

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

**ОТЧЕТ**  
**по летней практике, итерация №5**  
**Тема: Генетические Алгоритмы “Решение задачи судoku”**

Студенты гр. 3342

Преподаватель

Галеев А.Д; Романов

Е.А; Хайруллов Д.Л;

Жангиров Т.Р.

Санкт-Петербург

2025

**Цель работы.**

Написать программу решающую задачу sudoku с использованием генетического алгоритма.

**Задание.**

Для заданного игрового поля sudoku (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 последней итерации нашего проекта мы получили финальную рабочую версию программы. GUI полностью связан с работой генетического алгоритма. Есть возможность задать начальную расстановку для sudoku, выбрать загрузку данных из файла, выбрать начальные параметры генетического алгоритма. Получаемые данные проверяются: в случае несоответствия условиям расстановки по правилам sudoku или получения других невалидных данных в интерфейсе отображается сообщение с ошибкой. Для управления потоком выполнения алгоритма были реализованы кнопки старта, паузы, выполнения до конца, выполнения одного шага алгоритма. Все данные, такие как текущая расстановка с лучшей приспособленностью, график зависимости значения приспособленности от номера поколения, динамически отображаются в пользовательском интерфейсе.

### **Вклад участников:**

Романов Егор – добавление динамически обновляющегося по ходу вычисления алгоритма графика, реализация функционала кнопки паузы.

Хайруллов Динар – рефакторинг кода, тестирование финальной версии GUI и генетического алгоритма.

Галеев Амир – реализация функции проверки ввода поля для sudoku, изменение логики обновления графика и таблицы, панели управления алгоритмом.

## **Описание функций и структур данных.**

### **GUI**

#### **Класс UiMainWindow**

Основной класс интерфейса приложения для решения sudoku генетическим алгоритмом.

Методы класса:

`__init__()` — Инициализация основных атрибутов класса (алгоритм, параметры, флаги состояния).

`setup_ui(MainWindow)` — Настройка графического интерфейса (кнопки, таблицы, график, стили).

`retranslateUi(MainWindow)` — Установка текстовых значений для элементов интерфейса (перевод).

`on_row_clicked()` — Обработчик клика по строке в таблице результатов. Обновляет экран и график.

`update_table()` — Обновление таблицы результатов на основе данных из файла.

`update_screen(array)` — Обновление игрового поля (таблицы 9x9) переданными значениями.

`plot_graph(fitness_values)` — Отрисовка графика зависимости fitness от поколения.

`table_to_array(table)` — Преобразование QTableWidgetItem в двумерный список (матрицу 9x9).

`open_txt_file()` — Открытие и загрузка файла с начальным состоянием sudoku.

`update_progress_bar()` — Обновление прогресс-бара на основе текущего значения fitness.

`toggle_pause()` — Переключение режима паузы для алгоритма (вкл/выкл).

`table_check(table)` — Проверка корректности формата таблицы (только цифры 1-9 или 'x').

`enter_check(table)` — Проверка правил sudoku (уникальность в строках/столбцах).

`file_check(text)` — Проверка формата загружаемого файла (9 строк по 9 значений).

`functions()` — Подключение сигналов кнопок к соответствующим методам.

`start()` — Обработчик кнопки "Старт/Стоп". Запускает или останавливает алгоритм.

`start_one()` — Выполнение одной итерации алгоритма (один шаг).

`start_until_the_end()` — Запуск алгоритма до завершения (решения или достижения лимита поколений).

`_process_generation()` — Внутренний метод для обработки одного поколения в автоматическом режиме.

`_finish_algorithm()` — Завершение работы алгоритма и обновление интерфейса.

## **Генетический алгоритм**

Класс `GeneticAlgorithm`:

Реализует генетический алгоритм для решения sudoku. 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 sudoku

`get_data(data: Union[list, str], flag: str)` - Загружает начальное состояние sudoku 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_crossover_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 регенерация 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 sudoku

`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 sudoku 9x9 в строку

`str_to_field(string: str) -> list[list[int]]` - Восстанавливает поле sudoku 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.

Тестирования ввода невалидных данных:

При отсутствии значений в одном из основных полей параметра алгоритма и попытке его запуска выведется сообщение об этой ошибке. Результат данного тестирования представлен на рисунке 1.

The screenshot shows a web application for solving Sudoku puzzles. On the left, under the heading "Параметры" (Parameters), there is a section "Начальное поле" (Initial field) containing a 9x9 grid. The grid has most cells filled with 'x', with some numbers: 8, 1, 9 (highlighted in blue), 7, 2, 3, and 5. Below the grid is a button "Загрузить файл" (Load file). Further down are three input fields: "Макс. кол-во поколений" (Max number of generations) which is empty, "Размер популяции" (Population size) set to 4000, and "Вероятность мутации" (Mutation probability) set to 0,55. At the bottom of the parameters section, an error message is displayed: "Ошибка: Поле 'Макс. поколений' не должно быть пустым." (Error: The 'Max generations' field must not be empty). Below the error message is a progress indicator showing "0%". At the very bottom are four buttons: "Старт" (Start), a pause button, a fast-forward button, and a "+1" button. On the right side of the interface, there are two large empty boxes with headers "№" (Number) and "Best fitness".

