ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc198054750)

[1. Анализ существующих подходов к управлению автономными транспортными средствами 5](#_Toc198054751)

[1.1. Общая характеристика автономных автомобилей 5](#_Toc198054752)

[1.2. Обзор современных систем управления 6](#_Toc198054753)

[1.3. Методы искусственного интеллекта в управлении 8](#_Toc198054754)

[2. Теоретические основы генетических алгоритмов 10](#_Toc198054755)

[2.1. Основные принципы и компоненты генетического алгоритма 10](#_Toc198054757)

[2.2. Кодирование решений 11](#_Toc198054758)

[2.3. Операторы селекции, скрещивания и мутации 13](#_Toc198054759)

[2.4. Критерии завершения и адаптивные механизмы 15](#_Toc198054760)

[3. Разработка модели системы управления автономным автомобилем 17](#_Toc198054761)

[3.1. Постановка задачи и формализация модели 17](#_Toc198054765)

[3.2. Технологии 18](#_Toc198054770)

[3.3. Архитектура программного решения 19](#_Toc198054771)

[3.4. Представление окружающей среды и карты 21](#_Toc198054772)

[3.5. Реализация генетического алгоритма и алгоритма принятия решений 23](#_Toc198054773)

[4. Экспериментальное исследование и оценка эффективности 25](#_Toc198054774)

[4.1. Методика проведения экспериментов 25](#_Toc198054776)

[4.2. Анализ полученных результатов 25](#_Toc198054777)

[ЗАКЛЮЧЕНИЕ 28](#_Toc198054778)

[СПИСОК ЛИТЕРАТУРЫ 36](#_Toc198054779)

[ПРИЛОЖЕНИЕ 38](#_Toc198054780)

# 

# ВВЕДЕНИЕ

Разработка автономных транспортных средств является одним из приоритетных направлений современной науки и техники. В условиях стремительного роста городов, плотности движения, повышения требований к безопасности и энергоэффективности транспортных систем особое значение приобретает разработка интеллектуальных систем управления, способных адаптироваться к изменяющимся условиям окружающей среды и эффективно принимать решения в режиме реального времени.

В этой области значительное внимание уделяется методам искусственного интеллекта, в частности эволюционным алгоритмам. Генетические алгоритмы, основанные на принципах естественного отбора и наследования, показали себя с хорошей стороны, как мощный инструмент для решения задач оптимизации и поиска в условиях неопределенности и ограниченности априорной информации. Применение этих технологий в контексте автономного транспорта позволяет разрабатывать гибкие адаптивные стратегии вождения. Эти стратегии способны учитывать сложную структуру дорожной среды и множество возможных сценариев взаимодействия с другими участниками дорожного движения.

Целью данной работы является разработка и исследование системы управления автономным транспортным средством, использующей генетический алгоритм для принятия навигационных решений.

Для достижения поставленной цели были сформулированы следующие задачи:

1. провести обзор современных методов управления автономными транспортными средствами,
2. изучить принципы работы генетических алгоритмов,
3. разработать архитектуру системы управления автономным автомобилем на основе генетического алгоритма,
4. реализовать программную модель генетического алгоритма,
5. создать имитационную среду для тестирования работы системы,
6. провести серию вычислительных экспериментов для оценки эффективности предложенного алгоритма,
7. проанализировать результаты.

Объектом исследования является система управления автономным транспортным средством, а предметом — методы, обеспечивающие умный выбор маршрута и поведения системы на основе методов генетических алгоритмов.

Научная новизна работы заключается в применении генетических алгоритмов к задаче управления движением автономного транспортного средства. Практическая значимость исследования заключается в возможности использования полученных результатов при разработке интеллектуальных навигационных систем и роботизированных транспортных средств.

# ****Анализ существующих подходов к управлению автономными транспортными средствами****

# Общая характеристика автономных автомобилей

Автономные транспортные средства, или беспилотные автомобили, представляют собой интеллектуальные мобильные системы, способные выполнять задачи в дорожной среде без участия водителя [1]. В основе таких систем лежит принцип интеграции сенсорных, вычислительных и исполнительных компонентов, объединенных в единую архитектуру управления.

Современные автономные автомобили оснащаются различными датчиками, включая лидары, радары, ультразвуковые датчики, GPS, инерциальные модули и камеры машинного зрения. Эти устройства способствуют комплексному восприятию окружающей среды, позволяя формировать ее цифровую модель в режиме реального времени. На основе этой информации система управления производит всевозможные расчеты, такие, как оценка ситуации, планирование маршрута и выбор управляющих воздействий.

Уровень автономности автомобиля классифицируется по шкале SAE (Society of Automotive Engineers), которая включает в себя шесть уровней от 0 (когда водитель полностью контролирует ситуацию) до 5 (когда автомобиль полностью автономен в любых условиях) (рисунок 1) [2].



Рисунок 1 – Уровни автономности

В современной практике наиболее распространены уровни 2 и 3, когда система способна контролировать автомобиль в определенных ситуациях, но при этом требует постоянного контроля со стороны человека.

Основными требованиями к автономным автомобилям являются безопасность, надежность, устойчивость, способность адаптироваться к изменяющимся условиям и эффективно взаимодействовать с другими участниками дорожного движения. Для выполнения этих требований необходимы современные алгоритмы управления, способные обрабатывать многомерные данные, учитывать неопределенности и прогнозировать развитие дорожной ситуации.

# Обзор современных систем управления

Современные системы управления автономными транспортными средствами представляют собой многоуровневые архитектуры, объединяющие модули восприятия, принятия решений и управления движением. Каждая из этих подсистем выполняет определенные функции и взаимодействует в режиме реального времени для обеспечения безопасного и эффективного вождения в динамичной среде.

Наиболее известные промышленные и исследовательские системы, такие как Waymo, Tesla Autopilot, NVIDIA Drive, Baidu Apollo, Uber ATG и Yandex Autonomous Driving, используют гибридные подходы, сочетающие компьютерное зрение, обработку сигналов лидара и радара, а также алгоритмы машинного обучения и оптимизации (рисунок 2) [3].

Изображение выглядит как текст, транспортное средство, Наземный транспорт, машина

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2 – Пример устройства беспилотного автомобиля

Их архитектуры включают следующие ключевые компоненты:

* модуль восприятия, осуществляющий обработку сенсорных данных для идентификации объектов, дорожной разметки, пешеходов, знаков и других элементов окружающей среды;
* модуль локализации, обеспечивающий точное позиционирование автомобиля с помощью GPS, инерциальных систем и высокоточных картографических данных;
* модуль планирования, который генерирует краткосрочные и долгосрочные траектории с учетом текущей ситуации и прогнозирует поведение объекта;
* модуль управления движением, который непосредственно преобразует решения модуля планирования в команды для исполнительных механизмов (рулевое управление, ускорение, торможение).

Существующие реализации различаются по уровню автономности, степени использования машинного обучения и принципам организации взаимодействия между модулями [4]. Например, Tesla Autopilot опирается на визуальные данные с камер и нейросетевые модели, а Waymo использует более традиционный подход с активными датчиками и предварительно нанесенными картами.

Несмотря на значительный прогресс, многие из современных систем ограничены определенными условиями эксплуатации, требуют постоянного контроля со стороны водителя или демонстрируют нестабильную работу в сложных или нестандартных дорожных ситуациях. Это стимулирует разработку новых методов управления на основе адаптивных, самообучающихся и эволюционных алгоритмов, в частности генетических алгоритмов, способных решать навигационные задачи в условиях неопределенности и многокритериальности.

# Методы искусственного интеллекта в управлении

Методы искусственного интеллекта (далее — ИИ) играют ключевую роль в развитии автономных транспортных систем, обеспечивая возможность адаптивного принятия решений в условиях высокой неопределённости, динамичности среды и неполноты информации. Современные ИИ-подходы применяются на всех уровнях архитектуры автономного автомобиля — от восприятия окружающей среды до планирования маршрута и принятия тактических решений [5].

Наиболее активно в системах управления используются следующие направления ИИ:

* машинное обучение и, в частности, глубокое обучение применяются для задач распознавания объектов, классификации дорожных ситуаций, предсказания поведения участников движения и оценки рисков. Нейронные сети обучаются на больших объемах данных и способны выявлять сложные зависимости, что делает их незаменимыми в условиях, где невозможно явно задать все правила поведения;
* имитационное обучение позволяет обучать модели, повторяющие поведение опытного водителя, на основе записей с реальных поездок. Этот подход даёт возможность сформировать стратегию управления без явной формализации всех дорожных сценариев;
* обучение с подкреплением используется для разработки стратегий движения, где особь (автомобиль) обучается, взаимодействуя со средой и получая вознаграждение за эффективные действия. Этот подход особенно перспективен в задачах адаптивного управления, однако требует большого количества итераций и хорошо проработанной симуляционной среды;
* эволюционные вычисления, включая генетические алгоритмы, представляют собой методы глобального поиска, вдохновлённые биологическими механизмами наследования и отбора. Они демонстрируют высокую устойчивость к шуму, эффективно работают в многомерных пространствах и способны формировать управленческие решения, не опираясь на детерминированные модели среды;
* нечёткая логика и экспертные системы применяются для построения интерпретируемых стратегий управления, особенно в ситуациях, где требуется принимать решения на основе приближённых или неполных данных.

