Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительная техника

УТВЕРЖДАЮ
Заведующий кафедрой
_____О.В. Непомнящий
«___»_____2024 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 Информатика и вычислительная техника

Симулятор проезда перекрестков вне населенных пунктов

Руководитель _____ _____ старший преподаватель Л.В. Макуха
подпись *дата* *должность, ученая степень*

Выпускник _____ _____ Л.В. Ластовский
подпись *дата*

Нормоконтролёр _____ _____ Л.В. Макуха
подпись *дата*

Красноярск 2024

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительная техника

УТВЕРЖДАЮ
Заведующий кафедрой
_____О.В. Непомнящий
«___»_____2024 г.

ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

в форме бакалаврской работы

Студенту Ластовскому Леониду Витальевичу
фамилия, имя, отчество

Группа КИ20-06Б Направление (специальность) 09.03.01
номер код

Информатика и вычислительная техника
наименование

Тема выпускной квалификационной работы: Симулятор проезда перекрестков вне населенных пунктов

Утверждена приказом по университету № _____ от _____

Руководитель ВКР: Л.В. Макуха, старший преподаватель ВТ ИКИТ
инициалы, фамилия, учёная степень, должность, место работы

СФУ

Исходные данные для ВКР:
Задание на ВКР.

Перечень разделов ВКР:
1. Анализ предметной области.
2. Проектирование приложения симулятора.
3. Разработка симулятора.

Перечень графического материала: Презентация в формате Microsoft PowerPoint

Руководитель ВКР _____ Л.В. Макуха
подпись инициалы, фамилия

Задание принял к исполнению _____ Л.В. Ластовский
подпись инициалы, фамилия

«22» 12 2023 г.
дата

РЕФЕРАТ

Выпускная квалификационная работа по теме «Симулятор проезда перекрестков вне населенных пунктов» содержит 42 страниц текстового документа, 25 рисунков, 11 использованных источников.

СИМУЛЯТОР, ДОРОГА, ДТП, ЗАГОРОДНЫЕ ТРАССЫ, PYTHON.

Цель работы: разработать симулятор для проезда перекрестка вне населённых пунктов.

В первой главе проведен анализ предметной области, приведен анализ аналогов, поставлена цель и задачи для реализации симулятора.

Во второй главе произведен выбор языка программирования, среды разработки. Приведено проектирование вариантов использования, также разработан примерный интерфейс симулятора.

В третьей главе представлена реализация программы для симуляции проезда перекрестка вне населенных пунктов, описана подробная последовательность работы симуляции и устройства классов.

В результате работы над бакалаврской работой было разработан и реализован симулятор проезда перекрестков вне населенных пунктов.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире транспорт играет ключевую роль в жизни общества, обеспечивая мобильность населения и эффективность экономических процессов. Однако с увеличением количества транспортных средств возрастает и нагрузка на дорожную инфраструктуру, что приводит к росту количества дорожно-транспортных происшествий (ДТП). Особенно остро эта проблема проявляется на трассах и перекрестках вне населенных пунктов, которые часто становятся местами с повышенной аварийной опасностью.

Цель данной выпускной работы — разработка и исследование программы для симуляции проезда перекрестка вне населенных пунктов. Эта программа позволит моделировать различные сценарии дорожного движения и анализировать их влияние на безопасность и эффективность проезда через перекрестки. Симуляция дорожного движения является важным инструментом в планировании и управлении дорожной сетью, предоставляя ценную информацию для разработки мер по снижению количества аварий [1].

Актуальность работы обусловлена высоким уровнем дорожно-транспортных происшествий на перекрестках, что подтверждается статистическими данными и подчеркивает необходимость разработки новых подходов и технологий для их предотвращения. Например, исходя из данных Госавтоинспекции за 2023 год в России зарегистрировано 31608 случаев ДТП вне городов и населенных пунктов в которых погибло 8021 человек и пострадало 45017 людей, эти ДТП составляют около 30% от всех ДТП, произошедших за 2023 год [2]. Данная статистика подчеркивает необходимость разработки эффективных решений. Программа для симуляции проезда перекрестка позволит проводить комплексный анализ и выявлять опасные моменты в организации дорожного движения, способствуя разработке рекомендаций по улучшению дорожной безопасности.

1 Анализ предметной области

Дорожно-транспортные происшествия (ДТП) на дорогах и перекрёстках вне населенных пунктов представляют собой серьезную проблему для безопасности дорожного движения. Подобные ситуации обладают своей спецификой, требующей детального анализа и принятия эффективных мер для предотвращения происшествий.

Двухполосные дороги вне населенных пунктов с двусторонним движением (т. е. дороги, состоящая из двух встречных полос неразделенного движения, на которых смена полосы движения и объезд возможны только при встречном движении) составляют во всем мире важную часть сети загородных дорог. На дорогах данного типа обгон играет значительную роль, влияя как на безопасность дорожного движения, так и на мобильность с точки зрения пропускной способности. При этот отсутствие возможности совершить обгон может привести к образованию крупных пробок, но при этом взаимодействие со встречным транспортным потоком может значительно увеличить риск ДТП, например, из-за ложных оценок времени и расстояния до встречных транспортных средств, которые делают водители [3-4].

Основной причиной ДТП на дорогах вне населенных пунктов часто считают простое несоблюдение транспортным средством скоростного режима. Однако, по статистики возникновение аварийной ситуации на участках дорог вне населенных пунктов обусловлено не только несоблюдением скоростного режима, но и недостаточной или ограниченной видимостью перед транспортным средством, двигающимся на участке дороги в различных погодных условиях [5].

Кроме того, одной из наиболее частых и серьезных причин травм и смертельных исходов на дорогах вне населенных пунктов является обгон. В целом, маневр обгона является одним из самых сложных и трудных элементов вождения, требующий принятия быстрых и сложных решений, и включающим в себя ряд последовательных действий, таких как маневр смены полосы

движения, возможные действия по ускорению и замедлению, а также оценка относительной скорости обгоняемого и обгоняющего транспортных средств. Но этот маневр также может быть и неудачным по тем или иным причинам [6-7]. Все эти трудности особенно проявляются на двухполосных дорогах вне населенных пунктов, в основном из-за большого ранообразия влияющих факторов. Водители, совершающие обгон, должны пройти через определенные процессы принятия решений, чтобы оценить разрывы во встречном потоке и определить, оправдан ли обгон в постоянно меняющихся дорожных и транспортных условиях [8].

Суммируя вышесказанное можно скачать что, риск возникновения аварийной ситуации возрастает в случае формирования на участках дорог вне населенных пунктов предаварийной конфликтной дорожно-транспортной ситуации между транспортными средствами участников дорожного движения, сложившейся вследствие несоблюдения ими дистанции безопасности, но при этом часто характеризующейся наличием технической возможности у участников дорожного движения предупредить аварийную ситуацию. По статистике несвоевременность предупреждения об аварийной ситуации, возникшей вследствие формирования предаварийной конфликтной дорожно-транспортной ситуации между транспортными средствами участников дорожного движения, чревата возникновением следующих видов дорожно-транспортных происшествий:

– попутное столкновение: возникает между транспортными средствами участников дорожного движения, двигающихся на участке дороги в попутном направлении;

– встречное столкновение: возникает между транспортными средствами участников дорожного движения, двигающихся на участке дороги во встречном направлении;

– столкновение на перекрестке неравнозначных дорог: возникает между транспортными средствами участников дорожного движения, одно из которых

двигается по главной дороге, а другое - осуществляет выезд со второстепенной дороги на главную.

Исходя из представленных причин ДТП, для предотвращения ДТП возможны следующие технологические решения:

– системы предупреждения столкновений: Возможна разработка способа предупреждения столкновений транспортных средств на участках дорог вне населенных пунктов, обеспечивающего повышение безопасности дорожного движения на участках дорог вне населенных пунктов за счет своевременного и адресного предупреждения участников дорожного движения о возникновении различных аварийных ситуаций на участках дорог вне населенных пунктов [9];

– использование интеллектуальных транспортных систем (ИТС): Применение ИТС для мониторинга дорожного движения и предотвращения конфликтных ситуаций на перекрёстках.

