**СОДЕРЖАНИЕ**

[Введение 6](#_Toc168945236)

[1 Анализ рабочих процессов формирования маршрутов городских автобусов 7](#_Toc168945237)

[1.1 Основные принципы составления расписаний 7](#_Toc168945238)

[1.2 Общие требования к расписанию 8](#_Toc168945239)

[1.3 Обоснование целесообразности автоматизации рабочих процессов 9](#_Toc168945240)

[1.4 Цели и задачи исследования 10](#_Toc168945241)

[Особенности оптимизации расписания 11](#_Toc168945242)

[Присутствующие проблемы 11](#_Toc168945243)

[1.5 Существующие методы регулирования 13](#_Toc168945244)

[Использование геолокации. 13](#_Toc168945245)

[Централизованная система управления. 13](#_Toc168945246)

[Интеллектуальные системы управления. 14](#_Toc168945247)

[1.6 Требования к разрабатываемому программному продукту 14](#_Toc168945248)

[Функциональные требования 14](#_Toc168945249)

[Требования к интерфейсу пользователя 15](#_Toc168945250)

[Требования к программному обеспечению 16](#_Toc168945251)

[1.7 Моделирование рабочих процессов 16](#_Toc168945252)

[BPMN-модель рабочего процесса «Как есть» 16](#_Toc168945253)

[2 Разработка алгоритма регулирования интервалов движения автобусов 20](#_Toc168945254)

[2.1 BPMN-модель рабочего процесса «Как должно быть» 20](#_Toc168945255)

[2.2 Структура базы данных 22](#_Toc168945256)

[2.3 Описание пользовательского интерфейса 24](#_Toc168945257)

[Основное окно 24](#_Toc168945258)

[Верхняя панель инструментов 25](#_Toc168945259)

[Область построения графика 27](#_Toc168945260)

[Табличные виджеты 30](#_Toc168945261)

[Иерархическая структура логической схемы интерфейса 31](#_Toc168945262)

[2.4 Описание функционала приложения 34](#_Toc168945263)

[Используемые инструменты 34](#_Toc168945264)

[Диаграмма компонентов программы 36](#_Toc168945265)

[Основные функции 37](#_Toc168945266)

[2.5 Контрольный пример и инструкция пользователя 41](#_Toc168945267)

[3 Системотехнические расчёты 54](#_Toc168945268)

[3.1 Метод экспертных оценок 54](#_Toc168945269)

[3.2 Опрос экспертов 55](#_Toc168945270)

[3.3 Вычисление наиболее предпочтительного варианта регулирования 56](#_Toc168945271)

[Заключение 61](#_Toc168945272)

[Список использованных источников 63](#_Toc168945273)

[Приложение А 64](#_Toc168945274)

[Приложение Б 69](#_Toc168945275)

Введение

Современный город — это сложная система, в которой все элементы взаимосвязаны и взаимозависимы. Города сталкиваются с задачей организации движения общественного транспорта. Особенно актуальной эта проблема становится в условиях быстрого роста населения и увеличения количества автомобилей на дорогах. Одной из важнейших составляющих работы городского транспорта является обеспечение равномерного подъезда автобусов на остановках. Точное расписание и соблюдение оптимальных временных интервалов движения автобусов напрямую влияют на качество обслуживания пассажиров.

Однако существует проблема кучкования автобусов на остановках, когда несколько транспортных средств одновременно подъезжают к остановке для посадки и высадки пассажиров. Это может вызывать хаос и неудобства как для пассажиров, так и для водителей. Такая ситуация также может снижать эффективность обслуживания и увеличивать время задержек в движении автобусов. Эта проблема требует внимания и эффективных мер по регулированию движения автобусов на остановках, чтобы обеспечить более плавное и эффективное обслуживание пассажиров.

Одновременное нахождение автобусов на одной остановке может быть вызвано различными факторами, такими как несогласованное расписание движения автобусов, изменчивый пассажиропоток, отсутствие системы регулирования интервалов подъезда автобусов на остановку, поэтому эта проблема актуальна и по сей день.

Целью исследования является решение проблемы одновременного скопления автобусов на остановках путём создания сервиса регулирования интервалов подъезда на остановках города.

Задачи исследования:

* анализ процессов создания расписаний и выявление проблемы регулирования интервалов на остановках;
* изучение подходов к автоматизации регулирования интервалов автобусов и их недостатков;
* разработка BPMN-моделей текущих процессов и алгоритма оптимизации интервалов на остановках;
* проектирование базы данных для решения проблемы регулирования интервалов автобусов;
* создание дизайна пользовательского интерфейса приложения и его описание;
* разработка алгоритма регулирования интервалов подъезда автобусов к остановкам;
* описание контрольного примера для проверки результатов работы алгоритма.

В ходе работы рассмотрены существующие подходы к регулированию интервалов движения и способы определения оптимального времени прибытия автобусов на остановки. Будет проведён анализ существующих систем и сервисов, используемых для управления городским транспортом.

Результаты данной работы будут иметь практическую значимость в первую очередь для пользователей общественного транспорта – жителей города, а также для городских властей, управляющих транспортной системой. Разработанный сервис регулирования интервалов позволит создать более комфортные условия передвижения для всех граждан.

# Анализ рабочих процессов формирования маршрутов городских автобусов

## Основные принципы составления расписаний

Организация работы автобусов начинается с тщательного анализа потребностей города. Это включает изучение инфраструктуры, плотности населения, месторасположения школ, больниц, торговых центров и других ключевых объектов. На основе этого анализа разрабатываются маршруты. Определяются начальные и конечные точки маршрутов, выбираются промежуточные остановки, учитывая пассажиропоток и удобство доступа [1].

После составления маршрутов разрабатываются схемы движения автобусов по ним. Эти схемы включают в себя все остановки и направления движения. Затем создаются расписания движения, учитывая пиковую и не пиковую нагрузку. В часы пик вводятся дополнительные рейсы для обеспечения комфортного перемещения пассажиров.

Для обеспечения точности и удобства пассажиров используются современные технологии. Например, GPS-трекеры устанавливаются на автобусы для отслеживания их положения в реальном времени. Эта информация доступна через мобильные приложения и онлайн-сервисы, позволяя пассажирам заранее планировать свои поездки и знать точное время прибытия и отъезда автобуса с остановки.

Для каждого конкретного рейса составляются маршрутные листы [1]. Они включают информацию о времени начала и окончания работы водителя, интервалах и времени остановок. Эти листы периодически обновляются и корректируются в зависимости от изменений в графиках и условиях дорожного движения.

Расписание является основой организации движения автобусов на маршрутах, обязательно для выполнения всеми линейными работниками пасса­жирского автотранспорта. Им определяется количество рейсов, время движения между остановочными пунктами и т.д.

## Общие требования к расписанию

Расписание движения автобусов должно учитывать несколько важных аспектов [3]:

* удовлетворение потребности населения в перевозках: это включает в себя обеспечение покрытия всех районов города, учет пиковых часов и изменений в пассажиропотоке;
* использование вместимости автобусов: эффективное использование вместимости позволяет оптимизировать количество рейсов и экономить ресурсы;
* минимальные затраты времени пассажира на поездки: регулярность и предсказуемость расписания позволяют пассажирам планировать свое время и избегать лишних ожиданий;
* контроль за движением автобусов на всем протяжении маршрутов: это важно для пассажиров, которые используют разные участки маршрута;
* создание необходимых удобств в пути следования: удобное расположение остановок и условия для пассажиров с ограниченными возможностями;
* соблюдение режима и условий труда водителей и кондукторов: расписание должно учитывать необходимые перерывы и оптимальные условия работы персонала;
* эффективное использование автобусов: плотность расписания и предсказуемость позволяют максимально использовать автобусы и минимизировать их простои.

## Обоснование целесообразности автоматизации рабочих процессов

На сегодняшний день процессы регулирования интервалов подъезда автобусов в основном производится вручную, что может приводить к ошибкам и снизить производительность работы, а также вызвать излишние временные издержки. Автоматизация рабочих процессов регулирования оправдана по нескольким причинам:

* автоматизация позволяет прогнозировать и регулировать движение автобусов на основе текущей загрузки, трафика и других факторов, что позволяет эффективнее использовать имеющиеся транспортные средства;
* автоматизированное регулирование интервалов позволяет сократить время ожидания пассажиров на остановках за счет более регулярного и плавного движения автобусов, что улучшает качество обслуживания и удовлетворенность пассажиров;
* автоматизация позволяет лучше координировать движение автобусов на различных маршрутах и остановках, что способствует снижению перегрузок и пробок на дорогах и улучшает общую проходимость транспортного потока;
* автоматизация позволяет уменьшить человеческий фактор в процессе регулирования интервалов, что снижает вероятность ошибок и неоднородности в управлении транспортом;
* автоматизация обеспечивает более предсказуемое движение автобусов, что помогает пассажирам планировать свои поездки более эффективно и снижает неудобства, связанные с неожиданными задержками.

Для пассажиров:

* уменьшение времени ожидания на остановках;
* больше возможностей для планирования маршрутов;
* улучшение общего опыта использования общественного транспорта.

Для городских властей:

* эффективное использование транспортных ресурсов;
* экономия средств и оптимизация бюджета;
* развитие общественного транспорта как важного элемента инфраструктуры города.

Таким образом, автоматизация помогает оптимизировать использование ресурсов, улучшить обслуживание пассажиров и сделать движение общественного транспорта более эффективным и предсказуемым.

## Цели и задачи исследования

Рассмотрим подробнее такие задачи исследования, как:

* анализ процессов создания расписаний и выявление проблемы регулирования интервалов на остановках;
* изучение подходов к автоматизации регулирования интервалов автобусов и их недостатков.

### Особенности оптимизации расписания

Расписание движения автобусов ежегодно согласовывается с администрациями районов и городов, и утверждается государственным заказчиком на пассажирские перевозки. Этот процесс включает в себя множество аспектов, таких как прогнозирование пассажиропотоков, обеспечение равномерности расписания и адаптация к изменениям.

Прогнозирование пассажиропотоков является одним из важных аспектов разработки эффективного расписания движения автобусов. Это задача, требующая учета различных факторов, таких как день недели, время суток, сезонные колебания и особенности конкретного района города. Для этого используются статистические данные о поездках в разное время суток, анализ предыдущих периодов и учет особенностей каждого маршрута. Неверные прогнозы могут привести к недостатку или избытку автобусов на маршрутах, что влияет на комфорт и эффективность общественного транспорта.

Во избежание скопления или недостатка автобусов на маршрутах необходимо учитывать равномерность движения. Это означает, что автобусы должны прибывать на остановки с определенным интервалом, обеспечивая стабильность и комфорт для пассажиров.

Изменения в пассажиропотоке, дорожной ситуации или другие факторы могут потребовать быстрого корректирования расписания. Гибкость и возможность быстрого реагирования на такие изменения являются важными аспектами для обеспечения эффективной работы общественного транспорта.

### Присутствующие проблемы

Пассажиры часто сталкиваются с ситуациями, когда автобус опаздывает или приезжает раньше запланированного времени. Это может создавать неудобства и проблемы для пассажиров:

* при опоздании пассажир рискует не успеть на нужное ему время;
* ранний приезд может заставить пассажира ждать лишние минуты на остановке.

Водителю на длинном рейсе важнее всего вовремя оказаться на конечной остановке, а вот промежуточные он может проехать раньше срока [4]. Такое случается достаточно редко, но если всё-таки происходит, то действуют обычно следующим образом:

* можно позвонить перевозчику напрямую и узнать, по какой причине автобус не приехал в назначенное время;
* если дозвониться до водителя у диспетчера не получилось, или у этого перевозчика отсутствует диспетчерская, можно обратиться на горячую линию администрации города.

Нередки и ситуации, когда пассажир приходит на остановку, где, ожидая автобус, находится еще несколько человек. Если автобус приходит на остановку вовремя, но уже заполненным, пассажиры вынуждены оставаться на остановке, ожидая следующего, который также может прийти заполненным. Это может возникать из-за неравномерного пассажиропотока, пробок на дорогах и ограниченной вместимости автобусов.

Проблема одновременного подъезда нескольких автобусов к одной остановке может возникнуть в результате несогласованности графиков и интервалов движения. Автобусы разных парков имеют независимые расписания и могут возникать ситуации, когда к остановке подъезжают сразу несколько автобусов. Это может приводить к задержкам в отправлении, так как увеличивается время на высадку и посадку пассажиров.

В любом случае, даже после обращения к перевозчику, пассажир теряет время, что для него может обернуться негативными последствиями, такими как дисциплинарные взыскания в виде выговора, штрафа или увольнения, если человек спешил на работу, к примеру. Таким образом, проблема регулирования интервалов подъезда автобусов на остановки является актуальной и требует решения.

## Существующие методы регулирования

### Использование геолокации.

Системы, основанные на геолокации, позволяют отслеживать местоположение автобусов в реальном времени. Это дает возможность управлять информационными табло на остановках и предоставлять пассажирам актуальную информацию. Недостатком является то, что точность позиционирования может быть недостаточной из-за ошибок GPS, особенно в туннелях или рядом с высокими зданиями. Также, для работы этой системы в каждом автобусе необходимо установленное и настроенное на работу оборудование, позволяющее поддерживать связь со спутниками, но даже если такая система присутствует, не всегда удаётся быстро реагировать на изменения в трафике или другие дорожные события.

Пример: водители с проблемами с GPS могут отправлять неправильные данные о местонахождении автобуса, что влияет на информацию, предоставляемую пассажирам на остановках.

### Централизованная система управления.

Этот подход предполагает наличие центра регулирования, который непосредственно управляет движением городского общественного транспорта на основе информации о трафике, пассажиропотоке и других факторах. Однако существуют слабые места:

* необходимость в надёжном средстве связи и интеграции с различными транспортными системами. Должны быть задействованы человеческие ресурсы, координирующие работу такой системы, что означает некоторые экономические затраты в виде необходимости обучения персонала, расходов на зарплаты, аренды офисов и т. д.;
* возможность сбоев, в случае которых, в работе центрального узла может нарушиться работа всей транспортной сети, так как это проблема всех централизованных систем;

Пример: в случае сбоя в центральном узле управления автобусы могут двигаться без контроля, что приводит к хаосу на маршрутах.

### Интеллектуальные системы управления.

Эти системы объединяют технологии геолокации, машинного обучения и централизованного управления для оптимизации движения транспорта. Они способны предсказывать пассажиропотоки и оптимизировать маршруты. Из недостатков можно выделить значительные затраты на разработку и ввод в эксплуатацию подобных систем, а также необходимость в постоянном обновлении и поддержке.

