Санкт-Петербургский государственный университет
информационных технологий, механики и оптики
Факультет «Информационные технологии и программирование»
Кафедра «Компьютерные технологии»

К.В. Егоров, П.М. Райков

Описание проектирования и реализации игры «Побег» на основе автоматного программирования

Проектная документация

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

Санкт-Петербург
2007

Оглавление

Введение

Цель данной работы показать применение автоматного программирования для реализации специфического класса мультиагентных систем. Программа была спроектирована с помощью инструментального средства *UniMod* (http://unimod.sourceforge.net), который представляет собой подключаемую программу для среды разработки *Eclipse* (http://eclipse.org), реализуя в ней поддержку таких концепций как «Исполняемый *UML*» в виде генерирования кода на основе *UniMod*-моделей. Построение *UniMod*-моделей является одним из способов быстрого и простого проектирования программ на основе *SWITCH*-технологии [1].

1. Описание проекта

1.1. Функциональность

«Побег» – игра для одного игрока, задача которого состоит в том, чтобы уехать на своей машине от машин полицейских. В данной работе использование инструментального средства *UniMod* вызвано тем, что хитроумную логику компьютерных полицейских, чьей задачей является поимка игрока, удобно реализовывать с помощью автоматов. Таким образом, в работе была создана модель автоматов, применимых в условиях мультиагентности. Следовательно, можно, создав несколько таких полицейских, выпустить их вслед за преступником, и они, постоянно обмениваясь сообщениями, смогут, используя сообща все свои силы, поймать его.

1.2. Игровой мир

Игровое пространство представляет собой двумерную бесконечную плоскость (пустыню), «заселенную» редкой растительностью и многочисленными полицейскими. В начале раунда игрок является представителем «плохих парней» с ценным грузом где-то на бескрайних просторах этой пустыни.

После этого сразу же начинается погоня, полицейские машины постоянно появляются из-за пределов экрана и норовят поймать вас. Игра заканчивается, когда какая-нибудь полицейская машина врезается в машину игрока. При этом полицейские из-за своей страсти к быстрым победам и скромному водительскому стажу периодически сталкиваются друг с другом и взрываются.

К сожалению, для игрока, полиция не замечает своих потерь, и уже через несколько секунд на смену погибшим машинам приходят новые герои-полицейские, неожиданно появляющиеся прямо на пути игрока. Также отметим, что игрок может уехать от какой-то машины, и она, скрывшись за пределами экрана, прекратит погоню. Присутствие кактусов и камней на экране обусловлено исключительно эстетическим вкусом разработчиков, которые хотели внести разнообразие в игровой процесс. При этом если любая машина переедет их, то это никак не отразится на ее дальнейшем движении.

1.3. Интерфейс

Программа состоит из одного *SWING*-окна (рис. 1). Главное окно программы разделено на две части: панель управления и игровую панель. На панели управления находятся кнопки управления игрой («запуск», «пауза», «стоп»), регулятор скорости игры и текстовое поле с информацией о текущем счете игрока. На игровой панели – графическое поле, в котором отображается текущее состояние игры. Все картинки, используемые в программе, сделаны с помощью цифрового фотоаппарата. Машинки, кактусы и камни были сфотографированы на белом фоне и потом перенесены на компьютер. Анимация для взрыва была взята в виде готового *gif*-файла, найденного в Интернете.

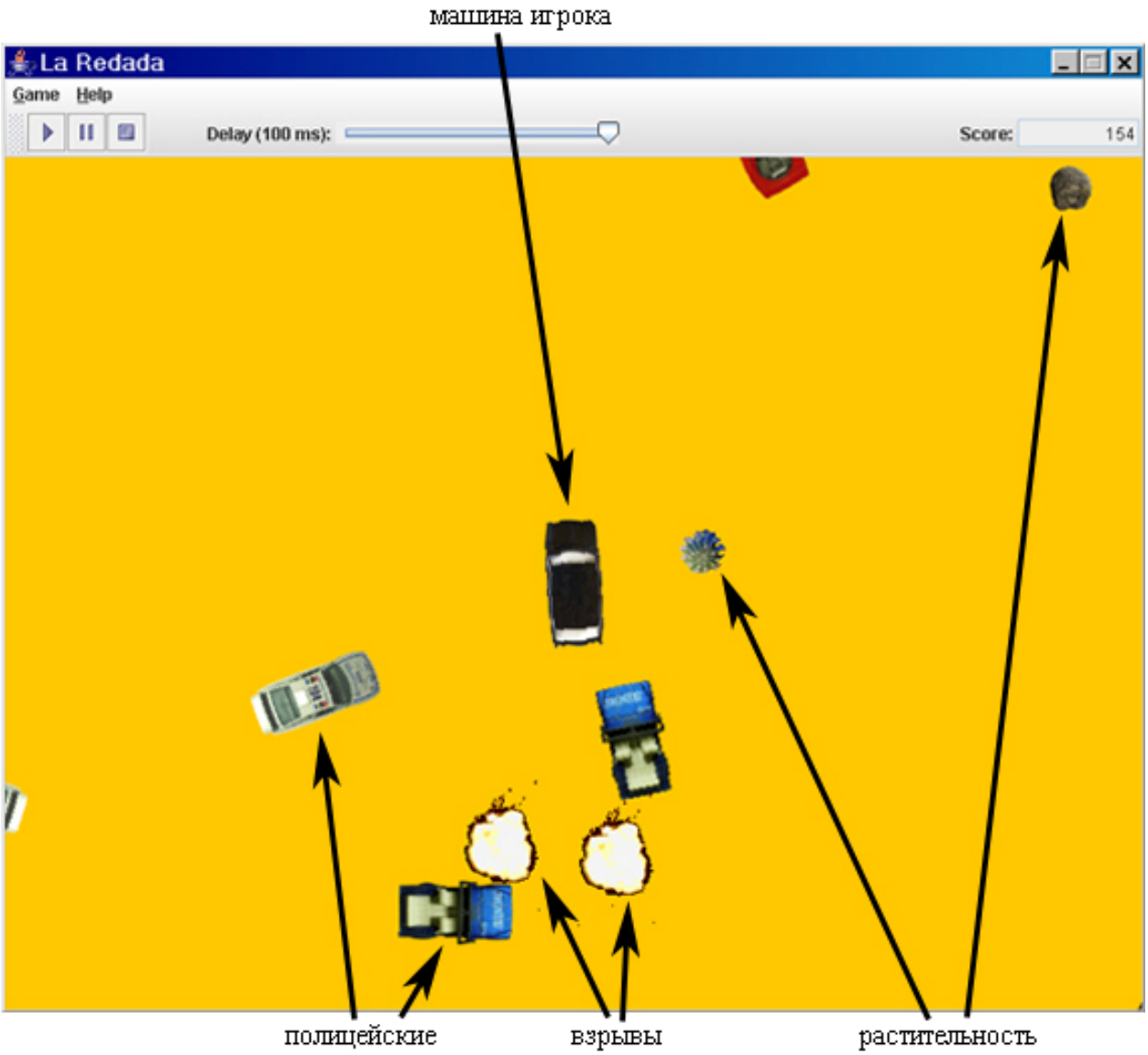


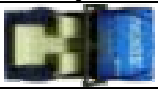
Рис. 1. Вид окна приложения с пояснениями

Отметим, что для игры требовалось придумать звучный вариант названия на одном из иностранных языков из-за того, что был создан не русскоязычный интерфейс. Так как действие игры по задумке авторов происходит в Мексике, то решено было взять испанский вариант слова «побег» – «la redada».

Управление машиной игрока производится мышкой – просто нужно поместить курсор в желаемую сторону направления движения машины.

В игре существует несколько моделей поведения полицейских, реализованных с помощью *UniMod*-моделей, им соответствуют свои изображения, перечисленные в табл. 1.

Таблица 1

***UniMod*-модель**	**Изображение**
CarModel1	
CarModel2	
CarModel3	
CarModel4	

1.4. Физическая модель

Действие игры осуществляется по таймеру. Каждые несколько миллисекунд он генерирует событие, по которому каждый объект перемещается в соответствующем направлении и с соответствующей скоростью, которая также изменяется согласно текущему ускорению.

Машины имеют три атрибута: *maxSpeed*, *maxAngle*, *acceleration*. Для каждой модели создается своя машина, параметры которой задаются в конфигурационном файле (Приложение 1). В табл. 2 указаны атрибуты машины и их смысл.

Таблица 2

Атрибут	**Семантика**	**Единицы измерения**
maxSpeed	Максимальная скорость машины за один такт таймера	Пиксель
maxAngle	Максимальный угол поворота машины за один такт таймера	Радианы
acceleration	Ускорение за один такт таймера	Пиксель за один такт таймера

2. Проектирование

Проектирование программы было выполнено при помощи инструментального средства *UniMod*, которое позволяет при помощи визуальных средств вынести всю логику программы в автоматы и разбить все остальные классы на два основных типа: поставщики событий и управляемые объекты (также используются вспомогательные классы). Всего было построено пять *UniMod*-моделей: модель игрового мира (*GameModel*) и четыре модели управления машинами полицейских (*CarModel1*, *CarModel2*, *CarModel3*, *CarModel4*).

2.1. Модель игрового мира — *GameModel*

2.1.1. Схема связей

Схема связей в виде диаграммы классов представлена на рис. 2. На ней представлены три поставщика событий (*p1*, *p2*, *p3*), три автомата (*A1*, *A2*, *A3*) и четыре объекта управления (*o1*, *o2*, *o3*, *o4*).

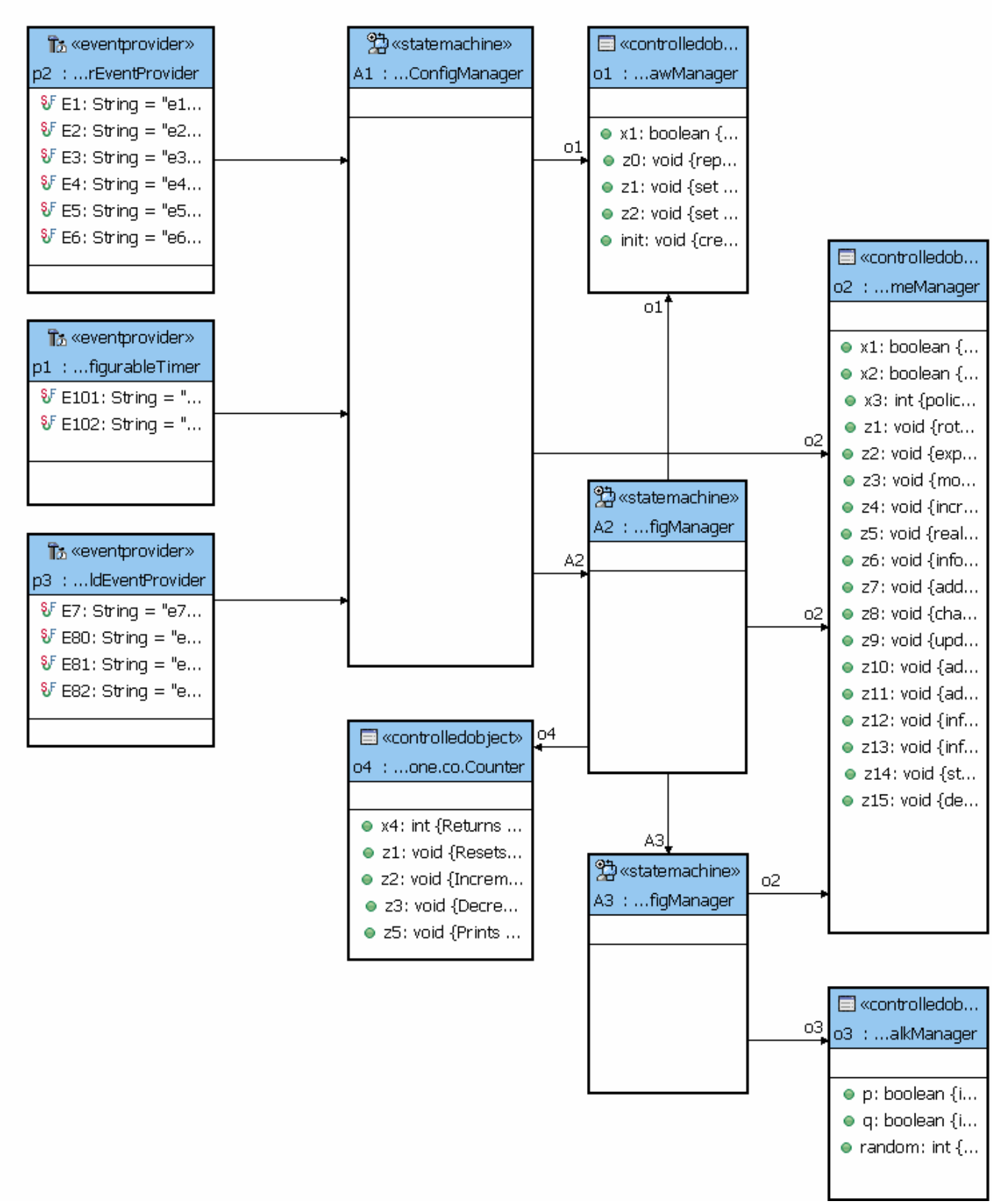


Рис. 2. Схема связей *GameModel*

2.1.2. Поставщики событий

2.1.2.1. Поставщик событий таймера (*p1*)

События, относящиеся к этому поставщику, инициируются таймером игровой модели и описаны в табл. 3.

