**Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации**

**Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Курсовая работа по дисциплине

«Системы и Алгоритмы обработки данных»

Выполнил студент группы БВТ2205  
 Ковалевский Стас

Москва, 2024

Содержание

1. ЦЕЛЬ ...........................................................................................3

2. ЗАДАЧИ .....................................................................................3

3. ХОД РАБОТЫ ...........................................................................3

4. СОЗДАНИЕ ДЕТЕРМИНИРОВАННОГО АЛГОРИТМА ...10

5. СОЗДАНИЕ ГЕНЕТИЧЕСКОГО АЛГОРИТМА ..................13

6. РЕАЛИЗАЦИЯ ФУНКЦИЙ (ПРОДОЛЖЕНИЕ) ..................23

7. ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ..30

8. ВЫВОД .....................................................................................36

9. ЛИСТИНГ .................................................................................36

10. Список литературы ................................................................55ГЕНЕТИЧЕСКОГ

ЦЕЛЬ: Основной целью данной курсовой работы является создания программы для управления маршрутным парком. Найти оптимум при генерации расписания.

ЗАДАЧИ:

1. **Регистрация водителей**
   * Добавление водителей типа "A" и "B".
   * Хранение данных о водителях в отдельных списках.
   * Ввод имени водителя и его типа через графический интерфейс.
2. **Управление водителями**
   * Удаление водителей из списка.
   * Смена типа водителя с "A" на "B" и наоборот.
   * Обновление списка водителей в реальном времени через графический интерфейс.
3. **Настройка параметров расписания**
   * Ввод количества рейсов на день.
   * Установка времени маршрута в минутах.
   * Определение рабочих дней и выходных для водителей.
4. **Создание расписания с использованием оптимизированного алгоритма**
   * Генерация расписания для водителей типа "A", типа "B" и комбинированного списка (A + B).
   * Учет выходных дней: водители типа "A" не работают в выходные.
   * Проверка на достаточность водителей для выполнения заданного количества рейсов.
   * Генерация дополнительного расписания, если не хватает водителей или нарушаются временные интервалы.
5. **Создание расписания с использованием генетического алгоритма**
   * Генерация расписания для водителей типа "A", "B" и комбинированного списка.
   * Реализация ключевых этапов генетического алгоритма:
     + **Crossover (скрещивание)**
     + **Mutation (мутация)**
     + **Fitness-функция** для оценки расписания.
   * Поиск оптимального расписания на основе количества выполненных рейсов.
   * Завершение алгоритма при отсутствии улучшений в расписании после нескольких поколений.
6. **Обработка ошибок и уведомления**
   * Проверка корректности ввода данных (например, количество рейсов и время маршрута).
   * Вывод сообщений о невозможности создания расписания при нехватке водителей.
   * Подсчет необходимого количества дополнительных водителей при недостатке ресурсов.
7. **Графический интерфейс пользователя (GUI)**
   * Форма для ввода данных о водителях, днях недели, количестве рейсов и времени маршрута.
   * Поле для вывода сгенерированного расписания в текстовом формате.
   * Кнопки для выполнения следующих действий:
     + Добавление водителей.
     + Установка времени маршрута.
     + Генерация расписания для водителей типа "A", "B" и комбинированного списка.
     + Генерация расписания с использованием генетического алгоритма.
     + Управление водителями (удаление и смена типа).
     + Сброс всех данных.
8. **Учет времени маршрутов и перерывов**
   * Расчет времени начала и окончания маршрутов.
   * Учет минимального времени перерывов между маршрутами.
   * Определение доступных временных слотов для маршрутов.
9. **Вывод расписания в текстовом виде**
   * Отображение итогового расписания в текстовом формате с указанием:
     + Водителя.
     + Типа маршрута.
     + Времени начала и окончания маршрута.
     + Количества маршрутов на смену.
10. **Сброс данных**
    * Возможность очистки всех полей и текстовых областей для повторного ввода данных.

ХОД РАБОТЫ: В ходе работы я буду использовать такие библиотеки как «pandas» и «tkinter».

Библиотека «pandas» используется для **хранения и представления расписания** в табличном формате и удобного вывода результатов.

Библиотека «tkinter» отвечает за **графический интерфейс**, предоставляя пользователю удобный способ взаимодействия с программой.

Часть 1. Написание функций и первого алгоритма – детермизированного.

Итак, для начала я инициализирую списки для хранения данных о водителях и словарь чтобы харнить данные о сменах

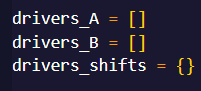


Рисунок 1 – инициализация списков и словаря

Далее создаю **ключевые параметры для маршрутов и смен водителей**.

По условию задачи: 2 типа маршрутов – до конечной (из пункта А в пункт Б) и замкнутый, он же кольцевой (из пункта А в пункт Б и обратно из пункта Б в пункт А). Также прописываю время рабочих смен: для водителей типа А смена составляет 8 часов (при отсутствии рабочих смен в выходные) и 12 часов для водителей типа Б – где рабочие смены могут быть полную неделю.

По умолчанию пропишу также время в пути на рейсе – 60 (минут).

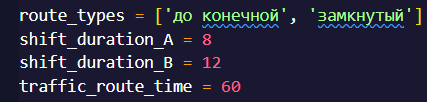


Рисунок 2 – ключевые параметы

Также ввожу ограничительные параметры: рабочая смена начинается с 6:00 утра и заканчиваятся не позднее 3:00 ночи.



Рисунок 3 – временные параметры

Теперь определяю функцию которая проверяет, является ли выбранный день выходным.

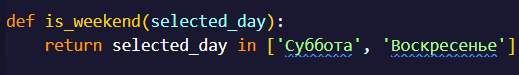


Рисунок 4 – функция проверки выходного дня

Теперь создам функцию которая рассчитывает **время окончания маршрута** на основе времени начала и продолжительности.

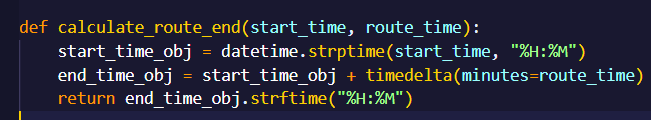


Рисунок 5 – функция рассчёта времени окончания маршрута

Следующая функция которая нормализует временные интервалы. Она решает проблему, когда **время окончания маршрута меньше времени начала** (что может происходить при переходе через полночь).

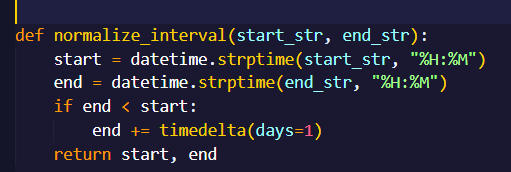


Рисунок 6 – функция нормализации интервалов

Еще одна функция проверяет, **перекрывается ли заданный временной интервал** с уже занятыми интервалами.

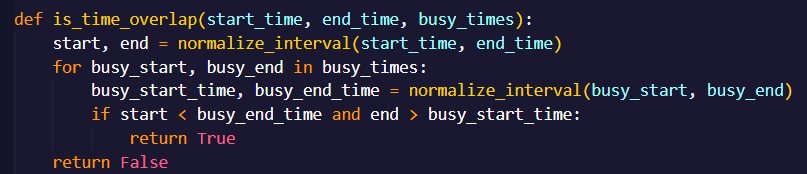


Рисунок 7 – функция проверки

Теперь создаю функцию которая находит **свободные временные интервалы** (слоты) для выполнения маршрутов у водителей на основе их занятости.

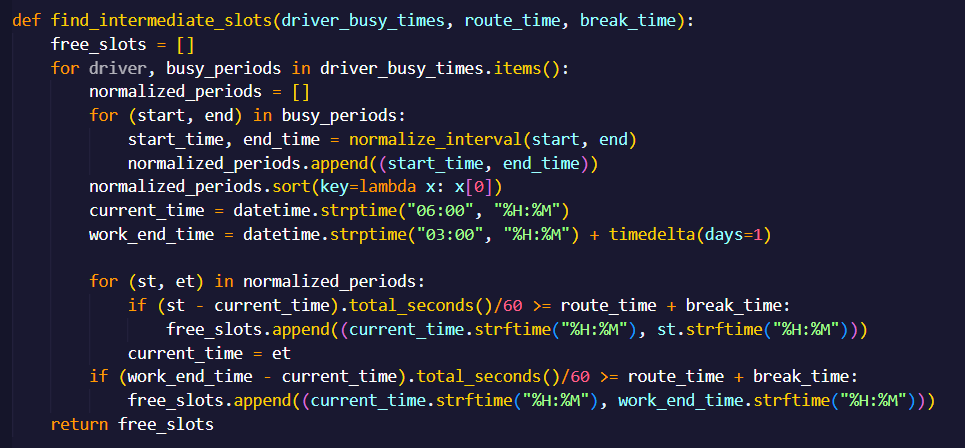


Рисунок 8 – функция поиска свободных интервалов

Добавлю функцию под рассчёт, **сколько дополнительных водителей** требуется для выполнения заданного количества маршрутов за смену.

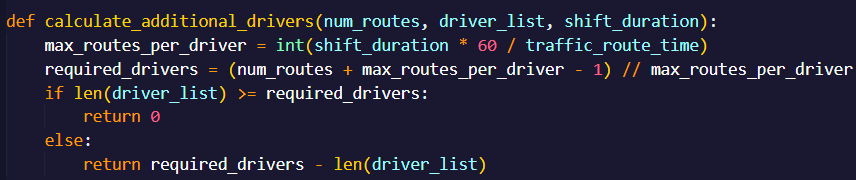


Рисунок 9 – функция рассчёта доп. Водителей

И нужно сделать обработку на то, что расписание нельзя сгенерировать из-за нехватки водителей.

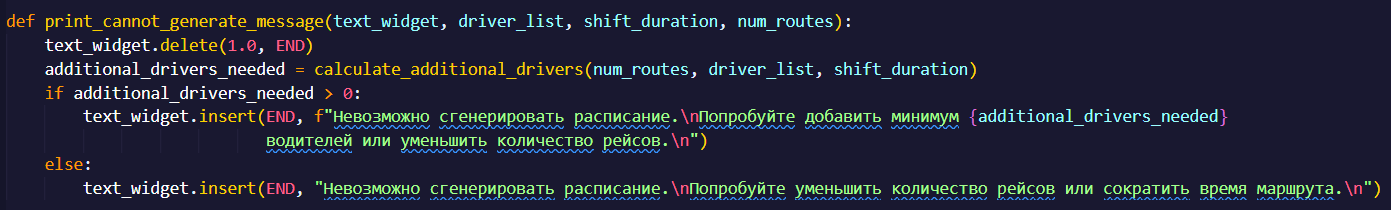


Рисунок 10 – функция обработки исключения

И такую же функцию только для расписания которое будет генерироваться с помощью генетического алгоритма.

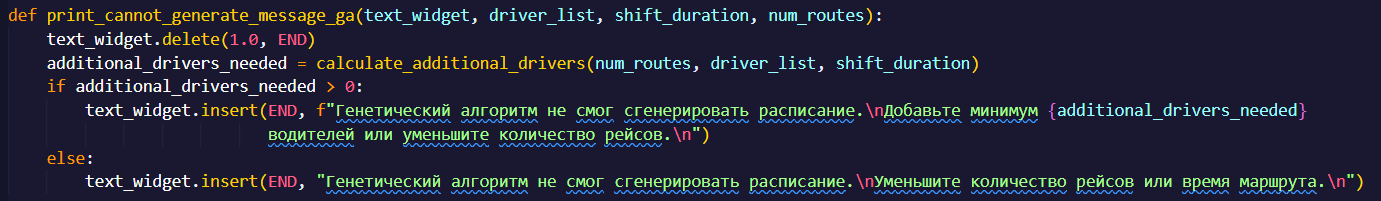


Рисунок 11 – обработка исключения

Теперь функция дляпроверки, можно ли назначить новый маршрут водителю, учитывая ограничения времени, занятости, длительности смены и перерывов.

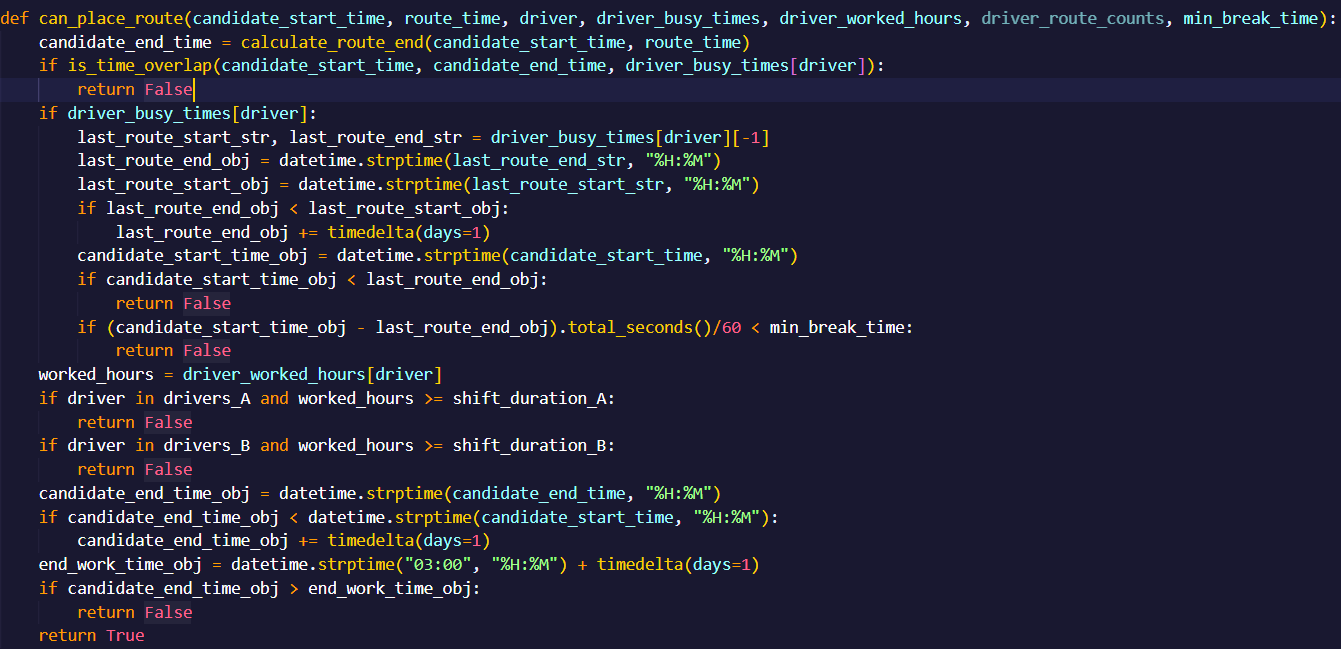


Рисунок 12 – функция проверки новых маршрутов

Также нужна функция которая пытается найти **свободный временной слот** для маршрута, который можно назначить одному из водителей с учетом всех ограничений.

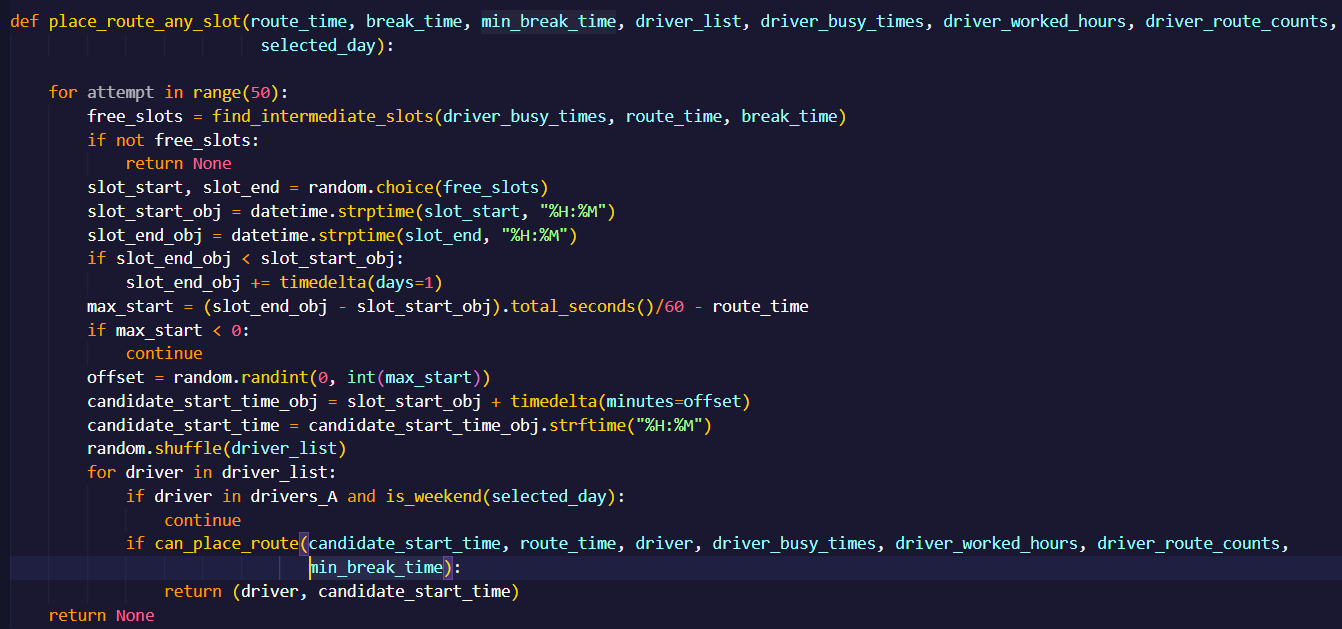


Рисунок 13 – функция поиска свободного слота

СОЗДАНИЕ ДЕТЕРМИЗИРОВАННОГО АЛГОРИТМА

Как он будет использоваться:

**Прямой перебор маршрутов**:

* Функция **последовательно** обрабатывает каждый маршрут из общего количества **num\_routes**.
* Для каждого маршрута проверяется, можно ли его назначить в текущее время или в другие слоты.