Применение ИИ-методов позволяет существенно повысить адаптивность и устойчивость автономных транспортных систем [6]. Однако остаются вызовы, связанные с объяснимостью принимаемых решений, устойчивостью к специфичным событиям и необходимостью верификации безопасности обученных моделей. В этой связи эволюционные алгоритмы, в том числе генетические, представляют собой привлекательную альтернативу или дополнение к другим ИИ-подходам, обеспечивая гибкость и возможность тонкой настройки стратегии управления в сложных условиях.

# ****Теоретические основы генетических алгоритмов****



# Основные принципы и компоненты генетического алгоритма

Генетические алгоритмы представляют собой стохастические методы глобального поиска и оптимизации, основанные на принципах естественного отбора и эволюции, сформулированных в рамках эволюционной биологии. Эти алгоритмы имитируют процесс эволюции, последовательно отбирая и модифицируя особи (решения) с целью повышения их приспособленности [7].

Основу работы генетического алгоритма составляют следующие ключевые компоненты:

* кодирование решений (репрезентация) — способ представления потенциальных решений задачи в виде генотипов (обычно в форме бинарных, вещественных или символьных строк), где каждая строка — это индивидуум популяции;
* функция приспособленности — критерий, по которому оценивается качество каждого индивидуума. Функция отражает, насколько хорошо данное решение соответствует целям задачи оптимизации;
* инициализация популяции — начальное формирование набора особей, часто случайным образом, что обеспечивает разнообразие решений на старте алгоритма;
* операторы селекции — механизмы отбора лучших особей для последующего размножения;
* оператор кроссовера (скрещивания) — процесс комбинирования генетического материала двух родительских особей для генерации новых потомков. Кроссовер способствует передаче удачных черт следующим поколениям;
* оператор мутации — случайная мутация отдельных генов в генотипе потомков, что позволяет поддерживать генетическое разнообразие и предотвращает преждевременную сходимость;
* критерий остановки — условие завершения работы алгоритма, которое может определяться достижением заданного качества решения, истечением времени, количества поколений или стабилизацией популяции.

Процесс работы генетического алгоритма заключается в итеративной эволюции популяции: на каждом шаге отбираются особи, которые подвергаются скрещиванию и мутации, после чего происходит оценка новой популяции и выбор наиболее приспособленных решений (рисунок 3) [8].

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3 – Генетический подход

Таким образом, генетический алгоритм реализует направленный поиск с элементами случайности, что позволяет ему эффективно решать широкий класс задач — от численной оптимизации до управления сложными динамическими системами.

Благодаря своей универсальности, гибкости и способности находить приближённые решения в условиях неполной информации, генетические алгоритмы находят широкое применение в инженерных задачах, включая управление автономными транспортными средствами.

# Кодирование решений

Процесс кодирования решений представляет собой ключевой этап в построении генетического алгоритма. Он определяет способ представления потенциальных решений данной проблемы (фенотипов) в формате, способствующем эволюционным операциям, таким как скрещивание и мутация. Эта форма представления, называемая генотипом, должна быть удобной для последующей алгоритмической обработки [9].

К наиболее распространенным типам кодирования относятся:

* бинарное кодирование — каждое решение представляется в виде строки из нулей и единиц. Этот подход является классическим для генетических алгоритмов и хорошо подходит для задач дискретной оптимизации, таких как маршруты, траектории, логические стратегии и выбор действий. Например, в задаче автономного управления транспортным средством бинарная строка может обозначать последовательность управляющих воздействий или структуру маршрута по дорожной сетке;
* реальное (числовое) кодирование используется в тех случаях, когда переменные, принимающие решения, принимают непрерывные значения. Вместо битов пользователь представляет собой набор действительных чисел. Этот подход широко используется в задачах параметрической оптимизации, включая настройку регуляторов и выбор параметров непрерывного управления;
* символьное (или целочисленное) кодирование хорошо подходит для комбинаторных задач, где решением является упорядоченный набор объектов (например, последовательность поворотов, действий или координаты точек маршрута);
* деревья и графовые структуры — применяются в задачах программной эволюции и автоматического построения стратегий. Эти структуры особенно актуальны в контексте генетического программирования и разработки адаптивных логик поведения для роботов.

Выбор метода кодирования оказывает значительное влияние на эффективность работы алгоритма. Он определяет, насколько легко могут быть реализованы операторы кроссовера и мутации, и насколько хорошо сохраняются семантически значимые части решений в процессе эволюции. Важно, чтобы кодирование обеспечивало достаточную выразительность для представления допустимых решений и не вводило чрезмерных ограничений на пространство поиска.

В области задач автономного управления транспортными средствами двоичное кодирование часто используется для представления последовательности действий (например, движение вперед, поворот, остановка), а также для моделирования дискретной дорожной обстановки. Такой подход обеспечивает простоту реализации и совместимость с базовыми эволюционными операциями, что делает его подходящим выбором на этапе прототипирования и моделирования.

# Операторы селекции, скрещивания и мутации

Эффективность генетического алгоритма в основном зависит от выбора и реализации эволюционных операторов, обеспечивающих процесс отбора, комбинирования и изменения особей в популяции. Ключевыми среди них являются операторы отбора, скрещивания (кроссинговера) и мутации, которые в совокупности обеспечивают целенаправленное и разнообразное исследование пространства решений.

Селекцию можно определить как процесс отбора особей из текущей популяции с целью участия в формировании нового поколения. Целью селекции является обеспечение того, чтобы предпочтение отдавалось особям, которые лучше приспосабливаются, что способствует «естественному отбору» наиболее эффективных решений. Основные методы селекции включают:

* рулеточную (пропорциональную) селекцию, где вероятность выбора особи пропорциональна её приспособленности;
* турнирную селекцию, при которой случайным образом выбирается подгруппа особей, и в потомство попадает наиболее приспособленная из них;
* ранговую селекцию, использующую ранжирование особей по уровню приспособленности с дальнейшим распределением вероятностей выбора.

Скрещивание (crossover) является основным оператором получения потомства, формирующим новые решения путем объединения генетического материала двух родительских особей.

Возможны следующие варианты реализации:

* одноточечный и двуточечный кроссовер, при которых обмен сегментами генотипов осуществляется по заранее определённым точкам;
* однородный кроссовер, предполагающий вероятностный выбор генов от каждого родителя;
* арифметический кроссовер, применяемый при вещественном кодировании, формирует потомков как линейную комбинацию параметров родителей.

Скрещивание способствует передаче «удачных» комбинаций признаков и освоению новых областей пространства решений, сохраняя при этом структуру предыдущих поколений.

Мутацию можно определить как операцию, которая вносит случайные изменения в отдельные гены индивидуума с целью поддержания генетического разнообразия популяции и предотвращения преждевременной конвергенции алгоритма. Типичные способы мутации включают:

* побитовую инверсию для бинарных строк (смена 0 на 1 и наоборот),
* случайное добавление шума или смещение параметров при вещественном представлении,
* перестановки или замены элементов в символьных строках и списках.

Интенсивность применения операторов (вероятность мутации и скрещивания) оказывает существенное влияние на равновесие между исследованием и использованием пространства решений. Чрезмерная мутация может привести к размыванию полезных функций, в то время как ее недостаток приводит к потере разнообразия и застою в поиске.

Следовательно, разумный отбор и координация действий операторов, включая селекцию, скрещивание и мутацию, играют важную роль в обеспечении эффективной эволюции популяции решений, тем самым играя ключевую роль в успешной реализации генетического алгоритма.

# Критерии завершения и адаптивные механизмы

Выбор критерия завершения является ключевым компонентом в разработке эффективного генетического алгоритма, поскольку он определяет точку, в которой эволюционный процесс завершается [10]. Эффективность вычислений и качество решения зависят от точности этого выбора.

Классические критерии завершения включают:

* фиксированное количество поколений — алгоритм прекращает работу по достижении заданного числа итераций, что удобно для задач с ограниченным временем расчёта;
* достижение порогового значения функции приспособленности — алгоритм завершает выполнение, если находится решение, удовлетворяющее заданному уровню качества;
* отсутствие прогресса — если на протяжении определённого числа поколений не наблюдается улучшения функции приспособленности, алгоритм считается исчерпавшим потенциал;
* истечение времени — завершение по ограничению по времени выполнения, часто применяемое в реальном времени или на встроенных системах.

В дополнение к строгим критериям завершения, современные реализации генетических алгоритмов используют адаптивные механизмы для повышения гибкости и эффективности алгоритма в динамично меняющихся условиях.

К числу таких механизмов относятся:

* адаптивная настройка параметров — изменение вероятностей кроссовера и мутации в зависимости от стадии эволюции или текущего разнообразия популяции;
* динамическая селекция — корректировка стратегии отбора в ответ на обнаружение стагнации или преждевременной сходимости;
* гибридизация с локальными методами оптимизации — интеграция локальных поисковых процедур, активируемых при стабилизации прогресса;
* механизмы элитизма и репопуляции — сохранение наилучших решений и частичная или полная замена популяции при необходимости восстановления разнообразия.

Применение адаптивных стратегий приобретает особое значение в сложных задачах управления, таких как эксплуатация автономных транспортных средств в нестабильной среде, где эффективность алгоритма при реагировании на непредсказуемые изменения входных условий имеет первостепенное значение.