Таблица 3. События поставщика событий *ConfigurableTimer*

Событие	Описание	Параметр
e101	Произошло событие таймера. Возникает раз в *GameData.sleepAmount* миллисекунд (это значение изменяется от 10ms до 100ms)	Нет параметра
e102	Произошло событие таймера. Возникает при каждом тридцатом событии *e101*	Нет параметра

2.1.2.2. Поставщик событий пользователя (*p2*)

События, относящиеся к этому поставщику, инициируются пользователем и описаны в табл. 4.

Таблица 4. События поставщика событий *UserEventProvider*

Событие	Описание	Параметр
e1	Начать игру	Нет параметра
e2	Приостановить игру	Нет параметра
e3	Остановить игру	Нет параметра
e4	Прервать выполнение автомата	Нет параметра
e5	Программа получила фокус	Нет параметра
e6	Программа потеряла фокус	Нет параметра

2.1.2.3. Поставщик событий игровой модели (*p3*)

События, относящиеся к этому поставщику, инициируются игровой моделью и описаны в табл. 5.

Таблица 5. События поставщика событий *WorldEventProvider*

Событие	Описание	Параметр
e80	Полицейские машины столкнулись	Нет параметра
e81	Вернуться в состояние выполнения ходов игры после изменения количества машин противника, принимающих участие в преследовании	Нет параметра
e82	Совершить вероятностный переход в цепи Маркова	Нет параметра

2.1.3. Объекты управления

2.1.3.1. Объект управления изображением игрового мира (*o1*)

Этот объект управления предоставляет входные и выходные воздействия, которые относятся к отображению игрового мира на игровую панель (табл. 6, 7).

Таблица 6. Входные воздействия объекта управления *DrawManager*

Входное воздействие	Описание
x1	Проверяет, в каком режиме рисования находится программа. При истинности рисуем большими буквами счет игры, в ином случае рисуем процесс игры

Таблица 7. Выходные воздействия объекта управления *DrawManager*

Выходное воздействие	Описание
z0	Нарисовать состояние игры на экране
z1	Установить режим рисования счета
z2	Установить режим рисования хода игры
init	Запустить приложение и автоматы машин противника

2.1.3.2. Объект управления игровой моделью (*o2*)

Этот объект управления предоставляет входные и выходные воздействия, которые относятся к созданию и изменению игрового мира (табл. 8, 9).

Таблица 8. Входные воздействия объекта управления *GameManager*

Входное воздействие	**Описание**
x1	Истина в случае столкновения машины игрока с машиной противника
x2	Истина в случае, если машина игрока в данный момент взрывается
x3	Количество машин противника, участвующих в погоне

Таблица 9. Выходные воздействия объекта управления *GameManager*

Выходное воздействие	**Описание**
z1	Повернуть игрока в сторону указателя мыши
z2	Взорвать машину игрока
z3	Передвинуть машину игрока на один ход
z4	Увеличить счет
z5	Распределить машины противника случайным образом
z6	Переслать сообщение таймера всем машинам, принимающим участие в гонке
z7	Добавить две машины к преследующим машинам
z8	Послать сообщение автомату, имитирующему Марковскую цепь, сделать вероятностный переход
z9	Обновить счет
z10	Добавить три машины к преследующим машинам
z11	Добавить одну машину к преследующим машинам
z12	Сообщить машинам противника о начале игры
z13	Сообщить машинам противника о закрытии программы
z14	Остановить машину игрока
z15	Уменьшить счет на 50 очков

2.1.3.3. Объект управления случайными числами (*o3*)

Этот объект управления предоставляет входные и выходные воздействия, которые относятся к созданию новых машин, появляющихся из-за пределов экрана (табл. 10).

Таблица 10. Входные воздействия объекта управления *RandomWalkManager*

Входное воздействие	**Описание**
p	Возвращает истину с вероятностью равной *p*. Задается в файле config.xml (Приложение 2)
q	Возвращает истину с вероятностью равной *q*. Задается в файле config.xml (Приложение 2)
random	Возвращает случайное число от 0 до 100

2.1.3.4. Объект управления подсчетом (*o4*)

Входит в стандартную поставку *UniMod*.

2.1.4. Автоматы

2.1.4.1. Автомат управления игрой из панели управления (*A1*)

Этот автомат представлен на рис. 3 и является базовым автоматом в данном приложении. Он управляет процессом игры, реагируя на нажатия клавиш: «старт», «стоп», «пауза».

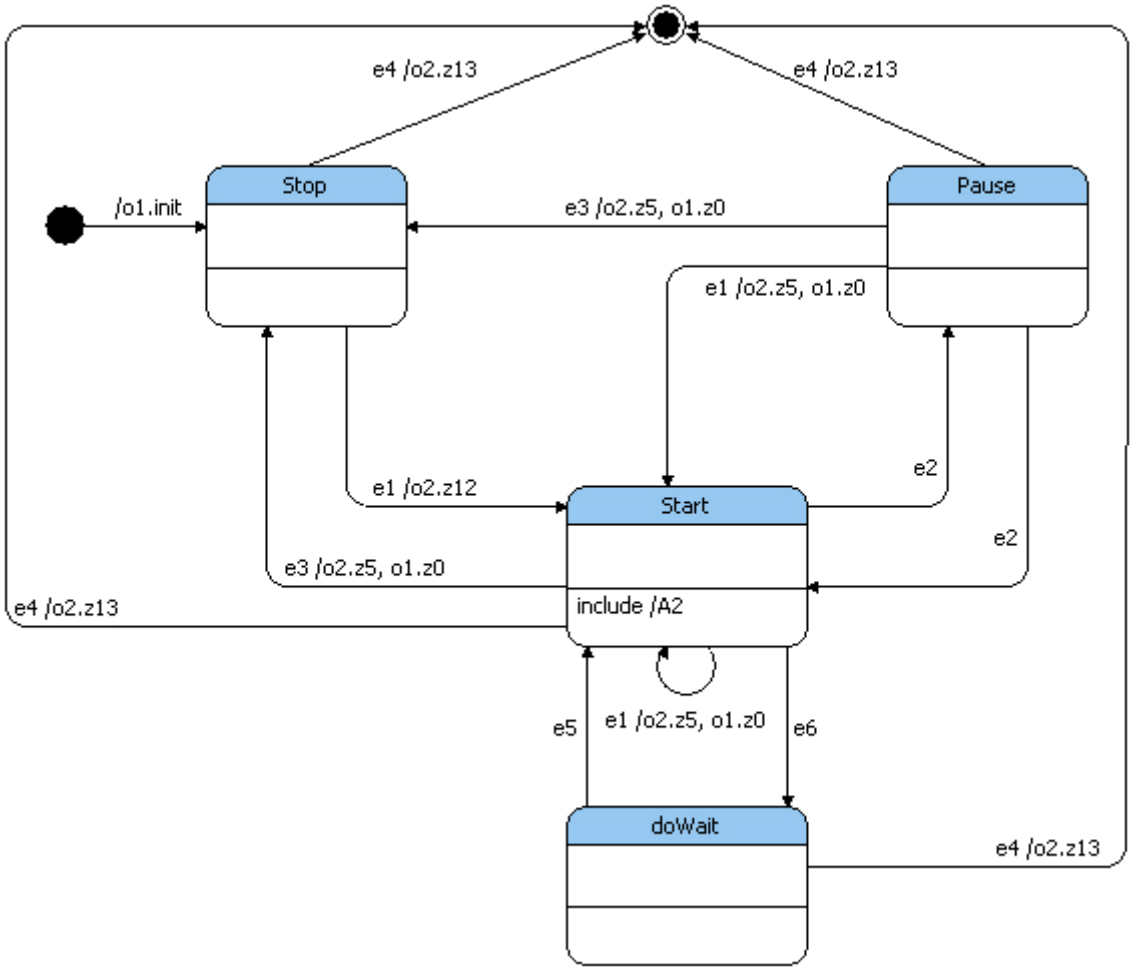


Рис. 3. Автомат *A1*

В табл. 11 представлены описания состояний автомата *A1*.

Таблица 11. *GameModel A1*

Состояние	**Описание**
Stop	В этом состоянии автомат находится в случае остановки игры или в начале игры до тех пор, пока пользователь не начал игру
Start	Состояние, в котором происходит процесс игры (движение машин)
Pause	Временная приостановка игры
doWait	Временная приостановка игры, но в отличие от состояния *Pause*, автомат переходит в это состояние при потере фокуса главным окном приложения

2.1.4.2. Автомат управления игрой во время движения (*A2*)

Данный автомат, представленный на рис. 4, отвечает за основную логику игры:

- двигает машинки;
- проверяет машины на столкновения;
- если произошло столкновение полицейских, то добавляет новые машины;
- если взорвалась машина игрока, заканчивает игру;

- отображает счет.

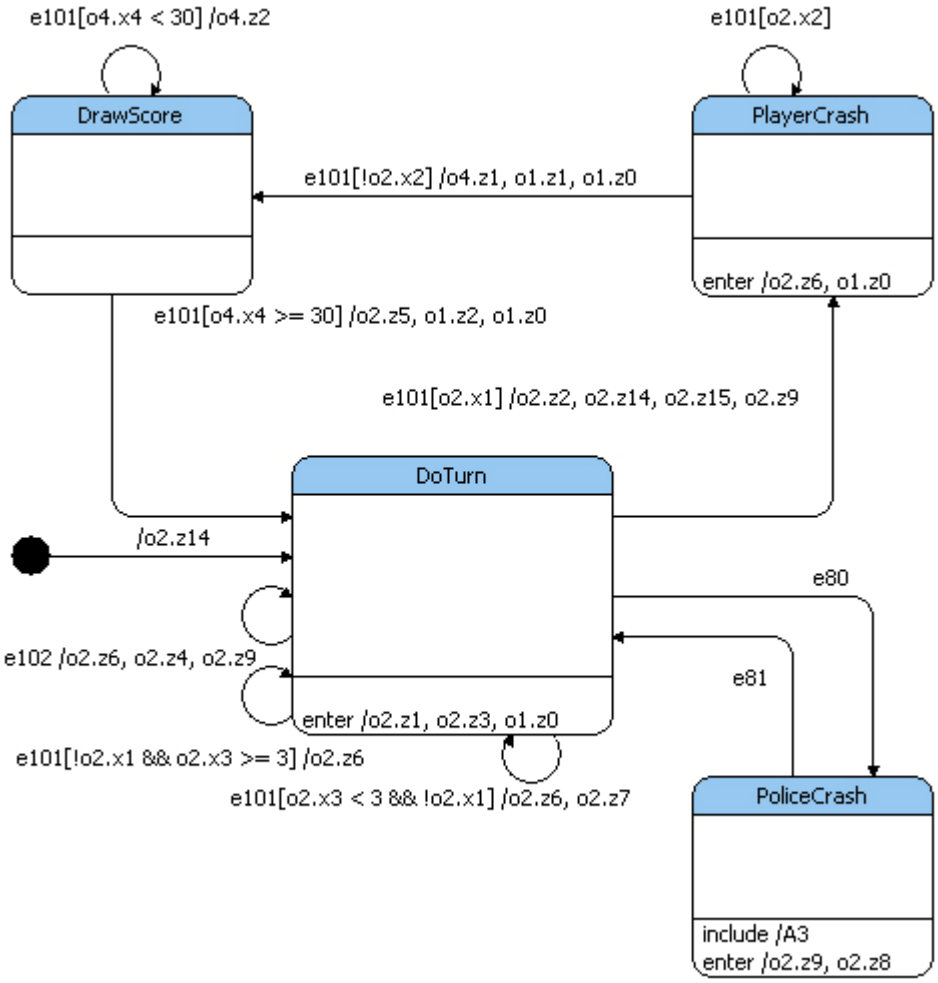


Рис. 4. Автомат *A2*

В табл. 12 представлены описания состояний автомата *A2*.