**Определённые шаги**:

* Логика выбора водителя, проверки времени и назначения маршрута **фиксирована**.
* Если время **выходит за пределы** рабочего дня, вызывается функция **place\_route\_any\_slot**, которая также действует последовательно с несколькими попытками.

**Фиксированная структура решений**:

* Если данные (список водителей, время смены и количество маршрутов) **не меняются**, то результат работы алгоритма всегда будет одинаковым, несмотря на некоторые случайные элементы.

**Детерминированная основа**:  
Алгоритм имеет фиксированную структуру действий и условия, по которым маршруты назначаются.

**Элементы случайности**:  
Используются для повышения равномерности распределения маршрутов и водителей (перемешивание списка и случайный выбор маршрута).

Это делает его более гибким и справедливым в распределении.

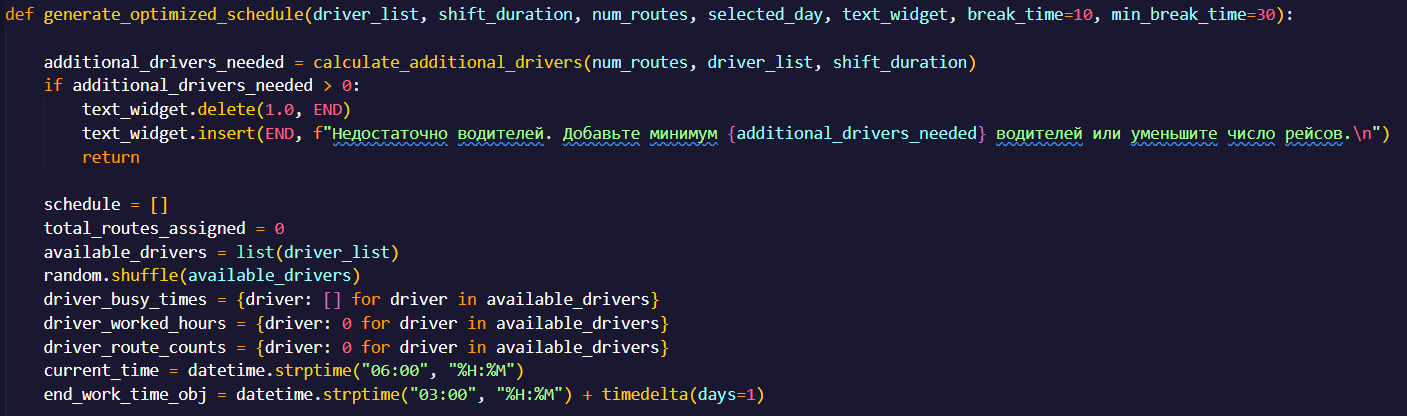


Рисунок 14 – реализация детермизированного алгоритма

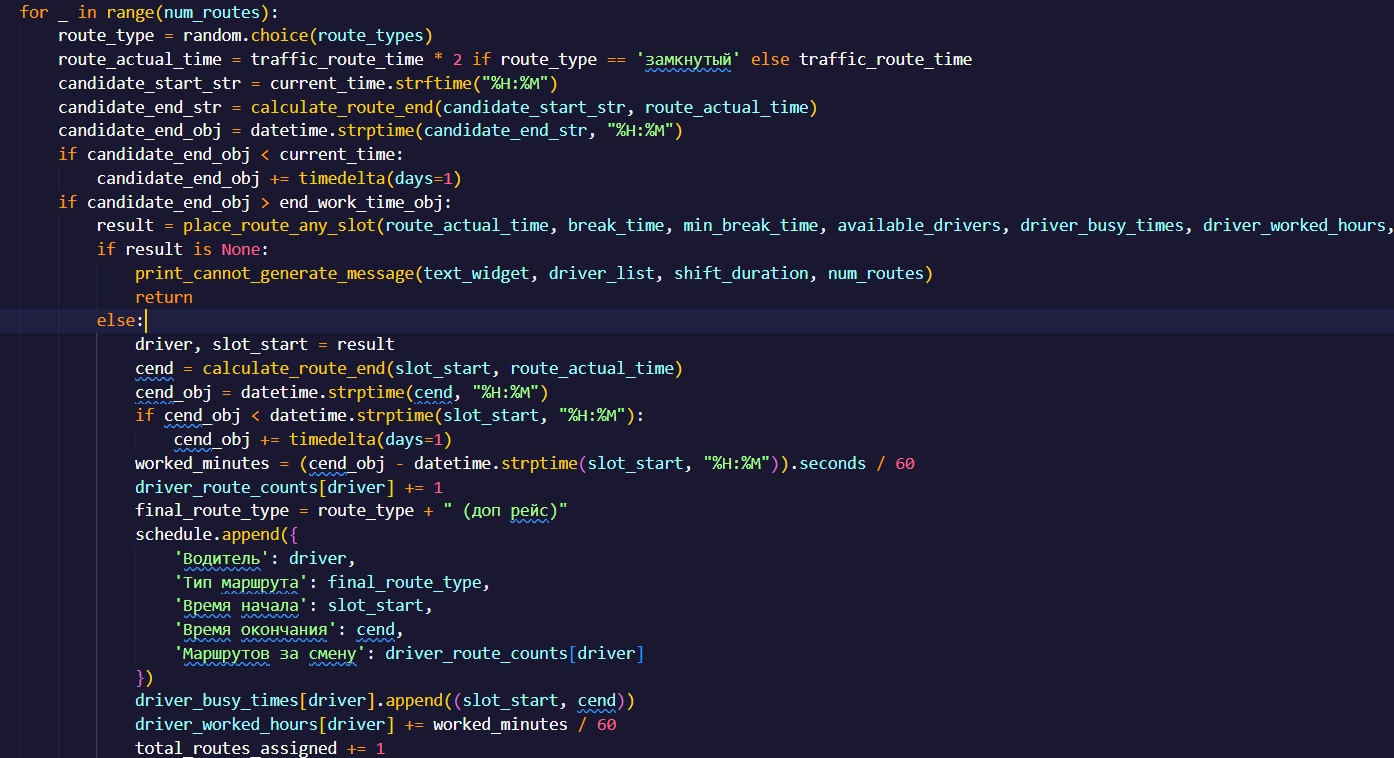


Рисунок 14.1 – реализация детермизированного алгоритма

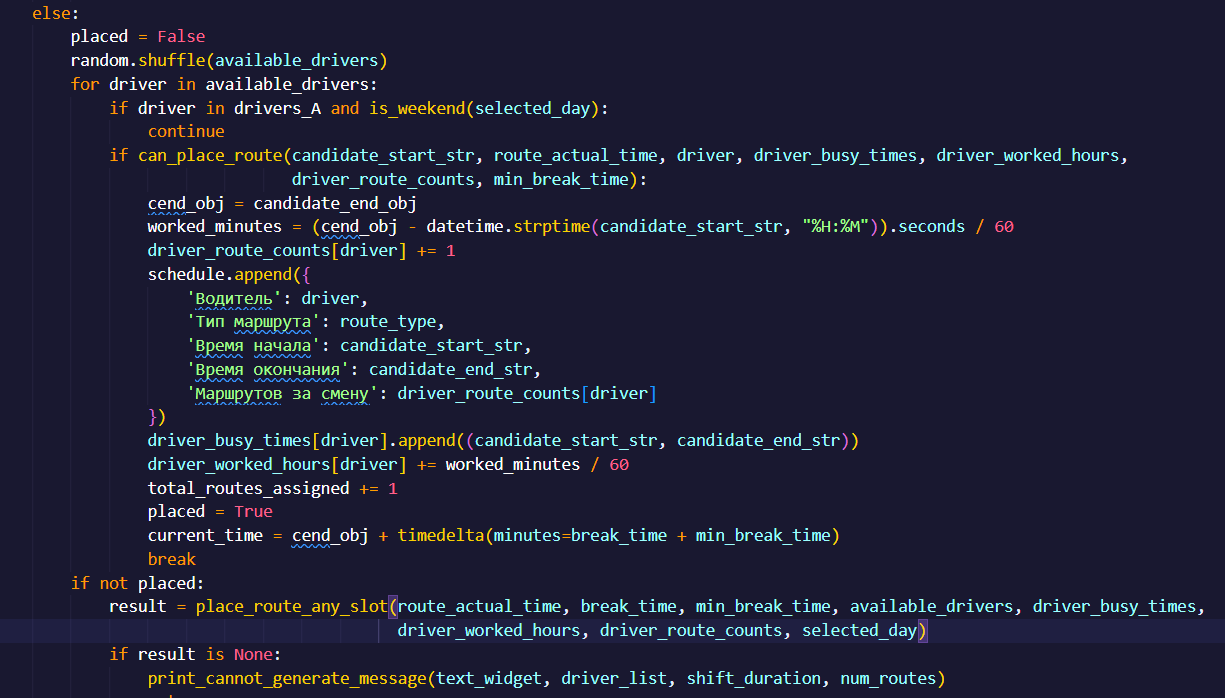


Рисунок 14.2 – реализация детермизированного алгоритма

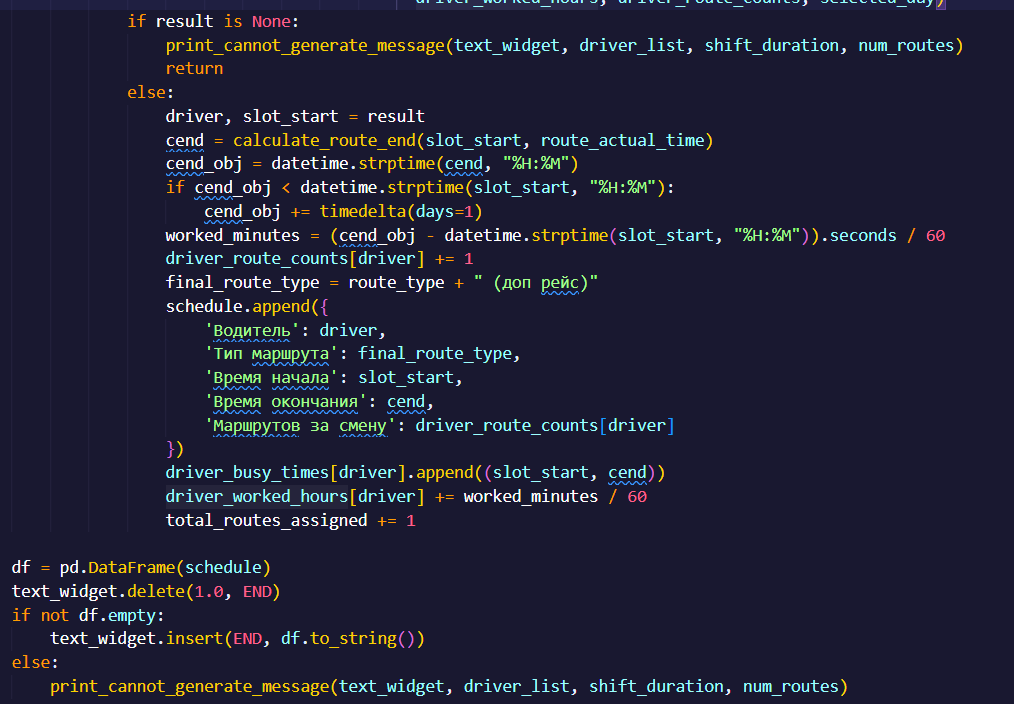


Рисунок 14.3 – реализация детермизированного алгоритма

РЕАЛИЗАЦИЯ ГЕНЕТИЧЕСКОГО АЛГОРИТМА ДЛЯ РАСПИСАНИЯ

**Теория**

Генетический алгоритм (**ГА**) — это метод **оптимизации и поиска**, вдохновленный процессами **естественного отбора** в биологии. Он особенно полезен для **сложных задач**, где перебор всех возможных вариантов занимает много времени, а детерминированные методы оказываются неэффективными.

### ****Основные принципы генетического алгоритма****

1. **Популяция**:  
   ГА начинает с **набора решений**, который называется **популяцией**.
   * Каждое решение в популяции называется **индивидом** или **особью**.
   * Эти решения могут быть случайными или частично оптимизированными.
2. **Хромосомы**:
   * Каждое решение представляется в виде **хромосомы** – структуры, которая содержит параметры решения.
   * Например, в задаче расписания хромосома будет представлять набор маршрутов для водителей.
3. **Оценка качества (Функция Fitness)**:
   * Каждому решению присваивается **оценка качества**.
   * Функция fitness определяет, насколько "хорош" данный индивид.
   * В задаче расписания это может быть количество назначенных маршрутов.
4. **Отбор**:
   * Лучшие решения из текущей популяции (особи с высокой оценкой качества) **отбираются** для дальнейшего размножения.
   * Это имитирует естественный отбор: выживают **самые приспособленные**.
5. **Скрещивание (Crossover)**:
   * Из двух "родителей" создается **новое решение** (ребенок), комбинируя их параметры.
   * Это позволяет передавать **лучшие черты** обоих родителей следующему поколению.
6. **Мутация**:
   * Для сохранения **разнообразия** в популяции у части решений случайно изменяются параметры.
   * Мутация помогает избежать "застревания" в локальном минимуме и позволяет находить новые варианты решений.
7. **Эволюция поколений**:
   * Новый набор решений (потомков) заменяет старую популяцию.
   * Процесс отбор → скрещивание → мутация повторяется несколько раз (поколений), пока не будет достигнуто оптимальное или приемлемое решение.

### ****Фишки генетического алгоритма****

1. **Эффективность для сложных задач**:  
   Генетический алгоритм не перебирает все возможные варианты, а постепенно улучшает решения, фокусируясь на перспективных.
2. **Гибкость**:  
   Подходит для задач с **многими ограничениями** и критериями, где классические методы неэффективны.
3. **Элементы случайности**:  
   Случайные мутации и комбинации решений позволяют исследовать **разнообразие решений** и избегать локальных минимумов.
4. **Приближенное решение**:  
   ГА не всегда находит идеальное решение, но часто находит **достаточно хорошее** решение за приемлемое время.

### ****Как генетический алгоритм внедряется в задачу составления расписания****

#### **Цель**

Оптимально распределить маршруты среди водителей, чтобы:

* Назначить **максимальное количество маршрутов**.
* Учитывать **ограничения времени** и перерывов.
* Сократить количество водителей, работающих сверх нормы.

#### **Этапы работы генетического алгоритма в задаче**

1. **Инициализация популяции**:
   * Создается **начальный набор расписаний**.
   * Каждое расписание случайно распределяет маршруты между водителями с учетом ограничений.
   * Например, одно расписание может распределить 80% маршрутов корректно, другое — только 50%.
2. **Функция оценки (fitness)**:
   * Для каждого расписания вычисляется количество **успешно назначенных маршрутов**.
   * Решения с **большим количеством назначенных маршрутов** получают высокую оценку качества.
3. **Отбор лучших решений**:
   * Из всей популяции отбираются **лучшие расписания** (с самой высокой оценкой).
   * Эти "родители" будут участвовать в создании новых решений.
4. **Скрещивание**:
   * Два лучших расписания (родители) комбинируются для создания нового "ребенка".
   * Например: первая половина расписания берется от первого родителя, а вторая половина — от второго.
5. **Мутация**:
   * В небольшом числе расписаний случайно изменяются маршруты.
   * Например, один маршрут может быть переназначен другому водителю.
   * Это добавляет разнообразие и позволяет находить **неожиданные улучшения**.
6. **Создание нового поколения**:
   * Новое поколение решений заменяет старую популяцию.
   * Процесс повторяется несколько раз (поколений), чтобы улучшать результаты.
7. **Остановка алгоритма**:
   * Алгоритм завершается, если:
     + Найдено **идеальное решение** (все маршруты назначены).
     + Не происходит улучшений в течение нескольких поколений.
8. **Выбор лучшего результата**:
   * Выводится расписание с **наибольшим количеством назначенных маршрутов** из последнего поколения.

### ****Преимущества генетического алгоритма для задачи расписания****

1. **Гибкость**:  
   Генетический алгоритм легко учитывает **множество ограничений**:
   * Время начала и конца смен.
   * Перерывы между маршрутами.
   * Разные категории водителей (например, A и B).
2. **Эффективность**:  
   Вместо перебора всех возможных комбинаций ГА находит **близкое к оптимальному решение** за несколько поколений.
3. **Масштабируемость**:  
   Хорошо работает с **большими объемами данных** (много маршрутов и водителей).
4. **Универсальность**:  
   ГА можно адаптировать под любые изменения условий задачи.

Генетический алгоритм в этой задаче позволяет распределить маршруты **оптимально и гибко**, преодолевая ограничения и находя **лучшие решения** даже в сложных ситуациях, где детерминированный подход может не справиться.