1.1 Анализ существующих продуктов

Для создания наиболее эффективного симулятора стоит рассмотреть уже имеющиеся симуляторы и выделить их преимущества и недостатки. Оба аналога представляют собой программное обеспечение для моделирования транспортных систем и потоков. Подходящими для рассмотрения программами являются Aimsun [10] и PTV VISSIM [11].

1.1.1 Aimsun

Aimsun — это программное обеспечение для моделирования транспортных потоков и управления транспортной инфраструктурой. Оно используется для анализа, проектирования и оптимизации систем транспортного движения. Aimsun предоставляет инструменты для создания динамических моделей дорожного движения, имитирующих поведение

транспортных средств и пассажиров в реальном времени. Интерфейс Aimsun изображен на рисунке 1.

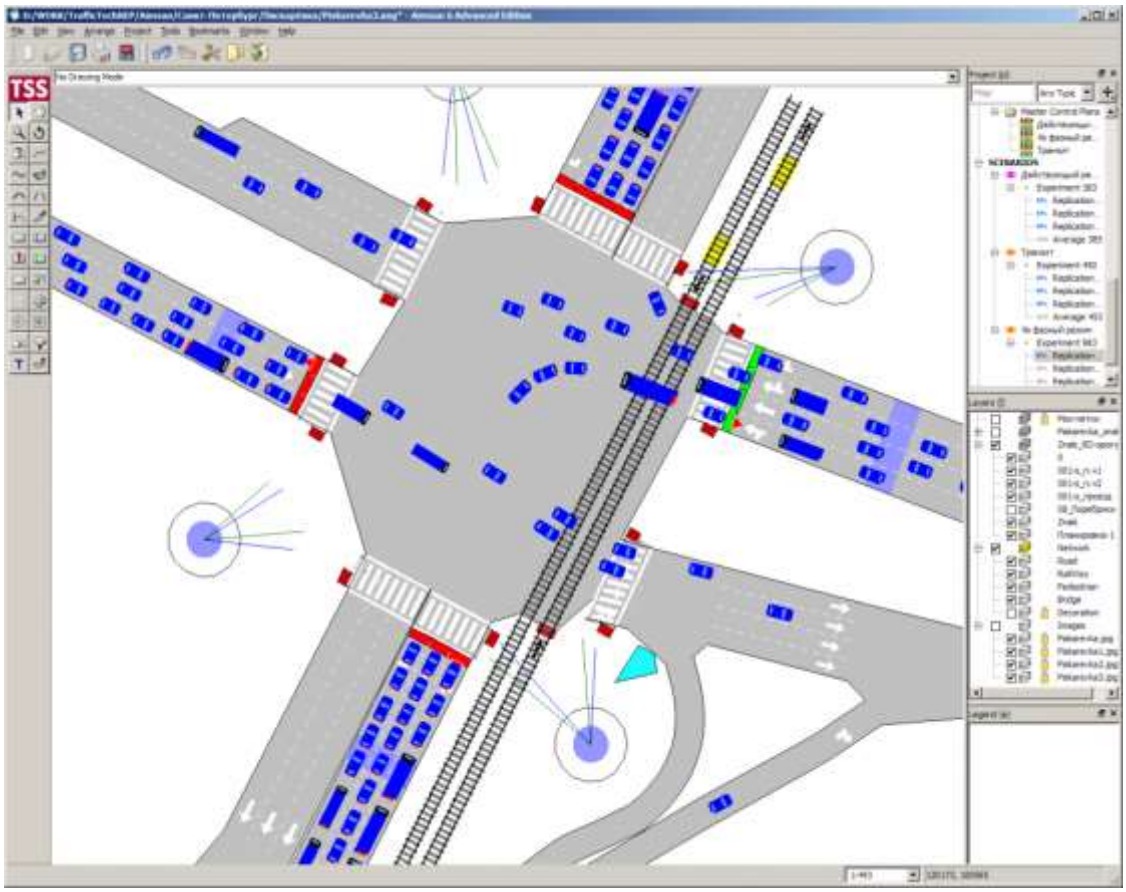


Рисунок 1 – Окно работы программы aimsun

К основным преимуществам Aimsun относиться возможность моделировать транспортные системы на различных уровнях, включая мезоскопическое и микроскопическое моделирование. А также динамическое моделирование транспортных сценариев, оценку эффективности инфраструктуры, анализ воздействия новых проектов на дорожное движение, симуляцию чрезвычайных ситуаций и другие.

Недостатками являются: сложность использования для новичков из-за богатства функциональности и возможностей, высокие системные требования, а также главным недостатком является платная модель распространения.

1.1.2 PTV VISSIM

PTV Vissim — это программное обеспечение для моделирования транспортных систем и дорожного движения. Оно разработано компанией PTV Group и предоставляет инструменты для анализа, проектирования и оптимизации транспортных сценариев. VISSIM специализируется на микроскопическом моделировании и позволяет создавать виртуальные модели дорожного движения с целью изучения воздействия различных факторов на транспортные потоки и эффективность инфраструктуры. Интерфейс Vissim изображен на рисунке 2.

Рисунок 2 – Окно работы программы PTV Vissim

К главным преимуществам этого аналога можно отнести то что PTV Vissim позволяет реализовать очень точное и детальное моделирование транспортных потоков, включая автомобили, общественный транспорт и пешеходов, используя модель следования с переменными временными интервалами. Также он имеет довольно большую гибкость и масштабируемость, Vissim дополнительно предоставляет множество инструментов для анализа и оценки транспортного потока.

К минусам относится большая сложность освоения, высокие требования к аппаратному обеспечению, более меньший инструментарий по сравнению с Aimsun, а также PTV Vissim, как и Aimsun распространяется на платной основе.

1.2 Формулировка требований к продукту

Финальная версия программы будет выглядеть следующим образом. Программа для симуляции будет представлять из себя приложение, открыв которое необходимо будет выполнить предстартовую настройку, сначала необходимо выбрать заранее подготовленный файл конфигурации перекрестка и прилегающего участка дороги, этот файл будет содержать все необходимые параметры этого перекрестка, а именно:

– информацию о всех дорогах, от куда и куда они ведут, длину этих дорог, количество полос, и другие параметры;

– информацию о всех полосах определенной дороги, у каждой полосы указывается максимальная разрешенная скорость и указания в каких направлениях будет возможен - разрешен маневр из этой полосы на перекрестке;

– информацию о всех перекрестках, какие дороги являются перекрестком и какие дороги в него входят и какие выходят.

Дальше пользователь должен в окне настройки симуляции задать остальные параметры симуляции:

– время работы симуляции;

– интенсивность появления новых машин в симуляции;

– настройку агрессивности водителей, какие именно уровни агрессивности будут присутствовать в симуляции, и вероятности их появления в симуляции;

– погодные условия.

После этого пользователь может запустить симуляцию, после проверки корректности введенных данных симуляция запуститься и будет визуализирована в главном окне.

По окончанию работы симуляции будет создано два файла, лог всей симуляции в котором описаны все действия водителей в каждом шагу симуляции (1 шаг 1/5 секунды). И файл событий, в котором будут перечислены все виды аварийных событий, а именно тип события, детали события и шаг на котором произошло событие. События будут делиться на разные уровни важности, для удобства последующего анализа, например, попытка перестроения на другую полосу для опережения автомобиля является событием 1 уровня, а ДТП будет событием 4 уровня. Также будет вестись регистрация событий обгона, экстренного возвращения на свою дорогу, экстренного торможения для возвращения.

В итоге в планируемой программе для симуляции проездов перекрестков вне населенных пунктов должны присутствовать следующие базовые функции:

– симулятор должен обеспечивать точное и реалистичное моделирование транспортного потока, учитывая разнообразные характеристики транспорта, их скорости, направления движения, плотность и потоки на участках дорог вне населенных пунктов;

– симулятор должен учитывать воздействие различных погодных условий (дождь, снег, туман) на безопасность движения и обеспечивать реалистичное моделирование таких сценариев;

– симулятор должен предоставлять средства документирования результатов симуляций.

