

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

Вычислительная техника

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

Профиль «Вычислительные машины, комплексы, системы
и сети»

БАКАЛАВРСКАЯ РАБОТА

на тему

**Бизнес-проект «ПРОГРАММА МОДЕЛИРОВАНИЯ
ИНТЕЛЛЕКТУАЛЬНОГО УПРАВЛЕНИЯ
ДОРОЖНЫМ ТРАФИКОМ.
ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС»**

Студент Лобанов Р.И. 30.05.2025
(подпись, дата)

Руководитель Митрохин М.А. 30.05.25
(подпись, дата)

Нормоконтролёр Бычков А.С. 30.05.25
(подпись, дата)

Работа допущена к защите (протокол заседания кафедры от

28.05.2025 г. № 16)

Заведующий кафедрой Митрохин М.А.
(подпись)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 01.06.25 № 4)

Секретарь ГЭК Гурина Е.И.
(подпись)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«06» 05 2025 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Лобанов Рамиль Ильясович гр. 21BVB1 факультета вычислительной техники.
2. Тема работы: Бизнес-проект «Программа моделирования интеллектуального управления дорожным трафиком. Пользовательский интерфейс».

Тема утверждена приказом ПГУ № 601/0 от "06" 05 2025 г.

3. Руководитель бакалаврской работы д.т.н., профессор, зав. каф. ВТ ПГУ Митрохин М.А.
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Необходимо создать программу моделирования интеллектуального управления дорожным трафиком, которая предназначена для анализа и оптимизации работы светофорных объектов. Программа должна быть создана с учётом пользовательского интерфейса, при помощи которого пользователь сможет анализировать результаты моделирования и самостоятельно задавать характеристики модели.

Программа должна получать входные данные о дорожном трафике либо генерировать их на основании настроек пользователя, а также отображать результаты моделирования при помощи графиков зависимостей характеристик дорожного движения.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ следующих предметных областей: изучить существующие средства разработки графических приложений на языке Python, изучить способы генерации и загрузки данных для моделирования, изучить интерфейсы выбранной среды имитационного моделирования, изучить способы предоставления результатов моделирования. Также необходимо изучить функционал библиотек Matplotlib, Traci и sumolib. Необходимо рассмотреть вопросы создания архитектуры программы, обеспечивающей взаимодействие между интерфейсом пользователя и остальными модулями, а также изучить вопросы генерации данных, подсчёта метрик и визуализации результатов.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	52	06.05.2025-09.05.2025	
2.	Анализ требований	13	10.05.2025	
3.	Выбор технологий, применяемых при проектировании и разработке	52	11.05.2025-14.05.2025	
4.	Описание алгоритмов	39	15.05.2025-17.05.2025	
5.	Разработка пользовательского интерфейса для программы моделирования интеллектуального управления дорожным трафиком	104	18.05.2025 - 25.05.2025	
6.	Оформление пояснительной записки	64	26.05.2025 - 30.05.2025	

Дата выдачи задания " 06 " 05 2025 г.

Руководитель бакалаврской работы  Митрохин М.А.
(подпись, дата)

Задание к исполнению принял студент  06.05.2025 Лобанов Р.И.
(подпись, дата)

Работу к защите допустить
Декан факультета  27.05.2025 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 109 листов, 32 рисунка, 2 таблицы, 22 источника и 3 приложения.

ИММИТАЦИОННОЕ МОДЕЛИРОВАНИЕ ТРАНСПОРТНЫХ ПОТОКОВ, ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС, ВИЗУАЛИЗАЦИЯ ДАННЫХ, ВЗАИМОДЕЙСТВИЕ С ПОЛЬЗОВАТЕЛЕМ, ГЕНЕРАЦИЯ ДАННЫХ, XML

Целью работы является разработка пользовательского интерфейса для настройки и управления моделированием, который включает визуализацию данных и выбор режимов моделирования.

Объектом исследования в работе являются интерфейс пользователя и способы взаимодействия с системой управления дорожным трафиком.

В работе применены подходы к созданию пользовательского интерфейса, методы визуализации данных (графики зависимости характеристик дорожного движения), инструменты настройки параметров и механизмы считывания характеристик дорожного трафика. Реализация выполнена с использованием Python-интерфейса для интеграции с средой моделирования SUMO.

В работе рассмотрены существующие фреймворки и библиотеки для создания пользовательского интерфейса, механизмы визуализации характеристик транспортного потока, а также протестированы методы генерации и форматирования данных для моделирования.

					ПГУ 1.090301.16.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Лобанов Р.И.		30.05.25	Бизнес-проект "Программа моделирования интеллектуального управления дорожным трафиком. Пользовательский интерфейс" Пояснительная записка	Лит.	Лист	Листов
Провер.		Митрохин М.А.		30.05.25			2	109
Ил. Контр		Бычков А.С.		30.05.25		ФВТ, каф. ВТ		
Утверд.		Митрохин М.А.		30.05.25		гр. 21ВВВ1		

Содержание

Перечень сокращений и обозначений	5
Введение.....	6
1 Программа моделирования интеллектуального управления	
дорожным движением	7
1.1 Постановка задачи	7
1.2 Анализ предметной области.....	7
1.2.1 Обзор аналогов.....	9
1.2.2 Выбор инструментов разработки	11
1.2.3 Выбор среды разработки	15
1.2.4 Выбор архитектуры программы.....	16
1.3 Проектирование и разработка	19
1.3.1 Механика моделирования в среде SUMO.....	19
1.3.2 Использование Python API (sumolib, tracI)	20
1.3.3 Формат данных и интеграция с реальными источниками	21
1.3.4 Разработка пользовательского интерфейса	21
1.3.5 Разработка модуля загрузки и генерации данных.....	32
1.3.6 Разработка модуля анализа и визуализации	35
1.4 Тестирование и анализ результатов	39
1.4.1 Тестирование и анализ интерфейса приложения	39
1.4.2 Тестирование алгоритма генерации данных.....	46
2 Охрана труда	51
2.1 Характеристика потенциальных рисков при работе с ПК	51
2.2 Оборудование рабочего места.....	51
2.3 Освещённость рабочего места.....	52
2.4 Электробезопасность	53
2.5 Пожарная безопасность	53
2.6 Санитарно-гигиеническая характеристика производства	54

					ПГУ 1.090301.16.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Лобанов Р.И.			Программа моделирования интеллектуального управления дорожным трафиком. Интерфейс пользователя Пояснительная записка	Лит.	Лист
Провер.		Митрохин М.А.					
Ф. Контр		Бычков А.С.				3	1
.						ФВТ, каф. ВТ <i>гр. 21BBB1</i>	
Утверд.		Митрохин М.А.					

3 Стартап проект	55
3.1 Анализ рынка	55
3.2 Целевая аудитория и каналы взаимодействия	56
3.3 Источники доходов и расходов.....	56
3.4 Риски проекта.....	57
3.5 Ключевые ресурсы.....	58
Заключение.....	59
Список использованных источников.....	60
Приложение А - UML – диаграммы	63
Приложение Б - Листинг программы	69
Приложение В - Презентация	102

Перечень сокращений и обозначений

ПЭВМ – персональная электронно-вычислительная машина

API (Application Programming Interface) – программный интерфейс приложения

SUMO (Simulation of Urban Mobility) – моделирование городской мобильности

TraCI (Traffic Control Interface) – интерфейс управления дорожным движением

XML (eXtensible Markup Language) – расширяемый язык разметки

					ПГУ 1.090301.16.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

Светофоры – важный элемент городского управления дорожным движением. Благодаря их использованию на дорогах организуется безопасность движения. Но со временем количество автомобилей растёт, что приводит к ещё большим заторам. Из-за этого используемые статичные алгоритмы управления светофорными объектами становятся всё более неэффективными, что ведёт к переполнению дорог и долгому ожиданию проезда перекрёстка. Решением этой проблемы может стать использование интеллектуальных алгоритмов управления светофорами, которые используют машинное обучение и нейронные сети.

Эти алгоритмы представляют собой систему, которая может адаптироваться к текущим условиям дорожного движения. Они позволяют светофорам реагировать на изменение дорожной ситуации и применять соответствующие решения для оптимизации движения. В результате их применения может достигаться более эффективное управление дорожным движением, что будет приводить к снижению заторов и других отрицательных эффектов.

Анализ эффективности интеллектуальных алгоритмов имеет важное значение, так как их использование может способствовать улучшению эффективности дорожного движения в целом, снижению вредных выбросов в атмосферу и др. Вместе с этим, можно предположить, что результаты управления дорожным трафиком в среде моделирования могут сильно отличаться от результатов реального использования в городской среде.

1 Программа моделирования интеллектуального управления дорожным движением

1.1 Постановка задачи

Задача состоит в разработке программы для исследования интеллектуальных алгоритмов управления фазами работы светофорных объектов и графического пользовательского интерфейса для неё.

Программа должна отвечать следующим требованиям:

- программа должна иметь графический пользовательский интерфейс;
- данные о трафике для моделирования должны генерироваться автоматически на основании настроек пользователя либо загружаться из готовых источников;
- блок визуализации моделирования должен показывать зависимости различных характеристик дорожного движения в виде графиков;
- программа должна работать в ОС Microsoft Windows 10 и выше.

Ожидаемые результаты:

- программа, позволяющая исследовать интеллектуальные алгоритмы в среде имитационного моделирования;
- доказательство эффективности Q-обучения и нейронных сетей по сравнению с классическими методами.

Технические требования:

- процессор: не менее 2 ядер;
- оперативная память: не менее 8 ГБ;
- дисковое пространство: не менее 8 ГБ;
- видеокарта: интегрированная (если не требуется использование тензорных ядер).

1.2 Анализ предметной области

Исследование интеллектуальных алгоритмов управления светофорными объектами неэффективно без применения современных инструментов

					ПГУ 1.090301.16.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		7

анализа и моделирования дорожного движения. Одним из наиболее перспективных направлений является имитационное моделирование, которое позволяет определять эффективность исследуемых алгоритмов, до их апробации на реальных перекрёстках.

Имитационное моделирование широко применяется для определения наилучшей схемы управления транспортными потоками в статье [1].

Отечественные [2] и зарубежные [3] исследования показали, что имитационное моделирование имеет потенциал в анализе работы транспортных систем.

Исследования [4] продемонстрировали эффективность различных алгоритмов, используемых при имитационном моделировании. Как упоминалось в статье [5], имитационные модели транспортных потоков являются перспективным инструментом для анализа и оценки систем управления городским дорожным движением.

Т. А. Кузнецов [6] в своей работе по проблемам моделирования дорожного движения рассмотрел задачу регулирования продолжительности зеленого света светофоров в зоне перекрестка. Авторы использовали методы имитационного моделирования для выбора оптимальных параметров работы светофоров, что позволило сократить время ожидания транспортных средств и уменьшить заторы на перекрестках.

В рамках изучения современных инструментов имитационного моделирования транспортных потоков авторы Захарова Ю. И. и Карнауха Е. С. [7] подтвердили значительное упрощение процесса изучения и контроля динамики транспортного потока средствами имитационного моделирования.

Наконец, в работах [8...10] доказана эффективность применения интеллектуальных алгоритмов управления светофорными объектами.

Их использование приводит к снижению временных задержек и транспортных заторов, а также к общему улучшению характеристик транспортного потока.

Особенно стоит отметить программное обеспечение [11], которое функционирует в нашем городе.

1.2.1 Обзор аналогов

В России существует несколько программных продуктов, целью которых является использование систем интеллектуального управления светофорными объектами.

Программное обеспечение AVEDEX [12], разработанное компанией MALLENOM TECHNOLOGIES — это система анализа транспортных и пешеходных потоков, использующая технологии машинного зрения и нейронные сети. К преимуществам данного ПО относятся: точная видеоаналитика, основанная на применении нейронных сетей; настройка учёта типов участников дорожного движения; работа по расписанию; интеграция с другими системами. К недостаткам можно отнести: невозможность на самообучения модели в процессе работы; централизованное управление данными с перекрёстков.

Научно-производственное объединение НПО «РБС», разрабатывает систему управления дорожным движением Авангард [13], которая представляет собой решение, направленное на мониторинг, анализ и оптимизацию транспортных потоков. К достоинствам можно отнести: сбор и анализ данных с разных источников (камеры, индукционные датчики); анализ аварийности на дорогах; создание виртуальных моделей дорожной сети в реальном времени.

Недостатки: обязательное наличие нескольких типов датчиков для работы системы; не используются методы машинного обучения и нейронные сети.

Компания Complitech разрабатывает систему Sentinel Smart Traffic [14], представляющую собой программное обеспечение для автоматического сбора и анализа данных о транспортных потоках. Достоинствами системы является: автоматический мониторинг нарушений ПДД; оперативная реакция на ДТП; система распознавания номерных знаков автомобилей. К недостатку можно отнести: высокие финансовые затраты на интеграцию системы.

Компания СпецДорПроект занимается разработкой интеллектуальной транспортной системы “Умный перекрёсток” [15] в Москве и других городах.

Она использует индуктивные детекторы для адаптивного управления светофорами, что сокращает время ожидания и повышает пропускную способность дорог. Достоинствами являются: осуществление приоритета движения для общественного транспорта; интеграция с общей ИТС города; Главным недостатком является высокая цена для использования системы.

В таблице 1 представлена сводная информация о достоинствах и недостатках каждого программного продукта.

Таблица 1 – Сводная таблица достоинств и недостатков аналогов

Программное обеспечение	Разработчик	Достоинство	Недостаток
AVEDEX	MALLENOM TECHNOLOGIES	Высокоточная видеоаналитика на основе нейросетей; Учёт нескольких типов участников движения; Работа по расписанию; Интеграция с другими системами.	Невозможность самообучения модели; Централизованное управление данными с перекрёстков.
Авангард	НПО «РБС»	Сбор данных с разных источников (камеры, датчики); Анализ аварийности; Виртуальные модели дорожной сети в реальном времени.	Требует несколько типов датчиков; Отсутствие машинного обучения).
Sentinel Smart Traffic	Complitech	Автоматический мониторинг нарушений ПДД; Оперативная реакция на ДТП;	Невозможность самообучения модели; Высокая стоимость внедрения.
Умный перекрёсток	СпецДорПроект	Приоритет для общественного транспорта; Интеграция с городской ИТС; Снижение времени ожидания на перекрёстках;	Высокая цена внедрения

1.2.2 Выбор инструментов разработки

Приложение предназначено для исследования работы интеллектуальных алгоритмов управления светофорными объектами на перекрёстке при помощи имитационного моделирования.

На сегодняшний день существует ряд популярных библиотек и фреймворков для создания графического интерфейса на Python с необходимым функционалом для создания нашей программы.

Tkinter [16] представляет собой стандартную библиотека Python для создания GUI, основанная на Tk. К достоинствам можно отнести тот факт, что она входит в стандартный набор инструментов Python; обеспечивает кроссплатформенную поддержку интерфейса; подходит для начинающих благодаря простой и понятной структуре. Недостатки: интерфейс выглядит устаревшим; библиотека предоставляет ограниченное количество виджетов, по сравнению с другими библиотеками. На рисунке 1 представлен пример интерфейса приложения на Tkinter.

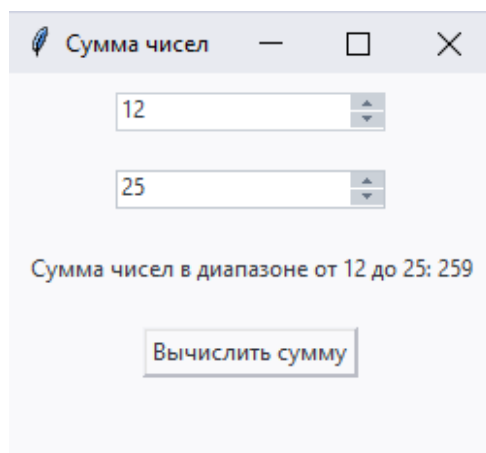


Рисунок 1 – Пример приложения на Tkinter

Фреймворк PyQt [17] предназначен для создания графических интерфейсов на языке Python, представляющий собой обёртку над фреймворком Qt. Он позволяет разрабатывать современные приложения для различных операционных систем, включая Windows, macOS и Linux. Так же он предлагает широкие возможности для создания GUI-приложений. Преимущества: предоставляет более 600 классов и 6000 функций для работы с интерфейсами;

поддерживает стили для визуального программирования интерфейсов; работает на всех платформах.

Недостатки: для коммерческого использования необходимо приобретение лицензии; может быть сложен в освоении из-за большого функционала.

На рисунке 2 представлен пример интерфейса приложения на PyQt.

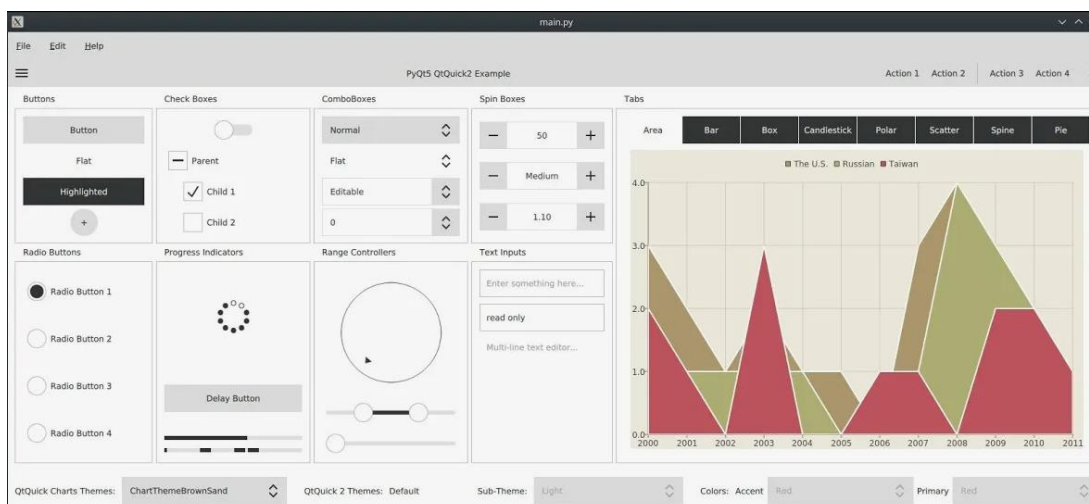


Рисунок 2 – Пример приложения на PyQt

Кроссплатформенный фреймворк wxPython [18] для создания графических интерфейсов на языке Python, является изменённой под Python библиотекой wxWidgets, написанной на C++. Он позволяет разрабатывать приложения с нативным внешним видом на Windows, macOS и Linux. Также он предоставляет виджеты для интерфейса, включая кнопки, текстовые поля, списки и др. К достоинствам можно отнести: богатый набор виджетов для интерфейса; кроссплатформенность. Недостатками являются: меньшая популярность по сравнению с другими библиотеками для GUI; ограниченные инструменты для дизайна приложений; лицензия, не позволяющая сохранять коммерческую тайну продукта. На рисунке 3 представлен пример приложения на wxPython.

Кроссплатформенный фреймворк BeeWare Toga [19] предназначен для создания графических интерфейсов на Python. Он позволяет разрабатывать нативные приложения для различных операционных систем, включая Windows, macOS, Linux, Android и iOS. Достоинства: использование нативных виджетов; наличие кроссплатформенности.

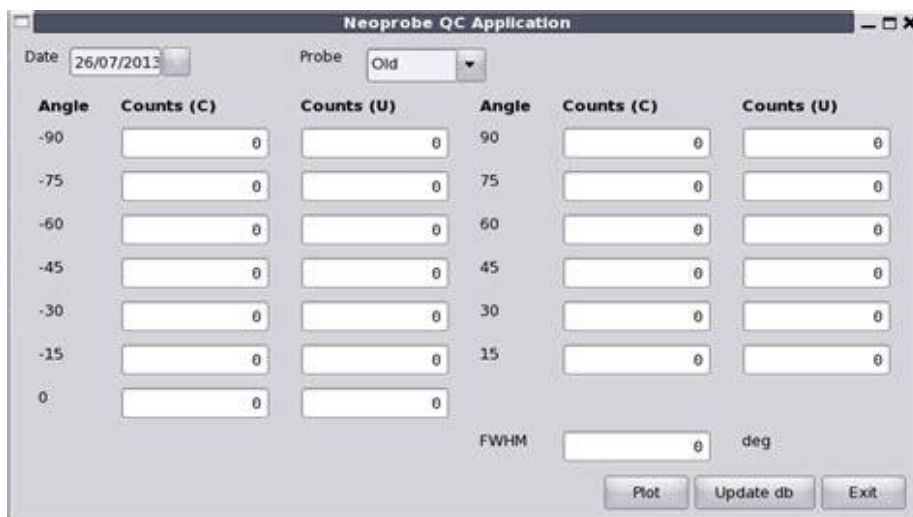


Рисунок 3 – Пример приложения на wxPython

Недостаток: ограниченный набор виджетов по сравнению с другими фреймворками.

Открытый фреймворк для создания кроссплатформенных приложений Kivy [20] позволяет разрабатывать приложения для Windows, macOS, Linux, Android и iOS с использованием одного кода. К преимуществам можно отнести: лицензия MIT; современный интерфейс; кроссплатформенность, хорошая документация. Недостатками являются: отсутствие нативного вида интерфейса; проблемы с поддержкой старых версий Python. На рисунке 4 представлен пример приложения на Kivy.



Рисунок 4 – Пример приложения на Kivy

Ниже показана сводная таблица достоинств и недостатков сред имитационного моделирования транспортных потоков.

Таблица 2 – Сводная таблица достоинств и недостатков средств разработки GUI

Программное обеспечение	Достоинство	Недостаток
Tkinter	Встроен в стандартную библиотеку Python; Кроссплатформенность (Windows, macOS, Linux); Простота освоения для начинающих.	Устаревший внешний вид интерфейсов; Ограниченный набор виджетов; Мало возможностей для кастомизации дизайна.
PyQt	Широкий функционал (600+ классов, 6000+ функций); Современный интерфейс; Кроссплатформенность.	Требует покупки лицензии для коммерческого использования; Сложный API для новичков.
wxPython	Нативный внешний вид для каждой ОС; Богатая коллекция виджетов; Кроссплатформенность;	Менее популярен среди разработчиков; Ограниченные возможности дизайна.
BeeWare Toga	Нативные элементы интерфейса; Поддержка мобильных платформ.	Ограниченный набор виджетов; Мало примеров и учебных материалов.
Kivy	Современный интерфейс; Кроссплатформенность; Лицензия MIT; Хорошая документация.	Требует последние версии Python; Не использует нативные элементы интерфейса.

Таким образом, при выборе инструмента для разработки графического интерфейса было отдано предпочтение Kivy по следующим причинам.

Во-первых, Kivy имеет разрешительную лицензию MIT, которая не запрещает коммерческое использование продукта.

Во-вторых, Kivy предоставляет инструменты для создания разнообразных интерфейсов. Благодаря широкому набору виджетов, мы можем создавать интерфейсы, которые необходимы для понятного интерфейса.

Также, Kivy имеет сообщество и хорошую документацию.

Основой разработки стал язык программирования Python, который подошел для данной задачи благодаря своей простоте, большому выбору библиотек и многофункциональному API с используемой средой имитационного моделирования.

Также Python предлагает большое количество фреймворков и библиотек, позволяющих создавать графические интерфейсы пользователя для различных приложений.

В качестве библиотеки для построения графиков была выбрана matplotlib. Она поддерживает линейные диаграммы, гистограммы и другие типы графиков. Её основное преимущество заключается в возможности настройки каждого элемента графика (оси, легенды, стили).

Выбранные инструменты разработки соответствует требованиям проекта, обеспечивая необходимый функционал для создания пользовательского интерфейса программы для исследования различных интеллектуальных алгоритмов управления светофорными объектами.

1.2.3 Выбор среды разработки

Современные среды разработки предоставляют встроенные инструменты для проверки и отладки программ, управления библиотеками, интеграции с системами контроля версий и других функций, которые упрощают процесс разработки.

Наиболее популярные из них: Visual Studio Code, PyCharm, Jupyter Notebook.

VS Code — это редактор кода от Microsoft, который благодаря расширению Python становится полноценной средой разработки. Он поддерживает автодополнение, отладку, виртуальные окружения, управление зависимостями и терминал. Также можно настроить интерфейс под себя.

