МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

УДК 681.3.06: 62—507

«УТВЕРЖДАЮ»
Ректор СПбГУИТМО
докт. техн. наук, профессор
Васильев В.Н.

ПРОМЕЖУТОЧНЫЙ ОТЧЕТ

по научно-исследовательской теме № 1.5.05
«Разработка основных положений создания программных систем управления со сложным поведением на основе объектно-ориентированного и автоматного подходов»

Этап 6

«Разработка методов построения мультиагентных систем с использованием объектов и автоматов»

Руководитель НИР
докт. техн. наук,
профессор,
заведующий кафедрой
«Технологии программирования»
Шалыто А.А.

Санкт-Петербург
2008

Список основных исполнителей

Руководитель НИР доктор технических наук, профессор, заведующий кафедрой «Технологии программирования»	Шалыто А.А.
Ответственный исполнитель магистрант	Царев Ф.Н.
Исполнители	
Доцент	Корнеев Г.А.
Доцент	Шопырин Д.Г.
Ассистент	Казаков М.А.
Аспирант	Мазин М.А.
Аспирант	Вельдер В.Э.
Аспирант	Маврин П.Ю.
Магистрант	Паращенко Д.А.
Студент	Соколов Д.О.
Студент	Давыдов А.А.
Студент	Буздалов М.В.

Реферат

Отчет содержит 118 страниц, 48 рисунков, 39 источников литературы.

Автомат, граф переходов, состояние, автоматное программирование, проектирование программ, объектно-ориентированное программирование, мультиагентные системы

Целью настоящего этапа работы является разработка методов построения мультиагентных систем с использованием объектов и автоматов. При этом рассматриваются мультиагентные системы, построенные на основе реактивных агентов.

В первом разделе кратко описываются основные положения методики построения мультиагентных систем на основе объектов и автоматов.

Во втором разделе описывается пример разработки на основе предложенного подхода мультиагентной системы «Соревнование беспилотных летательных аппаратов».

В третьем разделе описывается пример разработки на основе предложенного подхода мультиагентной системы роботов Lego Mindstorms.

Кроме описанных методов и примеров, в отчете приведены ссылки на другие методы построения мультиагентных систем на основе объектов и автоматов, разработанные авторами. Эти методы подробно рассмотрены в рамках соответствующих проектов, опубликованных на сайте http://is.ifmo.ru в разделе «Проекты».

Содержание

Введение

В данном отчете излагаются методы построения мультиагентных систем с использованием объектов и автоматов.

В отчете рассмотрены следующие вопросы:

- приводится методика построения мультиагентных систем на основе объектов и автоматов;
- приводятся два примера применения описанной методики, как для случая использования инструментального средства для генерации кода, так и для случая написания программного кода вручную.

Применение указанных методов подробно рассмотрено в рамках соответствующих проектов, опубликованных на сайте http://is.ifmo.ru в разделе «Проекты». Кроме описанных методов, в отчете приведены ссылки на другие методы совместного использования автоматного и объектно-ориентированного стилей программирования, разработанные авторами.

1. Построение мультиагентной системы на основе объектов и автоматов

При построении мультиагентных систем все шире используются реактивные агенты [1]. В качестве наиболее распространенной математической модели для построения агентов этого класса применяются конечные автоматы. В работе [2] была предложена технология автоматного программирования для построения таких агентов. Однако для их реализации использовался процедурный подход.

Рефлексные агенты [3] исследовались психологами [4], которые использовали отображение "вход-выход". Такому отображению соответствует модель автомата без памяти – комбинационная схема.

В результате прогресса в психологии возникло понимание того, что необходимо учитывать внутренние состояния агента [5]. В большинстве работ по искусственному интеллекту модель рефлексных агентов с внутренними состояниями (реактивных агентов) рассматривалась как слишком простая для того, чтобы на ее основе можно было добиться значительных успехов, но исследования, описанные в работах [6, 7], «позволили поставить под сомнение это предположение» [3].

Использование такого подхода позволяет избежать создания сложной модели внешней среды – вместо нее предлагается использовать реактивную систему (систему, которая реагирует на внешние события), поведение которой описывается конечным автоматом.

Важно отметить, что в *NASA* используется аналогичный подход [8] при создании программного обеспечения, входящего в проект *Mars Science Laboratory*.

Цель данного этапа работы состоит в рассмотрении различных методов реализации автоматов в рамках объектно-ориентированного подхода к программированию реактивных агентов.

1.1. Методы объектно-ориентированной реализации конечных автоматов в мультиагентных системах

В работе [9] было предложено проектировать программы для систем логического управления на основе использования конечных автоматов. Такой способ построения программ был назван «Switch-технология» или «автоматное программирование».

В работе [10] этот подход был развит для проектирования программного обеспечения событийных систем. Он является процедурным, и поэтому был назван «процедурное программирование с явным выделением состояний».

В работе [11] рассматриваемый подход был расширен на объектно-ориентированные программы и получил название «объектно-ориентированное программирование с явным выделением состояний». При этом автоматы реализовывались в качестве методов классов.

В ходе педагогического эксперимента, описанного в работе [12], в настоящее время выполнено более пятидесяти проектов с применением объектно-ориентированного программирования с явным выделением состояний. При выполнении проектов был создан ряд методов реализации автоматов.

1.1.1. Прямые методы объектно-ориентированной реализации автоматов в мультиагентных системах

Выполним классификацию этих методов и кратко опишем их.

1. Автоматы, как методы классов [10]. Этот подход близок к процедурному стилю программирования и может быть назван «обертывание автоматов в классы».
2. Автоматы, как классы. Базовый класс, реализующий типовые функции автоматов, не применяется [11].
3. Автоматы, как классы с использованием базового класса. Этот подход основан на совместном применении преимуществ, как объектного, так и автоматного стилей программирования. При этом автоматы разрабатываются, как наследники класса, реализующего базовую функциональность. Базовый класс и другие классы образуют библиотеку, предоставляемую разработчику.

В работе [12] приводится простейшая библиотека классов для разработки программного обеспечения в рамках объектно-ориентированного программирования с явным выделением состояний.

При использовании этой библиотеки проектирование каждого автомата состоит в создании по словесному описанию (декларация о намерениях) схемы связей, описывающей его интерфейс, и графа переходов, определяющего его поведение. По этим двум документам формально и изоморфно может быть построен фрагмент программы, соответствующий автомату.

Применяя объектную парадигму, автоматы разрабатываются, как наследники базового класса Automaton. Этот класс реализует типовые функции автоматов (основные и вспомогательные). В наследниках определяются только функции, специфические для автоматов.

Перечислим основные функции автоматов, реализованные в базовом классе:

- организация выполнения действий в вершинах графа переходов (для автоматов Мура), на его дугах и

петлях (для автоматов Мили), а также в вершинах, на дугах и петлях (для автоматов Мура-Мили);

- организация взаимодействия автоматов:
- вызов автоматов с определенными событиями;
- реализация вложенных автоматов;
- обмен номерами состояний между автоматами.

Отметим, что если взаимодействие по вложенности возможно только «сверху вниз» в иерархии автоматов, то остальные два способа могут осуществляться в обе стороны, как «сверху вниз», так и «снизу вверх».

Из вспомогательных функций автоматов в классе Automaton реализована поддержка протоколирования. При этом возможно:

- автоматическое протоколирование:
- при начале работы автомата в определенном состоянии с определенным событием;
- при переходах из состояния в состояние;
- при завершении работы автомата в определенном состоянии;
- добавление описаний входных и выходных воздействий автомата.

В классах наследниках переопределяется ряд функций базового класса и добавляются входные воздействия (события и переменные), внутренние переменные, выходные воздействия, объекты управления, а также вложенные и вызываемые автоматы.

В работах [13, 14] предлагаемый подход иллюстрируется примером моделирования лифта (программа Lift), который приведен на сайте http://is.ifmo.ru в разделе «Проекты».

Эта программа является объектно-ориентированной. Такую программу удобно разрабатывать на персональном компь-

ютере и легко переносить на PC-подобные контроллеры. Однако, кроме таких контроллеров, в системах управления используются также микроконтроллеры, для которых отсутствуют компиляторы с объектно-ориентированных языков. Поэтому для микроконтроллеров применяется процедурное программирование.

В работе [15] предлагается методика преобразования ядра объектно-ориентированной программы с явным выделением состояний на языке C++ в процедурную программу с явным выделением состояний на языке C.

В данном случае под ядром программы понимается ее фрагмент, в котором отсутствует интерфейсная часть и не реализованы функции входных и внутренних переменных, а также выходных воздействий.

Методика иллюстрируется примером переноса ядра программы Lift на микроконтроллер Siemens SAB 80C515. При этом использовалась среда Keil µVision 2. Полученная в результате программа также размещена на сайте http://is.ifmo.ru в разделе «Проекты».

В работе [16] предложена библиотека STOOL (Switch-Technology Object Oriented Library), в которой не только автомат, но и его логические составные части имеют соответствующие базовые классы. Кроме того, библиотека предоставляет возможность разработки многопоточного программного обеспечения.

Особенность предлагаемого подхода состоит в том, что автоматы предлагается использовать не как методы классов, а как объекты, являющиеся потомками класса Auto. При этом автоматы-методы можно легко свести к автоматам-объектам, но не наоборот.

Класс State представляет собой состояние автомата, а класс Info – описание автомата. Этот класс требуется для организации автоматического протоколирования.

Каждый автомат при запуске делает не более одного перехода.

Рассматриваются два варианта реализации алгоритмов с применением автоматов:

- автомат реализуется внутри цикла типа while;
- автомат используется непосредственно (без применения цикла).

Автоматы первого типа удобны при реализации вычислительных алгоритмов, а второго – для реактивных агентов.

Перегруженные операторы operator int() и operator=(int) класса State позволяют пользоваться экземпляром класса State так, как будто он является целочисленной переменной.

Использование объекта вместо скалярной переменной позволяет вынести из оператора switch все функции, отличные от основных – функций переходов, входных переменных и выходных воздействий. Это обеспечивает также возможность выделения глобального состояния системы и одинаковым образом реализовывать действия и деятельности.

Предлагаемый подход в рамках объектно-ориентированного программирования обеспечивает сохранение в программах оператора switch, позволяющего целостно, формально и изоморфно реализовывать графы переходов автоматов.

В работе [17] предложена еще одна библиотека для объектно-ориентированной реализации автоматов, названная Auto-Lib, и приведен пример ее использования.

В работе [18] предложена библиотека, позволяющая «собирать» простые автоматы из наследников базовых классов

«состояние автомата» и «переход между состояниями». Эта библиотека обеспечивает изоморфизм между текстом программы и графом переходов даже при наличии в нем групповых переходов (переходов из гиперсостояний).

В целях устранения реентерабельности (повторного вызова главной функции автомата до завершения ее выполнения) в работе [19] предложен метод «отложенного вызова автоматов», состоящий в том, что один из методов базового класса обеспечивает постановку приходящих автомату событий в очередь и последовательную их обработку в отдельном потоке, сформированном для этого автомата. Таким образом, при применении данного метода число потоков равно количеству автоматов, в то время, как в работе [9] применялся только один поток.

1.1.2. Реализация автоматов на основе интерпретации

В работе [20] предложен подход, позволяющий автоматически преобразовывать графы переходов в текстовое описание в формате XML. На языке Java разработана среда исполнения полученного XML-описания.

Сначала указанное описание однократно и целиком преобразуется в соответствующее внутреннее объектное представление программы. В результате образуется система, состоящая из среды исполнения и объектного представления программы. При этом каждое входное и выходное воздействие реализуется вручную, в соответствии с его функциональностью. Упомянутая система при появлении события анализирует его и входные переменные и выполняет выходные воздействия, а также запускает вложенные автоматы.

В работах [21, 22] описан программный пакет UniMod (http://unimod.sourceforge.net), который представляет собой встраиваемый модуль для среды Eclipse, реализующий

подход, изложенный в предыдущем пункте. Он позволяет решать задачи автоматизации построения событийных объектно-ориентированных программ с явным выделением состояний. При этом для проектирования конечных автоматов совместно применяются Switch-технология и унифицированный язык моделирования UML. В этом случае схема связей автомата изображается с помощью диаграммы классов, а граф переходов – с помощью диаграммы Statechart. Этот пакет состоит из следующих частей:

- ядро, содержащее объектную метамодель конечного автомата, алгоритмы разбора и интерпретации булевых формул, проверки корректности конечного автомата и среду исполнения XML-описания модели конечного автомата;
- встраиваемый модуль для среды разработки диаграмм на языке UML, который помогает создавать схемы связей автомата, графы переходов в виде UML-диаграмм, а также выполняет генерацию XML-описаний.

Таким образом, UniMod = Switch-technology + UML + Eclipse.

В работе [23] предложено использовать язык XML для автоматного описания динамики изменения внешнего вида виртуального устройства – видеопроигрывателя Crystal Player (http://www.crystalplayer.com).

1.2. Технология разработки программного обеспечения мультиагентной системы на основе автоматного подхода

Для систем реактивных агентов, этот подход может быть сформулирован следующим образом [9].

1. Строится схема взаимодействия агентов.

2. Для каждого агента разрабатывается диаграмма классов.

3. Для каждого класса создается его структурная схема.

4. Если поведение класса описывается системой автоматов, то строится схема их взаимодействия. Автоматы могут взаимодействовать тремя способами:

- по вложенности;
- по вызываемости;
- за счет обмена номерами состояний.

5. Для каждого автомата создается четыре документа:

- словесное (вербальное) описание поведения автомата;
- схема связей автомата, задающая его интерфейс, в которой указываются символьные обозначения входных переменных и выходных воздействий и их полные названия, а также названия источников и приемников информации;
- граф переходов, в котором состояния имеют названия, а все остальные составляющие помечены символами. Это позволяет компактно и формально описывать даже весьма сложное поведение агентов;
- по графу переходов формально и изоморфно строится текст программы.

Отметим, что входные переменные могут быть функциями, в том числе весьма сложными.

6. Для каждого класса пишется программа, изоморфная его структурной схеме [11]. Этот этап может быть осущест-

влен как вручную, так и с использованием инструментальных средств таких, как, например, *UniMod*.

7. Отлаженные на модели управляющие программы загружаются в соответствующие физические агенты (при необходимости).

8. Выполняется отладка системы, которая упрощается за счет возможности наблюдения за состояниями каждого автомата [9].

9. Осуществляется разработка и выпуск открытой проектной документации [12].

2. Мультиагентная система «Беспилотные летательные аппараты»

В настоящем разделе описывается реализация мультиагентной системы «Беспилотные летательные аппараты» на основе автоматов.

2.1. Постановка задачи

Проводится соревнование между двумя командами беспилотных летательных аппаратов. Цель соревнований состоит в том, чтобы один из аппаратов команды переместился 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 летательных аппаратов (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, посвящен настоящий раздел.

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 соответствую-

щую управляющую программу. При написании этой программы могут быть использованы любые программные средства, а не только *UniMod*.

2.1.1. 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).

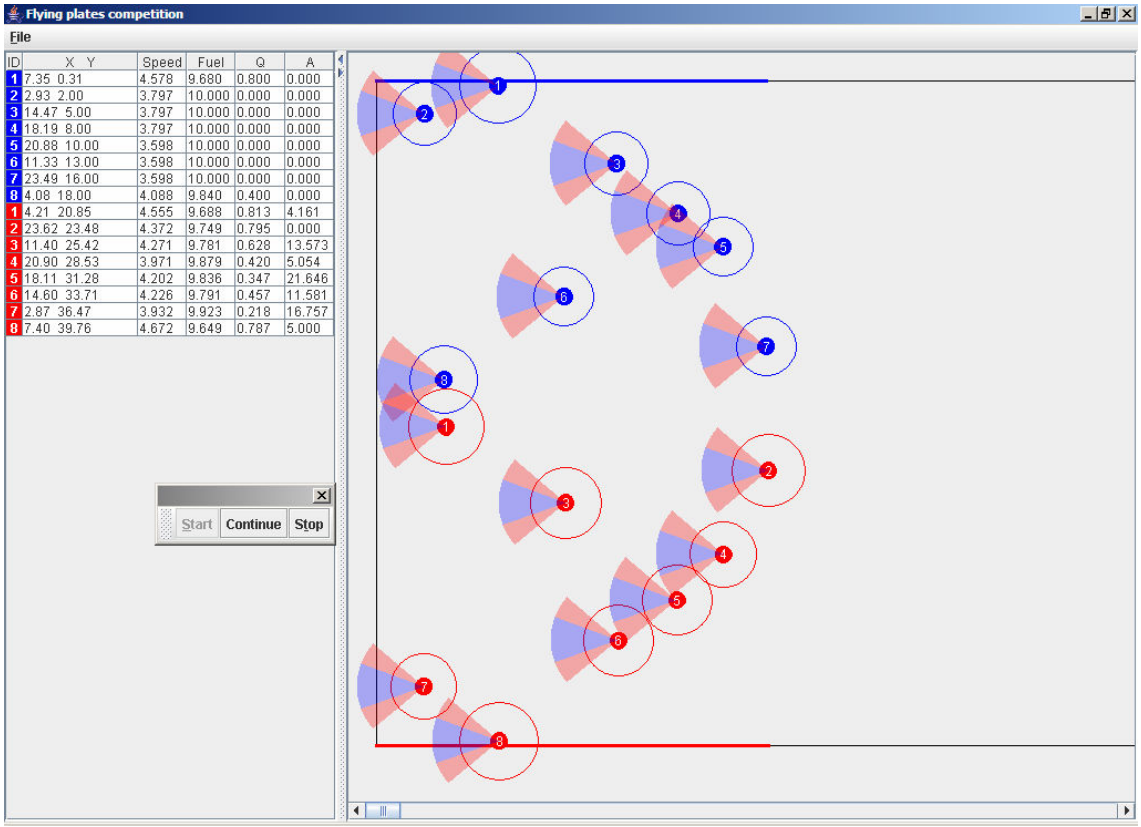

Рис. 1. Летающие тарелки на старте

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).

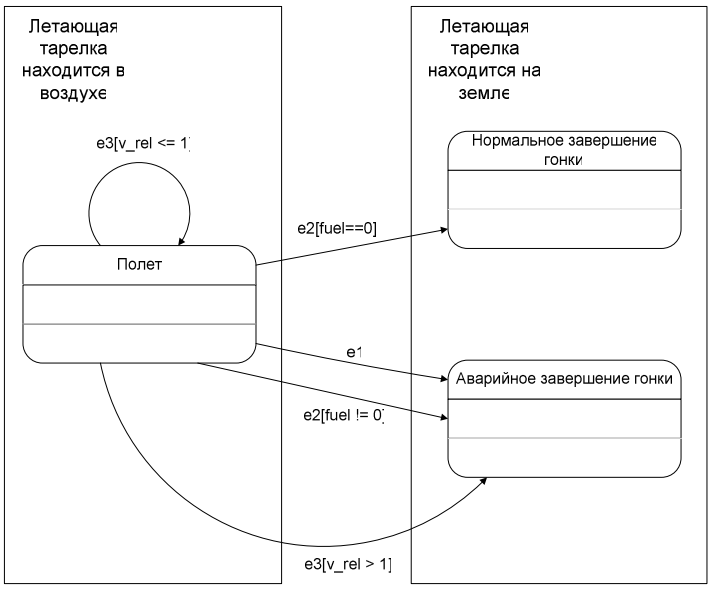

Рис. 2. Возможные состояния летающей тарелки и переходы между ними

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.

2.1.2. Динамика летающей тарелки

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

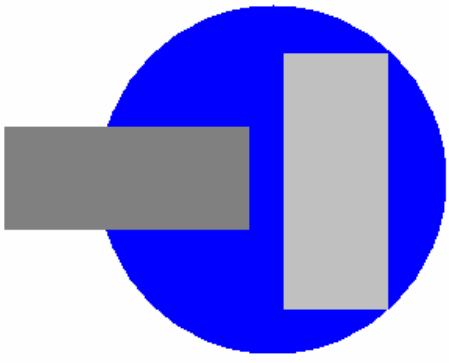

Рис. 3. Летающая тарелка

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°.

2.1.3. 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).

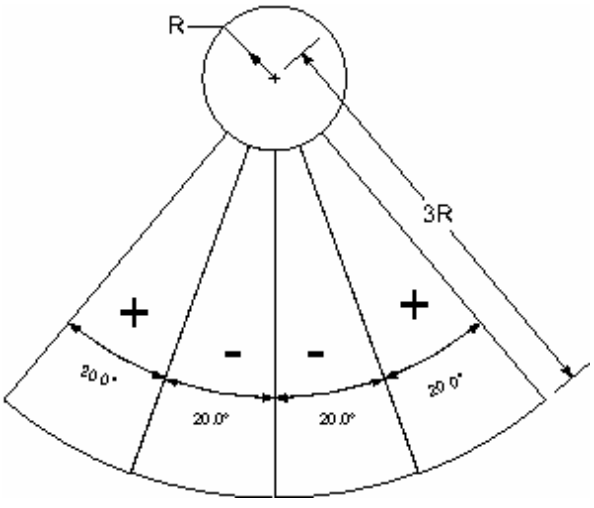

Рис. 4. Области аэродинамического взаимодействия

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.

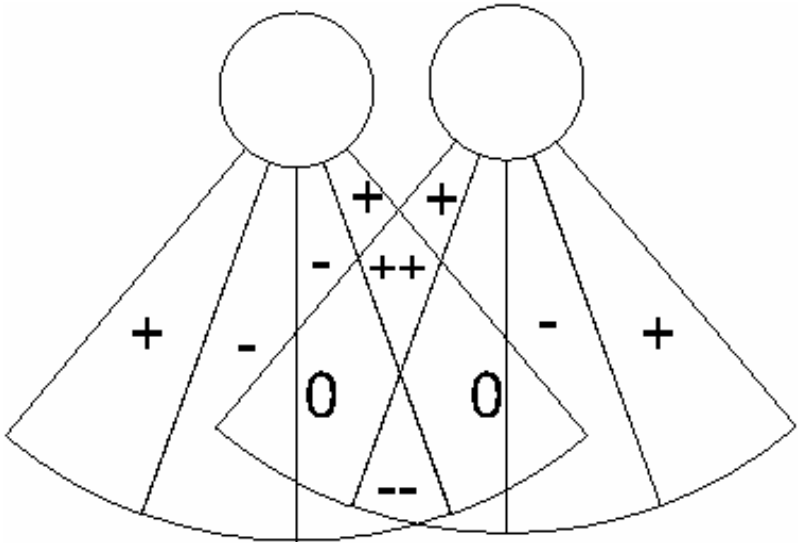

Рис. 5. Наложение областей аэродинамического взаимодействия от двух летающих тарелок

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.

2.1.3. 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.

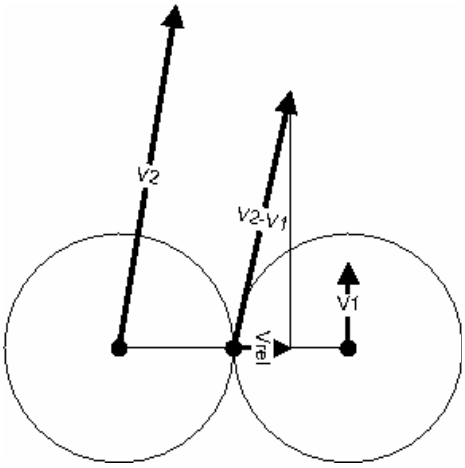

Рис. 6. Относительная скорость столкновения

2.1.4. 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_{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}.