Таблица 12. *GameModel A2*

Состояние	**Описание**
DoTurn	В данном состоянии происходит передвижение машины игрока и отправка сообщений таймера машинам полиции. Это основное состояние главного автомата в процессе игры, в котором он должен находиться большую часть игрового времени
PoliceCrash	В данное состояние автомат переходит при столкновении и взрыве машин противника с целью обновить счет и изменить количество машин противника, принимающих участие в преследовании
PlayerCrash	Взрыв машины игрока. Автомат находится в этом состоянии до тех пор, пока не закончится анимация взрыва
DrawScore	Ничего не происходит с машинами, только рисуется на экране текущий набранный счет

2.1.4.3. Автомат для генерации новых машин (*A3*)

Этот автомат представляет собой простую Марковскую цепь, которая была реализована с помощью средства *UniMod*. Она используется для генерации новых машин, которые появляются из-за границ экрана. В зависимости от того, в каком состоянии находится *A3*, игроку навстречу поедут одна или три машины.

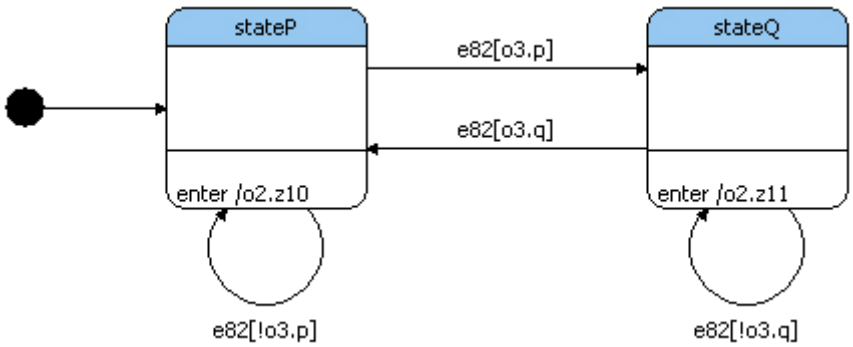


Рис. 5. Автомат *A3*

Возможность реализации Марковской цепи на основе автоматной модели была предложена А.А. Шалыто в работе [1]. В табл. 13 представлены описания состояний автомата *A3*.

Таблица 13. *GameModel A3*

Состояние	**Описание**
stateP	При входе в данное состояние количество машин противника увеличивается на три. Состояние можно покинуть с вероятностью *p* и остаться в нем с вероятностью *1 – p*
stateQ	При входе в данное состояние количество машин противника увеличивается на один. Состояние можно покинуть с вероятностью *q* и остаться в нем с вероятностью *1 – q*

2.2. *CarModel1*

Здесь реализовано управление машин с самой простой тактикой – постоянно двигаться на машину игрока.

2.2.1. Схема связей

Схема связей в виде диаграммы классов представлена на рис. 6. На ней представлены: один поставщик событий, автомат (*A1*) и один объекта управления.

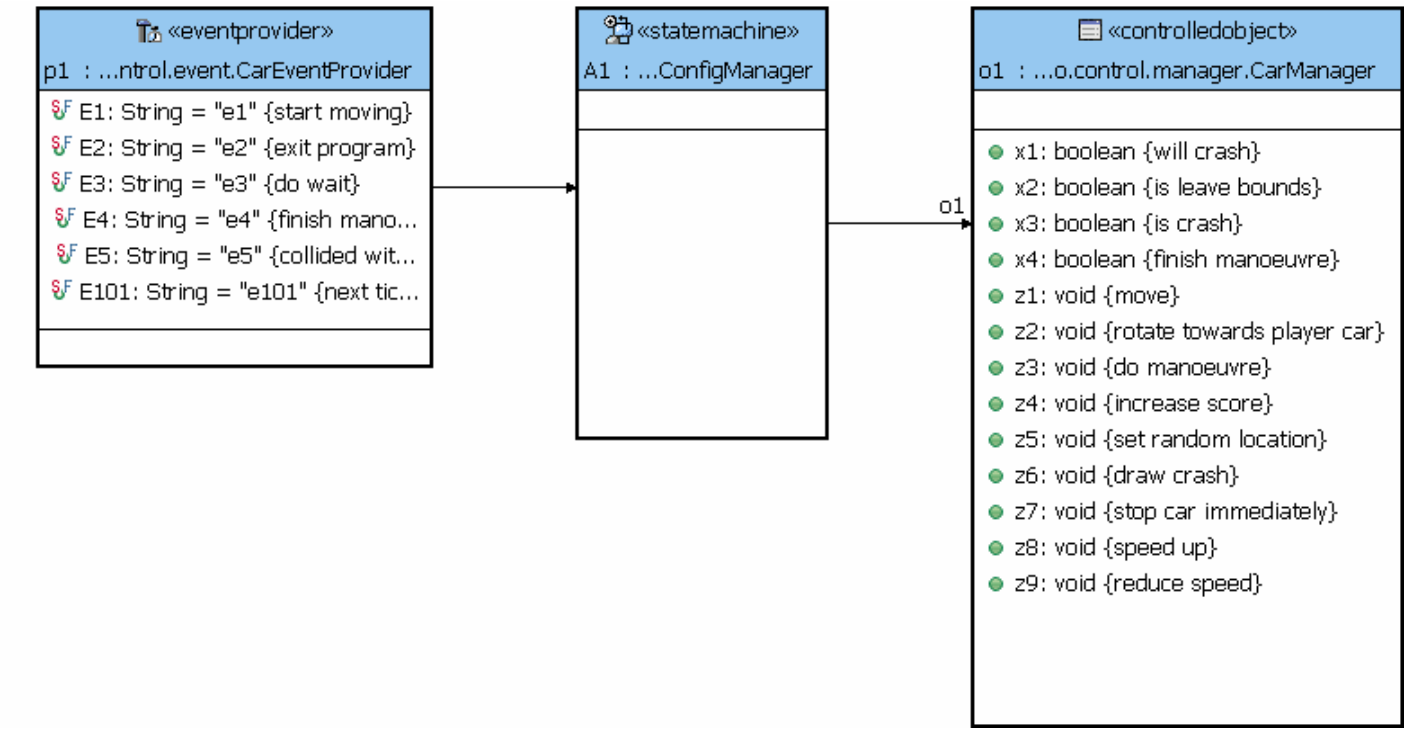


Рис. 6. Схема связей *CarModel1*

2.2.2. Поставщик событий машины

События, относящиеся к этому поставщику, инициируются игровым миром и описаны в табл. 14.

Таблица 14. События поставщика событий *CarEventProvider*

Событие	**Описание**	**Параметр**
e101	Произошло событие таймера. Оно пересылается главным автоматом, а не принимается непосредственно от таймера	Нет параметра
e1	Начать движение полицейской машины	Нет параметра
e2	Пользователь хочет покинуть программу	Нет параметра
e3	Приостановить движение машины	Нет параметра
e4	Уклонение от другой машины завершилось	Нет параметра
e5	Столкновение с другой машиной полиции	Нет параметра

2.2.3. Объект управления машиной полиции

Этот объект управления предоставляет входные и выходные воздействия, которые управляют конкретной машиной полиции (табл. 15, 16).

Таблица 15. Входные воздействия объекта управления *CarManager*

Входное воздействие	**Описание**
x1	Машина собирается столкнуться с другой машиной
x2	Машина покинула границы зоны видимости
x3	Машина разбилась
x4	Машина завершила выполнение маневра

Таблица 16. Выходные воздействия объекта управления *CarManager*

Выходное воздействие	**Описание**
z1	Передвинуть машину вперед на один ход
z2	Повернуть в ближайшем направлении к машине игрока
z3	Выполнить уклонение от столкновения с другой машиной
z4	Увеличить счет на 10 очков
z5	Установить машину на случайную позицию
z6	Нарисовать взрыв машины
z7	Прекратить движение машины
z8	Увеличить скорость движения
z9	Уменьшить скорость движения

2.2.4. Автомат управления машиной (*A1*)

Этот автомат, представленный на рис. 7, реализует тактику машины постоянной езды в направлении игрока.

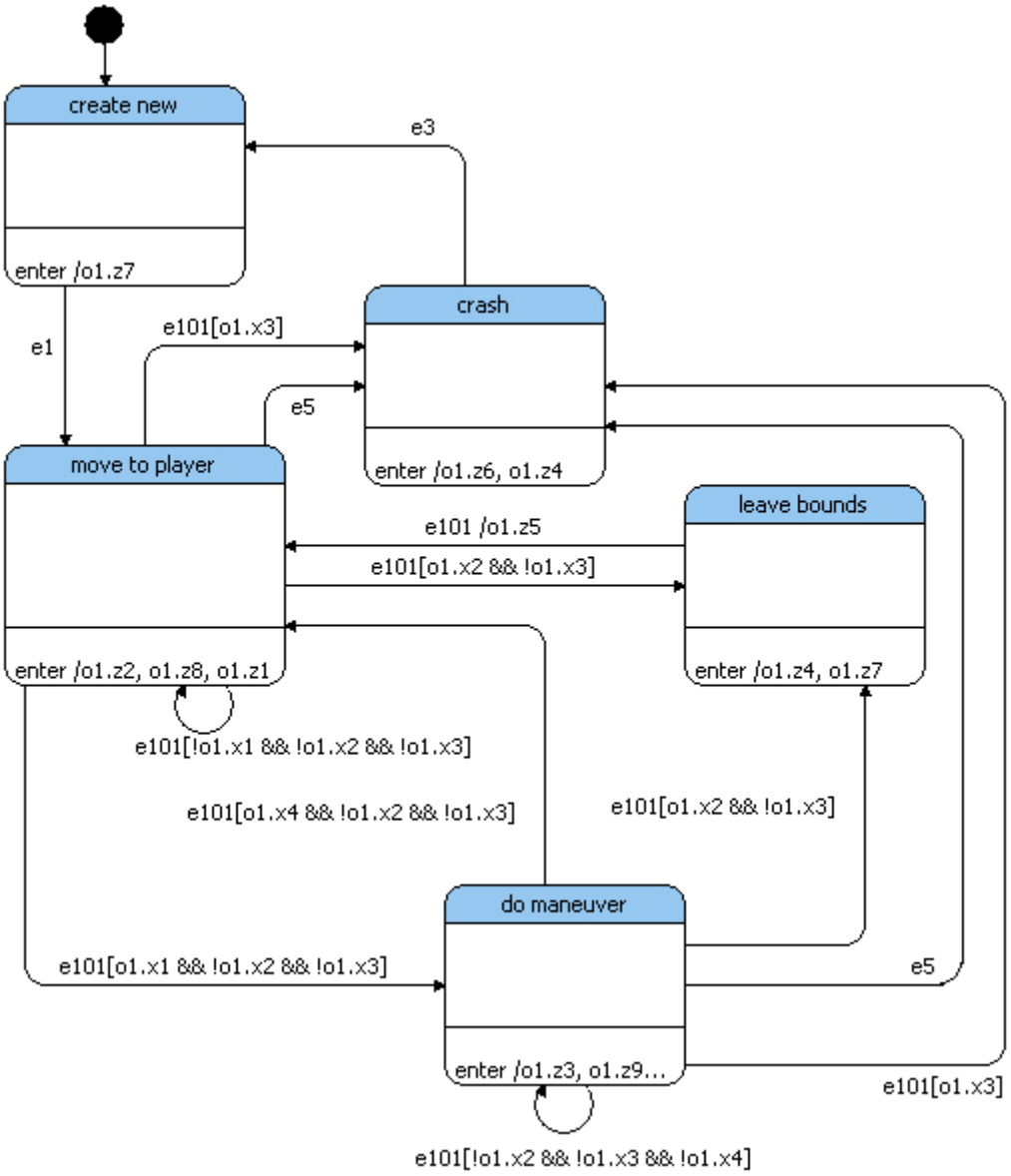


Рис. 7. Автомат *A1*

В табл. 17 представлены описания состояний автомата *A1*.

