МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по летней практике, итерация №3

Тема: Генетические Алгоритмы “Решение задачи судоку”

Студенты гр. 3342 Галеев, Романов, Хайруллов.

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2025

Цель работы.

Написать программу решающую задачу судоку с использованием генетического алгоритма.

Задание.

Для заданного игрового поля судоку (размеры 9x9) необходимо разместить цифры от 1 до 9, так, чтобы они не нарушали правил.

Выполнение работы.

В рамках нашей команды было принято решение распределить обязанности таким образом:

Галеев Амир – отвечает за разработку и совершенствование графического интерфейса пользователя (GUI);

Хайруллов Динар – отвечает за создание и реализацию одних из основных функциональных модулей алгоритма;

Романов Егор – обеспечивает создание одних из основных функций генетического алгоритма.

В ходе третьей итерации нашего проекта мы получили рабочую версию генетического алгоритма. Все необходимые функции были разработаны, и связаны между собой, что позволило алгоритму корректно работать.

Вклад участников:

Романов Егор разрабатывал функции генетического алгоритма, такие как функция скрещивания, селекции и функция оценки приспособленности сущности. Функция скрещивания объединяет родительские квадраты 3 на 3, функция селекции реализует турнирный метод обтора, а функция приспособленности оценивает кол-во уникальных элементов в строке. Также объединил основные функции генетического алгоритма для его работы.

Хайруллов Динар реализовывал функции мутации и генерации первоначальной популяции. Функция мутации случайным образом меняет местами два элемента. Такая мутация используется для внесения вариативности в алгоритм. Функции учитывают начальные положения элементов на доске и не меняет их. Также для тестирования была разработана другая функция мутации, которая принимает на вход список “плохих” строк или столбцов и меняет местами клетки в пределах этих строк или столбцов.

Галеев Амир переработал графический интерфейс с учетом всех полученных замечаний и частично интегрировал GUI с алгоритмом. Были

написаны функции для сохранения и чтения лучших особей поколения, функции для форматирования таблиц судоку. Были написаны функции для отображения данных в интерфейсе и функции для запуска алгоритма из самого интерфейса.

Описание функций и структур данных.

GUI

Класс: Ui_MainWindow

Класс отвечает за создание и настройку пользовательского интерфейса главного окна приложения с использованием PyQt5. Интерфейс включает в себя график, таблицу результатов, панель параметров и управляющие кнопки.

Метод: setupUi(self, MainWindow)

Метод настраивает все элементы интерфейса внутри главного окна MainWindow.

field_to_str(field) — преобразует поле 9×9 (список списков чисел) в строку из 81 символа.

str_to_field(string) — преобразует строку из 81 символа обратно в поле 9×9 (список списков чисел).

format_9x9_square(arr) — форматирует поле 9×9 в читаемый текст с разделителями между элементами.

save_data(id, data_array) — сохраняет массив данных по ключу id в файл и добавляет значение в список best_fitness.

read_data() — считывает и возвращает данные из JSON-файла data.json.

data_init() — инициализирует файл data.json с пустым списком best_fitness.

clean_data() — очищает файл data.json, удаляя всё содержимое.

Генетический алгоритм

Класс:FieldCreator

Создает начальную расстановку судоку и генерирует популяцию решенийс учетом уже занятых клеток поля.

Методы:

__init__(self) – Инициализирует списки чисел и фиксированных значений.

ReadFromFile(file_name) – Загружает судоку из файла (x – пустые клетки).

ReadFromList(arr) – Берет начальное состояние из двумерного списка.

GeneratePopulation(entities_amount) – Возвращает список случайных заполнений (особей) судоку с заданным кол-вом особей и с учетом уже занятых клеток из считанной начальной расстановки.

Функции оценки приспособленности (fitness)

fitness_full(individual) – Считает общее количество уникальных значений в строках, столбцах и квадратах 3×3. Максимальное значение — 243 (9 строк × 9 + 9 столбцов × 9 + 9 квадратов × 9).

fitness_cut(individual) – Считает только полностью правильные строки, столбцы и квадраты (где все 9 чисел уникальны). Максимальное значение — 27 (9 строк + 9 столбцов + 9 квадратов).

Функции селекции

group_tournament_selection(population, k=2) – Разбивает популяцию на группы по k особей и выбирает лучшую из каждой (по fitness_full).

Функции скрещивания (кроссовера)

one_point_crossing_sq(parent1, parent2, fixed_positions) – Обменивает случайные квадраты 3×3 между родителями, сохраняя фиксированные числа.

uniform_crossover_sq(parent1, parent2, fixed_positions) – Для каждого квадрата 3×3 случайно выбирает родителя и копирует его значения, сохраняя фиксированные числа.

Функции анализа и мутации

get_bad_rows(individual) – Возвращает индексы строк, где много дубликатов (низкая приспособленность).

Главный цикл генетического алгоритма

genetic_algorithm(population, fixed_positions, generations, population_size, mutation_rate) – Основной цикл алгоритма:

Селекция (group_tournament_selection)

Скрещивание (one_point_crossing_sq)

Мутация (случайная перестановка в плохих строках или случайных клетках)

Остановка при нахождении решения (fitness_full = 243)

Выводит прогресс и сохраняет лучшие решения.

Функция создания графика

plot_progress(fitness_values) – Функция построения графика особи с лучшим показателем приспособленности от кол-во популяций

Тестирование.

Тестирование для пустого поля:

Входные данные:

Пустое судоку-поле

x x x x x x x x x

x x x x x x x x x

x x x x x x x x x

x x x x x x x x x

x x x x x x x x x

x x x x x x x x x

x x x x x x x x x

x x x x x x x x x

x x x x x x x x x

Вероятность мутации: 0.55;

Размер начальной популяции: 500.