Рис. 1 — Пустое поле параметра алгоритма.

При вводе поля, нарушающего правила sudoku, будет выведена ошибка. Результат тестирования представлен на рисунке 2.

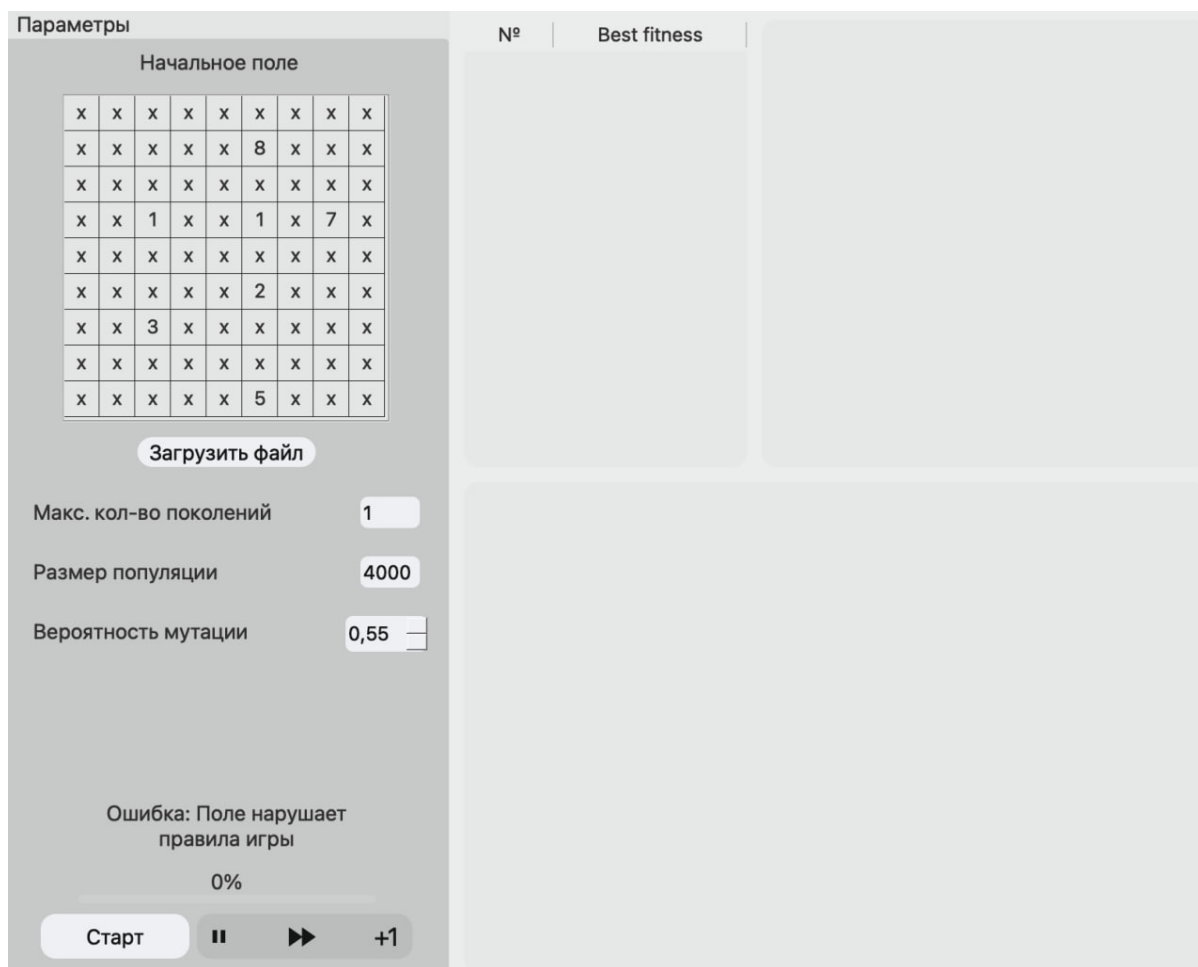


Рис. 2 — Поле, нарушающее правила sudoku.

Тестирование элементов управления потока выполнения алгоритма

GUI:

После ввода данных и запуска алгоритма в непрерывном режиме есть возможность воспользоваться кнопкой паузы. При ее активации программа останавливается на моменте нажатия. Есть возможность продолжить выполнять алгоритм по одному шагу в ручном режиме, либо же продолжить ее выполнение в непрерывном режиме. Результаты тестирования представлены на рисунках 3 и 4.