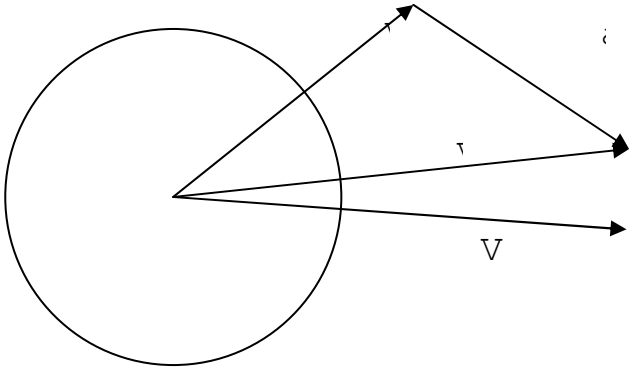

Рис. 7. Пересчет скорости летающей тарелки на шаге моделирования

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 со-

ответствии 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.

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

Для удобства визуализации оригинальные правила, предлагавшиеся на Всесибирской олимпиаде [26], 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.

2.1.6. 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.

2.2. Структура мультиагентной системы

В данном разделе приводится высокоуровневое описание программы.

2.2.1. Составные части мультиагентной системы

В описываемой программе можно выделить три основные сущности: интерфейс пользователя, игровой мир и система управления летающими тарелками. Схема взаимодействия между ними показана на рис. 8. 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.

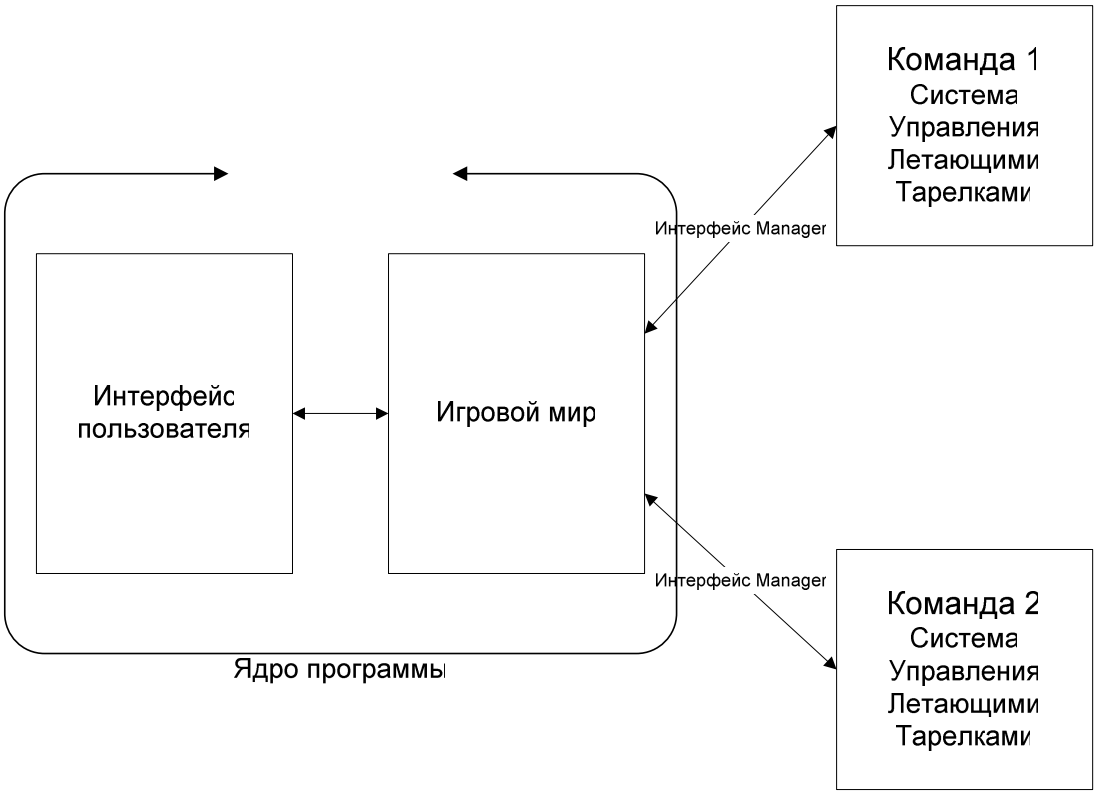

Рис. 8. Схема взаимодействия сущностей в программе

При этом сущности «Игровой мир» и «Интерфейс пользователя» неотделимы друг от друга и составляют ядро про-

граммы. 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. 2.1). 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.

2.2.2. 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, реализо-

ванной 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. 9.

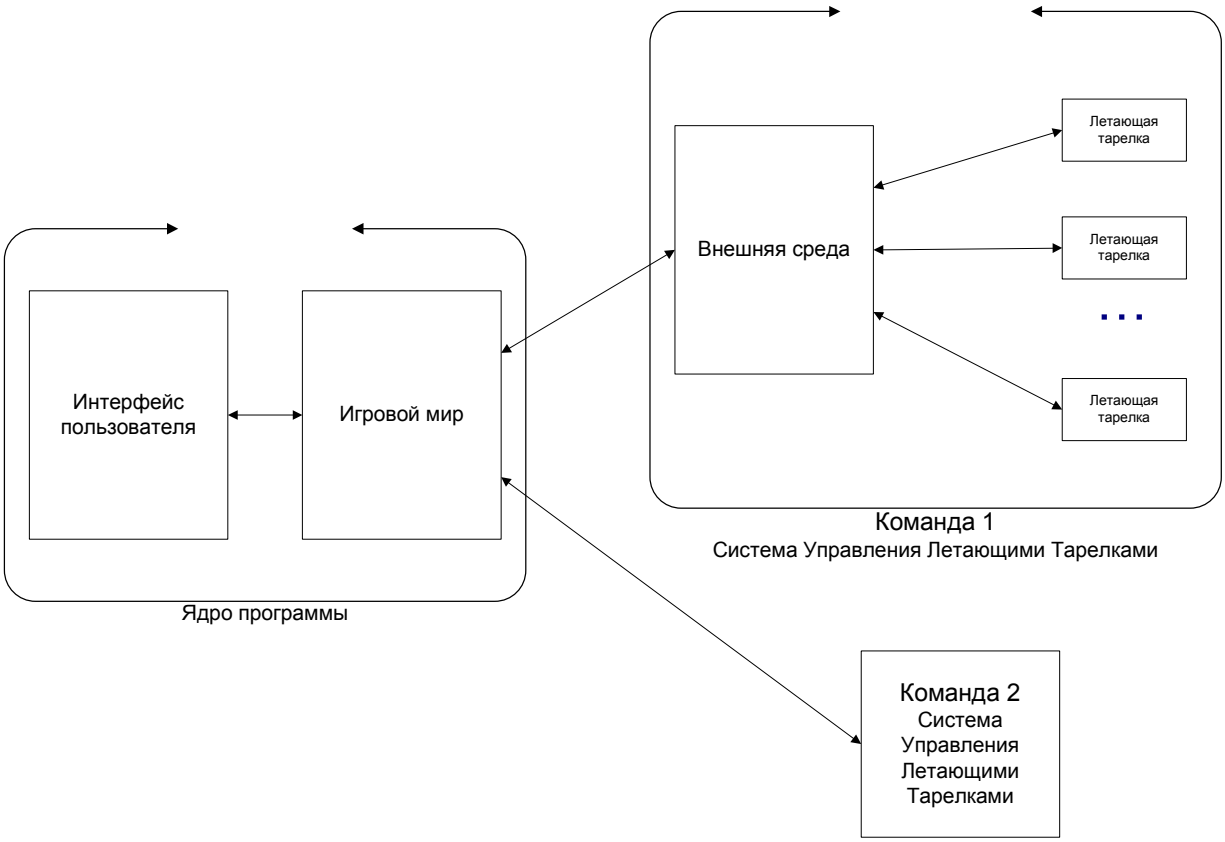

Рис. 9. Схема взаимодействия всех сущностей

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.

2.3. Ядро мультиагентной системы

В настоящем разделе описывется ядро мультиагентной системы «Беспилотные летательные аппараты», которое реализует внешнюю среду и обеспечивает взаимодействие между подключаемыми модулями-агентами.

2.3.1. 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.

2.3.2. Взаимодействие пользователя с программой

Взаимодействие пользователя с программой состоит в том, что пользователь выбирает две соревнующиеся команды летающих тарелок, отличающиеся системами управления (рис. 10); запускает, приостанавливает, завершает текущее соревнование; выбирает параметры визуализации процесса соревнования.

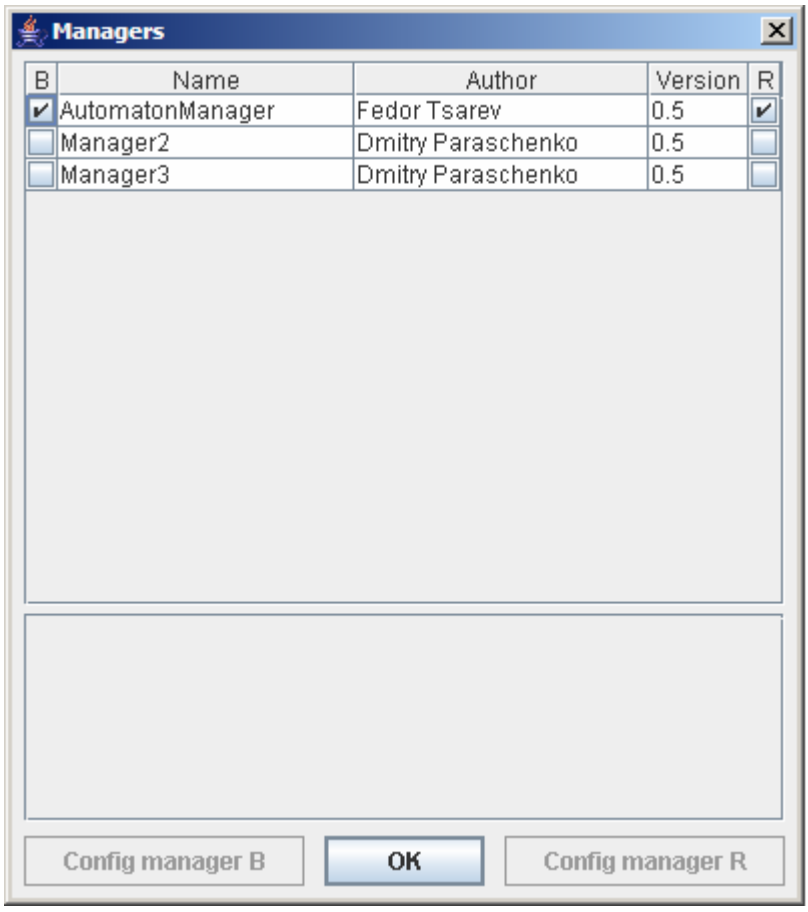

Рис. 10. Выбор соревнующихся систем управления беспилотными летательными аппаратами

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. 11.

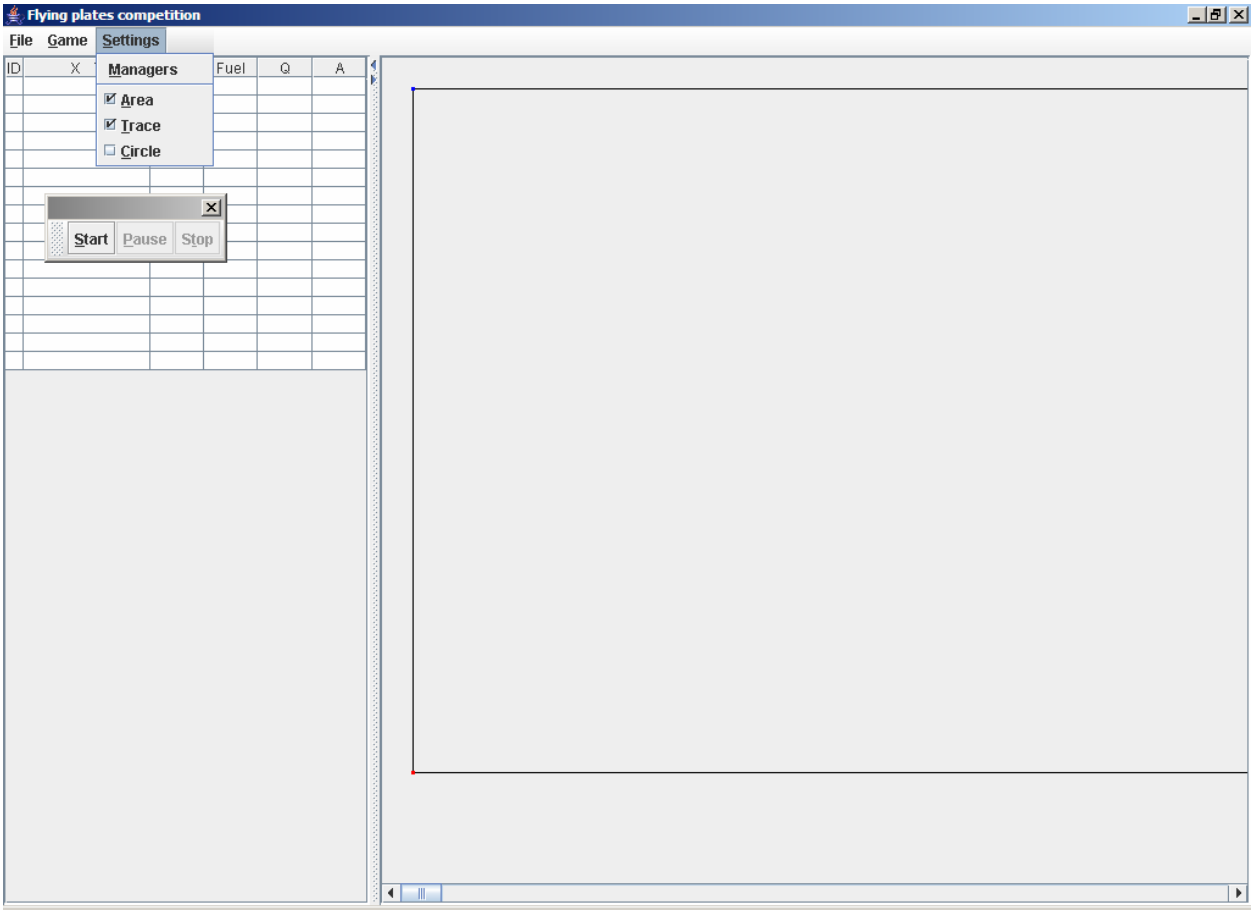

Рис. 11. Меню параметров визуализации и панель управления текущим соревнованием

2.3.3. Взаимодействие системы управления летающими тарелками с игровым миром

2.3.3.1. Проблема «нечестной игры»

Как упоминалось ранее, 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 про-

граммирования *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. 12).

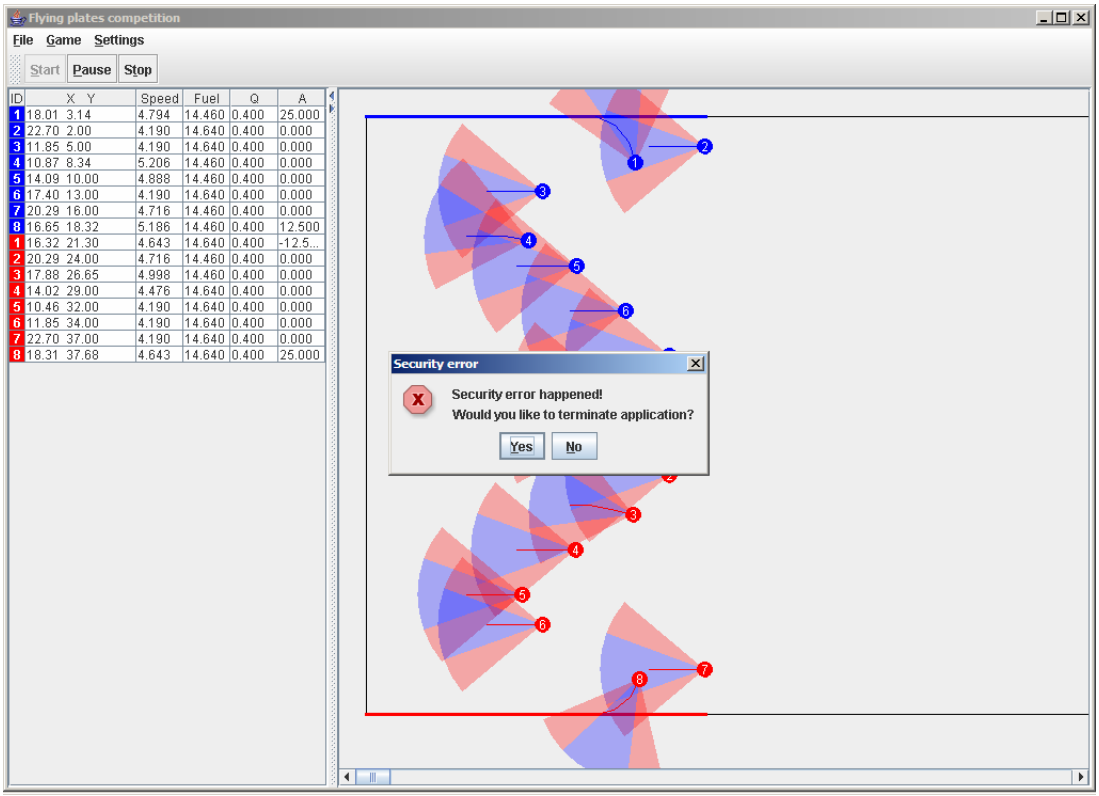

Рис. 12. Сообщение о том, что одна из команд ведет нечестную игру

2.3.4. Диаграмма связей

В отличие от *SWITCH*-технологии [10] 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. 13.

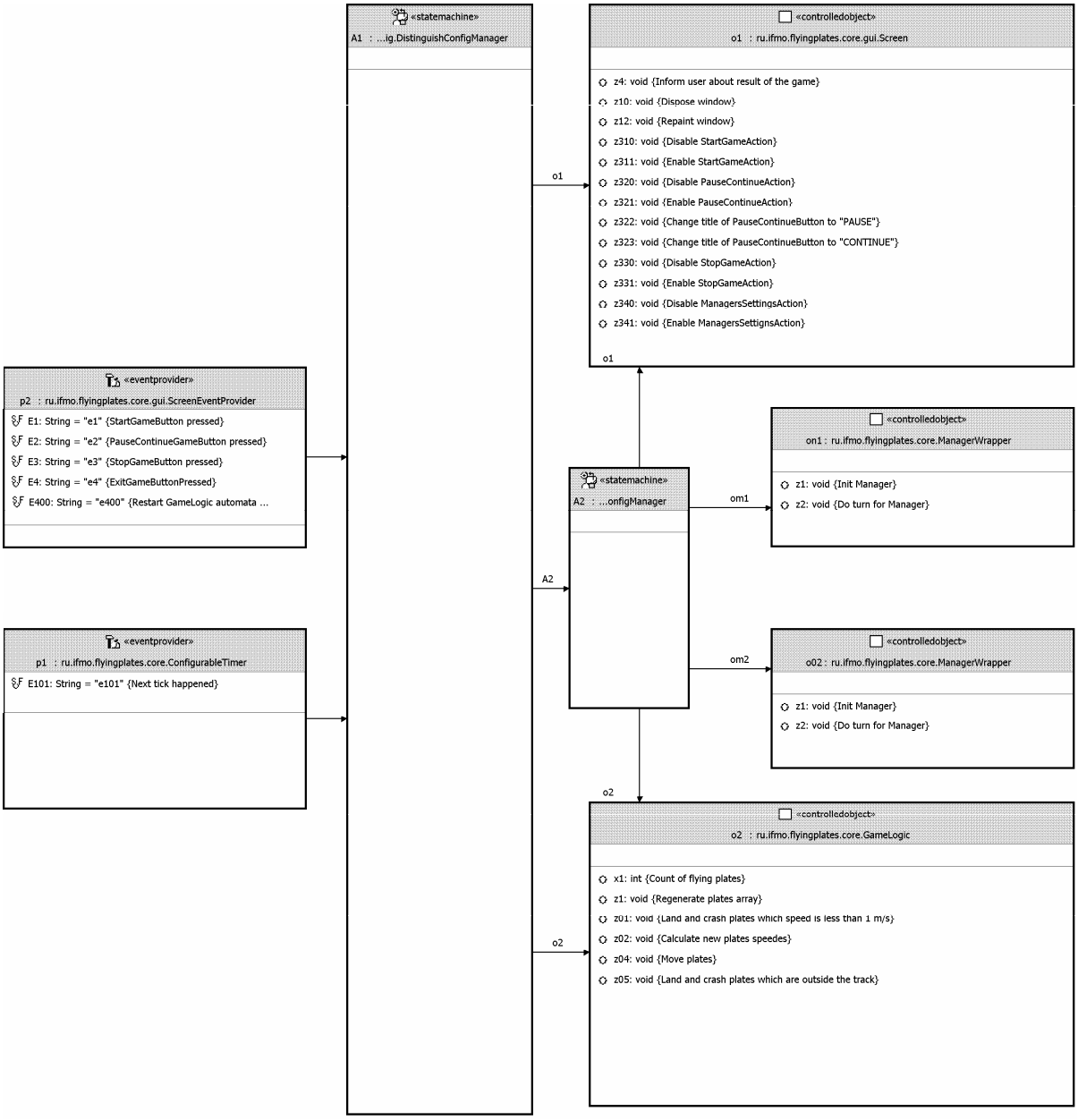

Рис. 13. Диаграмма связей

2.3.5. Поставщики событий

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

2.3.5.1. 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	

2.3.5.2. 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

Номер	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	

2.3.6. Автоматы

В этом разделе описаны автоматы.

2.3.6.1. Автомат A1 – управление интерфейсом пользователя

Автомат A1 с графом переходов, изображенным на рис. 14, управляет графическим интерфейсом пользователя (GUI).