**Практическая составляющая в коде**

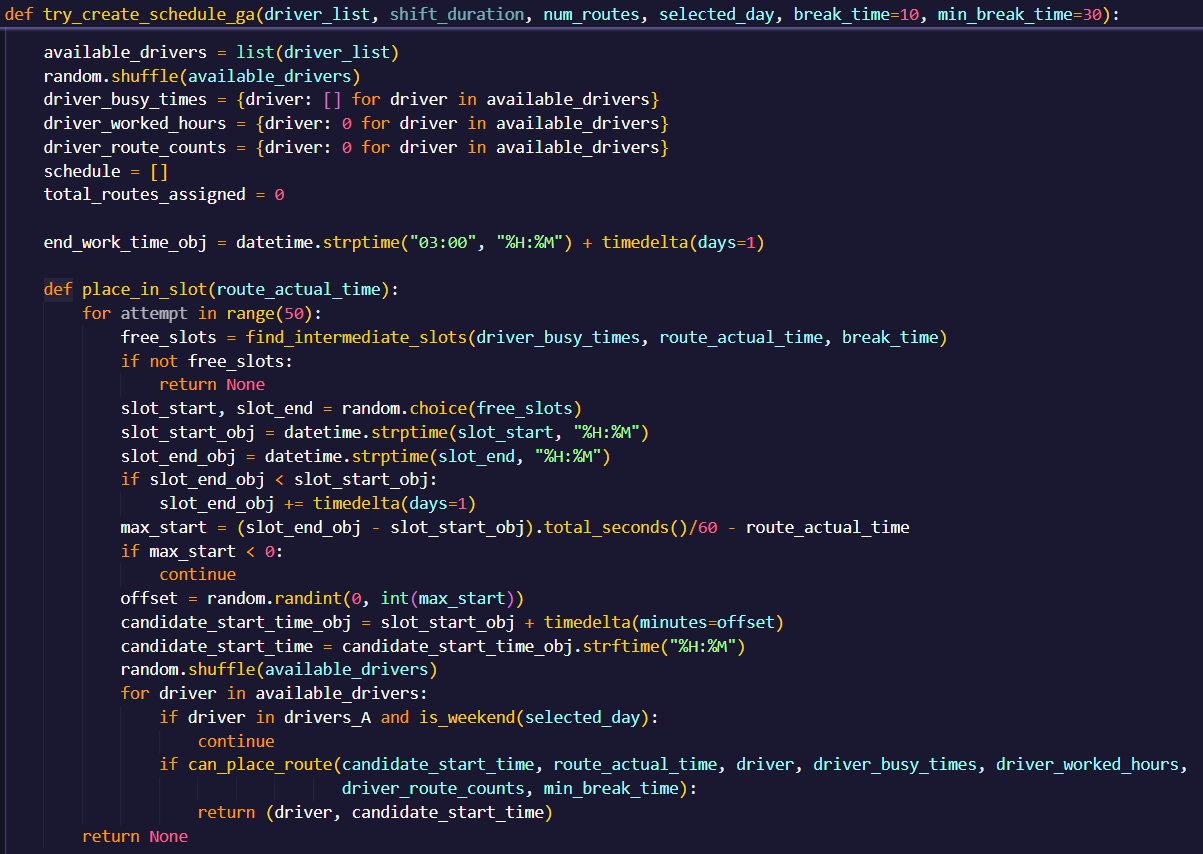
****

Рисунок 15 – реализация генетического алгоритма

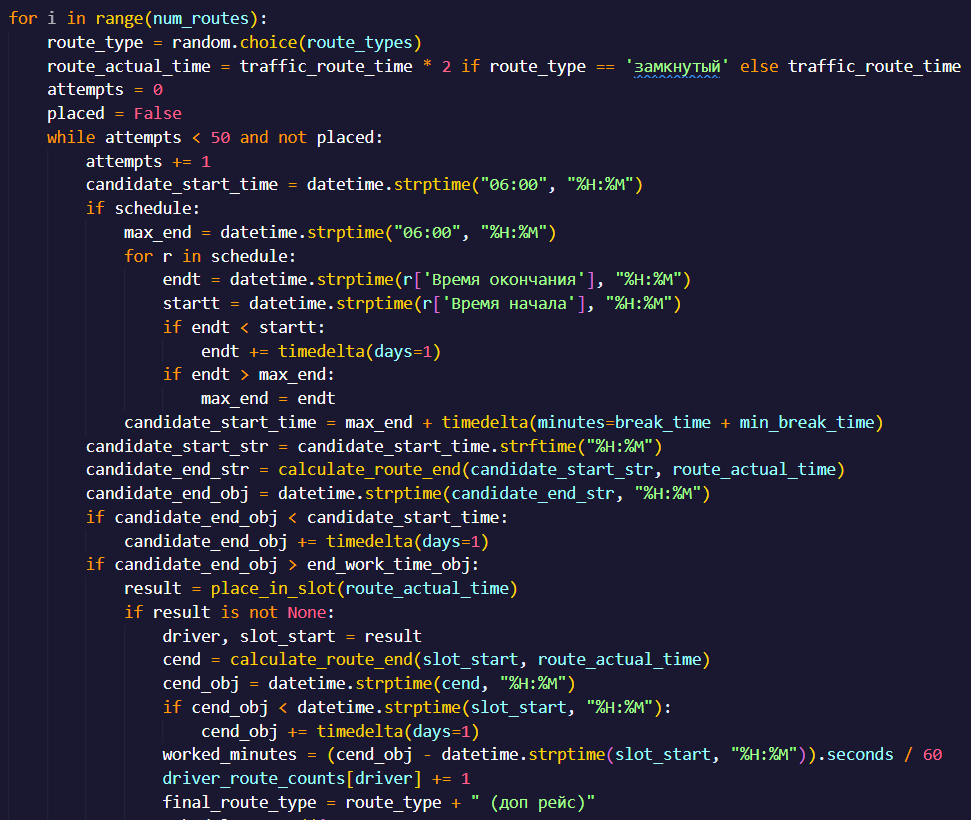
****

Рисунок 15.1 – реализация генетического алгоритма

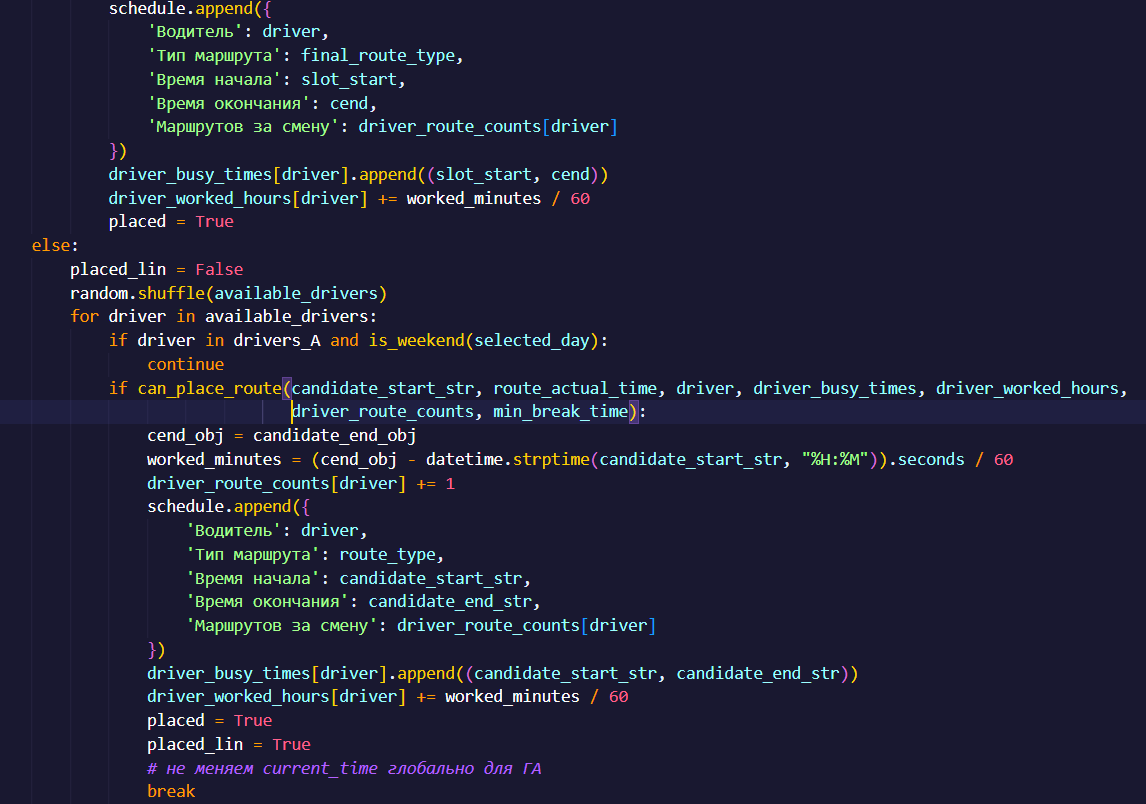
****

Рисунок 15.2 – реализация генетического алгоритма

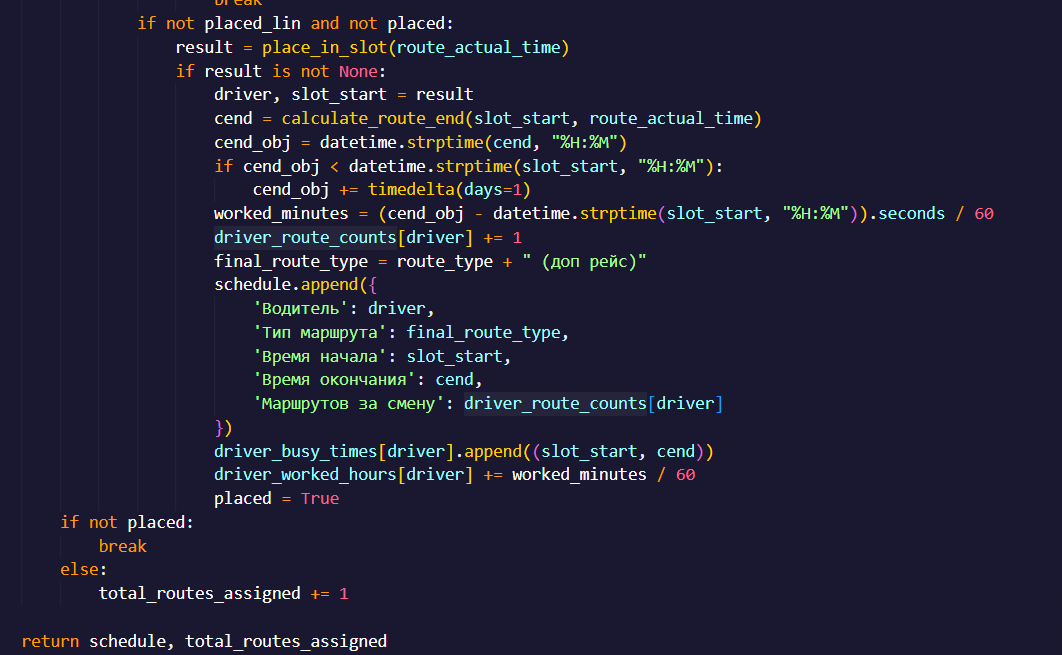
****

Рисунок 15.3 – реализация генетического алгоритма

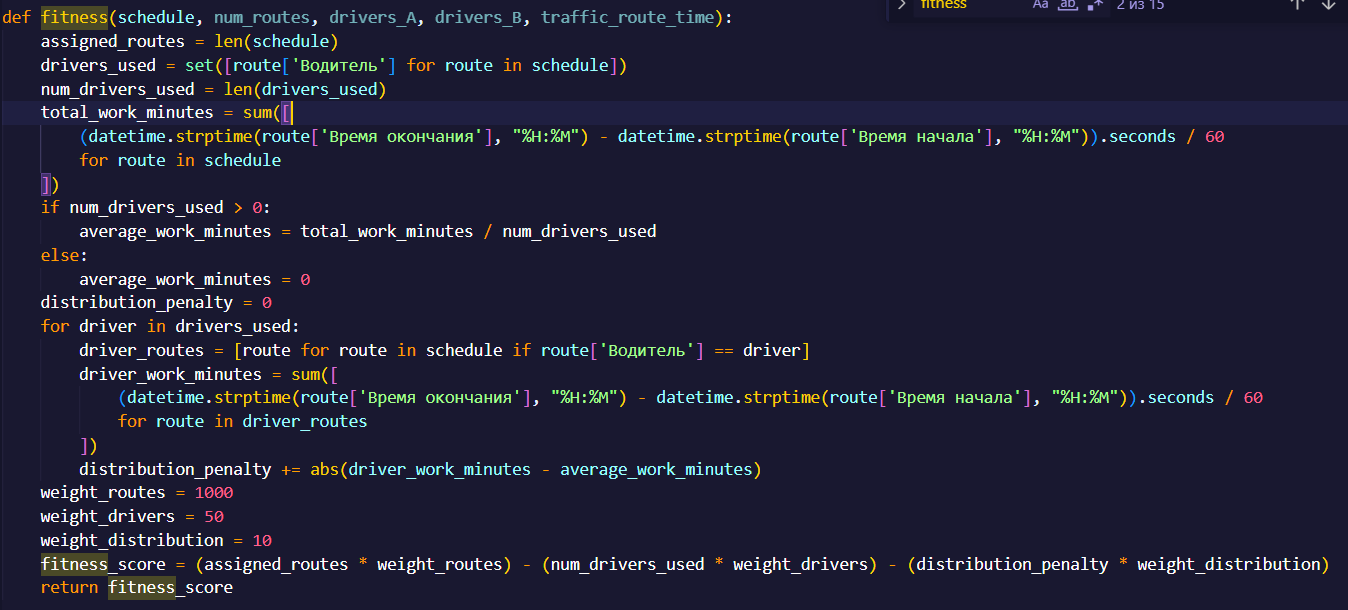
****

Рисунок 15.4 – реализация ключевой функции fitnes

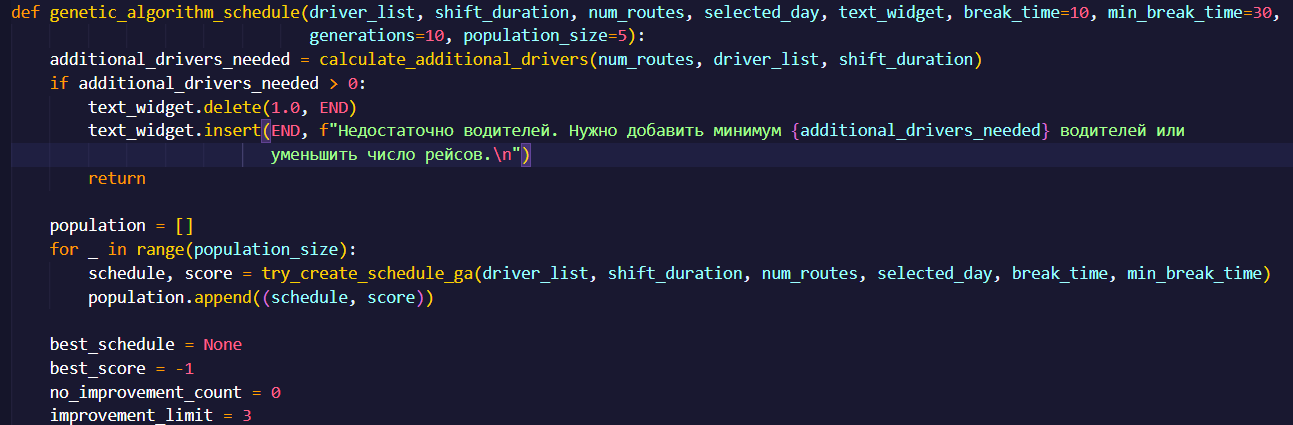
****

Рисунок 15.5 – реализация генетического алгоритма

****

Рисунок 15.6 – реализация генетического алгоритма

Ключевые моменты в моей реализации генетического алгоритма.

1. **Инициализация популяции**:
   * Генерируется **начальное множество решений** (популяция).
   * Используется функция **try\_create\_schedule\_ga**, которая случайным образом формирует расписания с учетом заданных ограничений.
   * Каждое расписание — это потенциальное решение задачи и **особь популяции**.
2. **Функция оценки качества (Fitness)**:
   * Оценка каждого расписания определяется функцией **fitness**, которая возвращает количество успешно назначенных рейсов.
   * Чем больше рейсов удалось распределить, тем выше качество расписания.
3. **Отбор лучших решений**:
   * Популяция сортируется по значению функции fitness.
   * Решения с **наибольшим числом рейсов** переходят в следующее поколение (это имитирует отбор "сильнейших" особей).
4. **Скрещивание (Crossover)**:
   * Для создания нового расписания (ребенка) комбинируются два лучших решения (родителя).
   * Функция **crossover** делит хромосому (расписание) пополам и объединяет части двух родителей.
5. **Мутация**:
   * Случайные изменения применяются к части расписаний, чтобы сохранить **генетическое разнообразие** и избежать локальных минимумов.
   * Реализовано в функции **mutate**, где случайно переставляются элементы расписания.
6. **Эволюция поколений**:
   * Новое поколение формируется из лучших решений предыдущего поколения и их потомков.
   * Это повторяется в цикле для заданного числа **поколений** или до тех пор, пока не будет достигнут лучший результат.
7. **Критерии остановки**:  
   Алгоритм завершает выполнение, если:
   * Найдено решение, где **все рейсы распределены**.
   * В течение нескольких поколений отсутствуют улучшения (параметр **improvement\_limit**).

**Ключевые моменты кода**

1. **Создание начального расписания**:
   * Функция **try\_create\_schedule\_ga**:
     + Генерирует расписание, перебирая свободные временные слоты и назначая маршруты водителям.
     + Использует вспомогательные функции **place\_in\_slot** и **can\_place\_route** для проверки допустимости назначения маршрута.
2. **Формирование новой популяции**:
   * **Отбор**: Два лучших решения (элита) автоматически переходят в следующее поколение.
   * **Скрещивание**: Функция **crossover** комбинирует решения двух родителей.
   * **Мутация**: Небольшое случайное изменение расписания, чтобы добавить вариативность.
3. **Оценка качества**:
   * Каждое расписание оценивается функцией **fitness**, которая подсчитывает количество назначенных рейсов.
4. **Улучшение поколений**:
   * Алгоритм ищет лучшие решения в течение нескольких итераций (поколений), каждый раз улучшая результат.
5. **Вывод результата**:
   * Если найдено хорошее решение, оно выводится как лучшее из текущего поколения.
   * Если идеального результата нет, возвращается **"наилучшее достигнутое решение"**.