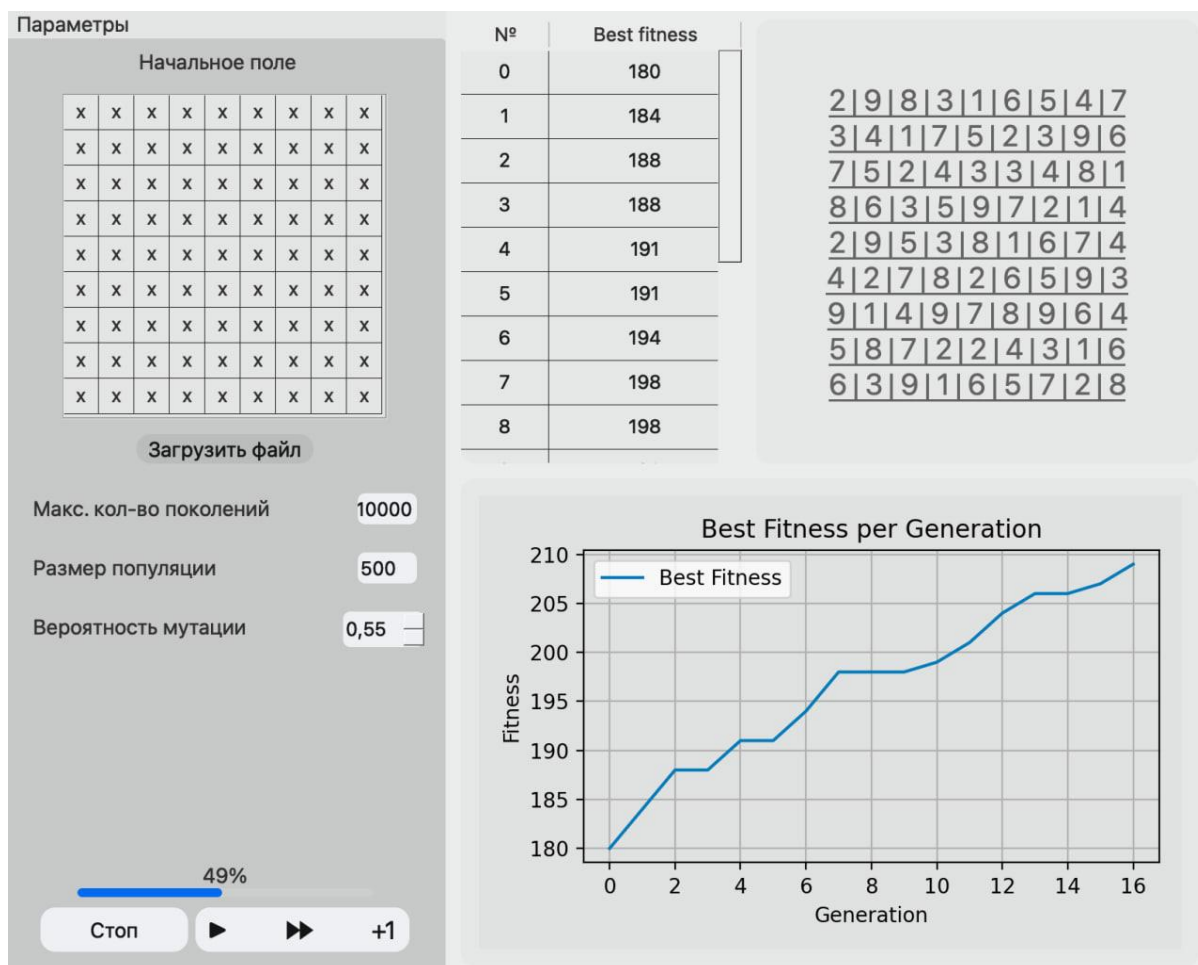


Рис. 3 — Режим паузы.

