Санкт – Петербургский государственный университет информационных технологий, механики и оптики.
Кафедра “Компьютерные технологии”

А. Ю. Хазановский, А. А. Шалыто

Применение автоматного программирования при моделировании мультиагентной системы беспилотных автомобилей

Программирование с явным выделением состояний

Проектная документация

Проект создан в рамках
«Движения за открытую проектную документацию»

Санкт-Петербург
2007

Введение

На сегодняшний день существует множество проблем, связанных с обеспечением оптимальности движения автомобилей. Все эти проблемы можно условно разделить на две группы: проблемы движения автотранспорта в городах и проблемы движения автотранспорта на трассах.

Каждая из этих групп обладает своей спецификой. Так, основные проблемы автомобильных потоков в населенных пунктах (в особенности, в крупных) определяются наличием большого количества транспорта и малой пропускной способностью магистралей.

Перечислим эти проблемы. Первая из них состоит в устранении пробок (заторов), возникновение которых связано, как правило, с наличием «узких» участков (участков с малой пропускной способностью, связывающих крупные жилые или рабочие массивы). Вторая проблема – организация движения при обстоятельствах, понижающих пропускную способность на отдельных участках, таких как ДТП, ремонт дороги или выход из строя дорожного оборудования (светофоров). Еще одна проблема, возникающая лишь в крупных городах – организация движения в моменты, когда количество автотранспорта достигает своего пика (например, в часы «пик» или во время проведения массовых мероприятий).

За пределами городов основное значение играют совсем другие факторы, так как проблем связанных с низкой пропускной способностью почти не бывает, а при возникновении, они решаются за счет расширения существующих магистралей или добавления новых, что не представляет большой проблемы при отсутствии плотной застройки (это «в идеале», без учета экономических факторов).

Таким образом, можно выделить следующие факторы, влияющие на движение автомобилей за пределами крупных населенных пунктов. Во-первых, это высокие скорости, которые, как правило, в разы превышают скорость движения автомобилей в городах, которая ограничена 60 км/ч, а на практике часто не превышает и 20 км/ч. Во-вторых, это наличие участков, где активно используемое шоссе пересекает небольшие населенные пункты. В этих случаях водители часто не осознают (или не хотят осознавать), что вероятность наличия на проезжей части людей, животных или других помех крайне высока, несмотря на то, что такой участок также является участком шоссе с хорошим покрытием и широкой проезжей частью. Наконец, следует назвать фактор ограниченной видимости, например, в темное время суток, в плохую погоду или при наличии закрытых поворотов. Из изложенного следует, что основной проблемой на дорогах, проходящих вне населенных пунктов, является безопасность. Конечно, такие проблемы, как организация движения при наличии обстоятельств, понижающих пропускную способность на отдельных участках, также существуют, но порой являются лишь следствием проблем безопасности.

Цель настоящей работы, спроектировать систему, позволяющую моделировать различные дорожные ситуации для того, чтобы иметь возможность прогнозировать и решать проблемы дорожного движения еще до их возникновения на реальных дорогах.

Для достижения этой цели было необходимо создать модель, подробно имитирующую не только физику поведения автомобилей, но и учитывающую также множество факторов, таких как:

- Различные характеристики автомобилей. Например, грузовой автомобиль с большой массой может не успеть вовремя затормозить перед каким-либо препятствием, в то время, как современный легковой автомобиль не испытает в аналогичной ситуации никаких трудностей.
- Различные характеристики (качества) водителей. Например, различное время реакции на события, происходящие на дороге.
- Различные манеры поведения водителей. Например, в одной и той же ситуации действия водителей могут варьироваться от правильных до абсолютно неадекватных. Так в ситуации, когда впереди идущий автомобиль начинает резкое торможение, одни водители начнут тормозить и проверять возможность обгона, другие сосредоточатся лишь на торможении, но найдутся и такие, которые не осознав ситуацию, сразу начнут маневр, не убедившись в отсутствие помех.
- Различные дорожные факторы. Например, темное время суток, осадки, скользкая или мокрая дорога, появление на дороге препятствий или любая комбинация этих факторов.

1. Существующие методы

На сегодняшний день применяется множество методов для решения проблем, перечисленных в разд.1. Большинство из них не являются предметом анализа в данной работе (улучшение качества дорог, ликвидация аварийно опасных автомобилей, не годных к эксплуатации и т.д.).

Остановимся на методах, которые призваны снизить аварийность, просчитать потенциальную пропускную способность трассы в случае ее изменения или опасность отдельных участков трасс в случае возникновения непредвиденных помех. Основным методом, с помощью которого на сегодняшний день определяется потенциальная опасность участка дороги, является метод ,использующий статистику, и ничего кроме нее. Он бесспорно эффективен, так как использует реальные данные о количестве ДТП (или просто аварийно опасных ситуаций). Но, к сожалению, эта информация может быть получена лишь в результате длительной эксплуатации.

Конечно, существуют методы прогнозирования аварийной опасности дорог. Одним из них является метод профессора В.Ф. Бабкова [7]. Качество дороги вычисляется по коэффициентам аварийности и безопасности. Основные трудности метода оценки по коэффициентам аварийности связаны с необходимостью скрупулезного учета множества параметров элементов дорог и характеристик движения, а методики оценки по коэффициентам безопасности — с необходимостью детально моделировать движение автомобилей расчетного транспортного потока. Кроме того, данный метод имеет своей целью не столько прогнозирование аварийно опасных участков, сколько моделирование всей трассы в целом, с учетом затрат на строительство и т.д.

Помимо этого существует программный продукт *Credo*. Система, предназначенная для проектирования нового строительства и реконструкции загородных автомобильных дорог всех технических категорий. Эта система находится в разработке и потому пока трудно объективно оценивать ее плюсы и минусы. Однако уже сейчас ясно, что основными задачами данной системы является проектирование новых дорог «с нуля». В этом процессе учитываются такие факторы, как влияние на экологию, экономическая обоснованность проекта и т.д. Следовательно, что данная система сможет решать задачи оценки безопасности и пропускной способности лишь косвенно. Подробнее существующие альтернативы будут рассмотрены в разд. 1.1. – 1.4.

1.1. Метод использования статистики

С помощью метода использования статистики в наши дни определяется потенциальная опасность участка дороги. Как правило, решения о проведении каких-либо работ, направленных на повышение безопасности, принимается лишь в тех случаях, когда статистика дает нежелательные данные. К сожалению, иногда в расчет берутся лишь серьезные происшествия, и вопрос ставится таким образом, что пока не произойдет какое-то (часто, оговоренное заранее) количество происшествий, ничего сделано не будет. Такой метод бесспорно эффективен, так как использует реальные данные о количестве ДТП (или просто аварийно опасных ситуаций). В этом показателе он не имеет конкурентов, так как другие методики (в том числе и реализованная авторами в данном проекте) лишь прогнозируют те происшествия, которые учитывает статистика. Однако, к сожалению, эта информация может быть получена лишь одним путем. Для этого дорога должна эксплуатироваться в течение долгого времени. И если участок является аварийно опасным, то выводы об этом будут сделаны лишь тогда, когда на нем произойдет достаточно ДТП, часть из которых, возможно, будет с летальным исходом.

Стоит отметить, что такое использование статистики «напрямую» - это лишь один из вариантов. Системы, прогнозирующие безопасность дорог во многом опираются на статистику. В данном проекте использовано множество статистических данных, таких как:

- средние скорости движения автомобилей в зависимости от дорожных условий;
- дистанции, соблюдаемые водителями в зависимости от скорости и дорожных условий;
- видимость при различных дорожных условиях;
- зависимость аварийности от различных факторов (учитывалась при проверке результатов работы).

Следовательно, использование статистических данных является обязательной частью моделирования. Кроме того, оно дает хорошие результаты при прогнозировании характеристик дороги, когда используются общие данные и данные, полученные на схожих участках.

Однако при этом метод, при котором никаких действий не предпринимается до достижения какого-то числа инцидентов, является слишком «дорогим», хотя и дает правдивые данные.

1.2. Инженерные методики расчета коэффициентов

При оценке качества проекта в отечественных проектах принято использовать методику профессора В.Ф. Бабкова [7]. Эта методика опирается на коэффициенты аварийности и безопасности.

Коэффициентами безопасности называют отношение максимальной скорости движения на участке к максимальной скорости въезда автомобилей на этот участок (начальная скорость движения).

Коэффициент аварийности представляет собой произведение частных коэффициентов, учитывающих влияние отдельных элементов плана и профиля, где каждый из таких коэффициентов – отношение количества ДТП на участке дорог с различными элементами плана и профиля к количеству ДТП на эталонном горизонтальном прямом участке дороги (данный коэффициент является лишь усложненной формой коэффициента из разд.1.1.).

При вычислении коэффициентов безопасности и аварийности учитывается множество параметров, которые не имеет смысла приводить в данной работе. Но при этом стоит отметить, что основные трудности методики оценки по коэффициентам аварийности связаны с необходимостью скрупулезного учета множества параметров элементов дорог и характеристик движения, а методики оценки по коэффициентам безопасности – с необходимостью детально моделировать движение автомобилей расчетного транспортного потока.

Методика из работы [7] предназначена в основном не столько для прогнозирования аварийно опасных участков, сколько для моделирования всей трассы в целом, с учетом затрат на строительство, рельефа местности и т.д. Таким образом, несмотря на то, что данная система хорошо зарекомендовала себя при проектирование новых дорог, можно утверждать, что для расчета и прогнозирования дорожных ситуаций она подходит не в полной мере.

1.3. Программные методы

Программные методы рассмотрим на примере системы *Credo*. Эта система предназначена для применения при новом строительстве и реконструкции загородных автомобильных дорог всех технических категорий, транспортных развязок, городских улиц и магистралей. Данная система, как и описанный в предыдущем пункте метод, предназначена для проектирования трасс в целом. Программа получает на вход подробный план местности, разработанный на основе изысканий. По данному плану формируются различные варианты трасс, которые сопровождаются различными графиками, отражающими различные характеристики дороги.

На сегодняшний день данная система находится в разработке и поэтому пока трудно оценивать ее плюсы и минусы, особенно в той части, которая пересекается с целями настоящей работы. Однако уже сейчас ясно, что основными задачами данной системы является проектирование новых дорог «с нуля». В этом процессе учитываются такие факторы, как влияние на экологию, экономическая обоснованность проекта и т.д. К сожалению, данная система сможет решать задачи оценки безопасности и пропускной способности лишь косвенно.

Таким образом, можно утверждать, что, несмотря на то, что у системы *Credo* и данной работы одна область применения, они решают абсолютно разные задачи и нацелены на достижение целей, которые лишь связаны между собой. Таким образом, упомянутые системы не могут никак заменять друг друга.

1.4. Выводы по главе 1

Помимо описанных систем существуют схожие с ними, упоминать которые здесь не обязательно. Кроме того, существуют системы и методы, которые решают задачи лишь косвенно связанные с обеспечением безопасности и расчетом пропускной способности дорог. Существуют также системы, решающие данные проблемы средствами, не подлежащими рассмотрению в данной работе (например, методы улучшения дорожного полотна). Отметим, что для понимания общей ситуации в рассматриваемой области достаточно описания приведенного в разд. 1.1 – 1.3.

Как показано в предыдущих пунктах, существующие системы либо не решают поставленных вопросов (разд. 1.2, 1.3), либо решают их не оптимально (разд. 1.1). Таким образом, на основании трех методов, описанных ранее, можно утверждать, что поднятый в данной работе вопрос является открытым и не имеет на сегодняшний день окончательного решения.

В данной работе будет показано, что предлагаемый подход может быть использован для решения вопросов безопасности и пропускной способности дорог. Будет обоснована его эффективность на основании проведенных тестов, а также показано, что данная методика способна если не дать четкую оценку, то, по крайней мере, показать тенденции, которые проявляются либо проявятся в будущем на исследуемых магистралях.

2. Общие принципы работы

Первоочередной задачей была разработка физической модели для моделирования, как самих автомобилей, так и их взаимодействия на шоссе. Для этого было написано несколько классов данных, основными из которых являются классы `Car` и `Road` (разд. 3.3 и 3.4). Эти классы содержат необходимые поля для реализации всех взаимодействий, возможных на дороге (взаимодействий между автомобилями, между автомобилем и дорогой, а также между автомобилем и препятствиями). Кроме этого класс `Car` содержит методы, необходимые для изменения состояния автомобиля (методы, «поворачивающие» руль, «нажимающие» на педали и т.д.) а также методы для получения информации обо

всех физических параметрах автомобиля (методы, возвращающие значения текущей скорости, ускорения, положения руля и т.д.).

Также необходимо было создать механизм, с помощью которого возможно было бы получать информацию более сложную, чем текущие характеристики автомобиля. Для этих целей был создан класс `QueryTool`. Этот класс позволяет получать различную информацию о ситуации на дороге или о параметрах конкретного автомобиля. Класс `QueryTool` имеет доступ к полям классов `Car` и `Road`. На основе данных из этих полей он проводит вычисления для получения дополнительной информации, например, о расстоянии между автомобилями. Условно методы вычисления можно разделить на три группы: методы, формирующие информацию об одном автомобиле, методы, формирующие информацию о взаимном расположении двух автомобилей и методы, предназначенные для получения информации обо всей дороге в целом. Перечислим некоторые методы этого класса:

- `isCarAhead` – позволяет проверить наличие автомобиля непосредственно впереди. Иначе говоря, ищет помеху, препятствующую продолжению движения.
- `isCarRight` – позволяет проверить наличие автомобиля непосредственно справа. Иначе говоря, ищет помеху, препятствующую перестроению вправо.
- `getYDistance` – позволяет узнать дистанцию до впереди идущего автомобиля. Используется для определения скорости относительно помехи. Далее по этой информации принимается решение о необходимости обгона или торможения.

В итоге в классе `QueryTool` реализованы методы, позволяющие получать различную информацию о ситуации на дороге.

Следующим этапом реализации было написание системы искусственного интеллекта, управляющего действиями автомобиля. Для этой цели применяется объектно-ориентированное программирование с явным выделением состояний, основанное на применении конечных автоматов. Такой подход оказался удобным для проектирования логики автопилота, ее реализации и последующей модификации. Для облегчения реализации и понимания логики работы автопилота было решено разделить управляющий автомат на два: первый для управления скоростью (изменяет ускорение автомобиля), второй для управления направлением движения (изменяет положение руля автомобиля). Подробнее о логике управления написано в разд. 4.2 – 4.4. Для достижения реалистичности процессов, происходящих на дороге, было создано по несколько автоматов управления скоростью и направлением (подробно они написаны в четвертой главе). Таким образом, удалось смоделировать различные манеры поведения водителей. Следовательно, в идентичных ситуациях автомобили, управляемые различными автоматами, могут принимать разные решения. Точно так же происходит и на реальных дорогах, когда в идентичные ситуации попадают, например новичок и опытный водитель. Для увеличения реалистичности на некоторых переходах автоматов запускается задержка, которая имитирует время реакции водителей.

В основу данной работы был положен реализованный ранее проект «Система управления автомобилем на шоссе» [6]. Проектная документация доступна по адресу: http://is.ifmo.ru/projects/carpilot// При выполнении работы [6] главной целью было доказательство обоснованности применения автоматного программирования при моделировании мультиагентных систем. Требования к реалистичности дорожных взаимодействий были не велики, а все автомобили имели одинаковые управляющие автоматы и потому вели себя идентично.

В работе [6] были созданы основные классы данных (`Car` – класс, описывающий автомобили, `Road` – класс, описывающий дорогу), класс, осуществляющий подсчет основных данных об автомобилях и их взаимном расположении, пара автоматных классов, реализующих искусственный интеллект (два класса `BasicDriverSpeed` и `BasicDriverSteer` работают только в паре и вместе управляют автомобилем, контролируя соответственно маневрирование и ускорение/торможение). Кроме того, был реализован интерфейс, с помощью которого происходит визуализация и управление (увидеть его можно на рисунках в 5 и 6 главах).

В данной работе была поставлена цель, достигнуть высокой реалистичности всех процессов, происходящих на моделируемой трассе, а также создать различные модели поведения автомобилей (различные автоматы управления). Для достижения этой цели был сделан ряд изменений и дополнений, основные из них:

- добавлены новые характеристики моделируемых объектов (автомобилей, трассы, водителей);
- все характеристические константы проверены и изменены (если это было необходимо) в соответствие с реальными данными. В работе [6] все константы (в том числе величины ускорения автомобилей, скорости перестроения и т.д.) были введены «с потолка»;
- добавлены новые участники движения с уникальным поведением (пожарный автомобиль);
- усовершенствована дорога, добавлены сужения и светофоры;
- добавлены четыре автоматных класса, реализующие две модели поведения водителей.

3. Разработка модели. Основные классы

В данном разделе рассматривается основная архитектура моделирующей программы. Приведено лишь описание основных модулей, которого достаточно для ознакомления с функционированием программы.

3. 1. Диаграмма взаимодействия классов

В диаграмме (рис. 1) все классы разделены на четыре группы: `RoadObjects` (классы, описывающие и хранящие данные), `DataManager` (классы, обеспечивающие доступ к данным), `Changers` (управляющие классы), `GUI` (классы взаимодействия с пользователем). Необходимо отметить, что классы группы `DataManager` реализуют взаимодействие между классами данных и управляющими классами. Благодаря этому программа легко модифицируется, а все взаимодействия строго систематизированы. Более подробное описание диаграммы приведено ниже.