PyCharm — это среда разработки от JetBrains, специально созданная для Python. Поддерживает множество функций на встроенном уровне: автодополнение и анализ кода, работу с базами данных и фреймворками.

Jupyter — это удобная среда для работы с кодом и анализа данных, визуализации и создания алгоритмов. Ячейки кода можно запускать по одной, вставлять графики, текст и формулы. Подходит для задач, где важен постепенный контроль выполнения.

Таким образом, в качестве среды разработки выбор остановился на PyCharm.

1.2.4 Выбор архитектуры программы

Приложение состоит из нескольких компонентов, взаимодействующих друг с другом для моделирования транспортных потоков, интеллектуального управления светофорами и анализа данных. Архитектура включает следующие основные блоки: пользовательский интерфейс; обработчик/генератор входных данных; ядро моделирования (SUMO); интеллектуальный контроллер светофоров; анализатор характеристик моделирования; визуализатор результатов.

Таким образом, структура программы представлена на рисунке 5.

Система моделирования дорожного трафика начинается с интерфейса управления, который обеспечивает взаимодействие пользователя с программой. Через графический интерфейс пользователь может задавать параметры моделирования, такие как выбор алгоритма управления светофорами, количество эпох обучения и способ генерации входных данных. Также предусмотрена возможность загрузки уже готовых данных о транспортных потоках или автоматического их создания на основе заданных параметров.

Обработчик/генератор входных данных играет важную роль в подготовке информации для моделирования. В зависимости от настроек пользователя он принимает данные из XML файла либо самостоятельно их генерирует на основе полученных от пользователя данных, затем он преобразует их в формат, понятный SUMO. Так же в этом модуле считывается информация о количестве машин за интервал усреднения [21]. Далее обработанная информация о проездах машин загружается непосредственно в конфигурационные файлы SUMO.

Основу симуляции составляет ядро моделирования, представленное средой SUMO. Она выполняет все вычисления, связанные с движением транспорта по сети. Она инициализирует дорожную среду, управляет перемещением автомобилей и имитирует работу светофоров в соответствии с выбранным алгоритмом. Для динамического управления используется библиотека TraCI, которая обеспечивает связь между SUMO и интеллектуальными алгоритмами управления. Она позволяет в реальном времени изменять фазы работы светофоров.

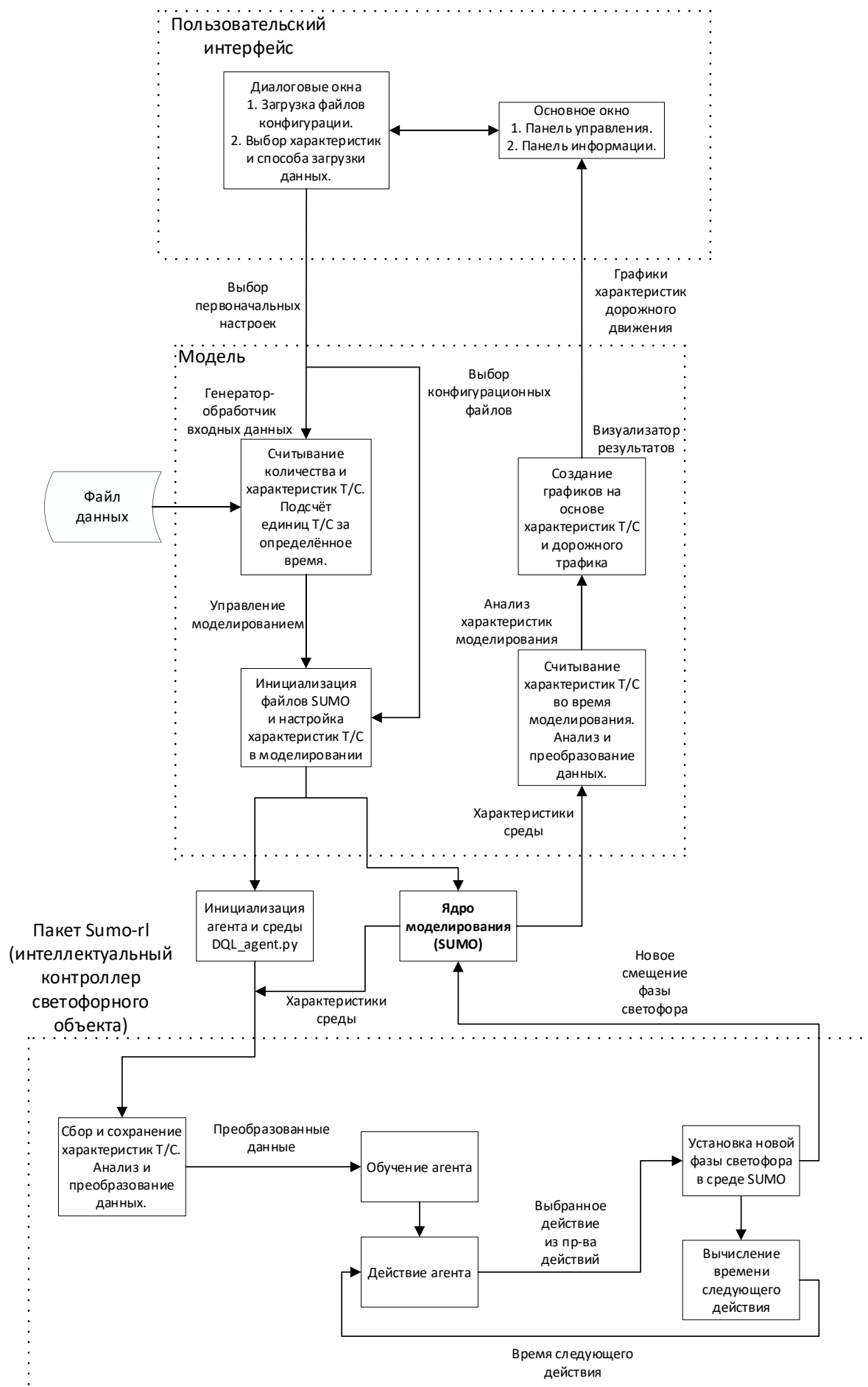


Рисунок 5 – Схема архитектуры программы

Одновременно с этим интеллектуальный контроллер управления светофорами анализирует дорожную ситуацию, принимает решения о смене фаз работы светофорного объекта и адаптируется к изменяющимся условиям трафика, постоянно обучаясь на основе поступающих данных.

В частности, используется алгоритм Q-обучения, который принимает решения, основываясь на Q-таблице. Также применяется подход с использованием нейронной сети, которая обучается предсказывать награды за действия в управлении светофорным объектом.

Для оценки эффективности алгоритмов применяется модуль анализа характеристик дорожного движения. Он собирает метрики, такие как среднее время ожидания, длина очередей и количество автомобилей на перекрёстке. Это позволяет проводить сравнение различных алгоритмов управления и определять наиболее эффективный из них. Все результаты сохраняются для последующего анализа.

Последним элементом является визуализатор результатов, который выполняет функции графического отображения полученных данных. Он позволяет строить графики зависимости различных параметров от времени моделирования. Таким образом, система не только моделирует транспортные потоки, но и представляет полученные результаты в виде графиков.

Взаимодействие между компонентами организовано по следующей схеме: пользователь через интерфейс управления инициирует моделирование, обработчик данных подготавливает необходимую информацию, ядро SUMO выполняет моделирование, интеллектуальный контроллер оптимизирует работу светофоров, а визуализатор предоставляет результаты в форме графиков зависимостей характеристик дорожного движения от времени моделирования.

Таким образом, выбранная архитектура и используемые инструменты разработки позволяют связать между собой анализ данных транспортных потоков, управление симуляцией и обучение интеллектуального алгоритма.

1.3 Проектирование и разработка

1.3.1 Механика моделирования в среде SUMO

SUMO предоставляет разнообразные методы моделирования, такие как: командная строка для запуска моделирования, конфигурационные файлы для определения среды, API для программного управления и сбора данных, а также графический интерфейс `sumo-gui` для визуализации моделирования. Благодаря этой совокупности инструментов, SUMO обеспечивает возможность выбора наиболее удобного подхода для решения задач моделирования транспортных систем.

Основные компоненты, необходимые для запуска модели: файл сценария (`.sumocfg`), файл сети дорог (`.net.xml`), файл маршрутов (`.row.xml`) файл результатов (`.output.xml`).

При использовании графического интерфейса `sumo-gui`, необходимо загрузить соответствующие файлы сценария и конфигурации, а затем визуализировать моделирование.

После этого можно наблюдать движение транспортных средств, состояние светофоров и другие параметры в режиме реального времени (рисунок 6).

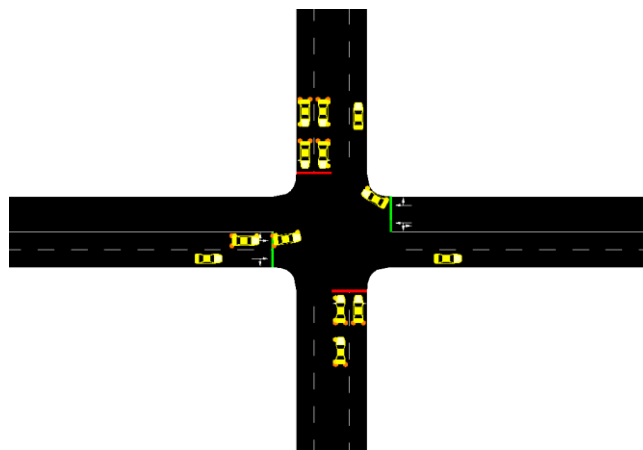


Рисунок 6 – Визуализация движения машин

Так же `sumo-gui` предоставляет возможность редактирования полигона для моделирования в ручном режиме при помощи графического редактора `Netedit`. Его можно использовать для создания или изменения формы и расположения дорожной сети.

Для возможности импорта дорожной сети служит файл `osmWebWizard.py`, который является скриптом в SUMO. Он позволяет пользователям импортировать данные из карты мира и получить разметку дорожной сети, которую можно использовать для проведения симуляции транспортного потока и исследования вариантов управления трафиком (рисунок 7).



Рисунок 7 – Результат использования `osmWebWizard.py`

1.3.2 Использование Python API (`sumolib`, `traci`)

При использовании Python API для программного управления и сбора данных, необходимо импортировать библиотеки `sumolib` и `traci`.

`sumolib` – это библиотека, предоставляющая функционал для работы с файлами SUMO, такими как сетевой файл (`.net.xml`), файл маршрутов (`.route.xml`) и другими. Она предлагает методы для чтения и записи этих файлов, а также для получения информации о дорожной сети, светофорах, полосах движения и других элементах модели.

`Traci` — это библиотека, которая предоставляет API для управления симуляцией транспортного потока в режиме реального времени. Она позволяет использовать функции для управления движением транспортных средств, получения информации о текущем состоянии моделирования, извлечения данных о транспортных средствах и других операций, которые позволяют создавать разнообразные сценарии.

Возможности библиотек важны при создании программы. `Sumolib` может использоваться для предварительной обработки данных о проездах транспортных

средств, а *traci* - для наблюдения за моделью на основе поступающих с датчиков данных. Такая система позволяет с высокой точностью [20] создавать транспортные модели.

1.3.3 Формат данных и интеграция с реальными источникам

Данные для моделирования из реальных источников должны быть представлены в xml файле *flows.xml*. В нём должна содержаться информация о *id* транспортного средства, его точке начала (*from*), конца (*to*) движения на перекрёстке, а также о времени отправления (*depart*).

Пример исходных данных:

```
<trip id="0" depart="0.0" from="A1B1" to="B1A1" />
<trip id="1" depart="0.0" from="A1B1" to="B1B0" />
```

Этот фрагмент кода описывает два проезда перекрёстка автомобилем. Файл содержит теги и атрибуты:

- *id* — уникальный идентификатор автомобиля;
- *depart* — время выезда из начальной точки (в секундах от начала моделирования);
- *from* — идентификатор начального участка дороги;
- *to* — идентификатор конечного участка.

Такой формат позволяет точно отразить движение отдельных автомобилей в процессе моделирования. Использование реальных маршрутов и времени выезда даёт возможность проводить моделирование, приближённое к действительности, анализировать загруженность конкретных участков дорог и оценивать эффективность интеллектуальных алгоритмов управления трафиком.

1.3.4 Разработка пользовательского интерфейса

Пользовательский интерфейс разработан с использованием библиотеки *Kivu*. Код графического интерфейса разделён на две части: разметку в файле *.kv* и обработку событий в файле *.py*.

Корневым виджетом является класс *Root*, который наследует *FloatLayout*. Корневой виджет *FloatLayout*, позволяет свободно расположение элементы. Класс *Root* содержит в себе следующие основные элементы управления: кнопки загрузки файлов конфигурации, настройки выбора и запуска режимов

моделирования, информацию о выбранных файлах и результатах моделирования.

Ниже представлена часть кода для обработки событий интерфейса в файле .py с методом и несколькими полями класса Root.

```
class Root(FloatLayout):
    net_file_path = ObjectProperty(None)
    route_file_path = ObjectProperty(None)
    def set_epochs(self, text):
        try:
            value = int(text)
            if value > 0:
                self.epochs = value
                print(f"[INFO] Количество эпох установлено: {self.epochs}")
            else:
                value = 1
        except Exception:
            return False
```

Класс Root содержит свойства net_info и route_info, которые используются для отображения текущего состояния загруженных файлов сети и маршрутов. Эти свойства связаны с метками на окне приложения. Метод set_epoch выполняет считывание количества эпох для обучения, которые ввёл пользователь и занесение значения во внутренние поля класса.

Ниже показана часть кода разметки интерфейса.

```
<Root>:
    Button:
        text: "Загрузить сеть"
        on_release: root.show_load_net()
    Button:
        text: "Загрузить маршруты"
        on_release: root.show_load_route()
```

Разметка описывает корневой класс Root и две кнопки. Каждая кнопка связана с функцией вызова, которая выполняет соответствующий алгоритм при нажатии.

Основной блок интерфейса – BoxLayout содержит в себе меню действий и информационное меню.

Меню действий представляет собой вертикальный список кнопок для

управления загрузкой данных, процессом моделирования и отображения результатов.

Информационное меню используется для отображения основных данных о моделировании: выбранных файлах, графиках моделирования и другой информации.

Общий вид интерфейса показан на рисунке 8.

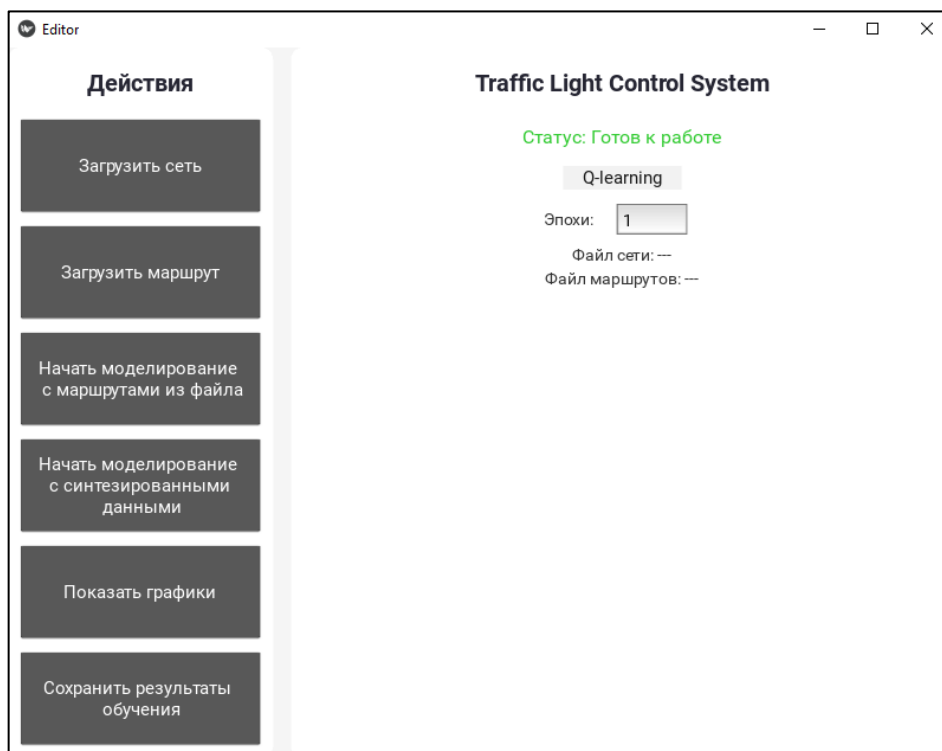


Рисунок 8 – Общий вид интерфейса

На левой стороне размещены кнопки: загрузить сеть, которая открывает окно выбора сетевого файла; загрузить маршрут – при нажатии открывает окно выбора файла маршрутов; начать моделирование с маршрутами из файлов, которая запускает моделирование с маршрутами из реальных источников; начать моделирование с синтезированными данными, которая при использовании открывает окно преднастройки синтезированных данных; сохранить результаты обучения – сохраняют выбранную обученную модель.

Функция `show_load_net` открывает окно выбора файла сети. После выбора файла в поле `net_info` сохраняется путь до файла дорожной сети. Окно реализовано при помощи виджета `LoadNetDialog`.

```
def show_load_net(self):
    content = LoadNetDialog(load_net=self.load_net,
cancel=self.dismiss_popup)
    self._popup = Popup(title="Load net file", content=content,
size_hint=(0.9, 0.9))
    self._popup.open()
```

Похожим образом реализована загрузка маршрутов через функцию show_load_route.

```
def show_load_route(self):
    content = LoadRouteDialog(load_route=self.load_route,
cancel=self.dismiss_popup)
    self._popup = Popup(title="Load route file", content=content,
size_hint=(0.9, 0.9))
    self._popup.open()
```

Ниже показана часть кода для разметки окна загрузки конфигурационных файлов.

```
<LoadNetDialog>:
    FileChooserIconView:
        id: filechooser
    BoxLayout:
        size_hint_y: None
        height: 50
        spacing: 12
        Button:
            text: "Отмена"
            on_release: root.cancel()
            background_color: 0.9, 0.3, 0.3, 1
        Button:
            text: "Загрузить"
            on_release: root.load_net(filechooser.path,
filechooser.selection)
            background_color: 0.3, 0.7, 0.3, 1
```

Разметка обозначает FileChooserIcon для отображения файлов и две кнопки для отмены и сохранения результатов выбора.

На рисунке 9 показан общий вид выбора конфигурационных файлов для моделирования.

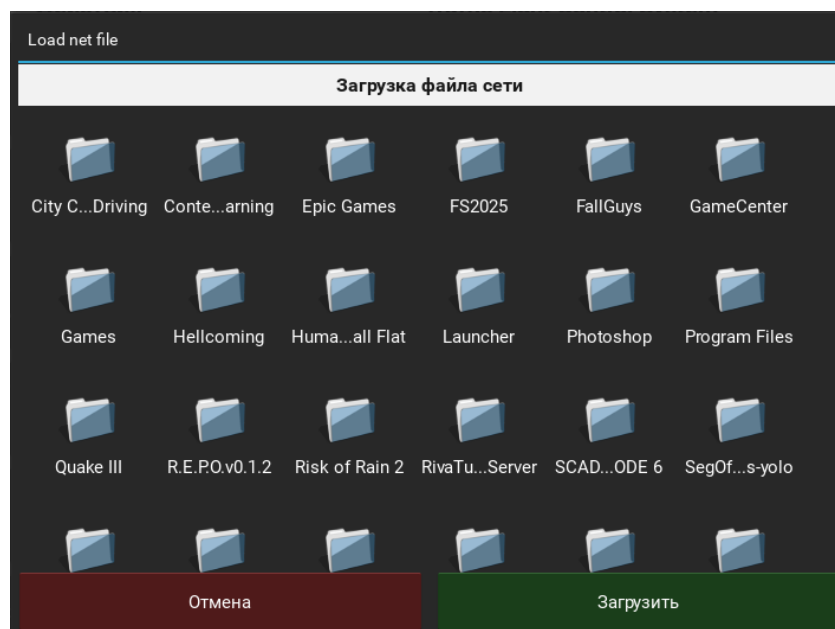


Рисунок 9 – Выбор конфигурационных файлов

При необходимости пользователь может сам задать параметры генерации маршрутов. Метод `show_route_config` вызывает окно с настройками для генерации данных.

```
def show_route_config(self):
    content = ConfigRouteDialog(config=self.config, ...)
    content.set_config_ratio(self.config)
    self._popup = Popup(title="Configure route", content=content,
        size_hint=(0.9, 0.9))
    self.popup.open()
```

Метод принимает текущую конфигурацию `self.config` и передает ее в диалоговое окно `ConfigRouteDialog`, которое содержит элементы управления для настройки. После инициализации диалога методом `set_config_ratio` создается экземпляр `Popup` с заголовком "Configure route", который занимает 90% ширины и высоты родительского окна `size_hint=(0.9, 0.9)`. Этот метод вызывается из пользовательского интерфейса по нажатию кнопки и позволяет настраивать параметры генерации маршрутов, такие как количество машин, интенсивности движения по направлениям и во времени.

Разметка для окна выбора характеристик для генерации трафика представлена в классе `ConfigRouteDialog`.

Изм.	Лист	№ докум.	Подпись	Дата

Предусмотрено сохранение и загрузка параметров, которые используются в режиме генерации маршрутов. Это сделано для удобства пользователя, чтобы он мог сохранять настройки и использовать их в будущих моделированиях в рамках одной сессии.

Ниже приведён пример кода для загрузки и сохранения параметра интенсивности движения по направлениям.

```
def set_road_ratio(self, _, __):
    ratio = []
    for text_input in self.ids.ratio_box_layout.children:
        text = text_input.text
        if text.replace('.', '', 1).isdigit():
            ratio.append(float(text))
        else:
            ratio.append(0.0)
    ratio.reverse()
    self.config['ratio'] = ratio
    self.load_road_ratio(ratio)
    self.save_button_disabled_flag = not (self.check_ratios_sum_to_one()
    and self.check_flux_sum_to_one())
```

Метод сохраняет лист с информацией об интенсивности по направлениям, считывает значения из всех текстовых полей в контейнере `ratio_box_layout` и преобразует их в числа с плавающей запятой.

```
def get_ratio(self, box_layout, text_input):
    for i in range(len(box_layout.children)):
        if text_input == box_layout.children[i]:
            return self.config['ratio'][i]
```

Через цикл метод перебирает все дочерние элементы в `box_layout` и при совпадении `text_input == box_layout.children[i]` возвращает `self.config['ratio'][i]`.

Перед запуском моделирования проводится проверка корректности введённых данных, чтобы предотвратить ошибки генерации маршрутов.

Метод `check_flux_sum_to_one` проверяет, что сумма всех коэффициентов распределения трафика (`ratio`) равна 1. Это необходимо для корректного задания времени появления транспортного средства в модели.

Метод `check_ratios_sum_to_one` проверяет, что сумма потоков трафика

flux по всем направлениям тоже равна 1. Проверка нужна для корректного вычисления вероятности появления машины на направлении.

На рисунке 10 показан вид ввода характеристик для генерации данных перед моделированием.

Рисунок 10 – Окно ввода характеристик

После выбора сети, маршрутов и параметров пользователь может запустить моделирование кнопкой. Метод `start_simulation` создаст конфигурационную структуру для моделирования в зависимости от настроек и передаст её в модуль `sim_main`. Сам процесс моделирования выполняется в отдельном потоке с целью избежать заморозки интерфейса.

Перед началом устанавливается флаг `self.is_modeling = True`, чтобы показать, что процесс моделирования начался. Затем проводится запуск основной функции симуляции `sim_main(self.sim_inf)`, куда передаётся структура.

Если в процессе моделирования возникает ошибка, то она перехватывается при помощи `try-except`. Затем устанавливается флаг ошибки `self.is_error = True` и в интерфейс выводится информация об ошибке.

После завершения моделирования программа собирает все изображения в список `self.graphics`.

По окончании работы моделирования устанавливается `self.is_modeling = False`.

В интерфейсе предусмотрены отдельная кнопка для визуализации графиков результатов моделирования. При нажатии на неё программа перебирает список сохранённых в `self.graphics` путей с изображениями и выводит их в центр информационного меню в столбик. Ниже представлена часть кода разметки для вывода графиков.

```

ScrollView:
    id: scrollView
    do_scroll: (True, True)
    size_hint: (1, 1)
    BoxLayout:
        orientation : 'vertical'
        size_hint_y : None
        id: scrollViewBox
        height: self.minimum_height

```

В коде `ScrollView` отвечает за вертикальную прокрутку. Внутри `ScrollView` находится `BoxLayout`, в который добавляются изображения графиков.

