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-судокуКод (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 \

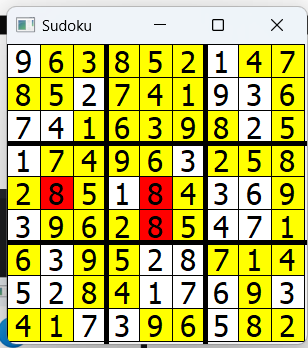
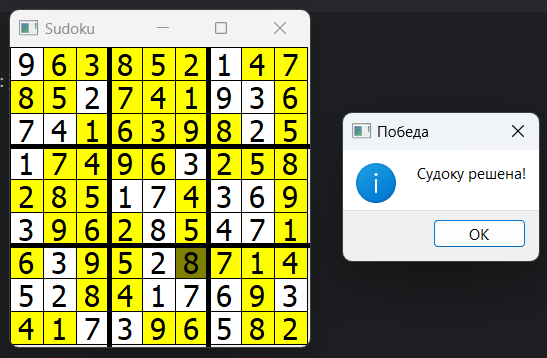
if self.matrix[x][y] > 0 else 9  
 elif event.button() == Qt.MiddleButton:  
 self.matrix[x][y] = 0  
  
 self.invalid\_indexes.clear()  
  
 # В одной плоскости  
 for i in range(self.matrix\_size):  
 num\_by\_indexes = defaultdict(list)  
  
 for j in range(self.matrix\_size):  
 num = self.matrix[i][j]  
 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.