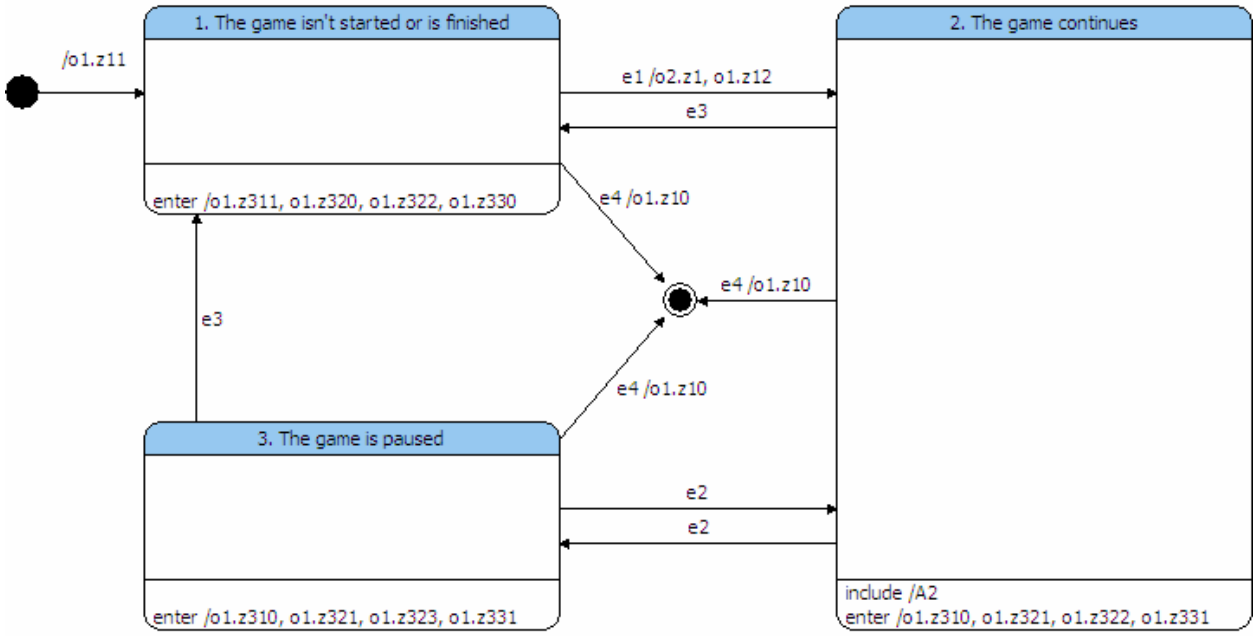

Рис. 14. Автомат A1

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. 14) 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.

2.3.6.2. Автомат A2 – управление главным циклом моделирования

Главный цикл моделирования [27] 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. 15) содержит состояния и переходы, реализующие эти этапы.

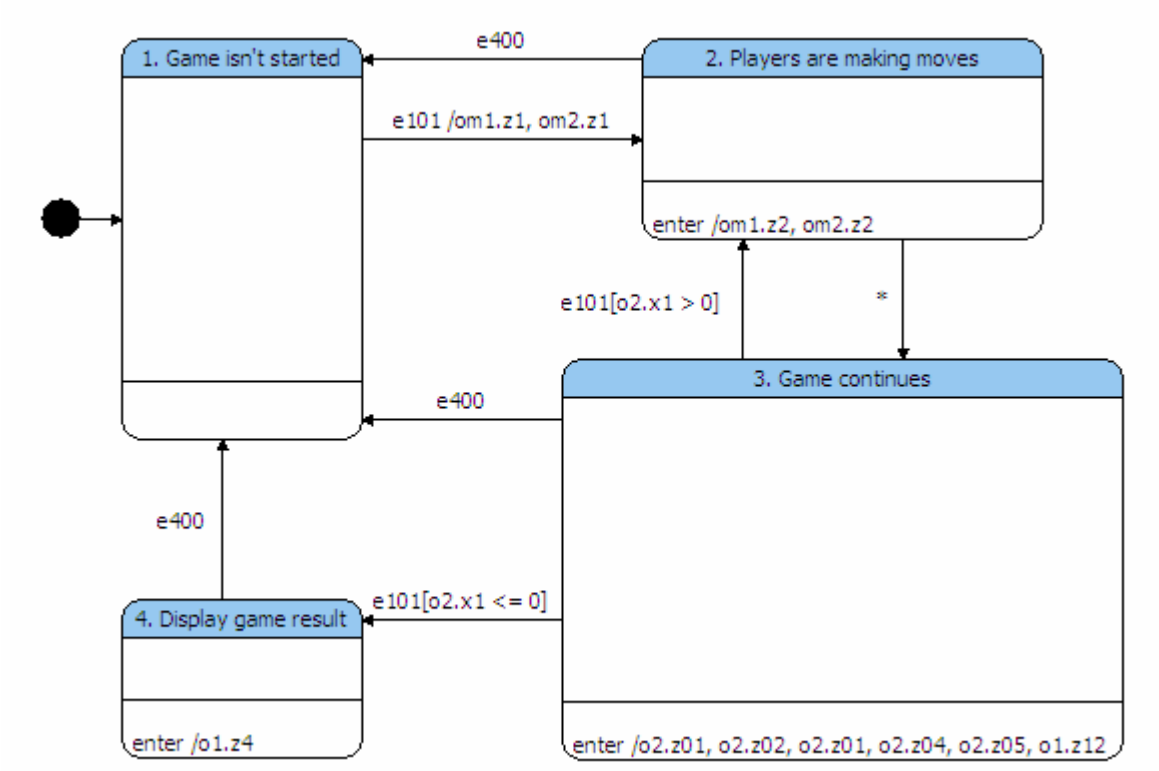

Рис. 15. Граф переходов автомата A2

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.

2.3.7. Объекты управления

В этом разделе описываются объекты управления.

2.3.7.1. Объект управления GameLogic

Объект управления GameLogic содержит выходные воздействия, связанные с реализацией игрового мира, подробно описанного в разд. 2.1. Табл. 4 содержит краткие описания выходных воздействия этого объекта управления, а табл. 5 – краткие описания входных воздействий.

Таблица 4. Выходные воздействия объекта управления GameLogic

Номер	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 скоро-	

		сти 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

Номер	Входное воздействие	Описание	Комментарий
1	x1	Число летающих тарелок, еще не завершивших полет	

2.3.7.2. 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

Номер	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	

2.3.7.3. 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

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

Отметим, что на диаграмме связей (рис. 13)g 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).

2.4. Описание агнета

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.

2.4.1. 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.

2.4.2. 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. 16. 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*.

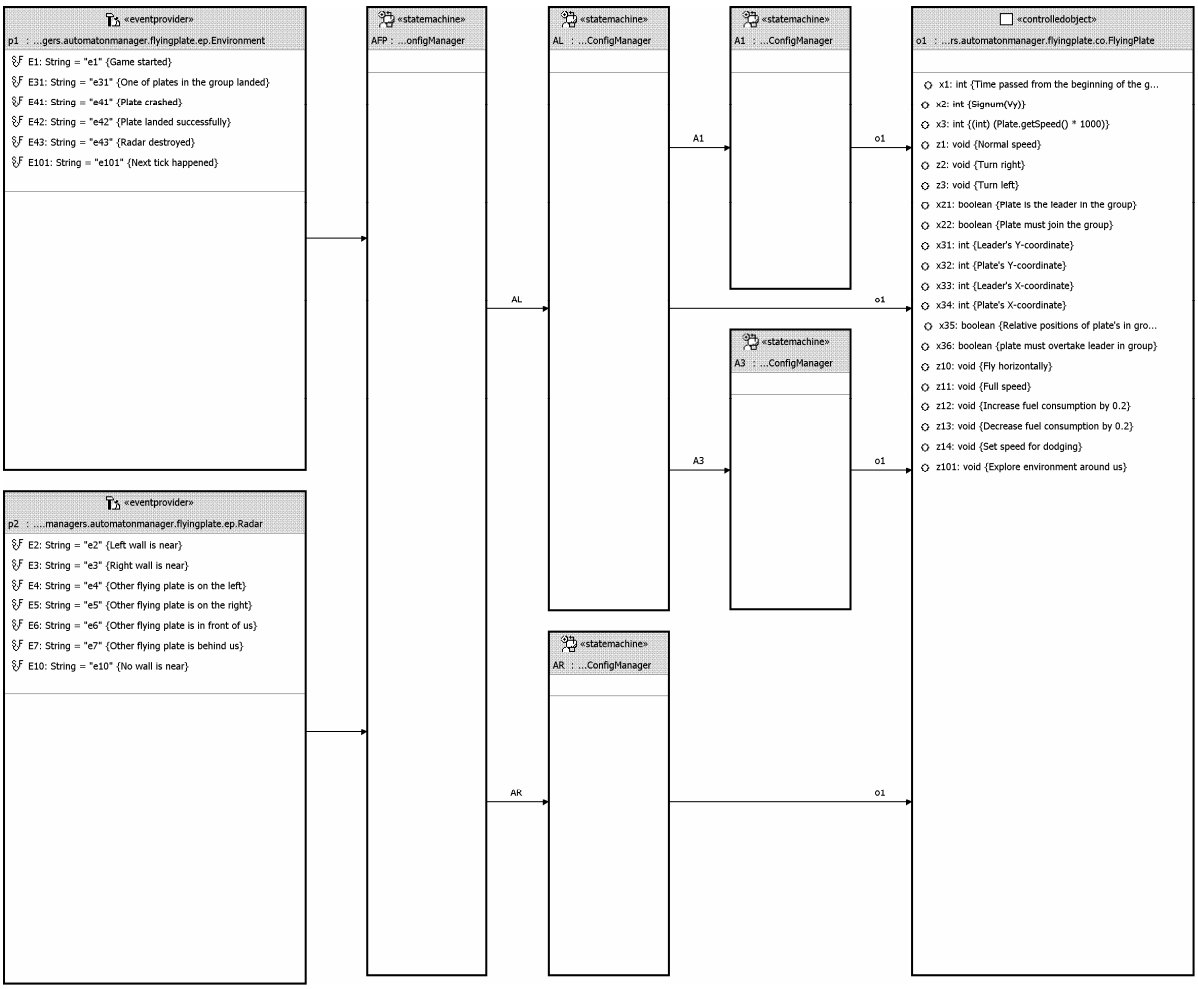

Рис. 16. Диаграмма связей летающей тарелки

2.4.2. Поставщики событий

В этом разделе описаны поставщики событий.

2.4.2.1. 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

2.4.2.2. 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	

2.4.3. Автоматы

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

2.4.3.1. Автомат AFP – *Состояние агента*

Этот автомат предназначен для управления летающей тарелкой на самом верхнем уровне. Его граф переходов показан на рис. 17.

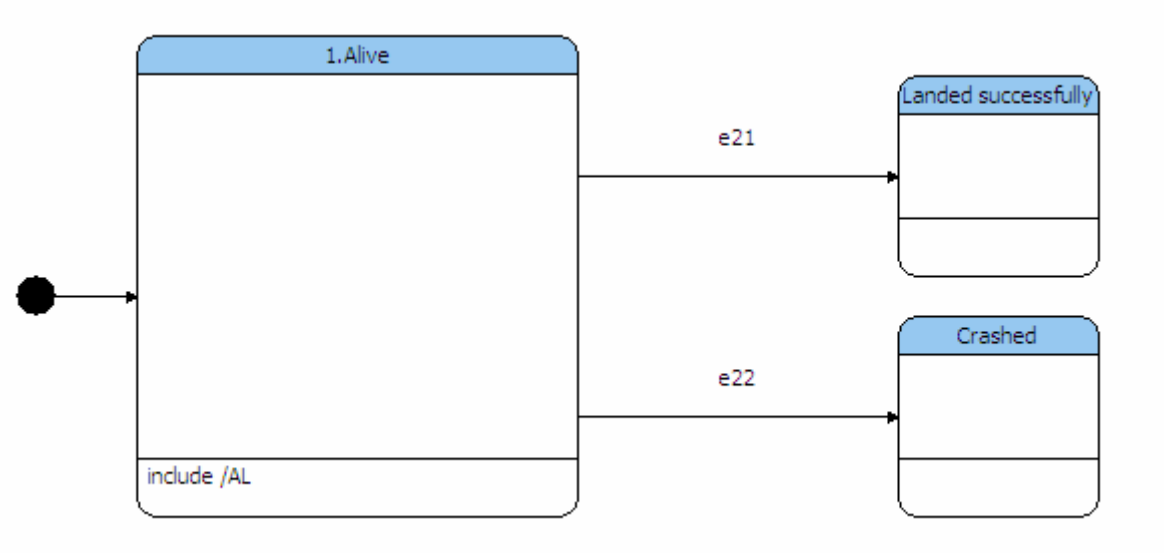

Рис. 17. Граф переходов автомата AFP

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*.

2.4.3.2. Автомат AL – *Режим полета*

Автомат AL «отвечает» за полет тарелки. Его граф переходов представлен на рис. 18.

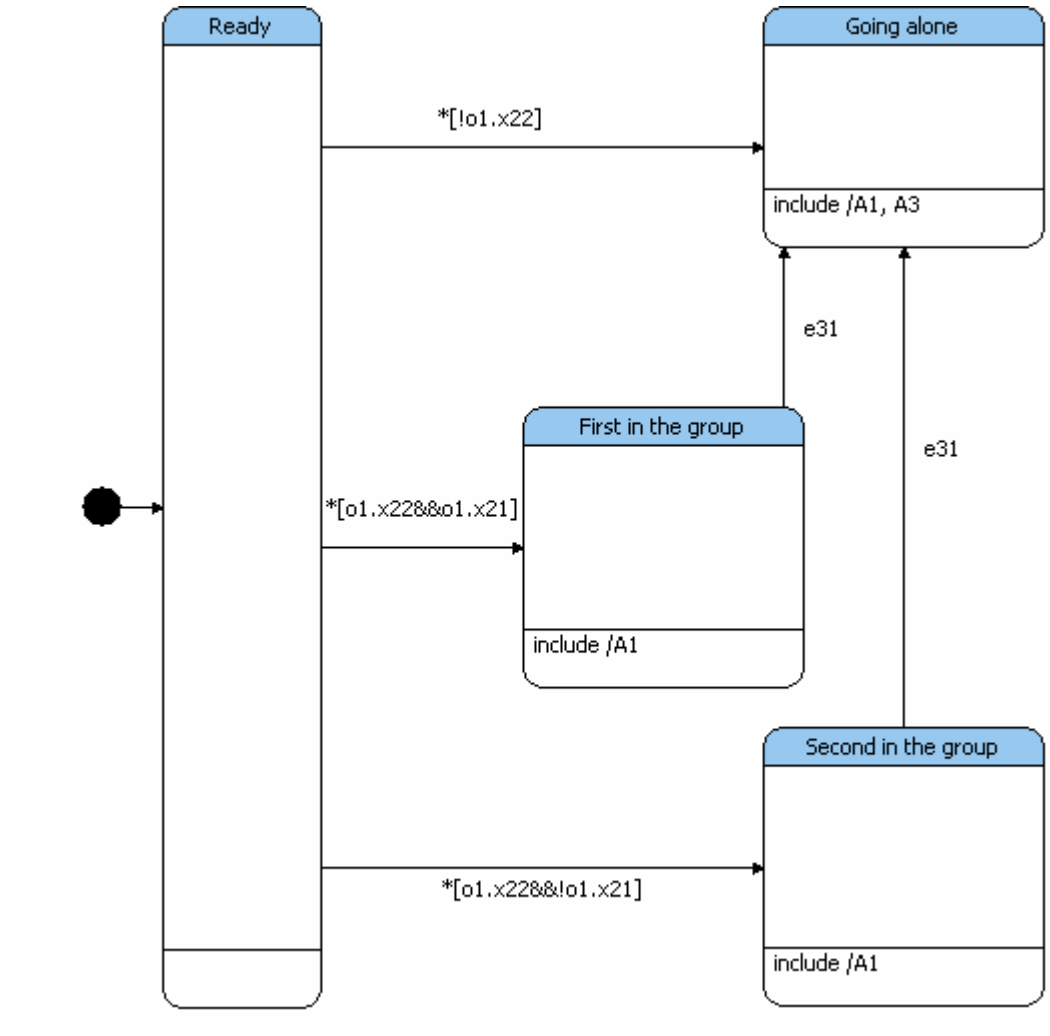

Рис. 18. Граф переходов автомата AL

Автомат AL имеет четыре 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.

2.4.3.3. 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. 19.

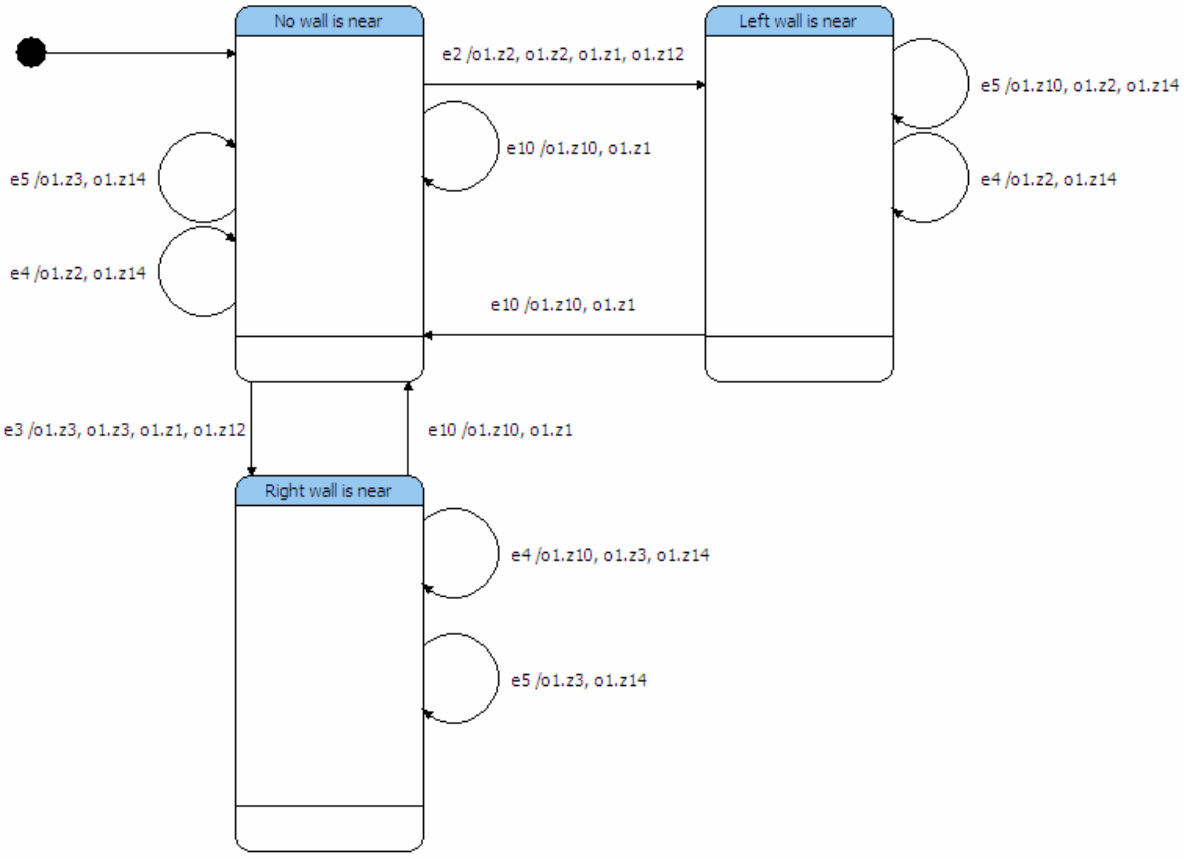

Рис. 19. Граф переходов автомата A1

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.

2.4.3.4. Автомат AR – *Радар*

Автомат AR, граф переходов которого показан на рис. 20, управляет радаром летающей тарелки.

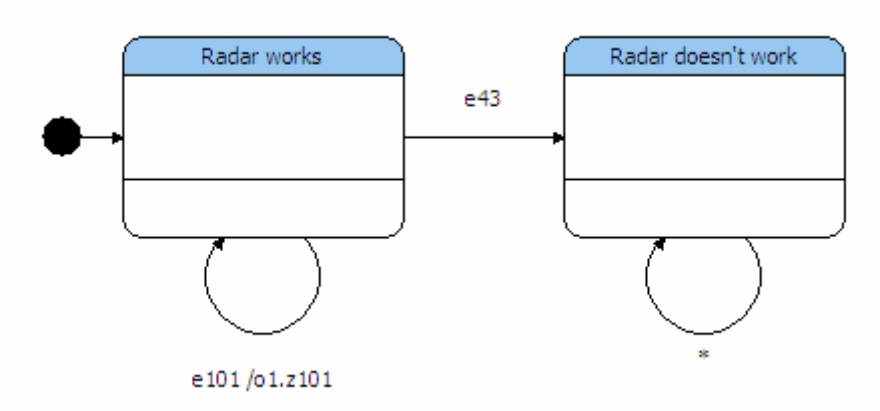

Рис. 20. Граф переходов автомата AR

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.

2.4.3.5. 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. 21.

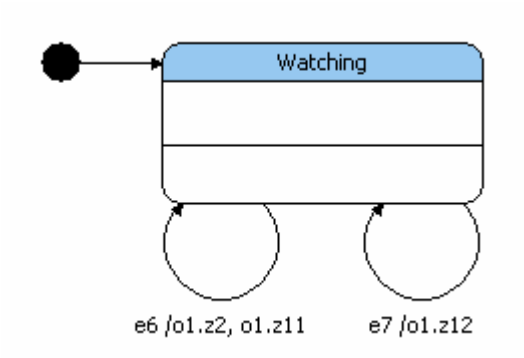

Рис. 21. Граф переходов автомата A3

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.

2.4.3.6. 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 Flying-Plate

Номер	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 Flying-Plate

Номер	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	

2.5. Статистика исходного кода и тестирование мультиагентной системы

2.5.1. Статистика исходного кода

Как отмечалось ранее, UML-диаграммы, создаваемые программистом, при использовании компиляционного подхода преобразуются в текст программы на языке *Java*. На рис. 22 показано соотношение объемов исходного кода, написанного вручную (в реализации летающей тарелки) и автоматически сгенерированного по *UniMod*-модели летающей тарелки.

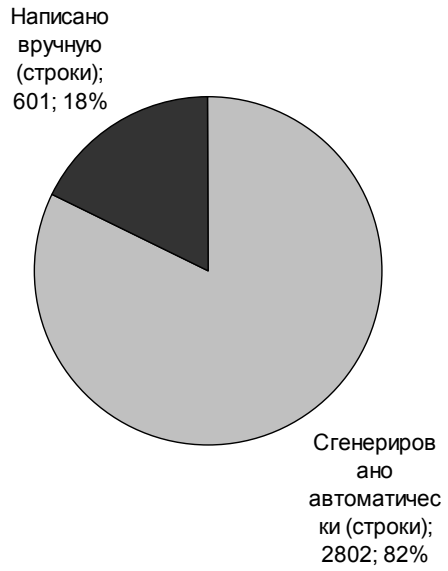

Рис. 22. Соотношение объемов исходного кода, сгенерированного автоматически и написанного вручную

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.

2.5.2. Тестирование мультиагентной системы

Тестирование стратегий проводилось при следующих значениях параметров агентов:

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

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

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

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

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

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

	$N_{ок}$	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: $N_{ОК}$ – 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

	$N_{ОК}$	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. 23, на котором показано распределение результатов «нашей» команды в соревнованиях против агрессивной стратегии.

