Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

Е. Ю.Васильева, А. С. Никитин, М. Ю. Чураков

Моделирование проезда регулируемых и нерегулируемых перекрёстков равнозначных дорог

Объектно-ориентированное программирование с явным выделением состояний

Проектная документация

Проект создан в рамках
«Движения за открытую проектную документацию»
http://is.ifmo.ru

Санкт-Петербург
2008

Оглавление

Введение

В существующих на данный момент правилах дорожного движения (ПДД) одним из наиболее сложных моментов является порядок разъезда на регулируемых и нерегулируемых перекрёстках равнозначных дорог. В материалах для подготовки к экзамену в ГИБДД обычно проводится детальный разбор лишь некоторых случаев. Однако, ситуации, которые возникают на практике, зачастую оказываются значительно сложнее, и требуют от водителя быстрого анализа дорожной обстановки и принятия решения. В таких случаях у водителя нет времени для того, чтобы перебрать в памяти все рассмотренные на уроках примеры и вспомнить, как необходимо действовать в конкретной ситуации. Для нормального управления автомобилем в сознании водителя должна быть чёткая схема действий для предотвращения аварийной ситуации. Задача обучения состоит как раз в том, чтобы сформировать в сознании учащегося ясный алгоритм принятия решения в любой дорожной обстановке и довести навыки управления транспортным средством до автоматизма. Основным правилом разъезда на нерегулируемых перекрестках является правило «правой руки», предписывающее уступить дорогу, если есть помеха справа, однако на практике возможна неверная трактовка этого правила, которая приведет к аварии.

Данный проект разработан с целью наглядной демонстрации правил проезда регулируемых и нерегулируемых перекрёстков равнозначных дорог. Пользователю достаточно лишь задать расположение автомобилей и направления их движения, чтобы увидеть, как в соответствии с ПДД должны разъезжаться водители. Автоматный подход, применяемый авторами, позволяет эффективно реализовать механизм принятия решений по выбору траектории и порядку начала движений.

Проект выполнен на языке программирования *Java* в среде *Eclipse* с помощью инструментального средства *UniMod*.

Настоящая работа является развитием проекта http://is.ifmo.ru/unimod-projects/model-traversing/. В ней наряду с нерегулируемым перекрестком равнозначных дорог рассматривается также и перекресток таких дорог, регулируемый светофором.

1. Постановка задачи

Авторами ставилась задача моделирования проезда регулируемых и нерегулируемых перекрёстков равнозначных дорог с учётом всех имеющихся правил, представленных в соответствующих разделах ПДД. Кроме того, требовалась реалистичность визуальной составляющей – автомобили должны двигаться по естественным траекториям, не задевая друг друга.

Как и в любой другой модели, в данном проекте используются некоторые допущения:

- дороги имеют по одной полосе для движения в каждом направлении;
- все автомобили движутся с одинаковой скоростью;
- разворот по малой траектории (без выезда на середину перекрёстка) невозможен.

1.1. Правила разъезда

Приведём выдержки из ПДД, которые реализуются в данном проекте:

> 13.11. На перекрестке равнозначных дорог водитель безрельсового транспортного средства обязан уступить дорогу транспортным средствам, приближающимся справа.
>
> 13.12. При повороте налево или развороте водитель безрельсового транспортного средства обязан уступить дорогу транспортным средствам, движущимся по равнозначной дороге со встречного направления прямо или направо.

Существуют ситуации, когда все участники движения, формально действуя по правилам, должны оставаться на месте, потому что для каждого из них присутствует помеха в виде другого автомобиля, имеющего преимущество. В этом случае водители транспортных средств должны договориться между собой о том, кто первым проедет перекрёсток.

Рассмотрим случай, изображённый на рис.1. Автомобили, приближающиеся снизу и справа, движутся в прямом направлении, а сверху и слева – налево и направо, соответственно. Порядок разъезда в этом случае определяется однозначно – лишь зелёная машина, находящаяся слева, не имеет помех, поэтому первой проезжает перекрёсток. Оранжевая машина не движется, так как зелёная для неё является помехой справа. Жёлтая машина не может повернуть налево и пропускает оранжевую, движущуюся со встречного направления. Синяя машина аналогично дожидается, пока исчезнет помеха справа.

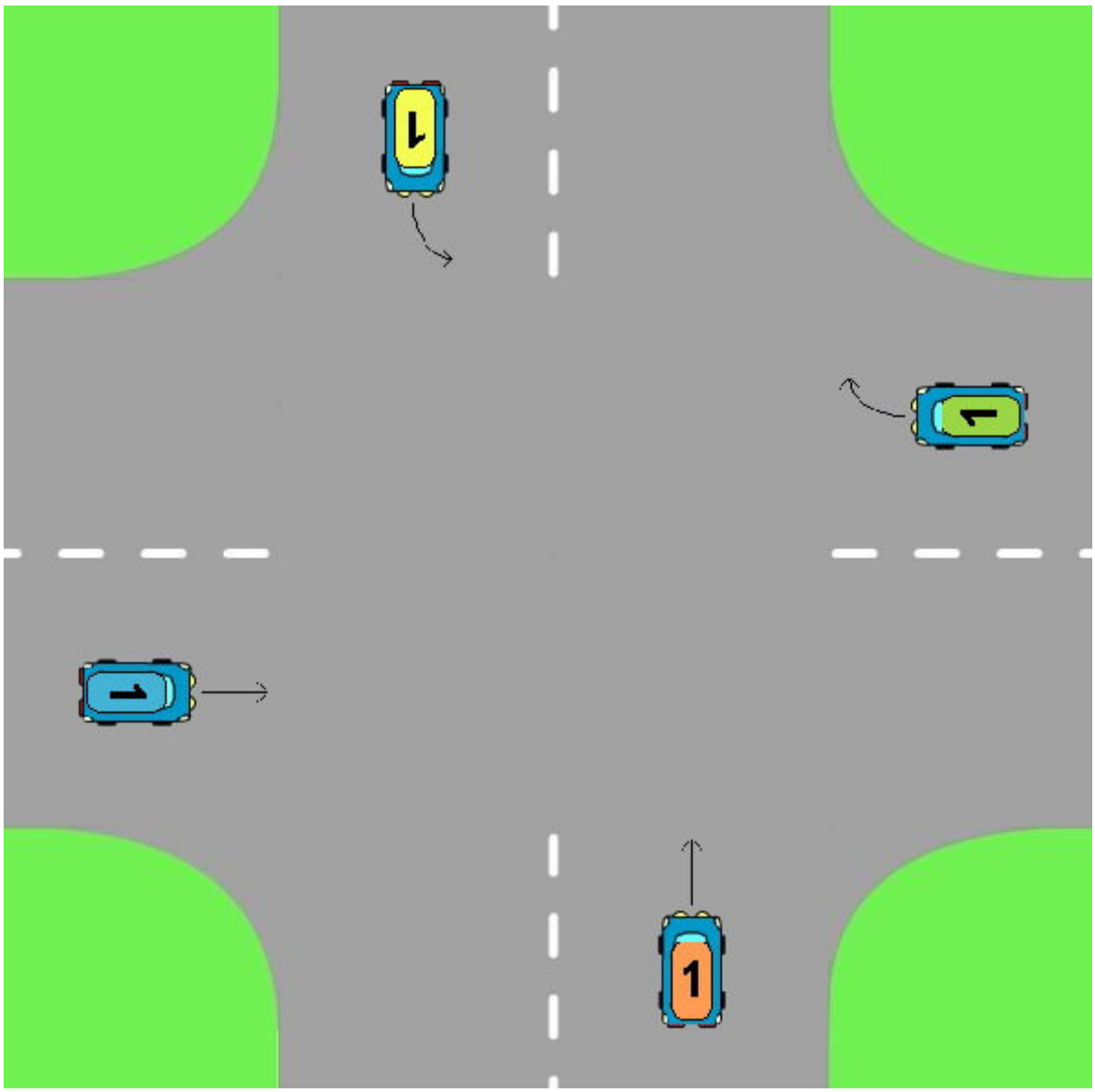

Рис. 1. Пример разъезда: направления движений

Одновременно с тем, как зелёная машина проезжает перекрёсток, жёлтая выезжает на середину перекрёстка, чтобы, пропустив оранжевую, закончить манёвр и покинуть перекрёсток. Далее для оранжевой машины исчезает помеха справа, и она может начать движение.

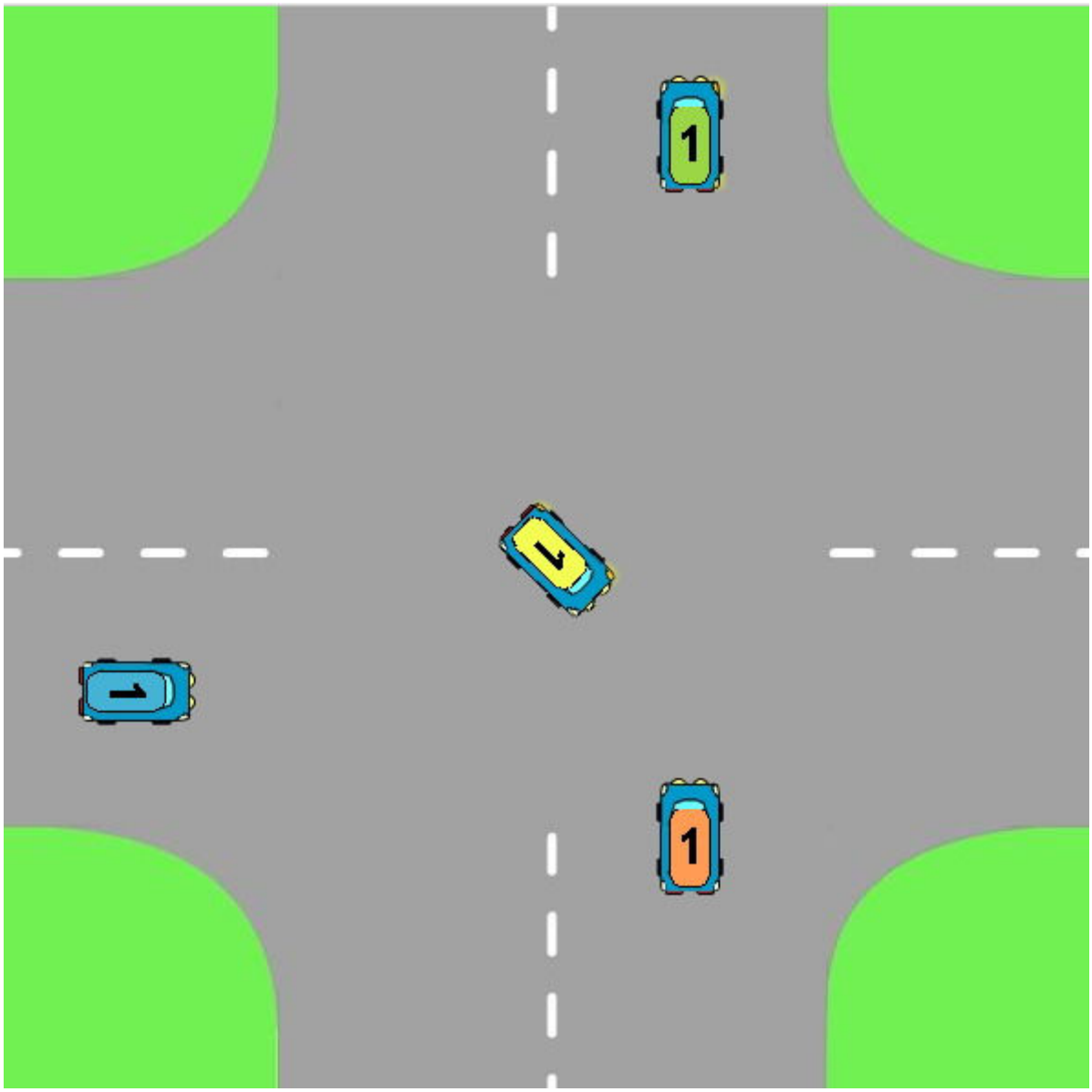

Рис. 2. Пример разъезда: зелёная машина поворачивает направо,
желтая – выезжает на середину перекрёстка, оранжевая – начинает двигаться

После того как оранжевая машина больше не создаёт помех, жёлтая и оранжевая машины могут продолжить движение. Однако их траектории пересекаются, поэтому одновременно двигаться они не могут. В этом случае жёлтая машина в соответствии с пунктом 13.11 ПДД должна уступить дорогу синей, так как для неё она является приближающейся справа.

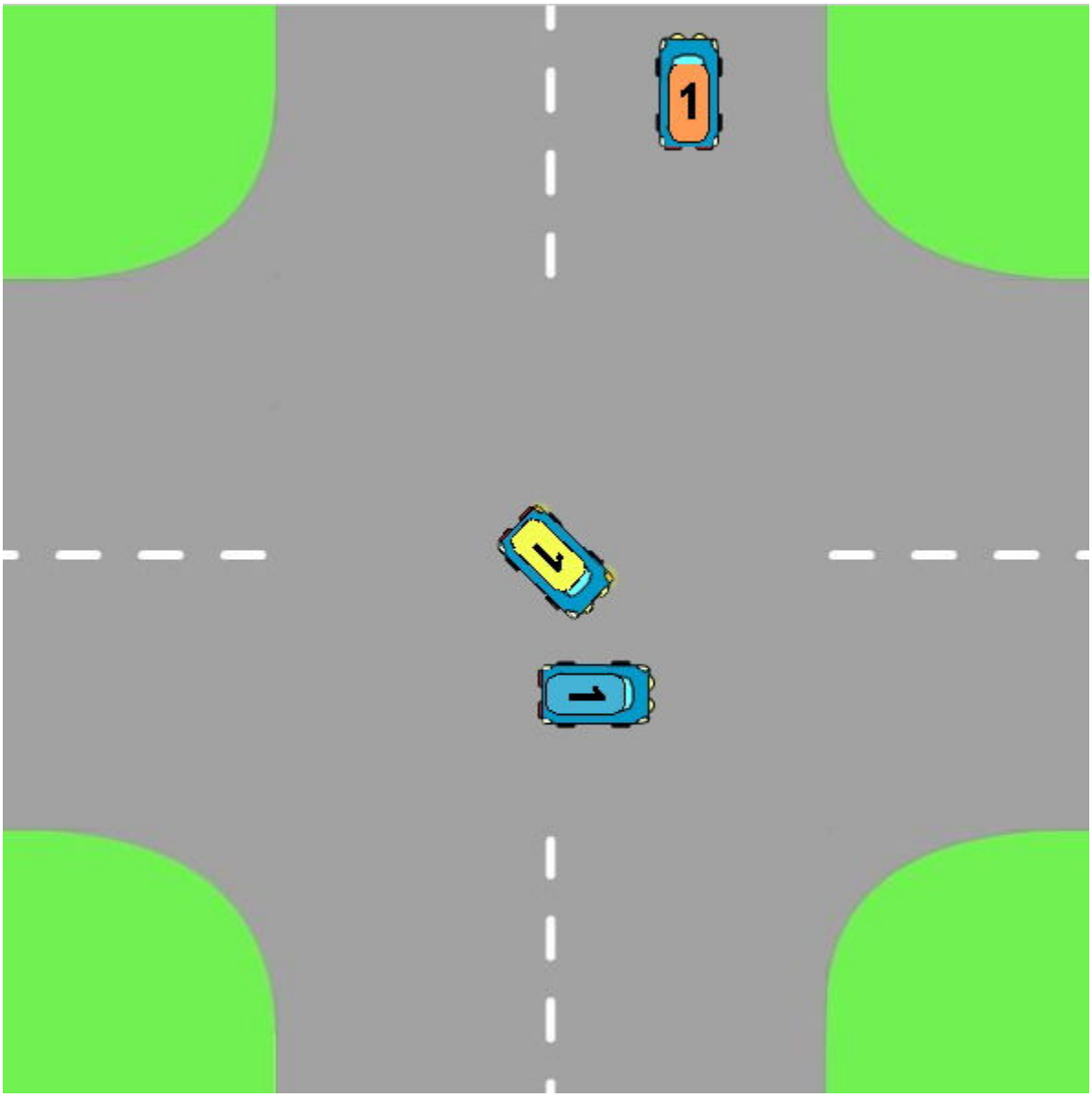

Рис. 3. Пример разъезда: жёлтая машина пропускает синюю

После этого жёлтая машина завершает поворот налево, не имея никаких помех, потому что все остальные участники движения уже успели покинуть перекрёсток.

1.2. Внешний вид программы и пользовательский интерфейс

После запуска программы (рис. 4) необходимо задать конфигурацию перекрестка (задать направления движения машин и механизмы их управления) – один из четырех автоматов:

1. Автомат, реализующий поведение водителя, соответствующее ПДД.
2. Автомат, подчиняющийся ПДД, но не включающий сигналы поворота.
3. Автомат, который не соблюдает ПДД.
4. Автомат, который моделирует движение автомобиля при регулируемом перекрестке.

Для того чтобы установить машину, движущуюся с одного из четырёх направлений, достаточно щёлкнуть на сером прямоугольнике левой кнопкой мыши. Для смены автомата – необходимо щёлкнуть ещё раз. Для изменения направления движения требуется щёлкнуть правой кнопкой мыши. Для того чтобы поставить светофор следует щелкнуть по серому квадрату на тротуаре, щелчок правой кнопкой приведет к смене цвета светофора.

Также можно указать такие дополнительные параметры отображения, как траектории и зоны, которые подсвечиваются цветом, соответствующим машине, которая её занимает.

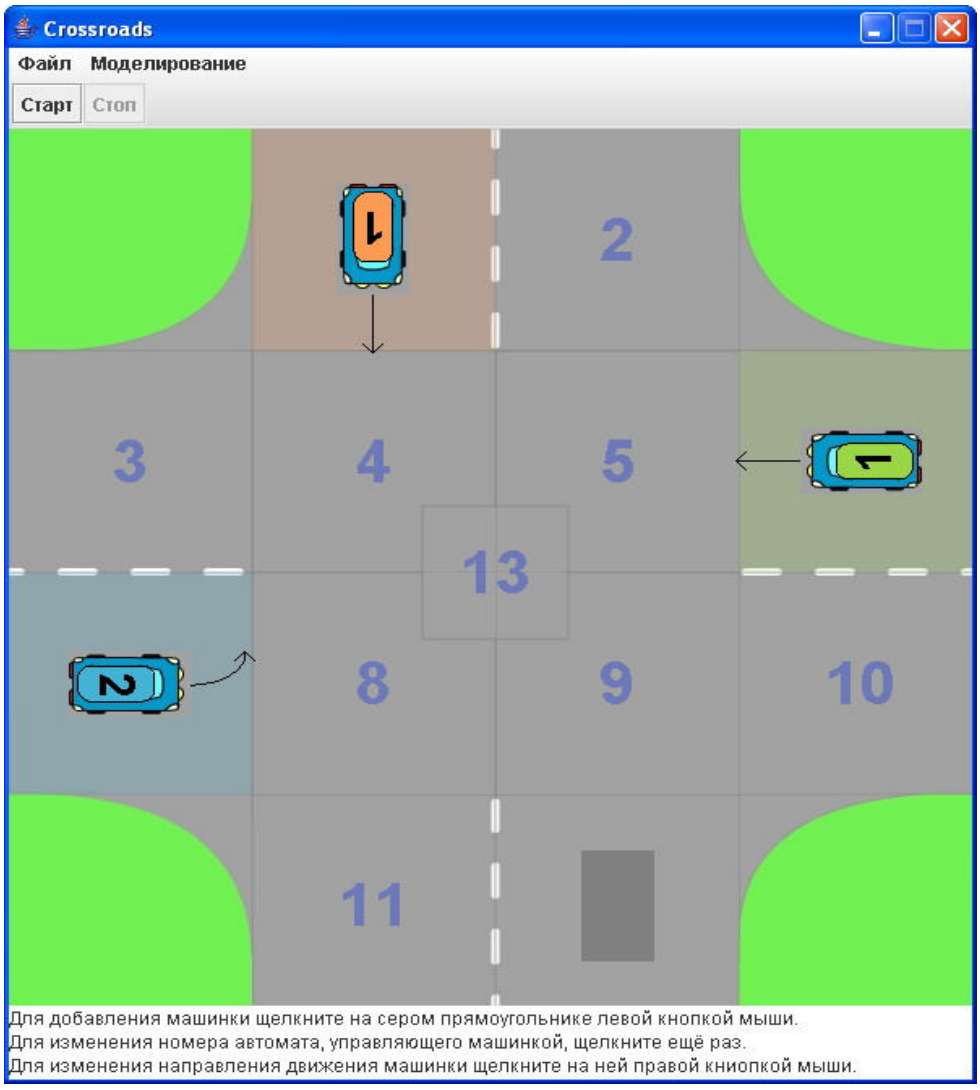

Рис. 4. Окно приложения

Далее всё управление осуществляется кнопками *Старт/Пауза/Продолжить* и *Стоп*.