Следовательно, разумный выбор критериев завершения в сочетании с адаптивными механизмами управления параметрами значительно повышает эффективность и стабильность генетического алгоритма, тем самым облегчая его применение в реальных инженерных задачах.

# Разработка модели системы управления автономным автомобилем



# Постановка задачи и формализация модели

Разработка эффективной системы управления автономным транспортным средством требует чёткой формализации задачи, учитывающей особенности динамики объекта, структуру окружающей среды и целевые критерии навигации. Управление автономным автомобилем в условиях частичной неопределённости и ограниченной априорной информации представляет собой задачу поиска оптимальной последовательности действий, обеспечивающей безопасное и эффективное передвижение к заданной цели.

Основной целью моделирования является построение алгоритма, способного генерировать допустимые и адаптивные управляющие воздействия, минимизирующие заданную целевую функцию, например длину маршрута, время прохождения трассы или вероятность столкновений. При этом система должна учитывать как глобальные цели навигации, так и локальные ограничения, накладываемые на траекторию движения.

В рамках настоящей работы задача формализуется следующим образом:

* автомобиль рассматривается как особь, движущаяся в дискретной или дискретизированной среде (сетке или графе), содержащей допустимые участки движения;
* состояние особи описывается координатами положения и направлением движения;
* управление осуществляется путём выбора последовательности действий из конечного множества допустимых манёвров (например, движение вперёд, поворот влево/вправо);
* цель особи — достичь конечной точки маршрута из заданной начальной позиции при минимизации функции стоимости, включающей метрики риска столкновений и соблюдения допустимых траекторий.

Поиск управляющей последовательности представляется как задача оптимизации в пространстве решений, где каждая особь соответствует потенциальному маршруту или стратегии поведения. Генетический алгоритм используется в качестве метода глобального поиска, способного эффективно исследовать большое пространство возможных решений и адаптироваться к условиям среды.



# Технологии

В процессе разработки модели системы управления автономным транспортным средством в качестве основной среды программирования был выбран язык Python, отличающийся простотой синтаксиса, широкими возможностями для прототипирования и высокой читаемостью кода. Ключевым принципом реализации являлось минимизировать зависимость от сторонних библиотек и реализовать алгоритм с нуля для глубокого понимания всех этапов вычислений, а также гибкости и адаптируемости модели под будущие модификации и расширения.

Для визуализации среды моделирования, включая трассу, положение и движение транспортного средства, применялась только одна внешняя графическая библиотека — Pyglet. Она обеспечила реализацию 2D-интерфейса, позволяющего отслеживать поведения особей в реальном времени.

Все ключевые компоненты генетического алгоритма — включая инициализацию популяции, операции селекции, скрещивания, мутации и оценки приспособленности — были реализованы вручную, без использования специализированных фреймворков для эволюционных вычислений. Это обеспечило максимальный контроль над логикой алгоритма и возможность дальнейшей адаптации под конкретные особенности дорожной среды или поведения автомобиля.

В проекте использовались встроенные модули Python:

* random — генерация случайных чисел, необходимых для стохастических операций эволюционного алгоритма;
* math — выполнение геометрических и тригонометрических вычислений;
* time — измерение производительности и задержек;
* itertools — работа с комбинаторикой и последовательностями;
* os и json — взаимодействие с файловой системой, сохранение параметров и результатов в конфигурационных файлах.

Такой подход позволил построить компактную и прозрачную систему, не перегруженную избыточным функционалом сторонних библиотек, что важно как с образовательной, так и с инженерной точки зрения.

# Архитектура программного решения

Программная реализация модели системы управления автономным транспортным средством построена на принципах модульности, расширяемости и тесной интеграции между компонентами восприятия среды, принятия решений и визуализации. Архитектура решения была спроектирована таким образом, чтобы обеспечить максимальную прозрачность логики работы каждого элемента системы и облегчить эксперименты с алгоритмами управления (рисунок 4).

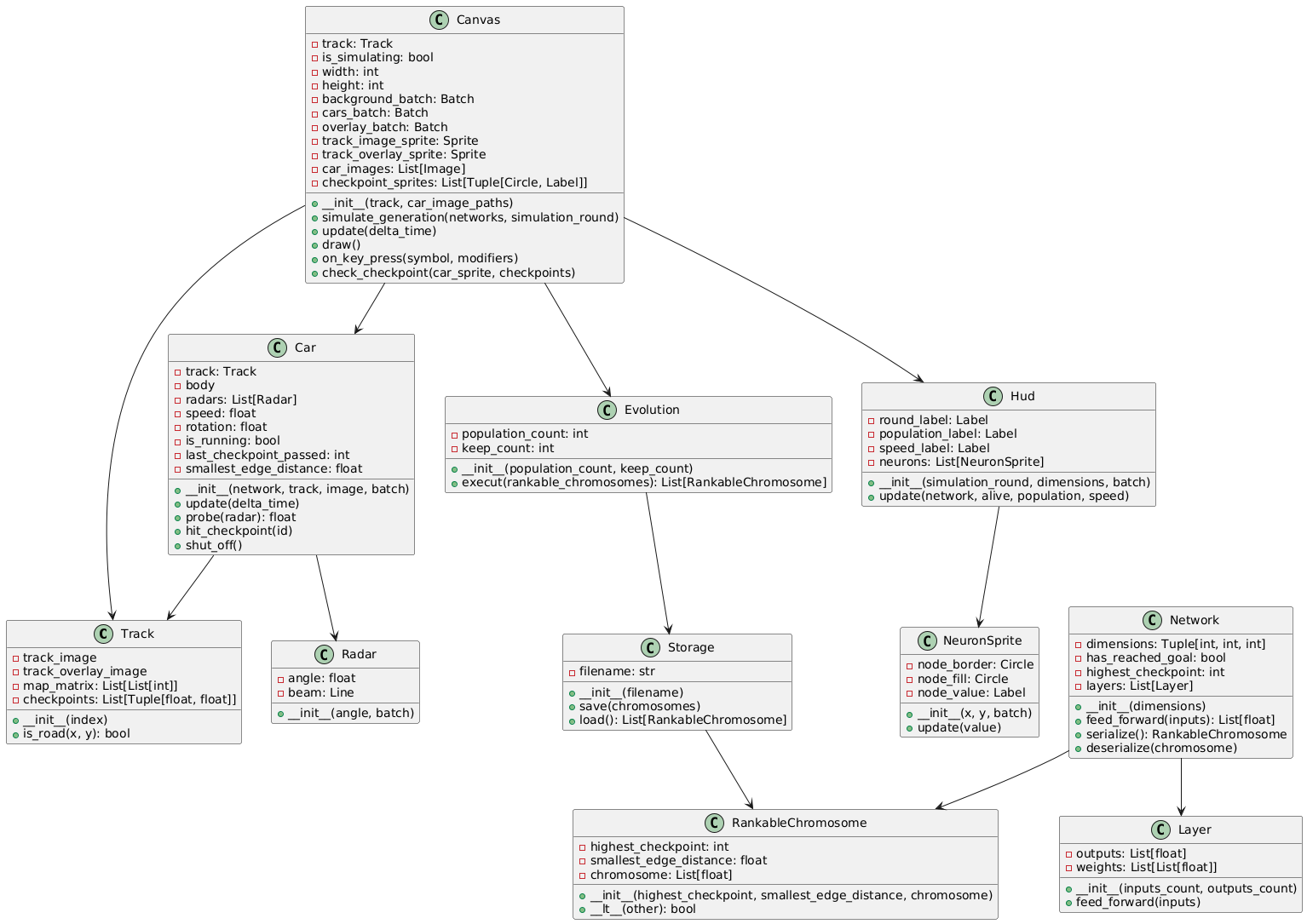


Рисунок 4 – Диаграмма классов

Центральным элементом является класс Canvas, который координирует взаимодействие всех компонентов симуляции. Он отвечает за загрузку 5 реализованных трасc (Track), размещение автомобилей (Car) на старте, запуск и обновление поколений, отрисовку визуальных элементов, а также обработку пользовательского ввода. Класс Track хранит графическое представление трассы, её логическую матрицу проходимости и контрольные точки (checkpoints), определяющие успешность навигации.

Автомобиль реализован через класс Car, включающий параметры движения, поворота, радары (Radar) для сенсорного восприятия препятствий, и связь с искусственной нейронной сетью (Network), управляющей его поведением. Сеть, в свою очередь, построена из иерархии объектов Layer, каждый из которых реализует базовую операцию прямого распространения сигнала (feed forward). Архитектура сети задаётся параметром dimensions, определяющим количество входных, скрытых и выходных нейронов.

Особое внимание в разработке было уделено реалистичности поведения транспортных средств. В систему внедрён эффект скольжения — потеря управления автомобилем при высоких скоростях и резких поворотах. Описывается он следующей формулой: , где x — текущая скорость автомобиля, t — пороговая скорость проскальзывания и m — модификатор снижения воздействия на рулевое управление. Это поведение учитывает ограниченность сцепления с дорогой и позволяет лучше смоделировать физику движения.