Выходные данные:

Sudoku solved!

243

[8, 2, 1, 7, 9, 6, 3, 5, 4]

[9, 5, 6, 1, 4, 3, 8, 7, 2]

[4, 7, 3, 5, 2, 8, 9, 6, 1]

[6, 9, 2, 8, 1, 4, 5, 3, 7]

[3, 8, 7, 9, 5, 2, 1, 4, 6]

[1, 4, 5, 6, 3, 7, 2, 9, 8]

[7, 1, 9, 2, 6, 5, 4, 8, 3]

[2, 3, 8, 4, 7, 9, 6, 1, 5]

[5, 6, 4, 3, 8, 1, 7, 2, 9]

График зависимости лучшего показателя функции приспособленности от номера популяции представлен на рисунке 1:

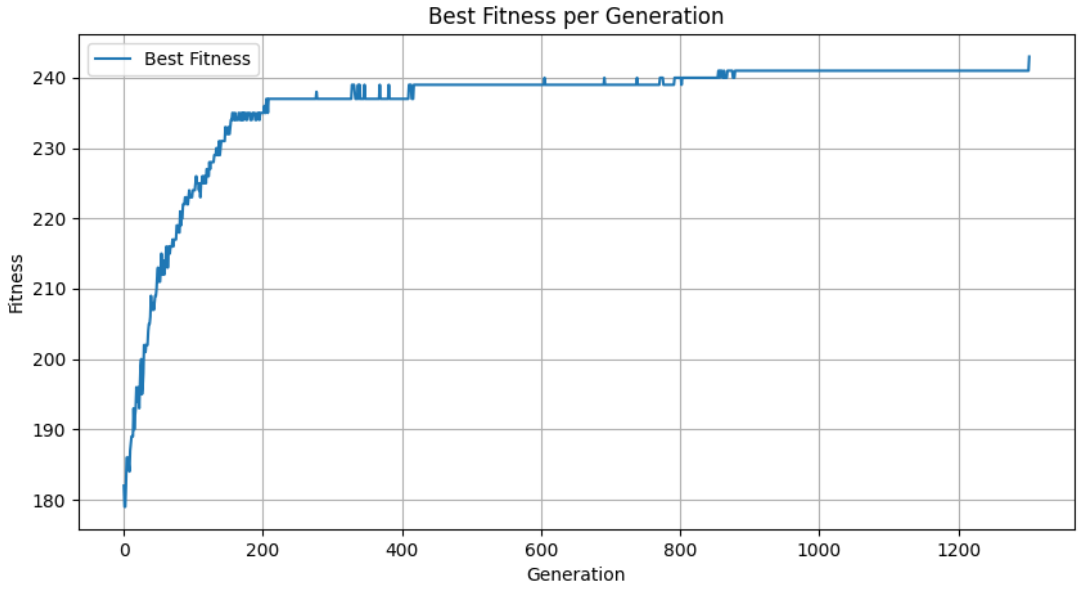


Рис. 1 — график зависимости приспособленности от популяции.

Тестирование для частично-заполненного поля:

Входные данные:

Пустое судоку-поле

x 3 x x x x x x x

x x x x x x x x x

x x x x x 2 x x x

x x x x x x x x x

x x x x x x x x 9

x x 4 x x x x x x

x x x x x x x x x

x 9 x x x x x x x

x x x x x x x x x

Вероятность мутации: 0.55;

Размер начальной популяции: 500.

Выходные данные:

Sudoku solved!

243

[4, 3, 9, 8, 7, 5, 2, 1, 6]

[2, 7, 6, 1, 3, 9, 8, 5, 4]

[1, 8, 5, 6, 4, 2, 3, 9, 7]

[6, 2, 3, 9, 1, 8, 4, 7, 5]

[8, 5, 7, 2, 6, 4, 1, 3, 9]

[9, 1, 4, 7, 5, 3, 6, 8, 2]

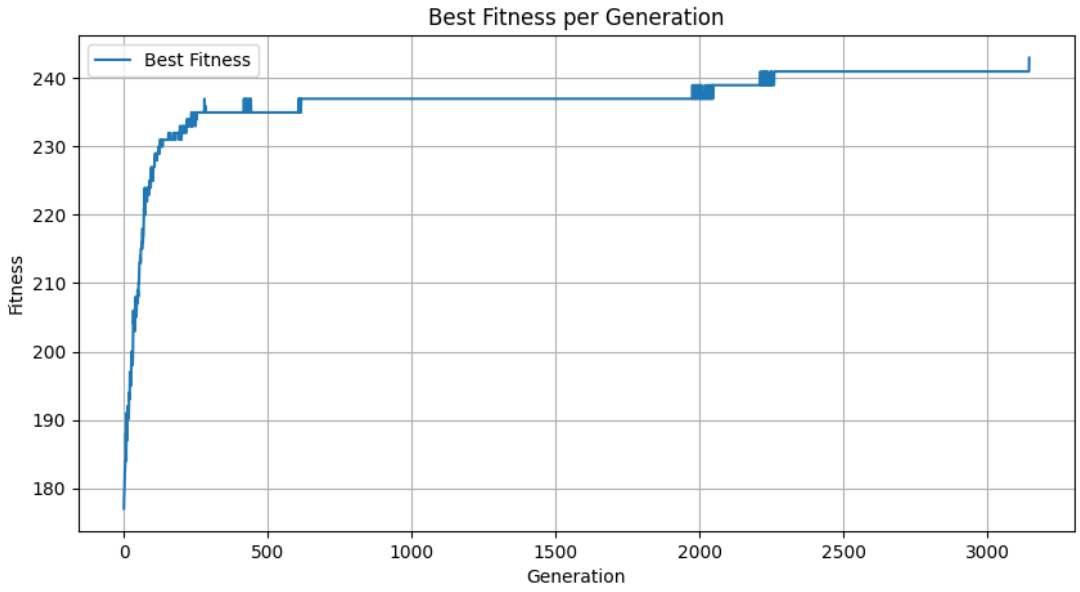
[3, 4, 1, 5, 9, 6, 7, 2, 8]

[7, 9, 2, 4, 8, 1, 5, 6, 3]

[5, 6, 8, 3, 2, 7, 9, 4, 1]

Видно, что изначальные элементы в решении сохранились.