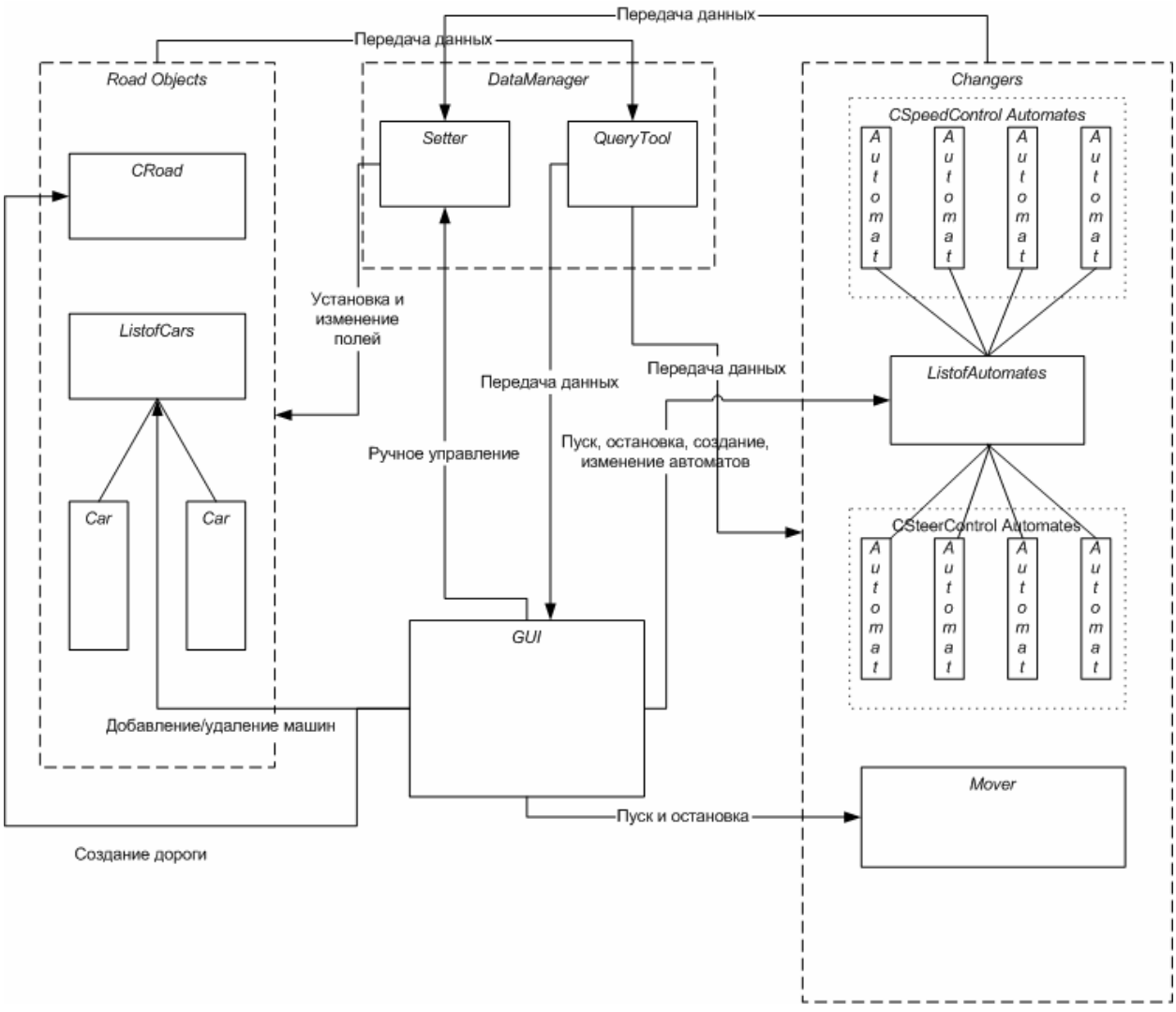

Рис. 1. Диаграмма взаимодействия классов

Все созданные экземпляры класса `Car` записываются в список `ListofCars`, где и хранятся на протяжении всего исполнения программы. Для каждого экземпляра `Car` создается по одному экземпляру `BasicDriverSpeed` и `BasicDriverSteer` (они записываются и хранятся в списке `ListofAutomates`). Эти два класса-автомата ведут управление соответствующим автомобилем на протяжении всего времени работы программы. В процессе работы могут быть добавлены новые автомобили с соответствующими им автоматами, которые принадлежат к указанным выше двум классам.

Классу `Mover` и автоматам необходимо обмениваться информацией с классами, хранящими данные. Обмен происходит через классы `QueryTool` и `Setter`. При этом через класс `QueryTool` можно получить как основные данные (координаты, размеры, скорости), так и информацию на основе вычислений, например, расстояния между автомобилями, наличие помех, расположение автомобилей относительно друг друга.

Класс `Setter` предназначен для исполнения команд автоматов. Когда автомат выполняет переход и дает команду на какое-либо изменение, эта команда передается в класс `Setter`, который и изменяет соответствующие поля класса `Car`.

Класс `Mover` реализует простейшую физику и изменяет координаты автомобилей на дороге с течением времени. На основе параметров, полученных из класса `QueryTool`, он проводит вычисления. Их результаты передаются в классы данных через класс `Setter`. Класс `Mover` функционирует в течение всего времени исполнения программы, его работа может быть приостановлена пользователем.

Классы `GUI` предназначены для визуализации среды моделирования и ручного управления движением. Они также работают через классы `Setter` и `QueryTool`. Все данные (выводимая на экран информация об автомобилях) запрашиваются классами `GUI` через класс `QueryTool`, который, в свою очередь, получает информацию из класса `ListofCars`.

В системе реализована возможность вмешательства в движение: ручное изменение параметров автомобилей или непосредственное управление. При таком вмешательстве изменения в классы данных вносятся через класс `Setter`.

3. 2. Диаграмма отношений классов

Отношения классов приведены на рис. 2. Классы объединены в группы, которые обозначены прерывистыми линиями. Обычным шрифтом описаны стандартные классы *Java*. С помощью стрелок указаны наследования.

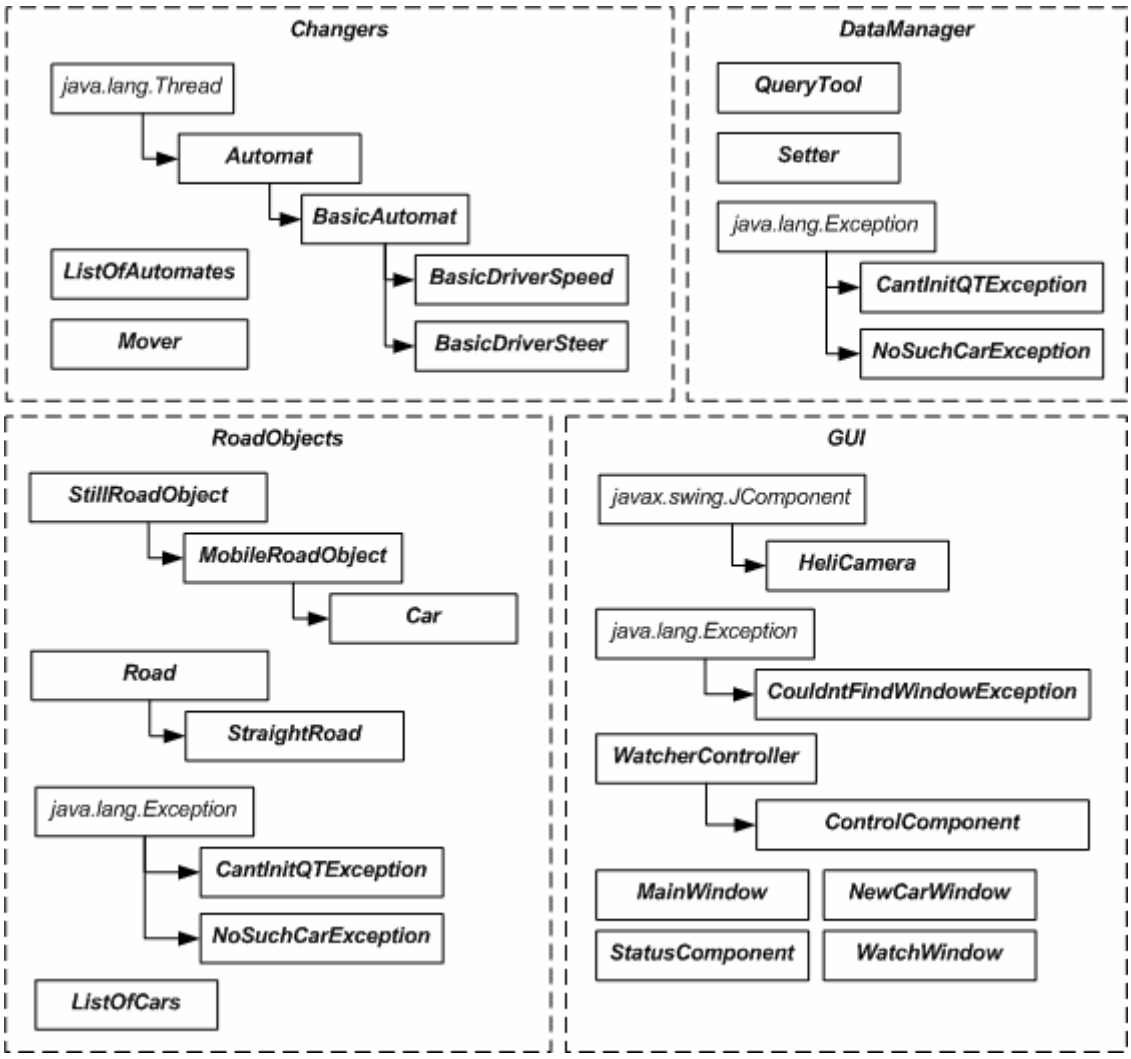

Рис. 2. Диаграмма отношений классов

3.3. Класс `Car`

3.3.1. Словесное описание

Этот класс, описывает автомобиль, который движется по шоссе. Класс `Car` является потомком класса `MobileRoadObject`, в свою очередь, являющегося потоком класса `StillRoadObject` (рис. 2).

Класс `StillRoadObject` описывает неподвижный объект, находящийся на дороге (например, яму, находящуюся на проезжей части), и содержит следующую информацию:

- длина объекта в метрах;
- ширина объекта в метрах;
- координата левого нижнего угла объекта по оси X;
- координата левого нижнего угла объекта по оси Y.

Потомок класса `StillRoadObject` – класс `MobileRoadObject` – описывает объект, равномерно движущийся (с постоянной скоростью и ускорением) по дороге. Он содержит следующие поля:

- текущая скорость объекта (пока еще абстракции) в км/ч;
- положение руля в данный момент времени.

Класс `Car` описывает автомобиль, находящийся на дороге. Кроме полей классов-предков, он содержит также поля, описывающие манеру поведения и технические характеристики автомобиля:

- приоритетная скорость в км/ч – скорость, с которой автомобиль должен двигаться при отсутствии помех;
- максимально возможная скорость движения автомобиля в км/ч;
- флаг «*нужен обгон*», сигнализирующий о необходимости осуществления обгона.
- мощность (ускорение) автомобиля в м/с^2 – описывает динамику автомобиля при разгоне;
- тормозное усилие – ускорение автомобиля при торможении в м/с^2;
- скорость перестроения – скорость движения автомобиля при перестроении, перпендикулярная направлению его движения (в км/ч);
- время реакции – время реакции автомобиля на изменения дорожной ситуации.

Таким образом, класс `Car` полностью описывает автомобиль, движущийся по шоссе. Этот класс содержит информацию о размерах, координатах, технических характеристиках автомобиля, а также о его состоянии в данный момент. Все созданные экземпляры класса `Car` хранятся в списке класса `ListofCars`. Автомобили могут быть созданы при запуске программы или добавлены во время ее выполнения.

Последние четыре поля из списка (мощность, тормозное усилие, скорость перестроения и время реакции) дают возможность создавать различные автомобили. С помощью этих параметров можно смоделировать как динамичный, хорошо управляемый автомобиль с умелым водителем, так и тяжелый грузовик, управляемый усталым водителем.

3.3.2. Структурная схема

Структурная схема класса `Car` приведена на рис. 3.

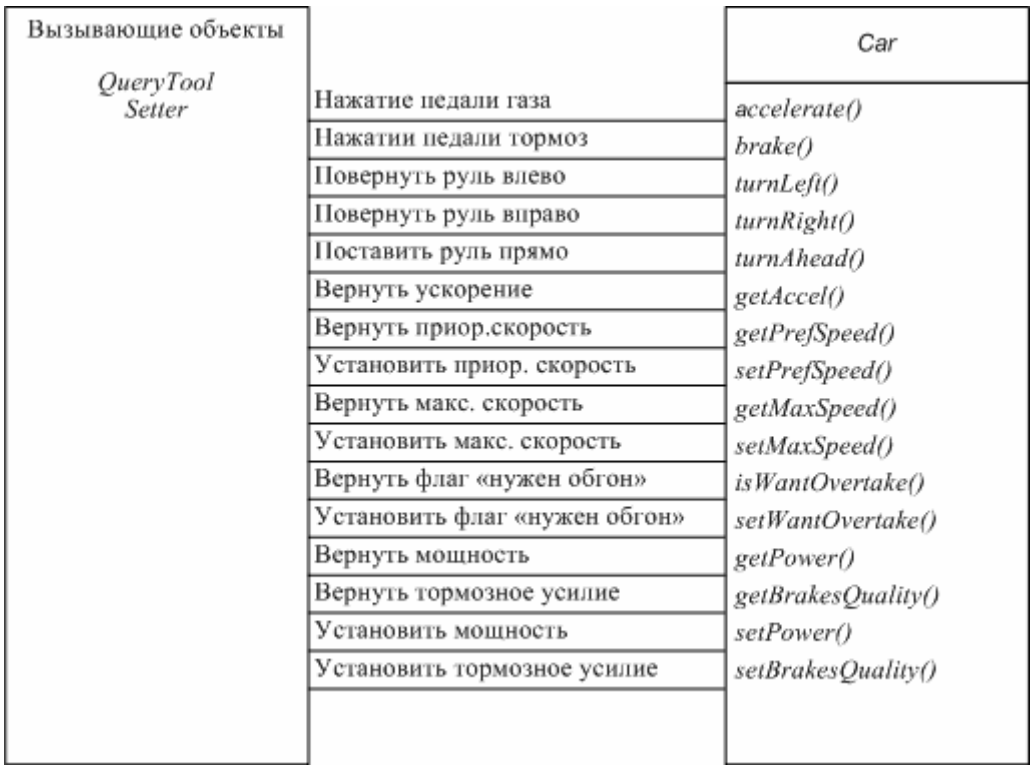

Рис. 3. Структурная схема класса `Car`

3.3.3. Листинг класса Car

Здесь приведен листинг класса Car. По сути, он содержит описание полей и методов для работы с ними.

```
public class Car extends MobileRoadObject{
   // Костанты.

   /** Предпочитаемая скорость по умлочанию (км/ч). */
   public final static double DEF_PREF_SPEED = 60;

   /** Максимальная скорость по умлочанию (км/ч). */
   public final static double DEF_MAX_SPEED = 120;

   /** Мощность по умлочанию (м/с^2). */
   public final static double DEF_POWER = 3;
   /** Тормозное усилие по умлочанию (м/с^2). */
   public final static double DEF_BRAKES = 7;

   /** Скорость перестроения по умлочанию (м/с). */
   public final static double DEF_STEER_QUALITY = 1;

   // Поля.

   /** Желаемая (предпочитаемая) скорость (км/ч). */
   private double prefSpeed;

   /** Максимальная скорость (км/ч). */
   private double maxSpeed;
```

```
    /** Ускорение по X. Другими словами - состояние педали газа (в м/с^2). */
    private double accel;

    /** Мощность машины - величина ускорения машины (в м/с^2). */
    private double power;

    /** Качество тормозов - величина интенсивности торможения (в м/с^2). */
    private double brakesQuality;

    /** Скорость перестроения - величина скорости с которой машина может перестраиваться (в м/с^2). */
    private double steerQuality;

    /** "Желания" автомобиля (нужно для взаимодействия автоматов): */

    /** "Желание перестроиться". */
    private boolean wantOvertake;


    // Конструкторы.

    /** Стандартный конструктор, все значения - "по умолчанию". */
    public Car() {
        prefSpeed = DEF_PREF_SPEED;
        maxSpeed = DEF_MAX_SPEED;
        accel = 0;
        power = DEF_POWER;
        brakesQuality = DEF_BRAKES;
        steerQuality = DEF_STEER_QUALITY;
        wantOvertake = false;
    }

    /**
     * Конструктор, устанавливающий все поля.
     * @param x - координата X левого нижнего угла (в метрах).
     * @param y - координата Y левого нижнего угла (в метрах).
     * @param width - ширина (в метрах).
     * @param lenght - длина (в метрах).
     * @param curSpeed - текущая скорость движения.
     * @param steerSpeed - "угол" поворота руля. Другими словами - скорость по Y.
     * @param prefSpeed - желаемая (предпочитаемая) скорость (км/ч).
     * @param maxSpeed - максимальная скорость (км/ч).
     * @param accel - ускорение по X. Другими словами - состояние педали газа (в м/с^2).
     * @param power - мощность машины - величина ускорения машины (в м/с^2).
     * @param steerQuality - Скорость перестроения - величина скорости с которой машина может
перестраиваться (в м/с^2).
     * @param brakesQuality - качество тормозов - величина интенсивности торможения (в м/с^2).
     * @param wantOvertake - "желание перестроиться".
     */
    public Car(double x, double y, double width, double lenght,
            double curSpeed, double steerSpeed,
            double prefSpeed, double maxSpeed, double accel,
            double power, double steerQuality, double brakesQuality, boolean wantOvertake) {
        super(x, y, width, lenght, curSpeed, steerSpeed);
        this.prefSpeed = prefSpeed;
        this.maxSpeed = maxSpeed;
        this.accel = accel;
        this.power = power;
        this.steerQuality = steerQuality;
        this.brakesQuality = brakesQuality;
        this.wantOvertake = wantOvertake;
    }
```

```
// Остальные методы.

/** Вызов метода "нажимает на тормоз". */
public synchronized void brake() {
   accel = (-1) * brakesQuality;
}

/** Вызов метода "нажимает на газ". */
public synchronized void accelerate() {
   accel = power;
}

/** Вызов метода "убирает ногу с педали газа". */
public synchronized void coolDown() {
   accel = 0;
}

/** Мгновенно поворачивает руль влево. */
public synchronized void turnLeft() {
   setSteerSpeed( (-1) * steerQuality);
}

/** Мгновенно поворачивает руль вправо. */
public synchronized void turnRight() {
   setSteerSpeed(steerQuality);
}

/** Мгновенно устанавливает руль прямо. */
public synchronized void turnAhead() {
   setSteerSpeed(0);
}

// Методы доступа к полям.

public double getAccel() {
   return accel;
}

public synchronized void setAccel(double accel) {
   this.accel = accel;
}

public double getPrefSpeed() {
   return prefSpeed;
}

public synchronized void setPrefSpeed(double prefSpeed) {
   this.prefSpeed = prefSpeed;
}

public double getMaxSpeed() {
   return maxSpeed;
}

public synchronized void setMaxSpeed(double maxSpeed) {
   this.maxSpeed = maxSpeed;
}

public boolean isWantOvertake() {
   return wantOvertake;
}
```

```
    public synchronized void setWantOvertake(boolean wantOvertake) {
        this.wantOvertake = wantOvertake;
    }

    public double getPower() {
        return power;
    }

    public void setPower(double power) {
        this.power = power;
    }

    public double getBrakesQuality() {
        return brakesQuality;
    }

    public void setBrakesQuality(double brakesQuality) {
        this.brakesQuality = brakesQuality;
    }

    public double getSteerQuality() {
        return steerQuality;
    }

    public void setSteerQuality(double steerQuality) {
        this.steerQuality = steerQuality;
    }
}
```

3.4. Класс `Road`

Этот класс описывает дорогу. Он содержит поля (длина и ширина дороги, которые устанавливаются при запуске программы) и реализует методы доступа к ним, а также методы, позволяющие определить расстояние от данной точки до правой или левой обочины. Также данный класс содержит информацию о светофорах, расположенных на дороге, и участках с плохой видимостью или плохим дорожным покрытием.