Эволюционный механизм обучения представлен классом Evolution, который управляет популяцией особей, отбирает наиболее успешные решения и формирует новое поколение с помощью генетических операторов. Каждая особь кодируется объектом RankableChromosome, содержащим параметры сети и метрики её эффективности (достигнутый чекпоинт и расстояние до края трассы). Отслеживание прогресса обучения осуществляется через консольный вывод метрик после каждого поколения.

Для анализа и повторного тестирования имеется возможность сохранения лучших нейросетей в формате JSON через модуль Storage. Эти сохранённые «мозги» могут быть впоследствии загружены в симулятор для проведения тест-драйвов без повторного обучения. Это позволяет оценивать устойчивость решений на новых трассах или при изменённых условиях.

Визуальное сопровождение симуляции реализовано с использованием классов Hud и NeuronSprite. Первый отображает текущую статистику симуляции — номер поколения, размер популяции и скорость автомобиля. Второй визуализирует активации нейронов в сети в реальном времени, что способствует отладке и интерпретации работы алгоритма.

Архитектура ориентирована на чёткое разделение ответственности между компонентами, что способствует удобству тестирования, адаптации под различные сценарии движения и последующей интеграции более сложных логик поведения. Благодаря независимой реализации ключевых компонентов алгоритма система легко модифицируется для оценки эффективности различных параметров, стратегий селекции и конфигураций нейросети.

# Представление окружающей среды и карты

В рамках разработанной модели окружающая среда представляет собой двумерную карту, отображающая трассу с различимыми дорожными и недоступными для движения участками (рисунок 5).

Изображение выглядит как Мультфильм, графическая вставка, снимок экрана, графический дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 5 – Пример одной из трасс

Это представление реализовано с использованием класса Track, играющего ключевую роль в формировании логической структуры карты.

Каждая трасса загружается из изображения, представляющего собой готовую графическую карту, и обрабатывается для получения логической матрицы проходимости (map\_matrix), где каждому пикселю или блоку соответствует бинарное значение: «дорога» или «препятствие» (рисунок 6).

Изображение выглядит как Графика, Шрифт, графический дизайн, зеленый

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 6 – Матрица дорожного покрытия

Такое представление среды позволяет использовать трассу не только в визуальных целях, но и как основу для вычислений при определении допустимых направлений движения, работы радаров и оценки столкновений.

Для контроля прогресса и оценки качества решений в каждой трассе задаются контрольные точки (checkpoints), координаты которых фиксируются в соответствующем списке. Эти точки выполняют функцию навигационных ориентиров, прохождение которых учитывается при расчёте приспособленности (fitness) каждой особи в процессе эволюционного обучения.

Метод is\_road(x, y) реализует логику проверки доступности заданной координаты, позволяя оперативно определять, находится ли точка в пределах допустимого для движения пространства. Это критически важно как для анализа сигналов от сенсоров (радаров), так и для принятия решений о корректировке движения.

Таким образом, карта среды служит основой не только для визуализации симуляции, но и для формализации задач навигации и взаимодействия с окружающей средой. Реализация карты через изображение и логическую модель обеспечивает универсальность: для добавления новой трассы достаточно подготовить изображение с заданной структурой, что делает систему легко расширяемой и пригодной для масштабных экспериментальных исследований.

# Реализация генетического алгоритма и алгоритма принятия решений

Генетический алгоритм в рамках данной работы реализует эволюционное обучение нейронных сетей, управляющих автономным транспортным средством в симулированной среде. Основная задача алгоритма — адаптивная настройка весов нейросетевой модели, обеспечивающей корректное поведение автомобиля при навигации по трассе.

Каждая особь в популяции представляет собой набор весов нейронной сети, закодированный в виде одномерного массива. Архитектура сети фиксирована и определяется заранее (в данной реализации — многослойный персептрон) (рисунок 7) [5].

Изображение выглядит как снимок экрана, круг, Графика, текст

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 7 – Структура нейросети

Основу обучения составляет многократное моделирование движения автомобиля по трассе, где эффективность оценивается с помощью функции приспособленности.

Функция приспособленности учитывает два ключевых параметра:

* количество успешно пройденных контрольных точек,
* расстояние до границ дороги.

Процесс эволюции включает:

* селекцию: отбор наиболее приспособленных особей для перехода в следующее поколение;
* скрещивание: комбинирование участков двух родительских хромосом с помощью одноточечного кроссовера (разреза), что позволяет сформировать новые хромосомы-потомки;
* мутацию: случайная замена отдельных генов на новое случайное значение из диапазона [-1, 1], обеспечивая разнообразие и предотвращение преждевременной сходимости.

Ключевые параметры алгоритма (выбранные на основании эмпирических наблюдений):

* размер популяции: 40,
* количество сохраняемых лучших решений (элита): 4,
* вероятность мутации: 20%,
* диапазон мутаций: ±1.

Алгоритм принятия решений осуществляется в реальном времени путём пропускания входных данных (результатов работы сенсоров — радаров) через нейронную сеть. Выходные значения нейросети соответствуют командам управления, в частности углу поворота и уровню ускорения. Таким образом, каждая особь в популяции представляет потенциальную стратегию управления, а процесс симуляции позволяет выявить наиболее эффективные из них.

Применение генетического алгоритма в данном контексте обеспечивает автономное, адаптивное обучение, что особенно важно при проектировании систем управления для сложных и динамичных сред, таких как дорожная обстановка.

# Экспериментальное исследование и оценка эффективности



# Методика проведения экспериментов

Для исследования влияния параметров генетического алгоритма на процесс обучения модели управления автономным автомобилем была организована серия экспериментов.

В рамках экспериментов варьировались следующие параметры:

* размер популяции: от 10 до 50 особей,
* вероятность мутации: от 1% до 30%,
* число поколений: от 10 до 50.

Для каждой комбинации параметров запускался процесс обучения, в ходе которого происходила адаптация популяции особей к условиям трассы. По завершении каждой симуляции фиксировались показатели пригодности особей. В качестве критериев оценки использовались:

* число пройденных контрольных точек,
* среднее расстояние до границ дороги,
* время нахождения в пределах допустимой траектории,
* стабильность поведения при повторных запусках.

Каждый эксперимент проводился в изолированной среде с заранее заданными начальными условиями. Результаты симуляций документировались для последующего анализа, включающего сравнение эффективности различных конфигураций параметров.

# Анализ полученных результатов

После проведения экспериментов с различными конфигурациями параметров генетического алгоритма был выполнен детальный анализ с целью выявления зависимостей между характеристиками алгоритма и качеством управления автономным автомобилем в симуляционной среде. Основное внимание уделялось таким параметрам, как размер популяции, вероятность мутации и число поколений. Целью экспериментов было определение конфигураций, обеспечивающих оптимальный баланс между скоростью обучения и стабильностью поведения особей в условиях динамически изменяющейся среды.

Для оценки качества работы системы использовались следующие ключевые метрики: средняя пригодность популяции, количество успешно пройденных контрольных точек, среднее расстояние до границ трассы и стабильность поведения в повторяющихся условиях.

Обработка результатов показала, что вероятность мутации оказывает ключевое влияние на динамику обучения. Так, высокая мутационная активность (свыше 20%) на ранних этапах экспериментов способствовала активному исследованию пространства решений. Однако сохранение этого уровня на более поздних этапах приводило к снижению устойчивости поведения особей: появлялись резкие колебания в пригодности, нестабильные траектории движения и деградация ранее выработанных стратегий. Наиболее стабильное и предсказуемое поведение наблюдалось в конфигурациях с адаптивной вероятностью мутации: высокая на первых поколениях (до 20%) и постепенное снижение к финалу обучения (до 5%).

Анализ влияния размера популяции показал, что слишком малая популяция (менее 15 особей) ограничивает генетическое разнообразие, снижая способность системы адаптироваться к разнообразным условиям. С другой стороны, увеличение популяции более чем до 50 особей лишь незначительно улучшает качество поведения, но существенно увеличивает время симуляции. Оптимальным размером популяции оказался диапазон от 30 до 40 особей, при котором достигался наилучший компромисс между разнообразием стратегий и вычислительными затратами.

На рисунке 8 приведена динамика изменения расстояния до края трассы в зависимости от номера раунда, как индикатора устойчивости поведения модели.

