Санкт-Петербургский государственный университет информационных технологий, механики и оптики
Факультет информационных технологий и программирования
Кафедра компьютерных технологий

Д.А. Паращенко, Ф.Н. Царев, А.А. Шалыто

Технология моделирования одного класса мультиагентных систем на основе автоматного программирования на примере игры «Соревнование летающих тарелок»

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

Санкт-Петербург
2006

Оглавление

Введение

Автоматное программирование (*SWITCH*-технология) [1, 2] акцентирует внимание разработчиков на *проектировании* программ. Иными словами, процесс реализации нового проекта начинается не с запуска среды разработки и написания кода `public static void main(String[] args)`, а с изучения предметной области, инструментального средства *UniMod* для поддержки автоматного программирования (*SWITCH*-технологии), выделения поставщиков событий и объектов управления. Затем проектируется один или несколько автоматов, получающих события и формирующих воздействия на объекты управления. Для каждого из автоматов создается схема связей и граф переходов.

Такой подход к проектированию программ позволяет выявить и устранить множество возникающих неясностей в постановке задачи, а также предусмотреть весьма неочевидные детали поведения системы. На сайте http://is.ifmo.ru размещено большое количество материалов, посвященных автоматному программированию, а также проекты, построенные по *SWITCH*-технологии и выполненные в рамках «Движения за открытую проектную документацию» [3] (http://is.ifmo.ru/works/open_doc/).

Для этих проектов необходимым требованием является опубликование проектной документации, по которой практически любой человек сможет понять и модифицировать программу, причем выполнит это существенно проще, чем в случае, когда она написана и документирована традиционным путем.

Проект *UniMod* [4, 5] (http://unimod.sourceforge.net) с открытым исходным кодом позволяет визуально проектировать и реализовывать автоматные программы.

Инструментальное средство *UniMod* базируется на трех «китах»: унифицированном языке моделирования *UML*, *SWITCH*-технологии и открытой среде разработки *Eclipse* [6].

Вклад *SWITCH*-технологии в этот проект заключается в том, что при ее использовании программы строятся так, как автоматизируются объекты управления в промышленности. В рамках этой технологии строится схема связей, которая состоит из источников (поставщиков) событий, системы управления (в общем случае, система взаимосвязанных автоматов) и объектов управления, на которые с целью выполнения действий воздействуют автоматы. Объекты управления, в свою очередь, используя входные переменные, могут формировать сигналы обратной связи в автоматы. Схема связей задает интерфейс автоматов. Новизна построения схемы связей в рассматриваемом проекте относительно «классической» *SWITCH*-технологии состоит в том, что в проекте эта схема является диаграммой классов, а раньше она была только схемой, предназначенной для описания интерфейса автомата – указания названий каждой входной и выходной переменной.

Такой принцип проектирования впервые в *Open Source* позволяет решить задачу построения программы в целом, а не только генерации кода по графам переходов, как это делается, например, с помощью сорока трех инструментальных средств, приведенных в энциклопедии Wikipedia (http://en.wikipedia.org/wiki/Finite_state_machine).

Среда разработки *Eclipse* (http://eclipse.org, http://pcweek.ru/?ID=604842) в рассматриваемом проекте используется для написания программ на языке программирования *Java*. При этом *UniMod* является встраиваемым модулем (plug-in) для этой среды. Он совместно разработан компанией *eVelopers Corporation* (http://www.evelopers.com) (http://www.pcweek.ru/?ID=504874, http://is.ifmo.ru/works/uml-switch-eclipse/) и кафедрой «Технологии программирования» СПбГУ ИТМО.

Классы, реализующие функциональность объектов управления и поставщиков событий, пишутся вручную на языке *Java*.

Существуют две возможности реализации программы. При интерпретационном подходе по *UML*-диаграммам **(**диаграммам классов, выполненных в виде схем связей, и диаграммам состояний (графам переходов)**)** автоматически создается *XML*-описание автоматов, которое затем поступает на вход интерпретатора – одному из компонентов инструментального средства *UniMod*. При компиляционном подходе по указанным диаграммам автоматически генерируется реализующий систему автоматов код на языке *Java*, который объединяется с остальной частью программы, написанной вручную.

Отметим, что в рассматриваемом проекте применяется новый смешанный, компиляционно-интерпретационный подход, который описан в разд. 2.2. Разработка этого подхода связана с тем, что в первой версии инструментального средства *UniMod* не было возможности красиво реализовать большое количество объектов одного класса. В связи с этим, авторами совместно с В.С. Гуровым (одним из разработчиков UniMod) было предложено для классов с одним объектом использовать компиляционный подход, а для классов, содержащих более одного объекта использовать интерпретационный подход.

С первым опытом применения инструментального средства *UniMod* можно ознакомиться в работе [7].

Опишем, как в данном случае работает приложение. Исходно автомат находится в начальном состоянии. События, формируемые поставщиками, обрабатываются автоматами в соответствии с их графами переходов. Соответствующий автомат при поступлении события может проверить различные логические условия и при выполнении одного из них выбрать переход в новое состояние. С переходом может быть ассоциирован набор выходных воздействий на объекты управления. Действия могут выполняться также при входе в новое состояние. В

состояниях могут указываться также вложенные в них автоматы. При поступлении определенного события соответствующий автомат может перейти в конечное состояние, завершив тем самым работу приложения.

В данной работе на основе инструментального средства *UniMod* разрабатывается технология моделирования одного класса мультиагентных систем [8]. Ее применение иллюстрируется на примере игровой программы, решающей задачу «Соревнование летающих тарелок» заочного тура Всесибирской олимпиады по информатике 2005 года [9]. Автором задачи является Дмитрий Иртегов (fat@nsu.ru). В настоящей работе в постановку задачи внесены некоторые изменения. Оригинальный текст условия задачи и список изменений приведены в разд. 2.6 и 2.7.

На сайте http://is.ifmo.ru/ приведены студенческие проекты, в которых конечные автоматы применяются для реализации искусственного интеллекта [10–15]. Настоящая работа является развитием этих работ. Существенным ее отличием является то, что наряду с искусственным интеллектом создаваемого агента также проектируется и внешняя среда (мир, в котором «живут» агенты). Еще одним отличием от указанных выше работ является то, что в данной работе искусственный интеллект разрабатывается для управления группой агентов, действующих для достижения общей цели.

Логически программа состоит из двух частей. Первая из них реализует ядро, включающее в себя интерфейс пользователя и систему моделирования внешней среды. Интерфейс пользователя и внешняя среда реализованы практически независимо, но в одно из «интерфейсных» состояний вложен автомат, управляющий моделью внешней среды. Вторая часть программы – это система управления летающими тарелками, основанная на мультиагентном подходе.

1. Описание внешней среды

Соревнование проводится между двумя командами летающих тарелок. Цель соревнований состоит в том, чтобы одна из тарелок команды переместилась на максимальное расстояние от линии старта. Состязание проходит на трассе, представляющей собой полубесконечную (бесконечную в одну сторону) полосу шириной 40 метров. Маневры, связанные с изменением высоты полета, не допускаются (таким образом, трасса соревнования двумерна).

Каждая команда состоит из `N` тарелок (агентов). Соревнующиеся команды назначаются пользователем из перечня имеющихся в наличии команд. Две команды с простыми стратегиями были созданы авторами с использованием традиционного метода, а разработке поведения третьей команды и среды, в которой соревнования проходят, посвящена настоящая работа.

В дальнейшем, кроме термина «соревнование», будем использовать термин «гонка».

В начале гонки агенты первой команды располагаются в воздухе случайным образом на некотором расстоянии от линии старта в левой половине трассы, которая на экране расположена горизонтально. Вторая команда размещается симметрично первой на правой половине трассы.

Для каждого агента заданы начальная скорость и направление движения. В простейшем случае начальные скорости всех агентов одинаковы, а направления – строго вперед. Система также позволяет делать начальные скорости и направления различными. Агенты в процессе полета могут поворачивать тем самым, мешая движению других агентов.

Каждый агент имеет определенный запас топлива, расходуемого в процессе движения. По команде «Старт» все агенты начинают движение с целью максимально удалиться от линии старта. Агенты в процессе полета

могут изменять скорость своего движения за счет изменения расхода топлива.

Летающие тарелки, покинувшие трассу, считаются прекратившими гонку. Выходом за пределы коридора считается пересечение центром летающей тарелки границы трассы.

Управление каждой командой выполняет программа, написанная на языке *Java*. Как отмечалось выше, в настоящей работе разрабатывается внешняя среда и одна из команд с использованием инструментального средства *UniMod*, созданного для поддержки автоматного программирования.

Для каждой новой команды, которая будет разрабатываться в дальнейшем, необходимо разработать соответствующую управляющую программу. При написании этой программы могут быть использованы любые программные средства, а не только *UniMod*.

1.1. Правила соревнований

В каждом соревновании каждая из команд на старте имеет восемь летающих тарелок с полным запасом топлива (10 $см^3$). Исходно тарелки первой команды случайным образом располагаются на первых 25 метрах левой половины трассы. Тарелки второй команды располагаются симметрично им в правой половине трассы (рис. 1).

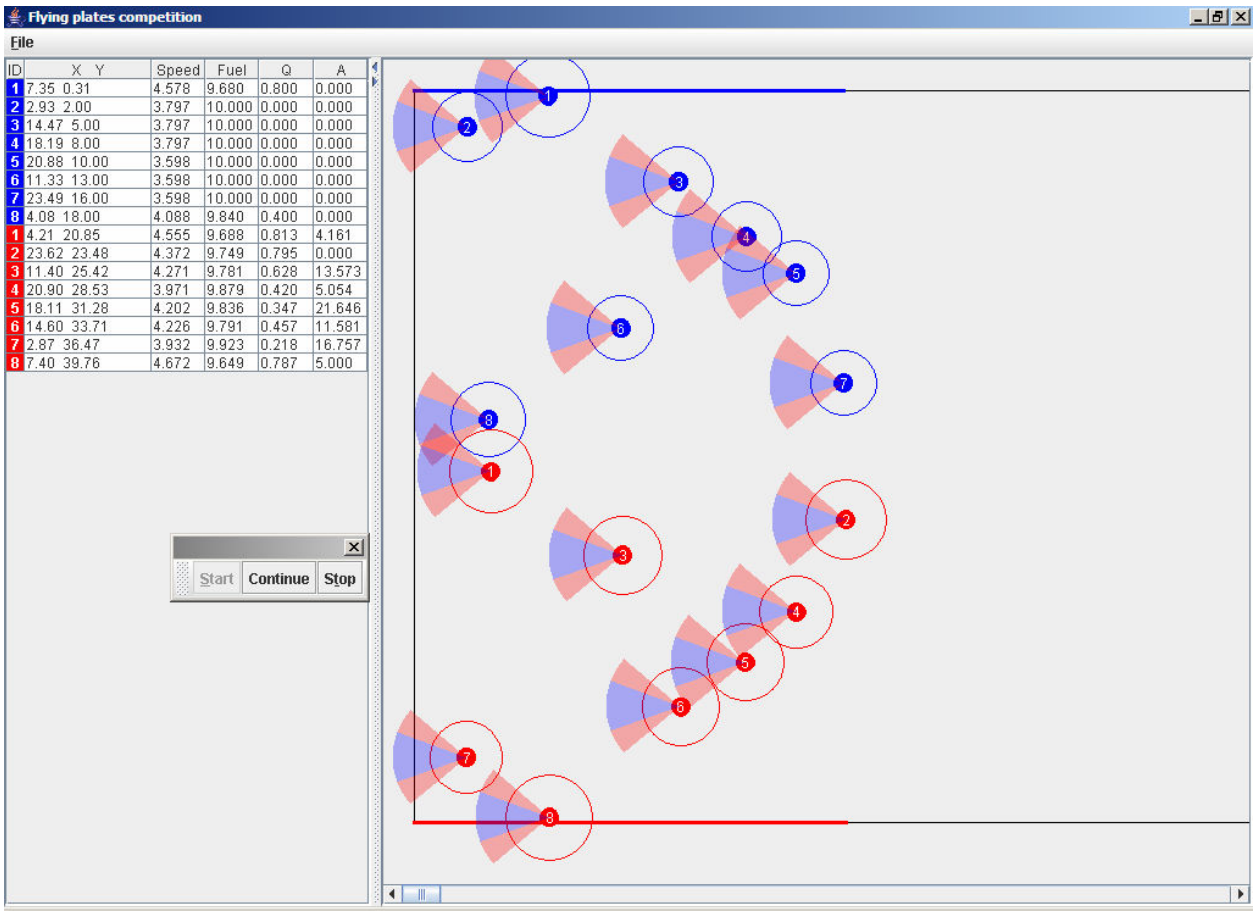

Рис. 1. Летающие тарелки на старте

В левой части экрана размещается таблица, в которой отображены параметры летающих тарелок в текущий момент времени: координаты, скорость, запас топлива, расход топлива, угол поворота аэродинамических рулей.

На этом рисунке слева также показаны кнопки управления моделированием:

- кнопка «Start» начинает процесс моделирования;
- кнопка «Continue» (на рис. 1 не показана) запускает процесс моделирования, при этом моделирование продолжается из того состояния, в котором оно было приостановлено;
- кнопка «Pause» приостанавливает процесс моделирования;
- кнопка «Stop» останавливает процесс моделирования.

При нажатии на кнопку «Pause» она заменяется кнопкой «Continue», и наоборот.

Жизненный цикл летающей тарелки выглядит следующим образом. Она может находиться в одном из трех состояний: «Полет», «Нормальное завершение гонки», «Аварийное завершение гонки» (рис. 2).

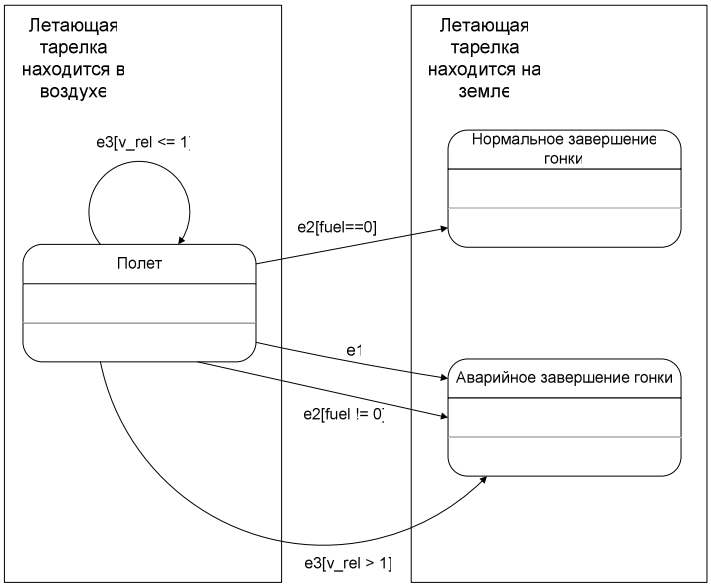

Рис. 2. Возможные состояния летающей тарелки и переходы между ними

Обозначения, используемые на рис. 2, приведены в табл. 1.

Таблица 1. Используемые обозначения

Обозначение	**Описание**
`e1`	Летающая тарелка покинула пределы трассы (ее центр пересек границу трассы)
`e2`	Скорость летающей тарелки стала меньше, чем один м/с
`e3`	Летающая тарелка столкнулась с другой летающей тарелкой
`v_rel`	Относительная скорость столкновения летающих тарелок
`fuel`	Количество топлива, которое осталось у летающей тарелки

Поясним поведение летающей тарелки. В начале гонки она находится в воздухе, исправна и способна продолжать участие в гонке. Этому соответствует состояние «Полет».

При выходе летающей тарелки за пределы трассы (событие `e1`) она завершает гонку аварийно.

Если скорость летающей тарелки падает ниже одного м/с (событие `e2`), и ее топливный бак не пуст (условие `fuel != 0`), то она завершает гонку аварийно. Если же при падении скорости ниже одного м/с (событие `e2`) топливный бак летающей тарелки пуст (условие `fuel == 0`), то она нормально завершает гонку.

Если летающая тарелка сталкивается с другой тарелкой (событие `e3`), то при относительной скорости столкновения, большей одного м/с (условие `v_rel > 1`), тарелка аварийно завершает гонку. При относительной скорости столкновения, не превышающей одного м/с (условие `v_rel <= 1`), тарелка продолжает полет.

Заметим, что поскольку начальный запас топлива у каждой тарелки конечен, то рано или поздно все тарелки обеих команд завершат гонку.

При подведении итогов гонки учитываются только результаты тарелок, нормально ее завершивших. Результатом команды считается наибольшее из расстояний, на которое удалились от линии старта ее летающие тарелки, нормально завершившие гонку. Если все летающие тарелки команды вышли из гонки аварийно, результат команды считается равным нулю. Победителем признается команда, прошедшая наибольшее расстояние. В случае равенства результатов гонка считается завершившейся вничью.

1.2. Динамика летающей тарелки

Летающая тарелка представляет собой дискообразное "летающее крыло" радиусом один метр. На рис. 3 представлен вид сверху летающей тарелки.

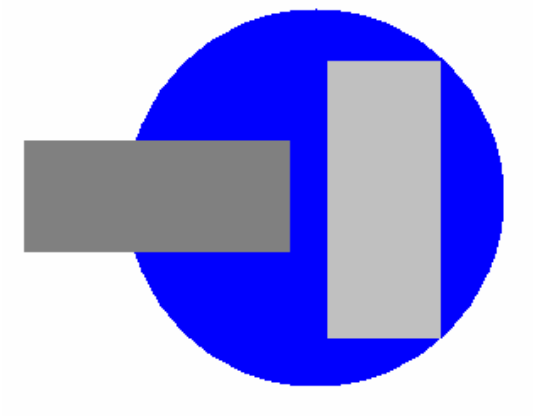

Рис. 3. Летающая тарелка

Тарелка имеет реактивный двигатель (на рис. 3 он условно показан горизонтальным прямоугольником), топливный бак (вертикальный прямоугольник) емкостью 15 см^3, аэродинамические рули и бортовой компьютер, способным регулировать расход топлива (и, как следствие, тягу двигателя) и положение аэродинамических рулей. Эти рули позволяют тарелке маневрировать. Тарелка может передвигаться со скоростями от одного метра в секунду. Максимальная скорость тарелки зависит от запаса топлива и сопротивления воздуха. Ограничение в один метр в секунду вызвано тем, что летающая тарелка с меньшей скоростью не может держаться в воздухе.

Летающая тарелка движется в соответствии со вторым законом Ньютона. Ее движение определяется двумя силами: сопротивлением воздуха `F` и тягой двигателя `T`. Если тяга не равна сопротивлению воздуха, то летающая тарелка движется с ускорением, которое может быть положительным (если тяга больше сопротивления воздуха) или отрицательным (если сопротивление воздуха больше тяги).

Ускорение определяется по формуле $a = \frac{T - F}{m}$, где `m` – масса летающей тарелки. При этом считается, что изменение массы тарелки за счет выгорания горючего пренебрежимо мало.

Сопротивление воздуха определяется по формуле $F = c_1 + c_2 v^2$, где `v` – скорость тарелки, а коэффициенты c_1 и c_2 определяются ее аэродинамическими характеристиками и одинаковы для всех тарелок обеих команд.

Тяга двигателя определяется по формуле $T = c_4 q$, где `q` – расход топлива в сантиметрах кубических в секунду. Расход топлива находится под контролем бортового компьютера тарелки, что позволяет изменять расход от нуля до единицы. Константа c_4 определяется характеристиками двигателя тарелки и одинакова для всех тарелок обеих команд.

Аэродинамические рули позволяют летающей тарелке поворачивать относительно ее текущего направления движения на угол, не превышающий 25°.

1.3. Аэродинамическое взаимодействие между летающими тарелками

При полете летающей тарелки от траектории ее полета в направлениях назад и в стороны под углом около 30° распространяются конические вихревые потоки воздуха. Если другая тарелка попадет в этот вихрь, то сопротивление воздуха ее полету резко **снизится** (рис. 4).

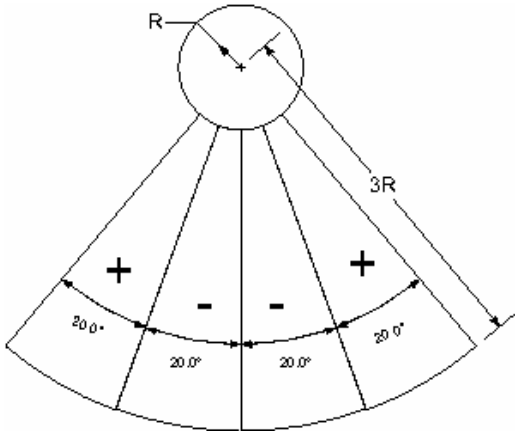

Рис. 4. Области аэродинамического взаимодействия

Отметим что, летающая тарелка, находящаяся за хвостом (два сектора по 20°) другой летающей тарелки, испытывает **дополнительное сопротивление** движению, обусловленное реактивной струей.

Поясним, как учитывается изменение сопротивления воздуха. Если центр второй летающей тарелки находится в областях, отмеченных на рис. 4 знаком "+", сопротивление воздуха ее движению падает на 50%. Если же центр второй тарелки находится в области, помеченной знаком "–", сопротивление воздуха возрастает на 50%.

Аэродинамические воздействия от нескольких летающих тарелок складываются, так что в зоне, отмеченной на рис. 5 знаками "++", сопротивление воздуха вообще отсутствует, а в зонах, помеченных знаком "0", воздействия компенсируют друг друга. При этом в результате наложения зон воздействия от трех и более летающих тарелок сопротивление воздуха не может стать отрицательным.

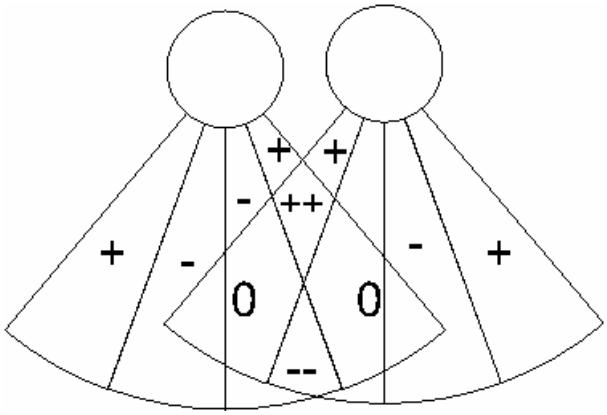

Рис. 5. Наложение областей аэродинамического взаимодействия от двух летающих тарелок

Учитывая изложенное, вычисление сопротивления воздуха происходит следующим образом. Пусть N_+ – количество тарелок, уменьшающих сопротивление воздуха в этой области, а N_- – количество тарелок, увеличивающих сопротивление воздуха. Пусть $\Delta N = N_+ - N_-$. Если $\Delta N = 0$, то в этой области нормальное аэродинамическое сопротивление, если $\Delta N = 1$ или $\Delta N = 2$, то сопротивление понижается на $50\Delta N$ процентов. Если ΔN отрицательно, то сопротивление в этой области повышается на $50|\Delta N|$ процентов.

1.4. Столкновение летающих тарелок

При столкновении двух тарелок происходит их абсолютно упругое соударение без передачи вращательного момента. Если относительная скорость столкновения была более одного метра в секунду, то обе участвовавшие в столкновении летающие тарелки повреждаются и начинают терять высоту. При этом они обе аварийно завершают гонку.

Под относительной скоростью столкновения понимается проекция векторной разности скоростей летающих тарелок на прямую, проходящую через центры летающих тарелок в момент столкновения (рис. 6). Вектор V_{rel} соответствует относительной скорости.

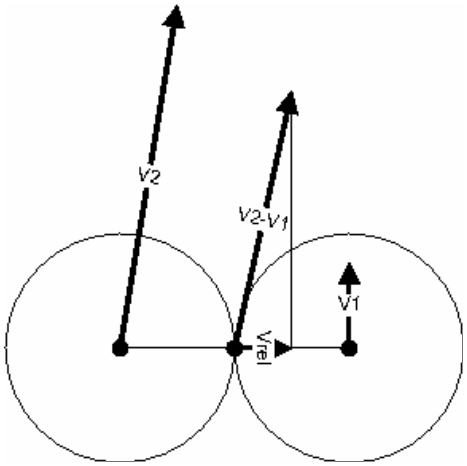

Рис. 6. Относительная скорость столкновения

1.5. Моделирование гонки

Моделирование гонки происходит по ходам, каждый из которых занимает `t` миллисекунд (параметр `t` читается из конфигурационного файла). В начале каждого хода игроки обладают информацией о координатах и скоростях всех летающих тарелок. Каждому игроку предоставляется возможность установить расход топлива и угол поворота каждой тарелки своей команды.

Каждые `t` миллисекунд (один ход) происходит обновление параметров. Покажем, как выполняется моделирование полета летающих тарелок за время одного хода.

1. Снятие с соревнования летающих тарелок, движущихся со скоростью, меньшей одного метра в секунду. При завершении полета летающими тарелками с пустыми баками пройденные ими расстояния засчитываются в результат команды.
2. Расчет ускорений летающих тарелок в соответствии с установленными расходами топлива и углами поворотов, а также аэродинамическим сопротивлением. Расчет новых скоростей летающих тарелок по формуле $\vec{V}_{temp} = \vec{V}_{old} + \vec{a} \cdot \Delta t$, где `V`$_{\texttt{temp}}$ –

вектор скорости летающей тарелки после учета ускорения, V_{old} – вектор старой летающей тарелки, a – вектор ускорения летающей тарелки. После этого происходит поворот вектора скорости на угол равный углу поворота аэродинамических рулей (рис. 7). В результате поворота получается вектор новой скорости летающей тарелки V_{new}.

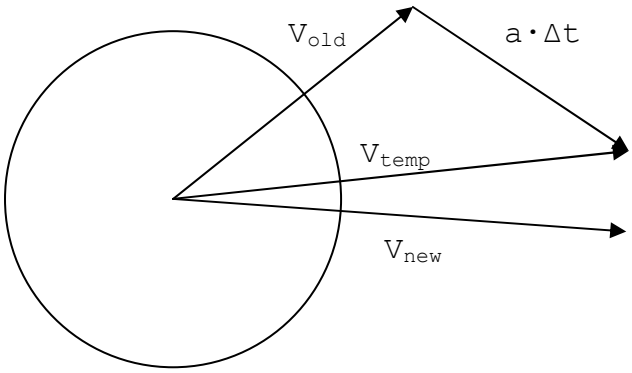

Рис 7. Пересчет скорости летающей тарелки на шаге моделирования

3. Снятие летающих тарелок, движущихся медленнее одного метра в секунду. Как и ранее, при завершении полета летающими тарелками с пустыми баками, пройденные ими расстояния засчитываются в результат команды.
4. Происходит равномерное прямолинейное движение летающих тарелок (считается, что за время шага моделирования скорости тарелок не меняются). Если при этом происходит соударение тарелок – расстояние между центрами каких-либо двух тарелок становится меньше двух метров, то их скорости и координаты изменяются в соответствии с законами сохранения импульса и энергии. При этом летающие тарелки, относительная скорость столкновения которых превосходила один метр в секунду, выбывают из гонки.
5. Проверка того, что все летающие тарелки находятся в пределах трассы. При выходе центра тарелки за пределы трассы, она выбывает из гонки.