Пример: системы машинного обучения могут адаптировать расписание автобусов к изменяющимся пассажиропотокам в разное время суток.

## Требования к разрабатываемому программному продукту

### Функциональные требования

Функциональные требования — это требования к поведению системы, они определяют функциональность разрабатываемого программного обеспечения, т. е. описывают возможности, которые предоставляет система:

* система должна обеспечивать работу с файлами Excel;
* необходимо предусмотреть объединение данных из нескольких файлов в один;
* система должна быть способна обработать маршрут длиной до 50 остановок;
* система должна инициализировать необходимые переменные при загрузке данных автоматически;
* должна быть реализована возможность построения графика расписания рейсов с использованием соответствующих библиотек;
* система должна отображать графики движения маршрутов с временными метками на оси X и остановками на оси Y;
* необходимо установить временные границы для отображения на графике;
* на графиках должна отображаться легенда для различных маршрутов и рейсов;
* необходимо предусмотреть подсчет количества коллизий для каждого рейса на каждой остановке;
* должна быть возможность отображения данных о движении в виде таблиц;
* необходимо предусмотреть возможность регулирования разными способами;
* должна быть реализована функция удаления выбранного маршрута из расписания;
* система должна сохранять удаленный маршрут и отдельные рейсы в Excel файлы с соответствующими названиями;
* необходимо предусмотреть возможность увеличения масштаба отображаемого графика;
* система должна обеспечивать уменьшение масштаба отображаемого графика;
* должна быть реализована функция обновления списка доступных маршрутов в выпадающем меню на основе загруженных данных.

### Требования к интерфейсу пользователя

Разрабатывая интерфейс пользователя, необходимо придерживаться принципов, определяющих взаимодействие с пользователем и саму структуру приложения в целом. Для реализации каждого принципа должен учитываться ряд требований:

* кнопки должны иметь интуитивно понятное текстовое или графическое обозначение. У кнопок с графическими обозначением должны быть иконки с соответствующим изображением выполняемых функций и не вызывать сомнений;
* все текстовые обозначения должны быть на русском языке, а элементы управления обладать чётко интерпретируемыми обозначениями;
* должны быть предусмотрены всплывающие окна с подсказками для облегчения ввода информации пользователем;
* в полях с ручным вводом данных должен присутствовать пример заполнения этого поля.

### Требования к программному обеспечению

Для функционирования приложения необходимо следующее:

* необходимо использовать объектно-ориентированный язык высокого уровня, например, Python, Java или C#;
* необходимо использовать библиотеки PyQt5 или PyQt6 для взаимодействия с файлом дизайна приложения;
* необходимо использовать библиотеку matplotlib для построения графиков;
* необходимо использовать библиотеку openpyxl для взаимодействия с файлами источника данных;
* необходимо использовать библиотеку Pandas для работы с данными и их анализа;

## Моделирование рабочих процессов

### BPMN-модель рабочего процесса «Как есть»

Рассмотрим процесс формирования интервалов подъезда автобусов на остановки.

Процесс начинается с получения заявки на изменение расписания, после чего проводится анализ приоритетов и необходимости изменения, согласование с представителем перевозчика и обсуждение особенностей эксплуатации автобусов. Заявка может быть подана как внутренними структурами (например, диспетчерами или представителями перевозчика), так и внешними (например, городской администрацией или пассажирами). Заявка содержит информацию о необходимости изменения расписания, причинах и предложениях по новому расписанию. Диспетчер или ответственный специалист анализирует заявку, определяет приоритет изменения расписания и его необходимость. После анализа заявки проводится обсуждение с представителем перевозчика, который эксплуатирует автобусы. В ходе обсуждения учитываются особенности эксплуатации, возможности перевозчика, а также финансовые и технические аспекты. На основе обсуждения и анализа определяются оптимальные интервалы движения автобусов, которые обеспечат максимальное соответствие пассажиропотоку и минимальное время ожидания на остановках. Расписание составляется с учетом оптимальных интервалов движения, времени в пути, промежуточных остановок и других факторов. Затем расписание передается городской администрации для согласования. Городская администрация проверяет соответствие расписания требованиям и нормативам, после чего расписание утверждается. Если расписание не соответствует требованиям и нормативам, оно возвращается диспетчеру для доработки. После внесения изменений расписание снова передается на согласование и утверждение. Утвержденное расписание передается диспетчеру и перевозчику, после чего публикуется, например, на официальном сайте перевозчика или городской администрации, в печатных изданиях или на остановках. Диаграмма приведена на рисунке 1.1.

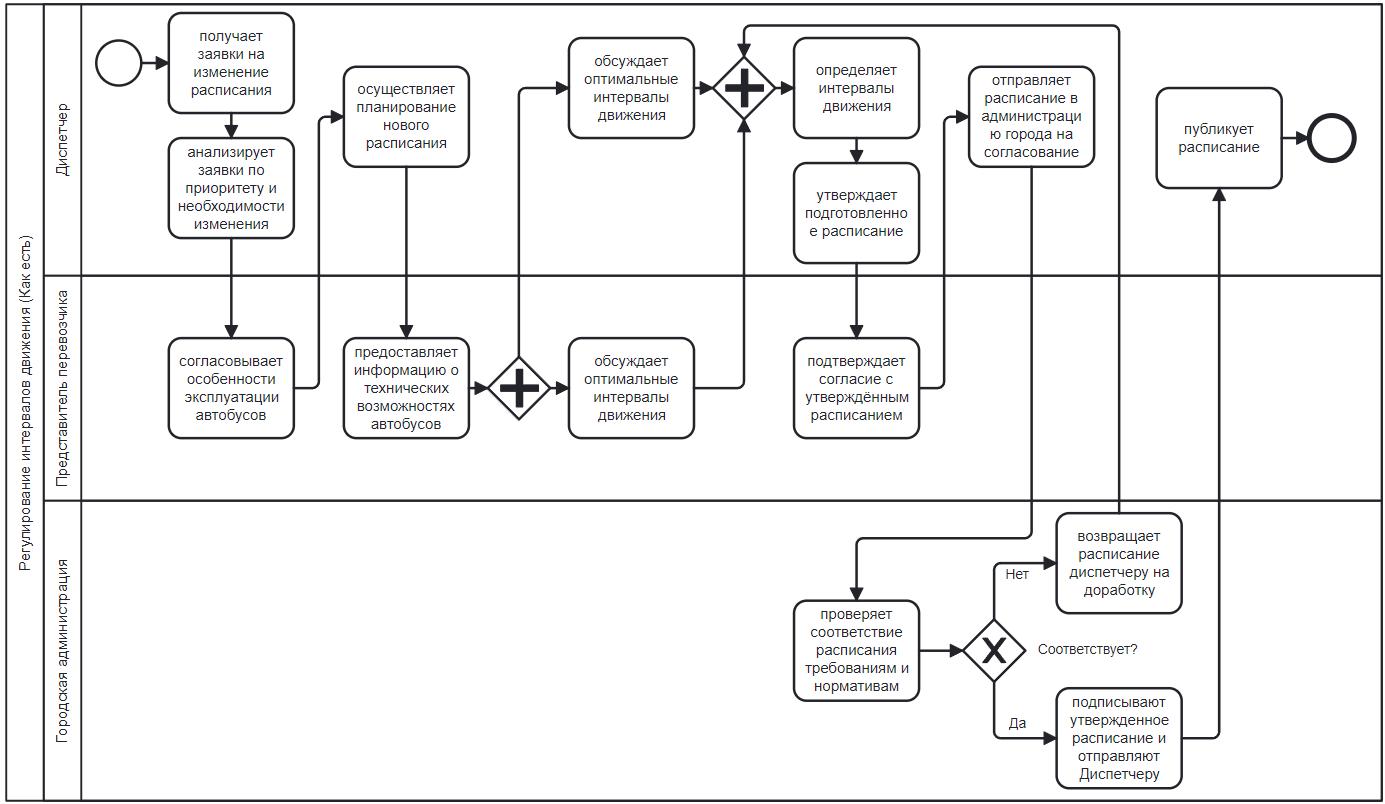


Рисунок 1.1 – BPMN-диаграмма модели «Как есть»

В модели BPMN используются роли «Диспетчер», «Представитель перевозчика» и «Городская администрация».

Так как процесс не учитывает действующие рейсы других перевозчиков, выявляется возможная проблема одновременного скопления автобусов разных маршрутов на остановке, вызванная независимостью расписаний движения автобусов разных маршрутов.

# Разработка алгоритма регулирования интервалов движения автобусов

## BPMN-модель рабочего процесса «Как должно быть»

Предлагаемый вариант процесса регулирования интервалов представляет собой схему управления расписанием движения автобусов, учитывающую все действующие рейсы. В модели используются роли «Администратор сервиса» и «Представитель перевозчика».

Сначала администратор запрашивает данные о маршрутных рейсах у перевозчика. Затем происходит проверка данных, преобразование и загрузка в хранилище (процесс ETL). Использование хранилища обусловлено удобством анализа данных. Далее данные загружаются в программу, в ней отображаются остановки, заявленные в маршрутных листах, и происходит расчет оптимальных интервалов движения автобусов. Вычисляется для каждой записи подъезда автобуса к остановке количество присутствующих автобусов на ней. В случае коллизий, анализируются рейсы, для которых значение этого показателя равно 1, и определяется, можно ли изменить время отправления для занимающего остановку автобуса. Если это возможно, то этот рейс отправляется на одну минуту раньше, а если нет, то задерживается следующий. После корректировки значение показателя вычисляется заново, и, если оно равно нулю, анализируется следующий рейс. Когда все противоречия устранены, то администратор отправляет изменённые маршрутные листы для всех рейсов каждого перевозчика на согласование. Диаграмма приведена на рисунке 2.1.

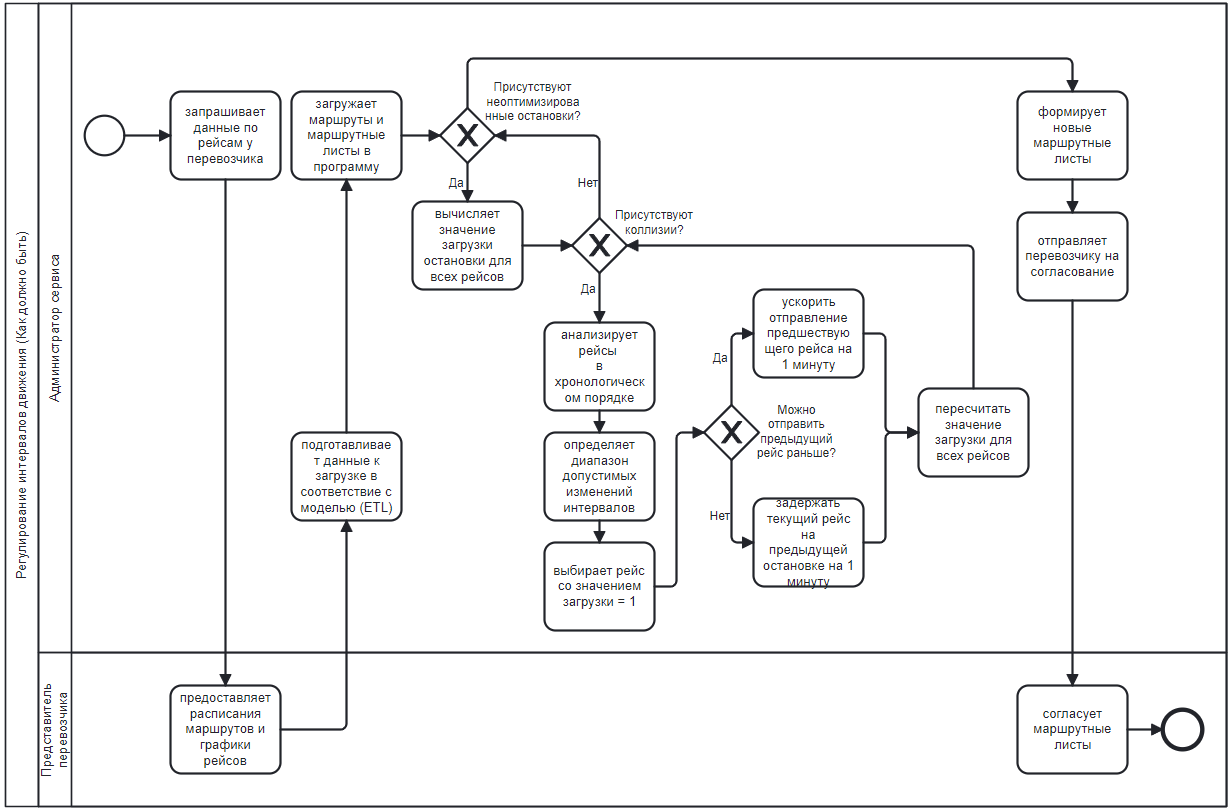


Рисунок 2.1 – BPMN-диаграмма модели «Как должно быть»

## Структура базы данных

На рисунке 2.2 приведена ER-модель базы данных для разрабатываемого сервиса:

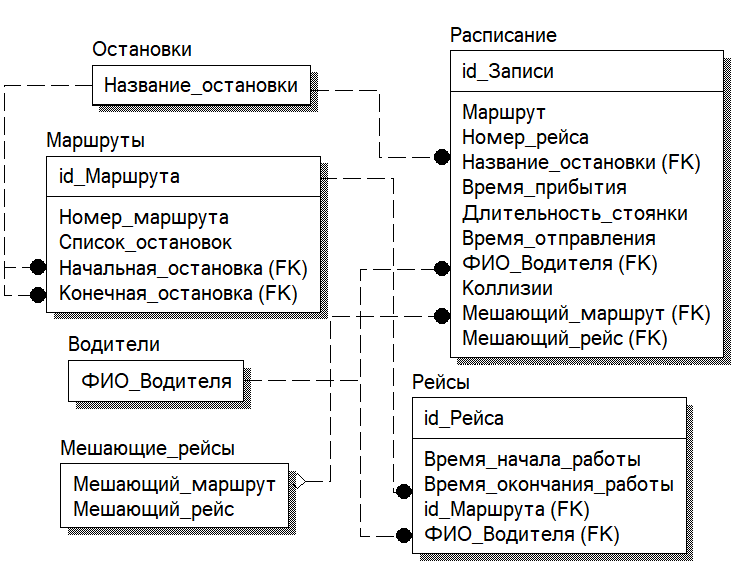


Рисунок 2.2 – ER-модель базы данных

ER-модель включает в себя следующие таблицы и атрибуты:

* таблица "Автобусы":
  1. id\_Автобуса - уникальный идентификатор автобуса (первичный ключ);
  2. Номер\_автобуса - номер автобуса;
  3. Вместимость - вместимость автобуса;
* таблица "Маршруты":
  1. id\_Маршрута - уникальный идентификатор маршрута (первичный ключ);
  2. Номер\_маршрута - номер маршрута;
  3. Начальная\_остановка – уникальный идентификатор остановки начала маршрута (внешний ключ);
  4. Конечная\_остановка – уникальный идентификатор остановки окончания маршрута (внешний ключ);
* таблица "Остановки" (Остановки):
  1. id\_Остановки - уникальный идентификатор остановки (первичный ключ);
  2. название\_остановки – название остановки;
* таблица "ФактыПрибытияИОтправления":
  1. id\_Факта - уникальный идентификатор факта (первичный ключ);
  2. id\_Остановки - идентификатор остановки (внешний ключ);
  3. Время - время прибытия на остановку или отправления с неё;
  4. Длительность\_остановки - длительность стоянки в минутах;
  5. ЗагруженностьОстановки – количество автобусов на остановке к моменту прибытия на неё;
  6. Прибытие/отправление – тип факта (прибытие или отправление);
  7. id\_Рейса (FK) – идентификатор рейса (внешний ключ);
  8. Номер\_остановки\_в\_маршруте – порядковый номер остановки для маршрута данного рейса;
* таблица «Рейсы»:
  1. id\_Рейса – уникальный идентификатор рейса (первичный ключ);
  2. Время\_начала\_работы – время начала работы рейса;
  3. Время\_окончания\_работы – время окончания работы рейса;
  4. id\_Маршрута – идентификатор маршрута (внешний ключ);
  5. id\_Автобуса – идентификатор автобуса (внешний ключ).