После завершения моделирования можно задать другой перекрёсток, для того чтобы посмотреть, как разъедутся автомобили в другом случае. При этом можно сохранять и загружать конфигурации перекрёстков из файла.

2. Проектирование

2.1. Диаграмма классов

В проекте используются две *Unimod*-модели. Одна для управления интерфейсом (`core.unimod`), а другая для принятия решений о проезде перекрёстка (`crossroad.unimod`). Для их связи используются дополнительные классы (рис. 5), написанные на языке *Java*, которые обеспечивают всю низкоуровневую работу с данными, отвечающими за внутреннее представление информации о перекрёстке и автомобилях.

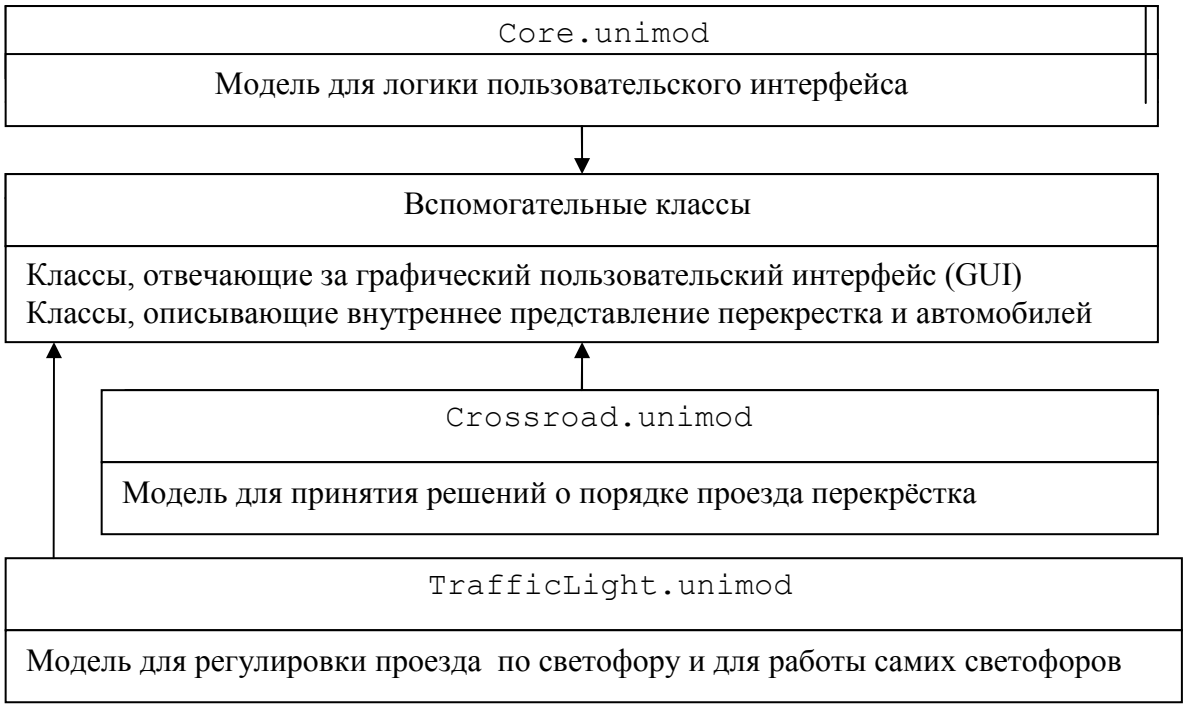

Рис.5. Общая схема классов

При запуске программа запускается из автомата *A1*, находящегося в `core.unimod`. Автоматы же из `crossroad.unimod` предварительно скомпилированы и динамически подключаются в процессе работы приложения.

2.2. Ядро UniMod

Как уже было сказано, в проекте используются две *Unimod*-модели. Одна для принятия решений о проезде перекрёстка (`crossroad.unimod`), а другая – для управления пользовательским интерфейсом (`core.unimod`). Остановимся сначала на последней.

На рис. 6 приведена диаграмма связей модели управления пользовательским интерфейсом `core.unimod`.

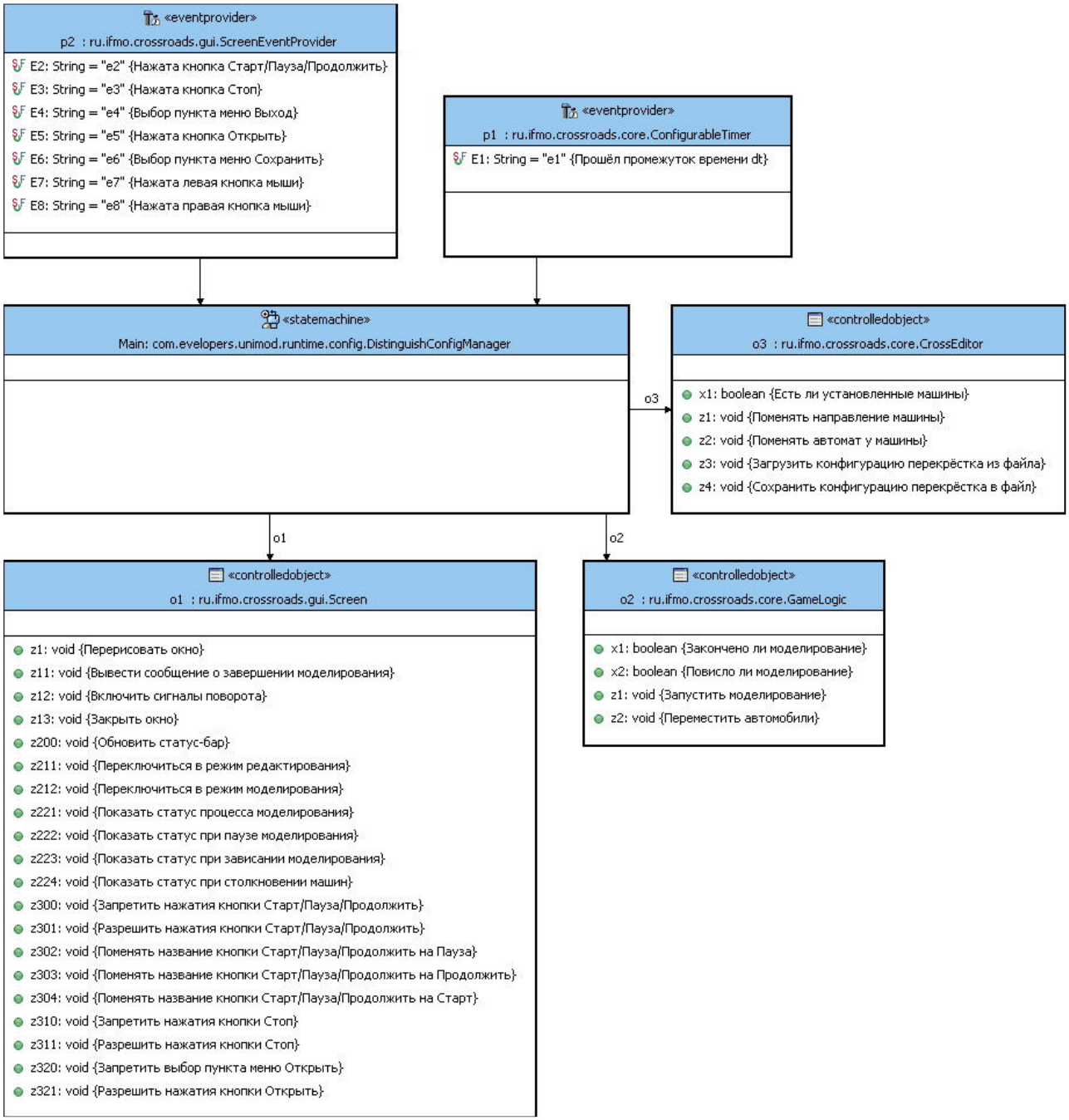

Рис. 6. Диаграмма связей `core.unimod`

Поставщики событий

ConfigurableTimer

Этот поставщик событий во время выполнения программы генерирует событие *e101* через определенный, заранее установленный промежуток времени *dt.* Является расширением стандартного поставщика событий `Timer`, который посылает событие лишь раз в секунду (табл. 1).

Таблица 1

Событие	Описание
E1	Прошёл промежуток времени dt

ScreenEventProvider

Передаёт воздействия пользователя (нажатие кнопок и выбор пунктов меню) автомату *A1* (табл. 2).

Таблица 2

Событие	Описание
E2	Нажата кнопка *Старт/Пауза/Продолжить*
E3	Нажата кнопка *Стоп*
E4	Выбор пункта меню *Выход*
E5	Нажата кнопка *Открыть*
E6	Выбор пункта меню *Сохранить*
E7	Нажата левая кнопка мыши
E8	Нажата правая кнопка мыши

Автомат Main

На рис. 7 приведен граф переходов автомата *Main.*

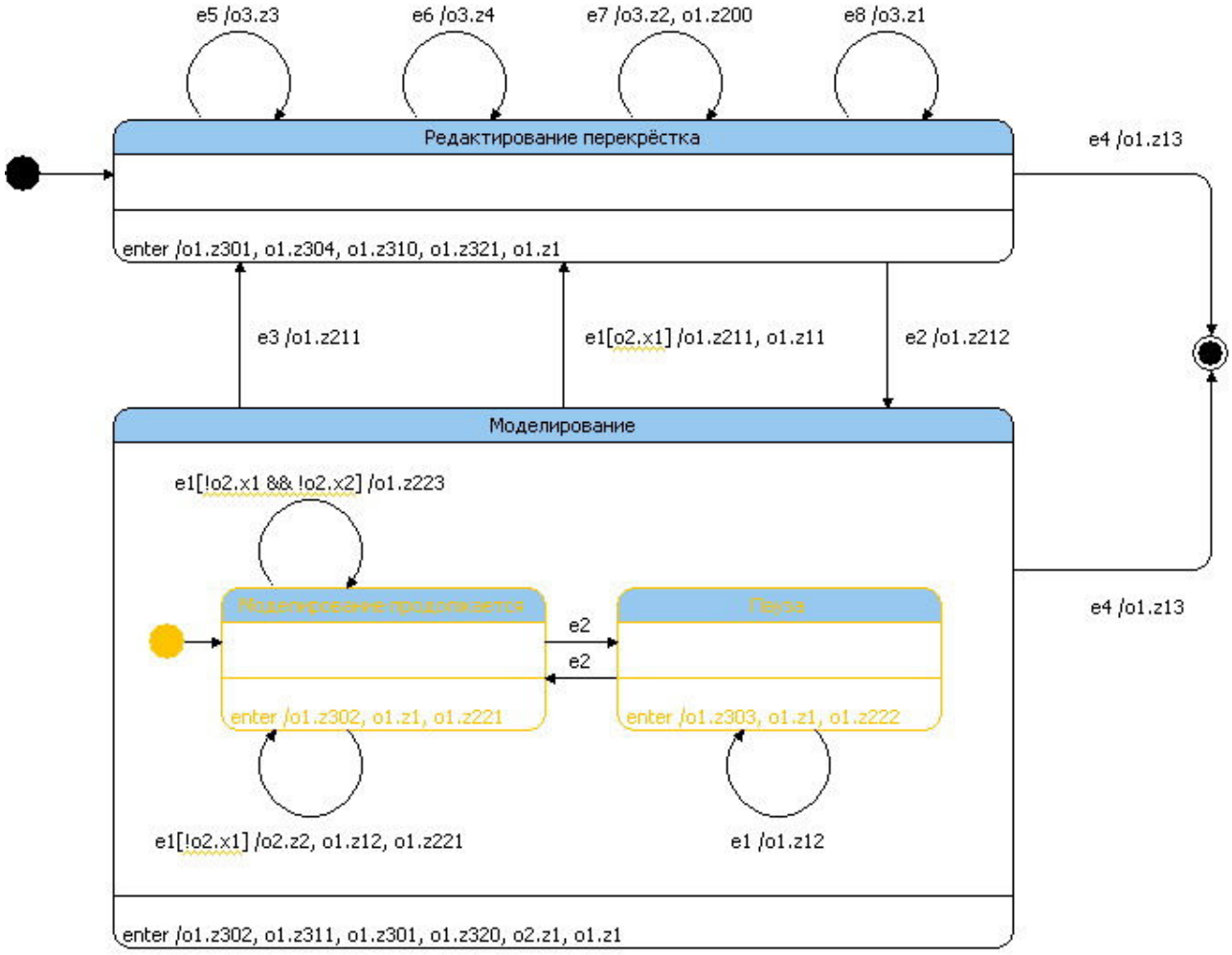

Рис. 7. Автомат `Main`

Управляет графическим интерфейсом пользователя, обеспечивая реагирование на нажатия кнопок и выборы пунктов меню. Вложенный автомат состояние *Моделирование* реализует главный цикл моделирования.

Объекты управления

Screen

Содержит входные и выходные воздействия, связанные с внешним видом окна и его перерисовкой (табл. 3).

Таблица 3

Выходное воздействие	Описание
Z1	Перерисовать окно
Z11	Вывести сообщение о завершении моделирования
Z12	Включить сигналы поворота
Z13	Закрыть окно
Z200	Обновить статус-бар
Z211	Переключиться в режим редактирования
Z212	Переключиться в режим моделирования
Z221	Показать статус процесса моделирования
Z222	Показать статус при паузе моделирования
Z223	Показать статус при зависании моделирования
Z224	Показать статус при столкновении машин
Z300	Запретить нажатия кнопки *Старт/Пауза/Продолжить*
Z301	Разрешить нажатия кнопки *Старт/Пауза/Продолжить*
Z302	Поменять название кнопки *Старт/Пауза/Продолжить* на *Пауза*
Z303	Поменять название кнопки *Старт/Пауза/Продолжить* на *Продолжить*
Z304	Поменять название кнопки *Старт/Пауза/Продолжить* на *Старт*
Z310	Запретить нажатия кнопки *Стоп*
Z311	Разрешить нажатия кнопки *Стоп*
Z320	Запретить выбор пункта меню *Открыть*
Z321	Разрешить нажатия кнопки *Открыть*

GameLogic

Содержит входные и выходные воздействия, связанные с процессом моделирования (табл. 4, 5).

Таблица 4

Входное воздействие	Описание
X1	Закончено ли моделирование
X2	Есть ли прогресс в моделирование (не повисло ли)

Таблица 5

Выходное воздействие	Описание
Z1	Запустить моделирование
Z2	Переместить автомобили

CrossEdit

Содержит входные и выходные воздействия, связанные с процессом редактирования перекрёстка (табл. 6, 7).

Таблица 6

Входное воздействие	Описание
X1	Есть ли установленные машины

Таблица 7

Выходное воздействие	Описание
Z1	Поменять направление машины
Z2	Поменять автомат у машины
Z3	Загрузить конфигурацию перекрёстка из файла
Z4	Сохранить конфигурацию перекрёстка в файл

2.3. Перекрестки UniMod

2.3.1. Схема взаимодействия автоматов

На рис. 8 приведена схема взаимодействия автоматов.

После того, как пользователь нажал кнопку *Старт,* создаётся экземпляр класса `Cross`, который отвечает за моделирование перемещения машин. Объект `Cross` при инициализации получает структуру данных, в которой указана конфигурация перекрестка, то есть расположение машин, их направления движения, номера автоматов, управляющих ими и состояние светофоров, если они есть. По этой информации `Cross` загружает `xml`-модели автоматов машин, и связывает каждый автомат с экземпляром класса `CarCO`, инкапсулирующего отдельную машину.

Таким образом, за исключением двух сообщений, получаемых от объекта `Cross` при начале моделирования, каждый автомат взаимодействует только со своим собственным экземпляром класса `CarCO`, который является для него одновременно и поставщиком событий и объектом управления.

Объект `Cross` служит диспетчером событий для объектов `CarCO`: каждое сообщение, сгенерированное каким-либо объектом `CarCO` направляется в объект Cross, где оно транслируется другим объектам `CarCO`, а те, в свою очередь, передают его своим автоматам.

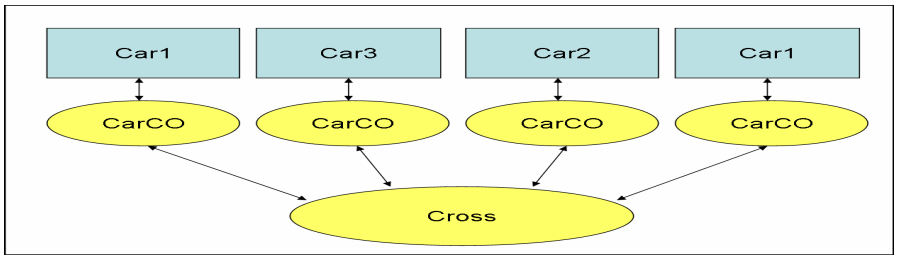

Рис. 8. Схема взаимодействия автоматов

2.3.2. Модель перекрестка

Территория перекрестка разбивается на 13 зон, как показано на рис. 9

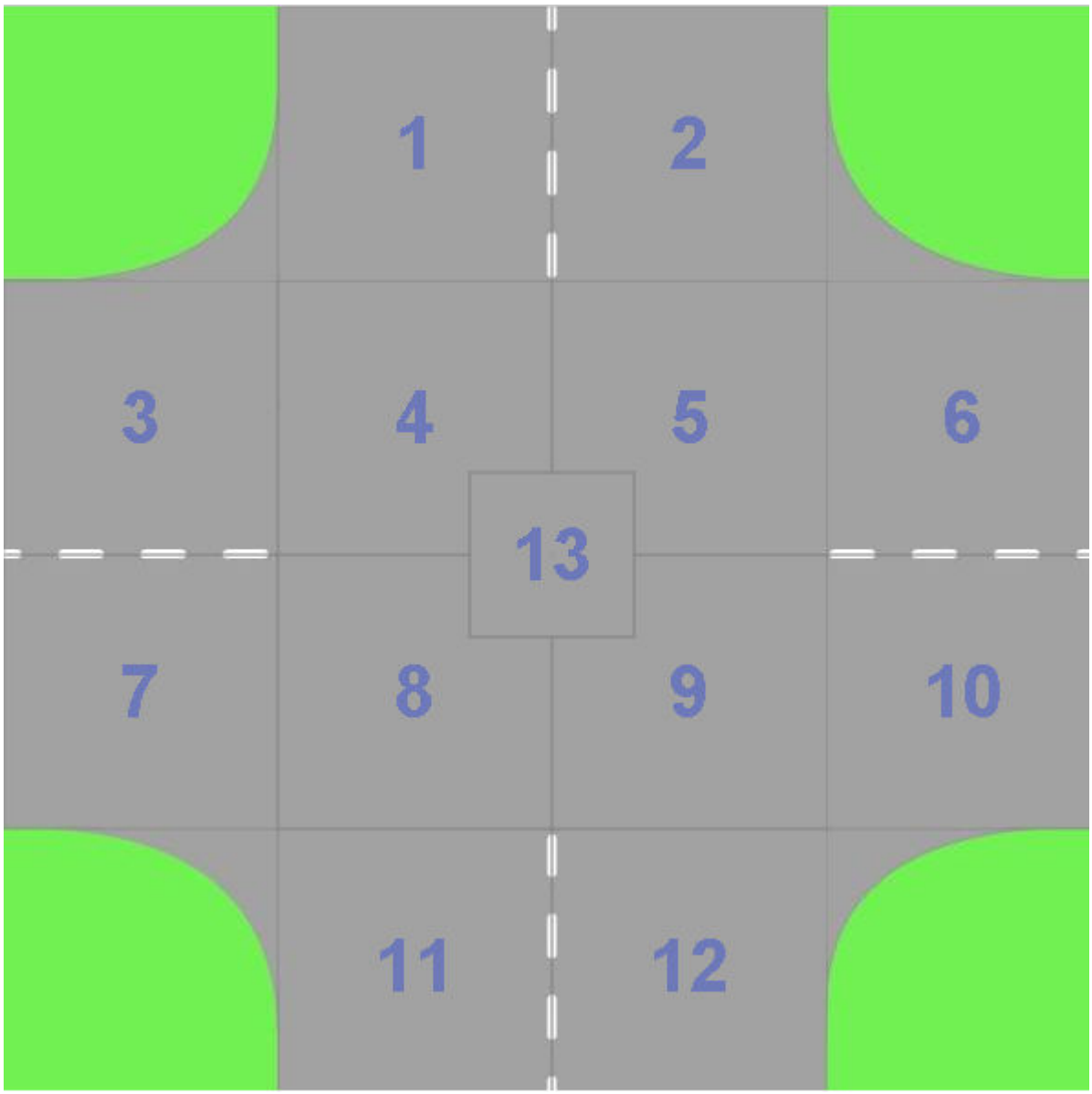

Рис. 9. Разбиение перекрёстка на зоны

Нумерация зон ведется относительно каждой конкретной машинки, то есть для каждой машины встречная машинка будет появляться в зоне 1, левая – в зоне 7, правая – в зоне 6 и т.д.