1.3 Выводы по первой главе

В первой главе был проведен анализ и причины совершения ДТП вне населённых пунктах, также приведены возможное технологические решения для предотвращения ДТП, проанализированы существующее программное обеспечение для моделирования транспортных систем и дорожного движения, а именно «Aimsun» и «PTV VISSIM». В результате рассмотрения этих аналогов были выявлены их преимущества и недостатки.

После сравнения инструментов были сформированы требования к разрабатываемой программе симулятору. На основе требований были рассмотрены технологии необходимые для реализации продукта.

2 Проектирование приложения симулятора

2.1 Проектирование вариантов использования

Исходя из требований системы были составлены диаграммы прецедентов. Данные диаграммы описывают как будет реализовано взаимодействие пользователя с системой.

Название прецедента: Ввод параметров для симуляции.

Цель сценария: Ввести параметры для симуляции.

Предусловие: Пользователь переходит в окно ввода параметров.

Основной сценарий: После перехода в окно ввода параметров, пользователь заполняет форму и нажимает кнопку для ее отправки. Диаграмма прецедента изображена на рисунке 3.

Постусловие: После корректного входа пользователю доступна возможность запустить симуляцию с заданными параметрами.

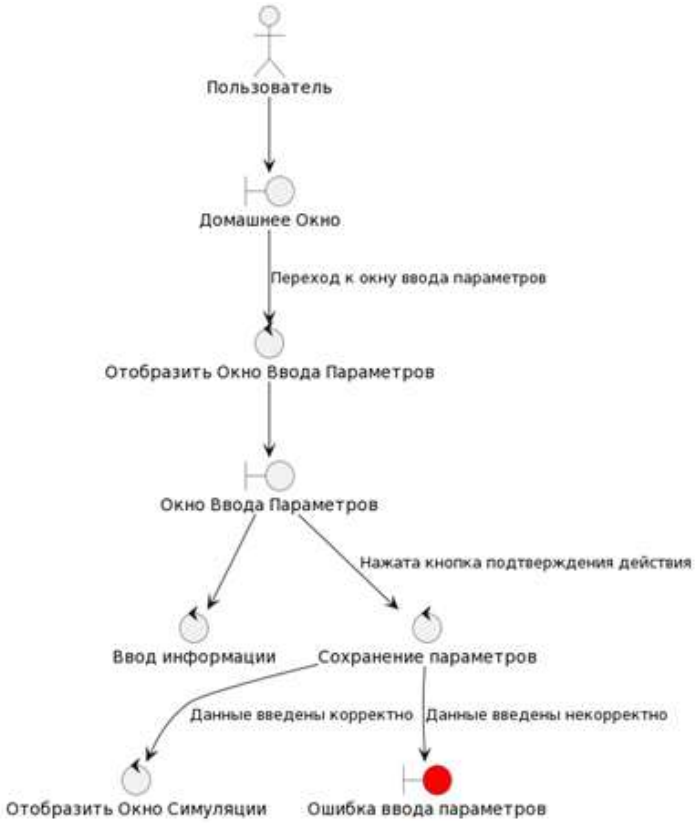


Рисунок 3 – Диаграмма прецедента ввода данных

Название прецедента: Запуск симуляции.

Цель сценария: Пользователь запускает симуляцию.

Предусловие: Пользователь должен ввести корректные данные для симуляции.

Основной сценарий: Пользователь переходит в окно запуска симуляции и запускает симуляцию, по прошествии необходимого времени выполнении, симуляция закончиться и результаты будут помещены в таблицу. Диаграмма прецедента изображена на рисунке 4.

Постусловие: Пользователь получает таблицу с результатами симуляции.

Рисунок 4 – Диаграмма прецедента запуска симуляции

2.2 Выбор инструментов

Для реализации приложения под систему Windows будет использоваться среда PyCharm и язык программирования Python, также будут использоваться библиотека Pygame.

Python выбран за его читаемость, простоту использования и мощную поддержку научных вычислений через библиотеки, такие как NumPy и SciPy, которые предлагают обширные возможности для математических расчетов и обработки данных.

Для создания визуализации используется Pygame. Эта библиотека позволяет разрабатывать приложения с графическим интерфейсом и анимацией.

2.3 Графический интерфейс

Графический интерфейс приложения представляет собой окно с рабочим пространством с кнопками, отвечающими за различные функции. Кнопки отвечают за использование параметров, запуска симуляции и отображение окна результатов.

Например, при нажатии кнопки «Параметры» будет вызвано дополнительное окно с настройками для создания симуляции с заданными условиями и параметрами. После указания параметров симуляции и нажатия кнопки «Применить» окно настройки закроется, а в рабочем пространстве основного окна появится предварительный результат создания перекрестка симуляции. Если результат создания симуляции не удовлетворил пользователя, он может изменить параметры в окне настройки и снова нажать кнопку «Применить», результат в основном окне изменится. При достижении удовлетворительного результата пользователь должен нажать кнопку «Запуск» и симуляция начнёт выполнение. На рисунке 7 представлено основное окно приложения.

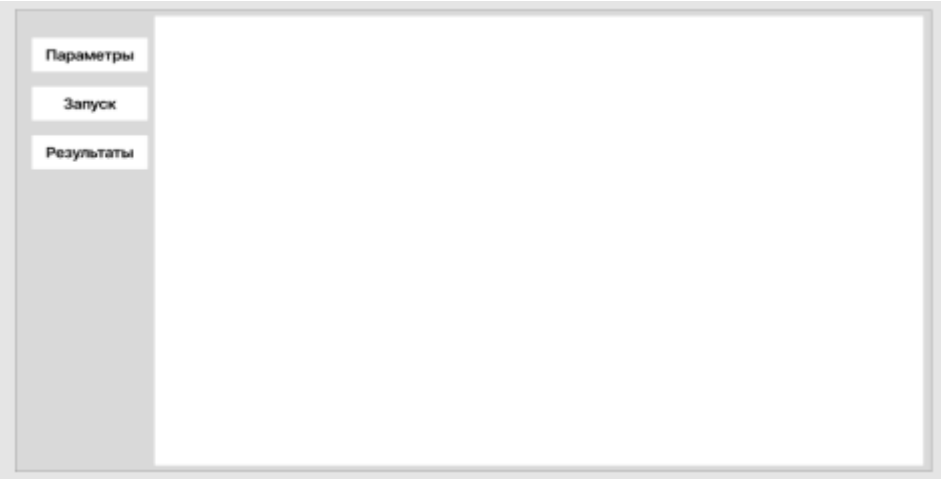


Рисунок 7 – Основное окно приложения

На рисунке 8 представлено одно из окон настройки

Рисунок 8 – Окно настройки

2.4 Вывод по второй главе

В результате выполнения главы были разработаны графический интерфейс и диаграммы прецедентов, которые показывают функциональные возможности программы. Также были предварительно рассмотрены и выбраны инструменты для разработки.

3 Разработка симулятора

В процессе проектирования и создания симулятора была создана диаграмма классов, которая отображена на рисунке 9.