Связи между таблицами:

* таблица Автобусы связана с таблицей Рейсы по полю id\_Автобуса (первичный ключ);
* таблица Маршруты связана с таблицей Остановки по полю Начальная\_остановка (внешний ключ) и по полю Конечная\_остановка (внешний ключ) и с таблицей Рейсы по полю id\_Маршрута (первичный ключ);
* таблица Остановки связана с таблицей Маршруты двумя связями по полю id\_Остановки (первичный ключ) и с таблицей ФактыПрибытияИОтправления по полю id\_Остановки (первичный ключ);
* таблица Рейсы связана с таблицей Автобусы по полю id\_Автобуса (внешний ключ), с таблицей Маршруты по полю id\_Маршрута (внешний ключ) и с таблицей ФактыПрибытияИОтправления по полю id\_Рейса (первичный ключ);
* таблица ФактыПрибытияИОтправления связана с таблицей Рейсы по полю id\_Рейса (внешний ключ) и с таблицей Остановки по полю id\_Остановки (внешний ключ).Начало формы

## Описание пользовательского интерфейса

### Основное окно

Основное окно приложения (рисунок 2.3) содержит множество элементов управления, организованных в удобном и понятном виде. Весь интерфейс разделен на несколько горизонтальных и вертикальных секций, обеспечивая логическую группировку элементов.

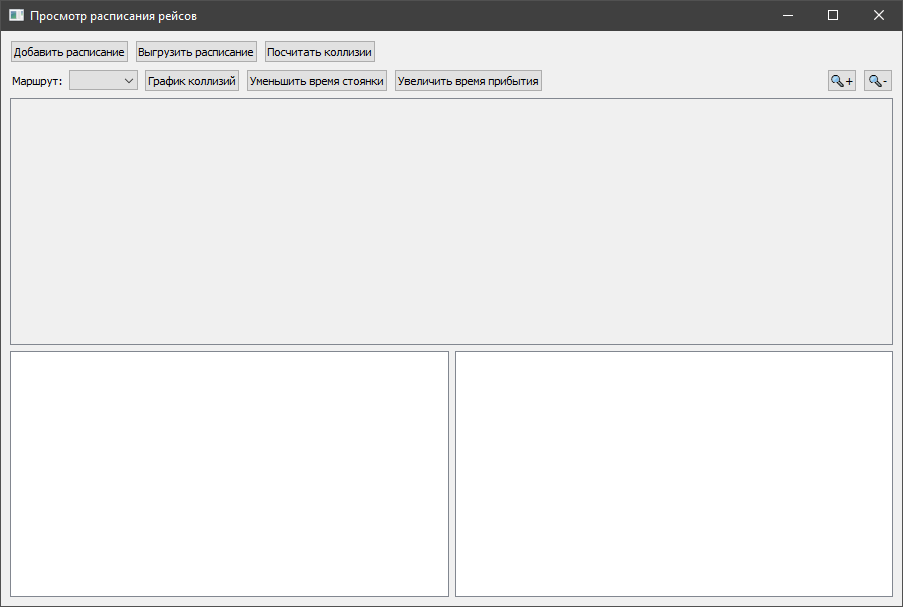


Рисунок 2.3 – Главное окно программы

### Верхняя панель инструментов

Кнопка "Добавить расписание" (рисунок 2.4): предназначена для загрузки расписания в приложение. Нажав эту кнопку, пользователь может выбрать файл, чтобы добавить расписание для дальнейшей обработки.



Рисунок 2.4 – Кнопка добавления расписания

Кнопка "Выгрузить расписание" (рисунок 2.5): используется для исключения выбранного маршрута из анализа и сохранения его текущего расписания в файл. Это позволяет сохранить внесенные изменения.



Рисунок 2.5 – Кнопка исключения расписания из анализа

Кнопка "Посчитать коллизии" (рисунок 2.6): активирует функцию расчёта коллизий в расписании. Это помогает выявить конфликтующие рейсы.



Рисунок 2.6 – Кнопка расчёта коллизий

Раскрывающийся список "Маршрут" (рисунок 2.7): этот элемент предназначен для выбора конкретного маршрута, который может быть сохранён при помощи кнопки «Выгрузить расписание».



Рисунок 2.7 – Раскрывающийся список маршрутов

Кнопка "График коллизий" (рисунок 2.8): показывает на графике только рейсы с коллизиями, скрывая остальные, помогая лучше понять проблемные места.



Рисунок 2.8 – Кнопка построения графика рейсов с коллизиями

Кнопка "Уменьшить время стоянки" (рисунок 2.9): уменьшает время стоянки проблемных рейсов, уменьшая количество коллизий.



Рисунок 2.9 – Кнопка уменьшения времени стоянки

Кнопка "Увеличить время прибытия" (рисунок 2.10): увеличивает время прибытия проблемных рейсов для аналогичной цели.



Рисунок 2.10 – Кнопка увеличения времени прибытия

Кнопки увеличения/уменьшения масштаба графика (рисунок 2.11): позволяют пользователю приблизить или отдалить график для более детального просмотра.



Рисунок 2.11 – Кнопки изменения масштаба графика

### Область построения графика

На рисунке 2.12 представлена рабочая область главного окна, визуализирующая загруженные расписания для лучшей наглядности и удобства работы пользователя.

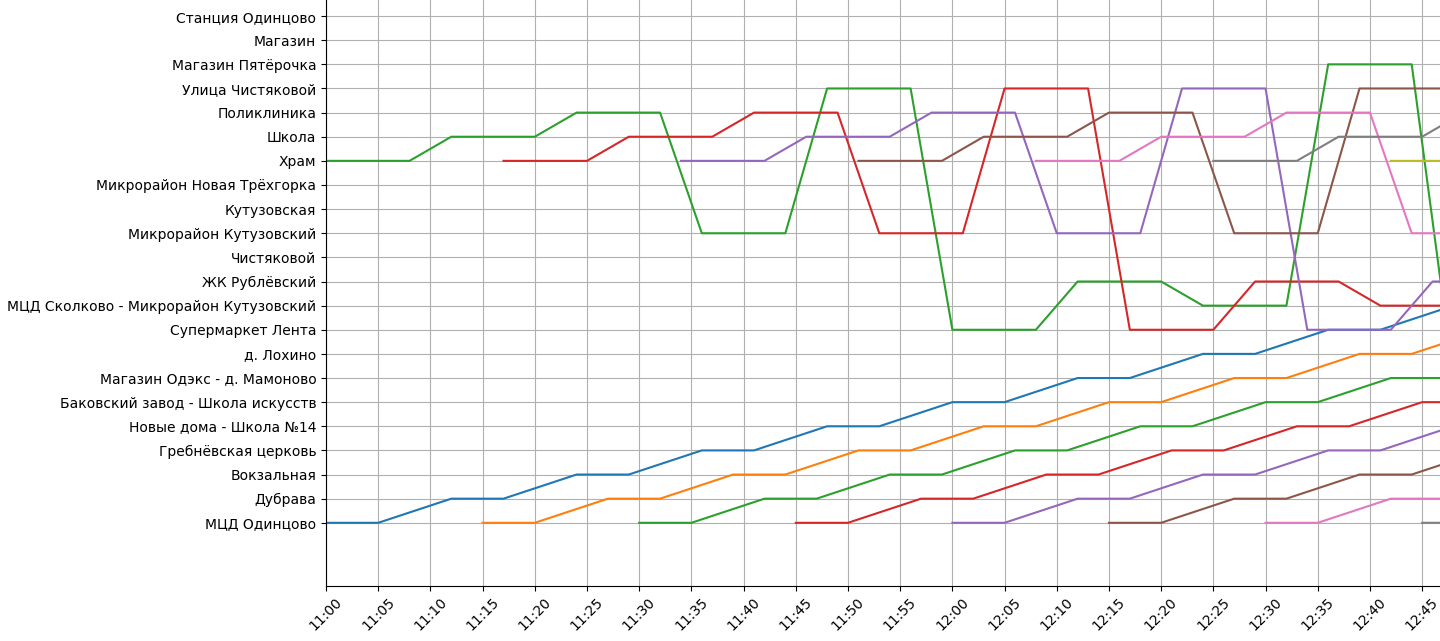


Рисунок 2.12 – Область построения графика

Эта область содержит график расписания рейсов. Пользователь может прокручивать эту область, чтобы увидеть все детали графика. Вертикальная ось содержит названия остановок (Рисунок 2.13), горизонтальная ось – временная шкала (Рисунок 2.14).

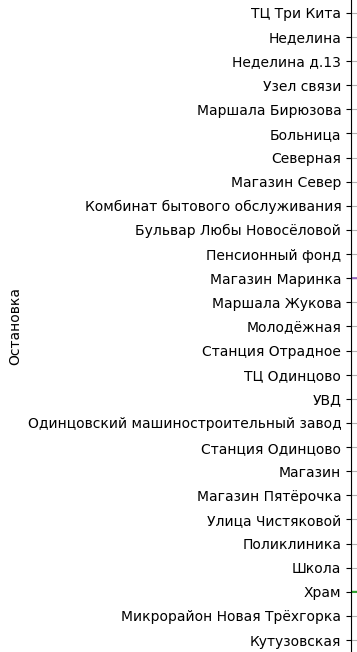


Рисунок 2.13 – Вертикальная ось графика

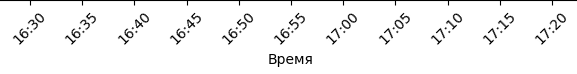


Рисунок 2.14 – Горизонтальная ось графика

Цветными линиями на графике изображается движение каждого рейса в расписании, позволяя пользователю легко определить, какой рейс нуждается в регулировании. В правом углу графика отображена легенда, на которой указано, какому рейсу соответствует тот или иной цвет линии.

### Табличные виджеты

В нижней части главного окна располагаются две таблицы, предоставляющие детальную информацию о расписании.

Таблица коллизий (рисунок 2.15): отображает рейсы с коллизиями. Это помогает пользователю быстро идентифицировать и решать проблемы.



Рисунок 2.15 – Виджет таблицы коллизий

Таблица полного расписания (рисунок 2.16): отображает все рейсы в виде таблицы, предоставляя пользователю полную информацию о движении.

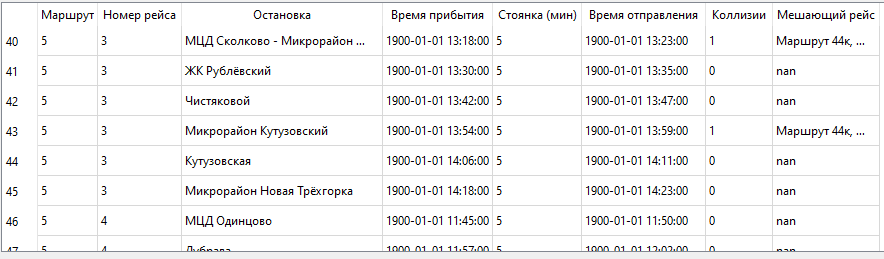


Рисунок 2.16 – Виджет таблицы полного расписания

### Иерархическая структура логической схемы интерфейса

Для визуализации поведения пользовательского интерфейса приложения приведём несколько диаграмм, отражающих набор доступных пользователю функций, зависящих от состояния системы, на рисунках 2.17, 2.18, 2.19 и 2.20:



Рисунок 2.17 – Логическая схема для начального состояния программы



Рисунок 2.18 – Логическая схема для состояния «расписание в памяти»

При нахождении в состоянии «Расписание в памяти», пользователь может перейти в начальное состояние, в состояние «Есть коллизии» или остаться в том же состоянии.



Рисунок 2.19 – Логическая схема для состояния «есть коллизии»

При нахождении в состоянии «Есть коллизии», пользователь может перейти в состояние «Нет коллизий» или остаться в том же состоянии.



Рисунок 2.20 – Логическая схема для состояния «нет коллизий»

## Описание функционала приложения

### Используемые инструменты

PyCharm является одним из самых популярных и мощных интегрированных сред разработки (IDE) для языка программирования Python. Он предоставляет удобный интерфейс и множество функций, которые облегчают разработку, отладку и тестирование кода. В этом проекте PyCharm был использован для написания и редактирования кода, навигации по проекту и интеграции с системами контроля версий. Одним из основных преимуществ PyCharm является его интеллектуальный редактор, который поддерживает автодополнение кода, рефакторинг, и быстрый доступ к документации. Эти функции значительно ускорили процесс разработки и помогли избежать многих ошибок.

GitHub — это веб-сервис для хостинга проектов, использующих систему контроля версий Git. В этом проекте GitHub был использован для хранения исходного кода и ведения истории изменений. С помощью GitHub можно легко отслеживать все изменения в коде, возвращаться к предыдущим версиям и эффективно решать конфликты, возникающие при объединении изменений. Также GitHub предоставляет инструменты для управления проектами, такие как issues и pull requests, которые помогли организовать процесс разработки и следить за выполнением задач.