График зависимости лучшего показателя функции приспособленности от номера популяции представлен на рисунке 2



:

Рис. 2 — график зависимости приспособленности от популяции.

Вывод.

На данной итерации был разработан основной генетический алгоритм, включающий основные функции для его работы: функции генерации первой популяции селекции, скрещивания, мутации, оценки приспособленности, также разработана финальная версия графического интерфейса, которая в дальнейшем может претерпеть лишь небольшие изменения.

На данном этапе графический интерфейс еще не связан с работой основных функций, но сам генетический алгоритм в большинстве случаев уже выполняет основную функцию: находит решение судоку (либо же максимально приближает решение к лучшему в определенных случаях). Это возможно благодаря тому, что основные функции для работы алгоритма реализованы и связаны между собой.

Было проведено тестирование, которое показывает решения, находимые алгоритмом, также отображает изменение приспособленности с течением работы алгоритма и созданием новых популяций.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: gui.py

```
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.figure import Figure
from PyQt5 import QtCore, QtGui, QtWidgets
from main import *
from data_saver import *


class Ui_MainWindow(object):
    def __init__(self):
        super().__init__()
        self._is_paused = False
        self._is_stopped = False


    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(820, 660)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Fixed)

sizePolicy.setHeightForWidth(MainWindow.sizePolicy().hasHeightForWidth())
        MainWindow.setSizePolicy(sizePolicy)
        MainWindow.setMinimumSize(QtCore.QSize(820, 660))
        MainWindow.setMaximumSize(QtCore.QSize(820, 660))
        font = QtGui.QFont()
        font.setBold(False)
        font.setWeight(50)
        MainWindow.setFont(font)
        MainWindow.setStyleSheet("background-color: rgb(235, 236, 236);\n"
                                 "color: rgb(48, 48, 48);")

        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
```

```
        # --- Вывод лучшего результата ---
        self.screen = QtWidgets.QLabel(self.centralwidget)
        self.screen.setGeometry(QtCore.QRect(510, 10, 300, 300))
        font = QtGui.QFont()
        font.setPointSize(20)
        font.setUnderline(True)
        self.screen.setFont(font)
        self.screen.setStyleSheet("border-color: rgb(220, 222, 221);\n"
                                  "background-color:    rgb(230,    231,
231);\n"
                                  "color: rgb(99, 99, 99);\n"
                                  "border-radius: 10px;")
        self.screen.setAlignment(QtCore.Qt.AlignCenter)
        self.screen.setObjectName("screen")

        # --- Таблица ---
        self.tableWidget = QtWidgets.QTableWidget(self.centralwidget)
        self.tableWidget.setGeometry(QtCore.QRect(310, 10, 190, 300))
        self.tableWidget.setStyleSheet("border-color:    rgb(220,    222,
221);\n"
                                       "background-color:  rgb(230,  231,
231);\n"
                                       "color: rgb(27, 28, 28);\n"
                                       "border-radius: 10px;")
        self.tableWidget.setColumnCount(2)
        self.tableWidget.setHorizontalHeaderLabels(["№", "Best fitness"])
        self.tableWidget.setColumnWidth(0, 60)
        self.tableWidget.horizontalHeader().setStretchLastSection(True)
        self.tableWidget.verticalHeader().setVisible(False)

self.tableWidget.setSelectionBehavior(QtWidgets.QAbstractItemView.SelectR
ows)

self.tableWidget.setEditTriggers(QtWidgets.QAbstractItemView.NoEditTrigge
rs)

        # --- График ---
        self.graph = QtWidgets.QWidget()
```

```
        self.graph.setParent(self.centralwidget)   # добавляем график на
центральный виджет!
        self.graph.setGeometry(QtCore.QRect(310, 320, 500, 330))
        self.graph.setStyleSheet("border-color: rgb(220, 222, 221);\n"
                                 "background-color:     rgb(230,     231,
231);\n"
                                 "color: rgb(27, 28, 28);\n"
                                 "border-radius: 10px;")
        self.graph_layout = QtWidgets.QVBoxLayout(self.graph)

        # --- Группа параметров ---
        self.groupBox = QtWidgets.QGroupBox(self.centralwidget)
        self.groupBox.setGeometry(QtCore.QRect(0, 0, 300, 660))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.groupBox.setFont(font)
        self.groupBox.setStyleSheet("background-color:    rgb(228,    229,
229);\n"
                                    "border-right-color:   rgb(177,   179,
179);\n"
                                    "color: rgb(48, 48, 48);")
        self.groupBox.setTitle("Параметры")
        self.groupBox.setMaximumSize(QtCore.QSize(300, 660))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.groupBox.setFont(font)
        self.groupBox.setStyleSheet("\n"
                                    "background-color:    rgb(228,    229,
229);\n"
                                    "border-right-color:   rgb(177,   179,
179);\n"
                                    "color: rgb(48, 48, 48);\n"
                                    "selection-color:    rgb(255,    255,
255);\n"
                                    "selection-background-color:  rgb(16,
81, 193);")
        self.groupBox.setAlignment(QtCore.Qt.AlignLeading                |
QtCore.Qt.AlignLeft | QtCore.Qt.AlignTop)
        self.groupBox.setObjectName("groupBox")
```

```
        # --- Кнопка запуска ---
        self.start_btn = QtWidgets.QPushButton(self.groupBox)
        self.start_btn.setGeometry(QtCore.QRect(72, 610, 100, 30))