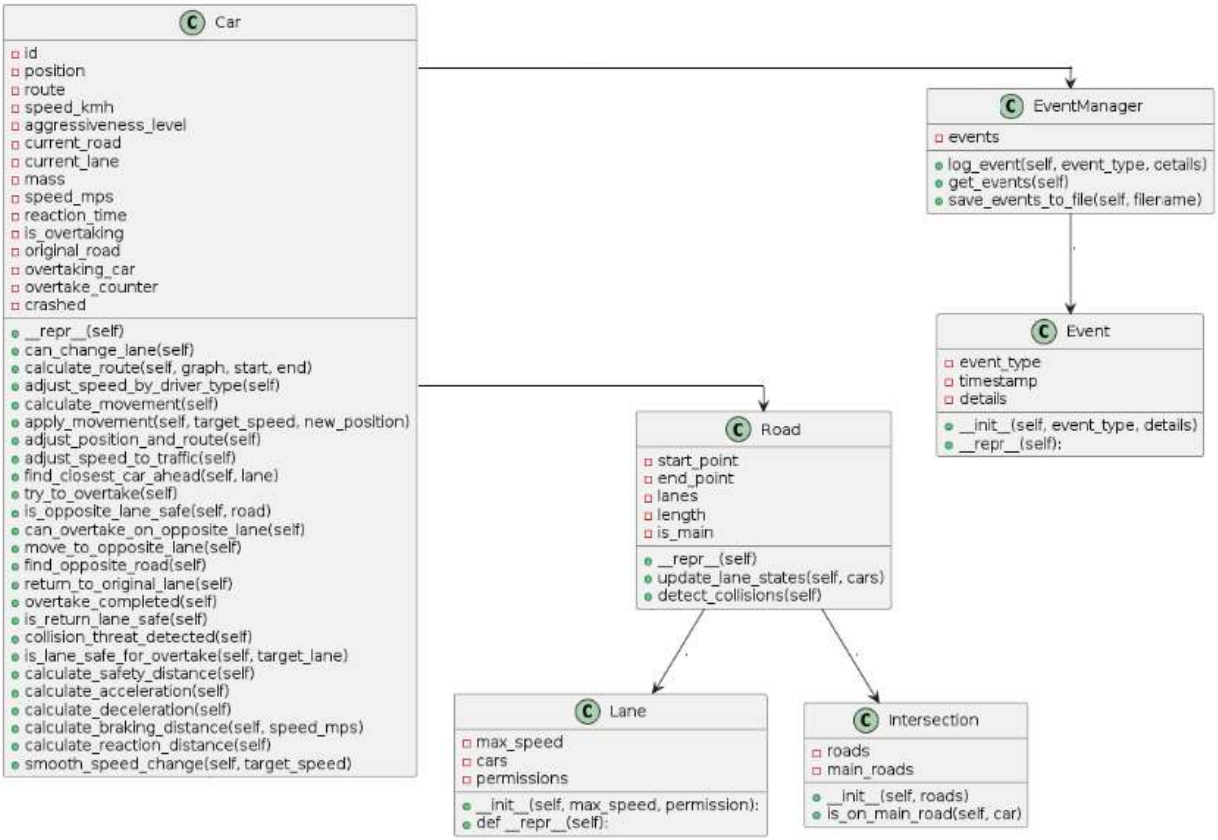


Рисунок 9 – Диаграмма классов

3.1 Разработка сущностей

Первым этапом разработки является создание сущностей. Каждая сущность представляется в виде класса. К самым главным сущностям относятся классы Car, Road, Lane и Intersection, именно с ними происходят главные взаимодействия в процессе симуляции, также есть несколько

вспомогательных классов таких как Event и EventManager, далее будут подробно рассмотрены все эти классы и их методы.

3.1.1 Описание класса Car

Класс Car представляет автомобиль в симуляции движения. Этот класс содержит все необходимые атрибуты и методы для моделирования поведения автомобиля на дороге. Основная цель класса – управлять поведением автомобиля в зависимости от его характеристик, таких как скорость, агрессивность водителя, и текущее состояние на дороге (например, обгон других автомобилей или реакция на дорожные условия). Далее перечислены основные аспекты класса Car.

Инициализация и идентификация:

– каждый автомобиль инициализируется с уникальным идентификатором, начальным и конечным пунктом маршрута, начальной скоростью и другими параметрами, такими как масса и уровень агрессивности;

– автомобиль вычисляет свой маршрут по предоставленному графу дорог и начинает движение с начальной точки.

Управление движением:

– автомобиль может регулировать свою скорость в зависимости от условий дорожного движения и личных характеристик водителя (например, уровень агрессивности);

– реализованы методы для обгона, включая проверку возможности перестроения и безопасного возвращения на полосу.

Детектирование и обработка столкновений:

– автомобиль может обнаруживать потенциальные столкновения на дороге и адаптировать своё поведение для избежания аварийных ситуаций.

Логика обгона:

– специфические методы для обгона, включая возможность обгона по встречной полосе и алгоритмы для безопасного возвращения на исходную полосу после обгона.

Прочие функции:

– класс включает в себя подробные методы для расчёта безопасного расстояния, скорости торможения и реакционного расстояния, а также обработку изменений в маршруте автомобиля.

Этот класс является ключевым компонентом симуляции, так как он определяет, как ведёт себя каждый автомобиль в различных дорожных условиях. В целом, класс Car помогает создать динамичную и реалистичную симуляцию движения транспортных средств. На рисунке 10 представлен фрагмент кода с описанием класса Car.

```
class Car:
    id_counter = 0

    def __init__(self, graph, start, end, speed_kmh=60, lane=0, aggressiveness_level=5, mass=1500, friction=0.7):
        Car.id_counter += 1
        self.id = Car.id_counter
        self.position = 0
        self.route = self.calculate_route(graph, start, end)
        self.base_speed_kmh = speed_kmh
        self.speed_kmh = speed_kmh
        self.aggressiveness_level = aggressiveness_level
        self.current_road = self.route[0] if self.route else None
        self.current_lane = lane
        self.mass = mass
        self.friction = friction
        self.reaction_time = 1.0
        self.speed_mps = (self.speed_kmh * 1000) / 3600
        self.is_overtaking = False
        self.original_road = None
        self.overtaking_car = None
        self.overtake_counter = 0
        self.crashed = False
        self.crash_time = 0


    def __repr__(self):
        return f"Car {self.id} (Aggressiveness level: {self.aggressiveness_level}) with speed {self.speed_kmh} km/h at position {self.position:.2f}"
```

Рисунок 10 – Класс Car

3.1.2 Описание класса Road

Класс Road представляет собой дорогу в симуляции транспортного потока. Этот класс включает в себя всю информацию о дороге, такую как

начальная и конечная точки, информация о полосах движения, длина дороги и статус основной дороги (если это применимо), далее приведены основные аспекты класса Road.

Инициализация:

– при создании экземпляра дороги задаются её начальная и конечная точки, информация о полосах (максимальная скорость и разрешения на полосах) и длина;

– дорога может быть помечена как главная, что может влиять на поведение автомобилей (например, приоритет на перекрёстках).

Управление полосами:

– класс содержит список объектов Lane, каждый из которых представляет собой полосу движения с определёнными атрибутами, такими как максимальная скорость и правила движения;

– метод update_lane_states обновляет состояние полос, например, очищает список автомобилей на каждой полосе и заново заполняет их на основе текущих позиций автомобилей.

Детектирование столкновений:

– метод detect_collisions проверяет наличие столкновений на дороге. Этот метод анализирует расстояние между автомобилями в каждой полосе и регистрирует столкновения, обновляя состояние задействованных автомобилей и регистрируя соответствующие события.

Представление дороги:

– метод __repr__ предоставляет строковое представление объекта полосы, включающее максимальную скорость и текущее количество автомобилей на полосе. Это облегчает отладку и мониторинг состояния полос в процессе симуляции.

Класс Road является ещё одним важным компонентом для моделирования дорожной сети, обеспечивая основу для движения и взаимодействия автомобилей в симуляции. Он позволяет управлять комплексными аспектами дорожного движения и обеспечивает

инфраструктуру для реалистичного моделирования транспортных потоков. Класс Road представлен на рисунке 11.

```
class Road:
    def __init__(self, start_point, end_point, lanes_info, length, is_main=False):
        self.start_point = start_point
        self.end_point = end_point
        self.lanes = [Lane(info['max_speed'], info['permission']) for info in lanes_info]
        self.length = length
        self.is_main = is_main

    def __repr__(self):
        return f"Road from {self.start_point} to {self.end_point}, {len(self.lanes)} lanes, {self.length} meters long"
```

Рисунок 11 – Класс Road

3.1.3 Описание класса Lane

Класс Lane представляет отдельную полосу движения на дороге в рамках симуляции транспортного потока. Этот класс включает в себя всю необходимую информацию для управления полосой и её характеристиками, такими как максимально разрешённая скорость и правила движения по полосе, основные аспекты класса Lane перечислены далее.