QtDesigner — это инструмент для создания графических пользовательских интерфейсов (GUI) на основе библиотеки Qt. В этом проекте QtDesigner использовался для разработки интерфейса программы. Он предоставляет визуальный способ создания интерфейсов, что позволяет быстро и легко добавлять, и настраивать элементы управления, такие как кнопки, поля ввода, таблицы и другие компоненты. Использование QtDesigner значительно упростило процесс создания интерфейса и позволило сосредоточиться на логике программы.

Pandas — это мощная библиотека для анализа и обработки данных в Python. В этом проекте pandas использовался для загрузки, обработки и анализа данных расписания автобусов. Библиотека предоставляет удобные инструменты для работы с табличными данными, такие как DataFrame, что позволяет легко манипулировать и анализировать данные. Pandas позволяет быстро загружать данные из файлов Excel, обрабатывать их и строить различные отчеты.

Matplotlib — это библиотека для создания визуализаций данных в Python. В этом проекте matplotlib использовался для построения графиков расписания рейсов и коллизий. Библиотека предоставляет широкий набор инструментов для создания различных типов графиков и диаграмм, что позволяет наглядно представлять данные и делать их более понятными.

PyQt5 — это модуль Python для библиотеки Qt, который позволяет создавать кроссплатформенные приложения с графическим интерфейсом. В этом проекте PyQt5 использовался для создания GUI программы, что обеспечило удобный и интуитивно понятный интерфейс для пользователей. PyQt5 предоставляет множество готовых компонентов и виджетов, которые можно легко настроить и интегрировать в приложение. Это позволило быстро разработать функциональный и привлекательный интерфейс, который значительно упростил взаимодействие пользователя с программой.

NumPy — это библиотека для работы с массивами и матрицами, а также для выполнения высокоуровневых математических функций. В этом проекте NumPy использовался для эффективной обработки и анализа данных, особенно при работе с большими объемами данных. Использование NumPy позволило значительно ускорить вычислительные процессы и оптимизировать алгоритмы анализа данных.

Openpyxl — это библиотека для работы с файлами Excel в формате .xlsx. В этом проекте openpyxl использовался для загрузки и сохранения данных расписания автобусов в файлах Excel. Эта библиотека предоставила удобный интерфейс для работы с Excel файлами, что позволило легко интегрировать данные из различных источников и сохранять результаты анализа в удобном для пользователей формате.

Использование этих инструментов и библиотек значительно упростило и ускорило процесс разработки приложения, главная задача приложения - анализ расписания автобусов и управление интервалами движения между рейсами.

### Диаграмма компонентов программы

На рисунке 2.21 представлена диаграмма компонентов программного обеспечения.

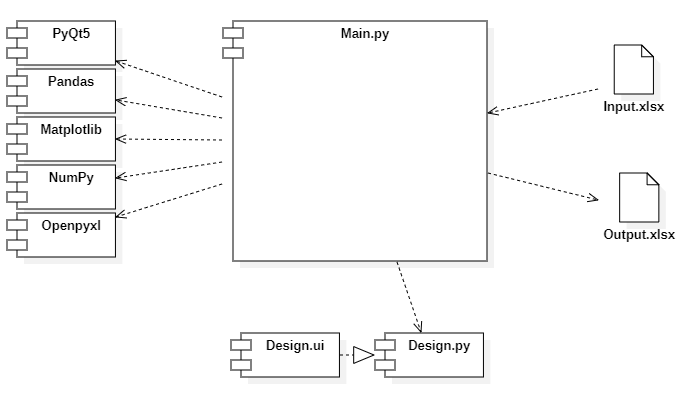


Рисунок 2.21 – Диаграмма компонентов приложения

### Основные функции

Загрузка расписания: функция load\_schedule позволяет пользователю выбрать и загрузить файл Excel с расписанием рейсов. Загрузка данных производится из файла в Pandas DataFrame (набор данных). Если нужно загрузить несколько файлов, этой функцией можно воспользоваться несколько раз, данные при этом объединяются. Функция может обрабатывать файлы как для одного, так и для нескольких маршрутов, в зависимости от содержимого файла. Файлы таблиц Excel принимаются как с расширением .xlsx, так и с расширением .xls, формата старого образца.

Визуализация расписания: функция plot\_schedule строит график расписания рейсов с использованием matplotlib. Рисуется координатная сетка, с подписями осей и названиями значений на этих осях. График отображает маршруты и рейсы, с временными метками на оси X и остановками на оси Y. На одной координатной плоскости рисуются линии разными цветами, изображающие движение каждого автобуса от начальной остановки до конечной, с учётом времени стоянки на каждой остановке. Помимо разноцветных линий, в местах их пересечений может располагаться яркий маркер в виде красной точки.

Отображение маркеров для проблемных рейсов: функция plot\_collisions\_markers предназначена для того, чтобы на графике движения автобусов при наличии конфликтующих ситуаций в точке скопления автобусов рисовался красный маркер, это удобный способ оперативно определить проблемные места в расписании.

Подсчет и отображение коллизий:

Функция calculate\_collisions вычисляет количество коллизий для каждого рейса на каждой остановке. В качестве источника данных выступают все загруженные в расписание файлы, преобразованные в Pandas DataFrame. Просматривается каждая строка в DataFrame, в цикле для каждой строки составляется список строк, для которых выполняется условие: ((Время прибытия <= Времени прибытия в текущей строке < Времени отправления) & (Индекс строки != индексу текущей строки)). Строки, найденные по данному условию, означают записи прибывших на остановку автобусов и не успевших её освободить к моменту приезда анализируемого автобуса. Этот список в формате «Маршрут №, Рейс №» заносится в столбец Мешающий рейс, а количество элементов этого списка заносится в столбец Коллизии. Функция дополнительно вызывается как при загрузке файла с расписанием, так и при регулировке интервалов функциями регулирования.

Функция show\_collisions заполняет таблицу коллизий только рейсами с коллизиями в таблице. Для этого на вход поступает набор данных всего расписания, в цикле просматривается каждая строка и фильтруется по условию «Значение в поле Коллизии > 0», после чего записывается в память в виде отфильтрованного набора данных и выводится в табличном виде на экран в левый табличный виджет главного окна. Функция вызывается каждый раз при вызове функции подсчёта коллизий, а также после вызова функций регулировки интервалов.

Функция show\_colliding\_routes заполняет таблицу всего расписания, принимая на вход весь набор данных загруженного расписания, затем отображает все маршруты и рейсы с указанием коллизий. Для вывода на экран используется правый табличный виджет. Функция так же вызывается каждый раз при вызове функции подсчёта коллизий, а также после вызова функций регулировки интервалов.

Регулирование интервалов подъезда автобусов:

Функция increase\_arrival\_time — это функция регулирования, принимающая на вход набор данных, отфильтрованный по ненулевому количеству мешающих автобусов, в котором просматриваются все записи с коллизиями, и увеличивает значение поля «время прибытия» текущего рейса для текущей остановки на 1 минуту, чтобы автобусы не встретились на остановке. Время не может быть изменено, если это изменение приведёт к равенству времени прибытия и отправления, означающее нулевое время стоянки, что приведёт к невозможности нормальной высадки и посадки пассажиров. В случае, когда два рейса имеют одинаковое время прибытия и мешают друг другу, регулируется только один из них, для этого при просмотре всего набора данных и нахождении строки, в поле «мешающий рейс» которой содержится значение маршрута и рейса текущей строки, индекс такой строки записывается в специальный список строк, которые необходимо исключить из обработки. При следующей итерации идет проверка на наличие индекса анализируемой строки в этом списке, если таковой находится в нем, то осуществляется переход к следующей строке.

Функция decrease\_stop\_time — это аналогичная функция регулирования, принимающая на вход набор данных, так же отфильтрованный по ненулевому количеству мешающих автобусов, в котором просматриваются все записи с коллизиями, и уменьшает значение поле «время отправления» рейсов, перечисленных в столбце Мешающий рейс на 1 минуту, чтобы устранить коллизии. Время не может быть изменено, если это изменение приведёт к равенству времени прибытия и отправления. В случае, когда два рейса имеют одинаковое время прибытия и мешают друг другу, регулируется только один из них, для этого при входе в цикл просмотра всего набора данных предварительно создаётся вспомогательный список исключенных из обработки индексов строк. Набор данных просматривается и при нахождении строки, в поле «мешающий рейс» которой содержится значение маршрута и рейса текущей строки, индекс такой строки записывается в список исключенных индексов, предотвращая двойное регулирование. При следующей итерации наличие индекса анализируемой строки в списке исключенных строк не даст выполнить регулировку необходимого поля и осуществится переход к следующей строке.

Эти функции применяются к рейсам с коллизиями разного уровня, начиная с уровня 1.

Построение графика коллизий: функция plot\_collisions строит график рейсов, у которых есть коллизии, аналогично функции plot\_schedule. Исключение линий рейсов без коллизий позволяет пользователю видеть проблемные участки расписания.

Удаление маршрута: функция remove\_route отключает отображение линий выбранного в раскрывающемся списке маршрута с графика, исключает строки, связанные с ним из таблицы коллизий, из таблицы всего расписания, при этом маршрут перед удалением сохраняется в отдельную директорию с названием ли номером маршрута, а в ней в несколько Excel файлов: один общий файл для всех рейсов этого маршрута и по одному небольшому файлу для каждого рейса с указанием в названии начало и конец следования. При наличии дополнительной информации вроде регистрационного знака транспортного средства или ФИО водителя эта информация также попадёт туда.

Масштабирование графика: функции increase\_scale и decrease\_scale увеличивают и уменьшают масштаб отображаемого графика, соответственно.

Заполнение выпадающего списка маршрутов: функция fill\_route\_combo\_box нужна для заполнения список доступных маршрутов в раскрывающемся списке при добавлении нового, или удалении существующего. Функция просматривает набор данных и выбирает из него уникальные значения столбца «Маршрут», затем заполняет коллекцию в элемент «раскрывающийся список».

## Контрольный пример и инструкция пользователя

Для демонстрации работоспособности приложения потребуется несколько файлов маршрутных листов, по одному на каждый маршрут. На рисунках 2.22, 2.23 и 2.24 приведен примерный вид содержимого этих файлов.



Рисунок 2.22 – Маршрутный лист маршрута №5



Рисунок 2.23 – Маршрутный лист маршрута №44к



Рисунок 2.24 – Маршрутный лист маршрута №339

Эти маршруты имеют смежные остановки, разный интервал запуска каждого следующего автобуса в рейс, разную длительность стоянки на остановках и одинаковое время начала работы. Для начала работы над регулировкой интервалов подъезда необходимо запустить приложение, откроется главное окно программы (Рис. 2.3), в котором пользователю при нажатии кнопки «Добавить расписание» (рисунок 2.4) предлагается выбрать файл Excel с данными маршрута (рисунок 2.25).

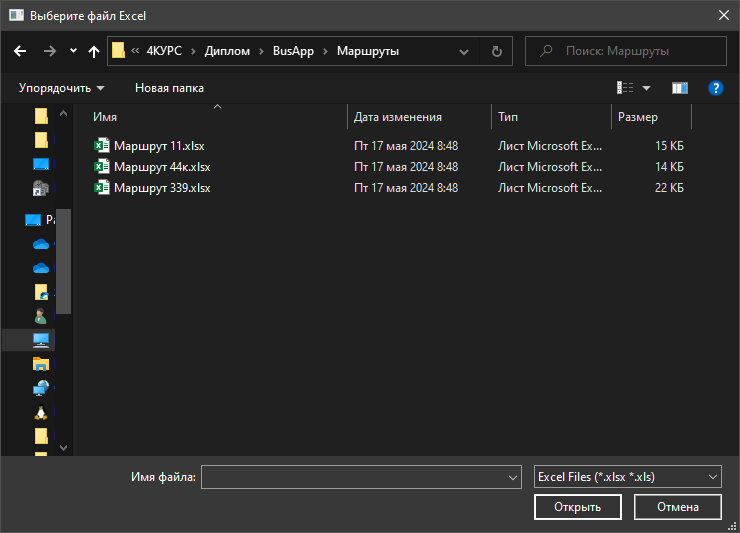


Рисунок 2.25 – Выбор файла с расписанием

При выборе файла с расписанием в области построения графика главного окна (рисунок 2.12) отобразится графически содержимое файла, а в раскрывающемся списке маршрутов (рисунок 2.7) появятся названия всех маршрутов, содержащихся в файле с расписанием. Если нажать любую другую кнопку, в то время как никаких данных не загружено, например, кнопку подсчёта коллизий (рисунок 2.6), появится окно с предупреждением, что данных для анализа нет (рисунок 2.26)

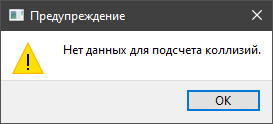


Рисунок 2.26 – Предупреждение об отсутствии данных для анализа

Загрузим файл расписания маршрута №339, главное окно программы отреагирует появлением графика с информацией об этом расписании (рисунок 2.27)

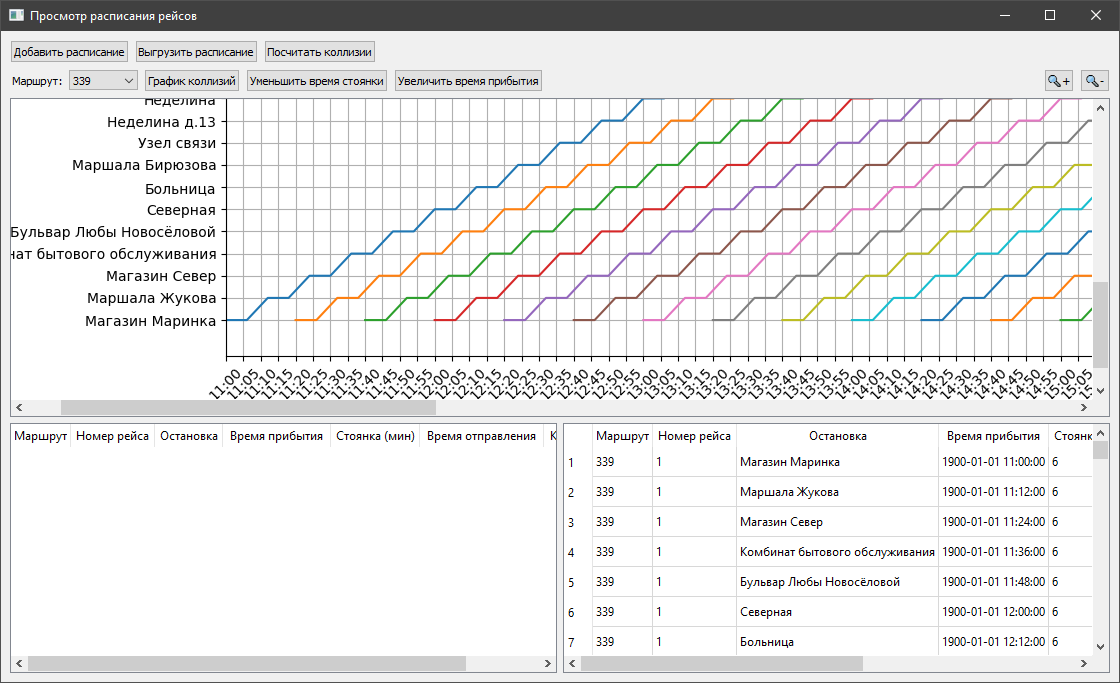


Рисунок 2.27 – Расписание маршрута №339

Теперь пользователю становятся доступны функции регулирования интервалов подъезда автобусов, для этого необходимо нажать кнопку подсчёта коллизий (рисунок 2.6), после чего главное окно приобретет вид, показанный на рисунке 2.27.