**Особенности и оптимизация**

1. **Обработка ограничений**:
   * Водители типа **A** не работают по выходным.
   * Временные ограничения на начало и окончание маршрутов.
   * Учет перерывов между маршрутами.
2. **Адаптация для задачи**:
   * Генетический алгоритм адаптирован под задачу расписания с конкретной структурой решения (хромосомы).
   * Используется случайное добавление маршрутов и проверка допустимости на каждом шаге.
3. **Случайность для разнообразия**:
   * **Перемешивание водителей** и **случайные мутации** позволяют исследовать разные решения, чтобы избежать локального минимума.
4. **Простая структура**:
   * Четкое разделение на функции: создание расписания, скрещивание, мутация и оценка

РЕАЛИЗАЦИЯ ФУНКЦИЙ (ПРОДОЛЖЕНИЕ)

Следующая функция служит для управления списком водителей (удаления и изменения типа водителя) в программе. Она содержит две вложенные функции: первая - Удаляет водителя из соответствующего списка и обновляет интерфейс, а вторая Перемещает водителя из одного в другой и обновляет интерфейс.

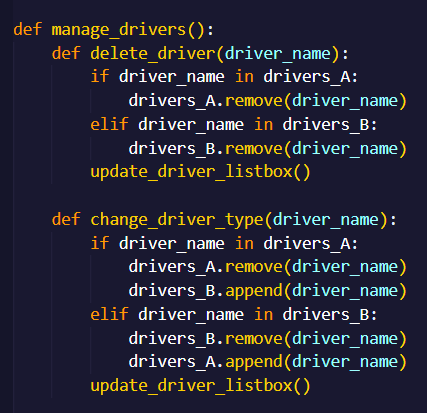


Рисунок 16 – функция списка водителей

Еще одна функция,которая обновляет интерфейс для управления списками водителей, предоставляя удобный и динамический способ удалять и менять их типы.



Рисунок 17 – функция обновления интерфейса

Далее напишу функции которые отвечают за **генерацию расписаний** для водителей **типа A** и **типа B** соответственно.

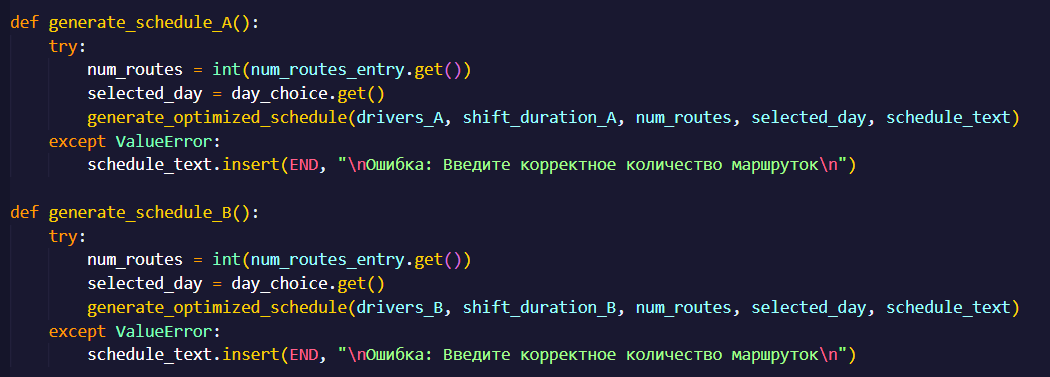


Рисунок 18 – функции генерации расписания

И еще одна такая же функция только для генерации совместного раписания из водителей типа А и типа Б. Также присутствует обработка ситуации «выходного дня» в которой водители типа А работать не могут.

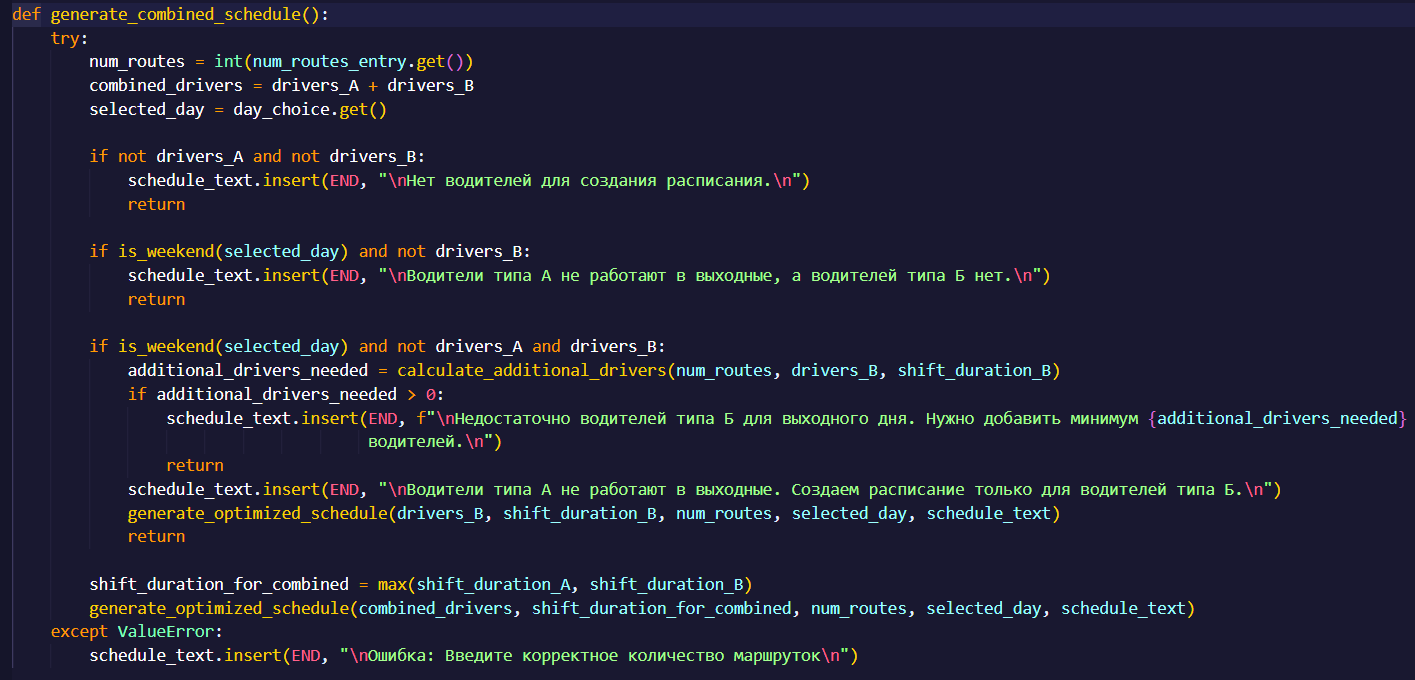


Рисунок 19 – генерация совместного расписания

И точно такие же по аналогии функцию создаю для генетического расписания.

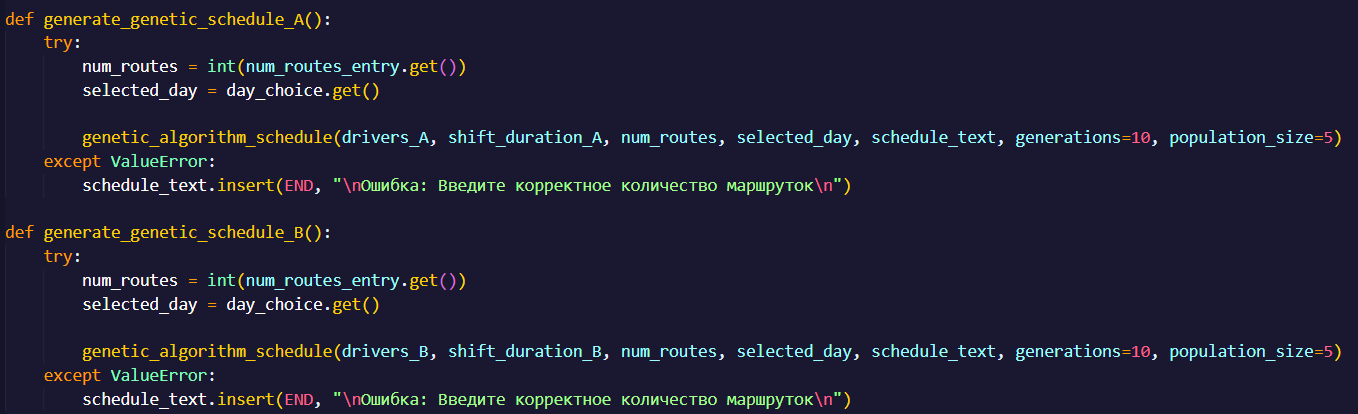


Рисунок 20 – генерация для каждого типа водителей

И для совместной генерации.

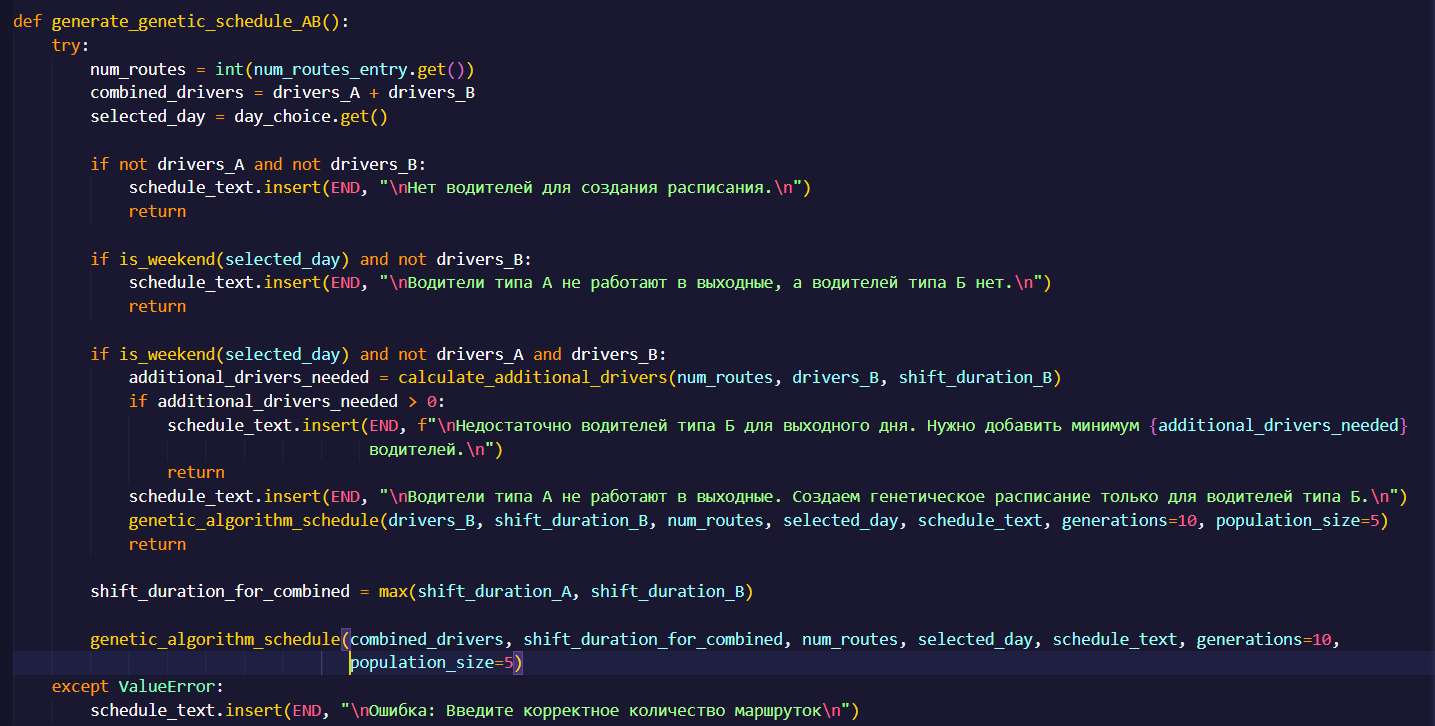


Рисунок 21 – функция для совместной генерации

Сделаю еще функци заглушки.



Рисунок 22 – функция заглушки

Следующая функция для полного сброса параметров по рейсам и очистки поля расписания.

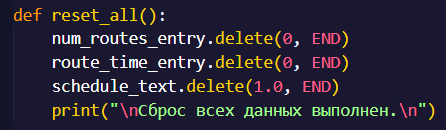


Рисунок 23 – функция сброса

Считаю нужным реализовать механизм для установки глобального времени маршрута на основе пользовательского ввода с проверкой корректности данных и сообщением о результате.

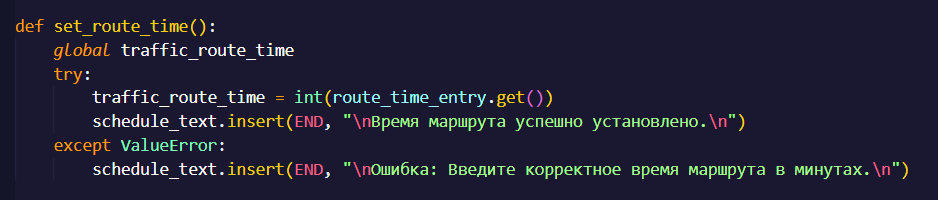


Рисунок 24 – функция механизма для установки времени

И теперь одна из главных функций: GUI - отвечает за создание и отображение графического интерфейса пользователя (GUI) для приложения с использованием библиотеки tkinter. Этот интерфейс позволяет регистрировать водителей, выбирать их тип и вводить другие параметры.

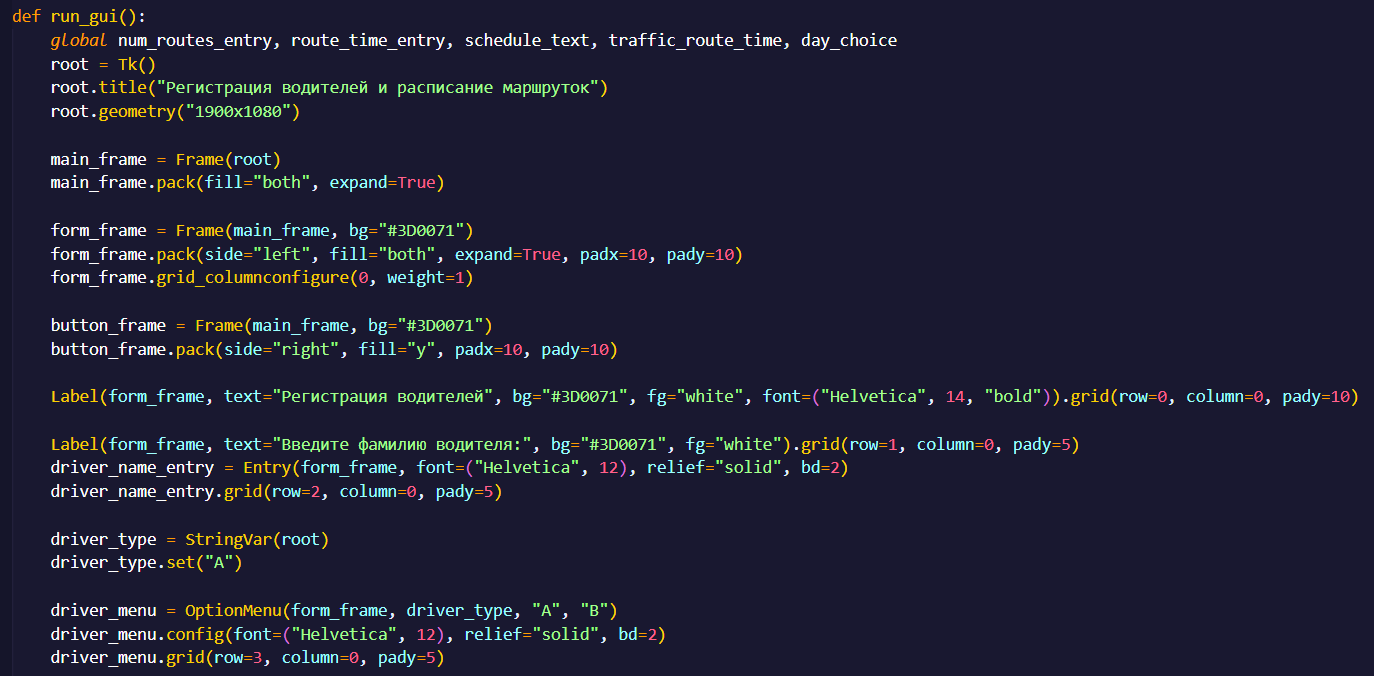


Рисунок 25 – функция GUI

Также присутствует функция отвечает за регистрацию водителей в графическом интерфейсе. Она обрабатывает ввод пользователя, сохраняет водителей в соответствующий список (тип A или B) и отображает статус операции.