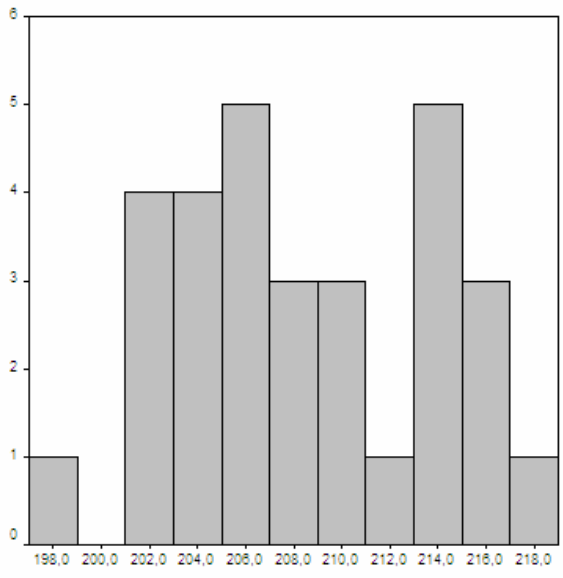

Рис. 23. Распределение результатов «нашей» команды в соревнованиях против агрессивной стратегии

Более детальная информация о результатах «нашей» команды может быть получена из рис. 24, на котором показано распределение количества агентов «нашей» команды, успешно закончивших соревнование против агрессивной стратегии.

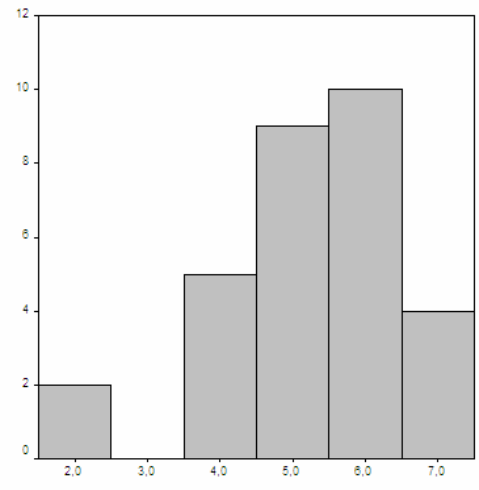

Рис. 24. Распределение количества агентов «нашей» команды, успешно завершивших соревнование против агрессивной стратегии

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).

В заключение главы отметим, что 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.

При этом отметим, что применение автоматного подхода позволяет существенно увеличить надежность программного обеспечения, так как более 80% программного кода удается генерировать автоматически по схеме связей и диаграммам переходов конечных автоматов.

3. Разработка мультиагентной системы роботов Lego Mindstorms

В настоящем разделе описывается мультиагентная система роботов Lego Mindstorms, которая состоит из транспортного робота и робота-поставщика предметов. Построение этой мультиагентной системы производится без использования инструментального средства *UniMod*, а путем реализации автоматных программ по диаграммам переходов автоматов вручную. Эта реализация осуществляется формально и изоморфно, что повышает качество программ по сравнению с традиционными методами.

3.1. Постановка задачи

Требуется создать систему, обеспечивающую доставку заданного количество предметов определенного размера из одного места в другое. Система должна состоять из двух роботов – транспортного робота и робота-поставщика предметов.

Доставка предметов ограничивается размерами небольшой комнаты или стола.

Транспортный робот должен ездить по определенному пути, который ведет от места доставки до места выдачи предметов и обратно. Под путем понимается черная линия на белом поле (бумаге). Характеристики пути описаны в разд. 2.

В начале транспортный робот находится в некотором месте – начале пути. Туда он и должен доставлять предметы. Место выдачи и место доставки предметов отмечены соответствующими кусками фольги.

Препятствий на пути нет.

Цвета черной линии и поля предполагаются равномерными. Характеристики “равномерности” изложены в разд. 2.

При считывании показаний светового датчика транспортный робот должен уметь различать путь, поле и фольгу.

Обмен информацией между транспортным роботом, роботом-поставщиком и пультом управления происходит посредством сообщений, передаваемых через инфракрасные порты.

Робот-поставщик стоит на месте выдачи предметов. Он может получать сообщения, передаваемые с помощью инфракрасного порта транспортного робота о выдаче необходимого количества предметов (одного, двух или трех).

Информация о количестве выдаваемых предметов передается транспортному роботу при помощи пульта с тремя функциональными кнопками, помеченными символами “1”, “2” и “3”. Каждая из кнопок соответствует определенному типу сообщения. Пульт нельзя перепрограммировать.

Проект должен быть выполнен при помощи конструктора *Lego Mindstorms*.

Конструирование механической части роботов должно быть выполнено с помощью свободно распространяемой программы *MLCAD*, позволяющей строить конфигурации роботов виртуально. Эта программа размещена по адресу http://www.lmsoftware.com/mlcad/.

3.2. Функциональные возможности комплекта *Lego Mindstorms*

Для реализации проекта использовались:

- два микроконтроллера;
- два световых датчика;
- два датчика касания;
- четыре двигателя;

- пульт управления;
- различные детали из стандартного набора *Lego Technics*.

Основной частью комплекта *Lego Mindstorms* является микроконтроллер, в который загружаются ОС и управляющая программа. Корпус микроконтроллера имеет три гнезда для подключения датчиков и три гнезда для подключения двигателей. Сверху на корпусе расположен небольшой дисплей. На него можно вывести до пяти цифр одновременно. Микроконтроллер основан на микропроцессоре *Renesas/H8300 Series* корпорации *Hitachi*.

Движение робота осуществляется с помощью двигателей, каждый из которых имеет четыре режима работы:

- вращение вперед;
- вращение назад;
- торможение (двигатель противодействует прокрутке);
- движение по инерции (двигатель свободно прокручивается).

В режимах вращения вперед и назад можно задать одну из семи мощностей работы.

Датчики могут находиться в одном из двух режимов:

- пассивном;
- активном.

При пассивном режиме, в отличие от активного, на датчики не подается напряжение.

Световой датчик может работать в обоих режимах. В пассивном режиме датчик получает информацию об интенсивности освещения, а в активном – излучает ярко-красный пучок света и измеряет интенсивность отраженного пучка.

Поле содержит элементы трех цветов – черного, белого и серебристого (фольга). Показания светового датчика лежат в диапазоне от 0 до 100. Интенсивность света, отраженного от черного цвета, составляет около 30-40 пунктов, от белого – 44-55 пунктов, а от фольги – 63-70 пунктов.

Датчик касания всегда работает в пассивном режиме.

На микроконтроллере находится инфракрасный порт (ИК-порт). С помощью этого порта производится загрузка управляющей программы в память микроконтроллера. Также с его помощью роботы могут обмениваться сообщениями друг с другом и принимать сообщения от дистанционного пульта управления.

Дистанционный пульт управления может посылать три типа сообщений, соответствующих кнопкам, помеченным символами “1”, “2” и “3”. При нажатой кнопке он постоянно посылает сообщения. Это следует учитывать при написании обработчика событий от ИК-порта.

3.3. Процесс разработки управляющей системы

В начале работы над проектом было собрано несколько простых роботов, и для каждого из них были написаны управляющие программы, используя рекомендации из книг [28, 29].

При этом программы писались на языке *Java* под ОС *leJOS*. В результате авторы убедились, что систему с достаточно сложным поведением с помощью указанных рекомендаций запрограммировать крайне трудно. При рекомендованном, по сути традиционном, программировании даже небольшие изменения по мере роста программы становилось вносить все сложнее и сложнее. В итоге, после нескольких дней работы по написанию кода, была создана малопонятная программа, работавшая нестабильно. При этом отладка была чрезвычайно

трудна, так как на дисплей используемого робота можно вывести только несколько цифр. Кроме того, загрузка программы в микроконтроллер занимала довольно много времени – три-четыре минуты. Поэтому каждое изменение программы было слишком длительным. Стало понятно, что дальше работать, используя традиционный подход, бесперспективно.

После этого использовали рекомендацию, предложенную разработчиками ОС *leJOS.* При применении операционной системы *leJOS* ее авторы предложили применять концепцию, основанную на представлении действий робота в виде нескольких объектов типа Behavior (поведение). Эти объекты не связаны друг с другом. Для того чтобы с ними работать, необходимо сначала загрузить их в объект типа Arbitrator (планировщик), а затем запустить Arbitrator. Он, в зависимости от приоритета объекта Behavior, а также определенного метода, возвращающего значение булевского типа и реализуемого для каждого такого объекта, выбирает тот объект Behavior, который должен работать в данный момент. Это похоже на то, как в современных операционных системах реализуется мультизадачность. Недостатком этого подхода является невозможность обмена информацией между объектами. Реализация рассматриваемой программы в рамках этого подхода прекратилась, когда понадобилось отличить кусок фольги, расположенный в одном конце пути, от куска фольги, расположенного в другом его конце. Это оказалось непосильной задачей для такого подхода. После этого было решено реализовать проект на основе автоматного подхода.

Сначала конечные автоматы использовались «приблизительно» – не так, как это сделано в окончательном варианте. Была создано некое описание алгоритма, с использованием графов переходов конечных автоматов. При этом, однако, не были четко прописаны условия перехода из одного

состояния в другое, а выходные воздействия описывались словами, как это любят делать многие. При реализации программы не выполнялся изоморфный переход от графов переходов к исходному тексту программы. Конечно, такой подход к разработке несколько ускорил само написание программы, но полностью от «скользких» мест избавиться не удалось. Программу по-прежнему было трудно отлаживать, количество ошибок уменьшилось, но не до конца. Эту реализацию все-таки удалось довести до «рабочего» состояния. Однако робот вел себя достаточно непредсказуемо при попадании в необычную ситуацию. Например, если транспортный робот резко сдвигался из-за неровности пути влево или вправо, то он благополучно «зависал», причем сказать что-либо об его поведении в этот момент было сложно.

Ввиду проблем с предыдущими тремя подходами, было решено использовать автоматный подход, основанный на SWITCH-технологии [9, 10].

Существует большое количество книг [28, 29] и проектов [30 – 32], посвященных программированию *Lego Mindstorms*. К сожалению, и у тех и у других есть недостатки: в книгах не приводится никакой технологии проектирования, а в проектах отсутствует проектная документация.

Стоит отметить интересную разработку скандинавских исследователей [33], в которой они использовали концепцию конечных автоматов для создания виртуальных моделей различных роботов, собранных из *Lego Mindstorms*. При помощи этих моделей можно было тестировать управляющие программы, не загружая их в микроконтроллер. Однако в этой работе концепция конечных автоматов не рассматривалась в качестве логической модели для самой управляющей системы. В работах [34 – 37] конечные автоматы также используются для построения разного рода виртуальных моделей различных

механизмов, однако проектная документация на них отсутствует.

3.4. Построение программы по схемам переходов автоматов

В настоящем разделе описывается алгоритм построения исходного кода программы по графу переходов автоматов. Это будет сделано для языка *Java*. По графу переходов, который приводится ниже, будет построен исходный код.

Отметим, что в **вершинах** графов переходов могут использоваться выходные воздействия, помеченные словами **in** и **out**. Выходные воздействия, помеченные **in**, выполняются сразу после **входа** в состояние, а воздействия с пометкой **out** – после **выхода** из состояния, но до того, как будут выполнены воздействия на переходе из этого состояния. В дальнейшем эти воздействия будут называться in-воздействиями и out-воздействиями. В языке *UML* [38] такие действия называются “действиями при входе в состояние” и “действиями при выходе из состояния” соответственно. Если при переходе состояние не изменилось, то ни in-воздействия, ни out-воздействия не выполняются. Таким образом, **переход** из одного состояния в другое сопровождается **сначала out-воздействиями** в **первом** состоянии, затем **воздействиями на переходе**, и уже **потом in-воздействиями** во **втором** состоянии.

Иногда переходы из состояния помечаются **приоритетами**. В этом случае **вначале** проверяются условия, соответствующие переходам с **меньшим номером** (большим приоритетом).

Блокировкам, выполненным за счет обмена номерами состояний, (например, y==4) всегда присваивается **наивысший** приоритет.

Иногда для устранения повторяющихся фрагментов кода используются ветвящиеся переходы. Они обозначаются при помощи кружка. Переход выполняется при условии, что выполняется как условие на стрелке, входящей в кружок, так и условие на стрелке, выходящей из кружка.

Помимо ветвящихся переходов также могут существовать **мультипереходы.** Они используются, если необходимо при **переходе** из одного состояние в другое совершать **различные выходные воздействия** при **различных** значениях **входных переменных.** Такие переходы обозначаются при помощи нескольких пометок на переходе между двумя состояниями.

Вложенные автоматы всегда реализуются до функций выходных воздействий.

На рис. 25, 26 изображены примеры графов переходов.

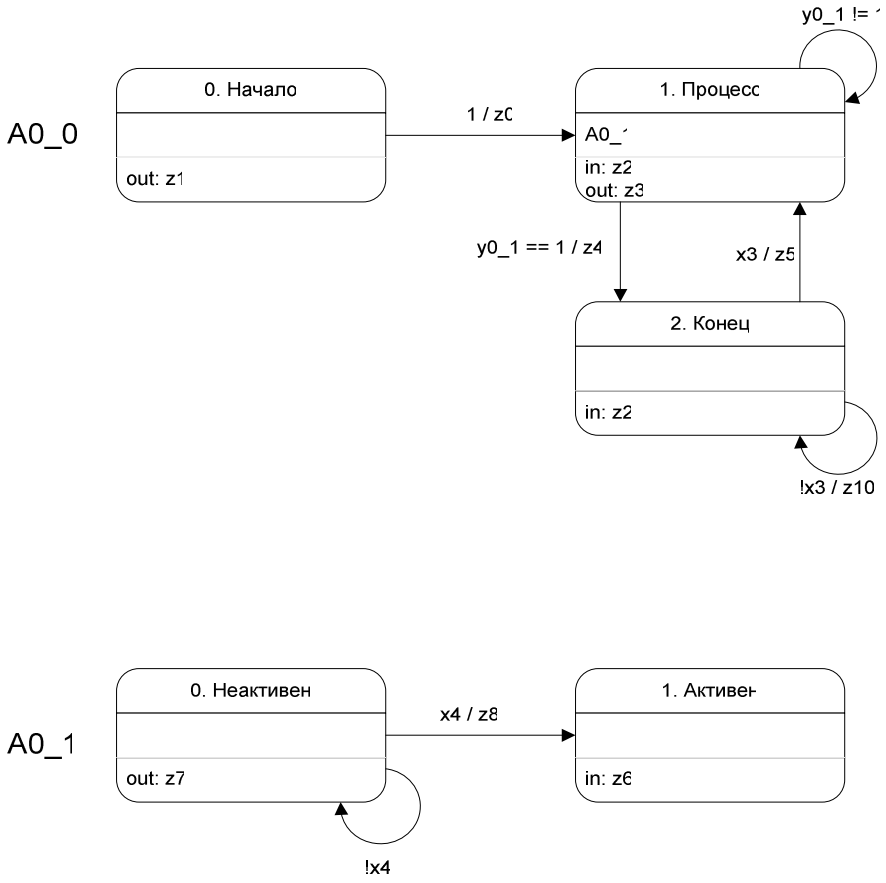

Рис. 25. Граф переходов с вложенным автоматом

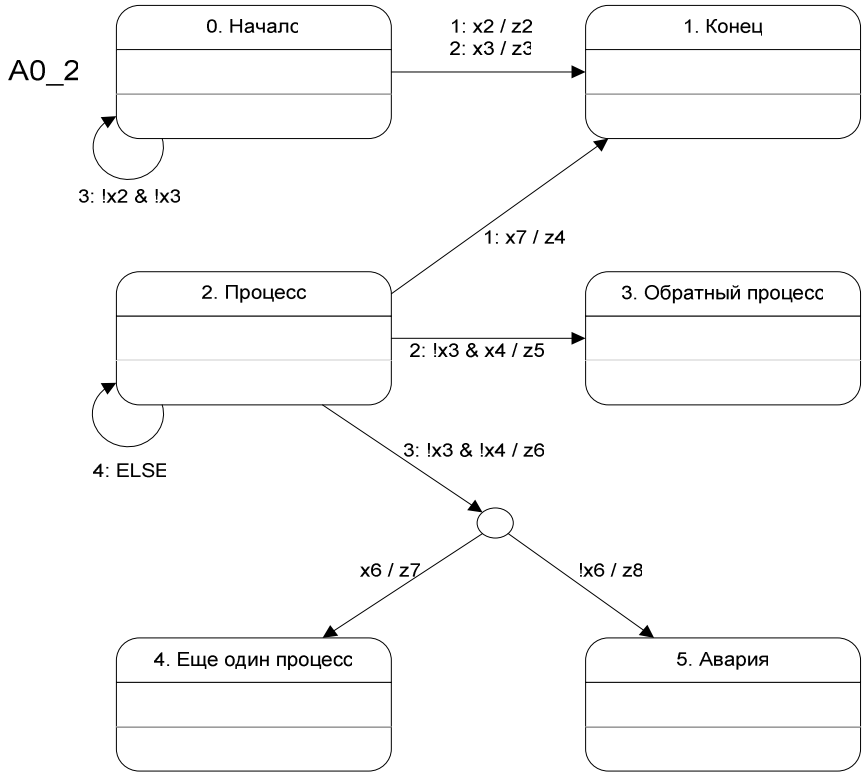

Рис. 26. Граф переходов с приоритетами, мультипереходами и ветвящимся переходом

3.5. Реализация автоматов на языке *Java*

Опишем процесс создания программы на языке *Java*. Одним из недостатков ОС *leJOS* является невозможность применения оператора `switch`. Поэтому вместо этого оператора будем использовать несколько идущих друг за другом конструкций `if…else`. Для экономии памяти все входные и выходные воздействия, а также автоматные функции реализованы в виде статических функций одного класса.

```
//Переменные и константы реализуются следующим образом
public static final int ES_POWER = 1;
public static final int VOLTAGE_TIME = 2000;
```

Функция булевского типа, соответствующая входной переменной с номером n, обозначаются как `xn`. Для случая `n==2` имеем следующий фрагмент кода:

```
public static boolean x2() {
     //Здесь может находиться произвольный код
}
```

Функция, соответствующая выходному воздействию с номером n, обозначается как zn. Для случая n=3 имеем следующий фрагмент кода:

```
private static void z3() {
     //Здесь может находиться произвольный код
}
```

Переменные, содержащие номер состояния автомата, являются статическими и обозначаются как yn. Здесь n – номер автомата. Для случая n=0_0 имеем следующий фрагмент кода:

```
public static int y0_0 = 0;
```

При этом первый ноль – номер робота, а второй – номер автомата в этом роботе.

Помимо переменных, содержащих номер состояния, заданы переменные булевского типа. Они обозначаются следующим образом: имя переменной, содержащей номер состояния автомата, а затем суффикс changed. Значение этой переменной равное true показывает, что состояние автомата изменилось, и следует выполнить in-воздействия в новом состоянии. Инициализация этих переменных производится значением false:

```
public static boolean y0_0changed = false;
```

Для удобства in- и out-воздействия автоматов вынесены в отдельные функции. Их названия образуются следующим образом – сначала префикс A, потом номер робота, потом символ подчеркивания, потом номер автомата в роботе, затем символ S (от слова “state”), номер состояния и, в зависимости от типа воздействия, суффикс IN или OUT.

```
//out-воздействия состояния 0 автомата A0_0 (рис. )
public static void A0_0S0OUT() {
     z1();
}


//in-воздействия состояния 1 автомата A0_0 (рис. )
public static void A0_0S1IN() {
```

```
    z2();
}
```

Теперь рассмотрим схему реализации автоматных функций на примере автомата A0_0 (рис. 25). Эта схема изображена на рис. 27.

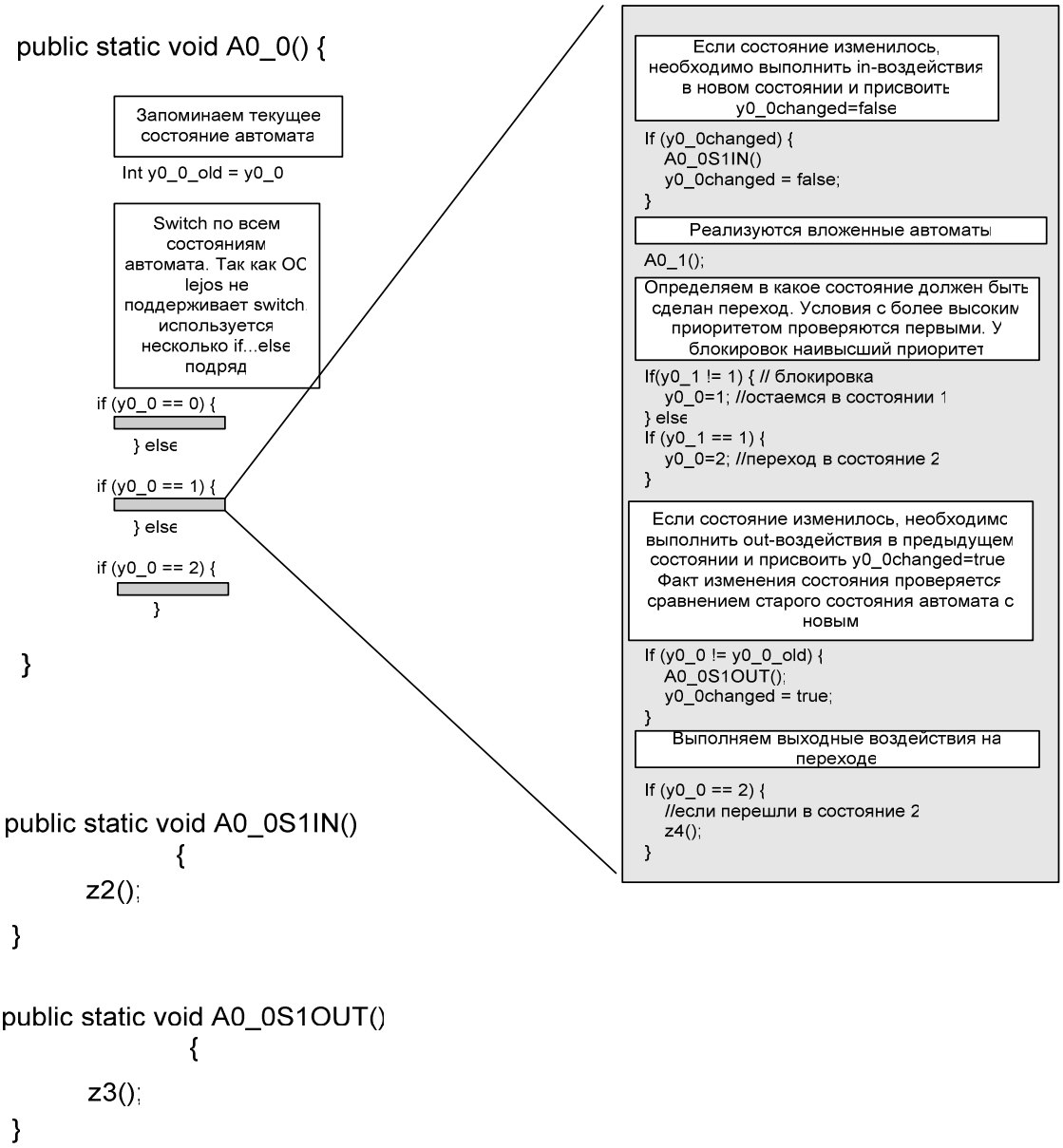

Рис. 27. Метод построения изоморфного Java-кода по графу переходов. Автомат A0_0

На рис. 28, 29 показаны схемы реализации мультипереходов и ветвящихся переходов.

