МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ САМАРСКОЙ ОБЛАСТИ

государственное бюджетное образовательное учреждение
среднего профессионального образования
«Поволжский государственный колледж»

А.Е.Осоргин

AnyLogic 6
Лабораторный практикум

Самара 2012

Печатается по решению редакционно-издательского совета
ГБОУ СПО «Поволжский государственный колледж».

Рецензенты:

Осоргин А.Е.
AnyLogic 6. Лабораторный практикум/ А.Е.Осоргин. – Изд. 2-е, перераб. и доп. – Самара: ПГК, 2012.

Курс лабораторно-практических работ предназначен для студентов, изучающих дисциплину «Компьютерное моделирование» с использованием программы AnyLogic версии 6.7. Данный курс также может быть полезен преподавателям и аспирантам, желающим повысить свой профессиональный уровень в области имитационного моделирования.

Материал курса изложен в доступной форме, снабжен большим количеством иллюстраций и заданий для самоподготовки. Все лабораторные работы проверены автором на выполнимость и могут быть использованы для самостоятельного освоения мощного инструмента имитационного моделирования – AnyLogic.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время моделирование составляет неотъемлемую часть современной фундаментальной и прикладной науки. Поэтому при изучении данного курса представляется целесообразным использовать пакеты прикладных программ для математических и научных расчетов, ориентированных на широкие круги пользователей.

Одним из таких пакетов является AnyLogic, сочетающий в себе мощный инструмент объектно-ориентированного моделирования с интерфейсом визуального программирования.

Версия 6, на которой базируется данный практикум, существенно переработана, имеет более удобный и логичный интерфейс, значительно повышена устойчивость работы и улучшена диагностика ошибок. К сожалению, модели, разработанные в предыдущей 5 версии практически не совместимы с новой 6 версией AnyLogic, что и послужило одним из побудительных мотивов к написанию данного практикума.

В AnyLogic модели представляются визуально-иерархическими. Простой графический язык моделирования оперирует понятиями объектов и связей между ними. AnyLogic позволяет легко и быстро создавать агентные модели на профессиональном уровне, благодаря языковым конструкциям для задания поведения агентов, их взаимодействия, моделирования среды, а также богатейшим анимационным возможностям.

При разработке учебных моделей, не удается обойтись без написания некоторого объема программного кода. Доля программирования при построении учебных моделей составила менее 10% общих трудозатрат на разработку модели, что позволило студентам не тратить время на базовые функции, а сосредоточиться на логике имитационной модели.

Использование программы AnyLogic позволяет студенту за ограниченное количество учебных часов научиться строить достаточно сложные и интересные компьютерные модели и в конечном итоге глубже изучить курс компьютерного моделирования.

БЛАГОДАРНОСТИ

При написании практикума автор использовал идеи некоторых моделей и их описания, изложенные в книге Юрия Карпова «Имитационное моделирование систем. Введение в моделирование с AnyLogic 5», а также документацию к AnyLogic 6, разработанную сотрудниками фирмы XJ Technologies. Автор приносит искреннюю благодарность за предоставленные материалы и возможности.

ЗАДАНИЕ 1

ПЕРВАЯ МОДЕЛЬ НА ANYLOGIC

ЦЕЛИ ЗАНЯТИЯ

- знакомство с программой AnyLogic на примере модели Balls,
- освоение интерфейса программы,
- ознакомление с технологией имитационного моделирования, реализованной в программе AnyLogic.

ФОРМА ОРГАНИЗАЦИИ ЗАНЯТИЯ

Фронтальная.

СТУДЕНТ ДОЛЖЕН ЗНАТЬ

- понятия: модель, имитационное моделирование,
- основы одного из алгоритмических языков: Java, Pascal или Basic,

СТУДЕНТ ДОЛЖЕН УМЕТЬ

- пользоваться операционной системой Windows,
- ориентироваться в многооконном интерфейсе прикладных программ.

ОБЕСПЕЧЕННОСТЬ

- компьютер с установленной программой AnyLogic версии 6,
- модель Balls,
- настоящий курс лабораторно-практических работ.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

1.1. ИНТЕРФЕЙС ПРОГРАММЫ

Когда Вы первый раз запустите AnyLogic, Вы увидите начальную страницу. Начальная страница содержит краткое описание основных возможностей программы, ссылки на примеры моделей, поставляемые вместе с AnyLogic, а также ссылки на веб-сайт компании XJ Technologies (разработчика этого программного продукта) и на форму обратной связи с компанией.

Чтобы закрыть начальную страницу, щелкните мышью по кнопке ("X") 

в панели заголовка начальной страницы.

С помощью кнопки **Открыть модель** панели инструментов или команды **Файл** / **Открыть** в главном меню выберите файл Balls. Это несложная модель прыгающего мяча. На экране Вы увидите следующее окно - рис. 1.1.

AnyLogic при открытии проекта открывает несколько панелей: проекты, диаграмма, палитра, ошибки и свойства. Рисунок 1.1 показывает основные составляющие

пользовательского интерфейса. Рассмотрим их поочередно.

Панель **Проекты** обеспечивает навигацию по элементам открытых моделей. Поскольку модель организована иерархически, то она отображается в виде дерева: сама модель образует верхний уровень дерева; классы активных объектов и эксперименты образуют следующий уровень и т.д.

Панель **Проекты** по умолчанию прикреплена к левой части рабочей области AnyLogic.

Полужирным шрифтом в дереве выделяется тот элемент, редактор которого активен в текущий момент.

Если Вы внесете в модель какие-то изменения и не сохраните их, то такая модель будет сразу же выделена в дереве - к имени модели будет добавлена звездочка (*).

Вы можете разворачивать и сворачивать ветви дерева элементов модели с помощью кнопок + и -.

У каждого класса активного объекта и эксперимента есть своя диаграмма, которая редактируется в графическом редакторе.

На диаграммах Вы можете:

- Нарисовать презентацию с помощью фигур и элементов управления.
- Задать поведение активного объекта с помощью событий и диаграмм действий.
- Задать структуру класса, добавив вложенные объекты.
- Добавить на презентацию визуализирующие графики, диаграммы.

Панель **Палитра** содержит элементы, которые могут быть добавлены на диаграмму класса активного объекта или эксперимента.

По умолчанию она прикреплена к правому краю окна приложения.

Панель **Палитры** состоит из нескольких вкладок (палитр), каждая из которых содержит элементы, относящиеся к определенной задаче:

- **Основная** содержит основные элементы, с помощью которых Вы можете задать динамику модели, ее структуру и данные.
- **Системная динамика** содержит: элементы диаграммы потоков и накопителей, а также параметр, соединитель и табличную функцию.
- **Диаграмма состояний** содержит блоки диаграмм, позволяющих графически задавать поведение объекта.
- **Диаграмма действий** содержит блоки структурированных блок-схем, позволяющих задавать алгоритмы визуально.
- **Статистика** содержит элементы, используемые для сбора, анализа и отображения результатов моделирования.
- **Презентация** содержит элементы для рисования презентаций: примитивные фигуры, а также элементы управления, для придания презентации интерактивности.
- **Внешние данные** содержит инструменты для работы с базами данных и текстовыми файлами.
- **Картинки** содержит набор картинок наиболее часто моделируемых объектов:

человек, грузовик, фура, погрузчик, склад, завод и т. д.

Панель **Свойства** используется для просмотра и изменения свойств выбранного в данный момент элемента модели.

Панель **Свойства** содержит несколько вкладок. Каждая вкладка содержит элементы управления, такие как поля ввода, флажки, переключатели, кнопки и т.д., с помощью которых Вы можете просматривать и изменять свойства элементов модели. Число вкладок и их внешний вид зависит от типа выбранного элемента.

Вы можете, как Вам угодно перемещать панели в пределах окна AnyLogic. Для восстановления принятых по умолчанию настроек расположения панелей нужно в главном меню вызвать **Окно / Восстановить расположение панелей**.

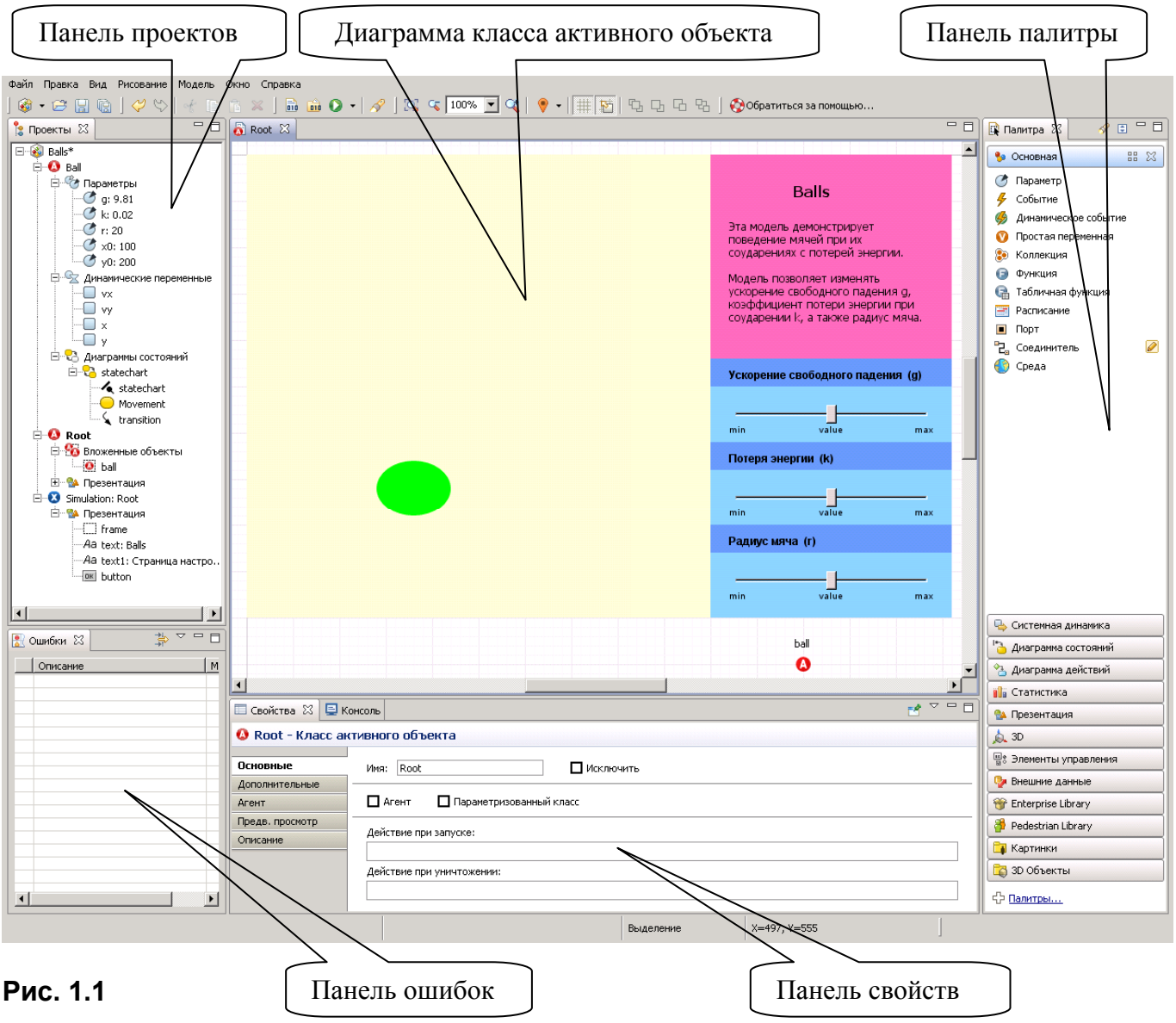


Рис. 1.1

На этапе компиляции модели AnyLogic производит проверку синтаксиса диаграмм, типов и параметров. Все обнаруженные на этапе компиляции и построения модели ошибки отображаются в панели **Ошибки**. Для каждой ошибки показывается ее описание и местоположение - имя элемента модели, при задании которого эта ошибка была допущена.

Активный объект является основным структурным элементом модели в AnyLogic. Активным объектом называется сущность, которая включает в себя данные, функции и

поведение как единое целое. Активный объект строится как класс, который может включать в качестве составных элементов экземпляры других классов активных объектов. Наш проект `Balls` включает два класса активных объектов: класс `Ball` и класс `Root`. На дереве проекта (рис. 1.2) как составные элементы класса `Ball` показаны `Параметры`, `Динамические переменные` и `Диаграммы состояний`, у класса `Root` составными его элементами показаны `Вложенные объекты` и `Презентация`.