Гонка продолжается до тех пор, пока ее не завершила хотя бы одна тарелка. После того, как ее закончит и эта тарелка, гонка завершается.

1.6. Отличия от задачи, предложенной на Всесибирской олимпиаде 2005 года

Для удобства визуализации оригинальные правила, предлагавшиеся на Всесибирской олимпиаде [8], были изменены авторами рассматриваемой работы. Приведем список изменений:

- в задаче на Всесибиркой олимпиаде моделирование производилось по шагам, каждый из которых занимал одну секунду, в рассматриваемой задаче шаг моделирования произволен и задается с точностью до миллисекунды;
- в задаче на Всесибиркой олимпиаде количество летающих тарелок в каждой из команд было равно десяти, а в условиях рассматриваемой задачи это количество произвольно (оно читается из конфигурационного файла).

Отметим, в чем состоит различие понятий «ход» и «шаг» в настоящем документе. Под **шагом** моделирования подразумевается квант времени в программе, все временные промежутке в программе должны быть ему кратны. При этом передача управления системам управления летающими тарелками (то есть разрешение каждой из них произвести **ход**) выполняется через промежутки времени равные шагу моделирования. Между двумя соседними ходами состояние внешней среды изменяется так, как будто между ними прошло время, равное шагу моделирования.

1.7. Интерфейс взаимодействия системы управления летающими тарелками и игрового мира (интерфейс `Manager`)

Система управления летающими тарелками должна быть реализована в виде класса на языке программирования *Java*. Класс должен реализовывать интерфейс `Manager` и может иметь произвольное имя.

Интерфейс `Manager` содержит три метода: `init(int player)`, `configManager(JDialog mainDialog)` и `doTurn()`.

Первый из этих методов вызывается перед началом гонки. Его задача состоит в том, чтобы инициализировать систему управления летающими тарелками. Параметр `player` при каждом вызове равен единице, либо двум и соответствует номеру команды, летающими тарелками которой будет управлять данная система управления.

Второй метод отвечает за отображение и функционирования окна настройки системы управления. Настройке могут подлежать любые параметры по усмотрению автора системы управления.

Третий метод отвечает за выполнение одного хода и вызывается на каждом шаге моделирования по времени. При этом система управления имеет доступ ко всем данным, относящимся к предыдущему ходу и его результатам.

2. Инструментальное средство *UniMod*

Открытое инструментальное средство *UniMod* [4, 5] базируется на трех открытых компонентах: унифицированном языке моделирования *UML*, *SWITCH*-технологии и среде разработки *Eclipse* [6]. Это средство является встраиваемым модулем (плагином) для указанной среды разработки.

UniMod позволяет строить и редактировать схемы связей и диаграммы состояний, обеспечивать проверку формальной корректности этих диаграмм, проводить отладку диаграмм в графическом режиме и т. д.

После построения диаграмм и автоматической проверки их корректности, по ним строится их *XML*-описание. Далее вручную пишутся следующие фрагменты программы на языке *Java*: для поставщиков событий – их объявления, инициализация и преобразование системных событий в автоматные, а для объектов управления – методы, реализующие входные переменные и выходные воздействия.

2.1. Интерпретационный и компиляционный подходы

Для реализации программ могут использоваться интерпретационный и компиляционный подходы.

При интерпретационном подходе (рис. 8) программист создает графическую *UML*-модель (схему связей и диаграммы переходов) и код на языке *Java*, реализующий функциональность источников событий и объектов управлений. Схема связей изображается в нотации диаграммы классов *UML*.

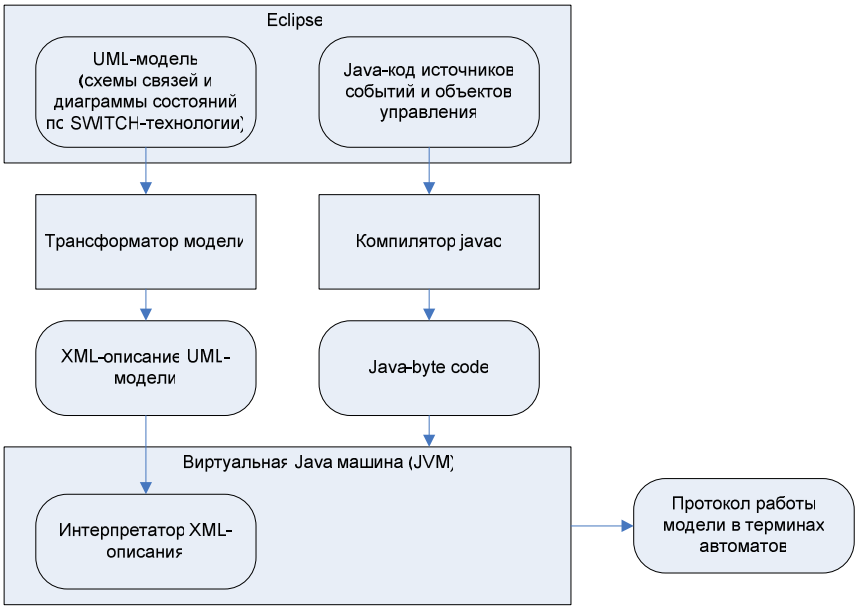

Рис. 8. Структурная схема интерпретационного подхода

Графическая *UML*-модель преобразуется в *XML*-описание, а код на языке *Java* компилятором *javac* преобразуется в байт-код. При работе программы *XML*-описание модели обрабатывается интерпретатором *XML*-описаний, который при необходимости вызывает методы, реализующие объекты управления и поставщики событий. В процессе работы программы ведется протокол в терминах автоматов.

При компиляционном подходе (рис. 9) программист также создает графическую *UML*-модель и пишет код на языке *Java*, который реализует функциональность объектов управления и поставщиков событий.

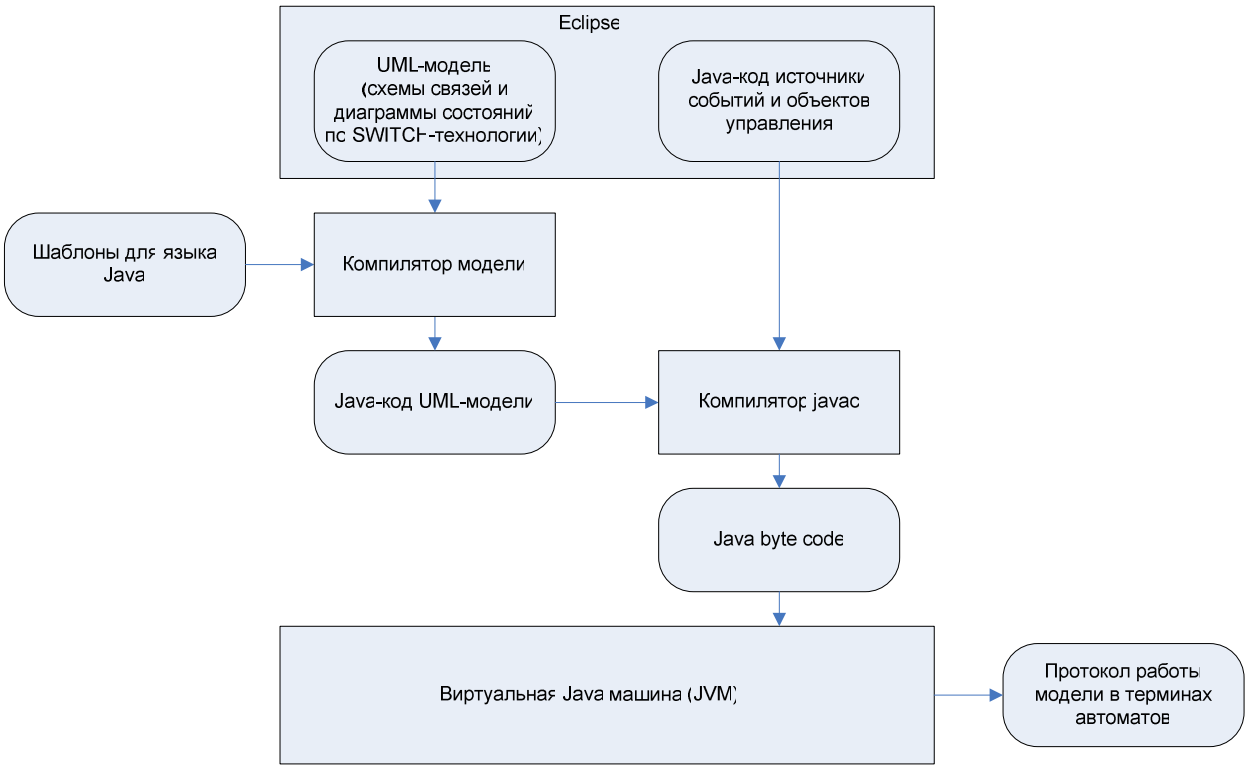

Рис. 9. Структурная схема компиляционного подхода

При помощи шаблонов *Velocity* *UML*-модель преобразуется в код на языке *Java*, который после этого с помощью компилятора *javac* совместно с кодом, написанным программистом вручную, преобразуется в байт-код, который исполняется виртуальной *Java*-машиной. Так же, как и в случае интерпретационного подхода, в процессе работы программы ведется протокол в терминах автоматов.

Использование шаблонов позволяет адаптировать компиляционный подход для языков программирования, отличных от языка *Java*.

2.2. Взаимодействие нескольких *UniMod*-моделей

В исходной версии инструментального средства *UniMod* имеется проблема, состоящая в том, что каждому объекту, входящему в класс, соответствует определенная модель. Так как каждая команда содержит `N` летающих тарелок, то необходимо было построить `N` одинаковых моделей, что крайне громоздко.

В результате консультаций с авторами проекта *UniMod* было принято решение для классов с одним объектом использовать компиляционный подход – преобразовывать модели в код на языке *Java* средствами *UniMod*, а для классов, содержащих более одного объекта (летающих тарелок) использовать интерпретационный подход. Во втором случае модели хранятся до выполнения программы в виде сгенерированных *XML*-описаний и их экземпляры создаются процессе работы программы из этих описаний. Такой подход был назван компиляционно-интерпретационным.

Каждая *UniMod*-модель состоит из одной схемы связей и произвольного числа диаграмм состояний, в то время как общая модель системы может содержать несколько *UniMod*-моделей. Связь между моделями выполняется через поставщики событий и объекты управления.

3. Структура программы

В данном разделе приводится высокоуровневое описание программы.

3.1. Составные части программы

В описываемой программе можно выделить три основные сущности: интерфейс пользователя, игровой мир и система управления летающими тарелками. Схема взаимодействия между ними показана на рис. 10. Система управления летающими тарелками присутствует на рисунке два раза, поскольку в игре участвуют две команды летающих тарелок – и для каждой из них необходима отдельная система управления.

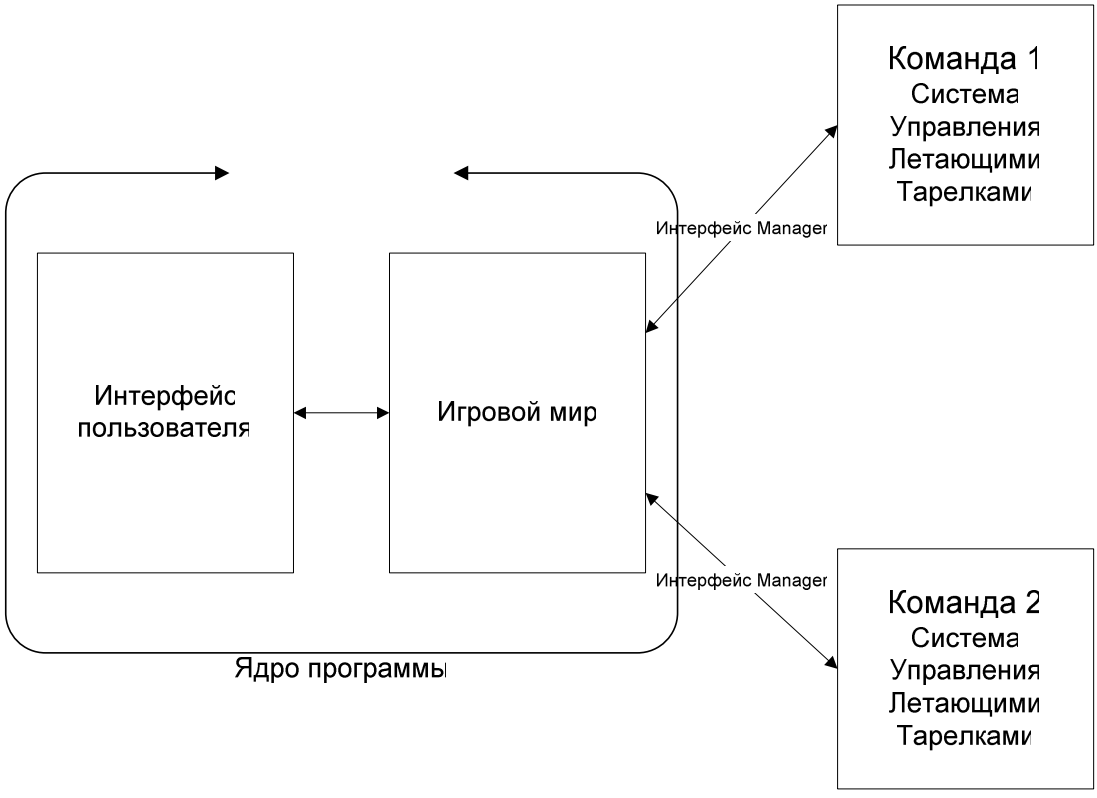

Рис. 10. Схема взаимодействия сущностей в программе

При этом сущности «Игровой мир» и «Интерфейс пользователя» неотделимы друг от друга и составляют ядро программы. Системы

управления летающими тарелками при этом являются встраиваемыми модулями (plug-in).

За счет этого достигается максимальная гибкость – разработчик стратегии может создавать такую внешнюю среду, какую он хочет. При этом главное – чтобы ее правила не противоречили глобальным правилам игры (разд. 2). При этом свобода заключается в том, что разработчик стратегии сам решает, какие данные летающей тарелке доступны, а какие – нет. За счет этого на базе одной и той же программы можно создавать системы управления летающими тарелками, базирующиеся на различных подходах: централизованном, мультиагентном и других. Авторы программы разработали систему управления летающими тарелками, основанную на мультиагентном подходе. Она описана в разд. 6.

В программе используется новый – компилятивно-интерпретационный подход. Конечные автоматы, находящиеся «внутри» ядра, реализуются на основе компилятивного подхода. Автоматы, находящиеся «внутри» летающей тарелки, реализуются на основе интерпретационного подхода.

3.2. Система управления летающими тарелками

Напомним, что задача разработки стратегии поведения летающих тарелок состоит в разработке для своей команды победной стратегии.

В ходе настоящей работы было сделано предположение, что победной будет стратегия, при которой одна половина агентов движется вперед и уклоняется от границ коридора и рядом находящихся агентов, а вторая половина агентов разбивается на пары. При этом агенты в парах движутся, поддерживая взаимное расположение для использования аэродинамического взаимодействия. Эта стратегия почти всегда оказывается победной по сравнению со стратегией, реализованной для

противника: один агент летит вперед, а другие – пытаются сбить агентов другой команды.

Разработанная авторами система управления летающими тарелками основана на автоматно-мультиагентном подходе. При таком подходе каждая летающая тарелка рассматривается в отдельности от других, а искусственный интеллект каждой летающей тарелки реализуется на основе *SWITCH*-технологии и конечных автоматов. Взаимодействие между летающими тарелками и игровым миром выполняется посредством сущности под названием «Внешняя среда». Поэтому итоговая схема взаимодействия всех сущностей в программе выглядит так, как показано на рис. 11.

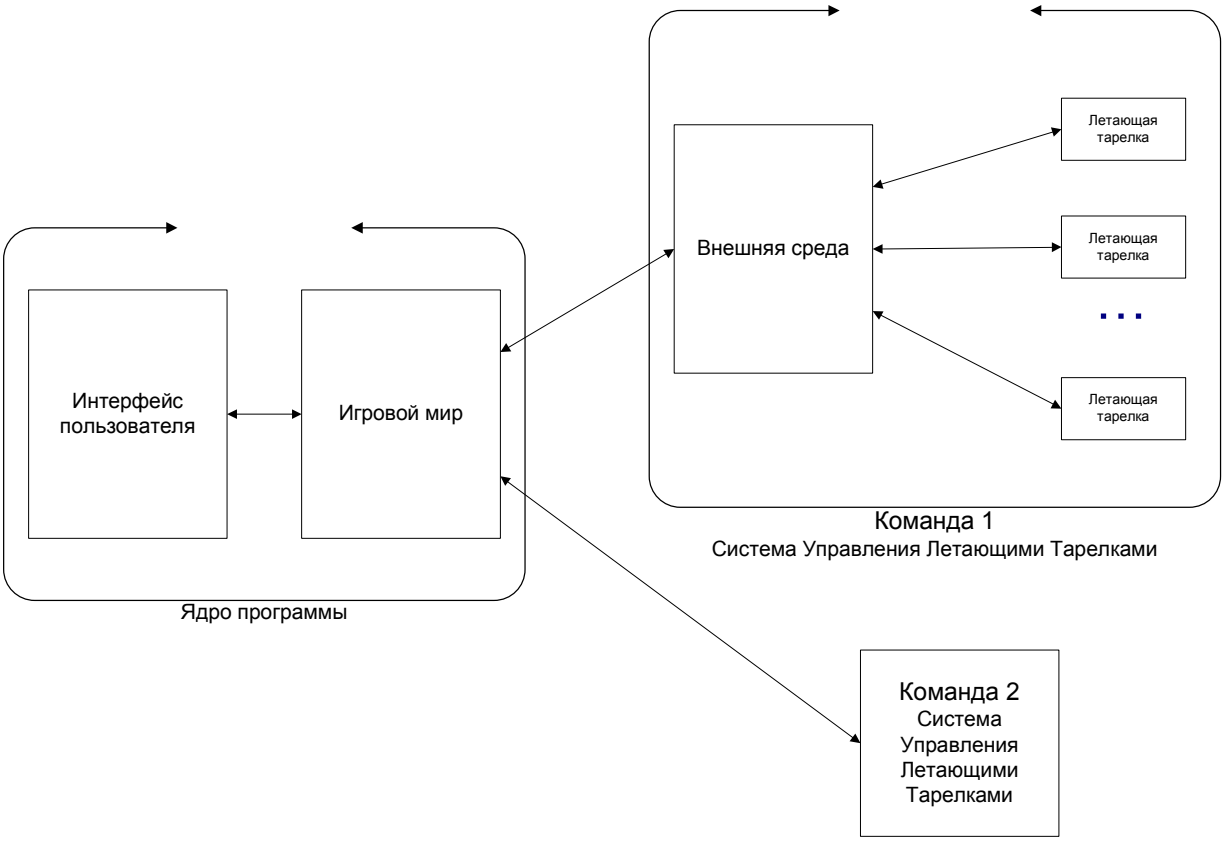

Рис. 11. Схема взаимодействия всех сущностей

При этом возникла необходимость в создании нескольких экземпляров летающей тарелки, каждая из которых управлялась бы своей системой конечных автоматов. По замыслу авторов, каждая летающая тарелка должна была управляться одинаковой системой автоматов.

Поэтому логично было бы спроектировать ее в единственном экземпляре, а потом – создать в необходимом количестве. Однако средствами *UniMod* последней на февраль 2006 года версии это сделать невозможно.

Одним из решений этой проблемы является создание восьми (по количеству летающих тарелок) одинаковых систем конечных автоматов с помощью *UniMod*. Однако это решение имеет множество недостатков таких, как, например, громоздкость и отсутствие гибкости – при изменении числа летающих тарелок пришлось бы добавлять новые автоматы или удалять лишние.

Вадим Гуров, один из разработчиков инструментального средства *UniMod*, предложил авторам более элегантное решение: создавать экземпляр системы управления летающей тарелкой из *XML*-описания *UniMod*-модели, а для связи двух *UniMod*-моделей использовать объект, являющийся объектом управления на одной модели и поставщиком событий – на другой. Детали этого подхода показаны в разд. 6.

4. Ядро программы

4.1. Основные функции

Как уже отмечалось, основными функциями ядра программы являются обеспечение взаимодействия пользователя с программой и обеспечение взаимодействия систем управления летающими тарелками с игровым миром. В связи с этим в ядре программы выделяются две части: интерфейс пользователя и система моделирования игрового мира. Их структура с точки зрения реализации на основе *SWITCH*-технологии описывается в разд. 4.4 – 4.7.

4.2. Взаимодействие пользователя с программой

Взаимодействие пользователя с программой состоит в том, что пользователь выбирает две соревнующиеся команды летающих тарелок,

отличающиеся системами управления (рис. 12); запускает, приостанавливает, завершает текущее соревнование; выбирает параметры визуализации процесса соревнования.

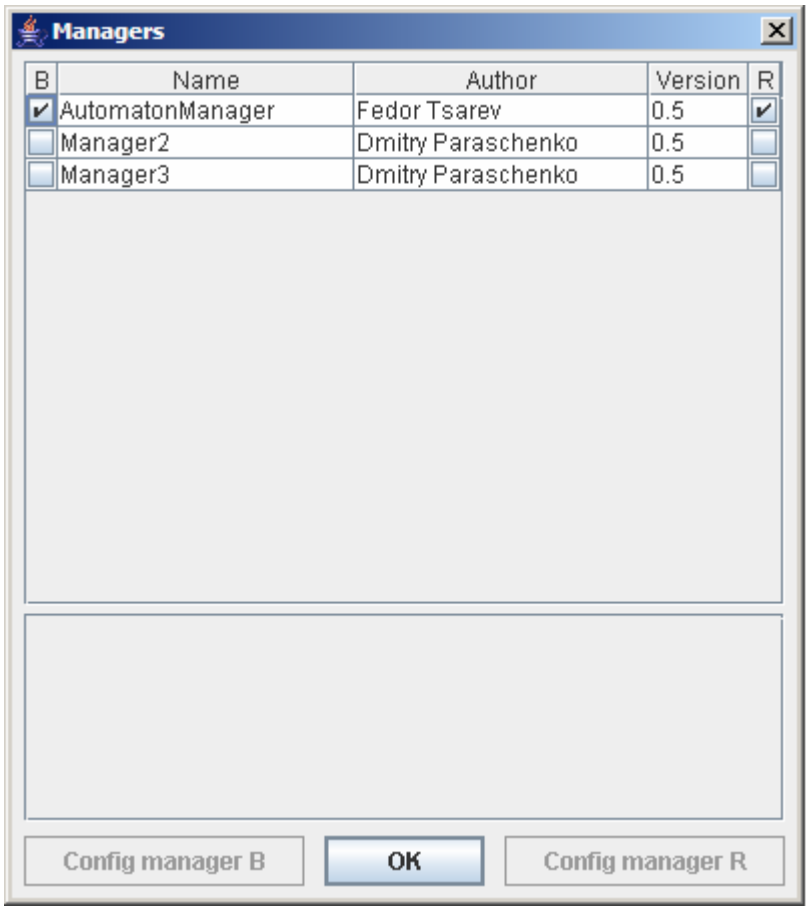

Рис. 12. Выбор соревнующихся систем управления летающими тарелками

Запуск, остановка и приостановка текущего соревнования выполняется с помощью кнопок `Start`, `Pause`, `Stop` панели инструментов, показанной в верхней части рис. 13.

Установка параметров визуализации процесса движения летающих тарелок выполняется с помощью меню, показанного на рис. 13.

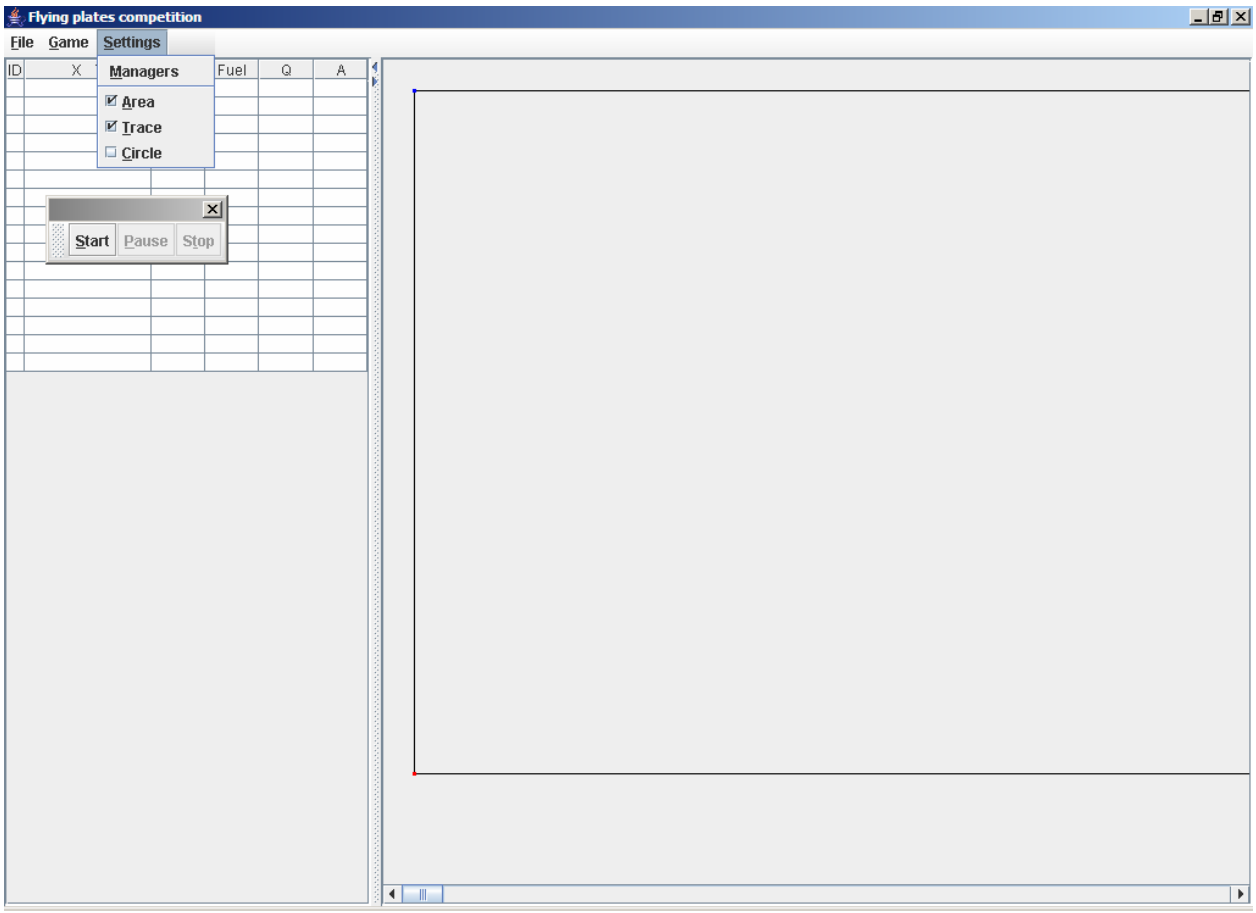

Рис. 13. Меню параметров визуализации и панель управления текущим соревнованием

4.3. Взаимодействие системы управления летающими тарелками с игровым миром

4.3.1. Проблема «нечестной игры»

Как упоминалось ранее (разд. 1.7), управление летающими тарелками обеспечивает система управления летающими тарелками, которая является программой на языке *Java*. Так как в процессе работы программы две системы управления летающими тарелками соревнуются, то возникает проблема обеспечения честности соревнований. Заключается она в следующем: поскольку обе системы управления летающими тарелками имеют доступ ко всей информации о летающих тарелках, каждая из которых представляет собой объект (в смысле языка программирования *Java*), необходимо как-то ограничивать возможность записи в некоторые поля этих объектов, не ограничивая возможности их чтения.

