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

ОТЧЕТ

по летней практике, итерация №4

Тема: Генетические Алгоритмы “Решение задачи судоку”

Студенты гр. 3342 Галеев А.Д; Романов Е.А; Хайруллов Д.Л;

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 четвертой итерации нашего проекта мы получили рабочую версию GUI связанную с генетическим алгоритмом. Все необходимые функции были разработаны, и связаны между собой, что позволило алгоритму корректно работать.

Вклад участников:

Романов Егор – были написаны новые функции скрещивания и мутаций, которые показали более высокую эффективность относительно старых реализаций. Была модифицирована логика формирования новых поколений.

Хайруллов Динар – был добавлен новый метод “сильной мутации”, который кардинально меняет расстановку у сущности, но при этом такая мутация срабатывает с очень малой вероятностью, с большей вероятностью все еще срабатывает старая мутация перестановки двух клеток. Добавлены новые функции скрещивания по столбцам и строкам. В сочетании со старым методом скрещивания по квадратам, скрещивания выполняется по одному из трех методов с вероятностью 1/3, что в теории должно улучшить разнообразие популяции. Осуществлен небольшой рефакторинг кода.

Галеев Амир – были написаны все проверки входящих данных с выводом ошибок и всеми необходимыми проверяющими функциями, GUI интерфейс – полностью связан с алгоритмом, были написаны функции для выполнения алгоритма по шагам (один шаг, до конца алгоритма), была написана функция для чтения поля из файла, скорее всего устранены все ошибки в интерфейсе.

Рис. 1 – кнопка загрузить файл

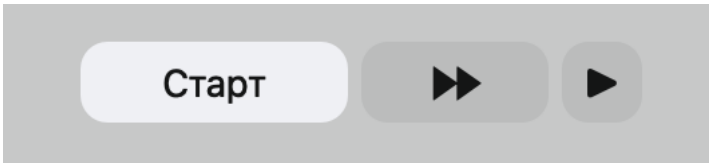


Рис. 2 – кнопки выполнения по шагам

Описание функций и структур данных.

GUI

Класс UiMainWindow – отвечает за внешний вид окна интерфейса, в нем расположены все кнопки поля и таблицы для взаимодействия с алгоритмом.

on_row_clicked - Обрабатывает клик по строке таблицы с результатами: Загружает соответствующее решение судоку на экран**.** Отображает график до выбранного поколения

update_table - Обновляет таблицу с результатами (tableWidget) из файла данных (self.data), добавляя поколения и их fitness-значения.

update_screen - Отображает переданный массив 9×9 на экран-таблицу tablescreen.

plot_graph - Отрисовывает график fitness-значений по поколениям на matplotlib-графике.

table_to_array - Считывает содержимое QTableWidget и возвращает его как список списков строк.

open_txt_file - Открывает .txt файл, проверяет его формат и, если корректен, загружает в таблицу tablescreen. В случае ошибки показывает сообщение.

table_check - Проверяет, что все ячейки 9×9 содержат цифры или символ x.

file_check - Проверяет, что файл содержит ровно 9 строк по 9 элементов, разделённых пробелами.

Functions - Подключает сигналы (нажатия кнопок и строки таблицы) к соответствующим функциям (on_row_clicked, start, и др.).

Start - Обрабатывает нажатие кнопки “Старт/Стоп”:

- При запуске: инициализирует алгоритм, проверяет поля, блокирует ввод
- При остановке: очищает данные, разблокирует ввод, сбрасывает кнопки

start_one - Выполняет один шаг алгоритма:

- Запускает одну итерацию генетического алгоритма
- Обновляет таблицу
- Проверяет, решена ли задача (если fitness = 243)

start_until_the_end - Запускает итерации алгоритма до максимального поколения или пока не будет решено судоку:

- Останавливается при достижении fitness = 243
- Обновляет таблицу

Генетический алгоритм

Класс GeneticAlgorithm:

Реализует генетический алгоритм для решения судоку. Основные функции:

Генерация начальной популяции возможных решений

Оценка качества решений с помощью фитнес-функций

Эволюция решений через селекцию, скрещивание и мутацию

Сохранение и визуализация результатов

Методы класса:

1. Инициализация и загрузка данных

__init__() - Инициализирует пустую популяцию и структуры для хранения судоку

get_data(data: Union[list, str], flag: str) - Загружает начальное состояние судоку из файла (flag='f') или списка (flag='l')

2. Генерация популяции

GeneratePopulation(entities_amount: int) - Создает начальную популяцию из entities_amount случайных решений с учетом фиксированных клеток

3. Оценка решений (фитнес-функции)

fitness_full(individual: list[list[int]]) - Полная оценка качества (учитывает строки, столбцы и блоки 3x3)

fitness_cut(individual: list[list[int]]) - Упрощенная оценка (считает только совершенные строки/столбцы/блоки)

4. Селекция

group_tournament_selection(k: int = 2) - Турнирный отбор с размером группы k

5. Скрещивание

one_point_crossing_sq(parent1: list[list[int]], parent2: list[list[int]]) - Обмен случайными блоками 3x3

uniform_crossover_sq(parent1: list[list[int]], parent2: list[list[int]]) - Равномерное скрещивание по блокам

uniform_crossover_cell(parent1: list[list[int]], parent2: list[list[int]]) - Равномерное скрещивание по клеткам

uniform_crossover_row(parent1: list[list[int]], parent2: list[list[int]]) - Скрещивание целых строк

uniform_crossover_column(parent1: list[list[int]], parent2: list[list[int]]) - Скрещивание целых столбцов

6. Мутации

random_mutation(entity: list[list[int]]) - Случайная перестановка двух клеток

row_shuffle_mutation(entity: list[list[int]]) - Перемешивание значений в строке

mutation_change_5_percent(entity: list[list[int]]) - Изменение 5% клеток

very_random_mutation() - Полная перегенерация особи

7. Основной алгоритм

main_cycle(generations: int, population_size: int, mutation_rate: float) - Главный цикл эволюции;

one_iteration(population_size: int, mutation_rate: float, generation: int) - Одна итерация эволюции