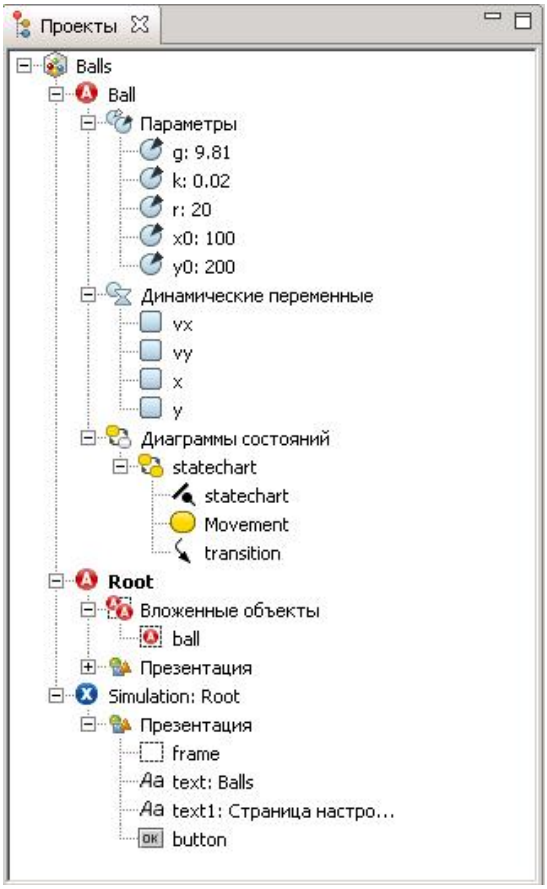


Рис. 1.2

Одна из ветвей в дереве проекта имеет название `Simulation`, это эксперимент, который может быть выполнен с моделью. С помощью экспериментов задаются конфигурационные настройки модели. AnyLogic поддерживает несколько типов экспериментов, каждый из которых соответствует своей задаче моделирования. AnyLogic поддерживает следующие типы экспериментов:

- Простой эксперимент
- Варьирование параметров
- Оптимизация
- Сравнение "прогонов"
- Монте-Карло
- Анализ чувствительности
- Калибровка
- Нестандартный

Эксперименты: **Сравнение "прогонов"**, **Монте-Карло**, **Анализ чувствительности, Калибровка** и **Нестандартный** доступны только в AnyLogic Professional.

1.1.1. ДИАГРАММА КЛАССА АКТИВНОГО ОБЪЕКТА

В нашей модели диаграмма класса активного объекта – мяча задается в окне с именем **Ball,** в котором содержатся его переменные (координаты мяча `x`, `y` и его скорости `Vx` и `Vy`), параметры (`g`, `r`, `k`, `x0` и `y0`) и диаграмма состояний с именем `statechart` (рис. 1.3).

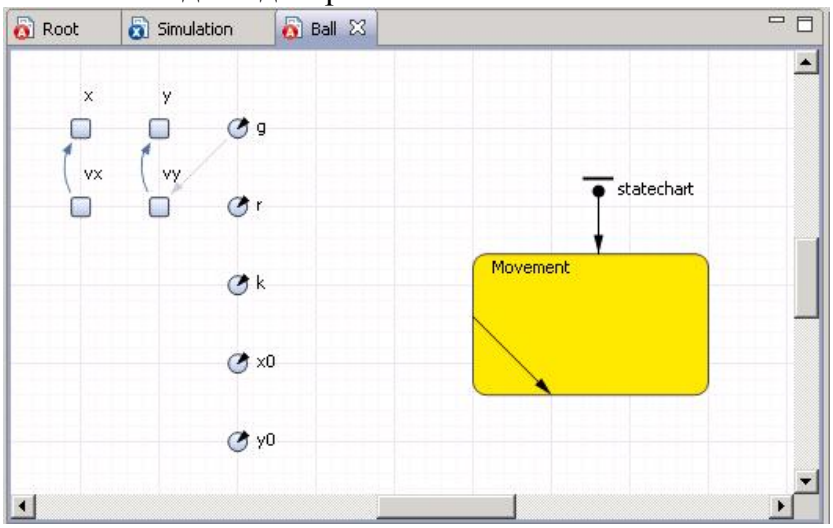


Рис. 1.3

AnyLogic отображает получившиеся зависимости между переменными с помощью тонких голубых стрелок.

Стрелка, направленная от переменной `Vx` к переменной `x` означает, что переменная

`Vx` упоминается в формуле переменной `x`. Стрелки зависимостей рисуются автоматически, всегда синхронизированы с формулами переменных и автоматически появляются или исчезают на диаграмме, как только переменная появится в формуле или будет исключена. Для улучшения наглядности можно редактировать внешний вид стрелок зависимостей, а именно изменять их цвет и радиус закругления.

Структура корневого активного объекта `Root` задана в окне с именем **Root**. В модели (рис. 1.1) активный объект `Root` содержит иконку с именем `ball` – один экземпляр активного объекта `Ball` и анимацию.

1.1.2. ПАНЕЛЬ СВОЙСТВ ОБЪЕКТОВ МОДЕЛИ

Каждый элемент модели обладает теми или иными свойствами или параметрами. При выделении какого-либо элемента в панели **Проекта** или **Диаграммы** внизу появляется окно свойств именно этого выделенного элемента. Окно **Свойства** (рис. 1.4) используется для просмотра и изменения свойств элементов.

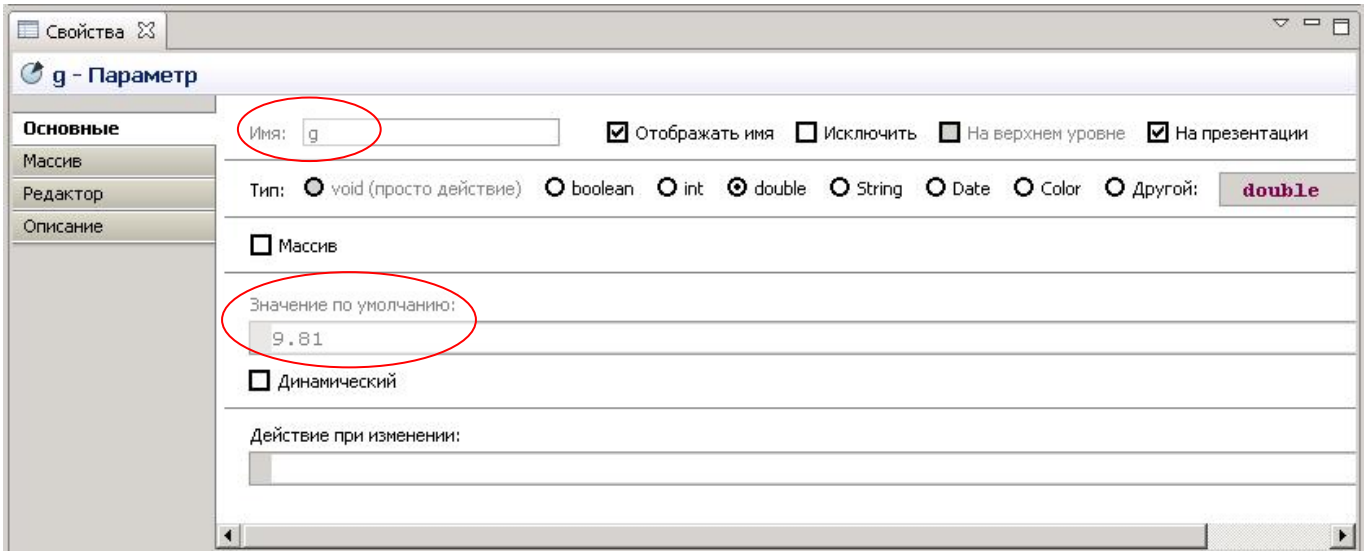


Рис. 1.4

Например, при выборе параметра `g` в окне редактора диаграммы объекта `Ball` внизу в панели свойств появятся 4 вкладки: **Общие, Массив, Редактор** и **Описание**. На вкладке **Общие** кроме имени этого объекта указаны его параметры: тип (`double`), значение по умолчанию (`9.81`) и отмечено, что следует показывать этот параметр на презентации и отображать его имя на диаграмме объекта `Ball`.

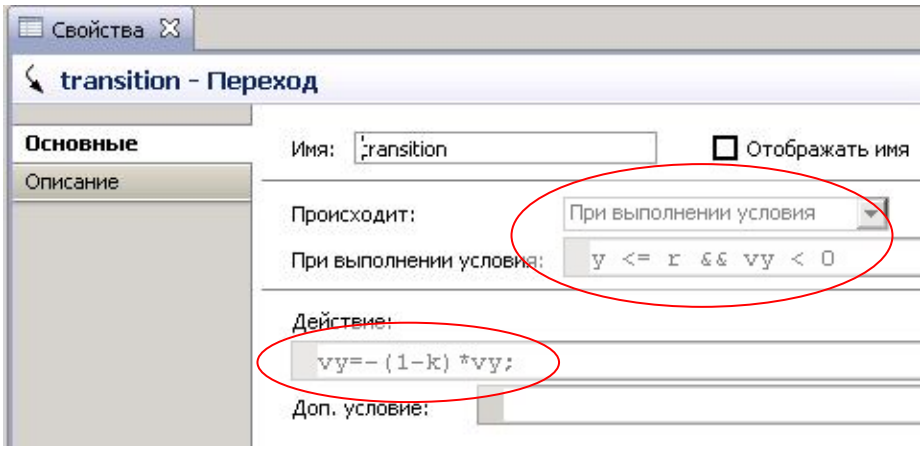
1.1.3. ПОВЕДЕНИЕ АКТИВНОГО ОБЪЕКТА

Поведение мяча представлено в стейтчарте, рис. 1.3, который состоит из Начала диаграммы состояний, одного состояния с именем `Movement` и одного перехода. Переход срабатывает при выполнении условия касания поверхности земли при движении мяча вниз, которое можно записать выражением:

```
y <= r && Vy < 0
```

Когда вертикальная координата `y` центра мяча с радиусом `r` будет отстоять от поверхности на `r` и при этом скорость мяча `Vy` будет направлена вниз, переход стейтчарта сработает.

Это выражение записано в поле **При выполнении условия** окна свойств перехода (рис. 1.5). При выполнении данного условия мяч отскакивает, то есть его скорость меняет свой знак на противоположный и уменьшается на долю `k`, моделирующую потерю энергии при отскоке. Это отражено в поле **Действие** окна свойств перехода.



Именно так, стейтчарты следят за событиями. При наступлении нужного события выполняется необходимое действие. И условие наступления события, и само действие, меняющее переменные модели, записываются операторами языка Ява.

Рис. 1.5

1.1.4. ПРЕЗЕНТАЦИЯ

В нашей модели строится двумерное динамическое представление, которое показывает, что происходит с моделью с течением времени. Для модели `Balls` в диаграмме `Root` построено изображение мяча в виде закрашенной окружности.

Презентация позволяет более наглядно представить динамику моделируемой системы, т. е. поведение ее во времени. Для презентации геометрических фигур, например окружности, их параметры – координаты, радиус, цвет и т. п., связываются с переменными и параметрами модели. Изменение переменных модели во времени приводит к изменению во времени внешнего вида геометрических фигур, что позволяет наглядно представить динамику моделируемой системы.