Инициализация:

– каждая полоса инициализируется с указанием максимальной скорости и разрешений, которые определяют возможные манёвры на этой полосе (например, поворот направо, налево или движение прямо);

– класс также содержит список автомобилей, находящихся на этой полосе, что позволяет управлять движением по полосе и отслеживать состояние транспортных средств.

Управление автомобилями:

– полоса содержит список автомобилей, которые в данный момент находятся на ней. Это включает в себя функциональность добавления и удаления автомобилей, что необходимо для актуализации движения транспорта при изменении условий на дороге или после выполнения манёвров.

Представление полосы:

– метод __repr__ предоставляет строковое представление объекта полосы, включающее максимальную скорость и текущее количество автомобилей на полосе. Это облегчает отладку и мониторинг состояния полос в процессе симуляции.

Класс Lane играет ключевую роль в управлении движением на дорогах внутри симуляции. Он обеспечивает детализацию дорожной сети, позволяя точно моделировать различные сценарии движения и поведения автомобилей на разных полосах. Класс служит важным строительным блоком для класса Road, составляя основу для создания многослойных и реалистичных дорожных условий. Класс Lane представлен на рисунке 12.

```
class Lane:
    def __init__(self, max_speed, permission):
        self.max_speed = max_speed
        self.cars = []
        self.permissions = permission

    def __repr__(self):
        return f"Lane with max speed {self.max_speed} km/h"
```

Рисунок 12 – Класс Lane

3.1.4 Описание класса Event и EventManager

Класс Event представляет событие в симуляции транспортного потока. Этот класс используется для записи различных действий или изменений в состоянии системы, например столкновений или изменений в движении автомобилей, далее перечислены аспекты класса Event.

Инициализация:

– событие инициализируется с типом события и деталями, которые описывают, что именно произошло. Это может включать данные о столкновениях, перестроениях или других значимых изменениях в симуляции;

– каждое событие также содержит временную метку (timestamp), которая указывает на шаг симуляции, когда событие произошло, обеспечивая хронологическую точность.

Класс EventManager управляет регистрацией и хранением всех событий в симуляции. Этот класс представляет собой центральное хранилище для всех событий, позволяя легко доступ к историческим данным для анализа или отчётности, аспекты класса EventManager представлены далее.

Инициализация:

– при создании экземпляра класса создаётся пустой список событий, который будет использоваться для хранения всех зарегистрированных событий.

Регистрация и извлечение событий:

– метод log_event используется для создания и добавления нового события в список. Этот метод принимает тип и детали события, создаёт экземпляр Event и добавляет его в список событий;

– метод get_events предоставляет доступ ко всем текущим зарегистрированным событиям, что может быть полезно для анализа или мониторинга состояния симуляции.

Сохранение событий в файл:

– метод save_events_to_file позволяет сохранить все события в файл, что облегчает долгосрочное хранение данных и анализ вне контекста программы. Формат сохранения включает временные метки и описания событий, что делает данные легко интерпретируемыми.

Классы Event и EventManager вместе формируют систему управления событиями для симуляции, обеспечивая регистрацию, хранение и анализ важных изменений и действий во время выполнения симуляции. Классы Event и EventManager изображены на рисунке 13.

```python
class Event:
    def __init__(self, event_type, details):
        self.event_type = event_type
        self.timestamp = step
        self.details = details

    def __repr__(self):
        return f"Event(type={self.event_type}, details={self.details}, time={self.timestamp})"


class EventManager:
    def __init__(self):
        self.events = []

    def log_event(self, event_type, details):
        event = Event(event_type, details)
        self.events.append(event)
        print(event)

    def get_events(self):
        return self.events

    def save_events_to_file(self, filename):
        with open(filename, 'w') as file:
            for event in self.events:
                file.write(f"{event.timestamp}: {event.event_type} at step {event.details}\n")
```

Рисунок 13 – Классы Event и EventManager

3.2 Разработка логики симуляции

Логику симуляции можно условно разбить на два блока: Основной блок симуляции, включающий в себя основное тело цикла симуляции, а также вспомогательные методы для отрисовки дорог, машин и их движения, и другие вспомогательные функции. Второй блок — это отдельные функции расчета логики, например, логики реагирования, расчёты скорости и так далее. Далее мы рассмотрим оба блока и функции, которые находятся в этих блоках.

3.2.1 Основной блок симуляции

Ключевой функцией блока является метод simulate_traffic. Этот метод является основным механизмом управления выполнением симуляции транспортного потока в программе. Он интегрирует различные компоненты системы (автомобили, дороги, события) и управляет их взаимодействием и обновлением состояний во времени. Основным предназначением метода является моделирование поведения автомобилей на дорожной сети в заданный период времени.

Он выполняет следующие функции:

– инициализация и управление временем симуляции: устанавливает параметры окружения Pygame, такие как размеры окна, частоту кадров и шрифт для отображения текста;

– циклическое обновление сценария: Метод управляет циклом симуляции, в которой на каждом шаге происходит обновление состояний всех объектов, и отрисовка графики.

Далее представлены детали реализации метода simulate_traffic. В самом начале идёт инициализация Pygame, а именно запускается сам Pygame, настраивается окно для отображения и создаются объекты для управления временем и отображения текста.

Основной цикл игры:

– обработка событий Pygame: реагирует на события, такие как закрытие окна, чтобы корректно завершить работу программы;

– очистка экрана: подготавливает экран к новому кадру, заполняя его черным цветом.

Обновление и отрисовка дорог:

– обновляет состояние каждой дороги, распределяя автомобили по полосам. Отрисовывает каждую дорогу на экране, используя функцию draw_road, функция draw_road продемонстрирована на рисунке 14.

```python
def draw_road(screen, road):
    lane_width = 10
    lane_count = len(road.lanes)
    road_start_x, road_start_y = road.start_point
    road_end_x, road_end_y = road.end_point

    # Определяем направление дороги
    horizontal = abs(road_start_x - road_end_x) > abs(road_start_y - road_end_y)

    for i in range(lane_count):
        if horizontal:
            # Горизонтальное направление
            if road_start_x < road_end_x:
                # Слева направо: полосы под дорогой
                offset_y = -(i + 1) * lane_width
            else:
                # Справа налево: полосы над дорогой
                offset_y = (i + 1) * lane_width
            pygame.draw.line(screen, (200, 200, 200), (road_start_x, road_start_y + offset_y),
                             (road_end_x, road_end_y + offset_y), 5)
        else:
            # Вертикальное направление
            if road_start_y < road_end_y:
                # Сверху вниз: полосы справа от дороги
                offset_x = (i + 1) * lane_width
            else:
                # Снизу вверх: полосы слева от дороги
                offset_x = -(i + 1) * lane_width
            pygame.draw.line(screen, (200, 200, 200), (road_start_x + offset_x, road_start_y),
                             (road_end_x + offset_x, road_end_y), 5)
```

Рисунок 14 – Функция draw_road

Детекция столкновений:

– проверяет каждую дорогу на предмет столкновений и обрабатывает возникающие столкновения, исключая поврежденные автомобили из активной симуляции.

Обновление состояний автомобилей:

– рассчитывает новое состояние (скорость и позицию) каждого автомобиля и применяет эти изменения.