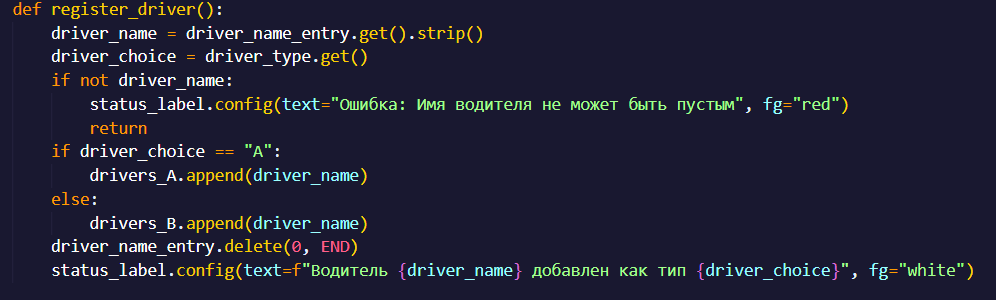


Рисунок 26 – функция регистрации водителей в GUI

И заключительное создание виджетов (текстовых полей, кнопок и меток) для управления расписанием водителей. Виджеты размещаются в графическом интерфейсе с использованием библиотеки **tkinter**.

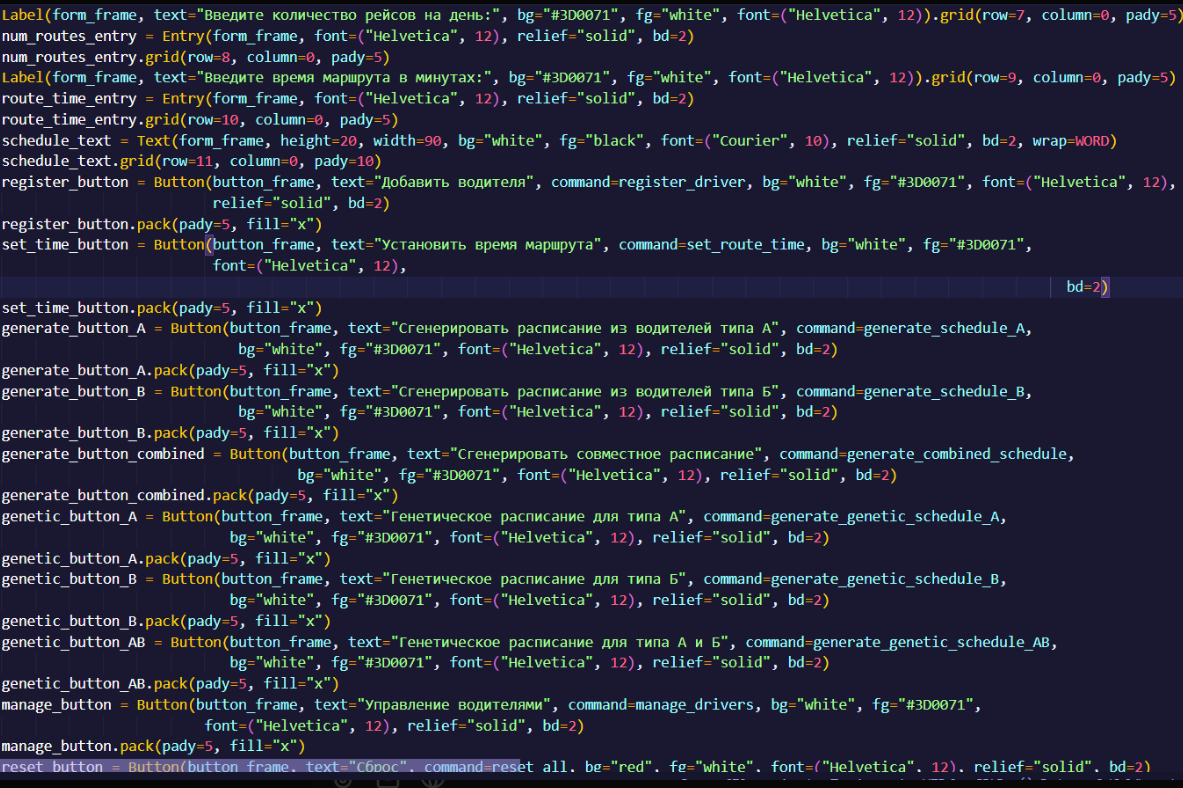


Рисунок 27 – создание виджетов

Заключительный блок для инициализации и запуска графического интерфейса.

**root.mainloop()** запускает основной цикл Tkinter, который "оживляет" интерфейс и удерживает его открытым до тех пор, пока пользователь не закроет окно.

**run\_gui()** подготавливает и отображает интерфейс перед его запуском.

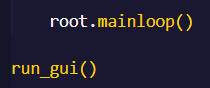


Рисунок 28 – инициализация + запуск интерфейса

Код для приложения по созданию расписания полностью написан, включая функции для управления водителями, генерации расписаний с использованием детерминированного и генетического алгоритмов, а также полноценного графического интерфейса на **Tkinter**.

На данном этапе можно перейти к **тестированию** приложения для проверки его работы, корректности генерации расписаний и выявления возможных ошибок или недочетов.

ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ

Для начала необходимо убедиться что все библиотеки установлены. Если не установлены то с помощью команды pip install +название библиотеки можно установить. Например: pip install tkinter

После установки библиотек можно перейти к запуску файла. Команда: python schedule\_generator\_test.py

Программа запускается.

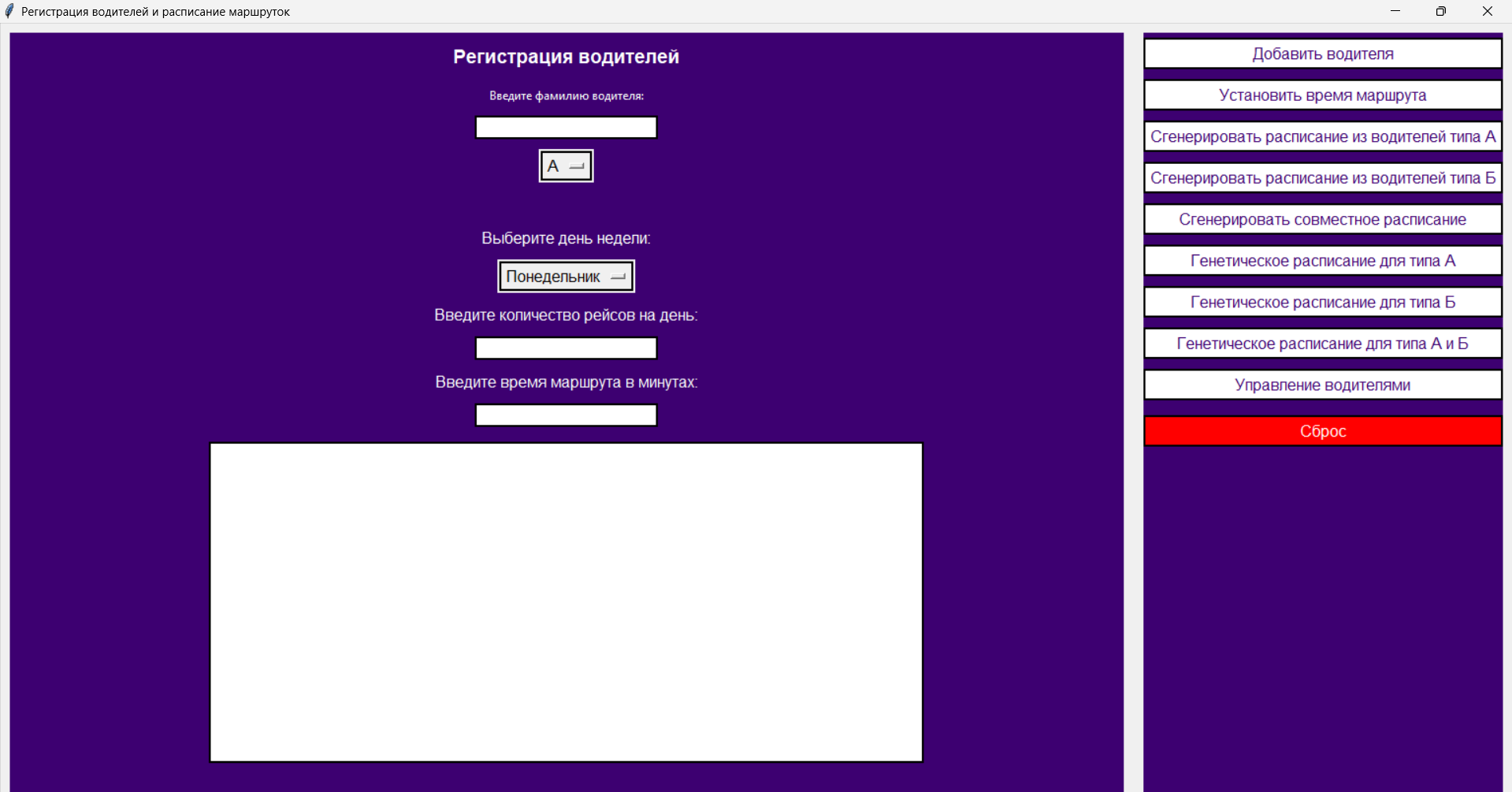


Рисунок 29 – успешный запуск программы

Свое приложение я реализовал в 2 поля: в левой половине находятся поля для вввода и кнопки, а также поле в котором будет генерироваться расписание. Вторая половина (справа) – панель всех кнопок.

Итак,начнем тестирование. Начну сразу с обработки различных ошибок.

Первая обработка: при нажатии на кнопку Добавить водителя (если поле для ввода имени сотрудника пусто) будет выпадать ошибка о том, что поле водителя не может быть пустым. И обязательно надо ввести имя в соответствующее поле.

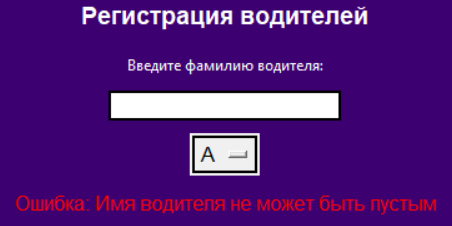


Рисунок 30 – обработка пустого ввода имени

Следующая обработка: при нажатии на кнопку установить время маршрута – если оно также не вводилось в соответсвущее поле, то предупреждение об этом появляется сразу в терминале.

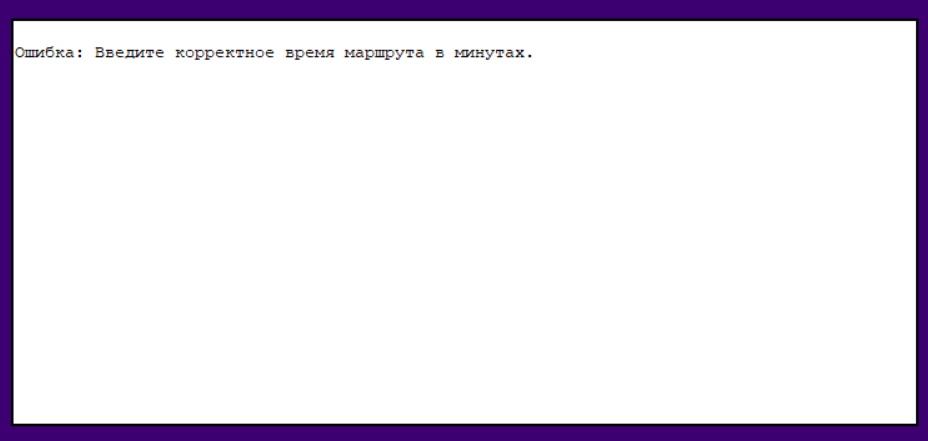


Рисунок 31 – обработка отсутствия введенного времени

Следующая обработка: если не указано количество маршрутов, но нажата одна из кнопок по созданию расписания, то также происходит выпадающее предупреждение с просьбой проверить эти параметры.

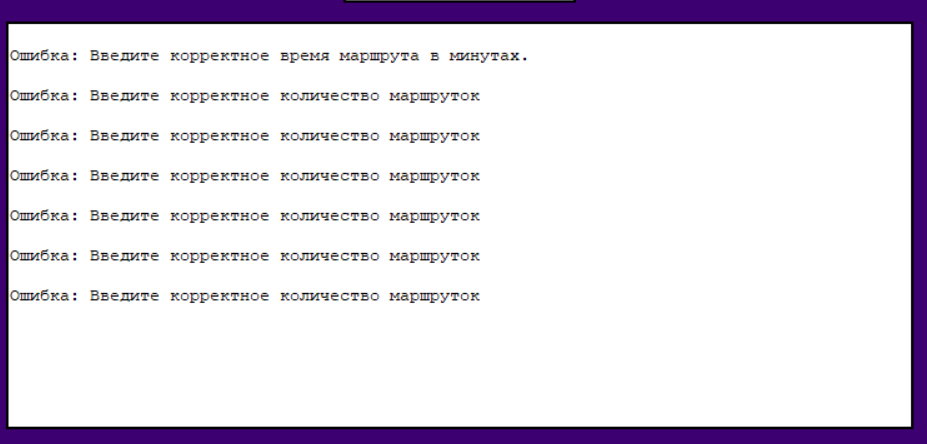


Рисунок 32 – обработка пустой генерации

Ещё одна обработка: предположим,что водитель успешно добавлен,время маршрута и количество рейсов успешно установлено, но при нажатии на кнопку «Сгенерировать расписание типа А» (только из водителей А) выходит обработка: невозможно сгенерировать расписание, попробуйте изменить количество рейсов или время маршрутов. Решение: можно или действительно сократить время одного рейса, но более резонным будет вариант с добавлением еще одного водителя. Эта обработка работает для всех типов генераций.

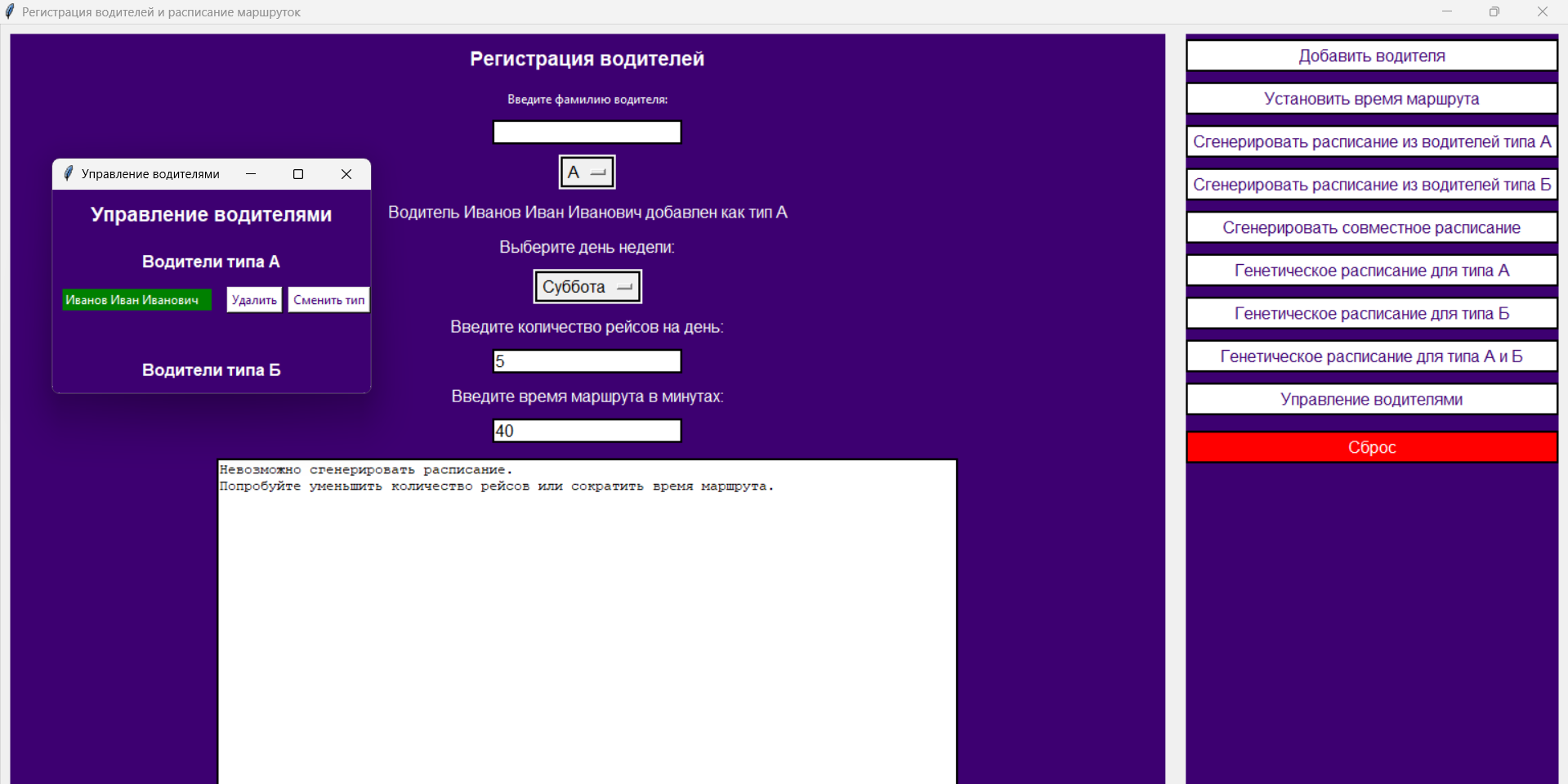


Рисунок 32 – обработка отсутсвия расписания

Еще одна обработка (удобная пользователю) – обработка с точным указанием о том, сколько минимум водитилей какого типа надо добавить.

Например – время маршрута установлено, количество рейсов тоже задано,но водители этого типа не создавались. В таком случае выходит обработка: добавьте минимум столько то водителей такого то типа.

На скриншоте ситуация когда требуется выполнить всего 5 рейсов.