Машина может двигаться по одной из восьми возможных траекторий: одна траектория поворота направо, одна – прямо (рис. 10), три – налево (рис. 11) и три – на разворот (рис. 12). (Рассматривается случай, когда разворот по малой траектории, не выходящей за пределы зон 8 и 9, невозможен.) Траектория выбирается автоматом в начале моделирования и не может быть изменена после того, как машина начала движение.

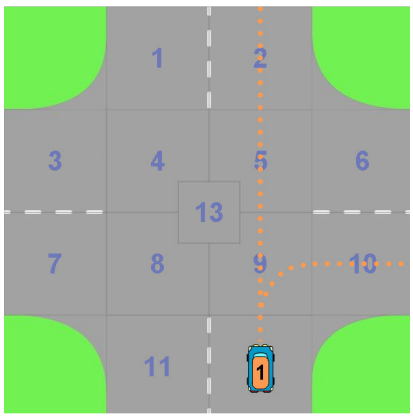

Рис. 10.
Траектории движения прямо и направо

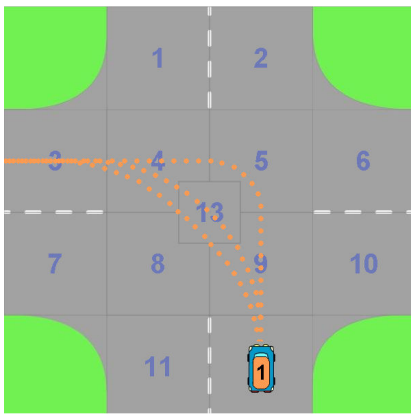

Рис. 11.
Траектории движения налево

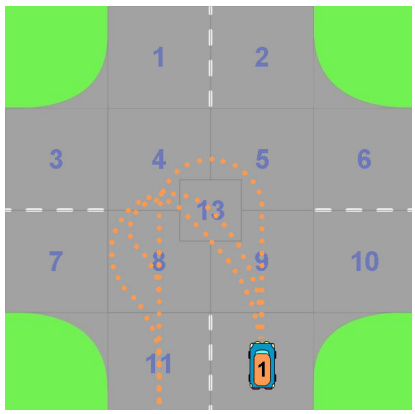

Рис. 12.
Траектории разворота

Направление движения каждой машины задается при редактировании конфигурации. Если выбрано направление налево или на разворот, то конкретная траектория движения (одна из трех) выбирается автоматом в начале моделирования.

Каждая траектория проходит по нескольким зонам перекрестка. Во время движения объект `CarCO` машины отслеживает номер зоны, в которой находится машина. При этом предполагается, что машина находится в той зоне, в которую попадает точка, являющаяся центром машины. При смене зоны объект `CarCO` генерирует сообщение о покидании зоны, которое через объект `Cross` транслируется остальным машинам.

Кроме сообщений о покидании зон другими машинами, автомат получает сообщения, когда его машина пересекает одну из разделительных полос.

Моделирование начинается с того, что объект `Cross` рассылает два сообщения всем машинам:

1. «Включить указатели поворота».
   При получении этого сообщения автомат должен включить необходимые указатели поворота для того, чтобы другие машины могли правильно оценить дорожную обстановку и выбрать тактику действий
2. «Начать движение».
   При получении этого сообщения автомат может выбрать траекторию движения (если машина двигается налево или на разворот), и, если необходимо, начать движение.

После выбора траектории движения автомат может совершать только два выходных воздействия: «Ехать» и «Ждать». Для принятия решения автомат использует входные переменные, которые характеризуют окружающую обстановку: расположение машин на перекрестке и их указатели поворота.

2.3.3. Диаграмма связей

На рис. 13 изображена диаграмма связей с четырьмя однотипными автоматами для управления машиной.

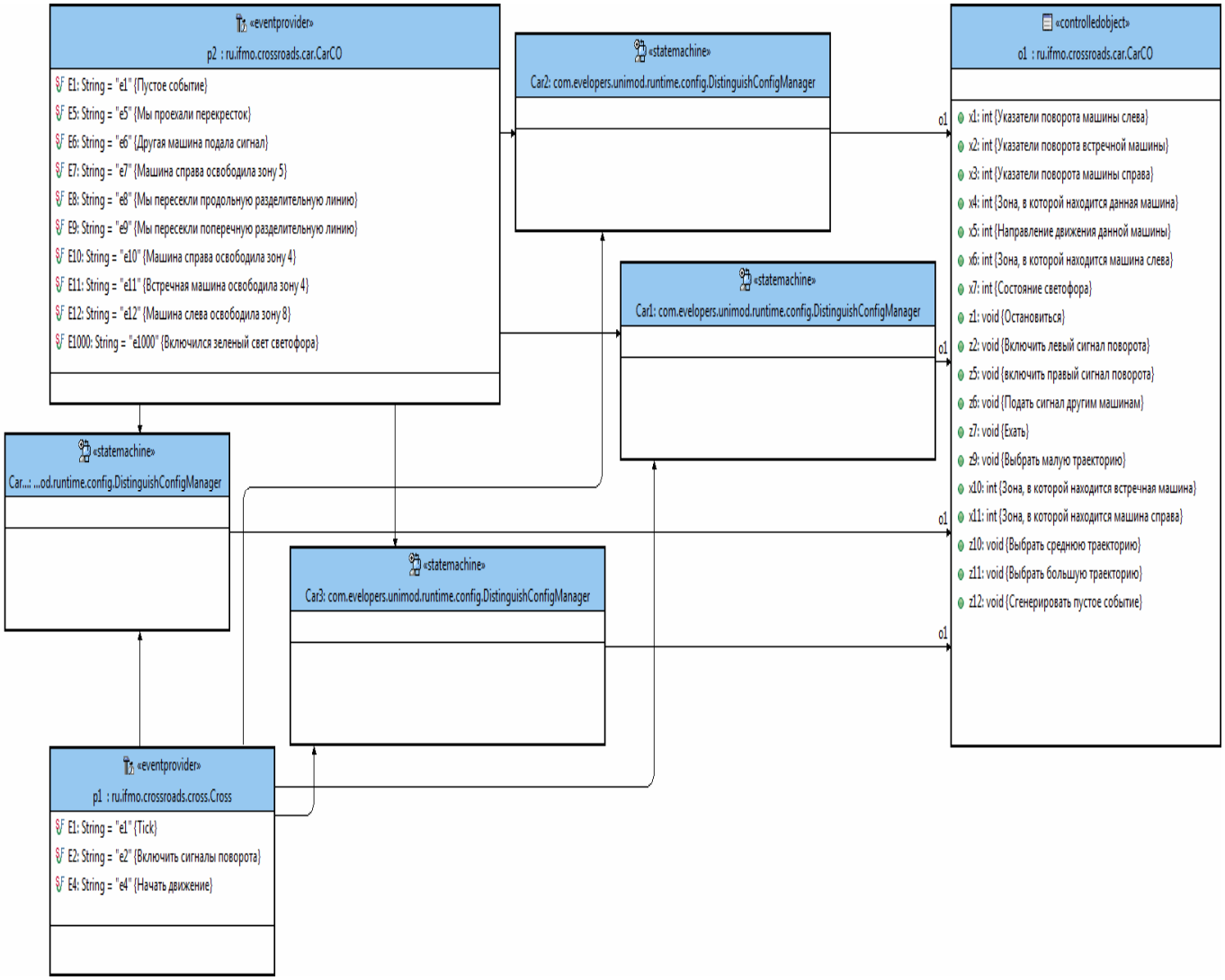

Рис. 13. Диаграмма связей `crossroad.unimod`

Поставщики событий

CarCO

Объект `CarCO` инкапсулирует отдельную машину и отвечает за моделирование перемещения вдоль траектории. Каждая машина может находиться в одном из четырех состояний:

- `MOVING` – машина едет;
- `WAITING` – машина стоит на месте;
- `CRASHED` – машина столкнулась с другой машиной;
- `FINISHED` – машина проехала перекресток.

Объект `CarCO` является основным поставщиком событий для автомата (рис. 8).

Таблица 8

Событие	Описание
E1	Событие по умолчанию
E5	Мы проехали перекрёсток
E6	Другая машина подала сигнал
E7	Машина справа покинула зону 5
E8	Мы пересекли продольную разделительную линию
E9	Мы пересекли поперечную разделительную линию
E10	Машина справа покинула зону 4
E11	Встречная машина покинула зону 4
E12	Машина слева покинула зону 4

Объект `CarCO` отвечает за перемещение машин вдоль выбранной траектории и отслеживание пересекаемых зон. Как только машина покидает какую-либо зону, объект `CarCO` отправляет соответствующее сообщение объекту `Cross` для трансляции остальным машинам.

Сообщения, получаемые автоматом от объекта `CarCO`, можно разделить на два типа:

1. Собственные сообщения.
   К ним относятся три сообщения: о пересечении продольной и поперечной разделительной полосы и сообщение о завершении движения.
2. Сообщения от других машин.
   Эти сообщения генерируются другими объектами `CarCO` и транслируются через объект `Cross`. К ним относятся все сообщения об освобождении зон и специальное сообщение «Кто-то махнул рукой», которое служит для разрешения взаимной блокировки машин.

Приведем пример трансляции сообщений. Допустим, машина слева покинула зону 8 (рис. 14). Относительно машины слева зона 8 является зоной 9, а зона 9 – зоной 5. Сообщения будут передаваться следующим образом:

1. Объект `CarCO` машины слева фиксирует, что машина перешла из зоны 9 в зону 5, генерирует сообщение «Я покинула зону 9» и передаёт его объекту `Cross`.
2. Объект `Cross` переводит это сообщение в следующее: «Машина слева покинула зону 8»
3. Объект `Cross` отправляет сообщение «Машина справа покинула зону 5» объекту `CarCO` машины сверху.
4. Объект `CarCO` машины сверху генерирует сообщение *E7* и отправляет его своему автомату.
5. Объект `Cross` отправляет сообщение «Встречная машина покинула зону 4» объекту `CarCO` машины справа.
6. Объект `CarCO` машины справа генерирует сообщение *E11* и отправляет его своему автомату.

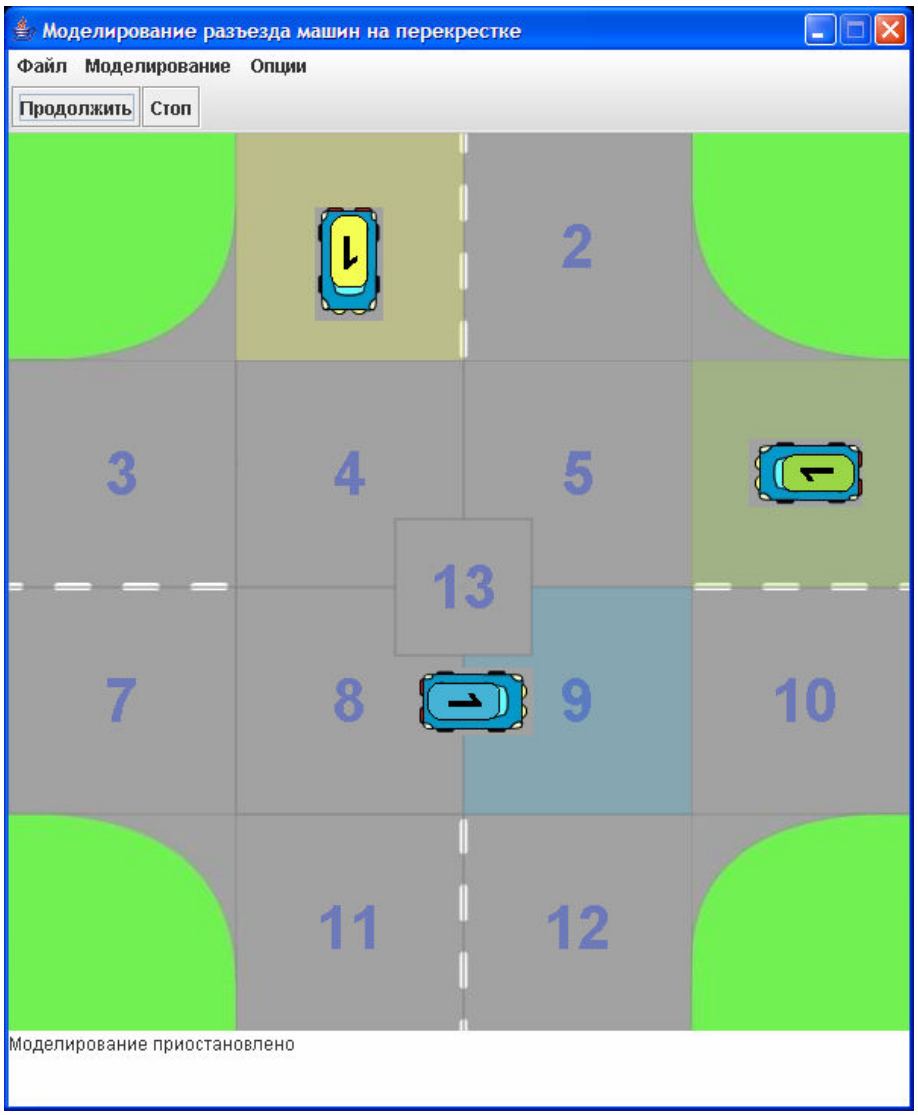

Рис. 14. Пример трансляции сообщений

Cross

Объект `Cross` выполняет три функции:

1. В начале моделирования всем автоматам рассылает сообщения *E2* и *E4* (табл. 9).
2. По тику таймера перемещает машины, находящиеся в состоянии `MOVING`.
3. Транслирует сообщения, сгенерированные объектами `CarCO`, всем остальным машинам.

Таблица 9

Событие	Описание
E2	Включить сигналы поворота
E4	Начать движение

Отметим, что сначала все автоматы получают сообщение *E2*, и лишь после того, как все автоматы обработали это сообщение, рассылается сообщение *E4*. Это важно, так как при получении сообщения *E4* автомат должен выбрать траекторию движения, при этом он может руководствоваться только «внешней» информацией о машинах: их расположении и указателях поворота, и ничего не знает об истинных направлениях движения других машин. В частности, автомат не может различить, собирается ли другая машина повернуть налево или развернуться.

Объекты управления

CarCO

Автомат может получить следующую информацию о дорожной обстановке:
– зона, в которой находится какая-либо машина;
– состояние другой машины;
– направление движения данной машины.

Состояния машин кодируются следующим образом:
0 – отсутствие машины;
1 – указатели поворота выключены;
2 – включен левый указатель поворота;
3 – включен правый указатель поворота;
4 – включены и левый и правый указатель поворота (аварийные огни).

Направления движения машин кодируются следующим образом:
0 – назад;
1 – влево;
2 – вперед;
3 – направо.

Кодировка зон изображена на рис. 9.

Перечень входных переменных приведен в табл. 10.

Таблица 10

Входное воздействие	Описание
X1	Указатели поворота машины слева
X2	Указатели поворота машины спереди
X3	Указатели поворота машины справа
X4	Зона, в которой мы находимся
X5	Направление нашего движения
X6	Зона, в которой находится машина слева
X10	Зона, в которой находится встречная машина
X11	Зона, в которой находится машина справа

Выходными воздействиями автомата (табл. 11) являются:
– управление состоянием указателей поворота;
– управление движением (ехать/стоять);
– подача сигнала другим водителям в случае неопределенности очередности проезда;
– выбор траектории поворота налево или разворота (малая/средняя/большая траектории);
– генерация события по умолчанию.

Последнее действие необходимо, если автомат должен пройти несколько состояний при обработке одного события. Оно является исключительно вспомогательным и служит для упрощения устройства автомата.

Таблица 11

Выходное воздействие	Описание
Z1	Стоять
Z2	Включить левый сигнал поворота
Z5	Включить правый сигнал поворота
Z6	Подать сигнал другим машинам
Z7	Ехать
Z9	Выбрать малую траекторию
Z10	Выбрать среднюю траекторию
Z11	Выбрать большую траекторию
Z12	Сгенерировать событие по умолчанию

Автомат *Car1*

Граф переходов автомата *Car1* приведен на рис. 15. Автомат *Car1* представляет собой правильную реализацию правил проезда нерегулируемых перекрестков равнозначных дорог.