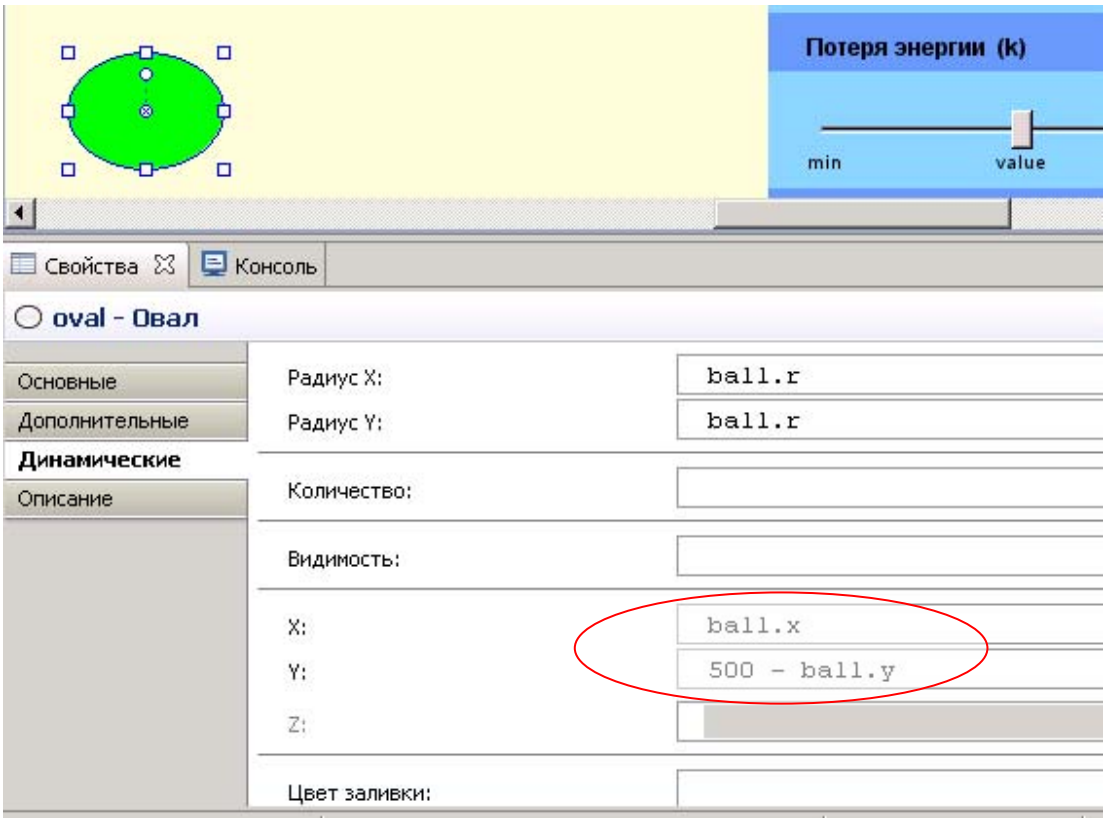


Рис. 1.6

На рис. 1.6 показано, что в окне свойств зеленого круга координаты `X` и `Y` его центра и

радиус `r` имеют динамические значения, которые связаны с переменными `x`, `y` и `r` экземпляра `ball` класса активного объекта `Ball`. Таким образом, изменение данных переменных будет вызывать перемещение центра и изменение радиуса окружности, моделирующей мяч, при работе модели.

Щелкните мышью на различных анимационных объектах в панели диаграммы `Root`. Вы увидите, что для различных элементов модели окно свойств содержит различные параметры, характеризующие именно данный элемент. Например, щелкните мышью в презентации на слайдере. В окне свойств на вкладке **Основные** будет представлена информация о связи слайдера с конкретным параметром модели и геометрическими параметрами самого слайдера.

Обратите внимание, что в качестве координаты Y мы взяли значение переменной `y` со знаком минус и смещением 500 единиц. Это сделано потому, что ось Y на презентации направлена вниз, а не вверх, и знак минус позволяет нам перевернуть ось Y и опустить ее до нужного уровня.

Итак, модель `Balls` построена и готова к запуску.

1.2. РЕЖИМ ВЫПОЛНЕНИЯ МОДЕЛИ

При запуске модели можно выполнять различные эксперименты с моделью. Рассмотрим основные средства управления экспериментом.

1.2.1. ЗАПУСК МОДЕЛИ

Запуск модели производится кнопкой **Запустить** на панели инструментов. При запуске эксперимента AnyLogic автоматически производит построение запускаемой модели. Поэтому в случае обнаружения ошибки Вам будет показано сообщение об ошибке, а более подробная информация будет выведена в панель Консоль.

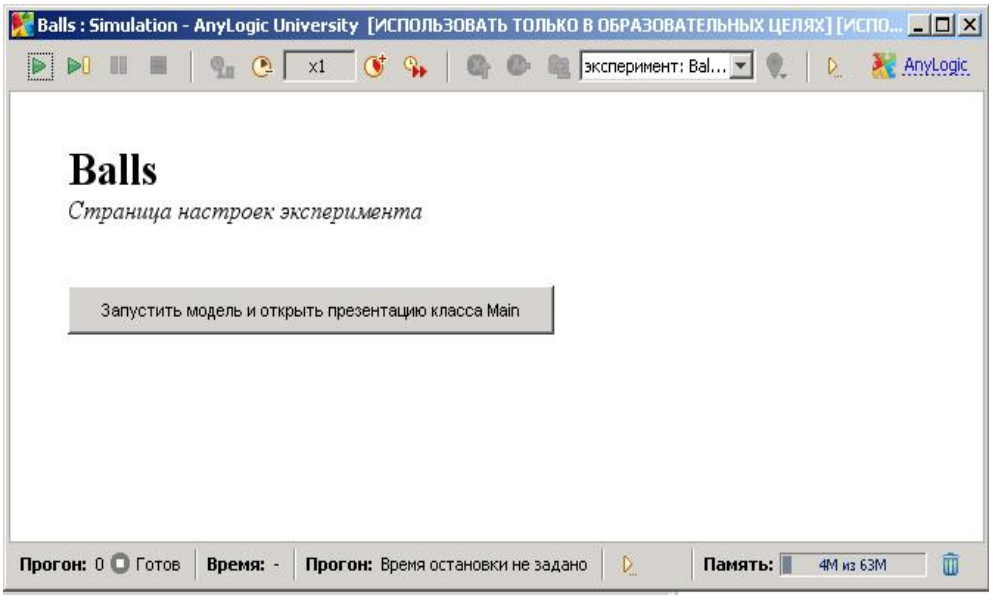


Рис. 1.7

При отсутствии ошибок откроется окно презентации эксперимента, рис. 1.7., которое содержит кнопку **Запустить модель и открыть презентацию класса Main**.

Когда Вы запустите модель с помощью этой кнопки, откроется окно презентации либо эксперимента, либо одного из активных объектов запущенной модели, рис. 1.8. На презентации будут видны все элементы, в свойствах которых были установлены флажки **На презентации**.

При проведении компьютерных экспериментов можно использовать все кнопки, показанные в верхней части окна рис. 1.8:

- запуск или продолжение моделирования
- запуск выполнения модели по шагам
- пауза
- останов модели и возврат в окно презентации эксперимента

В нижней части окна виден статус модели (пауза или выполнение, № прогона и др. информация).

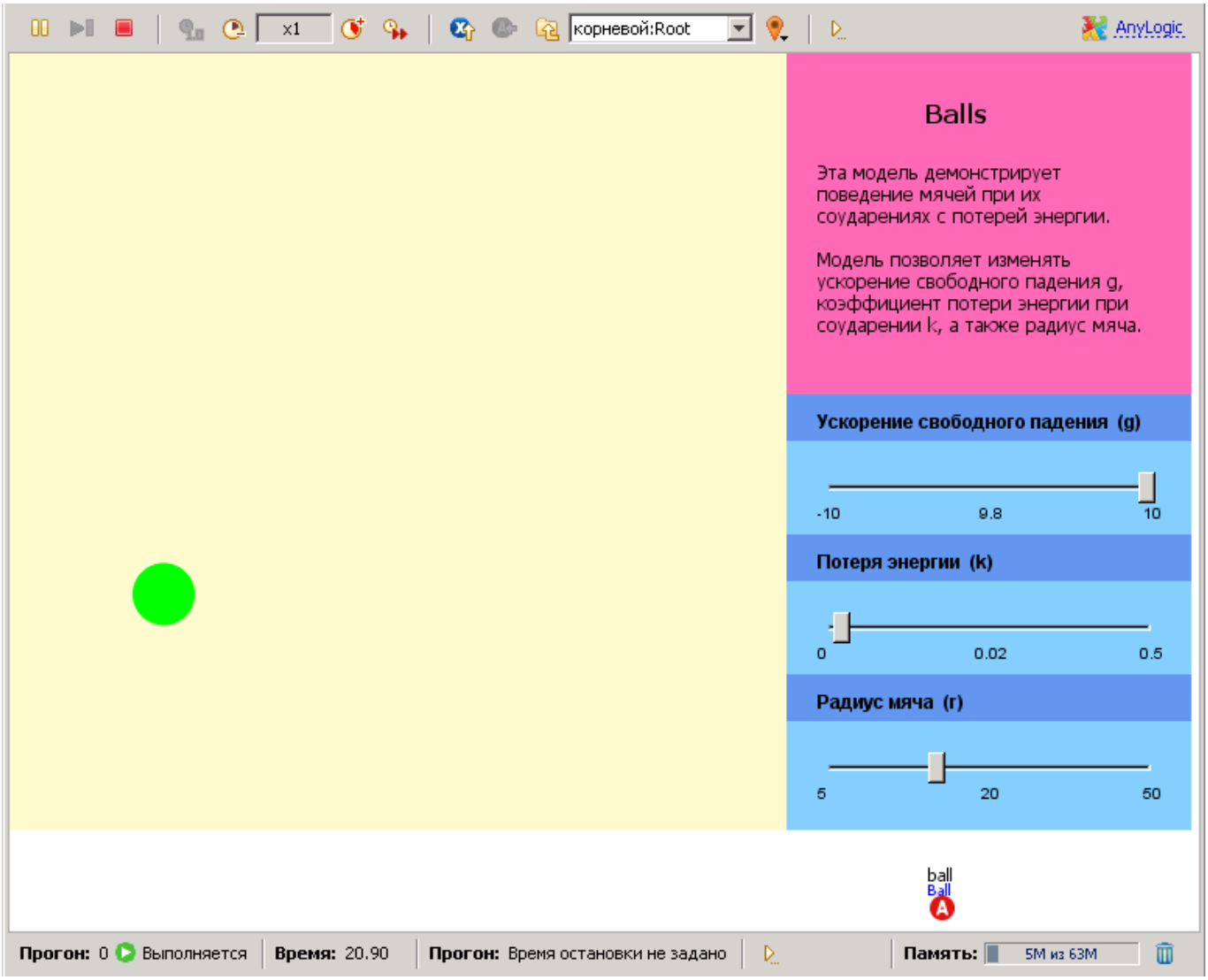


Рис. 1.8

1.2.2. ЭКСПЕРИМЕНТЫ С МОДЕЛЬЮ

На рис. 1.8. кроме движущегося изображения мяча видны текстовый комментарий и "бегунки" или "слайдеры" – подвижные указатели для изменения параметров модели во время ее выполнения. Перемещая бегунки слайдеров, можно менять три параметра – ускорение свободного падения, долю потери скорости мяча при каждом отскоке и радиус мяча. Изменение параметров позволяет исследовать поведение модели в различных условиях – это и есть компьютерный эксперимент.

Проведите несколько экспериментов с моделью, изменяя параметры модели.

1.2.3. УПРАВЛЕНИЕ СКОРОСТЬЮ ВЫПОЛНЕНИЯ МОДЕЛИ И ИЗОБРАЖЕНИЕМ

Модель AnyLogic может выполняться либо в режиме виртуального, либо в режиме реального времени.

В режиме виртуального времени модель выполняется без привязки к физическому времени – она просто выполняется настолько быстро, насколько это возможно. Этот режим лучше всего подходит в том случае, когда требуется моделировать работу системы в течение достаточно длительного периода времени.

В режиме реального времени задается связь модельного времени с физическим, то есть задается количество единиц модельного времени, выполняемых в одну секунду. Это часто требуется, когда Вы хотите, чтобы презентация модели отображалась с той же скоростью, что и в реальной жизни.

Переключение между виртуальным и реальным временем исполнения модели осуществляется кнопкой **Виртуальное/реальное время** панели управления окна презентации, а уменьшение масштаба времени выполняется с помощью двух кнопок **Замедлить**, **Ускорить** и расположенного между ними поля. Это поле указывает коэффициент ускорения модельного времени относительно физического (здесь 1x означает единичный коэффициент ускорения).

Выполните несколько экспериментов с различными скоростями выполнения модели, используя кнопки управления.

1.2.4. НАСТРОЙКА ПРЕЗЕНТАЦИИ

Для повышения качества изображения можно использовать настройки презентации:

Обновить вид	Обновляет содержимое окна презентации.
Адаптивная частота отрисовки	Устанавливает режим адаптивной частоты отрисовки.
Уменьшить частоту отрисовки	Уменьшает частоту отрисовки кадров.
Приоритет (частота отрисовки: скорость выполнения)	Отображает текущее значение частоты отрисовки кадров. Вы можете изменить это значение с помощью кнопок **Уменьшить частоту отрисовки** и **Увеличить частоту отрисовки**.
Увеличить частоту отрисовки	Увеличивает частоту отрисовки кадров.
Отрисовка со сглаживанием	Включает/выключает режим сглаживания. Кнопка отображается нажатой, если режим сглаживания выбран.

1.2.5. НАВИГАЦИЯ ПО МОДЕЛИ

Расположенный в панели управления окна презентации выпадающий список **Навигация** открывает организованный в виде дерева список объектов модели,

обеспечивая простую навигацию по модели и быстрый доступ к любым ее объектам, рис.1.9. Корнем дерева объектов является корневой объект запущенного эксперимента. Если структура модели меняется во время выполнения модели, то эти изменения тут же отображаются в дереве объектов модели.