Отрисовка автомобилей:

– отображает каждый автомобиль на экране, используя функцию draw_car, Функция draw_car продемонстрирована на рисунке 15.

```python
def draw_car(screen, car, font):
    if car.current_road:
        road_length = car.current_road.length
        road_start_x, road_start_y = car.current_road.start_point
        road_end_x, road_end_y = car.current_road.end_point
        pos_ratio = car.position / road_length
        lane_width = 10
        horizontal = abs(road_start_x - road_end_x) > abs(road_start_y - road_end_y)

        if horizontal:
            # Горизонтальное направление
            car_pos_x = int(road_start_x + (road_end_x - road_start_x) * pos_ratio)
            if road_start_x < road_end_x:
                # Слева направо
                car_pos_y = int(road_start_y - (car.current_lane + 1) * lane_width)
            else:
                # Справа налево
                car_pos_y = int(road_start_y + (car.current_lane + 1) * lane_width)
        else:
            # Вертикальное направление
            car_pos_y = int(road_start_y + (road_end_y - road_start_y) * pos_ratio)
            if road_start_y < road_end_y:
                # Сверху вниз
                car_pos_x = int(road_start_x + (car.current_lane + 1) * lane_width)
            else:
                # Снизу вверх
                car_pos_x = int(road_start_x - (car.current_lane + 1) * lane_width)

        car_color = (255, 0, 0) if not car.crashed else (0, 255, 0)  # Зеленый если в ДТП
        pygame.draw.circle(screen, car_color, (car_pos_x, car_pos_y), 5)

        speed_surface = font.render(f'{round(car.speed_kmh)} km/h', True, (255, 255, 255))
        aggression_surface = font.render(f'Agg: {car.aggressiveness_level}', True, (66, 170, 255))
        id_surface = font.render(f'{car.id}', True, (255, 255, 255))

        screen.blit(id_surface, (car_pos_x - 7, car_pos_y - 20))
        screen.blit(speed_surface, (car_pos_x + 15, car_pos_y + 10))
        screen.blit(aggression_surface, (car_pos_x + 15, car_pos_y + 20))
```

Рисунок 15 – Функция draw_car

Добавление новых автомобилей:

– периодически генерирует новые автомобили для поддержания активности трафика в симуляции. Функция create_car продемонстрирована на рисунке 16.

```
def create_car(graph, points):
    start = random.choice(points)
    end = random.choice([p for p in points if p != start])
    route = find_shortest_path(graph, start, end)
    if not route:
        return None
    speed = random.randint(40, 100)
    lane = random.randint(0, len(route[0].lanes) - 1)
    aggressiveness_level = random.randint(1, 10)
    return Car(graph, start, end, speed, lane, aggressiveness_level)
```

Рисунок 16 – Функция create_car

Обновление экрана и синхронизация времени:

– обновляет содержимое экрана и управляет темпом симуляции, используя встроенные инструменты Pygame для управления временем.

Логирование событий:

– записывает происходящие события в лог-файл, предоставляя данные для анализа после завершения симуляции.

Завершение симуляции:

– завершает выполнение Pygame и освобождает ресурсы после достижения заданного количества шагов или при закрытии программы пользователем.

Метод simulate_traffic предоставляет полный контроль и управление симуляцией, включая визуализацию, обработку данных и управление временем, что делает его ключевым компонентом в структуре программы моделирования транспортного потока, функция simulate_traffic продемонстрирован на рисунке 17.

```python
def simulate_traffic(cars, graph, points, steps=5000, new_car_interval=15):
    pygame.init()
    screen = pygame.display.set_mode((1000, 600))
    clock = pygame.time.Clock()
    font = pygame.font.Font(None, 16)
    running = True
    dtp = 0
    global step
    while running and step < steps:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
        screen.fill((0, 0, 0))
        all_cars = set(cars)
        for destination_info in graph.values():
            for road in destination_info.values():
                draw_road(screen, road)
                road.update_lane_states(cars)
                crashed_cars = road.detect_collisions()
                if crashed_cars:
                    dtp += 1
                all_cars.difference_update(crashed_cars)
        updated_cars = []
        cars = list(all_cars)
        for car in cars:
            draw_car(screen, car, font)
            car.calculate_movement()
            car.apply_movement(car.speed_kmh, car.position + car.speed_mps * 0.2)
            print(f"{car} on {car.current_road} in lane {car.current_lane}")
            if car.current_road:
                updated_cars.append(car)
        cars = updated_cars
        if step % new_car_interval == 0:
            new_car = create_car(graph, points)
            if new_car:
                cars.append(new_car)
                print(f"New car added: {new_car}")
        pygame.display.flip()
        clock.tick(30)
        step += 1
        print(f"Step {step}:")
        print(f"dtp {dtp}:")
        event_manager.save_events_to_file(filename="event_log.txt")
    pygame.quit()
```

Рисунок 17 – Функция simulate_traffic

3.2.2 Второй блок симуляции

В функции simulate_traffic вызывается метод класса Car – calculate_movement. Этот метод отвечает за вычисление новой скорости и позиции автомобиля на текущем шаге симуляции. Этот метод учитывает характеристики водителя, состояние дорожного движения и различные

физические параметры, чтобы смоделировать реалистичное поведение автомобиля.

Регулировка скорости автомобиля в зависимости от типа водителя:

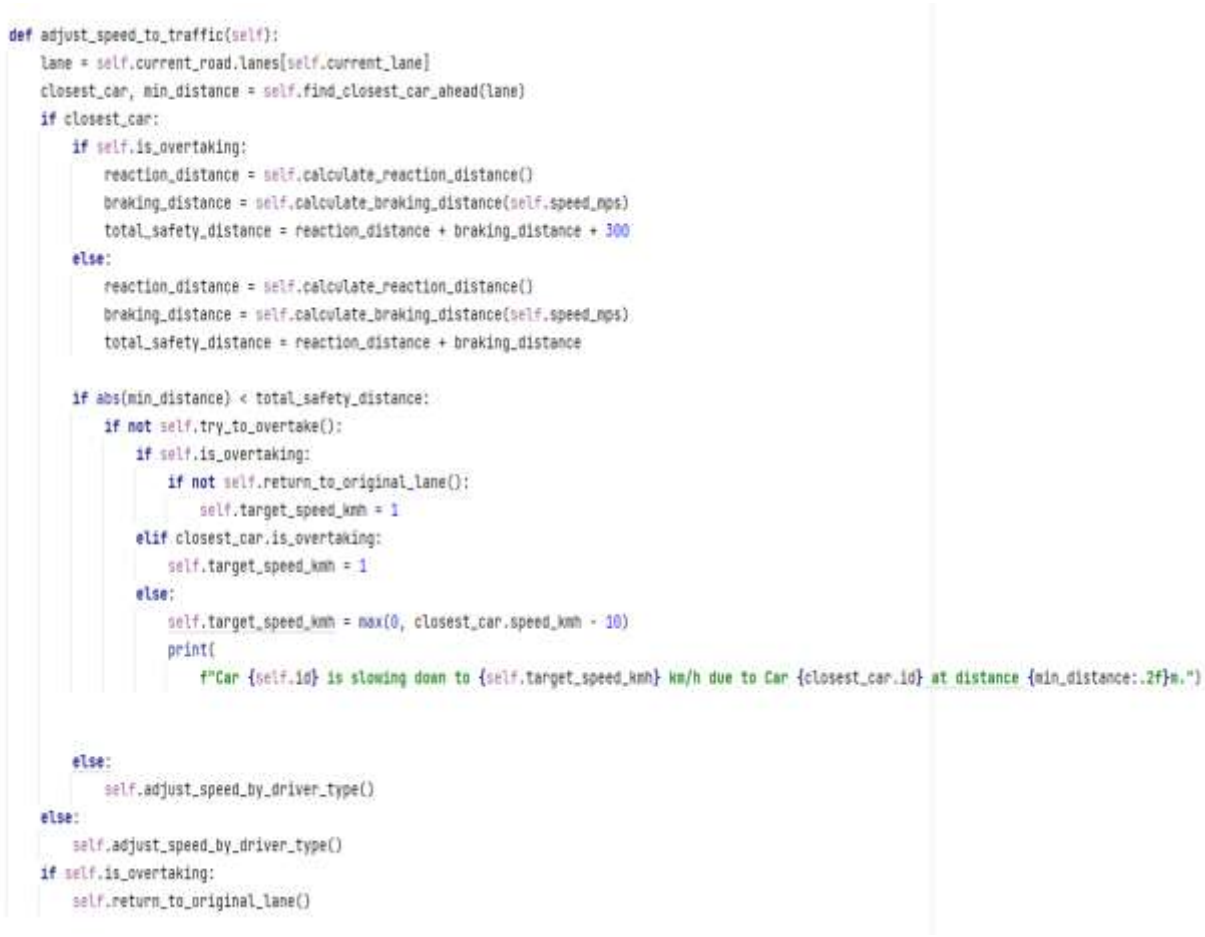
– вызывается метод adjust_speed_by_driver_type который корректирует целевую скорость автомобиля в зависимости от уровня агрессивности водителя и ограничений скорости на текущей полосе. Метод adjust_speed_by_driver_type изображен на рисунке 18.

```
def adjust_speed_by_driver_type(self):
    lane_speed_limit = self.current_road.lanes[self.current_lane].max_speed
    aggression_factor = (self.aggressiveness_level - 5) / 10.0
    self.target_speed_kmh = lane_speed_limit * (1 + aggression_factor)
```