Авторы программы нашли решение этой проблемы. Каждая из летающих тарелок (объект типа `Plate`, приложение 1) хранит внутри себя множество объектов, которым разрешено менять ее поля. При этом установка полей выполняется методами с сигнатурами вида `void set???(Object setter, double value)`.

Внутри этих методов сначала проверяется принадлежность объекта `setter` множеству объектов, которые могут изменять поля этой летающей тарелки. Если выясняется, что объекту `setter` не разрешено изменять поля этой летающей тарелки, то генерируется ошибка `SecurityError`. При генерации этой ошибки пользователю сообщается о ней и предлагается завершить выполнение программы (рис. 14).

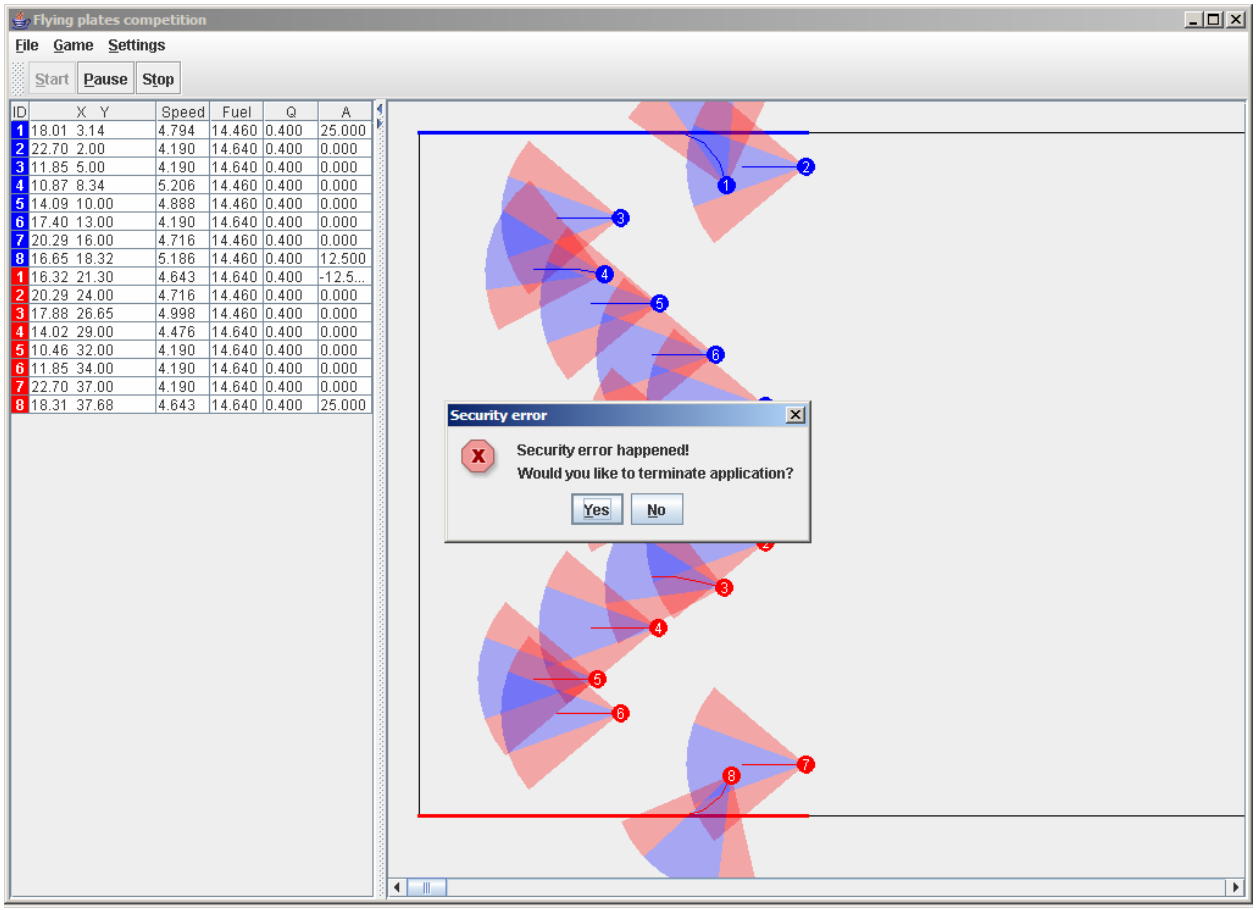

Рис. 14. Сообщение о том, что одна из команд ведет нечестную игру

4.4. Диаграмма связей

В отличие от *SWITCH*-технологии [1] при использовании инструментального средства *UniMod* схема связей является не картинкой, а диаграммой классов *UML*, изображенной нетрадиционным путем (она ориентирована не сверху вниз, а слева направо). Диаграмма связей показана на рис. 15.

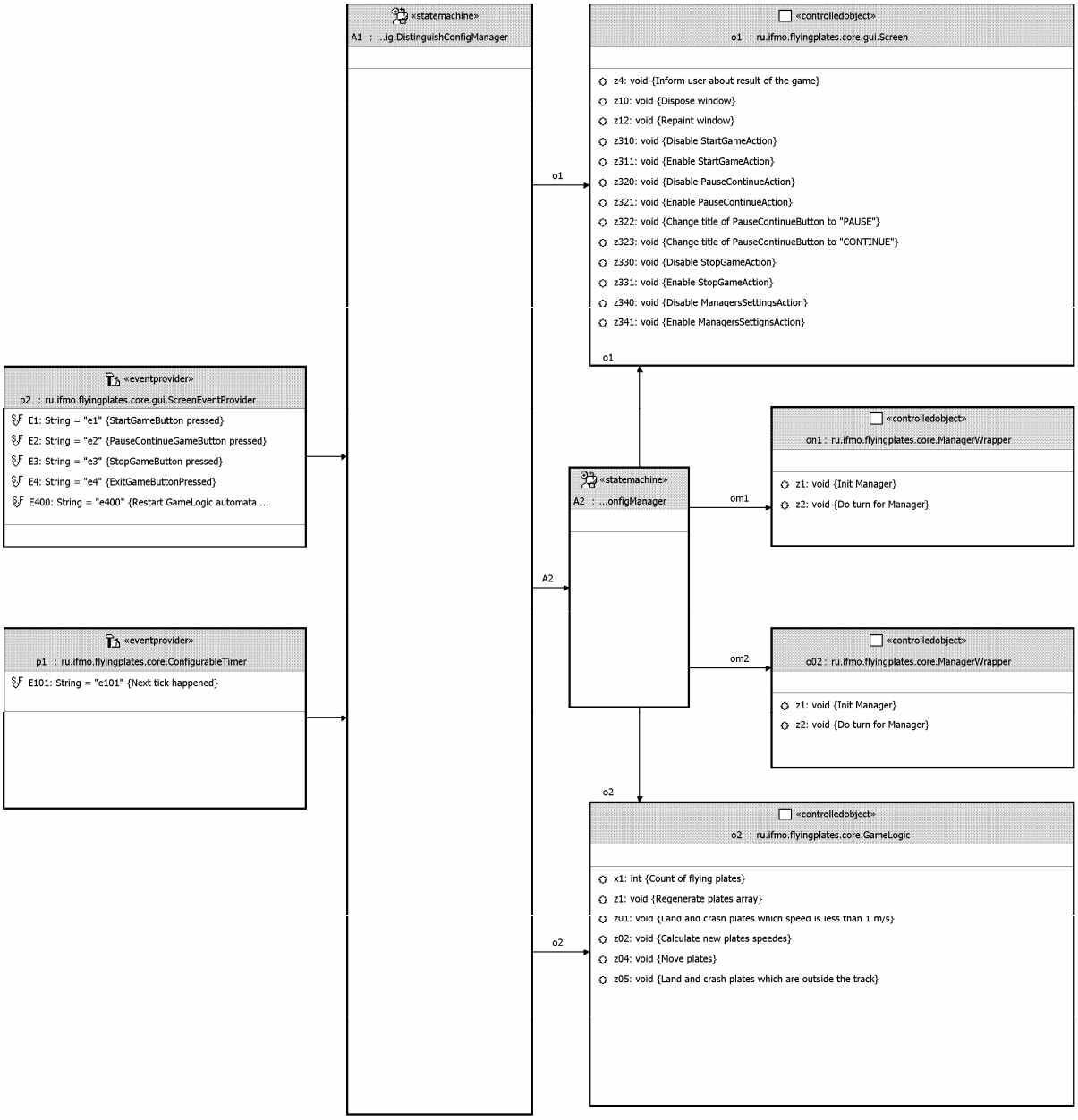

Рис. 15. Диаграмма связей

4.5. Поставщики событий

В этом разделе описываются поставщики событий (они находятся слева на диаграмме связей).

4.5.1. Поставщик событий `ConfigurableTimer`

Этот поставщик событий предназначен для синхронизации работы программы со временем. Этот поставщик событий генерирует одно и то же событие `e101` через каждые `Config.TIME_STEP` миллисекунд. Это событие используется для выполнения очередной итерации главного цикла игры.

Поставщик событий `ConfigurableTimer` является расширением стандартного поставщика событий `Timer`, входящего в средство *UniMod*, который может генерировать события лишь с интервалом в одну секунду (табл. 2).

Таблица 2. События поставщика событий `ConfigurableTimer`

	Событие	Описание	Параметр	Комментарий
1.	`e101`	Прошло `Config.TIME_STEP` миллисекунд с момента предыдущего события `e101`	нет параметров	

4.5.2. Поставщик событий `ScreenEventProvider`

Этот поставщик событий предназначен для передачи воздействий пользователя автомату `A1`, управляющему интерфейсом. Он реализует события, связанные с выбором различных пунктов меню пользователем (табл. 3).

Таблица 3. События поставщика событий `ScreenEventProvider`

	Событие	Описание	Параметр	Комментарий
1.	`e1`	Выбран пункт меню «Начать игру»	нет параметров	
2.	`e2`	Выбран пункт меню «Приостановить/продолжить игру»	нет параметров	
3.	`e3`	Выбран пункт меню «Остановить игру»	нет параметров	
4.	`e4`	Выбран пункт меню «Выйти из игры»	нет параметров	
5.	`e400`	Необходимо перезапустить автомат `A2`	нет параметров	

4.6. Автоматы

В этом разделе описаны автоматы.

4.6.1. Автомат `A1` – управление интерфейсом пользователя

Автомат `A1` с графом переходов, изображенным на рис. 16, управляет графическим интерфейсом пользователя (GUI).

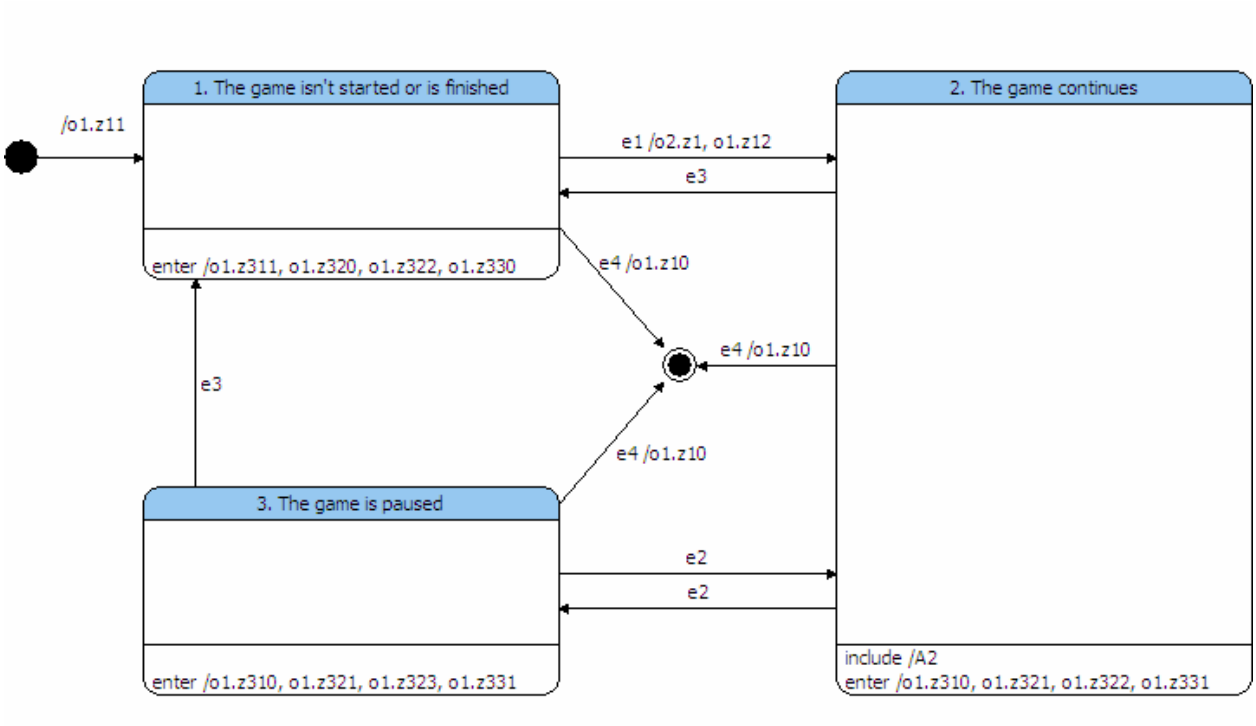

Рис. 16. Автомат `A1`

Начальное состояние обозначено черным кругом, а конечное – двойным кругом с закрашенной серединой. Состояния обозначены прямоугольниками со скругленными углами, а переходы – стрелками, рядом с которыми написано условие перехода. Общая форма условия перехода:

e# [логическое выражение] / список выходных воздействий

(вместо символа # стоит некоторое число).

Логическое выражение может состоять из:

- входных переменных в нотации *o#.x#* (здесь, как и выше, # обозначает любую цифру);
- логических операций `&&` (и), `||` (или), *!* (не);
- операций сравнения > (больше), < (меньше), >= (больше или равно), <= (меньше или равно), == (равно), != (не равно);
- скобок `(`, `)`.

Допустим, автомат находится в каком-либо состоянии. Если поступило событие, то для всех переходов по этому событию проверяется логическое условие (если оно присутствует). Если найден переход, для

которого логическое условие истинно, то сначала выполняются все выходные воздействия, указанные на переходе после символа /, а затем – выходные воздействия, предусмотренные для исполнения в момент входа в состояние (список таких воздействий приводится в нижней части прямоугольника-состояния после слов «*enter /*»). В том случае, если такой переход отсутствует, автомат остается в прежнем состоянии. При проектировании графа переходов с помощью средства *UniMod* выполняется автоматическая проверка условий переходов на полноту и непротиворечивость.

Автомат `A1` (рис. 16) отвечает за управления пользовательским интерфейсом. Игра продолжается только тогда, когда автомат находится в состоянии 2. В это состояние вложен автомат `A2`, отвечающий за основной цикл игры.

Если автомат находится в некотором состоянии *S* и отсутствует переход по некоторому событию, то он остается в состоянии *S,* в то время как всем вложенным в это состояние автоматам передается данное событие.

Когда автомат `A1` находится в состоянии 2 и получает событие от таймера (`e101`), переход по этому событию не осуществляется, и событие передается единственному автомату, вложенному в состояние 2 автомата `A1`, – автомату `A2`.

4.6.2. Автомат `A2` – управление главным циклом моделирования

Главный цикл моделирования [16] включает в себя следующие этапы:

- инициализация;
- получение информации от систем управления летающими тарелками;
- реализация физической модели – перемещение летающих тарелок, обработка столкновений и выходов за пределы трассы;

Автомат A2 (рис. 17) содержит состояния и переходы, реализующие эти этапы.

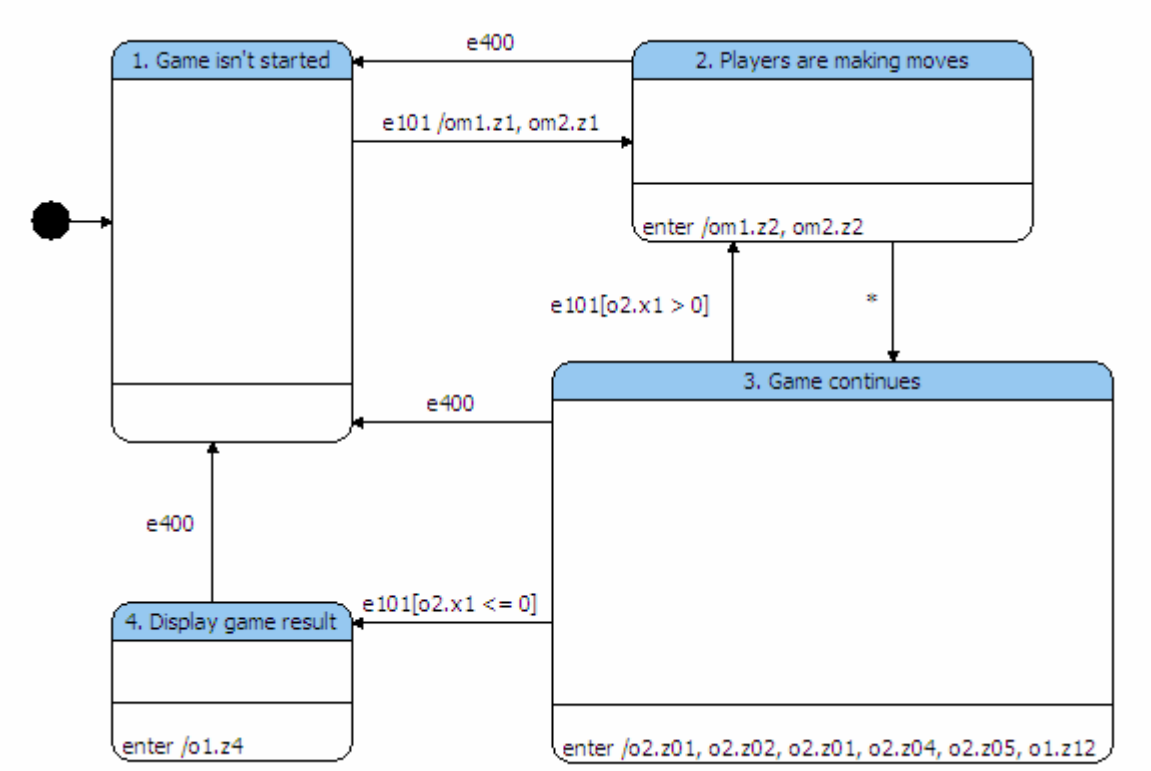

Рис. 17. Граф переходов автомата A2

Первоначально автомат находится в состоянии *«1. Game isn't started»*. При получении события от таймера (e101) выполняется переход из состояния *«1. Game isn't started»* в состояние *«2. Players are making moves»* и реализуются выходные воздействия om1.z1 и om2.z1, с помощью которых каждая из систем управления летающими тарелками производит свою инициализацию. При входе в состояние *«2. Players are making moves»* выполняются выходные воздействия om1.z2 и om2.z2, в которой каждая из команд выполняет свой ход.

В состоянии *«3. Game continues»* происходит моделирование движения тарелок с учетом всех особенностей имеющейся модели среды (разд. 1). При входе в это состояние выполняются все описанные в разд. 1.5 этапы моделирования. Они перечислены в нижней части прямоугольника, соответствующего этому состоянию, после слов «*enter /*».

Затем при получении события от таймера (`e101`) выполняется либо переход в состояние *«3. Players are making moves»* (если игра еще не завершилась – тогда игроки должны сделать очередные ходы), либо переход в состояние *«4. Display game result»*, в котором пользователю сообщается о результатах игры.

Символ `*` на переходе между состояниями 2 и 3 означает то, что это переход выполняется при поступлении любого события.

4.7. Объекты управления

В этом разделе описываются объекты управления.

4.7.1. Объект управления `GameLogic`

Объект управления `GameLogic` содержит выходные воздействия, связанные с реализацией игрового мира, подробно описанного в разд. 1. Табл. 4 содержит краткие описания выходных воздействия этого объекта управления, а табл. 5 – краткие описания входных воздействий.

Таблица 4. Выходные воздействия объекта управления `GameLogic`

	Выходное воздействие	Описание	Комментарий
1.	`z1`	Установка летающих тарелок на стартовые позиции	
2.	`z01`	«Приземлить» тарелки, скорости которых меньше одного м/с	
3.	`z02`	Вычислить новые скорости летающих тарелок	
4.	`z04`	Переместить все летающие тарелки в соответствии с их скоростями	
5.	`z05`	«Приземлить» тарелки, находящиеся вне трассы	

Таблица 5. Входные воздействия объекта управления `GameLogic`

	Входное воздействие	Описание	Комментарий
1.	`x1`	Количество летающих тарелок, еще не завершивших полет	

4.7.2. Объект управления `Screen`

Объект управления `Screen` содержит выходные воздействия, связанные с изменениями внешнего вида главного окна приложения. Выходные воздействия объекта управления представлены в табл. 6.

Таблица 6. Выходные воздействия объекта управления `Screen`

	Выходное воздействие	Описание	Комментарий
1.	`z4`	Вывести результаты игры на экран	
2.	`z10`	Закрыть окно	
3.	`z12`	Перерисовать окно	
4.	`z310`	Сделать кнопку «Start» неактивной	
5.	`z311`	Сделать кнопку «Start» активной	
6.	`z320`	Сделать кнопку «Pause»/ «Continue» неактивной	
7.	`z321`	Сделать кнопку «Pause»/ «Continue» активной	
8.	`z322`	Установить надпись на кнопке «Pause»/ «Continue» в «Pause»	
9.	`z323`	Установить надпись на кнопке «Pause»/ «Continue» в «Continue»	
10.	`z330`	Сделать кнопку «Stop» неактивной	
11.	`z331`	Сделать кнопку «Stop» активной	

4.7.3. Объект управления `ManagerWrapper`

Объект управления `ManagerWrapper` представляет собой оболочку для системы управления летающими тарелками. Он содержит выходные воздействия (табл. 7), связанные с взаимодействием между игровым миром и системой управления группой летающих тарелок. Каждое из этих воздействий, в свою очередь, вызывает определенный метод объекта, реализующего систему управления летающими тарелками. Через этот

объект, в свою очередь, с внешним миром взаимодействуют летающие тарелки.

Таким образом, при изменении системы управления летающими тарелками необходимо изменять не диаграмму связей, а лишь параметры экземпляра объекта `ManagerWrapper` (методы «реальной» системы управления летающими тарелками вызываются при помощи технологии *Reflection* языка *Java*). За счет этого достигается простота встраивания собственных систем управления летающими тарелками в программу.

При этом единственным требованием к указанным системам является требование реализации интерфейса `Manager`.

Таблица 7. Выходные воздействия объекта управления `ManagerWrapper`

	Выходное воздействие	Описание	Комментарий
1.	`z1`	Инициализировать систему управления группой летающих тарелок	
2.	`z2`	Выполнить очередной ход	

Отметим, что на диаграмме связей (рис. 15) присутствуют два экземпляра объекта `ManagerWrapper` (они обозначены `on1` и `on2`). Это объясняется тем, что в игре участвуют две команды (для каждой из них необходим отдельный объект `ManagerWrapper`).

5. Как создать собственную систему управления летающими тарелками?

Ранее упоминалось, что программа состоит из двух частей: ядра и набора систем управления летающими тарелками. Этот раздел представляет собой краткое руководство по созданию собственной системы управления летающими тарелками.

5.1. Обязательные требования к системе управления летающими тарелками

Система управления летающими тарелками должна представлять собой класс, написанный на языке программирования *Java*. Этот класс должен иметь аннотацию `ManagerInfo` и реализовывать интерфейс `Manager`, описанный в разд. 1.7.

5.1.1. Описание аннотации `ManagerInfo`

Аннотация `ManagerInfo` содержит следующие поля:

- **name** – название стратегии;
- **author** – информация об авторах;
- **version** – версия реализации стратегии;
- **comment** – краткое описание.

Эта информация доступна в диалоге выбора соревнующихся команд (их систем управления).

5.1.2. Создание JAR-архива

Для того, чтобы ядро обнаружило систему управления летающими тарелками, последняя должна быть упакована в JAR-архив и помещена в директорию **`managers`**. При создании JAR-архива в манифесте (файл ‘`META-INF/MANIFEST.MF`’) параметр `Main-Class` должен содержать полное имя класса системы управления летающими тарелками.

6. Летающая тарелка (агент)

В этом разделе описаны поставщики событий, автоматы и объекты управления, относящиеся к летающей тарелке (агенту). Отметим, что описываемые объекты находятся в другой *UniMod*-модели, нежели описанные в предыдущих разделах, поэтому совпадение имен вполне допустимо.