8. Вспомогательные методы

print_ind(ind: list[list[int]]) - Красивый вывод судоку

calculate_population_diversity(population: list[list[list[int]]]) – Оценка разнообразия популяции

Дополнительные функции

def plot_progress(fitness_values) – Создание графика зависимости лучшей приспособленности от популяции

ReadFromFile(file_name: str) -> None – Чтение начального поля из файла

ReadFromList(arr: list[list[str]]) -> None – Чтение начального поля из массива

field_to_str(field: list[list[int]]) -> str - Преобразует поле судоку 9x9 в строку

str_to_field(string: str) -> list[list[int]] - Восстанавливает поле судоку из строки

format_9x9_square(arr) - Форматирует массив 9x9 для красивого вывода

save_data(id: int, data_array: list) - Сохраняет данные поколения в файл

read_data() - Читает все сохраненные данные из файла

data_init() - Инициализирует файл данных

clean_data() - Очищает файл данных

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:

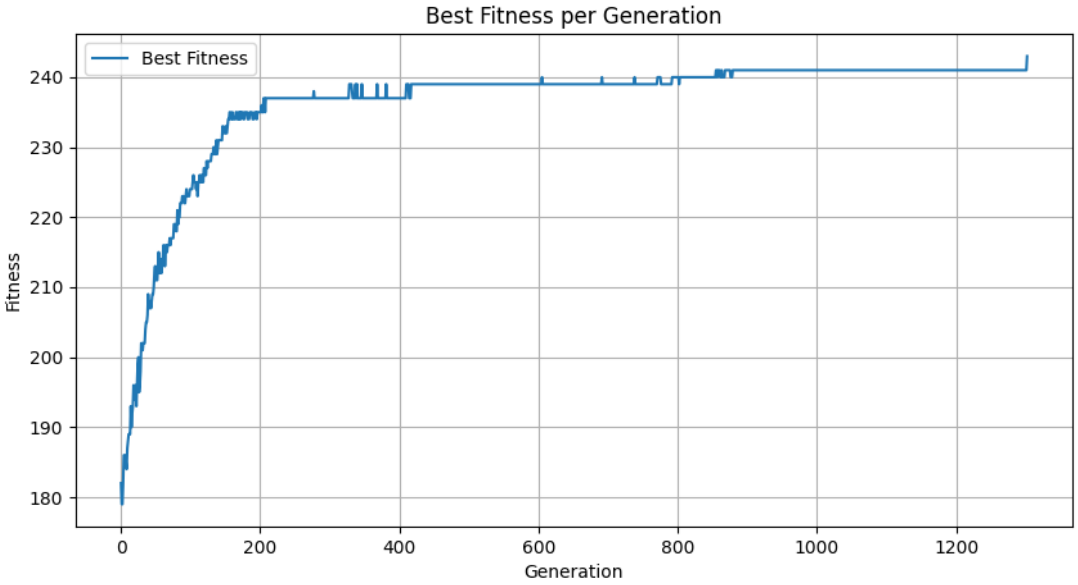


Рис. 3 — график зависимости приспособленности от популяции.

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]

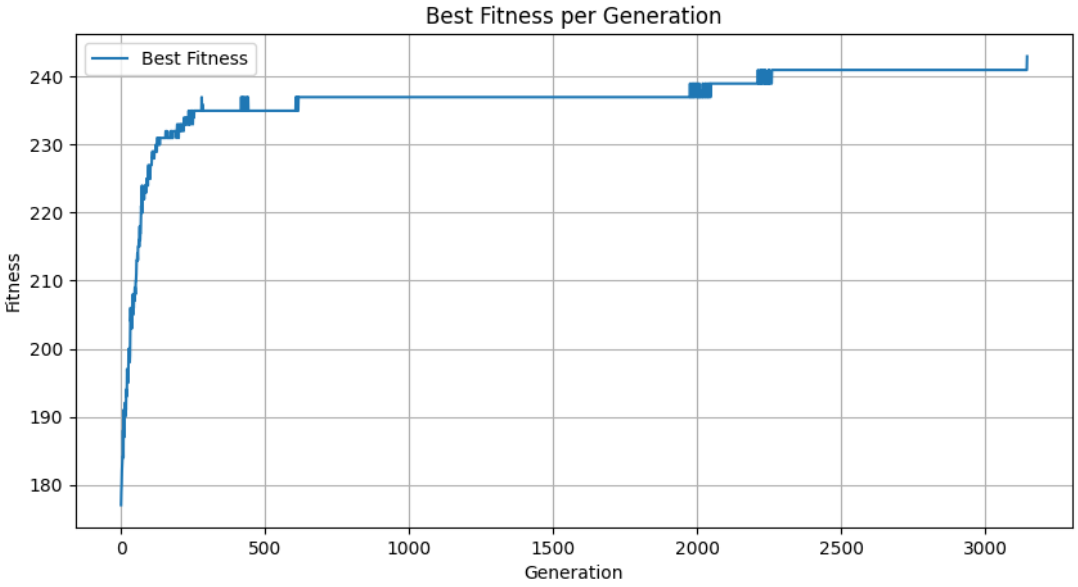
[3, 4, 1, 5, 9, 6, 7, 2, 8]

[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. 4 — график зависимости приспособленности от популяции.