Таблица 17. *CarModel1 A1*

Состояние	**Описание**
create new	Состояние, в котором полицейская машина ожидает вступления в погоню
move to player	Движение полицейской машины к машине игрока
do maneuver	Совершение маневра уклонения от столкновения с другой машиной преследования. Автомат переходит в данное состояние после проверки того, что данная машина может столкнуться с какой-нибудь другой полицейской машиной
leave bounds	Полицейская машина покинула границы видимой области экрана. После этого она появляется в новом месте и продолжает преследование
crash	Машина столкнулась с другой полицейской машиной и больше не принимает участие в преследовании

2.3. *CarModel2*

Здесь реализовано управление машин со следующей тактикой – движение на машину игрока или в точку пересечения траекторий машин игрока и полицейского, если игрок и полицейский прибудут туда одновременно.

2.3.1. Схема связей

Схема связей в виде диаграммы классов представлена на рис. 8. На ней представлен один поставщик событий, автоматы (*A1*, *A2*) и один объект управления.

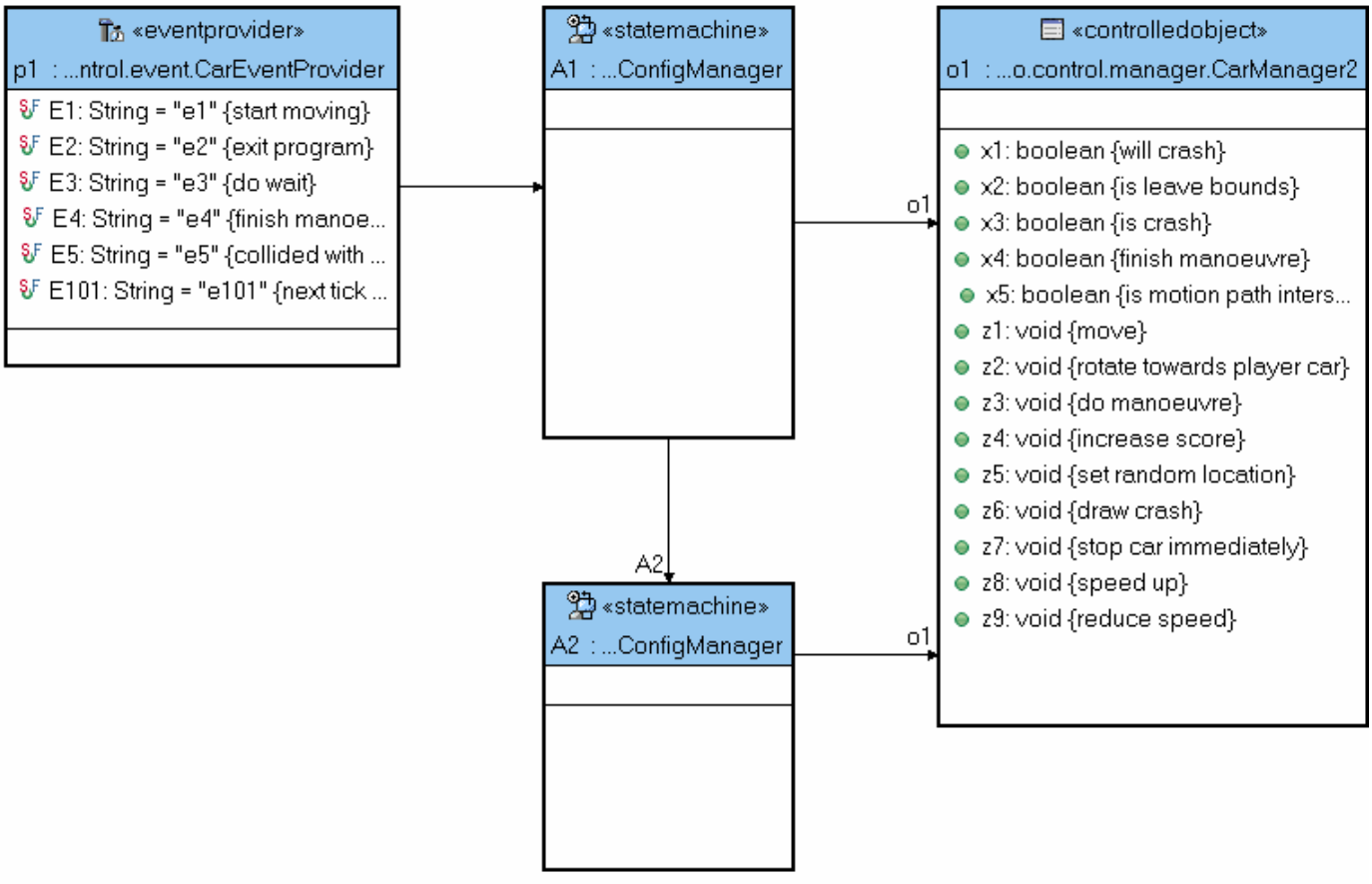


Рис. 8. Схема связей *CarModel2*

2.3.2. Поставщики событий

Они идентичны поставщикам событий из *CarModel1*.

2.3.3. Объект управления машиной полиции

Этот объект управления предоставляет входные и выходные воздействия, которые управляют конкретной машиной полиции (табл. 18). Он наследуется от соответствующего объекта управления из *CarModel1*.

Таблица 18. Входные воздействия объекта управления *CarManager2*

Входное воздействие	**Описание**
x1 – x4	Наследуются от объекта управления *CarManager*
x5	Является истиной в случае прибытия машины противника и игрока в точку пересечения своих траекторий в одно и тоже время

2.3.4. Автоматы

2.3.4.1. Автомат основного управления машиной (*A1*)

Этот автомат, представленный на рис. 9, реализует основную часть управления полицейской машины. В табл. 17 представлены описания состояний автомата *A1*.

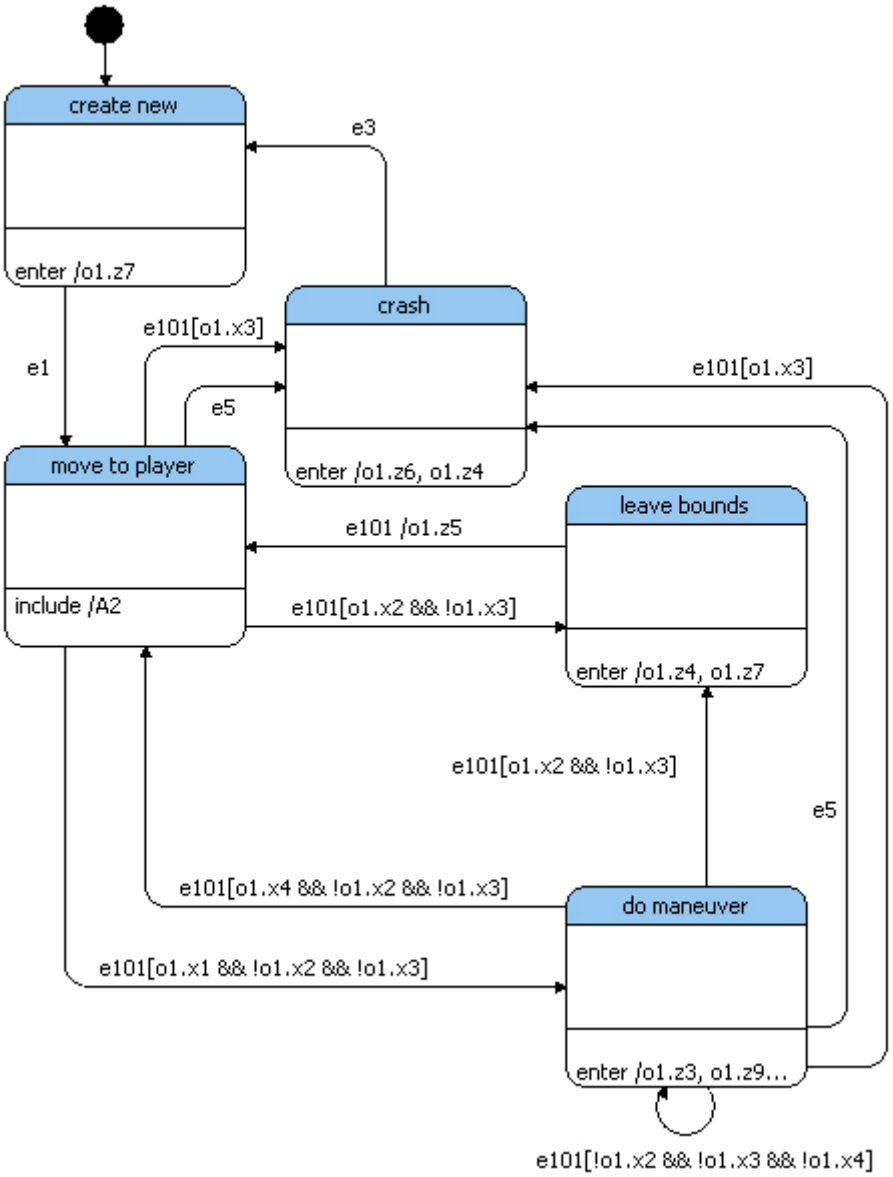


Рис. 9. Автомат *A1*

2.3.4.2. Автомат расширенного управления машиной (*A2*)

Этот автомат, представленный на рис. 10, реализует новую тактику полицейской машины – ехать наперерез игроку.

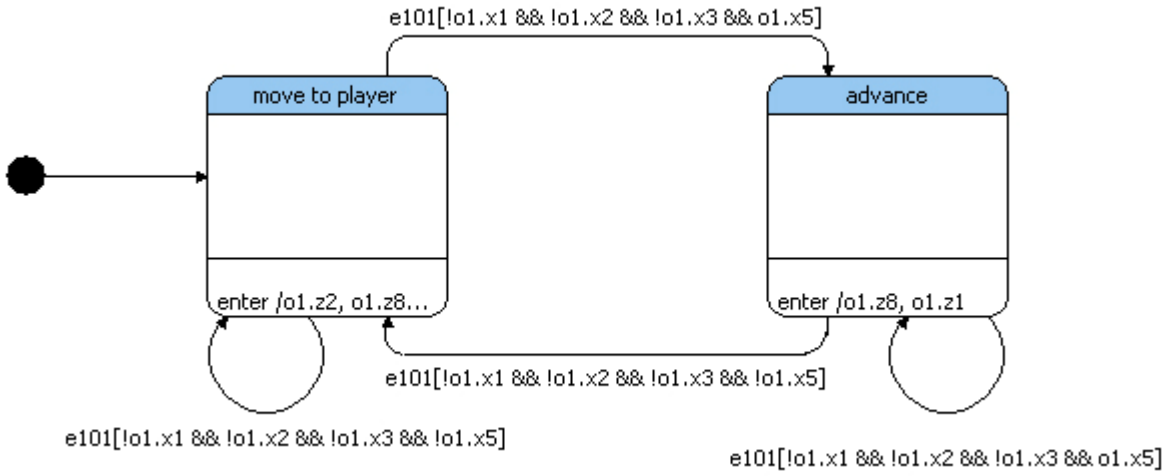


Рис. 10. Автомат *A2*

В табл. 19 представлены описания состояний автомата *A2*.