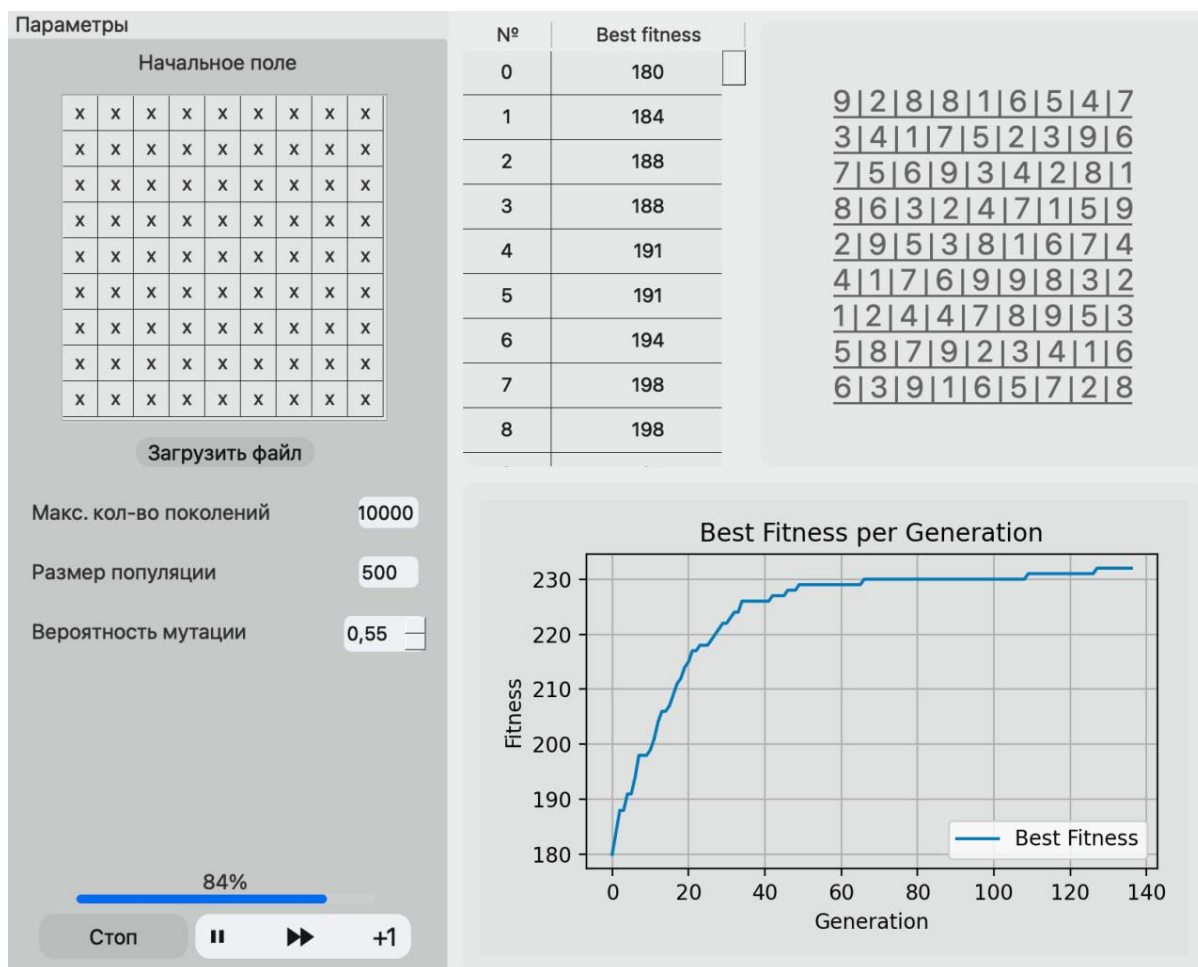


Рис. 4 — Возобновление непрерывного выполнения после паузы.

## **Вывод.**

На данной итерации был доработан графический пользовательский интерфейс, добавлен весь необходимый функционал для полноценной работы пользователя с реализованным генетическим алгоритмом.

GUI полностью связан с генетическим алгоритмом. Были реализованы элементы интерфейса для управления потоком выполнения алгоритма. Выполнена вся визуальная часть интерфейса, отображающая характеристики особей поколения: изменение значения приспособленности с каждым новым поколением, графическое представление особей (поле судoku с текущей лучшей расстановкой чисел), шкала прогресса алгоритма.

Было проведено тестирование, которое демонстрирует работоспособность реализованного графического пользовательского интерфейса и работоспособность самого алгоритма, который улучшает приспособленность особей, пока не найдет индивида с максимальным значением этого показателя, что и является сигналом о решении поставленной задачи.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: gui.py

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

class UiMainWindow(object):
    def __init__(self):
        self.i = 0
        self.alg = GeneticAlgorithm()
        self.population_size = 0
        self.population = self.alg.GeneratePopulation(10)
        self.fixed_positions = 0
        self.generations = 0
        self.p_mutation = 0
        self.is_start = False
        self.figure = None
        self.ax = None
        self.line = None
        self.canvas = None
        self.is_pause = False

    def setup_ui(self, MainWindow):

        # =====
        #   Н а с т р о й к и   о к н а
        #   =====
        self.MainWindow = 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)
self.centralwidget = QtWidgets.QWidget(MainWindow)
self.centralwidget.setObjectName("centralwidget")
MainWindow.setStyleSheet("background-color: rgb(235, 236,
236);\n"
                           "color: rgb(48, 48, 48);")