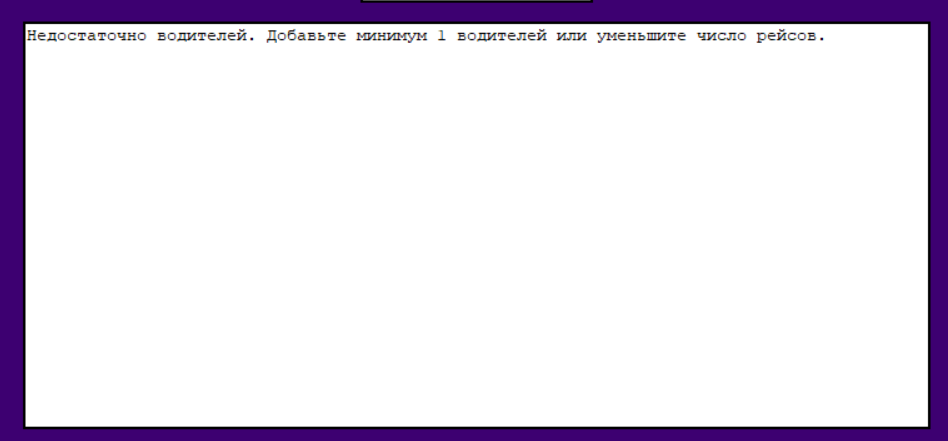


Рисунок 34 – обработка с точным указанием

Еще одна обработка: есть водители типа А, нет водителей типа Б. Все параметры успешно заданы, но выбран выходной день недели. Так как по условию водители типа А не могут работать в выходные – следовательно обработка это и отвечает.

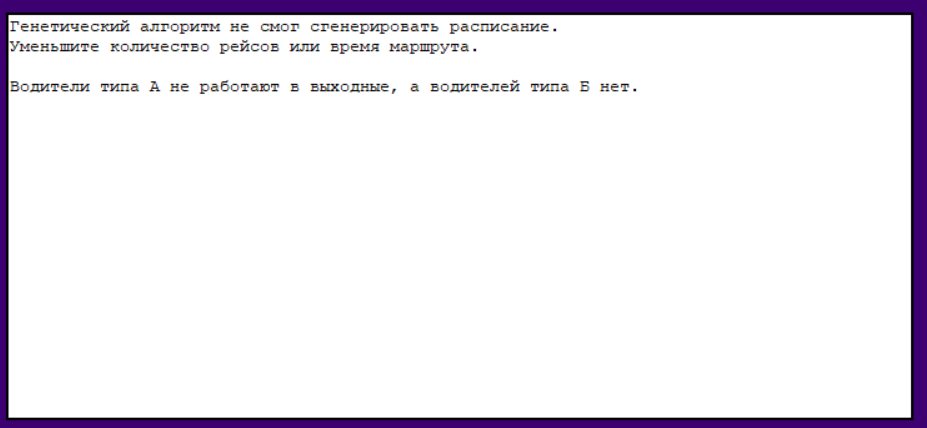


Рисунок 35 – неверно выбранный день недели относительно водителей

Теперь перейду к успешным генерациям. Первый алгоритм: добавлю несколько водителей и установлю требование: 19 рейсов на день обязательно должно быть. Время в пути 40 минут. Если рейс круговой (замкнутый) то время считается как в два раза больше.

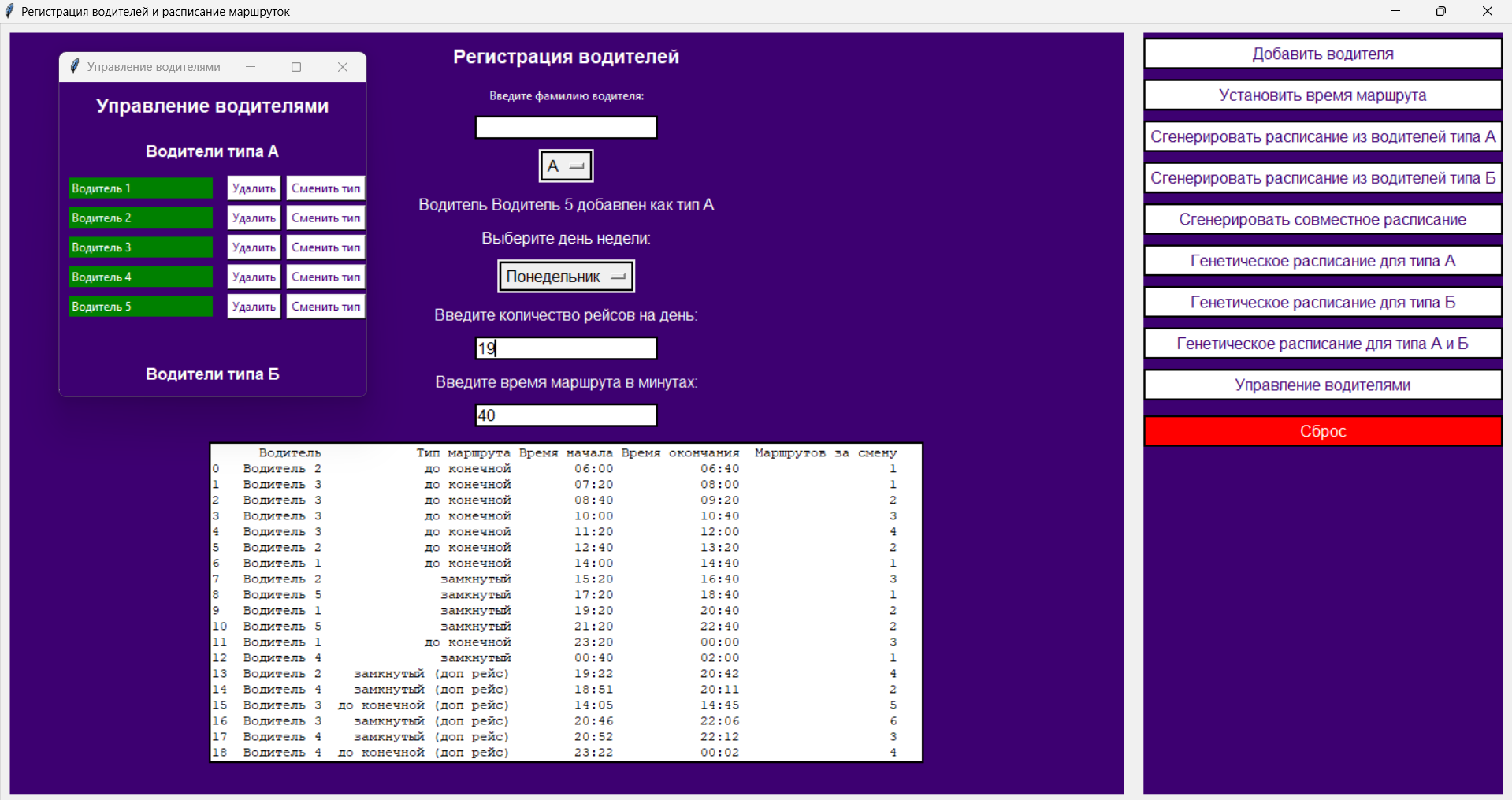


Рисунок 36 – успешно созданное расписание.

Стоить обратить внимание на то,что по условию работа маршрутного парка начинается с 6 утра и заканчивается не позднее 3:00 ночи. Но не со всеми вводными параметрами всегда получится умещать все рейсы в этот промежуток, ведь у водитилей также предусмотрен перерыв.

Поэтому те рейсы которые не успевают реализоваться в отведенное время в порядке очереди расписания – записываются с пометкой (доп рейс) – это значит,что этот рейс будет включен просто по ходу дня между несколькими рейсами (например в дневное время).

И также сгенерирую расписание генетическим алгоритмом.

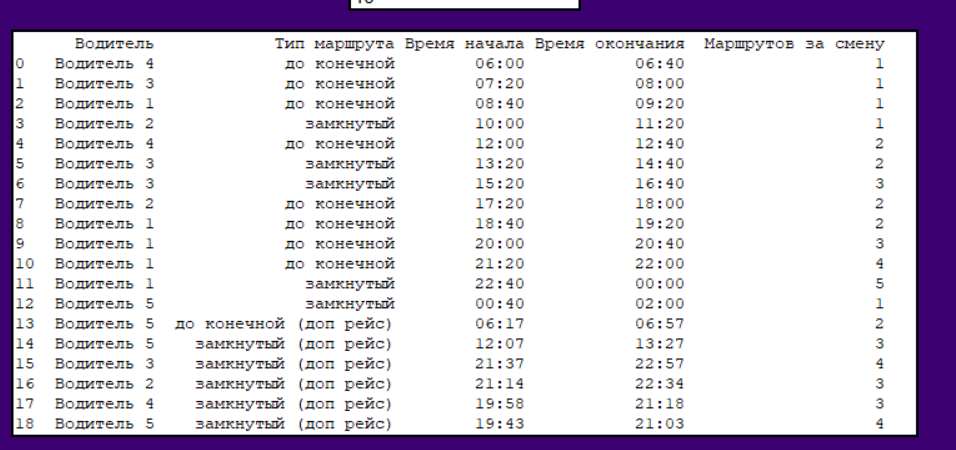


Рисунок 37 – генетическое расписание

Это расписание генерируется дольше,но результат оптимума по итогу может быть даже лучше.

**Сравнение**: если требуется акцент на скорость генерации – то следует выбирать детермизированное расписание.

Отдельно я реализовал интерфейс управления водителями: можно удалять и изменять тип водителям.

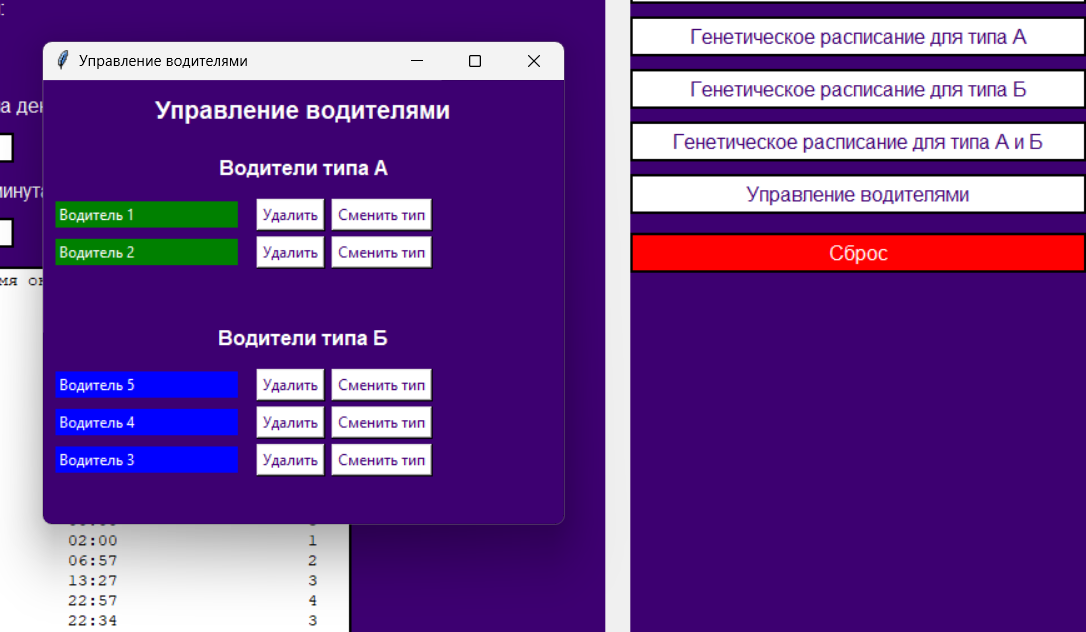


Рисунок 38 – управление водителями

ВЫВОД

В результате проделанной работы было создано удобное и функциональное приложение для управления водителями и генерации расписаний маршруток. Приложение обладает интуитивно понятным графическим интерфейсом, разработанным с использованием библиотеки Tkinter, что обеспечивает простоту использования даже для пользователей без технического опыта

ЛИСТИНГ КОДА

from tkinter import \*

import pandas as pd

import random

from datetime import datetime, timedelta

drivers\_A = []

drivers\_B = []

drivers\_shifts = {}

route\_types = ['до конечной', 'замкнутый']

shift\_duration\_A = 8

shift\_duration\_B = 12

traffic\_route\_time = 60

start\_work\_time = '06:00'

end\_work\_time = '03:00'

def is\_weekend(selected\_day):

    return selected\_day in ['Суббота', 'Воскресенье']

def calculate\_route\_end(start\_time, route\_time):

    start\_time\_obj = datetime.strptime(start\_time, "%H:%M")

    end\_time\_obj = start\_time\_obj + timedelta(minutes=route\_time)

    return end\_time\_obj.strftime("%H:%M")

def normalize\_interval(start\_str, end\_str):

    start = datetime.strptime(start\_str, "%H:%M")

    end = datetime.strptime(end\_str, "%H:%M")

    if end < start:

        end += timedelta(days=1)

    return start, end

def is\_time\_overlap(start\_time, end\_time, busy\_times):

    start, end = normalize\_interval(start\_time, end\_time)

    for busy\_start, busy\_end in busy\_times:

        busy\_start\_time, busy\_end\_time = normalize\_interval(busy\_start, busy\_end)

        if start < busy\_end\_time and end > busy\_start\_time:

            return True

    return False

def find\_intermediate\_slots(driver\_busy\_times, route\_time, break\_time):

    free\_slots = []

    for driver, busy\_periods in driver\_busy\_times.items():

        normalized\_periods = []

        for (start, end) in busy\_periods:

            start\_time, end\_time = normalize\_interval(start, end)

            normalized\_periods.append((start\_time, end\_time))

        normalized\_periods.sort(key=lambda x: x[0])

        current\_time = datetime.strptime("06:00", "%H:%M")

        work\_end\_time = datetime.strptime("03:00", "%H:%M") + timedelta(days=1)

        for (st, et) in normalized\_periods:

            if (st - current\_time).total\_seconds()/60 >= route\_time + break\_time:

                free\_slots.append((current\_time.strftime("%H:%M"), st.strftime("%H:%M")))

            current\_time = et

        if (work\_end\_time - current\_time).total\_seconds()/60 >= route\_time + break\_time:

            free\_slots.append((current\_time.strftime("%H:%M"), work\_end\_time.strftime("%H:%M")))

    return free\_slots

def calculate\_additional\_drivers(num\_routes, driver\_list, shift\_duration):

    max\_routes\_per\_driver = int(shift\_duration \* 60 / traffic\_route\_time)

    required\_drivers = (num\_routes + max\_routes\_per\_driver - 1) // max\_routes\_per\_driver

    if len(driver\_list) >= required\_drivers:

        return 0

    else:

        return required\_drivers - len(driver\_list)

def print\_cannot\_generate\_message(text\_widget, driver\_list, shift\_duration, num\_routes):

    text\_widget.delete(1.0, END)

    additional\_drivers\_needed = calculate\_additional\_drivers(num\_routes, driver\_list, shift\_duration)

    if additional\_drivers\_needed > 0:

        text\_widget.insert(END, (

            f"Невозможно сгенерировать расписание.\nПопробуйте добавить минимум {additional\_drivers\_needed} водителей или "

            "уменьшить количество рейсов.\n"

        ))

    else:

        text\_widget.insert(END, (

            "Невозможно сгенерировать расписание.\nПопробуйте уменьшить количество рейсов или сократить время маршрута.\n"

        ))

def print\_cannot\_generate\_message\_ga(text\_widget, driver\_list, shift\_duration, num\_routes):

    text\_widget.delete(1.0, END)

    additional\_drivers\_needed = calculate\_additional\_drivers(num\_routes, driver\_list, shift\_duration)

    if additional\_drivers\_needed > 0:

        text\_widget.insert(END, (

            f"Генетический алгоритм не смог сгенерировать расписание.\nДобавьте минимум {additional\_drivers\_needed} водителей или "

            "уменьшите количество рейсов.\n"

        ))

    else:

        text\_widget.insert(END, (

            "Генетический алгоритм не смог сгенерировать расписание.\nУменьшите количество рейсов или время маршрута.\n"

        ))

def can\_place\_route(candidate\_start\_time, route\_time, driver, driver\_busy\_times, driver\_worked\_hours, driver\_route\_counts, min\_break\_time):

    candidate\_end\_time = calculate\_route\_end(candidate\_start\_time, route\_time)

    if is\_time\_overlap(candidate\_start\_time, candidate\_end\_time, driver\_busy\_times[driver]):

        return False

    if driver\_busy\_times[driver]:

        last\_route\_start\_str, last\_route\_end\_str = driver\_busy\_times[driver][-1]

        last\_route\_end\_obj = datetime.strptime(last\_route\_end\_str, "%H:%M")

        last\_route\_start\_obj = datetime.strptime(last\_route\_start\_str, "%H:%M")

        if last\_route\_end\_obj < last\_route\_start\_obj:

            last\_route\_end\_obj += timedelta(days=1)

        candidate\_start\_time\_obj = datetime.strptime(candidate\_start\_time, "%H:%M")

        if candidate\_start\_time\_obj < last\_route\_end\_obj:

            return False

        if (candidate\_start\_time\_obj - last\_route\_end\_obj).total\_seconds()/60 < min\_break\_time:

            return False

    worked\_hours = driver\_worked\_hours[driver]

    if driver in drivers\_A and worked\_hours >= shift\_duration\_A:

        return False

    if driver in drivers\_B and worked\_hours >= shift\_duration\_B:

        return False

    candidate\_end\_time\_obj = datetime.strptime(candidate\_end\_time, "%H:%M")

    if candidate\_end\_time\_obj < datetime.strptime(candidate\_start\_time, "%H:%M"):

        candidate\_end\_time\_obj += timedelta(days=1)

    end\_work\_time\_obj = datetime.strptime("03:00", "%H:%M") + timedelta(days=1)

    if candidate\_end\_time\_obj > end\_work\_time\_obj:

        return False

    return True

def place\_route\_any\_slot(route\_time, break\_time, min\_break\_time, driver\_list, driver\_busy\_times, driver\_worked\_hours, driver\_route\_counts,