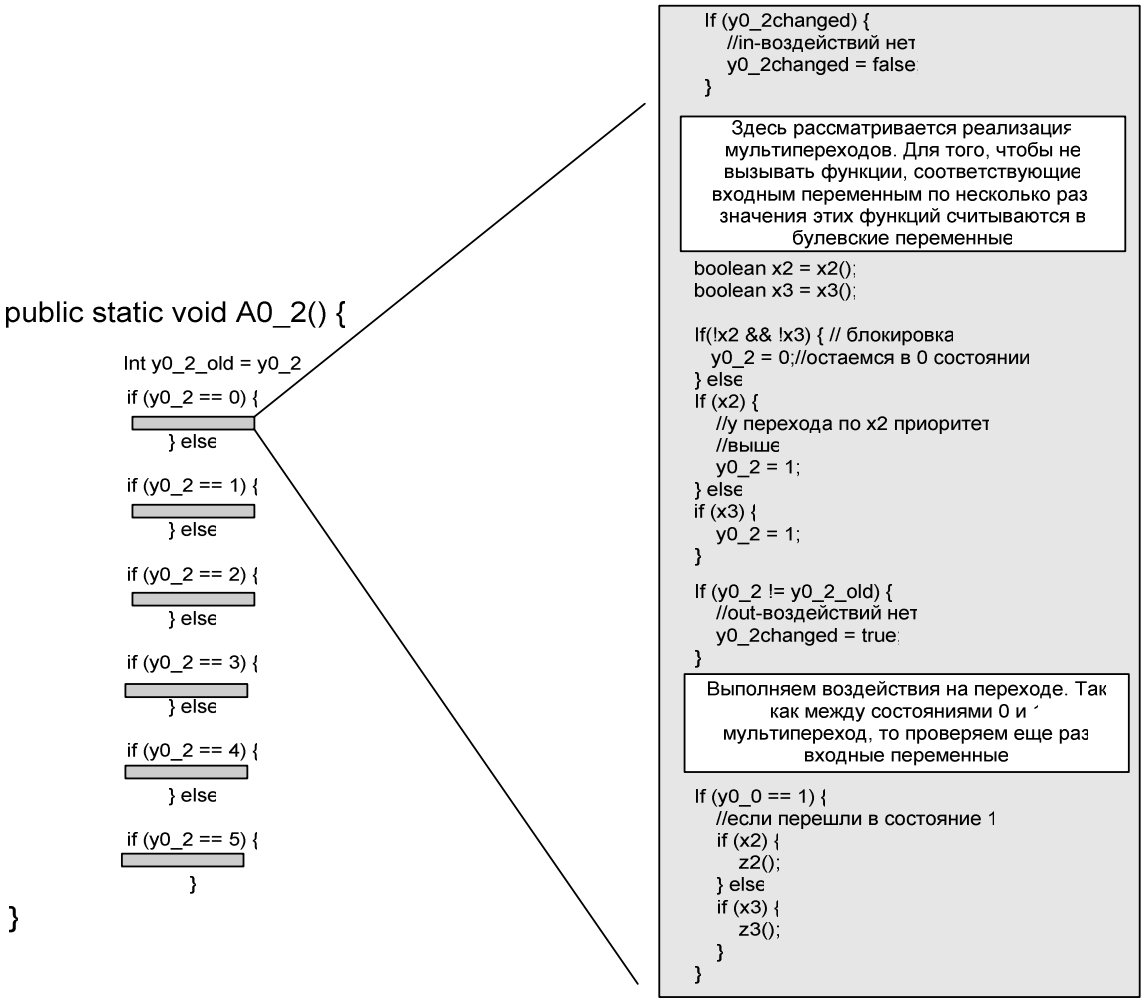

Рис. 28. Реализация мультипереходов. Автомат A0_2

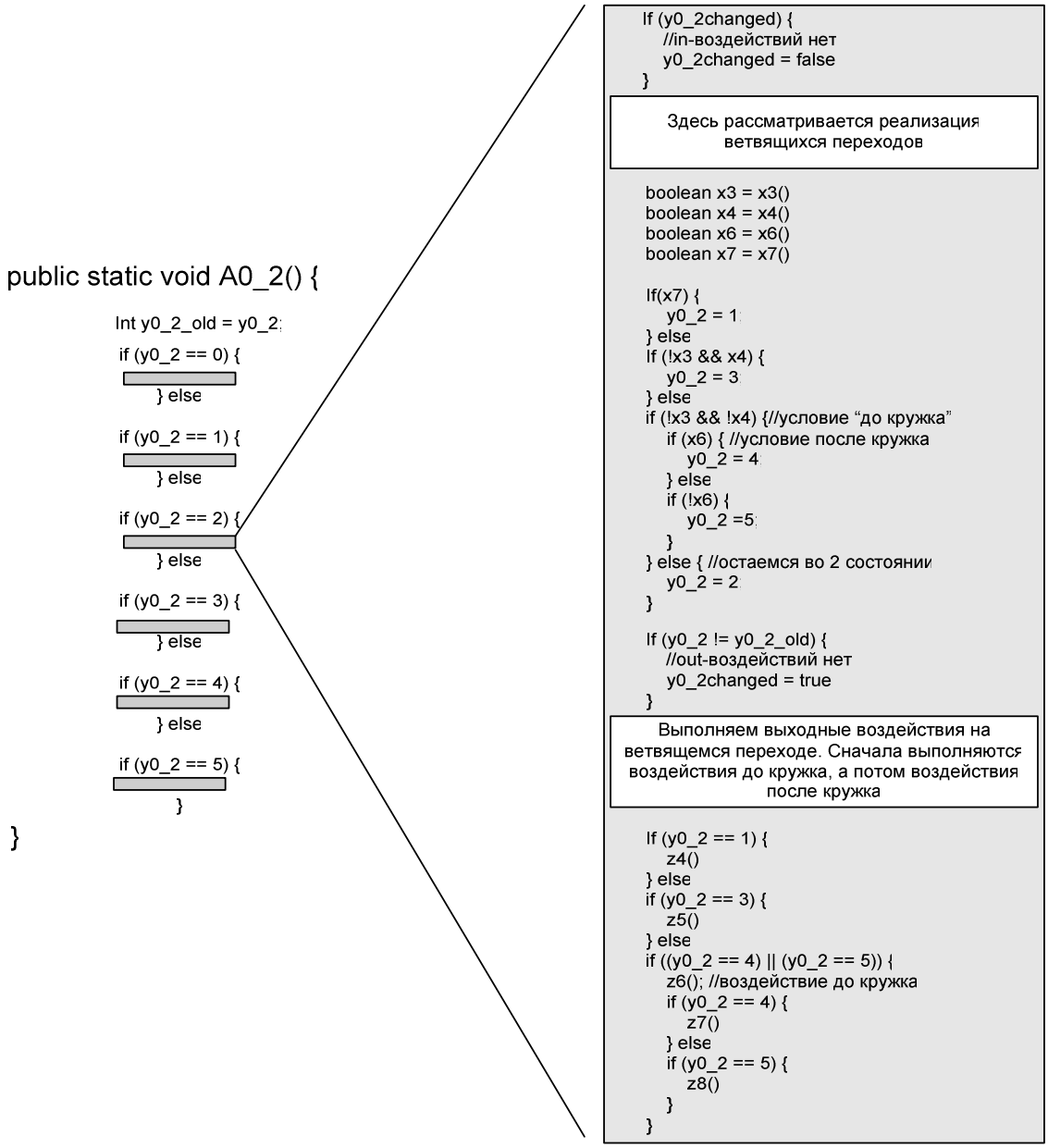

Рис. 29. Реализация ветвящихся переходов. Автомат A0_2

При программировании на языке *Java* основная функция main выглядит следующим образом:

```
public static void main(String args[]) {
     while (true) {
          A0_0();
     }
     }
```

Функция автомата A0_0, как внешнего (головного) автомата, вызывается в бесконечном цикле.

3.6. Взаимодействие между автоматами

Система управления транспортным роботом состоит из нескольких вложенных автоматов, каждый из которых выполняет определенную задачу. Для того, чтобы внешний автомат мог дать команду вложенным в него автоматам, ему необходимо выполнить соответствующее выходное воздействие. Вложенный автомат узнает о том, что ему была дана команда что-то сделать, когда соответствующая входная переменная примет значение `true`. После этого он должен сбросить эту переменную. Это выполняется при помощи соответствующего выходного воздействия. Если вложенному автомату для работы нет необходимости получать команды от внешнего автомата, он может запретить получение команд от внешнего автомата. Разрешение и запрещение получения команд выполняется при помощи выходных воздействий вложенного автомата. Внешний автомат также может получать команды от вложенного автомата и разрешать/запрещать получение команд. **Таким образом, один автомат может выставить входную переменную другому автомату посредством своего выходного воздействия. Другой автомат может сбросить значение этой переменной также при помощи своего выходного воздействия.**

Рассмотрим фрагмент *Java*-кода из реализации управляющей программы для транспортного робота. Эта программа будет подробно рассмотрена далее. Здесь этот фрагмент кода приводится только для иллюстрации. Для удобства все булевские переменные и передаваемые параметры, относящиеся к одному автомату, имеют префикс `An_`, где `n` – номер автомата. Переменная `An_AkListener` принимает значение `true`, если команды от автомата `Ak` разрешены автомату `An`, и `false` в противном случае.

```
//Выходное воздействие автомата A2, передающее команду
//автомату A3
```

```
public static void z2_6() {
     if (A3_A2Listener) {
          A3_findPath = true;
     }
}

//Входная переменная автомата A3, принимает значение true,
//если пришла команда от автомата A2
public static boolean x3_1() {
     return A3_findPath;
}
//Следующие два выходных воздействия автомата A3
//разрешают получение команд от автомата A2
public static void z3_2_0() {
     A3_A2Listener = false;
}
public static void z3_2_1() {
     A3_A2Listener = true;
}
//Это выходное воздействие автомата A3 сбрасывает
//переменную A3_findPath и соответственно x3_1
public static void z3_2_2() {
     A3_findPath = false;
}
```

3.7. Обмен сообщениями по инфракрасному порту в операционной системе *leJOS*

Операционная система *leJOS* предоставляет интерфейс *Serial* для реализации обмена сообщениями по ИК-порту. Этот интерфейс дает возможность посылать однобайтные сообщения от одного робота к другому, а также обрабатывать нажатия кнопок на пульте дистанционного управления. Разработчики ОС *leJOS* пытались реализовать стандартные *Java* интерфейсы для обмена информацией, но эта попытка оказалась неудачной. В качестве протокола нижнего уровня разработчики ОС *leJOS* использовали протокол, разработанный в Стэнфордском университете [39]. Принцип его работы можно охарактеризовать следующим образом. Для того чтобы послать какую-либо информацию, необходимо передать в систему специальный префикс, который определяет параметры передачи, а затем уже собственно информацию. Число f7, на-

пример, является указанием системе о том, что следующий байт необходимо послать по ИК-порту.

Посылка сообщений реализуется следующим образом. Достаточно записать в массив из двух байтов следующую информацию: в первый байт – шестнадцатеричное число f7, а второй байт – то, что необходимо передать. После этого вызвать функцию Serial.sendPacket(byte[] packet/*массив*/, 0, 2 /*число байт в массиве*/). В результате по ИК-порту будет передано однобайтовое сообщение другому роботу.

Прием сообщений немного сложнее – сообщения от пульта и от другого робота отличаются префиксами. Значение **первого байта** сообщения от **пульта** всегда равно **210**, второго – 0. Если в третьем байте единица стоит на позиции, соответствующей нулевой степени двух, то, следовательно, была нажата кнопка “1”. Если единица стоит на позиции для единичной степени двух, то была нажата кнопка “2”. Если она расположена на позиции для второй степени двух, то была нажата кнопка “3”. Значение **первого байта** сообщения от **другого робота не равно 210**. Следующий байт содержит посланную информацию. Метод Serial.isPacketAvailable() возвращает `true`, если был получено сообщение. Метод Serial.readPacket(byte[]) читает полученное сообщение в массив байтов.

Функция updateSignals(), реализованная как для транспортного робота, так и для робота поставщика, просматривает принятые сообщения и при необходимости обновляет значения булевских переменных, соответствующих факту принятия сообщения от пульта и/или другого робота, а также обновляет значение целочисленной переменной `remoteValue`, которая показывает, какая кнопка была нажата на пульте. Эта функция должна вызываться всегда в начале выполнения функции, соответствующей входной переменной или выходному

воздействию, ответственному за прием и передачу сообщений по ИК-порту. Одной из особенностей пульта является то, что он постоянно посылает сообщения при нажатой кнопке – он, как многие другие пульты, не прекращает передачу после отправки одного сообщения. Так как транспортному роботу необходимо только первое сообщение из этой “очереди”, остальные сообщения удаляются без обработки.

В данном разделе рассматривается реализация обмена сообщениями по ИК-порту для транспортного робота. Реализация для робота-поставщика практически идентична.

Чтобы проверить, есть ли сообщения от другого робота или от пульта, необходимо вызвать функцию updateSignals(), а затем проверить значения соответствующих булевских переменных. В случае транспортного робота следует проверить значения переменных dispencerSignal и remoteSignal. В случае если одно из этих значений true, значит пришло сообщение от робота-поставщика или дистанционного пульта соответственно. Также можно запретить сообщения от робота-поставщика и от пульта, присвоив переменным dispencerActive и remoteActive значение false. Разрешить сообщения, можно присвоив переменным dispencerActive и remoteActive значение true.

Значения переменных dispencerSignal и remoteSignal используются при формировании значений, соответствующих входных переменных. Значения переменных dispencerActive и remoteActive также формируют значения соответствующих входных переменных. Они также могут быть изменены при помощи соответствующих выходных воздействий.

```
//Функция updateSignals()
public static void updateSignals() {
     while (Serial.isPacketAvailable()) {
          Serial.readPacket(packet);
     //первый байт не равен 210, пришло сообщение от
     //робота-поставщика
          if ((packet[0] & 255) != 210) {
```

```
                if (dispencerActive) {
                     //Пришло сообщение от робота-постащика
                     dispencerSignal = true;
                }
           } else {
     //Пришло сообщение от пульта
                if (!remoteSignal && remoteActive) {
                     int c1=packet[1] & 255;
                     int c2=packet[2] & 255;
                     if (c1 == 0) {
                          //Нажата кнопка 1
                          if (c2==0x01) remoteValue = 1;
                          //Нажата кнопка 2
                          else if (c2==0x02) remoteValue = 2;
                          //Нажата кнопка 3
                          else if (c2==0x04) remoteValue = 3;
                     }
                     //Пришло сообщение от пульта. Обновляем
                     //значение переменной remoteSignal
                     remoteSignal = true;
                }
           }
     }
}
```

//Пример входной переменной, возвращающей значение //true, если получено сообщение по ИК-порту от пульта и //значение false – в противном случае

```
public static boolean x1_0_0() {
     //Сначала вызывается функция updateSignals() updateSignals();
     //Затем возвращается значение булевской переменной
     //remoteSignal
     return remoteSignal;
}

//Пример выходного воздействия, запрещающего сообщения от пульта
public static void z1_3_0() {
          updateSignals();
          remoteActive = false;
}

//Пример выходного воздействия, разрешающего сообщения от пульта
public static void z1_3_1() {
          updateSignals();
          remoteActive = true;
}
//Пример выходного воздействия, обрабатывающего сообщение от
//пульта.
//Число предметов считывается в переменную candyCount
     public static void z1_3_2() {
          updateSignals();
          if (remoteSignal) {
               remoteSignal = false;
               candyCount = remoteValue;
```

```
        }
    }
```

3.6. Взаимодействие агентов

Взаимодействие агентов описывается при помощи соответствующей схемы. В данном проекте она оказалось довольно простой – пульт посылает три типа сообщений, робот-поставщик посылает сообщение транспортному роботу о конце погрузки и транспортный робот посылает сообщение роботу-поставщику. В сообщении указано о числе предметов для доставки. Схема взаимодействия роботов приведена на рис. 30.

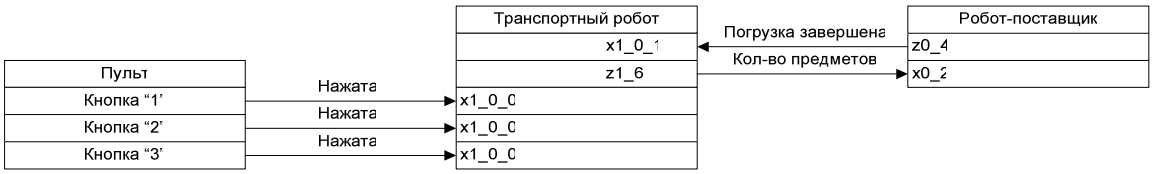

Рис. 30. Схема взаимодействия роботов

3.7. Транспортный робот

3.7.1. Модель транспортного робота

На рис. 31 приведен виртуальный транспортный робот, сконструированный с помощью программы *MLCAD*. Файл с описанием транспортного робота для программы *MLCAD* размещен по адресу http://is.ifmo.ru/projects/lego.

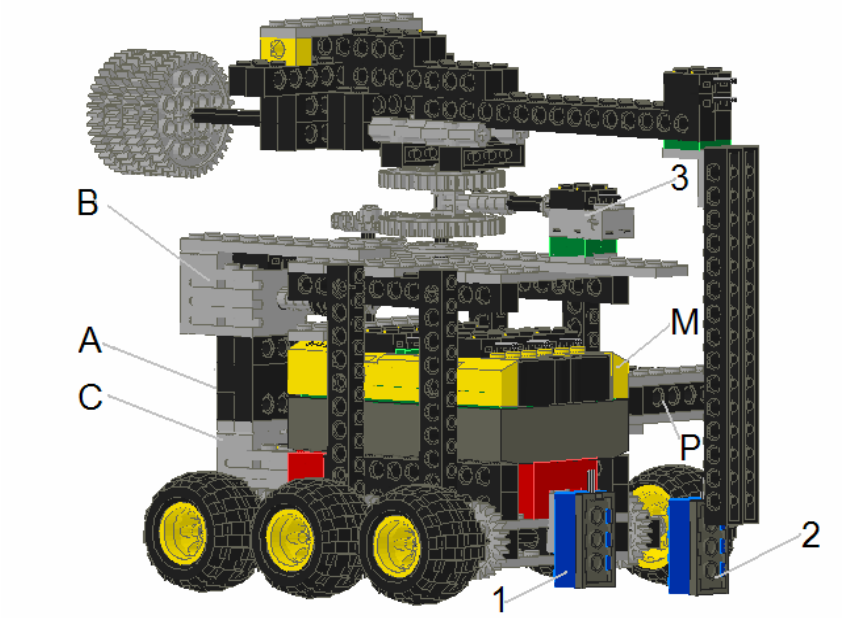

Рис. 31. Виртуальный транспортный робот

На этом рисунке отмечены следующие части робота:

- световой датчик 1 подключен к гнезду 1 корпуса микроконтроллера (номер гнезда указан на корпусе микроконтроллера);
- световой датчик поиска пути 2, размещенный на вращающейся штанге и подключенный к гнезду 2;
- датчик касания 3, необходимый для центрирования штанги, подключен к гнезду 3;
- двигатель *A*, вращающий левые колеса, подключен к гнезду A;
- двигатель *C*, вращающий правые колеса, подключен к гнезду C;
- двигатель *B*, отвечающий за вращение штанги, подключен к гнезду B;
- поддон для предметов *P*;
- микроконтроллер с инфракрасным портом *M*.

Датчик 1 закреплен неподвижно. Датчик 2 размещен на вращающейся штанге, прикрепленной к двигателю B. Она способна поворачиваться на 180° влево и вправо попеременно.

На основе схемы виртуального робота на рис. 31 собран транспортный робот (рис. 32).

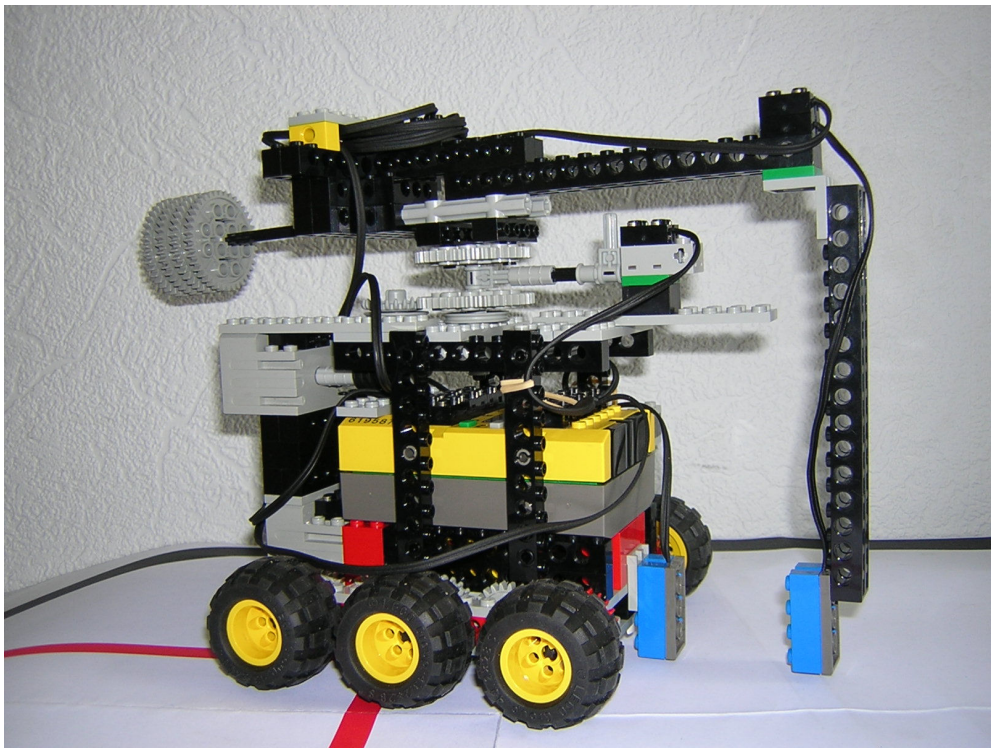

Рис. 32. Транспортный робот

На рис. 33 показано прохождение поворота транспортным роботом.