6.1. Краткое описание стратегии летающих тарелок

В ходе настоящей работы было сделано предположение, что победной будет стратегия, при которой одна половина агентов движется вперед и уклоняется от границ коридора и рядом находящихся агентов, а вторая половина агентов разбивается на пары. При этом агенты в парах движутся, поддерживая взаимное расположение для использования аэродинамического взаимодействия. Эта стратегия почти всегда оказывается победной по сравнению со стратегией, реализованной для противника: один агент летит вперед, а другие – пытаются сбить агентов другой команды.

В этой версии программы особое внимание уделено тому, чтобы агенты избегали столкновений с другими агентами и со стенами. Взаимодействие же между агентами практически отсутствует.

6.2. Диаграмма связей

В данной версии программы поведение всех агентов «нашей» команды описывается одной и той же системой взаимодействующих автоматов. Диаграмма связей, в которую входят эти автоматы, показана на рис. 18. Автомат *Состояние агента AFP* имеет три состояния: «Летит», «Нормальное завершение», «Аварийное завершение».

В первое состояние этого автомата вложены автомат *Режим полета AL* и автомат *Радар AR*. В состояние «Полет в одиночку» автомата *Режим полета* вложены автомат *Уклонение от границ коридора и агентов справа и слева A1* и автомат *Уклонение от агентов спереди и сзади A3*.

В состояние «Полет первым в паре» автомата *Режим полета AL* вложен автомат *Полет первым в паре AG1*, а в состояние «Полет вторым в паре» автомата *Режим полета AL* вложен автомат *Полет вторым в паре AG2*.

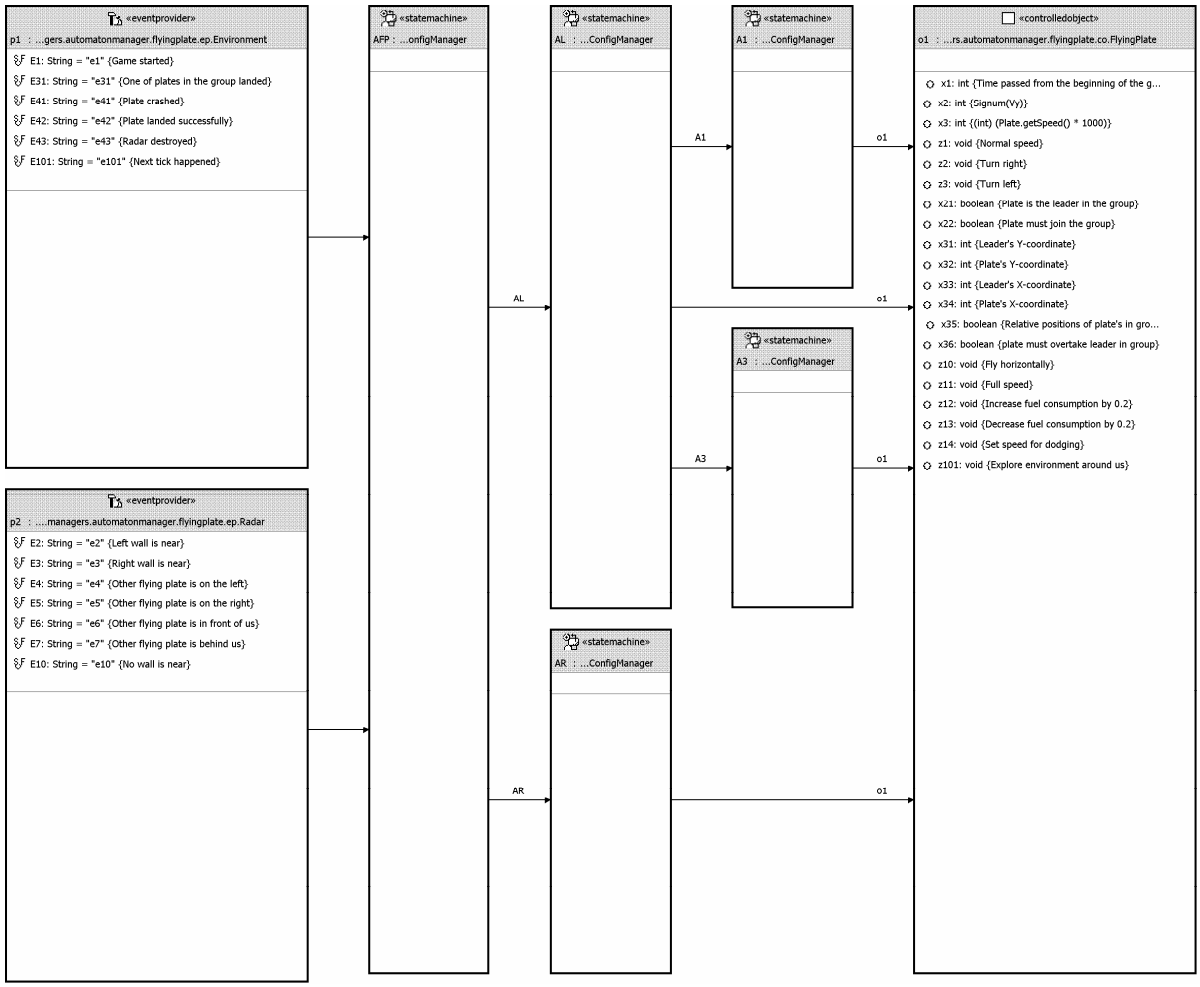

Рис. 18. Диаграмма связей летающей тарелки

6.3. Поставщики событий

В этом разделе описаны поставщики событий.

6.3.1. Поставщик событий `Environment`

Этот поставщик событий служит для передачи воздействий на летающую тарелку от игрового мира. События, генерируемые этим поставщиком событий, перечислены в табл. 8.

Таблица 8. События поставщика событий `Environment`

	Событие	Описание	Параметр	Комментарий
1.	`e1`	Соревнование началось	нет параметров	
2.	`e31`	Второй из агентов в паре разбился или приземлился	нет параметров	
3.	`e41`	Тарелка разбилась	нет параметров	
4.	`e42`	Тарелка благополучно приземлилась	нет параметров	
5.	`e43`	Радар сломался	нет параметров	
6.	`e101`	Поставщик событий `ConfigurableTimer` из ядра программы создал событие `e101`	нет параметров	Таким образом, событие e101 из ядра программы как бы ретранслируется летающей тарелке

6.3.2. Поставщик событий `Radar`

Поставщик событий `Radar` содержит события (табл. 9), связанные с изменением взаимного расположения данной летающей тарелки относительно других летающих тарелок и стен.

Таблица 9. События поставщика событий `Radar`

	Событие	Описание	Параметр	Комментарий
1.	`e2`	Агент находится близко к левой границе трассы	нет параметров	
2.	`e3`	Агент находится близко к правой границе трассы	нет параметров	
3.	`e4`	Другой агент находится слева	нет параметров	
4.	`e5`	Другой агент находится справа	нет параметров	
5.	`e6`	Другой агент находится спереди	нет параметров	
6.	`e7`	Другой агент находится сзади	нет параметров	
7.	`e10`	Рядом нет границ трассы	нет параметров	

6.4. Автоматы

В этом разделе описаны автоматы, реализующие искусственный интеллект летающей тарелки.

6.4.1. Автомат `AFP` – *Состояние агента*

Этот автомат предназначен для управления летающей тарелкой на самом верхнем уровне. Его граф переходов показан на рис. 19.

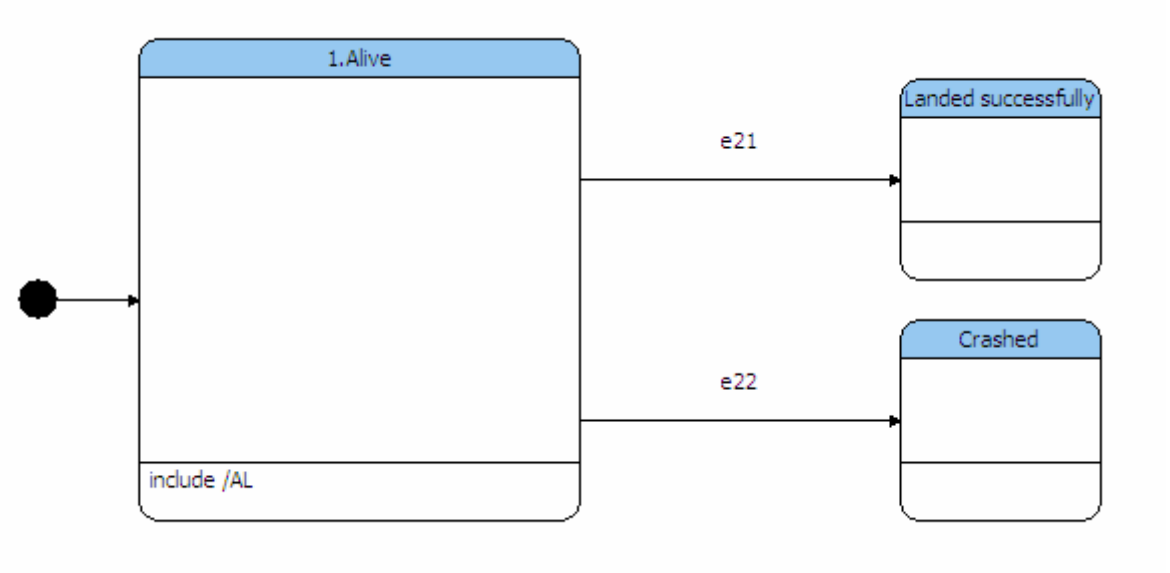

Рис. 19. Граф переходов автомата `AFP`

Автомат `AFP` содержит состояния, соответствующие всем трем возможным этапам жизненного цикла летающей тарелки. Состояние *Alive* соответствует тому, что летающая тарелка летит, – в него вложен автомат `AL`, отвечающий за управление полетом тарелки. Состояние *Landed successfully* соответствует тому, что летающая тарелка благополучно приземлилась, а состояние *Crashed* – тому, что тарелка разбилась.

В начале гонки автомат находится в состоянии *Alive*. При поступлении события `e21` (агент благополучно приземлился) автомат переходит в состояние *Landed successfully*, а при поступлении события `e22` (агент разбился) автомат – в состояние *Crashed*.

6.4.2. Автомат `AL` – *Режим полета*

Автомат `AL` «отвечает» за полет тарелки. Его граф переходов представлен на рис. 20.

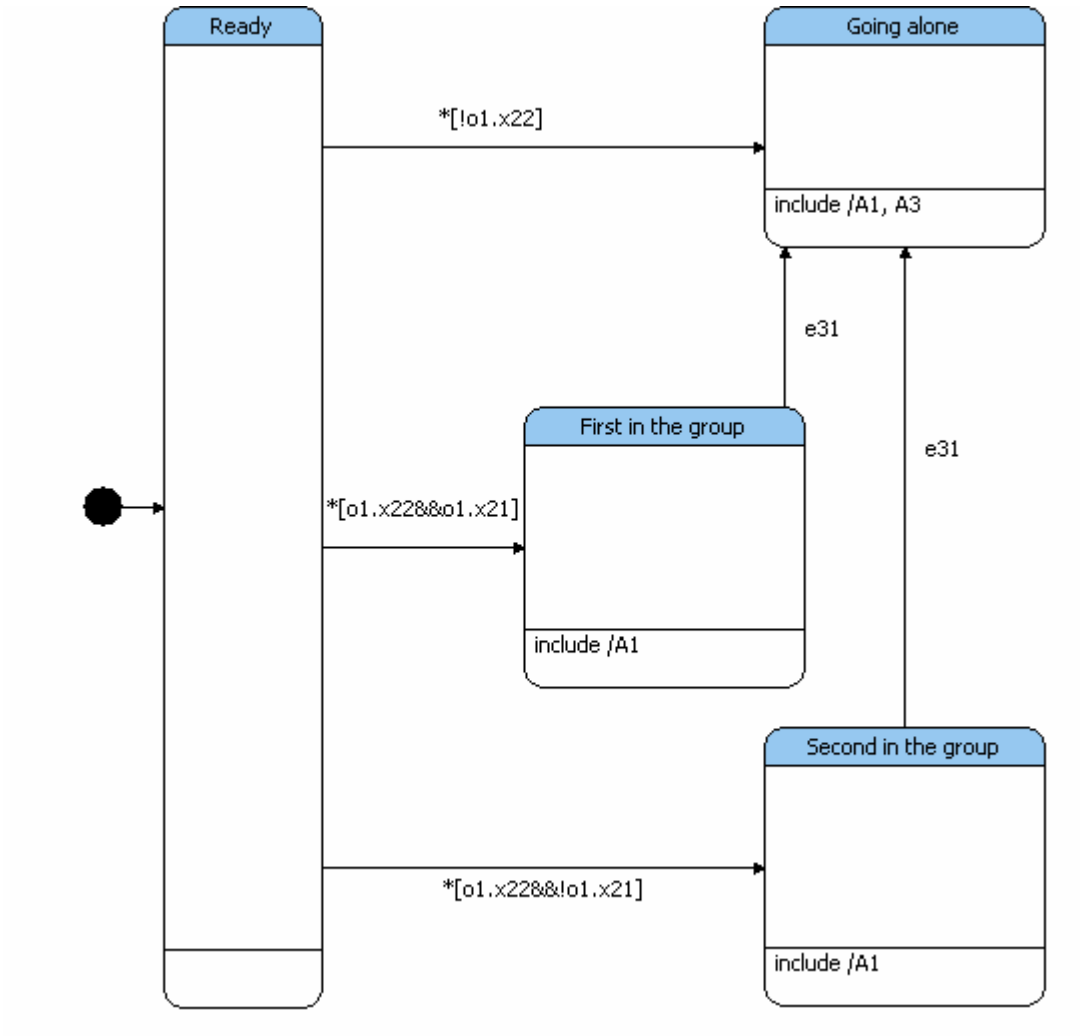

Рис. 20. Граф переходов автомата `AL`

Автомат `AL` имеет 4 состояния. Состояние *Ready* соответствует принятию решения, в каком режиме будет лететь летающая тарелка. Состояние *Ready* реализуется только в самом начале гонки. Оставшиеся три состояния соответствуют этим режимам полета. Состояние *Going alone* соответствует полету в одиночку. В этом режиме управление летающей тарелкой выполняют автоматы `A1` и `A3`, вложенные в это состояние.

Состояние *First in the group* соответствует тому, что агент летит первым в паре, а состояние *Second in the group* – тому, что агент летит вторым в паре. В описываемой версии программы в каждое из этих состояний вложен автомат `A1`.

6.4.3. Автомат `A1` - *Уклонение от границ коридора и агентов справа и слева*

Этот автомат «следит» за тем, чтобы летающая тарелка не сталкивалась со стенами и другими летающими тарелками. Его граф переходов показан на рис. 21.

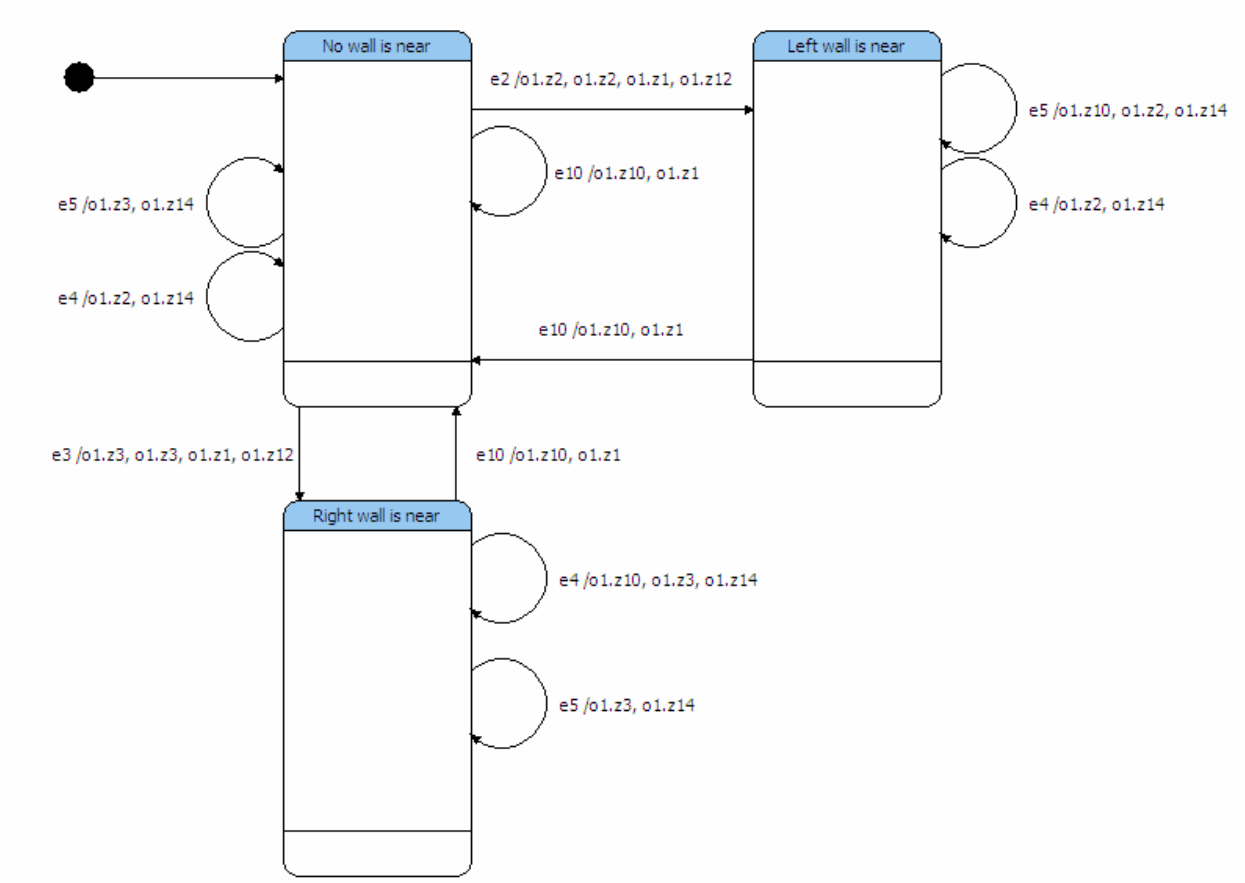

Рис. 21. Граф переходов автомата `A1`

Автомат `A1` имеет три состояния. Состояние *«No wall is near»* соответствует тому, что агент находится далеко от границ коридора, в котором происходят соревнования. Находясь в нем, автомат обрабатывает события, связанные с появлением вблизи агента границ трассы, либо других агентов.

При появлении вблизи агента левой границы трассы (событие `e2`) автомат переходит в состояние *Left wall is near*. При этом переходе агент поворачивает направо, а расход топлива устанавливается равным 0.6.

Аналогично происходит обработка события `e3` (агент находится вблизи правой границы трассы), только вместо поворота направо производится поворот влево.

6.4.4. Автомат `AR` – *Радар*

Автомат `AR`, граф переходов которого показан на рис. 22, управляет радаром летающей тарелки.

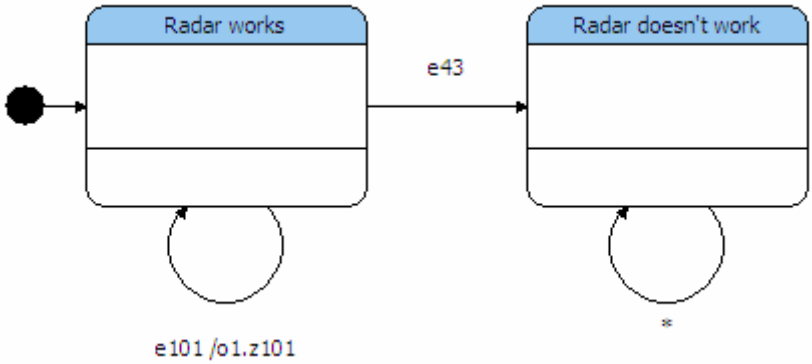

Рис. 22. Граф переходов автомата `AR`

Автомат `AR` имеет два состояния. Состояние *Radar works* соответствует тому, что радар исправен. Находясь в нем, автомат по событию `e101` вызывает метод `o1.z101`, который выполняет «исследование» окружающей среды. По событию `e43` (радар поврежден) автомат переходит в состояние *Radar doesn't work*, в котором он при поступлении любого события не выполняет никаких действий.

6.4.5. Автомат `A3` – *Уклонение от агентов спереди и сзади*

Этот автомат «следит» за агентами, находящимися в непосредственной близости спереди, либо сзади. Его граф переходов показан на рис. 23.

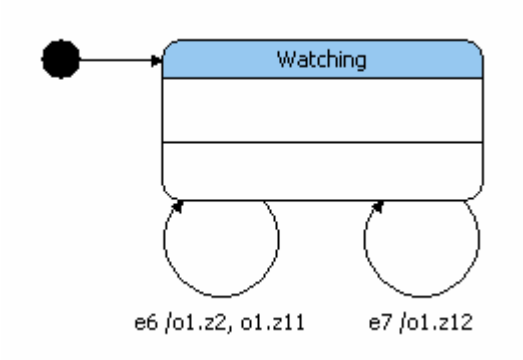

Рис. 23. Граф переходов автомата `A3`

Автомат `A3` имеет одно состояние и два перехода. Один из переходов соответствует обнаружению летающей тарелки спереди – в этом случае управляемая летающая тарелка уменьшает расход топлива (и, как следствие, свою скорость). Обнаружение летающей тарелки сзади обрабатывается аналогично, но в этом случае расход топлива увеличивается.

6.5. Объект управления `FlyingPlate`

Этот объект управления представляет собой летающую тарелку, которой производится управление. Он содержит выходные воздействия, связанные с управлением аэродинамическими рулями летающей тарелки и управлением расходом топлива. Выходные воздействия этого объекта управления перечислены в табл. 10.

Таблица 10. Выходные воздействия объекта управления `FlyingPlate`

	Выходное воздействие	Описание	Комментарий
1.	`z1`	Установить расход топлива равным 0.4	
2.	`z2`	Повернуть направо	
3.	`z3`	Повернуть налево	
4.	`z10`	Лететь горизонтально	
5.	`z11`	Максимальный расход топлива	
6.	`z12`	Увеличить расход топлива на 0.2	
7.	`z13`	Уменьшить расход топлива на 0.2	
8.	`z14`	Установить расход топлива равным 0.8	

Входные воздействия объекта управления `FlyingPlate` перечислены в табл. 11.

Таблица 11. Входные воздействия объекта управления `FlyingPlate`

	Входное воздействие	Описание	Комментарий
1.	`x1`	Время, прошедшее с начала гонки	Измеряется количеством событий e101
2.	`x2`	Знак проекции скорости агента на ось Oy	
3.	`x3`	Величина скорости агента	
4.	`x21`	Агент должен лететь первым в паре	
5.	`x22`	Агент должен лететь в паре с другим агентом	
6.	`x31`	Y-координата первого в паре	
7.	`x32`	Y-координата агента	
8.	`x33`	X-координата первого в паре	
9.	`x34`	X-координата агента	
10.	`x35`	Взаимное расположение агентов в паре правильно	
11.	`x36`	Агент должен обогнать первого в паре	

7. Реализация

Классы на языке *Java* (приложения 1, 2) реализуют функциональность объектов управления и источников событий.

В процессе работы программы ведется протокол, в который заносятся:

- поступающие события;
- состояния, в которых находятся автоматы;

- значения входных переменных, когда делается выбор перехода в новое состояние;
- переходы;
- выходные воздействия.

8. Статистика и тестирование программы

8.1. Статистика исходного кода

Как отмечалось в разд. 2, UML-диаграммы, создаваемые программистом, при использовании компиляционного подхода преобразуются в текст программы на языке *Java*. На рис. 24 показано соотношение объемов исходного кода, написанного вручную (в реализации летающей тарелки) и автоматически сгенерированного по *UniMod*-модели летающей тарелки.

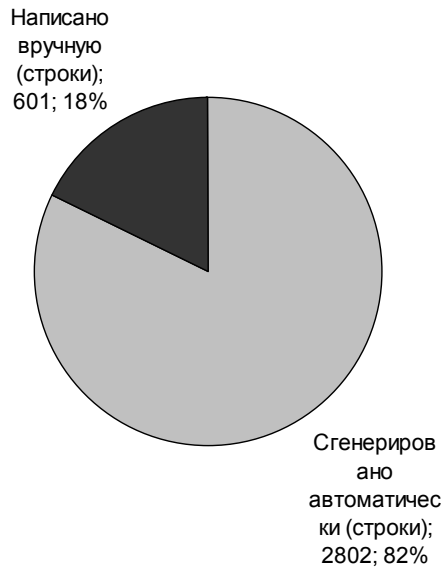

Рис. 24. Соотношение объемов исходного кода, сгенерированного автоматически и написанного вручную

Таким образом, можно сделать вывод о том, что основная функциональность системы управления реализована на основе автоматов. Размер текста программы, сгенерированного автоматически, более чем в четыре раза превосходит размер текста, написанного вручную. Как следствие, существенно повышается надежность программы особенно в

связи с тем, что при построении диаграмм автоматически проверяется их корректность.

8.2. Тестирование мультиагентной системы

Тестирование стратегий проводилось при следующих значениях параметров агентов (разд. 1):

$$c_1 = 0.625; \quad (1)$$

$$c_2 = 0.025; \quad (2)$$

$$c_4 = 3.125; \quad (3)$$

$$\Delta t = 0.3 \text{ секунды}; \quad (4)$$

$$L = 7. \quad (5)$$

Каждая команда состояла из восьми агентов, каждый из которых имел запас топлива в 15 единиц, диаметр агента был равен одному метру, ширина поля – 40 метров. Начальные скорости агентов были равны четырем метрам в секунду, начальные направления – строго вперед. Начальные положения агентов были случайно выбраны на первых 25 метрах поля. Отметим, что в случае необходимости агенты могут быть расположены и детерминировано.

При тестировании в качестве соперников использовались две стратегии. Одна из них (далее называемая «простой») состояла в том, что все агенты двигались строго вперед. Вторая (агрессивная) – в том, что все агенты, кроме одного, двигаются в сторону агентов «нашей» команды с целью вытолкнуть их за пределы трассы или сбить. Оставшийся же агент движется строго вперед.

«Наша» стратегия соревновалась с простой в 30 соревнованиях. Статистика результатов соревнований представлена в табл. 12.