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.start_btn.sizePolicy().hasHeightForWidt
h())
        self.start_btn.setSizePolicy(sizePolicy)
        self.start_btn.setMinimumSize(QtCore.QSize(100, 30))
        self.start_btn.setMaximumSize(QtCore.QSize(100, 30))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.start_btn.setFont(font)
        self.start_btn.setStyleSheet("""
            QPushButton {
                background-color: rgb(239, 240, 244);
                border-color: rgb(147, 147, 147);
                color: rgb(20, 21, 21);
                selection-color: rgb(255, 255, 255);
                selection-background-color: rgb(16, 81, 193);
                border-radius: 10px;
            }
            QPushButton:pressed {
                background-color: rgb(200, 200, 200); /* более тёмный
цвет для затемнения */
            }
        """)
        self.start_btn.setObjectName("start_btn")

        self.pushButton = QtWidgets.QPushButton(self.groupBox)
        self.pushButton.setGeometry(QtCore.QRect(178, 610, 50, 30))
        self.pushButton.setMinimumSize(QtCore.QSize(50, 30))
        self.pushButton.setMaximumSize(QtCore.QSize(50, 30))
        self.pushButton.setStyleSheet("""
            QPushButton {
```

```
                background-color: rgb(239, 240, 244);
                border-color: rgb(147, 147, 147);
                color: rgb(20, 21, 21);
                selection-color: rgb(255, 255, 255);
                selection-background-color: rgb(16, 81, 193);
                border-radius: 10px;
            }
            QPushButton:pressed {
                background-color: rgb(200, 200, 200); /* более тёмный
цвет для затемнения */
            }
        """)
        self.pushButton.setObjectName("pushButton")
        self.pushButton_2 = QtWidgets.QPushButton(self.groupBox)
        self.pushButton_2.setGeometry(QtCore.QRect(233, 610, 30, 30))
        self.pushButton_2.setMinimumSize(QtCore.QSize(30, 30))
        self.pushButton_2.setMaximumSize(QtCore.QSize(30, 30))
        font = QtGui.QFont()
        font.setPointSize(16)
        font.setBold(False)
        font.setWeight(50)
        self.pushButton_2.setFont(font)
        self.pushButton_2.setStyleSheet("""
            QPushButton {
                background-color: rgb(239, 240, 244);
                border-color: rgb(147, 147, 147);
                color: rgb(20, 21, 21);
                selection-color: rgb(255, 255, 255);
                selection-background-color: rgb(16, 81, 193);
                border-radius: 10px;
            }
            QPushButton:pressed {
                background-color: rgb(200, 200, 200); /* более тёмный
цвет для затемнения */
            }
        """)
        self.pushButton_2.setObjectName("pushButton_2")
        self.pushButton_3 = QtWidgets.QPushButton(self.groupBox)
        self.pushButton_3.setGeometry(QtCore.QRect(37, 610, 30, 30))
```

```
        self.pushButton_3.setMinimumSize(QtCore.QSize(30, 30))
        self.pushButton_3.setMaximumSize(QtCore.QSize(30, 30))
        font = QtGui.QFont()
        font.setPointSize(16)
        font.setBold(False)
        font.setWeight(50)
        self.pushButton_3.setFont(font)
        self.pushButton_3.setStyleSheet("""
            QPushButton {
                background-color: rgb(239, 240, 244);
                border-color: rgb(147, 147, 147);
                color: rgb(20, 21, 21);
                selection-color: rgb(255, 255, 255);
                selection-background-color: rgb(16, 81, 193);
                border-radius: 10px;
            }
            QPushButton:pressed {
                background-color: rgb(200, 200, 200); /* более тёмный
цвет для затемнения */
            }
        """)
        self.pushButton_3.setObjectName("pushButton_3")

        # --- Кнопка ---
        self.spin_mutation = QtWidgets.QDoubleSpinBox(self.groupBox)
        self.spin_mutation.setGeometry(QtCore.QRect(230, 460, 55, 24))
        self.spin_mutation.setMaximumSize(QtCore.QSize(55, 16777215))
        self.spin_mutation.setStyleSheet("background-color: rgb(239, 240,
244);\n"
                                         "border-color: rgb(147, 147,
147);\n"
                                         "color: rgb(20, 21, 21);\n"
                                         "selection-color: rgb(255, 255,
255);\n"
                                         "selection-background-color:
rgb(16, 81, 193);\n"
                                         "border-radius: 5px;")
        self.spin_mutation.setObjectName("spin_mutation")
```

```
        # --- Кнопка ---
        self.spin_crossover = QtWidgets.QDoubleSpinBox(self.groupBox)
        self.spin_crossover.setGeometry(QtCore.QRect(230, 420, 55, 24))
        self.spin_crossover.setMaximumSize(QtCore.QSize(55, 16777215))
        self.spin_crossover.setStyleSheet("background-color: rgb(239, 240,
244);\n"
                                          "border-color:   rgb(147,   147,
147);\n"
                                          "color: rgb(20, 21, 21);\n"
                                          "selection-color: rgb(255, 255,
255);\n"
                                          "selection-background-color:
rgb(16, 81, 193);\n"
                                          "border-radius: 5px;")
        self.spin_crossover.setObjectName("spin_crossover")

        # Настройки для spin_mutation
        self.spin_mutation.setDecimals(2)
        self.spin_mutation.setRange(0.01, 0.99)
        self.spin_mutation.setSingleStep(0.01)

        # Настройки для spin_crossover
        self.spin_crossover.setDecimals(2)
        self.spin_crossover.setRange(0.01, 0.99)
        self.spin_crossover.setSingleStep(0.01)

        self.label_mutation = QtWidgets.QLabel(self.groupBox)