Таблица 19. *CarModel2 A2*

Состояние	**Описание**
move to player	Движение полицейской машины к машине игрока
advance	В данном состоянии машина полиции не поворачивается в сторону машины игрока, а едет на опережение в точку пересечения их траекторий, при условии прибытия туда одновременно

2.4. *CarModel3*

Здесь реализовано управление машин со следующей тактикой – движение на машину игрока или в точку пересечения траекторий машин игрока и полицейского, если игрок и полицейский прибудут туда одновременно. Если полицейский прибывает раньше игрока, то он замедляет скорость.

2.4.1. Схема связей

Схема связей в виде диаграммы классов представлена на рис. 11. На ней представлены: один поставщик событий, автоматы (*A1*, *A2*) и один объект управления.

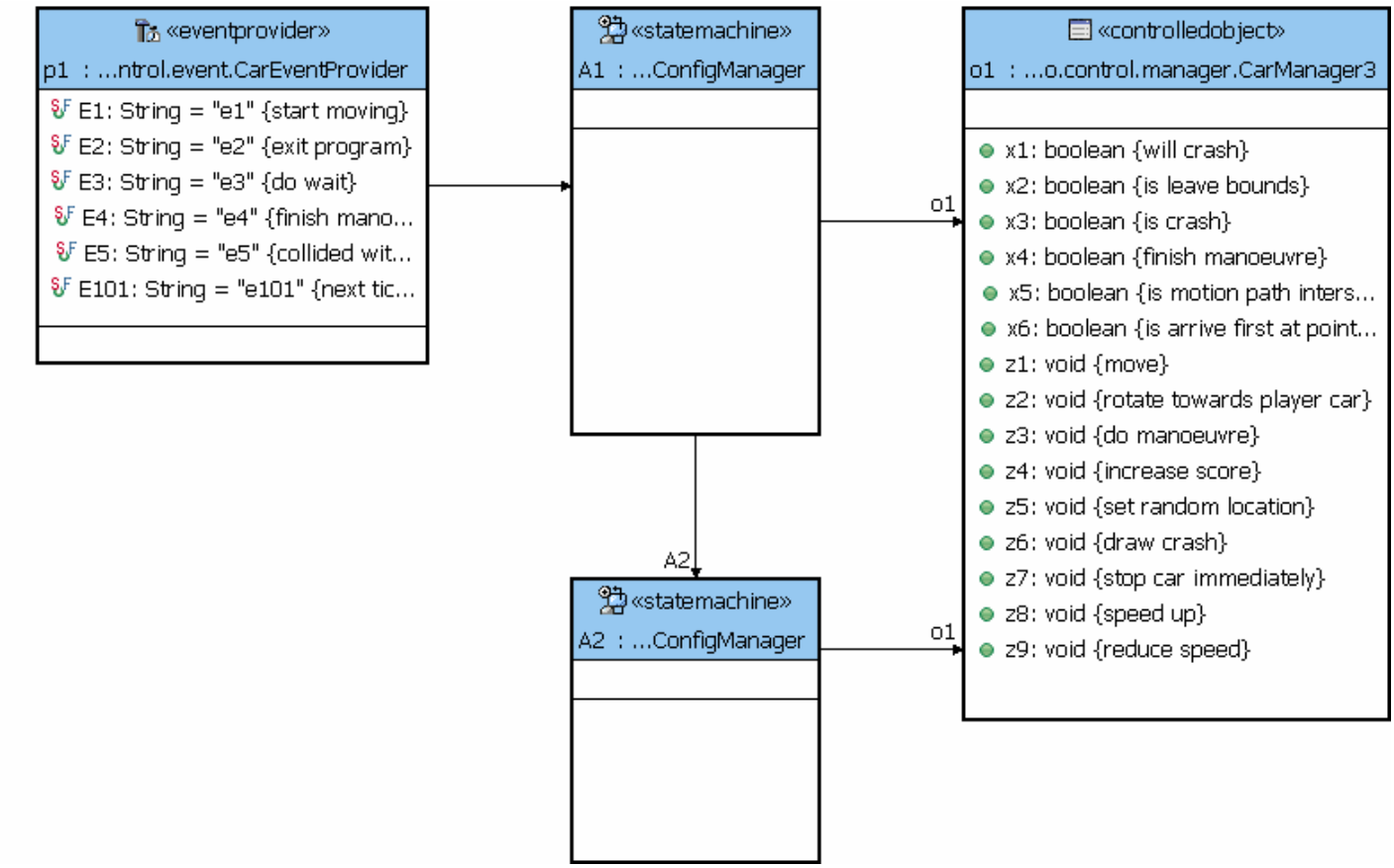


Рис. 11. Схема связей *CarModel3*

2.4.2. Поставщики событий

Они идентичны поставщикам событий из *CarModel1* и *CarModel2*.

2.4.3. Объект управления машиной полиции

Этот объект управления предоставляет входные и выходные воздействия, которые управляют конкретной машиной полиции (табл. 20). Наследуется от соответствующего объекта управления из *CarModel2*.

Таблица 20. Входные воздействия объекта управления *CarManager3*

Входное воздействие	**Описание**
x1 – x5	Наследуются от объекта управления *CarManager2*
x6	Является истиной в случае прибытия машины противника раньше машины игрока в точку пересечения траекторий

2.4.4. Автоматы

2.4.4.1. Автомат основного управления машиной (*A1*)

Этот автомат, представленный на рис. 9, реализует основную часть управления полицейской машины. Он ничем не отличается от соответствующего автомата из *CarModel2*.

2.4.4.2. Автомат расширенного управления машиной (*A2*)

Этот автомат, представленный на рис. 12, реализует новую тактику полицейской машины – ехать наперерез игроку и, если надо, ждать его в точке возможного пересечения их траекторий.

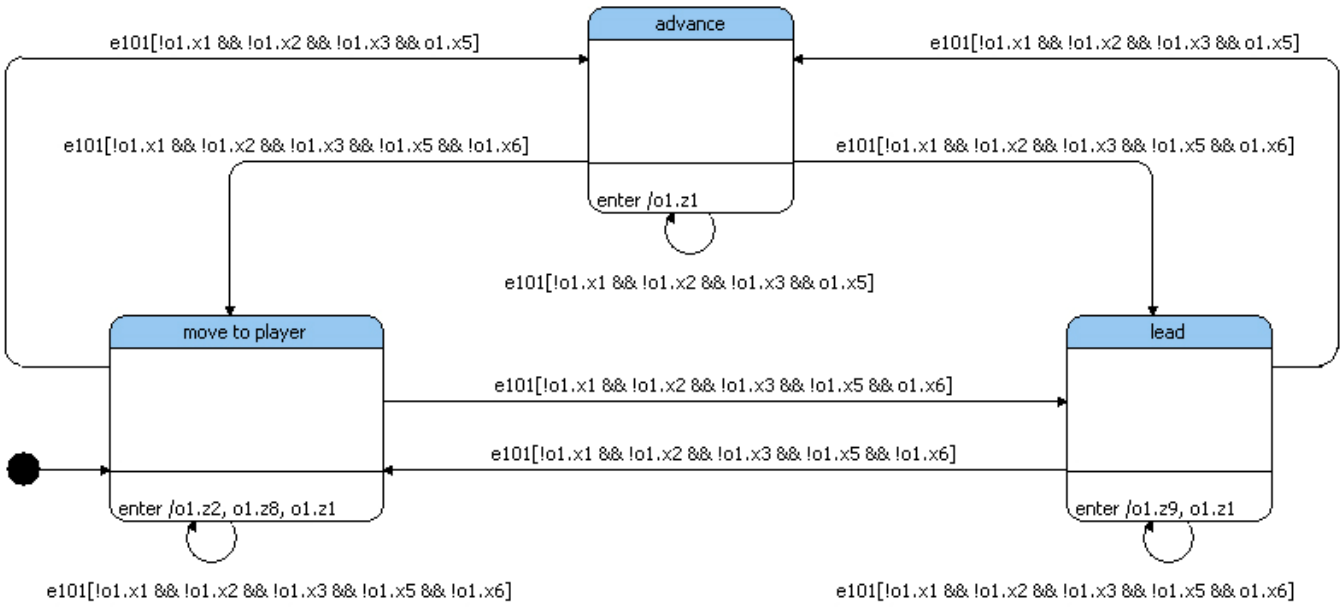


Рис. 12. Автомат *A2*

В табл. 21 представлены описания состояний автомата *A2*.

Таблица 21. *CarModel3 A2*

Состояние	**Описание**
move to player	Движение полицейской машины к машине игрока
advance	В данном состоянии машина полиции не поворачивается в сторону машины игрока, а едет на опережение в точку пересечения их траекторий, при условии прибытия туда одновременно
lead	В данном состоянии машина полиции не поворачивается в сторону машины игрока, а едет на опережение в точку пересечения их траекторий, при условии прибытия туда раньше игрока. Чтобы не проскочить данную точку, полицейский уменьшает скорость

2.5. *CarModel4*

Здесь реализовано управление машин со следующей тактикой – когда полицейский видит, что он прибудет в точку пересечения траекторий раньше игрока, то он переходит в состояние “перехват” и рассылает остальным машинам приглашение участвовать в перехвате. Одна из машин, принявшая приглашение, едет в точку перехвата, а остальные продолжают независимое преследование.

2.5.1. Схема связей

Схема связей в виде диаграммы классов представлена на рис. 13. На ней представлены: один поставщик событий, автоматы (*A1*, *A2*) и один объекта управления.

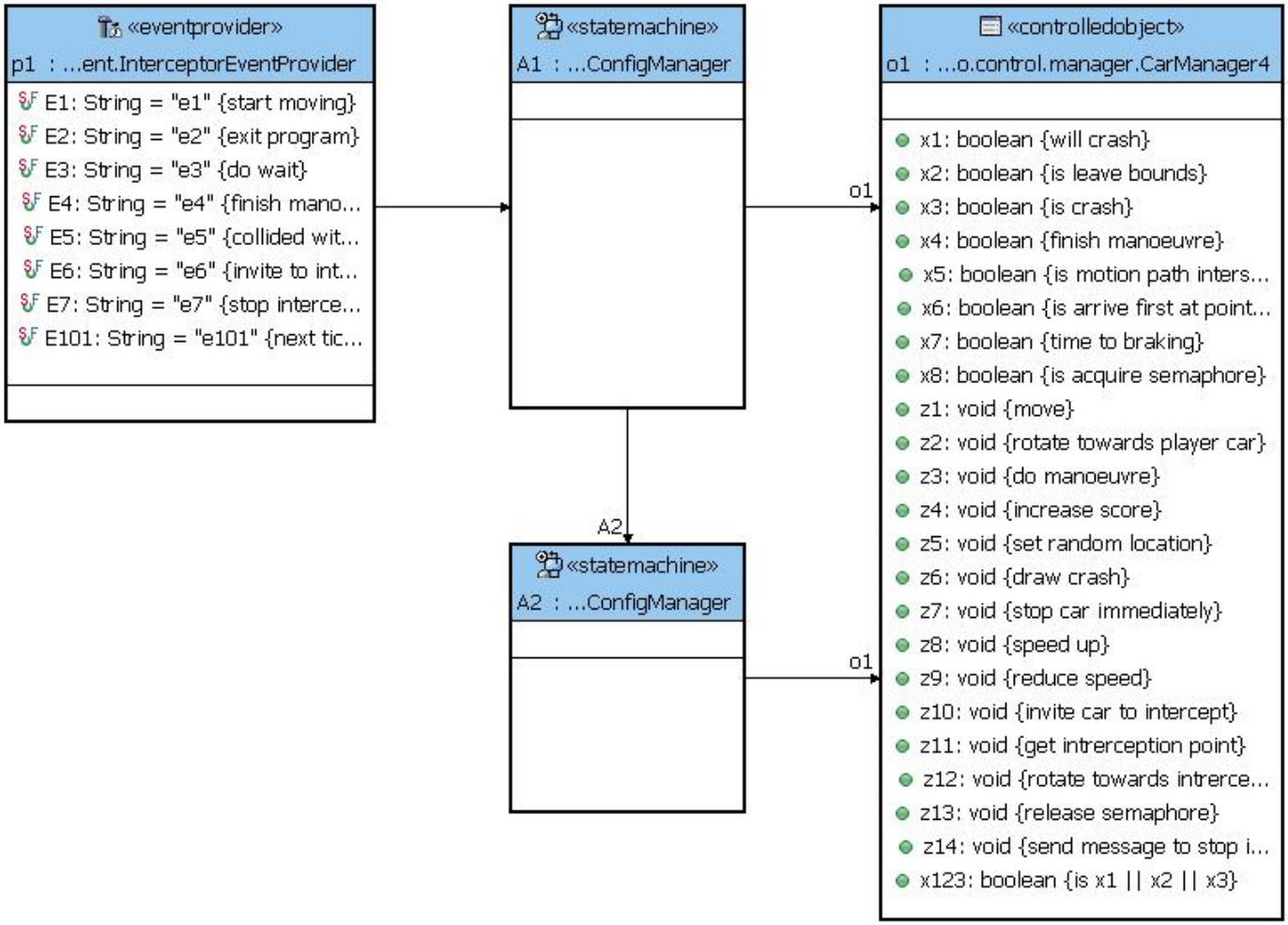


Рис. 13. Схема связей *CarModel4*

2.5.2. Поставщик событий машины

События, относящиеся к этому поставщику, инициируются игровым миром и описаны в табл. 22.