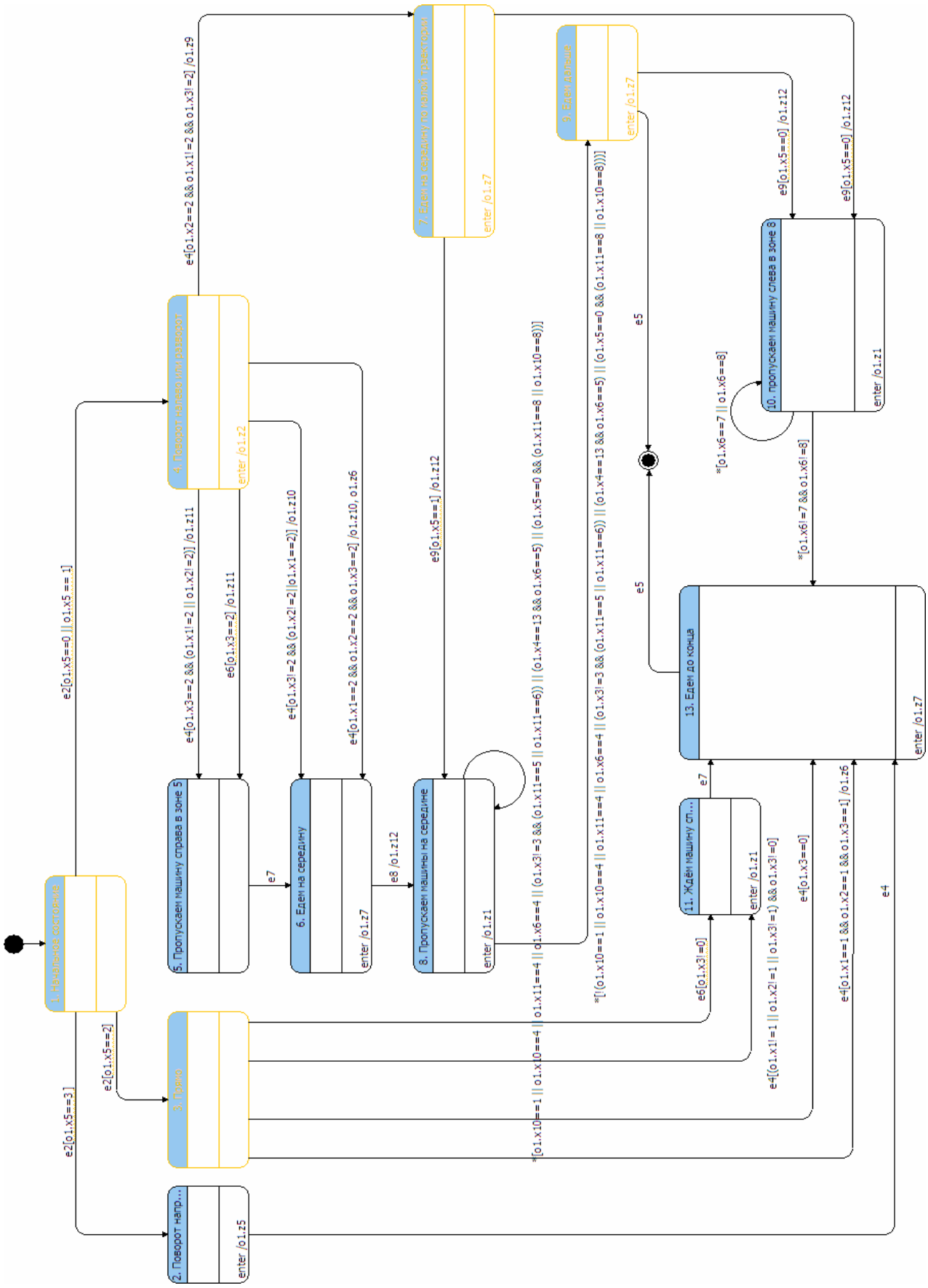

Рис. 15. Автомат *Car1*

Сначала автомат находится в начальном состоянии (состояние 1).
При получении первого сообщения о включении указателей поворота (сообщение *E2* от объекта `Cross`), автомат анализирует выбранное направление движения (`o1.x5`) и переходит в соответствующее состояния, включая при этом необходимые указатели поворота.

Рассмотрим три случая:

1. ***Поворот направо*** (состояние 2)
   В этом случае машина никого не пропускает и едет вдоль траектории до конца
2. ***Движение прямо*** (состояние 3)
   Отдельно рассмотрим случай потенциальной взаимной блокировки.
   a. *Все четыре машины намерены двигаться прямо*.
      В этом случае возникает неопределенность и, в соответствии с ПДД, водители должны договориться об очередности проезда. В автомате это реализовано следующим образом: тот автомат, который первым обрабатывает сообщение *E4* «Начать движение», в этом случае вызывает действие `o1.z6` «Подать сигнал другим машинам» и начинает двигаться (состояние 12). Другие автоматы получают сообщение *E6* «Другая машина подала сигнал» раньше сообщения *E4* и при его обработке переходят в состояние «ждём машину справа» (состояние 11), а сообщение *E4* игнорируют.
   b. *Иначе*
      В любой другой ситуации машина должна пропустить машину справа. Поэтому при получении сообщения *E4* «начать движение» автомат проверяет, есть ли машина справа и, если есть, переходит в состояние «Ждём машину справа» (состояние 11), а иначе – в состояние «Едем до конца» (состояние 12).

   В состоянии «Ждём машину справа» (состояние 11) автомат вызывает действие `o1.z1` «Стоять» и ждёт получения сообщения *E7* «Машина справа покинула зону 5». После этого он переходит в состояние «Едем до конца» (состояние 12).

3. ***Поворот налево или разворот*** (состояние 4)
   В дальнейшем, для простоты описания, фразу «Поворот налево или разворот» будем заменять на «Движение налево».

   В соответствии с ПДД, при движении налево машина должна выехать на середину перекрестка, пропустить машину справа и встречную машину, а если она разворачивается – то и машину слева, и завершить проезд. Конкретная траектория движения машины зависит от направлений движения других машин:

   a. Если встречная машина тоже двигается налево, то имеет место разъезд правыми бортами, то есть машины двигаются одновременно, но так, чтобы не столкнуться. В автомате это соответствует выбору малой траектории (состояние 7).
   b. Если машина справа тоже двигается налево, выехала на середину и остановилась, например, пропуская машину слева, то нужно объехать её сзади. В автомате это соответствует выбору большой траектории (состояние 5).
   c. В остальных случаях выбирается средняя траектория, проходящая через середину перекрестка (состояние 6).

   Так же, как и в случае движения всех машин прямо, возникает неопределенность в случае движения всех четырех машин налево. Ситуация разрешается аналогичным образом: машина, которая первой получает сообщение *E4*, подаёт сигнал и начинает двигаться по средней траектории (состояние 6). Остальные машины при получении сообщения e6 переходят в состояние «Пропускаем машину справа в зоне 5» (состояние 5).

Рассмотрим два случая:

a. *Движение по средней или большой траектории* (состояния 5, 6).
   Если машина справа двигается налево (а, следовательно, выбрана большая траектория), то необходимо сначала её пропустить, а лишь затем выехать на середину (состояние 5). В случае выбора средней траектории автомат сразу переходит в состояние «ехать на середину» (состояние 6), так как его траектория до середины перекрестка не пересекается с траекторией машины справа.

   Считается, что машина доехала до середины перекрестка, когда её центр пересек продольную разделительную линию. В этот момент автомат получает сообщение *E8* «Мы пересекли продольную разделительную линию» и переходит в состоянии 8 «Пропускаем машины в зоне 4». В этом состоянии автомат пребывает до тех пор, пока не выполнится сложное условие, которое мы разберем позже. Это условие проверяется при получении любого сообщения. Для того, чтобы проверить его при попадании в состояние 8, при переходе из состояния 7 генерируется событие по умолчанию, которое инициирует проверку условия сразу после попадания в состояние 8.

   Теперь разберем это условие. Машина должна стоять на месте на середине перекрестка, если верно одно из следующих утверждений:

   - Встречная машина находится в зоне 1 – в своей начальной зоне.
     Куда бы ни ехала встречная машина, нужно её пропустить.

   - Какая-либо машина находится в зоне 4.
     Через зону 4 проходит наша траектория. Поэтому в любом случае требуется дождаться её освобождения.

   - Машина справа двигается прямо или налево и находится в зоне 5 или 6.
     Траектория машины справа пересекает нашу траекторию, и она ещё не доехала до пересечения. Надо её пропустить.

   - Данная машина находится в зоне 13, и машина слева находится в зоне 5.
     Машина слева или разворачивается, или поворачивает налево. При этом невозможно определить это, так как знаем только, что у неё включен левый указатель поворота. Однако если она разворачивается, то она будет помехой справа. Поэтому помеху необходимо на всякий случай пропустить.

   - Наша машина разворачивается, а зона 8 занята другой машиной.
     При этом необходимо дождаться освобождения зоны 8, так как через неё проходит наша траектория.

   Как только все условия становятся ложными, автомат переходит в состояние 9 «Едем дальше». Затем, если данная машина разворачивается (`o1.x5 == 0`), то при пересечении поперечной разделительной линии она снова останавливается (состояние 10) и пропускает машину слева. После этого завершает проезд перекрестка (состояние 12).

b. *Движение по малой траектории* (состояние 7)

Автомат выбирает малую траекторию, если встречная машина тоже двигается налево, а ни машина слева, ни машина справа не двигаются налево.
В этом случае машина двигается вдоль малой траектории до пересечения поперечной разделительной линии (состояние 7). Затем, если она поворачивает налево, то автомат переходит в состояние 8 «Пропускаем машины на середине», а если разворачивается – в состояние 10 «Пропускаем машину слева в зоне 8». Оба этих состояния уже были рассмотрены ранее.

Автомат *Car2*

На рис. 16 приведен граф переходов автомата *Car2*.

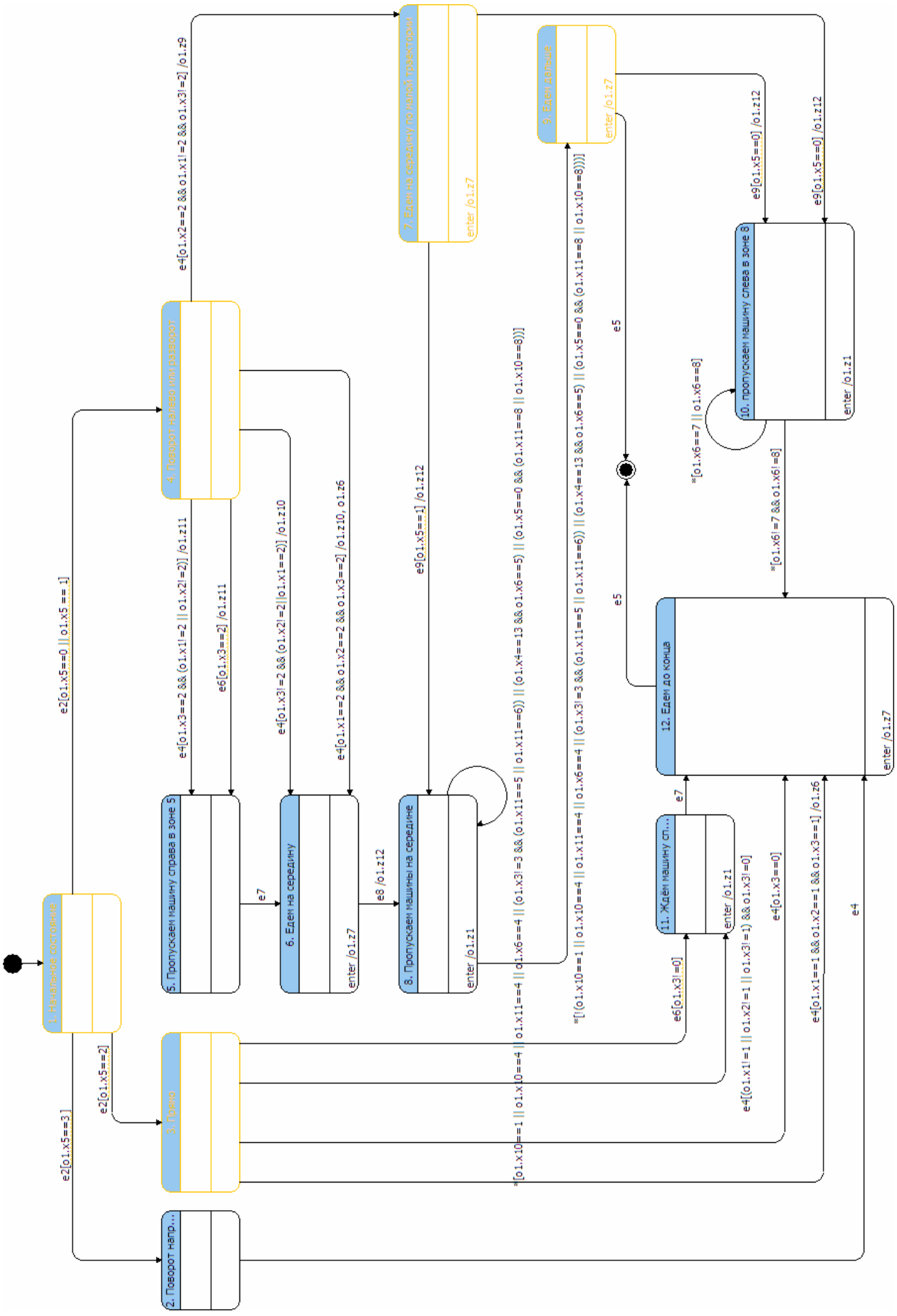

Рис. 16. Автомат *Car2*

Автомат `Car2` является копией автомата *Car1* за исключением того, что он не включает указатели поворота, когда это требуется. Таким образом, автомат *Car2* моделирует действия водителя, который соблюдает правила дорожного движения, но не знает, что у машины не работают указатели поворота, или забывает их включить. Это может привести к аварии, так как остальные участники дорожного движения будут принимать решение о траектории и очередности проезда, полагая, что данная машина движется прямо.

Автомат *Car3*

Автомат *Car3*, напротив, включает необходимые указатели поворота, но совершенно не соблюдает правила дорожного движения: он сразу же начинает движение и никого не пропускает на своём пути. Таким образом, этот автомат имитирует «наглого водителя».

Автомат *CarTL*

Этот автомат имитирует проезд автомобиля с принятием во внимание дорожных светофоров. Автомат реализует поведение «честного водителя».

2.4. Светофоры *UniMod*

2.4.1. Диаграмма связей

На рис. 17 изображена диаграмма связей для работы светофоров.

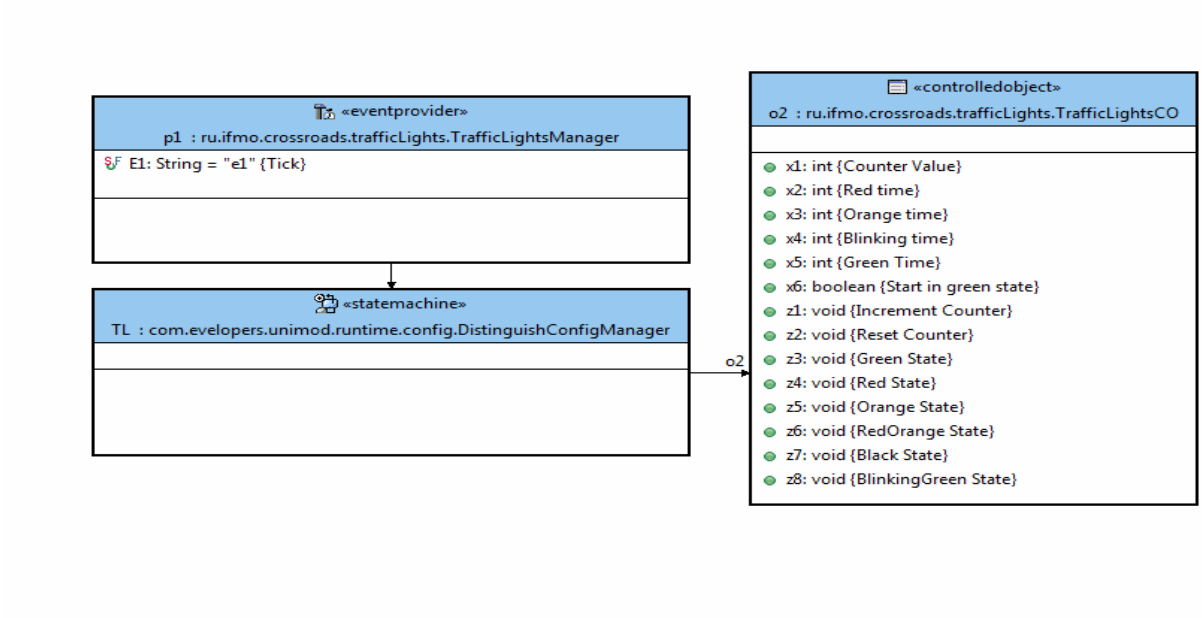

Рис. 17. Диаграмма связей *TrafficLights.unimod*

2.4.2. Автомат *TL*

На рис. 18 приведен автомат, моделирующий работу светофора.

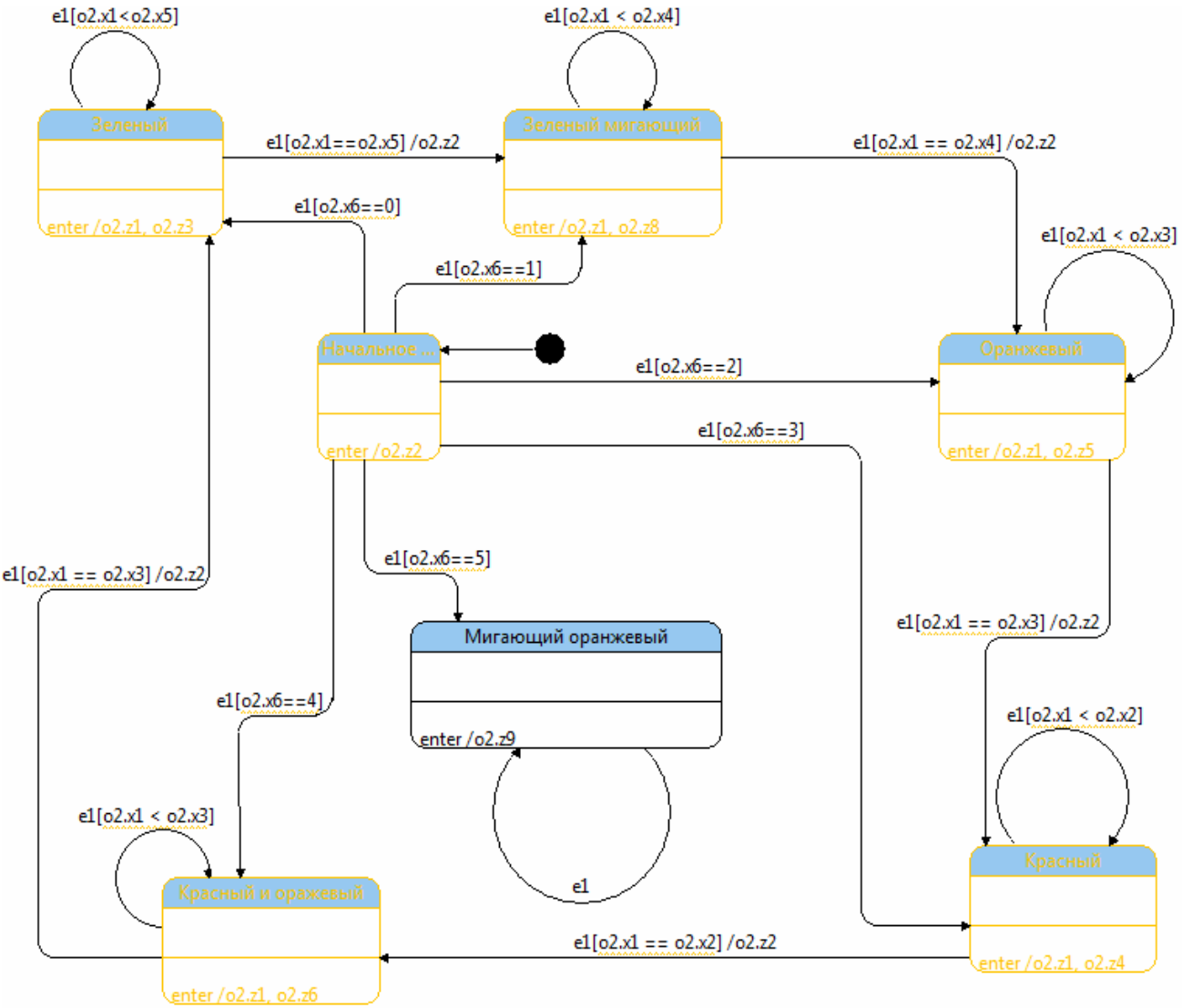

Рис. 18. Автомат *TL*

Входные и выходные воздействия этого автомата описаны в табл.13, 14.