        self.label_mutation.setGeometry(QtCore.QRect(20, 460, 200, 20))
        self.label_mutation.setMinimumSize(QtCore.QSize(200, 20))
        self.label_mutation.setMaximumSize(QtCore.QSize(200, 20))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.label_mutation.setFont(font)
        self.label_mutation.setStyleSheet("background-color:    rgba(255,
255, 255, 0);")
        self.label_mutation.setObjectName("label_mutation")

        self.label_crossover = QtWidgets.QLabel(self.groupBox)
        self.label_crossover.setGeometry(QtCore.QRect(20, 420, 200, 20))
```

```
        self.label_crossover.setMinimumSize(QtCore.QSize(200, 20))
        self.label_crossover.setMaximumSize(QtCore.QSize(200, 20))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.label_crossover.setFont(font)
        self.label_crossover.setStyleSheet("background-color:   rgba(255,
255, 255, 0);")
        self.label_crossover.setObjectName("label_crossover")

        self.label_population = QtWidgets.QLabel(self.groupBox)
        self.label_population.setGeometry(QtCore.QRect(20, 340, 200, 20))
        self.label_population.setMinimumSize(QtCore.QSize(200, 20))
        self.label_population.setMaximumSize(QtCore.QSize(200, 20))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.label_population.setFont(font)
        self.label_population.setStyleSheet("background-color:  rgba(255,
255, 255, 0);")
        self.label_population.setObjectName("label_population")

        self.label_max = QtWidgets.QLabel(self.groupBox)
        self.label_max.setGeometry(QtCore.QRect(20, 300, 200, 20))
        self.label_max.setMinimumSize(QtCore.QSize(200, 20))
        self.label_max.setMaximumSize(QtCore.QSize(200, 20))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.label_max.setFont(font)
        self.label_max.setStyleSheet("background-color:   rgba(255,   255,
255, 0);")
        self.label_max.setObjectName("label_max")

        self.enter_population_size = QtWidgets.QLineEdit(self.groupBox)
        self.enter_population_size.setGeometry(QtCore.QRect(240, 340, 40,
21))
        self.enter_population_size.setMaximumSize(QtCore.QSize(40,
16777215))
        self.enter_population_size.setStyleSheet("background-color:
rgb(239, 240, 244);\n"
```

```
                                                "border-color:  rgb(147,
147, 147);\n"
                                                "color:    rgb(20,    21,
21);\n"
                                                "selection-color:
rgb(255, 255, 255);\n"
                                                "selection-background-
color: rgb(16, 81, 193);\n"
                                                "border-radius: 5px;")
        self.enter_population_size.setObjectName("enter_population_size")

        self.enter_max = QtWidgets.QLineEdit(self.groupBox)
        self.enter_max.setGeometry(QtCore.QRect(240, 300, 40, 21))
        self.enter_max.setMaximumSize(QtCore.QSize(40, 16777215))
        self.enter_max.setStyleSheet("background-color:   rgb(239,   240,
244);\n"
                                     "border-color:     rgb(147,     147,
147);\n"
                                     "color: rgb(20, 21, 21);\n"
                                     "selection-color:    rgb(255,    255,
255);\n"
                                     "selection-background-color: rgb(16,
81, 193);\n"
                                     "border-radius: 5px;")
        self.enter_max.setObjectName("enter_max")

        self.label_population_2 = QtWidgets.QLabel(self.groupBox)
        self.label_population_2.setGeometry(QtCore.QRect(20,   380,   200,
20))
        self.label_population_2.setMinimumSize(QtCore.QSize(200, 20))
        self.label_population_2.setMaximumSize(QtCore.QSize(200, 20))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.label_population_2.setFont(font)
        self.label_population_2.setStyleSheet("background-color: rgba(255,
255, 255, 0);")
        self.label_population_2.setObjectName("label_population_2")
        self.enter_population_size_2 = QtWidgets.QLineEdit(self.groupBox)
```

```
        self.enter_population_size_2.setGeometry(QtCore.QRect(240,   380,
40, 21))
        self.enter_population_size_2.setMaximumSize(QtCore.QSize(40,
16777215))
        self.enter_population_size_2.setStyleSheet("background-color:
rgb(239, 240, 244);\n"
                                                   "border-color: rgb(147,
147, 147);\n"
                                                   "color:   rgb(20,   21,
21);\n"
                                                   "selection-color:
rgb(255, 255, 255);\n"
                                                   "selection-background-
color: rgb(16, 81, 193);\n"
                                                   "border-radius: 5px;")

self.enter_population_size_2.setObjectName("enter_population_size_2")

        # Ограничение ввода только чисел от 0 до 10000
        int_validator = QtGui.QIntValidator(0, 10000)
        self.enter_population_size.setValidator(int_validator)
        self.enter_max.setValidator(int_validator)
        self.enter_population_size_2.setValidator(int_validator)

        self.tablescreen = QtWidgets.QTableWidget(self.groupBox)
        self.tablescreen.setGeometry(QtCore.QRect(40, 60, 220, 220))
        self.tablescreen.setMinimumSize(QtCore.QSize(220, 220))
        self.tablescreen.setMaximumSize(QtCore.QSize(220, 220))
        self.tablescreen.setObjectName("tablescreen")
        rows, cols = 9, 9
        self.tablescreen.setRowCount(rows)
        self.tablescreen.setColumnCount(cols)
        cell_size = 24
        for i in range(rows):
            self.tablescreen.setRowHeight(i, cell_size)
        for j in range(cols):