Для вызова метода `show_metrics` по нажатию кнопки используется следующий код:

```

Button:
    text: 'Показать графики'
    on_release: root.show_metrics(scrollViewBox)

```

Через событие вывод графиков обрабатывается в функции `show_metrics`. Её код представлен ниже.

```

def show_metrics(self, scrollViewBox):
    if self.graphics is not None:
        scrollViewBox.clear_widgets()
        scrollViewBox.height = 0
        for graph in self.graphics:
            img = ScrollViewImage(source=graph, height=500)
            scrollViewBox.add_widget(img)
            scrollViewBox.height += img.height - 20

```

В начале выполняется проверка, что поле `self.graphics` не пустое. Затем очищаются все виджеты, чтобы удалить старые графики. После для каждого графика создаётся свой виджет `ScrollViewImage`,

в который загружается соответствующий график. Созданный виджет добавляется в scrollViewBox и отображается в окне приложения.

На рисунке 11 показан вид интерфейса с отображёнными графиками.

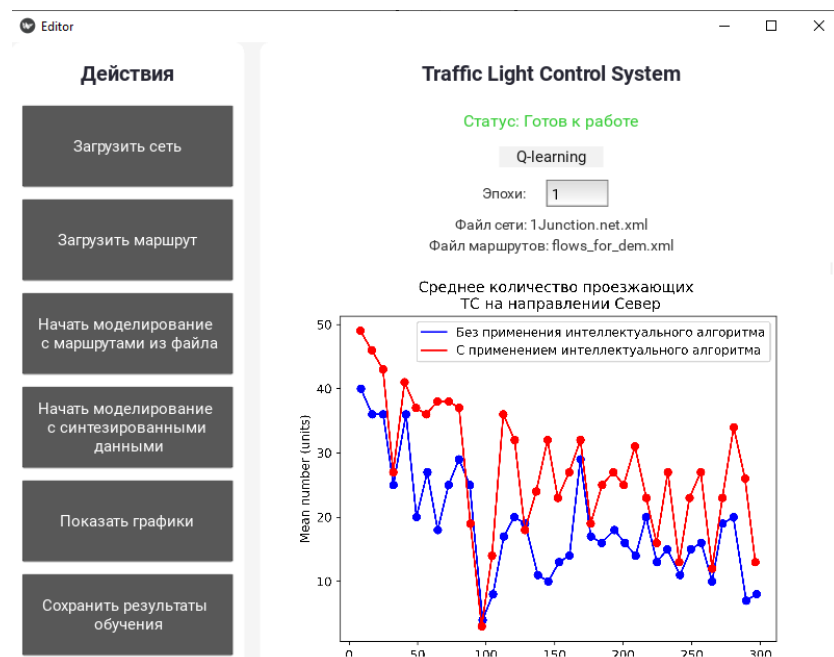


Рисунок 11 – Визуализация графиков

После завершения обучения у пользователя есть возможность сохранения результатов обучения на произвольное место на диске. Это позволяет хранить несколько обученных моделей.

Для сохранения используется отдельный диалог – класс SaveDialog. Когда пользователь выбирает сохранение, то открывается окно с выбором места сохранения и названия файла при помощи FileChooserIconView. Расширение файла устанавливается автоматически, в зависимости от выбранного алгоритма. Имя файла вводится в TextInput.

Логика сохранения файла представлена в методе save.

```
def save(self, path, file_name):
    if self.algorithm == "Q-learning":
        full_path = str(path + "/" + file_name + ".pkl")
        shutil.copy2('database/dataDWT.pkl', full_path)
    elif self.algorithm == "Neural Net":
        full_path = str(path + "/" + file_name + ".pth")
        shutil.copy2('database/dataQ.pth', full_path)
    self.cancel()
```

В зависимости от выбранного алгоритма выбирается файл с результатами обучения, который необходимо сохранить. Файлы копируются в выбранную пользователем папку path с новым именем file_name.

Информационная панель служит для отображения текущего статуса работы системы, вывода текущих файлов конфигурации, а также выбора количества эпох обучения и интеллектуального алгоритма.

Общий вид информационной панели показан на рисунке 12.

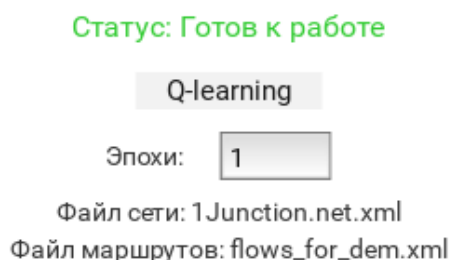


Рисунок 12 – Информационная панель

Ниже представлен элемент информационной панели.

```
Label:
    id: status_label
    text: "Статус: Моделирование" if root.is_modeling else ("Статус: Ошибка
моделирования!" if root.is_error else "Статус: Готов к работе")
    color: (0.9, 0.54, 0.15, 1) if root.is_modeling else (0.93, 0.25, 0.13,
1) if root.is_error else (0.2, 0.8, 0.2, 1)
    font_size: '16sp'
    size_hint_y: None
    height: "30dp"
```

Данный Label автоматически меняет текст и цвет на основе текущего состояния системы. Если моделирование активно root.is_modeling, то отображается “статус: Моделирование”. Если возникла ошибка root.is_error, то отображается “статус: Ошибка”. В остальных случаях система находится в статусе “Готов к работе”.

В программе возможен выбор интеллектуального алгоритма управления светофорными объектами при помощи виджета Spinner. Ниже показан фрагмент разметки для выбора алгоритма.

Spinner:

```
text: root.algorithm
values: ["Q-learning", "Neural Net"]
on_text: root.set_algorithm(self.text)
size_hint: None, None
size: "100dp", "20dp"
background_normal: ''
background_color: (0.95, 0.95, 0.95, 1)
color: 0.1, 0.1, 0.1, 1
```

При помощи поля text устанавливается текущий выбранный алгоритм, который хранится в поле self.algorithm. При выборе нового значения вызывается метод self.set_algorithm, который обновляет поле self.algorithm.

Для ввода количества эпох обучения в интерфейсе создан элемент ввода при помощи виджета TextInput.

Label:

```
text: "Эпохи:"
size_hint_x: None
width: "60dp"
font_size: '14sp'
color: 0.2, 0.2, 0.2, 1
```

TextInput:

```
id: epoch_input
text: str(root.epochs)
size_hint: None, None
size: "60dp", "26dp"
multiline: False
input_filter: 'int'
font_size: '14sp'
on_text: root.set_epochs(self.text)
```

Значение изначально равно текущему количеству эпох в self.epochs. При помощи свойства input_filter: 'int' ввод ограничивается только целыми числами. После изменения текста вызывается метод self.set_epochs, который сохраняет новое значение.

Метод set_epochs отвечает за установку количества эпох для обучения.

```
def set_epochs(self, text):
    try:
        if value > 0:
```

```

        self.epochs = value
        print(f"[INFO] Количество эпох установлено: {self.epochs}")
    else:
        value = 1
    except Exception:
        return False

```

Если полученное число больше нуля, то оно сохраняется в поле `self.epochs`. В случае некорректного ввода, произойдёт исключение и метод вернёт `False`, предотвратив аварийное завершение программы.

1.3.5 Разработка модуля загрузки и генерации данных

Для работы симулятора SUMO нужно создать файл транспортных потоков, который будет включать порядковый номер машины, начальный и конечный пункт движения, а также время появления в сети моделирования. В программе реализовано два способа формирования конфигурационного файла с информацией о проездах транспортных средств: загрузка данных из реальных источников и генерация синтетических данных.

Логика выбора варианта использования данных представлена в функции `sim_main`.

```

def sim_main(sim_inf):
    ...
    try:
        if sim_inf['mode'] == "Flows":
            os.system(
                f"duarouter --flow-files={sim_inf['route_path'][0]} --net-
                file=1Junction.net.xml "
                "--output-file=grid.rou.xml --begin 0 --end 1000000000"
            )
        ...

```

В случае выбора режима “Flows”, программа загружает указанный пользователем файл с маршрутами транспортных средств. Затем при помощи утилиты SUMO `duarouter` преобразует его в конфигурационный файл `grid.rou.xml`, который используется в моделировании. Такой способ упрощает работу пользователя и предоставляет ему стандартизированный способ для сбора данных о движении транспортных средств на перекрёстке для

последующего использования этих данных в SUMO.

Если выбран режим “generate”, то данные о количестве машин, интенсивности по направлениям движения и во времени пользователю необходимо будет ввести самостоятельно. Генерация данных осуществляется при помощи последовательного вызова трёх функций: `generate_ways`, `flux_to_route` и утилиты `duarouter`, которая вызывается через командную строку. Ниже представлен код функции `generate_ways`.

```
def generate_ways():
    ways_move = []
    incoming_routes = []
    exiting_routes = []
    temp = sumolib.net.readNet('1Junction.net.xml')
    routes = temp.getEdges()
    for route in routes:
        if len(route.getIncoming()) > 1 and len(route.getOutgoing()) == 0:
            exiting_routes.append(route.getID())
        else:
            incoming_routes.append(route.getID())
    ways_move.extend([[x, exiting_routes] for x in incoming_routes])
    return ways_move
```

Эта функция создаёт список допустимых направлений движения для каждого направления движения на перекрёстке, используя данные сетевого файла. При помощи библиотеки `sumolib` дорожная среда сохраняется в переменную `temp` для удобной работы с сетью дорог и перекрёстком. Далее в цикле для каждой въездной дороги формируется лист всех допустимых выездных направлений. Результат работы сохраняется в структуру `ways_move`, которая имеет вид: `[входное_ребро, [выходные_ребра]]`.

Далее функция `flux_to_route` рассчитывает для каждого автомобиля его начальный и конечный маршрут, а также время выезда на перекрёсток. Вычисление ведётся на основе информации о количестве машин `cars`, коэффициентах интенсивности по направлениям `ratio` и времени `flux`.

Ниже показан код этой функции.

```

def flux_to_route(ratio, flux_list, period, ways, cars):
    ...
    for way in ways:
        ratio_flux_list = [
            cars / len(ways) * ratio[incoming_routes] * flux_list[i] for i
in range(len(flux_list))
        ]
        for exiting_routes in way[1]:
            while i < len(ratio_flux_list):
                if offset >= math.floor(ratio_flux_list[i]):
                    error += ratio_flux_list[i] - offset
                    if error >= 1:
                        v = V(period * i + period / ratio_flux_list[i] *
offset, way[0], exiting_routes)
                        set_list.append(v)
                        error -= 1
                        i += 1
                        offset = 0
                        continue
                    v = V(period * i + period / ratio_flux_list[i] * offset,
way[0], exiting_routes)
                    set_list.append(v)
                    offset += 1
                i = 0
            incoming_routes += 1
        set_list.sort(key=lambda x: x.depart)
        os.system(f"randomTrips.py -n 1Junction.net.xml -o flows.xml -e
{len(set_list)}")
        i = 0
        tree = ET.parse('flows.xml')
        for trip in tree.getroot():
            trip.set('depart', str(set_list[i].depart))
            trip.set('from', set_list[i].attr_from)
            trip.set('to', set_list[i].to)
            i += 1
        tree.write("flows.xml")

```

В начале функция начинает перебор всех входных направлений из структуры ways.

Для каждого направления рассчитывается список `ratio_flux_list`. В нём для каждого периода определяется сколько автомобилей должно быть сгенерировано. Это значение зависит от общего числа автомобилей, потока на конкретном направлении и интенсивности потока в период времени.

Затем для каждого выходного направления создаются объекты `v`, которые описывают каждое транспортное средство. Для расчёта времени появления каждой машины используется формула, которая учитывает текущий период и количество автомобилей, уже сгенерированных в этом периоде. Стоит отметить, что сохраняется дробная ошибка `error`, которая возникает при округлении количества машин. Она компенсируется при достижении достаточного накопления `offset`. В конце каждый объект, содержащий время появления в сети `depart`, точку отправления `from` и назначения `to` добавляется в список машин `set_list`.

Когда завершилось создание всех объектов, то они сортируются по времени выезда, чтобы сохранить временную корректность моделирования. Затем вызывается утилита `randomTrips.py`, которая генерирует файл `flows.xml`. Затем этот файл перезаписывается значениями из листа `set_list`.

На выходе функция формирует готовый XML – файл, который содержит описание всех сгенерированных транспортных средств.

1.3.6 Разработка модуля анализа и визуализации

Модуль анализа и визуализации создан для сбора и агрегации результатов моделирования и создания графиков, которые показывают различные характеристики дорожного движения в зависимости от времени моделирования.

Центральной частью модуля является метод `_get_period_metrics`, который собирает данные о метриках `self.metrics` в моделировании за периоды. Часть метода представлена ниже.

```
def _get_period_metrics(self):
    for ts_id in self.ts_ids:
        buff = dict(
            {"step": 0, ts_id + "_stopped": 0, ts_id +
             "_accumulated_waiting_time": 0, ts_id + "_average_speed": 0})
        buff.update({"vehiclesIDson_lane " + lane[:-2]: set() for lane in
```

```

self.traffic_signals[ts_id].lanes})
    buff.update({"ts_time": [0, 0]})
    buff.update(
        {"vehicleMeanWaitingTime_lane " + lane[:-2]: float() for lane in
self.traffic_signals[ts_id].lanes})
    buff.update(
        {"vehicleTotalHalting_lane " + lane[:-2]: float() for lane in
self.traffic_signals[ts_id].lanes})
    for i in range(len(self.metrics)):
        if int(self.metrics[i].get("step")) <= last_period +
int(self.period):
            buff |= {ts_id + "_accumulated_waiting_time":
                    float(self.metrics[i].get(ts_id +
"_accumulated_waiting_time")) + float(buff.get(
                    ts_id + "_accumulated_waiting_time"))}
            else:
            buff |= {"step": self.metrics[i].get("step")}
            buff |= {ts_id + "_accumulated_waiting_time":
                    float(buff.get(ts_id +
"_accumulated_waiting_time")) / float(
                    self.period / self.delta_time)}
            buff |= {"vehiclesIDson_lane " + lane[:-2]:
len(buff.get("vehiclesIDson_lane " + lane[:-2], [])) for
                    lane in self.traffic_signals[ts_id].lanes}
            buff |= {"vehicleTotalHalting_lane " + lane[:-2]: float(
                    buff.get("vehicleTotalHalting_lane " + lane[:-2]))
                    /
float(self.period / self.delta_time) for lane in
self.traffic_signals[ts_id].lanes}
            self.metricsPeriod.append(buff.copy())
            buff = dict(
                {"step": 0, ts_id + "_stopped": 0, ts_id +
"_accumulated_waiting_time": 0,
                ts_id + "_average_speed": 0})

```

Метод начинает работу с перебора светофорных объектов ts_id и создаёт для каждого из них словарь buff, в который записываются нулевые значения для некоторых характеристик движения. Также в словарь добавляется массив времени фаз светофора.

Затем алгоритм проходит по каждому шагу моделирования, в которых были собраны метрики. Если текущий шаг попадает в заданный период $\text{last_period} + \text{self.period}$, то его метрики накапливаются в словаре `buff`. Для каждой метрики создан свой словарь, в который добавляется новое значение.

Если текущий шаг моделирования выходит за пределы текущего периода, то есть $\text{int}(\text{self.metrics}[i].\text{get}(\text{"step"})) \leq \text{last_period} + \text{int}(\text{self.period})$, то происходит обработка накопленных данных. Вычисляются средние данные для каждой накопленной метрики и результат сохраняется в список `self.metricsPeriod`. После этого словарь `buff` обнуляется и начинается сбор данных для следующего периода. Стоит отметить, что метрики, собранные в этом методе, используются в качестве пространства наблюдения агента.

Затем метод `save_csv` сохраняет созданные метрики `self.metricsPeriod` в формате `.csv`. На вход она принимает путь к выходному файлу и номер эпизода для создания уникального имени. Ниже представлен код метода.

```
def save_csv(self, out_csv_name, episode):
    self._get_period_metrics()
    super(TLSCEnvironment, self).save_csv(out_csv_name, episode)
    if out_csv_name is not None:
        df = pd.DataFrame(self.metricsPeriod)
        Path(Path(out_csv_name).parent).mkdir(parents=True, exist_ok=True)
        df.to_csv(out_csv_name + f"_conn{self.label}_ep{episode}" +
            f"_period{self.period}" + ".csv", index=False)
        for id in self.ts_ids:
            Path(Path(out_csv_name).parent).mkdir(parents=True,
            exist_ok=True)
            df.to_csv(out_csv_name + f"_conn{self.label}_ep{episode}" +
            f"_ts{id}" + ".csv",
                index=False)
```

В коде преобразуется структура собранных метрик `self.metrics` в табличный формат `df` при помощи библиотеки `pandas`. Затем происходит проверка наличия директории по заданному пути и необходимости создаёт её. Файл сохраняется с указанием номера соединения `self.label`, эпизода обучения `self.episode` и длительности периода `self.period`.

Ниже представлен пример функции carsHaltingInPeriod. Остальные функции визуализации имеют аналогичную структуру и логику работы.

```
def carsHaltingInPeriod(metricsPeriodOn, metricsPeriodOff, attempt):
    filenames = []
    edges_labels = ['Север', 'Восток', 'Юг', 'Запад']
    for i, (laneOn, laneOff, edge) in enumerate(
        zip([laneOn for laneOn in metricsPeriodOn[0] if
            "vehicleTotalHalting_lane " in laneOn], [laneOff for laneOff in
            metricsPeriodOff[0] if "vehicleTotalHalting_lane " in laneOff],
            edges_labels)):
        plt.plot([step["step"] / 60 for step in metricsPeriodOn[:]],
            [l[laneOn] for l in metricsPeriodOn[:]],
            label="С применением интеллектуального алгоритма",
            color="red")
        plt.plot([step["step"] / 60 for step in metricsPeriodOn[:]],
            [l[laneOn] for l in metricsPeriodOn[:]], 'o', color="red")

        plt.plot([step["step"] / 60 for step in metricsPeriodOff[:]],
            [l[laneOff] for l in metricsPeriodOff[:]],
            label="Без применения интеллектуального алгоритма",
            color="blue")
        plt.plot([step["step"] / 60 for step in metricsPeriodOff[:]],
            [l[laneOff] for l in metricsPeriodOff[:]], 'o',
            color="blue")
        plt.legend(loc='best')
        plt.title(f"Среднее количество ожидающих \n ТС на направлении
            {edges_labels[i]}")
        plt.ylabel('Mean halting number (units)')
        plt.xlabel('Simulation time (minutes)')
        filename = f'graph//vehicleMeanHalting_' + str(attempt) +
            str(edges_labels[i]) + '.png'
        plt.savefig(filename, dpi=600)
        plt.close()
        filenames.append(filename)
    return filenames
```

На вход функция получает два массива метрик: metricsPeriodOn и metricsPeriodOff, которые хранят данные моделирования с включенным и выключенным агентом.

Третьим параметром является `attempt`, который используется для формирования уникальных имён для графиков в разных эпохах моделирования.

Для каждого из четырёх направлений движения: север, юг, запад, восток строится график, который показывают зависимость количества машин, ожидающих проезда перекрёстка, от времени моделирования. Полосы движения определяются автоматически — из словарей метрик извлекаются значения, содержащие подстроку `vehicleTotalHalting_lane`.

Для каждого направления формируются две кривые: красная линия — для моделирования с интеллектуальным алгоритмом; синяя линия — для моделирования без интеллектуального алгоритма.

Горизонтальная временная ось `[step["step"] / 60]` на графике указана в минутах, по вертикальной линии откладывается среднее количество остановившихся транспортных средств. При помощи `plt.legend(loc='best')`, `plt.title` и `plt.label` добавляется легенда и соответствующий заголовок для графика.

Полученный график сохраняется в виде файла `.png` в папке `graph`. Название каждого файла включает в себя название характеристики, номер попытки и название направления.

Таким образом функции визуализации позволяют показывать зависимости различных характеристик дорожного движения от времени моделирования.

1.4 Тестирование и анализ результатов

В данном разделе проводится проверка работоспособности разработанной системы. Основное внимание уделяется взаимодействию пользователя с интерфейсом, корректности запуска моделирования, а также стабильности работы алгоритмов генерации и обработки данных.

1.4.1 Тестирование и анализ интерфейса приложения

Пользовательский интерфейс является главным элементом взаимодействия пользователя с системой моделирования. Его тестирование производилось на всех этапах разработки. Проверялась доступность и понятность элементов управления, реакцию системы на действия пользователя и корректность отображения графиков и прочей информации.

					ПГУ 1.090301.16.001 ПЗ	Лист
						39
Изм.	Лист	№ докум.	Подпись	Дата		

В рамках тестирования были проверены следующие варианты использования программы.

Проверка запуска моделирования с выбором интеллектуального алгоритма. Пользователь загружает файл сети (рисунок 13), выбирает способ загрузки файла маршрутов (рисунок 14) и выбирает соответствующий режим и количество эпох на главном экране (рисунок 15). Затем он запускает моделирование нажатием на кнопку. После этого появляется информация о процессе моделирования (рисунок 16).

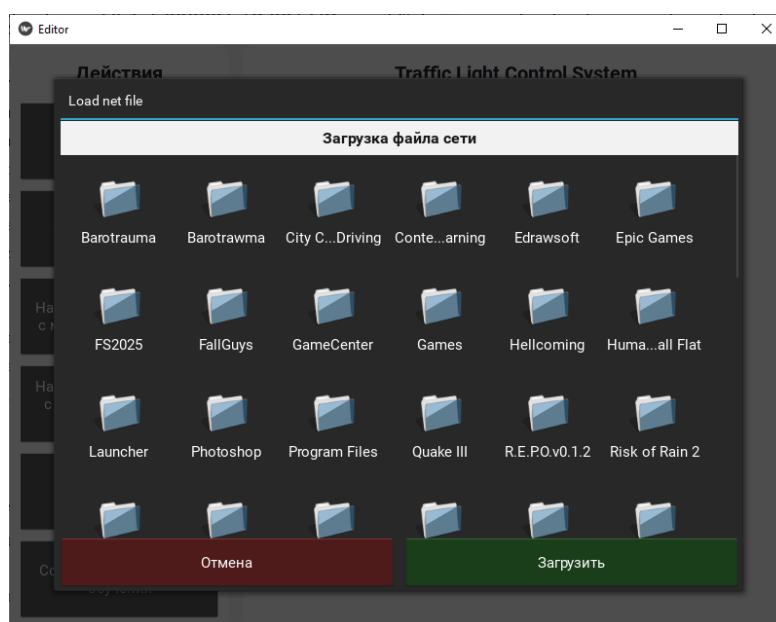


Рисунок 13 – Загрузка файла сети

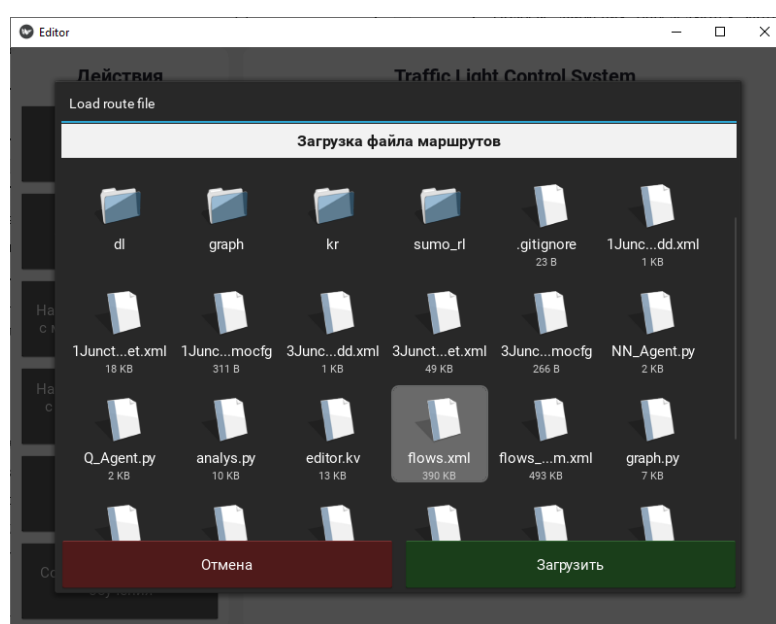


Рисунок 14 – Вариант загрузки файла маршрутов

Traffic Light Control System

Статус: Готов к работе

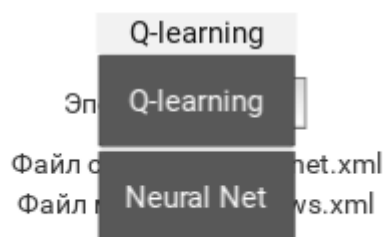


Рисунок 15 – Выбор интеллектуального алгоритма

Traffic Light Control System

Статус: Моделирование

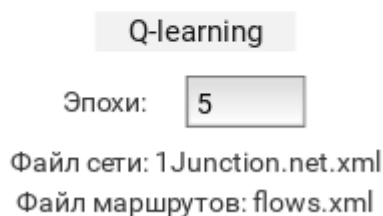


Рисунок 16 – Информация о процессе моделирования

После работы приложение вывело статус “Готов к работе”, что говорит о том, что моделирование прошло успешно (рисунок 17).