Таблица 12. Результаты соревнований против простой стратегии

	N_{OK}	R_{our}	R_{op}	Δ
Максимум	8	225.6843	187.2180	39.4271
Минимум	5	205.1690	179.2453	20.1622
Среднее	7.3667	214.6608	185.0916	29.5691

В табл. 12 используются следующие обозначения: N_{OK} – количество агентов «нашей» команды, успешно завершивших соревнование; R_{our} – результат «нашей» команды; R_{op} – результат соперника; $\Delta = R_{our} - R_{op}$.

На основе этих экспериментальных результатов можно сделать вывод о том, что «наша» стратегия существенно лучше простой. В одном из соревнований преимущество составило 39 метров, а в среднем оно было около 30 метров. Обычно успешно завершали соревнование более семи агентов, однако существуют такие начальные расположения, при которых два или три агента не могут избежать выхода за пределы трассы или столкновения друг с другом.

Также было проведено тридцать соревнований между агрессивной стратегией и «нашей». Их результаты представлены в табл. 13.

Таблица 13. Результаты соревнований против агрессивной стратегии

	N_{OK}	R_{our}	R_{op}	Δ
Максимум	7	217.7150	195.2719	36.4381
Минимум	2	197.4160	179.0959	7.4978
Среднее	5.2333	208.5202	187.8240	20.6961

Условные обозначения, используемые в табл. 13, такие же, как и в табл. 12.

Более точная информация о результатах «нашей» команды может быть получена из рис. 25, на котором показано распределение результатов «нашей» команды в соревнованиях против агрессивной стратегии.

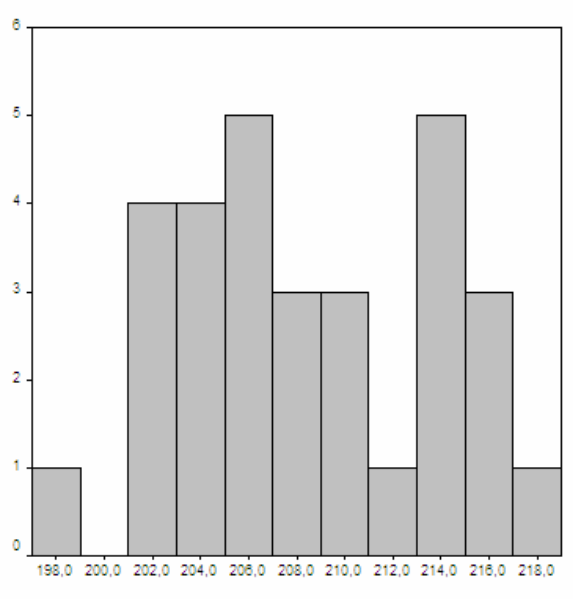

Рис. 25. Распределение результатов «нашей» команды в соревнованиях против агрессивной стратегии

Более детальная информация о результатах «нашей» команды может быть получена из рис. 26, на котором показано распределение количества агентов «нашей» команды, успешно закончивших соревнование против агрессивной стратегии.

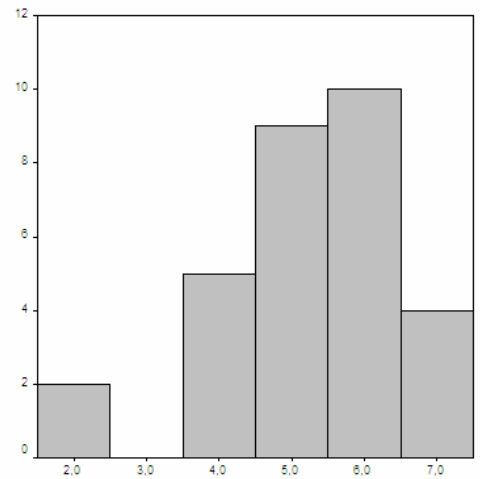

Рис. 26. Распределение количества агентов «нашей» команды, успешно завершивших соревнование против агрессивной стратегии

Из приведенных результатов можно сделать вывод о том, что «наша» стратегия успешно противостоит агрессивной. Среднее преимущество «нашей» стратегии равно 20 метрам, и агенты «нашей» команды успешно избегали столкновений с агентами соперника (в среднем соревнование успешно завершали от пяти до шести агентов).

Заключение

SWITCH-технология динамично развивается. Недавно появился мощный набор инструментов *UniMod*, который позволяет более доступно и наглядно применить автоматный подход для проектирования систем любой степени сложности.

Данный проект построения мультиагентной системы наряду со многими другими, опубликованными на сайте http://is.ifmo.ru, демонстрирует хорошую применимость автоматного подхода и *SWITCH*-технологии для управления «реактивными» системами [2], а также в тех предметных областях, где можно выделить несколько объектов, сложным образом взаимодействующих. Авторы также пришли к выводу, что системы подобные представленной трудно проектировать и реализовывать без использования конечных автоматов, так как при разработке такого рода систем обычным способом приходится слишком много всего держать в голове.

Программу, представленную в данной работе, можно улучшить в нескольких направлениях, например, реализовать взаимодействие между агентами и их построение в группы, а также добавить возможность обучения агентов.

Источники

1. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб. Наука, 1998. http://is.ifmo.ru/books/switch/6
2. *Шалыто А. А., Туккель Н. И.* SWITCH-технология – автоматный подход к созданию программного обеспечения реактивных систем // Программирование. 2001. № 5, с. 45–62. http://is.ifmo.ru/works/switch/1/.
3. *Шалыто А. А.* Новая инициатива в программировании. Движение за открытую проектную документацию // Мир ПК. 2003. № 9, с. 52–56. http://is.ifmo.ru/works/open_doc/
4. UniMod project. http://unimod.sourceforge.net
5. *Гуров В.С., Мазин М.А., Нарвский А.С., Шалыто А.А.* UML. SWITCH-технология. Eclipse. // Информационно-управляющие системы, 2004, № 6, с.12–17. http://is.ifmo.ru/works/uml-switch-eclipse/
6. Eclipse project. http://eclipse.org
7. *Гуров В. С., Нарвский А. С., Шалыто А. А.* ICQ и автоматы //Технология клиент-сервер. 2004, № 3, с. 3–11 http://is.ifmo.ru/works/icq/.
8. Заочный тур всесибирской олимпиады 2005 по информатике. http://olimpic.nsu.ru/widesiberia/archive/wso6/2005/rus/1tour/problem/problem.html
9. *Рассел С., Норвиг П.* Искусственный интеллект. Современный подход. М.: Вильямс. 2006.
10. *Шалыто А. А., Туккель Н. И.* Танки и автоматы //BYTE/Россия. 2003, № 2, с 69–73. http://is.ifmo.ru/projects/tanks/
11. *Шалыто А. А., Кузнецов Д. В.* Система управления танком для игры «Robocode». http://is.ifmo.ru/projects/robocode2/

12. *Ярцев Б.М., Шалыто А.А.* Разработка программного обеспечения роботов Lego Mindstorms на основе автоматного подхода (Проект Isenguard) http://is.ifmo.ru/projects/lego/
13. *Шалыто А.А., Наумов Л.А.* Методы объектно-ориентированной реализации реактивных агентов на основе конечных автоматов // Искусственный интеллект. 2004. №4. с. 756–762. http://is.ifmo.ru/works/_aut_oop.pdf
14. *Yartsev B., Korneev G., Shalyto A., Kotov V.* Automata-Based Programming of the Reactive Multi-Agent Control Systems. *Proceedings of International Conference Integration of Knowledge Intensive Multi-Agent Systems: Modeling, Exploration and Engineering.* KIMAS-05. Boston: IEEE Boston Section. 2005. http://is.ifmo.ru/articles_en/_kimas05-2.pdf
15. *Shalyto A., Naumov L., Korneev G.* Methods of Object-Oriented Reactive Agents Implementation on the Basis of Finite Automata. *Proceedings of International Conference Integration of Knowledge Intensive Multi-Agent Systems: Modeling, Exploration and Engineering.* KIMAS-05. Boston: IEEE Boston Section. 2005. http://is.ifmo.ru/articles_en/_kimas05-1.pdf
16. *Ла Мот А., Ратклифф Д., Семинаторе М., Талер Д.* Секреты программирования игр. СПб.: Питер, 1995.

Приложение 1. Исходные коды ядра

Main.java

```
01 package ru.ifmo.flyingplates;
02
03 import java.io.IOException;
04
05 import javax.xml.parsers.ParserConfigurationException;
06
07 import org.apache.commons.logging.LogFactory;
08 import org.apache.log4j.xml.DOMConfigurator;
09 import org.xml.sax.SAXException;
10
11 import com.evelopers.common.exception.CommonException;
12 import com.evelopers.unimod.adapter.standalone.Run;
13 import com.evelopers.unimod.core.stateworks.Model;
14 import com.evelopers.unimod.debug.ExceptionHandlerImpl;
15 import com.evelopers.unimod.runtime.EventProcessorException;
16 import com.evelopers.unimod.runtime.EventProcessorListener;
17 import com.evelopers.unimod.runtime.ExceptionHandler;
18 import com.evelopers.unimod.runtime.ModelEngine;
19 import com.evelopers.unimod.runtime.interpretation.InterpretationHelper
;
20 import com.evelopers.unimod.runtime.logger.SimpleLogger;
21 import com.evelopers.unimod.transform.TransformException;
22 import com.evelopers.unimod.transform.xml.XMLToModel;
23
24 public class Main {
25     protected void run(String arg) throws IOException, SAXException, Pa
rserConfigurationException, TransformException, InterruptedException, Even
tProcessorException, CommonException {
26         DOMConfigurator.configure(Run.class.getResource("log4j.xml"));
27         Model model = XMLToModel.loadAndCompile(Main.class.getClassLoad
er().getResourceAsStream(arg));
28         InterpretationHelper helper = InterpretationHelper.getInstance(
);
29         ModelEngine engine = helper.createStandAloneModelEngine(model,
true);
30         EventProcessorListener logger = new SimpleLogger(LogFactory.get
Log(Run.class));
31         engine.getEventProcessor().addEventProcessorListener(logger);
32         ExceptionHandler eh = new ExceptionHandlerImpl();
33         engine.getEventProcessor().addExceptionHandler(eh);
34         engine.start();
35     }
36
37     public static void main(String[] args) throws IOException, SAXExcep
tion, ParserConfigurationException, TransformException, InterruptedExcepti
on, EventProcessorException, CommonException {
38         Main m = new Main();
39         m.run("A1.xml");
40     }
41 }
```

Plate.java

```
001 package ru.ifmo.flyingplates.common;
002
003 import java.awt.Color;
004 import java.util.HashSet;
005 import java.util.Set;
006
```

```
007 import ru.ifmo.flyingplates.core.ManagerWrapper;
008 import ru.ifmo.flyingplates.core.exceptions.SecurityError;
009 import ru.ifmo.flyingplates.utils.Vector;
010
011 public class Plate {
012   private static final int FLYING = 0;
013
014   private static final int CRASHED = 1;
015
016   private static final int LANDED = 2;
017
018   private Object owner;
019
020   private int player;
021
022   private int id;
023
024   private int state;
025
026   private Vector position;
027
028   private Vector speed;
029
030   private double fuel;
031
032   private Set<Object> setters;
033
034   private double q;
035
036   private double a;
037
038   private Color[][] colors = new Color[][] {
039     {new Color(0, 0, 255), new Color(0, 0, 127), new Color(127, 127, 0
)},
040     {new Color(255, 0, 0), new Color(127, 0, 0), new Color(0, 127, 127
)}};
041
042   private Color[] borderColors = new Color[] {
043       new Color(0, 0, 255),
044       new Color(255, 0, 0)};
045
046   public Plate(Object owner, int player, int id, Vector position, Vect
or speed, double initFuel) {
047     assert(position != null);
048     assert(speed != null);
049
050     this.owner = owner;
051     this.player = player;
052     this.id = id;
053     this.state = FLYING;
054     this.position = position;
055     this.speed = speed;
056     this.fuel = initFuel;
057
058     this.setters = new HashSet<Object>();
059     this.setters.add(owner);
060
061     this.q = 0;
062     this.a = 0;
063   }
064
065   public Color getColor() {
066     return colors[player][state];
```

```
067   }
068
069   public Color getBorderColor() {
070     return borderColors[player];
071   }
072
073   public int getPlayer() {
074     return player;
075   }
076
077   public int getId() {
078     return id;
079   }
080
081   public Vector getPosition() {
082     return position;
083   }
084
085   public Vector getSpeed() {
086     return speed;
087   }
088
089   public double getFuel() {
090     return fuel;
091   }
092
093   public boolean isFlying() {
094     return state == FLYING;
095   }
096
097   public boolean isCrashed() {
098     return state == CRASHED;
099   }
100
101   public boolean isLanded() {
102     return state == LANDED;
103   }
104
105   public void land() {
106     state = LANDED;
107   }
108
109   public void crash() {
110     state = CRASHED;
111   }
112
113   public void setPosition(Object source, Vector position) {
114     if (owner != source)
115       throw new SecurityError();
116     assert(position != null);
117     this.position = position;
118   }
119
120   public void setSpeed(Object source, Vector speed) {
121     if (owner != source)
122       throw new SecurityError();
123     assert(speed != null);
124     this.speed = speed;
125   }
126
127
128   public void decFuel(Object source, double decFuel) {
129     if (owner != source)
```

```
130       throw new SecurityError();
131     assert(decFuel >= 0);
132     fuel -= decFuel;
133   }
134
135   public double getA() {
136     return a;
137   }
138
139   public double getQ() {
140     return q;
141   }
142
143   public void setA(Object source, double a) {
144     if (!setters.contains(source))
145       throw new SecurityError();
146     int angle = Config.getMaximalRotateAngle();
147     if (a < -angle)
148       a = -angle;
149     if (a > angle)
150       a = angle;
151     this.a = a;
152   }
153
154   public void setQ(Object source, double q) {
155     if (!setters.contains(source))
156       throw new SecurityError();
157     if (q < 0)
158       q = 0;
159     if (q > 1.0)
160       q = 1.0;
161     if (q > fuel)
162       q = fuel;
163     this.q = q;
164   }
165
166   public void addSetter(Object source, Object setter) {
167     if (!(setters.contains(source) || source.getClass() == ManagerWrap
per.class))
168       throw new SecurityError();
169     setters.add(setter);
170   }
171 }
```

ManagerInfo.java

```
01 package ru.ifmo.flyingplates.common;
02
03 import java.lang.annotation.ElementType;
04 import java.lang.annotation.Retention;
05 import java.lang.annotation.RetentionPolicy;
06 import java.lang.annotation.Target;
07
08 /**
09  * Description.
10  *
11  * @author Dmitry Paraschenko
12  * @version 11.01.2006, 14:53:13
13  */
14 @Retention(RetentionPolicy.RUNTIME)
15 @Target({ElementType.TYPE})
16 public @interface ManagerInfo {
17     String name();
18     String author();
```

```
19      String version();
20      String comment();
21  }
```

Plates.java

```
01  package ru.ifmo.flyingplates.common;
02
03  import java.util.ArrayList;
04  import java.util.List;
05
06  import ru.ifmo.flyingplates.core.GameLogic;
07
08  public class Plates {
09
10    public static List<Plate> getPlates() {
11      return GameLogic.getPlates();
12    }
13
14    public static List<Plate> getPlates(int player) {
15      List<Plate> res = new ArrayList<Plate>();
16      for (Plate plate: getPlates()) {
17        if (plate.getPlayer() == player)
18          res.add(plate);
19      }
20      return res;
21    }
22
23    public static Plate getPlate(int player, int id) {
24      for (Plate plate: getPlates()) {
25        if (plate.getPlayer() == player && plate.getId() == id)
26          return plate;
27      }
28      return null;
29    }
30
31    @Deprecated
32    public static Plate getPlate(int plate) {
33      return GameLogic.getPlates().get(plate);
34    }
35  }
```

Manager.java

```
01  package ru.ifmo.flyingplates.common;
02
03  import javax.swing.JDialog;
04
05  public interface Manager {
06      public JDialog configManager(JDialog mainDialog);
07
08      public void init(ClassLoader cl, int player);
09
10    public void doTurn();
11  }
```

Config.java

```
001 package ru.ifmo.flyingplates.common;
002
003 import java.io.IOException;
004 import java.io.InputStream;
005 import java.util.Properties;
006
007 public class Config {
008   private static final String FILE_NAME = "config.properties";
```

```
009
010   private static final Properties properties;
011
012   static {
013     InputStream is = null;
014     Properties p = new Properties();
015     is = Config.class.getClassLoader().getResourceAsStream(FILE_NAME);
016     if (is != null) {
017       try {
018         p.load(is);
019       } catch (IOException e) {
020         e.printStackTrace();
021       } finally {
022         try {
023           is.close();
024         } catch (IOException e) {
025           e.printStackTrace();
026         }
027       }
028     }
029     properties = p;
030   }
031
032   public static final String PLATES_COUNT = "plates.count";
033
034   public static final String INIT_FUEL = "init.fuel";
035
036   public static final String INIT_SPEED = "init.speed";
037
038   public static final String PIXELS_IN_METER = "meter2pixels";
039
040   public static final String FIELD_HEIGHT = "field.height";
041
042   public static final String FIELD_WIDTH = "field.width";
043
044   public static final String PLATE_DIAMETER = "plate.diameter";
045
046   public static final String CONSTANT_F1 = "cf1";
047   public static final String CONSTANT_F2 = "cf2";
048   public static final String CONSTANT_T = "ct";
049
050   public static final String TIME_STEP = "time.step";
051   public static final String TIMER_DELAY = "timer.delay";
052
053   public static final String MAXIMAL_ROTATE_ANGLE = "maximal.rotate.an
gle";
054
055   public static final String INFLUENCE_DISTANCE = "influence.distance"
;
056
057   public static String SCREEN_INFLUENCE_AREA = "screen.influence.area"
;
058   public static String SCREEN_TRACE = "screen.trace";
059   public static String SCREEN_CIRCLE = "screen.circle";
060
061   public static final String MANAGERS_MANAGER1 = "managers.manager1";
062   public static final String MANAGERS_MANAGER2 = "managers.manager2";
063
064
065   public static int getPlatesCount() {
066     return Integer.parseInt(properties.getProperty(PLATES_COUNT, "8"))
;
067   }
```

```
068
069   public static double getInitFuel() {
070     return Double.parseDouble(properties.getProperty(INIT_FUEL, "15.0"
));
071   }
072
073   public static double getInitSpeed() {
074     return Double.parseDouble(properties.getProperty(INIT_SPEED, "4.0"
));
075   }
076
077   public static int getPixelsInMeter() {
078     return Integer.parseInt(properties.getProperty(PIXELS_IN_METER, "1
4"));
079   }
080
081   public static int getFieldHeight() {
082     return Integer.parseInt(properties.getProperty(FIELD_HEIGHT, "40")
);
083   }
084
085   public static int getFieldWidth() {
086     return Integer.parseInt(properties.getProperty(FIELD_WIDTH, "1000"
));
087   }
088
089   public static int getPlateDiameter() {
090     return Integer.parseInt(properties.getProperty(PLATE_DIAMETER, "1"
));
091   }
092
093     public static double getConstantF1() {
094     return Double.parseDouble(properties.getProperty(CONSTANT_F1, "0.6
25"));
095   }
096
097   public static double getConstantF2() {
098     return Double.parseDouble(properties.getProperty(CONSTANT_F2, "0.0
25"));
099   }
100
101   public static double getConstantT() {
102     return Double.parseDouble(properties.getProperty(CONSTANT_T, "3.12
5"));
103   }
104
105   public static double getTimeStep() {
106     return Double.parseDouble(properties.getProperty(TIME_STEP, "0.3")
);
107   }
108
109   public static double getTimerDelay() {
110     return Double.parseDouble(properties.getProperty(TIMER_DELAY, "0.3
"));
111   }
112
113   public static int getMaximalRotateAngle() {
114     return Integer.parseInt(properties.getProperty(MAXIMAL_ROTATE_ANGL
E, "25"));
115   }
116
117   public static int getInfluenceDistance() {
118     return Integer.parseInt(properties.getProperty(INFLUENCE_DISTANCE,
```

```
 "7"));
119   }
120
121   public static boolean isScreenInfluenceAreaEnabled() {
122     return Boolean.parseBoolean(properties.getProperty(SCREEN_INFLUENC
E_AREA, "true"));
123   }
124
125   public static boolean isScreenTraceEnabled() {
126     return Boolean.parseBoolean(properties.getProperty(SCREEN_TRACE, "
true"));
127   }
128
129   public static boolean isScreenCircleEnabled() {
130     return Boolean.parseBoolean(properties.getProperty(SCREEN_CIRCLE,
"false"));
131   }
132 }
```

GameLogic.java

```
001 package ru.ifmo.flyingplates.core;
002
003 import java.util.ArrayList;
004 import java.util.Collections;
005 import java.util.HashMap;
006 import java.util.List;
007 import java.util.Random;
008
009 import ru.ifmo.flyingplates.common.Config;
010 import ru.ifmo.flyingplates.common.Plate;
011 import ru.ifmo.flyingplates.utils.Vector;
012
013 import com.evelopers.unimod.runtime.ControlledObject;
014 import com.evelopers.unimod.runtime.context.StateMachineContext;
015
016 public class GameLogic implements ControlledObject {
017   private static final List<Plate> plates = new ArrayList<Plate>();
018   private static final List<Plate> unmodifiablePlates = Collections.un
modifiableList(plates);
019
020   private static final double inf = 1e20;
021   private static final double eps = 1e-9;
022
023   public static List<Plate> getPlates() {
024     return unmodifiablePlates;
025   }
026
027   /**
028    * @unimod.action.descr Count of flying plates
029    */
030   public int x1(StateMachineContext context) {
031     int res = 0;
032     for (Plate plate: plates)
033       if (plate.isFlying())
034         res++;
035     return res;
036   }
037
038   /**
039    * @unimod.action.descr Regenerate plates array
040    */
041   public void z1(StateMachineContext context) {
042     Random random = new Random();
```

```
043     double[] randomXCoordinates = new double[Config.getPlatesCount()];
044     for (int i = 0; i < Config.getPlatesCount(); i++) {
045       randomXCoordinates[i] = 5 + random.nextDouble() * 15;
046     }
047
048     plates.clear();
049
050     for (int i = 0; i < Config.getPlatesCount(); i++) {
051       plates.add(new Plate(this, 0, i,
052           new Vector(randomXCoordinates[i],
053               Config.getFieldHeight() * i / (2 * Config.getPlatesCount
() - 1)),
054           new Vector(Config.getInitSpeed(), 0),
055           Config.getInitFuel()));
056     }
057     for (int i = 0; i < Config.getPlatesCount(); i++) {
058       plates.add(new Plate(this, 1, i,
059           new Vector(randomXCoordinates[Config.getPlatesCount() - i -
 1],
060               Config.getFieldHeight() * (Config.getPlatesCount() + i)
061               / (2 * Config.getPlatesCount() - 1)),
062           new Vector(Config.getInitSpeed(), 0),
063           Config.getInitFuel()));
064     }
065   }
066
067   /**
068    * @unimod.action.descr Land and crash plates which speed is less th
an 1 m/s
069    */
070   public void z01(StateMachineContext context) {
071     for (Plate plate: plates) {
072       if (plate.isFlying() && plate.getSpeed().getLength() < 1) {
073         if (plate.getFuel() <= 0.001) {
074           plate.land();
075         } else {
076           plate.crash();
077         }
078       }
079     }
080   }
081
082   /**
083    * @unimod.action.descr Calculate new plates speedes
084    */
085   public void z02(StateMachineContext context) {
086     HashMap<Plate, Double> koeff = new HashMap<Plate, Double>();
087     for (Plate plate1: plates) {
088       if (!plate1.isFlying())
089         continue;
090       double koef = 1;
091       for (Plate plate2: plates) {
092         if (!plate2.isFlying())
093           continue;
094         if (plate1.getPlayer() == plate2.getPlayer() && plate1.getId()
 == plate2.getId())
095           continue;
096         Vector speed = plate2.getSpeed();
097         Vector delta = plate2.getPosition().subtract(plate1.getPositio
n());
098         if (delta.getLength() > Config.getInfluenceDistance())
099           continue;
100         double ca = speed.multiply(delta) / (speed.getLength() * delta
```

```
.getLength());
101         if (ca >= Math.cos(20 * Math.PI / 180)) {
102           koef += 0.5;
103         } else if (ca >= Math.cos(40 * Math.PI / 180)) {
104           koef -= 0.5;
105         }
106       }
107       if (koef < 0)
108         koef = 0;
109       koeff.put(plate1, koef);
110     }
111     for (Plate plate: plates) {
112       if (!plate.isFlying())
113         continue;
114       Vector speed = plate.getSpeed().rotate(plate.getA() * Math.PI /
180);
115       double T = 0;
116       plate.decFuel(this, plate.getQ() * Config.getTimeStep());
117       T = Config.getConstantT() * plate.getQ();
118       double F = Config.getConstantF1() + Config.getConstantF2() *
          speed.getLength() * speed.getLength();
119       double inc = Config.getTimeStep() * (T -
 F * koeff.get(plate)) / speed.getLength();
120       plate.setSpeed(this, speed.multiply(1 + inc));
121     }
122   }
123
124   /**
125    * @unimod.action.descr Move plates
126    */
127   public void z04(StateMachineContext context) {
128     ArrayList<Plate> cr = new ArrayList<Plate>();
129     double t = Config.getTimeStep();
130     while (t > 0) {
131       double min = inf;
132       boolean found = false;
133       for (Plate plate1: plates) {
134         if (!plate1.isFlying())
135           continue;
136         for (Plate plate2: plates) {
137           if (!plate2.isFlying())
138             continue;
139           if (!(plate1.getPlayer() < plate2.getPlayer() || plate1.
getPlayer() == plate2.getPlayer() && plate1.getId() < plate2.getId()))
140             continue;
141
142           Vector v1 = plate1.getSpeed();
143           Vector pos1 = plate1.getPosition();
144           Vector v2 = plate2.getSpeed();
145           Vector pos2 = plate2.getPosition();
146
147           Vector deltaV = v1.subtract(v2);
148           Vector deltaPos = pos1.subtract(pos2);
149
150           double a = deltaV.getLength() * deltaV.getLength();
151           double b = 2 * deltaV.multiply(deltaPos);
152           double c = deltaPos.getLength() * deltaPos.getLength() -
 Config.getPlateDiameter() * Config.getPlateDiameter();
153           double d = b * b - 4 * a * c;
154
155           if (d < 0)
156             continue;
157           if (Math.abs(a) <= eps)
```

```
158             continue;
159
160           double t1 = (-b + Math.sqrt(d)) / (2 * a);
161           double t2 = (-b - Math.sqrt(d)) / (2 * a);
162           if (t1 < eps)
163             t1 = inf;
164           if (t2 < eps)
165             t2 = inf;
166           t1 = Math.min(t1, t2);
167           if (t1 > t)
168             continue;
169           if (t1 < min) {
170             min = t1;
171             found = true;
172           }
173         }
174       }
175       if (!found)
176         break;
177       cr.clear();
178       t -= min;
179       for (Plate plate: plates) {
180         if (!plate.isFlying())
181           continue;
182         plate.setPosition(this, plate.getPosition().add(plate.
getSpeed().multiply(min)));
183       }
184
185       for (Plate plate1: plates) {
186         if (!plate1.isFlying())
187           continue;
188         for (Plate plate2: plates) {
189           if (!plate2.isFlying())
190             continue;
191           if (!(plate1.getPlayer() < plate2.getPlayer() || plate1.
getPlayer() == plate2.getPlayer() && plate1.getId() < plate2.getId()))
192             continue;
193
194           if (plate1.getPosition().subtract(plate2.getPosition()).
getLength() - Config.getPlateDiameter() < 1e-7) {
195
196             Vector v1 = plate1.getSpeed();
197             Vector pos1 = plate1.getPosition();
198             Vector v2 = plate2.getSpeed();
199             Vector pos2 = plate2.getPosition();
200
201             Vector axis = pos1.subtract(pos2);
202               double d = axis.getLength();
203               axis = axis.multiply(d);
204               double v1a = v1.multiply(axis);
205               double v2a = v2.multiply(axis);
206               double vRel = Math.abs(v1a - v2a);
207               if (vRel > 1) {
208                 cr.add(plate1);
209                 cr.add(plate2);
210                 continue;
211               } else {
212                 System.err.println("Hit happened");
213                 double v2an = v1a;
214                 double v1an = v2a;
215                 v1 = v1.subtract(axis.multiply(v1a)).add(axis.
multiply(v1an));
216                 v2 = v2.subtract(axis.multiply(v2a)).add(axis.
```

```
multiply(v2an));
                plate1.setSpeed(this, v1);
                plate2.setSpeed(this, v2);
              }
            }
          }
        }
        for (Plate plate: cr)
          plate.crash();
      }
      for (Plate plate: plates) {
        if (!plate.isFlying())
          continue;
        plate.setPosition(this, plate.getPosition().add(plate.getSpeed()
.multiply(t)));
      }
    }

    /**
     * @unimod.action.descr Land and crash plates which are outside the
track
     */
    public void z05(StateMachineContext context) {
      for (Plate plate: plates) {
        if (!plate.isFlying())
          continue;
        Vector pos = plate.getPosition();
        if (pos.x <= 0) {
          plate.crash();
          plate.setPosition(this, new Vector(0, pos.y));
        } else if (pos.y <= 0) {
          plate.crash();
          plate.setPosition(this, new Vector(pos.x, 0));
        } else if (pos.y >= Config.getFieldHeight()) {
          plate.crash();
          plate.setPosition(this, new Vector(pos.x, Config.
getFieldHeight()));
        } else
          continue;
        plate.setSpeed(this, new Vector(0, 0));
      }
    }
}
```