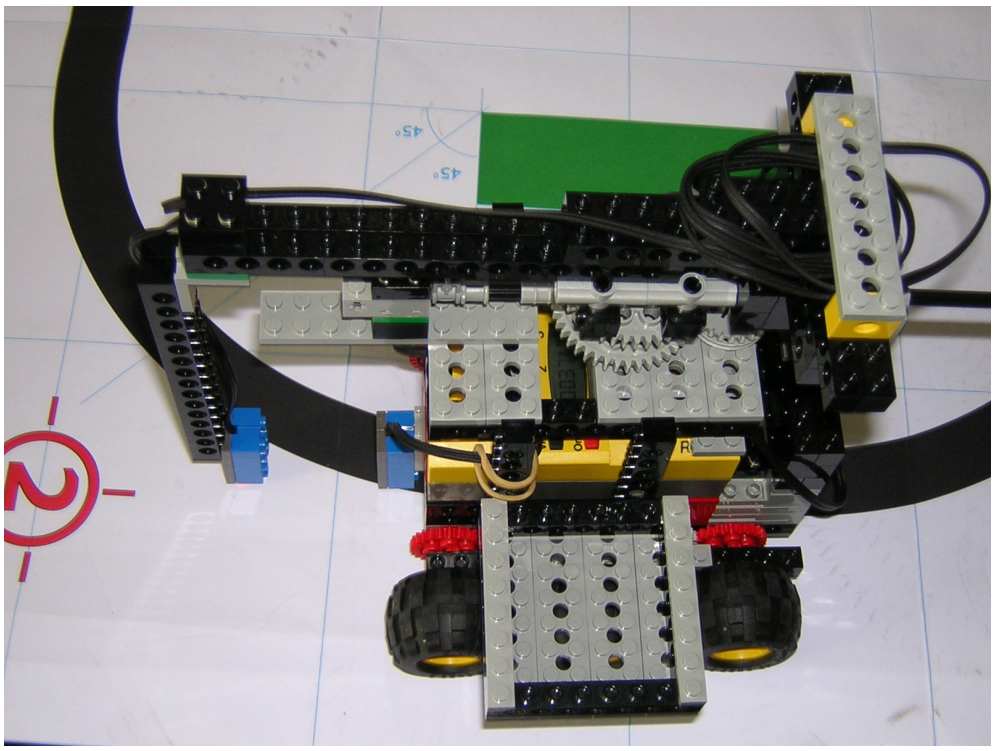

Рис. 33. Прохождение поворота транспортным роботом

3.7.2. Особенности конструкции транспортного робота

Благодаря использованию червячной передачи для связи двигателей с колесами, робот имеет возможность практически мгновенно останавливаться. Тормозной путь достаточно мал. Червячная передача не позволяет роботу двигаться самопроизвольно.

Робот является колесным и способен с высокой точностью вращаться вокруг своего центра. Это объясняется тем, что колеса имеют глубокий протектор, повышающий сцепление с «дорогой» и предотвращающий проскальзывание на поворотах. Сначала был построен гусеничный робот, однако авторы были вынуждены отказаться от него, так как его сильно сносило с пути на поворотах.

Для соединения мотора со штангой, к которой прикреплен световой датчик поиска пути 2, также используется червячная передача. Это позволяет уменьшить инерцию штанги при вращении. Штанга способна вращаться на 180° влево и вправо попеременно. При повороте более чем на 180° провод, соединяющий штангу с микроконтроллером, намотается на шестеренки, что приведет к аварии. Управляющая программа противодействует таким поворотам.

На другом конце штанги прикреплен противовес, который обеспечивает плавность хода. Благодаря уравновешенности конструкции штанги и червячной передачи, штангу можно центрировать относительно светового датчика 1 с достаточно высокой точностью (погрешность не более 5°).

Для центрирования применяется датчик касания 3, установленный на корпусе робота. В тот момент, когда световые датчики и штанга находятся на одной прямой, датчик срабатывает и передает соответствующий сигнал своему микроконтроллеру для остановки двигателя *B*.

Световой датчик поиска пути 2, закрепленный на конце штанги, соединяется с микроконтроллером проводом, который укреплен вдоль всей штанги. Световой датчик 1 закреплен на корпусе робота спереди, его положение фиксировано.

ИК-порт встроен в микроконтроллер и расположен в передней части робота. Конструкция робота выбрана таким образом, чтобы не препятствовать распространению сигнала от порта.

Поддон находится слева от микроконтроллера. В его конструкции предусмотрены бортики, чтобы предметы не вываливались.

В целом конструкция робота, если не считать штанги, крепкая и устойчивая. Возможна иная конструкция робота, в которой штанга крепится не над микроконтроллером, а под ним.

3.8. Управление транспортным роботом

3.8.1. Спецификация на управляющую систему

Транспортный робот должен, обработав сообщение от пульта, развернуться, проехать по пути, используя штангу со световым датчиком. Доехав до участка пути, помеченного фольгой, он должен подъехать к роботу-поставщику, послать ему сообщение о количестве предметов к выдаче, получить предметы, отъехать, развернуться, проехать по пути до участка, помеченного фольгой, и ожидать следующего сообщения от пульта.

3.8.2. Алгоритм для движения по пути

Для корректного движения по пути был разработан соответствующий алгоритм. Он довольно прост, но без знания его весьма проблематично понять логику работы автоматов

A2, A3 и A5, входящих в состав управляющей системы транспортного робота. Предположим, что первоначально световой датчик 1 находится на пути. Транспортный робот также ориентирован по пути. Пока световой датчик 1 находится на пути, робот движется вперед. Как только световой датчик 1 покажет отличное от пути значение, следует этап корректировки этого значения – в течение определенного промежутка времени робот продолжит двигаться вперед, потом показания фиксированного светового датчика 1 снимаются повторно. Это сделано для того, чтобы была возможность отличить фольгу от белого поля. Как уже говорилось ранее, показания светового датчика лежат в промежутке от 0 до 100. Когда световой датчик снимает показания на границе между фольгой и путем, он зачастую выдает число, соответствующее белому полю, что ведет к неправильной работе. С этой целью был введен этап корректировки показаний. Если после этапа корректировки световой датчик 1 возвращает цвет пути, то робот продолжает двигаться вперед. Если цвет белый, то робот пытается найти путь при помощи штанги. Если цвет фольги, то робот доехал до конца пути и останавливается. Если цвет определен некорректно, то робот также останавливается, а алгоритм завершает работу с сообщением об ошибке.

Теперь опишем то, как робот ищет путь. Сначала штанга движется влево, если путь после определенного промежутка времени не найден, то она движется вправо. Если и справа путь не найден – то выводится сообщение об ошибке. Если путь найден справа или слева, то робот поворачивает в соответствующем направлении, штанга возвращается в центрированное положение. После того, как световой датчик 1 обнаружит путь, то робот совершает еще небольшой поворот в том же направлении, чтобы световой датчик 1 оказался на

центре пути. Это делается для более эффективного движения по пути. Далее транспортный робот продолжает движение вперед.

3.8.3. Общее описание управляющей системы

Транспортный робот управляется четырьмя автоматами – A1, A2, A3 и A5. Автоматы взаимодействуют друг с другом посредством вложенности и командами. Основным автоматом является автомат A1. Он отвечает за обработку сообщений от пульта управления, обмен сообщениями с роботом-поставщиком, а также за поворот после получения сообщений от пульта и робота-поставщика. Автомат A2 отвечает за движение по пути. Автомат A3 управляет штангой, на которой закреплен световой датчик 2. Автомат A5 используется для возвращения штанги в центральное положение – центрирование штанги. Идентификаторы входных переменных и выходных воздействий, общих для автоматов транспортного робота, записываются в форме z_0.. или x_0.., где вместо точек стоит уникальное выражение, состоящее из цифр и символов подчеркивания. Входные переменные и выходные воздействия, специфичные для одного автомата, записываются в форме z_n.. или x_n.., где вместо символа n стоит номер этого автомата.

3.8.3.1. Перечень констант

В этом разделе перечислены все константы, определенные для транспортного робота. Значения констант, определенных опытным путем, приведены в приложении 3.

VOLTAGE_TIME – время, в течение которого выведено на дисплей напряжение на батареях. Измеряется в миллисекундах.

TURN_TIME – время, необходимое для поворота транспортного робота на 180°. Определяется экспериментально.

BACK_TIME – время, необходимое для транспортному роботу, чтобы отъехать от робота-поставщика. Определяется экспериментально.

ROT_TIME – максимальное время поворота транспортного робота, если он исправен.

ROT_EX_TIME – время, необходимое для поворота транспортного робота влево или вправо, чтобы световой датчик 1 сместился с границы пути к его центру.

CORRECT_TIME – время, необходимое для движения вперед с целью подтверждения цвета светового датчика 1.

ES_POWER – значение мощности работы двигателей колес для движения вперед и назад.

ER_POWER – значение мощности работы двигателей колес для поворота.

RR_TIME – константа, соответствующая времени, за которое штанга повернется на 180°.

RB_TIME – константа, соответствующая максимальному возможному времени возвращения штанги назад, если она исправна.

ERR_POWER – значение мощности работы двигателя штанги.

BLACK_UP – верхняя граница значения на световых датчиках, соответствующая черному цвету (цвету пути).

BLACK_DOWN – нижняя граница, соответствующая черному цвету (цвету пути).

WHITE_UP – верхняя граница значения на световых датчиках, соответствующая белому цвету (цвету поля).

WHITE_DOWN – нижняя граница, соответствующая черному цвету (цвету пути).

FOIL_UP – верхняя граница значения на световых датчиках, соответствующая фольге (конец пути).

FOIL_DOWN – нижняя граница, соответствующая фольге (конец пути).

OTHER_UP – верхняя граница значения на световых датчиках, соответствующая некорректному цвету.

OTHER_DOWN – нижняя граница, соответствующая некорректному цвету.

3.8.3.2. Перечень входных переменных, общих для автоматов A2 и A3

x0_4_0 – значение светового датчика 1 соответствует полю.

x0_4_1 – значение светового датчика 1 соответствует пути.

x0_4_2 – значение светового датчика 1 соответствует фольге.

x0_4_3 – значение светового датчика 1 ничему не соответствует.

x0_5_0 – значение светового датчика 2 соответствует полю.

x0_5_1 – значение светового датчика 2 соответствует пути.

x0_5_2 – значение светового датчика 2 соответствует фольге.

x0_5_3 – значение светового датчика 2 ничему не соответствует.

3.8.3.3. Перечень выходных воздействий, общих для автоматов A1, A2, A3 и A5

z0_5_1 – включить таймер 1.

z0_6_1 – остановить и обнулить таймер 1.

z0_5_2 – включить таймер 2.

z0_6_2 – остановить и обнулить таймер 2.

z0_5_3 – включить таймер 3.

z0_6_3 – остановить и обнулить таймер 3.

z0_5_4 – включить таймер 4.

z0_6_4 – остановить и обнулить таймер 4.

z0_7 – запустить левый и правый двигатели вперед со значением мощности ES_POWER.

z0_8 – запустить левый и правый двигатели назад со значением мощности ES_POWER.

z0_9 – остановить левый и правый двигатели.

z0_10 – запустить левый двигатель назад, а правый вперед со значением мощности ER_POWER. Для поворота налево.

z0_11 – запустить левый двигатель вперед, а правый назад со значением мощности ER_POWER. Для поворота направо.

3.8.4. Автомат A1. Управление транспортным роботом

3.8.4.1. Краткое описание

Автомат A1 получает сообщение от пульта, далее он (посредством автомата A2) проводит транспортный робот по пути, получает предметы от робота-поставщика и возвращается к месту доставки (также посредством автомата A2).

3.8.4.2. Перечень констант

VOLTAGE_TIME – время, в течение которого выведено на дисплей напряжение на батареях. Измеряется в миллисекундах.

TURN_TIME – время, необходимое для разворота робота. Определяется экспериментально.

BACK_TIME – время, необходимое для подъезда и отъезда от робота-поставщика. Определяется экспериментально.

3.8.4.3. Перечень переменных

candyCount – количество предметов для доставки. Начальное значение – ноль.

3.8.4.4. Перечень входных переменных

x1_0_0 – сообщение от пульта по ИК-порту получено. Не может быть выставлена одновременно с переменной x1_0_1.

x1_0_1 – сообщение от робота-поставщика по ИК-порту получено. Не может быть выставлена одновременно с переменной x1_0_0.

x1_2 – таймер 1 превысил значение TURN_TIME.

x1_3 – таймер 1 превысил значение BACK_TIME.

x1_4 – таймер 1 превысил значение VOLTAGE_TIME.

x1_6 – получена команда от автомата A2 посредством выходного воздействия z2_3 о том, что конец пути достигнут. На схеме связей и далее по тексту последнее обозначается как A2(z2_3).

3.8.4.5. Блокировки

y2 == 3 – транспортный робот не смог проехать по заданному пути (авария).

3.8.4.6. Перечень состояний y1

0 – Начало.

1 – Вывод напряжения на дисплей.

2 – Ожидание сообщения от пульта.

3 – Поворот.

4 – Движение по линии.

5 – Въезд.

6 – Ожидание сообщения от робота-поставщика

7 – Реверс.

8 – Поворот после реверса.

9 – Движение по линии назад.

10 – Авария.

3.8.4.7. Перечень выходных воздействий

z0_5_1 – включить таймер 1.

z0_6_1 – остановить и обнулить таймер 1.

z0_7 – запустить левый и правый двигатели вперед со значением мощности ES_POWER.

z0_8 – запустить левый и правый двигатели назад со значением мощности ES_POWER.

z0_9 – остановить левый и правый двигатели.

z0_11 – запустить левый двигатель вперед, правый назад со значением мощности ER_POWER. Для поворота направо.

z1_0 – вывести на дисплей напряжение на батареях.

z1_1 – перевести световой датчик 1 в активный режим.

z1_2 – перевести световой датчик 2 в активный режим.

z1_3_0 – запретить сообщения от пульта.

z1_3_1 – разрешить сообщения от пульта.

z1_3_2 – обработать сообщение от пульта.

z1_4_0 – запретить сообщения от робота-поставщика.

z1_4_1 – разрешить сообщения от робота-поставщика.

z1_4_2 – обработать сообщения от робота-поставщика.

z1_5 – вывести на дисплей сообщение об аварии – число 666.

z1_6 – послать сообщение роботу-поставщику о количестве предметов для погрузки.

z1_7_0 – запретить получение команд от автомата A2 (A1_A2_Listener = false).

z1_7_1 – разрешить получение команд от автомата A2 (A1_A2_Listener = true).

z1_7_2 – сбросить переменную x1_6.

z1_8 – выдать команду автомату A2 о начале работы посредством входной переменной x2_1. На схеме связей и далее по тексту последнее обозначается как A2(x2_1).

На рис. 34 приведена схема связей автомата A1, а на рис. 35 – его граф переходов.

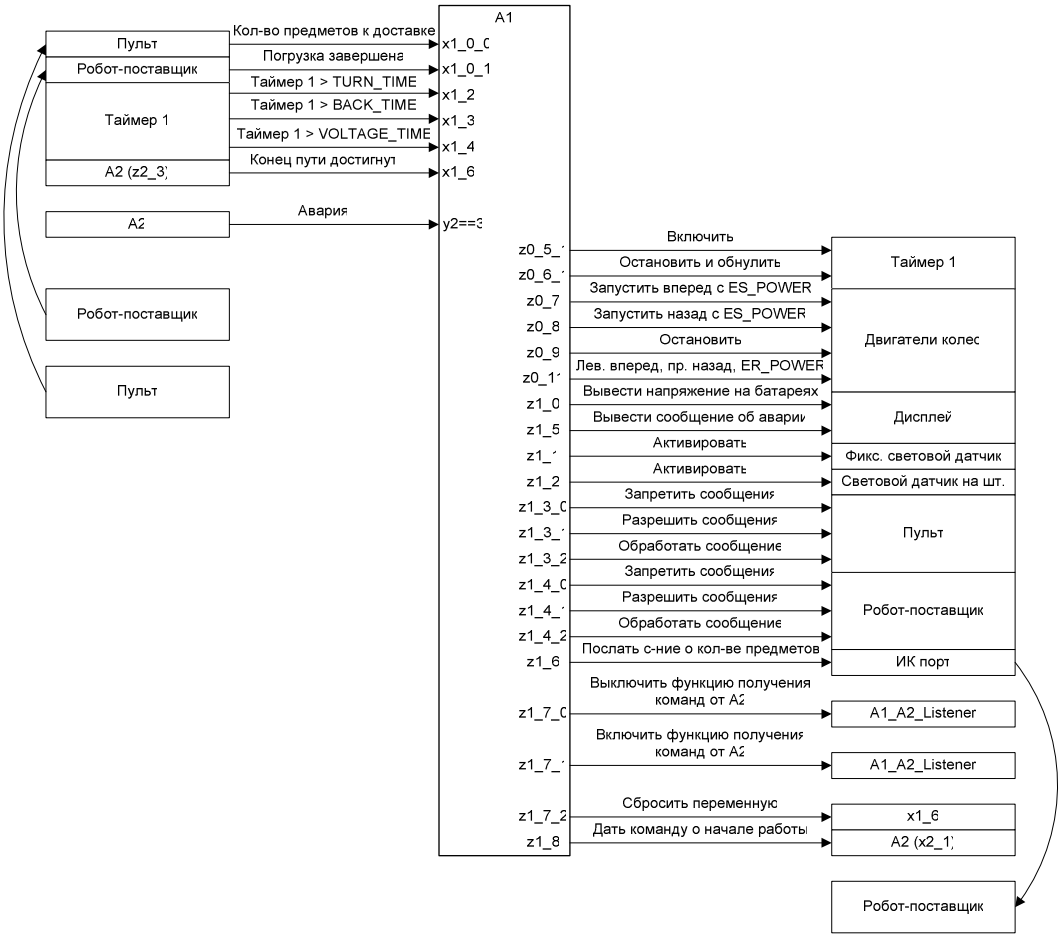

Рис. 34. Схема связей автомата A1. Управление транспортным роботом

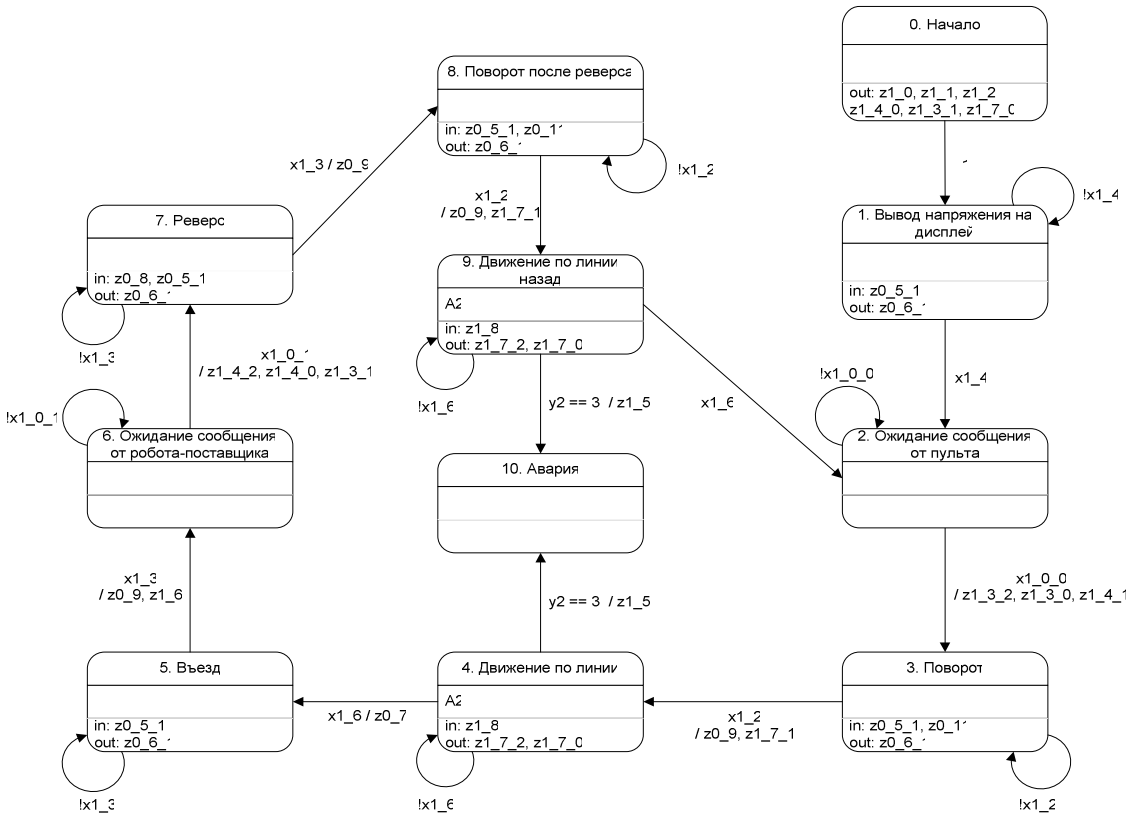

Рис. 35. Граф переходов автомата A1. Управление транспортным роботом

3.8.5. Автомат A2. Движение по пути

3.8.5.1. Краткое описание

Логика работы этого автомата реализует алгоритм, описанный в разд. 3.8.2.

Автомат A2 управляет транспортным роботом при его движении по пути. Робот едет равномерно по пути, либо пока не доедет до конца пути (переменная x0_4_2 == true), либо не съедет с пути на поле (переменная x0_4_0 == true). В случае, если робот съезжает с пути, то автомат A2 при помощи штанги ищет потерянный путь и возвращает робот обратно на этот путь.

3.8.5.2. Перечень констант

ROT_EX_TIME – время, необходимое для поворота робота влево или вправо для того, чтобы световой датчик 1 сместился от границы пути к его центру.

CORRECT_TIME – время, необходимое для движения вперед с целью подтверждения цвета светового датчика 1.

ES_POWER – значение мощности работы двигателей колес для движения вперед и назад.

ER_POWER – значение мощности работы двигателей колес для поворота.

ROT_TIME – максимальное время поворота транспортного робота, если он исправен.

3.8.5.3. Блокировки

y3 == 5 – авария. Путь не был найден.

3.8.5.4. Перечень входных переменных

x0_4_0 – значение светового датчика 1 соответствует полю.

x0_4_1 – значение светового датчика 1 соответствует пути.

x0_4_2 – значение светового датчика 1 соответствует фольге.

x0_4_3 – значение светового датчика 1 ничему не соответствует.

x0_5_0 – значение светового датчика 2 соответствует полю.

x0_5_1 – значение светового датчика 2 соответствует пути.

x0_5_2 – значение светового датчика 2 соответствует фольге.

x0_5_3 – значение светового датчика 2 ничему не соответствует.

x2_1 – получена команда от автомата A1 посредством выходного воздействия z1_8, о том, что следует начинать работу. На схеме связей и далее по тексту последнее обозначается как A1(z1_8).

x2_3 – получена команда от автомата A3 посредством выходного воздействия z3_0, о том, что путь найден слева. На схеме связей и далее по тексту последнее обозначается как A3(z3_0).

x2_4 – получена команда от автомата A3 посредством выходного воздействия z3_1, о том, что путь найден справа. На схеме связей и далее по тексту последнее обозначается как A3(z3_1).

x2_7 – таймер 2 превысил значение CORRECT_TIME.

x2_8 – таймер 2 превысил значение ROT_EX_TIME.

x2_9 – таймер 2 превысил значение ROT_TIME.

3.8.5.5. Перечень состояний y2

0 – Начало пути.

1 – Движение вперед.

2 – Подтверждение цвета.

3 – Авария.

4 – Корректировка.

5 – Поиск пути.

6 – Поворот.

7 – Центрирование по пути.