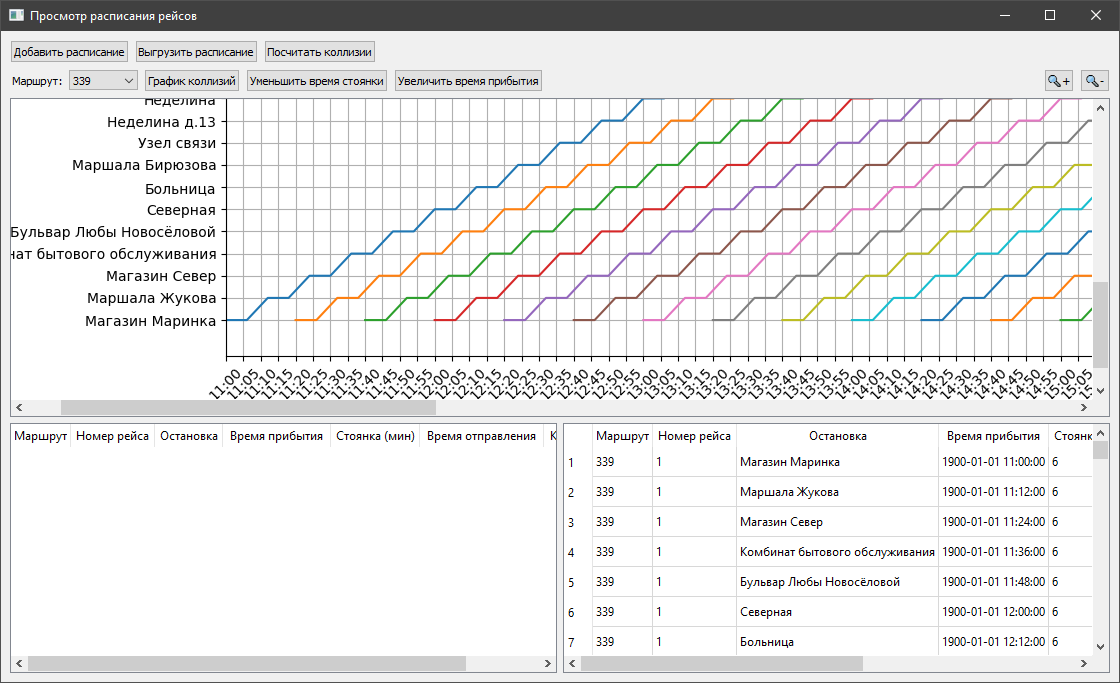


Рисунок 2.27 – Вид главного окна после подсчёта коллизий

В левом табличном виджете данные отсутствуют, это означает, что расписание не имеет коллизий, и автобусы на остановках не кучкуются. Добавим к анализу больше данных, для этого загрузим расписания еще двух маршрутов, №5 и №44к. После подсчёта коллизий окно примет вид (рисунок 2.28).

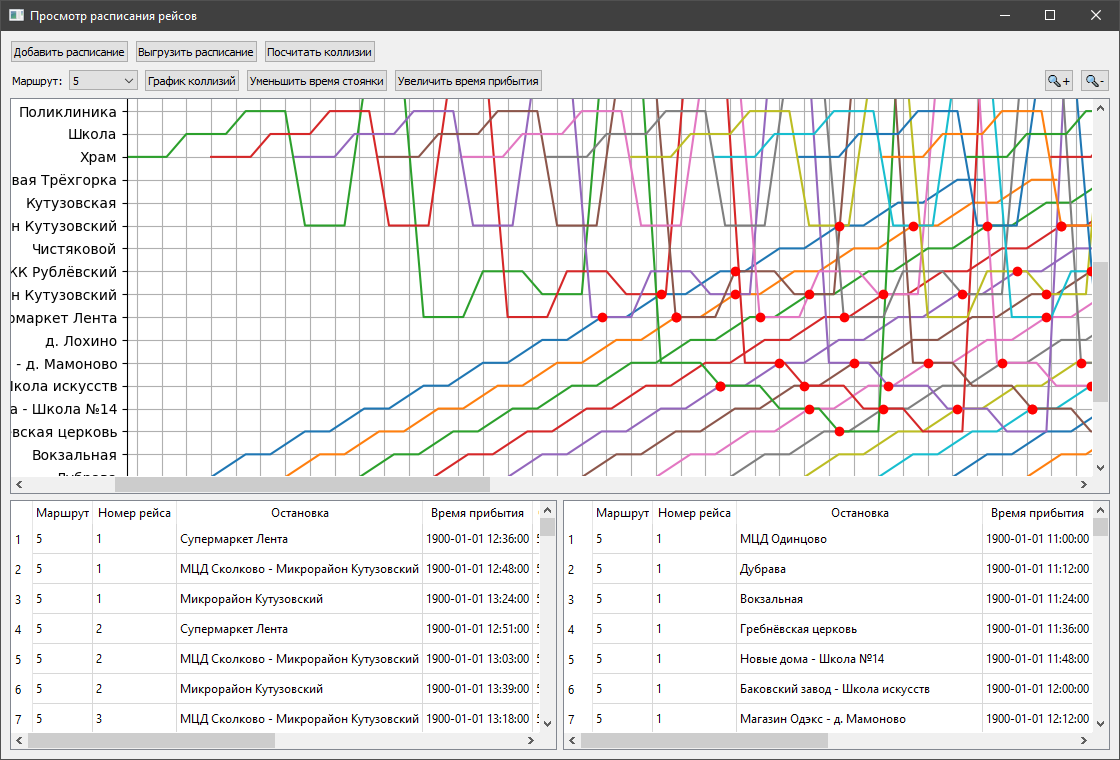


Рисунок 2.28 – Главное окно после подсчёта коллизий трёх маршрутов

На графике появились красные точки – маркеры коллизий. Это означает, что в данный момент времени на остановке находится больше одного автобуса, что затрудняет правильную посадку и высадку пассажиров. В левом табличном виджете появились данные о таких случаях, каждая запись означает один случай нахождения автобуса на остановке при подъезде к ней. На данном этапе количество таких коллизий равно 75. Уменьшить их количество могут функции регулировки: увеличения времени прибытия и уменьшения длительности стоянки. Пользователь может по своему усмотрению, в зависимости от предпочтений, пользоваться любой из них. Воспользуемся функцией уменьшения времени стоянки (рисунок 2.9), после чего пересчитаем коллизии (рисунок 4.9)

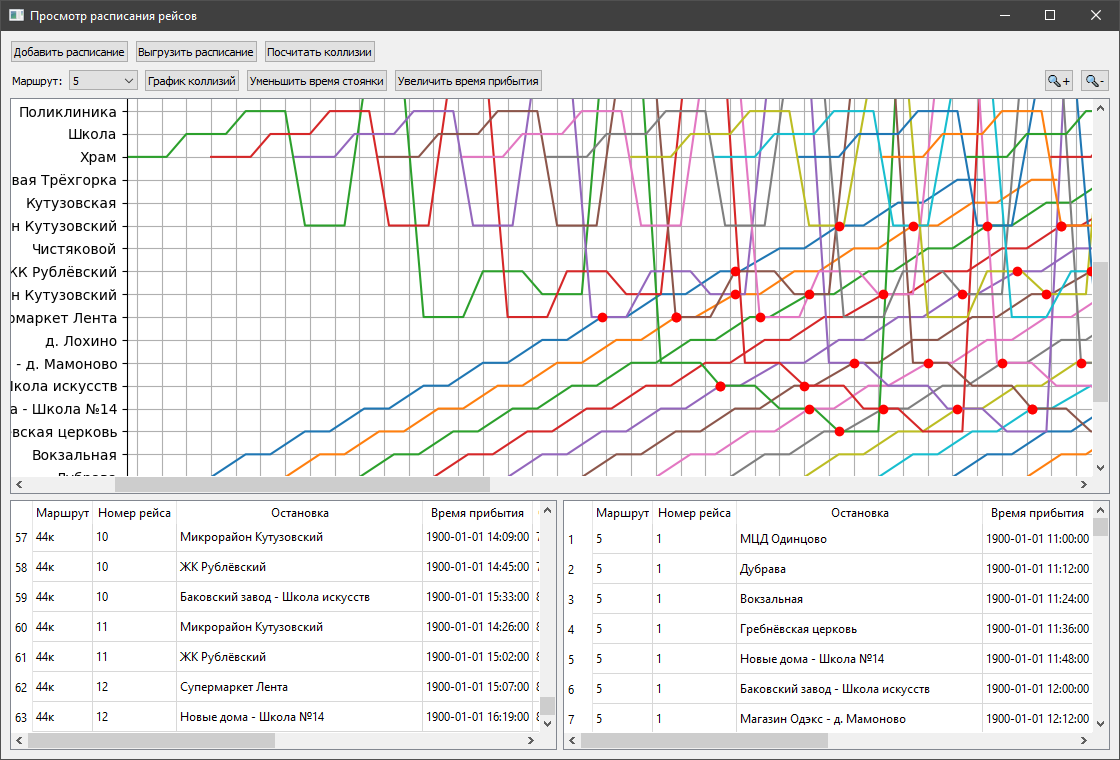
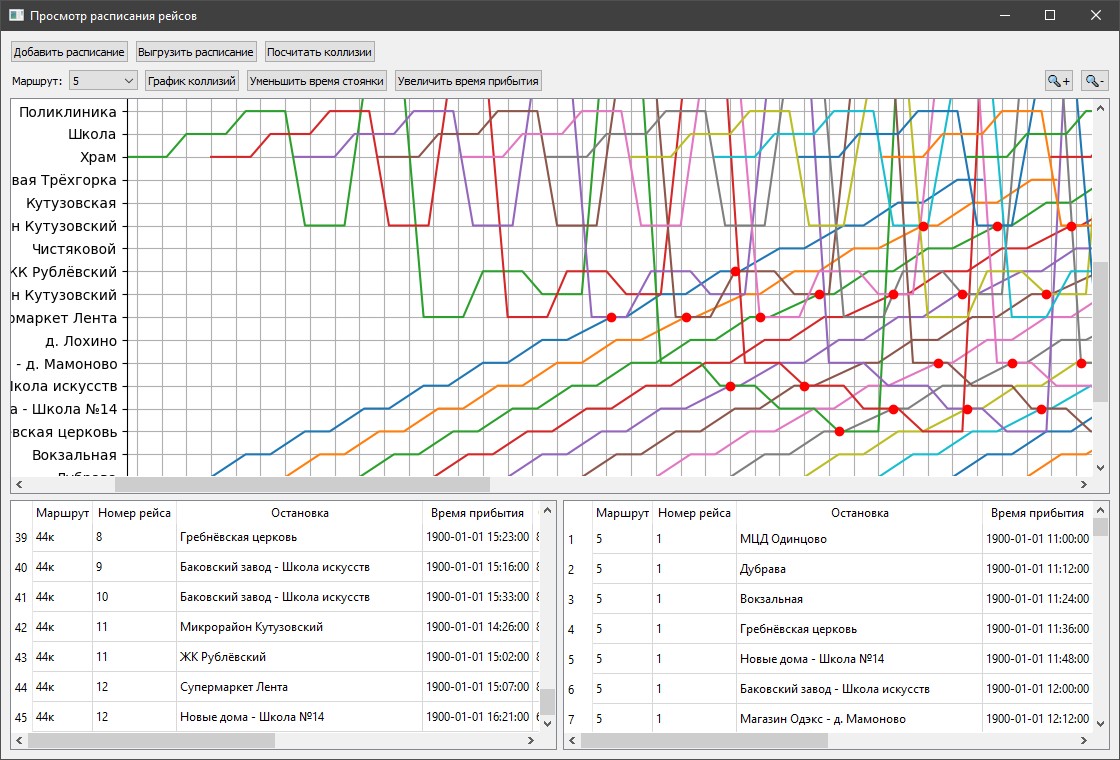
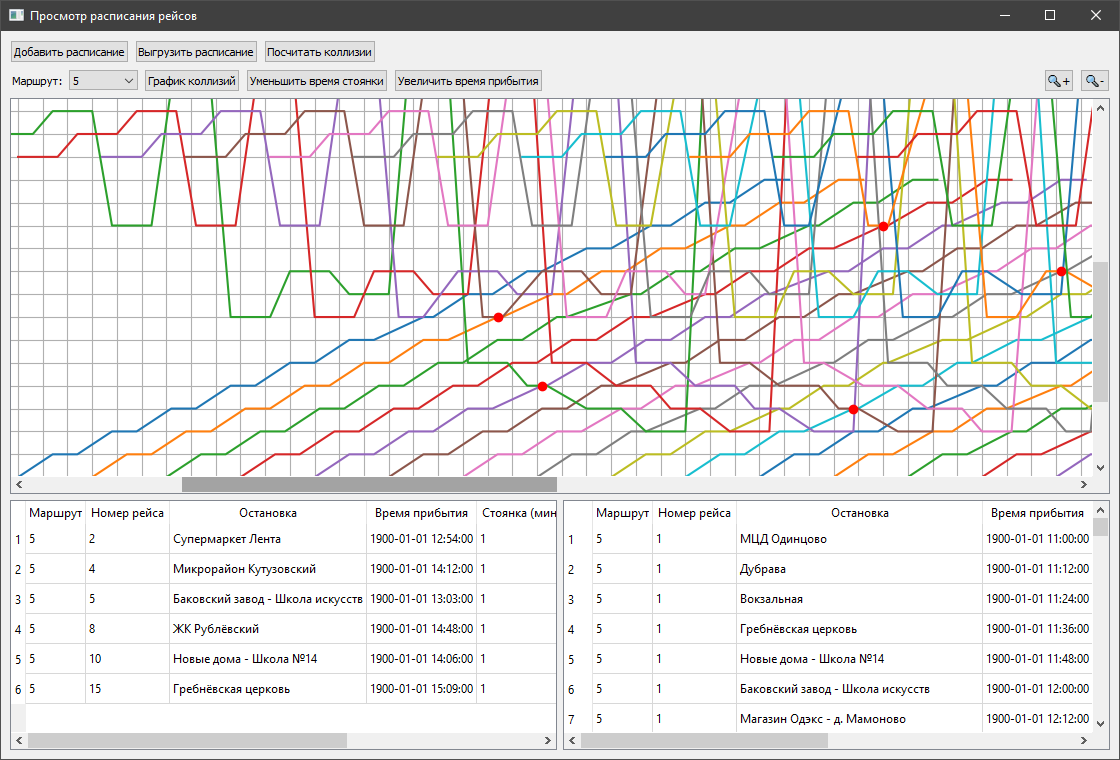


