    if 0 <= self.x_highlight_cell < self.matrix_size \
                        and 0 <= self.y_highlight_cell < self.matrix_size:
        x, y = self.x_highlight_cell, self.y_highlight_cell

        if not self.def_num_matrix[x][y]:
            painter.setBrush(Qt.darkYellow)
            painter.drawRect(
                x * self.cell_size,
                y * self.cell_size,
                self.cell_size,
                self.cell_size
            )

    painter.restore()

def _draw_cell_numbers(self, painter: QPainter):
    painter.save()

    for i in range(self.matrix_size):
        for j in range(self.matrix_size):
            num = self.matrix[i][j]
            if not num:
                continue

            num = str(num)

            factor = (self.cell_size / 2) / painter.fontMetrics().width(num)
            if factor < 1 or factor > 1.25:
```

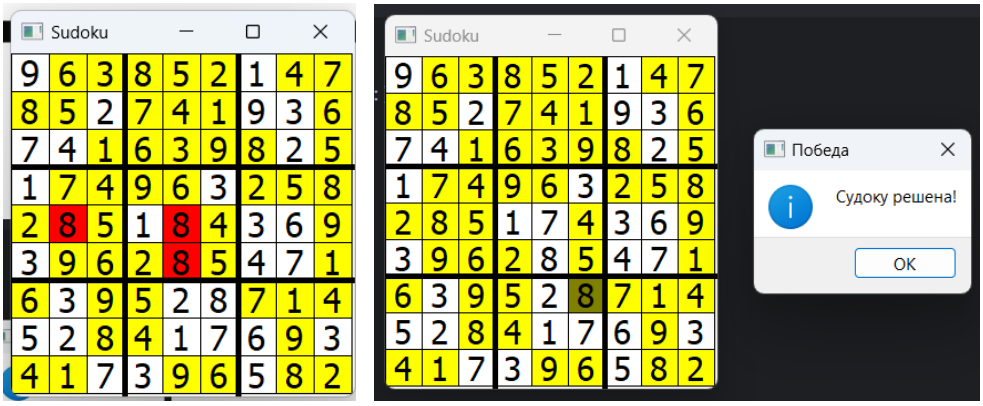
```
                    f = painter.font()
                    point_size = f.pointSizeF() * factor
                    if point_size > 0:
                        f.setPointSizeF(point_size)
                        painter.setFont(f)

                x = i * self.cell_size
                y = j * self.cell_size
                w, h = self.cell_size, self.cell_size
                painter.drawText(x, y, w, h, Qt.AlignCenter, num)

        painter.restore()

    def _draw_grid(self, painter: QPainter):
        painter.save()

        y1, y2 = 0, 0

        factor = min(self.width(), self.height()) / self.default_size
        size = self.default_pen_size_1
        size2 = self.default_pen_size_2

        if factor < 1 or factor > 1.25:
            size *= factor
            if size < self.min_default_pen_size_2:
                size = self.min_default_pen_size_2

        def _is_border(i):
            return i % self.sub_matrix_size == 0 and i and i < self.matrix_size

        for i in range(self.matrix_size + 1):
            painter.setPen(QPen(Qt.black, size2 if _is_border(i) else size))
            painter.drawLine(0, y1, self.cell_size * self.matrix_size, y2)

            y1 += self.cell_size
            y2 += self.cell_size

        x1, x2 = 0, 0
        for i in range(self.matrix_size + 1):
            painter.setPen(QPen(Qt.black, size2 if _is_border(i) else size))
            painter.drawLine(x1, 0, x2, self.cell_size * self.matrix_size)

            x1 += self.cell_size
            x2 += self.cell_size

        painter.restore()

    def paintEvent(self, event):
        super().paintEvent(event)

        painter = QPainter(self)

        self._draw_background_cell(painter)

        self._draw_cell_numbers(painter)

        self._draw_grid(painter)


if __name__ == '__main__':
    app = QApplication(sys.argv)

    w = Widget()
    # w.resize(200, 200)
    w.show()

    sys.exit(app.exec_())
```

Код (генератор поля):

```
import random


class grid:
    def __init__(self, n=3):
        self.n = n
        self.table = [
            [
                ((i * n + i // n + j) % (n * n) + 1)
                for j in range(n * n)
            ]
            for i in range(n * n)
        ]

    def show(self):
        for row in self.table:
            print(row)

    def transposing(self):
        self.table = map(list, zip(*self.table))
        self.table = list(self.table)

    def swap_rows_small(self):
        area = random.randrange(0, self.n, 1)
        line1 = random.randrange(0, self.n, 1)
        # получение случайного района и случайной строки
        N1 = area * self.n + line1
        # номер 1 строки для обмена

        line2 = random.randrange(0, self.n, 1)
        # случайная строка, но не та же самая
        while line1 == line2:
            line2 = random.randrange(0, self.n, 1)

        N2 = area * self.n + line2

        # номер 2 строки для обмена
        self.table[N1], self.table[N2] = self.table[N2], self.table[N1]

    def swap_columns_small(self):
        self.transposing()
        self.swap_rows_small()
        self.transposing()

    def swap_rows_area(self):
        area1 = random.randrange(0, self.n, 1)

        area2 = random.randrange(0, self.n, 1)
        while area1 == area2:
            area2 = random.randrange(0, self.n, 1)

        for i in range(0, self.n):
            N1, N2 = area1 * self.n + i, area2 * self.n + i
            self.table[N1], self.table[N2] = self.table[N2], self.table[N1]

    def swap_columns_area(self):
        self.transposing()
        self.swap_rows_area()
        self.transposing()

    def mix(self, amt=10):
        mix_func = ['self.transposing()',
                    'self.swap_rows_small()',
                    'self.swap_columns_small()',
```

```
                    'self.swap_rows_area()',
                    'self.swap_columns_area()']
        for i in range(1, amt):
            id_func = random.randrange(0, len(mix_func), 1)
            eval(mix_func[id_func])


def gen():
    example = grid()
    example.mix()

    n = example.n * example.n

    mask = [
        [0 for _ in range(n)]
        for _ in range(n)
    ]

    iterator = 0
    while iterator < 25:
        i = random.randrange(0, n)
        j = random.randrange(0, n)
        if mask[i][j] == 0 and sum(mask[i]) < n - 2 and sum(mask[:][j]) < n - 2:
            iterator += 1

            mask[i][j] = 1
            example.table[i][j] = 0

    return example.table, (example.n, example.n)
```

Пример работы:



Вывод: В ходе лабораторной работы приобрели практические навыки создания приложений на языке Python, QT приложения.