ManagerWrapper.java

```
package ru.ifmo.flyingplates.core;

import java.util.List;

import ru.ifmo.flyingplates.common.Manager;
import ru.ifmo.flyingplates.common.Plate;
import ru.ifmo.flyingplates.core.settings.ManagerContainer;
import ru.ifmo.flyingplates.core.settings.ManagersSettings;

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

public final class ManagerWrapper implements ControlledObject {
  private static int playerCnt = 0;

  private final int player;
  private Manager manager;

```

```
19   public ManagerWrapper() {
20     player = playerCnt++;
21     assert(player == 0 || player == 1);
22   }
23
24   /**
25    * @unimod.action.descr Init Manager
26    */
27   public void z1(StateMachineContext context) {
28     Manager man = null;
29     ClassLoader cl = ManagerWrapper.class.getClassLoader();
30     List<ManagerContainer> mcs = ManagersSettings.getManagersSettings()
.getManagerContainers();
31     for (ManagerContainer mc: mcs) {
32       if (mc.getManager(player) != null) {
33         man = mc.getManager(player);
34         cl = mc.getClassLoader(player);
35         break;
36       }
37     }
38     manager = man;
39
40     for (Plate plate: GameLogic.getPlates()) {
41       if (plate.getPlayer() == player) {
42         plate.addSetter(this, manager);
43       }
44     }
45     manager.init(cl, player);
46   }
47
48   /**
49    * @unimod.action.descr Do turn for Manager
50    */
51   public void z2(StateMachineContext context) {
52     manager.doTurn();
53   }
54 }
```

ConfigurableTimer.java

```
01 package ru.ifmo.flyingplates.core;
02
03 import java.util.TimerTask;
04
05 import ru.ifmo.flyingplates.common.Config;
06
07 import com.evelopers.unimod.core.stateworks.Event;
08 import com.evelopers.unimod.runtime.EventProvider;
09 import com.evelopers.unimod.runtime.ModelEngine;
10 import com.evelopers.unimod.runtime.context.StateMachineContextImpl;
11
12 public class ConfigurableTimer implements EventProvider{
13   /**
14    * @unimod.event.descr Next tick happened
15    */
16   public static final String E101 = "e101";
17
18   private java.util.Timer t;
19   private ModelEngine engine;
20
21   public void init(ModelEngine engine) {
22     this.engine = engine;
23
24     t = new java.util.Timer(false);
```

```
25      t.schedule(new TimerTask() {
26            public void run() {
27              ConfigurableTimer.this.engine.getEventManager().handle(
28                  new Event(E101), StateMachineContextImpl.create());
29            }
30          }, 100, (int) (1000 * Config.getTimerDelay()));
31    }
32
33    public void dispose() {
34      t.cancel();
35    }
36  }
```

InterfaceNotImplementedException.java

```
1  package ru.ifmo.flyingplates.core.exceptions;
2
3  public class InterfaceNotImplementedException extends Exception {
4    private static final long serialVersionUID = 1L;
5  }
```

SecurityError.java

```
01 package ru.ifmo.flyingplates.core.exceptions;
02
03 import javax.swing.JOptionPane;
04
05 public class SecurityError extends Error {
06   private static final long serialVersionUID = 1L;
07
08   public SecurityError() {
09     if (JOptionPane.showConfirmDialog(null,
10         "Security error happened!\n" +
11         "Would you like to terminate application?"
12         , "Security error",
13         JOptionPane.YES_NO_OPTION, JOptionPane.ERROR_MESSAGE) ==
JOptionPane.YES_OPTION) {
14       this.printStackTrace();
15       System.exit(0);
16     }
17   }
18 }
```

AnnotationNotFoundException.java

```
1  package ru.ifmo.flyingplates.core.exceptions;
2
3  public class AnnotationNotFoundException extends Exception {
4    private static final long serialVersionUID = 1L;
5  }
```

ScreenEventProvider.java

```
01 package ru.ifmo.flyingplates.core.gui;
02
03 import com.evelopers.common.exception.CommonException;
04 import com.evelopers.unimod.core.stateworks.Event;
05 import com.evelopers.unimod.runtime.EventProvider;
06 import com.evelopers.unimod.runtime.ModelEngine;
07 import com.evelopers.unimod.runtime.context.Parameter;
08 import com.evelopers.unimod.runtime.context.StateMachineContextImpl;
09
10 public class ScreenEventProvider implements EventProvider {
11   /**
12    * @unimod.event.descr StartGameButton pressed
13    */
14   public static final String E1 = "e1";
```

```
15
16    /**
17     * @unimod.event.descr PauseContinueGameButton pressed
18     */
19    public static final String E2 = "e2";
20
21    /**
22     * @unimod.event.descr StopGameButton pressed
23     */
24    public static final String E3 = "e3";
25
26    /**
27     * @unimod.event.descr ExitGameButtonPressed
28     */
29    public static final String E4 = "e4";
30
31    /**
32     * @unimod.event.descr Restart GameLogic automata (A2)
33     */
34    public static final String E400 = "e400";
35
36    private static ModelEngine engine;
37
38    public void init(ModelEngine engine) throws CommonException {
39      if (ScreenEventProvider.engine != null)
40        return;
41  //    try {
42  //      PrintStream out = new PrintStream(new File("plates.log"));
43  //      System.setOut(out);
44  //    } catch (FileNotFoundException e) {
45  //      e.printStackTrace();
46  //    }
47      ScreenEventProvider.engine = engine;
48    }
49
50      public static void fireEvent(String eventName, Parameter parameter)
 {
51          if (engine != null) {
52              if (parameter == null) {
53                  engine.getEventManager().handle(new Event(eventName),
54                          StateMachineContextImpl.create());
55              } else {
56                  engine.getEventManager().handle(
57                          new Event(eventName, new Parameter[] { paramete
r }),
58                          StateMachineContextImpl.create());
59              }
60          }
61      }
62
63    public void dispose() {
64    }
65  }
```

GameResultDialog.java

```
01  package ru.ifmo.flyingplates.core.gui;
02
03  import java.awt.Dimension;
04  import java.awt.GridBagConstraints;
05  import java.awt.GridBagLayout;
06  import java.awt.Insets;
07  import java.awt.event.ActionEvent;
08  import java.awt.event.ActionListener;
```

```
09
10 import javax.swing.JButton;
11 import javax.swing.JDialog;
12 import javax.swing.JFrame;
13 import javax.swing.JLabel;
14 import javax.swing.JPanel;
15 import javax.swing.JScrollPane;
16
17 public class GameResultDialog extends JDialog implements ActionListener
 {
18   private static final long serialVersionUID = 1L;
19
20   private JButton btnOK;
21
22   public GameResultDialog(JFrame owner, double dist1, double dist2) {
23     super(owner, "The game is over", true);
24     this.setResizable(false);
25     JPanel panel = new JPanel();
26     this.getContentPane().add(new JScrollPane(panel));
27     GridBagLayout gbl = new GridBagLayout();
28     panel.setLayout(gbl);
29     gbl.columnWeights = new double[] { 1 };
30     gbl.rowWeights = new double[] { 1, 1, 1, 1 };
31     panel.add(new JLabel("Player1 distance: " + String.format("%.4f",
dist1)),
32         new GridBagConstraints(0, 0, 2, 1, 0, 0,
33             GridBagConstraints.WEST, GridBagConstraints.HORIZONTAL,
34             new Insets(0, 10, 0, 10), 0, 0));
35     panel.add(new JLabel("Player2 distance: " + String.format("%.4f",
dist2)),
36         new GridBagConstraints(0, 1, 2, 1, 0, 0,
37             GridBagConstraints.WEST, GridBagConstraints.HORIZONTAL,
38             new Insets(0, 10, 0, 10), 0, 0));
39     String result = "";
40     if (Math.abs(dist1 - dist2) < 1e-3) {
41       result = "Game result: 'draw'.";
42     } else if (dist1 > dist2) {
43       result = "Game result: 'first player won'.";
44     } else /* if (dist2 > dist1) */{
45       result = "Game result: 'second player won'.";
46     }
47     panel.add(new JLabel(result), new GridBagConstraints(0, 2, 2, 1, 0,
 0,
48         GridBagConstraints.WEST, GridBagConstraints.HORIZONTAL,
49         new Insets(0, 10, 0, 10), 0, 0));
50     btnOK = new JButton("OK");
51     btnOK.setMinimumSize(new Dimension(80, 25));
52     btnOK.setPreferredSize(new Dimension(80, 25));
53     btnOK.addActionListener(this);
54     panel.add(btnOK, new GridBagConstraints(0, 3, 2, 1, 0, 0,
55         GridBagConstraints.CENTER, GridBagConstraints.NONE, new Insets(
56             0, 10, 0, 10), 0, 0));
57     this.getRootPane().setDefaultButton(btnOK);
58     this.setSize(300, 150);
59     this.setLocation(owner.getX() + (owner.getWidth() -
 this.getWidth())
60         / 2, owner.getY() + (owner.getHeight() -
 this.getHeight()) / 2);
61     this.setVisible(true);
62   }
63
64   public void actionPerformed(ActionEvent e) {
65     if (e.getSource() == btnOK)
```

```
66        this.setVisible(false);
67    }
68 }
```

ManagersDialog.java

```
001 package ru.ifmo.flyingplates.core.gui;
002
003 import javax.swing.*;
004 import javax.swing.table.AbstractTableModel;
005
006 import ru.ifmo.flyingplates.core.settings.ManagerContainer;
007 import ru.ifmo.flyingplates.core.settings.ManagersSettings;
008
009 import java.awt.*;
010 import java.awt.event.ActionListener;
011 import java.awt.event.ActionEvent;
012 import java.util.List;
013
014 /**
015  * This class extends <code>JDialog</code> in order to allow user to
016  * configure available managers.
017  *
018  * @author Dmitry Paraschenko
019  * @version 09.09.2004, 14:08:16
020  */
021 public class ManagersDialog extends JDialog implements ActionListener
{
022   private static final long serialVersionUID = 1L;
023
024   private ManagersSettings managers;
025
026     private Runnable updateCommentLabelRunnable;
027     private JDialog configManagerDialog1;
028     private JDialog configManagerDialog2;
029
030     private JButton btnOK;
031     private JButton btnConfig1;
032     private JButton btnConfig2;
033     private JLabel commentLabel;
034     private ListSelectionModel managersListSelectionModel;
035
036     /**
037      * Creates a modal dialog allowing to configure managers.
038      *
039      * @param owner the <code>Frame</code> from which the dialog is di
splayed
040      * @param managers available managers
041      */
042     public ManagersDialog(JFrame owner, ManagersSettings managers) {
043         super(owner, "Managers", true);
044         this.managers = managers;
045
046         this.setSize(new Dimension(400, 450));
047         Dimension screenSize = Toolkit.getDefaultToolkit().
getScreenSize();
048         setLocation((screenSize.width -
 getWidth()) / 2, (screenSize.height - getHeight()) / 2 - 30);
049
050         updateCommentLabelRunnable = new UpdateCommentLabelRunnable();
051
052         JTable managersTable = new JTable();
053         managersTable.setModel(new ManagersTableModel());
054         managersListSelectionModel = new ManagersListSelectionModel();
```

```
055         managersTable.setSelectionModel(managersListSelectionModel);
056         managersTable.setDragEnabled(false);
057         managersTable.setRowHeight(16);
058         managersTable.setSelectionMode(ListSelectionModel.
SINGLE_SELECTION);
059         managersTable.getColumnModel().getColumn(0).setMinWidth(16);
060         managersTable.getColumnModel().getColumn(0).setMaxWidth(16);
061         managersTable.getColumnModel().getColumn(0).
setPreferredWidth(16);
062         managersTable.getColumnModel().getColumn(1).setMinWidth(50);
063         managersTable.getColumnModel().getColumn(1).
setPreferredWidth(150);
064         managersTable.getColumnModel().getColumn(2).setMinWidth(50);
065         managersTable.getColumnModel().getColumn(2).
setPreferredWidth(150);
066         managersTable.getColumnModel().getColumn(3).setMinWidth(50);
067         managersTable.getColumnModel().getColumn(3).
setPreferredWidth(50);
068         managersTable.getColumnModel().getColumn(4).setMinWidth(16);
069         managersTable.getColumnModel().getColumn(4).setMaxWidth(16);
070         managersTable.getColumnModel().getColumn(4).
setPreferredWidth(16);
071
072         commentLabel = new JLabel();
073
074         btnConfig1 = new JButton("Config manager B");
075         btnConfig1.setMinimumSize(new Dimension(140, 25));
076         btnConfig1.setPreferredSize(new Dimension(140, 25));
077         btnConfig1.addActionListener(this);
078         btnConfig1.setEnabled(false);
079
080         btnConfig2 = new JButton("Config manager R");
081         btnConfig2.setMinimumSize(new Dimension(140, 25));
082         btnConfig2.setPreferredSize(new Dimension(140, 25));
083         btnConfig2.addActionListener(this);
084         btnConfig2.setEnabled(false);
085
086         btnOK = new JButton("OK");
087         btnOK.setMinimumSize(new Dimension(80, 25));
088         btnOK.setPreferredSize(new Dimension(80, 25));
089         btnOK.addActionListener(this);
090
091         GridBagLayout gbl = new GridBagLayout();
092         gbl.columnWeights = new double[] {1, 1, 1};
093         gbl.rowWeights = new double[] {3, 1, 0};
094
095         JPanel commentPanel = new JPanel();
096         commentPanel.setLayout(new GridBagLayout());
097         commentPanel.add(commentLabel, new GridBagConstraints(0, 0, 1,
 1, 0, 0, GridBagConstraints.CENTER, GridBagConstraints.BOTH,
new Insets(5, 5, 5, 5), 0, 0));
098
099         Container contentPane = this.getContentPane();
100         contentPane.setLayout(gbl);
101         contentPane.add(new JScrollPane(managersTable), new
GridBagConstraints(0, 0, 3, 1, 0, 0, GridBagConstraints.CENTER,
GridBagConstraints.BOTH, new Insets(5, 5, 2, 5), 0, 0));
102         contentPane.add(new JScrollPane(commentPanel), new
GridBagConstraints(0, 1, 3, 1, 0, 0, GridBagConstraints.CENTER,
GridBagConstraints.BOTH, new Insets(2, 5, 2, 5), 0, 0));
103         contentPane.add(btnConfig1, new GridBagConstraints(0, 2, 1, 1,
 0, 0, GridBagConstraints.WEST, GridBagConstraints.NONE,
new Insets(5, 5, 5, 5), 0, 0));
```

```
104         contentPane.add(btnOK, new GridBagConstraints(1, 2, 1, 1, 0, 0
, GridBagConstraints.CENTER, GridBagConstraints.NONE, new Insets(5, 5, 5,
5), 0, 0));
105         contentPane.add(btnConfig2, new GridBagConstraints(2, 2, 1, 1,
 0, 0, GridBagConstraints.EAST, GridBagConstraints.NONE, new Insets(5, 5,
5, 5), 0, 0));
106
107         this.getRootPane().setDefaultButton(btnOK);
108     }
109
110     /**
111      * Shows dialog.
112      */
113     public void showDialog() {
114         SwingUtilities.invokeLater(updateCommentLabelRunnable);
115         this.setVisible(true);
116     }
117
118     public void actionPerformed(ActionEvent e) {
119         Object source = e.getSource();
120         if (source == btnOK) {
121             setVisible(false);
122         } else if (source == btnConfig1) {
123             configManagerDialog1.setVisible(true);
124         } else if (source == btnConfig2) {
125             configManagerDialog2.setVisible(true);
126         }
127     }
128
129     private class ManagersTableModel extends AbstractTableModel {
130       private static final long serialVersionUID = 1L;
131
132       private String[] column = {"B", "Name", "Author", "Version", "R"
};
133
134         public int getRowCount() {
135             return managers.getManagerContainers().size();
136         }
137
138         public int getColumnCount() {
139             return 5;
140         }
141
142         public String getColumnName(int columnIndex) {
143             return column[columnIndex];
144         }
145
146         public Class<?> getColumnClass(int columnIndex) {
147             switch (columnIndex) {
148                 case 0:
149                 case 4:
150                     return Boolean.class;
151                 case 1:
152                 case 2:
153                 case 3:
154                     return String.class;
155                 default:
156                     return null;
157             }
158         }
159
160         public boolean isCellEditable(int rowIndex, int columnIndex) {
161             return columnIndex == 0 || columnIndex == 4;
```

```
162         }
163
164         public Object getValueAt(int rowIndex, int columnIndex) {
165             switch (columnIndex) {
166                 case 0:
167                     return managers.getManagerContainers().
get(rowIndex).getManager(0) != null;
168                 case 1:
169                     return managers.getManagerContainers().
get(rowIndex).getManagerInfo().name();
170                 case 2:
171                     return managers.getManagerContainers().
get(rowIndex).getManagerInfo().author();
172                 case 3:
173                     return managers.getManagerContainers().
get(rowIndex).getManagerInfo().version();
174                 case 4:
175                     return managers.getManagerContainers().
get(rowIndex).getManager(1) != null;
176                 default:
177                     return null;
178             }
179         }
180
181         public void setValueAt(Object value, int rowIndex,
int columnIndex) {
182             switch (columnIndex) {
183                 case 0: {
184                   for (ManagerContainer mc: managers.
getManagerContainers())
185                     mc.setManagerEnabled(0, false);
186                     ManagerContainer mc = managers.
getManagerContainers().get(rowIndex);
187                     mc.setManagerEnabled(0, mc.getManager(0) == null);
188                     break;
189                 }
190                 case 4: {
191                   for (ManagerContainer mc: managers.
getManagerContainers())
192                     mc.setManagerEnabled(1, false);
193                     ManagerContainer mc = managers.
getManagerContainers().get(rowIndex);
194                     mc.setManagerEnabled(1, mc.getManager(1) == null);
195                     break;
196                 }
197             }
198             this.fireTableDataChanged();
199         }
200     }
201
202     private class ManagersListSelectionModel extends
DefaultListSelectionModel {
203       private static final long serialVersionUID = 1L;
204
205         private ManagersListSelectionModel() {
206             super();
207             this.setSelectionMode(ListSelectionModel.SINGLE_SELECTION)
;
208         }
209
210         public void setSelectionInterval(int index0, int index1) {
211             super.setSelectionInterval(index0, index1);
212             SwingUtilities.invokeLater(updateCommentLabelRunnable);
```

```
213         }
214
215         public void addSelectionInterval(int index0, int index1) {
216             super.addSelectionInterval(index0, index1);
217             SwingUtilities.invokeLater(updateCommentLabelRunnable);
218         }
219
220         public void removeSelectionInterval(int index0, int index1) {
221             super.removeSelectionInterval(index0, index1);
222             SwingUtilities.invokeLater(updateCommentLabelRunnable);
223         }
224
225         public void setAnchorSelectionIndex(int index) {
226             super.setAnchorSelectionIndex(index);
227             SwingUtilities.invokeLater(updateCommentLabelRunnable);
228         }
229
230         public void setLeadSelectionIndex(int index) {
231             super.setLeadSelectionIndex(index);
232             SwingUtilities.invokeLater(updateCommentLabelRunnable);
233         }
234
235         public void clearSelection() {
236             super.clearSelection();
237             SwingUtilities.invokeLater(updateCommentLabelRunnable);
238         }
239
240         public void insertIndexInterval(int index, int length, boolean
 before) {
241             super.insertIndexInterval(index, length, before);
242             SwingUtilities.invokeLater(updateCommentLabelRunnable);
243         }
244
245         public void removeIndexInterval(int index0, int index1) {
246             super.removeIndexInterval(index0, index1);
247             SwingUtilities.invokeLater(updateCommentLabelRunnable);
248         }
249     }
250
251     private class UpdateCommentLabelRunnable implements Runnable {
252         public void run() {
253             int index = managersListSelectionModel.getMinSelectionInde
x();
254             List<ManagerContainer> mcl = managers.getManagerContainers
();
255             if (index < 0 || index >= mcl.size())
256                 return;
257             commentLabel.setText(mcl.get(index).getManagerInfo().comme
nt());
258         }
259     }
260 }
```