3.8.5.6. Перечень выходных воздействий

z0_5_2 – включить таймер 2.

z0_6_2 – остановить и обнулить таймер 2.

z0_7 – запустить левый и правый двигатели колес вперед со значением мощности ES_POWER.

z0_9 – остановить левый и правый двигатели колес.

z0_10 – для поворота налево запустить левый двигатель назад, а правый вперед со значением мощности ER_POWER.

z0_11 – для поворота направо запустить левый двигатель вперед, а правый назад со значением мощности ER_POWER.

z2_3 – выдать команду автомату A1 посредством входной переменной x1_6 о том, что достигнут конец пути. На схеме связей и далее по тексту последнее обозначается как A1(x1_6).

z2_4_0 – запретить получение команд от автомата A1(A2_A1_Listener = false).

z2_4_1 – разрешить получение команд от автомата A1(A2_A1_Listener = true).

z2_4_2 – сбросить переменную (x2_1).

z2_5_0 – запретить получение команд от автомата A3(A2_A3_Listener = false).

z2_5_1 – разрешить получение команд от автомата A3(A2_A3_Listener = true).

z2_5_2 – сбросить переменные x2_3 и x2_4.

z2_6 – дать команду автомату A3 посредством переменной x3_1 о начале работы. На схеме связей и далее по тексту последнее обозначается как A3(x3_1).

На рис. 36 приведена схема связей автомата A2, а на рис. 37 – его граф переходов.

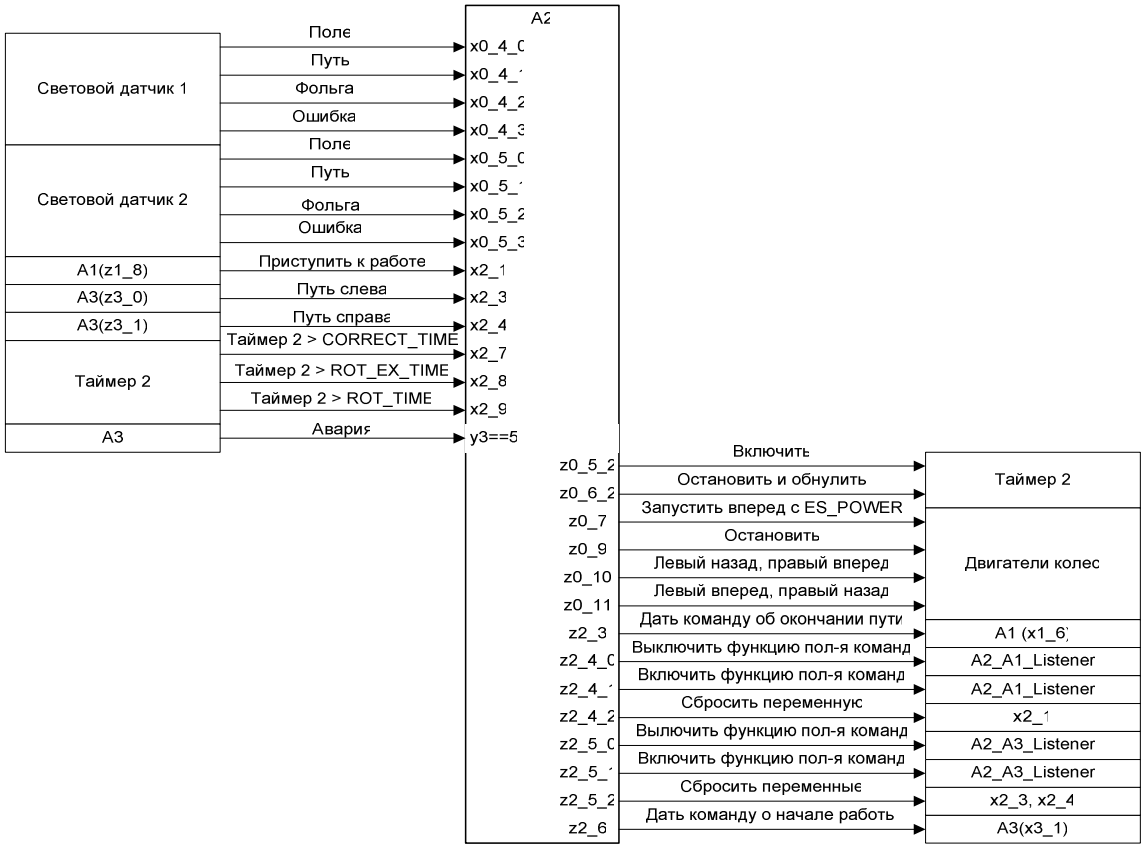

Рис. 36. Схема связей автомата A2. Транспортный робот. Движение по пути

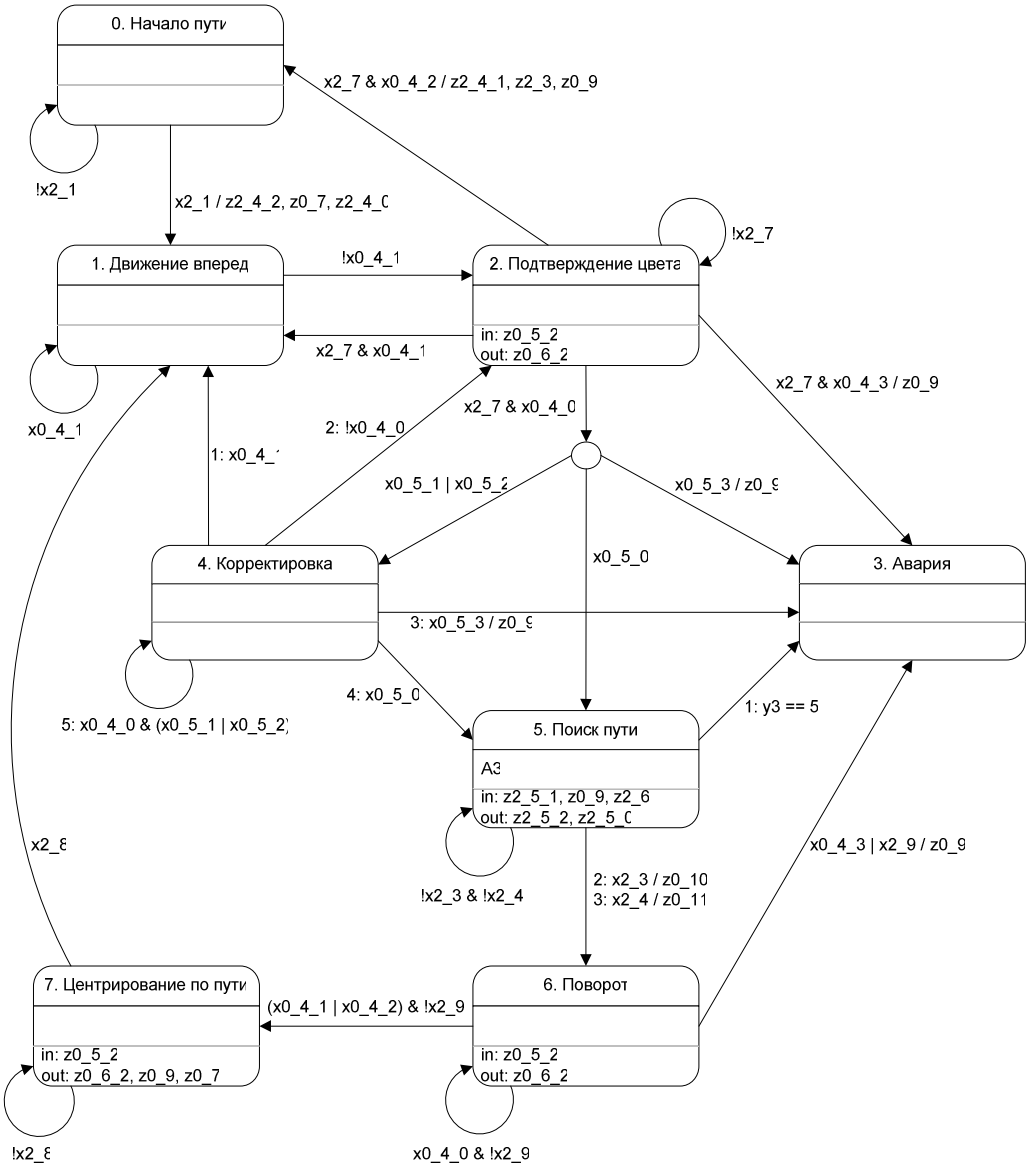

Рис. 37. Граф переходов автомата A2. Транспортный робот. Движение по пути

3.8.6. Автомат A3. Управление штангой

3.8.6.1. Краткое описание

Автомат A3 ищет путь при помощи штанги. Вначале автомат запускает штангу для движения налево. Если путь слева не найден, то он ищет путь справа. Если путь был найден либо слева, либо справа, то автомату A2 дается команда о том, что путь был найден соответственно слева или справа. В противном случае автомат A3 переходит в состояние “Авария”.

3.8.6.2. Перечень констант

RR_TIME – константа, соответствующая времени, за которое штанга повернется на 180°.

ERR_POWER – значение мощности работы двигателя штанги.

3.8.6.3. Перечень входных переменных

x0_5_0 – значение светового датчика 2 соответствует полю.

x0_5_1 – значение светового датчика 2 соответствует пути.

x0_5_2 – значение светового датчика 2 соответствует фольге.

x0_5_3 – значение светового датчика 2 ничему не соответствует.

x3_1 – получена команда от автомата A2 посредством выходного воздействия z2_6 о том, что требуется начать работу. На схеме связей и далее по тексту последнее обозначается как A2(z2_6).

x3_3 – таймер 3 превысил значение RR_TIME.

x3_5 – получена команда от автомата A5 посредством выходного воздействия z5_4 о том, что штанга центрирована. На схеме связей и далее по тексту последнее обозначается как A5(z5_4).

3.8.6.4. Перечень состояний

0 – Штанга по центру.

1 – Поиск пути слева.

2 – Центрирование слева.

3 – Поиск пути справа.

4 – Центрирование. Путь не найден.

5 – Авария.

6 – Центрирование справа.

3.8.6.5. Перечень выходных воздействий

z0_5_3 – включить таймер 3.

z0_6_3 – остановить и обнулить таймер 3.

z3_0 – дать команду автомату A2 посредством входной переменной x2_3, о том, что путь найден слева. На схеме связей и далее по тексту последнее обозначается как A2(x2_3).

z3_1 – дать команду автомату A2 посредством входной переменной x2_4, о том, что путь найден справа. На схеме связей и далее по тексту последнее обозначается как A2(x2_4).

z3_2_0 – запретить получение команд от автомата A2 (A3_A2_Listener = false).

z3_2_1 – разрешить получение команд от автомата A2 (A3_A2_Listener = true).

z3_2_2 – сбросить переменную x3_1.

z3_4_0 – запретить получение команд от автомата A5 (A3_A5_Listener = false).

z3_4_1 – разрешить получение команд от автомата A5 (A3_A5_Listener = true).

z3_4_2 – сбросить переменную x3_5

z3_5 – запустить двигатель штанги влево с мощностью ERR_POWER.

z3_6 – запустить двигатель штанги вправо с мощностью ERR_POWER.

z3_7 – остановить двигатель штанги.

z3_8 – выдать команду A5 посредством входной переменной x5_1 о том, что требуется центрировать штангу. На схеме связей и далее по тексту последнее обозначается как A5(x5_1).

На рис. 38 приведена схема связей автомата A3, а на рис. 39 – его граф переходов.

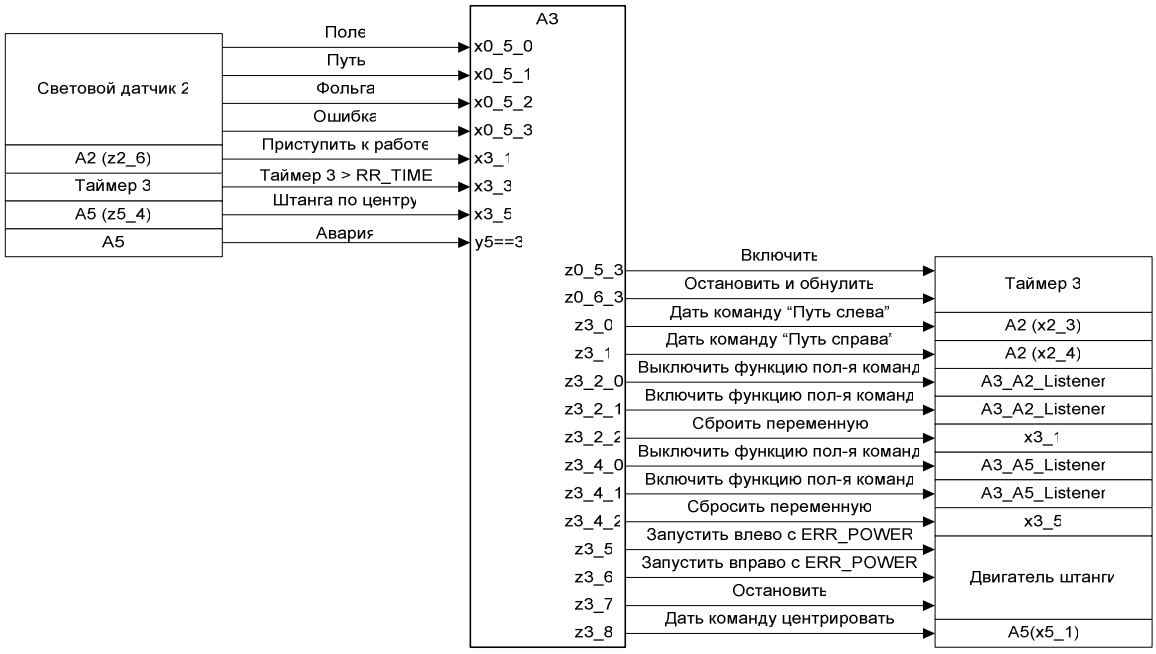

Рис. 38. Схема связей автомата A3. Транспортный робот. Управление штангой

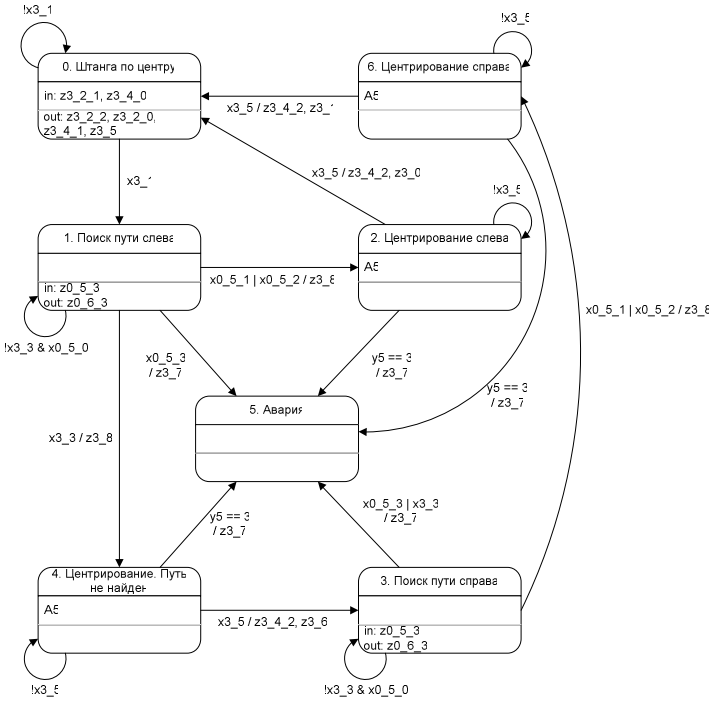

Рис. 39. Граф переходов автомата A3. Транспортный робот. Управление штангой

3.8.7. Автомат A5. Центрирование штанги

3.8.7.1. Краткое описание

Автомат A5 выполняет центрирование штанги путем изменения направления движения штанги и отслеживания показаний датчика касаний 3.

3.8.7.2. Перечень констант

RB_TIME – константа, соответствующая максимальному времени возвращения штанги, если она исправна.

3.8.7.3. Перечень входных переменных

x5_0 – датчик касания 3 штанги нажат.

x5_1 – пришла команда от автомата A3 посредством выходного воздействия z3_8 о том, что надо начать работу. На схеме связей и далее по тексту последнее обозначается как A3(z3_8).

x5_2 – таймер 4 превысил значение RB_TIME.

3.8.7.4. Перечень состояний

0 – Ожидание сообщения.

1 – Поиск датчика.

2 – Движение обратно.

3 – Авария.

3.8.7.5. Перечень выходных воздействий

z0_5_4 – включить таймер 4.

z0_6_4 – остановить и обнулить таймер 4.

z5_1_0 – запретить получение команд от автомата A3 (A5_A3_Listener = false).

z5_1_1 – разрешить получение команд от автомата A3 (A5_A3_Listener = true).

z5_1_2 – сбросить переменную x5_1.

z5_2 – изменить направление движения штанги.

z5_3 – остановить штангу.

z5_4 – дать команду автомату A3 посредством входной переменной x3_5 о том, что штанга по центру. На схеме связей и далее по тексту последнее обозначается как A5(x5_1).

На рис. 40 приведена схема связей автомата A5, а на рис. 41 – его граф переходов.

Рис. 40. Схема связей автомата A5. Транспортный робот. Центрирование штанги

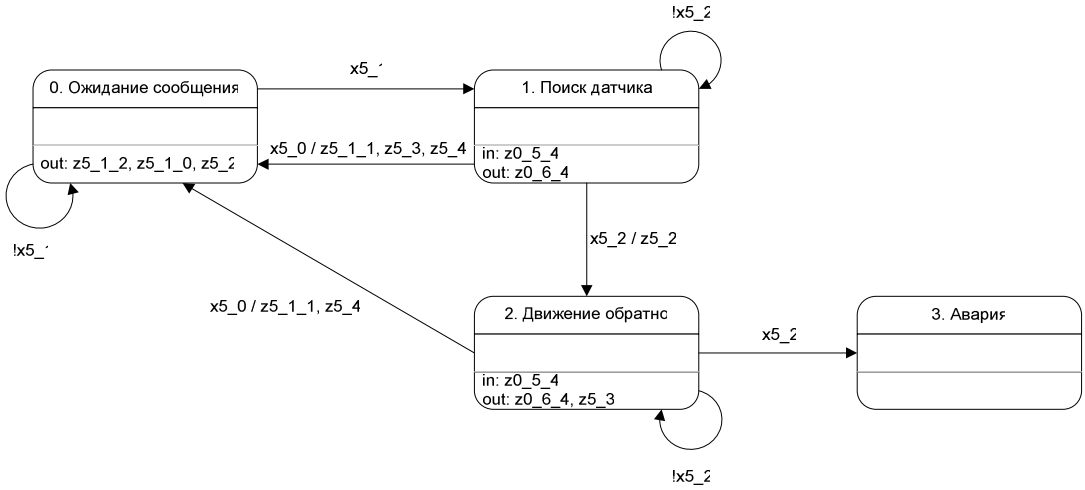

Рис. 41. Граф переходов автомата A5. Транспортный робот. Центрирование штанги

3.9. Робот-поставщик

На рис. 42 приведен виртуальный робот-поставщик, сконструированный с помощью программы *MLCAD*. Файл с описанием робота-поставщика размещен по адресу http://is.ifmo.ru/projects/lego

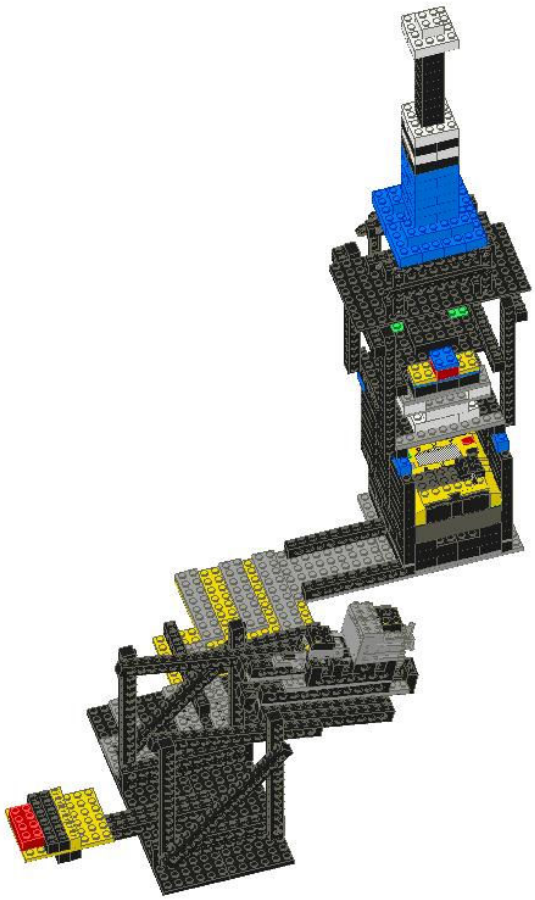

Рис. 42. Виртуальный робот-поставщик

Робот-поставщик, как следует из приведенного рисунка, состоит из двух основных частей: устройство для выдачи предметов и башня.

На роботе-поставщике установлены двигатель и сенсор касания, при помощи которого можно подсчитывать количество оборотов сбрасывающего рычажка. При каждом обороте рычажка, выдается один предмет с наклонной плоскости. Таким образом, обеспечивается выдача необходимого количества предметов. Поддон транспортного робота находится сбоку, устройство для выдачи предметов также находится также

сбоку от микроконтроллера, обеспечивающего функционирование робота-поставщика.

На рис. 43 приведен робот-поставщик.

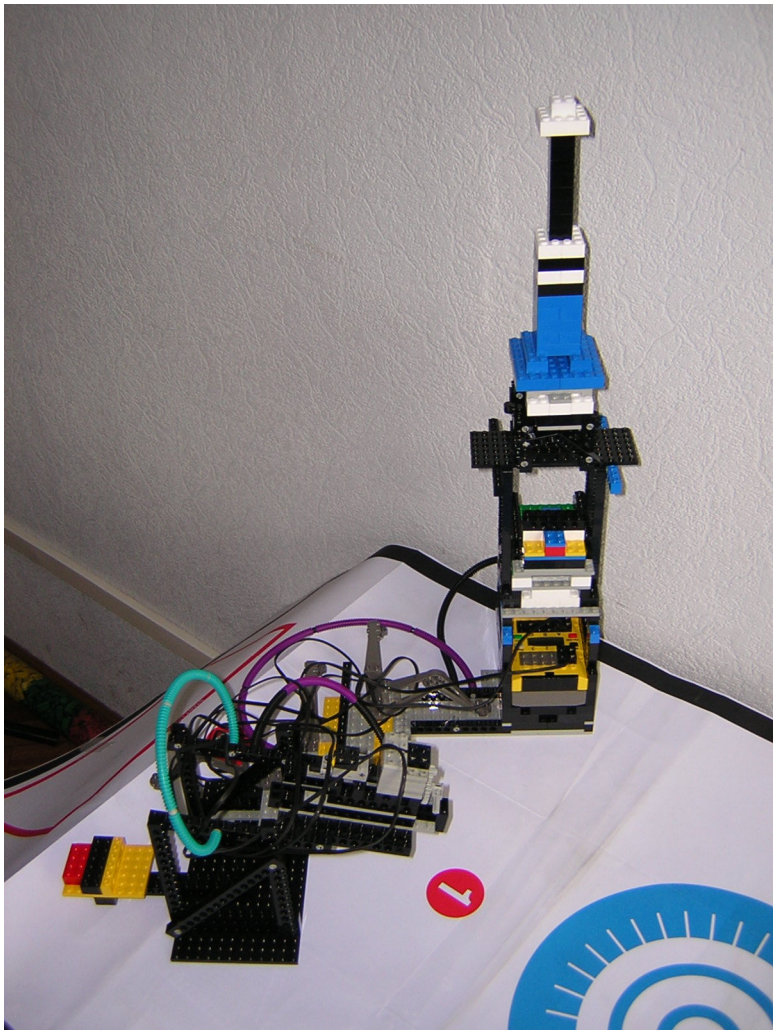

Рис. 43. Робот-поставщик

На рис. 44 показан микроконтроллер робота-поставщика, расположенный в основании башни.