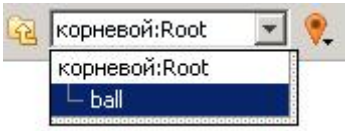


Рис. 1.9

Если выбрать объект `Ball`, то мы увидим его структурную диаграмму с динамично изменяющимися значениями переменных `Y` и `Vy`. AnyLogic поддерживает различные инструменты для сбора, отображения и анализа данных во время выполнения модели. Простейшим способом просмотра текущего значения и истории изменения значений переменной или параметра во время выполнения модели является использование окна **инспекта**. Щелкните мышью по значку переменной в окне презентации. Будет отображено небольшое желтое окно - это и есть окно инспекта, рис. 1.10.

Установите подходящий размер окна путем перетаскивания мышью нижнего правого угла окна инспекта. Если нужно, переместите окно, перетаскивая его мышью за панель названия окна.

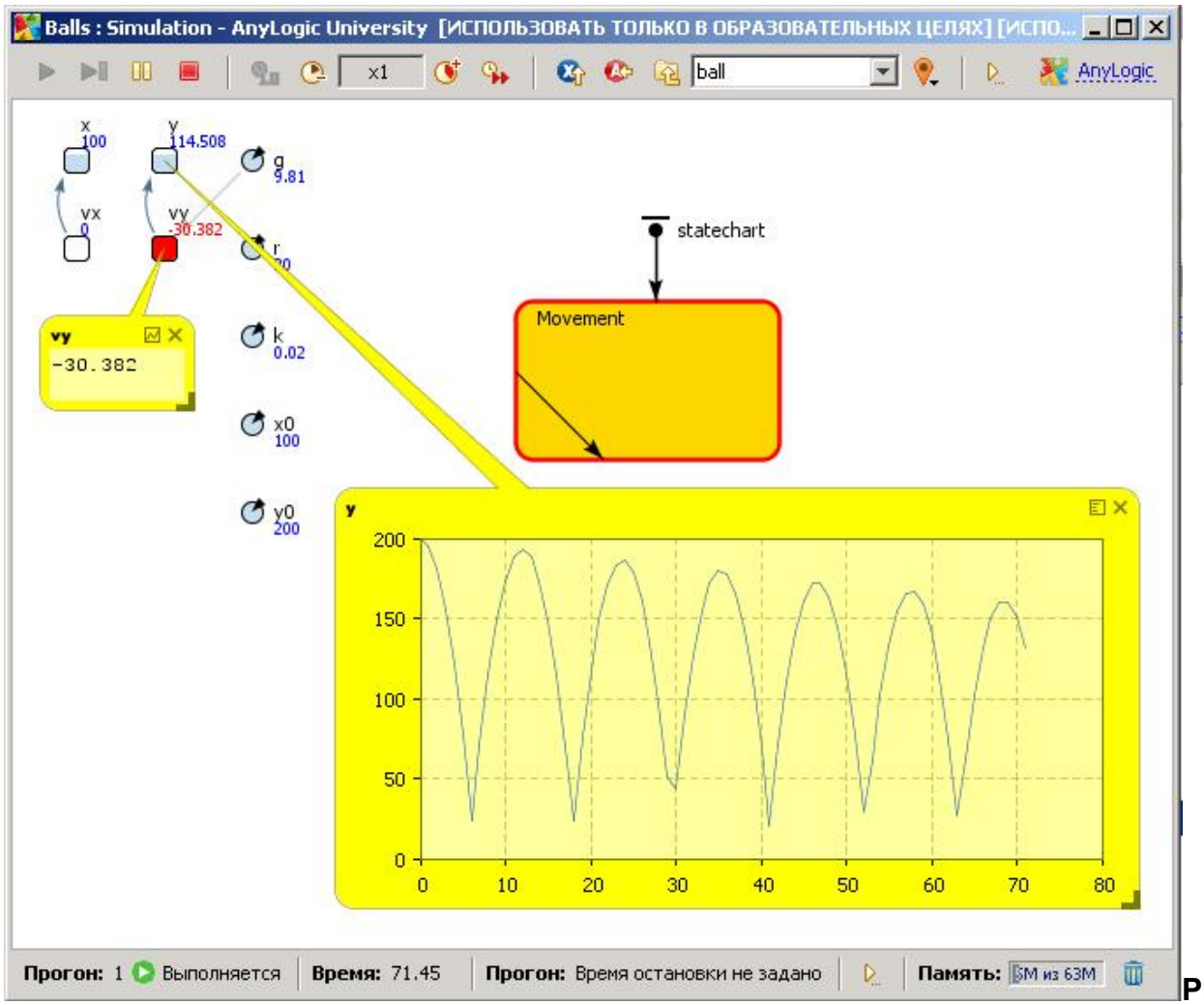


Рис. 1.10

Чтобы переключить окно инспекта в режим графика щелкните мышью по символу графика  , находящегося в правом верхнем углу окна инспекта.

1.3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чем назначение стейтчартов?
2. Какие классы активных объектов включает проект `balls`?
3. Продемонстрируйте на примере модели `balls` что такое структурная диаграмма.
4. Каким образом в модели `balls` реализован отскок мяча?
5. Для чего применяются динамические значения параметров в окне презентации?
6. Как запустить модель на выполнение?
7. В чем назначение слайдеров?
8. Что такое виртуальное время?
9. Как изменить цвет мяча на презентации?
10. Как переключиться из режима виртуального времени в реальное?
11. Как изменить скорость выполнения модели?
12. Как изменяются режимы отрисовки изображения?
13. В чем смысл эксперимента в программе AnyLogic?
14. Какие типы экспериментов поддерживаются программой AnyLogic?
15. Как изменить текущие значения переменных и параметров модели при ее выполнении?
16. Как показать график изменения переменной модели?
17. Как в модели balls задается потеря энергии при отскоке?

ЗАДАНИЕ 2

ДОРАБОТКА МОДЕЛИ BALLS

ЦЕЛИ ЗАНЯТИЯ

- продолжение знакомства с программой AnyLogic на примере модели Balls,
- освоение методов редактирования модели,
- ознакомление с технологией обработки событий (стейтчартами).

ФОРМА ОРГАНИЗАЦИИ ЗАНЯТИЯ

Фронтальная.

СТУДЕНТ ДОЛЖЕН ЗНАТЬ

- понятия: модель, имитационное моделирование, стейтчарт.
- основы алгоритмического языка Java

СТУДЕНТ ДОЛЖЕН УМЕТЬ

- пользоваться операционной системой Windows,
- ориентироваться в интерфейсе программы AnyLogic.

ОБЕСПЕЧЕННОСТЬ

- компьютер с установленной программой AnyLogic версии 6,
- модель Balls,
- настоящий курс лабораторно-практических работ.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

Выполним ряд упражнений с моделью `Balls`, которые дадут представление о средствах разработки моделей в AnyLogic.

2.1. ИЗМЕНЕНИЕ ЦВЕТА МЯЧА ПРИ ОТСКОКЕ

Дополним анимационное представление мяча динамическим цветом, так, чтобы при отскоке его цвет на несколько секунд изменялся на красный. Для этого нужно запомнить момент отскока и установить красный цвет окружности в презентации на небольшой интервал времени, следующий за этим моментом.

Создайте переменную `t0`, которая будет фиксировать момент отскока. Для этого перейдите на диаграмму класса активного объекта `Ball`, затем в панели **Палитра** откройте вкладку **Системная динамика** и перенесите иконку (**Параметр**) на диаграмму. В поле **Имя** открывшегося окна свойств этого параметра введите `t0`, а в поле **Значение по умолчанию** введите `-1` (рис. 2.1). Для того чтобы параметр `t0` фиксировал момент отскока, нужно значение текущего времени в модели при выполнении условия "отскок" запомнить в этом параметре. За наступлением данного условия следит стейтчарт,

поэтому выделите мышью переход стейтчарта (рис. 2.2), и в поле **Действие** добавьте выражение: `t0 = time();`

При каждом вызове функция `time()` возвращает текущее значение модельного времени.

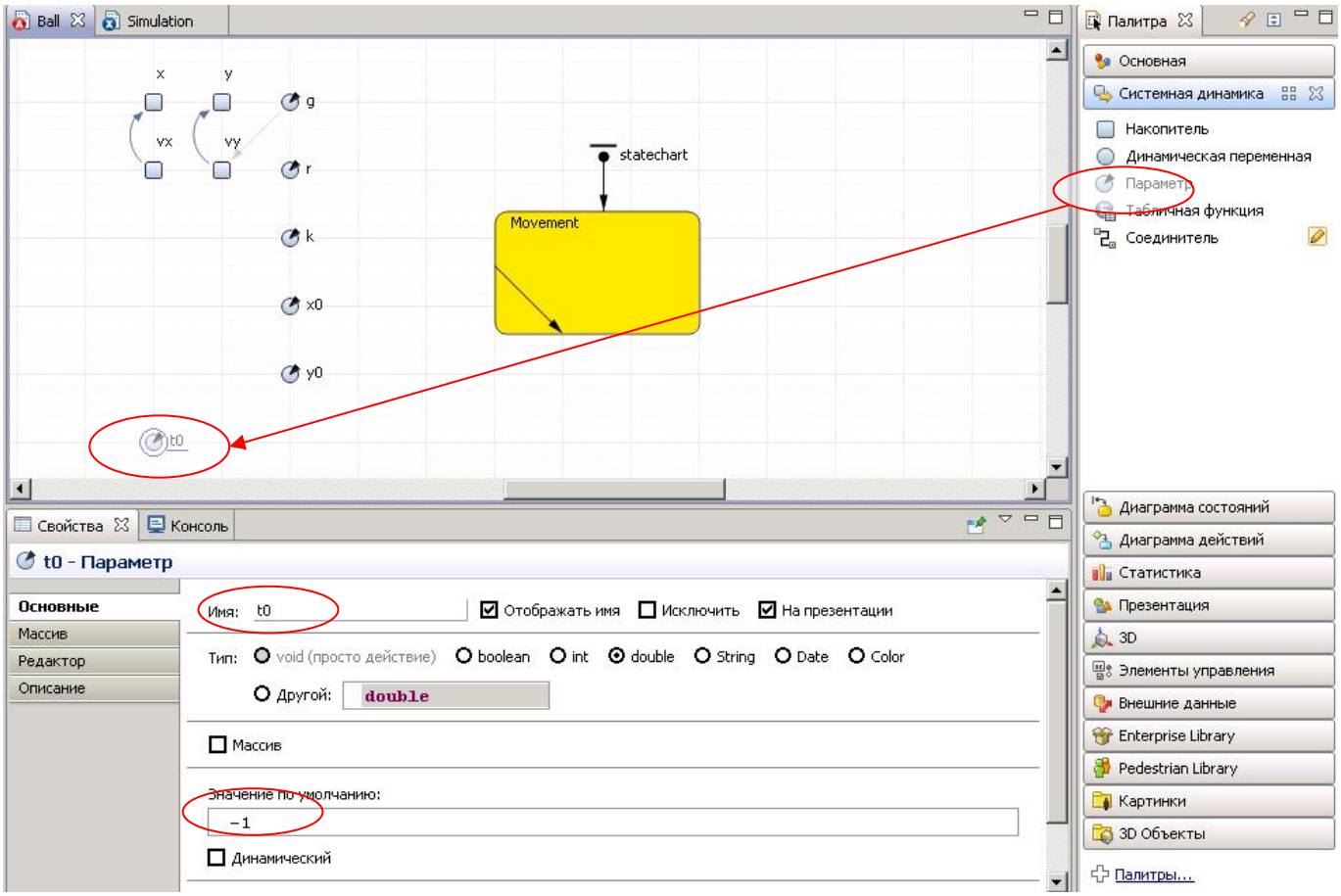
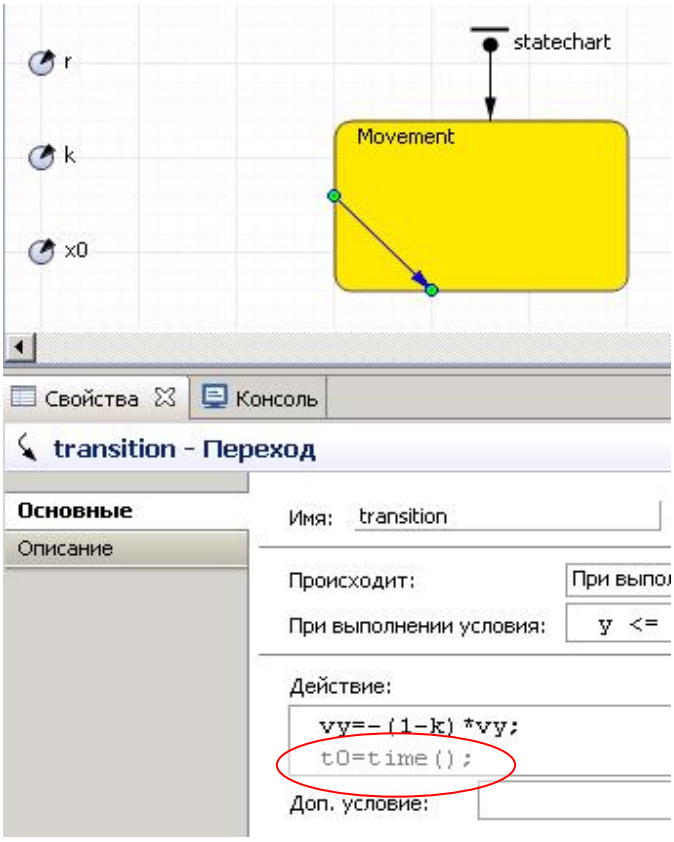


Рис. 2.1



Параметр `t0` имеет начальное значение -1, а при работе модели хранит значение момента времени последнего отскока. Для того чтобы каждый раз при отскоке мяча его цвет изменялся на красный (в течение 0.3 сек.), нужно перейти на диаграмму класса `Root`, выделить зеленый овал (мяч), в панели свойств этого овала открыть вкладку **Динамические** и установить в поле **Цвет заливки** динамическое значение цвета (рис. 2.3):

```
time()<ball.t0+0.3? red: lime
```

Это условное выражение устанавливает цвет заливки изображения мяча `ball` красным в течение 0.3 сек. после каждого отскока.