Рисунок 2.29 – Главное окно после уменьшения времени стоянки

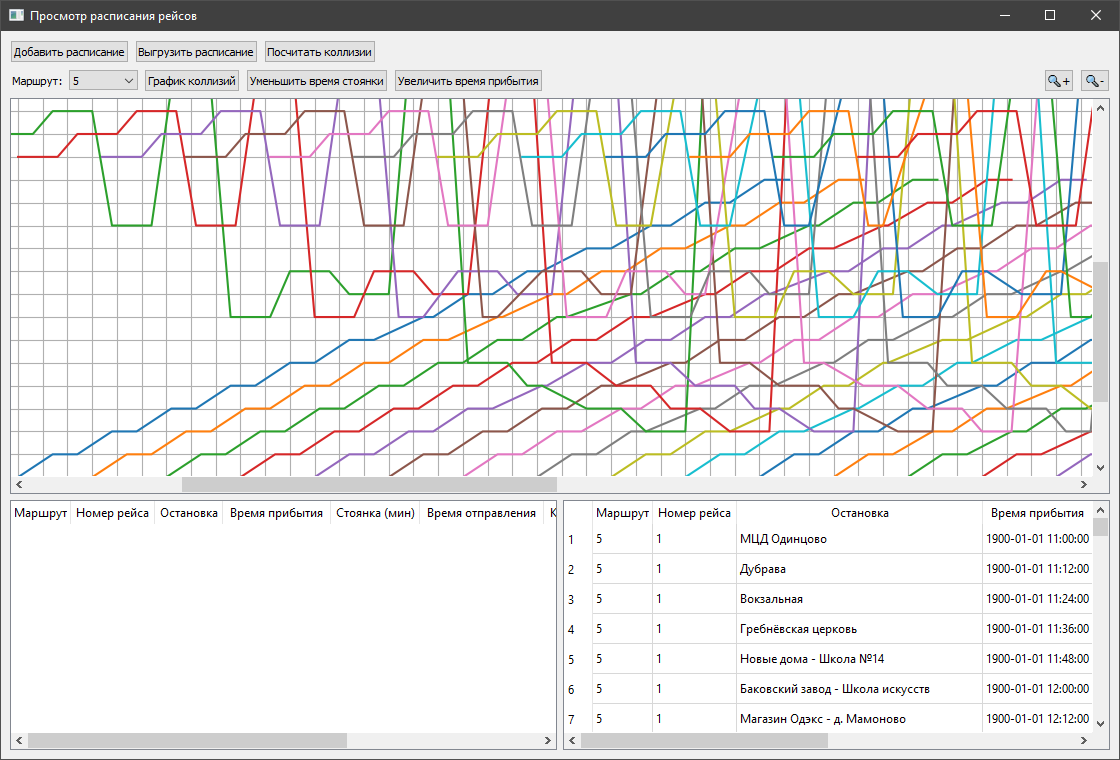
После успешной регулировки график изменился, и количество коллизий сократилось до 51. Воспользуемся второй функцией, увеличения времени прибытия на остановку (рисунок 2.10), затем снова пересчитаем коллизии (рисунок 2.30).

 Рисунок 2.30 – Вид графика после увеличения времени прибытия

Пользователь может повторять операции регулировки до тех пор, пока количество коллизий сократится до минимума (рисунок 2.31).

Рисунок 2.31 –График расписания после нескольких итераций регулировки

Так как для оставшихся рейсов с коллизиями время прибытия увеличить невозможно в связи с минимальной длительностью стоянки, воспользуемся функцией уменьшения стоянки для мешающих им рейсов. Результат приведён на рисунке 2.32.

 Рисунок 2.32 – Результат регулировки интервалов подъезда автобусов

В левом табличном виджете больше не осталось проблемных записей, а в правом для всех записей количество коллизий стало равно нулю, работа над расписанием завершена, теперь нужно сохранить изменённые маршрутные листы в отдельные файлы для каждого маршрута. Для этого выбираем маршрут из списка (рисунок 2.7) и нажимаем кнопку «Выгрузить расписание» (рисунок 2.5), в результате с графика пропадут линии, соответствующие выбранному маршруту, и появится уведомление о том, что расписание сохранено в файл (рисунок 2.33).

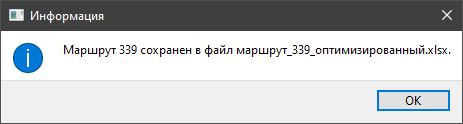


Рисунок 2.33 – Уведомление об успешном сохранении расписания в файл

После последовательного сохранения расписаний всех анализируемых маршрутов, появится уведомление об отсутствии данных для отображения графика (рисунок 2.34), на этом работа над расписанием завершена, и главное окно примет начальный вид, представленный на рисунке 2.3.

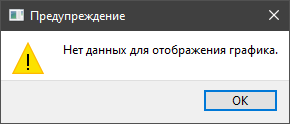


Рисунок 2.34 – Предупреждение об отсутствии данных для отображения

На рисунке 2.35 представлено содержимое директории с сохранёнными после обработки файлами для маршрута 5.

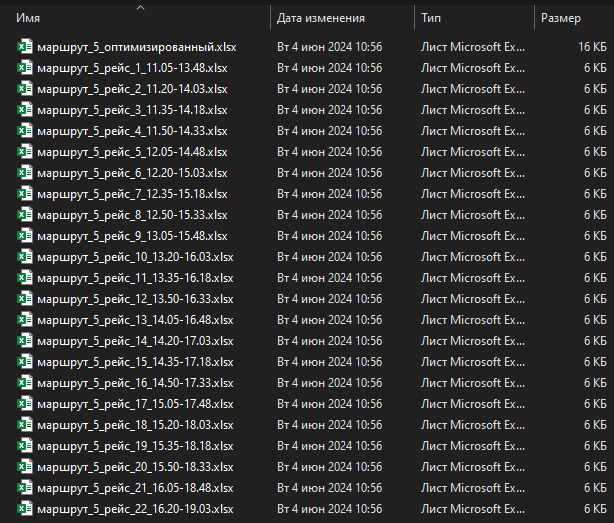


Рисунок 2.35 – Перечень файлов выходных данных

# Системотехнические расчёты

## Метод экспертных оценок

Для обоснования целесообразности создания сервиса регулирования интервалов подъезда автобусов на остановках города используем метод экспертных оценок.

В рамках данного исследования рассматриваются три способа регулирования интервалов подъезда:

* без использования автоматизации, вручную при помощи табличного процессора Microsoft Excel – данный способ является устаревшим и постепенно выводится из эксплуатации;
* с помощью программы для планирования маршрутов общественного транспорта Merakas Pikas;
* с помощью сервиса регулирования интервалов подъезда автобусов на остановках города, который разрабатывается в данном исследовании.

Для выбора наиболее предпочтительного способа пригласим экспертов, специализирующихся в данной сфере, но не связанных с деятельностью автобусного транспорта, и обработаем их мнения. Группой экспертов будет выступать несколько работников метрополитена, диспетчеры и машинисты.

Критерии, по которым будем проводить опрос экспертов, следующие:

* быстродействие;
* полнота функционала;
* удобство использования;
* независимость от импорта.

Эксперты будут опрошены независимо друг от друга, чтобы получить индивидуальные оценки по каждому критерию для трех способов регулирования интервалов движения автобусов. Для оценки коэффициентов важности критериев, попросим экспертов дать ранжированные оценки степени важности критериев. Затем мы проверим гипотезу о случайной простановке оценок важности критериев (рангов), затем вычислим коэффициенты предпочтительности вариантов выбора и сможем сделать вывод о наиболее эффективном способе регулирования интервалов подъезда автобусов на остановки города.

## Опрос экспертов

В ходе опроса группы приглашенных экспертов о степени соответствия выбранным критериям (быстродействие, полнота функционала, удобство использования, независимость от импорта) каждого способа регулирования интервалов движения автобусов (вручную при помощи Excel, при помощи Pikas или при помощи разрабатываемого в ходе исследования ПО), были получены следующие оценки (таблица 1). Значения варьируются от 1 до 5, где 1 – минимальная предпочтительность, 5 – максимальная.

Таблица 1 – Оценки способов по каждому критерию

|  |  |  |  |
| --- | --- | --- | --- |
| Критерии | Способ | | |
| Ручной режим | Merakas Pikas | Разработанный сервис регулирования |
| Быстродействие | 1 | 3 | 4 |
| Полнота функционала | 4 | 2 | 4 |
| Удобство использования | 2 | 5 | 4 |
| Независимость от импорта | 5 | 5 | 3 |

Критерии, проранжированные экспертами по степени важности при регулировании интервалов подъезда автобусов приведены в таблице 2.

Таблица 2 – Оценки степени важности критериев

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Критерии | Эксперты | | | | |
| 1 | 2 | 3 | 4 | 5 |
| Быстродействие | 1 | 1 | 2 | 2 | 4 |
| Полнота функционала | 3 | 2 | 3 | 3 | 2 |
| Удобство использования | 2 | 2 | 2 | 1 | 1 |
| Независимость от импорта | 4 | 4 | 4 | 4 | 3 |

Каждый эксперт должен был независимо от других присвоить ранг 1 наиболее важному свойству, ранг 2 - следующему по значимости, и так далее. При этом эксперты могут придать одинаковые ранг двум или более свойствам.

## Вычисление наиболее предпочтительного варианта регулирования

Проведём расчёт коэффициентов важности критериев выбора. Необходимо произвести нормализацию таблицы ранжировок (таблица 3), затем, рассчитав коэффициент конкордации и проверив его значимость при  = 0,05, чтобы проверить гипотезу о согласованности мнений экспертов.

Таблица 3 – нормализованная таблица ранжировок критериев

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Критерии | Эксперты | | | | | xj |  |
| 1 | 2 | 3 | 4 | 5 |
| Быстродействие | 1 | 1 | 1,5 | 2 | 4 | 9,5 | 9 |
| Полнота функционала | 3 | 2,5 | 3 | 3 | 2 | 13,5 | 1 |
| Удобство использования | 2 | 2,5 | 1,5 | 1 | 1 | 8 | 20,25 |
| Независимость от импорта | 4 | 4 | 4 | 4 | 3 | 19 | 42,25 |

Также в таблицу внесли суммы рангов по критерию (xj) и квадраты отклонения от средней суммы рангов , где m – количество экспертов, а n – количество критериев.

Выполним проверку правильности приведения таблицы ранжировок к нормальному виду, воспользовавшись формулой 3.1:

*xj ) = 0,5×m×n×(n+1)* ,(3.1)

где xj – суммарный ранг по j-му критерию;

m – число экспертов;

n – количество критериев.

Так как вычисленные значения совпадают в обеих частях уравнения, можно сделать вывод, что нормализация таблицы ранжировок выполнена верно. Теперь нужно вычислить сумму S квадратов отклонения от средней суммы рангов:

Теперь, так как в таблице ранжировок присутствуют повторяющиеся ранги, необходимо вычислить поправку коэффициента конкордации для повторяющихся рангов Ti по формуле 3.2:

*Ti = tij3 – tij )* ,(3.2)

где Ti – показатель одинаковых рангов в i-ой ранжировке;

n – количество критериев выбора;

tij – число одинаковых рангов у j-го эксперта.

*T1 = 0;*

*T2 = 6;*

*T3 = 6;*

*T4 = 0;*

*T5 = 0.*

Затем мы можем вычислить коэффициент конкордации k0 по формуле 3.3:

,(3.3)

где S – сумма квадратов отклонения от средней суммы рангов;

m – количество экспертов;

n – количество критериев;

Ti – показатель одинаковых рангов в i-ой ранжировке.

Зная значение коэффициента конкордации, можно проверить гипотезу о том, что эксперты проставляют ранги случайным образом. При случайной простановке рангов статистика имеет 𝜒2 распределение с степенями свободы, само распределение представлено на таблице 4, а статистика вычисляется по формуле 3.4.

Таблица 4 – Распределение 𝜒2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Степени свободы | Уровень значимости α | | | | | |
| 0,01 | 0,025 | 0,05 | 0,95 | 0,975 | 0,99 |
| 1 | 6,6 | 5,0 | 3,8 | 0,0039 | 0,00098 | 0,00016 |
| 2 | 9,2 | 7,4 | 6,0 | 0,103 | 0,051 | 0,020 |
| 3 | 11,3 | 9,4 | 7,8 | 0,352 | 0,216 | 0,115 |
| 4 | 13,3 | 11,1 | 9,5 | 0,711 | 0,484 | 0,297 |

, (3.4)

где – статистика;

m – количество экспертов;

n – количество критериев;

k0 – коэффициент конкордации.

Так как полученное значение статистики больше табличного, мы отвергаем гипотезу, согласованность во мнениях экспертов присутствует.

Для оценки коэффициентов важности критериев выбора воспользуемся формулой 3.5:

,(3.5)

где – коэффициент важности j-го критерия;

m – количество экспертов;

n – количество критериев;

xj – сумма рангов j-го критерия.

Сумма всех коэффициентов важности критериев равна единице, что означает, что расчёты произведены верно.