Таблица 13

Входное воздействие	Описание
X1	Значение счетчика
X2	Время для красного света
X3	Время для оранжевого света
X4	Время для мигающего зеленого света
X5	Время для зеленого света
X6	Старт в зеленом свете

Таблица 14

Выходное воздействие	Описание
Z1	Увеличение счетчика
Z2	Сбросить счетчик
Z3	Зеленое состояние
Z4	Красное состояние
Z5	Оранжевое состояние
Z6	Красно-оранжевое состояние
Z7	Черное состояние
Z8	Состояние мигающего зеленого

Заключение

Возможные пути совершенствования проекта состоят в рассмотрении более широких классов: неравнозначных дорог и добавлении других транспортных средств таких, как трамваи и автомобили со специальными сигналами. Однако это приведёт к существенному усложнению автоматов.

Литература

1. *Шалыто А.А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. http://is.ifmo.ru/books/switch/1
2. *Паращенко Д. А., Царев Ф. Н., Шалыто А. А.* Технология моделирования одного класса мультиагентных систем на основе автоматного программирования на примере игры «Соревнование летающих тарелок». Проектная документация. СПбГУ ИТМО. 2006. http://is.ifmo.ru/unimod-projects/plates/
3. *Правила дорожного движения.* Утверждены постановлением Правительства РФ № 1090 от 23.10.93 (в редакции от 28.02.2006).

Приложение. Исходные коды

Main.java

```
package ru.ifmo.crossroads;

import java.io.IOException;

import javax.xml.parsers.ParserConfigurationException;

import org.apache.commons.logging.LogFactory;
import org.apache.log4j.xml.DOMConfigurator;
import org.xml.sax.SAXException;

import com.evelopers.common.exception.CommonException;
import com.evelopers.unimod.adapter.standalone.Run;
import com.evelopers.unimod.core.stateworks.Model;
import com.evelopers.unimod.debug.ExceptionHandlerImpl;
import com.evelopers.unimod.runtime.EventProcessorException;
import com.evelopers.unimod.runtime.EventProcessorListener;
import com.evelopers.unimod.runtime.ExceptionHandler;
import com.evelopers.unimod.runtime.ModelEngine;
import com.evelopers.unimod.runtime.interpretation.InterpretationHelper;
import com.evelopers.unimod.runtime.logger.SimpleLogger;
import com.evelopers.unimod.transform.TransformException;
import com.evelopers.unimod.transform.xml.XMLToModel;

public class Main {
    protected void run(String arg) throws IOException, SAXException,
    ParserConfigurationException, TransformException, InterruptedException,
    EventProcessorException, CommonException {
        DOMConfigurator.configure(Run.class.getResource("log4j.xml"));
        Model model = XMLToModel.loadAndCompile(Main.class.getClassLoader()
                .getResourceAsStream(arg));
        InterpretationHelper helper = InterpretationHelper.getInstance();
        ModelEngine engine = helper.createStandAloneModelEngine(model, true);
        EventProcessorListener logger = new SimpleLogger(LogFactory
                .getLog(Run.class));
        engine.getEventProcessor().addEventProcessorListener(logger);
        ExceptionHandler eh = new ExceptionHandlerImpl();
        engine.getEventProcessor().addExceptionHandler(eh);
        engine.start();
    }

    public static void main(String[] args) throws IOException, SAXException,
    ParserConfigurationException, TransformException, InterruptedException,
    EventProcessorException, CommonException {
        Main m = new Main();
        m.run("unimod/Core/Main.xml");
    }
}
```

Car.java

```
package ru.ifmo.crossroads.car;

import java.awt.Color;
import java.awt.Point;

import ru.ifmo.crossroads.common.CarState;
import ru.ifmo.crossroads.common.Direction;
```

```
import ru.ifmo.crossroads.common.Position;
import ru.ifmo.crossroads.common.Side;
import ru.ifmo.crossroads.common.Zone;
import ru.ifmo.crossroads.cross.Cross;
import ru.ifmo.crossroads.trajectory.Trajectory;

/**
 * Отвечает за текущее положение и перемещение машины.
 */
public abstract class Car implements MoveableCar {
    /** Хранит информацию о перекрёстке */
      protected Cross cross = null;

    /** Сторона перекрестка, на которой находится машина */
    protected Side side;

    /** Направление движения машины - задается пользователем */
    protected Direction dir;

    /** Траектория движения машины. Выбирается автоматом */
    protected Trajectory traj = Trajectory.Forward;

    /** Расстояние, пройденное машиной вдоль своей траектории */
    protected double distance = 0;

    /** Отвечает за мигание правого сигнала поворота */
    protected boolean bRight = false;

    /** Отвечает за мигание левого сигнала поворота */
    protected boolean bLeft = false;

    /** Цвет */
    protected Color color = Color.orange;

    /** Состояние машины */
    protected CarState state = CarState.WAITING;

    /** Зона, в которой сейчас находится машина */
    protected Zone zone = Zone.OutDownRight;

    /** Скорость движения машины */
    private double speed = 4;


    public Car() {}

    /**
     * @param side направление, откуда приближается машина
     * @param dir направление движения машины
     * @param cross конкретный перекрёсток, где находится машина
     */
    public Car(Side side, Direction dir, Cross cross) {
        this.side = side;
        this.cross = cross;
        this.dir = dir;
        if (dir == Direction.BACK) traj = Trajectory.BackMiddle;
        if (dir == Direction.LEFT) traj = Trajectory.LeftMiddle;
        if (dir == Direction.FORWARD) traj = Trajectory.Forward;
        if (dir == Direction.RIGHT) traj = Trajectory.Right;
    }

    /**
     * @return направление движения машины
     */
```

```
public Direction getDirection() {
    return dir;
}

/**
 * @return текущие координаты центра машины
 */
public Point getCenterPoint() {
    return getLocalPosition().getCarCenter();
}

/**
 * @return текущее локальное положение машины
 */
protected Position getLocalPosition() {
    if (isFinished()) return traj.getLastPosition();
    return traj.getPosition(distance);
}

/**
 * @return информацию о включённых сигналах поворота
 */
public int getIndicators() {
    if (!bLeft && !bRight) return 1;
    else if (bLeft && !bRight) return 2;
    else if (!bLeft && bRight) return 3;
    else return 4;
}

/**
 * @return зону, в которой находится машина
 */
public Zone getZone() {
    return side.untransposeZone(getLocalZone());
}

/**
 * Выводит инфомацию о машине на консоль
 * @param s строка с информацией
 */
public void log(String s) {
    System.out.println("Машина " + side + ": " + s);
}

/**
 * @return мигает ли левый поворотник
 */
public boolean blinkingLeft() {
    return bLeft;
}

/**
 * @return мигает ли правый поворотник
 */
public boolean blinkingRight() {
    return bRight;
}

/**
 * @return разбилась ли машина
 */
public boolean isCrashed() {
    return state == CarState.CRASHED;
}
```

```
    /**
     * "Разбивает машину". Останавливает движение, включает аварийную сигнализацию
     *      и пишет в консоль соответствующее сообщение.
     */
    public void crash() {
        state = CarState.CRASHED;
        bLeft = true;
        bRight = true;
        log("Разбилась");
    }

    /**
     * @return текущее абсолютное положение машины
     */
    public Position getPosition() {
        return side.untransposePosition(getLocalPosition());
    }

    /**
     * @return закончено ли моделирование движения машины
     */
    public boolean isFinished() {
        return state == CarState.FINISHED;
    }

    /**
     * @return продолжает ли машина движение
     */
    public boolean isMoving() {
        return state == CarState.MOVING;
    }

    /**
     * Отправляет сообщение на перекресток о том, что машина покинула зону
<code>z</code>
     * @param z зона, которую покидает машина
     */
    private void sendLeaveZoneMessage(Zone z) {
        cross.distributeLeaveZoneMessage(side, side.untransposeZone(z));
    }

    /**
     * @return сторону, с которой направляется машина
     */
    public Side getSide() {
        return side;
    }

    /**
     * Перемещает машину вдоль выбранной траектории
     * @param dt интервал времени, определяющий степень дискретизации движения
     */
    public void move(int dt) {
        if (isMoving()) {
            distance += dt * speed;
            Zone z;
            if (distance >= traj.getLen()) {
                state = CarState.FINISHED;
                distance = traj.getLen();
            }
            z = Zone.fromPosition(traj.getPosition(distance));
            if (z != zone && zone != Zone.OutOfCross) {
                Zone oldZone = zone;
```

```
                zone = z;
                sendLeaveZoneMessage(oldZone);
            } else zone = z;
        }
    }

    /**
     * @return текущая локальная зона, в которой находится машина
     */
    public Zone getLocalZone() {
        return zone;
    }

    /**
     * @param s пройденный путь
     * @return точку, в которой окажется машина, пройдя путь <code>s</code>
     */
    public Point getFuturePosition(double s) {
        return side.untransposePoint(traj.getPosition(distance + s));
    }

    /**
     * @return оставшееся расстояние до конца траектории
     */
    public double getLeftDistance() {
        return distance < traj.getLen() ? (traj.getLen() - distance) : 0;
    }
}
```

CarCO.java

```
package ru.ifmo.crossroads.car;

import java.awt.Point;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;

import org.apache.commons.logging.LogFactory;
import com.evelopers.common.exception.CommonException;
import com.evelopers.unimod.adapter.standalone.Run;
import com.evelopers.unimod.core.stateworks.Event;
import com.evelopers.unimod.core.stateworks.Model;
import com.evelopers.unimod.debug.ExceptionHandlerImpl;
import com.evelopers.unimod.runtime.ControlledObject;
import com.evelopers.unimod.runtime.ControlledObjectsMap;
import com.evelopers.unimod.runtime.EventProcessorListener;
import com.evelopers.unimod.runtime.EventProvider;
import com.evelopers.unimod.runtime.ExceptionHandler;
import com.evelopers.unimod.runtime.ModelEngine;
import com.evelopers.unimod.runtime.context.StateMachineContext;
import com.evelopers.unimod.runtime.context.StateMachineContextImpl;
import com.evelopers.unimod.runtime.interpretation.InterpretationHelper;
import com.evelopers.unimod.runtime.logger.SimpleLogger;
import com.evelopers.unimod.transform.TransformException;
import com.evelopers.unimod.transform.xml.XMLToModel;

import ru.ifmo.crossroads.common.CarState;
import ru.ifmo.crossroads.common.Direction;
import ru.ifmo.crossroads.common.Side;
import ru.ifmo.crossroads.common.Zone;
import ru.ifmo.crossroads.cross.Cross;
import ru.ifmo.crossroads.trajectory.Trajectory;
```

```
/**
 * Инкапсулирует сущность машины. Каждый автомат взаимодействует только со своим
 * собственным экземпляром класса CarCO, который является для него одновременно
 * и поставщиком событий, и контролируемым объектом.
 */
public class CarCO extends Car implements ControlledObject, EventProvider{
    /** Автомат, управляющий машиной */
      private ModelEngine engine = null;

    /** Номер автомата */
    private int automatId = 1;

    /** @unimod.event.descr Мы проехали перекресток*/
    public static final String E5 = "e5";

    /** @unimod.event.descr Другая машина подала сигнал */
    public static final String E6 = "e6";

    /** @unimod.event.descr Машина справа освободила зону 5 */
    public static final String E7 = "e7";

    /** @unimod.event.descr Мы пересекли продольную разделительную линию */
    public static final String E8 = "e8";

    /** @unimod.event.descr Мы пересекли поперечную разделительную линию */
    public static final String E9 = "e9";

    /** @unimod.event.descr Машина справа освободила зону 4 */
    public static final String E10 = "e10";

    /** @unimod.event.descr Встречная машина освободила зону 4 */
    public static final String E11 = "e11";

    /** @unimod.event.descr Машина слева освободила зону 8 */
    public static final String E12 = "e12";

    /** @unimod.event.descr Пустое событие */
    public static final String E1 = "e1";

    /** Количество автоматов */
    public static final int CarAutomatNum = 3;

    /**
     * @return номер автомата у данной машины
     */
    public int getAutomatId(){
        return automatId;
    }

    public CarCO() {}

    /**
     * @param automatId номер автомата
     * @param xmlFile xml-файл с автоматом
     * @param side сторона, с которой едет машина
     * @param dir направление движения машины
     * @param cross перекрёсток, на котором находится машина
     */
    public CarCO(int automatId, String xmlFile, Side side, Direction dir, Cross
cross) {
        super(side, dir, cross);
        this.automatId = automatId;
        loadEngine(xmlFile);
```

```
    }

    /**
     * Список из объектов CarCO
     */
    private class CarCOMap implements ControlledObjectsMap {
        private ControlledObject co = null;

        public CarCOMap(ControlledObject co){
            this.co = co;
        }

        public ControlledObject getControlledObject(String coName) {
            return co;
        }
    }

    /**
     * Посылает автомату сообщение о том, что машина со стороны <code>s</code>
     * покинула зону <code>z</code>
     * @param s сторона, с которой направлялась машина
     * @param z зона, которую покинула машина
     */
    public void sendLeaveZoneEvent(Side s, Zone z) {
        if (s == Side.RIGHT && z == Zone.UpRight) sendEventToCar(E7);
        else if (s == Side.RIGHT && z == Zone.UpLeft) sendEventToCar(E10);
        else if (s == Side.UP && z == Zone.UpLeft) sendEventToCar(E11);
        else if (s == Side.LEFT && z == Zone.DownLeft) sendEventToCar(E12);
        else sendEventToCar(E1);
    }

    /**
     * Перемещает машину вдоль выбранной траектории, вызывая метод, определённый в
     * <code>Car</code>, и посылая соответствующие сообщения другим машинам
     * @param dt интервал времени, определяющий степень дискретизации движения
     */
    public void move(int dt) {
        Point p1 = getCenterPoint();
        super.move(dt);
        Point p2 = getCenterPoint();
        if ((p1.x > 0) && (p2.x <= 0)) sendEventToCar(E8);
        if ((p2.y != 0) && (p1.y * p2.y <= 0)) sendEventToCar(E9);
    }

    /**
     * Посылает автомату сообщение о том, что нам махнули рукой
     */
    public void sendWaveHandEvent() {
        sendEventToCar(E6);
    }

    /**
     * Посылает сообщение другим машинам
     * @param event событие, сообщение о котором посылается другим машинам
     */
    public void sendEventToCar(String event) {
        log("Обрабатываю сообщение " + event);
        engine.getEventManager().handleAndWait(new Event(event),
StateMachineContextImpl.create());
    }

    /**
     * @unimod.action.descr Указатели поворота машины, находящейся слева
     */
```

```
    public int x1(StateMachineContext context) {
        return cross.getCarIndicators(side.untransposeSide(Side.LEFT));
    }

    /**
     * @unimod.action.descr Указатели поворота машины, находящейся  спереди
     */
    public int x2(StateMachineContext context) {
        return cross.getCarIndicators(side.untransposeSide(Side.UP));
    }

    /**
     * @unimod.action.descr Указатели поворота машины, находящейся справа
     */
    public int x3(StateMachineContext context) {
        return cross.getCarIndicators(side.untransposeSide(Side.RIGHT));
    }

    /**
     * @unimod.action.descr Направление движения машины
     */
    public int x5(StateMachineContext context) {
      return dir.ordinal();
    }

    /**
     * @unimod.action.descr Стоять
     */
    public void z1(StateMachineContext context) {
        if (state != CarState.MOVING) return;
      log("стою");
        state = CarState.WAITING;
    }

    /**
     * Подгружает автомат для машины
     * @param xmlFile xml-файл с автоматом
     */
    private void loadEngine(String xmlFile) {
        Model model;
        try {
            InputStream is =
Cross.class.getClassLoader().getResourceAsStream(xmlFile);
            System.out.println(new File(xmlFile).getAbsolutePath());
            if (is == null) {
                System.out.println("is == null");
                System.exit(0);
            }
            model = XMLToModel.loadAndCompile(is);
        } catch (IOException e) {
            System.out.println("[ERROR]   Can't load xml: " + xmlFile);
            throw new RuntimeException();
        } catch (TransformException e) {
            System.err.println(e.getMessage());
            throw new RuntimeException();
        }

        InterpretationHelper helper = InterpretationHelper.getInstance();

        try {
            engine = helper.createBuildInModelEngine(model,
                    new CarCOMap(this), true);

        } catch (CommonException e) {
```

```
            System.err.println(e.getMessage());
            throw new RuntimeException();
        }
        try {
            engine.start();
        } catch (CommonException e) {
            e.printStackTrace();
        }
        EventProcessorListener logger = new SimpleLogger(LogFactory
                .getLog(Run.class));
        engine.getEventProcessor().addEventProcessorListener(logger);

        ExceptionHandler eh = new ExceptionHandlerImpl();
        engine.getEventProcessor().addExceptionHandler(eh);
    }

    /**
     * @unimod.action.descr Включить левый сигнал поворота
     */
    public void z2(StateMachineContext context) {
      bLeft = true;
        log("Включаю левый поворотник");
    }

    /**
     * @unimod.action.descr включить правый сигнал поворота
     */
    public void z5(StateMachineContext context) {
        if (state == CarState.CRASHED || state == CarState.FINISHED) return;
      bRight = true;
        log("Включаю правый поворотник");
    }

    /**
     * @unimod.action.descr Подать сигнал другим машинам
     */
    public void z6(StateMachineContext context) {
        if (state == CarState.CRASHED || state == CarState.FINISHED) return;
        log("Машу рукой");
      cross.distributeWaveHandMessage(side);
    }

    /**
     * @unimod.action.descr Ехать
     */
    public void z7(StateMachineContext context) {
        if (state != CarState.WAITING) return;
        log("Еду");
        state = CarState.MOVING;
    }

    /**
     * @unimod.action.descr Выбрать малую траекторию
     */
    public void z9(StateMachineContext context) {
        if (distance > 0) return;
      if (dir != Direction.BACK && dir != Direction.LEFT) return;
        log("Выбираю малую траекторию");
        if (dir == Direction.BACK) traj = Trajectory.BackShort;
        else traj = Trajectory.LeftShort;
    }

    /**
     * @unimod.action.descr Выбрать среднюю траекторию
```

```
     */
    public void z10(StateMachineContext context) {
        if (distance > 0) return;
        if (dir != Direction.BACK && dir != Direction.LEFT) return;
        log("Выбираю среднюю траекторию");
        if (dir == Direction.BACK) traj = Trajectory.BackMiddle;
        else traj = Trajectory.LeftMiddle;
    }

    /**
     * @unimod.action.descr Выбрать большую траекторию
     */
    public void z11(StateMachineContext context) {
        if (distance > 0) return;
        if (dir != Direction.BACK && dir != Direction.LEFT) return;
        log("Выбираю большую траекторию");
        if (dir == Direction.BACK) traj = Trajectory.BackLong;
        else traj = Trajectory.LeftLong;
    }

    /**
     * @unimod.action.descr Зона, в которой находится машина, находящаяся слева
     */
    public int x6(StateMachineContext context) {
      return
side.transposeZone(cross.getCarZone(side.untransposeSide(Side.LEFT))).ordinal();
    }

    /**
     * @unimod.action.descr Зона, в которой находится встречная машина
     */
    public int x10(StateMachineContext context) {
        return
side.transposeZone(cross.getCarZone(side.untransposeSide(Side.UP))).ordinal();
    }

    /**
     * @unimod.action.descr Зона, в которой находится машина, находящаяся справа
     */
    public int x11(StateMachineContext context) {
        return
side.transposeZone(cross.getCarZone(side.untransposeSide(Side.RIGHT))).ordinal();
    }

    /**
     * @unimod.action.descr Сгенерировать пустое событие
     */
    public void z12(StateMachineContext context) {
      engine.getEventManager().handle(new Event(E1),
StateMachineContextImpl.create());
    }

    public void dispose() {}

    public void init(ModelEngine engine) throws CommonException {}

    /**
     * @unimod.action.descr Зона, в которой находится данная машина
     */
    public int x4(StateMachineContext context) {
      return zone.ordinal();
    }
}
```