3.5. Выводы по главе 3

В главе 3 описана структура реализуемой программы и классы данных. Приведена лишь основная архитектура, необходимая для понимания общих принципов. В эту главу не вошло описание классов, принимающих решения – классов, реализующих искусственный интеллект (о них речь пойдет в главе 4).

4. Разработка модели поведения

4.1. Реализация автоматов управления

Каждая машина управляется двумя параллельно работающими автоматами, которые образуют систему управления. В этой системе каждый автомат выполняет свою функцию: один управляет рулем, а другой – торможением и ускорением. Взаимодействие между автоматами происходит через специальный флаг «Нужен обгон», который выставляется и сбрасывается автоматом `DriverSpeed`. Значение этого флага используется автоматом `DriverSteer`.

Каждый автомат реализован в виде класса. При инициализации каждого экземпляра автоматного класса, создается и запускается отдельный поток, в котором с заданной частотой выполняются переходы, соответствующие графу переходов данного автомата. Это объясняется тем, что для управления каждым автомобилем создаются отдельные экземпляры одних и тех же автоматных классов. Каждый экземпляр класса должен управлять своей машиной постоянно.

Автоматные классы, относящиеся к одной системе управления, имеют совпадающие методы, относящиеся к входным и выходным воздействиям. Поэтому удобно создать промежуточный автоматный класс `BasicAutomat`. Этот класс описывает совпадающие методы и имеет пустой граф переходов.

В силу того, что автоматные классы имеют еще и другие одинаковые методы (например, инициализация и работа с потоками), целесообразно создать некий скелет – класс `Automat` с пустым графом переходов, который был бы предком автоматных классов. Таким образом, автоматные классы наследуют «скелетный» класс, перегружая (заменяя) только его метод, описывающий граф переходов.

Выбранная архитектура построения автоматов обеспечивает возможность создания различных систем управления, каждая из которых может содержать любое количество автоматных классов.

Для моделирования различных манер вождения создано по несколько автоматов типа `DriverSpeed` и `DriverSteer` (подробнее в разд. 4.2. – 4.4.).

4.2. Класс `BasicAutomat`

4.2.1. Словесное описание

Этот класс – предок всех классов типа `DriverSpeed` и `DriverSteer`. Он является автоматом с пустым графом переходов и содержит только необходимые методы и константы для дальнейшей реализации двух автоматных классов типа `DriverSpeed` и `DriverSteer`. Они перегружают только метод `makeSwitch`, реализуя в нем граф переходов, так как все необходимые методы для вычислений условий переходов уже описаны в данном классе.

4.2.2. Структурная схема

Структурная схема класса `BasicAutomat` приведена на рис. 4.

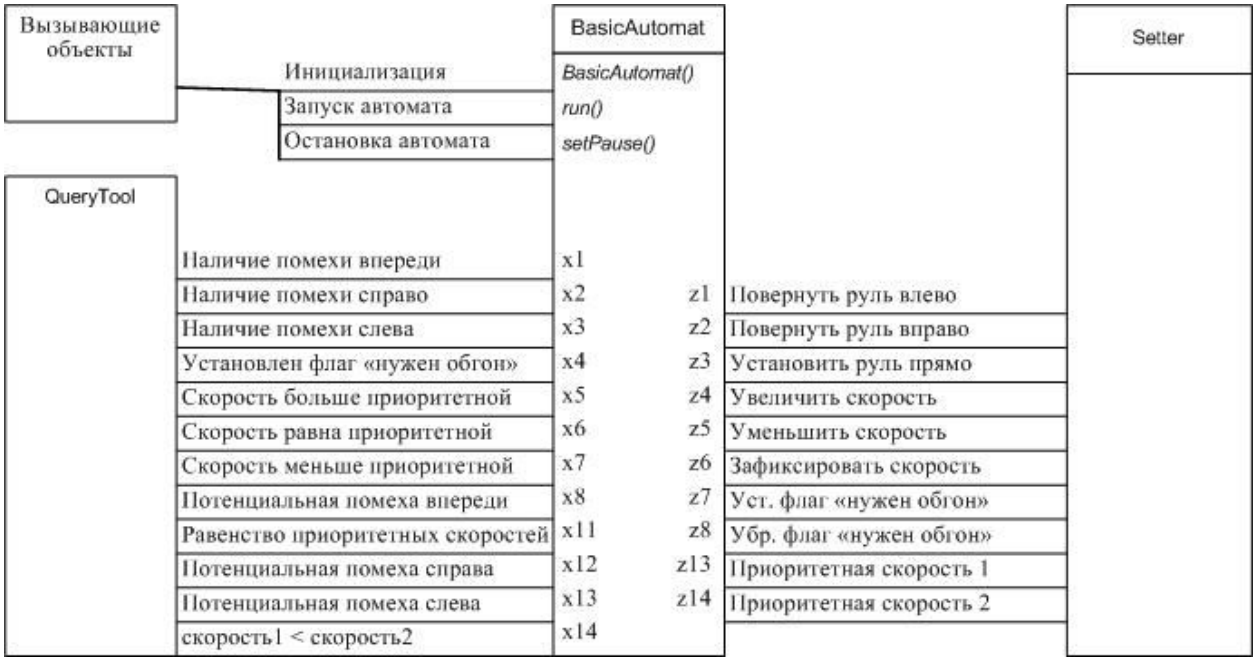

Рис. 4. Структурная схема класса `BasicAutomat`

4.2.3. Пояснения к структурной схеме

Входные переменные:

- `X1` – Наличие помехи впереди. По расстоянию до впереди идущей машины и разности скоростей вычисляется время до достижения этого автомобиля. Оно сравнивается с пороговым значением.
- `X2` – Наличие помехи справа. Требуется при смещении вправо (когда данная машина пытается занять правый ряд). Для этого предполагается небольшое смещение вправо, и определяется, не помешает ли какая-либо машина выполнить данный маневр.
- `X4` – Установлен флаг «Н*ужен обгон*». Используется для взаимодействия автоматов `BasicDriverSpeed` и `BasicDriverSteer`.
- `X5` – Текущая скорость машины больше приоритетной.
- `X6` – Текущая скорость машины равна приоритетной.
- `X7` – Текущая скорость машины меньше приоритетной.
- `X8` – Наличие "потенциальной" помехи впереди. Она определяется следующим образом. Вычисляется время до достижения «нашей» машиной впереди идущего автомобиля в предположении, что «наша» машина движется с приоритетной скоростью. Полученная величина сравнивается с пороговым значением.

- `X11` – Равенство приоритетных скоростей. Сравниваются приоритетные скорости «нашей» машины и идущей непосредственно справа от нее.
- `X12` – Наличие помехи справа с учетом желания перестроиться вправо. Учитывая расстояние, на которое необходимо сместиться вправо, чтобы совершить обгон впереди идущей машины, находим машины, которым «наша» машина может создать помеху. Вычисляем время до столкновения. Оно сравнивается с пороговым временем. Здесь же учитывается, не помешает ли «нашей» машине правый край дороги совершить обгон.
- `X13` – Наличие помехи слева с учетом желания перестроиться влево. Учитывая расстояние, на которое необходимо сместиться влево, чтобы совершить обгон впереди идущей машины, находим машины, которым наша машина может создать помеху. Вычисляем время до столкновения. Оно сравнивается с пороговым временем. Здесь же учитывается, не помешает ли «нашей» машине левый край дороги совершить обгон.
- `X14` – Текущая скорость «нашей» машины меньше скорости впереди идущей машины.

`Выходные переменные:`

- `Z1` – Повернуть руль влево.
- `Z2` – Повернуть руль вправо.
- `Z3` – Установить руль прямо.
- `Z4` – Увеличить скорость – устанавливает значение ускорения машины максимально возможным.
- `Z5` – Уменьшить скорость – устанавливает значение ускорения машины равным его тормозному ускорению.
- `Z6` – Зафиксировать скорость – устанавливает значение ускорения машиной равным нулю.
- `Z7` – Установить флаг «Нужен обгон».
- `Z8` – Убрать флаг «Нужен обгон».
- `Z13` – Временно увеличивает приоритетную скорость на константу.
- `Z14` – Устанавливает измененное методом `Z13` значение приоритетной скорости на первоначальное.

4.2.4 Листинг класса *BasicAutomat*

В этом пункте приведен листинг, описывающий входные переменные. Комментарии приведены прямо в коде.

```
protected boolean x1() {
    try {
      int aheadCar = qTool.getAheadCar(car);
      double dist = qTool.getYDistance(car, aheadCar);
```

```
        double relSpeed = qTool.getSpeed(car) - qTool.getSpeed(aheadCar);
        if (relSpeed < 0) {
            // Мы движемся медленнее впереди идущей машины.
            return false;
        }
        if (dist / relSpeed < SAFE_TAU) {
            return true;
        } else {
            return false;
        }

    } catch (CouldntFindCarException e) {
        e.printStackTrace();
        return true;
    } catch (NoSuchCarException e) {
        // Впереди нет ни одной машины.
        return false;
    }
}

/**
 * Наличие помехи справа с учётом желания перестроиться вправо. Учитывая расстояние, на которое
 * необходимо сместиться вправо, чтобы совершить обгон впереди идущей машины, находим машины,
 * которым мы можем создать помеху. Вычисляем время до столкновения. Оно сравнивается с пороговым
        * временем
 * SAFE_TAU. Если время до столкновения меньше SAFE_TAU, то метод вернет true, в противном
 *случае – false.
 * Здесь же учитывается, не помешает ли нам правый край дороги совершить обгон.
 * @return true, если есть помеха, мешающая обгону справа.
 */
protected boolean x12() {
    try {
        int aheadCar = qTool.getAheadCar(car);
        double rightEdge = qTool.getX(aheadCar) + qTool.getCarWidth(aheadCar) + qTool.getCarWidth(car);
        if (rightEdge > qTool.getRoadWidth()) {
            // Мешает дорога.
            return true;
        } else {
            // Дорога не мешает.

            // Создадим ghostCar – воображаемая машина с габаритами данной, но находящаяся
            // в том месте, куда мы хотим перестроиться.
            StillRoadObject ghostCar = new StillRoadObject(
                    qTool.getX(aheadCar) + qTool.getCarWidth(aheadCar), qTool.getY(car),
                    qTool.getCarWidth(car), 0);

            try {
                // Впереди идущая машина для ghostCar.
                int carAheadOfGhost = qTool.getAheadCar(ghostCar, car);
                double aheadDist = qTool.getYDistance(ghostCar, carAheadOfGhost) - qTool.getCarLenght(car);
                double aheadRelSpeed = qTool.getSpeed(car) - qTool.getSpeed(carAheadOfGhost);

                // Если помеха спереди
                if (aheadDist / aheadRelSpeed < SAFE_TAU && aheadRelSpeed > 0 ||
                        // или непосредственно справа
                        aheadDist < 0) {
                    return true;
                }
            } catch (NoSuchCarException e) {
                // Машины впереди нет.
            }
```

```
            try {
                // Сзади идущая машина для ghostCar.
                // Внимание! Тут всё наоборот - чтоб положительные скорости и расстояния получались.
                int carBehindOfGhost = qTool.getBehindCar(ghostCar, car);
                double behindDist = qTool.getYDistance(carBehindOfGhost, ghostCar);
                double behindRelSpeed = qTool.getSpeed(carBehindOfGhost) - qTool.getSpeed(car);

                // Если помеха сзади.
                if (behindDist / behindRelSpeed < SAFE_TAU && behindRelSpeed > 0) {
                    return true;
                }
            } catch (NoSuchCarException e) {
                // Машины сзади нет.
            }
            return false;


        }
    } catch (CouldntFindCarException e) {
        e.printStackTrace();
        return false;
    } catch (NoSuchCarException e) {
        return false;
    }
}

/**
 * Наличие помехи слева с учётом желания перестроиться вправо. Учитывая расстояние, на которое
 * необходимо сместиться слева, чтобы совершить обгон впереди идущей машины, находим машины,
которым мы
 * можем создать помеху. Вычисляем время до столкновения. Оно сравнивается с пороговым временем
 * SAFE_TAU. Если время до столкновения меньше SAFE_TAU, то метод вернёт true, в противном
 * случае – false.
 * Здесь же учитывается, не помешает ли нам слева край дороги совершить обгон.
 * @return true, если есть помеха, мешающая обгону слева.
 */
protected boolean x13() {
    try {
        int aheadCar = qTool.getAheadCar(car);
        double leftEdge = qTool.getX(aheadCar) - qTool.getCarWidth(car) - DIST_EPS;
        if (leftEdge < 0) {
            // Мешает дорога.
            return true;
        } else {
            // Дорога не мешает.
            StillRoadObject ghostCar = new StillRoadObject(
                    qTool.getX(aheadCar) - qTool.getCarWidth(car), qTool.getY(car),
                    qTool.getCarWidth(car), 0);

            try {
                // Впереди идущая машина для ghostCar.
                int carAheadOfGhost = qTool.getAheadCar(ghostCar, car);
                double aheadDist = qTool.getYDistance(ghostCar, carAheadOfGhost) - qTool.getCarLenght(car);
                double aheadRelSpeed = qTool.getSpeed(car) - qTool.getSpeed(carAheadOfGhost);

                // Если помеха спереди
                if (aheadDist / aheadRelSpeed < SAFE_TAU && aheadRelSpeed > 0 ||
                        // или непосредственно справа
                        aheadDist < 0) {
                    return true;
                }
            } catch (NoSuchCarException e) {
```

```
            // Машины впереди нет.
        }

        try {
            // Сзади идущая машина для ghostCar.
            // Внимание! Тут всё наоборот – чтоб положительные скорости и расстояния получались
            int carBehindOfGhost = qTool.getBehindCar(ghostCar, car);
            double behindDist = qTool.getYDistance(carBehindOfGhost, ghostCar);
            double behindRelSpeed = qTool.getSpeed(carBehindOfGhost) - qTool.getSpeed(car);

            // Если помеха сзади.
            if (behindDist / behindRelSpeed < SAFE_TAU && behindRelSpeed > 0) {
                return true;
            }
        } catch (NoSuchCarException e) {
            // Машины сзади нет.
        }
        return false;


    }
  } catch (CouldntFindCarException e) {
    e.printStackTrace();
    return false;
  } catch (NoSuchCarException e) {
    return false;
  }
}

/**
 * Наличие другой машины или края дороги непосредственно справа от машины на расстоянии менее
 * DIST_EPS.
 * @return true, если помеха есть. false – в обратном случае.
 */
protected boolean x2() {
  try {
    double rightEdge = qTool.getX(car) + qTool.getCarWidth(car) + DIST_EPS;

    if (rightEdge - qTool.getRoadWidth() > 0) {
        // Мешает дорога.
        return true;
    } else {
        // Дорога не мешает.
        StillRoadObject ghostCar = new StillRoadObject(
                qTool.getX(car) + qTool.getCarWidth(car), qTool.getY(car),
                DIST_EPS, 0);

        try {
            // Впереди идущая машина для ghostCar.
            int carAheadOfGhost = qTool.getAheadCar(ghostCar, car);
            double aheadDist = qTool.getYDistance(ghostCar, carAheadOfGhost) - qTool.getCarLenght(car);
            double aheadRelSpeed = qTool.getSpeed(car) - qTool.getSpeed(carAheadOfGhost);

            // Если помеха спереди
            if (aheadDist / aheadRelSpeed < SAFE_TAU && aheadRelSpeed > 0 ||
                    // или непосредственно справа
                    aheadDist < 0) {
                return true;
            }
        } catch (NoSuchCarException e) {
            // Машины впереди нет.
        }
```

```
            try {
                // Сзади идущая машина для ghostCar.
                // Внимание! Тут всё наоборот – чтоб положительные скорости и расстояния получались.
                int carBehindOfGhost = qTool.getBehindCar(ghostCar, car);
                double behindDist = qTool.getYDistance(carBehindOfGhost, ghostCar);
                double behindRelSpeed = qTool.getSpeed(carBehindOfGhost) - qTool.getSpeed(car);

                // Если помеха сзади.
                if (behindDist / behindRelSpeed < SAFE_TAU && behindRelSpeed > 0) {
                    return true;
                }
            } catch (NoSuchCarException e) {
                // Машины сзади нет.
            }
            return false;
        }
    } catch (CouldntFindCarException e) {
        e.printStackTrace();
        return false;
    }

}

/**
 * Установлен флаг "Нужен обгон".
 * @return true, если установлен.
 */
protected boolean x4() {
    try {
        return qTool.getWantOverTake(car);
    } catch (CouldntFindCarException e) {
        e.printStackTrace();
    }
    return false;
}

/**
 * Текущая скорость машины больше (в пределах допуска SPEED_EPS) приоритетной.
 * @return true, если больше.
 */
protected boolean x5() {
    try {
        return (qTool.getSpeed(car) > qTool.getPrefSPeed(car) + SPEED_EPS);
    } catch (CouldntFindCarException e) {
        e.printStackTrace();
        return false;
    }
}

/**
 * Текущая скорость машины равна (в пределах допуска SPEED_EPS) приоритетной.
 * @return true, если равна.
 */
protected boolean x6() {
    try {
        return (Math.abs(qTool.getSpeed(car) - qTool.getPrefSPeed(car)) < SPEED_EPS);
    } catch (CouldntFindCarException e) {
        e.printStackTrace();
        return false;
    }
}
```

```
/**
 * Текущая скорость машины меньше (в пределах допуска SPEED_EPS) приоритетной.
 * @return true, если меньше.
 */
protected boolean x7() {
   try {
      return (qTool.getSpeed(car) + SPEED_EPS < qTool.getPrefSPeed(car));
   } catch (CouldntFindCarException e) {
      e.printStackTrace();
      return false;
   }
}

/**
 * Наличие "потенциальной" помехи впереди. По расстоянию до впереди идущей машины и разности её
 * текущей
 * скорости с нашей приоритетной скоростью вычисляется время до достижения этого автомобиля
 * (до возможного столкновения) в предположении, что наша машина будет двигаться с приоритетной
 * скоростью.
 * Оно сравнивается с пороговым временем SAFE_TAU. Если время до столкновения меньше SAFE_TAU,
       * то метод
 * вернёт true, в противном случае – false. Подробнее – см. документацию.
 * @return true, если впереди идущая машина представляет помеху движению с приоритетной скоростью.
 */
protected boolean x8() {
   try {
      int aheadCar = qTool.getAheadCar(car);
      double dist = qTool.getYDistance(car, aheadCar);
      double relSpeed = qTool.getPrefSPeed(car) - qTool.getSpeed(aheadCar);
      if (relSpeed < 0) {
         // Мы движемся медленнее впереди идущей машины.
         return false;
      }
      if (dist / relSpeed < SAFE_TAU) {
         return true;
      } else {
         return false;
      }

   } catch (CouldntFindCarException e) {
      e.printStackTrace();
      return true;
   } catch (NoSuchCarException e) {
      // Впереди нет ни одной машины.
      return false;
   }
}

/**
 * Равенство приоритетных скоростей. Сравниваются приоритетные скорости данной машины
 * и идущей непосредственно справа от нее.
 * @return true, если указанные скорости равны в пределах BIG_SPEED_EPS.
 */
protected boolean x11() {
   try {
      int rightCar = qTool.getRightCar(car);
      double hisSpeed = qTool.getSpeed(rightCar);
      if (Math.abs(hisSpeed - qTool.getSpeed(car)) < BIG_SPEED_EPS) {
         return true;
      } else {
         return false;
```

```
            }
        } catch (CouldntFindCarException e) {
            // Добавить код.
            e.printStackTrace();
            return false;
        } catch (NoSuchCarException e) {
            // Справа нет ни одной машины.
            return false;
        }
    }

    /**
     * Текущая скорость машины меньше скорости впереди идущей машины.
     * @return true, если меньше.
     */
    protected boolean x14() {
        try {
            int aheadCar = qTool.getAheadCar(car);
            double relSpeed = qTool.getSpeed(car) - qTool.getSpeed(aheadCar) + SPEED_EPS;
            if (relSpeed < 0) {
                // Мы движемся медленнее впереди идущей машины.
                return true;
            } else {
                return false;
            }
        } catch (CouldntFindCarException e) {
            e.printStackTrace();
            return false;
        } catch (NoSuchCarException e) {
            return false;
        }
    }
```