            self.tablescreen.setColumnWidth(j, cell_size)
        for i in range(rows):
            for j in range(cols):
```

```
                item = QtWidgets.QTableWidgetItem("x")
                item.setTextAlignment(QtCore.Qt.AlignCenter)
                self.tablescreen.setItem(i, j, item)
        self.tablescreen.horizontalHeader().setVisible(False)
        self.tablescreen.verticalHeader().setVisible(False)

self.tablescreen.setSelectionBehavior(QtWidgets.QAbstractItemView.SelectI
tems)

        self.label_f = QtWidgets.QLabel(self.groupBox)
        self.label_f.setGeometry(QtCore.QRect(90, 30, 110, 16))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.label_f.setFont(font)
        self.label_f.setStyleSheet("background-color: rgba(255, 255, 255,
0);\n"
                                   "color: rgb(48, 48, 48);")
        self.label_f.setAlignment(QtCore.Qt.AlignCenter)
        self.label_f.setObjectName("label_f")

        self.error_label = QtWidgets.QLabel(self.groupBox)
        self.error_label.setGeometry(QtCore.QRect(0, 570, 300, 40))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.error_label.setFont(font)
        self.error_label.setStyleSheet("background-color: rgba(255, 255,
255, 0);")
        self.error_label.setAlignment(QtCore.Qt.AlignCenter)
        self.error_label.setObjectName("error_label")

        # --- Подпись к результату ---
        self.label      =      QtWidgets.QLabel("Лучший      результат",
self.centralwidget)
        self.label.setGeometry(QtCore.QRect(600, 30, 121, 16))
        self.label.setStyleSheet("background-color: rgb(0, 0, 0, 0);")
        font = QtGui.QFont()
        font.setPointSize(13)
        self.label.setFont(font)
        self.label.setObjectName("label")
```

```
        MainWindow.setCentralWidget(self.centralwidget)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

        # Подключение обработчиков
        self.functions()

    # --- Обработчик выбора строки ---
    def on_row_clicked(self):
        row = self.tableWidget.currentRow()
        key = self.tableWidget.item(row, 0).text()
        label_text                                                    =
format_9x9_square(str_to_field((self.data.get(key))[1]))
        self.screen.setText(label_text)
        print(self.data.get('best_fitness')[:int(key)])
        self.plot_graph(self.data.get('best_fitness')[:int(key)])

    # --- Обновить таблицу ---
    def update_table(self):
        self.data = read_data()
        keys = list(self.data.keys())[1:]  # пропустить первый элемент
        self.tableWidget.setRowCount(len(keys))
        for row, key in enumerate(keys):
            value = self.data[key]
            item1 = QtWidgets.QTableWidgetItem(str(key))
            item1.setTextAlignment(QtCore.Qt.AlignCenter)
            item2  =  QtWidgets.QTableWidgetItem(str(value[0])  if  value
else "")
            item2.setTextAlignment(QtCore.Qt.AlignCenter)
            self.tableWidget.setItem(row, 0, item1)
            self.tableWidget.setItem(row, 1, item2)

    # --- Отрисовщик графика ---
    def plot_graph(self, fitness_values):
        if hasattr(self, 'canvas'):
            self.graph_layout.removeWidget(self.canvas)
            self.canvas.setParent(None)
```

```
        figure = Figure(figsize=(10, 5), tight_layout=True)
        self.canvas = FigureCanvas(figure)
        ax = figure.add_subplot(111)
        ax.plot(fitness_values, label='Best Fitness')
        ax.set_xlabel("Generation")
        ax.set_ylabel("Fitness")
        ax.set_title("Best Fitness per Generation")
        ax.legend()
        ax.grid(True)

        self.graph_layout.addWidget(self.canvas)

    def table_to_array(self, table: QtWidgets.QTableWidget) -> list:
        rows = table.rowCount()
        cols = table.columnCount()
        data = []

        for i in range(rows):
            row_data = []
            for j in range(cols):
                item = table.item(i, j)
                value = item.text() if item else ''
                row_data.append(value)
            data.append(row_data)

        return data

    # --- Перевод интерфейса ---
    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "ГА Судоку"))
        self.groupBox.setTitle(_translate("MainWindow", "Параметры"))
        self.start_btn.setText(_translate("MainWindow", "Запуск"))
        self.label_mutation.setText(_translate("MainWindow", "Вероятность
мутации"))
        self.label_crossover.setText(_translate("MainWindow",
"Вероятность скрещивания"))
        self.label_population.setText(_translate("MainWindow",    "Размер
популяции"))
```

```
        self.label_max.setText(_translate("MainWindow", "Макс. кол-во поколений"))
        self.label_f.setText(_translate("MainWindow", "Начальное поле"))
        self.error_label.setText(_translate("MainWindow", ""))
        self.label.setText(_translate("MainWindow", "Лучший результат"))
        self.pushButton.setText(_translate("MainWindow", "Стоп"))
        self.pushButton_2.setText(_translate("MainWindow", "▷"))
        self.label_population_2.setText(_translate("MainWindow", "Кол-во случайных клеток"))
        self.pushButton_3.setText(_translate("MainWindow", "◁"))

    # --- Обработчик функций ---
    def functions(self):
        self.tableWidget.clicked.connect(lambda: self.on_row_clicked())
        self.start_btn.clicked.connect(lambda: self.start())

    def start(self):
        print("Программа запущена")
        array = self.table_to_array(self.tablescreen)
        main_start(array, int(self.enter_population_size.text()), int(self.enter_max.text()), float(self.spin_mutation.text().replace(',', '.')))
        self.update_table()

if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

Название файла: sudoku_field.py