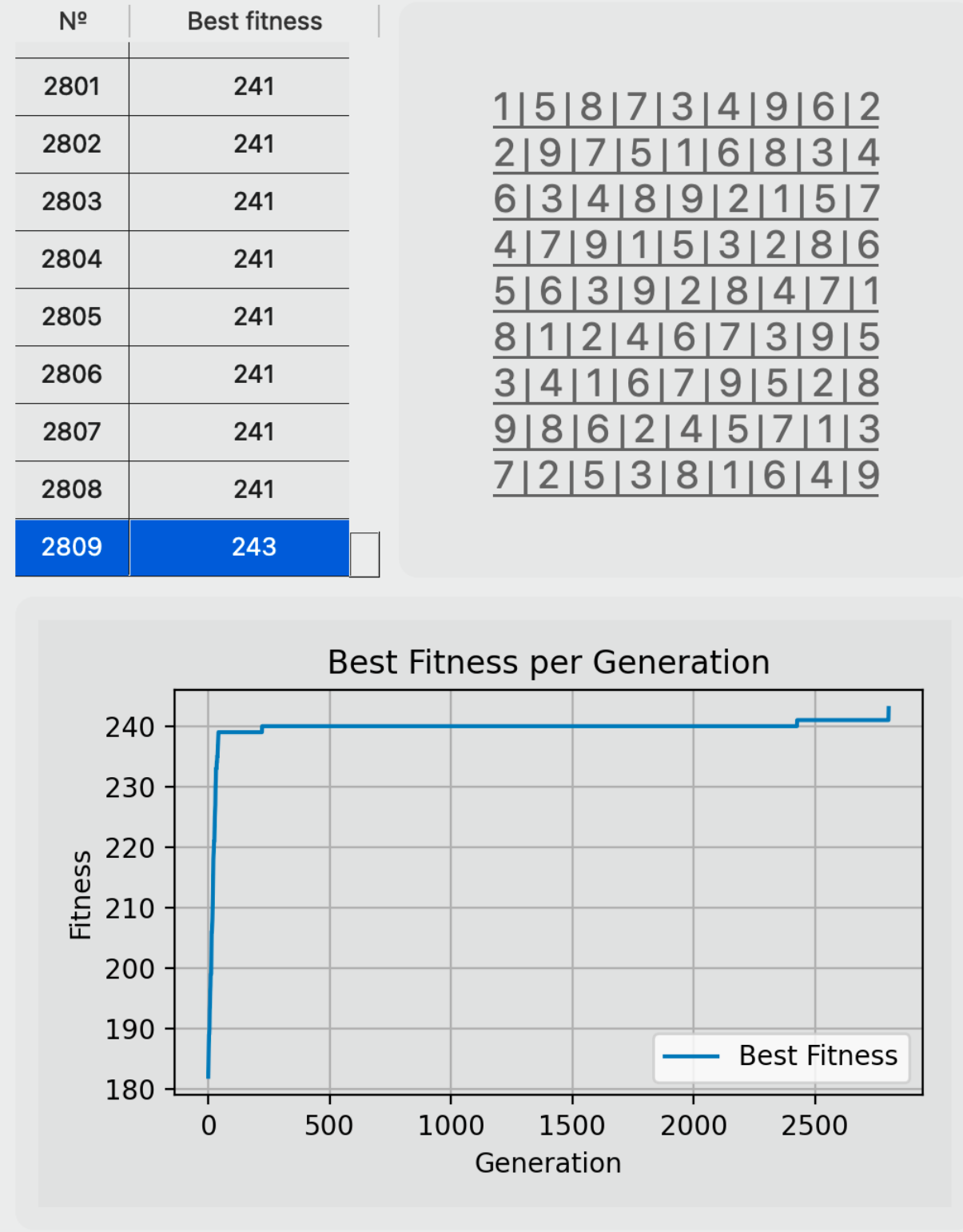


Рис. 5 — Интерфейс вывода результатов

Вывод.

На данной итерации был доработан основной генетический алгоритм, также разработана финальная версия графического интерфейса, которая в дальнейшем может претерпеть лишь небольшие изменения.

Графический интерфейс уже связан с работой функций и работает исправно, сам генетический алгоритм в большинстве случаев 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
from main import *


class UiMainWindow(object):
    def __init__(self):
        self.i = 0
        self.best_fitness_values = []
        self.field_creator = FieldCreator()
        self.population_size = 0
        self.population = self.field_creator.GeneratePopulation(10)
        self.fixed_positions = 0
        self.generations = 0
        self.p_mutation = 0
        self.is_start = False

    def setup_ui(self, MainWindow):

        # ======================================
        #   Настройки окна
        # ======================================
        self.MainWindow = MainWindow
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(820, 660)
        sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)

sizePolicy.setHeightForWidth(MainWindow.sizePolicy().hasHeightForWid
th())
        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)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
```

```
        self.centralwidget.setObjectName("centralwidget")
        MainWindow.setStyleSheet("background-color: rgb(235, 236,
236);\n"
                                 "color: rgb(48, 48, 48);")

        # ======================================
        #   Окно вывода лучшего результата
        # ======================================
        self.label = QtWidgets.QLabel("Лучший результат",
self.centralwidget)
        self.label.setGeometry(QtCore.QRect(600, 30, 121, 16))
        font = QtGui.QFont()
        font.setPointSize(13)
        self.label.setFont(font)
        self.label.setObjectName("label")
        self.label.setStyleSheet("background-color: rgb(255, 255,
255, 0);")

        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.setAlignment(QtCore.Qt.AlignCenter)
        self.screen.setObjectName("screen")
        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;")

        # ======================================
        #   Таблица результатов
        # ======================================
        self.tableWidget = QtWidgets.QTableWidget(self.centralwidget)
        self.tableWidget.setGeometry(QtCore.QRect(310, 10, 190, 300))
        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.Se
lectRows)

self.tableWidget.setEditTriggers(QtWidgets.QAbstractItemView.NoEditT
riggers)
        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;")

        # ========================================
        #   Г р а ф и к
        # ========================================
        self.graph = QtWidgets.QWidget()
        self.graph.setParent(self.centralwidget)
        self.graph.setGeometry(QtCore.QRect(310, 320, 500, 330))
        self.graph_layout = QtWidgets.QVBoxLayout(self.graph)
        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;")