4.3. Автомат `BasicDriverSpeed`

В данном разделе приведено описание одного из созданных автоматов типа `DriverSpeed`. Автоматы данного типа управляют торможением и ускорением.

4.3.1. Словесное описание

Этот автоматный класс предназначен для управления скоростью автомобилей, торможением и ускорением. При этом каждый экземпляр класса управляет скоростью одного автомобиля. При инициализации запускается отдельный поток, в котором с заданной частотой совершаются переходы в автомате. Класс получает информацию о дорожной обстановке через класс `QueryTool`. На основе этих данных и выполняются переходы. Результатом работы является изменение ускорения машины. Данный автомат является одним из двух автоматов, непосредственно управляющих автомобилем. Взаимодействие со вторым автоматом (`BasicDriverSteer`) выполняется через флаг «*Нужен обгон*» (как это именно это происходит, описано в разд.4.3.3.).

Основными функциями данного автомата являются:

- поддержание скорости, равной приоритетной;
- заблаговременное изменение скорости при обнаружении помех;
- принятие решения о необходимости перестроения;

- устранение «барьеров» на дороге. Иначе говоря, устранение ситуаций, когда несколько машин едут шеренгой и имеют приблизительно равные скорости. В таком случае они перекрывают большую часть дороги на длительное время.

Для каждой машины создается отдельный экземпляр данного автомата.

4.3.2. Структурная схема

Структурная схема класса `BasicDriverSpeed` приведена на рис.5.

Рис. 5. Структурная схема класса `BasicDriverSpeed`

Пояснения к схеме приведены в разд. 4. 2. 3.

4.3.3. Граф переходов

Граф переходов представлен на рис.6.

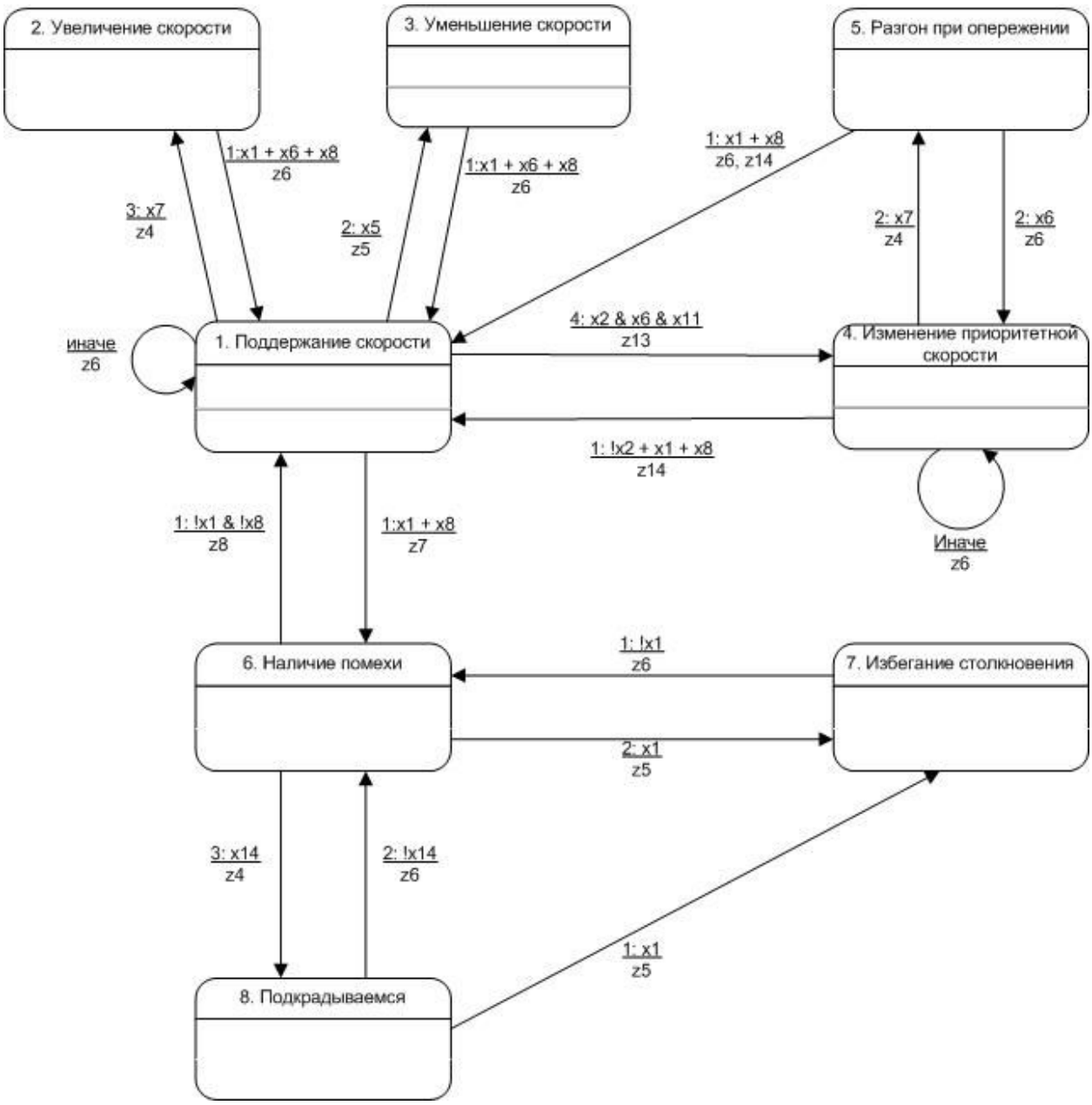

Рис. 6. Граф переходов автомата `BasicDriverSpeed`

Ниже приводятся некоторые пояснения, объясняющие смысл состояний и переходов в графе переходов автомата `BasicDriverSpeed`.

Первое и основное состояние «*1. Поддержание скорости*». Нахождение автомата в этом состояние означает, что не выполняется никаких маневров. Переходы в это состояние осуществляются из состояний, которые изменяют скорость (при выполнении маневра или опережения). Состояния «*2. Увеличение скорости*» и «*3. Уменьшение скорости*» предназначены для изменения скорости с целью приблизить ее к приоритетной. Например, переход в состояние 3 происходит, если скорость меньше нормы (меньше приоритетной), при условии, что не требуется выполнить переход,

связанный с возникновением помехи. Автомат остается в этом состояние до тех пор, пока скорость не сравняется с приоритетной или не возникнет помеха впереди (в том числе и потенциальная).

Состояние «*4. Изменение приоритетной скорости»* создано для разрешения конкретной ситуации. Допустим, наша машина движется с приоритетной скоростью, а справа от нее движется другой автомобиль с той же скоростью. В таком случае возникает заслон – две машины (возможно больше) движутся рядом и заслоняют большую часть дороги. Если отдельно не разбирать эту ситуацию, то такой заслон не исчезнет, пока одна из машин не встретит помеху спереди, а если машины движутся с маленькой скоростью, то заслон может не исчезнуть вовсе.

Переход в состояние 4 выполняется, когда текущая скорость обгоняемой машины равна приоритетной скорости нашего автомобиля. При переходе приоритетная скорость увеличивается и автомобиль начинает разгоняться, если не возникнет помех. Таким образом, «наш» автомобиль опережает движущийся справа и получает возможность для перестроения вправо, которое выполняет автомат `BasicDriverSteer`. Состояние 5 необходимо, чтобы автомобиль разгонялся после увеличения приоритетной скорости. Обратный переход из состояний 4 и 5 в состояние 1 происходит либо, когда исчезает помеха справа (опережение закончено), либо когда возникает помеха впереди. Работа автопилота в данном случае рассмотрена на примере в разд. 5.2.

Переход в состояние «*6. Наличие помехи»* происходит при обнаружении впереди помехи или потенциальной помехи. При этом устанавливается флаг «нужен обгон», который используется автоматом, управляющим рулем (`BasicDriverSteer`). Состояния 7 и 8 предназначены для регулирования скорости относительно идущей впереди помехи. При переходе в состояние «*7. Избегание столкновения»* выполняется торможение, переход в него происходит, если машина впереди едет быстрее контролируемой машины (и если расстояние мало, то она является помехой). Состояние «*8. Подкрадываемся»* выполнено аналогично, но в данном случае скорость увеличивается, чтобы догнать помеху.

4.3.4. Альтернативный автомат управления скоростью

Граф переходов представлен на рис.7.

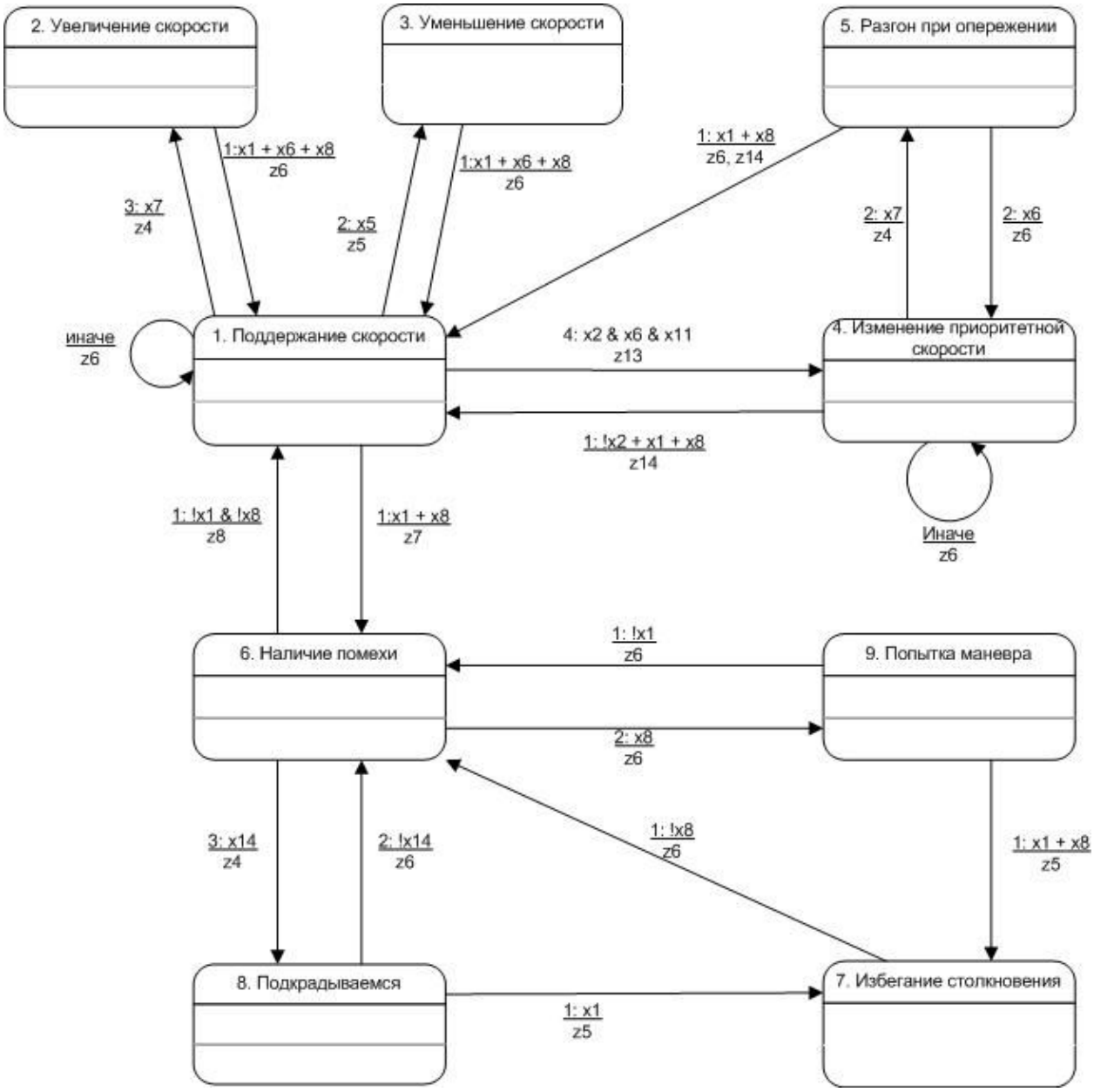

Рис.7. Граф переходов альтернативного автомата управления скоростью

Основное отличие данного автомата от описанного выше – наличие состояния «9. Попытка маневра». Переходя в данное состояние и передавая автомату управления маневрированием флаг «Нужен обгон», автомат ждет, пока не появится реальная помеха x1 (потенциальная помеха уже есть x8). При этом автомат не снижает скорость, а лишь фиксирует ее. В итоге вполне вероятно, что автомату не придется тормозить, произойдет это в том случае, если автомобиль успеет быстро перестроиться. В ситуации, если, например, дорого сильно загружена, и возможности для маневра ограничены, автомобиль не сумеет быстро перестроиться, и тогда автомат начнет торможение, когда приблизится к

впереди идущему автомобилю (то есть, потенциальная помеха x8 превратится в реальную x1).

Таким образом, данный автомат моделирует чуть менее осторожного (чем основной автомат), но более грамотного водителя.

4.4. Автомат `BasicDriverSteer`

В данном разделе приведено описание одного из созданных автоматов типа `DriverSteer`.

4.4.1. Словесное описание

Класс предназначен для управления маневрированием автомобиля. При инициализации запускается отдельный поток, в котором совершаются переходы в автомате с заданной частотой. Класс запрашивает информацию о дорожной обстановке с помощью экземпляра `QueryTool`. На основе этих данных и совершаются переходы. Результатом работы является решение о повороте руля.

Например, сложилась следующая ситуация:

- есть помеха впереди (x1);
- нет помехи слева (!x13);
- скорость меньше нормы (x7).

Тогда будет принято решение о перестроении (повороте руля) влево (z1) для устранения помехи впереди. Тем самым достигается возможность увеличить скорость. Основными функциями данного автомата являются:

- выбор и осуществление оптимального маневра;
- контроль безопасности маневра;
- освобождение левых полос, при наличии свободного пространства справа.

Данный класс является одним из двух классов, непосредственно управляющих автомобилем (второй – `BasicDriverSpeed`). Для каждого автомобиля создается свой экземпляр данного класса.

4.4.2. Структурная схема

Структурная схема класса `BasicDriverSteer` представлена на рис. 8.

Рис. 8. Структурная схема класса `BasicDriverSteer`

Пояснения к схеме приведены в разд. 4. 2. 3.

4.4.3. Граф переходов автомата `BasicDriverSteer`

Граф переходов автомата `BasicDriverSteer` приведен на рис.9. Ниже приводятся некоторые пояснения, объясняющие смысл состояний и переходов.