Traffic Light Control System

Статус: Готов к работе

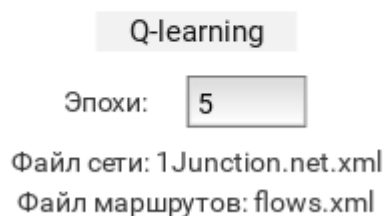


Рисунок 17 – Успешное моделирование

Все действия выполнялись без задержек и ошибок, что подтвердило

корректную реализацию логики работы с этим алгоритмом.

Проверка отсутствия выбора или неверного выбора конфигурационного файла разметки или маршрутов транспортных средств. В случае неверного выбора файла интерфейс выведет сообщение об ошибке и аварийно завершит работу, сохраняя функционал системы для будущих запусков. Это демонстрирует логику обработки исключений в программе. На рисунках 18 и 19 показано тестирование неверного выбора конфигурационных файлов или отсутствия выбора файла.

Traffic Light Control System

Статус: Ошибка моделирования!

Q-learning

Эпохи: 5

Файл сети: analys.py

Файл маршрутов: test.xml

Рисунок 18 – Результат неверного выбора конфигурационных файлов

Traffic Light Control System

Статус: Ошибка моделирования!

Q-learning

Эпохи: 1

Файл сети: 1Junction.net.xml

Файл маршрутов: —

Рисунок 19 – Отсутствие выбора файла

В следующем сценарии тестируется возможность повторного запуска моделирования с изменёнными характеристиками режима генерации данных. После завершения одного моделирования пользователь запускает новое, с другими характеристиками генерации данных.

Моделирование было проведено на данных, представленных на рисунке 20.

Общее количество машин:	3500							
Распределение по периодам:	0. 125	0. 125	0. 125	0. 125	0. 125	0. 125	0. 125	0. 125
	+				-			
	С		В		Ю		З	
Соотношение направлений:	0.25		0.25		0.25		0.25	
Отмена				Начать моделирование				

Рисунок 20 – Первое тестовое моделирование

Графики моделирования (рисунок 21) и системное уведомление SUMO показывают, что моделирование успешно прошло за 110 минут симуляции.

Step #6076.00 (0ms ?*RT. ?UPS, TraCI: 276ms, vehicles TOT 3500 ACT 0 BUF 0)

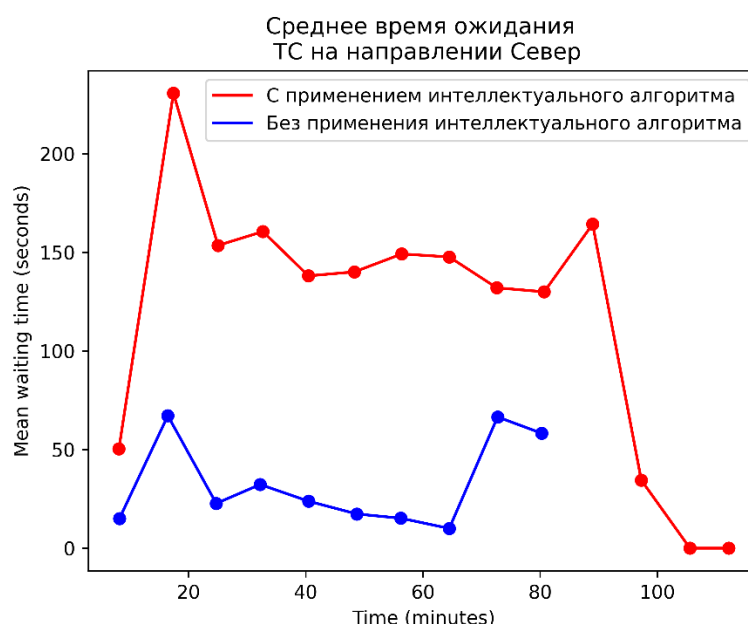


Рисунок 21 – Результат первого тестового моделирования

Второе тестовое моделирование проводилось со следующими настройками генерации (рисунок 22).

Общее количество машин:	6500									
Распределение по периодам:	0.07	0.13	0.11	0.09	0.06	0.14	0.08	0.05	0.12	0.15
	+					-				
	С		В		Ю		З			
Соотношение направлений:	0.25		0.3		0.2		0.25			
Отмена					Начать моделирование					

Рисунок 21 – Второе тестовое моделирование

Графики моделирования (рисунок 23) и системное уведомление SUMO показывают, что моделирование успешно прошло за 180 минут симуляции.

Step #11267.00 (0ms ?*RT. ?UPS, TraCI: 279ms, vehicles TOT 6500 ACT 0 BUF 0)

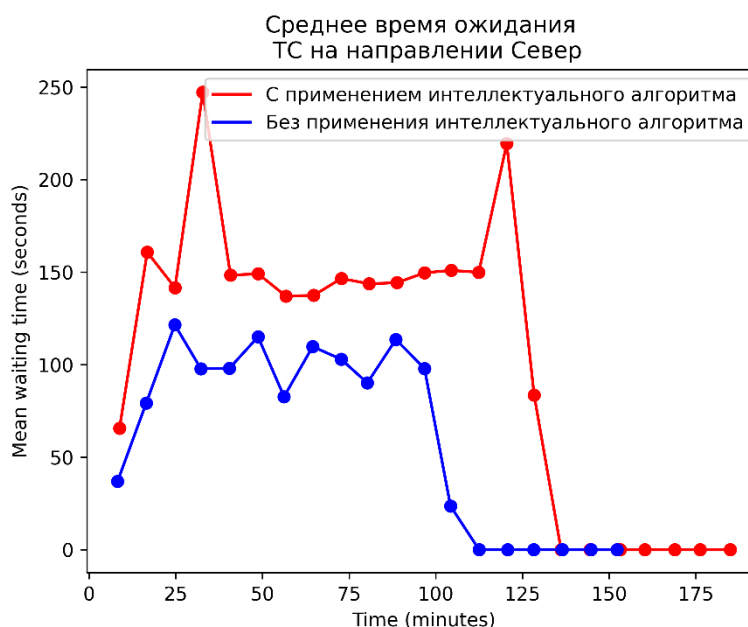


Рисунок 23 – Результат второго тестового моделирования

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.16.001 ПЗ

Лист

44

Таким образом, по результаты двух моделирований видно, что пользователь может успешно изменять характеристики для генерации данных. Это показало корректную реализацию механизма повторного моделирования.

После обучения модели файл обучения возможно сохранить в указанное пользователем место. Это даёт пользователю возможность обучения нескольких моделей для разных задач. Сохранение файла происходит в зависимости от выбранного алгоритма обучения в информационной панели. Ниже приведен сценарий сохранения файлов обученной модели Q-обучения.

После нажатия кнопки “Сохранить результаты обучения” открывается диалоговое окно с файловой системой (рисунок 24).

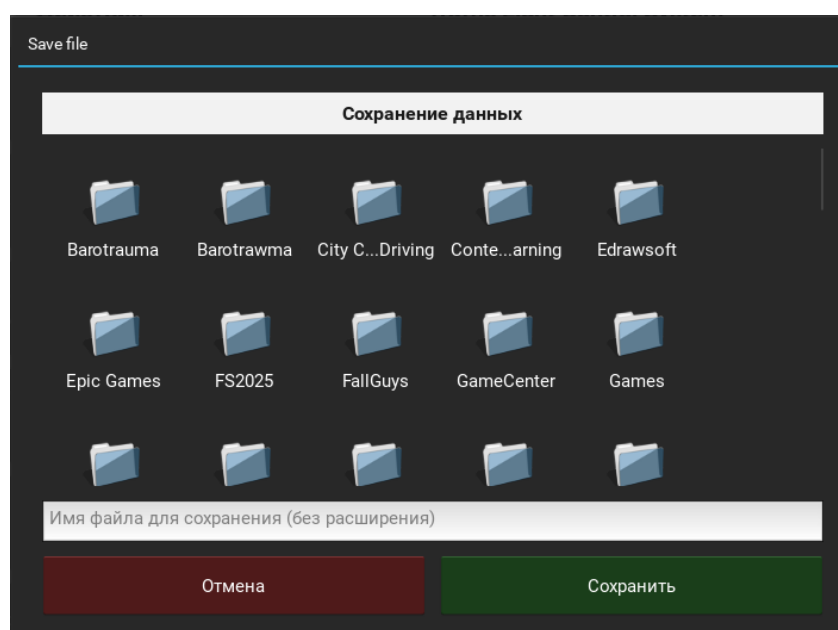


Рисунок 24 – Сохранение результатов обучения

Для проверки Q-таблица сохранена под именем Test_weights в директории Model_weights.

Проверка директории показала, что файл успешно сохранился в нужном разрешении .pkl (рисунок 25).

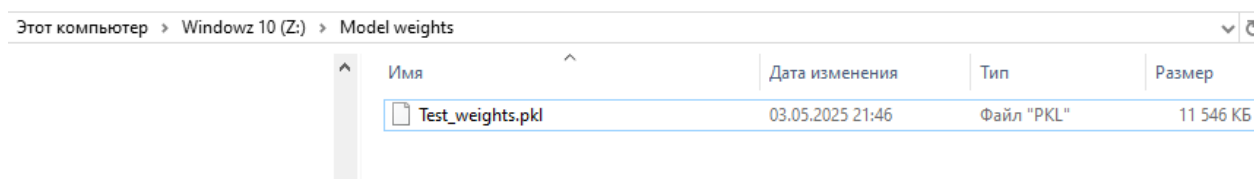


Рисунок 25 – Проверка сохранения результатов обучения

1.4.2 Тестирование алгоритма генерации данных

Алгоритм генерации данных создаёт дорожную ситуацию на основе параметров, которые вводит пользователь через графический интерфейс. К таким параметрам относится общее количество машин в моделировании, интенсивность по каждому направлению и интенсивность во времени.

Тестирование проводилось по нескольким сценариям, которые проверяли равномерное и неравномерное распределение трафика во времени и по направлениям.

На рисунке 26 показан вариант тестирования равномерного сценария моделирования. Создано 5 периодов, в каждом из которых создаётся 20 % от общего числа машин. Интенсивность для каждого направления равна 25 % от общего числа машин. Количество машин равно 5500 единиц.

Load route file

Общее количество машин: 5500

Распределение по периодам: 0.2 0.2 0.2 0.2 0.2

Соотношение направлений: 0.25 0.25 0.25 0.25

Отмена Начать моделирование

Рисунок 26 – Конфигурация для равномерного сценария

Результаты моделирования показаны на рисунке 27.

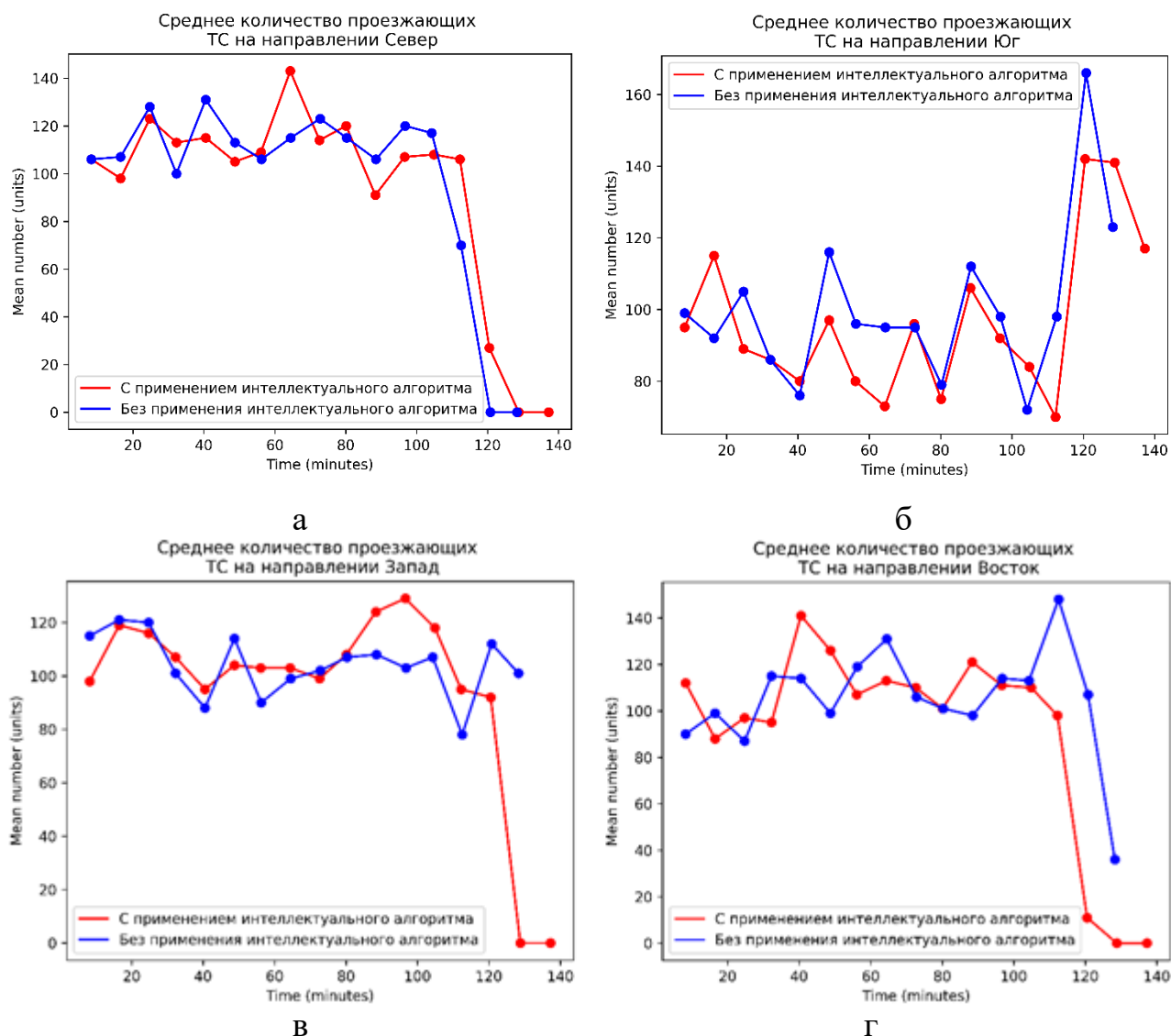


Рисунок 27 – Результаты моделирования для равномерного сценария (а – направление север, б – направление юг, в – направление восток, г – направление запад)

По графикам видно, что на всех направлениях, кроме направления юг, сохраняется равномерное распределение машин во времени около 80 – 120 автомобилей в период. Направление юг является наиболее нестабильным из-за накопления машин из предыдущих периодов.

На втором этапе проводилось тестирование алгоритма на ассиметричных сценариях моделирования. В первом случае задаётся повышенная нагрузка на отдельные временные интервалы. Конфигурация для генерации такого набора данных представлена на рисунке 28. Результаты моделирования показаны на рисунке 29.

Изм.	Лист	№ докум.	Подпись	Дата

Create simulation config

Общее количество машин:

5000

Распределение по периодам:

0.05

0.0

0.0

0.0

0.15

0.05

0.2

0.55

+

-

Соотношение направлений:

С

В

Ю

З

0.25

0.25

0.25

0.25

Отмена

Начать моделирование

Рисунок 28 – Конфигурация для неравномерного сценария

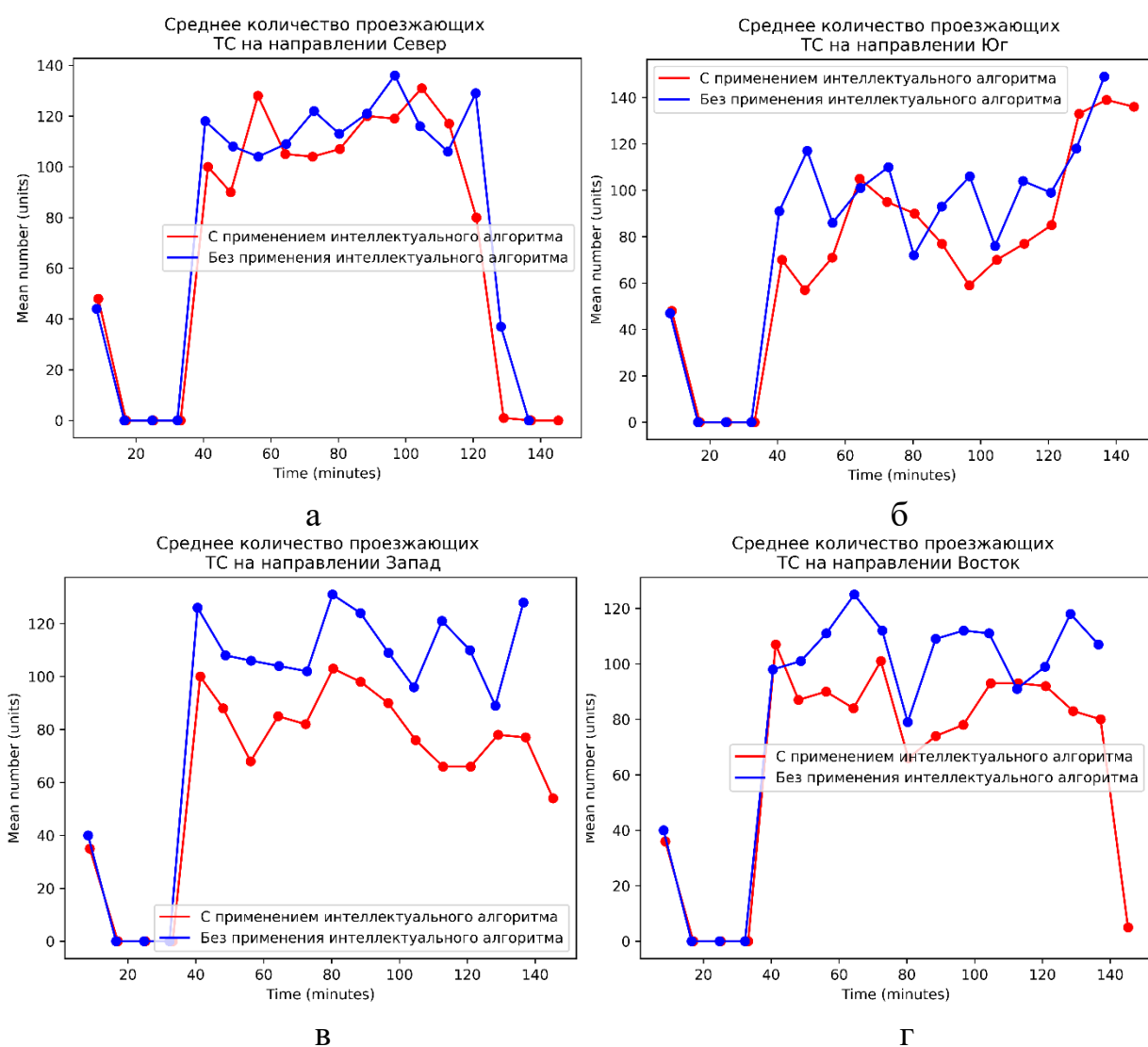


Рисунок 29 – Результаты моделирования для неравномерного сценария
(а – направление север, б – направление юг, в – направление восток, г –
направление запад)

По графикам видно, что на всех направлениях в промежутке 15-40 минут наблюдается уменьшение количества транспортных средств до нуля, что соответствует конфигурации моделирования.

Во втором случае задана увеличенная нагрузка на направление запад, как показано на рисунке 30. На остальных направлениях установлена минимальная загрузка.

The screenshot shows a configuration window titled "Create simulation config". It contains the following elements:

- Общее количество машин:** A large input field containing the value "1250".
- Распределение по периодам:** Five small input fields, each containing the value "0.2".
- Соотношение направлений:** Four input fields with values "0.05", "0.2", "0.05", and "0.7". Above these fields are labels "С", "В", "Ю", and "З" respectively.
- Buttons:** "Отмена" (Cancel) and "Начать моделирование" (Start simulation).

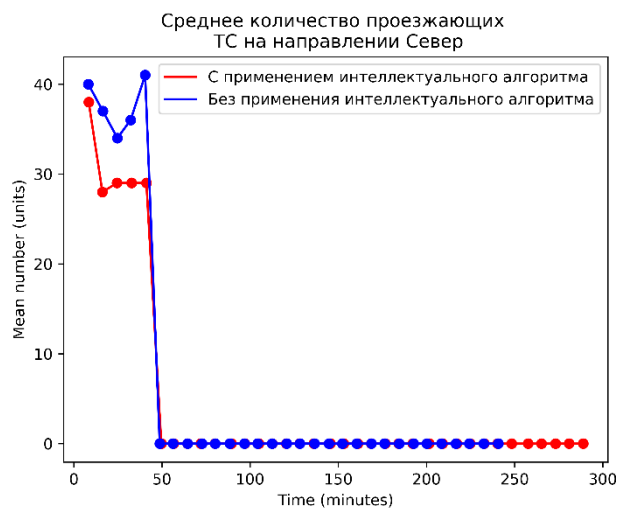
Рисунок 30 – Конфигурация для увеличенного трафика на направлении

На рисунке 31 показан результат моделирования в виде графиков зависимости количества машин с направления от времени моделирования.

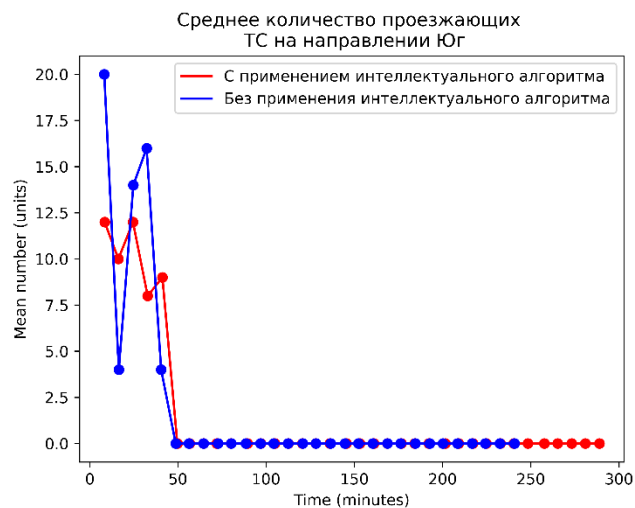
По графикам видно, что на направлении запад наблюдается увеличенное по сравнению с другими направлениями количество машин. Это подтверждает работоспособность алгоритма и его способность правильной работы даже при резкой несимметричной нагрузке с одного из направлений.

В результате тестирования алгоритма генерации данных для моделирования подтверждена его работоспособность в условиях различных параметров. Алгоритм корректно трансформирует значения, которые вводит пользователь, в параметры транспортного потока.

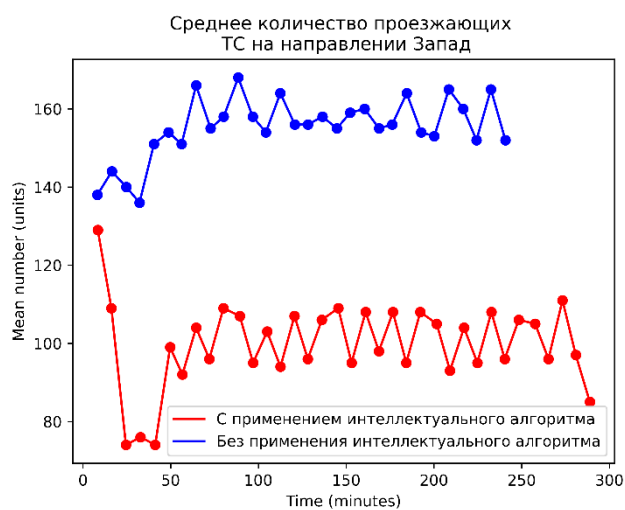
Таким образом, алгоритм может использоваться для создания сценариев моделирования, давая возможность генерации данных о трафике для тестирования интеллектуальных алгоритмов управления светофорами.