```
import numpy as np
"""
Получение начальной расстановки и создание популяции
"""
class FieldCreator():
    def __init__(self):
        self.main_permutation = list(range(1, 10)) * 9
        self.insert_list_indexes = []
        self.insert_list_symbols = []
    def ReadFromFile(self, file_name: str) -> None:
        with open(file_name, 'r') as file:
            file_field    =    [item.split('    ')    for    item    in
file.read().split('\n')]

        for x in range(9):
            for y in range(9):
                symbol = file_field[x][y]
                if symbol != 'x':
                    self.insert_list_indexes.append((x, y))
                    self.insert_list_symbols.append(int(symbol))
                    self.main_permutation.remove(int(symbol))

    def ReadFromList(self, arr: list[list[str]]) -> None:
        for x in range(9):
            for y in range(9):
                symbol = arr[x][y]
                if symbol != 'x':
                    self.insert_list_indexes.append((x, y))
                    self.insert_list_symbols.append(int(symbol))
                    self.main_permutation.remove(int(symbol))


    def GeneratePopulation(self, entities_amount: int) -> list:
        population = []
        for _ in range(entities_amount):
            new_entity                                                  =
np.random.permutation(self.main_permutation).tolist()
```

```
            for index in range(len(self.insert_list_symbols)):
                new_entity.insert(self.insert_list_indexes[index][0] * 9
+ self.insert_list_indexes[index][1], self.insert_list_symbols[index])
            new_entity = [new_entity[i:i + 9] for i in range(0, 81, 9)]
            population.append(new_entity)

        return population



def main():
    field_creator = FieldCreator()
    field_creator.ReadFromFile('example.txt')
    print(field_creator.insert_list)
    population = field_creator.GeneratePopulation(5)
    for item in population:
        print('\n'.join(['  '.join(list(map(str,  item[i])))  for  i  in
range(9)]) + '\n')



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

Название файла: mutation.py

```
import random
"""
Мутация 2 рандомных клеток в одной рандомной строке
Мутация 2 рандомных клеток в одном рандомном столбце
Мутация 2 абсолютно рандомных клеток
"""


def random_mutation(entity: list, insert_list_indexes: list) -> None:
    numbers = list(range(0, 81))
    for item in insert_list_indexes:
        numbers.remove(item[0] * 9 + item[1])

    first_sum = random.choice(numbers)
```

```
    numbers.remove(first_sum)
    second_sum = random.choice(numbers)

    first_x = first_sum // 9
    first_y = first_sum % 9

    second_x = second_sum // 9
    second_y = second_sum % 9

    sym = entity[first_x][first_y]
    entity[first_x][first_y] = entity[second_x][second_y]
    entity[second_x][second_y] = sym

def  mutation_among_bad_rowsncolumns(entity:  list,  insert_list_indexes:
list, bad_rows_columns_indexes: list, row_column_flag: bool) -> None:
    numbers = []
    if row_column_flag == True:
        for item in bad_rows_columns_indexes:
            numbers += list(range(item * 9, item * 9 + 9))

        for item in insert_list_indexes:
            if item[0] in bad_rows_columns_indexes:
                numbers.remove(item[0] * 9 + item[1])
    else:
        for item in bad_rows_columns_indexes:
            numbers += list(range(item, 81, 9))

        for item in insert_list_indexes:
            if item[1] in bad_rows_columns_indexes:
                numbers.remove(item[0] * 9 + item[1])

    first_sum = random.choice(numbers)
    numbers.remove(first_sum)
    second_sum = random.choice(numbers)

    first_x = first_sum // 9
    first_y = first_sum % 9

    second_x = second_sum // 9
```

```
    second_y = second_sum % 9

    # sym = entity[first_x][first_y]
    # entity[first_x][first_y] = entity[second_x][second_y]
    # entity[second_x][second_y] = sym

    entity[first_x][first_y],        entity[second_x][second_y]        =
entity[second_x][second_y], entity[first_x][first_y]
```