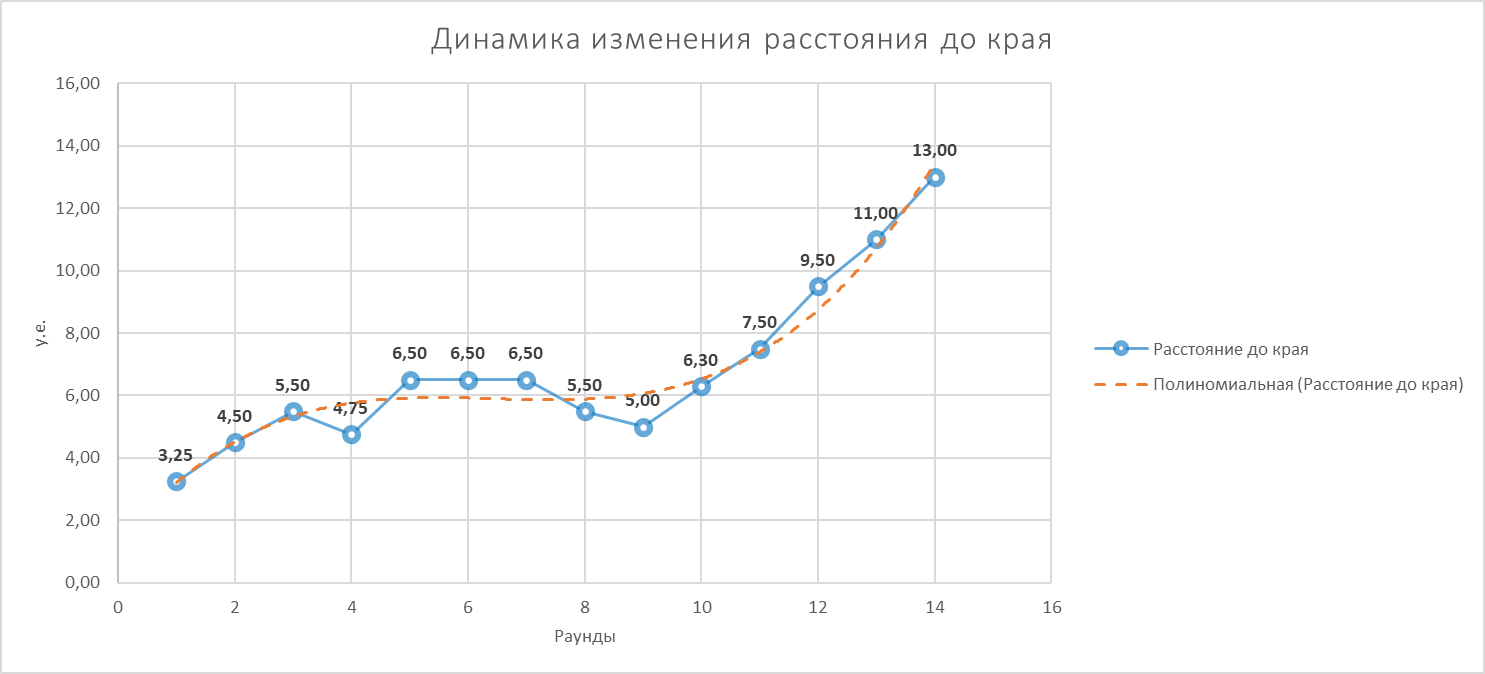


Рисунок 8 – Динамика изменения расстояния до края

Этот показатель демонстрирует способность автомобиля избегать границ дороги, что напрямую связано с безопасностью движения.

В начальной фазе (1–4 раунды) наблюдается постепенное улучшение поведения, затем в раундах 5–7 закрепляется устойчивый уровень. Временное снижение показателя в 8–10 раундах может быть связано с действием мутации, а последующий резкий рост (раунды 11–14) демонстрирует успешную адаптацию.

Показатель пригодности популяции позволил оценить скорость адаптации. Во всех конфигурациях с адаптивной мутацией и сбалансированной популяцией резкий рост пригодности наблюдался уже к 5–7 поколению, что свидетельствует о высокой скорости выработки устойчивого поведения. На поздних этапах происходила стабилизация пригодности, указывающая на сходимость алгоритма.

Полученные данные позволяют сделать вывод, что параметры генетического алгоритма критически важны для формирования качественного поведения автономного автомобиля. Их грамотная настройка позволяет достичь высокой эффективности и устойчивости управления в симулируемой среде.

# ЗАКЛЮЧЕНИЕ

Исходя из поставленной цели, выполненных задач и полученных результатов, можно сделать следующие выводы:

* был проведён аналитический обзор современных подходов к управлению автономными транспортными средствами, что позволило выделить преимущества применения методов искусственного интеллекта и эволюционного подхода;
* подробно изучены принципы работы генетических алгоритмов, включая механизмы селекции, кроссинговера, мутации и оценки приспособленности, что стало основой для построения обучаемой системы управления;
* разработана архитектура системы, объединяющая ключевые компоненты: восприятие среды (радары), принятие решений (нейросеть), управление движением (модель автомобиля) и визуализацию процессов, что обеспечило модульность, прозрачность и расширяемость;
* реализована программная модель генетического алгоритма;
* создана имитационная среда, включающая различные типы трасс, реалистичную физику движения с учётом эффекта скольжения, а также визуализацию активности нейронной сети и отображение ключевых метрик;
* проведены вычислительные эксперименты, подтверждающие способность системы к обучению: за несколько поколений наблюдается рост числа успешно завершивших трассу автомобилей, увеличение средней достигнутой контрольной точки и снижение отклонений от траектории;
* реализован удобный консольный вывод метрик эффективности на каждом поколении, а также сохранение параметров обученных нейросетей в JSON-файлы, что позволяет проводить повторные тест-драйвы и анализировать поведение без необходимости переобучения.

Таким образом, разработанная система продемонстрировала эффективность применения генетических алгоритмов для адаптивного управления автономными транспортными средствами и может быть использована в дальнейшем для исследований в области самообучающихся систем и интеллектуальной мобильности.

За время выполнения выпускной квалификационной работы были приобретены следующие компетенции (таблица 1):

Таблица 1 – Компетенции

|  |  |  |
| --- | --- | --- |
| Компетенция | Расшифровка компетенции | Описание приобретенных знаний, умений и навыков |
| УК-1 | Способен осуществлять поиск, критический анализ и синтез информации, применять системный подход для решения поставленных задач | Проведен анализ современных подходов к управлению автономным транспортом, систематизированы знания о генетических алгоритмах, разработана и протестирована система управления. |
| УК-2 | Способен определять круг задач в рамках поставленной цели и выбирать оптимальные способы их решения, исходя из действующих правовых норм, имеющихся ресурсов и ограничений | Сформулированы задачи на основе поставленной цели, выбраны методы проектирования и тестирования, применимы к условиям ограниченных вычислительных ресурсов. |
| УК-3 | Способен осуществлять социальное взаимодействие и реализовывать свою роль в команде | Получен опыт взаимодействия с научным руководителем и экспертами, а также навыки обсуждения проектных решений. |
| УК-4 | Способен осуществлять деловую коммуникацию в устной и письменной формах на государственном языке Российской Федерации и иностранном(ых) языке(ах) | Подготовлена защита и презентация, освоены навыки устной и письменной научной коммуникации. |
| УК-5 | Способен воспринимать межкультурное разнообразие общества в социально-историческом, этическом и философском контекстах | При изучении зарубежного опыта использованы материалы на английском языке, сделан анализ подходов, принятых в разных странах. |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| Компетенция | Расшифровка компетенции | Описание приобретенных знаний, умений и навыков |
| УК-6 | Способен управлять своим временем, выстраивать и реализовывать траекторию саморазвития на основе принципов образования в течение всей жизни | Разработан и реализован план поэтапной работы над проектом, освоены новые технологии и методы самостоятельно. |
| УК-7 | Способен поддерживать должный уровень физической подготовленности для обеспечения полноценной социальной и профессиональной деятельности | В процессе выполнения выпускной квалификационной работы соблюдался баланс между умственной и физической активностью, что способствовало поддержанию работоспособности и концентрации, необходимых для длительной научно-исследовательской и проектной деятельности. |
| УК-8 | Способен создавать и поддерживать в повседневной жизни и в профессиональной деятельности безопасные условия жизнедеятельности для сохранения природной среды, обеспечения устойчивого развития общества, в том числе при угрозе и возникновении чрезвычайных ситуаций и военных конфликтов | В ходе работы над проектом проявлялось осознанное отношение к вопросам безопасности и устойчивого развития. На личном уровне соблюдались правила безопасности и ответственного поведения, что способствовало формированию устойчивых привычек и экологической культуры. |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| Компетенция | Расшифровка компетенции | Описание приобретенных знаний, умений и навыков |
| УК-9 | Способен принимать обоснованные экономические решения в различных областях жизнедеятельности | В процессе разработки системы управления автономным автомобилем учитывались критерии эффективности, в том числе оптимизация вычислительных ресурсов и снижение затрат на реализацию. При выборе методов и архитектурных решений анализировались их ресурсоемкость, сложность внедрения и потенциальная экономическая целесообразность, что позволило сформировать практические навыки принятия рациональных технических и экономических решений. |
| УК-10 | Способен формировать нетерпимое отношение к проявлениям экстремизма, терроризма, коррупционному поведению, противодействовать им в профессиональной деятельности | В процессе выполнения работы и подготовки проекта придерживался принципов академической честности, соблюдения норм этики и права. Сформированы навыки ответственного поведения, соответствующие принципам открытости, гуманизма и законопослушности. |
| ОПК-1 | Способен применять фундаментальные знания, полученные в области математических и (или) естественных наук, и использовать их в профессиональной деятельности | Использованы математические модели и принципы эволюционных вычислений при построении алгоритма. |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| Компетенция | Расшифровка компетенции | Описание приобретенных знаний, умений и навыков |
| ОПК-2 | Способен применять компьютерные/суперкомпьютерные методы, современное программное обеспечение, в том числе отечественного происхождения, для решения задач профессиональной деятельности | Разработан программный модуль с использованием современных средств разработки и тестирования. |
| ОПК-3 | Способен к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных моделей, созданию информационных ресурсов глобальных сетей, образовательного контента, прикладных баз данных, тестов и средств тестирования систем и средств на соответствие стандартам и исходным требованиям | Разработана программная реализация генетического алгоритма, архитектура системы и модель имитации. |
| ОПК-4 | Способен участвовать в разработке технической документации программных продуктов и комплексов с использованием стандартов, норм и правил, а также в управлении проектами создания информационных систем на стадиях жизненного цикла | Сформирована техническая документация по архитектуре системы и этапам разработки. |
| ОПК-5 | Способен инсталлировать и сопровождать программное обеспечение информационных систем и баз данных, в том числе отечественного происхождения, с учетом информационной безопасности | Обеспечена корректная установка, настройка и тестирование программного комплекса. |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| Компетенция | Расшифровка компетенции | Описание приобретенных знаний, умений и навыков |
| ОПК-6 | Способен понимать принципы работы современных информационных технологий и использовать их для решения задач профессиональной деятельности | Использованы знания о системах навигации, управления и искусственного интеллекта для построения модели автономного управления. |
| ПК-1 | Проверка работоспособности и рефакторинг кода программного обеспечения | Проведена отладка и оптимизация кода модели генетического алгоритма. |
| ПК-2 | Интеграция программных модулей и компонент и верификация выпусков программного продукта | Программный модуль интегрирован с имитационной средой, проведено тестирование работы. |
| ПК-3 | Разработка требований и проектирование программного обеспечения | Программный модуль интегрирован с имитационной средой, проведено тестирование работы. |
| ПК-4 | Оценка и выбор варианта архитектуры программного средства | Проведен анализ вариантов и реализована архитектура управления с учётом специфики автономного транспорта. |
| ПК-5 | Разработка тестовых случаев, проведение тестирования и исследование результатов | Реализованы тестовые сценарии и проведён анализ поведения системы в имитационной среде. |
| ПК-6 | Обеспечение и оптимизация функционирования баз данных | Освоены принципы эффективной структуры хранения данных, обеспечивающей быструю выборку и устойчивую работу при обработке информации. |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| Компетенция | Расшифровка компетенции | Описание приобретенных знаний, умений и навыков |
| ПК-7 | Обеспечение информационной безопасности на уровне базы данных | Изучены методы ограничения доступа к информации, обеспечения целостности данных и защиты от несанкционированного вмешательства. Получен практический опыт реализации базовых механизмов безопасности на уровне хранения и обработки данных. |
| ПК-8 | Выполнение работ по созданию (модификации) и сопровождению информационных систем, автоматизирующих задачи организационного управления и бизнес-процессы | Разработан программный прототип информационной системы, автоматизирующей задачи управления движением автономного автомобиля. |
| ПК-9 | Создание и сопровождение требований и технических заданий на разработку и модернизацию систем и подсистем малого и среднего масштаба и сложности | Сформулировано техническое задание на разработку и тестирование системы. |
| ПК-10 | Способен к коммуникации, восприятию информации, умению логически, верно, аргументировано и ясно строить устную и письменную речь для решения профессиональных задач | В процессе выполнения выпускной квалификационной работы развиты навыки структурированной подачи технической информации, оформления пояснительной записки, подготовки презентации и устной защиты результатов. Осуществлялось взаимодействие с научным руководителем, в ходе которого вырабатывались аргументированные решения и формулировались корректные технические обоснования. |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| Компетенция | Расшифровка компетенции | Описание приобретенных знаний, умений и навыков |
| ПК-11 | Способен использовать действующее законодательство и другие правовые документы в своей деятельности, демонстрировать готовность и стремление к совершенствованию и развитию общества на принципах гуманизма, свободы и демократии | Освоены основы нормативно-правового регулирования информационных технологий и ответственности разработчика. Сформировано понимание важности соблюдения правовых норм и этических стандартов, а также стремление к развитию технологий, способствующих улучшению качества жизни и устойчивому развитию общества. |