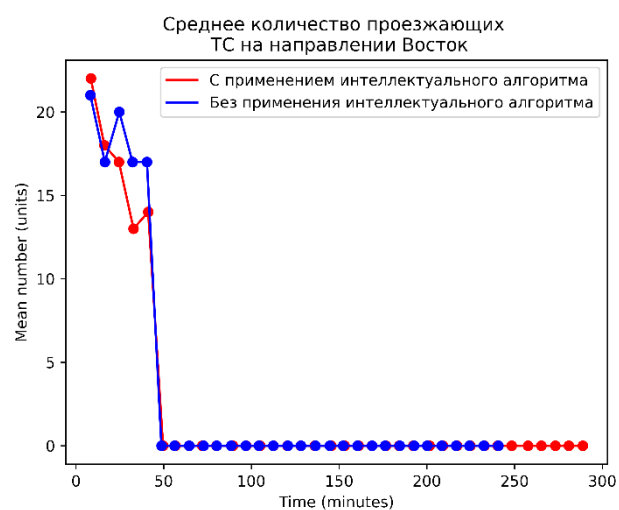
а



б



в



г

Рисунок 31 – Результаты моделирования для увеличенного трафика на направлении (а – направление север, б – направление юг, в – направление восток, г – направление запад)

2 Охрана труда

В данном разделе рассматриваются требования по обеспечению безопасности жизнедеятельности и охраны труда пользователей ПЭВМ, участвующих в разработке и эксплуатации программного обеспечения, а также способы минимизации пагубных факторов на здоровье.

2.1 Характеристика потенциальных рисков при работе с ПК

В соответствии с ГОСТ 12.0.003-2015 «Система стандартов безопасности труда. Электронно-вычислительная техника. Общие требования безопасности», при эксплуатации ПК выделяются следующие основные риски:

- риски поражения электрическим током из-за: неисправной проводки; отсутствия заземления; попадания жидкости на технику; короткого замыкания при перегрузке сети;
- риски заболеваний опорно-двигательной системы из-за: неправильного положения тела; неправильной высоты стула и стола;
- риски заболеваний зрения из-за: использования ПК без перерывов для глаз; неправильной яркости экрана и расположения монитора.

2.2 Оборудование рабочего места

Основные требования, которые должны соблюдаться при пользовании рабочем местом:

- расстояние от монитора до глаз должно быть не менее 500 – 700 мм. Монитор должен располагаться ниже уровня глаз под углом 15-20°;
- высота стола должна быть 680 – 800 мм (в зависимости от роста пользователя);
- кресло должно иметь регулируемую высоту сиденья и спинку, а также глубину сиденья 400 - 450 мм.

На рисунке 32 показана правильная организация рабочего пространства для ПК.

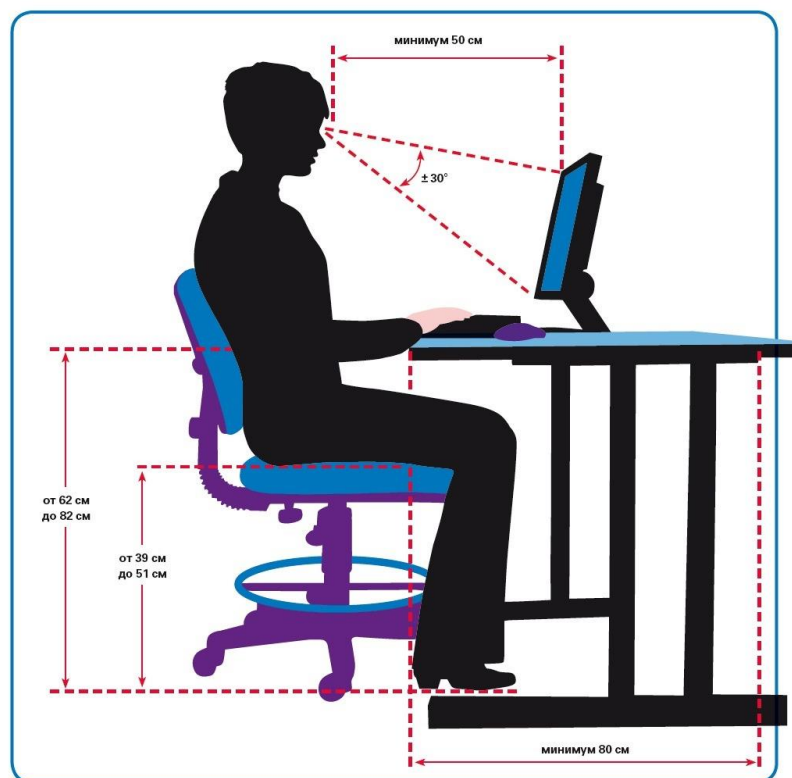


Рисунок 32 – Правильная организация рабочего места

Рабочее место с монитором должно гарантировать пользователю удобную работу с ПК в положении сидя. При нахождении за ПК должны выполняться следующие условия: прямая спина с опорой на спинку кресла; расслабленные плечи; кисти и запястья в прямом положении; ноги должны устойчиво стоять на полу.

2.3 Освещённость рабочего места

Для обеспечения здоровья органов зрения на рабочем месте должны соблюдаться стандарты освещения. Согласно санитарным правилам, общая освещённость рабочего места должна составлять не менее 400 люкс. Источники освещения должны обеспечивать равномерный рассеянный свет без бликов на мониторе. Цветовая температура источников освещения должна составлять 4000 – 5000К, что соответствует белому цвету. Недопустимо применение мерцающих источников света и расположение в поле зрения пользователя ярких источников света.

2.4 Электробезопасность

Электробезопасность при работе с ПК регламентируется ГОСТ 12.1.019-2017 «Электробезопасность. Исходя из стандарта, всё электрооборудование должно иметь защитное заземление. ГОСТ предусматривает использование только исправного оборудования с целой изоляцией и компонентами.

Кабели должны быть проложены в местах, где они не будут подвергаться механическим воздействиям и повреждениям. Стандарт требует, чтобы все электрические компоненты были защищены от случайного прикосновения, а корпуса электрооборудования были заземлены.

Пользователи оборудования должны быть обучены правилам электробезопасности и оказании первой помощи при поражении электрическим током. Требования стандарта включают наличие предупреждающих знаков и плакатов для предотвращения поражения током.

2.5 Пожарная безопасность

Основные риски пожарной безопасности связаны с возможным перегревом оборудования, короткими замыканиями и возгоранием компонентов. Для предотвращения опасных ситуаций необходимо поддерживать исправное состояние электрооборудования с средствами защиты. Особое внимание необходимо обратить на правильную организацию рабочего пространства - компьютеры должны располагаться на расстоянии не менее 0,5 метра от отопительных приборов и горючих материалов. В помещениях с компьютерами необходимо наличие средств пожаротушения.

Для повышения противопожарной безопасности необходимо соблюдать следующие правила: проходы и запасные выходы не должны быть перекрыты посторонними предметами; в здании должна быть установлена автоматическая пожарная сигнализация и планы эвакуации;

2.6 Санитарно-гигиеническая характеристика производства

Санитарно-гигиеническая характеристика производства регламентируется в соответствии с СанПиН 1.2.3685-21 и другими нормативными документами. Основными критериями являются: влажность и температура воздуха в помещении, количество света, скорость движения воздуха.

Рабочие столы должны располагаться не более, чем в 6 метрах от оконных проемов. Окна и рабочие места должны регулярно очищаться от пыли и загрязнений. Температурный режим в помещении должен поддерживаться на уровне 22 – 25 °С при влажности воздуха около 60 %. Необходимо наличие вентиляции или системы кондиционирования. Уровень шума не должен превышать 65 дБ.

В помещении необходимо обеспечить: доступ к питьевой воде; доступ к огнетушителям (1 единицу на 50 м³ помещения); аптечки первой помощи.

Все сотрудники должны проходить инструктажи по технике безопасности не реже 1 раза в год.

					ПГУ 1.090301.16.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		54

3 Стартап проект

Стартап проект направлен на разработку и исследование применения интеллектуальных алгоритмов управления светофорными объектами. Система будет адаптироваться к условиям дорожного трафика, тем самым оптимизируя движение на перекрёстке.

Целью деятельности стартапа является получение прибыли для улучшения благосостояния его участников и улучшение эффективности движения на перекрёстках за счёт применения современных технологий. Основной аудиторией являются государственные структуры и частные компании, занимающиеся разработкой решений в области ИТС.

3.1 Анализ рынка

Рынок ИТС в России за последние годы демонстрирует устойчивый рост. Этому способствуют исследования в этой области и федеральные программы.

По данным CNews, инвестиции в решения для транспорта с 2019 г. превысили 100 млрд ₽. Объём рынка цифровых транспортных решений к концу 2024 г. оценивается в 40–50 млрд ₽. В ходе национального проекта «Безопасные качественные дороги» в 2024 г. было выделено 3,15 млрд ₽ на внедрение ИТС в 62 городах в 56 субъектах России.

Факторы роста рынка ИТС и умных городов:

Увеличение числа транспортных средств требует внедрения интеллектуальных решений для управления дорожным движением.

Государственные программы, которые представляют собой инвестиции в транспортную инфраструктуру для повышения эффективности дорожного движения.

По прогнозам, к 2027–2029 гг. рынок российских ИТС будет расти ежегодно на 8–10 % за счёт инвестиции в ИТС проекты, развития инфраструктуры и внедрения интеллектуальных для прогнозирования трафика в реальном времени.

3.2 Целевая аудитория и каналы взаимодействия

Целевой аудиторией стартапа являются муниципальные органы власти, заинтересованные в улучшении транспортной инфраструктуры города; коммерческие организации, которые занимаются разработкой, внедрением или исследованием ИТС.

Ценовые предложения для каждой группы будут подбираться индивидуально, в зависимости от масштаба внедряемой системы и потребностей заказчика.

Для коммуникации с целевой аудиторией планируется использовать следующие каналы: участие в профильных выставках и конференциях; сотрудничество с органами власти; публикации в журналах и профессиональных сообществах; демонстрация возможностей системы.

К каналам взаимодействия также можно отнести рассылку коммерческих предложений, прямые встречи с заинтересованными лицами, на которых будет продемонстрированы возможности системы.

После заключения договора обязательным будет сопровождение внедрения системы. Команда будет заниматься настройкой и развёртыванием системы на перекрёстке.

Наличие плана внедрения и сопровождения продукта позволит обеспечить удовлетворённость клиентов и эффективное взаимодействие в будущем.

3.3 Источники доходов и расходы

Стартовый капитал стартапа составляет 300 000 Р, финансируемых ООО «ЦИФРАЛЕКТ». Первый контракт в 300 000 Р покрывает размер стартового капитала. По прогнозам, в первый год доход достигнет 1 000 000 Р при расходах на уровне 570 000 Р. Это приведёт к прибыли 430 000 Р.

Точка безубыточности достигается после заключения 2 – 3 контрактов. При средней выручке 250 000 – 300 000 с одного контракта критический уровень расходов будет покрыт. При 2 – 3 контрактах в год выручка составит около 900 000 Р, что даёт запас для дальнейшего развития.

Изм.	Лист	№ докум.	Подпись	Дата

Рентабельность собственного капитала в стартапе равна 75,4 %, а чистый оборотный капитал составляет около 100 000 Р. Потребность в дополнительном финансировании при текущих темпах реализации проектов отсутствует, но для ускорения масштабирования возможен привлечение до 500 000 Р.

Распределение расходов выглядит следующим образом: на разработку программного обеспечения планируется направить 250 000 Р, маркетинг — 100 000 Р, прочие операционные затраты — 220 000 Р. В результате, при 2 – 3 исполненных контрактах чистая прибыль составит порядка 330 000 Р.

Таким образом, при текущем темпе роста стартап выйдет на прибыль уже на 11-м месяце работы, обеспечивая 75,4 % годовой рентабельности без привлечения внешних инвестиций. При дополнительном финансировании развитие проекта может ускориться в два раза, что позволит быстрее расширить клиентскую базу и увеличить объёмы выручки.

3.4 Риски проекта

Стартап может столкнуться с некоторыми рисками, которые могут повлиять на его разработку и доходы. Ниже представлены некоторые из них.

Изменение ситуации на рынке. Возможны экономические спады, изменения приоритетов в финансировании программ развития ИТС и как следствия снижение спроса на ИСТ решения.

Меры реагирования: заключение долгосрочных контрактов с заказчиками, гибкость в ценообразовании и корректирование решений под разных клиентов.

Появление новых технологий. Быстрое развитие технологий может сделать продукт устаревшим или менее эффективным по сравнению с конкурентами.

Меры реагирования: постоянное обновление функционала программы, отслеживание развития технологий, проведение тестовых внедрений новых функций.

Деятельность конкурентов. Крупные компании могут предлагать более привлекательные по цене и функционалу решения.

Меры реагирования: создания долгосрочных B2G контрактов и подстраивание решения под нужды каждого клиента.

					ПГУ 1.090301.16.001 ПЗ	Лист
						57
Изм.	Лист	№ докум.	Подпись	Дата		

Недостаток финансирования. Ограниченные ресурсы могут замедлить развитию проекта.

Меры реагирования: привлечение инвестиций, оптимизация расходов и участие в конкурсах и программах.

3.5 Ключевые ресурсы

Для успешной реализации стартапа необходимо наличие нескольких ресурсов, которые обеспечивают разработку, сопровождение и масштабирование продукта. Ниже представлены некоторые из них.

Команда специалистов. Основой стартапа является профессиональная команда, которая включает в себя разработчиков ПО, специалистов в области машинного обучения и нейронных сетей, а также специалистов для работы с клиентами.

Вычислительные мощности. Для обучения моделей и обработки данных требуются вычислительные мощности с поддержкой тензорных вычислений. Эти ресурсы важны как при разработке, так и при сопровождении продукта.

Связь с партнёрами. Связи с государственными учреждениями, коммерческими и исследовательскими организациями упрощает внедрение системы и затраты на масштабирование.

Каналы коммуникаций. Продвижение стартапа возможно через профессиональные сообщества, выставки, соцсети, а также прямое взаимодействие с заинтересованными лицами.

Финансовые ресурсы. Базовое финансирование необходимо для покрытия первичных расходов и создания резерва. Возможность дополнительного финансирования в виде инвестиций и грантов повысит надёжность проекта.

Заключение

В ходе выполнения выпускной квалификационной работы были изучены методы имитационного моделирования транспортных потоков в среде моделирования SUMO. Особое внимание было уделено созданию пользовательского интерфейса для работы с приложением и повышения удобства взаимодействия с программой.

Получены навыки создания и интеграции пользовательского интерфейса с системой моделирования и интеллектуальными алгоритмами. Графический интерфейс реализован при помощи фреймворка kivu. Программа позволяет не только запускать моделирования и управлять их параметрами, но и визуализировать результаты в виде графиков, а также сохранять метрики в файлах csv. Разработанный интерфейс делает приложение доступным для пользователя без глубоких знаний в программировании.

В настоящее время данная программа используется в [22] для интеллектуального управления дорожным движением на перекрёстках города Пензы.

В дальнейшем планируется расширить возможности интерфейса, добавив поддержку новых алгоритмов, а также выбор гиперпараметров интеллектуальных алгоритмов.

Список использованных источников

1. А. Д. Сальникова, А. В. Баните, Д. Г. Плотников, А. С. Кашталинский. Формирование системы транспортно-пересадочных узлов в городской агломерации // Автоматика на транспорте. – 2023. – Т. 9, № 1. – С. 87–98.
2. И. В. Макаров, Р. Г. Хабибуллин, К. А. Шубенкова. Совершенствование управления транспортными потоками города с использованием имитационного моделирования // Имитационное моделирование. Теория и практика (ИММОД-2009): материалы 4-й Всероссийская научно-практическая конференция, Санкт-Петербург, 21–23 окт. 2009 г. – Санкт-Петербург, 2009.
3. R. A. Shaharuddin, M. Y. Misro. Controlling Traffic Congestion in Urbanised City: A Framework Using Agent-Based Modelling and Simulation Approach // ISPRS International Journal of Geo-Information. – 2023. – Vol. 12, № 6. – P. 226.
4. T. Alghamdi, S. Mostafi, S. Abdelkader, K. Elgazzar. Comparative Study on Traffic Modeling Techniques for Predicting and Simulating Traffic Behavior // Future Internet. – 2021. – Vol. 14. – P. 294.
5. P. Yedavalli, K. Kumar, P. Waddell. Microsimulation analysis for network traffic assignment (MANTA) at metropolitan-scale for agile transportation planning // Transportmetrica A: Transport Science. – 2021. – P. 1278–1299.
6. Т. А. Кузнецов. Оптимизация движения транспортного потока на регулируемом перекрестке с использованием имитационного моделирования // Политехнический молодежный журнал. – 2022. – № 7 (72).
7. Ю. И. Захаров, Е. С. Карнаух. Основные современные инструменты имитационного моделирования транспортных потоков // Вісник Придніпровської державної академії будівництва та архітектури. - 2023. - С. 87-98.
8. И. А. Саленек, Я. А. Селиверстов, С. А. Селиверстов, Н. И. Носкова. Оптимизация циклов светофорного регулирования методом обучения с подкреплением // Сборник научных трудов XXVI Международной научно-практической конференции. - Санкт-Петербург: Санкт-Петербургский политехнический университет Петра Великого, 2023. - 13-14 окт.

9. M. N. Abdullah, H. A. Jeiad, S. S. Hamid. Intelligent Traffic Lights Control Using Q-learning // International Journal of Science and Research (IJSR). 2018. – Vol. 7, № 9. – P. 365-369.

10. О. П. Тимофеева, С. С. Палицына. Обучение нейронной сети интеллектуальной системы управления городскими светофорами // Труды Нижегородского государственного технического университета им. Р.Е. Алексеева. - Нижний Новгород, 2016. - № 1(112).

11. Сетевое адаптивное управление светофорами: успешный кейс Пензы [Электронный ресурс] – URL: <https://www.itsjournal.ru/articles/digital-region/setevoe-adaptivnoe-upravlenie-svetoforami-ushpeshnyy-keys-penzy> (дата обращения: 13.05.2025).

12. AVEDEX [Электронный ресурс] – URL: <https://mallenom-tech.ru/> (дата обращения: 25.03.2025).

13. Авангард [Электронный ресурс] – URL: <https://fvf-rbs.ru/> (дата обращения: 25.03.2025).

14. Sentinel Smart Traffic [Электронный ресурс] – URL: <https://complitech.org/> (дата обращения: 25.03.2025).

15. Умный перекрёсток [Электронный ресурс] – URL: <https://www.spetsdor.ru/> (дата обращения: 25.03.2025).

16. Tkinter [Электронный ресурс] – URL: <https://docs.python.org/3/library/tkinter.html> (дата обращения 26.03.2025).

17. PyQt [Электронный ресурс]– URL: <https://wiki.python.org/moin/PyQt> (дата обращения 26.03.2025).

18. wxPython [Электронный ресурс] – URL: <https://wxpython.org/> (дата обращения 26.03.2025).

19. BeeWare Toga [Электронный ресурс] – URL: <https://beeware.org/project/toga/> (дата обращения 26.03.2025).

20. Kivy [Электронный ресурс]– URL: <https://kivy.org/> (дата обращения 26.03.2025).

21. М. А. Митрохин, А. О. Аляев, Р. И. Лобанов, В. Е. Кирьянов.
Моделирование локальных транспортных потоков в условиях неполноты
исходных данных // Автоматика на транспорте. – 2023. – № 9(4).

22. Нейросетевая видеоаналитика автомобильного трафика [Электронный
ресурс] – URL: <https://traffic.codeinside.ru> (дата обращения 13.05.2025).

					ПГУ 1.090301.16.001 ПЗ	Лист
						62
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А

(Обязательное)

Программа моделирования интеллектуального управления дорожным

трафиком

UML-диаграммы

					ПГУ 1.090301.16.001 ПЗ	Лист
						63
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 - Диаграмма вариантов использования



ПГУ 1.090301.01 002.Д13									
Изм.	Лист	Изм.	Лист	Изм.	Лист	Программа интеллектуальной автоматизированной системы управления дорожным движением. Планово-оперативное управление УДС – диспетчеризация движения и контроль за состоянием			
Разработчик	Александр А.Д.	Проверен		Дата		Лист	Место	Масштаб	
Проверен	Морозов М.А.								
Г. изд.						Лист	Листа		
Разработчик	Бонин А.С.					ФБТ, стр. 2188881			
И. изд.	Морозов М.А.								
Уч.									