        # ========================================
        #   Г р у п п а  п а р а м е т р о в
        # ========================================
        self.groupBox = QtWidgets.QGroupBox(self.centralwidget)
        self.groupBox.setGeometry(QtCore.QRect(0, 0, 300, 660))
        self.groupBox.setMinimumSize(QtCore.QSize(300, 660))
        self.groupBox.setMaximumSize(QtCore.QSize(300, 660))
        self.groupBox.setFont(font)
        font = QtGui.QFont()
        font.setPointSize(14)
        self.groupBox.setFont(font)
        self.groupBox.setAlignment(QtCore.Qt.AlignLeading |
QtCore.Qt.AlignLeft | QtCore.Qt.AlignTop)
        self.groupBox.setObjectName("groupBox")
        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);")

        # ======================================
        #   К н о п к а  з а п у с к а
        # ======================================
        self.start_btn = QtWidgets.QPushButton(self.groupBox)
        self.start_btn.setGeometry(QtCore.QRect(45, 610, 100, 30))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.start_btn.setFont(font)
        self.start_btn.setObjectName("start_btn")
        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)}")

        # ======================================
        #   К н о п к а  з а г р у з и т ь
        # ======================================
        self.download_btn = QtWidgets.QPushButton(self.groupBox)
        self.download_btn.setGeometry(QtCore.QRect(90, 290, 120, 20))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.download_btn.setFont(font)
        self.download_btn.setObjectName("download_btn")
        self.download_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)}")

        # ======================================
```

```
        #   В е р о я т н о с т ь  м у т а ц и и
        # ======================================
        self.label_mutation = QtWidgets.QLabel(self.groupBox)
        self.label_mutation.setGeometry(QtCore.QRect(20, 490, 200,
20))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.label_mutation.setFont(font)
        self.label_mutation.setObjectName("label_mutation")
        self.label_mutation.setStyleSheet("background-color:
rgba(255, 255, 255, 0);")

        self.spin_mutation = QtWidgets.QDoubleSpinBox(self.groupBox)
        self.spin_mutation.setGeometry(QtCore.QRect(230, 490, 55,
24))
        self.spin_mutation.setObjectName("spin_mutation")
        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.setDecimals(2)
        self.spin_mutation.setRange(0.01, 0.99)
        self.spin_mutation.setSingleStep(0.01)

        # ======================================
        #   В е р о я т н о с т ь  с к р е щ и в а н и я
        # ======================================
        self.label_crossover = QtWidgets.QLabel(self.groupBox)
        self.label_crossover.setGeometry(QtCore.QRect(20, 450, 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.setObjectName("label_crossover")
        self.label_crossover.setStyleSheet("background-color:
rgba(255, 255, 255, 0);")
```

```
        self.spin_crossover = QtWidgets.QDoubleSpinBox(self.groupBox)
        self.spin_crossover.setGeometry(QtCore.QRect(230, 450, 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")

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

        # ======================================
        #   Размер популяции
        # ======================================
        self.label_population = QtWidgets.QLabel(self.groupBox)
        self.label_population.setGeometry(QtCore.QRect(20, 370, 200,
20))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.label_population.setFont(font)
        self.label_population.setObjectName("label_population")
        self.label_population.setStyleSheet("background-color:
rgba(255, 255, 255, 0);")

        self.enter_population_size =
QtWidgets.QLineEdit(self.groupBox)
        self.enter_population_size.setGeometry(QtCore.QRect(240, 370,
40, 21))

self.enter_population_size.setObjectName("enter_population_size")
        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;")

        int_validator = QtGui.QIntValidator(0, 10000)
        self.enter_population_size.setValidator(int_validator)

        # ======================================
        #   Макс. кол-во поколений
        # ======================================
        self.label_max = QtWidgets.QLabel(self.groupBox)
        self.label_max.setGeometry(QtCore.QRect(20, 330, 200, 20))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.label_max.setFont(font)
        self.label_max.setObjectName("label_max")
        self.label_max.setStyleSheet("background-color: rgba(255,
255, 255, 0);")

        self.enter_max = QtWidgets.QLineEdit(self.groupBox)
        self.enter_max.setGeometry(QtCore.QRect(240, 330, 40, 21))
        self.enter_max.setObjectName("enter_max")
        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.setValidator(int_validator)

        # ======================================
        #   Количество случайных клеток
        # ======================================
        self.label_random = QtWidgets.QLabel(self.groupBox)
        self.label_random.setGeometry(QtCore.QRect(20, 410, 200, 20))
        font = QtGui.QFont()
        font.setPointSize(14)
```

```
        self.label_random.setFont(font)
        self.label_random.setObjectName("label_random")
        self.label_random.setStyleSheet("background-color: rgba(255,
255, 255, 0);")

        self.enter_n_random = QtWidgets.QLineEdit(self.groupBox)
        self.enter_n_random.setGeometry(QtCore.QRect(240, 410, 40,
21))
        self.enter_n_random.setObjectName("enter_n_random")
        self.enter_n_random.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_n_random.setValidator(int_validator)

        # ======================================
        #   Таблица для ввода старта
        # ======================================
        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.setAlignment(QtCore.Qt.AlignCenter)
        self.label_f.setObjectName("label_f")
        self.label_f.setStyleSheet("background-color: rgba(255, 255,
255, 0);\n"
                                   "color: rgb(48, 48, 48);")

        self.tablescreen = QtWidgets.QTableWidget(self.groupBox)
        self.tablescreen.setGeometry(QtCore.QRect(40, 60, 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.Se
lectItems)

        # ======================================
        #   Вывод ошибки
        # ======================================
        self.error_label = QtWidgets.QLabel(self.groupBox)
        self.error_label.setGeometry(QtCore.QRect(0, 560, 300, 41))
        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")