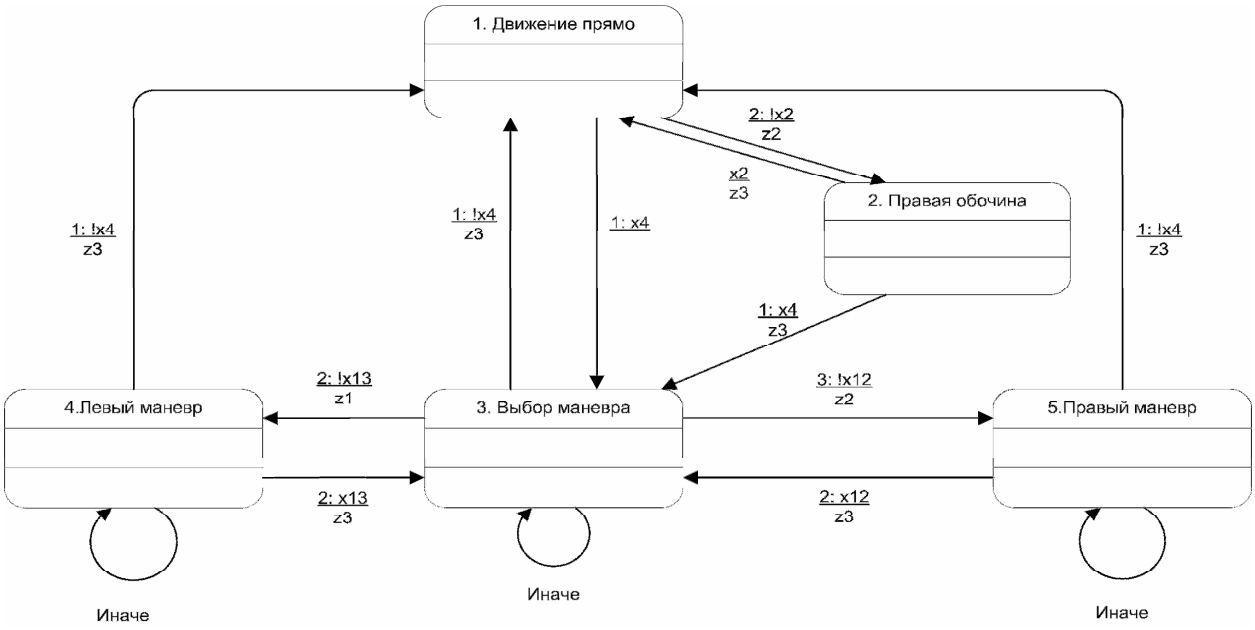

Рис.9. Граф переходов автомата `BasicDriverSteer`

Основным состоянием этого графа переходов является состояние «1. Движение прямо». В этом состоянии автомобиль движется прямо, прижавшись к правой обочине. В случае полного отсутствия помех автомат перейдет в это состояние и останется в нем, пока не будет обнаружена помеха. Состояние «2. Правая обочина» предназначено для перестроения автомобиля вправо в случае наличия свободного места. Это необходимо для освобождения пространства на проезжей части и отвечает требованиям правил дорожного движения. После опережения автомобиля, двигавшегося справа от нашего, (если нет помех) автомат переходит в это состояние и выполняет перестроение вправо, и в итоге «наш» автомобиль оказывается впереди автомобиля, который ранее обгонял. Выход из этого состояния происходит либо при обнаружении правой помехи (это значит, что выполнено перестроение вправо на максимально возможное расстояние), либо при установке флага «Нужен обгон». В последнем случае автомат переходит в состояние «3. Выбор маневра».

Второй переход из состояния 1 осуществляется в случае, когда установлен флаг «Нужен обгон». Этот флаг устанавливает автомат скорости BasicDriverSpeed, когда он не может поддерживать приоритетную скорость, двигаясь прямо. В этом случае автомат переходит в состояние «3. Выбор маневра», в котором выбирает направление перестроения (левое перестроение обладает большим приоритетом). В состояниях «4. Левый маневр» и «5. Правый маневр» машина выполняет перестроение, контролируя появление помех на путь маневра. Выход из этих состояний происходит либо при обнаружении помехи в направлении маневра, либо при исчезновении помехи спереди. Если появилась помеха, препятствующая маневру, то автомат возвращается в состояние 3, поскольку маневр по-прежнему необходим. Если исчезла помеха спереди (и как следствие

исчез флаг «Нужен обгон»), это означает, что маневр завершен и автомат переходит в состояние 1.

4.4.4. Альтернативный автомат управления маневрированием

Граф переходов представлен на рис.10.

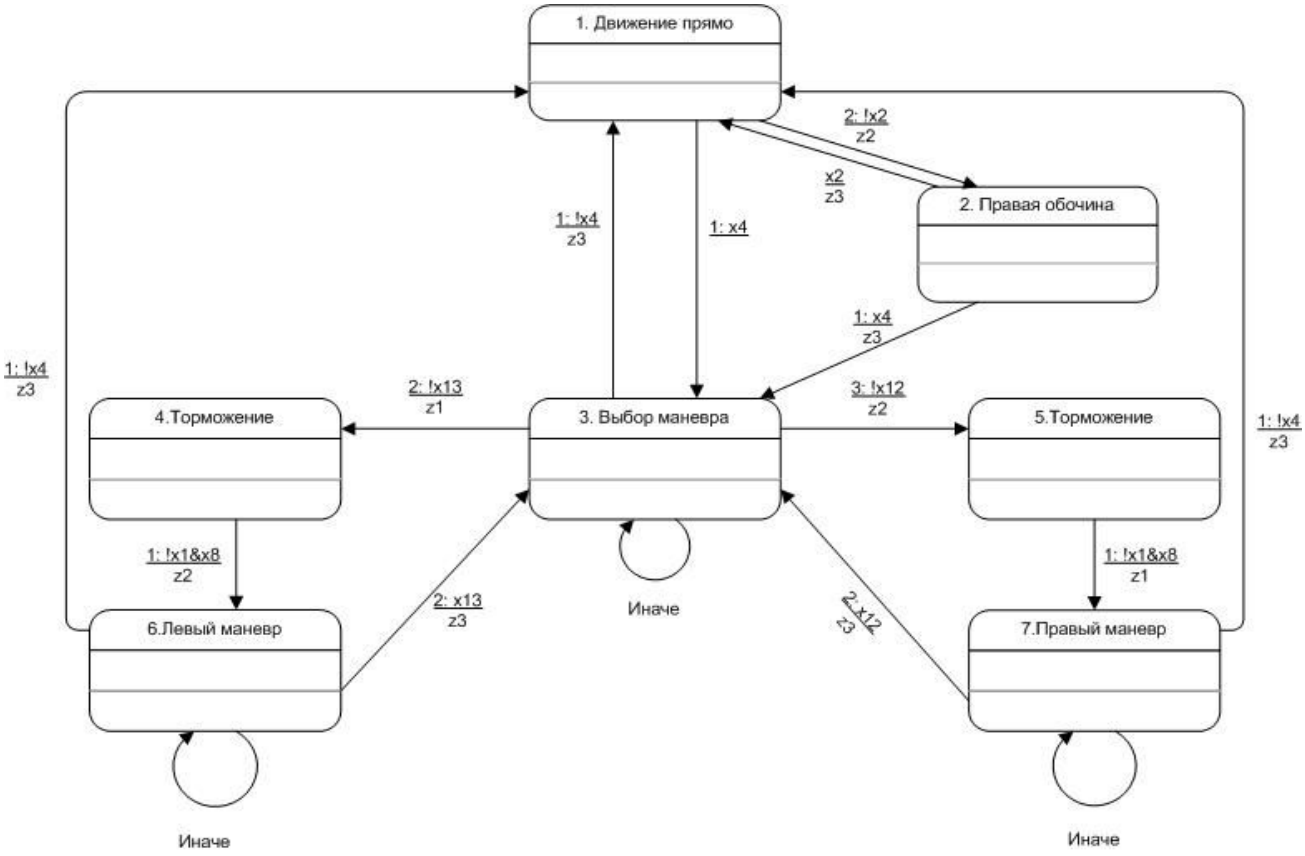

Рис. 10. Граф переходов альтернативного автомата управления маневрированием

Данный автомат моделирует неумелого «однозадачного» водителя. Данный водитель может в конкретный момент времени делать лишь одно действие (либо маневрировать либо тормозить/разгоняться). При взаимодействии основных автоматов, рассмотренных ранее, часто возникали ситуации, когда помеху обрабатывают сразу оба автомата. Один из них снижал скорость, а другой выбирал или осуществлял маневр. Такая система действий является абсолютно правильной, но, к сожалению, не все реальные водителя следуют ей.

Рассмотрим подробно действия данного автомата. После получения флага «Нужен обгон» и перехода в состояние «3.Выбор маневра», данный автомат, как и основной, выбирает какой маневр совершить. Однако вместо того, чтобы начать выбранный маневр, автомат переходит в одно из двух состояний «Торможение» и находится там до тех пор, пока, не исчезнет реальная помеха. Таким образом, имитируется поведение водителя, который полностью сосредоточился на торможении и думает исключительно о том, чтобы затормозить перед впереди идущим автомобилем, забыв о возможности маневра. Когда же

исчезает реальная помеха и необходимость тормозить, автомат начинает совершать маневр.

4.5. Выводы по главе 4

Данная глава посвящена описанию классов, управляющих автомобилями. Сначала описываются общие положения, которым должны удовлетворять эти классы. Затем описывается предок автоматных классов, что дает общее понимание принципов принятия решений и работы управляющих классов. Далее следует подробное описание двух типов классов, реализующий искусственный интеллект, который обеспечивает управление каждым автомобилем на шоссе. Приведены конкретные примеры и схемы классов данного типа. Описаны принципы и объяснена логика, в соответствие с которой принимаются решения. Кроме того, при описании графов переходов, приведено сравнение действий разных автоматов в различных ситуациях.

5. Тестирование

В данном разделе приведены примеры функционирования модели в нормальных условиях. В примерах отсутствуют помехи и непредвиденные ситуации (они будут рассмотрены в шестой главе), таким образом, проверяется лишь работоспособность. Также исключены человеческие факторы, имитирующие ошибки и задержки, свойственные водителям. Целью данного раздела является обоснование работоспособности и реалистичности созданной модели на основе приведенных примеров и рассмотренных ситуаций.

5.1. Двойной обгон

Иллюстрируется так называемый «двойной обгон», ситуация, когда две машины одновременно обгоняют третью. Единственная сложность данной ситуации заключается в том, что последняя машина должна отслеживать смещение впереди идущего автомобиля (второй). Иначе говоря, в начальной ситуации для обгона будет достаточно смещения влево «на корпус», но потом, когда второй автомобиль сам начнет обгон, последний автомобиль должен будет выполнить смещение влево вновь.

На рис.11 показана начальная ситуация, когда белый автомобиль догнал пару машин, одна из которых начинает обгон другой (все автомобили двигаются снизу вверх).

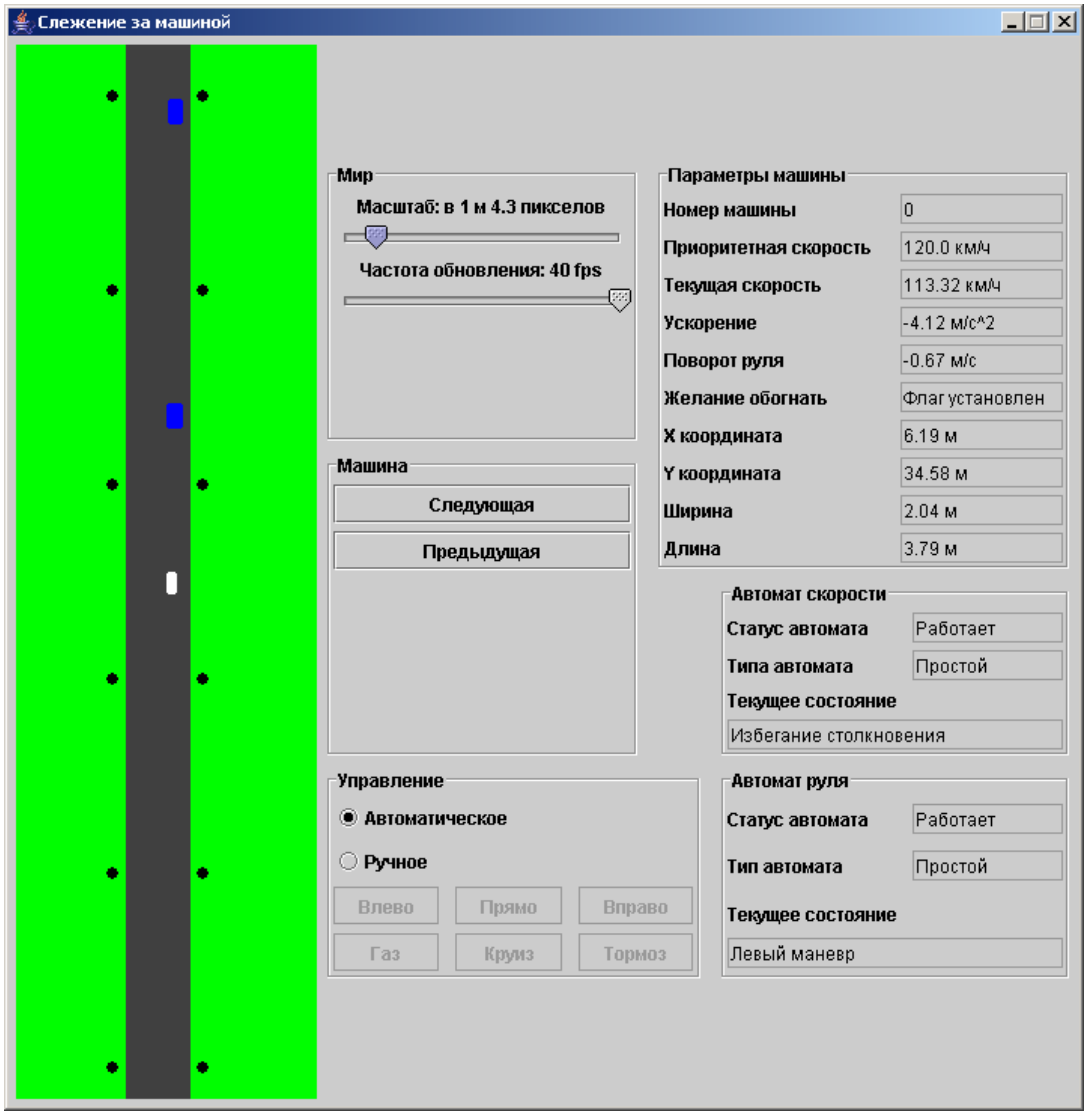

Рис.11. Двойной обгон. Этап 1

На рис.12 представлен промежуточный этап двойного обгона, когда две обгоняющие машины заняли свободные полосы и продолжают обгон.

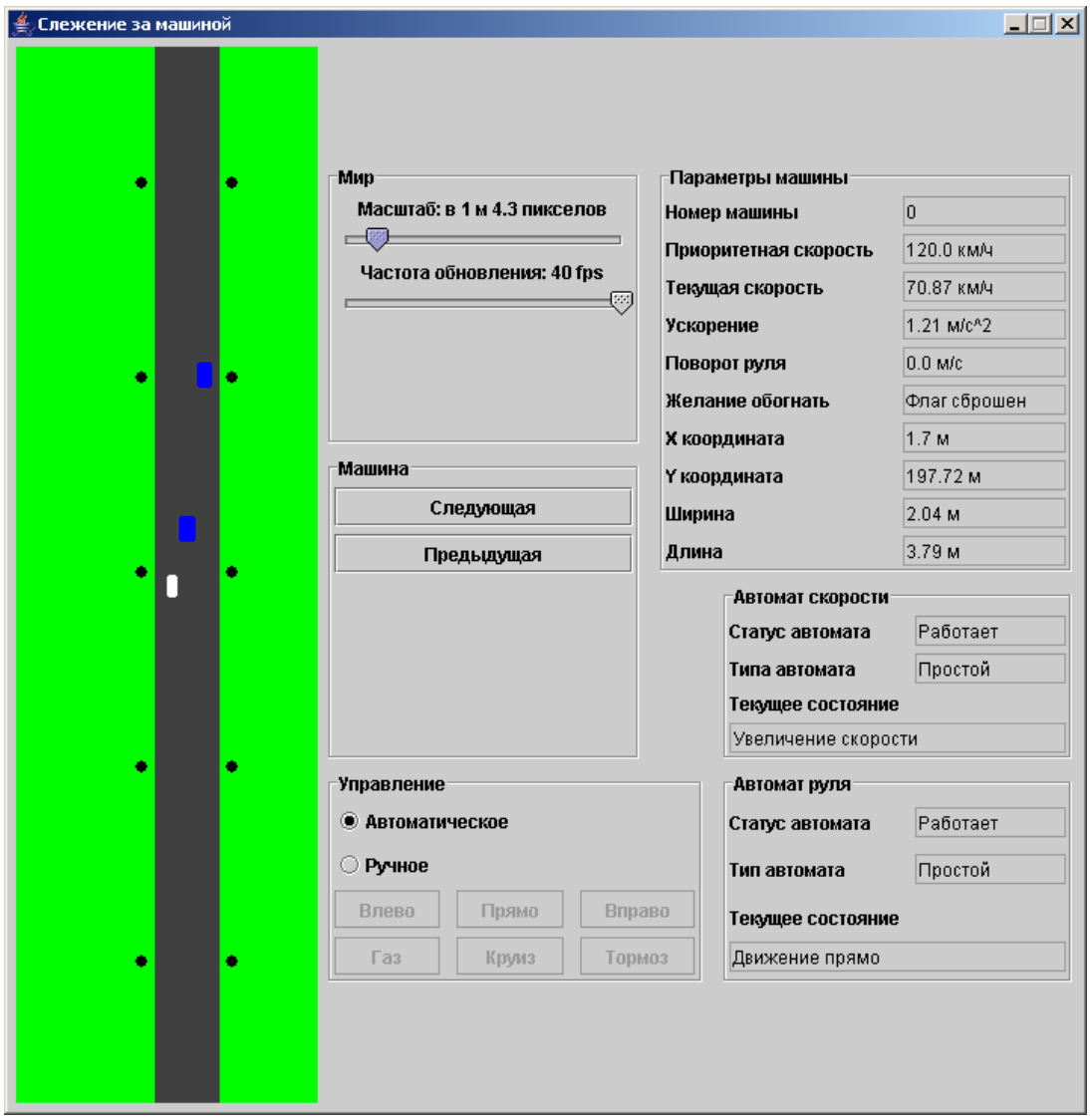

Рис. 12. Двойной обгон. Этап 2

На рис.13 показана заключительная стадия. Обгон уже выполнен, и автомобили удаляются друг от друга, перемещаясь вправо.