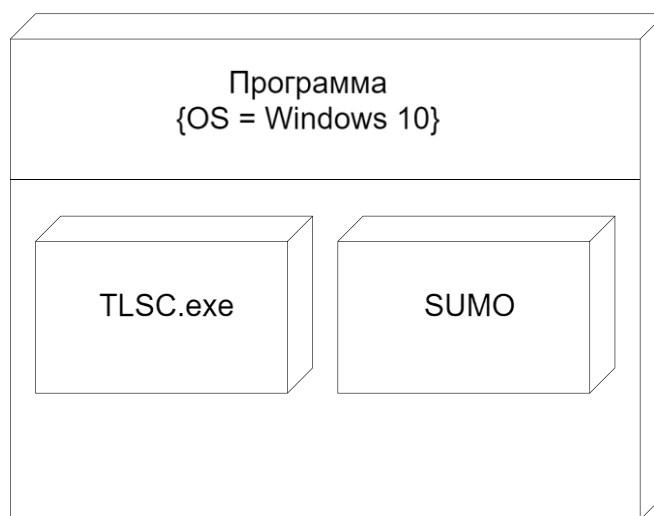
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.16.001 ПЗ

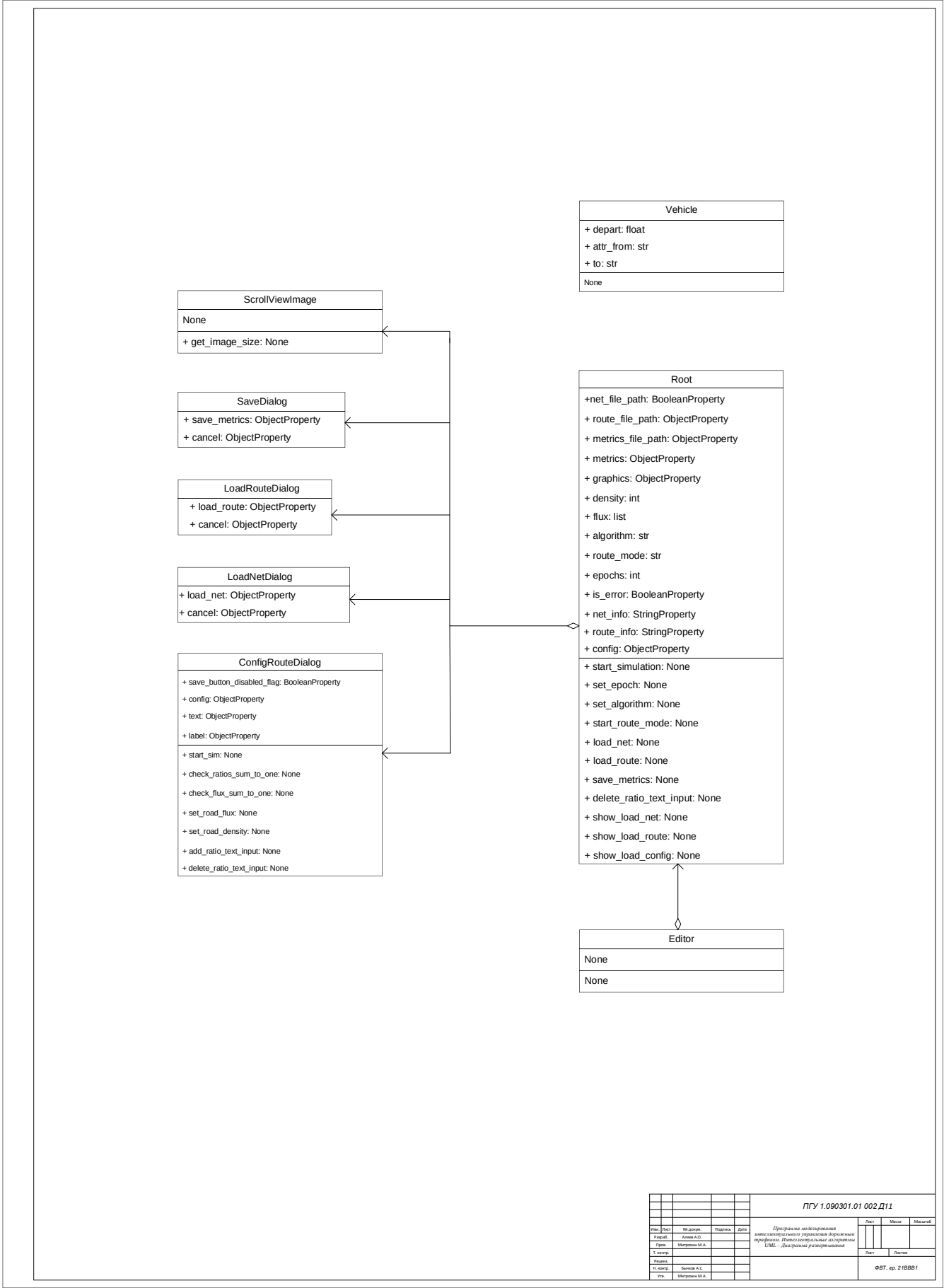
Лист

64

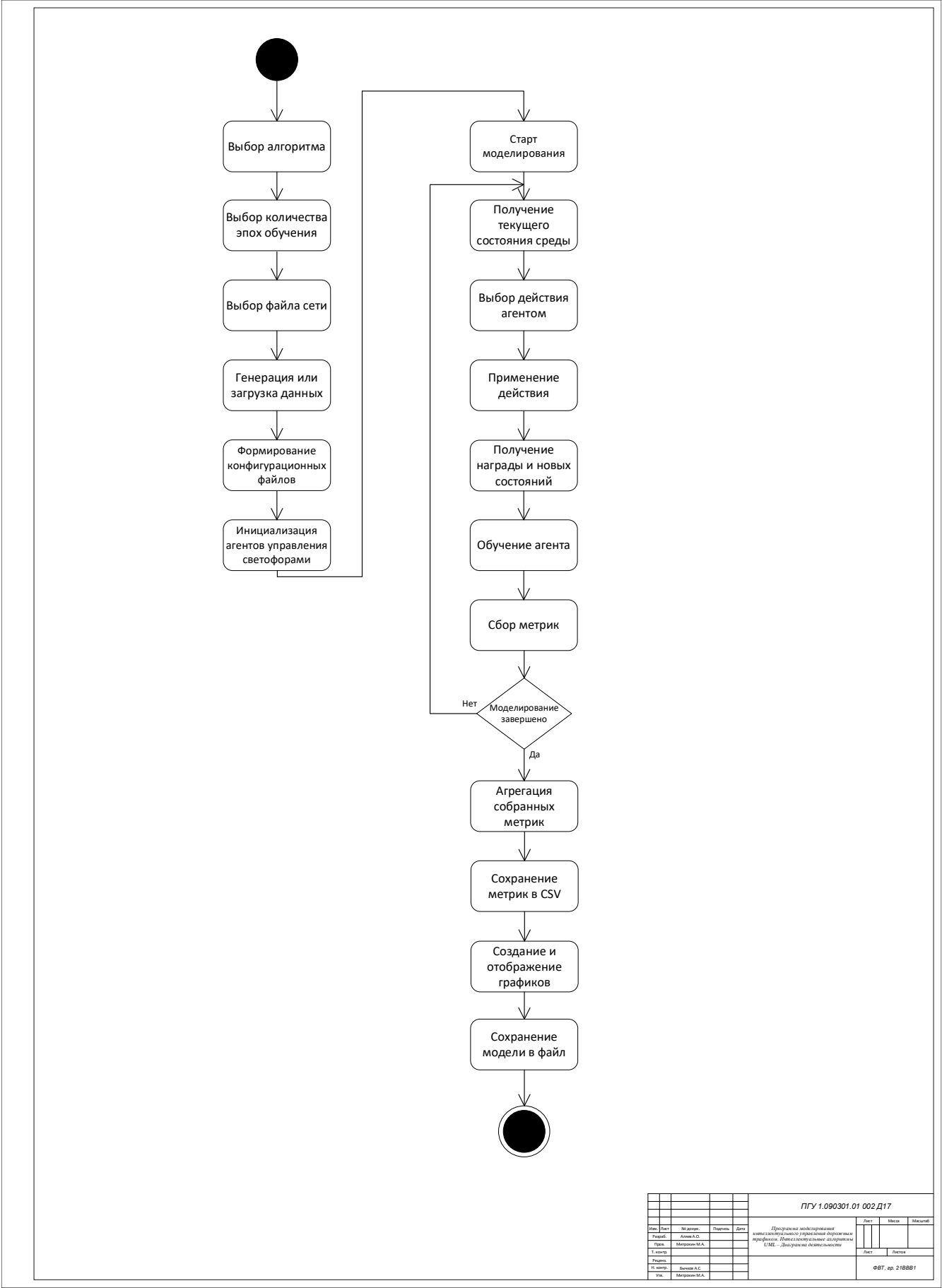
Приложение А.2 - Диаграмма развёртывания

[illegible]

Приложение А.3 - Диаграмма классов



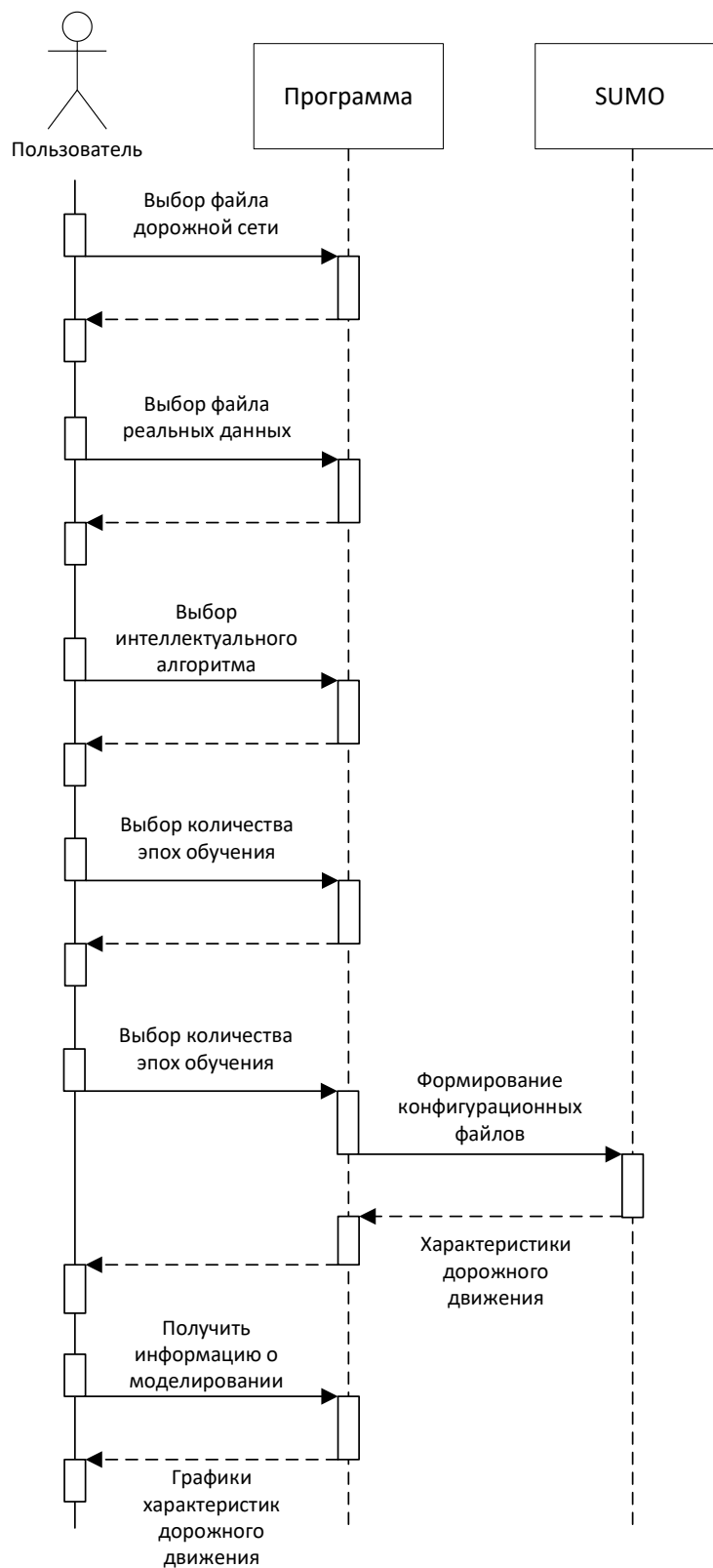
Приложение А.4 - Диаграмма деятельности



ПГУ 1.090301.01 002 Д17									
Имя	Пост	№ докум.	Параметр	Дата	Приложение к методическим указаниям по выполнению лабораторных работ по дисциплине «Интеллектуальные системы управления движением транспортных средств»				
Фамилия	Имя	Отчество	Подпись	Дата	ФБТ, стр. 218881				
Фамилия	Имя	Отчество	Подпись	Дата					
Фамилия	Имя	Отчество	Подпись	Дата					
Фамилия	Имя	Отчество	Подпись	Дата					
Фамилия	Имя	Отчество	Подпись	Дата					
Фамилия	Имя	Отчество	Подпись	Дата					
Фамилия	Имя	Отчество	Подпись	Дата					
Фамилия	Имя	Отчество	Подпись	Дата					

Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.3 - Диаграмма последовательности

[illegible]

Приложение Б

(Обязательное)

Программа моделирования интеллектуального управления дорожным
трафиком

Листинг программы

					ПГУ 1.090301.16.001 ПЗ	Лист
						69
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «main.py»

```

from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.floatlayout import FloatLayout
from kivy.factory import Factory
from kivy.properties import ObjectProperty
from kivy.properties import BooleanProperty
from kivy.properties import StringProperty
from kivy.uix.popup import Popup
from threading import Thread
from kivy.uix.image import Image
import sim_main
import os
import shutil
from kivy.core.window import Window
from kivy.uix.textinput import TextInput
from kivy.config import Config

Window.fullscreen = False

class LoadNetDialog(FloatLayout):
    load_net = ObjectProperty(None)
    cancel = ObjectProperty(None)

class ConfigRouteDialog(BoxLayout):
    save_button_disabled_flag = BooleanProperty(True)
    config = ObjectProperty(None)
    text = ObjectProperty(None)
    label = ObjectProperty(None)

    start_simulation = ObjectProperty(None)
    load_density = ObjectProperty(None)
    load_road_ratio = ObjectProperty(None)
    load_road_flux = ObjectProperty(None)

    cancel = ObjectProperty(None)

    def start_sim(self):
        self.start_simulation()
        self.cancel()

    def check_ratios_sum_to_one(self):
        return sum(self.config['ratio']) == 1

    def check_flux_sum_to_one(self):
        try:
            values = [float(x) if x.strip() else 0.0 for x in
self.config['flux']] # Конвертация во float
            return sum(values) == 1
        except Exception:
            return False # Если он ловит неконвертируемое

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.16.001 ПЗ

Лист

70

```

def set_road_flux(self):
    self.load_road_flux((self.ids.text_input_north_flux.text,
                        self.ids.text_input_east_flux.text,
                        self.ids.text_input_south_flux.text,
                        self.ids.text_input_west_flux.text,
                        ))

    self.save_button_disabled_flag = not
(self.check_ratios_sum_to_one() and self.check_flux_sum_to_one())

def set_density(self):
    self.load_density(self.ids.text_input_density.text)

def set_road_ratio(self, _, __):
    ratio = []
    for text_input in self.ids.ratio_box_layout.children:
        text = text_input.text
        if text.replace('.', '', 1).isdigit():
            ratio.append(float(text))
        else:
            ratio.append(0.0)

    ratio.reverse()
    self.config['ratio'] = ratio
    self.load_road_ratio(ratio)

    self.save_button_disabled_flag = not
(self.check_ratios_sum_to_one() and self.check_flux_sum_to_one())

def get_ratio(self, box_layout, text_input):
    for i in range(len(box_layout.children)):
        if text_input == box_layout.children[i]:
            return self.config['ratio'][i]

def add_ratio_text_input(self, box_layout):
    self.config['ratio'].append(0)
    text_input = TextInput(
        text=str(self.config['ratio'][len(self.config['ratio'])
- 1]),
        multiline="False",
    )
    text_input.bind(text=self.set_road_ratio)
    box_layout.add_widget(text_input)

def set_config_ratio(self, config):
    self.ids.ratio_box_layout.clear_widgets()
    for i in range(len(self.config['ratio'])):
        text_input = TextInput(
            text=str(config['ratio'][i]),
            multiline="False"
        )

```

```

text_input.bind(text=self.set_road_ratio)
                self.ids.ratio_box_layout.add_widget(text_input)
                self.save_button_disabled_flag =
not                (self.check_ratios_sum_to_one()                                and
self.check_flux_sum_to_one())
def delete_ratio_text_input(self, ratio_box_layout):
    ratio_box_layout.remove_widget(
        ratio_box_layout.children[0]
    )
    del self.config['ratio'][-1]
    self.load_road_ratio(self.config['ratio'])

    self.save_button_disabled_flag = not
(self.check_ratios_sum_to_one() and self.check_flux_sum_to_one())

class LoadRouteDialog(FloatLayout):
    load_route = ObjectProperty(None)
    cancel = ObjectProperty(None)

class SaveDialog(FloatLayout):
    save_metrics = ObjectProperty(None)
    cancel = ObjectProperty(None)
    file_name = ObjectProperty(None)
    algorithm = StringProperty(None)

    def save(self, save_metrics, file_name):
        if self.algorithm == "Q-learning":
            full_path = str(save_metrics + "/" + file_name +
".pkl")
            shutil.copy2('database/dataDWT.pkl', full_path)
        elif self.algorithm == "Neural Net":
            full_path = str(save_metrics + "/" + file_name +
".pth")
            shutil.copy2('database/dataQ.pth', full_path)
        self.cancel()

class Root(FloatLayout):
    net_file_path = ObjectProperty(None)
    route_file_path = ObjectProperty(None)
    metrics_file_path = ObjectProperty(None)
    metrics = ObjectProperty(None)
    graphics = ObjectProperty(None)
    density = 0 # Обращаться по self.density
    ratio = [1]
    flux = [0, 0, 0, 0]
    algorithm = "Q-learning"
    route_mode = "Flows"
    epochs = 1
    is_modeling = BooleanProperty(False) # для статусов
    is_error = BooleanProperty(False) # для ошибок
    net_info = StringProperty("Файл сети: ---") # для загрузки
информации о выбранном файле сети
    route_info = StringProperty("Файл маршрутов: ---") # для
загрузки информации о выбранном файле маршрутов

```

Изм.	Лист	№ докум.	Подпись	Дата

```

config = ObjectProperty({'density': density,
                        'ratio': ratio,
                        'flux': flux}) # Обращаться по
self.density`
sim_inf = ObjectProperty(None)

def set_epochs(self, text):
    try:
        value = int(text)
        if value > 0:
            self.epochs = value
            print(f"[INFO] Количество эпох установлено:
{self.epochs}")
        else:
            value = 1
    except Exception:
        return False

def set_route_mode(self, mode):
    self.route_mode = mode

def start_simulation(self):
    if self.route_mode == 'Flows':
        self.sim_inf = {
            'mode': self.route_mode,
            'algorithm': self.algorithm,
            'route_path': self.route_file_path,
            'net_path': self.net_file_path,
            'epochs': self.epochs
        }
    elif self.route_mode == 'generate':
        self.sim_inf = {
            'mode': self.route_mode,
            'algorithm': self.algorithm,
            'config': self.config,
            'net_path': self.net_file_path,
            'epochs': self.epochs
        }

def run():
    self.is_modeling = True
    self.graphics = []

    try:
        self.graphics = sim_main.sim_main(self.sim_inf)
    except Exception as e:
        print(f"[Ошибка]: {e}")
        self.is_error = True
        self.status_text = "Статус: Ошибка моделирования!"
        self.is_modeling = False
        raise

    if self.is_error:
        return

```

```

# только если всё прошло успешно:
    for img in os.listdir("graph/"):
        self.graphics.append("graph/" + str(img))

    self.is_modeling = False

    Thread(target=run, daemon=True).start()

def set_algorithm(self, value):
    self.algorithm = value
    print(f"Выбран алгоритм: {self.algorithm}")

def load_road_ratio(self, ratio):
    self.ratio = ratio
    self.config['ratio'] = ratio

def load_road_flux(self, flux):
    self.flux = flux
    self.config['flux'] = flux

def load_density(self, density):
    if str.isdecimal(density):
        self.density = density
        self.config['density'] = density
    else:
        self.density = 'Введите число'
        self.config['density'] = density

def dismiss_popup(self):
    self._popup.dismiss()

def show_load_net(self):
    content = LoadNetDialog(load_net=self.load_net,
cancel=self.dismiss_popup)
    self._popup = Popup(title="Load net file", content=content,
        size_hint=(0.9, 0.9))
    self._popup.open()

def show_load_route(self):
    content = LoadRouteDialog(load_route=self.load_route,
cancel=self.dismiss_popup)
    self._popup = Popup(title="Load route file", content=content,
        size_hint=(0.9, 0.9))
    self._popup.open()

def show_route_config(self):
    content = ConfigRouteDialog(config=self.config,
        load_density=self.load_density,

load_road_ratio=self.load_road_ratio,
        load_road_flux=self.load_road_flux,

start_simulation=self.start_simulation,
        cancel=self.dismiss_popup, )
    content.set_config_ratio(self.config)

```

```

self._popup = Popup(title="Load route file", content=content,
                    size_hint=(0.9, 0.9))
self._popup.open()

def show_save(self):
    content = SaveDialog(save_metrics=self.save_metrics,
cancel=self.dismiss_popup, algorithm=self.algorithm)
    self._popup = Popup(title="Save file", content=content,
                    size_hint=(0.9, 0.9))
    self._popup.open()

def show_metrics(self, scrollViewBox):
    if self.graphics is not None:
        scrollViewBox.clear_widgets()
        scrollViewBox.height = 0
        for graph in self.graphics:
            img = ScrollViewImage(source=graph, height=500)
            scrollViewBox.add_widget(img)
            scrollViewBox.height += img.height - 20 # Space before
scroll WHY????
            # content = MetricsDialog(metrics = self.metrics, graphics =
self.graphics, cancel = self.dismiss_popup)
            # self._popup = Popup(title="Metrics and graphics",
content=content,
            # size_hint=(0.9, 0.9))
            # self._popup.open()

def change_label(self, root, text):
    root.ids.label.text = text

def load_net(self, path, filename):
    self.net_file_path = filename
    self.net_info = f"Файл сети: {os.path.basename(filename[0])}"
    self.dismiss_popup()

def load_route(self, path, filename):
    self.route_file_path = filename
    self.route_info = f"Файл маршрутов:
{os.path.basename(filename[0])}"
    self.dismiss_popup()

def save_metrics(self, path, filename):
    with open(os.path.join(path, filename), 'w') as stream:
        stream.write(self.text_input.text)

    self.dismiss_popup()

class ScrollViewImage(Image):
    def get_image_size(self):
        return 300, 300

class Editor(App):
    pass

```

```

Factory.register('Root', cls=Root)
Factory.register('LoadNetDialog', cls=LoadNetDialog)
Factory.register('LoadRouteDialog', cls=LoadRouteDialog)
Factory.register('ConfigRouteDialog', cls=ConfigRouteDialog)
Factory.register('SaveDialog', cls=SaveDialog)
# Factory.register('MetricsDialog', cls=MetricsDialog)

if __name__ == '__main__':
    Editor().run()

```

Файл «editor.kv»

```

Root:
    id: main_root
    orientation: 'vertical'
    padding: 10
    spacing: 10
    canvas.before:
        Color:
            rgba: 0.96, 0.96, 0.96, 1
        Rectangle:
            pos: self.pos
            size: self.size

    BoxLayout:
        id: left_menu
        orientation: 'horizontal'
        spacing: 15

        BoxLayout:
            orientation: 'vertical'
            size_hint_x: 0.4
            padding: 10
            spacing: 10
            canvas.before:
                Color:
                    rgba: 1, 1, 1, 1
                RoundedRectangle:
                    pos: self.pos
                    size: self.size
                    radius: [10]

            Label:
                text: "Действия"
                font_size: '20sp'
                color: 0.15, 0.15, 0.2, 1
                bold: True
                size_hint_y: None
                height: 40

            Button:
                text: 'Загрузить сеть'
                on_release: root.show_load_net()

```

```

Button:
    text: 'Загрузить маршрут'
    on_release: root.show_load_route()
Button:
    text: 'Начать моделирование \n с маршрутами из файла'
    halign: 'center'
    valign: 'center'
    on_release:
        root.set_route_mode('Flows')
        root.start_simulation()
Button:
    text: 'Начать моделирование \n с синтезированными \n данными'
    halign: 'center'
    valign: 'center'
    on_release:
        root.set_route_mode('generate')
        root.show_route_config()
Button:
    text: 'Показать графики'
    on_release: root.show_metrics(scrollViewBox)
Button:
    text: 'Сохранить результаты \n обучения'
    halign: 'center'
    valign: 'center'
    on_release: root.show_save()
BoxLayout:
    id: labels_layout
    orientation: 'vertical'
    padding: 10
    spacing: 10
    canvas.before:
        Color:
            rgba: 1, 1, 1, 1
        RoundedRectangle:
            pos: self.pos
            size: self.size
            radius: [10]

    Label:
        text: "Traffic Light Control System"
        font_size: '20sp'
        bold: True
        color: 0.15, 0.15, 0.2, 1
        size_hint_y: None
        height: "40dp"

    Label:
        id: status_label
        text: "Статус: Моделирование" if root.is_modeling else
("Статус: Ошибка моделирования!" if root.is_error else "Статус: Готов
к работе")
        color: (0.9, 0.54, 0.15, 1) if root.is_modeling else (0.93,
0.25, 0.13, 1) if root.is_error else (0.2, 0.8, 0.2, 1)

```

```

        font_size: '16sp'
        size_hint_y: None
        height: "30dp"

AnchorLayout:
    anchor_x: 'center'
    anchor_y: 'center'
    size_hint_y: None
    height: "20dp"

    Spinner:
        text: root.algorithm
        values: ["Q-learning", "Neural Net"]
        on_text: root.set_algorithm(self.text)
        size_hint: None, None
        size: "100dp", "20dp"
        background_normal: ''
        background_color: (0.95, 0.95, 0.95, 1)
        color: 0.1, 0.1, 0.1, 1

AnchorLayout:
    anchor_x: 'center'
    anchor_y: 'center'
    size_hint_y: None
    height: "30dp"

    BoxLayout:
        orientation: 'horizontal'
        size_hint: None, None
        size: "150dp", "26dp"
        spacing: 10

        Label:
            text: "Эпохи:"
            size_hint_x: None
            width: "60dp"
            font_size: '14sp'
            color: 0.2, 0.2, 0.2, 1

        TextInput:
            id: epoch_input
            text: str(root.epochs)
            size_hint: None, None
            size: "60dp", "26dp"
            multiline: False
            input_filter: 'int'
            font_size: '14sp'
            on_text: root.set_epochs(self.text)

Label:
    id: info_label
    text: root.net_info
    font_size: '14sp'
    color: 0.2, 0.2, 0.2, 1
    size_hint_y: None
    height: "10dp"

```

```

Label:
    id: info_label
    text: root.route_info
    font_size: '14sp'
    color: 0.2, 0.2, 0.2, 1
    size_hint_y: None
    height: "10dp"

ScrollView:
    id: scrollView
    do_scroll: (True, True) #up and down
    size_hint: (1, 1)
    BoxLayout:
        orientation : 'vertical'
        size_hint_y : None
        id: scrollViewBox
        height: self.minimum_height
<ScrollViewImage>:
    height : 500
    source: 'images/notebook.png'
    size_hint_y: None