        # ======================================
        #   Кнопка включить до результата
        # ======================================
        self.to_end = QtWidgets.QPushButton(self.groupBox)
        self.to_end.setGeometry(QtCore.QRect(150, 610, 70, 30))
        font = QtGui.QFont()
        font.setPointSize(25)
        self.to_end.setFont(font)
        self.to_end.setObjectName("to_end")
        self.to_end.setEnabled(False)
        self.to_end.setStyleSheet("QPushButton {background-color:
rgb(187, 188, 188);"
                                  "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}")

        # ========================================
        #   Кнопка один шаг
        # ========================================
        self.one_step = QtWidgets.QPushButton(self.groupBox)
        self.one_step.setGeometry(QtCore.QRect(225, 610, 30, 30))
        font = QtGui.QFont()
        font.setPointSize(17)
        font.setBold(False)
        font.setWeight(50)
        self.one_step.setFont(font)
        self.one_step.setObjectName("one_step")
        self.one_step.setEnabled(False)
        self.one_step.setStyleSheet("QPushButton {background-color:
rgb(187, 188, 188);"
                                    "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}")

        # ========================================
        #   Обработчики
        # ========================================
        MainWindow.setCentralWidget(self.centralwidget)
        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)
        self.functions()

    # ========================================
    #   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", "Старт"))
```

```
        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.download_btn.setText(_translate("MainWindow", "Загрузить файл"))
        self.to_end.setText(_translate("MainWindow", "⏩"))
        self.one_step.setText(_translate("MainWindow", "▶️"))
        self.label_random.setText(_translate("MainWindow", "Кол-во случайных клеток"))

    # ========================================
    #   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)])

    # ========================================
    #   Обновление таблицы
    # ========================================
    def update_table(self):
        self.data = read_data()
        keys = list(self.data.keys())[1:]
        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)

    # ======================================
    #   Обновление экрана результата
    # ======================================
    def update_screen(self, array):
        for i in range(9):
            for j in range(9):
                item = QtWidgets.QTableWidgetItem(array[i][j])
                self.tablescreen.setItem(i, j, item)

    # ======================================
    #   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)
        figure.patch.set_facecolor((224 / 255, 225 / 255, 225 / 255))
        self.canvas = FigureCanvas(figure)
        ax = figure.add_subplot(111)
        ax.set_facecolor((224 / 255, 225 / 255, 225 / 255))

        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)

    # ======================================
    #   Таблица в массив
    # ======================================
    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

    # ========================================
    #   Открыть файл
    # ========================================
    def open_txt_file(self):
        file_name, _ =
QtWidgets.QFileDialog.getOpenFileName(self.MainWindow, "Выберите текстовый файл", "",
                                                                "Text
Files (*.txt)")
        if file_name:
            try:
                with open(file_name, "r", encoding="utf-8") as f:
                    text = f.read()
                if self.file_check(text) and
self.table_check([item.split(' ') for item in text.split('\n')]):
                    self.error_label.setText("")
                    self.update_screen([item.split(' ') for item in
text.split('\n')])
                else:
                    self.error_label.setText("Ошибка: Неверный формат поля")

            except Exception as e:
                QtWidgets.QMessageBox.warning(self.MainWindow, "Ошибка", f"Не удалось прочитать файл:\n{str(e)}")

    # ========================================
    #   Проверка поля
    # ========================================
    def table_check(self, table):
        for i in range(9):
            for j in range(9):
```

```
                if not table[i][j].isdigit() and not table[i][j] ==
'x':
                    return False
        return True

    def file_check(self, text: str) -> bool:
        lines = text.strip().splitlines()

        if len(lines) != 9:
            return False

        for i, line in enumerate(lines):
            parts = line.strip().split()
            if len(parts) != 9:
                return False
        return True

    # ========================================
    #   Обработчик функций
    # ========================================
    def functions(self):
        self.tableWidget.clicked.connect(lambda:
self.on_row_clicked())
        self.start_btn.clicked.connect(self.start)
        self.one_step.clicked.connect(self.start_one)
        self.to_end.clicked.connect(self.start_until_the_end)
        self.download_btn.clicked.connect(self.open_txt_file)

    # ========================================
    #   Подтверждение данных
    # ========================================
    def start(self):
        if self.is_start:
            print("Программа остановлена")
            self.start_btn.setText("Старт")
            clean_data()
            self.to_end.setEnabled(False)
            self.one_step.setEnabled(False)
            self.download_btn.setEnabled(True)
            self.spin_mutation.setReadOnly(False)
            self.spin_crossover.setReadOnly(False)
            self.enter_population_size.setReadOnly(False)
            self.enter_n_random.setReadOnly(False)
            self.enter_max.setReadOnly(False)
```

```
            self.tablescreen.setEnabled(True)
            self.to_end.setStyleSheet("background-color: rgb(187,
188, 188);"
                                      "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")
            self.one_step.setStyleSheet("background-color: rgb(187,
188, 188);"
                                        "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")
            self.download_btn.setStyleSheet("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")
            self.is_start = False
        else:
            is_correct = False

            while not is_correct:
                population_text =
self.enter_population_size.text().strip()
                generations_text = self.enter_max.text().strip()
                n_random = self.enter_n_random.text().strip()
                table = self.table_to_array(self.tablescreen)

                if not generations_text:
                    self.error_label.setText("Ошибка: Поле '
Макс. поколений'\nне должно быть пустым.")
                    return

                if not population_text:
                    self.error_label.setText("Ошибка: Поле '
Размер популяции'\nне должно быть пустым.")
```