                          selected\_day):

    for attempt in range(50):

        free\_slots = find\_intermediate\_slots(driver\_busy\_times, route\_time, break\_time)

        if not free\_slots:

            return None

        slot\_start, slot\_end = random.choice(free\_slots)

        slot\_start\_obj = datetime.strptime(slot\_start, "%H:%M")

        slot\_end\_obj = datetime.strptime(slot\_end, "%H:%M")

        if slot\_end\_obj < slot\_start\_obj:

            slot\_end\_obj += timedelta(days=1)

        max\_start = (slot\_end\_obj - slot\_start\_obj).total\_seconds()/60 - route\_time

        if max\_start < 0:

            continue

        offset = random.randint(0, int(max\_start))

        candidate\_start\_time\_obj = slot\_start\_obj + timedelta(minutes=offset)

        candidate\_start\_time = candidate\_start\_time\_obj.strftime("%H:%M")

        random.shuffle(driver\_list)

        for driver in driver\_list:

            if driver in drivers\_A and is\_weekend(selected\_day):

                continue

            if can\_place\_route(candidate\_start\_time, route\_time, driver, driver\_busy\_times, driver\_worked\_hours, driver\_route\_counts,

                               min\_break\_time):

                return (driver, candidate\_start\_time)

    return None

def generate\_optimized\_schedule(driver\_list, shift\_duration, num\_routes, selected\_day, text\_widget, break\_time=10, min\_break\_time=30):

    additional\_drivers\_needed = calculate\_additional\_drivers(num\_routes, driver\_list, shift\_duration)

    if additional\_drivers\_needed > 0:

        text\_widget.delete(1.0, END)

        text\_widget.insert(END, (

            f"Недостаточно водителей. Добавьте минимум {additional\_drivers\_needed} водителей или уменьшите число рейсов.\n"

        ))

        return

    schedule = []

    total\_routes\_assigned = 0

    available\_drivers = list(driver\_list)

    random.shuffle(available\_drivers)

    driver\_busy\_times = {driver: [] for driver in available\_drivers}

    driver\_worked\_hours = {driver: 0 for driver in available\_drivers}

    driver\_route\_counts = {driver: 0 for driver in available\_drivers}

    current\_time = datetime.strptime("06:00", "%H:%M")

    end\_work\_time\_obj = datetime.strptime("03:00", "%H:%M") + timedelta(days=1)

    for \_ in range(num\_routes):

        route\_type = random.choice(route\_types)

        route\_actual\_time = traffic\_route\_time \* 2 if route\_type == 'замкнутый' else traffic\_route\_time

        candidate\_start\_str = current\_time.strftime("%H:%M")

        candidate\_end\_str = calculate\_route\_end(candidate\_start\_str, route\_actual\_time)

        candidate\_end\_obj = datetime.strptime(candidate\_end\_str, "%H:%M")

        if candidate\_end\_obj < current\_time:

            candidate\_end\_obj += timedelta(days=1)

        if candidate\_end\_obj > end\_work\_time\_obj:

            result = place\_route\_any\_slot(route\_actual\_time, break\_time, min\_break\_time, available\_drivers, driver\_busy\_times, driver\_worked\_hours, driver\_route\_counts, selected\_day)

            if result is None:

                print\_cannot\_generate\_message(text\_widget, driver\_list, shift\_duration, num\_routes)

                return

            else:

                driver, slot\_start = result

                cend = calculate\_route\_end(slot\_start, route\_actual\_time)

                cend\_obj = datetime.strptime(cend, "%H:%M")

                if cend\_obj < datetime.strptime(slot\_start, "%H:%M"):

                    cend\_obj += timedelta(days=1)

                worked\_minutes = (cend\_obj - datetime.strptime(slot\_start, "%H:%M")).seconds / 60

                driver\_route\_counts[driver] += 1

                final\_route\_type = route\_type + " (доп рейс)"

                schedule.append({

                    'Водитель': driver,

                    'Тип маршрута': final\_route\_type,

                    'Время начала': slot\_start,

                    'Время окончания': cend,

                    'Маршрутов за смену': driver\_route\_counts[driver]

                })

                driver\_busy\_times[driver].append((slot\_start, cend))

                driver\_worked\_hours[driver] += worked\_minutes / 60

                total\_routes\_assigned += 1

        else:

            placed = False

            random.shuffle(available\_drivers)

            for driver in available\_drivers:

                if driver in drivers\_A and is\_weekend(selected\_day):

                    continue

                if can\_place\_route(candidate\_start\_str, route\_actual\_time, driver, driver\_busy\_times, driver\_worked\_hours,

                                   driver\_route\_counts, min\_break\_time):

                    cend\_obj = candidate\_end\_obj

                    worked\_minutes = (cend\_obj - datetime.strptime(candidate\_start\_str, "%H:%M")).seconds / 60

                    driver\_route\_counts[driver] += 1

                    schedule.append({

                        'Водитель': driver,

                        'Тип маршрута': route\_type,

                        'Время начала': candidate\_start\_str,

                        'Время окончания': candidate\_end\_str,

                        'Маршрутов за смену': driver\_route\_counts[driver]

                    })

                    driver\_busy\_times[driver].append((candidate\_start\_str, candidate\_end\_str))

                    driver\_worked\_hours[driver] += worked\_minutes / 60

                    total\_routes\_assigned += 1

                    placed = True

                    current\_time = cend\_obj + timedelta(minutes=break\_time + min\_break\_time)

                    break

            if not placed:

                result = place\_route\_any\_slot(route\_actual\_time, break\_time, min\_break\_time, available\_drivers, driver\_busy\_times,

                                              driver\_worked\_hours, driver\_route\_counts, selected\_day)

                if result is None:

                    print\_cannot\_generate\_message(text\_widget, driver\_list, shift\_duration, num\_routes)

                    return

                else:

                    driver, slot\_start = result

                    cend = calculate\_route\_end(slot\_start, route\_actual\_time)

                    cend\_obj = datetime.strptime(cend, "%H:%M")

                    if cend\_obj < datetime.strptime(slot\_start, "%H:%M"):

                        cend\_obj += timedelta(days=1)

                    worked\_minutes = (cend\_obj - datetime.strptime(slot\_start, "%H:%M")).seconds / 60

                    driver\_route\_counts[driver] += 1

                    final\_route\_type = route\_type + " (доп рейс)"

                    schedule.append({

                        'Водитель': driver,

                        'Тип маршрута': final\_route\_type,

                        'Время начала': slot\_start,

                        'Время окончания': cend,

                        'Маршрутов за смену': driver\_route\_counts[driver]

                    })

                    driver\_busy\_times[driver].append((slot\_start, cend))

                    driver\_worked\_hours[driver] += worked\_minutes / 60

                    total\_routes\_assigned += 1

    df = pd.DataFrame(schedule)

    text\_widget.delete(1.0, END)

    if not df.empty:

        text\_widget.insert(END, df.to\_string())

    else:

        print\_cannot\_generate\_message(text\_widget, driver\_list, shift\_duration, num\_routes)

def try\_create\_schedule\_ga(driver\_list, shift\_duration, num\_routes, selected\_day, break\_time=10, min\_break\_time=30):

    available\_drivers = list(driver\_list)

    random.shuffle(available\_drivers)

    driver\_busy\_times = {driver: [] for driver in available\_drivers}

    driver\_worked\_hours = {driver: 0 for driver in available\_drivers}

    driver\_route\_counts = {driver: 0 for driver in available\_drivers}

    schedule = []

    total\_routes\_assigned = 0

    end\_work\_time\_obj = datetime.strptime("03:00", "%H:%M") + timedelta(days=1)

    def place\_in\_slot(route\_actual\_time):

        for attempt in range(50):

            free\_slots = find\_intermediate\_slots(driver\_busy\_times, route\_actual\_time, break\_time)

            if not free\_slots:

                return None

            slot\_start, slot\_end = random.choice(free\_slots)

            slot\_start\_obj = datetime.strptime(slot\_start, "%H:%M")

            slot\_end\_obj = datetime.strptime(slot\_end, "%H:%M")

            if slot\_end\_obj < slot\_start\_obj:

                slot\_end\_obj += timedelta(days=1)

            max\_start = (slot\_end\_obj - slot\_start\_obj).total\_seconds()/60 - route\_actual\_time

            if max\_start < 0:

                continue

            offset = random.randint(0, int(max\_start))

            candidate\_start\_time\_obj = slot\_start\_obj + timedelta(minutes=offset)

            candidate\_start\_time = candidate\_start\_time\_obj.strftime("%H:%M")

            random.shuffle(available\_drivers)

            for driver in available\_drivers:

                if driver in drivers\_A and is\_weekend(selected\_day):

                    continue

                if can\_place\_route(candidate\_start\_time, route\_actual\_time, driver, driver\_busy\_times, driver\_worked\_hours,

                                   driver\_route\_counts, min\_break\_time):

                    return (driver, candidate\_start\_time)

        return None

    for i in range(num\_routes):

        route\_type = random.choice(route\_types)

        route\_actual\_time = traffic\_route\_time \* 2 if route\_type == 'замкнутый' else traffic\_route\_time

        attempts = 0

        placed = False

        while attempts < 50 and not placed:

            attempts += 1

            candidate\_start\_time = datetime.strptime("06:00", "%H:%M")

            if schedule:

                max\_end = datetime.strptime("06:00", "%H:%M")

                for r in schedule:

                    endt = datetime.strptime(r['Время окончания'], "%H:%M")

                    startt = datetime.strptime(r['Время начала'], "%H:%M")

                    if endt < startt:

                        endt += timedelta(days=1)

                    if endt > max\_end:

                        max\_end = endt

                candidate\_start\_time = max\_end + timedelta(minutes=break\_time + min\_break\_time)

            candidate\_start\_str = candidate\_start\_time.strftime("%H:%M")

            candidate\_end\_str = calculate\_route\_end(candidate\_start\_str, route\_actual\_time)

            candidate\_end\_obj = datetime.strptime(candidate\_end\_str, "%H:%M")

            if candidate\_end\_obj < candidate\_start\_time:

                candidate\_end\_obj += timedelta(days=1)

            if candidate\_end\_obj > end\_work\_time\_obj:

                result = place\_in\_slot(route\_actual\_time)

                if result is not None:

                    driver, slot\_start = result

                    cend = calculate\_route\_end(slot\_start, route\_actual\_time)

                    cend\_obj = datetime.strptime(cend, "%H:%M")

                    if cend\_obj < datetime.strptime(slot\_start, "%H:%M"):

                        cend\_obj += timedelta(days=1)

                    worked\_minutes = (cend\_obj - datetime.strptime(slot\_start, "%H:%M")).seconds / 60

                    driver\_route\_counts[driver] += 1

                    final\_route\_type = route\_type + " (доп рейс)"

                    schedule.append({

                        'Водитель': driver,

                        'Тип маршрута': final\_route\_type,

                        'Время начала': slot\_start,

                        'Время окончания': cend,

                        'Маршрутов за смену': driver\_route\_counts[driver]

                    })

                    driver\_busy\_times[driver].append((slot\_start, cend))

                    driver\_worked\_hours[driver] += worked\_minutes / 60

                    placed = True

                    total\_routes\_assigned += 1

            else:

                placed\_lin = False

                random.shuffle(available\_drivers)

                for driver in available\_drivers:

                    if driver in drivers\_A and is\_weekend(selected\_day):

                        continue

                    if can\_place\_route(candidate\_start\_str, route\_actual\_time, driver, driver\_busy\_times, driver\_worked\_hours,

                                       driver\_route\_counts, min\_break\_time):

                        cend\_obj = candidate\_end\_obj

                        worked\_minutes = (cend\_obj - datetime.strptime(candidate\_start\_str, "%H:%M")).seconds / 60

                        driver\_route\_counts[driver] += 1

                        schedule.append({

                            'Водитель': driver,

                            'Тип маршрута': route\_type,

                            'Время начала': candidate\_start\_str,

                            'Время окончания': candidate\_end\_str,

                            'Маршрутов за смену': driver\_route\_counts[driver]

                        })

                        driver\_busy\_times[driver].append((candidate\_start\_str, candidate\_end\_str))

                        driver\_worked\_hours[driver] += worked\_minutes / 60

                        placed = True

                        placed\_lin = True

*# не меняем current\_time глобально для ГА*

                        break

                if not placed\_lin and not placed:

                    result = place\_in\_slot(route\_actual\_time)

                    if result is not None:

                        driver, slot\_start = result

                        cend = calculate\_route\_end(slot\_start, route\_actual\_time)

                        cend\_obj = datetime.strptime(cend, "%H:%M")

                        if cend\_obj < datetime.strptime(slot\_start, "%H:%M"):

                            cend\_obj += timedelta(days=1)

                        worked\_minutes = (cend\_obj - datetime.strptime(slot\_start, "%H:%M")).seconds / 60

                        driver\_route\_counts[driver] += 1

                        final\_route\_type = route\_type + " (доп рейс)"

                        schedule.append({

                            'Водитель': driver,

                            'Тип маршрута': final\_route\_type,

                            'Время начала': slot\_start,

                            'Время окончания': cend,

                            'Маршрутов за смену': driver\_route\_counts[driver]

                        })

                        driver\_busy\_times[driver].append((slot\_start, cend))

                        driver\_worked\_hours[driver] += worked\_minutes / 60

                        placed = True

                        total\_routes\_assigned += 1

        if not placed:

            break

        else:

            total\_routes\_assigned += 1

    return schedule, total\_routes\_assigned

def crossover(schedule1, schedule2):

    half = len(schedule1)//2

    child = schedule1[:half] + schedule2[half:]

    return child

def mutate(schedule):

    if random.random() < 0.1 and len(schedule) > 1:

        i, j = random.sample(range(len(schedule)), 2)

        schedule[i], schedule[j] = schedule[j], schedule[i]

    return schedule

def fitness(schedule, num\_routes, drivers\_A, drivers\_B, traffic\_route\_time):

    assigned\_routes = len(schedule)

    drivers\_used = set([route['Водитель'] for route in schedule])

    num\_drivers\_used = len(drivers\_used)

    total\_work\_minutes = sum([

        (datetime.strptime(route['Время окончания'], "%H:%M") - datetime.strptime(route['Время начала'], "%H:%M")).seconds / 60

        for route in schedule

    ])

    if num\_drivers\_used > 0:

        average\_work\_minutes = total\_work\_minutes / num\_drivers\_used

    else:

        average\_work\_minutes = 0

    distribution\_penalty = 0

    for driver in drivers\_used:

        driver\_routes = [route for route in schedule if route['Водитель'] == driver]

        driver\_work\_minutes = sum([

            (datetime.strptime(route['Время окончания'], "%H:%M") - datetime.strptime(route['Время начала'], "%H:%M")).seconds / 60

            for route in driver\_routes

        ])

        distribution\_penalty += abs(driver\_work\_minutes - average\_work\_minutes)

    weight\_routes = 1000

    weight\_drivers = 50

    weight\_distribution = 10

    fitness\_score = (assigned\_routes \* weight\_routes) - (num\_drivers\_used \* weight\_drivers) - (distribution\_penalty \* weight\_distribution)

    return fitness\_score

def genetic\_algorithm\_schedule(driver\_list, shift\_duration, num\_routes, selected\_day, text\_widget, break\_time=10, min\_break\_time=30,

                               generations=10, population\_size=5):

    additional\_drivers\_needed = calculate\_additional\_drivers(num\_routes, driver\_list, shift\_duration)

    if additional\_drivers\_needed > 0:

        text\_widget.delete(1.0, END)

        text\_widget.insert(END, (

            f"Недостаточно водителей. Нужно добавить минимум {additional\_drivers\_needed} водителей или "

            "уменьшить число рейсов.\n"