Теперь мы можем произвести расчёт коэффициентов предпочтительности вариантов выбора , применив формулу 3.6:

, (3.6)

где – коэффициент предпочтительности выбора μ-го метода;

n – количество критериев выбора;

– коэффициент важности j-го критерия выбора;

qjμ – оценка μ-го метода по j-ому критерию.

Исходя из проведённых расчётов методом экспертных оценок делаем вывод, что наиболее предпочтительным вариантом выбора является разработанный в данном исследовании сервис регулирования интервалов подъезда автобусов на остановках города.

Заключение

В рамках данной работы была проведена глубокая исследовательская работа, направленная на разработку и внедрение сервиса регулирования интервалов подъезда автобусов на остановках города. Основная цель заключалась в оптимизации расписания движения общественного транспорта с целью уменьшения скопления автобусов на остановках и повышения эффективности их работы.

В первом разделе был проведен анализ текущей организации движения автобусов. Выявлены основные принципы составления расписаний и общие требования к ним. Подчеркнута необходимость автоматизации рабочих процессов для повышения точности и надежности расписаний. Рассмотрены преимущества автоматизации, которые включают уменьшение временных затрат и улучшение качества обслуживания пассажиров. Описаны существующие методы регулирования, включая использование геолокации и централизованных систем управления. Также были сформулированы функциональные требования, требования к интерфейсу и программному обеспечению.

Во втором разделе проведено моделирование рабочих процессов "Как есть" и "Как должно быть" с использованием BPMN. Был описан пользовательский интерфейс, описан функционал приложения. Разработан контрольный пример использования сервиса с объяснением работы программы на практике.

Третий раздел содержит расчеты, в том числе метод экспертных оценок и опрос экспертов, который позволил выбрать наиболее предпочтительный вариант регулирования интервалов подъезда автобусов.

Итоги каждого раздела подтверждают, что разработанный сервис регулирования интервалов подъезда автобусов позволяет значительно улучшить организацию движения общественного транспорта, снижая временные задержки и повышая общий уровень комфорта для пассажиров. Внедрение предложенных методов и инструментов способствует более эффективному управлению городскими пассажирскими перевозками и созданию комфортных условий для всех участников дорожного движения.

Список использованных источников

1. Ефремов И. С., Кобозев В. М., Юдин В.А. Теория городских пассажирских перевозок: Учеб пособие для вузов. – М.: Высш. школа, 1980. – 535с.
2. Буйначев С. К. Основы программирования на языке Python: учебное пособие / С. К. Буйначев, Н. Ю. Боклаг. – Екатеринбург: Изд-во Урал. ун-та, 2014. – 91, [1] c.
3. Статья «Составление расписания движения автобусов» // URL: http://www.transportguide.ru/trguides-953-1.html
4. Статья «Что делать, если мой автобус не приехал?» // URL: https://www.tutu.ru/geo/journal/polezno-passazhiru/article/chto-delat-esli-moj-avtobus-ne-priekhal/
5. Никита Прияцелюк. Python GUI: создаём простое приложение с PyQt и Qt Designer [Электронный ресурс] / Режим доступа: URL: https://tproger.ru/translations/python-gui-pyqt

ПРИЛОЖЕНИЕ А

Файл Design.py:

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.resize(1117, 575)

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.verticalLayout = QtWidgets.QVBoxLayout(self.centralwidget)

self.verticalLayout.setObjectName("verticalLayout")

self.horizontalLayout = QtWidgets.QHBoxLayout()

self.horizontalLayout.setObjectName("horizontalLayout")

self.AddButton = QtWidgets.QPushButton(self.centralwidget)

self.AddButton.setObjectName("AddButton")

self.horizontalLayout.addWidget(self.AddButton)

self.ClearButton = QtWidgets.QPushButton(self.centralwidget)

self.ClearButton.setObjectName("ClearButton")

self.horizontalLayout.addWidget(self.ClearButton)

self.CollisionButton = QtWidgets.QPushButton(self.centralwidget)

self.CollisionButton.setObjectName("CollisionButton")

self.horizontalLayout.addWidget(self.CollisionButton)

spacerItem = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)

self.horizontalLayout.addItem(spacerItem)

self.verticalLayout.addLayout(self.horizontalLayout)

self.horizontalLayout\_2 = QtWidgets.QHBoxLayout()

self.horizontalLayout\_2.setObjectName("horizontalLayout\_2")

self.label = QtWidgets.QLabel(self.centralwidget)

self.label.setMaximumSize(QtCore.QSize(55, 16777215))

self.label.setObjectName("label")

self.horizontalLayout\_2.addWidget(self.label)

self.route\_combo\_box = QtWidgets.QComboBox(self.centralwidget)

self.route\_combo\_box.setMaximumSize(QtCore.QSize(100, 16777215))

self.route\_combo\_box.setObjectName("route\_combo\_box")

self.horizontalLayout\_2.addWidget(self.route\_combo\_box)

self.CollisionGraphButton = QtWidgets.QPushButton(self.centralwidget)

self.CollisionGraphButton.setMaximumSize(QtCore.QSize(100, 16777215))

self.CollisionGraphButton.setObjectName("CollisionGraphButton")

self.horizontalLayout\_2.addWidget(self.CollisionGraphButton)

self.DecreaseStopTimeButton = QtWidgets.QPushButton(self.centralwidget)

self.DecreaseStopTimeButton.setObjectName("DecreaseStopTimeButton")

self.horizontalLayout\_2.addWidget(self.DecreaseStopTimeButton)

self.IncreaseArrivalButton = QtWidgets.QPushButton(self.centralwidget)

self.IncreaseArrivalButton.setObjectName("IncreaseArrivalButton")

self.horizontalLayout\_2.addWidget(self.IncreaseArrivalButton)

spacerItem1 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)

self.horizontalLayout\_2.addItem(spacerItem1)

spacerItem2 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)

self.horizontalLayout\_2.addItem(spacerItem2)

self.ZoomInButton = QtWidgets.QPushButton(self.centralwidget)

self.ZoomInButton.setMaximumSize(QtCore.QSize(30, 16777215))

self.ZoomInButton.setObjectName("ZoomInButton")

self.horizontalLayout\_2.addWidget(self.ZoomInButton)

self.ZoomOutButton = QtWidgets.QPushButton(self.centralwidget)

self.ZoomOutButton.setMaximumSize(QtCore.QSize(30, 16777215))

self.ZoomOutButton.setObjectName("ZoomOutButton")

self.horizontalLayout\_2.addWidget(self.ZoomOutButton)

self.verticalLayout.addLayout(self.horizontalLayout\_2)

self.verticalLayout\_5 = QtWidgets.QVBoxLayout()

self.verticalLayout\_5.setObjectName("verticalLayout\_5")

self.scrollArea = QtWidgets.QScrollArea(self.centralwidget)

self.scrollArea.viewport().setProperty("cursor", QtGui.QCursor(QtCore.Qt.ArrowCursor))

self.scrollArea.setWidgetResizable(True)

self.scrollArea.setObjectName("scrollArea")

self.scrollAreaWidgetContents = QtWidgets.QWidget()

self.scrollAreaWidgetContents.setGeometry(QtCore.QRect(0, 0, 1095, 242))

self.scrollAreaWidgetContents.setObjectName("scrollAreaWidgetContents")

self.verticalLayout\_2 = QtWidgets.QVBoxLayout(self.scrollAreaWidgetContents)

self.verticalLayout\_2.setObjectName("verticalLayout\_2")

self.graphLabel = QtWidgets.QLabel(self.scrollAreaWidgetContents)

self.graphLabel.setText("")

self.graphLabel.setObjectName("graphLabel")

self.verticalLayout\_2.addWidget(self.graphLabel)

self.scrollArea.setWidget(self.scrollAreaWidgetContents)

self.verticalLayout\_5.addWidget(self.scrollArea)

self.horizontalLayout\_3 = QtWidgets.QHBoxLayout()

self.horizontalLayout\_3.setSizeConstraint(QtWidgets.QLayout.SetMinimumSize)

self.horizontalLayout\_3.setObjectName("horizontalLayout\_3")

self.tableWidget = QtWidgets.QTableWidget(self.centralwidget)

self.tableWidget.setMaximumSize(QtCore.QSize(16777215, 250))

self.tableWidget.setObjectName("tableWidget")

self.tableWidget.setColumnCount(0)

self.tableWidget.setRowCount(0)

self.horizontalLayout\_3.addWidget(self.tableWidget)

self.collidingTable = QtWidgets.QTableWidget(self.centralwidget)

self.collidingTable.setMaximumSize(QtCore.QSize(16777215, 250))

self.collidingTable.setObjectName("collidingTable")

self.collidingTable.setColumnCount(0)

self.collidingTable.setRowCount(0)

self.horizontalLayout\_3.addWidget(self.collidingTable)

self.verticalLayout\_5.addLayout(self.horizontalLayout\_3)

self.verticalLayout.addLayout(self.verticalLayout\_5)

MainWindow.setCentralWidget(self.centralwidget)

self.retranslateUi(MainWindow)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))

self.AddButton.setText(\_translate("MainWindow", "Добавить расписание"))

self.ClearButton.setText(\_translate("MainWindow", "Выгрузить расписание"))

self.CollisionButton.setText(\_translate("MainWindow", "Посчитать коллизии"))

self.label.setText(\_translate("MainWindow", " Маршрут:"))

self.CollisionGraphButton.setText(\_translate("MainWindow", "График коллизий"))

self.DecreaseStopTimeButton.setText(\_translate("MainWindow", "Уменьшить время стоянки"))

self.IncreaseArrivalButton.setText(\_translate("MainWindow", "Увеличить время прибытия"))

self.ZoomInButton.setText(\_translate("MainWindow", "🔍︎+"))

self.ZoomOutButton.setText(\_translate("MainWindow", "🔍︎-"))

ПРИЛОЖЕНИЕ Б

Файл Main.py

import os

import re

import sys

import matplotlib.dates as mdates

import matplotlib.pyplot as plt

import pandas as pd

from PyQt5.QtGui import QPixmap

from PyQt5.QtWidgets import QApplication, QFileDialog, QMessageBox, QTableWidgetItem

from PyQt5.QtWidgets import QMainWindow

from matplotlib.dates import DateFormatter

import design

class MainWindow(QMainWindow, design.Ui\_MainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.df = None

self.setupUi(self) # Это нужно для инициализации нашего дизайна

self.setWindowTitle("Просмотр расписания рейсов")

self.AddButton.clicked.connect(self.load\_schedule)

self.ClearButton.clicked.connect(self.remove\_route)

self.CollisionButton.clicked.connect(self.calculate\_collisions)

self.CollisionGraphButton.clicked.connect(self.plot\_collisions)

self.ZoomInButton.clicked.connect(self.increase\_scale)

self.ZoomOutButton.clicked.connect(self.decrease\_scale)

self.DecreaseStopTimeButton.clicked.connect(self.decrease\_stop\_time)

self.IncreaseArrivalButton.clicked.connect(self.increase\_arrival\_time)

def increase\_arrival\_time(self):

# Проверяем, что df уже есть и не пустой

if self.df is None or self.df.empty:

QMessageBox.warning(self, "Предупреждение", "Нет данных для увеличения времени прибытия.")

return

# Функция для обработки коллизий

def handle\_collisions():

nonlocal collisions\_df, excluded\_indexes

for index, row in collisions\_df.iterrows():

if index in excluded\_indexes:

continue

# Увеличиваем время прибытия на 1 минуту

updated\_arrival\_time = row['Время прибытия'] + pd.Timedelta(minutes=1)

# Проверяем, что время прибытия не стало позже времени отправления

if updated\_arrival\_time < row['Время отправления']:

if row['Стоянка (мин)'] <= 1:

# Увеличиваем время отправления на 1 минуту, если стоянка меньше или равна 1 минуте

self.df.at[index, 'Время отправления'] = row['Время отправления'] + pd.Timedelta(minutes=1)

else:

# Уменьшаем время стоянки

self.df.at[index, 'Стоянка (мин)'] -= 1

# Устанавливаем новое время прибытия

self.df.at[index, 'Время прибытия'] = updated\_arrival\_time

# Проверяем, является ли текущая запись мешающим рейсом для другой записи

if f"Маршрут {row['Маршрут']}, рейс {row['Номер рейса']}" in self.df['Мешающий рейс'].values:

interfering\_rows = self.df[

self.df['Мешающий рейс'] == f"Маршрут {row['Маршрут']}, рейс {row['Номер рейса']}"]

for interfering\_row in interfering\_rows.itertuples():

excluded\_indexes.append(interfering\_row.Index)

# Удаляем текущую запись из дальнейшего регулирования

excluded\_indexes.append(index)

excluded\_indexes = []

# Сначала обрабатываем коллизии уровня 1

collisions\_df = self.df[self.df['Коллизии'] == 1]

handle\_collisions()

# Проверяем, остались ли коллизии уровня 1

# if not self.df[self.df['Коллизии'] == 1].empty:

collisions\_df = self.df[self.df['Коллизии'] == 1]

handle\_collisions()

# else:

# Если коллизий уровня 1 не осталось, обрабатываем коллизии более высокого уровня

for level in range(2, self.df['Коллизии'].max() + 1):

collisions\_df = self.df[(self.df['Коллизии'] == level) & (~self.df.index.isin(excluded\_indexes))]

handle\_collisions()

# Перерисовываем график

self.plot\_schedule(self.df)

def decrease\_stop\_time(self):

# Проверяем, что df уже есть и не пустой

if self.df is None or self.df.empty:

QMessageBox.warning(self, "Предупреждение", "Нет данных для уменьшения времени стоянки.")

return

# Функция для обработки коллизий

def handle\_collisions():

nonlocal collisions\_df, excluded\_indexes

for index, row in collisions\_df.iterrows():

if index in excluded\_indexes:

continue

interfering\_routes = row['Мешающий рейс']

# Используем регулярное выражение для извлечения маршрута и рейса

match = re.search(r"Маршрут (\w+), рейс (\d+)", interfering\_routes)

if match:

interfering\_route = match.group(1)

interfering\_trip = match.group(2)

else:

QMessageBox.warning(self, "Предупреждение", "Не удалось извлечь маршрут и рейс.")

continue

# Фильтруем DataFrame для записи с мешающим рейсом

interfering\_row = self.df[(self.df['Маршрут'] == interfering\_route) &

(self.df['Номер рейса'] == interfering\_trip) &

(self.df['Остановка'] == row['Остановка'])]

# Проверяем, что такая запись найдена

if not interfering\_row.empty:

interfering\_index = interfering\_row.index[0]