Проделанная работа показала, что использование генетических алгоритмов для оптимизации управления автономным автомобилем является эффективным методом, который позволяет адаптировать поведение автомобиля к различным условиям. Однако настройка параметров алгоритмов и анализ полученных решений требуют более глубокого подхода и тщательной настройки по сравнению с традиционными методами управления, что обусловлено сложностью эволюционных процессов и необходимостью учета множества факторов для достижения оптимального результата.

# СПИСОК ЛИТЕРАТУРЫ

1) История беспилотных автомобилей [Электронный ресурс]. ⎯ 2020. ⎯ URL: https://habr.com/ru/companies/itelma/articles/505872/ (дата обращения 09.01.2025).

2) Беспилотные автомобили: объяснение 6 уровней автономности [Электронный ресурс]. ⎯ 2018. ⎯ URL: https://vc.ru/transport/48947-bespilotnye-avtomobili-obyasnenie-6-urovnei-avtonomnosti (дата обращения 14.01.2025).

3) 10 компаний, ведущих в области автономных и электрических транспортных средств [Электронный ресурс]. ⎯ 2025. ⎯ URL: https://dzen.ru/a/Z9KCrZ214EmRcZOR (дата обращения 23.01.2025).

4) Как устроены беспилотные автомобили, и кто их делает [Электронный ресурс]. ⎯ 2021. ⎯ URL: https://auto.ru/mag/article/selfdrivingsetup/?utm\_referrer=https%3A%2F%2Fyandex.ru%2F (дата обращения 25.01.2025).

5) Методы искусственного интеллекта и применение их на транспорте [Электронный ресурс]. ⎯ 2023. ⎯ URL: https://cyberleninka.ru/article/n/metody-iskusstvennogo-intellekta-i-primenenie-ih-na-transporte/viewer (дата обращения 10.02.2025).

6) Глаза боятся, а ИИ делает: как эмоции помогают ИИ лучше управлять автомобилем [Электронный ресурс]. ⎯ 2023. ⎯ URL: https://habr.com/ru/companies/bothub/articles/783034/ (дата обращения 15.02.2025).

7) Что такое генетический алгоритм? [Электронный ресурс]. ⎯ 2019. ⎯ URL: https://dzen.ru/a/XEX77NvJAwCtsNml (дата обращения 24.02.2025).

8) Генетические алгоритмы [Электронный ресурс]. ⎯ 2024. ⎯ URL: https://ru.wikipedia.org/wiki/Генетический\_алгоритм (дата обращения 02.03.2025).

9) Генетический алгоритм [Электронный ресурс]. ⎯ 2007. ⎯ URL: https://masters.donntu.ru/2019/fknt/domina/lib/genalg.pdf (дата обращения 14.03.2025).

10) Критерий остановки генетического алгоритма при решении задач оптимизации [Электронный ресурс]. ⎯ 2012. ⎯ URL: https://conf.nsc.ru/files/conferences/ym2012/fulltext/137477/139423/Критерий%20остановки\_Лукьянов.pdf (дата обращения 23.03.2025).

# ПРИЛОЖЕНИЕ

from pyglet.window import Window, key

from pyglet import image

from pyglet.sprite import Sprite

from pyglet.graphics import Batch

from pyglet.shapes import Circle

from pyglet.text import Label

import time

import random

import math

from car import Car

from hud import Hud

class Canvas(Window):

FRAME\_DURATION = 1 / 60

CHECKPOINT\_RADIUS = 15

CHECKPOINT\_HIT\_DISTANCE = 40

def \_\_init\_\_(self, track, car\_image\_paths):

super().\_\_init\_\_(width=960, height=540)

self.track = track

self.is\_simulating = True

# Batches

self.background\_batch = Batch()

self.cars\_batch = Batch()

self.overlay\_batch = Batch()

# Sprites

self.track\_image\_sprite = Sprite(track.track\_image, batch=self.background\_batch)

self.track\_overlay\_sprite = Sprite(track.track\_overlay\_image, batch=self.overlay\_batch)

# Car images

self.car\_images = [image.load(path) for path in car\_image\_paths]

# Checkpoints

self.checkpoint\_sprites = self.\_create\_checkpoint\_sprites(track.checkpoints)

def \_create\_checkpoint\_sprites(self, checkpoints):

sprites = []

for i, (x, y) in enumerate(checkpoints):

circle = Circle(x, y, self.CHECKPOINT\_RADIUS, color=(255, 255, 255, 100), batch=self.background\_batch)

label = Label(str(i), x=x, y=y, anchor\_x="center", anchor\_y="center", color=(255, 255, 255, 255),

batch=self.background\_batch)

sprites.append((circle, label))

return sprites

def simulate\_generation(self, networks, simulation\_round):

self.hud = Hud(simulation\_round, networks[0].dimensions, self.overlay\_batch)

self.car\_sprites = [

Car(network, self.track, random.choice(self.car\_images), self.cars\_batch)

for network in networks

]

self.population\_total = len(self.car\_sprites)

self.population\_alive = self.population\_total

last\_time = time.perf\_counter()

while self.is\_simulating and self.population\_alive > 0:

current\_time = time.perf\_counter()

elapsed\_time = current\_time - last\_time

if elapsed\_time > self.FRAME\_DURATION:

last\_time = current\_time

self.dispatch\_events()

self.update(elapsed\_time)

self.draw()

self.\_finalize\_generation()

def \_finalize\_generation(self):

for car in self.car\_sprites:

car.network.highest\_checkpoint = car.last\_checkpoint\_passed

car.network.smallest\_edge\_distance = car.smallest\_edge\_distance

car.network.has\_reached\_goal = (car.last\_checkpoint\_passed == len(self.checkpoint\_sprites) - 1)

def update(self, delta\_time):

for car in self.car\_sprites:

car.update(delta\_time)

if car.is\_running:

if not self.track.is\_road(car.body.x, car.body.y):

car.shut\_off()

self.\_check\_checkpoint(car)

running\_cars = [c for c in self.car\_sprites if c.is\_running]

self.population\_alive = len(running\_cars)

if running\_cars:

self.hud.update(running\_cars[0].network, self.population\_alive, self.population\_total, running\_cars[0].speed)

def draw(self):

self.clear()

self.background\_batch.draw()

self.cars\_batch.draw()

self.overlay\_batch.draw()

self.flip()

def on\_key\_press(self, symbol, modifiers):

if symbol == key.ESCAPE:

self.is\_simulating = False

print("Симуляция прервана.")