# =====
#   О к н о   в ы  в о д а   л у ч ш е г о   р е з у л ь т а т а
#   =====
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;")

# =====
#   Т а б л и ц а   р е з у л ь т а т о в
#   =====
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.horizontalHeader().setSectionResizeMode(QtWidgets.Q
HeaderView.Fixed)
        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(25, 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, 410, 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, 410, 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)

```

```

# =====
#   Р а з м е р   п о п у л я ц и и
# =====
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)

# =====
#   П а н е л ь   у п р а в л е н и я
# =====
self.label_bar = QtWidgets.QLabel(self.groupBox)
self.label_bar.setGeometry(QtCore.QRect(130, 610, 145, 30))
self.label_bar.setStyleSheet("background-color: rgb(187, 188,
188);\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: 10px;")

self.label_bar.setText("")
self.label_bar.setObjectName("label_bar")
self.label_bar.raise_()

# =====
#   К н о п к а   п а у з ы
# =====
self.pause = QtWidgets.QPushButton(self.groupBox)
self.pause.setGeometry(QtCore.QRect(130, 610, 30, 30))
self.pause.setMinimumSize(QtCore.QSize(30, 30))
self.pause.setMaximumSize(QtCore.QSize(30, 30))
font = QtGui.QFont()
font.setPointSize(17)
font.setBold(False)
font.setWeight(50)
self.pause.setFont(font)
self.pause.setEnabled(False)
self.pause.setObjectName("pause")
self.pause.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;}")

# =====
#   М а к с .   к о л - в о   п о к о л е н и й
# =====
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)

        # =====
        #   Т а б л и ц а   д л я   в в о д а   с т а р т а
        #   =====
        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.SelectItems)

# =====
#   В ы в о д   о ш и б к и
# =====
self.error_label = QtWidgets.QLabel(self.groupBox)
self.error_label.setGeometry(QtCore.QRect(0, 530, 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")

# =====
#   К н о п к а   в к л ю ч и т ь   д о   р е з у л ь т а т а
# =====
self.to_end = QtWidgets.QPushButton(self.groupBox)
self.to_end.setGeometry(QtCore.QRect(165, 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);"}")

```



```

color: rgb(20, 21, 21); "border-color: rgb(147, 147, 147);
255);" "selection-color: rgb(255, 255,
rgb(16, 81, 193);" "selection-background-color:
"border-radius: 10px}")

# =====
#   Ш к а л а   в ы п о л н е н и я
# =====
self.progressBar = QtWidgets.QProgressBar(self.groupBox)
self.progressBar.setGeometry(QtCore.QRect(50, 580, 201, 40))
font = QtGui.QFont()
font.setPointSize(14)
self.progressBar.setFont(font)
self.progressBar.setStyleSheet("background-color: rgb(213,
215, 213, 0)")
self.progressBar.setProperty("value", 0)
self.progressBar.setAlignment(QtCore.Qt.AlignHCenter |
QtCore.Qt.AlignTop)
self.progressBar.setObjectName("progressBar")

# =====
#   К н о п к а   о д и н   ш а г
# =====
self.one_step = QtWidgets.QPushButton(self.groupBox)
self.one_step.setGeometry(QtCore.QRect(240, 610, 35, 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}")

# =====
#   О б р а б о т ч и к и
#   =====
MainWindow.setCentralWidget(self.centralwidget)
self.retranslateUi(MainWindow)
QtCore.QMetaObject.connectSlotsByName(MainWindow)
self.functions()

# =====
#   П е р е в о д   и н т е р ф е й с а
#   =====
def retranslateUi(self, MainWindow):
    _translate = QtCore.QCoreApplication.translate
    MainWindow.setWindowTitle(_translate("MainWindow", "Г А   С у
д о к у"))

    self.groupBox.setTitle(_translate("MainWindow", "П а р а м е
т р ы"))

    self.start_btn.setText(_translate("MainWindow", "С т а р т"))
    self.label_mutation.setText(_translate("MainWindow", "В е р
о я т н о с т ь   м у т а ц и и"))

    self.label_population.setText(_translate("MainWindow", "П а
з м е р   п о п у л я ц и и"))

    self.label_max.setText(_translate("MainWindow", "М а к с .   к
о л - В о   п о к о л е н и й"))

    self.label_f.setText(_translate("MainWindow", "Н а ч а л ь н
о е   п о л е"))

    self.error_label.setText(_translate("MainWindow", ""))
    self.download_btn.setText(_translate("MainWindow", "З а г р
у з и т ь   ф а й л"))

    self.to_end.setText(_translate("MainWindow", "▶▶"))
    self.pause.setText(_translate("MainWindow", "⏸"))
    self.one_step.setText(_translate("MainWindow", "+1"))