Рис. 2.2

2.2. МОДЕЛЬ С ДВУМЯ МЯЧАМИ

Добавим в модель второй мяч. Перейдите на диаграмму класса активного объекта `Root` и перенесите мышью на него еще один экземпляр мяча. Появившийся объект автоматически получит имя `ball1` (рис. 2.4). При этом в окне свойств нового экземпляра мяча мы увидим те же значения параметров мяча, которые были определены для активного объекта `Ball`.

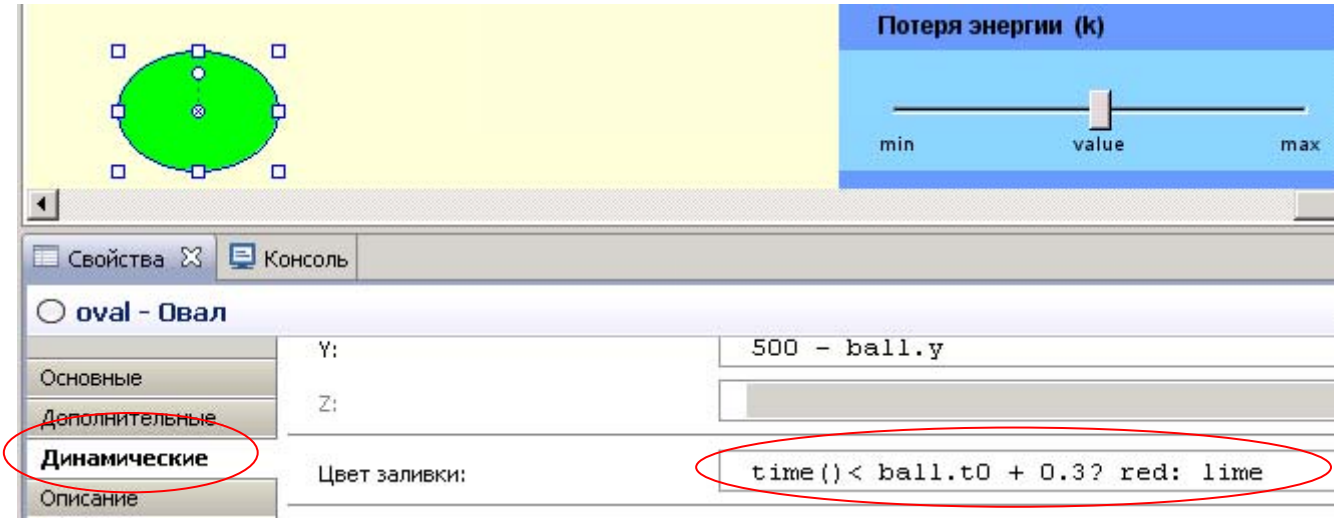


Рис. 2.3

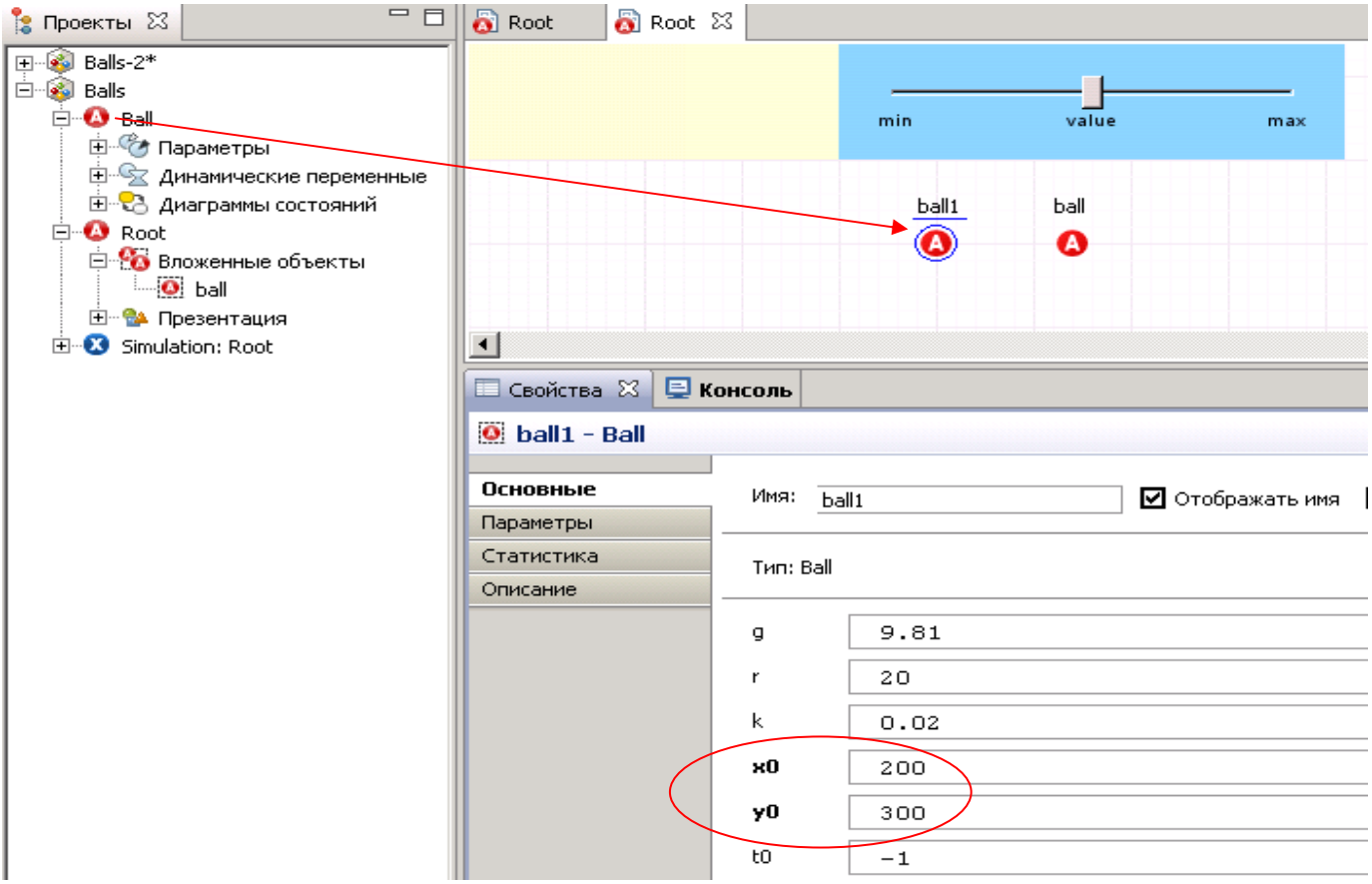


Рис. 2.4

Установите начальные значения `x0` и `y0` нового мяча равными 200 и 300 соответственно.

Чтобы на презентации показать движение второго мяча, продублируйте изображение

первого мяча с помощью клавиш <Ctrl+C> и <Ctrl+V>. Параметры нового изображения овала (координаты и цвета) связаны с характеристиками объекта `ball`. Их нужно связать с новым объектом – шаром с именем `Ball1`. То есть, вместо `Ball.x, Ball.y` и `Ball.t0` в соответствующих полях нужно записать `Ball1.x, Ball1.y` и `Ball1.t0` (рис. 2.5). А для значения радиусов **Радиус X** и **Радиус Y** нужно установить `Ball1.r` вместо `Ball.r`.

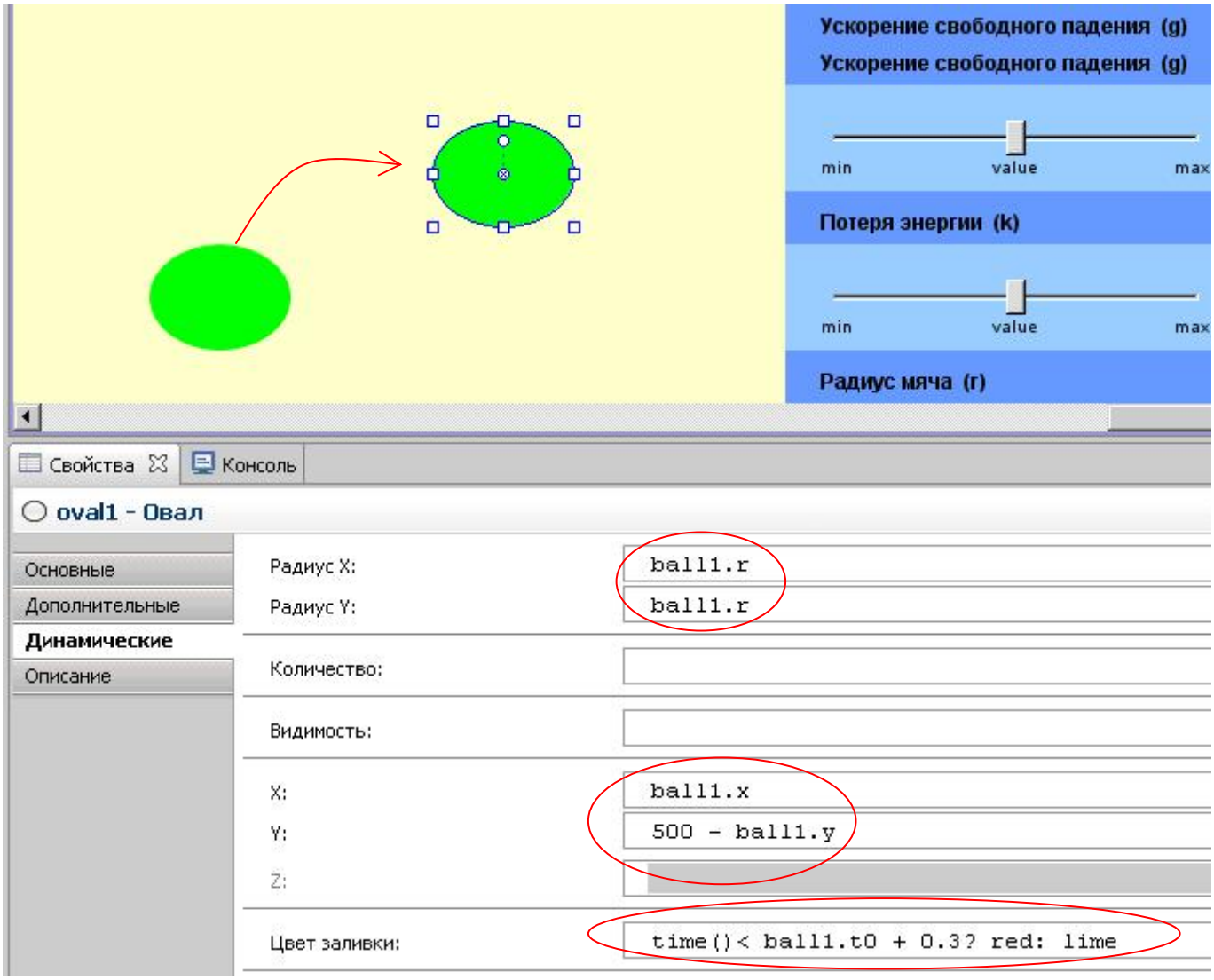


Рис. 2.5

Теперь при запуске модели будут имитироваться независимые движения двух шаров.

Продемонстрируйте свою модель преподавателю.