Screen.java

```
001 package ru.ifmo.flyingplates.core.gui;
002
003 import java.awt.BorderLayout;
004 import java.awt.Dimension;
005 import java.awt.GridBagConstraints;
006 import java.awt.GridBagLayout;
007 import java.awt.Insets;
008 import java.awt.Toolkit;
009 import java.awt.event.ActionEvent;
```

```
010 import java.awt.event.WindowAdapter;
011 import java.awt.event.WindowEvent;
012
013 import javax.swing.AbstractAction;
014 import javax.swing.Action;
015 import javax.swing.JCheckBoxMenuItem;
016 import javax.swing.JFrame;
017 import javax.swing.JMenu;
018 import javax.swing.JMenuBar;
019 import javax.swing.JPanel;
020 import javax.swing.JScrollPane;
021 import javax.swing.JSplitPane;
022 import javax.swing.JTable;
023 import javax.swing.JToolBar;
024
025 import com.evelopers.unimod.runtime.ControlledObject;
026 import com.evelopers.unimod.runtime.context.StateMachineContext;
027
028 import ru.ifmo.flyingplates.common.Config;
029 import ru.ifmo.flyingplates.common.Plate;
030 import ru.ifmo.flyingplates.common.Plates;
031 import ru.ifmo.flyingplates.core.settings.ManagersSettings;
032
033 public class Screen extends JFrame implements ControlledObject {
034   private static final long serialVersionUID = 1L;
035
036   public final FieldPanel fieldPanel;
037
038   public final Action startGameAction;
039   public final Action pauseGameAction;
040   public final Action stopGameAction;
041   public final ManagersSettingsAction managersSettingsAction;
042
043   public final JCheckBoxMenuItem cbArea;
044   public final JCheckBoxMenuItem cbTrace;
045   public final JCheckBoxMenuItem cbCircle;
046
047   private JSplitPane splitPanel;
048   private JPanel infoPanel;
049   private JTable infoTable;
050
051     private ManagersDialog managersDialog;
052
053     private ManagersSettings managersSettings;
054
055   public Screen() {
056     super("Flying plates competition");
057     this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
058
059     managersSettings = new ManagersSettings();
060       managersDialog = new ManagersDialog(this, managersSettings);
061
062       fieldPanel = new FieldPanel(this);
063
064     infoTable = new JTable(new PlatesTableModel());
065     infoTable.getColumnModel().getColumn(0).setMaxWidth(15);
066     infoTable.getColumnModel().getColumn(0).setMinWidth(15);
067     infoTable.getColumnModel().getColumn(1).setPreferredWidth(1500);
068     infoTable.getColumnModel().getColumn(2).setPreferredWidth(500);
069     infoTable.getColumnModel().getColumn(3).setPreferredWidth(500);
070     infoTable.getColumnModel().getColumn(4).setPreferredWidth(500);
071     infoTable.getColumnModel().getColumn(5).setPreferredWidth(500);
072     infoTable.getColumnModel().getColumn(0).setCellRenderer(new
```

```
PlateCellRenderer());
073
074     infoTable.getTableHeader().setReorderingAllowed(false);
075     infoTable.setColumnSelectionAllowed(false);
076     infoTable.setSelectionMode(0);
077     infoTable.setRowHeight(15);
078     GridBagLayout gbl = new GridBagLayout();
079     gbl.columnWeights = new double[] {1.0};
080     gbl.rowWeights = new double[] {1.0};
081     infoPanel = new JPanel(gbl);
082     infoPanel.add(new JScrollPane(infoTable), new GridBagConstraints(0
, 0, 1, 1, 1, 1, GridBagConstraints.CENTER, GridBagConstraints.BOTH, new I
nsets(0, 0, 0, 0), 0, 0));
083
084     splitPanel = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, infoPanel
, new JScrollPane(fieldPanel));
085     splitPanel.setOneTouchExpandable(true);
086     splitPanel.setDividerLocation(300);
087     this.getContentPane().add(splitPanel, BorderLayout.CENTER);
088     Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize()
;
089     this.setSize(screenSize.width, screenSize.height - 30);
090     this.setLocation(0, 0);
091     this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
092     this.addWindowListener(new WindowAdapter() {
093       public void windowClosing(WindowEvent e) {
094         ScreenEventProvider.fireEvent(ScreenEventProvider.E4, null);
095       }
096     });
097
098     JMenuBar menu = new JMenuBar();
099
100     JMenu mFile;
101     mFile = new JMenu(new FileAction());
102     mFile.add(new ExitAction());
103     menu.add(mFile);
104
105     mFile = new JMenu(new GameAction());
106     startGameAction = new StartGameAction();
107     mFile.add(startGameAction);
108     pauseGameAction = new PauseContinueGameAction();
109     mFile.add(pauseGameAction);
110     stopGameAction = new StopGameAction();
111     mFile.add(stopGameAction);
112     menu.add(mFile);
113
114     mFile = new JMenu(new SettingsAction());
115     managersSettingsAction = new ManagersSettingsAction();
116         mFile.add(managersSettingsAction);
117     mFile.addSeparator();
118     cbArea = new JCheckBoxMenuItem(new AreaAction());
119     cbArea.setSelected(Config.isScreenInfluenceAreaEnabled());
120     mFile.add(cbArea);
121     cbTrace = new JCheckBoxMenuItem(new TraceAction());
122     cbTrace.setSelected(Config.isScreenTraceEnabled());
123     mFile.add(cbTrace);
124     cbCircle = new JCheckBoxMenuItem(new CircleAction());
125     cbCircle.setSelected(Config.isScreenCircleEnabled());
126     mFile.add(cbCircle);
127     menu.add(mFile);
128
129     JToolBar jtb = new JToolBar();
130     jtb.add(startGameAction);
```

```
131     jtb.add(pauseGameAction);
132     jtb.add(stopGameAction);
133     this.add(jtb, BorderLayout.NORTH);
134
135     this.setJMenuBar(menu);
136     this.setVisible(true);
137   }
138
139   private class ExitAction extends AbstractAction {
140     private static final long serialVersionUID = 1L;
141
142     public ExitAction() {
143       putValue(Action.NAME, "Exit");
144       putValue(Action.SHORT_DESCRIPTION, "Exit application");
145       putValue(Action.MNEMONIC_KEY, new Integer('x'));
146     }
147
148     public void actionPerformed(ActionEvent e) {
149       ScreenEventProvider.fireEvent(ScreenEventProvider.E4, null);
150     }
151   }
152
153   private class AreaAction extends AbstractAction {
154     private static final long serialVersionUID = 1L;
155
156     public AreaAction() {
157       putValue(Action.NAME, "Area");
158       putValue(Action.SHORT_DESCRIPTION, "TODO");
159       putValue(Action.MNEMONIC_KEY, new Integer('A'));
160       setEnabled(true);
161     }
162
163     public void actionPerformed(ActionEvent e) {
164     }
165   }
166
167   private class CircleAction extends AbstractAction {
168     private static final long serialVersionUID = 1L;
169
170     public CircleAction() {
171       putValue(Action.NAME, "Circle");
172       putValue(Action.SHORT_DESCRIPTION, "TODO");
173       putValue(Action.MNEMONIC_KEY, new Integer('C'));
174       setEnabled(true);
175     }
176
177     public void actionPerformed(ActionEvent e) {
178     }
179   }
180
181   private class FileAction extends AbstractAction {
182     private static final long serialVersionUID = 1L;
183
184     public FileAction() {
185       putValue(Action.NAME, "File");
186       putValue(Action.MNEMONIC_KEY, new Integer('F'));
187     }
188
189     public void actionPerformed(ActionEvent e) {
190     }
191   }
192
193   private class GameAction extends AbstractAction {
```

```
194     private static final long serialVersionUID = 1L;
195
196     public GameAction() {
197       putValue(Action.NAME, "Game");
198       putValue(Action.MNEMONIC_KEY, new Integer('G'));
199     }
200
201     public void actionPerformed(ActionEvent e) {
202     }
203   }
204
205   private class PauseContinueGameAction extends AbstractAction {
206     private static final long serialVersionUID = 1L;
207
208     public static final String PAUSE = "Pause";
209
210     public static final String CONTINUE = "Continue";
211
212     public PauseContinueGameAction() {
213       this.putValue(Action.NAME, PAUSE);
214       this.putValue(Action.SHORT_DESCRIPTION, "Pause/Continue game");
215       this.putValue(Action.MNEMONIC_KEY, new Integer('p'));
216       this.setEnabled(false);
217     }
218
219     public void actionPerformed(ActionEvent e) {
220       ScreenEventProvider.fireEvent(ScreenEventProvider.E2, null);
221     }
222   }
223
224   private class SettingsAction extends AbstractAction {
225     private static final long serialVersionUID = 1L;
226
227     public SettingsAction() {
228       putValue(Action.NAME, "Settings");
229       putValue(Action.MNEMONIC_KEY, new Integer('S'));
230     }
231
232     public void actionPerformed(ActionEvent e) {
233     }
234   }
235
236   private class StartGameAction extends AbstractAction {
237     private static final long serialVersionUID = 1L;
238
239     public StartGameAction() {
240       putValue(Action.NAME, "Start");
241       putValue(Action.SHORT_DESCRIPTION, "Start game");
242       putValue(Action.MNEMONIC_KEY, new Integer('s'));
243       setEnabled(true);
244     }
245
246     public void actionPerformed(ActionEvent e) {
247       ScreenEventProvider.fireEvent(ScreenEventProvider.E1, null);
248       ScreenEventProvider.fireEvent(ScreenEventProvider.E400, null);
249     }
250   }
251
252   private class StopGameAction extends AbstractAction {
253     private static final long serialVersionUID = 1L;
254
255     public StopGameAction() {
256       putValue(Action.NAME, "Stop");
```

```
257       putValue(Action.SHORT_DESCRIPTION, "Stop game");
258       putValue(Action.MNEMONIC_KEY, new Integer('t'));
259       setEnabled(false);
260     }
261
262     public void actionPerformed(ActionEvent e) {
263       ScreenEventProvider.fireEvent(ScreenEventProvider.E3, null);
264     }
265   }
266
267   private class TraceAction extends AbstractAction {
268     private static final long serialVersionUID = 1L;
269
270     public TraceAction() {
271       putValue(Action.NAME, "Trace");
272       putValue(Action.SHORT_DESCRIPTION, "TODO");
273       putValue(Action.MNEMONIC_KEY, new Integer('T'));
274       setEnabled(true);
275     }
276
277     public void actionPerformed(ActionEvent e) {
278     }
279   }
280
281     private class ManagersSettingsAction extends AbstractAction {
282       private static final long serialVersionUID = 1L;
283
284       private ManagersSettingsAction() {
285             putValue(Action.NAME, "Managers");
286             putValue(Action.SHORT_DESCRIPTION, "Configure managers");
287             putValue(Action.MNEMONIC_KEY, (int) 'M');
288         }
289
290         public void actionPerformed(ActionEvent e) {
291             managersDialog.showDialog();
292 //            updateManagersMenu();
293         }
294     }
295
296
297 //----------------
298   /**
299    * @unimod.action.descr Dispose window
300    */
301   public void z10(StateMachineContext context) {
302     this.dispose();
303   }
304
305   /**
306    * @unimod.action.descr Repaint window
307    */
308   public void z12(StateMachineContext context) {
309     this.repaint();
310   }
311
312   /**
313    * @unimod.action.descr Disable StartGameAction
314    */
315   public void z310(StateMachineContext context) {
316     this.startGameAction.setEnabled(false);
317   }
318
319   /**
```

```
320     * @unimod.action.descr Enable StartGameAction
321     */
322    public void z311(StateMachineContext context) {
323      this.startGameAction.setEnabled(true);
324    }
325
326    /**
327     * @unimod.action.descr Disable PauseContinueAction
328     */
329    public void z320(StateMachineContext context) {
330      this.pauseGameAction.setEnabled(false);
331    }
332
333    /**
334     * @unimod.action.descr Enable PauseContinueAction
335     */
336    public void z321(StateMachineContext context) {
337      this.pauseGameAction.setEnabled(true);
338    }
339
340    /**
341     * @unimod.action.descr Change title of PauseContinueButton to "PAUS
E"
342     */
343    public void z322(StateMachineContext context) {
344      this.pauseGameAction.putValue(Action.NAME,
345          PauseContinueGameAction.PAUSE);
346    }
347
348    /**
349     * @unimod.action.descr Change title of PauseContinueButton to "CONT
INUE"
350     */
351    public void z323(StateMachineContext context) {
352      this.pauseGameAction.putValue(Action.NAME,
353          PauseContinueGameAction.CONTINUE);
354    }
355
356    /**
357     * @unimod.action.descr Disable StopGameAction
358     */
359    public void z330(StateMachineContext context) {
360      this.stopGameAction.setEnabled(false);
361    }
362
363    /**
364     * @unimod.action.descr Enable StopGameAction
365     */
366    public void z331(StateMachineContext context) {
367      this.stopGameAction.setEnabled(true);
368    }
369
370    /**
371     * @unimod.action.descr Disable ManagersSettingsAction
372     */
373    public void z340(StateMachineContext context) {
374      this.managersSettingsAction.setEnabled(false);
375    }
376
377    /**
378     * @unimod.action.descr Enable ManagersSettignsAction
379     */
380    public void z341(StateMachineContext context) {
```

```
381     this.managersSettingsAction.setEnabled(true);
382   }
383
384   /**
385    * @unimod.action.descr Inform user about result of the game
386    */
387   public void z4(StateMachineContext context) {
388     double[] maxX = new double[2];
389     for (Plate plate: Plates.getPlates()) {
390       if (!plate.isLanded())
391         continue;
392       if (maxX[plate.getPlayer()] < plate.getPosition().x)
393         maxX[plate.getPlayer()] = plate.getPosition().x;
394     }
395     new GameResultDialog(this, maxX[0], maxX[1]);
396   }
397 }
```

FieldPanel.java

```
001 package ru.ifmo.flyingplates.core.gui;
002
003 import java.awt.BasicStroke;
004 import java.awt.Color;
005 import java.awt.Dimension;
006 import java.awt.Graphics;
007 import java.awt.Graphics2D;
008 import java.awt.Stroke;
009 import java.awt.geom.Ellipse2D;
010 import java.awt.geom.Line2D;
011 import java.util.ArrayList;
012 import java.util.HashMap;
013 import java.util.List;
014
015 import javax.swing.JPanel;
016
017 import ru.ifmo.flyingplates.common.Config;
018 import ru.ifmo.flyingplates.common.Plate;
019 import ru.ifmo.flyingplates.common.Plates;
020 import ru.ifmo.flyingplates.utils.Vector;
021
022 public class FieldPanel extends JPanel {
023   private static final long serialVersionUID = 1L;
024
025   private static final int MARGIN_UP = 25;
026   private static final int MARGIN_DOWN = 25;
027   private static final int MARGIN_LEFT = 25;
028   private static final int MARGIN_RIGHT = 25;
029
030   private Object oldPlates = null;
031
032   private HashMap<Plate, ArrayList<Vector>> traces = new
HashMap<Plate, ArrayList<Vector>>();
033
034   private Ellipse2D.Double ellipse = new Ellipse2D.Double();
035
036   private Stroke strokeNormal = new BasicStroke(1);
037   private Stroke strokeBold = new BasicStroke(3);
038
039   private Screen par;
040
041   public FieldPanel(Screen par) {
042     this.par = par;
043     Dimension size = new Dimension(MARGIN_LEFT + Config.getFieldWidth(
```

```
)
044         * Config.getPixelsInMeter() + MARGIN_RIGHT, MARGIN_UP + Config
.getFieldHeight()
045         * Config.getPixelsInMeter() + MARGIN_DOWN);
046     this.setSize(size);
047     this.setPreferredSize(size);
048     this.setDoubleBuffered(true);
049   }
050
051   public void paint(Graphics g) {
052     super.paint(g);
053     Graphics2D graphics2D = (Graphics2D) g;
054     List<Plate> plates = Plates.getPlates();
055     if (plates == null)
056       return;
057
058     double[] maxX = new double[2];
059     for (Plate plate: plates) {
060       if (plate.isCrashed())
061         continue;
062       if (maxX[plate.getPlayer()] < plate.getPosition().x)
063         maxX[plate.getPlayer()] = plate.getPosition().x;
064     }
065
066     graphics2D.setStroke(strokeNormal);
067     graphics2D.setColor(Color.BLACK);
068     graphics2D.draw(new Line2D.Double(MARGIN_LEFT, MARGIN_UP,
MARGIN_LEFT,
069         MARGIN_UP + Config.getFieldHeight() * Config.getPixelsInMeter(
)));
070     graphics2D.draw(new Line2D.Double(MARGIN_LEFT + Config.
getFieldWidth()
071             * Config.getPixelsInMeter(), MARGIN_UP, MARGIN_LEFT
072             + Config.getFieldWidth() * Config.getPixelsInMeter(),
MARGIN_UP
073             + Config.getFieldHeight() * Config.getPixelsInMeter()));
074     graphics2D.draw(new Line2D.Double(MARGIN_LEFT + maxX[0] *
Config.getPixelsInMeter(),
075         MARGIN_UP, MARGIN_LEFT + Config.getFieldWidth() *
Config.getPixelsInMeter(),
076         MARGIN_UP));
077     graphics2D.draw(new Line2D.Double(MARGIN_LEFT + maxX[1] * Config.g
etPixelsInMeter(),
078         MARGIN_UP + Config.getFieldHeight() *
Config.getPixelsInMeter(), MARGIN_LEFT
079             + Config.getFieldWidth() * Config.getPixelsInMeter(),
MARGIN_UP
080             + Config.getFieldHeight() * Config.getPixelsInMeter()));
081
082     graphics2D.setStroke(strokeBold);
083     graphics2D.setColor(Color.BLUE);
084     graphics2D.draw(new Line2D.Double(MARGIN_LEFT, MARGIN_UP,
MARGIN_LEFT
085         + maxX[0] * Config.getPixelsInMeter(), MARGIN_UP));
086
087     graphics2D.setStroke(strokeBold);
088     graphics2D.setColor(Color.RED);
089     graphics2D.draw(new Line2D.Double(MARGIN_LEFT, MARGIN_UP
090             + Config.getFieldHeight() * Config.getPixelsInMeter(),
MARGIN_LEFT
091             + maxX[1] * Config.getPixelsInMeter(), MARGIN_UP + Config.
getFieldHeight()
092             * Config.getPixelsInMeter()));
```

```
093
094     graphics2D.setStroke(strokeNormal);
095
096     if (par.cbArea.isSelected()) {
097       /*
098        * Рисуем зоны положительного и отрицательного
099        * аэродинамического влияния тарелок.
100        */
101
102       for (Plate plate: plates) {
103         if (!plate.isFlying())
104           continue;
105
106         int angle = (int) Math.round(Math.atan2(-
plate.getSpeed().y, plate.getSpeed().x) * 180 / Math.PI) - 180;
107
108         graphics2D.setPaint(new Color(1f, 0f, 0f, 0.3f));
109         graphics2D.fillArc((int) (MARGIN_LEFT + (plate.getPosition().x
  - Config.getInfluenceDistance()) * Config.getPixelsInMeter()),
110             (int) (MARGIN_UP + (plate.getPosition().y -
 Config.getInfluenceDistance()) * Config.getPixelsInMeter()),
111             2 * Config.getInfluenceDistance() * Config.getPixelsInMete
r(),
112             2 * Config.getInfluenceDistance() * Config.getPixelsInMete
r(),
113             angle - 40, 20);
114
115         graphics2D.fillArc((int) (MARGIN_LEFT + (plate.getPosition().x
  - Config.getInfluenceDistance()) * Config.getPixelsInMeter()),
116             (int) (MARGIN_UP + (plate.getPosition().y -
 Config.getInfluenceDistance()) * Config.getPixelsInMeter()),
117             2 * Config.getInfluenceDistance() * Config.getPixelsInMete
r(),
118             2 * Config.getInfluenceDistance() * Config.getPixelsInMete
r(),
119             angle + 20, 20);
120
121         graphics2D.setPaint(new Color(0f, 0f, 1f, 0.3f));
122
123         graphics2D.fillArc((int) (MARGIN_LEFT + (plate.getPosition().x
  - Config.getInfluenceDistance()) * Config.getPixelsInMeter()),
124             (int) (MARGIN_UP + (plate.getPosition().y -
 Config.getInfluenceDistance()) * Config.getPixelsInMeter()),
125             2 * Config.getInfluenceDistance() * Config.getPixelsInMete
r(),
126             2 * Config.getInfluenceDistance() * Config.getPixelsInMete
r(),
127             angle - 20, 40);
128       }
129     }
130
131     if (par.cbTrace.isSelected()) {
132       // Рисуем traces of plates
133       if (oldPlates != plates) {
134         oldPlates = plates;
135         traces.clear();
136       }
137
138       for (Plate plate: plates) {
139         ArrayList<Vector> trace = traces.get(plate);
140         if (trace == null) {
141           trace = new ArrayList<Vector>();
142           traces.put(plate, trace);
```

```
143         }
144         trace.add(plate.getPosition());
145         graphics2D.setColor(plate.getColor());
146         for (int t = 1; t < trace.size(); t++) {
147           Vector pp = trace.get(t - 1);
148           Vector cp = trace.get(t);
149           graphics2D.drawLine(MARGIN_LEFT + (int) (pp.x *
Config.getPixelsInMeter()),
150               MARGIN_UP + (int) (pp.y * Config.getPixelsInMeter()),
151               MARGIN_LEFT + (int) (cp.x * Config.getPixelsInMeter()),
152               MARGIN_UP + (int) (cp.y * Config.getPixelsInMeter()));
153         }
154       }
155     }
156
157     if (par.cbCircle.isSelected()) {
158       // Рисуем окружности вокруг тарелок, отмасштабированные Config.T
IME_STEP
159       graphics2D.setColor(Color.DARK_GRAY);
160       for (Plate plate: plates) {
161         if (!plate.isFlying())
162           continue;
163         ellipse.setFrame(MARGIN_LEFT + (plate.getPosition().x -
 plate.getSpeed().getLength() * Config.getTimeStep()) *
Config.getPixelsInMeter(),
164             MARGIN_UP + (plate.getPosition().y -
 plate.getSpeed().getLength() * Config.getTimeStep()) *
Config.getPixelsInMeter(),
165             2 * plate.getSpeed().getLength() * Config.getTimeStep() *
Config.getPixelsInMeter(),
166             2 * plate.getSpeed().getLength() * Config.getTimeStep() *
Config.getPixelsInMeter());
167         graphics2D.draw(ellipse);
168       }
169     }
170     // Рисуем тарелки
171     for (Plate plate: plates) {
172       ellipse.setFrame(MARGIN_LEFT + (plate.getPosition().x -
 0.5 * Config.getPlateDiameter()) * Config.getPixelsInMeter(),
173           MARGIN_UP + (plate.getPosition().y -
 0.5 * Config.getPlateDiameter()) * Config.getPixelsInMeter(),
174           Config.getPlateDiameter() * Config.getPixelsInMeter(),
175           Config.getPlateDiameter() * Config.getPixelsInMeter());
176       graphics2D.setPaint(plate.getColor());
177       graphics2D.fill(ellipse);
178       graphics2D.setColor(plate.getBorderColor());
179         graphics2D.draw(ellipse);
180
181       String text = "" + (plate.getId() + 1);
182       int textX = (int) (ellipse.getCenterX() - 3);
183       int textY = (int) (ellipse.getCenterY() + 5);
184       graphics2D.setColor(Color.WHITE);
185       graphics2D.drawString(text, textX, textY);
186     }
187   }
188 }
```

ColorCellRenderer.java

```
01 package ru.ifmo.flyingplates.core.gui;
02
03 import java.awt.Color;
04
05 import javax.swing.table.DefaultTableCellRenderer;
06
```

```
07 public class ColorCellRenderer extends DefaultTableCellRenderer {
08   private static final long serialVersionUID = 1L;
09
10   public void setValue(Object value) {
11       setBackground((Color) value);
12     }
13 }
```

PlateCellRenderer.java

```
01 package ru.ifmo.flyingplates.core.gui;
02
03 import java.awt.Color;
04 import java.awt.Component;
05 import java.awt.Graphics;
06 import java.awt.Graphics2D;
07
08 import javax.swing.JLabel;
09 import javax.swing.JTable;
10 import javax.swing.table.TableCellRenderer;
11
12 import ru.ifmo.flyingplates.common.Plate;
13
14 public class PlateCellRenderer implements TableCellRenderer {
15   private static ColoredLabel label = new ColoredLabel();
16
17   public Component getTableCellRendererComponent(JTable table, Object v
alue, boolean isSelected, boolean hasFocus, int row, int column) {
18     if (value == null)
19       return null;
20     Plate p = (Plate) value;
21     label.setText("" + (p.getId() + 1));
22     label.backgroundColor = p.getColor();
23     label.borderColor = p.getBorderColor();
24     return label;
25   }
26
27   private static class ColoredLabel extends JLabel {
28     private static final long serialVersionUID = 1L;
29
30     private Color backgroundColor;
31     private Color borderColor;
32
33     private ColoredLabel() {
34       setHorizontalAlignment(JLabel.CENTER);
35       setForeground(Color.WHITE);
36     }
37
38     public void paint(Graphics g) {
39       Graphics2D g2 = (Graphics2D) g;
40       g2.setPaint(backgroundColor);
41       g2.fillRect(0, 0, this.getWidth() - 1, this.getHeight() - 1);
42       super.paint(g);
43       g2.setColor(borderColor);
44       g2.drawRect(0, 0, this.getWidth() - 1, this.getHeight() - 1);
45     }
46   }
47 }
```

PlatesTableModel.java

```
01 package ru.ifmo.flyingplates.core.gui;
02
03 import java.util.List;
04 import java.util.Locale;
```

```
05
06 import javax.swing.table.AbstractTableModel;
07
08 import ru.ifmo.flyingplates.common.Config;
09 import ru.ifmo.flyingplates.common.Plate;
10 import ru.ifmo.flyingplates.common.Plates;
11
12 public class PlatesTableModel extends AbstractTableModel {
13   private static final long serialVersionUID = 1L;
14
15   private static final String[] name = new String[] {
16     "ID", "X    Y", "Speed", "Fuel", "Q", "A"};
17
18
19     public String getColumnName(int columnIndex) {
20       return name[columnIndex];
21     }
22
23     public int getRowCount() {
24     return 2 * Config.getPlatesCount();
25   }
26
27   public int getColumnCount() {
28     return name.length;
29   }
30
31   public Object getValueAt(int rowIndex, int columnIndex) {
32     List<Plate> plates = Plates.getPlates();
33     if (plates == null || plates.size() == 0)
34       return null;
35     Plate plate = plates.get(rowIndex);
36     switch (columnIndex) {
37       case 0:
38         return plate;
39       case 1:
40         return String.format(Locale.US, "%.2f  %.2f",
plate.getPosition().x, plate.getPosition().y);
41       case 2:
42         return String.format(Locale.US, "%.3f", plate.getSpeed().
getLength());
43       case 3:
44         return String.format(Locale.US, "%.3f", plate.getFuel());
45       case 4:
46         return String.format(Locale.US, "%.3f", plate.getQ());
47       case 5:
48         return String.format(Locale.US, "%.3f", plate.getA());
49       default:
50         return null;
51     }
52   }
53
54 }
```

ManagerContainer.java

```
01 package ru.ifmo.flyingplates.core.settings;
02
03 import java.io.File;
04 import java.io.IOException;
05 import java.net.URL;
06 import java.net.URLClassLoader;
07 import java.util.jar.Attributes;
08 import java.util.jar.JarFile;
09
```

```
10 import ru.ifmo.flyingplates.common.Manager;
11 import ru.ifmo.flyingplates.common.ManagerInfo;
12 import ru.ifmo.flyingplates.core.exceptions.AnnotationNotFoundException
;
13 import ru.ifmo.flyingplates.core.exceptions.InterfaceNotImplementedExce
ption;
14
15 /**
16  * Description.
17  *
18  * @author Dmitry Paraschenko
19  * @version 13.01.2006, 13:58:10
20  */
21 public class ManagerContainer {
22     private ManagerInfo info;
23     private ClassLoader[] classLoader = new ClassLoader[2];
24     private Class<?>[] managerClass = new Class<?>[2];
25     private Manager[] manager = new Manager[2];
26
27     @SuppressWarnings("unchecked")
28     public ManagerContainer(File file) throws IOException,
ClassNotFoundException, AnnotationNotFoundException,
InterfaceNotImplementedException {
29         String mainClass = (new JarFile(file)).getManifest().
getMainAttributes().getValue(Attributes.Name.MAIN_CLASS);
30 //        ClassLoader cl = new URLClassLoader(new URL[]{file.toURL()});
31         for (int i = 0; i < 2; i++) {
32           classLoader[i] = new URLClassLoader(new URL[]{file.toURL()});
33           managerClass[i] = classLoader[i].loadClass(mainClass);
34           info = managerClass[i].getAnnotation(ManagerInfo.class);
35           if (info == null)
36               throw new AnnotationNotFoundException();
37           Class[] inter = managerClass[i].getInterfaces();
38           boolean ok = false;
39           for (Class c : inter) {
40               if (c == Manager.class)
41                   ok = true;
42           }
43           if (!ok)
44               throw new InterfaceNotImplementedException();
45         }
46     }
47
48     public ManagerInfo getManagerInfo() {
49         return info;
50     }
51
52     public Manager getManager(int id) {
53         return manager[id];
54     }
55
56     public void setManagerEnabled(int id, boolean enabled) {
57         if (enabled) {
58             try {
59                 manager[id] = (Manager) managerClass[id].newInstance();
60             } catch (InstantiationException e) {
61                 e.printStackTrace();
62             } catch (IllegalAccessException e) {
63                 e.printStackTrace();
64             }
65         } else {
66             manager[id] = null;
67         }
```

```
68      }
69
70      public ClassLoader getClassLoader(int id) {
71        return classLoader[id];
72      }
73  }
```