# Уменьшаем время отправления на 1 минуту

updated\_departure\_time = interfering\_row.iloc[0]['Время отправления'] - pd.Timedelta(minutes=1)

# Проверяем, что время отправления не стало раньше времени прибытия

if updated\_departure\_time > interfering\_row.iloc[0]['Время прибытия']:

if self.df.at[interfering\_index, 'Стоянка (мин)'] <= 1:

# Уменьшаем время прибытия на 1 минуту, если стоянка меньше или равна 1 минуте

self.df.at[interfering\_index, 'Время прибытия'] = interfering\_row.iloc[0][

'Время прибытия'] - pd.Timedelta(

minutes=1)

else:

# Уменьшаем время стоянки

self.df.at[interfering\_index, 'Стоянка (мин)'] -= 1

# Устанавливаем новое время отправления

self.df.at[interfering\_index, 'Время отправления'] = updated\_departure\_time

# Удаляем текущую запись из дальнейшего регулирования

self.df.at[index, 'Коллизии'] = 0

self.df.at[index, 'Мешающий рейс'] = ''

excluded\_indexes.append(index)

# Удаляем текущую запись из дальнейшего регулирования

excluded\_indexes.append(index)

excluded\_indexes = []

# Сначала обрабатываем коллизии уровня 1

collisions\_df = self.df[self.df['Коллизии'] == 1]

handle\_collisions()

# Проверяем, остались ли коллизии уровня 1

collisions\_df = self.df[self.df['Коллизии'] == 1]

handle\_collisions()

# Если коллизий уровня 1 не осталось, обрабатываем коллизии более высокого уровня

for level in range(2, self.df['Коллизии'].max() + 1):

collisions\_df = self.df[(self.df['Коллизии'] == level) & (~self.df.index.isin(excluded\_indexes))]

handle\_collisions()

# Перерисовываем график

self.plot\_schedule(self.df)

def calculate\_collisions(self):

# Проверяем, что df уже есть и не пустой

if self.df is None or self.df.empty:

QMessageBox.warning(self, "Предупреждение", "Нет данных для подсчета коллизий.")

return

# Создаем столбец "Коллизии" и заполняем его нулями

self.df['Коллизии'] = 0

# Перебираем каждую строку DataFrame для подсчета коллизий и нахождения мешающих рейсов

for index, row in self.df.iterrows():

current\_time = row['Время прибытия']

current\_stop = row['Остановка']

# Фильтруем DataFrame для поиска коллизий

colliding\_buses = self.df[(self.df['Время прибытия'] <= current\_time) &

(self.df['Время отправления'] > current\_time) &

(self.df['Остановка'] == current\_stop) &

(self.df.index != index)]

# Подсчитываем коллизии

current\_route\_bus\_count = len(colliding\_buses)

# Обновляем значение столбца "Коллизии" для текущей строки

self.df.at[index, 'Коллизии'] = current\_route\_bus\_count

# Если есть коллизии, находим мешающий рейс

if current\_route\_bus\_count > 0:

colliding\_routes = ', '.join(

f"Маршрут {r}, рейс {n}" for r, n in

zip(colliding\_buses['Маршрут'], colliding\_buses['Номер рейса']))

# Обновляем значение столбца "Мешающий рейс" для текущей строки

self.df.at[index, 'Мешающий рейс'] = colliding\_routes

# Теперь ваш DataFrame содержит столбцы "Коллизии" и "Мешающий рейс"

self.show\_collisions()

def show\_collisions(self):

# Очищаем таблицу перед заполнением новыми данными

self.tableWidget.clear()

# Фильтруем DataFrame по коллизиям

collisions\_df = self.df[self.df['Коллизии'] > 0]

# Преобразование времени в формат "HH:MM"

if 'Время отправления' in collisions\_df.columns:

collisions\_df['Время отправления'] = pd.to\_datetime(collisions\_df['Время отправления']).dt.strftime('%H:%M')

if 'Время прибытия' in collisions\_df.columns:

collisions\_df['Время прибытия'] = pd.to\_datetime(collisions\_df['Время прибытия']).dt.strftime('%H:%M')

# Устанавливаем количество строк и столбцов в таблице

self.tableWidget.setRowCount(len(collisions\_df))

self.tableWidget.setColumnCount(len(collisions\_df.columns))

# Задаем заголовки столбцов

headers = collisions\_df.columns.tolist()

self.tableWidget.setHorizontalHeaderLabels(headers)

# Заполняем таблицу данными из DataFrame

for i, row in enumerate(collisions\_df.itertuples(index=False), start=0):

for j, value in enumerate(row, start=0):

item = QTableWidgetItem(str(value))

self.tableWidget.setItem(i, j, item)

self.tableWidget.resizeColumnsToContents()

self.show\_colliding\_routes()

def show\_colliding\_routes(self):

# Очищаем таблицу перед заполнением новыми данными

self.collidingTable.clear()

colliding\_routes\_df = self.df.copy()

# Преобразование времени в формат "HH:MM"

if 'Время отправления' in colliding\_routes\_df.columns:

colliding\_routes\_df['Время отправления'] = pd.to\_datetime(

colliding\_routes\_df['Время отправления']).dt.strftime('%H:%M')

if 'Время прибытия' in colliding\_routes\_df.columns:

colliding\_routes\_df['Время прибытия'] = pd.to\_datetime(colliding\_routes\_df['Время прибытия']).dt.strftime(

'%H:%M')

# Устанавливаем количество строк и столбцов в таблице

num\_rows, num\_cols = colliding\_routes\_df.shape

self.collidingTable.setRowCount(num\_rows)

self.collidingTable.setColumnCount(num\_cols)

# Заполняем таблицу данными из DataFrame

for i, row in enumerate(colliding\_routes\_df.itertuples(index=False), start=0):

for j, value in enumerate(row, start=0):

item = QTableWidgetItem(str(value))

self.collidingTable.setItem(i, j, item)

# Дополнительно, если необходимо, можно установить заголовки столбцов

headers = colliding\_routes\_df.columns.tolist()

self.collidingTable.setHorizontalHeaderLabels(headers)

# И установить размеры столбцов по содержимому

self.collidingTable.resizeColumnsToContents()

def load\_schedule(self):

file\_path, \_ = QFileDialog.getOpenFileName(self, "Выберите файл Excel", "", "Excel Files (\*.xlsx \*.xls)")

if file\_path:

try:

new\_df = pd.read\_excel(file\_path)

if self.df is None:

self.df = new\_df

else:

self.df = pd.concat([self.df, new\_df], ignore\_index=True)

self.plot\_schedule(self.df)

self.fill\_route\_combo\_box()

except Exception as e:

QMessageBox.critical(self, "Ошибка", f"Произошла ошибка при загрузке файла: {e}")

# Создаем столбец "Коллизии" и заполняем его нулями

self.df['Коллизии'] = 0

self.df['Маршрут'] = self.df['Маршрут'].astype(str)

self.df['Номер рейса'] = self.df['Номер рейса'].astype(str)

# # Преобразуем значения времени в строки только времени

# self.df['Время прибытия'] = self.df['Время прибытия'].apply(lambda x: x.strftime('%H:%M'))

# self.df['Время отправления'] = self.df['Время отправления'].apply(lambda x: x.strftime('%H:%M'))

def plot\_schedule(self, df):

if df.empty:

QMessageBox.warning(self, "Предупреждение", "Нет данных для отображения графика.")

self.graphLabel.clear()

return

plt.figure(figsize=(30, 9))

unique\_routes = df['Маршрут'].unique()

# Указываем формат времени

time\_format = "%H:%M"

# Преобразуем время прибытия и отправления в datetime

df['Время прибытия'] = pd.to\_datetime(df['Время прибытия'], format=time\_format)

df['Время отправления'] = pd.to\_datetime(df['Время отправления'], format=time\_format)

# Устанавливаем общие временные границы для всех рейсов и маршрутов

all\_times = pd.to\_datetime(df[['Время прибытия', 'Время отправления']].stack().unique())

min\_time = all\_times.min()

max\_time = all\_times.max()

legend\_handles = []

for route in unique\_routes:

route\_data = df[df['Маршрут'] == route]

for trip in route\_data['Номер рейса'].unique():

trip\_data = route\_data[route\_data['Номер рейса'] == trip]

times = []

stops = []

for \_, row in trip\_data.iterrows():

arrival\_time = row['Время прибытия']

departure\_time = row['Время отправления']

stop = row['Остановка']

times.append(arrival\_time)

stops.append(stop)

times.append(departure\_time)

stops.append(stop)

line, = plt.plot(times, stops, label=f'Маршрут {route} Рейс {trip}')

legend\_handles.append(line)

plt.xticks(rotation=45)

plt.xlabel('Время')

plt.ylabel('Остановка')

plt.title('Расписание рейсов')

# Устанавливаем временные границы

plt.xlim([min\_time, max\_time])

# Устанавливаем интервал меток на оси X каждые 5 минут

plt.gca().xaxis.set\_major\_locator(mdates.MinuteLocator(interval=5))

plt.gca().xaxis.set\_major\_formatter(DateFormatter('%H:%M'))

if legend\_handles:

plt.legend(handles=legend\_handles, loc='upper left', bbox\_to\_anchor=(1, 1))

plt.grid(True)

plt.tight\_layout()

self.calculate\_collisions()

# Вызов новой функции для добавления маркеров коллизий

self.plot\_collisions\_markers()

# Сохраняем график во временный файл

temp\_file = 'temp\_plot.png'

plt.savefig(temp\_file)

plt.close()

# Отображаем график в QLabel

pixmap = QPixmap(temp\_file)

self.graphLabel.setPixmap(pixmap)

self.graphLabel.setScaledContents(True)

def plot\_collisions\_markers(self):

if self.df is None or 'Коллизии' not in self.df.columns:

return

collisions\_df = self.df[self.df['Коллизии'] > 0]

if collisions\_df.empty:

return

for \_, row in collisions\_df.iterrows():

plt.plot(row['Время прибытия'], row['Остановка'], 'ro') # Красные маркеры для прибытия

# plt.plot(row['Время отправления'], row['Остановка'], 'ro') # Красные маркеры для отправления

def plot\_collisions(self):

# Проверяем наличие DataFrame

if self.df is None:

QMessageBox.warning(self, "Предупреждение", "Данные не загружены.")

return

# Проверяем наличие столбца "Коллизии"

if 'Коллизии' not in self.df.columns:

QMessageBox.warning(self, "Предупреждение", "Отсутствует столбец 'Коллизии' в данных.")

return

# Фильтруем DataFrame для получения рейсов с коллизиями

collisions\_df = self.df[self.df['Коллизии'] > 0]

if collisions\_df.empty:

QMessageBox.warning(self, "Предупреждение", "Нет данных о коллизиях.")

self.graphLabel.clear()

return

# Используем существующую функцию для построения графика

self.plot\_schedule(collisions\_df)

def remove\_route(self):

if self.df is None:

QMessageBox.warning(self, "Предупреждение", "Сначала загрузите файл расписания.")

return

# Убедитесь, что колонка 'Маршрут' приведена к строковому типу данных

self.df['Маршрут'] = self.df['Маршрут'].astype(str).str.strip()

# Получаем выбранный маршрут из комбобокса и удаляем пробелы

selected\_route = self.route\_combo\_box.currentText().strip()

route\_df = self.df[self.df['Маршрут'] == selected\_route]

# Изменение формата времени в DataFrame

route\_df['Время отправления'] = route\_df['Время отправления'].dt.strftime('%H:%M')

route\_df['Время прибытия'] = route\_df['Время прибытия'].dt.strftime('%H:%M')

# Удаление столбца 'Коллизии'

route\_df = route\_df.drop(columns=['Коллизии'])

# Создание папки для сохранения файлов маршрута

directory = f"Маршрут\_{selected\_route}"

if not os.path.exists(directory):

os.makedirs(directory)

# Сохранение измененного DataFrame в общий файл Excel для всего маршрута

filename = os.path.join(directory, f"маршрут {selected\_route} отрегулированный.xlsx")

route\_df.to\_excel(filename, index=False)

# Сохранение данных каждого рейса в отдельный файл

for trip in route\_df['Номер рейса'].unique():

trip\_data = route\_df[route\_df['Номер рейса'] == trip]

start\_time = trip\_data['Время отправления'].min().replace(":", ".")

end\_time = trip\_data['Время прибытия'].max().replace(":", ".")

driver\_name = trip\_data['Водитель'].iloc[0] # Предполагается, что у рейса один водитель

trip\_filename = os.path.join(directory,

f"маршрут {selected\_route} рейс {trip}, {start\_time}-{end\_time}, {driver\_name}.xlsx")

trip\_data.to\_excel(trip\_filename, index=False)

# Обновление DataFrame без выбранного маршрута

self.df = self.df[self.df['Маршрут'] != selected\_route]

self.plot\_schedule(self.df)

self.fill\_route\_combo\_box()

QMessageBox.information(self, "Информация",

f"Маршрут {selected\_route} сохранен в папку {directory} и файлы для каждого рейса.")

def fill\_route\_combo\_box(self):

unique\_routes = self.df['Маршрут'].unique().astype(str)

self.route\_combo\_box.clear()

self.route\_combo\_box.addItems(unique\_routes)

def increase\_scale(self):

current\_pixmap = self.graphLabel.pixmap()

if current\_pixmap is not None:

# Получаем текущее изображение

image = current\_pixmap.toImage()

# Уменьшаем масштаб изображения на 20%

new\_width = int(image.width() \* 1.2)

new\_height = int(image.height() \* 1.2)

# Преобразуем изображение в новый размер

scaled\_image = image.scaled(new\_width, new\_height)

# Создаем новый pixmap и устанавливаем его в QLabel

new\_pixmap = QPixmap.fromImage(scaled\_image)

self.graphLabel.setPixmap(new\_pixmap)

self.graphLabel.setScaledContents(True)

def decrease\_scale(self):

current\_pixmap = self.graphLabel.pixmap()

if current\_pixmap is not None:

# Получаем текущее изображение

image = current\_pixmap.toImage()

# Уменьшаем масштаб изображения на 20%

new\_width = int(image.width() \* 0.8)

new\_height = int(image.height() \* 0.8)

# Преобразуем изображение в новый размер

scaled\_image = image.scaled(new\_width, new\_height)

# Создаем новый pixmap и устанавливаем его в QLabel

new\_pixmap = QPixmap.fromImage(scaled\_image)

self.graphLabel.setPixmap(new\_pixmap)

self.graphLabel.setScaledContents(True)

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec\_())