# =====
#   О б р а б о т ч и к   в ы б о р а   с т р о к и
#   =====
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)])

# =====
#   О б н о в л е н и е   т а б л и ц ы
# =====
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)

# =====
#   О б н о в л е н и е   э к р а н а   р е з у л ь т а т а
# =====
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)

# =====
#   О т р и с о в щ и к   г р а ф и к а
# =====
# 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)

def plot_graph(self, fitness_values):
    # Если график ещё не инициализирован
    if not hasattr(self, 'canvas') or self.canvas is None:
        self.figure = Figure(figsize=(10, 5), tight_layout=True)
        self.figure.patch.set_facecolor((224/255, 225/255,
225/255))

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

        self.graph_layout.addWidget(self.canvas)
        self.canvas.show()

    self.ax.clear()
    self.ax.plot(fitness_values, label='Best Fitness')
    self.ax.set_xlabel("Generation")
    self.ax.set_ylabel("Fitness")
    self.ax.set_title("Best Fitness per Generation")
    self.ax.legend()
    self.ax.grid(True)
    self.ax.set_facecolor((224/255, 225/255, 225/255))

    self.canvas.draw()
    QtWidgets.QApplication.processEvents()

```

```

# =====
#   Т а б л и ц а   В   м а с с и в
# =====
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

# =====
#   О т к р ы т ь   ф а й л
# =====
def open_txt_file(self):
    file_name, _ =
QtWidgets.QFileDialog.getOpenFileName(self.MainWindow, "В ы б е р и т
е   т е к с т о в ы й   ф а й л", "",
                                                                    "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("О ш и б к а : Н е в е р н
ый   ф о р м а т   п о л я")

        except Exception as e:

```

```

QtWidgets.QMessageBox.warning(self.MainWindow, "О ш
и б к а", f"Не удалось прочитать файл:\n{str(e)}")

```

```

# =====
#   О б н о в л е н и е   з а г р у з к и
# =====
def update_progress_bar(self):
    min_rate = 243 - self.alg.best_fitness_values[0]
    best = min_rate - (243 - self.alg.best_fitness_values[-1])

    value = round((best / min_rate) * 100)
    self.progressBar.setValue(value)

# =====
#   О б н о в л е н и е   з а г р у з к и
# =====
# def pause_alg(self):
#     if self.is_pause:
#         self.is_pause = False
#         self.pause.setText("II")
#     else:
#         self.is_pause = True
#         self.pause.setText("▶")

def toggle_pause(self):

    self.is_pause = not self.is_pause

    if not self.is_pause: # Е с л и   с н и м а е м   п а у з у
        if hasattr(self, 'generation_timer'):
            QTimer.singleShot(0, lambda: (
                self.generation_timer.start(0), # З а п у с к а
                self.pause.setText("II"),
                self.start_btn.setEnabled(False),
                self.start_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);"

```

```

background-color: rgb(16, 81, 193);"
10px")
    ))

    else:
        self.pause.setText("▶")
        self.start_btn.setEnabled(True)
        self.start_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")

# =====
#   П р о в е р к а   п о л я
# =====
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 enter_check(self, table):
    for i in range(9):
        line = []
        column = []
        for j in range(9):
            if table[i][j].isdigit():
                if not (0 < int(table[i][j]) <= 9):
                    return False
                if table[i][j] not in line:
                    line.append(table[i][j])
            else:
                return False
        if table[j][i].isdigit():

```

```

        if table[j][i] not in column:
            column.append(table[j][i])
        else:
            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

# =====
#   О б р а б о т ч и к   ф у н к ц и й
# =====
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.pause.clicked.connect(self.toggle_pause)
    self.download_btn.clicked.connect(self.open_txt_file)

# =====
#   П о д т в е р ж д е н и е   д а н н ы х
# =====
def start(self):
    if self.is_start:
        print("П р о г р а м м а   о с т а н о в л е н а ")
        self.start_btn.setText("С т а р т ")
        if self.is_pause:
            self.is_pause = False
            self.pause.setText("||") # У с т а н а в л и в а е м
з н а ч о к   п а у з ы

```