<PopupHeader@Label>:
    size_hint_y: None
    height: 35
    font_size: '16sp'
    bold: True
    color: 0.1, 0.1, 0.1, 1
    canvas.before:
        Color:
            rgba: 0.95, 0.95, 0.95, 1
        Rectangle:
            pos: self.pos
            size: self.size
<LoadNetDialog>:
    BoxLayout:
        orientation: "vertical"

        size: root.size
        pos: root.pos

        PopupHeader:
            text: "Загрузка файла сети"

        FileChooserIconView:
            id: filechooser
BoxLayout:
    size_hint_y: None
    height: 50
    spacing: 12
    Button:

```

```

text: "Отмена"
        on_release: root.cancel()
        background_color: 0.9, 0.3, 0.3, 1
    Button:
        text: "Загрузить"
        on_release: root.load_net(filechooser.path,
filechooser.selection)
        background_color: 0.3, 0.7, 0.3, 1

<LoadRouteDialog>:
    BoxLayout:
        orientation: "vertical"

        size: root.size
        pos: root.pos

        PopupHeader:
            text: "Загрузка файла маршрутов"
FileChooserIconView:
    id: filechooser

BoxLayout:
    size_hint_y: None
    height: 50
    spacing: 12
    Button:
        text: "Отмена"
        on_release: root.cancel()
        background_color: 0.9, 0.3, 0.3, 1
    Button:
        text: "Загрузить"
        on_release: root.load_route(filechooser.path,
filechooser.selection)
        background_color: 0.3, 0.7, 0.3, 1
<ConfigRouteDialog>:
    BoxLayout:
        orientation: 'vertical'
        size_hint: (0.8, 0.8)
        pos_hint: {'center_x': 0.5, 'center_y': 0.5}
        spacing: 10
        padding: 10

        GridLayout:
            cols: 2
            rows: 3
            spacing: 5
            size_hint_y: 0.8

Label:
    text: "Общее количество машин:"
    size_hint_x: 0.5
    halign: 'left'
    valign: 'middle'
    text_size: self.width, None

```

Изм.	Лист	№ докум.	Подпись	Дата

```

TextInput:
    id: text_input_density
    size_hint_x: 0.3
    height: 30
    text: str(root.config['density'])
    multiline: False
    input_filter: 'int'
    on_text: root.set_density()

Label:
    text: "Распределение по периодам:"
    size_hint_x: 0.5
    halign: 'left'
    valign: 'middle'
    text_size: self.width, None
BoxLayout:
    orientation: 'vertical'
    size_hint_x: 0.5
    spacing: 5

    BoxLayout:
        id: ratio_box_layout
        orientation: 'horizontal'
        spacing: 5
        size_hint_y: 0.7

    BoxLayout:
        orientation: 'horizontal'
        spacing: 5
        size_hint_y: 0.3
        Button:
            text: '+'
            size_hint_x: 0.5
            on_release: root.add_ratio_text_input(ratio_box_layout)

Button:
    id: remove_period_btn
    text: '-'
    size_hint_x: 0.5
    disabled: len(ratio_box_layout.children) <= 1
    on_release: root.delete_ratio_text_input(ratio_box_layout)

Label:
    text: "Соотношение направлений:"
    size_hint_x: 0.5
    halign: 'left'
    valign: 'middle'
    text_size: self.width, None
GridLayout:
    cols: 4
    rows: 2
    spacing: 5
    size_hint_x: 0.5
Label:
    text: "C"

```

```

Label:
    text: "B"
Label:
    text: "Ю"
Label:
    text: "3"

TextInput:
    id: text_input_north_flux
    size_hint_x: 0.25
    height: 30
    text: str(root.config['flux'][0])
    multiline: False
    input_filter: 'float'
    on_text: root.set_road_flux()
TextInput:
    id: text_input_east_flux
    size_hint_x: 0.25
    height: 30
    text: str(root.config['flux'][1])
    multiline: False
    input_filter: 'float'
    on_text: root.set_road_flux()
TextInput:
    id: text_input_south_flux
    size_hint_x: 0.25
    height: 30
    text: str(root.config['flux'][2])
    multiline: False
    input_filter: 'float'
    on_text: root.set_road_flux()
TextInput:
    id: text_input_west_flux
    size_hint_x: 0.25
    height: 30
    text: str(root.config['flux'][3])
    multiline: False
    input_filter: 'float'
    on_text: root.set_road_flux()

BoxLayout:
    orientation: 'horizontal'
    size_hint_y: 0.1
    spacing: 20
    padding: [0, 10, 0, 0]
Button:
    text: "Отмена"
    size_hint_x: 0.4
    on_release: root.cancel()
Button:
    id: save_button
    text: "Начать моделирование"

```

```

        size_hint_x: 0.4
        disabled: root.save_button_disabled_flag
        on_release: root.start_sim()

<SaveDialog>:
    text_input: text_input
    BoxLayout:
        size: root.size
        pos: root.pos
        orientation: "vertical"
        padding: 20
        spacing: 12
    PopupHeader:
        text: "Сохранение данных"

FileChooserIconView:
    id: filechooser
    on_selection: text_input.text = self.selection and
self.selection[0] or ''

TextInput:
    id: text_input
    size_hint_y: None
    height: 35
    multiline: False
    hint_text: "Имя файла для сохранения (без расширения)"
BoxLayout:
    size_hint_y: None
    height: 50
    spacing: 12
    Button:
        text: "Отмена"
        on_release: root.cancel()
        background_color: 0.9, 0.3, 0.3, 1
    Button:
        text: "Сохранить"
        on_release: root.save(filechooser.path, text_input.text)
        background_color: 0.3, 0.7, 0.3, 1

```

Файл «sim_main.py»

```

import time
import Q_Agent
import NN_Agent
import graph
import routeChange
import junction_config
from datetime import datetime, timedelta
import os
import traci as traci
import numpy as np
import csv

```

```

class Vehicle:
    vehicle_type = ""
    start = ""
    finish = ""
    event_time = 0
    start_road = ""
    finish_road = ""
    accumulating_time = 0.0

    def __init__(self, vehicle_type, start, finish, event_time,
start_road, finish_road, accumulating_time):
        self.vehicle_type = vehicle_type
        self.start = start
        self.finish = finish
        self.event_time = event_time
        self.start_road = start_road
        self.finish_road = finish_road
        self.accumulating_time = accumulating_time

def calculate_waiting_time():
    edges = ["A1B1", "B2B1", "C1B1", "B0B1"]
    waiting_list = []

    for edge in edges:
        vehicle_ids = traci.edge.getLastStepVehicleIDs(edge)
        if vehicle_ids:
            waiting_time = traci.edge.getWaitingTime(edge) /
len(vehicle_ids)
            else:
                waiting_time = 0
            waiting_list.append(waiting_time)

    return waiting_list

def count_cars_from_to_direct(dict_vehicle_list,
from_to_directions_dict, period, period_count):
    for junction in dict_vehicle_list:
        for junction1 in from_to_directions_dict.keys():
            if junction1 == junction:
                vehicle_list = dict_vehicle_list.get(junction)
                from_to_direction =
from_to_directions_dict.get(junction1)
                for vehicle in vehicle_list:
                    to_direction =
from_to_direction.get(vehicle.start)
                    time = int(vehicle.event_time // period)
                    if not to_direction.get(vehicle.finish):
                        to_direction[vehicle.finish] = [0 for _ in
range(period_count)]
                    to_direction[vehicle.finish][time] += 1

```

```

def run_rl_mode(period, epoch, initial_green, net_path, algorithm):
    branches_passed = 0
    filenames = []
    for attempt in range(epoch * 2):
        agent_is_on = (attempt % 2 == 0)
        if agent_is_on:
            if algorithm == "Q-learning":
                env_agent_on, initial_green = Q_Agent.sumoRl(period,
0.5, 0.6, initial_green, agent_is_on, net_path)
            elif algorithm == "Neural Net":
                env_agent_on, initial_green = NN_Agent.sumoRl(period,
0.5, 0.6, initial_green, agent_is_on, net_path)

            else:
                #env_agent_off, initial_green = NN_Agent.sumoRl(period,
0.5, 0.6, initial_green, agent_is_on, net_path)
                initial_green = {"B1": [45, 45]}
                env_agent_off, initial_green = Q_Agent.sumoRl(period,
0.5, 0.6, initial_green, agent_is_on, net_path)

        traci.close()
        branches_passed += 1

        if branches_passed == 2:
            linear_filenames = []
            filenames = graph.graph(env_agent_on, env_agent_off,
attempt // 2)
            for i in range(len(filenames)):
                for j in filenames[i]:
                    linear_filenames.append(j)
            env_agent_on = None
            env_agent_off = None
            branches_passed = 0

    return linear_filenames

def sim_main(sim_inf):
    period = 480
    mode = '1Junction'
    direct_count = 4
    data_type = "dl"

    try:
        names_list, directions_dict, edges_dict, junction_count =
junction_config.get_junctions_config(
            "junction_config.ini")
        if sim_inf['mode'] == "Flows":
            os.system(
                f"duarouter --flow-files={sim_inf['route_path'][0]} --
net-file=1Junction.net.xml "
                "--output-file=grid.rou.xml --begin 0 --end 100000000"
            )
        elif sim_inf['mode'] == 'generate':
            cars = int(sim_inf['config']['density'])

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.16.001 ПЗ

Лист

85

```

ratio = [float(x) for x in sim_inf['config']['flux']] # !!!
flux_list = sim_inf['config']['ratio'] # !!!
ways = routeChange.generate_ways()
routeChange.flux_to_route(ratio, flux_list, period, ways, cars)
os.system(
    "duarouter --flow-files=flows.xml --net-file=1Junction.net.xml
"
    "--output-file=grid.rou.xml --begin 0 --end 100000000"
)
start = time.time()
epoch = 1
initial_green = {"B1": [45, 45]}
linear_filenames = run_rl_mode(period, sim_inf["epochs"],
initial_green, sim_inf['net_path'], sim_inf['algorithm'])

end = time.time()
print(str(end - start))
except Exception as e:
    print(f"[Ошибка]: {e}")
    raise
return linear_filenames

```

Файл «graph.py»

```

import matplotlib
from matplotlib import pyplot as plt
matplotlib.use('Agg') # Отключаем GUI для нормальной работы в
другом потоке

def graph(envOn, envOff, attempt):
    graph_filenames = []
    trafficLightAllTime(envOn.metrics, attempt)
    graph_filenames.append(eachDirectionsCars(envOn.metricsPeriod,
envOff.metricsPeriod, attempt))

graph_filenames.append(carsWaitTimeInPeriod(envOn.metricsPeriod,
envOff.metricsPeriod, attempt))
    graph_filenames.append(carsHaltingInPeriod(envOn.metricsPeriod,
envOff.metricsPeriod, attempt))
    return graph_filenames
def trafficLightAllTime(metrics_on, attempt):
    colors = ['r', 'b']
    edges_labels = ['Север - Юг', 'Запад - Восток']
    for tl, color, edge in zip([tl for tl in metrics_on[0] if
"ts_time" in tl], colors, edges_labels):
        plt.plot([step["step"] / 60 for step in metrics_on[:,],
[tl["ts_time"][0] for tl in metrics_on[:,], color='r',
label=edges_labels[0]]
        plt.plot([step["step"] / 60 for step in metrics_on[:,],
[tl["ts_time"][1] for tl in metrics_on[:,], color='b',
label=edges_labels[1]]
        plt.legend(loc='center right')
        plt.ylabel('Trafficlight time (seconds)')
        plt.xlabel('Simulation time (minutes)')

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.16.001 ПЗ

Лист

86

```

filename = f'graph//TrafficlightAllTime_' + str(attempt) + '.png'
plt.savefig(filename, dpi=600)
plt.close()

def carsHaltingInPeriod(metricsPeriodOn, metricsPeriodOff, attempt):
    filenames = []
    edges_labels = ['Север', 'Восток', 'Юг', 'Запад']
    for i, (laneOn, laneOff, edge) in enumerate(
        zip([laneOn for laneOn in metricsPeriodOn[0] if
            "vehicleTotalHalting_lane " in laneOn], [laneOff for laneOff in
            metricsPeriodOff[0] if "vehicleTotalHalting_lane " in laneOff],
            edges_labels)):
        plt.plot([step["step"] / 60 for step in metricsPeriodOn[:]],
            [l[laneOn] for l in metricsPeriodOn[:]],
            label="С применением интеллектуального алгоритма",
            color="red")
        plt.plot([step["step"] / 60 for step in metricsPeriodOn[:]],
            [l[laneOn] for l in metricsPeriodOn[:]], 'o', color="red")

        plt.plot([step["step"] / 60 for step in metricsPeriodOff[:]],
            [l[laneOff] for l in metricsPeriodOff[:]],
            label="Без применения интеллектуального алгоритма",
            color="blue")
        plt.plot([step["step"] / 60 for step in metricsPeriodOff[:]],
            [l[laneOff] for l in metricsPeriodOff[:]], 'o',
            color="blue")
        plt.legend(loc='best')
        plt.title(f"Среднее количество ожидающих \n ТС на направлении
{edges_labels[i]}")
        plt.ylabel('Mean halting number (units)')
        plt.xlabel('Simulation time (minutes)')
        filename = f'graph//vehicleMeanHalting_' + str(attempt) +
str(edges_labels[i]) + '.png'
        plt.savefig(filename, dpi=600)
        plt.close()
        filenames.append(filename)
    return filenames

def eachDirectionsCars(metricsPeriodOn, metricsPeriodOff, attempt):
    filenames = []
    edges_labels = ['Север', 'Восток', 'Юг', 'Запад']
    for i, (laneOn, laneOff, edge) in enumerate(
        zip([laneOn for laneOn in metricsPeriodOn[0] if
            "vehiclesIDson_lane " in laneOn], [laneOff for laneOff in
            metricsPeriodOff[0] if "vehiclesIDson_lane " in laneOff],
            edges_labels)):
        plt.plot([step["step"] / 60 for step in metricsPeriodOn[:]],
            [l[laneOn] for l in metricsPeriodOn[:]],
            label="С применением интеллектуального алгоритма",
            color="red")
        plt.plot([step["step"] / 60 for step in metricsPeriodOn[:]],
            [l[laneOn] for l in metricsPeriodOn[:]], 'o', color="red")

```

```

plt.plot([step["step"] / 60 for step in metricsPeriodOff[:]],
[l[laneOff] for l in metricsPeriodOff[:]],
        label="Без применения интеллектуального алгоритма",
color="blue")
    plt.plot([step["step"] / 60 for step in metricsPeriodOff[:]],
[l[laneOff] for l in metricsPeriodOff[:]], 'o',
        color="blue")
    plt.title(f"Среднее количество проезжающих \n ТС на направлении
{edges_labels[i]}")
    plt.legend(loc='best')
    plt.ylabel('Mean number (units)')
    plt.xlabel('Time (minutes)')
    filename = f'graph//carsFromDirection_' + str(attempt) +
str(edges_labels[i]) + '.png'
    plt.savefig(filename, dpi=600)
    plt.close()
    filenames.append(filename)
return filenames

def carsWaitTimeInPeriod(metricsPeriodOn, metricsPeriodOff,
attempt):
    filenames = []
    edges_labels = ['Север', 'Восток', 'Юг', 'Запад']
    for i, (laneOn, laneOff, edge) in enumerate(
        zip([laneOn for laneOn in metricsPeriodOn[0] if
"vehicleMeanWaitingTime_lane " in laneOn], [laneOff for laneOff in
metricsPeriodOff[0] if "vehicleMeanWaitingTime_lane " in laneOff],
edges_labels)):
        all_values_without_alg = []
        values_with_alg = [l[laneOn] for l in metricsPeriodOn[:]]
        values_without_alg = [l[laneOff] for l in
metricsPeriodOff[:]]
        # Добавляем значения в общие списки
        all_values_with_alg.extend(values_with_alg)
        all_values_without_alg.extend(values_without_alg)
        plt.plot([step["step"] / 60 for step in
metricsPeriodOn[:]], [l[laneOn] for l in metricsPeriodOn[:]],
        label="С применением интеллектуального алгоритма",
color="red")
        plt.plot([step["step"] / 60 for step in
metricsPeriodOn[:]], [l[laneOn] for l in metricsPeriodOn[:]], 'o',
color="red")
        plt.plot([step["step"] / 60 for step in metricsPeriodOff[:]],
[l[laneOff] for l in metricsPeriodOff[:]],
        label="Без применения интеллектуального алгоритма",
color="blue")
        plt.plot([step["step"] / 60 for step in metricsPeriodOff[:]],
[l[laneOff] for l in metricsPeriodOff[:]], 'o',
color="blue")
    plt.legend(loc='best')
    plt.title(f"Среднее время ожидания \n ТС на направлении
{edges_labels[i]}")
    plt.ylabel('Mean waiting time (seconds)')
    plt.xlabel('Time (minutes)')

```

Изм.	Лист	№ докум.	Подпись	Дата

```

filename = f'graph//vehicleMeanWaitingTime_' + str(attempt) +
str(edges_labels[i]) + '.png'
plt.savefig(filename, dpi=600)
plt.close()
filenames.append(filename)
avg_with_alg = sum(all_values_with_alg) / len(all_values_with_alg)
if all_values_with_alg else 0
avg_without_alg = sum(all_values_without_alg) /
len(all_values_without_alg) if all_values_without_alg else 0
print(f"\nОбщее среднее время ожидания ТС для
{str(edges_labels[i])}:")
print(f"С алгоритмом: {avg_with_alg:.2f} секунд")
print(f"Без алгоритма: {avg_without_alg:.2f} секунд")
if avg_without_alg > 0:
improvement = ((avg_without_alg - avg_with_alg) /
avg_without_alg) * 100
print(f"Улучшение: {improvement:.2f}%")
return filenames

```

Файл «Q_agent.py»

```

from sumo_rl.agents import QAgent
from sumo_rl.environment.env import TLSCEnvironment
from sumo_rl.exploration.epsilon_greedy_q import EpsilonGreedyQ
import pickle
import os

def sumoRl(period, alpha, gamma, initial_green, agentIsOn,
net_path):
    env = TLSCEnvironment(
        net_file=net_path[0],
        route_file="grid.rou.xml",
        out_csv_name="dataFiles/output.csv",
        use_gui=False,
        num_seconds=21000,
        min_green=10,
        max_green=80,
        initial_green=initial_green,
        period=period,
        reward_fn="queue",
        agentIsOn=agentIsOn
    )
    initial_states = env.reset()
    ql_agents = {
        ts: QAgent(
            starting_state=env.encode(initial_states[ts], ts),
            state_space=env.observation_space,
            action_space=env.action_space,
            alpha=alpha,
            gamma=gamma,
            exploration_strategy=EpsilonGreedyQ(
                initial_epsilon=0.3, min_epsilon=0.001, decay=0.001
            ),
        ),
    }

```

```

for ts in env.ts_ids
}
done = {"__all__": False}
out_csv = f"dataFiles/output.csv"
if os.path.exists('database//dataDWT.pkl'):
    with open('database//dataDWT.pkl', 'rb') as f:
        for agent_id in ql_agents.keys():
            ql_agents[agent_id].q_table |= pickle.load(f)
while not done["__all__"]:
    actions = {ts: ql_agents[ts].act() for ts in ql_agents.keys()}
    s, r, done, _ = env.step(action=actions)
    for agent_id in ql_agents.keys():
        ql_agents[agent_id].learn(next_state=env.encode(s[agent_id],
agent_id), reward=r[agent_id])
initial_green = env.save_tl_phases()
env.save_csv(out_csv, 1)
q_saving = ql_agents[agent_id].q_table
with open('database//dataDWT.pkl', 'wb') as f:
    pickle.dump(q_saving, f)
return env, initial_green

```

Файл «NN_agent.py»

```

from sumo_rl.agents import NNAgent
from sumo_rl.environment.env import TLSCEnvironment
from sumo_rl.exploration import EpsilonGreedyNN
import pickle
import torch
import os

def sumoRl(period, alpha, gamma, initial_green, agentIsOn,
net_path):
    env = TLSCEnvironment(
        net_file=net_path[0],
        route_file="grid.rou.xml",
        out_csv_name="dataFiles//output.csv",
        use_gui=False,
        num_seconds=21000,
        min_green=10,
        max_green=80,
        initial_green=initial_green,
        period=period,
        reward_fn="diff-waiting-time",
        agentIsOn=agentIsOn,
    )
    initial_states = env.reset()
    lossList = []
    ql_agents = {
        ts: NNAgent(
            starting_state=env.encode(initial_states[ts], ts),
            action_space=env.action_space,
            alpha=alpha,
            gamma=gamma,

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.16.001 ПЗ

Лист

90

```

exploration_strategy=EpsilonGreedyNN(
    initial_epsilon=0.3, min_epsilon=0.01, decay=0.01
),
    lossList=lossList
)
for ts in env.ts_ids
}
done = {"__all__": False}
out_csv = f"dataFiles/output.csv"
if os.path.exists('database//dataQ.pth'):
    for agent_id in ql_agents.keys():
        ql_agents[agent_id].q_network.load_state_dict(torch.load('database/
/dataQ.pth'))
while not done["__all__"]:
    actions = {ts: ql_agents[ts].act() for ts in ql_agents.keys()}
    s, r, done, _ = env.step(action=actions)
    for agent_id in ql_agents.keys():
        ql_agents[agent_id].learn(ql_agents[agent_id].state,
actions[agent_id],

next_state=env.encode(s[agent_id], agent_id), reward=r[agent_id])
initial_green = env.save_tl_phases()
env.save_csv(out_csv, 1)
for agent_id in ql_agents.keys():
    torch.save(ql_agents[agent_id].q_network.state_dict(),
'database//dataQ.pth')
    params = ql_agents[agent_id].q_network.state_dict()
weights = []
for param_tensor in ql_agents[agent_id].q_network.state_dict():
weights.append(ql_agents[agent_id].q_network.state_dict()[param_ten
sor].detach().numpy())
return env, initial_green

```

Файл «route_change.py»

```

import xml.etree.cElementTree as ET
import random
import numpy as np
import sumolib.route
import math
import os

class V:
    depart = 0
    attr_from = ''
    to = ''

    def __init__(self, depart, attr_from, to):
        self.depart = depart
        self.attr_from = attr_from
        self.to = to

```

```

def change_time_in_period(car_count_list, period,
dict_vehicle_list):
    total_cars = sum(car_count_list)
    for vehicle_list in dict_vehicle_list.values():
        time = 0
        j = 0
        for num_cars in car_count_list:
            for y in range(num_cars):
                a = np.random.exponential(scale=1 / 2.1859)
                time_interval = period / num_cars * a
                vehicle_list[j].event_time = time + y +
time_interval
                if vehicle_list[j].event_time < 0:
                    vehicle_list[j].event_time = 0
                j += 1
            time += period

def remove_cars_in_junction(dict_vehicle_list):
    incoming_routes = []
    exiting_routes = []
    exiting_routes_set = set()
    temp = sumolib.net.readNet('3Junction.net.xml')
    routes = temp.getEdges()

    for route in routes:
        if len(route.getIncoming()) <= 1 and
len(route.getOutgoing()) > 0:
            incoming_routes.append(route.getID()[2])
            exiting_routes = list(exiting_routes_set)
            new_dict_vehicle_list = dict(dict_vehicle_list) # Create a new
dictionary for the result
            for key, vehicles in dict_vehicle_list.items():
                new_vehicles = []
                for car in vehicles:
                    if car.start in incoming_routes:
                        new_vehicles.append(car)
                new_dict_vehicle_list[key] = new_vehicles

    return new_dict_vehicle_list

def reroute_cars(dict_vehicle_list, flux_function):

    def continue_routes(car, junction_list, flux_function,
incoming_junction, outgoing_junction):
        if incoming_junction in junction_list:
            flux_subdict =
flux_function.get(incoming_junction).get(outgoing_junction)
            total_sum = 0
            random_number = round(random.uniform(0, 1), 8)
            outgoing_junction = incoming_junction
            for incoming_junction, value in flux_subdict.items():
                if random_number > total_sum + value:
                    total_sum += value