Рисунок 18 – Метод adjust_speed_by_driver_type

Адаптация скорости к текущим дорожным условиям:

– в методе adjust_speed_to_traffic учитывается наличие других автомобилей на полосе и их скорость, чтобы предотвратить столкновения и обеспечить безопасное движение, Метод adjust_speed_to_traffic изображен на рисунке 19.



```
def adjust_speed_to_traffic(self):
    lane = self.current_road.lanes[self.current_lane]
    closest_car, min_distance = self.find_closest_car_ahead(lane)
    if closest_car:
        if self.is_overtaking:
            reaction_distance = self.calculate_reaction_distance()
            braking_distance = self.calculate_braking_distance(self.speed_mps)
            total_safety_distance = reaction_distance + braking_distance + 300
        else:
            reaction_distance = self.calculate_reaction_distance()
            braking_distance = self.calculate_braking_distance(self.speed_mps)
            total_safety_distance = reaction_distance + braking_distance

        if abs(min_distance) < total_safety_distance:
            if not self.try_to_overtake():
                if self.is_overtaking:
                    if not self.return_to_original_lane():
                        self.target_speed_kmh = 1
                elif closest_car.is_overtaking:
                    self.target_speed_kmh = 1
                else:
                    self.target_speed_kmh = max(0, closest_car.speed_kmh - 10)
                    print(
                        f"Car {self.id} is slowing down to {self.target_speed_kmh} km/h due to Car {closest_car.id} at distance {min_distance:.2f}m.")


        else:
            self.adjust_speed_by_driver_type()
    else:
        self.adjust_speed_by_driver_type()
    if self.is_overtaking:
        self.return_to_original_lane()
```

Рисунок 19 – Метод adjust_speed_to_traffic

Вычисление ускорения или торможения:

– рассчитывает, насколько автомобиль должен ускориться или замедлиться, исходя из разницы между текущей и целевой скоростью, а также физических характеристик автомобиля.

Обновление позиции автомобиля:

– определяет новую позицию автомобиля на дороге, исходя из его текущей скорости и состояния (например, обгоняет ли он другое транспортное средство).

Метод calculate_movement выполняет ключевую роль в симуляции, обеспечивая реалистичное моделирование движения автомобиля в зависимости от различных факторов, таких как агрессивность водителя, дорожные условия и физические характеристики автомобиля. Этот метод интегрирует различные аспекты поведения автомобиля для достижения

правдоподобного результата в симуляции, метод calculate_movement изображен на рисунке 20.

```
def calculate_movement(self):
    self.adjust_speed_by_driver_type()
    self.adjust_speed_to_traffic()
    self.check_intersection()
    acceleration = self.calculate_acceleration()
    time_step = 0.2  # шаг времени в секундах для каждого обновления

    if self.target_speed_kmh > self.speed_kmh:
        # Ускорение
        speed_change = min((self.target_speed_kmh - self.speed_kmh), (acceleration * time_step * 3600) / 1000)
    else:
        # Торможение
        current_braking_distance = self.calculate_braking_distance(self.speed_mps)
        if current_braking_distance == 0:
            current_braking_distance = 1  # avoid division by zero
        deceleration = (self.speed_mps ** 2) / (2 * current_braking_distance)
        speed_change = max((self.target_speed_kmh - self.speed_kmh), (-deceleration * time_step * 3600) / 1000)

    self.speed_kmh += speed_change
    self.speed_mps = (self.speed_kmh * 1000) / 3600

    if self.is_overtaking:
        new_position = self.position - (self.speed_mps * time_step)
        if new_position <= 0:
            self.update_road_on_overtake()
            new_position = self.position - (self.speed_mps * time_step)
        self.position = new_position
        self.speed_mps = self.speed_mps * -1
    else:
        new_position = self.position + self.speed_mps * time_step

    return self.speed_kmh, new_position
```

Рисунок 20 – Метод calculate_movement

Если в методе adjust_speed_to_traffic обнаружена впереди идущая машина с помощью метода find_closest_car_ahead который изображен на рисунке 21 и, если расстояние до этой машины меньше рассчитанного безопасного расстояния тогда вызывается проверка метода try_to_overtake.

```
def find_closest_car_ahead(self, lane):
    closest_car = None
    min_distance = 100
    for car in lane.cars:
        if self.is_overtaking:
            if car != self and car.position < self.position:
                distance = car.position - self.position
                if abs(distance) < min_distance:
                    min_distance = distance
                    closest_car = car
        else:
            if car != self and car.position > self.position:
                distance = car.position - self.position
                if distance < min_distance:
                    min_distance = distance
                    closest_car = car

    return closest_car, min_distance
```

Рисунок 21 – Метод find_closest_car_ahead

Метод try_to_overtake отвечает за попытку автомобиля перестроиться на другую полосу для опережения или обгона по встречной полосе, если текущая ситуация на дороге позволяет это сделать. Этот метод учитывает агрессивность водителя, наличие свободных полос и возможность безопасного выполнения манёвра обгона, далее перечислены задачи, которые выполняет данный метод.

Проверка текущего состояния обгона:

– определяет, находится ли автомобиль уже в процессе обгона.

Проверка возможности перестроения на левую или правую полосу:

– оценивает возможность безопасного перестроения на соседние полосы (как левые, так и правые) с помощью метода is_lane_safe_for_overtake который изображен на рисунке 22.

```
def is_lane_safe_for_overtake(self, target_lane):
    safety_distance = self.calculate_safety_distance()
    for car in target_lane.cars:
        if abs(car.position - self.position) < safety_distance:
            return False
    return True
```

Рисунок 22 – Метод is_lane_safe_for_overtake

Проверка возможности обгона по встречной полосе:

– если автомобиль находится на самой левой полосе, проверяется возможность обгона по встречной полосе при высокой агрессивности водителя. Вызывается ряд методов для проверки соседний встречной полосы, методы изображены на рисунке 23.

```
def can_overtake_on_opposite_lane(self):
    opposite_road = self.find_opposite_road()
    if opposite_road and self.is_opposite_lane_safe(opposite_road):
        return True
    return False

def is_opposite_lane_safe(self, road):
    if len(road.lanes[0].cars) == 0:
        return True  # Полоса свободна для перестроения

    # Проверка безопасности на встречной полосе
    for car in road.lanes[0].cars:
        if (car.position + self.position) < self.current_road.length:
            if abs(self.current_road.length - (car.position + self.position)) < 100:  # 100 метров для безопасности
                return False
    return True
```

Рисунок 23 – Методы для проверки встречной полосы

Выполнение манёвра обгона:

– если перестроение возможно и безопасно, автомобиль выполняет обгон, изменяя свою полосу и состояние, если выполняется обгон через встречную полосу, то вызывается соответствующий метод move_to_opposite_lane который изображён на рисунке 24.

```
def can_overtake_on_opposite_lane(self):
    opposite_road = self.find_opposite_road()
    if opposite_road and self.is_opposite_lane_safe(opposite_road):
        return True
    return False

def is_opposite_lane_safe(self, road):
    if len(road.lanes[0].cars) == 0:
        return True  # Полоса свободна для перестроения

    # Проверка безопасности на встречной полосе
    for car in road.lanes[0].cars:
        if (car.position + self.position) < self.current_road.length:
            if abs(self.current_road.length - (car.position + self.position)) < 100:  # 100 метров для безопасности
                return False
    return True
```