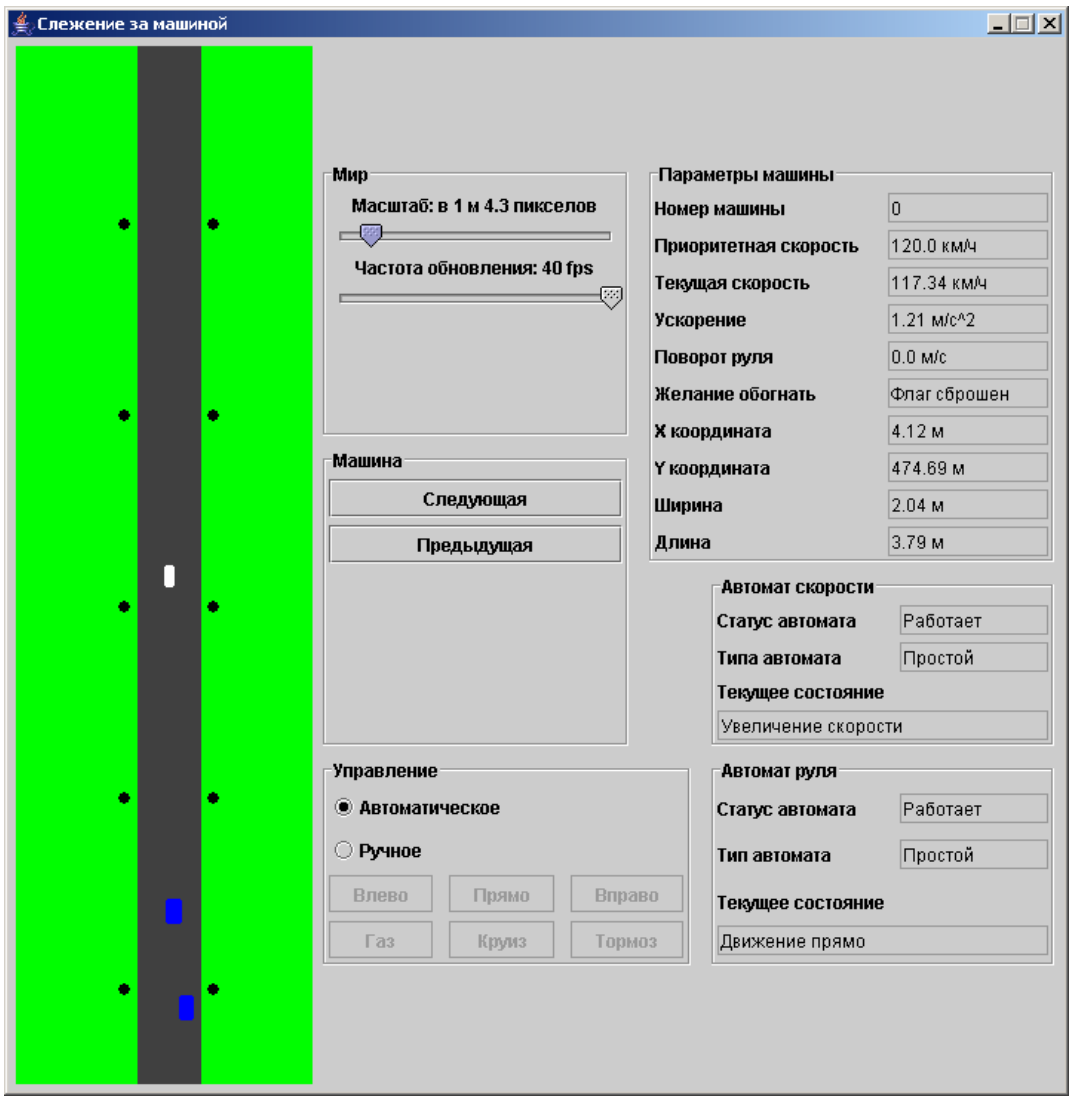

Рис.13. Двойной обгон. Этап 3

На данном примере были протестированы и другие созданные автоматы. Стоит отметить, что когда последним автомобилем управлял «спокойный» автомат, имитирующий неопытного водителя, обгон осуществлялся в два этапа. Сначала обгон осуществил второй автомобиль, а затем последний.

5.2. Механизм повышения приоритетной скорости

Данный механизм предназначен для ликвидации «заторов» на дороге. Иначе говоря, когда несколько автомобилей двигаются «бок о бок» (шеренгой) с близкими скоростями, что может затруднить движение. В таком случае они перекрывают большую часть дороги на длительное время, и пока эти автомобили разъезжаются, чтобы получить возможность перестроиться вправо, сзади может возникнуть затор.

На рис.14 белый автомобиль совершает опережение темного автомобиля, причем приоритетная скорость белой машины (выделена на рисунке) приблизительно равна скорости темной машины. Если обгон совершать без изменения скорости, то он может занять значительное время, что приведет к созданию помехи нормальному движению на шоссе.

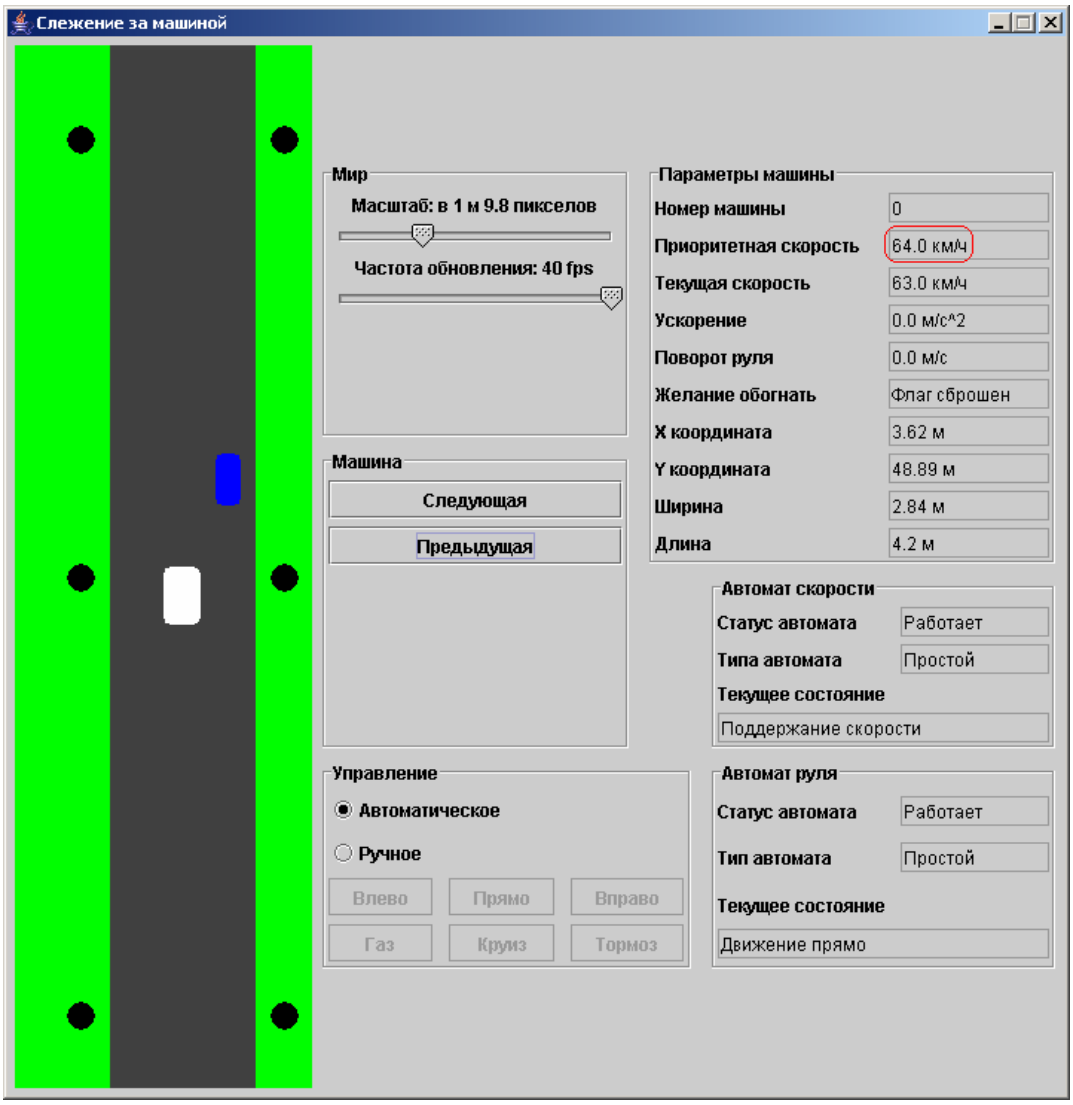

Рис.14. Обгон с ускорением. Этап 1

Поэтому в данной ситуации автопилот белого автомобиля увеличивает его приоритетную скорость, что проиллюстрировано на рис.15. При этом автопилот старается держаться указанной скорости, и машина начинает разгон (текущая скорость вскоре возрастет до приоритетной).

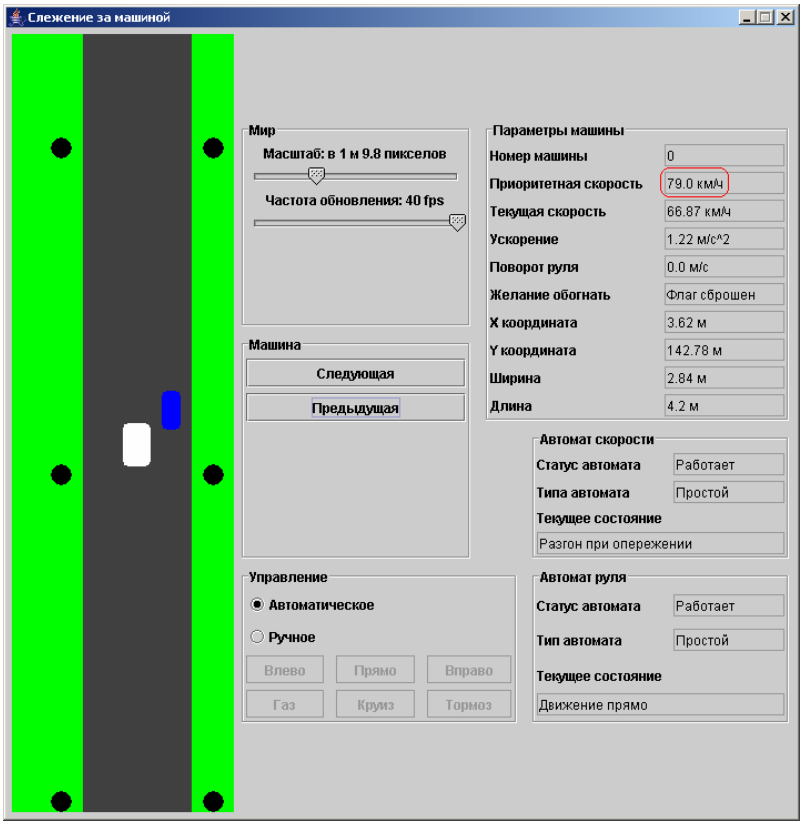

Рис. 15. Обгон с ускорением. Этап 2

Когда обгон выполнен, автопилот устанавливает значение приоритетной скорости таким, каким оно было до начала маневра (рис. 16) и продолжает движение в обычном темпе.

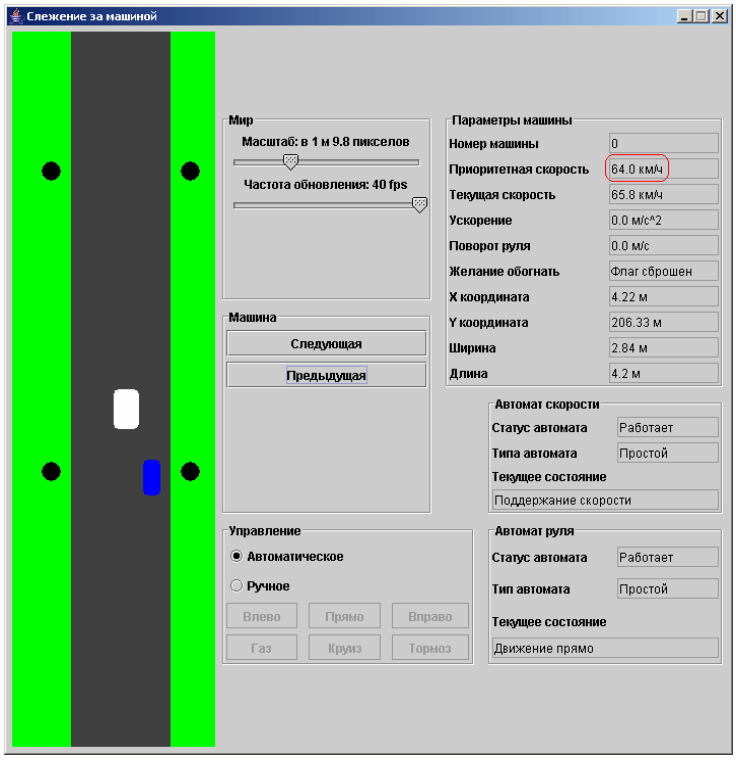

Рис. 16. Обгон с ускорением. Этап 3

5.3. Препятствия на дороге

Иллюстрируется ситуация, когда на дороге возникло непредвиденное препятствие, в данном случае – ДТП с участием двух автомобилей (рис. 17). К месту ДТП подъезжают два автомобиля, причем автомобиль Car2 не рассматривает ДТП как препятствие, так как движется левее. В то же время ДТП находится на пути выделенного автомобиля (*Car1*) и потому его автопилот принимает решение о маневре (на рисунке текущее состояние автомата руля – выбор маневра). Однако в данный момент маневр невозможен, так как смещение вправо не даст никакого результата (помеха не будет устранена), а смещение влево нежелательно, так как автомобиль *Car1* «подрежет» автомобиль *Car2*. В итоге автомобиль *Car1* на маленькой скорости движется к препятствию, ожидая появления возможности для перестроения.

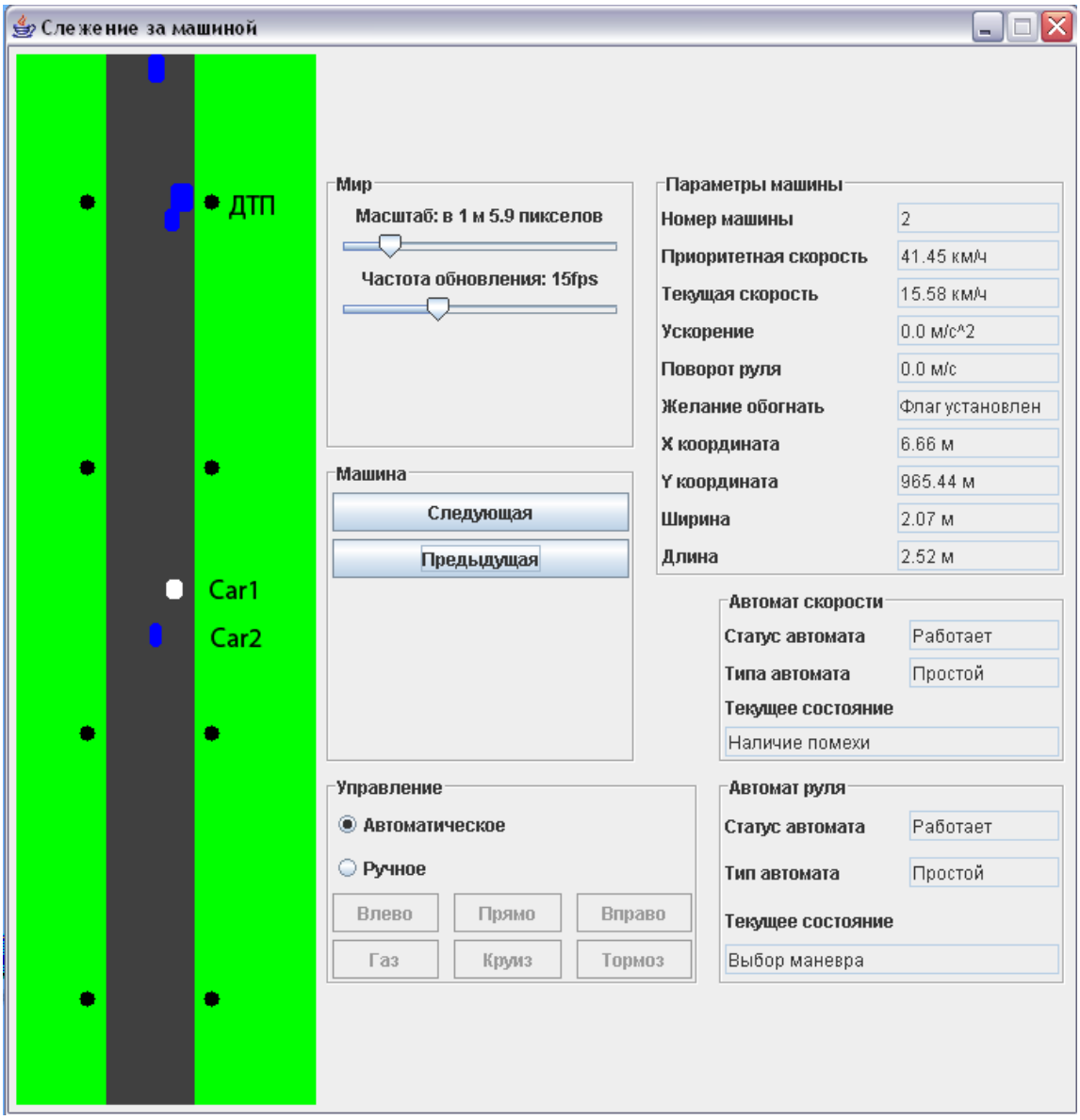

Рис. 17. Объезд препятствия. Этап 1

На следующем этапе (рис.18) автомобиль *Car1* продолжает медленно приближаться к препятствию, и возникает момент, когда он уже в состоянии перестроиться влево, не создавая никому помехи. Потому автопилот принимает решение о выполнении перестроения влево. Автомобиль *Car2* в это время продолжает движение прямо, так как по-прежнему не имеет помех на своем пути.