2.3. ПРОИЗВОЛЬНЫЕ ПЕРЕМЕЩЕНИЯ МЯЧА

В нашей модели мячи движутся строго вертикально, отскакивая от горизонтальной поверхности. Это происходит потому, что начальная скорость мячей по координате `x` равна 0. Если мы изменим начальные скорости мячей по `x`, нам необходимо будет описать поведение мячей при столкновении с вертикальными стенками и потолком.

Зададим случайные начальные значения скоростей `Vx` и `Vy`. Для этого перейдите на диаграмму класса активного объекта `Ball`, выделите переменную `Vx` и в поле **Начальное значение** этой переменной замените значение 0 на значение `uniform(-100, 100)`. При

этом у различных экземпляров активного объекта `Ball` начальная скорость по координате `x` будет задана случайно из диапазона (-100, +100) метров в секунду. То же самое сделайте для переменной `Vy`.

Для моделирования отскока мяча от потолка нужно на переходе стейтчарта изменить условие столкновения мяча с поверхностью. Мячи двигаются в пространстве, размером 500x500 метров. В поле **При выполнении условия** панели свойств перехода стейтчарта активного объекта `Ball` выражение:

```
y <= r && vy < 0
```

измените на:

```
y <= r && vy < 0 || y >= 500 - r && vy > 0
```

При этом, выполняемое действие должно остаться без изменения, а именно: смена направления скорости `Vy` с частичной ее потерей.

Для того чтобы мяч отскакивал от вертикальных стен, нужно записать это условие в стейтчарте добавлением дополнительного перехода. Откройте палитру **Диаграмма состояний** и сделайте двойной щелчок мышью по иконке **Переход**, включив, тем самым, режим рисования. Нарисуйте переход внутри состояния `Movement`, как показано на рис. 2.6.

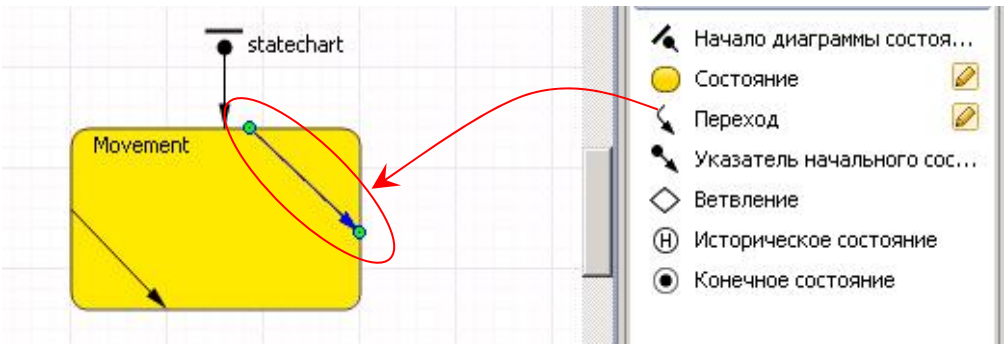


Рис. 2.6

В окне свойств этого перехода в поле **Происходит** нужно выбрать вариант `При выполнении условия`, в поле **При выполнении условия** следует записать условие касания мяча о вертикальную стенку:

```
x <= r && vx < 0 || x >= 500 - r && vx > 0
```

а в поле **Действие** записать изменение направления составляющей `Vx` скорости мяча и запомнить момент времени, когда произошло касание стенки для последующего изменения цвета мяча, рис. 2.7:

```
vx = -(1 - k) * vx;
t0 = time();
```

Запустите модель. Поэкспериментируйте с ней, используя слайдеры. Продемонстрируйте модель преподавателю.

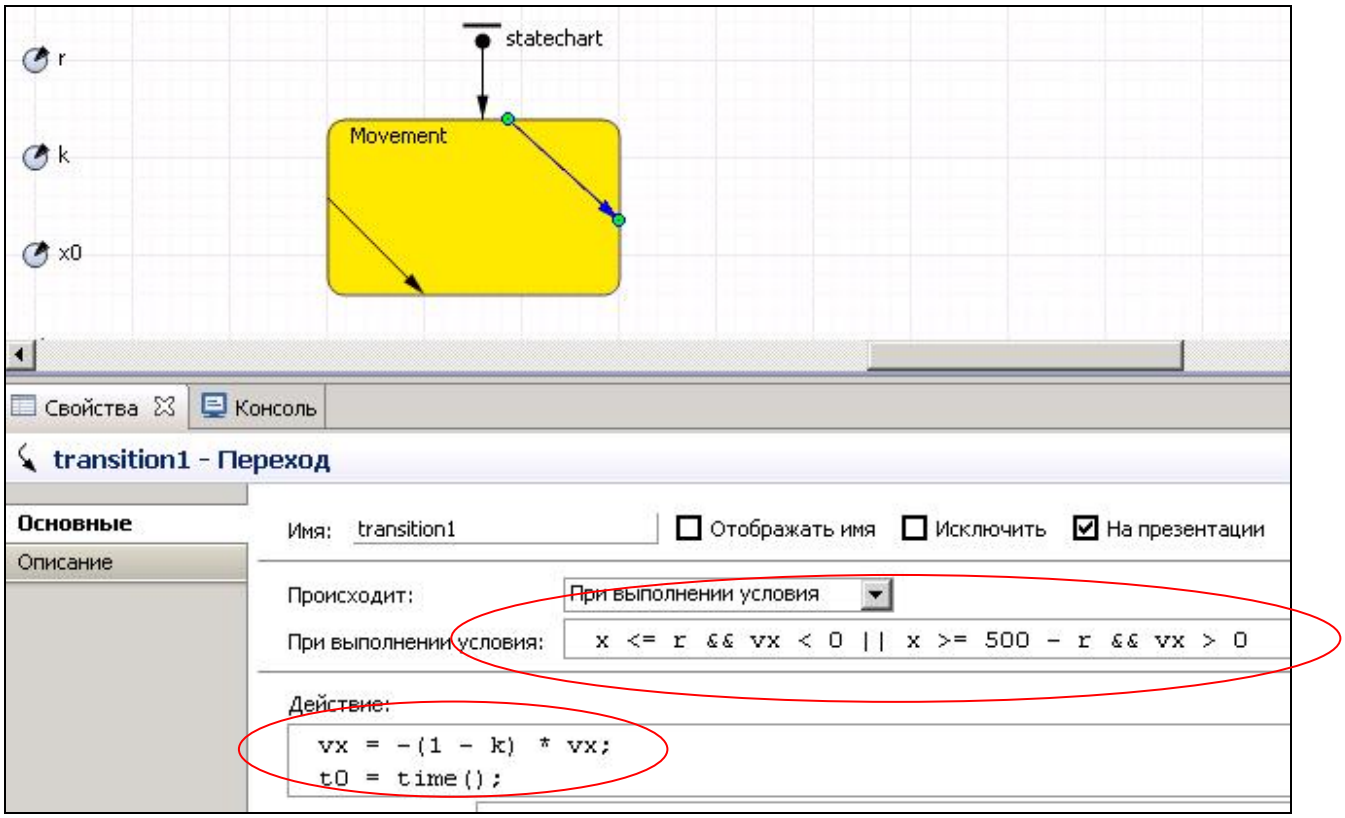


Рис. 2.7

2.4. КОНТРОЛЬНЫЕ ЗАДАНИЯ

1. Измените модель таким образом, чтобы слайдеры действовали на оба мяча одновременно. (5+)
2. Измените модель таким образом, чтобы подсчитывалось время, прошедшее с момента последнего отскока мяча. (5)
3. Измените модель таким образом, чтобы при отскоке от горизонтальных стенок мяч становился желтым на 0,5 сек., а при отскоке от боковых – зеленым. (5)
4. Измените модель таким образом, чтобы на один мяч сила тяжести действовала по оси X, а на другой по оси Y. (5)
5. Измените модель таким образом, чтобы в верхней-правой четверти пространства мячи были синего цвета, а в остальном пространстве – голубого, а при отскоке цвет менялся на желтый на 0,5 сек. (5)
6. Измените модель таким образом, чтобы при каждом отскоке изменялся диаметр мяча на 20%. (4)
7. Измените модель таким образом, чтобы мяч №1 менял свой цвет тогда, когда мяч №2 соударяется со стенками. (4)
8. Измените модель таким образом, чтобы в верхней половине пространства мячи были желтого цвета, а в нижней – голубого, при отскоке цвет не меняется. (4)
9. Измените модель таким образом, чтобы при полете вправо мяч был зеленого цвета, а при полете влево - пурпурного, при отскоке цвет не меняется. (4)
10. Измените модель таким образом, чтобы подсчитывалось количество отскоков мяча. (4)
11. Измените модель таким образом, чтобы при полете вверх мяч был красного цвета, а при полете вниз - голубого, при отскоке цвет не меняется. (4)
12. Измените модель таким образом, чтобы моделировалось движение мяча в правой половине пространства. (4)
13. Измените модель таким образом, чтобы моделировалось движение мяча в верхней

половине пространства. (4)
14. Измените направление движения мяча на горизонтальное. (3)
15. Измените модель таким образом, чтобы при отскоке цвет одного из мячей становился зеленым, а другого – желтым. (3)
16. Измените модель таким образом, чтобы мяч вначале выполнения модели был в правом верхнем углу. (3)
17. Измените модель таким образом, чтобы при отскоке мячи меняли цвет на 1 сек. (3)
18. Измените направление движения мяча на вертикальное. (3)

ЗАДАНИЕ 3

ПОСТРОЕНИЕ МОДЕЛИ СЕРДЦА

ЦЕЛИ ЗАНЯТИЯ

- Научиться создавать модели с нуля.
- Создавать графики в презентации модели.
- Создавать слайдеры для управления параметрами модели.

ФОРМА ОРГАНИЗАЦИИ ЗАНЯТИЯ

Фронтальная.

СТУДЕНТ ДОЛЖЕН ЗНАТЬ

- понятия: проект, активный объект, переменная, параметр, презентация, эксперимент,
- основы алгоритмического языка Java,
- интерфейс программы AnyLogic.

СТУДЕНТ ДОЛЖЕН УМЕТЬ

- выполнять лабораторно-практическое задание №2,
- редактировать параметры модели в программе AnyLogic,

ОБЕСПЕЧЕННОСТЬ

- компьютер с установленной программой AnyLogic версии 6,
- настоящий курс лабораторно-практических работ.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

3.1. ПОСТАНОВКА ЗАДАЧИ

Мы рассмотрим простейшую математическую модель, описывающую процессы, похожие на биение сердца. Эта модель описана двумя дифференциальными уравнениями первого порядка:

$$\frac{dx}{dt} = \frac{(x - x^3 - b)}{\varepsilon}; \qquad \frac{db}{dt} = x - x_0,$$

где: `x` – радиус сердца, `x0` – его начальное значение, `b` – переменная, а ε – параметр `eps`.

В этой модели мы исследуем характер зависимостей переменных `x` и `b` от времени при разных значениях параметра `eps`, а также построим фазовую диаграмму зависимости радиуса `x` от переменной `b`.

3.2. СОЗДАНИЕ НОВОГО ПРОЕКТА

Запустите AnyLogic. Закройте все ранее открытые проекты, для этого выберите в основном меню **Файл / Закрыть все**.

Для построения нового проекта щелкните кнопку **Создать** панели инструментов, либо выберите в основном меню **Файл / Создать / Модель**.

В появившемся диалоговом окне установите рабочую папку: рабочий_диск:\номер_группы\ФИОстудента, наберите `heart` как имя модели и щелкните **Далее**. В следующем окне оставьте выбранным по умолчанию **Начать создание модели с «нуля»** и щелкните **Готово**. Новый проект под названием `heart` будет создан.

Окно редактора нового проекта содержит несколько панелей. Слева в панели проектов автоматически строится дерево проекта. Оно обеспечивает навигацию по всем элементам проекта, которые будут создаваться при построении модели. Для нашего проекта в нем уже создан корневой класс активного объекта с именем `Main`, а для проведения экспериментов с моделью уже создан один эксперимент с именем `Simulation`. Центральная панель – панель графического редактора класса активного объекта в которой мы будем производить основные построения модели.

Нижняя панель – это панель свойств выделенного элемента модели. В данном случае панель показывает свойства класса корневого объекта с именем `Main`, если окно его структуры активно. Измените имя корневого объекта нашей модели, назвав его `Heart` (вместо установленного по умолчанию имени `Main`). Для этого в поле **Имя** вкладки **Основные** панели **Свойства** корневого объекта введите `Heart` вместо `Main`. В панели проектов имя корневого объекта сразу изменится.

Поскольку AnyLogic создает для каждого класса активного объекта соответствующий Java класс, при задании имени класса активного объекта нужно руководствоваться правилами названия классов в Java. Пожалуйста, начинайте имя класса с заглавной буквы.