Название файла: main.py

```
from PyQt5.QtWidgets import QApplication, QMainWindow
from PyQt5 import QtWidgets
import random
import sys
from sudoku_field import *
from mutation import *
import matplotlib.pyplot as plt
from data_saver import *

test = [
    [5, 9, 3, 8, 1, 9, 6, 7, 2],
[8, 2, 4, 6, 3, 7, 5, 1, 4],
[6, 1, 7, 4, 5, 2, 8, 3, 9],
[1, 3, 5, 7, 2, 8, 4, 9, 6],
[4, 8, 9, 5, 6, 3, 7, 2, 1],
[7, 6, 2, 1, 9, 4, 3, 8, 5],
[9, 4, 6, 3, 7, 1, 2, 5, 8],
[3, 5, 1, 2, 8, 6, 9, 4, 7],
[2, 7, 8, 9, 4, 5, 1, 6, 3],
]


# ========================== print individual
def print_ind(ind):
    print('\n'.join([' '.join(list(map(str, ind[i]))) for i in range(9)])
+ '\n')
# ========================== print individual

# ========================== fitness calculation
```

```
def fitness_full(individual):
    amount = 0
    for line in individual:
        amount += len(set(line))

    for column in zip(*individual):
        amount += len(set(column))

    for i in range(0, 9, 3):
        for j in range(0, 9, 3):
            square = [individual[x][y] for x in range(i, i+3) for y in
range(j, j+3)]

            amount += len(set(square))

    return amount

def fitness_cut(individual):
    amount = 0
    for line in individual:
        if len(set(line)) == 9:
            amount += 1

    for column in zip(*individual):
        if len(set(column)) == 9:
            amount += 1

    for i in range(0, 9, 3):
        for j in range(0, 9, 3):
            square = [individual[x][y] for x in range(i, i+3) for y in
range(j, j+3)]

            if len(set(square)) == 9:
                amount += 1

    return amount

# ========================== fitness calculation
```

```
# ========================== selection
def group_tournament_selection(population, k=2):
    random.shuffle(population)
    best_individuals = []

    for i in range(0, len(population), k):
        group = population[i:i + k]
        best_in_group = max(group, key=fitness_full)
        best_individuals.append(best_in_group)

    return best_individuals
# ========================== selection

# ========================== crossing
def one_point_crossing_sq(parent1, parent2, fixed_positions):
    child = [row.copy() for row in parent1]

    num_squares_to_swap = random.randint(1, 7)

    all_squares = [(i, j) for i in range(0, 9, 3) for j in range(0, 9, 3)]

    squares_to_swap = random.sample(all_squares, num_squares_to_swap)

    for block_row, block_col in squares_to_swap:
        for i in range(block_row, block_row + 3):
            for j in range(block_col, block_col + 3):

                if (i, j) not in fixed_positions:
                    child[i][j] = parent2[i][j]

    return child

def uniform_crossover_sq(parent1, parent2, fixed_positions):
    child = [[0 for _ in range(9)] for _ in range(9)]

    for block_row in range(0, 9, 3):
        for block_col in range(0, 9, 3):
            source = parent1 if random.random() < 0.55 else parent2
```

```
            for i in range(3):
                for j in range(3):
                    x, y = block_row + i, block_col + j
                    if (x, y) in fixed_positions:
                        # Сохраняем фиксированную ячейку
                        child[x][y] = parent1[x][y]
                    else:
                        child[x][y] = source[x][y]

    return child
# ========================== crossing
# ========================== mutation data
def get_bad_rows(individual):
    low_fitness_rows = []

    for i in range(9):
        row_fitness = 0
        for j in range(9):
            val = individual[i][j]

            fitness = 0

            if individual[i].count(val) == 1:
                fitness += 1

            column = [individual[x][j] for x in range(9)]
            if column.count(val) == 1:
                fitness += 1

            block_i = (i // 3) * 3
            block_j = (j // 3) * 3
            block = [individual[x][y] for x in range(block_i, block_i + 3)
                                      for y in range(block_j, block_j + 3)]
            if block.count(val) == 1:
                fitness += 1

            row_fitness += fitness
```

```
        if row_fitness < 27:
            low_fitness_rows.append(i)

    return low_fitness_rows

def get_bad_columns(individual):
    columns_indexes = []
    for j in range(9):
        column = []
        for i in range(len(individual)):
            column.append(individual[i][j])
        # if len(set(column)) < 9:
        print(column)
            # columns_indexes.append(j)

    # return columns_indexes

# ========================== mutation data

def  genetic_algorithm(population,  fixed_positions,  generations=10000,
population_size=100, mutation_rate=0.55):
    best_fitness_values = []
    data_init()

    for generation in range(generations):
        population = sorted(population, key=fitness_full, reverse=True)
        best = population[0]
        best_fitness = fitness_full(best)
        best_fitness_values.append(best_fitness)

        # Сохранение данных о поколении
        data = [best_fitness, field_to_str(best)]
        save_data(generation, data)

        print(f"Generation {generation}, Best fitness: {best_fitness}")
        if best_fitness == 243:
            print("Sudoku solved!")
            plot_progress(best_fitness_values)
            return best
```

```
        selected = group_tournament_selection(population)

        next_generation = []
        while len(next_generation) < population_size:
            parent1, parent2 = random.sample(selected, 2)
            child = one_point_crossing_sq(parent1, parent2,
fixed_positions)

            if random.random() < mutation_rate:
                # if len(get_bad_rows(child)) > 0:
                #         mutation_among_bad_rows(child, fixed_positions,
get_bad_rows(child), True)
                # else:
                    random_mutation(child, fixed_positions)

            next_generation.append(child)

        population = next_generation

    print("Max generations reached.")
    plot_progress(best_fitness_values)
    return max(population, key=fitness_full)

# Функция построения графика
def plot_progress(fitness_values):
    plt.figure(figsize=(10, 5))
    plt.plot(fitness_values, label='Best Fitness')
    plt.xlabel("Generation")
    plt.ylabel("Fitness")
    plt.title("Best Fitness per Generation")
    plt.legend()
    plt.grid(True)
    plt.show()

def main_start(field: list[list[str]], population_size: int,
generation_size: int, p_mutation: float):
    field_creator = FieldCreator()
    field_creator.ReadFromList(field)
```

```
    population = field_creator.GeneratePopulation(population_size)
    solution = genetic_algorithm(population, field_creator.insert_list_indexes, generation_size, population_size, p_mutation)

if __name__ == "__main__":
    field_creator = FieldCreator()
    field_creator.ReadFromFile('example.txt')
    # # print(field_creator.insert_list)
    population = field_creator.GeneratePopulation(500)

    # print(get_bad_rows(test))

    # for ind in population:
    #     print(fitness_full(ind))

    #         one_point_crossing_sq(population[0], population[1], field_creator.insert_list_indexes)

    # for item in population:
    #       print('\n'.join([' '.join(list(map(str, item[i]))) for i in range(9)]) + '\n')
    # print(fitness_full(test))

    solution = genetic_algorithm(population, field_creator.insert_list_indexes, 10000, 500)
    print(fitness_full(solution))
    for row in solution:
        print(row)
```