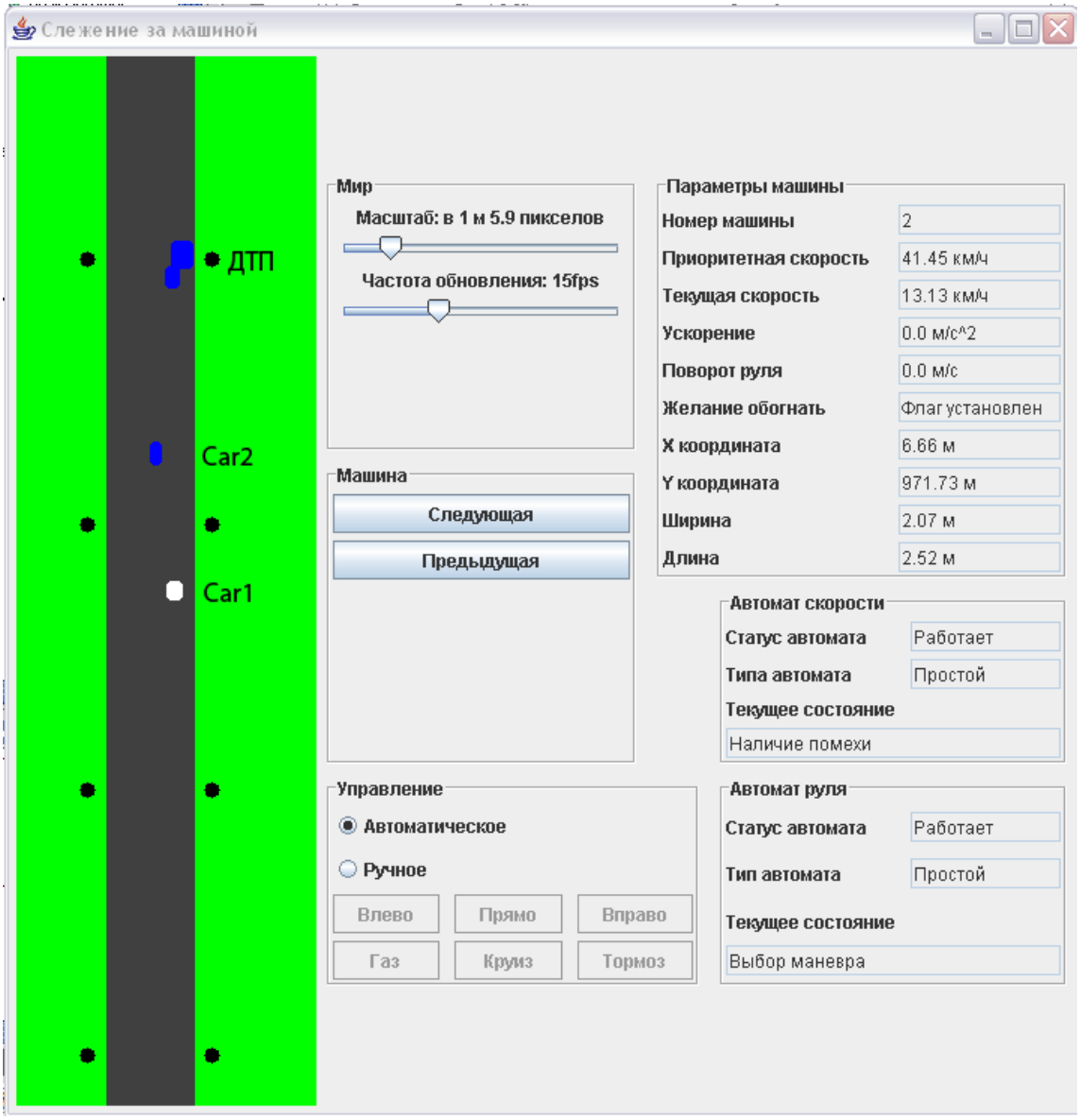

Рис. 18. Объезд препятствия. Этап 2

В итоге автомобиль *Car1* занимает свободный ряд левее ДТП (рис.19) и разгоняется до приоритетной скорости (на рисунке разгон уже выполнен и автомобиль движется с приоритетной скоростью). При этом сзади появился автомобиль *Car3*, которому автомобиль *Car1* не создал помех при перестроении, но стал помехой, когда закончил маневр (скорость автомобиля *Car3* значительно выше скорости автомобиля *Car1*). Автопилот автомобиля *Car3* после получения информации о помехе произвел перестроение влево и таким образом занял свободный ряд. В итоге оба автомобиля движутся в своих рядах, не имея помех для набора или поддержания скорости.

Автомобиль *Car2*, скорость которого была высока с самого начала (и не снижалась), уже уехал далеко вперед.

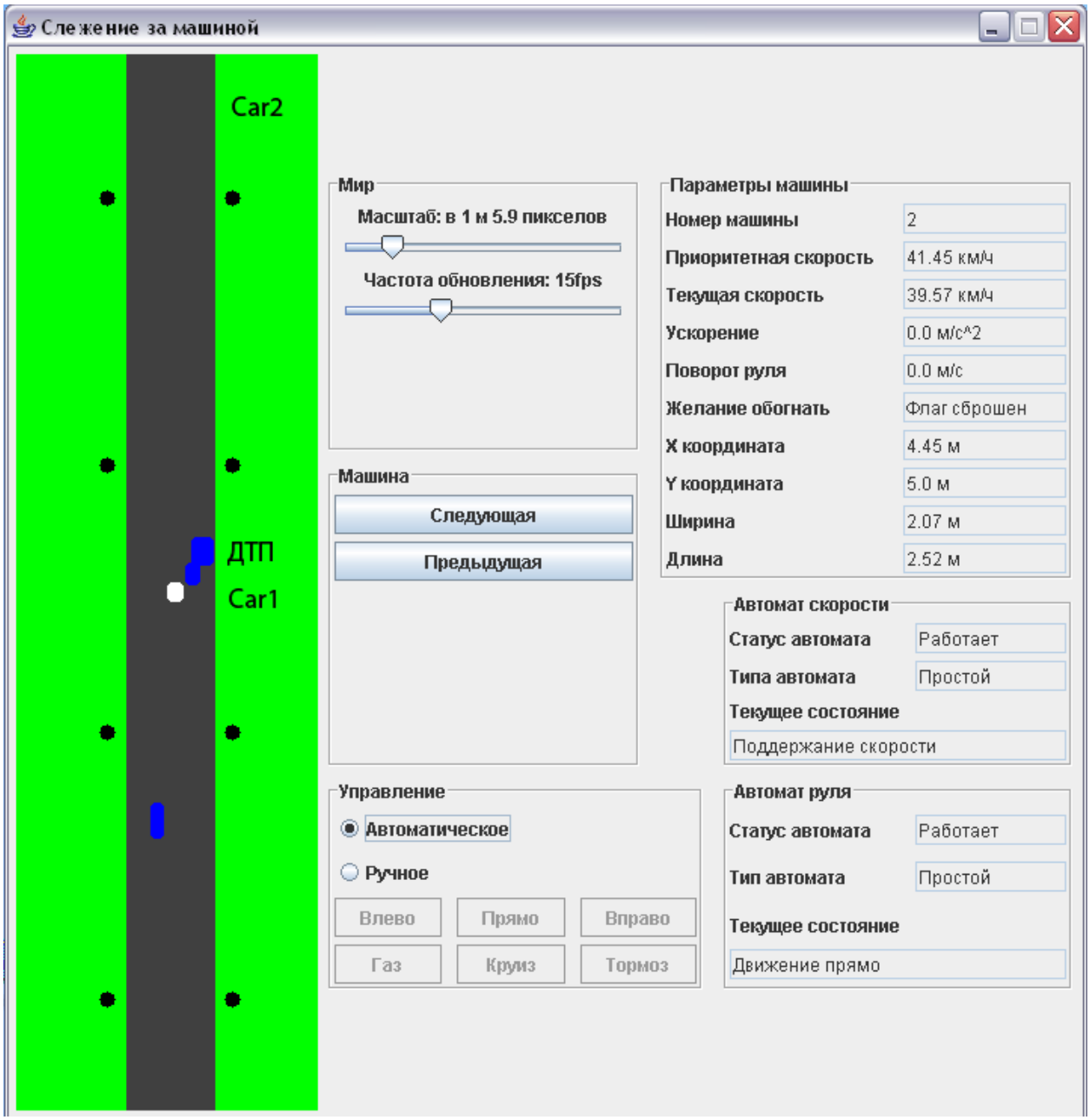

Рис. 19. Объезд препятствия. Этап 3

На последнем рисунке (рис. 20) приведена заключительная стадия объезда препятствия. Автомобили *Car1* и *Car3* проехали место ДТП и либо движутся с приоритетными скоростями, либо набирают их. При появлении возможности оба автомобиля попытаются перестроиться вправо.

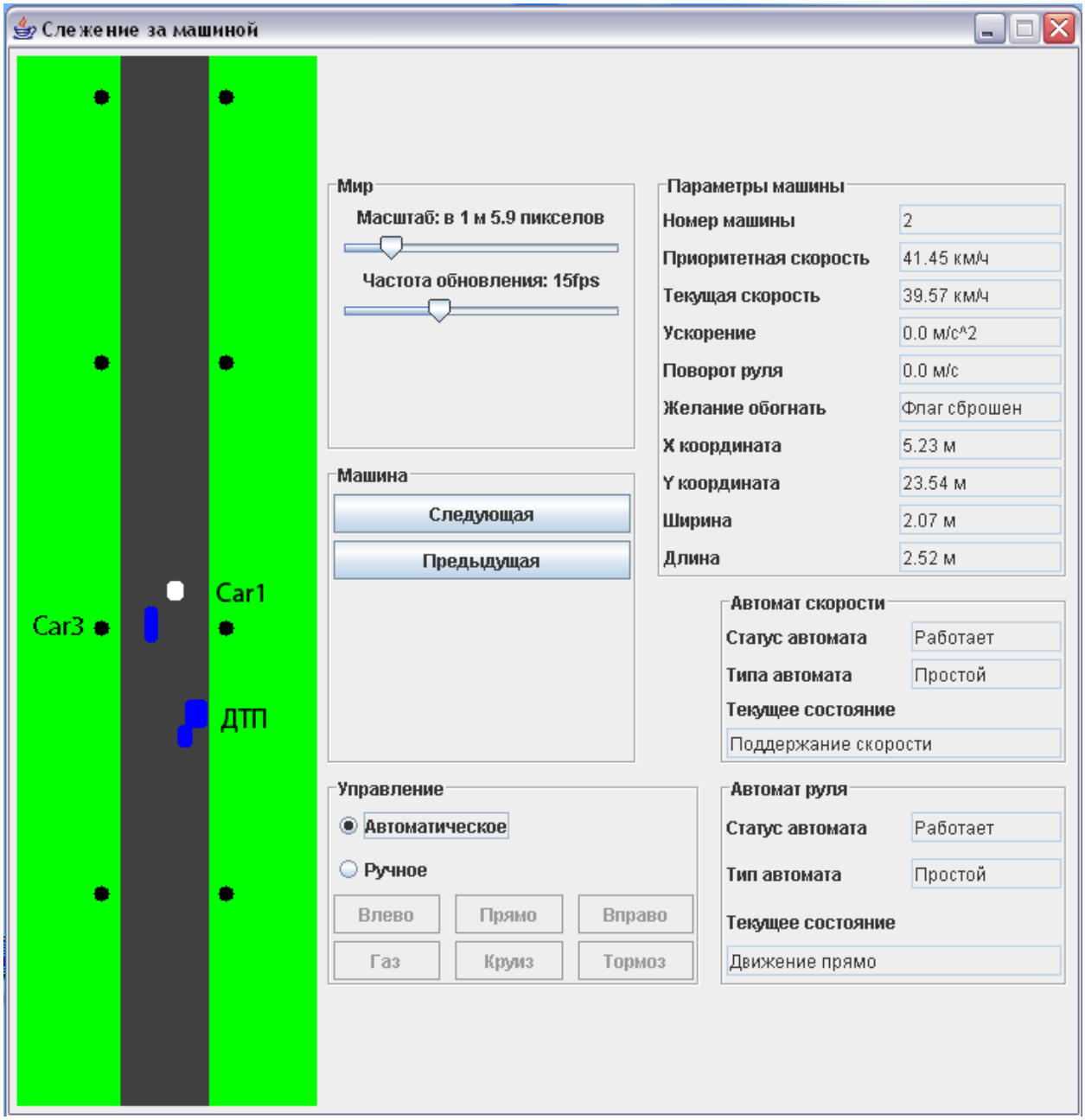

Рис. 20. Препятствие. Этап 4

На основе трех приведенных примеров показаны общие принципы действия автоматов управления автомобилями. Стоит отметить, что в более сложных ситуациях, когда количество автомобилей, которые нужно учитывать, принимая решение, велико, автоматы также ведут себя естественно и логично. Хотя показать и объяснить (как в трех приведенных примерах) все моменты, которые учитываются при принятии решения, весьма затруднительно и очень громоздко.

Таким образом, можно утверждать, что созданная система способна моделировать движение реальных автомобилей на шоссе. На данный момент достигнута достаточная степень реалистичности. Стоит отметить, что при использовании большего количества статистических данных (характеристики автомобилей, сведения о средних скоростях и т.д.) можно еще больше увеличить реалистичность для более точного моделирования.

5.4. Светофор

Рассматривается участок шоссе, на котором расположен пешеходный светофор (рис. 21). Один раз в минуту этот светофор приостанавливает движение автомобилей на 25 с (красный сигнал). Перед этим на семь секунд загорается желтый сигнал, который извещает о том, что вскоре движение будет приостановлено. На рис. 21 показан момент, когда зеленый сигнал сменился на желтый. Выделенный автомобиль (*Car1*), заметив это, начинает снижать скорость (при этом он воспринимает красный или желтый сигналы светофора как помеху). Для этого его автопилот устанавливает отрицательное значение ускорения (поле выделено на рисунке). При этом автомобиль начинает замедление для того, чтобы остановиться у «стоп-линии». Автомобили, движущиеся за автомобилем *Car1*, действуют аналогично.

Автомобили *Car2* и движущийся за ним следом в момент, когда загорелся желтый сигнал светофора, были близко к «стоп-линии» и двигались с большой скоростью. Потому они, не успевая затормозить, пересекают «стоп-линию» на желтый сигнал светофора.

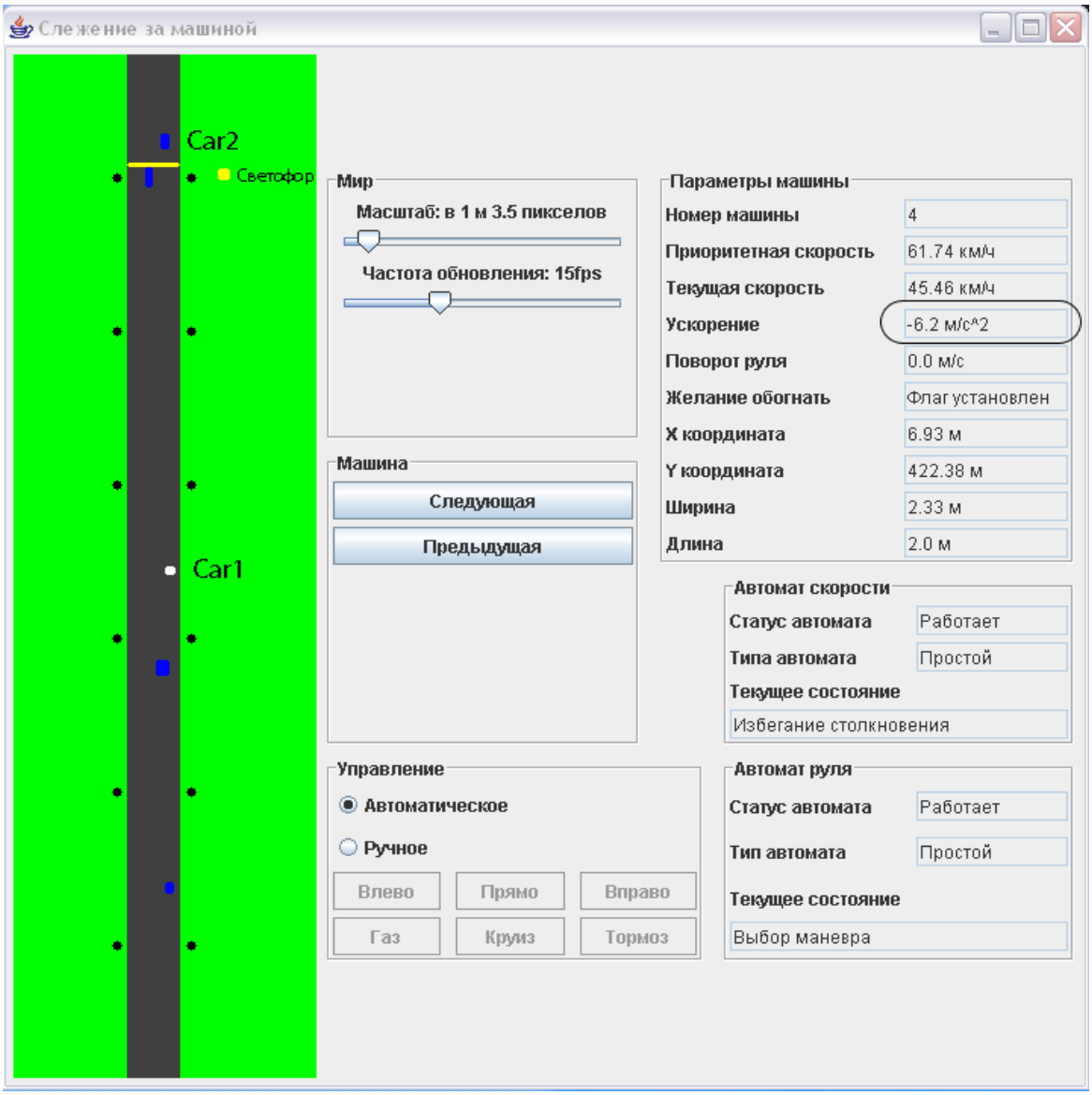

Рис. 21. Светофор. Этап 1

На следующем рисунке (рис. 22) изображен момент, когда желтый сигнал светофора сменился красным. Автомобиль *Car1* почти остановился и с очень низкой скоростью подъезжает к «стоп-линии». Остальные автомобили действуют аналогично, но их скорости выше, так как они находятся дальше от светофора. Автомобили, «проскочившие» на желтый сигнал продолжили движение и уже уехали их поля видимости.