```

        self.pause.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")
        clean_data()

        if hasattr(self, 'canvas'):
            self.graph_layout.removeWidget(self.canvas)
            self.canvas.setParent(None)
        self.screen.setText('')
        self.progressBar.setValue(0)
        self.tableWidget.setRowCount(0)
        self.to_end.setEnabled(False)
        self.one_step.setEnabled(False)
        self.pause.setEnabled(False)
        self.download_btn.setEnabled(True)
        self.spin_mutation.setReadOnly(False)
        self.enter_population_size.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.pause.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.label_bar.setStyleSheet("background-color: rgb(187,
188, 188);\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: 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()
            table = self.table_to_array(self.tablescreen)

            if not generations_text:
                self.error_label.setText("О ш и б к а :  П о л е '
М а к с .  п о к о л е н и й '\nн е  д о л ж н о  б ы т ь  п у с т ы м .")
                return

            if not population_text:

```

```

        self.error_label.setText("Ошибка: Поле '
Размер популяции'\nне должно быть пустым.")
        return

    if not self.table_check(table):
        self.error_label.setText("Ошибка: Неве р н
о з а д а н о п о л е")
        return

    if not self.enter_check(table):
        self.error_label.setText("Ошибка: П о л е н
а р у ш а е т \nп р а в и л а   и г р ы")
        return

    is_correct = True

    print("Программа запущена")
    self.error_label.setText("")
    self.start_btn.setText("С т о п")
    clean_data()
    data_init()
    self.i = 0
    self.alg.main_permutation, self.alg.insert_list_indexes,
self.alg.insert_list_symbols =
ReadFromList(self.table_to_array(self.tablescreen))
    self.population_size =
int(self.enter_population_size.text())
    self.alg.population = []
    self.alg.GeneratePopulation(self.population_size)
    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.enter_population_size.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.pause.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.label_bar.setStyleSheet("background-color: rgb(239,
240, 244);\n"
                                "border-color: rgb(147, 147,
147);\n"
                                "color: rgb(20, 21, 21);\n")

```

```

255, 255);\n"
rgb(16, 81, 193);\n"

self.is_start = True

# =====
#   О д и н   ш а г
# =====
def start_one(self):
    self.alg.one_iteration(self.population_size, self.p_mutation,
self.i)
    self.update_progress_bar()
    self.update_table()
    data = read_data()
    label_text =
format_9x9_square(str_to_field(data[str(self.i)][1]))
    self.screen.setText(label_text)
    self.plot_graph(self.data.get('best_fitness')[:int(self.i)])
    if self.alg.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);"
147); color: rgb(20, 21, 21); "
255);"
rgb(16, 81, 193);"
self.one_step.setStyleSheet("background-color: rgb(187,
188, 188);"
147); color: rgb(20, 21, 21);"
255, 255);"
rgb(16, 81, 193);"

if self.i == self.generations:
    self.to_end.setEnabled(False)

```

```

        self.one_step.setEnabled(False)
        self.pause.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.pause.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.label_bar.setStyleSheet("background-color: rgb(187,
188, 188);\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: 10px;")

        self.i += 1

        # =====
        #   Д о  р е з у л ь т а т а

```

```

# =====
def start_until_the_end(self):
    self.pause.setEnabled(True)
    self.start_btn.setEnabled(False)
    self.start_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_pause = False
    self.pause.setText("II") # У с т а н а в л и в а е м з н а ч о
к п а у з ы
    self.pause.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")

    if not hasattr(self, 'generation_timer'):
        self.generation_timer = QTimer()

self.generation_timer.timeout.connect(self._process_generation)

    self.current_generation = self.i
    self.generation_timer.start(0)

def _process_generation(self):
    """Обработка одного поколения"""
    if self.is_pause:
        self.generation_timer.stop()
        return

    self.alg.one_iteration(self.population_size, self.p_mutation,
self.i)

```

```

# У с л о в и я   з а в е р ш е н и я
if (self.i >= self.generations or
    self.alg.best_fitness_values[-1] == 243):
    self._finish_algorithm()
    return

# О б н о в л е н и е   и н т е р ф е й с а
self.update_progress_bar()
self.update_table()
data = read_data()
label_text =
format_9x9_square(str_to_field(data[str(self.i)][1]))
self.screen.setText(label_text)
self.plot_graph(self.data.get('best_fitness')[:int(self.i)])

# П е р е х о д   к   с л е д у ю щ е м у   п о к о л е н и ю
self.i += 1

def _finish_algorithm(self):
    """З а в е р ш е н и е   р а б о т ы   а л г о р и т м а"""
    self.generation_timer.stop()
    # self.i = self.current_generation

    # В а ш   о р и г и н а л ь н ы й   к о д   с т и л и з а ц и и
    print("Sudoku solved!")
    self.to_end.setEnabled(False)
    self.one_step.setEnabled(False)
    self.pause.setEnabled(False)
    self.start_btn.setEnabled(True)
    self.start_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.to_end.setStyleSheet("background-color: rgb(187, 188,
188);"

```



```

color: rgb(20, 21, 21); "border-color: rgb(147, 147, 147);
255);" "selection-color: rgb(255, 255,
rgb(16, 81, 193);" "selection-background-color:
"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.pause.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.label_bar.setStyleSheet("background-color: rgb(187, 188,
188);\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: 10px;")
# for generation in range(self.i + 1, self.generations):
#     while self.is_pause:
#         time.sleep(2)
#     self.alg.one_iteration(self.population_size,
self.p_mutation, generation)
#
#     self.update_progress_bar()
#     self.update_table()
#     data = read_data()

```