Таблица 22. События поставщика событий *InterceptorEventProvider*

Событие	**Описание**	**Параметр**
e1 – e5, e101	События наследуются от *CarEventProvider*	Нет параметра
e6	Получение приглашения от другой полицейской машины участвовать в перехвате	Точка, в которой произойдет перехват
e7	Получение извещения об отмене перехвата	Нет параметра

2.5.3. Объект управления машиной полиции

Этот объект управления предоставляет входные и выходные воздействия, которые управляют конкретной машиной полиции (табл. 23, 24). Он наследуется от соответствующего объекта управления из *CarModel3*.

Таблица 23. Входные воздействия объекта управления *CarManager4*

Входное воздействие	**Описание**
x1 – x5	Наследуются от объекта управления *CarManager3*
x7	Наступило время для торможения
x8	Истина в том случае, если в перехвате участвуют меньше двух машин
x123	Выполнение *x1 | | x2 | | x3*

Таблица 24. Выходные воздействия объекта управления *CarManager4*

Выходное воздействие	**Описание**
z1 – z9	Наследуются от объекта управления *CarManager3*
z10	Пригласить другие машины участвовать в перехвате
z11	Получить точку, в которой планируется перехват
z12	Поворачиваться в направлении точки перехвата
z13	Позволить другим машинам участвовать в перехвате
z14	Послать сообщение о прекращении перехвата

2.5.4. Автоматы

2.5.4.1. Автомат основного управления машиной (*A1*)

Этот автомат, представленный на рис. 9, реализует основную часть управления полицейской машины. Он ничем не отличается от соответствующих автоматов из *CarModel2* и *CarModel3*.

2.5.4.2. Автомат расширенного управления машиной (*A2*)

Этот автомат, представленный на рис. 14, реализует новую тактику полицейской машины – перехватывать игрока в какой-то точке вместе с другой полицейской машиной. В табл. 25 представлены описания состояний автомата *A2*.

Таблица 25 *CarModel4 A2*

Состояние	**Описание**
move to player	Движение полицейской машины к машине игрока
Advance	В данном состоянии машина полиции не поворачивается в сторону машины игрока, а едет на опережение в точку пересечения их траекторий, при условии прибытия туда одновременно
try semaphore	Полицейская машина проверяет возможность участия в перехвате. Ей может быть не позволено участвовать в нем в случае, если уже две машины задействованы в данной операции
Interception	Машина полиции едет в точку перехвата и должна прибыть туда раньше игрока
move to interception point	Машина, принявшая приглашение участвовать в перехвате, которая двигается в назначенную точку для перехвата
Braking	Состояние, в котором полицейская машина тормозит, чтобы остановится в точке, назначенной для перехвата машины игрока

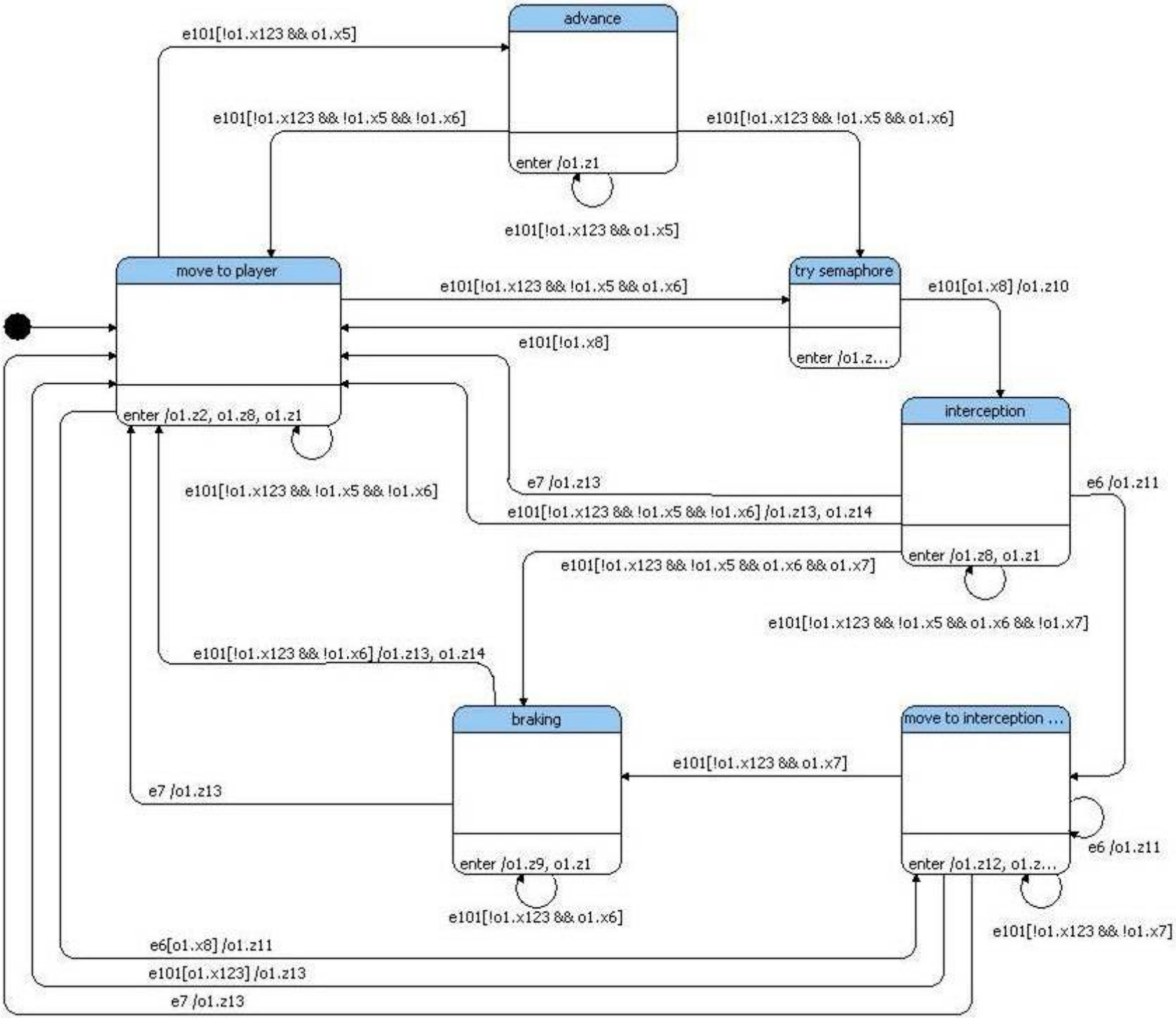


Рис. 14. Автомат *A2*

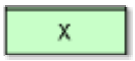
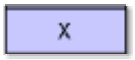
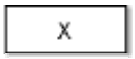
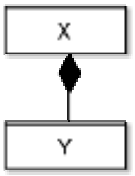
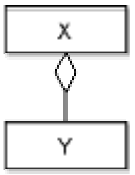
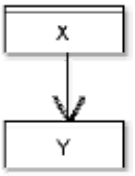
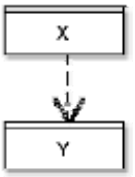
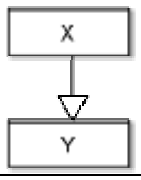
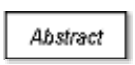
3. Реализация

3.1. Диаграмма классов

Для создания диаграммы классов использовалась свободно распространяемая программа *ArgoUML* [6]. Для подробного изучения языка *UML* авторы рекомендуют книгу [2] и материалы в Интернете [3-5].

Полная диаграмма классов данного проекта представлена на рис. 15. Заметим, что здесь опущены все стандартные классы *Java*, чтобы упростить диаграмму. При построении диаграммы использовались стандартные обозначения, приведенные в табл. 26.

Таблица 26. Обозначения на диаграмме

Обозначение	Семантика
	Поставщики событий. Зеленые стрелки из них идут в *UniMod*-модель, которой они принадлежат
	Объекты управления. Синие стрелки в них идут из *UniMod*-модели, которой они принадлежат
	Собственные классы, написанные вручную
	*UniMod*-модель
	Отношение «композиция». Класс Y создается классом X и не может существовать без него
	Отношение «агрегация». Класс Y создает и использует класс X в своей внутренней реализации
	Отношение «ассоциация». Класс X использует класс Y
	Отношение «зависимость». При изменении класса Y возможно нужно будет поменять класс X
	Отношение «обобщение». Класс X наследуется от класса Y
	Класс Abstract – абстрактный (название класса написано курсивом)

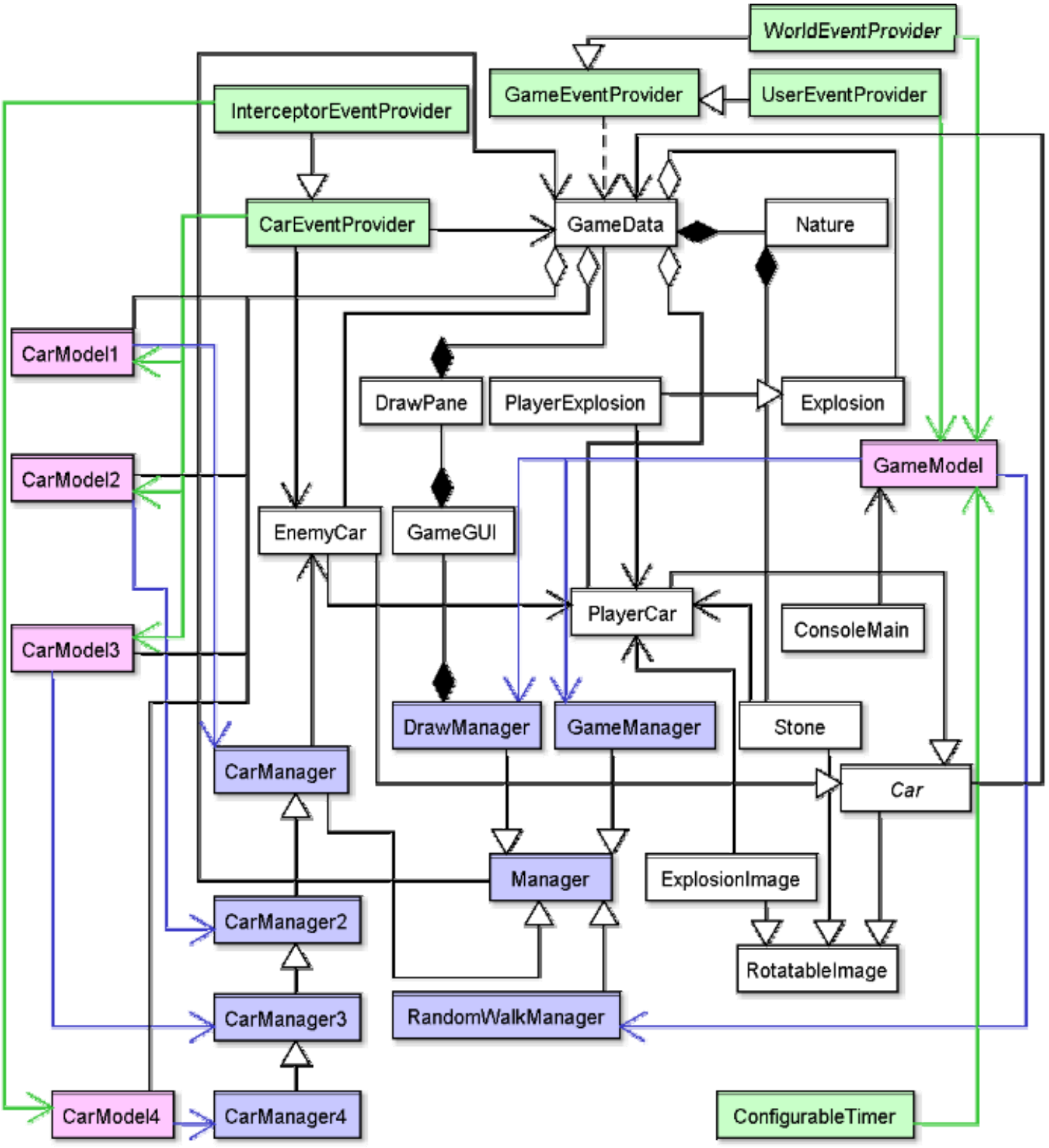


Рис. 15. Диаграмма классов проекта

Краткие характеристики классов, изображенных на диаграмме, приведены в табл. 27.