```
                    return

                if not n_random:
                    self.error_label.setText("Ошибка: Поле '
Кол-во случайных клеток'\nне должно быть пус
тым.")
                    return

                if not (0 <= int(n_random) <= 81):
                    self.error_label.setText("Ошибка: Поле '
Кол-во случайных клеток'\nдолжно быть в диап
азоне 0-81.")
                    return

                if not self.table_check(table):
                    self.error_label.setText("Ошибка: Неверн
о задано поле")
                    return

                is_correct = True


            print("Программа запущена")
            self.error_label.setText("")
            self.start_btn.setText("Стоп")
            clean_data()
            data_init()
            self.i = 0
            self.best_fitness_values = []

self.field_creator.ReadFromList(self.table_to_array(self.tablescreen
))
            self.population_size =
int(self.enter_population_size.text())
            self.population =
self.field_creator.GeneratePopulation(self.population_size)
            self.fixed_positions =
self.field_creator.insert_list_indexes
            self.generations = int(self.enter_max.text())
            self.p_mutation =
float(self.spin_mutation.text().replace(',', '.'))
            self.update_table()
            self.to_end.setEnabled(True)
            self.one_step.setEnabled(True)
            self.download_btn.setEnabled(False)
```

```
            self.spin_mutation.setReadOnly(True)
            self.spin_crossover.setReadOnly(True)
            self.enter_population_size.setReadOnly(True)
            self.enter_n_random.setReadOnly(True)
            self.enter_max.setReadOnly(True)
            self.tablescreen.setEnabled(False)
            self.to_end.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)}")
            self.one_step.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)}")
            self.download_btn.setStyleSheet("background-color:
rgb(187, 188, 188);"
                                            "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")
            self.is_start = True

    # ========================================
    #   О д и н  ш а г
    # ========================================
    def start_one(self):
        self.i += 1
        self.one_iter()
        self.update_table()
        if self.best_fitness_values[-1] == 243:
            print("Sudoku solved!")
```

```
            self.to_end.setEnabled(False)
            self.one_step.setEnabled(False)
            self.to_end.setStyleSheet("background-color: rgb(187,
188, 188);"
                                      "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")
            self.one_step.setStyleSheet("background-color: rgb(187,
188, 188);"
                                        "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")

    # ======================================
    #   До результата
    # ======================================
    def start_until_the_end(self):
        for generation in range(self.i, self.generations):
            self.one_iter(generation)
            print(f"Generation {self.i + generation}, Best fitness:
{self.best_fitness_values[-1]}")
            if self.best_fitness_values[-1] == 243:
                self.i += generation
                print("Sudoku solved!")
                self.to_end.setEnabled(False)
                self.one_step.setEnabled(False)
                self.to_end.setStyleSheet("background-color: rgb(187,
188, 188);"
                                          "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")
                self.one_step.setStyleSheet("background-color:
rgb(187, 188, 188);"
                                            "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")
                self.update_table()
                break
        self.update_table()

    # ========================================
    #   О д н а  и т е р а ц и я
    # ========================================
    def one_iter(self, generation = 0):
        population = sorted(self.population, key=fitness_full,
reverse=True)
        best = population[0]
        best_fitness = fitness_full(best)
        self.best_fitness_values.append(best_fitness)

        # С о х р а н е н и е  д а н н ы х  о  п о к о л е н и и
        data = [best_fitness, field_to_str(best)]
        save_data(self.i + generation, data)

        if best_fitness == 243:
            return best

        selected = group_tournament_selection(population)

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

            if random.random() < self.p_mutation:
                random_mutation(child, self.fixed_positions)

            next_generation.append(child)

        self.population = next_generation


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
```

```
MainWindow = QtWidgets.QMainWindow()
ui = UiMainWindow()
ui.setup_ui(MainWindow)
MainWindow.show()
sys.exit(app.exec_())
```

Название файла: data_saver.py

```
import json
import os

FILENAME = 'data.json'


def field_to_str(field: list[list[int]]) -> str:
    string = ''
    for x in range(9):
        for y in range(9):
            string += str(field[x][y])
    return string


def str_to_field(string: str) -> list[list[int]]:
    field = []
    for x in range(9):
        row = []
        for y in range(9):
            row.append(int(string[x * 9 + y]))
        field.append(row)
    return field


def format_9x9_square(arr):
    lines = []
    for i in range(9):
        row = arr[i] if i < len(arr) else [0] * 9
        row_9 = row[:9] + [0] * (9 - len(row))
        line = " | ".join(str(x) for x in row_9)
        lines.append(line)
    return "\n".join(lines)


def save_data(id: int, data_array: list):
    if os.path.exists(FILENAME):
        with open(FILENAME, 'r', encoding='utf-8') as f:
            try:
                all_data = json.load(f)
            except json.JSONDecodeError:
                all_data = {}
    else:
        all_data = {}
```

```
    all_data['best_fitness'].append(data_array[0])
    all_data[str(id)] = data_array

    with open(FILENAME, 'w', encoding='utf-8') as f:
        json.dump(all_data, f, ensure_ascii=False, indent=4)


def read_data():
    if not os.path.exists(FILENAME):
        return None

    with open(FILENAME, 'r', encoding='utf-8') as f:
        try:
            all_data = json.load(f)
        except json.JSONDecodeError:
            return None

    return all_data


def data_init():
    all_data = {}
    all_data['best_fitness'] = []
    with open(FILENAME, 'w', encoding='utf-8') as f:
        json.dump(all_data, f, ensure_ascii=False, indent=4)


def clean_data():
    with open(FILENAME, 'w', encoding='utf-8') as f:
        pass
```

Название файла: plot_drawing.py

```
import matplotlib.pyplot as plt

# 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()
```

Название файла: reader_writer.py

```
def ReadFromFile(file_name: str) -> None:
    main_permutation = list(range(1, 10)) * 9
    insert_list_indexes = []
    insert_list_symbols = []