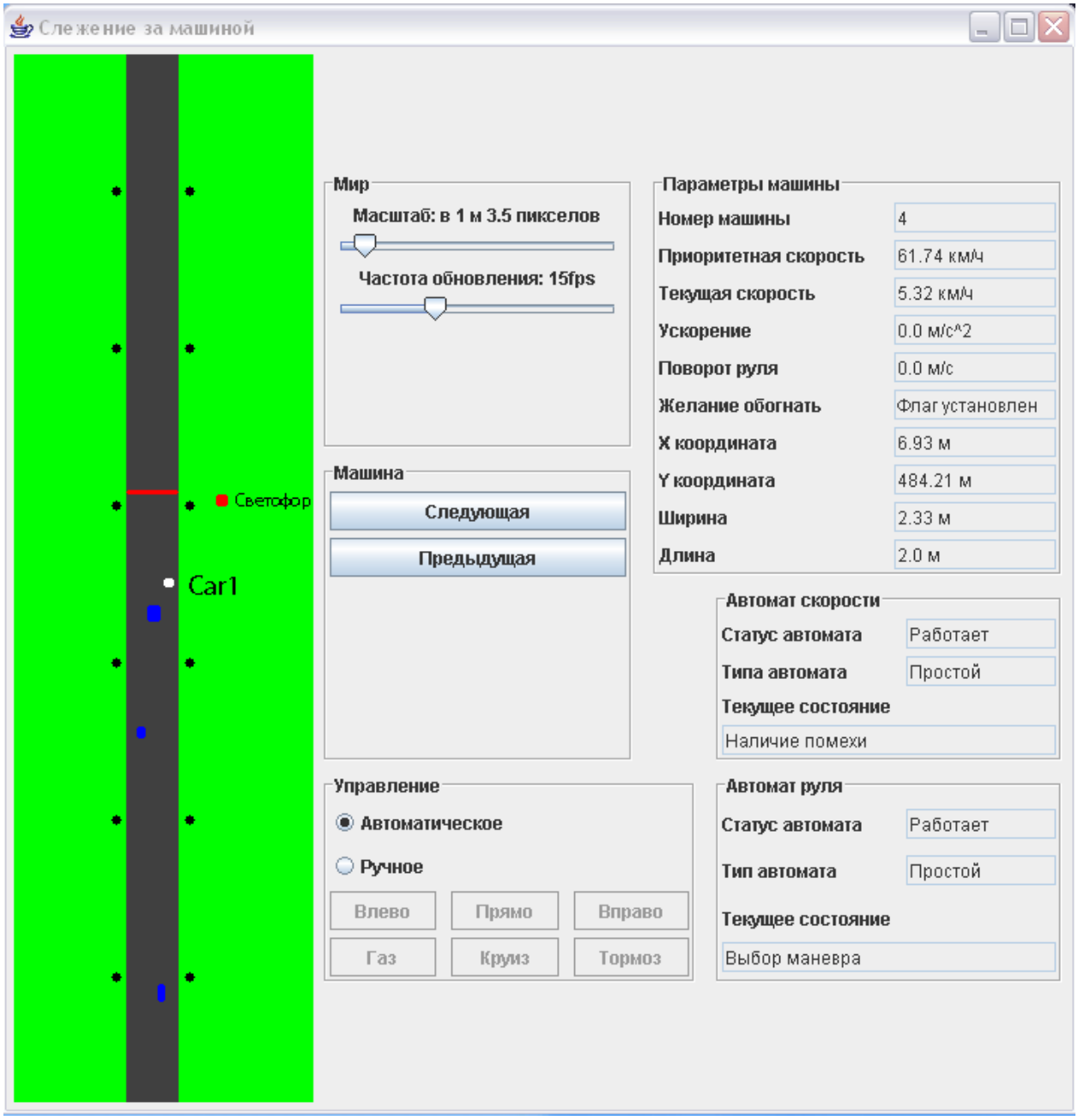

Рис. 22. Светофор. Этап 2

На рис. 23 показан последний фрагмент цикла светофора. Красный сигнал сменился зеленым. Автомобиль *Car1* уже успел разогнаться до своей приоритетной скорости и удаляется от светофора. Остальные автомобили ускоряются и проезжают светофор.

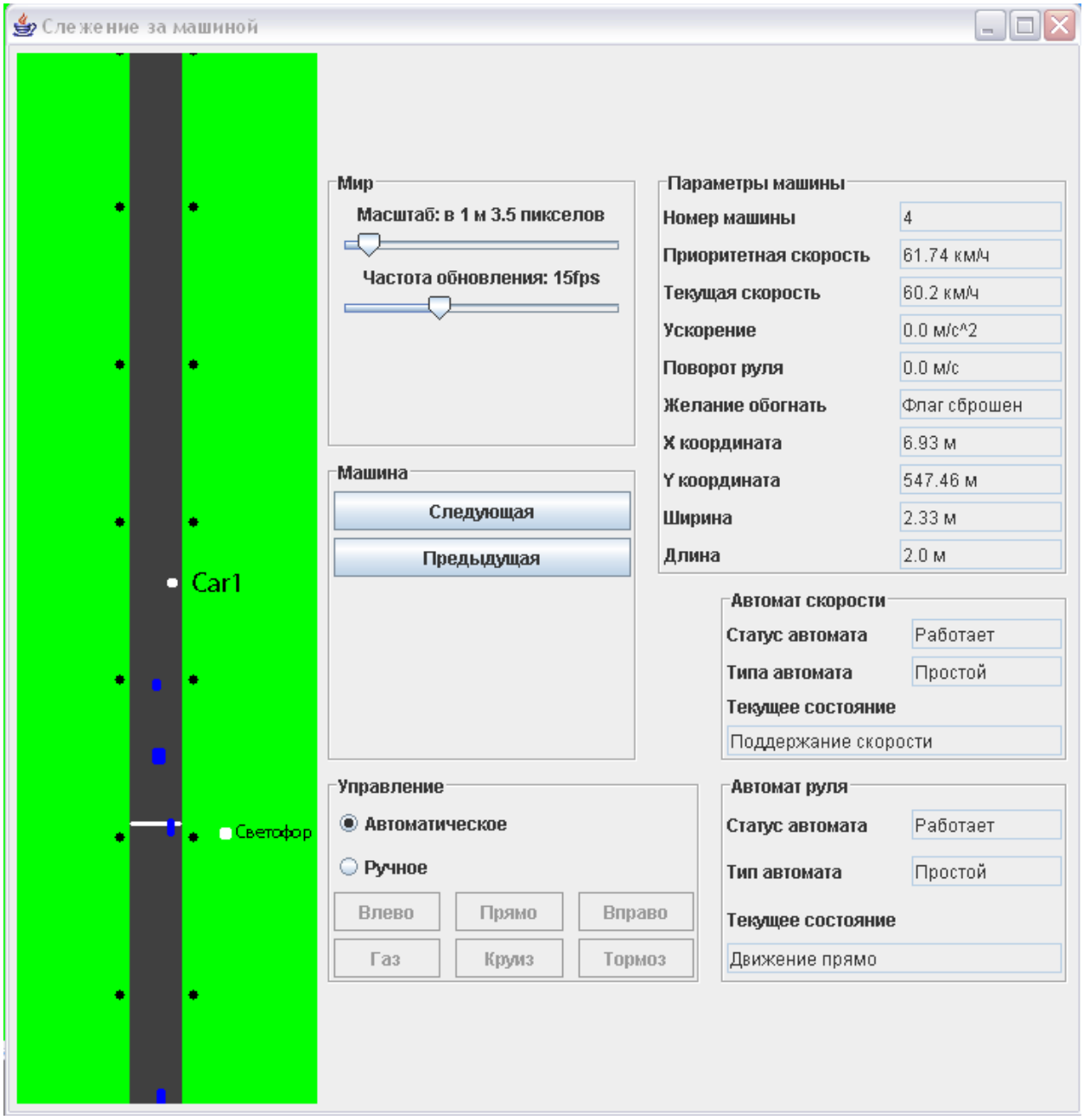

Рис. 23. Светофор. Этап 3

5.5. Выводы по главе 5

На основе проведенного тестирования, часть которого рассмотрена в разд. 5.1 – 5.4., можно сделать вывод, что реализованная модель достаточно точно и подробно имитирует поведение реальных автомобилей на шоссе. Таким образом, данные, полученные на основе моделирования с помощью рассмотренной методики, действительно могут быть использованы для прогнозирования дорожных ситуаций.

Бесспорно, реалистичность может быть увеличена, если учитывать большее количество факторов, влияющих на дорожное движение, таких например, как различное нервное состояние водителей, небольшие препятствия на дороге (ямы, выбоины) и т.д. Однако в данной работе стояла цель продемонстрировать применимость рассмотренного подхода, а не внедрить его. Потому вторичные факторы не учитывались.

Если данная система будет разрабатываться далее, то к факторам, уже учтенным при моделировании, будут добавлены другие факторы, которые также влияют на поведение автомобилей и водителей, но влияние которых не столь явно и сильно.

6. Моделирование

Для проверки системы в рабочем режиме был смоделирован участок дороги, образцом для которого послужил участок Пулковского шоссе города Санкт-Петербурга. Основные данные смоделированного участка:

- Длина – 1 км.
- Ширина – 12 м. (четыре полосы для движения).
- Дорожные условия – идеальные.
- Плотность потока – средняя. На рассматриваемом участке длиной 1 км находится в один момент времени от 30 до 40 автомобилей.
- Автомобили – разнообразные. При моделировании использовались различные транспортные средства: легковые автомобили с разными характеристиками, грузовые автомобили, мотоциклы.

Некоторые из выводов в данном разделе будут делаться для эталонного автомобиля, который обладает заданными средними характеристиками (одинаковыми во всех вариантах моделирования). Например, приоритетная скорость эталонного автомобиля равна 80 км/ч, динамика 13 с (время разгона до 100 км/ч).

6.1. Без препятствий

В первом случае участок дороги был пустым, на нем не было никаких препятствий.

Средняя скорость автомобилей составляла 82% – 100% от их приоритетных скоростей. Иначе говоря, автомобили ехали почти с той скоростью, с которой хотели. Это вполне логично, так как никаких препятствий не было, а плотность потока была сравнительно невысока для рассматриваемого участка.

Пропускная способность составила 2688 автомобилей в час. Пока эту цифру сравнивать не с чем.

Дорожно-транспортных происшествий (ДТП) не зафиксировано. Это также логично, так как автомобили маневрировали лишь во время обгонов.

Эталонный автомобиль проходил участок за 44 – 49 с.

6.2. Светофор

В данной модели рассматривался тот же участок шоссе, но на его середине (500 м) был установлен пешеходный светофор со следующим циклом:

- красный сигнал – 20 с;
- желтый сигнал – 5 с;

- зеленый сигнал – 25 с.

На данном участке были получены следующие результаты

Средняя скорость автомобилей составляла 43% – 97% от их приоритетных скоростей. Минимальные значения были у автомобилей, которым пришлось стоять почти целый цикл на светофоре. Так как поток не плотный, то не было автомобилей, которым пришлось бы стоять более одного цикла. Максимальные значения средней скорости были у автомобилей, которые подъезжали к светофору, когда загорался зеленый, и не теряли времени.

Пропускная способность составила 2581 автомобиль в час. Это лишь на 4% меньше, чем в случае участка без светофора. Произошло это потому, что почти все автомобили, которые въехали на данный участок, проехали его с небольшой задержкой, вызванной наличием светофора. Иначе говоря, не один автомобиль не потерял более одной минуты.

ДТП не зафиксировано. Стоит отметить, что когда в рассмотренный поток был добавлен динамичный автомобиль с очень высоким ускорением при торможении, было зафиксировано ДТП. Автомобиль, двигавшийся за ним, не успел затормозить, когда динамичный автомобиль начал торможение на светофоре.

Эталонный автомобиль проходил участок за 44 – 85 с.

6.3. Пост ГАИ

В данной модели рассматривается тот же участок шоссе. Однако на его середине (500м) установлен пост ГАИ. В данном месте происходит сужение проезжей части до двух полос (исчезают крайние). Были получены следующие результаты:

Средняя скорость автомобилей составляла 16% – 94% от их приоритетных скоростей. Минимальные значения были у автомобилей, которым пришлось стоять в небольшом заторе, который образовался, перед сужением. Так как поток не плотный, образовавшийся затор был не велик и не увеличивался со временем. Максимальные значения средней скорости были у автомобилей, которые проехали участок с сужением до образования затора.

Пропускная способность составила 2260 автомобилей в час. Это на 15% меньше, чем в случае участка без препятствий. Стоит отметить, что когда при сохранении всех остальных данных, плотность была увеличена на 15 – 20% , пропускная способность почти не изменилась, но затор был заметно больше и средние скорости упали существенно. Таким образом, можно утверждать, что пропускная способность около 2200 – 2300 автомобилей является максимальной для данного участка.

Зафиксировано одно ДТП. Автомобиль не смог перестроиться и не успел затормозить перед сужением. Стоит отметить, что данный автомобиль обладал очень высокими динамическими характеристиками и ехал почти на их пределе. Водите аккуратно.

Эталонный автомобиль проходил участок за 44 – 318 с.

6.4. Выводы по главе 6

В данной главе описаны конкретные эксперименты, которые были проведены с помощью разработанной системы. В этих экспериментах моделируются различные участки дорог с препятствиями на них. В результате моделирования получены конкретные оценки прогнозируемых величин, как для потока автомобилей в целом, так и для конкретно выбранного эталонного автомобиля. Показано, что полученные данные логичны и могут использоваться на практике.

Заключение

В рамках данной работы была рассмотрена проблема создания системы моделирования дорожного движения. Главным требованием являлось достижение высокой реалистичности при имитации движения.

На основе анализа приведенного в первой главе были выявлены недостатки существующих методик. Кроме того, были сделаны выводы относительно конкретных целей, которые преследует данный метод. При этом внимание было сфокусировано на вопросах безопасности и пропускной способности магистралей, проходящих вне населенных пунктов.

Во второй главе была описана предполагаемая структура программы. Были разъяснены средства, с помощью которых предполагалось решать каждую из локальных задач, которые вместе и составляют систему. Описано разделение функций между модулями программы. Все идеи, изложенные в данной главе, были реализованы.

Третья глава полностью посвящена рассмотрению деталей реализации системы. Описаны все ключевые классы программы, принципы их работы, функции, которые они выполняют, а также их взаимодействие. Подробно рассмотрены основные классы данных. В этот раздел не вошло описание классов, принимающих решения – классов, реализующих искусственный интеллект.

Четвертая глава посвящена описанию классов, управляющих автомобилями. Сначала описываются общие положения, которым должны удовлетворять эти классы. Затем описывается предок автоматных классов, что дает общее понимание принципов принятия решений и работы управляющих классов. Далее следует подробное описание двух типов классов, реализующий искусственный интеллект, который обеспечивает управление каждым автомобилем на шоссе. Приведены конкретные примеры и схемы классов данного типа. Описаны принципы и объяснена логика, в соответствие с которой принимаются решения.

В последних двух главах описывается функционирование созданной модели, и проводятся эксперименты, которые и были целью работы.

В пятой главе рассмотрены типичные ситуации, возникающие на дороге. Продемонстрировано правильное функционирование модели, а также показано, что элементы управления принимают правильные и адекватные решения. Таким образом,

было показано, что созданная модель достаточно точно имитирует движение автомобилей на шоссе, а автоматы управления способны моделировать действия реальных водителей.

Данные выводы позволили провести эксперименты, описанные в шестой главе. В этих экспериментах моделируются различные участки дорог с препятствиями на них. В результате моделирования получены конкретные оценки прогнозируемых величин, как для потока автомобилей в целом, так и для конкретно выбранного эталонного автомобиля.

Авторы предполагают, что разработанная система если и не может в данном виде использоваться на практике, то, очевидно, показывает целесообразность применения моделирования для прогнозирования дорожной обстановки. Доказано, что системы, реализованные на принципах, использованных в данном проекте, способны точно и адекватно имитировать реальное движение. На их основе могут быть сделаны достоверные выводы относительно существующих дорог и автомобилей.

В заключение, отметим, что данная модель имеет много направлений для дальнейшего развития и модификации.

Литература

1. *Шалыто* А. А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. http://is.ifmo.ru/books/switch/1
2. *Ла Мот А., Ратклифф Д., Семинаторе М.* Секреты программирования игр. СПб.: Питер, 1995.
3. *Туккель Н. И., Шалыто А. А.* Система управления танком для игры *Robocode.* Объектно-ориентированное программирование с явным выделением состояний. Проектная документация.. http://is.ifmo.ru/projects/tanks/
4. *Наумов А. С., Шалыто А. А.* Система управления лифтом. Проектная документация. http://is.ifmo.ru/projects/elevator/
5. *Туккель Н. И., Шалыто А. А.* SWITCH-технология – автоматный подход к созданию программного обеспечения «реактивных» систем //Программирование. 2001. № 5, с.45–62. http://is.ifmo.ru/works/switch/1.
6. *Клименко В. В., Хазановский А. Ю., Шалыто А. А.* Система управления автомобилем на шоссе. http://is.ifmo.ru/projects/carpilot/
7. *Бабков В. Ф., Андреев О. В.* Проектирование автомобильных дорог: учебник для вузов. М.: Транспорт, 1987.