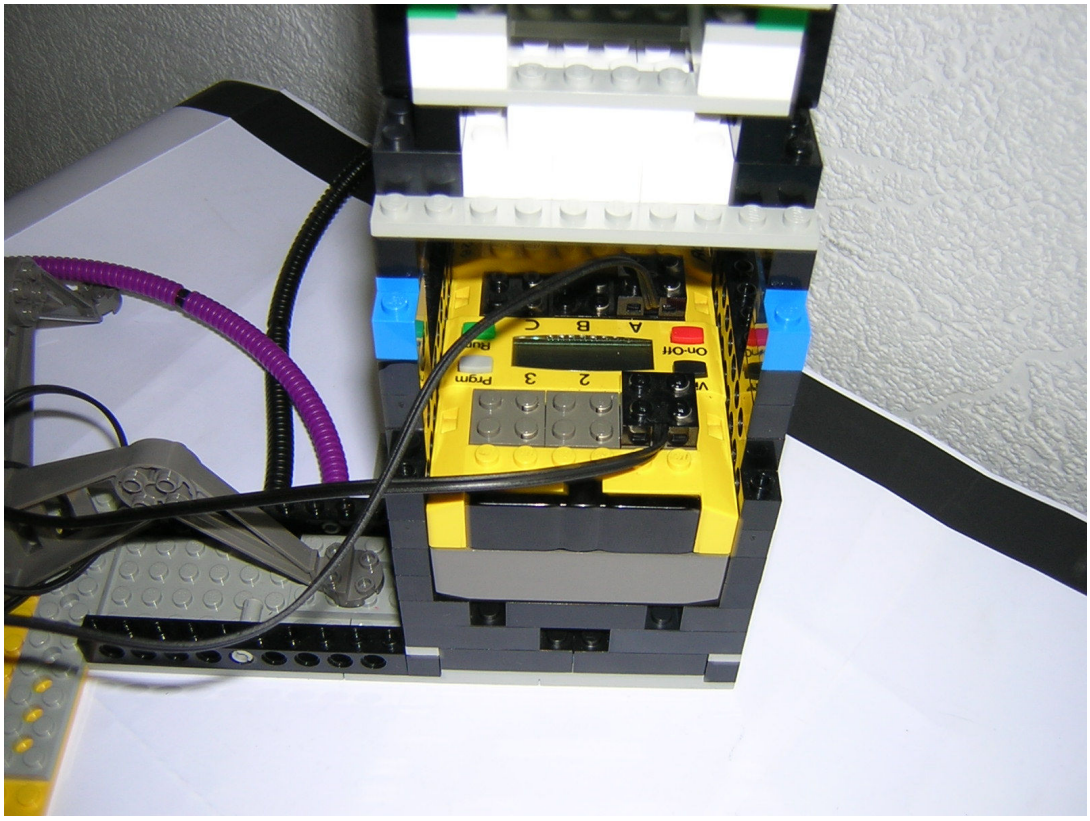

Рис. 44. Микроконтроллер робота-поставщика

На рис. 45 показано взаимодействие роботов при выдаче предметов.

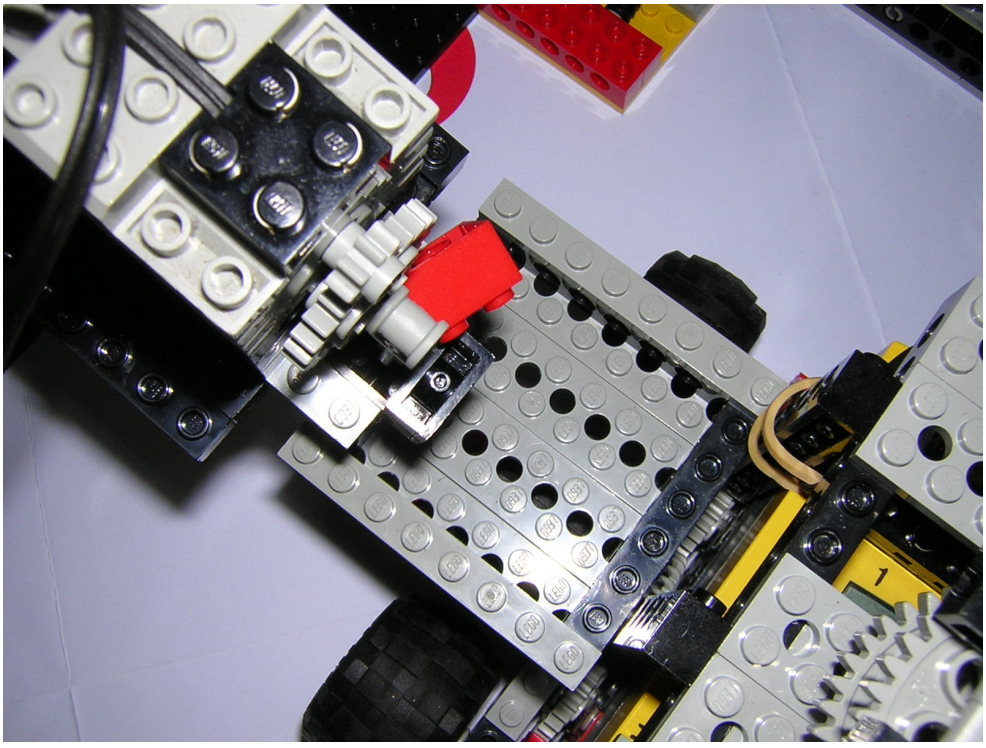

Рис. 45. Выдача предмета

На рис. 46 показано взаимодействие роботов после выдачи предмета.

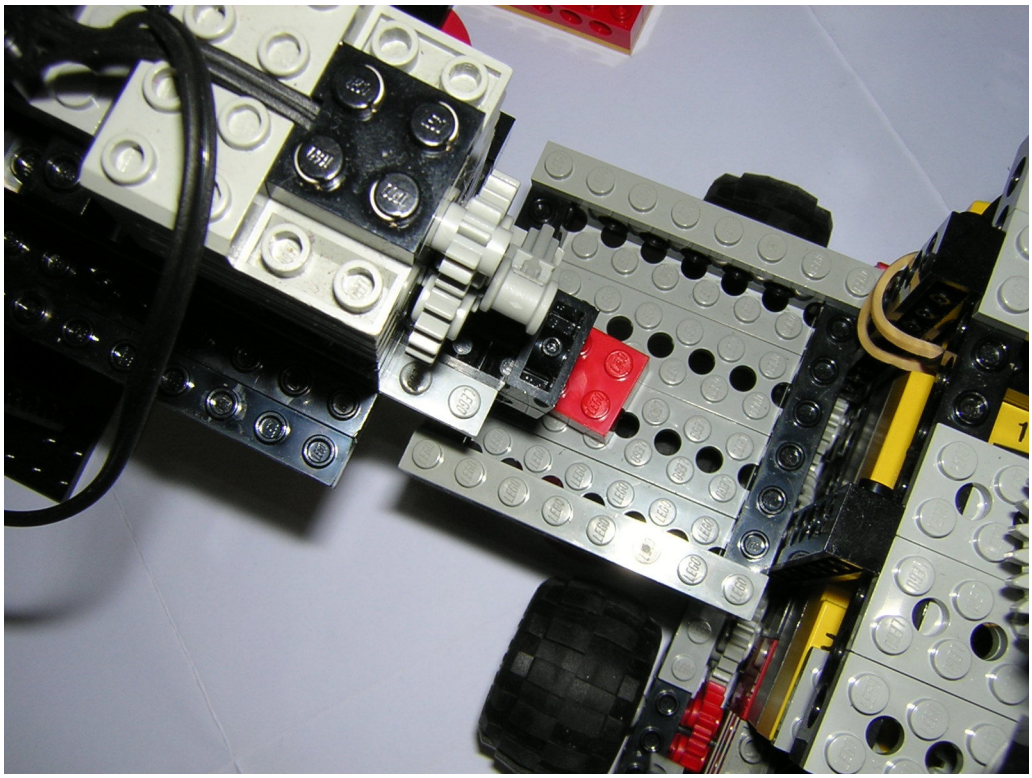

Рис. 46. После выдачи предмета

3.10. Управление роботом-поставщиком

3.10.1. Спецификация на управляющую систему

Управляющая система робота-поставщика при запуске должна сначала вывести информацию о напряжении на батареях, затем ожидать сообщения о начале выдачи от транспортного робота. После получения этого сообщения, система должна выдать указанное количество предметов. После их выдачи, система вновь должна ожидать дальнейшие сообщения от транспортного робота.

3.10.2. Автомат A0

3.10.2.1. Краткое описание

Автомат A0 ожидает сообщения от транспортного робота о количестве предметов к доставке. После получения этого сообщения автомат включает двигатель *B* (выходное воздействие z0_1) и не выключает, пока необходимое количество предметов не будет погружено. После этого автомат посылает сообщение транспортному роботу об окончании погрузки (выходное воздействие z0_4).

3.10.2.2. Перечень констант

Здесь перечислены все константы, определенные для робота-поставщика.

VOLTAGE_TIME – время, в течение которого на дисплей выводится напряжение на батареях. Измеряется в миллисекундах.

ES_POWER – значение мощности работы двигателя для выдачи предметов.

3.10.2.3. Перечень переменных

candyCount – количество предметов для выдачи. Начальное значение – ноль.

3.10.2.4. Перечень входных переменных

x0_0 – датчик касания нажат.

x0_1 – перменная candyCount больше нуля.

x0_2 – получено сообщение от транспортного робота о количестве предметов для выдачи.

x0_3 – таймер превысил значение VOLTAGE_TIME.

3.10.2.5. Перечень состояний y0

0 – Начало.

1 – Ожидание сообщения.

2 – Выдача предмета.

3 – Подготовка к выдаче.

4 – Вывод напряжения на дисплей.

3.10.2.6. Перечень выходных воздействий

z0_0 – вывести на дисплей напряжение на батареях.

z0_1 – запустить двигатель со значением мощности ES_POWER.

z0_2 – остановить двигатель.

z0_3 – уменьшить на единицу количество предметов, которые необходимо выдать.

z0_4 – послать сообщение транспортному роботу об окончании погрузки.

z0_5_0 – запретить сообщения от робота-поставщика.

z0_5_1 – разрешить сообщения от робота-поставщика.

z0_5_2 – обработать сообщение от робота-поставщика.

z0_6 – включить таймер.

z0_7 – остановить и обнулить таймер.

На рис. 47 приведена схема связей автомата A0, а на рис. 48 – его граф переходов.

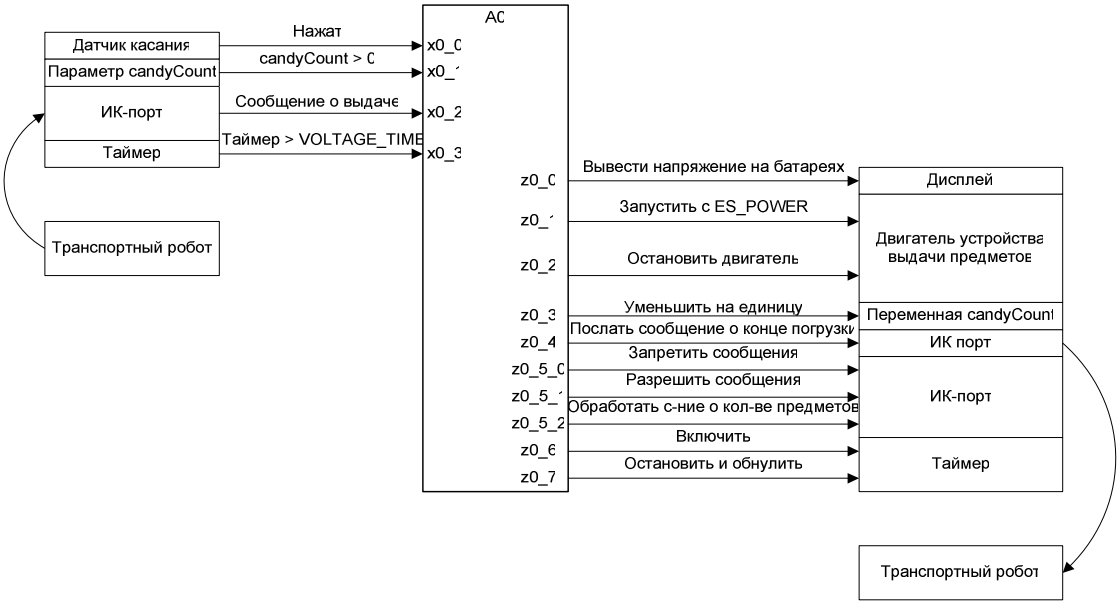

Рис. 47. Схема связей автомата A0. Управление роботом-поставщиком

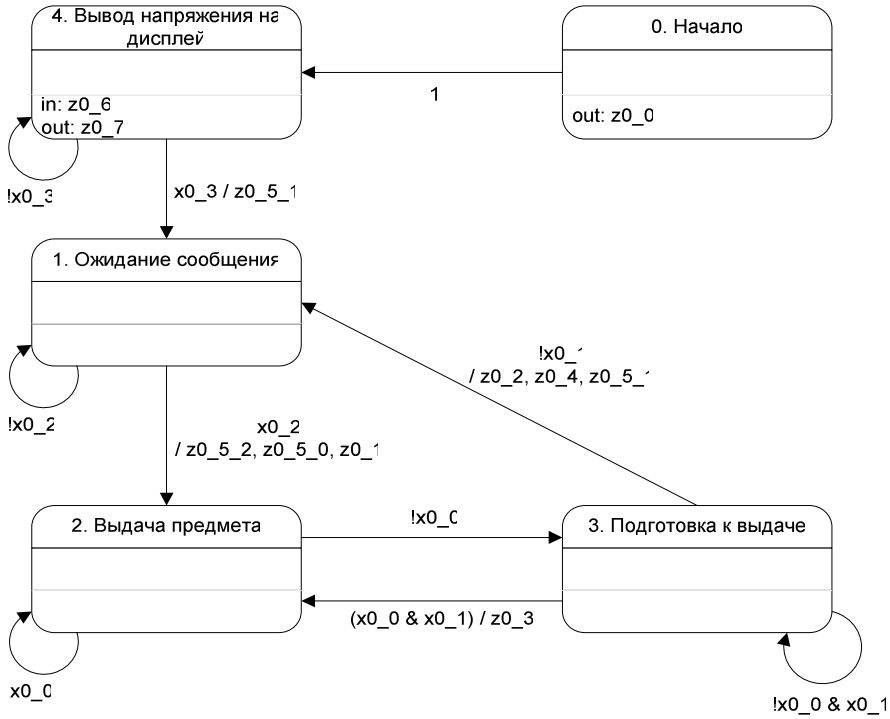

Рис. 48. Граф переходов автомата A0. Управление роботом-поставщиком

Заключение

В ходе работы получены следующие научные результаты.

1. Предложена классификация методов совместного использования объектно-ориентированного и автоматного подходов к программированию. Все методы неоднократно применены в соответствующих проектах, опубликованных на сайте http://is.ifmo.ru в разделе «Проекты», что подтверждает широкую область применимости и эффективность предложенных методов.
2. Разработана технология разработки программного обеспечения мультиагентной системы с использованием объектов и автоматов.
3. На основе этой технологии реализовано два примера мультиагентных систем – при реализации одной из них использовалась автоматическая генерация кода, а при реализации другой код по диаграммам автоматов создавался вручную.
4. Результаты работы внедрены в учебном процессе на кафедре «Компьютерные технологии» СПбГУ ИТМО при чтении лекций по курсу «Теория автоматов в программировании».

Литература

1. **Luger G.** Artificial Intelligence. Structures and strategies for Complex Problems Solving. Addison Wesley. 2002.
2. **Naumov L., Shalyto A.** Automata Theory for Multi-Agent Systems Implementation // Proceedings of Integration of Knowledge Intensive Multi-Agent Systems. MA, Boston. 2003. http://is.ifmo.ru, раздел «Наука».
3. **Russel S., Norvig P.** *Artificial Intelligence. A Modern Approach.* Prentice Hall. 2003. (Рассел С., Норвиг П. Искусственный интеллект. Современный подход. М.: Вильямс. 2006.)
4. **Skinner B.F.** *Science and Human behavior.* London: Macmillan, 1953.
5. **Putnam H.** Mind and machines /*Dimensions of Mind.* London: Macmillan, 1960, p.138-164.
6. **Rosenschein S.J.** Formal Theories of Knowledge in AI and Robostics *New Generation Computing.* 1985. 3(4), p.345-357.
7. **Brooks R.A.** A Robust Layered Control System for a Mobil Robot *IEEE Journal of Robotics and Automation.* 1986. 2, p.14-23.
8. **Regan P., Hamilton S.** NASA’s Mission Reliable. *Computer*. 2004, January. p. 59–68.
9. **Шалыто А.А.** SWITCH–технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
10. **Шалыто А.А., Туккель Н.И.** SWITCH-технология — автоматный подход к созданию программного обеспечения

«реактивных» систем // Программирование. 2001. № 5. http://is.ifmo.ru, раздел «Статьи».

11. **Шалыто А.А., Туккель Н.И.** Танки и автоматы // BYTE/Россия. 2003. № 2. http://is.ifmo.ru, раздел «Статьи».
12. **Shalyto A., Naumov L.** Foundation for Open Project Documentation // Linux Summit-2004. http://linuxsummit.org.
13. **Наумов А.С., Шалыто А.А.** Система управления лифтом. СПбГУ ИТМО, 2003. http://is.ifmo.ru, раздел «Проекты».
14. **Наумов Л.А., Шалыто А.А.** Искусство программирования лифта. Объектно-ориентированное программирование с явным выделением состояний // Информационно-управляющие системы. 2003. №6. http://is.ifmo.ru, раздел «Статьи».
15. **Корнеев Г.А., Шалыто А.А.** Реализация конечных автоматов с использованием объектно-ориентированного программирования // Труды X Всероссийской научно-методической конференции "Телематика-2003". 2003. Т.2. http://tm.ifmo.ru.
16. **Шопырин Д.Г., Шалыто А.А.** Объектно-ориентированный подход к автоматному программированию. СПбГУ ИТМО, 2003. http://is.ifmo.ru, раздел «Проекты».
17. **Фельдман П.И., Шалыто А.А.** Совместное использование объектного и автоматного подходов в программировании. СПбГУ ИТМО, 2004. http://is.ifmo.ru, раздел «Проекты».
18. **Заякин Е.А., Шалыто А.А.** Метод устранения повторных фрагментов кода при реализации конечных авто-

матов. СПбГУ ИТМО, 2003. http://is.ifmo.ru, раздел «Проекты».

19. **Канжелев С.Ю., Шалыто А.А.** Моделирование кнопочного телефона с использованием SWITCH-технологии. Вариант 2. СПбГУ ИТМО, 2004. http://is.ifmo.ru, раздел «Проекты».

20. **Гуров В.С., Нарвский А.С., Шалыто А.А.** Автоматизация проектирования событийных объектно-ориентированных программ с явным выделением состояний // Труды X Всероссийской научно-методической конференции "Телематика-2003". 2003. Т.1. http://tm.ifmo.ru.

21. **Гуров В.С., Мазин М.А., Шалыто А.А.** UniMod – программный пакет для разработки объектно-ориентированных приложений на основе автоматного подхода // Труды XI Всероссийской научно-методической конференции "Телематика-2004". 2004. Т.1. http://tm.ifmo.ru.

22. **Гуров В.С., Мазин М.А., Шалыто А.А. UML.** Switch-технология, Eclipse. Информационно-управляющие системы. 2004. №6. http://is.ifmo.ru, раздел «Статьи».

23. **Бондаренко К.А., Шалыто А.А.** Разработка XML – формата для описания внешнего вида видеопроигрывателя с использованием конечных автоматов. СПбГУ ИТМО. 2003. http://is.ifmo.ru, раздел «Проекты».

24. **Гуисов М.И., Кузнецов А.Б., Шалыто А.А.** Интеграция механизма обмена сообщениями в Switch-технологию. СПбГУ ИТМО. 2003. http://is.ifmo.ru, раздел «Проекты».

25. **Гуисов М.И., Кузнецов А.Б., Шалыто А.А.** Задача Д. Майхилла «Синхронизация цепи стрелков». Вариант 2.

СПбГУ ИТМО. 2003. http://is.ifmo.ru, раздел «Проекты».

26. Заочный тур всесибирской олимпиады 2005 по информатике. http://olimpic.nsu.ru/widesiberia/archive/wso6/2005/rus/1tour/problem/problem.html

27. **Ла Мот А., Ратклифф Д., Семинаторе М., Талер Д.** Секреты программирования игр. СПб.: Питер, 1995.

28. ***Baum D.*** Definitive Guide to Lego Mindstroms. NY: Appress, 2000.

29. ***Baum D., Gasperi M., Hempel R., Villa L.*** Extreme Mindstroms. An advanced guide to Lego Mindstorms. NY: Appress, 2000.

30. ***Киви Б.*** Сделай сам // Компьютерра. 2002. № 5.

31. ***Brown JP.*** Serious Lego. http://jpbrown.i8.com/cubesolver.html

32. ***Crosby M.*** Lego projects. http://www.mastincrosbie.com/mark/lego/lego.html

33. **Iversen T., Kristoffersen K., Larsen K., et al.** Model-Checking Real-Time Control Programs. Verifying Lego Mindstorms Systems Using UPPAAL // 12 Euromicro Conference on Real-Time Systems 2000.

34. **Alur R., Kannan S., Yannakakis M.** Communicating hierarchical state machines // *Proceedings of the 26th International Colloquium on Automata, Languages, and Programming*, LNCS 1644, 1999. pp. 169–178. http://cis.upenn.edu/~alur/Icalp99chsm.html

35. **Alur R.** Timed Automata. NATO-ASI 1998 Summer School on Verification of Digital and Hybrid Systems. http://www.cis.upenn.edu/~alur/Nato97.ps.gz

36. **Alur R., Dill D.** Automata-theoretic Verification of Real-Time Systems // *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, 1992. pp. 177-186.

37. **Laursen M., Madsen R., Mortensen S.** Verifying Distributed LEGO RCX Programs Using UPPAAL. http://citeseer.ist.psu.edu/laursen99verifying.html

38. **Буч Г., Рамбо Д., Якобсон И.** UML. Руководство пользователя, М.:ДМК. 2000.

39. *leJOS* infrared serial protocol. http://graphics.stanford.edu/~kekoa/rcx/#Protocol