def \_check\_checkpoint(self, car):

for i, (x, y) in enumerate(self.track.checkpoints):

if self.\_distance(car.body.x, car.body.y, x, y) < self.CHECKPOINT\_HIT\_DISTANCE:

car.hit\_checkpoint(i)

@staticmethod

def \_distance(x1, y1, x2, y2):

return math.hypot(x2 - x1, y2 - y1)

from pyglet.sprite import Sprite

from pyglet.shapes import Line

import math

class Radar:

MAX\_LENGTH\_PIXELS = 200

STEP\_SIZE = 2

def \_\_init\_\_(self, angle, batch):

self.angle = angle

self.beam = Line(0, 0, 0, 0, color=(255, 255, 255, 127), batch=batch)

self.beam.width = 2

def update\_beam(self, origin\_x, origin\_y, rotation\_deg, is\_road\_func):

self.beam.x = origin\_x

self.beam.y = origin\_y

angle\_rad = math.radians(rotation\_deg + self.angle)

probe\_length = 0

x2, y2 = origin\_x, origin\_y

while probe\_length < self.MAX\_LENGTH\_PIXELS and is\_road\_func(x2, y2):

probe\_length += self.STEP\_SIZE

x2 = origin\_x + probe\_length \* math.cos(angle\_rad)

y2 = origin\_x + probe\_length \* math.sin(angle\_rad)

self.beam.x2 = x2

self.beam.y2 = y2

return probe\_length

class Car:

MAX\_SPEED = 6.0

SLIPPING\_SPEED = MAX\_SPEED \* 0.75

RADAR\_ANGLES = [-70, -35, 0, 35, 70]

def \_\_init\_\_(self, network, track, image, batch):

self.network = network

self.track = track

# Image setup

image.anchor\_x = 25

image.anchor\_y = 25

self.body = Sprite(image, batch=batch)

self.body.x, self.body.y = track.checkpoints[0]

# Radars

self.radars = [Radar(angle, batch) for angle in self.RADAR\_ANGLES]

# State

self.speed = 0.0

self.rotation = 0.0

self.is\_running = True

self.last\_checkpoint\_passed = 0

self.smallest\_edge\_distance = Radar.MAX\_LENGTH\_PIXELS

def update(self, delta\_time):

render\_speed = delta\_time \* 60

self.speed -= 0.05 # Friction

if self.is\_running:

inputs = [radar.update\_beam(self.body.x, self.body.y, self.rotation, self.track.is\_road) / Radar.MAX\_LENGTH\_PIXELS

for radar in self.radars]

self.\_update\_smallest\_distance(inputs)

acceleration, steer = self.network.feed\_forward(inputs)

if acceleration > 0:

self.speed += 0.1

self.speed = min(self.speed, self.MAX\_SPEED)

steer\_impact = self.\_calculate\_steer\_impact()

self.rotation -= steer \* self.speed \* steer\_impact \* render\_speed \* 3

else:

self.speed -= 0.05 \* self.speed # Slow stop

if self.speed < 0:

self.speed = 0.0

self.shut\_off()

self.\_update\_position(render\_speed)

def \_update\_smallest\_distance(self, radar\_values):

for value in radar\_values:

distance = value \* Radar.MAX\_LENGTH\_PIXELS

if distance < self.smallest\_edge\_distance:

self.smallest\_edge\_distance = distance

def \_calculate\_steer\_impact(self):

if self.speed > self.MAX\_SPEED:

return -self.speed / self.MAX\_SPEED + self.SLIPPING\_SPEED / self.MAX\_SPEED + 1

return 1

def \_update\_position(self, render\_speed):

rad = math.radians(self.rotation)

self.body.rotation = -self.rotation

self.body.x += self.speed \* render\_speed \* math.cos(rad)

self.body.y += self.speed \* render\_speed \* math.sin(rad)

def hit\_checkpoint(self, checkpoint\_id):

if checkpoint\_id - self.last\_checkpoint\_passed == 1:

self.last\_checkpoint\_passed = checkpoint\_id

elif checkpoint\_id < self.last\_checkpoint\_passed:

self.shut\_off() # Went backwards

def shut\_off(self):

self.is\_running = False

self.radars = None

import random

import itertools

class Evolution:

def \_\_init\_\_(self, population\_count, keep\_count):

self.population\_count = population\_count

self.keep\_count = keep\_count

def execute(self, rankable\_chromosomes):

# Selection: оставляем лучших

sorted\_chromosomes = sorted(rankable\_chromosomes)

keep\_chromosomes = [w.chromosome for w in sorted\_chromosomes[:self.keep\_count]]

# Копируем лучших прямо в новое поколение

offspring = list(keep\_chromosomes)

# Генерация потомков (кроссовер)

while len(offspring) < self.population\_count:

parent1, parent2 = random.sample(keep\_chromosomes, 2)

child1, child2 = self.crossover(parent1, parent2)

offspring.extend((child1, child2))

# Обрезаем до нужного размера (если вдруг перебор)

offspring = offspring[:self.population\_count]

# Мутация

for i in range(self.keep\_count, len(offspring)):

self.mutate(offspring[i])

assert len(offspring) == self.population\_count, "Количество потомков не равно заданному размеру популяции"

return offspring

def crossover(self, parent1, parent2):

if len(parent1) != len(parent2):

raise ValueError("Разные длины хромосом при скрещивании")

split\_index = random.randint(1, len(parent1) - 1) # 1..n-1 — чтобы не копировать полностью одного

child1 = parent1[:split\_index] + parent2[split\_index:]

child2 = parent2[:split\_index] + parent1[split\_index:]

return child1, child2

def mutate(self, chromosome, mutation\_chance=0.2):

for i in range(len(chromosome)):

if random.random() < mutation\_chance:

chromosome[i] = random.uniform(-1.0, 1.0)

from pyglet.text import Label

from pyglet.shapes import Circle

class NeuronSprite:

RADIUS = 20

BORDER\_WIDTH = 2

def \_\_init\_\_(self, x, y, batch):

self.node\_border = Circle(x, y, self.RADIUS + self.BORDER\_WIDTH, color=(0, 0, 0, 255), batch=batch)

self.node\_fill = Circle(x, y, self.RADIUS, color=(255, 255, 255, 255), batch=batch)

self.node\_value = Label(

text="", x=x, y=y,

color=(255, 255, 255, 255),

font\_size=12,

anchor\_x="center", anchor\_y="center",

batch=batch

)

def update(self, value):

self.node\_value.text = f"{value:.2f}"

green = int(max(0, value) \* 200)

red = int(abs(min(0, value)) \* 200)

self.node\_fill.color = (red, green, 0, 255)

class Hud:

LABEL\_COLOR = (0, 0, 0, 255)

FONT\_SIZE = 12

HUD\_Y = 520

HUD\_SPACING\_X = 100

NEURON\_SPACING\_Y = 50

NEURON\_SPACING\_X = 50

CANVAS\_HEIGHT = 540

def \_\_init\_\_(self, simulation\_round, dimensions, batch):

self.round\_label = Label(

text=f"Round: {simulation\_round}",

x=20, y=self.HUD\_Y,

color=self.LABEL\_COLOR,

font\_size=self.FONT\_SIZE,

batch=batch

)

self.population\_label = Label(

text="", x=20 + self.HUD\_SPACING\_X, y=self.HUD\_Y,

color=self.LABEL\_COLOR,

font\_size=self.FONT\_SIZE,

batch=batch

)

self.speed\_label = Label(

text="", x=20 + self.HUD\_SPACING\_X \* 2, y=self.HUD\_Y,

color=self.LABEL\_COLOR,

font\_size=self.FONT\_SIZE,

batch=batch

)

self.neurons = []

self.\_create\_neuron\_sprites(dimensions, batch)

def \_create\_neuron\_sprites(self, dimensions, batch):

x = 40

for neuron\_count in dimensions:

total\_layer\_height = neuron\_count \* self.NEURON\_SPACING\_Y - 10

y\_start = self.CANVAS\_HEIGHT - (self.CANVAS\_HEIGHT - total\_layer\_height) / 2

for i in range(neuron\_count):

y = y\_start - i \* self.NEURON\_SPACING\_Y

self.neurons.append(NeuronSprite(x, y, batch))

x += self.NEURON\_SPACING\_X

def update(self, network, alive, population, speed):

self.population\_label.text = f"Population: {alive}/{population}"

self.speed\_label.text = f"Speed: {speed:.2f}"

index = 0

for input\_value in network.inputs:

self.neurons[index].update(input\_value)

index += 1

for layer in network.layers:

for output\_value in layer.outputs:

if index < len(self.neurons): # безопасность

self.neurons[index].update(output\_value)

index += 1

import random

import math

class Layer:

def \_\_init\_\_(self, input\_count, output\_count):

self.outputs = [0.0] \* output\_count

self.weights = [

[random.uniform(-1, 1) for \_ in range(input\_count)]

for \_ in range(output\_count)