3.3. ПОСТРОЕНИЕ МОДЕЛИ

В нашей модели должны присутствовать две переменные состояния – `x` и `b`, и два параметра – `x0` и `eps`, где `x0` - начальное значение `x`. Начальное значение переменной `b` зададим константой.

В модели переменная `x` определяется дифференциальным уравнением:

$$dx/dt = (x - x^3 - b)/eps$$

с начальным значением `x`, равным `x0`. В AnyLogic можно подобные зависимости задавать именно в таком аналитическом виде. Для того чтобы таким образом определить переменную `x`, зададим ее в форме накопителя.

Для этого перетащите мышью элемент **Накопитель,** расположенный во вкладке **Системная динамика** панели **Палитра** на диаграмму класса активного объекта `Heart`. Пиктограмма появится в поле редактора с именем `stock`. Одновременно внизу вместо окна свойств объекта `Heart` появится окно свойств выделенного накопителя. В это окно в

поле имени **Имя** вместо предопределенного имени `stock` введите `x` (рис. 3.2) и нажмите клавишу <Enter>.

В поле **Начальное значение** введите `x0`, после чего, см. рис. 3.2, определим формулу для вычисления `x` в поле **dx/dt =** следующим образом:

```
(x - x*x*x - b) /eps
```

Если Вы работаете с программой AnyLogic версии 6.6 и выше, то поле **Начальное значение** будет недоступным. Для того чтобы ввести в это поле формулу, нужно выбрать **Режим задания уравнения** `Произвольный`, см. рис. 3.1.

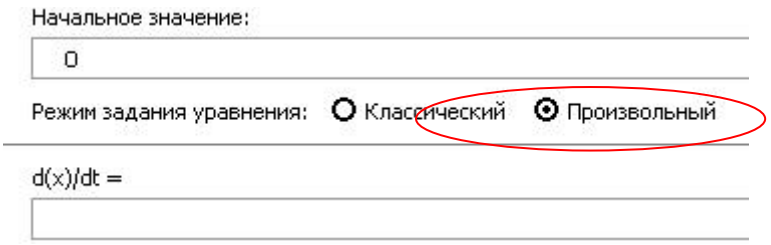


Рис. 3.1

При выделенной пиктограмме накопителя ☐ его имя можно перемещать по диаграмме класса. Саму пиктограмму накопителя также можно перемещать при нажатой на ней левой кнопке мыши.

Вторая переменная `b` задана дифференциальным уравнением `db/dt = x - x0`. Ее создадим в модели аналогичным образом. Установим начальное значение `b` равное 0. В поле **Начальное значение** окна свойств переменной `b` величину `0` можно не записывать: если это поле пусто, по умолчанию значение переменной считается нулевым.

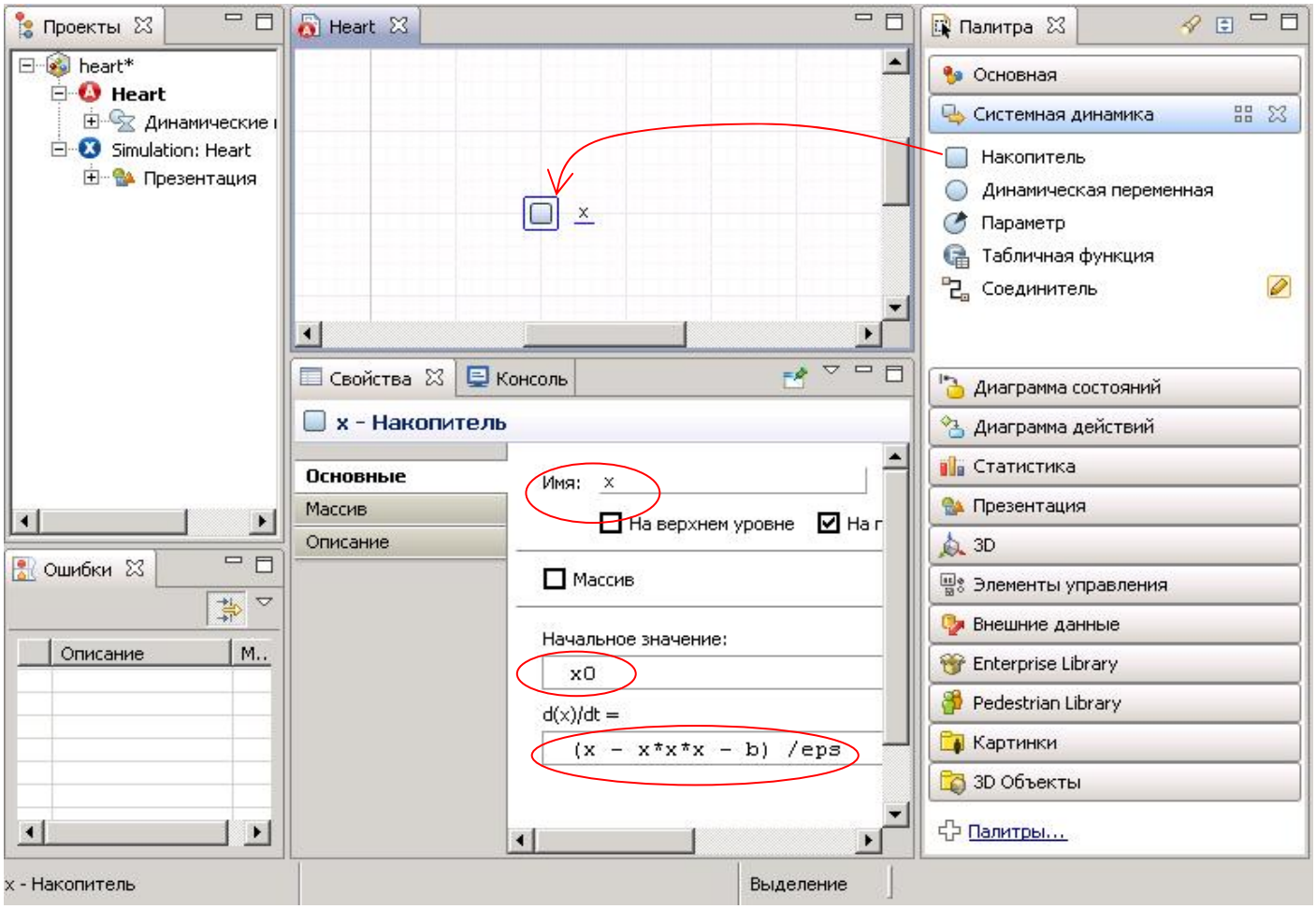


Рис. 3.2

На диаграмму класса активного объекта можно помещать текстовые комментарии. Для этого перетащите элемент **Текст** со вкладки **Презентация** панели палитр на диаграмму класса, рядом с накопителями. Введите следующий комментарий: `Структура активного объекта`. Его можно редактировать в поле **Текст** окна свойств этого комментария.

Для проверки правильности синтаксиса (модели можно использовать кнопку **Построить модель** панели инструментов или нажать клавишу F7. Если щелкнуть на этой кнопке, то выполнится компиляция разрабатываемой модели в программный код на языке Java. Щелкните по кнопке **Построить модель**. В нашем примере обнаружились ошибки (рис. 3.3): действительно, нами не определены параметры `x0` и `eps`.

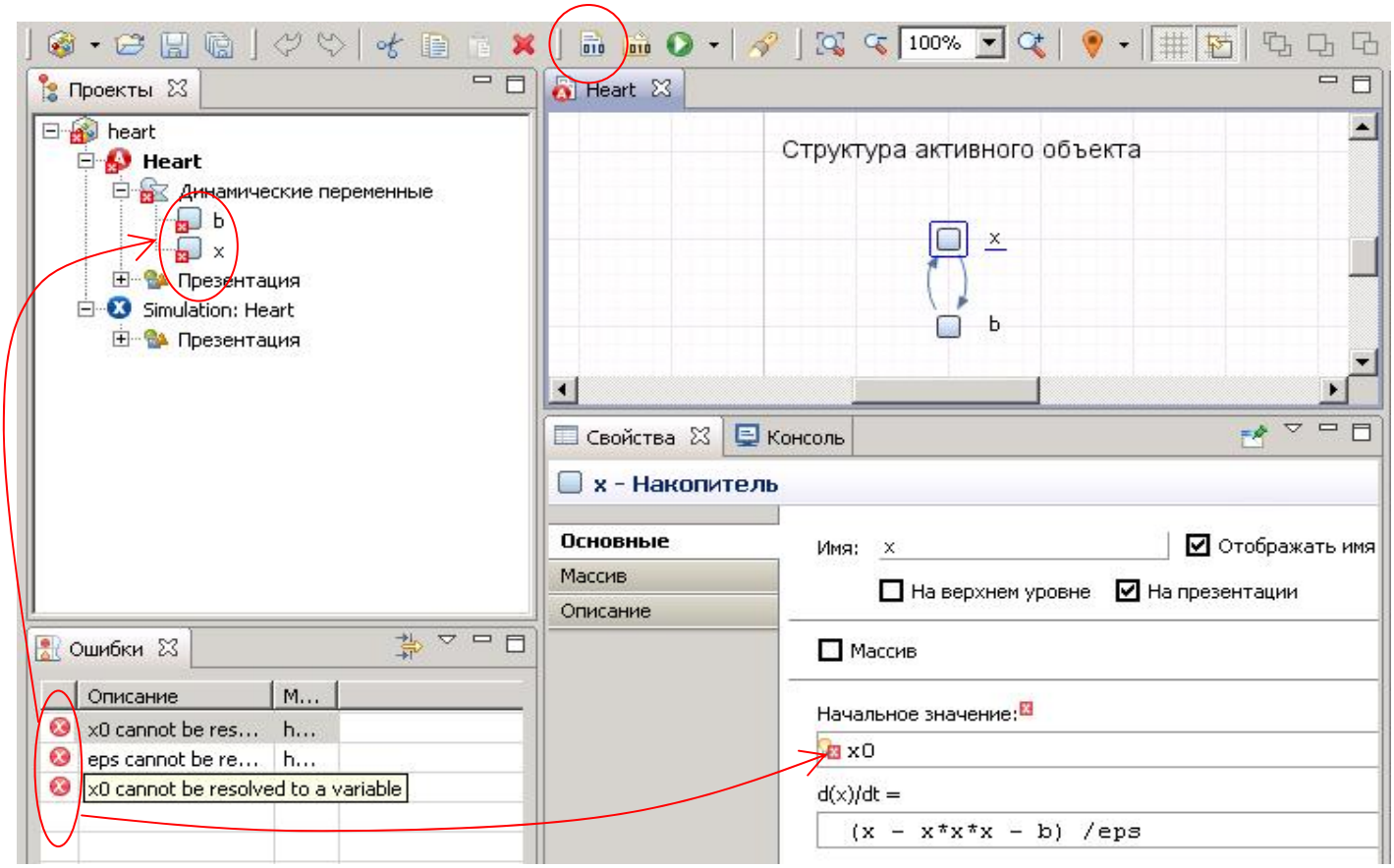


Рис. 3.3

На наличие ошибки указывает появившиеся символы [X] в панели **Ошибки**. Для каждой ошибки показывается ее описание и местоположение - имя элемента модели, при задании которого эта ошибка была допущена. Двойной щелчок мышью по ошибке, в зависимости от того, где она была совершена, приведет к открытию того или иного редактора или панели. Если, например, это графическая ошибка, то будет открыт графический редактор, отображающий диаграмму соответствующего класса активного объекта, в котором будут выделены фигуры, которые были неправильно нарисованы.

Для завершения построения модели указанные пропущенные параметры нужно задать. Пусть `x0 = 0.5, eps = 0.01`. Перетащите мышью пиктограмму **Параметр** с вкладки **Основная** панели **Палитры** на диаграмму класса активного объекта `Heart`. Назовите его `x0`, затем на вкладке **Основные** панели свойств этого параметра введите в поле **Значение по умолчанию** – 0.5. Остальные поля оставьте без изменения. Переменная `eps` со значением 0.01 задается так же. Снова выполните проверку синтаксиса: нажмите кнопку **Построить модель**. В результате на экране вы получите следующее - рис. 3.4.

Надпись «Построение удачно завершено» в левом нижнем углу окна программы свидетельствует о том, что синтаксис модели правильный.