CarData.java

```
package ru.ifmo.crossroads.car;

import java.awt.Point;

import ru.ifmo.crossroads.common.Direction;
import ru.ifmo.crossroads.common.Position;
import ru.ifmo.crossroads.common.Side;
import ru.ifmo.crossroads.common.Zone;

public interface CarData {
    public Position getPosition();
    public boolean blinkingLeft();
    public boolean blinkingRight();
    public boolean isCrashed();
    public Point getFuturePosition(double s);
    public double getLeftDistance();
    public int getAutomatId();
    public Direction getDirection();
    public Side getSide();
    public Zone getZone();
}
```

CarInfo.java

```
package ru.ifmo.crossroads.car;

import java.awt.Point;
import java.io.Serializable;

import ru.ifmo.crossroads.common.Direction;
import ru.ifmo.crossroads.common.Position;
import ru.ifmo.crossroads.common.Side;
import ru.ifmo.crossroads.common.Zone;

/**
 * Содержит информацию о машине
 */
public class CarInfo implements CarData, Serializable{
      private static final long serialVersionUID = 1L;

      /** Номер автомата */
      private int automatId = 0;

      /** Направление движения машины */
    private Direction dir = Direction.FORWARD;

    /** Сторона, откуда едет машина */
    private Side side = Side.DOWN;

    /** Путь к файлу с автоматами для машин */
    private static final String CAR_AUTOMATON_FILE = "..\\cars\\car";

    public boolean blinkingLeft() {
        return false;
    }

    public boolean blinkingRight() {
        return false;
    }

    /**
```

```
     * @param side сторона, откуда едет машина
     * @param dir направление движения машины
     * @param automatId номер автомата
     */
    public CarInfo(Side side, Direction dir, int automatId) {
        this.side = side;
        this.dir = dir;
        this.automatId = automatId;
    }

    /**
     * @return номер автомата у данной машины
     */
    public int getAutomatId() {
        return automatId;
    }

    /**
     * @return направление движения машины
     */
    public Direction getDirection() {
        return dir;
    }

    /**
     * Устанавливает автомат
     * @param id номер автомата
     */
    public void setAutomatId(int id){
        automatId = id;
    }

    /**
     * Устанавливает направление движения
     * @param d направление движения
     */
    public void setDirection(Direction d){
        dir = d;
    }

    public Point getFuturePosition(double s) {
        return null;
    }

    public Zone getZone(){
        return side.untransposeZone(Zone.OutDownRight);
    }

    public double getLeftDistance() {
        return 0;
    }

    public Position getPosition() {
        return side.untransposePosition(new Position(50, -150, Math.PI / 2));
    }

    /**
     *  @return сторону, откуда едет машина
     */
    public Side getSide() {
        return side;
    }

    public boolean isCrashed() {
```

```
        return false;
    }

    /**
     * @return точный путь к xml-файлу, содержащему автомат
     */
    public String getFileName() {
        return CAR_AUTOMATON_FILE + Integer.toString(automatId) + ".xml";
    }
}
```

MoveableCar.java

```
package ru.ifmo.crossroads.car;

public interface MoveableCar extends CarData {
    public void move(int dt);
    public boolean isMoving();
    public boolean isFinished();
}
```

CarState.java

```
package ru.ifmo.crossroads.common;

/**
 * Состояние машины. Разбилась, закончила движение, движется,
 * временно остановилась, чтобы пропустить другие машины.
 */
public enum CarState{
    CRASHED, FINISHED, MOVING, WAITING;
}
```

Direction.java

```
package ru.ifmo.crossroads.common;

/**
 * Направление движения
 */
public enum Direction {
    BACK{
        @Override
        public String toString() {
            return "назад";
        }

    },
    LEFT{
        @Override
        public String toString() {
            return "налево";
        }

    },
    FORWARD{
        @Override
        public String toString() {
            return "прямо";
        }

    },
```

```
    RIGHT{
        @Override
        public String toString() {
            return "направо";
        }
    },
    NULL{
        @Override
        public String toString() {
            return "отсутствует";
        }
    };
}
```

Position.java

```
package ru.ifmo.crossroads.common;

import java.awt.Point;
import java.awt.geom.AffineTransform;
import java.awt.geom.Area;
import java.awt.geom.Rectangle2D;

import ru.ifmo.crossroads.cross.Cross;

/**
 * Описывает положение машины на перекрестке
 */
public class Position extends Point {
      private static final long serialVersionUID = 1L;

      /** Определяет предел, после которого машины сталкиваются */
      private final static double MIN_CRASH_AREA = 10.0;

      /**
       * Угол поворота. В радианах, отсчитывается от положительного
       * направления оси X против часовой стрелки.
       */
    public double angle;

    /**
     * @param x абсцисса
     * @param y ордината
     * @param angle угол поворота
     */
    public Position(long x, long y, double angle) {
        super((int)x, (int)y);
        this.angle = angle;
    }

    /**
     * @return площадь, занимаемую машиной
     */
    public Area getCarArea() {
        Area a = new Area(new Rectangle2D.Double(-Cross.CarLength / 2, -
Cross.CarWidth / 2,
                Cross.CarLength, Cross.CarWidth));
        a.transform(new AffineTransform(Math.cos(angle), Math.sin(angle), -
Math.sin(angle), Math.cos(angle), x, y));
        return a;
    }

    /**
```

```
     * Определяет столкновения
     * @param p позиция машины
     * @return было ли столкновение
     */
    public boolean isCrash(Position p) {
        Zone z = Zone.fromPosition(p);
        if (z == Zone.OutOfCross || z == Zone.OutOfCross) return false;
        Area a1 = getCarArea(), a2 = p.getCarArea();
        a1.intersect(a2);
        if (a1.isEmpty()) return false;
        Rectangle2D r = a1.getBounds();
        return r.getWidth() * r.getHeight() > MIN_CRASH_AREA;
    }

    public static void main(String args[]) {
        Position p1 = new Position(30, 30, Math.PI / 4), p2 = new Position(50, 20,
Math.PI / 2);
        System.out.println(p1.isCrash(p2));
    }

    /**
     * @return координаты центра машины
     */
    public Point getCarCenter() {
        final double offcenter = 4;
        return new Point((int)Math.round(x - offcenter * Math.cos(angle)),
                (int)Math.round(y - offcenter * Math.sin(angle)));
    }
}
```

Side.java

```
package ru.ifmo.crossroads.common;

import java.awt.Point;

/**
 * Сторона перекрёстка
 */
public enum Side {
    DOWN{
        @Override
        public String toString() {
            return "снизу";
        }

        public Point transposePoint(Point p){
            return p;
        }
    },
    LEFT{
        @Override
        public String toString() {
            return "слева";
        }

        public Position transposePosition(Position p){
            transposePoint(p);
            p.angle += Math.PI / 2;
            return p;
        }

        public Point transposePoint(Point p){
```

```
            int y = p.x;
            p.x = -p.y;
            p.y = y;
            return p;
        }
    },
    UP{
        @Override
        public String toString() {
            return "сверху";
        }

        public Position transposePosition(Position p){
            transposePoint(p);
            p.angle -= Math.PI;
            return p;
        }

        public Point transposePoint(Point p){
            p.x = -p.x;
            p.y = -p.y;
            return p;
        }
    },
    RIGHT{
        @Override
        public String toString() {
            return "справа";
        }

        public Position transposePosition(Position p){
            transposePoint(p);
            p.angle -= Math.PI / 2;
            return p;
        }

        public Point transposePoint(Point p){
            int y = -p.x;
            p.x = p.y;
            p.y = y;
            return p;
        }
    };

    /** Перевод абсолютной стороны перекрестка в относительную */
    public Side transposeSide(Side s){
        return Side.values()[(s.ordinal() - this.ordinal() + 4) % 4];
    }

    /** Перевод относительной стороны перекрестка в абсолютную */
    public Side untransposeSide(Side s){
        return Side.values()[(this.ordinal() + s.ordinal()) % 4];
    }

    /** Перевод абсолютной зоны в относительную */
    public Zone transposeZone(Zone z){
        if (z == Zone.OutOfCross) return z;
        return Zone.fromPoint(transposePoint(z.center()));
    }

    /** Перевод относительной зоны в абсолютную */
    public Zone untransposeZone(Zone z){
        if (z == Zone.OutOfCross) return z;
        return Zone.fromPoint(untransposePoint(z.center()));
```

```
    }

    /** Перевод абсолютной позиции в относительную */
    public Position transposePosition(Position p){
        transposePoint(p);
        return p;
    }

    /** Перевод относительной позиции в абсолютную */
    public Position untransposePosition(Position p){
        if (this == RIGHT) return LEFT.transposePosition(p);
        if (this == LEFT) return RIGHT.transposePosition(p);
        return transposePosition(p);
    }

    /** Перевод абсолютной позиции в относительную */
    public abstract Point transposePoint(Point p);

    /** Перевод относительной позиции в абсолютную */
    public Point untransposePoint(Point p){
        if (this == RIGHT) return LEFT.transposePoint(p);
        if (this == LEFT) return RIGHT.transposePoint(p);
        return transposePoint(p);
    }
}
```

Zone.java

```
package ru.ifmo.crossroads.common;

import java.awt.Point;
import java.awt.Rectangle;
import java.awt.geom.Area;
import java.awt.geom.Rectangle2D;

import ru.ifmo.crossroads.cross.Cross;

/**
 * Зона
 */
public enum Zone {
    OutOfCross(0,0,0,0),
    OutUpLeft(-Cross.LaneSize, Cross.LaneSize, 0, Cross.LaneSize +
Cross.ExtraSize),
    OutUpRight(0, Cross.LaneSize, Cross.LaneSize, Cross.LaneSize +
Cross.ExtraSize),
    OutLeftUp(-Cross.LaneSize-Cross.ExtraSize, 0, -Cross.LaneSize, Cross.LaneSize),
    UpLeft(-Cross.LaneSize, 0, 0, Cross.LaneSize),
    UpRight(0, 0, Cross.LaneSize, Cross.LaneSize),
    OutRightUp(Cross.LaneSize, 0, Cross.LaneSize+Cross.ExtraSize, Cross.LaneSize),
    OutLeftDown(-Cross.LaneSize-Cross.ExtraSize, -Cross.LaneSize, -Cross.LaneSize,
0),
    DownLeft(-Cross.LaneSize, -Cross.LaneSize, 0, 0),
    DownRight(0, -Cross.LaneSize, Cross.LaneSize, 0),
    OutRightDown(Cross.LaneSize, -Cross.LaneSize, Cross.LaneSize+Cross.ExtraSize,
0),
    OutDownLeft(-Cross.LaneSize, -Cross.LaneSize-Cross.ExtraSize, 0, -
Cross.LaneSize),
    OutDownRight(0, -Cross.LaneSize-Cross.ExtraSize, Cross.LaneSize, -
Cross.LaneSize),
    Middle(-Cross.CentreSize/2, -Cross.CentreSize/2, Cross.CentreSize/2,
Cross.CentreSize/2);

    private Rectangle rect;
```

```
    /**
     * Зона, соответствующая точкам (x1, y1) и (x2, y2)
     */
    private Zone(int x1, int y1, int x2, int y2) {
        this.rect = new Rectangle(x1, y1, x2 - x1, y2 - y1);
    }

    /**
     * @return прямоугольник, занимаемый зоной
     */
    public Rectangle getRect() {
        return rect;
    }

    /**
     * @param x абсцисса
     * @param y ордината
     * @return принадлежит ли точка (x, y) зоне
     */
    public boolean pointInZone(long x, long y) {
        return getArea().contains(x, y);
    }

    /**
     * @return координаты центра зоны
     */
    public Point center() {
        return new Point((int)Math.round(rect.getCenterX()),
(int)Math.round(rect.getCenterY()));
    }

    @Override
    public String toString() {
        return Integer.toString(ordinal());
    }

    /**
     * @return площадку, занимаемую зоной
     */
    public Area getArea() {
        Area a = new Area(new Rectangle2D.Double(rect.x, rect.y, rect.width,
rect.height));
        if (this == UpLeft || this == UpRight || this == DownLeft || this ==
DownRight)
            a.subtract(Middle.getArea());
        return a;
    }

    /**
     * @param p точка
     * @return зону, которой принадлежит точка
     */
    public static Zone fromPoint(Point p) {
        if (Middle.pointInZone(p.x, p.y)) return Middle;
        Zone[] zones = values();
        for (int i = 1; i <= 12; i++)
            if (zones[i].pointInZone(p.x, p.y)) return zones[i];
        return OutOfCross;
    }

    /**
     * @param p положение машины
     * @return зону, где находится машина
```

```
     */
    public static Zone fromPosition(Position p) {
        return fromPoint(p.getCarCenter());
    }
}
```

ConfigurableTimer.java

```
package ru.ifmo.crossroads.core;

import java.util.TimerTask;

import com.evelopers.unimod.core.stateworks.Event;
import com.evelopers.unimod.runtime.ControlledObject;
import com.evelopers.unimod.runtime.EventProvider;
import com.evelopers.unimod.runtime.ModelEngine;
import com.evelopers.unimod.runtime.context.StateMachineContextImpl;
import com.evelopers.unimod.runtime.context.StateMachineContext;

/**
 * Таймер, посылающий сообщение e1 через равные промежутки времени
 */
public class ConfigurableTimer implements EventProvider, ControlledObject{
      /**
       * @unimod.event.descr Прошёл промежуток времени dt
       */
      public static final String E1 = "e1";
      private final static int timerPeriod = 100;
      private java.util.Timer t = null;
      private ModelEngine engine;
      private TimerTask task = null;

      public void init(ModelEngine engine) {
            this.engine = engine;
            t = new java.util.Timer(false);
        task = new TimerTask() {
            public void run() {
                ConfigurableTimer.this.engine.getEventManager().handle(
                        new Event(E1), StateMachineContextImpl.create());
            }
        };
            t.schedule(task, timerPeriod, timerPeriod);
      }

      /** Удалить таймер*/
      public void dispose() {
            t.cancel();
      }

      /** Приостановить таймер */
    public void pause() {
        if (task != null) task.cancel();
    }

    /** Запустить заново таймер */
    public void resume() {
        t.cancel();
        if (task != null) t.schedule(task, timerPeriod, timerPeriod);
    }

    /**
     * @unimod.action.descr Pause
     */
```

```
    public void z1(StateMachineContext context) {
      pause();
    }

    /**
     * @unimod.action.descr Resume
     */
    public void z2(StateMachineContext context) {
      resume();
    }
}
```

CrossEditor.java

```
package ru.ifmo.crossroads.core;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;

import javax.swing.JFileChooser;

import ru.ifmo.crossroads.car.CarCO;
import ru.ifmo.crossroads.car.CarInfo;
import ru.ifmo.crossroads.common.Direction;
import ru.ifmo.crossroads.common.Side;
import com.evelopers.unimod.runtime.ControlledObject;
import com.evelopers.unimod.runtime.context.StateMachineContext;

/**
 * Редактор перекрёстков
 */
public class CrossEditor implements ControlledObject, Serializable {
      private static final long serialVersionUID = 1L;

      /** Массив с информацией о машинах */
      public static CarInfo[] cars = new CarInfo[4];

      /** Информация о выбранной машине */
    public static CarInfo selected = null;

    /** @unimod.action.descr Есть ли установленные машины */
    public boolean x1(StateMachineContext context) {
      for (int i = 0; i < 4; i++)
            if (cars[i].getAutomatId() != 0) return true;
      return false;
    }

    /** @unimod.action.descr Поменять направление машины */
    public void z1(StateMachineContext context) {
        if (selected == null) return;
        selected.setDirection(Direction.values()[(selected.getDirection().ordinal()
+ 1) % 4]);
    }

    /** @unimod.action.descr Поменять автомат у машины */
    public void z2(StateMachineContext context) {
      if (selected == null) return;
```

```
        selected.setAutomatId((selected.getAutomatId() + 1) % (CarCO.CarAutomatNum
+ 1));
    }
    static {
        for (int i = 0; i < 4; i++)
            cars[i] = new CarInfo(Side.values()[i], Direction.FORWARD, 0);
    }

    /**
     * @unimod.action.descr Загрузить конфигурацию перекрёстка из файла
     * @throws ClassNotFoundException
     */
    public void z3(StateMachineContext context) throws ClassNotFoundException {
        final JFileChooser fc = new JFileChooser("tests");
        int returnVal = fc.showOpenDialog(null);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            File file = fc.getSelectedFile();
            System.out.println("Opening: " + file.getName() + ".");

            try {
                ObjectInputStream is = new ObjectInputStream(new
FileInputStream(file.getAbsolutePath()));
                cars = (CarInfo[]) is.readObject();
                is.close();
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                System.out.println("Error during deserialization.");
                e.printStackTrace();
            }
        } else {
            System.out.println("Open command cancelled by user.");
        }
    }

    /** @unimod.action.descr Сохранить конфигурацию перекрёстка в файл */
    public void z4(StateMachineContext context) {
        final JFileChooser fc = new JFileChooser("tests");
        int returnVal = fc.showSaveDialog(null);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            File file = fc.getSelectedFile();
            System.out.println("Saving: " + file.getName() + ".");
            try {
                ObjectOutputStream os = new ObjectOutputStream(new
FileOutputStream(file.getAbsolutePath()));
                os.writeObject(cars);
                os.close();
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                System.out.println("Error during serialization.");
                e.printStackTrace();
            }
        } else {
            System.out.println("Save command cancelled by user.");
        }
    }
}
```

GameLogic.java

```
package ru.ifmo.crossroads.core;
```

```
import ru.ifmo.crossroads.cross.Cross;
import ru.ifmo.crossroads.cross.CrossLogic;

import com.evelopers.unimod.runtime.ControlledObject;
import com.evelopers.unimod.runtime.context.StateMachineContext;

/**
 * Отвечает за процесс моделирования
 */
public class GameLogic implements ControlledObject {

      public static CrossLogic cross = new Cross();

      /** @unimod.action.descr Закончено ли моделирование */
      public boolean x1(StateMachineContext context) {
            return cross.isFinished();
      }

      /** @unimod.action.descr Повисло ли моделирование */
      public boolean x2(StateMachineContext context) {
            return cross.isAlive();
      }

      /** @unimod.action.descr Запустить моделирование */
      public void z1(StateMachineContext context) {
        cross.init(CrossEditor.cars);
            cross.start();
      }

      /** @unimod.action.descr Переместить автомобили */
      public void z2(StateMachineContext context) {
            cross.moveCars();
      }
}
```