    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':
                insert_list_indexes.append((x, y))
                insert_list_symbols.append(int(symbol))
                main_permutation.remove(int(symbol))
    return main_permutation, insert_list_indexes,
insert_list_symbols

def ReadFromList(arr: list[list[str]]) -> None:
    main_permutation = list(range(1, 10)) * 9
    insert_list_indexes = []
    insert_list_symbols = []
    for x in range(9):
        for y in range(9):
            symbol = arr[x][y]
            if symbol != 'x':
                insert_list_indexes.append((x, y))
                insert_list_symbols.append(int(symbol))
                main_permutation.remove(int(symbol))
    return main_permutation, insert_list_indexes,
insert_list_symbols
```

Название файла: genetic_algorithm.py

```
import random
import numpy as np
from typing import overload, Literal, Union
import math

from data_saver import *
```

```
from reader_writer import *
from plot_drawing import *


class GeneticAlgorithm():
    def __init__(self):
        self.main_permutation = list(range(1, 10)) * 9
        self.insert_list_indexes = []
        self.insert_list_symbols = []
        self.population = []
        self.best_fitness_values = []

    # ========================== getting data
    @overload
    def get_data(self, data: list, flag: Literal['l']) -> None:
        ...
    @overload
    def get_data(self, data: str, flag: Literal['f']) -> None:
        ...
    def get_data(self, data: Union[list, str], flag: str) -> None:
        if flag == 'l' and isinstance(data, list):
            self.main_permutation,         self.insert_list_indexes,
self.insert_list_symbols = ReadFromList(data)
        elif flag == 'f' and isinstance(data, str):
            self.main_permutation,         self.insert_list_indexes,
self.insert_list_symbols = ReadFromFile(data)
        else:
            raise ValueError("Некорректные аргументы")
    # ==========================

    # ========================== generating population
    def GeneratePopulation(self, entities_amount: int) -> None:
        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)]
            self.population.append(new_entity)
    # ==========================

    # ========================== fitness calculation
    def fitness_full(self, 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(self, 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
    # ==========================

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

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

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

    # ========================== crossing
    def one_point_crossing_sq(self, parent1, parent2):
        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 self.insert_list_indexes:
                        child[i][j] = parent2[i][j]

        return child

    def uniform_crossover_sq(self, parent1, parent2):
        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 self.insert_list_indexes:
                            # Сохраняем фиксированную
ячейку
                            child[x][y] = parent1[x][y]
                        else:
                            child[x][y] = source[x][y]

        return child

    def uniform_crossover_cell(self, parent1, parent2):
        child = [row.copy() for row in parent1]

        for i in range(9):
            for j in range(9):
                if (i, j) in self.insert_list_indexes:
                    continue
```

```
                child[i][j] = parent1[i][j] if random.random() < 0.5
else parent2[i][j]

        return child

    def   uniform_crossover_row(self,   parent1:   list[list[int]],
parent2:list[list[int]]) -> list[list[int]]:
        child = [row.copy() for row in parent1]
        for row in range(9):
            if random.random() < 0.5:
                for column in range(9):
                    child[row][column] = parent1[row][column]
            else:
                for column in range(9):
                    child[row][column] = parent2[row][column]

        return child


    def  uniform_crossover_column(self,  parent1:  list[list[int]],
parent2: list[list[int]]) -> list[list[int]]:
        child = [row.copy() for row in parent1]
        for column in range(9):
            if random.random() < 0.5:
                for row in range(9):
                    child[row][column] = parent1[row][column]
            else:
                for row in range(9):
                    child[row][column] = parent2[row][column]

        return child
    # ==========================

    # ========================== mutation
    def random_mutation(self, entity: list[list[int]]) -> None:
        numbers = list(range(0, 81))
        for item in self.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

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

    def row_shuffle_mutation(self, entity: list[list[int]]) -> None:
        fixed_set = set(self.insert_list_indexes)
        row_idx = random.randint(0, 8)
        mutable_indices = [j for j in range(9) if (row_idx, j) not
in fixed_set]

        if len(mutable_indices) < 2:
            return

        values_to_shuffle = [entity[row_idx][j] for j in
mutable_indices]
        random.shuffle(values_to_shuffle)

        for idx, j in enumerate(mutable_indices):
            entity[row_idx][j] = values_to_shuffle[idx]

    def mutation_change_5_percent(self, entity: list[list[int]]) ->
None:
        fixed_set = set(self.insert_list_indexes)

        mutable_positions = [(i, j) for i in range(9) for j in
range(9) if (i, j) not in fixed_set]
```

```
        if not mutable_positions:
            return

        num_to_mutate = max(1, round(0.05 * len(mutable_positions)))

        positions_to_mutate = random.sample(mutable_positions,
num_to_mutate)

        for i, j in positions_to_mutate:
            current_val = entity[i][j]
            new_val = random.choice([v for v in range(1, 10) if v !=
current_val])
            entity[i][j] = new_val
    # ==========================

    # ========================== main cycle
    def main_cycle(self, generations, population_size,
mutation_rate):

        data_init()