Рисунок 24 – Метод move_to_opposite_lane

Метод try_to_overtake играет важную роль в обеспечении динамики движения в симуляции, позволяя моделировать агрессивное поведение водителей и сложные манёвры обгона, метод try_to_overtake изображен на рисунке 25.

```
def try_to_overtake(self):
    # Проверяем, не находится ли автомобиль уже в процессе обгона

    if not self.is_overtaking:
        # Проверка возможности перестроения в левую полосу
        if self.aggressiveness_level > 2 and self.current_lane - 1 >= 0:
            left_lane = self.current_road.lanes[self.current_lane - 1]
            if self.is_lane_safe_for_overtake(left_lane):
                if self.overtake_counter >= (15 - (self.aggressiveness_level / 2)):
                    self.overtake_counter = 0
                    self.current_lane -= 1
                    return True

        # Проверка возможности перестроения в правую полосу
        elif self.aggressiveness_level > 2 and self.current_lane + 1 < len(self.current_road.lanes):
            right_lane = self.current_road.lanes[self.current_lane + 1]
            if self.is_lane_safe_for_overtake(right_lane):
                if self.overtake_counter >= (15 - (self.aggressiveness_level / 2)):
                    self.overtake_counter = 0
                    self.current_lane += 1
                    return True

        # Проверка возможности обгона по встречной полосе
        elif self.aggressiveness_level > 6 and self.current_lane == 0 and not self.is_overtaking and\
                self.can_overtake_on_opposite_lane() and self.position < self.current_road.length * 0.8:
            self.overtaking_car, _ = self.find_closest_car_ahead(self.current_road.lanes[self.current_lane])
            if self.overtaking_car:
                if self.overtake_counter >= (15 - (self.aggressiveness_level / 2)):
                    self.overtake_counter = 0
                    self.move_to_opposite_lane()
                    return True

        self.overtake_counter += 1
    return False
```

Рисунок 25 – Функция try_to_overtake

3.3 Вывод по третьей главе

В третьей главе представлен подход к построению логики для симулятора проезда перекрестков вне населенных пунктах, приведены подробные примеры функций и методов.

В результате разработан симулятор, который соответствует всем поставленным требованиям, симулятор обеспечивает умеренно точное и достаточно реалистичное моделирование транспортного потока, учитывая разнообразные характеристики транспорта, их скорости, направления движения, плотность и потоки на участках дорог вне населенных пунктов, а также учитывает воздействие различных погодных условий. Основа симулятора написана так чтобы была возможность последующего улучшения и добавления новых функций.

ЗАКЛЮЧЕНИЕ

В результате работы был спроектирован и реализован симулятор проезда перекрёстков вне населённых пунктов.

В ходе выполнения работы был произведен анализ существующих аналогов симуляторов, выбраны средства разработки, обозначен функционал программного продукта и выполнена его практическая реализация.

Симулятор реализован с использованием программной среды PyCharm, языка программирования Python с использованием дополнительных библиотек и модулей. Разработанный симулятор отвечает всем поставленным задачам и первоначальным задумкам. В процессе работы над симулятором удалось реализовать приемлемую логику поведения машин на дороге, создать инструмент для полного отслеживания и документирования всех важных аварийных и предаварийных событий, а также создать визуализацию всей симуляции для более информативного вывода.

В дальнейшем планируется внедрение функции дополнительных функций для общего улучшения симуляции.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ластовский, Л. В. Симулятор для проезда перекрестков вне населенных пунктов / Л. В. Ластовский // Научный аспект. Информационные технологии. – 2024. – № 5: официальный сайт. – 2024. – URL: https://na-journal.ru/5-2024-informacionnye-tekhnologii/12650-simulyator-dlya-proezda-perekrestkov-vne-naselennyh-punktov (дата обращения: 10.06.2024).

2. Сведения о показателях состояния безопасности дорожного движения // Официальный сайт Госавтоинспекции : официальный сайт. – 2024. – URL: https://stat.gibdd.ru/ (дата обращения: 2.06.2024).

3. Hegeman, G. Overtaking assistant assessment using traffic simulation / G. Hegeman, A. Tapani, S. Hoogendoorn // Transportation Research Part C: Emerging Technologies. – 2009. – Т. 17, № 6. – С. 617–630.

4. Richter, T. Causes, consequences and countermeasures of overtaking accidents on two-lane rural roads / T. Richter, S. Ruhl, J. Ortlepp, E. Bakaba // Transportation Research Procedia. – 2017. – № 25. – С.1989-2001.

5. Maosheng, L. Study on the Impact of Traffic Accidents in Key Areas of Rural Roads / L. Maosheng, X. Hui, S. Panpan // Accident Analysis and Sustainable Road Safety Research : Sustainability – 2021. – Т. 13, – № 14. – С. 7802.

6. Hillel, B. The tendency of drivers to pass other vehicles / B. Hillel, D. Shinar // Transportation Research Part F: Traffic Psychology and Behaviour. – 2005. – Т. 8, № 6. – С. 429–439.

7. Tavakoli, A. dentifying Significant Variables Influencing Overtaking Maneuvers on Two-lane, Two-way Rural Roads in Iran / A. Tavakoli, E. Ayazi. M. Ravasani // Periodica Polytechnica Transportation Engineering. – 2016. – Т. 44, № 3. – С. 155–163.

8. Asaithambi, G. Overtaking behaviour of vehicles on undivided roads in non-lane based mixed traffic conditions / G. Asaithambi, G. Shravani // Journal of

Traffic and Transportation Engineering (English Edition). – 2017. – Т. 4, № 3. – С. 252–261.

9. Патент № 2770723 Российская федерация, МПК G08G 1/0967 (2022.01), G08G 1/16 (2022.01), B60W 30/08 (2022.01), H04W 4/30 (2022.01), H04W 4/44 (2022.01). Способ предупреждения столкновений транспортных средств на участках дорог вне населенных пунктов : № 2021125343 : заявл. 26.08.2021 : опубл. 21.04.2022 / В. А. Зеер, Е. В. Гражданцев, А. Ю. Сидоров, Р. С. Глухих, Д. Л. Окладников, П. С. Литвинов – 48 с.

10. Aimsun digital mobility solutions : официальный сайт. – 2024. – URL: https://www.aimsun.com/ (дата обращения: 2.06.2024).

11. Multimodal Traffic Simulation Software : официальный сайт. – 2024. – URL: https://www.ptvgroup.com/en/products/ptv-vissim (дата обращения: 2.06.2024).

ПРИЛОЖЕНИЕ А

Справка о публикации

Общество с ограниченной ответственностью "Аспект"
443068, г.Самара, ул.Николая Панова, д.16, оф.34
ОГРН: 1126316002152 ИНН/КПП:
6316172906/631601001
оф. сайт: na-journal.ru e-mail: public@na-journal.ru

СПРАВКА № 05/24-20-300

О ПУБЛИКАЦИИ СТАТЬИ

Редакция журнала «Научный аспект» (свидетельство ПИ № ФС 77-84349, ISSN 2226-5694) подтверждает, что статья Ластовского Леонида Витальевича (научный руководитель – Макуха Любовь Витальевна) «Симулятор для проезда перекрестков вне населенных пунктов» включена в проект выпуска журнала «Научный аспект №5-2024».

- Статье присвоен номер 05/24-20-300 в базе издательства.
- Электронная версия статьи опубликована на официальном сайте журнала

https://na-journal.ru/5-2024-informacionnye-tekhnologii/12650-simulyator-dlya-proezda-perekrestkov-vne-naselennyh-punktov

Метаданные статьи индексируются в реферативной базе данных научных публикаций РИНЦ (ИФ 0,04).

Директор издательства
ООО «Аспект» Хасиятуллов Марат Габделахатович

Рисунок А.1 – Справка о публикации научной статьи

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
Кафедра вычислительная техника

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
«17» 06 2024 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 Информатика и вычислительная техника

Симулятор проезда перекрестков вне населенных пунктов

Руководитель _____ 17.06.24 старший преподаватель Л.В. Макуха
подпись *дата* *должность, ученая степень*

Выпускник _____ 17.06.24 Л.В. Ластовский
подпись *дата*

Нормоконтролёр _____ 17.06.24 Л.В. Макуха
подпись *дата*

Красноярск 2024