ManagersSettings.java

```
001 package ru.ifmo.flyingplates.core.settings;
002
003 import java.util.ArrayList;
004 import java.util.Collections;
005 import java.util.List;
006 import java.io.File;
007 import java.io.FilenameFilter;
008 import java.io.IOException;
009
010 import ru.ifmo.flyingplates.core.exceptions.AnnotationNotFoundExceptio
n;
011 import ru.ifmo.flyingplates.core.exceptions.InterfaceNotImplementedExc
eption;
012
013 /**
014  * Description.
015  *
016  * @author Dmitry Paraschenko
017  * @version 18.01.2006, 13:12:40
018  */
019 public class ManagersSettings {
020     private static ManagersSettings ms;
021
022   private ArrayList<ManagerContainer> managers = new ArrayList<Manager
Container>();
023
024     public ManagersSettings() {
025       ms = this;
026         File managersDir = new File("managers");
027         if (!managersDir.exists()) {
028             managersDir.mkdirs();
029         }
030         FilenameFilter filter = new FilenameFilter() {
031             public boolean accept(File dir, String name) {
032                 return name.endsWith(".jar");
033             }
034         };
035         String[] names = managersDir.list(filter);
036         for (String s : names) {
037             try {
038                 File file = new File(managersDir, s);
039                 managers.add(new ManagerContainer(file));
040             } catch (IOException e) {
041                 e.printStackTrace();
042             } catch (ClassNotFoundException e) {
043                 e.printStackTrace();
044             } catch (InterfaceNotImplementedException e) {
045                 e.printStackTrace();
046             } catch (AnnotationNotFoundException e) {
047                 e.printStackTrace();
048             } catch (Exception e) {
049                 e.printStackTrace();
050             }
051         }
052         managers.trimToSize();
```

```
053         if (managers.size() > 0) {
054           ManagerContainer mc = managers.get(0);
055           mc.setManagerEnabled(0, true);
056           mc.setManagerEnabled(1, true);
057         }
058     }
059
060 //    public ManagersSettings(Node node) {
061 //        this();
062 //        logger.entering(ManagersSettings.class.getCanonicalName(), "
<init>", node);
063 //        if (node.getNodeType() != Node.ELEMENT_NODE || !node.getNode
Name().equals("managers"))
064 //            throw new RuntimeException();
065 //        HashSet<String> lp = new HashSet<String>();
066 //        NodeList nl = node.getChildNodes();
067 //        for (int nid = 0; nid < nl.getLength(); nid++) {
068 //            Node n = nl.item(nid);
069 //            if (n.getNodeType() != Node.ELEMENT_NODE || !n.getNodeNa
me().equals("manager"))
070 //                continue;
071 //            lp.add(n.getAttributes().getNamedItem("name").getNodeVal
ue());
072 //        }
073 //        for (ManagerContainer pc: managers) {
074 //            if (lp.contains(pc.getManagerInfo().name()))
075 //                pc.setManagerEnabled(true);
076 //        }
077 //        logger.exiting(ManagersSettings.class.getCanonicalName(), "<
init>");
078 //    }
079 //
080 //    public void storeToXML(PrintWriter pw) {
081 //        logger.entering(ManagersSettings.class.getCanonicalName(), "
storeToXML");
082 //        pw.println("<managers>");
083 //        for (ManagerContainer pc: managers) {
084 //            if (pc.getManager() != null) {
085 //                pw.println("<manager name=\"" + pc.getManagerInfo().
name() + "\" />");
086 //            }
087 //        }
088 //        pw.println("</managers>");
089 //        logger.exiting(ManagersSettings.class.getCanonicalName(), "s
toreToXML");
090 //    }
091
092
093     public List<ManagerContainer> getManagerContainers() {
094       return Collections.unmodifiableList(managers);
095     }
096
097     public static ManagersSettings getManagersSettings() {
098       return ms;
099     }
100 }
```

Vector.java

```
01 package ru.ifmo.flyingplates.utils;
02
03 public class Vector {
04   public final double x;
05   public final double y;
```

```
06
07   public Vector(double x, double y) {
08     this.x = x;
09     this.y = y;
10   }
11
12   public double getLength() {
13     return Math.sqrt(x * x + y * y);
14   }
15
16   public Vector multiply(double a) {
17     return new Vector(x * a, y * a);
18   }
19
20   public double multiply(Vector v) {
21     return x * v.x + y * v.y;
22   }
23
24   public Vector add(Vector v) {
25     return new Vector(x + v.x, y + v.y);
26   }
27
28   public Vector subtract(Vector v) {
29     return new Vector(x - v.x, y - v.y);
30   }
31
32   public Vector rotate(double ang) {
33     return new Vector( x * Math.cos(ang) + y * Math.sin(ang),
34                 -x * Math.sin(ang) + y * Math.cos(ang));
35   }
36
37 }
```

Приложение 2. Исходные коды летающей тарелки

AutomatonManager.java

```
001 package ru.ifmo.flyingplates.managers.automatonmanager;
002
003 import java.io.IOException;
004 import java.io.InputStream;
005
006 import javax.swing.JDialog;
007
008 import org.apache.commons.logging.LogFactory;
009
010 import ru.ifmo.flyingplates.common.Config;
011 import ru.ifmo.flyingplates.common.Manager;
012 import ru.ifmo.flyingplates.common.ManagerInfo;
013 import ru.ifmo.flyingplates.common.Plates;
014 import ru.ifmo.flyingplates.managers.automatonmanager.flyingplate.co.F
lyingPlate;
015
016 import com.evelopers.common.exception.CommonException;
017 import com.evelopers.unimod.adapter.standalone.Run;
018 import com.evelopers.unimod.core.stateworks.Event;
019 import com.evelopers.unimod.core.stateworks.Model;
020 import com.evelopers.unimod.debug.ExceptionHandlerImpl;
021 import com.evelopers.unimod.runtime.ControlledObject;
```

```
022 import com.evelopers.unimod.runtime.ControlledObjectsMap;
023 import com.evelopers.unimod.runtime.EventProcessorListener;
024 import com.evelopers.unimod.runtime.EventProvider;
025 import com.evelopers.unimod.runtime.ExceptionHandler;
026 import com.evelopers.unimod.runtime.ModelEngine;
027 import com.evelopers.unimod.runtime.context.StateMachineContextImpl;
028 import com.evelopers.unimod.runtime.interpretation.InterpretationHelpe
r;
029 import com.evelopers.unimod.runtime.logger.SimpleLogger;
030 import com.evelopers.unimod.transform.TransformException;
031 import com.evelopers.unimod.transform.xml.XMLToModel;
032
033 /**
034  *
035  * @author Fedor Tsarev
036  *
037  */
038 @ManagerInfo(name="AutomatonManager", author="Fedor Tsarev", version="
0.5", comment="MultiAgent Automaton-Based Manager, ver. 0.5")
039 public class AutomatonManager implements Manager, EventProvider {
040
041 //  public static final String PLATE_AUTOMATON_FILE = "e:\\eclipse_wor
kspace\\FlyingPlates\\resources\\afp.xml";
042 //  public static final String PLATE_AUTOMATON_FILE = "D:/Development/
Projects/Eclipse_ws/FlyingPlates/resources/afp.xml";
043   public static final String PLATE_AUTOMATON_FILE = "AFP.xml";
044
045   public static ModelEngine[] engines;
046   private static int platesCount = Config.getPlatesCount();
047
048   private ModelEngine engine;
049   private ClassLoader cl;
050
051   public void dispose() {
052     // TODO Auto-generated method stub
053   }
054
055   /**
056    * Creates automata for flyingplates.
057    *
058    */
059   public void initPlatesAutomata() {
060
061     platesCount = Config.getPlatesCount();
062     engines = new ModelEngine[platesCount];
063
064     for (int i = 0; i < platesCount; i++) {
065           Model model;
066       /* transform XML to state machine in-memory model */
067           try {
068             InputStream is = cl.getResourceAsStream(PLATE_AUTOMATON_FI
LE);
069             if (is == null) {
070               System.out.println("is == null");
071               System.exit(0);
072             }
073             model = XMLToModel.loadAndCompile(is);
074           } catch (IOException e) {
075             System.out.println("[ERROR]   Can't load xml: " +
PLATE_AUTOMATON_FILE);
076             // TODO
077             throw new RuntimeException();
078           } catch (TransformException e) {
```

```
079             System.err.println(e.getMessage());
080             // TODO
081             throw new RuntimeException();
082           }
083
084           /* create runtime engine */
085           InterpretationHelper helper = InterpretationHelper.
getInstance();
086
087           ModelEngine engine1;
088
089           try {
090             FlyingPlate fp = new FlyingPlate(player, i);
091             Plates.getPlate(player, i).addSetter(this, fp);
092             engine1 = helper.createBuildInModelEngine(model,
new FlyingPlateCOMap(fp),true);
093             fp.setModelEngine(engine1);
094
095           } catch (CommonException e) {
096             System.err.println(e.getMessage());
097             // TODO
098             throw new RuntimeException();
099           }
100
101           EventProcessorListener logger = new SimpleLogger(LogFactory.
getLog(Run.class));
102           engine1.getEventProcessor().addEventProcessorListener(logger
);
103
104           ExceptionHandler eh = new ExceptionHandlerImpl();
105           engine1.getEventProcessor().addExceptionHandler(eh);
106
107           engines[i] = engine1;
108       }
109     }
110
111     public void init(ModelEngine engine) throws CommonException {
112 //      this.engine = engine;
113     }
114
115     public void sendEventToPlate(int pNum, String event) {
116       System.out.println("[EVENT]    Sending event " + event + " to plat
e number " + pNum);
117       engines[pNum].getEventManager().handleAndWait(new Event(event),
118           StateMachineContextImpl.create());
119     }
120
121     public void init(ClassLoader cl, int player) {
122       this.cl = cl;
123       this.player = player;
124       initPlatesAutomata();
125       for (int i = 0; i < Config.getPlatesCount(); i++) {
126         sendEventToPlate(i, "e1");
127       }
128       System.out.println("Init is done");
129     }
130
131     public void doTurn() {
132       for (int i = 0; i < Config.getPlatesCount(); i++) {
133         sendEventToPlate(i, FlyingPlate.NEXT_TICK_HAPENNED);
134       }
135       System.out.println("Turn is done");
136     }
```

```
137
138   public JDialog configManager(JDialog mainDialog) {
139     // TODO Auto-generated method stub
140     return null;
141   }
142
143   private int player = -1;
144
145   /**
146    * Controlled object map implementation for
147    * flying plate. Returns controlled object,
148    * which is specified when constructing.
149    * @author Fedor Tsarev
150    *
151    */
152   private static class FlyingPlateCOMap implements ControlledObjectsMa
p {
153
154     private ControlledObject co;
155
156     public FlyingPlateCOMap(ControlledObject co) {
157       this.co = co;
158     }
159
160     public ControlledObject getControlledObject(String coName) {
161       return co;
162     }
163   }
164 }
```

FlyingPlate.java

```
001 package ru.ifmo.flyingplates.managers.automatonmanager.flyingplate.co;
002
003 import com.evelopers.unimod.core.stateworks.Event;
004 import com.evelopers.unimod.runtime.ControlledObject;
005 import com.evelopers.unimod.runtime.EventManager;
006 import com.evelopers.unimod.runtime.ModelEngine;
007
008 import ru.ifmo.flyingplates.common.Config;
009 import ru.ifmo.flyingplates.common.Plate;
010 import ru.ifmo.flyingplates.common.Plates;
011 import ru.ifmo.flyingplates.utils.Vector;
012
013 import com.evelopers.unimod.runtime.context.StateMachineContext;
014 import com.evelopers.unimod.runtime.context.StateMachineContextImpl;
015
016 /**
017  *
018  * @author Fedor Tsarev
019  *
020  */
021 public class FlyingPlate implements ControlledObject {
022
023   public static String NORMAL_FUEL_CONSUMPTION = "e1";
024   public static String LEFT_BORDER_IS_NEAR = "e2";
025   public static String RIGHT_BORDER_IS_NEAR = "e3";
026   public static String OTHER_PLATE_ON_THE_LEFT = "e4";
027   public static String OTHER_PLATE_ON_THE_RIGHT = "e5";
028   public static String OTHER_PLATE_IN_FRONT = "e6";
029   public static String OTHER_PLATE_BEHIND = "e7";
030   public static String FLY_HORIZONTALLY = "e10";
031   public static String NEXT_TICK_HAPENNED = "e101";
032
```

```
033   public static String LAND_OK = "e21";
034   public static String LAND_CRASHED = "e22";
035
036   private int flyingPlateNumber;
037   private int player;
038
039   private Plate controlledPlate;
040
041   private ModelEngine engine;
042   private EventManager em;
043
044   private final double EPS = 1e-9;
045
046   /**
047    *
048    * @param fpn
049    */
050   public FlyingPlate(int player, int fpn) {
051     flyingPlateNumber = fpn;
052     this.player = player;
053   }
054
055   public void setModelEngine(ModelEngine me) {
056     this.engine = me;
057     em = engine.getEventManager();
058   }
059
060   protected void sendEventToPlate(String event) {
061     System.err.println("Plate #" + flyingPlateNumber + "[EVENT]    Sen
ding event " + event + " to plate number ");
062     em.handle(new Event(event),
063         StateMachineContextImpl.create());
064   }
065
066   private void setControlledPlate() {
067     if (controlledPlate == null) {
068       controlledPlate = Plates.getPlate(player, flyingPlateNumber);
069     }
070   }
071
072   /**
073    * @unimod.action.descr Normal speed
074    *
075    */
076   public void z1(StateMachineContext context) {
077     setControlledPlate();
078     controlledPlate.setQ(this, 0.40);
079   }
080
081   /*
082    * Normalize all parameters of the plate
083    */
084   protected void normalize() {
085   }
086
087   private int leadersNumber() {
088     return (flyingPlateNumber % 2 == 0) ? flyingPlateNumber :
flyingPlateNumber + 1;
089   }
090
091   /**
092    * @unimod.action.descr Turn right
093    */
```

```
094   public void z2(StateMachineContext context) {
095     setControlledPlate();
096     controlledPlate.setA(this, controlledPlate.getA() - 12.5);
097   }
098 
099   /**
100    * @unimod.action.descr Turn left
101    */
102   public void z3(StateMachineContext context) {
103     setControlledPlate();
104     controlledPlate.setA(this, controlledPlate.getA() + 12.5);
105   }
106 
107   /**
108    * @unimod.action.descr Fly horizontally
109    */
110   public void z10(StateMachineContext context) {
111     setControlledPlate();
112     controlledPlate.setA(this, controlledPlate.getA() +
113         Math.atan2(controlledPlate.getSpeed().y, controlledPlate.
getSpeed().x) * 180 / Math.PI);
114 //    controlledPlate.setA(this, Math.atan2(controlledPlate.getSpeed()
.y, controlledPlate.getSpeed().x) * 180 / Math.PI);
115   }
116 
117   /**
118    * @unimod.action.descr Time passed from the beginning of the game
119    */
120   public int x1(StateMachineContext context) {
121     /*TODO: automatically generated by UniMod method*/
122     return -1;
123   }
124 
125   /**
126    * @unimod.action.descr Full speed
127    */
128   public void z11(StateMachineContext context) {
129     setControlledPlate();
130     controlledPlate.setQ(this, 1);
131   }
132 
133   /**
134    * @unimod.action.descr Plate is the leader in the group
135    */
136   public boolean x21(StateMachineContext context) {
137     return (flyingPlateNumber % 2) == 0;
138   }
139 
140   /**
141    * @unimod.action.descr Plate must join the group
142    */
143   public boolean x22(StateMachineContext context) {
144     return false;
145 //    if (flyingPlateNumber == 0) return false;
146 //    if (flyingPlateNumber == Config.PLATES_COUNT - 1) return false;
147 //    return true;
148   }
149 
150   /**
151    * @unimod.action.descr Increase fuel consumption by 0.2
152    */
153   public void z12(StateMachineContext context) {
154     setControlledPlate();
```

```
155       controlledPlate.setQ(this, controlledPlate.getQ() + 0.2);
156     }
157
158     /**
159      * @unimod.action.descr Decrease fuel consumption by 0.2
160      */
161     public void z13(StateMachineContext context) {
162       setControlledPlate();
163       controlledPlate.setQ(this, controlledPlate.getQ() - 0.2);
164     }
165
166     /**
167      * @unimod.action.descr Set speed for dodging
168      */
169     public void z14(StateMachineContext context) {
170       setControlledPlate();
171       controlledPlate.setQ(this, 1.0);
172     }
173
174     //((int) Y * 1000))
175     /**
176      * @unimod.action.descr Leader's Y-coordinate
177      */
178     public int x31(StateMachineContext context) {
179       /*TODO: automatically generated by UniMod method*/
180       return -1;
181     }
182
183     /**
184      * @unimod.action.descr Plate's Y-coordinate
185      */
186     public int x32(StateMachineContext context) {
187       setControlledPlate();
188       return (int) (controlledPlate.getPosition().y * 1000);
189     }
190
191     /**
192      * @unimod.action.descr Leader's X-coordinate
193      */
194     public int x33(StateMachineContext context) {
195       /*TODO: automatically generated by UniMod method*/
196       return -1;
197     }
198
199     /**
200      * @unimod.action.descr Plate's X-coordinate
201      */
202     public int x34(StateMachineContext context) {
203       setControlledPlate();
204       return (int) (controlledPlate.getPosition().x * 1000);
205     }
206
207     /**
208      * @unimod.action.descr Relative positions of plate's in group are n
ormal
209      */
210     public boolean x35(StateMachineContext context) {
211       return true;
212     }
213
214     /**
215      * @unimod.action.descr plate must overtake leader in group
216      */
```

```
217   public boolean x36(StateMachineContext context) {
218     /*TODO: automatically generated by UniMod method*/
219     return false;
220   }
221
222   /**
223    * @unimod.action.descr Explore environment around us
224    */
225   public void z101(StateMachineContext context) {
226     setControlledPlate();
227
228     // Îáðàáàòûâàåì ïðèáëèæåíèÿ ê ñòåíàì
229     if ((Math.abs(controlledPlate.getPosition().y - 0) < 2)) {
230       // Too close to left border
231       sendEventToPlate(FlyingPlate.LEFT_BORDER_IS_NEAR);
232     } else if ((Math.abs(controlledPlate.getPosition().y -
 Config.getFieldHeight()) < 2)) {
233       // Too close to right border
234       sendEventToPlate(FlyingPlate.RIGHT_BORDER_IS_NEAR);
235     } else {
236       sendEventToPlate(FlyingPlate.FLY HORIZONTALLY);
237     }
238
239     // Checking if there is a plate near us
240     for (Plate otherPlate : Plates.getPlates()) {
241
242       if (controlledPlate == otherPlate) continue;
243       if (otherPlate.isCrashed())
244         continue;
245       if (otherPlate.isLanded())
246         continue;
247
248       if (!canHit(otherPlate))
249         continue;
250
251       if (controlledPlate.getPosition().subtract(otherPlate.getPositio
n()).getLength() <= 10) {
252         if (((controlledPlate.getPosition().y -
 otherPlate.getPosition().y) > 0) &&
253             ((controlledPlate.getPosition().y -
 otherPlate.getPosition().y) <= 5) &&
254             (Math.abs(controlledPlate.getPosition().x -
 otherPlate.getPosition().x) < 2)) {
255           // Somebody on the left
256           sendEventToPlate(FlyingPlate.OTHER_PLATE_ON_THE_LEFT);
257         }
258         if (((controlledPlate.getPosition().y -
 otherPlate.getPosition().y) < 0) &&
259             ((controlledPlate.getPosition().y -
 otherPlate.getPosition().y) >= -5) &&
260             (Math.abs(controlledPlate.getPosition().x -
 otherPlate.getPosition().x) < 2)) {
261           // Somebody on the right
262           sendEventToPlate(FlyingPlate.OTHER_PLATE_ON_THE_RIGHT);
263         }
264
265         if ((Math.abs(controlledPlate.getPosition().y -
 otherPlate.getPosition().y) < 3) &&
266             ((controlledPlate.getPosition().x -
 otherPlate.getPosition().x) >= -5) &&
267             ((controlledPlate.getPosition().x -
 otherPlate.getPosition().x) < 0)) {
268           // Somebody in front of us
```

```
269           sendEventToPlate(FlyingPlate.OTHER_PLATE_IN_FRONT);
270         }
271
272         if ((Math.abs(controlledPlate.getPosition().y -
 otherPlate.getPosition().y) < 3) &&
273             ((controlledPlate.getPosition().x -
 otherPlate.getPosition().x) <= 5) &&
274             ((controlledPlate.getPosition().x -
 otherPlate.getPosition().x) > 0)) {
275           // Somebody behind us
276           sendEventToPlate(FlyingPlate.OTHER_PLATE_BEHIND);
277         }
278
279       }
280     }
281   }
282
283   /**
284    * Calculates if the plate can hit the <code>other</code>.
285    * @param other other flying plate.
286    * @return true if the hit can happen, false - otherwise.
287    */
288   private boolean canHit(Plate other) {
289     setControlledPlate();
290     Plate thisPlate = controlledPlate;
291
292     Vector v1 = thisPlate.getSpeed();
293     Vector pos1 = thisPlate.getPosition();
294     Vector v2 = other.getSpeed();
295     Vector pos2 = other.getPosition();
296
297     Vector deltaV = v1.subtract(v2);
298     Vector deltaPos = pos1.subtract(pos2);
299
300     double a = deltaV.getLength() * deltaV.getLength();
301     double b = 2 * deltaV.multiply(deltaPos);
302     double c = deltaPos.getLength() * deltaPos.getLength() -
 4 * Config.getPlateDiameter() * Config.getPlateDiameter();
303     double d = b * b - 4 * a * c;
304
305     if (d < 0)
306       return false;
307     if (Math.abs(a) <= 1e-9)
308       return false;
309
310     double t1 = (-b + Math.sqrt(d)) / (2 * a);
311     double t2 = (-b - Math.sqrt(d)) / (2 * a);
312     System.out.println("t1 = " + t1 + "; t2 = " + t2 + " " + other.get
Id()  + " " + thisPlate.getId());
313     return (t1 >= 0) || (t2 >= 0);
314   }
315
316   /**
317    * @unimod.action.descr Signum(Vy)
318    */
319   public int x2(StateMachineContext context) {
320     setControlledPlate();
321     if (controlledPlate.getSpeed().y > 0) return 1;
322     if (controlledPlate.getSpeed().y < 0) return -1;
323     return 0;
324   }
325
326   /**
```

```
327     * @unimod.action.descr (int) (Plate.getSpeed() * 1000)
328     */
329    public int x3(StateMachineContext context) {
330      /*TODO: automatically generated by UniMod method*/
331      return -1;
332    }
333
334 }
```

Radar.java

```
01 package ru.ifmo.flyingplates.managers.automatonmanager.flyingplate.ep;
02
03 import com.evelopers.common.exception.CommonException;
04 import com.evelopers.unimod.runtime.EventProvider;
05 import com.evelopers.unimod.runtime.ModelEngine;
06
07 public class Radar implements EventProvider {
08   /**
09    * @unimod.event.descr Left wall is near
10    */
11   public static final String E2 = "e2";
12
13   /**
14    * @unimod.event.descr Right wall is near
15    */
16   public static final String E3 = "e3";
17
18   /**
19    * @unimod.event.descr No wall is near
20    */
21   public static final String E10 = "e10";
22   /**
23    * @unimod.event.descr Other flying plate is on the left
24    */
25   public static final String E4 = "e4";
26   /**
27    * @unimod.event.descr Other flying plate is on the right
28    */
29   public static final String E5 = "e5";
30   /**
31    * @unimod.event.descr Other flying plate is in front of us
32    */
33   public static final String E6 = "e6";
34   /**
35    * @unimod.event.descr Other flying plate is behind us
36    */
37   public static final String E7 = "e7";
38
39   public void dispose() {
40     // TODO Auto-generated method stub
41
42   }
43
44   public void init(ModelEngine engine) throws CommonException {
45     // TODO Auto-generated method stub
46
47   }
48
49 }
```

CommunicationChannelListener.java

```
01 package ru.ifmo.flyingplates.managers.automatonmanager.flyingplate.ep;
02
03 import com.evelopers.common.exception.CommonException;
04 import com.evelopers.unimod.runtime.EventProvider;
05 import com.evelopers.unimod.runtime.ModelEngine;
06
07 public class CommunicationChannelListener implements EventProvider {
08
09   public void dispose() {
10     // TODO Auto-generated method stub
11
12   }
13
14   public void init(ModelEngine engine) throws CommonException {
15     // TODO Auto-generated method stub
16
17   }
18
19 }
```

Environment.java

```
01 package ru.ifmo.flyingplates.managers.automatonmanager.flyingplate.ep;
02
03 import com.evelopers.common.exception.CommonException;
04 import com.evelopers.unimod.runtime.EventProvider;
05 import com.evelopers.unimod.runtime.ModelEngine;
06
07 public class Environment implements EventProvider {
08   /**
09    * @unimod.event.descr Game started
10    */
11   public static final String E1 = "e1";
12
13   /**
14    * @unimod.event.descr One of plates in the group landed
15    */
16   public static final String E31 = "e31";
17   /**
18    * @unimod.event.descr Next tick happened
19    */
20   public static final String E101 = "e101";
21
22   /**
23    * @unimod.event.descr Plate crashed
24    */
25   public static final String E41 = "e41";
26
27   /**
28    * @unimod.event.descr Plate landed successfully
29    */
30   public static final String E42 = "e42";
31
32   /**
33    * @unimod.event.descr Radar destroyed
34    */
35   public static final String E43 = "e43";
36
37   public void dispose() {
38     // TODO Auto-generated method stub
```

```
39
40    }
41    public void init(ModelEngine engine) throws CommonException {
42      // TODO Auto-generated method stub
43
44    }
45
46  }
```

MessageProcessor.java

```
01  package ru.ifmo.flyingplates.managers.automatonmanager.flyingplate.ep;
02
03  import com.evelopers.common.exception.CommonException;
04  import com.evelopers.unimod.runtime.EventProvider;
05  import com.evelopers.unimod.runtime.ModelEngine;
06
07  public class MessageProcessor implements EventProvider {
08
09    /**
10     * @unimod.event.descr Message processed
11     */
12    public static final String E201 = "e201";
13    /**
14     * @unimod.event.descr Message received
15     */
16    public static final String E202 = "e202";
17
18    public void dispose() {
19      // TODO Auto-generated method stub
20
21    }
22
23    public void init(ModelEngine engine) throws CommonException {
24      // TODO Auto-generated method stub
25
26    }
27
28  }
```