        for generation in range(generations):
            population = sorted(self.population, key =
self.fitness_full, reverse=True)
            best = population[0]
            best_fitness = self.fitness_full(best)
            self.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}, diversity:
{calculate_population_diversity(population)}")
```

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

            selected = self.group_tournament_selection(2)

            #   next_generation   =   population[:math.ceil(0.05   *
len(population))]
            next_generation = []
            while len(next_generation) < population_size:
                parent1, parent2 = random.sample(selected, 2)
                #   child   =   self.uniform_crossover_cell(parent1,
parent2)
                # child = []
                # if random.random() < crossover_rate:
                #   child   =   self.one_point_crossing_sq(parent1,
parent2)

                # else:
                #     next_generation.append(parent1)
                #     next_generation.append(parent2)
                #     continue

                child = []
                #   child   =   self.one_point_crossing_sq(parent1,
parent2, fixed_positions)
            #   child1   =   uniform_crossover_cell(parent1,   parent2,
fixed_positions)
            #   child2   =   uniform_crossover_cell(parent1,   parent2,
fixed_positions)
            #   child3   =   uniform_crossover_cell(parent1,   parent2,
fixed_positions)
                cross_mode = random.choice([1,2,3])
                if cross_mode == 1:
```

```
                    child    =    self.uniform_crossover_sq(parent1,
parent2)
                if cross_mode == 2:
                    child    =    self.uniform_crossover_row(parent1,
parent2)
                if cross_mode == 3:
                    child  =  self.uniform_crossover_column(parent1,
parent2)

                if    (random_number    :=    random.random())    <
mutation_rate:
                    # self.row_shuffle_mutation(child)
                    self.random_mutation(child)

                    # if len(get_bad_rows(child)) > 0:
                    #                 mutation_among_bad_rows(child,
fixed_positions, get_bad_rows(child), True)
                    # else:

                # В О Т  С Ю Д А  Я  Д О Б А В И Л  М Е Г А  Р А Н Д О М Н У
Ю  М У Т А Ц И Ю!!!!!!!!!!!!!!!!!!
                # elif random_number < 0.05:
                #     child = self.very_random_mutation()



                next_generation.append(child)

            self.population = next_generation

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

    """
    ============================
```

```
    Т Е С Т О В Ы Е  Ф У Н К Ц И И  А Л Г О Р И Т М А
    ============================
    """
    def very_random_mutation(self) -> list[list[int]]:
        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)]

        return new_entity

    """
    П р и  и с п о л ь з о в а н и и  ф у н к ц и и - и т е р а ц и и  в  н а ч
а л е  н у ж н о  д о б а в и т ь  data_init() и з data_saver.py

    Ф у н к ц и я  и т е р а ц и и  д о л ж н а  б ы т ь  в н у т р и  ц и к л
а  for generation in generations, г д е  generations - э т о  к о л - в
о  п о п у л я ц и й
    Э т о й  ф у н к ц и и  п е р е д а в а т ь  н о м е р  п о п у л я ц и и
generation
    П о с л е  ц и к л а  д о б а в и т ь :

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

    К о р о ч е  о р и е н т и р о в а т ь с я  н а  т о  к а к  р а б о т а е
т main_cycle е с л и  ч т о
    """
    def    one_iteration(self,    population_size,    mutation_rate,
generation):

        # /finding best fitness
```

```
        population = sorted(self.population, key=self.fitness_full,
reverse=True)
        best = population[0]
        best_fitness = self.fitness_full(best)
        self.best_fitness_values.append(best_fitness)

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

        # / printing debug data
        print(
            f"Generation {generation}, Best fitness: {best_fitness},
diversity: {calculate_population_diversity(population)}")
        #    print(f"Generation    {generation},    Best    fitness:
{best_fitness}, diversity: ")
        if best_fitness == 243:
            print("Sudoku solved!")
            plot_progress(self.best_fitness_values)
            return best

        # / selection
        selected = self.group_tournament_selection(10)

        # / best 5% of selected generation survives
        next_generation      =      population[:math.ceil(0.05      *
len(population))]
        # next_generation = []

        # / crossing + mutation
        while len(next_generation) < population_size:
            parent1, parent2 = random.sample(selected, 2)
            # child = self.uniform_crossover_cell(parent1, parent2)
            # child = []
            # if random.random() < crossover_rate:
            # child = self.one_point_crossing_sq(parent1, parent2)
```

```
            # else:
            #     next_generation.append(parent1)
            #     next_generation.append(parent2)
            #     continue

            child = []
            # child = self.one_point_crossing_sq(parent1, parent2,
fixed_positions)
            # child1 = uniform_crossover_cell(parent1, parent2,
fixed_positions)
            # child2 = uniform_crossover_cell(parent1, parent2,
fixed_positions)
            # child3 = uniform_crossover_cell(parent1, parent2,
fixed_positions)
            cross_mode = random.choice([1, 2, 3])
            if cross_mode == 1:
                child = self.uniform_crossover_sq(parent1, parent2)
            if cross_mode == 2:
                child = self.uniform_crossover_row(parent1, parent2)
            if cross_mode == 3:
                child    =    self.uniform_crossover_column(parent1,
parent2)

            if (random_number := random.random()) < mutation_rate:
                # self.row_shuffle_mutation(child)
                self.random_mutation(child)

                # if len(get_bad_rows(child)) > 0:
                #     mutation_among_bad_rows(child, fixed_positions,
get_bad_rows(child), True)
                # else:

            # ВОТ СЮДА Я ДОБАВИЛ МЕГА РАНДОМНУЮ МУТАЦИЮ!!!!!!!!!!!!!!!!!!
            # elif random_number < 0.05:
            #     child = self.very_random_mutation()
```

```
            next_generation.append(child)

        # / generating new population
        self.population = next_generation


# ============================================================
def   main_start(field:   list[list[str]],   population_size:   int,
generation_size: int, p_mutation: float):
    gen_alg = GeneticAlgorithm()
    gen_alg.get_data(field, 'l')
    gen_alg.GeneratePopulation(population_size)
    solution  =  gen_alg.main_cycle(generation_size,  population_size,
p_mutation)

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

    def grid_to_hashable(grid):
        return tuple(tuple(row) for row in grid)

    unique_grids  =  set(grid_to_hashable(individual)  for  individual
in population)
    total = len(population)

    if total == 0:
        return 0.0

    diversity_percent = (len(unique_grids) / total) * 100
    return diversity_percent
# ==========================
```

```
if __name__ == "__main__":
    gen_alg = GeneticAlgorithm()
    # gen_alg.insert_list_indexes =[(1, 1), (5, 6), (2, 8)]
    gen_alg.get_data('example.txt', 'f')
    gen_alg.GeneratePopulation(8000)

    solution = gen_alg.main_cycle(10000, 8000, 0.5)

    print(gen_alg.fitness_full(solution))
    print_ind(solution)
```