Таблица 27. Описание классов

Класс	Описание
package ru.ifmo	
ConsoleMain	Класс, который создает и загружает головной автомат из модели *GameModel*
package ru.ifmo.control	
DrawPane	Компонент, в котором рисуется весь процесс игры: машины полиции и игрока, камни, кактусы и взрывы. В конструкторе этого класса создается экземпляр объекта *GameData*
GameData	Класс, который хранит данные об игре, в том числе, о машине игрока, полиции, элементах местности, счете, задержки между ходами. В связи с этим, ссылку на него хранят большинство классов. В конструкторе класса загружаются *cars.xml* (Приложение 1) и *config.xml* (Приложение 2), из которых извлекается информация для инициализации данных игры и характеристик машин. Там же создаются автоматы полицейских машин
GameGUI	Класс пользовательского интерфейса. Он является главным окном приложения, в котором создаются все элементы управления и вывода информации пользователю, в том числе *DrawPane*
Nature	Класс, отвечающий за элементы местности и их отображение на экране. К элементам местности относятся кактусы и камни
package ru.ifmo.control.event	
CarEventProvider	Поставщик событий для машин противника. Экземпляры этого класса знают о машинах полиции и обо всех автоматах, управляющих ими
ConfigurableTimer	Поставщик событий таймера для главного автомата. Посылает событие каждые *GameData.sleepAmount* миллисекунд (это значение изменяется от 10ms до 100ms). Автоматам машин противника это сообщение пересылает главный автомат в случае обработки данного сообщения
GameEventProvider	Поставщик событий для игры. Предоставляет базовый набор функций отправки сообщений главному автомату
InterceptorEventProvider	Поставщик событий для автоматов, управляющих машинами с функцией перехвата машины игрока
UserEventProvider	Поставщик событий для главного автомата, который управляет состоянием игры: начать, приостановить или прекратить игру
WorldEventProvider	Поставщик событий игровых моментов, таких как взрывы полицейских машин и изменение их числа

Класс	Описание
package ru.ifmo.control.manager	
CarManager	Объект управления машиной самого простого типа. Каждый экземпляр класса данного типа знает полицейскую машину, которой он управляет и предоставляет элементарный набор функций для управления ею. Объект позволяет проверять столкновение машин, возможность столкновения, покидание границ видимой области, осуществление управление машиной, такое как движение вперед, поворот, поворот в сторону машины игрока, уменьшение и увеличение скорости
CarManager2	Объект управления машиной второго типа, является улучшенной версией *CarManager*, которая позволяет проверять время прибытия в точку пересечения траекторий машин и двигаться туда, в случае возможности прибыть туда одновременно с машиной игрока
CarManager3	Объект управления машиной третьего типа, расширяет класс *CarManager2* и позволяет ехать в точку пересечения траекторий машины полиции и игрока, при этом изменить скорость для прибытия туда одновременно
CarManager4	Объект управления машиной четвертого типа, расширяет класс *CarManager3*, добавляя возможность осуществления перехвата машины игрока, который заключается в прибытии на место пересечения траекторий движения машин и остановки там, перекрывая путь для проезда игроку. Полицейская машина, обнаружившая возможность перехвата, рассылает приглашения участвовать в нем всем остальным машинам, среди которых только одна принимает приглашение и едет в точку, назначенную для перехвата
DrawManager	Класс, отвечающий за рисование машин и действия связанные с рисованием в классе *DrawPane*. Он предоставляет возможность запустить главное окно приложения и автоматы полицейских машин
GameManager	Объект управления ходом игры и машиной игрока. Проверяет столкновение с машинами полиции, состояние машины игрока, предоставляет набор функций для управления ею, позволяет изменять количество машин противника и информировать их о событиях таймера
Manager	Общий предок всех объектов управления, предоставляющий для них возможность установки данных игры типа *GameData*
RandomWalkManager	Объект управления автоматом, эмулирующего простейшую Марковскую цепь с двумя состояниями и вероятностными переходами из одного состояния в другое, что позволяет нелинейно изменять количество

Класс	Описание
	полицейских машин
package `ru.ifmo.image`	
Car	Абстрактный класс, предоставляющий возможность управления машиной и ее рисованием на экране. Машина характеризуется такими параметрами, как максимально развиваемая скорость, текущая скорость, ускорение, скорость поворота руля, которая влияет на угол поворота автомобиля
EnemyCar	Расширяет класс *Car* и добавляет функциональные возможности характерные для машины противника, такие как случайное появление на экране и рисование относительно машины игрока
Explosion	Класс, предоставляющий возможность отображения на экране анимированного взрыва. Каждый отдельный кадр взрыва представляет собой объект типа *ExplosionImage*
ExplosionImage	Изображение одного кадра взрыва в системе координат относительно машины игрока
PlayerCar	Расширяет класс *Car* и предоставляет функциональные возможности, характерные для машины игрока
PlayerExplosion	Взрыв машины игрока
RotatableImage	Класс, позволяющий рисовать изображение, загруженное из файла. Предоставляет методы для перемещения и поворота изображения в абсолютной системе координат. Изображение характеризуется двумя координатами и направляющим углом
Stone	Элемент местности, такой как камень или кактус. Этот объект позволяет рисовать картинки с изображением природных объектов, относительно машины игрока
package `UniMod`	
CarModel1	Автомат управления полицейской машиной самого простого типа с тактикой движения на игрока и с возможностью попытки уклонения от других машин. Управляет объектом управления *CarManager*
CarModel2	Автомат, расширяющий возможности *CarModel1*, который по возможности направляет машину в точку пересечения ее траектории с прямой движения машины игрока. Управляет объектом управления *CarManager2*
CarModel3	Автомат, расширяющий возможности *CarModel2*, который направляет полицейскую машину в точку пересечения ее траектории с прямой движения машины игрока. Если полицейский успевает прибыть туда раньше, то сбавляет скорость, иначе действует как *CarModel2*. Управляет объектом управления *CarManager3*
CarModel4	Автомат, расширяющий возможности *CarModel3*,

Класс	Описание
	который направляет полицейскую машину в точку пересечения ее траектории с прямой движения машины игрока. Если полицейский успевает прибыть туда раньше, то посылает остальным машинам полиции приглашение участвовать в перехвате, иначе действует как *CarModel3*. Остановившись в точке пересечения траекторий, машина ждет подъезда игрока до тех пор, пока последний не изменит направление движения. Управляет объектом управления *CarManager4*
GameModel	Основной автомат в программе, отвечает за состояние игры, такое как начать игру, приостановить, продолжить или прекратить. Так же уменьшает или увеличивает количество машин, участвующих в преследовании; руководит машиной игрока, у которой всегда нажат газ и пользователю остается только мышкой корректировать направление движения машины

3.2. Запуск программы

Так как программа была спроектирована при помощи инструментального средства *UniMod*, она может быть запущена двумя способами: интерпретационным и компилятивным [7].

В этом проекте использовался компилятивно-интерпретационный подход. Головной автомат из модели *GameModel* может запускаться при помощи интерпретатора *UniMod*, использующего соответствующий *XML*-файл, или при помощи *Java*-машины, использующей соответствующий скомпилированный класс-файл. Для уменьшения размеров проекта было решено написать класс из двух строк – *ConsoleMain*, который только запускает головной автомат. При этом создание моделей *CarModel1*, *CarModel2*, *CarModel3*, *CarModel4* происходит с помощью компилятивного подхода.

Программа в виде *Java*-апплета опубликована на сайте http://is.ifmo.ru в разделе «*UniMod*-проекты».

3.3. Тестирование программы

При написании программы были созданы четыре тактики управления машинами полицейских: *CarModel1*, *CarModel2*, *CarModel3* и *CarModel4*. Какие тактики будут использоваться во время очередной игры, настраивается при помощи специального *XML*-файла (Приложение 1). В данной конфигурации файла заданы параметры игры, при которых будет создано по одной машине, отвечающей моделям *CarModel1*, *CarModel2* и *CarModel3*, и три машины, соответствующих модели *CarModel4*. Во время выполнения программы будет создано всего шесть объектов-машин, соответствующих своим моделям, которые участвуют в погоне.

Программа тестировалась следующим образом: запускался головной автомат, который выводил лог своих действий в файл (Приложение 4). Этот лог имеет следующую структуру: вся программа действует по таймеру, который каждый несколько миллисекунд дает команду о начале следующего хода. В начале каждого хода программа записывает координаты всех машин и взрывов в данный лог, а потом пересчитывает новые координаты машин в соответствии с их текущими направлениями. Таким образом, лог представляет собой набор координат всех машин, записанных в последовательные моменты времени. Далее запускалась специально написанная на языке *Java* программа (Приложение 5), которая переводила данные из лога в графический формат, строя по

координатам машин их траектории и выделяя места их взрывов. В итоге, получается изображение, пригодное для визуального анализа. Пример построения такого «графического лога» представлен на рис. 16. Здесь введены следующие обозначения:

- красная линия – траектория движения игрока;
- цветные линии – траектории движения полицейских;
- цветной кружок – место начала движения машины;
- черный кружок – место взрыва машины;