Cross.java

```
package ru.ifmo.crossroads.cross;


import ru.ifmo.crossroads.car.CarCO;
import ru.ifmo.crossroads.car.CarData;
import ru.ifmo.crossroads.car.CarInfo;
import ru.ifmo.crossroads.car.MoveableCar;
import ru.ifmo.crossroads.common.Direction;
import ru.ifmo.crossroads.common.Position;
import ru.ifmo.crossroads.common.Side;
import ru.ifmo.crossroads.common.Zone;

import com.evelopers.common.exception.CommonException;
import com.evelopers.unimod.runtime.EventProvider;
import com.evelopers.unimod.runtime.ModelEngine;

/**
 * Сущность перекрёстка
 */
public class Cross implements EventProvider, CrossLogic {
      /** Размеры составных частей перекрёстка */
    public final static int LaneSize = 100,
                            CentreSize = 60,
                            ExtraSize = 100,
                            CarLength = 44,
                            CarWidth = 24;
```

```
    /** Счётчик тиков таймера */
    private int tickCount = 0;

    /** Массив объектов управления, отвечающих за сущности машин */
    private CarCO cars[] = new CarCO[] {null, null, null, null};

    /** @unimod.event.descr Включить сигналы поворота */
    public static final String E2 = "e2";

    /** @unimod.event.descr Начать движение */
    public static final String E4 = "e4";

    /**
     * @param side сторона
     * @return информацию о сигналах поворота машины, находящейся со стороны
<code>s</code>
     */
    public int getCarIndicators(Side side) {
        if (cars[side.ordinal()] == null) return 0;
        return cars[side.ordinal()].getIndicators();
    }

    /**
     * @param side сторона
     * @return информацию о том, куа движетсся машина, находящаяся со стороны
<code>s</code>
     */
    public Direction getCarDirection(Side side){
        if (cars[side.ordinal()] == null) return Direction.NULL;
        return cars[side.ordinal()].getDirection();
    }

    /**
     * Посылает сообщение о том, что машина покинула зону
     * @param s сторона, где находилась машина
     * @param z зона
     */
    public void distributeLeaveZoneMessage(Side s, Zone z){
        log("рассылаю сообщение о том, что машина " + s + " освободила зону "  +
z);
        for (int i = 0; i < 4; i++){
            if (cars[i] == null || i==s.ordinal()) continue;
            Side carSide = cars[i].getSide();
            cars[i].sendLeaveZoneEvent(carSide.transposeSide(s),
carSide.transposeZone(z));
        }
    }

    /**
     * Посылает сообщение о взмахе рукой
     * @param s сторона, где находилась машина
     */
    public void distributeWaveHandMessage(Side s){
        log("рассылаю сообщение " + s + " (подать сигнал от машины " + s + ")");
        for (int i = 0; i < 4; i++){
            if (cars[i]==null || cars[i].getSide()==s) continue;
            cars[i].sendWaveHandEvent();
        }
    }

    /**
     * Посылает сообщение всем машинам
     * @param m посылаемое сообщение
```

```
     */
    private void distributeMessage(String m){
        log("рассылаю сообщение " + m);
        for (CarCO car : cars)
            if (car != null) car.sendEventToCar(m);
    }

    /**
     * Пишет в консоль
     * @param s строчка, которую надо вывести на консоль
     */
    public void log(String s){
        System.out.println("Перекресток: " + s);
    }

    /**
     * Выводит на консоль сообщения о первоначальной обстановке на перекрёстке
     */
    public void logInit(){
        StringBuilder sb = new StringBuilder("начальное расположение машин: \n");
        for (int i = 0; i < 4; i++){
            if (cars[i] != null){
                sb.append("Машина "+ Side.values()[i] + " хочет ехать " +
cars[i].getDirection() + "\n");
            }
        }
        log(sb.toString());
    }

    public void dispose() {}

    public void init(ModelEngine engine) throws CommonException {}

    /**
     * @return количество машин
     */
    public int getCarNum(){
        int s = 0;
        for (int i = 0; i < 4; i++)
            if (cars[i] != null) s++;
        return s;
    }

    /**
     *
     */
    public CarData[] getCars() {
        CarData[] a = new CarData[getCarNum()];
        int j = 0;
        for (int i = 0; i < 4; i++)
            if (cars[i] != null) a[j++] = cars[i];
        return a;
    }

    /**
     *
     */
    public void init(CarInfo carInfo[]) {
        for (int i = 0; i < 4; i++) {
            CarInfo car = carInfo[i];
            if (car.getAutomatId()!= 0) {
                Side side = Side.values()[i];
```

```
                cars[i] = new CarCO(car.getAutomatId(), car.getFileName(), side,
car.getDirection(), this);
            } else cars[i] = null;
        }
        logInit();
    }

    /** Закончено ли моделирование */
    public boolean isFinished() {
        for (MoveableCar car : cars){
            if (car != null && !car.isCrashed() && !car.isFinished()) return false;
        }
        return true;
    }

    /** Не зависло ли моделирование */
    public boolean isAlive(){
        for (MoveableCar car : cars){
            if (car != null && car.isMoving()) return true;
        }
        return false;
    }

    /** Зафиксированы ли столкновения */
    public boolean isCrashed(){
        for (MoveableCar car : cars){
            if (car != null && car.isCrashed()) return true;
        }
        return false;
    }

    /** Проверка на столкновения */
    private void checkCrashes(){
        Position p1, p2;
        for (int i = 0; i < 4; i++){
            if (cars[i]==null || cars[i].isFinished()) continue;
            p1 = cars[i].getPosition();
            for (int j = i + 1; j < 4; j++){
                if (cars[j] == null) continue;
                p2 = cars[j].getPosition();
                if (p1.isCrash(p2)){
                    cars[i].crash();
                    cars[j].crash();
                }
            }
        }
    }

    /** Переместить машины */
    public void moveCars() {
        tickCount++;
        if (tickCount < 4) return;
        int dt = 1;
        for (MoveableCar car : cars){
            if (car != null && !car.isCrashed() && !car.isFinished()) car.move(dt);
        }
        checkCrashes();
    }

    /**
     * @param s сторона
     * @return зону, где находится машина, двигающаяся со стороны <code>s</code>
     */
    public Zone getCarZone(Side s){
```

```
        if (cars[s.ordinal()] != null) return cars[s.ordinal()].getZone();
        return Zone.OutOfCross;
    }

    /** Начало моделирования */
    public void start() {
        tickCount = 0;
        distributeMessage(E2);
        distributeMessage(E4);
    }
}
```

CrossLogic.java

```
package ru.ifmo.crossroads.cross;


import ru.ifmo.crossroads.car.CarData;
import ru.ifmo.crossroads.car.CarInfo;

public interface CrossLogic {
    public CarData[] getCars();
    public void moveCars();
    public void start();
    public boolean isFinished();
    public boolean isAlive();
    public boolean isCrashed();
    public void init(CarInfo cars[]);
}
```

FieldPanel.java

```
package ru.ifmo.crossroads.gui;

import java.awt.AlphaComposite;
import java.awt.Color;
import java.awt.Composite;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.Rectangle;
import java.awt.Point;
import java.awt.Toolkit;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;
import java.awt.geom.*;
import java.awt.image.BufferedImage;
import javax.swing.JPanel;

import ru.ifmo.crossroads.car.CarData;
import ru.ifmo.crossroads.car.CarInfo;
import ru.ifmo.crossroads.common.Direction;
import ru.ifmo.crossroads.common.Side;
import ru.ifmo.crossroads.common.Zone;
import ru.ifmo.crossroads.core.CrossEditor;
import ru.ifmo.crossroads.core.GameLogic;
import ru.ifmo.crossroads.trajectory.Trajectory;

/**
```

```
 * Область для отрисовки перекрёстка
 */
public class FieldPanel extends JPanel{
      private static final long serialVersionUID = 1L;
      /** Ширина окна */
      private static final int SCREEN_WIDTH = 600;

      /** Высота окна */
      private static final int SCREEN_HEIGHT = 600;

      /** Путь до изображений машин */
    private final static String CAR_IMAGE_DIR = "D:/eclipse/Crossroads/res/";

    /** Показывать ли сигналы поворота */
    public static boolean showIndicators = true;

    /** Рисовать ли траектории */
    public static boolean drawTrajectories = false;

    /** Рисовать ли зоны */
    public static boolean drawZones = false;

    /** Путь к изображению перекрёстка */
    private static final String CROSS_IMAGE =
"D:/eclipse/Crossroads/res/crossroads.jpg";

    /** Изображение перекрёстка */
    private Image crossImage = null;

    /** Массив изображений машин */
    private Image[] carImages = new Image[5];

    /** Включён ли режим редактирования */
    public boolean editMode = true;

    /** Преобразование координат */
    public AffineTransform transform = new AffineTransform(3.0 / 2, 0, 0, -3.0 / 2,
300, 300);

    /** Обратное преобразование */
    public AffineTransform backtransform = new AffineTransform(2.0 / 3.0, 0, 0, -
2.0 / 3.0, -200, 200);

    /** Массив цветов машин */
    public static final Color carColors[] = new Color[]{
        new Color(255,155,83),
        new Color(69,179,213),
        new Color(245,255,83),
        new Color(154,214,70)
    };

    /** Внутренняя область окна */
      public final Screen screen;

      /** Обновить статус */
    public void updateStatus() {
        screen.updateStatus();
    }

      public FieldPanel(Screen screen) {
            this.screen = screen;
            Dimension size = new Dimension(SCREEN_WIDTH, SCREEN_HEIGHT);
        carImages[1] = Toolkit.getDefaultToolkit().getImage(CAR_IMAGE_DIR +
"car.jpg");
```

```
        carImages[2] = Toolkit.getDefaultToolkit().getImage(CAR_IMAGE_DIR +
"car_l.jpg");
        carImages[3] = Toolkit.getDefaultToolkit().getImage(CAR_IMAGE_DIR +
"car_r.jpg");
        carImages[4] = Toolkit.getDefaultToolkit().getImage(CAR_IMAGE_DIR +
"car_c.jpg");
        crossImage = Toolkit.getDefaultToolkit().getImage(CROSS_IMAGE);
        this.setSize(size);
            this.setPreferredSize(size);

            this.setDoubleBuffered(true);
        this.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                super.mouseClicked(e);
                switch(e.getButton()){
                case MouseEvent.BUTTON1:
                    ScreenEventProvider.fireEvent(ScreenEventProvider.E7, null);
                    break;
                case MouseEvent.BUTTON3:
                    ScreenEventProvider.fireEvent(ScreenEventProvider.E8, null);
                    break;
                }
            }
        });

        this.addMouseMotionListener(new MouseMotionAdapter(){
            @Override
            public void mouseMoved(MouseEvent e) {
                if (!editMode) return;
                Point p = new Point();
                backtransform.transform(e.getPoint(), p);
                CarInfo c = null;
                for (CarInfo car : CrossEditor.cars) {
                    if (car.getPosition().getCarArea().contains(p.x, p.y)) {
                        c = car;
                    }
                }
                if (CrossEditor.selected != c){
                    CrossEditor.selected = c;
                    repaint();
                }
                updateStatus();
                super.mouseMoved(e);
            }
        });
      }

    public Rectangle toScreen(Rectangle r) {
        return null;
    }

    public Color getCarColor(Side s) {
        return carColors[s.ordinal()];
    }

    public void drawZones(Graphics2D g, CarData[] cars) {
        g.transform(transform);
        g.setColor(Color.gray);
        float alpha = 0.3f;
        int type = AlphaComposite.SRC_OVER;
        AlphaComposite comp = AlphaComposite.getInstance(type, alpha);
        Composite oldComp = g.getComposite();
        g.setComposite(comp);
```

```
        for (Zone z : Zone.values()) {
            if (z != Zone.OutOfCross) {
                g.draw(z.getArea());
            }
        }
        for (CarData c : cars) {
            Zone z = c.getZone();
            if (z != Zone.OutOfCross && c.getAutomatId() != 0) {
                g.setColor(getCarColor(c.getSide()));
                g.fill(z.getArea());
            }
        }
        g.transform(backtransform);
        g.setComposite(AlphaComposite.getInstance(type, 0.4f));
        g.setColor(new Color(30, 60, 220));
        for (Zone z : Zone.values()) {
            if (z != Zone.OutOfCross) {
                Rectangle2D r = z.getArea().getBounds();
                Point p = new Point((int)r.getCenterX(), (int)r.getCenterY());
                transform.transform(p, p);
                g.setFont(new Font("Arial", Font.BOLD, 40));
                FontMetrics fm = g.getFontMetrics();
                String str = Integer.toString(z.ordinal());
                r = fm.getStringBounds(str, null);
                g.drawString(str, (float)(p.x - r.getCenterX()), (float)(p.y -
r.getCenterY()));
            }
        }
        g.setComposite(oldComp);
    }

    /**
     * Рисует траектории
     * @param g графический контекст
     * @param cars массив данных о машинах
     */
    public void drawTrajectories(Graphics g, CarData cars[]) {
        for (CarData car : cars)
           drawTrajectory(g, car, getCarColor(car.getSide()));
    }

    /**
     * Рисует траекторию
     * @param g графический контекст
     * @param car данные о машине
     * @param col цвет
     */
    public void drawTrajectory(Graphics g, CarData car, Color col) {
        g.setColor(col);
        for (double s = car.getLeftDistance(); s >= 0 ; s -= 12) {
            Point d = car.getFuturePosition(s);
            transform.transform(d, d);
            g.fillOval((int) d.x - 3, (int) d.y - 3, 6, 6);
        }
    }

    /**
     * Рисует всевозможные траектории
     * @param g графический контекст
     */
    public void drawAllTrajectories(Graphics2D g) {
        CarData car = new CarInfo(Side.DOWN, Direction.FORWARD, 1);
        g.setColor(getCarColor(car.getSide()));
```

```
        Trajectory tr[] = {Trajectory.BackLong, Trajectory.BackMiddle,
Trajectory.BackShort};
        for (Trajectory traj : tr) {
            for (double s = 0; s < traj.getLen(); s += 12) {
                Point d = traj.getPosition(s);
                transform.transform(d, d);
                g.fillOval((int) d.x - 4, (int) d.y - 4, 8, 8);
            }
        }
        drawCar(g, car, false, false, false);
    }

    /**
     * Отрисовка перекрёстка и машин
     * @param g графический контекст
     */
    public void paint(Graphics g) {
        super.paint(g);
        Graphics2D g2d = (Graphics2D) g;

        g2d.drawImage(crossImage, 0, 0, this);
        CarData cars[];
        if (editMode) cars = CrossEditor.cars;
        else cars = GameLogic.cross.getCars();
        if (drawZones) drawZones(g2d, cars);
        if (!editMode && drawTrajectories) drawTrajectories(g2d, cars);

        for (CarData car : cars)
            drawCar(g2d, car, showIndicators && !editMode, editMode, editMode &&
(car == CrossEditor.selected));
    }

    /**
     * Рисует направления движений
     * @param g графический контекст
     * @param width ширина
     * @param height высота
     * @param dir направление
     */
    public void drawDirection(Graphics g, int width, int height, Direction dir) {
        int arrowSize = 6;
        g.setColor(Color.black);
        if (dir == Direction.FORWARD) {
            g.drawLine(width / 2, height, width / 2, 0);
            g.drawLine(width / 2 - arrowSize, arrowSize, width / 2, 0);
            g.drawLine(width / 2 + arrowSize, arrowSize, width / 2, 0);
        } else if (dir == Direction.LEFT) {
            g.drawArc(-width / 2, arrowSize, width, (height - arrowSize) * 2, 0,
90);
            g.drawLine(0, arrowSize, arrowSize, arrowSize * 2);
            g.drawLine(0, arrowSize, arrowSize, 0);
        } else if (dir == Direction.RIGHT) {
            g.drawArc(width / 2, arrowSize, width, (height - arrowSize) * 2, 90,
180);
            g.drawLine(width, arrowSize, width - arrowSize, arrowSize * 2);
            g.drawLine(width, arrowSize, width - arrowSize, 0);
        } else if (dir == Direction.BACK) {
            g.drawLine(width / 2, height, width / 2, height - arrowSize);
            g.drawArc(arrowSize, 0, width / 2 - arrowSize, (height - arrowSize) *
2, 0, 180);
            g.drawLine(arrowSize, height-arrowSize, 0, height-arrowSize*2);
            g.drawLine(arrowSize, height - arrowSize, arrowSize * 2, height -
arrowSize * 2);
        }
```

```
    }

    /**
     * Отрисовка машины
     * @param g графический контекст
     * @param car данные о машине
     * @param drawIndicators рисовать ли индикаторы
     * @param drawDirection рисовать ли направления
     * @param selected выбрана ли машина
     */
      public void drawCar(Graphics2D g, CarData car, boolean drawIndicators,
boolean drawDirection, boolean selected) {
        Image mImage = null;
        Point p = car.getPosition();
        if (!drawIndicators) mImage = carImages[1];
        else if (car.blinkingLeft() && !car.blinkingRight())
            mImage = carImages[2];
        else if (car.blinkingRight() && !car.blinkingLeft())
            mImage = carImages[3];
        else if (car.blinkingLeft() && car.blinkingRight())
            mImage = carImages[4];
        else
            mImage = carImages[1];
        int mWidth = mImage.getWidth(this),
            mHeight = mImage.getHeight(this);
        int arrowSpace = car.getAutomatId() == 0 ? 0 : 40;
        BufferedImage bim = new BufferedImage(mWidth, mHeight + arrowSpace,
BufferedImage.TYPE_INT_ARGB);
        Graphics2D gbuf = (Graphics2D) bim.createGraphics();
        if (car.getAutomatId() == 0) {
            gbuf.setColor(Color.gray);
            gbuf.fillRect(0, 0, mWidth, mHeight);
        } else {
            gbuf.setComposite(AlphaComposite.Clear);
            gbuf.fillRect(0, 0, mWidth, arrowSpace + mHeight);
            gbuf.setComposite(AlphaComposite.Src);
            if (drawDirection)
                drawDirection(gbuf, mWidth, arrowSpace, car.getDirection());
            gbuf.drawImage(mImage, 0, arrowSpace, mWidth, mHeight, null);
            gbuf.setColor(getCarColor(car.getSide()));
            gbuf.fillRoundRect(10, 22 + arrowSpace, 24, 47, 14, 16);
            gbuf.setColor(Color.black);
            gbuf.drawRoundRect(10, 22 + arrowSpace, 24, 47, 14, 16);
            gbuf.setColor(Color.black);
            gbuf.setFont(new Font("Arial", Font.BOLD, 30));
            gbuf.drawString(Integer.toString(car.getAutomatId()), mWidth / 3, 3 *
mHeight / 4 + arrowSpace);
        }

            AffineTransform xform = new AffineTransform();
        transform.transform(p, p);
        xform.translate(p.x, p.y);
        xform.scale(0.9, 0.9);
        if(selected) xform.scale(1.1, 1.1);
            xform.translate(-mWidth / 2, -mHeight / 2 - arrowSpace);
            xform.rotate(Math.PI / 2 - car.getPosition().angle, mWidth / 2, mHeight
/ 2 + arrowSpace);

            g.drawImage(bim, xform, this);
      }
}
```