```

        #         label_text =
format_9x9_square(str_to_field(data[str(generation)][1]))
        #         self.screen.setText(label_text)
        #
self.plot_graph(self.data.get('best_fitness')[:int(generation)])
        #
        #         if self.alg.best_fitness_values[-1] == 243:
        #             self.i += generation
        #             print("Sudoku solved!")
        #             break
        # self.to_end.setEnabled(False)
        # self.one_step.setEnabled(False)
        # self.pause.setEnabled(False)
        # self.start_btn.setEnabled(True)
        # self.start_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.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.pause.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.label_bar.setStyleSheet("background-color: rgb(187,
188, 188);\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: 10px;")

# def start_until_the_end(self):
#     self.thread = EvolutionThread(
#         alg=self.alg,
#         start_gen=self.i,
#         total_gen=self.generations,
#         pop_size=self.population_size,
#         p_mutation=self.p_mutation,
#         data=self.data
#     )
#     self.thread.update_signal.connect(self.on_update)
#     self.thread.finished_signal.connect(self.on_finished)
#     self.thread.start()
#
# def on_update(self, label_text, plot_data):
#     self.update_progress_bar()
#     self.update_table()
#     self.screen.setText(label_text)
#     self.plot_graph(plot_data)
#
# def on_finished(self):
#     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")
#         print("Sudoku solved or finished!")
#
# class EvolutionThread(QThread):
#     update_signal = pyqtSignal(str, list) # прогресс, таб
лица, текст, график
#     finished_signal = pyqtSignal()
#
#     def __init__(self, alg, start_gen, total_gen, pop_size,
p_mutation, data):
#         super().__init__()
#         self.alg = alg
#         self.i = start_gen
#         self.generations = total_gen
#         self.population_size = pop_size
#         self.p_mutation = p_mutation
#         self.data = data
#         self._stop_flag = False
#
#     def run(self):
#         for generation in range(self.i, self.generations):
#             if self._stop_flag:
#                 break
#
#                 self.alg.one_iteration(self.population_size,
self.p_mutation, generation)
#                 label_text =
format_9x9_square(str_to_field(read_data()[str(generation)][1]))
#                 plot_data = self.data.get('best_fitness')[:generation
+ 1]

```

```

#
#         self.update_signal.emit(label_text, plot_data)
#
#         if self.alg.best_fitness_values[-1] == 243:
#             break
#
#         self.finished_signal.emit()
#
#     def stop(self):
#         self._stop_flag = True

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 math
from typing import overload, Literal, Union
from matplotlib.animation import FuncAnimation

import numpy as np
```



```

import matplotlib.pyplot as plt

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("Не корректные 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:
                        # С о х р а н я е м  ф и к с и р о в а н н у ю
я ч е й к у

                        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()
            # Включим интерактивный режим
            plt.ion() # Turn on interactive mode
            fig, ax = plt.subplots()
            line, = ax.plot([], [], 'r-') # Пустая линия для
г р а ф и к а

            ax.set_xlabel('П о к о л е н и е')
            ax.set_ylabel('Л у ч ш а я   п р и с п о с о б л е н н о с т ь')
            ax.set_title('Г р а ф и к   п р и с п о с о б л е н н о с т и   в   р е
а л ь н о м   в р е м е н и')

            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)

        self.update_plot(range(generation+1),
self.best_fitness_values, line, ax, fig)

        # 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 = []
            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)

            # very random func
            # elif random_number < 0.05:
            #     child = self.very_random_mutation()

        next_generation.append(child)

        self.population = next_generation
plt.ioff() # В ы к л ю ч а е м  и н т е р а к т и в н ы й  р е ж и
м

plt.show() # П о к а з ы в а е м  ф и н а л ь н ы й  г р а ф и к

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

"""
=====
ТЕСТОВЫЕ ФУНКЦИИ АЛГОРИТМА
=====
"""

def update_plot(self, generations, best_fitness, line, ax, fig):
    # best_fitness.append(fitness)

    line.set_data(generations, best_fitness)

```

```

ax.relim() # Пересчет границ
ax.autoscale_view() # Автомасштабирование
fig.canvas.draw() # Перерисовка
fig.canvas.flush_events() # Очистка буфера собы
т и й

```

```

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

```

```

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 = []
        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)

    # very random mutation
    # elif random_number < 0.05:
    #     child = self.very_random_mutation()

```

```

        next_generation.append(child)

    # / generating new population
    self.population = next_generation
# =====

# ===== 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.get_data('example.txt', 'f')
    gen_alg.GeneratePopulation(400)

    solution = gen_alg.main_cycle(10000, 400, 0.5)

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