]

def feed\_forward(self, inputs):

for i, weight\_set in enumerate(self.weights):

weighted\_sum = sum(input\_val \* weight for input\_val, weight in zip(inputs, weight\_set))

self.outputs[i] = math.tanh(weighted\_sum)

class Network:

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

"""

dimensions: список количества нейронов на каждом уровне, например [5, 4, 2]

"""

self.dimensions = dimensions

self.has\_reached\_goal = False

self.highest\_checkpoint = 0

self.smallest\_edge\_distance = 100 # нужно инициализировать, если используется в RankableChromosome

self.layers = [

Layer(dimensions[i], dimensions[i + 1])

for i in range(len(dimensions) - 1)

]

def feed\_forward(self, inputs):

self.inputs = list(inputs) # сохраняем для HUD

for layer in self.layers:

layer.feed\_forward(inputs)

inputs = list(layer.outputs)

return self.layers[-1].outputs

def serialize(self):

chromosome = [

weight

for layer in self.layers

for neuron\_weights in layer.weights

for weight in neuron\_weights

]

return RankableChromosome(

highest\_checkpoint=self.highest\_checkpoint,

smallest\_edge\_distance=self.smallest\_edge\_distance,

chromosome=chromosome,

)

def deserialize(self, chromosome):

"""

Применяет значения хромосомы к весам сети

"""

index = 0

for layer in self.layers:

for i in range(len(layer.weights)):

for j in range(len(layer.weights[i])):

layer.weights[i][j] = chromosome[index]

index += 1

class RankableChromosome:

def \_\_init\_\_(self, highest\_checkpoint, smallest\_edge\_distance, chromosome):

self.highest\_checkpoint = highest\_checkpoint

self.smallest\_edge\_distance = smallest\_edge\_distance

self.chromosome = chromosome

def \_\_lt\_\_(self, other):

"""

Сортировка по:

1. Больше пройденных контрольных точек

2. Больше расстояние до ближайшего края (в случае равных КП)

"""

if self.highest\_checkpoint == other.highest\_checkpoint:

return self.smallest\_edge\_distance > other.smallest\_edge\_distance

return self.highest\_checkpoint > other.highest\_checkpoint

import os

import json

import itertools

from pyglet import image

class Track:

ROAD\_COLOR = (75, 75, 75, 255)

IMAGE\_FOLDER = "images"

WIDTH = 960

HEIGHT = 540

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

self.track\_image = image.load(os.path.join(self.IMAGE\_FOLDER, f"track{index}.png"))

self.overlay\_image = image.load(os.path.join(self.IMAGE\_FOLDER, f"track{index}-overlay.png"))

self.checkpoints = self.\_load\_checkpoints(index)

self.map\_matrix = self.\_generate\_map\_matrix()

def \_load\_checkpoints(self, index):

json\_path = os.path.join(self.IMAGE\_FOLDER, f"track{index}.json")

with open(json\_path, "r") as file:

data = json.load(file)

return data["checkpoints"]

def \_generate\_map\_matrix(self):

pitch = self.track\_image.width \* len("RGBA")

pixels = self.track\_image.get\_data("RGBA", pitch)

pixel\_tuples = list(itertools.batched(pixels, 4))

binary\_map = [1 if tuple(pixel) == self.ROAD\_COLOR else 0 for pixel in pixel\_tuples]

width = self.track\_image.width

height = self.track\_image.height

return [binary\_map[i:i + width] for i in range(0, width \* height, width)]

def is\_road(self, x, y):

"""Проверка, находится ли точка (x, y) на дороге"""

if not (0 <= int(x) < self.WIDTH and 0 <= int(y) < self.HEIGHT):

return False

return self.map\_matrix[int(y)][int(x)] == 1

import json

from typing import List, Any

class Storage:

def \_\_init\_\_(self, filename: str):

self.filename = filename

def save(self, chromosomes: List[Any]) -> None:

"""Сохраняет список хромосом в файл."""

try:

with open(self.filename, "w", encoding="utf-8") as file:

json.dump({"chromosomes": chromosomes}, file, ensure\_ascii=False, indent=2)

except IOError as e:

print(f"Ошибка при сохранении: {e}")

def load(self) -> List[Any]:

"""Загружает список хромосом из файла. Возвращает пустой список в случае ошибки."""

try:

with open(self.filename, "r", encoding="utf-8") as file:

data = json.load(file)

return data.get("chromosomes", [])

except (FileNotFoundError, json.JSONDecodeError) as e:

print(f"Ошибка при загрузке: {e}")

return []

import os

from canvas import Canvas

from network import Network

from racetrack import Track

from storage import Storage

# --- Конфигурация ---

TRACK\_INDEX = 3

NETWORK\_DIMENSIONS = (5, 4, 2)

CAR\_IMAGE\_COUNT = 5

POPULATION\_SIZE = 4

MAX\_SIMULATION\_ROUNDS = 5

SAVE\_FILE = "brain.json"

def load\_car\_images(count: int) -> list[str]:

"""Загружает пути к изображениям автомобилей."""

return [os.path.join("images", f"car{i}.png") for i in range(count)]

def create\_networks(count: int, dimensions: tuple) -> list[Network]:

"""Создает список нейросетей заданной конфигурации."""

return [Network(dimensions) for \_ in range(count)]

def main():

car\_image\_paths = load\_car\_images(CAR\_IMAGE\_COUNT)

track = Track(TRACK\_INDEX)

canvas = Canvas(track, car\_image\_paths)

storage = Storage(SAVE\_FILE)

networks = create\_networks(POPULATION\_SIZE, NETWORK\_DIMENSIONS)

best\_chromosomes = storage.load()

for chromosome, network in zip(best\_chromosomes, networks):

network.deserialize(chromosome)

simulation\_round = 1

while simulation\_round <= MAX\_SIMULATION\_ROUNDS and canvas.is\_simulating:

canvas.simulate\_generation(networks, simulation\_round)

simulation\_round += 1

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

main()

import os

from canvas import Canvas

from racetrack import Track

from network import Network

from evolution import Evolution

from storage import Storage

# --- Константы ---

TRACK\_INDEX = 4

NETWORK\_DIMENSIONS = (5, 4, 2)

POPULATION\_COUNT = 40

MAX\_GENERATIONS = 50

KEEP\_COUNT = 4

CAR\_IMAGE\_COUNT = 5

SAVE\_FILE = "brain.json"

# --- Утилиты ---

def load\_car\_images(count: int) -> list[str]:

return [os.path.join("images", f"car{i}.png") for i in range(count)]

def create\_networks(count: int, dimensions: tuple) -> list[Network]:

return [Network(dimensions) for \_ in range(count)]

def restore\_best\_chromosomes(storage: Storage, networks: list[Network]):

"""Загружает сохранённые хромосомы и применяет их к начальной популяции."""

best\_chromosomes = storage.load()

for chromosome, network in zip(best\_chromosomes, networks):

network.deserialize(chromosome)

def print\_generation\_stats(networks: list[Network], generation: int):

avg\_checkpoint = sum(n.highest\_checkpoint for n in networks) / len(networks)

avg\_edge\_dist = sum(n.smallest\_edge\_distance for n in networks[:KEEP\_COUNT]) / KEEP\_COUNT

goals\_reached = sum(n.has\_reached\_goal for n in networks)

print(f"\n=== Результаты поколения {generation} ===")

print(f"- Среднее количество пройденных контрольных точек: {avg\_checkpoint:.2f}")

print(f"- Среднее расстояние до границы (топ {KEEP\_COUNT}): {avg\_edge\_dist:.2f}")

print(f"- Машин достигло цели: {goals\_reached} из {POPULATION\_COUNT}\n")

# --- Основной процесс ---

def main():

car\_image\_paths = load\_car\_images(CAR\_IMAGE\_COUNT)

canvas = Canvas(Track(TRACK\_INDEX), car\_image\_paths)

storage = Storage(SAVE\_FILE)

evolution = Evolution(POPULATION\_COUNT, KEEP\_COUNT)

networks = create\_networks(POPULATION\_COUNT, NETWORK\_DIMENSIONS)

restore\_best\_chromosomes(storage, networks)

generation = 1

while generation <= MAX\_GENERATIONS and canvas.is\_simulating:

print(f"\n=== Simulating Generation {generation} ===")

canvas.simulate\_generation(networks, generation)

if not canvas.is\_simulating:

break

print\_generation\_stats(networks, generation)

# Генерация нового поколения

serialized = [net.serialize() for net in networks]

offspring = evolution.execut(serialized)

storage.save(offspring[:KEEP\_COUNT]) # сохраняем лучших

# Создание новой популяции из потомков

networks = []

for chromosome in offspring:

net = Network(NETWORK\_DIMENSIONS)

net.deserialize(chromosome)

networks.append(net)

generation += 1

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

main()