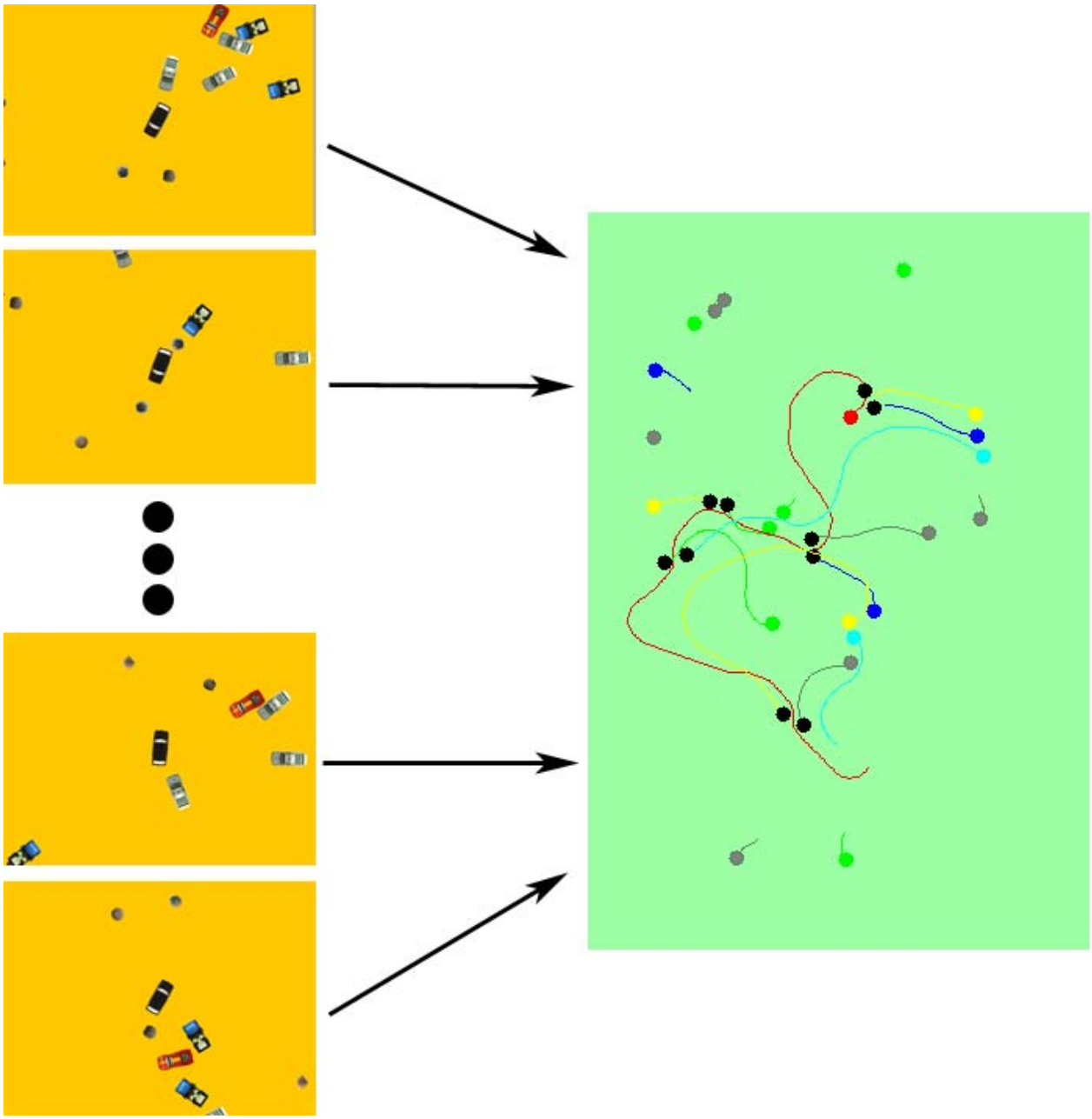


Рис. 16. Построение диаграммы хода игры

Заметим, что с помощью данной диаграммы можно устанавливать и анализировать различные свойства всей системы, например, можно проследить, как машина управляется игроком или сосчитать, как часто полицейские машины поворачивают. Диаграмма, представленная на рис. 16, позволяет установить, что все черные кружки располагаются парами – полицейские машины взрываются при столкновении друг с другом. Также видны очень короткие траектории и цветные кружки вообще без выходящих линий – это полицейские машины, которые просто не успели развернуться и начать погоню и, как

следствие, быстро выбывающие из игры. Так, с помощью автоматного программирования была создана формализованная модель игры «Побег» с возможностями графического анализа её состояний по статической диаграмме хода игры.

Заключение

Данный проект показывает возможность применения автоматного программирования для построения достаточно сложного класса мультиагентных систем. При этом было применено два средства проектирования программ – инструментальное средство *UniMod* и программа *ArgoUML*, в которой строились диаграммы классов. Совмещение этих методов визуального построения программ значительно повышает читабельность программы и позволяет исключить возникновение большинства ошибок, связанных с неправильной логикой работы проекта или с неправильным дизайном всей системы.

Литература

1. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. http://is.ifmo.ru/books/switch/6/
2. *Буч Г.*, *Рамбо Д.*, *Джекобсон А*. UML. Руководство пользователя. М.: ДМК, 2000.
3. *UML Resource Page*. http://uml.org/
4. *Теория UML*. http://www.info-system.ru/designing/methodology/uml/theory/theory.html
5. *Practical UML*. http://dn.codegear.com/article/31863/
6. *ArgoUML*. http://argouml.tigris.org/
7. *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». http://is.ifmo.ru/unimod-projects/plates/

Приложение 1. Файл `cars.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<game>
      <player
            name = "playerCar"
            image = "images/car3.png"
            maxSpeed = "5"
            maxAngle = "0.08"
            acceleration = "0.15"
      />
      <car
            name = "enemyCar1"
            image = "images/car4.png"
            maxSpeed = "4"
            maxAngle = "0.04"
            acceleration = "0.12"
            class = "unimod.Car1ModelEventProcessor"
      />

      <car
            name = "enemyCar2"
            image = "images/car4.png"
            maxSpeed = "4"
            maxAngle = "0.04"
            acceleration = "0.12"
            class = "unimod.Car2ModelEventProcessor"
            amount = "1"
      />

      <car
            name = "advancedCar"
            image = "images/car2.png"
            maxSpeed = "4.5"
            maxAngle = "0.05"
            acceleration = "0.15"
            class = "unimod.Car3ModelEventProcessor"
      />

      <car
            name = "interceptorCar"
            image = "images/car1.png"
            maxSpeed = "4.5"
            maxAngle = "0.05"
            acceleration = "0.15"
            class = "unimod.Car4ModelEventProcessor"
            amount = "3"
      />
</game>
```

Приложение 2. Файл `config.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
      <comment>Game data properties</comment>

      <entry key="sleepAmount">30</entry>

      <entry key="probability_p">0.7</entry>
      <entry key="probability_q">0.6</entry>

      <entry key="InvisibleRegionSizePercent">0.3</entry>
      <entry key="intersectionSize">5</entry>
      <entry key="crashSteps">8</entry>

      <entry key="natureImagesFolder">images/nature</entry>
</properties>
```

Приложение 3. *XML*-schema задания машин

```
<?xml version="1.0" encoding="utf-8" ?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
	<xsd:element name="game" type="game" />
	<xsd:complexType name="game">
		<xsd:sequence>
			<xsd:element
				name="player"
				type="Player"
				maxOccurs="1"
				/>
			<xsd:element
				name="car"
				type="Car"
				minOccurs="1"
				maxOccurs="unbounded"
				/>
		</xsd:sequence>
	</xsd:complexType>

	<xsd:attributeGroup name = "carAttrs">
		<xsd:attribute
			name="name"
			type="xsd:string"
			use="required"
			/>
		<xsd:attribute
			name="image"
			type="xsd:anyURI"
			use="required"
			/>
		<xsd:attribute
			name="maxSpeed"
			type="xsd:double"
			use="required"
			/>
		<xsd:attribute
			name="maxAngle"
			type="xsd:double"
			use="required"
			/>
		<xsd:attribute
			name="acceleration"
			type="xsd:double"
			use="required"
			/>
	</xsd:attributeGroup>

	<xsd:complexType name="Car">
		<xsd:attributeGroup ref="carAttrs"/>
		<xsd:attribute
			name="class"
			type="xsd:string"
			use="required"
			/>
```

```
            <xsd:attribute
                  name="amount"
                  type="xsd:positiveInteger"
                  default="1" />
      </xsd:complexType>

      <xsd:complexType name="Player">
            <xsd:attributeGroup ref="carAttrs"/>
      </xsd:complexType>
</xsd:schema>
```

Приложение 4. Отрывок из файла `trace.log`

```
-----
player = (400,000000, 300,000000)
police1 = (800,866150, 149,679163)
police2 = (803,037886, 637,612778)
police3 = (805,466110, 285,012364)
police4 = (817,867361, 561,561097)
police5 = (835,906095, 210,802486)
-----
player = (400,149520, 299,988013)
police1 = (800,971579, 149,621851)
police2 = (803,156270, 637,593148)
police3 = (805,535113, 284,914187)
police4 = (817,850445, 561,442296)
police5 = (835,997887, 210,725194)
-----
player = (400,882791, 299,833251)
police1 = (801,479220, 149,302223)
police2 = (803,740586, 637,457368)
police3 = (805,847962, 284,402341)
police4 = (817,728073, 560,855025)
police5 = (836,431110, 210,310248)
-----
player = (402,143128, 299,352447)
police1 = (801,870460, 149,024137)
police2 = (804,201315, 637,322724)
police3 = (806,074974, 283,979416)
police4 = (817,604009, 560,391335)
police5 = (836,758629, 209,959347)
-----
player = (405,808493, 296,772297)
police1 = (801,870460, 149,024137)
```

Приложение 5. Файл `DrawLog.java`

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.StringTokenizer;
import javax.imageio.ImageIO;
import javax.vecmath.Point2d;

public class DrawLog {
	private static Point2d lastP = null;
	private static Point2d[] lastPolice = new Point2d[6];

	private static void drawPlayer(Graphics g, Point2d p) {
		g.setColor(Color.RED);
		if(lastP == null) {
			g.fillOval((int) p.x - 5, (int) p.y - 5, 10, 10);
			lastP = p;
		} else {
			g.drawLine((int) p.x, (int) p.y,
					(int) lastP.x, (int) lastP.y);
			lastP.x = p.x;
			lastP.y = p.y;
		}
	}

	private static void drawPolice(Graphics g, Point2d p, char i) {
		int n = i - '1';
		switch(i) {
			case '1': g.setColor(Color.BLUE); break;
			case '2': g.setColor(Color.YELLOW); break;
			case '3': g.setColor(Color.GRAY); break;
			case '4': g.setColor(Color.CYAN); break;
			case '5': g.setColor(Color.GREEN); break;
			case '6': g.setColor(Color.PINK); break;
			default: g.setColor(Color.BLUE); break;
		}
		if(lastPolice[n] == null) {
			lastPolice[n] = new Point2d(p.x, p.y);
			g.fillOval((int) p.x - 5, (int) p.y - 5, 10, 10);
		} else {
			g.drawLine((int) p.x, (int) p.y,
					(int) lastPolice[n].x, (int) lastPolice[n].y);
			lastPolice[n].x = p.x;
			lastPolice[n].y = p.y;
		}
	}

	private static Point2d readP(StringTokenizer tokenizer) {
		String t = tokenizer.nextToken();
		double x = Double.parseDouble(t);
		x = (x + 300)/2; //Params are manually adjusted for every image.
		tokenizer.nextToken();
		t = tokenizer.nextToken();
		double y = Double.parseDouble(t);
		y = (y + 500)/2; //Params are manually adjusted for every image.
		return new Point2d(x, y);
```

```
	}

	private static void drawExplosion(Graphics g, Point2d p) {
		g.setColor(Color.BLACK);
		g.fillOval((int) p.x - 5, (int) p.y - 5, 10, 10);
	}

	public static void main(String[] args) {
		for(int i = 0; i < 6; i++) lastPolice[i] = null;
		try {
			BufferedImage my = new BufferedImage(1000, 1000,
					BufferedImage.TYPE_INT_RGB);
			Graphics g = my.getGraphics();
			g.setColor(Color.WHITE);
			g.fillRect(0, 0, 1000, 1000);
			FileReader reader = new FileReader("trace.log");
			BufferedReader bufferedReader = new BufferedReader(reader);
			String nextLine;
			while ((nextLine = bufferedReader.readLine()) != null) {
				StringTokenizer tokenizer = new StringTokenizer(
					nextLine, "=() ,");
				String who = tokenizer.nextToken();
				if(who.equals("explosion")) {
					String t = tokenizer.nextToken();
					int i = Integer.parseInt(t);
					lastPolice[i-1] = null;
					drawExplosion(g, readP(tokenizer));
				} else if(who.equals("player")) {
					drawPlayer(g, readP(tokenizer));
				} else if(who.contains("police")){
					drawPolice(g, readP(tokenizer), who.charAt(6));
				} else if(who.contains("crash")) {
					break;
				}
			 }
			 bufferedReader.close();
			 ImageIO.write(my, "png", new File("new.png"));
			 System.out.println("Done");
		} catch (FileNotFoundException e1) {
			e1.printStackTrace();
		} catch (IOException e2) {
			e1.printStackTrace();
		}
	}
}
```