Screen.java

```
package ru.ifmo.crossroads.gui;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.AbstractAction;
import javax.swing.Action;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JOptionPane;
import javax.swing.JTextArea;
import javax.swing.JToolBar;

import ru.ifmo.crossroads.car.CarInfo;
import ru.ifmo.crossroads.core.CrossEditor;
import com.evelopers.unimod.runtime.ControlledObject;
import com.evelopers.unimod.runtime.context.StateMachineContext;

/**
 * Отвечает за GUI и логику при воздействиях пользователя
 */
public class Screen extends JFrame implements ControlledObject {
      private static final long serialVersionUID = 1L;
      public final FieldPanel fieldPanel;
      public final Action startGameAction;
      public final Action stopGameAction;
      public final Action openAction;
    public final Action saveAction;
    public final JCheckBoxMenuItem miTrajectories;
    public final JCheckBoxMenuItem miZones;
    public final Action drawTrajectories;
    public final Action drawZones;
    public final JTextArea statusBar;
    public final String HINT_STATUS = "Для добавления машинки щелкните на сером
прямоугольнике левой кнопкой мыши. \nДля изменения номера автомата, управляющего
машинкой, щелкните ещё раз. \nДля изменения направления движения машинки щелкните
на ней правой книопкой мыши.";
    public final String AUTO1_STATUS = "Машинка с номером 1 ездит по всем правилам
дорожного движения";
    public final String AUTO2_STATUS = "Машинка с номером 2 никогда не включает
указатели поворота";
    public final String AUTO3_STATUS = "Машинка с номером 3 никого не пропускает";
    public final String AUTO0_STATUS = "Щелкните здесь для добавления машинки";
    public final String PAUSE_STATUS = "Моделирование приостановлено";
    public final String MODELING_STATUS = "Машинки едут...";
    public final String BUZZ_STATUS = "МОДЕЛИРОВАНИЕ ЗАВИСЛО! Это значит, что все
машинки приняли решение стоять на месте.";
    public final String CRASH_STATUS = "ПРОИЗОШЛА АВАРИЯ!!! 2 или более машин
столкнулись";

    /**
     * Установить сообщение в статус
     * @param s строка с сообщением
     */
    public void setStatus(String s) {
        statusBar.setText(s);
    }
```

```
    /**
     * Обновить статус
     */
    public void updateStatus() {
        String s = null;
        if (!fieldPanel.editMode) {
            return;
        } else {
            CarInfo c = CrossEditor.selected;
            if (c != null) {
                int k = c.getAutomatId();
                switch(k) {
                      case 0: s = AUTO0_STATUS; break;
                      case 1: s = AUTO1_STATUS; break;
                      case 2: s = AUTO2_STATUS; break;
                      case 3: s = AUTO3_STATUS; break;
                }
            } else s = HINT_STATUS;
        }
        if (s != null) statusBar.setText(s);
    }

      public Screen() {
            super("Моделирование разъезда машин на перекрестке");
            this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
          fieldPanel = new FieldPanel(this);
            this.getContentPane().add(fieldPanel, BorderLayout.CENTER);
            this.setSize(600, 600 + 50 + 30 + 50);
            this.setLocation(0, 0);
            this.setResizable(false);
            this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
            this.addWindowListener(new WindowAdapter() {
                  public void windowClosing(WindowEvent e) {
                        ScreenEventProvider.fireEvent(ScreenEventProvider.E4,
null);
                  }
            });

            JMenuBar menu = new JMenuBar();
            JMenu mFile, mModeling, mOptions;
            mFile = new JMenu(new FileAction());
            openAction = new OpenAction();
            mFile.add(openAction);
        saveAction = new SaveAction();
        mFile.add(saveAction);
            mFile.add(new ExitAction());

        mModeling = new JMenu("Моделирование");
            startGameAction = new StartPauseContinueAction();
        mModeling.add(startGameAction);
            stopGameAction = new StopGameAction();
        mModeling.add(stopGameAction);

        mOptions = new JMenu("Опции");

        drawTrajectories = new ShowTrajectoriesAction();
        miTrajectories = new JCheckBoxMenuItem(drawTrajectories);
        miTrajectories.setSelected(FieldPanel.drawTrajectories);
        mOptions.add(miTrajectories);

        drawZones = new DrawZonesAction();
        miZones = new JCheckBoxMenuItem(drawZones);
        miZones.setSelected(FieldPanel.drawZones);
```

```
        mOptions.add(miZones);

        menu.add(mFile);
            menu.add(mModeling);
        menu.add(mOptions);

            JToolBar jtb = new JToolBar();
            jtb.add(startGameAction);
            jtb.add(stopGameAction);
            jtb.setFloatable(false);
            this.add(jtb, BorderLayout.NORTH);

            this.setJMenuBar(menu);
        statusBar = new JTextArea(HINT_STATUS);
        statusBar.setEditable(false);
        statusBar.setPreferredSize(new Dimension(10, 50));
        this.add(statusBar, BorderLayout.SOUTH);
            this.setVisible(true);
    }

    private class FileAction extends AbstractAction {
            private static final long serialVersionUID = 1L;

            public FileAction() {
                  putValue(Action.NAME, "Файл");
                  putValue(Action.MNEMONIC_KEY, new Integer('F'));
            }

            public void actionPerformed(ActionEvent e) {}
    }

    private class OpenAction extends AbstractAction {
            private static final long serialVersionUID = 1L;

            public OpenAction() {
                  putValue(Action.NAME, "Открыть");
                  putValue(Action.MNEMONIC_KEY, new Integer('O'));
            }

            public void actionPerformed(ActionEvent e) {
                  ScreenEventProvider.fireEvent(ScreenEventProvider.E5, null);
            }
    }

private class SaveAction extends AbstractAction {
        private static final long serialVersionUID = 1L;

        public SaveAction() {
            putValue(Action.NAME, "Сохранить");
            putValue(Action.MNEMONIC_KEY, new Integer('S'));
        }

        public void actionPerformed(ActionEvent e) {
            ScreenEventProvider.fireEvent(ScreenEventProvider.E6, null);
        }
}

private class ShowTrajectoriesAction extends AbstractAction {
        private static final long serialVersionUID = 1L;

        public ShowTrajectoriesAction() {
            putValue(Action.NAME, "Отображать траектории");
            putValue(Action.MNEMONIC_KEY, new Integer('T'));
        }
```

```
        public void actionPerformed(ActionEvent e) {
            FieldPanel.drawTrajectories = !FieldPanel.drawTrajectories;
            miTrajectories.setSelected(FieldPanel.drawTrajectories);
            repaint();
        }
    }

    private class DrawZonesAction extends AbstractAction {
        private static final long serialVersionUID = 1L;

        public DrawZonesAction() {
            putValue(Action.NAME, "Отображать зоны");
            putValue(Action.MNEMONIC_KEY, new Integer('Z'));
        }

        public void actionPerformed(ActionEvent e) {
            FieldPanel.drawZones = !FieldPanel.drawZones;
            miZones.setSelected(FieldPanel.drawZones);
            repaint();
        }
    }

      private class ExitAction extends AbstractAction {
            private static final long serialVersionUID = 1L;

            public ExitAction() {
                  putValue(Action.NAME, "Выход");
                  putValue(Action.SHORT_DESCRIPTION, "Выход");
                  putValue(Action.MNEMONIC_KEY, new Integer('x'));
            }

            public void actionPerformed(ActionEvent e) {
                  ScreenEventProvider.fireEvent(ScreenEventProvider.E4, null);
            }
      }

      private class StartPauseContinueAction extends AbstractAction {
            private static final long serialVersionUID = 1L;
            public static final String PAUSE = "Пауза";
            public static final String CONTINUE = "Продолжить";
        public static final String START = "Старт";

            public StartPauseContinueAction() {
                  this.putValue(Action.NAME, PAUSE);
                  this.putValue(Action.SHORT_DESCRIPTION,
"Старт/Пауза/Продолжить");
                  this.putValue(Action.MNEMONIC_KEY, new Integer('s'));
                  this.setEnabled(false);
            }

            public void actionPerformed(ActionEvent e) {
                  ScreenEventProvider.fireEvent(ScreenEventProvider.E2, null);
            }
      }

      private class StopGameAction extends AbstractAction {
            private static final long serialVersionUID = 1L;

            public StopGameAction() {
                  putValue(Action.NAME, "Стоп");
                  putValue(Action.SHORT_DESCRIPTION, "Стоп");
                  putValue(Action.MNEMONIC_KEY, new Integer('t'));
                  setEnabled(false);
```

```
        }

        public void actionPerformed(ActionEvent e) {
              ScreenEventProvider.fireEvent(ScreenEventProvider.E3, null);
        }
  }


/**
 * @unimod.action.descr Перерисовать окно
 */
public void z1(StateMachineContext context) {
    repaint();
}

  /**
   * @unimod.action.descr Вывести сообщение о завершении моделирования
   */
  public void z11(StateMachineContext context) {
    JOptionPane.showMessageDialog(this, "Моделирование завершено!");
  }

/**
 * @unimod.action.descr Включить сигналы поворота
 */
public void z12(StateMachineContext context) {
    FieldPanel.showIndicators = !FieldPanel.showIndicators;
}

  /**
   * @unimod.action.descr Закрыть окно
   */
  public void z13(StateMachineContext context) {
        this.dispose();
  }

/**
 * @unimod.action.descr Обновить статус-бар
 */
public void z200(StateMachineContext context) {
  fieldPanel.updateStatus();
}

/**
 * @unimod.action.descr Переключиться в режим редактирования
 */
public void z211(StateMachineContext context) {
    fieldPanel.editMode = true;
}

/**
 * @unimod.action.descr Переключиться в режим моделирования
 */
public void z212(StateMachineContext context) {
  fieldPanel.editMode = false;
}

/**
 * @unimod.action.descr Показать статус при паузе моделирования
 */
public void z222(StateMachineContext context) {
  setStatus(PAUSE_STATUS);
}
```

```
    /**
     * @unimod.action.descr Показать статус процесса моделирования
     */
    public void z221(StateMachineContext context) {
        setStatus(MODELING_STATUS);
    }

    /**
     * @unimod.action.descr Показать статус при зависании моделирования
     */
    public void z223(StateMachineContext context) {
        setStatus(BUZZ_STATUS);
    }

    /**
     * @unimod.action.descr Показать статус при столкновении машин
     */
    public void z224(StateMachineContext context) {
      setStatus(CRASH_STATUS);
    }

      /**
       * @unimod.action.descr Запретить нажатия кнопки Старт/Пауза/Продолжить
       */
      public void z300(StateMachineContext context) {
            this.startGameAction.setEnabled(false);
      }

      /**
       * @unimod.action.descr Разрешить нажатия кнопки Старт/Пауза/Продолжить
       */
      public void z301(StateMachineContext context) {
            this.startGameAction.setEnabled(true);
      }

      /**
       * @unimod.action.descr Поменять название кнопки Старт/Пауза/Продолжить на
Пауза
       */
      public void z302(StateMachineContext context) {
            this.startGameAction.putValue(Action.NAME,
                        StartPauseContinueAction.PAUSE);
      }

      /**
       * @unimod.action.descr Поменять название кнопки Старт/Пауза/Продолжить на
Продолжить
       */
      public void z303(StateMachineContext context) {
            this.startGameAction.putValue(Action.NAME,
                        StartPauseContinueAction.CONTINUE);
      }

    /**
     * @unimod.action.descr Поменять название кнопки Старт/Пауза/Продолжить на
Старт
     */
    public void z304(StateMachineContext context) {
        this.startGameAction.putValue(Action.NAME,
                StartPauseContinueAction.START);
    }


      /**
```

```
	 * @unimod.action.descr Запретить нажатия кнопки Стоп
	 */
	public void z310(StateMachineContext context) {
		this.stopGameAction.setEnabled(false);
	}

	/**
	 * @unimod.action.descr Разрешить нажатия кнопки Стоп
	 */
	public void z311(StateMachineContext context) {
		this.stopGameAction.setEnabled(true);
	}

	/**
	 * @unimod.action.descr Запретить выбор пункта меню Открыть
	 */
	public void z320(StateMachineContext context) {
		this.openAction.setEnabled(false);
	}

	/**
	 * @unimod.action.descr Разрешить нажатия кнопки Открыть
	 */
	public void z321(StateMachineContext context) {
		this.openAction.setEnabled(true);
	}
}
```

ScreenEventProvider.java

```
package ru.ifmo.crossroads.gui;

import com.evelopers.common.exception.CommonException;
import com.evelopers.unimod.core.stateworks.Event;
import com.evelopers.unimod.runtime.EventProvider;
import com.evelopers.unimod.runtime.ModelEngine;
import com.evelopers.unimod.runtime.context.Parameter;
import com.evelopers.unimod.runtime.context.StateMachineContextImpl;

/**
 * Поставщик событий, вызванных действиями пользователя
 */
public class ScreenEventProvider implements EventProvider {
	/** @unimod.event.descr Нажата кнопка Старт/Пауза/Продолжить */
	public static final String E2 = "e2";

	/** @unimod.event.descr Нажата кнопка Стоп */
	public static final String E3 = "e3";

	/** @unimod.event.descr Выбор пункта меню Выход */
	public static final String E4 = "e4";

	/** @unimod.event.descr Нажата кнопка Открыть */
	public static final String E5 = "e5";

	/** @unimod.event.descr Выбор пункта меню Сохранить */
    public static final String E6 = "e6";

    /** @unimod.event.descr Нажата левая кнопка мыши */
    public static final String E7 = "e7";

    /** @unimod.event.descr Нажата правая кнопка мыши */
    public static final String E8 = "e8";
```

```
      private static ModelEngine engine;

      public void init(ModelEngine engine) throws CommonException {
            if (ScreenEventProvider.engine != null)
                  return;
            ScreenEventProvider.engine = engine;
      }

    public static void fireEvent(String eventName, Parameter parameter) {
        if (engine != null) {
            if (parameter == null) {
                engine.getEventManager().handle(new Event(eventName),
                        StateMachineContextImpl.create());
            } else {
                engine.getEventManager().handle(
                        new Event(eventName, new Parameter[] { parameter }),
                        StateMachineContextImpl.create());
            }
        }
    }

      public void dispose() {}
}
```

LinePart.java

```
package ru.ifmo.crossroads.trajectory;

import ru.ifmo.crossroads.common.Position;

/**
 * Линейная часть траектории
 */
public class LinePart implements Part {
    /** Координаты начала и конца отрезка */
      private long x1, y1, x2, y2;

      /** Длина */
    private double len;

    /**
     * @param x1 абсцисса начала отрезка
     * @param y1 ордината начала отрезка
     * @param x2 абсцисса конца отрезка
     * @param y2 ордината конце отрезка
     */
    public LinePart(long x1, long y1, long x2, long y2) {
        super();
        this.x1 = x1;
        this.y1 = y1;
        this.x2 = x2;
        this.y2 = y2;
        len = Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
    }

    /**
     * @return длину данной части траектории
     */
    public double getLen() {
        return len;
    }
```

```
    /**
     * @return угол наклона отрезка
     */
    private double getAngle() {
        return Math.atan2(y2 - y1, x2 - x1);
    }

    /**
     * @param s пройденный путь
     * @return положение после прохождения пути <code>s</code>
     */
    public Position getPosition(double s) {
        return new Position(Math.round(x1 + (x2 - x1) * (s / len)),
                Math.round(y1 + (y2 - y1) * (s / len)),
                getAngle());
    }

    /**
     * @return конечное положение
     */
    public Position getLastPosition() {
        return new Position(x2, y2, getAngle());
    }
}
```

Part.java

```
package ru.ifmo.crossroads.trajectory;

import ru.ifmo.crossroads.common.Position;

/**
 * Часть траектории. Вся траектория состоит из списка объектов типа Part
 */
public interface Part {
      /**
     * @return длину данной части траектории
     */
    public double getLen();

    /**
     * @param s пройденный путь
     * @return положение после прохождения пути <code>s</code> от начала траектории
     */
    public Position getPosition(double s);

    /**
     * @return конечное положение
     */
    public Position getLastPosition();
}
```

RoundPart.java

```
package ru.ifmo.crossroads.trajectory;

import ru.ifmo.crossroads.common.Position;

/**
 * Часть траектории, описываемая окружностью
 */
public class RoundPart implements Part {
```

```
      /** Координаты центра, радиус, начальный и конечный углы окружности */
    private double a1, a2, radius, x1, y1;

    /** Длина траектории */
    private double len;

    /**
     * @param a1 абсцисса центра
     * @param a2 ордината центра
     * @param radius радиус
     * @param x1 начальный угол
     * @param y1 конечный угол
     */
    public RoundPart(double a1, double a2, double radius, double x1, double y1) {
        super();
        this.a1 = a1;
        this.a2 = a2;
        this.radius = radius;
        this.x1 = x1;
        this.y1 = y1;
        len = radius*(Math.abs(a2 - a1));
    }

    /**
     * @return длину данной части траектории
     */
    public double getLen() {
        return len;
    }

    /**
     * @return угол наклона
     */
    private double getAngle(double s) {
        return a1 + s / radius * (a1 < a2 ? 1 : -1);
    }

    /**
     * @param s пройденный путь
     * @return положение после прохождения пути <code>s</code>
     */
    public Position getPosition(double s) {
        double angle = getAngle(s);
        return new Position(Math.round(x1 + radius * Math.cos(angle)),
                Math.round(y1 + radius * Math.sin(angle)),
                angle + Math.PI / 2 * (a1 < a2 ? 1 : -1));
    }

    /**
     * @return конечное положение
     */
    public Position getLastPosition() {
        return getPosition(getLen());
    }
}
```

Trajectory.java

```
package ru.ifmo.crossroads.trajectory;

import ru.ifmo.crossroads.common.Position;

/**
```

```
 * Траектории. Определены заранее.
 */
public enum Trajectory {
    LeftShort{
        protected void init() {
            parts = new Part[] {
                    new RoundPart(0, Math.PI / 4, 170, -120, -150),
                    new LinePart(2, -30, -30, 2),
                    new RoundPart(Math.PI / 4, Math.PI / 2, 170, -150, -120),
                    new LinePart(-150, 50, -250, 50)
                    };
        }
    },
    LeftMiddle{
        protected void init() {
            parts = new Part[] {
                    new LinePart(50, -150, 50, -120),
                    new RoundPart(0, Math.PI / 2, 170, -120, -120),
                    new LinePart(-120, 50, -250, 50)
                    };
        }
    },
    LeftLong{
        protected void init() {
            parts = new Part[] {
                    new LinePart(50, -150, 50, 0),
                    new RoundPart(0, Math.PI / 2, 50, 0, 0),
                    new LinePart(0, 50, -250, 50)
                    };
        }
    },
    Right{
        protected void init() {
            parts = new Part[] {
                    new LinePart(50, -150, 50, -100),
                    new RoundPart(Math.PI, Math.PI / 2, 50, 100, -100),
                    new LinePart(100, -50, 250, -50)
                    };
        }
    },
    Forward{
        protected void init() {
            parts = new Part[] {
                    new LinePart(50, -150, 50, 250)
                    };
        }
    },
    BackShort{
        protected void init() {
            parts = new Part[] {
                    new RoundPart(0, Math.PI / 4, 170, -120, -150),
                    new LinePart(2, -30, -30, 2),
                    new RoundPart(Math.PI / 4, 5 * Math.PI / 4, 20 * Math.sqrt(2),
-50, -18),
                    new RoundPart(Math.PI / 4, 0, 68, -118, -86),
                    new LinePart(-50, -86, -50, -250)
                    };
        }
    },
    BackMiddle{
        protected void init() {
            parts = new Part[] {
                    new LinePart(50, -150, 50, -120),
                    new RoundPart(0, Math.PI / 4, 170, -120, -120),
```

```
                    new RoundPart(Math.PI / 4, 5 * Math.PI / 4, Math.sqrt(3200), -
40, -40),
                    new RoundPart(Math.PI / 4, 0, 102, -152, -152),
                    new LinePart(-50, -152, -50, -250)
                    };
        }
    },
    BackLong{
        protected void init() {
            parts = new Part[] {
                    new LinePart(50, -150, 50, 0),
                    new RoundPart(0, Math.PI, 50, 0, 0),
                    new LinePart(-50, 0, -50, -250)
                    };
        }
    };

    /** Части траекторий */
    protected Part[] parts;

    /** Длина */
    protected double len;

    abstract void init();

    private Trajectory() {
        init();
        len = 0;
        for (Part p : parts) len += p.getLen();
    }

    /**
     * @param s пройденный путь
     * @return положение после прохождения пути <code>s</code> от начала траектории
     */
    public Position getPosition(double s) {
        if (s < 0) s = 0;
        for (int i = 0; i < parts.length; i++) {
            if (s < parts[i].getLen()) return parts[i].getPosition(s);
            s -= parts[i].getLen();
        }
        return getLastPosition();
    }

    /**
     * @return конечное положение
     */
    public Position getLastPosition() {
        return parts[parts.length-1].getLastPosition();
    }

    /**
     * @return длину данной части траектории
     */
    public double getLen() {
        return len;
    }
}
```