```

```

else:
    return continue_routes(car, junction_list,
flux_function, incoming_junction, outgoing_junction)
    else:
        return outgoing_junction + incoming_junction

junction_list = list(dict_vehicle_list.keys())
for junction in junction_list:
    for key1, vehicles in dict_vehicle_list.items():
        for car in vehicles:
            car.start_road = car.start + key1
            car.finish_road = continue_routes(car, junction_list,
flux_function, key1, car.start)

def set_parameters(dict_vehicle_list):
    tree = ET.parse('flows.xml')
    j = 0
    i = 0
    junctionList = list(dict_vehicle_list.keys())
    for i in range(len(junctionList)):
        for key1, vehicles in dict_vehicle_list.items():
            for car in vehicles:
                car.startRoad = car.start + key1
                car.finishRoad = key1 + car.finish
    junction = list(dict_vehicle_list.keys())
    cars = list(dict_vehicle_list.values())
    for trip in tree.getroot():
        start = cars[j][i].startRoad
        finish = cars[j][i].finishRoad
        eventTime = cars[j][i].eventTime
        trip.set('from', start)
        trip.set('to', finish)
        trip.set('depart', str(eventTime))
        if i < len(cars[j]) - 1:
            i += 1
        else:
            i = 0
            j += 1
        start = ''
        finish = ''
        eventTime = 0
    root = tree.getroot()
    sorted_root = ET.Element(root.tag)
    sorted_elements = sorted(root, key=lambda child:
float(child.get('depart')))
    for element in sorted_elements:
        sorted_root.append(element)
    tree = ET.ElementTree(sorted_root)
    tree.write('flows.xml')

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.16.001 ПЗ

Лист

93

```

def generate_ways():
    ways_move = []
    incoming_routes = []
    exiting_routes = []
    temp = sumolib.net.readNet('1Junction.net.xml')
    routes = temp.getEdges()

    for route in routes:
        if len(route.getIncoming()) > 1 and
len(route.getOutgoing()) == 0:
            exiting_routes.append(route.getID())
        else:
            incoming_routes.append(route.getID())

    ways_move.extend([[x, exiting_routes] for x in
incoming_routes])
    return ways_move

def flux_to_route(ratio, flux_list, period, ways, cars):
    i = 0
    offset = 0
    error = 0
    set_list = [] * cars
    incoming_routes = 0

    for way in ways:
        ratio_flux_list = [
            cars / len(ways) * ratio[incoming_routes] *
flux_list[i] for i in range(len(flux_list))
        ]

        for exiting_routes in way[1]:
            while i < len(ratio_flux_list):
                if offset >= math.floor(ratio_flux_list[i]):
                    error += ratio_flux_list[i] - offset
                    if error >= 1:
                        v = V(period * i + period /
ratio_flux_list[i] * offset, way[0], exiting_routes)
                        set_list.append(v)
                        error -= 1
                        i += 1
                        offset = 0
                        continue
                    v = V(period * i + period / ratio_flux_list[i]
* offset, way[0], exiting_routes)
                    set_list.append(v)
                    offset += 1

            i = 0
            incoming_routes += 1

    set_list.sort(key=lambda x: x.depart)
    os.system(f"randomTrips.py -n 1Junction.net.xml -o flows.xml -e
{len(set_list)}")

```

```

i = 0
tree = ET.parse('flows.xml')

for trip in tree.getroot():
    trip.set('depart', str(set_list[i].depart))
    trip.set('from', set_list[i].attr_from)
    trip.set('to', set_list[i].to)
    i += 1

tree.write("flows.xml")

```

Файл «env.py»

```

class TLSCEnvironment(SumoEnvironment):
    def __init__(
        self,
        net_file: str,
        route_file: str,
        initial_green: dict,
        out_csv_name: Optional[str] = None,
        use_gui: bool = False,
        virtual_display: Tuple[int, int] = (3200, 1800),
        begin_time: int = 0,
        num_seconds: int = 4500,
        max_depart_delay: int = -1,
        waiting_time_memory: int = 1000,
        time_to_teleport: int = -1,
        delta_time: int = 5,
        yellow_time: int = 2,
        min_green: int = 10,
        max_green: int = 50,
        single_agent: bool = False,
        reward_fn: Union[str, Callable, dict] = "diff-waiting-
time",
        observation_class: ObservationFunction =
DefaultObservationFunction,
        add_system_info: bool = True,
        add_per_agent_info: bool = True,
        sumo_seed: Union[str, int] = "random",
        fixed_ts: bool = False,
        sumo_warnings: bool = True,
        additional_sumo_cmd: Optional[str] = None,
        render_mode: Optional[str] = None,
        period: int = 480,
        vehicleDictionary: dict = {},
        agentIsOn: bool = True,
    ) -> None:
        super(TLSCEnvironment, self).__init__(net_file,
                                                route_file,
                                                initial_green,
                                                out_csv_name,
                                                use_gui,
                                                virtual_display,
                                                begin_time,
                                                num_seconds,

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.16.001 ПЗ

Лист

95

```

max_depart_delay,
waiting_time_memory,
time_to_teleport,
delta_time,
yellow_time,
min_green,
max_green,
single_agent,
reward_fn,
observation_class,
add_system_info,
add_per_agent_info,
sumo_seed,
fixed_ts,
sumo_warnings,
additional_sumo_cmd,
render_mode,
agentIsOn)

self.period = period
self.agentIsOn = agentIsOn
self.vehicleDictionary = vehicleDictionary
self.metricsPeriod = []

def _get_period_metrics(self):
    last_period = 0
    for ts_id in self.ts_ids:
        buff = dict(
            {"step": 0, ts_id + "_stopped": 0, ts_id +
            "_accumulated_waiting_time": 0, ts_id + "_average_speed": 0})
        buff.update({"vehiclesIDson_lane " + lane[:-2]: set() for
        lane in self.traffic_signals[ts_id].lanes})
        buff.update({"ts_time": [0, 0]})
        buff.update(
            {"vehicleMeanWaitingTime_lane " + lane[:-2]: float()
        for lane in self.traffic_signals[ts_id].lanes})
        buff.update(
            {"vehicleTotalHalting_lane " + lane[:-2]: float() for
        lane in self.traffic_signals[ts_id].lanes})
        for i in range(len(self.metrics)):
            if int(self.metrics[i].get("step")) <= last_period +
            int(self.period):
                buff |= {ts_id + "_stopped":
                    float(self.metrics[i].get(ts_id +
                    "_stopped")) + float(buff.get(ts_id + "_stopped"))}
                buff |= {ts_id + "_accumulated_waiting_time":
                    float(self.metrics[i].get(ts_id + "_accumulated_waiting_time")) +
                    float(buff.get(
                        ts_id + "_accumulated_waiting_time"))}
        buff |= {ts_id + "_average_speed":
                    float(self.metrics[i].get(ts_id + "_average_speed")) +
                    float(
                        buff.get(ts_id + "_average_speed"))}

```

					ПГУ 1.090301.16.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		96

```

for lane in self.traffic_signals[ts_id].lanes:
    buff |= {"vehiclesIDson_lane " + lane[:-2]: set(
        self.metrics[i].get("vehiclesIDson_lane " + lane[:-2], []))
    | set(
        buff.get("vehiclesIDson_lane " + lane[:-2], []))}
    if len(self.metrics[i].get("vehiclesIDson_lane " + lane[:-2]))
> 0:
        buff |= {"vehicleMeanWaitingTime_lane " + lane[:-2]:
            float(buff.get("vehicleMeanWaitingTime_lane "
+ lane[:-2])) + self.metrics[
                i].get(
                    "vehicles_accumulated_waiting_time_on_lane
" + lane[:-2])}
        buff |= {"vehicleTotalHalting_lane " + lane[:-2]:
            float(buff.get("vehicleTotalHalting_lane " +
lane[:-2])) + self.metrics[i].get(
                "vehicles_halting_on_lane " + lane[:-2])}
buff |= {"ts_time": [x + y for x, y in
zip(self.metrics[i].get("ts_time"), buff.get("ts_time"))]}
else:
    buff |= {"step": self.metrics[i].get("step")}
    buff |= {
        ts_id + "_stopped": float(buff.get(ts_id + "_stopped")) /
float(self.period / self.delta_time)}
    buff |= {ts_id + "_accumulated_waiting_time":
        float(buff.get(ts_id +
"_accumulated_waiting_time")) / float(
            self.period / self.delta_time)}
    buff |= {ts_id + "_average_speed": float(buff.get(ts_id +
"_average_speed"))}
    buff |= {"vehiclesIDson_lane " + lane[:-2]:
len(buff.get("vehiclesIDson_lane " + lane[:-2], [])) for
        lane in self.traffic_signals[ts_id].lanes}
    buff |= {"vehicleTotalHalting_lane " + lane[:-2]: float(
        buff.get("vehicleTotalHalting_lane " + lane[:-2]))
        /
float(self.period / self.delta_time) for
        lane in self.traffic_signals[ts_id].lanes}
buff |= {"ts_time": [time / self.period for time in
buff.get("ts_time")]}
buff |= {"vehicleMeanWaitingTime_lane " + lane[:-2]: float(
    buff.get("vehicleMeanWaitingTime_lane " + lane[:-2])) /
float(self.period / self.delta_time) for
        lane in self.traffic_signals[ts_id].lanes}
self.metricsPeriod.append(buff.copy())
buff = dict(
    {"step": 0, ts_id + "_stopped": 0, ts_id +
"_accumulated_waiting_time": 0,
    ts_id + "_average_speed": 0})
buff.update(
    {"vehiclesIDson_lane " + lane[:-2]: set() for lane in
self.traffic_signals[ts_id].lanes})
buff.update({"ts_time": [0, 0]})

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.16.001 ПЗ

Лист

97

```

buff.update(
    {"vehicleMeanWaitingTime_lane " + lane[:-2]:
float() for lane in
    self.traffic_signals[ts_id].lanes})
    buff.update(
    {"vehicleTotalHalting_lane " + lane[:-2]:
float() for lane in
    self.traffic_signals[ts_id].lanes})
    last_period += self.period

def save_csv(self, out_csv_name, episode):
    """Save metrics of the simulation to a .csv file.

    Args:
        out_csv_name (str): Path to the output .csv file. E.g.:
"results/my_results
        episode (int): Episode number to be appended to the output
file name.
    """
    self._get_period_metrics()
    super(TLSCEnvironment, self).save_csv(out_csv_name, episode)
    if out_csv_name is not None:
        df = pd.DataFrame(self.metricsPeriod)
        Path(Path(out_csv_name).parent).mkdir(parents=True,
exist_ok=True)
        df.to_csv(out_csv_name + f"_conn{self.label}_ep{episode}" +
f"_period{self.period}" + ".csv", index=False)

        for id in self.ts_ids:
            Path(Path(out_csv_name).parent).mkdir(parents=True,
exist_ok=True)
            df.to_csv(out_csv_name +
f"_conn{self.label}_ep{episode}" + f"_ts{id}" + ".csv",
index=False)

```

Файл «nn_agent.py»

```

import torch.optim as optim
import numpy as np
from sumo_rl.exploration.epsilon_greedy_nn import EpsilonGreedyNN

class QNetwork(nn.Module):
    def __init__(self, state_size, action_size):
        super(QNetwork, self).__init__()
        self.fc1 = nn.Linear(state_size, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, action_size)
    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x

```

```

class NNAgent:
    """Q-learning Agent class."""

    def __init__(self, starting_state, action_space, alpha=0.5,
gamma=0.01, exploration_strategy=EpsilonGreedyNN(),
        lossList=[]):
        """Initialize Q-learning agent."""
        self.state = starting_state
        self.state_size = len(self.state)
        self.action_space = action_space
        self.action_size = self.action_space.n
        self.alpha = alpha
        self.gamma = gamma
        self.exploration = exploration_strategy
        self.q_network = QNetwork(int(self.state_size),
int(self.action_size))
        self.optimizer = optim.Adam(self.q_network.parameters(),
lr=alpha)
        self.acc_reward = 0
        self.device = torch.device("cuda" if
torch.cuda.is_available() else "cpu")
        self.lossList = lossList
    def act(self):
        self.state = torch.tensor(self.state,
dtype=torch.float32).to(self.device)
        q_values = self.q_network(self.state)
        action = self.exploration.choose(q_values, self.action_space)
        return action

    def learn(self, state, action, next_state, reward, done=False):
        state = torch.tensor(state,
dtype=torch.float32).to(self.device)
        next_state = torch.tensor(next_state,
dtype=torch.float32).to(self.device)
        q_values = self.q_network(state)
        q_values_next = self.q_network(next_state)
        q_value = q_values[action]
        if done:
            target = reward
        else:
            target = reward + self.gamma * torch.max(q_values_next)
        loss = nn.MSELoss()(q_value, target)
        self.optimizer.zero_grad()
        self.lossList.append(int(loss))
        loss.backward()
        self.optimizer.step()
        self.acc_reward += reward

```

Файл «ql_agent.py»

```

from sumo_rl.exploration.epsilon_greedy_q import EpsilonGreedyQ
class QAgent:
    def __init__(self, starting_state, state_space, action_space,
alpha=0.5, gamma=0.95, exploration_strategy=EpsilonGreedyQ()):

```

					ПГУ 1.090301.16.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		99

```

self.state = starting_state
self.state_space = state_space
self.action_space = action_space
self.action = None
self.alpha = alpha
self.gamma = gamma
self.q_table = {self.state: [0 for _ in range(action_space.n)]}
self.exploration = exploration_strategy
self.acc_reward = 0

def act(self):
    """Choose action based on Q-table."""
    self.action = self.exploration.choose(self.q_table, self.state,
self.action_space)
    return self.action

def learn(self, next_state, reward, done=False):
    """Update Q-table with new experience."""
    if next_state not in self.q_table:
        self.q_table[next_state] = [0 for _ in
range(self.action_space.n)]
    if next_state in self.q_table:
        a = 1
        s = self.state
        s1 = next_state
        a = self.action
        delta_time = self.alpha * (
            reward + self.gamma * max(self.q_table[s1]) -
self.q_table[s][a]
        )
        self.q_table[s][a] = self.q_table[s][a] + self.alpha * (
            reward + self.gamma * max(self.q_table[s1]) -
self.q_table[s][a]
        )
        self.state = s1
        self.acc_reward += reward

```

Файл «epsilon_greedy_nn.py»

```

import numpy as np

class EpsilonGreedyNN:
    """Epsilon Greedy Exploration Strategy."""

    def __init__(self, initial_epsilon=1.0, min_epsilon=0.0,
decay=0.99):
        """Initialize Epsilon Greedy Exploration Strategy."""
        self.initial_epsilon = initial_epsilon
        self.epsilon = initial_epsilon
        self.min_epsilon = min_epsilon
        self.decay = decay

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.16.001 ПЗ

Лист

100

```

def choose(self, q_values, action_space):
    """Choose action based on epsilon greedy strategy."""
    if np.random.rand() < self.epsilon:
        action = int(action_space.sample())
    else:
        action = np.argmax(q_values.cpu().detach().numpy())

    self.epsilon = max(self.epsilon * self.decay, self.min_epsilon)
    # print(self.epsilon)
    return action

def reset(self):
    """Reset epsilon to initial value."""
    self.epsilon = self.initial_epsilon

```

Файл «epsilon_greedy_q.py»

```

import numpy as np

class EpsilonGreedyQ:

    def __init__(self, initial_epsilon=1.0, min_epsilon=0.0,
decay=0.99):
        self.initial_epsilon = initial_epsilon
        self.epsilon = initial_epsilon
        self.min_epsilon = min_epsilon
        self.decay = decay

    def choose(self, q_table, state, action_space):
        if np.random.rand() < self.epsilon:
            action = int(action_space.sample())
        else:
            action = np.argmax(q_table[state])

        self.epsilon = max(self.epsilon * self.decay,
self.min_epsilon)
        # print(self.epsilon)
        return action

    def reset(self):
        """Reset epsilon to initial value."""
        self.epsilon = self.initial_epsilon

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.16.001 ПЗ

Лист

101

Приложение В

(Обязательное)

Программа моделирования интеллектуального управления дорожным

трафиком

Презентация

					ПГУ 1.090301.16.001 ПЗ	Лист
						102
Изм.	Лист	№ докум.	Подпись	Дата		

Бизнес-проект «Программа моделирования интеллектуального управления дорожным трафиком»

Докладчики:
студенты группы 218881
Аляев Александр
Лобанов Рамиль

1

Проблема

Загруженность дорог в крупных городах России остаётся крупной проблемой, приводя к значительным экономическим потерям, увеличению выбросов вредных веществ в атмосферу и снижению качества жизни.

Две трети россиян регулярно стоят в серьёзных пробках — ежедневно или по несколько раз в неделю. Такие результаты были получены в ходе опроса пользователей Авто.ру.

Существующие системы светофорного регулирования во многих городах работают по фиксированному временному алгоритмам. Это приводит к неэффективному распределению фаз работы светофорного объекта и увеличению пробок.



2

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.16.001 ПЗ

Лист

103

Цель и задачи работы

Цель работы – разработать программу и пользовательский интерфейс для неё, реализующую интеллектуальные методы управления светофорными объектами на перекрестке, а также визуализирующую их эффективность при помощи графиков зависимостей различных характеристик дорожного движения.

Задачи работы:

- проанализировать существующие среды имитационного моделирования, фреймворки и библиотеки для создания GUI и алгоритмы интеллектуального управления транспортными потоками;
- разработать имитационную модель дорожного трафика и пользовательский интерфейс для неё, основанную на агентном подходе;
- разработать алгоритм для генерации данных о трафике на основе информации от пользователя;
- разработать интеллектуальные алгоритмы управления светофорным объектом и интегрировать их в имитационную модель;
- провести сравнительный анализ эффективности интеллектуальных алгоритмов в управлении светофорами по сравнению со статическими методами.

3

Стек технологий



5

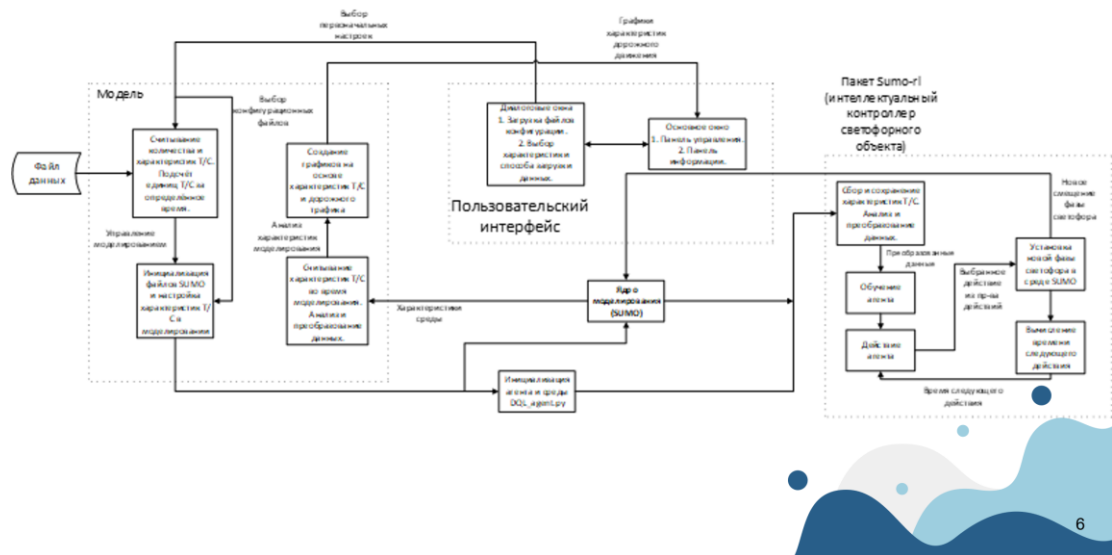
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.16.001 ПЗ

Лист

104

Структура программы



Среда SUMO (Simulation of Urban Mobility)

- Микроскопический симулятор транспортных потоков.
- Поддерживает интеграцию с внешними агентами через API TraCI.

Конфигурационные файлы:

- Файл сценария .sumocfg определяет параметры моделирования
- Файл сети дорог .net.xml хранит геометрию дорожной сети
- Файлы маршрутов .flows.xml и .rows.xml определяют пути и интервалы движения транспортных средств
- Файл результатов .output.xml содержит информацию о характеристиках моделирования
- Файл дополнений .add.xml хранит информацию о дополнительных параметрах дорожной сети (датчики, скоростные ограничения и т.п.)



Пример движения транспортных средств в SUMO



7

Изм.	Лист	№ докум.	Подпись	Дата

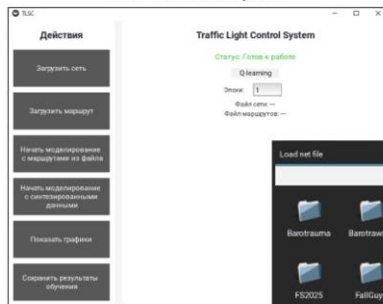
ПГУ 1.090301.16.001 ПЗ

Лист

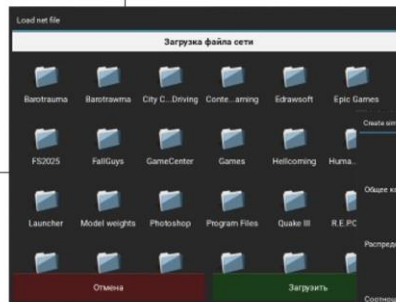
105

Интерфейс пользователя

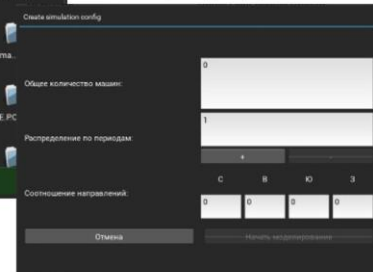
Главный экран



Загрузка файлов



Генерация данных



8

Генерация и загрузка данных



Общая схема генерации данных

Формула генерации : $A = T * i + \left(\frac{T}{r_i}\right) k$,
 где A – итоговое время появления ТС в моделировании в секундах;
 T – период усреднения данных;
 i – номер текущего периода;
 r_i – Шаг моделирования;
 k – номер машины в моделировании .

Пример формата загружаемых данных:

```

<trip id="0" depart="0.0" from="A1B1" to="B1A1"/>
<trip id="1" depart="0.0" from="A1B1" to="B1B0"/>
    
```

9

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.16.001 ПЗ

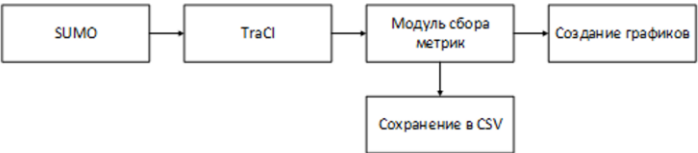
Лист

106

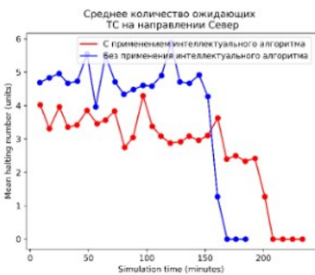
Сбор метрик и создание графиков

Собираемые метрики:

- Среднее время ожидание проезда на перекрёстке
- Длина очереди на перекрёстке
- Количество машин на перекрёстке
- Продолжительность фазы работы светофора
- Средняя скорость по направлениям



Пример графика (среднее количество машин)



Бизнес - план



Изм.	Лист	№ докум.	Подпись	Дата

Целевая аудитория и каналы взаимодействия

Аудитория :

- Муниципальные транспортные службы – заинтересованы в улучшении пропускной способности транспортной системы
- Разработчики ИТС – используют алгоритмы в составе более сложных продуктов
- Исследовательские группы – занимаются анализом и улучшением транспортной инфраструктуры

Каналы взаимодействия :

- Участие в профильных выставках и конференциях
- Сотрудничество с органами власти
- Публикации в научных журналах
- Прямые встречи с заинтересованными лицами



16

Перспективы коммерциализации

Инвестиции в область: с 2019 г. инвестиции в ИТС в

- России превысили 100 млрд Р. В ходе национального проекта «Безопасные качественные дороги» в 2024 г. было выделено 3,15 млрд Р на внедрение ИТС в 62 городах в 56 субъектах России.

Рыночная потребность: по прогнозам к 2027 – 2029 гг. рынок российских ИТС будет расти ежегодно на 8 – 10 % за счёт инвестиций в ИТС проекты.

Простота интеграции : система может быть внедрена на основе существующей городской инфраструктуры без необходимости капитальных изменений .



17

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.16.001 ПЗ

Лист

108



Спасибо за внимание!



19

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.16.001 ПЗ

Лист

109