        ))

        return

    population = []

    for \_ in range(population\_size):

        schedule, score = try\_create\_schedule\_ga(driver\_list, shift\_duration, num\_routes, selected\_day, break\_time, min\_break\_time)

        fitness\_score = fitness(schedule, num\_routes, drivers\_A, drivers\_B, traffic\_route\_time)

        population.append((schedule, fitness\_score))

    best\_schedule = None

    best\_score = -float('inf')

    no\_improvement\_count = 0

    improvement\_limit = 3

    for gen in range(generations):

        evaluated\_population = []

        for individual in population:

            schedule = individual[0]

            current\_fitness = fitness(schedule, num\_routes, drivers\_A, drivers\_B, traffic\_route\_time)

            evaluated\_population.append((schedule, current\_fitness))

        evaluated\_population.sort(key=lambda x: x[1], reverse=True)

        if evaluated\_population[0][1] > best\_score:

            best\_score = evaluated\_population[0][1]

            best\_schedule = evaluated\_population[0][0]

            no\_improvement\_count = 0

        else:

            no\_improvement\_count += 1

        if best\_score >= num\_routes \* 1000:

            break

        if no\_improvement\_count >= improvement\_limit:

            break

        new\_population = evaluated\_population[:2]

        while len(new\_population) < population\_size:

            parent1 = random.choice(evaluated\_population)[0]

            parent2 = random.choice(evaluated\_population)[0]

            child = crossover(parent1, parent2)

            child = mutate(child)

            child\_fitness = fitness(child, num\_routes, drivers\_A, drivers\_B, traffic\_route\_time)

            new\_population.append((child, child\_fitness))

        population = new\_population

    text\_widget.delete(1.0, END)

    if best\_schedule and len(best\_schedule) > 0:

        df = pd.DataFrame(best\_schedule)

        if best\_score >= num\_routes \* 1000:

            text\_widget.insert(END, "Генетический алгоритм нашел идеальное решение:\n")

        else:

            text\_widget.insert(END, "Генетический алгоритм не нашел идеальное решение, но вот лучший результат:\n")

        text\_widget.insert(END, df.to\_string())

    else:

        print\_cannot\_generate\_message\_ga(text\_widget, driver\_list, shift\_duration, num\_routes)

def crossover(schedule1, schedule2):

    half = len(schedule1)//2

    child = schedule1[:half] + schedule2[half:]

    return child

def mutate(schedule):

    if random.random() < 0.1 and len(schedule) > 1:

        i, j = random.sample(range(len(schedule)), 2)

        schedule[i], schedule[j] = schedule[j], schedule[i]

    return schedule

def fitness(schedule, num\_routes, drivers\_A, drivers\_B, traffic\_route\_time):

    assigned\_routes = len(schedule)

    drivers\_used = set([route['Водитель'] for route in schedule])

    num\_drivers\_used = len(drivers\_used)

    total\_work\_minutes = sum([

        (datetime.strptime(route['Время окончания'], "%H:%M") - datetime.strptime(route['Время начала'], "%H:%M")).seconds / 60

        for route in schedule

    ])

    if num\_drivers\_used > 0:

        average\_work\_minutes = total\_work\_minutes / num\_drivers\_used

    else:

        average\_work\_minutes = 0

    distribution\_penalty = 0

    for driver in drivers\_used:

        driver\_routes = [route for route in schedule if route['Водитель'] == driver]

        driver\_work\_minutes = sum([

            (datetime.strptime(route['Время окончания'], "%H:%M") - datetime.strptime(route['Время начала'], "%H:%M")).seconds / 60

            for route in driver\_routes

        ])

        distribution\_penalty += abs(driver\_work\_minutes - average\_work\_minutes)

    weight\_routes = 1000

    weight\_drivers = 50

    weight\_distribution = 10

    fitness\_score = (assigned\_routes \* weight\_routes) + (num\_drivers\_used \* weight\_drivers) - (distribution\_penalty \* weight\_distribution)

    return fitness\_score

def manage\_drivers():

    def delete\_driver(driver\_name):

        if driver\_name in drivers\_A:

            drivers\_A.remove(driver\_name)

        elif driver\_name in drivers\_B:

            drivers\_B.remove(driver\_name)

        update\_driver\_listbox()

    def change\_driver\_type(driver\_name):

        if driver\_name in drivers\_A:

            drivers\_A.remove(driver\_name)

            drivers\_B.append(driver\_name)

        elif driver\_name in drivers\_B:

            drivers\_B.remove(driver\_name)

            drivers\_A.append(driver\_name)

        update\_driver\_listbox()

    def update\_driver\_listbox():

        for widget in driver\_listbox\_frame.winfo\_children():

            widget.destroy()

        Label(driver\_listbox\_frame, text="Водители типа А", bg="#3D0071", fg="white", font=("Helvetica", 12, "bold")).pack(pady=10)

        for driver in drivers\_A:

            driver\_frame = Frame(driver\_listbox\_frame, bg="#3D0071")

            driver\_frame.pack(fill=X, pady=2)

            Label(driver\_frame, text=driver, bg="green", fg="white", width=20, anchor=W).pack(side=LEFT, padx=10)

            Button(driver\_frame, text="Удалить", command=lambda d=driver: delete\_driver(d), bg="white", fg="#3D0071").pack(side=LEFT, padx=5)

            Button(driver\_frame, text="Сменить тип", command=lambda d=driver: change\_driver\_type(d), bg="white", fg="#3D0071").pack(side=LEFT)

        Label(driver\_listbox\_frame, text="", bg="#3D0071").pack(pady=5)

        Label(driver\_listbox\_frame, text="Водители типа Б", bg="#3D0071", fg="white", font=("Helvetica", 12, "bold")).pack(pady=10)

        for driver in drivers\_B:

            driver\_frame = Frame(driver\_listbox\_frame, bg="#3D0071")

            driver\_frame.pack(fill=X, pady=2)

            Label(driver\_frame, text=driver, bg="blue", fg="white", width=20, anchor=W).pack(side=LEFT, padx=10)

            Button(driver\_frame, text="Удалить", command=lambda d=driver: delete\_driver(d), bg="white", fg="#3D0071").pack(side=LEFT, padx=5)

            Button(driver\_frame, text="Сменить тип", command=lambda d=driver: change\_driver\_type(d), bg="white", fg="#3D0071").pack(side=LEFT)

    manage\_window = Toplevel()

    manage\_window.title("Управление водителями")

    manage\_window.configure(bg="#3D0071")

    Label(manage\_window, text="Управление водителями", bg="#3D0071", fg="white", font=("Helvetica", 14, "bold")).pack(pady=10)

*global* driver\_listbox\_frame

    driver\_listbox\_frame = Frame(manage\_window, bg="#3D0071")

    driver\_listbox\_frame.pack(fill=BOTH, expand=True)

    update\_driver\_listbox()

def generate\_schedule\_A():

    try:

        num\_routes = int(num\_routes\_entry.get())

        selected\_day = day\_choice.get()

        generate\_optimized\_schedule(drivers\_A, shift\_duration\_A, num\_routes, selected\_day, schedule\_text)

    except ValueError:

        schedule\_text.insert(END, "\nОшибка: Введите корректное количество маршруток\n")

def generate\_schedule\_B():

    try:

        num\_routes = int(num\_routes\_entry.get())

        selected\_day = day\_choice.get()

        generate\_optimized\_schedule(drivers\_B, shift\_duration\_B, num\_routes, selected\_day, schedule\_text)

    except ValueError:

        schedule\_text.insert(END, "\nОшибка: Введите корректное количество маршруток\n")

def generate\_combined\_schedule():

    try:

        num\_routes = int(num\_routes\_entry.get())

        combined\_drivers = drivers\_A + drivers\_B

        selected\_day = day\_choice.get()

        if not drivers\_A and not drivers\_B:

            schedule\_text.insert(END, "\nНет водителей для создания расписания.\n")

            return

        if is\_weekend(selected\_day) and not drivers\_B:

            schedule\_text.insert(END, "\nВодители типа А не работают в выходные, а водителей типа Б нет.\n")

            return

        if is\_weekend(selected\_day) and not drivers\_A and drivers\_B:

            additional\_drivers\_needed = calculate\_additional\_drivers(num\_routes, drivers\_B, shift\_duration\_B)

            if additional\_drivers\_needed > 0:

                schedule\_text.insert(END, (

                    f"\nНедостаточно водителей типа Б для выходного дня. Нужно добавить минимум {additional\_drivers\_needed} водителей.\n"

                ))

                return

            schedule\_text.insert(END, (

                "\nВодители типа А не работают в выходные. Создаем расписание только для водителей типа Б.\n"

            ))

            generate\_optimized\_schedule(drivers\_B, shift\_duration\_B, num\_routes, selected\_day, schedule\_text)

            return

        shift\_duration\_for\_combined = max(shift\_duration\_A, shift\_duration\_B)

        generate\_optimized\_schedule(combined\_drivers, shift\_duration\_for\_combined, num\_routes, selected\_day, schedule\_text)

    except ValueError:

        schedule\_text.insert(END, "\nОшибка: Введите корректное количество маршруток\n")

def generate\_genetic\_schedule\_A():

    try:

        num\_routes = int(num\_routes\_entry.get())

        selected\_day = day\_choice.get()

        genetic\_algorithm\_schedule(drivers\_A, shift\_duration\_A, num\_routes, selected\_day, schedule\_text, generations=50, population\_size=20)

    except ValueError:

        schedule\_text.insert(END, "\nОшибка: Введите корректное количество маршруток\n")

def generate\_genetic\_schedule\_B():

    try:

        num\_routes = int(num\_routes\_entry.get())

        selected\_day = day\_choice.get()

        genetic\_algorithm\_schedule(drivers\_B, shift\_duration\_B, num\_routes, selected\_day, schedule\_text, generations=50, population\_size=20)

    except ValueError:

        schedule\_text.insert(END, "\nОшибка: Введите корректное количество маршруток\n")

def generate\_genetic\_schedule\_AB():

    try:

        num\_routes = int(num\_routes\_entry.get())

        combined\_drivers = drivers\_A + drivers\_B

        selected\_day = day\_choice.get()

        if not drivers\_A and not drivers\_B:

            schedule\_text.insert(END, "\nНет водителей для создания расписания.\n")

            return

        if is\_weekend(selected\_day) and not drivers\_B:

            schedule\_text.insert(END, "\nВодители типа А не работают в выходные, а водителей типа Б нет.\n")

            return

        if is\_weekend(selected\_day) and not drivers\_A and drivers\_B:

            additional\_drivers\_needed = calculate\_additional\_drivers(num\_routes, drivers\_B, shift\_duration\_B)

            if additional\_drivers\_needed > 0:

                schedule\_text.insert(END, (

                    f"\nНедостаточно водителей типа Б для выходного дня. Нужно добавить минимум {additional\_drivers\_needed} водителей.\n"

                ))

                return

            schedule\_text.insert(END, (

                "\nВодители типа А не работают в выходные. Создаем генетическое расписание только для водителей типа Б.\n"

            ))

            genetic\_algorithm\_schedule(drivers\_B, shift\_duration\_B, num\_routes, selected\_day, schedule\_text, generations=50, population\_size=20)

            return

        shift\_duration\_for\_combined = max(shift\_duration\_A, shift\_duration\_B)

        genetic\_algorithm\_schedule(combined\_drivers, shift\_duration\_for\_combined, num\_routes, selected\_day, schedule\_text, generations=50,

        population\_size=20)

    except ValueError:

        schedule\_text.insert(END, "\nОшибка: Введите корректное количество маршруток\n")

def new\_function\_placeholder():

    pass

def reset\_all():

    num\_routes\_entry.delete(0, END)

    route\_time\_entry.delete(0, END)

    schedule\_text.delete(1.0, END)

    print("\nСброс всех данных выполнен.\n")

def set\_route\_time():

*global* traffic\_route\_time

    try:

        traffic\_route\_time = int(route\_time\_entry.get())

        schedule\_text.insert(END, "\nВремя маршрута успешно установлено.\n")

    except ValueError:

        schedule\_text.insert(END, "\nОшибка: Введите корректное время маршрута в минутах.\n")

def run\_gui():

*global* num\_routes\_entry, route\_time\_entry, schedule\_text, traffic\_route\_time, day\_choice

    root = Tk()

    root.title("Регистрация водителей и расписание маршруток")

    root.geometry("1900x1080")

    main\_frame = Frame(root)

    main\_frame.pack(fill="both", expand=True)

    form\_frame = Frame(main\_frame, bg="#3D0071")

    form\_frame.pack(side="left", fill="both", expand=True, padx=10, pady=10)

    form\_frame.grid\_columnconfigure(0, weight=1)

    button\_frame = Frame(main\_frame, bg="#3D0071")

    button\_frame.pack(side="right", fill="y", padx=10, pady=10)

    Label(form\_frame, text="Регистрация водителей", bg="#3D0071", fg="white", font=("Helvetica", 14, "bold")).grid(row=0, column=0, pady=10)

    Label(form\_frame, text="Введите фамилию водителя:", bg="#3D0071", fg="white").grid(row=1, column=0, pady=5)

    driver\_name\_entry = Entry(form\_frame, font=("Helvetica", 12), relief="solid", bd=2)

    driver\_name\_entry.grid(row=2, column=0, pady=5)

    driver\_type = StringVar(root)

    driver\_type.set("A")

    driver\_menu = OptionMenu(form\_frame, driver\_type, "A", "B")

    driver\_menu.config(font=("Helvetica", 12), relief="solid", bd=2)

    driver\_menu.grid(row=3, column=0, pady=5)

    def register\_driver():

        driver\_name = driver\_name\_entry.get().strip()

        driver\_choice = driver\_type.get()

        if not driver\_name:

            status\_label.config(text="Ошибка: Имя водителя не может быть пустым", fg="red")

            return

        if driver\_choice == "A":

            drivers\_A.append(driver\_name)

        else:

            drivers\_B.append(driver\_name)

        driver\_name\_entry.delete(0, END)

        status\_label.config(text=f"Водитель {driver\_name} добавлен как тип {driver\_choice}", fg="white")

    status\_label = Label(form\_frame, text="", bg="#3D0071", fg="white", font=("Helvetica", 12))

    status\_label.grid(row=4, column=0, pady=5)

    Label(form\_frame, text="Выберите день недели:", bg="#3D0071", fg="white", font=("Helvetica", 12)).grid(row=5, column=0, pady=5)

    day\_choice = StringVar(root)

    day\_choice.set("Понедельник")

    days\_menu = OptionMenu(form\_frame, day\_choice, "Понедельник", "Вторник", "Среда", "Четверг", "Пятница", "Суббота", "Воскресенье")

    days\_menu.config(font=("Helvetica", 12), relief="solid", bd=2)

    days\_menu.grid(row=6, column=0, pady=5)

    Label(form\_frame, text="Введите количество рейсов на день:", bg="#3D0071", fg="white", font=("Helvetica", 12)).grid(row=7, column=0, pady=5)

    num\_routes\_entry = Entry(form\_frame, font=("Helvetica", 12), relief="solid", bd=2)

    num\_routes\_entry.grid(row=8, column=0, pady=5)

    Label(form\_frame, text="Введите время маршрута в минутах:", bg="#3D0071", fg="white", font=("Helvetica", 12)).grid(row=9, column=0, pady=5)

    route\_time\_entry = Entry(form\_frame, font=("Helvetica", 12), relief="solid", bd=2)

    route\_time\_entry.grid(row=10, column=0, pady=5)

    schedule\_text = Text(form\_frame, height=20, width=90, bg="white", fg="black", font=("Courier", 10), relief="solid", bd=2, wrap=WORD)

    schedule\_text.grid(row=11, column=0, pady=10)

    register\_button = Button(button\_frame, text="Добавить водителя", command=register\_driver, bg="white", fg="#3D0071", font=("Helvetica", 12), relief="solid", bd=2)

    register\_button.pack(pady=5, fill="x")

    set\_time\_button = Button(button\_frame, text="Установить время маршрута", command=set\_route\_time, bg="white", fg="#3D0071", font=("Helvetica", 12), relief="solid", bd=2)

    set\_time\_button.pack(pady=5, fill="x")

    generate\_button\_A = Button(button\_frame, text="Сгенерировать расписание из водителей типа А", command=generate\_schedule\_A, bg="white", fg="#3D0071", font=("Helvetica", 12), relief="solid", bd=2)

    generate\_button\_A.pack(pady=5, fill="x")

    generate\_button\_B = Button(button\_frame, text="Сгенерировать расписание из водителей типа Б", command=generate\_schedule\_B, bg="white", fg="#3D0071", font=("Helvetica", 12), relief="solid", bd=2)

    generate\_button\_B.pack(pady=5, fill="x")

    generate\_button\_combined = Button(button\_frame, text="Сгенерировать совместное расписание", command=generate\_combined\_schedule, bg="white", fg="#3D0071", font=("Helvetica", 12), relief="solid", bd=2)

    generate\_button\_combined.pack(pady=5, fill="x")

    genetic\_button\_A = Button(button\_frame, text="Генетическое расписание для типа А", command=generate\_genetic\_schedule\_A, bg="white", fg="#3D0071", font=("Helvetica", 12), relief="solid", bd=2)

    genetic\_button\_A.pack(pady=5, fill="x")

    genetic\_button\_B = Button(button\_frame, text="Генетическое расписание для типа Б", command=generate\_genetic\_schedule\_B, bg="white", fg="#3D0071", font=("Helvetica", 12), relief="solid", bd=2)

    genetic\_button\_B.pack(pady=5, fill="x")

    genetic\_button\_AB = Button(button\_frame, text="Генетическое расписание для типа А и Б", command=generate\_genetic\_schedule\_AB, bg="white", fg="#3D0071", font=("Helvetica", 12), relief="solid", bd=2)

    genetic\_button\_AB.pack(pady=5, fill="x")

    manage\_button = Button(button\_frame, text="Управление водителями", command=manage\_drivers, bg="white", fg="#3D0071", font=("Helvetica", 12), relief="solid", bd=2)

    manage\_button.pack(pady=5, fill="x")

    reset\_button = Button(button\_frame, text="Сброс", command=reset\_all, bg="red", fg="white", font=("Helvetica", 12), relief="solid", bd=2)

    reset\_button.pack(pady=10, fill="x")

    root.mainloop()

run\_gui()

**Список литературы**

1. **Документация Python**  
   Python Software Foundation. Python 3.11. Официальная документация [Электронный ресурс]. – Режим доступа: <https://docs.python.org/3/> – Дата обращения: 27.11.2024.
2. **Tkinter – руководство по созданию GUI**  
   Библиотека Tkinter для Python. TkDocs – Официальное руководство [Электронный ресурс]. – Режим доступа: <https://tkdocs.com/> – Дата обращения: 25.11.2024.
3. **Алгоритмы оптимизации и генетические алгоритмы**  
   Mitchell M. *An Introduction to Genetic Algorithms*. – MIT Press, 1998. – 221 с.
4. **Принципы работы генетических алгоритмов**  
   Niko Schrum. "Genetic Algorithms Explained" [Электронный ресурс]. – Режим доступа: <https://towardsdatascience.com/genetic-algorithm-explained-1934f320b2e6> – Дата обращения: 01.12.2024.