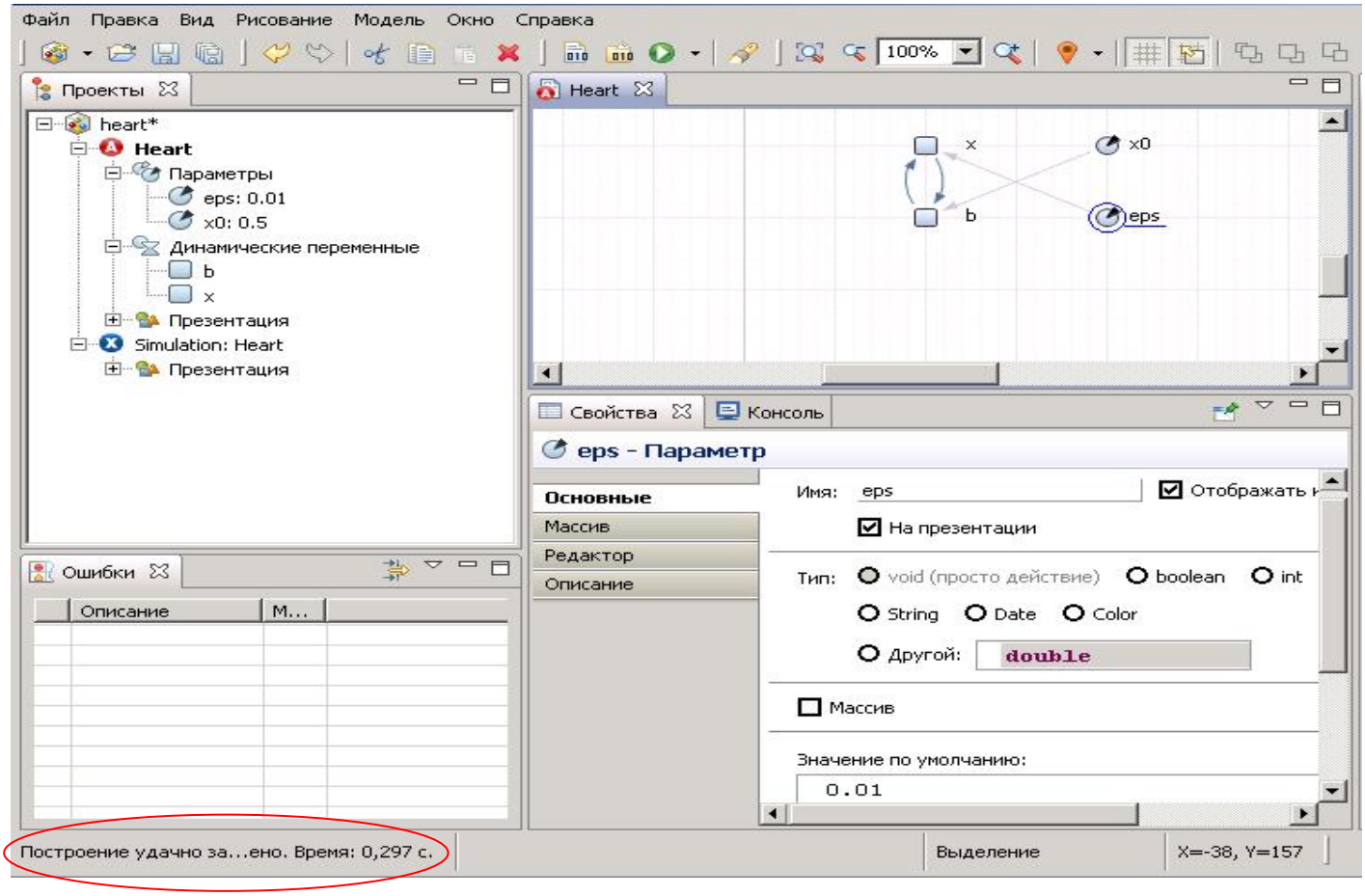


Рис. 3.4

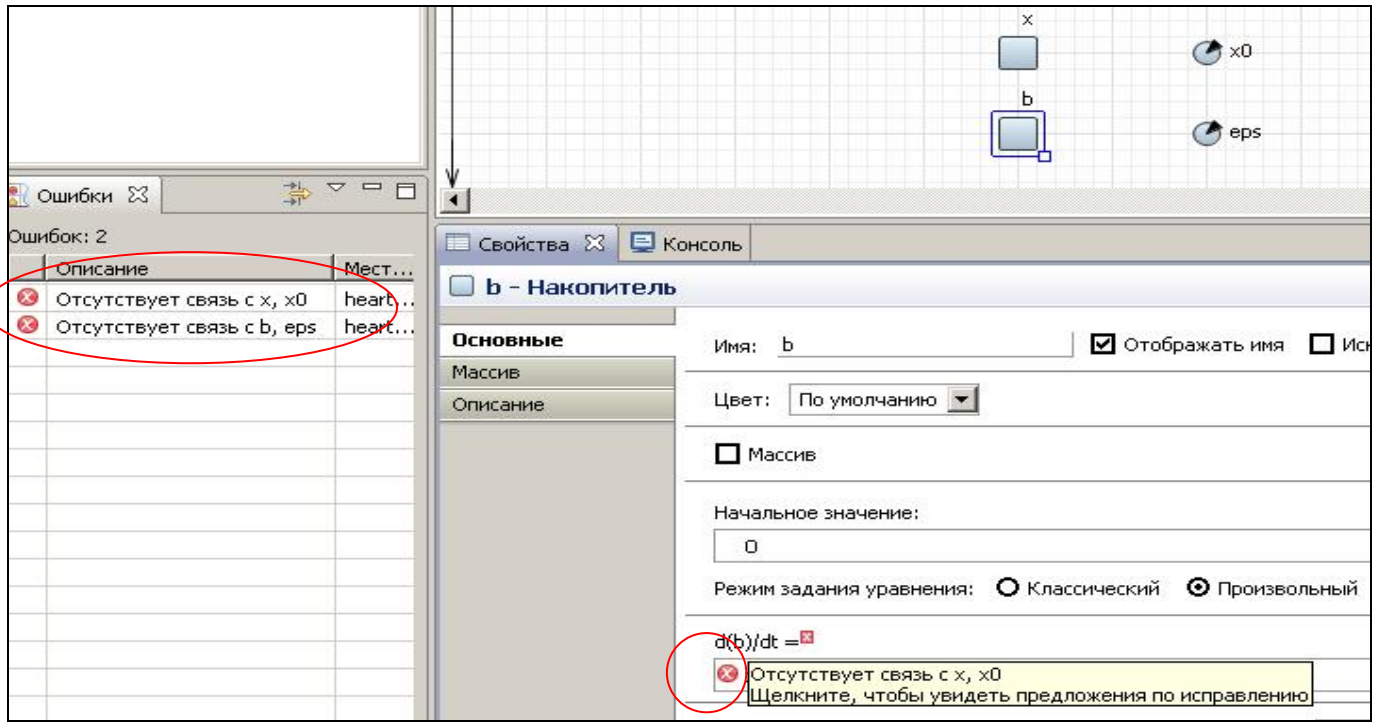


Рис. 3.5

В версии 6.6 программы AnyLogic связи между переменными автоматически не построятся, и в панели **Ошибки** мы увидим, что две ошибки остались неисправленными, а

в их описании будет указано, что: `Отсутствует связь с …`, см. рис.3.5.

Для задания связи между переменными выделите мышью переменную, например `b`, и в панели свойств установите курсор в поле с уравнением `db/dt = x - x0`. Затем щелкните мышью по красному крестику слева от уравнения, откроется всплывающее окно с предложением создать связь, рис. 3.6.

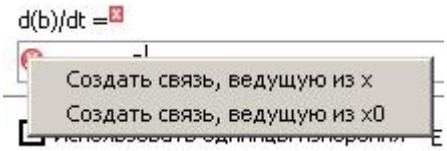
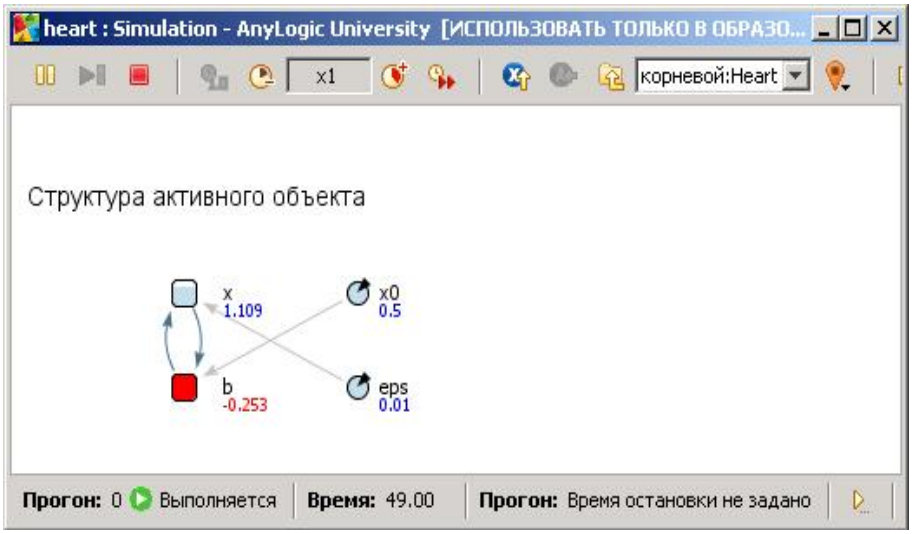


Рис. 3.6

Создайте все связи и выполните проверку синтаксиса, если все правильно, то появится сообщение об удачном построении модели.

3.4. ЗАПУСК МОДЕЛИ



Щелкнув на кнопке запуска модели, после компиляции откроется окно презентации эксперимента. Запустив эксперимент, увидим структуру активного объекта: переменные и параметры с их значениями (рис. 3.7). Переменные `b` и `x` в этом окне начнут изменяться в соответствии с определенными для них уравнениями.

Рис. 3.7

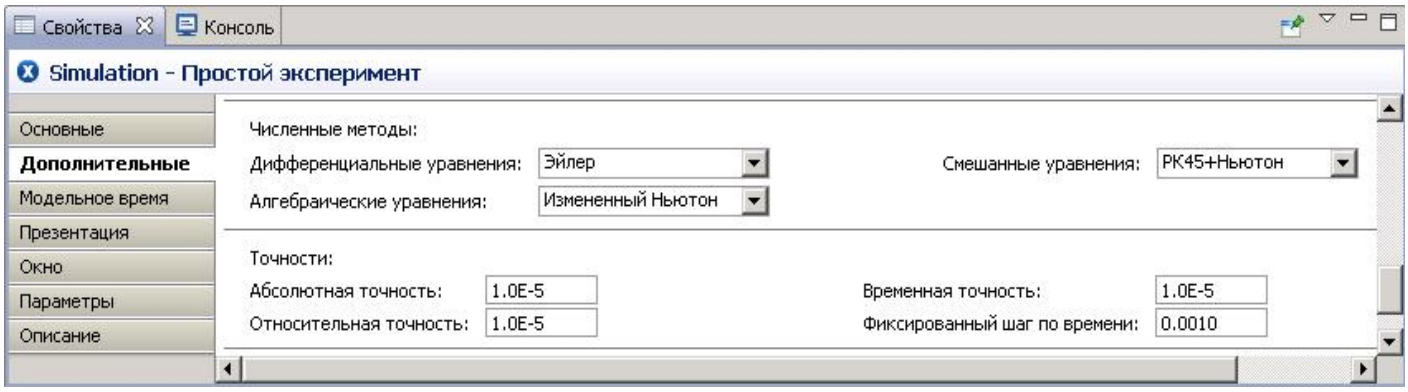
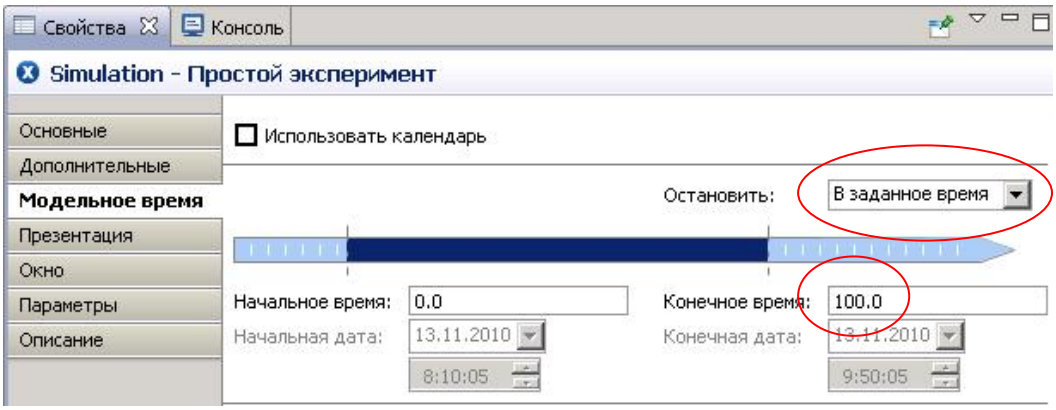


Рис. 3.8

Системы дифференциальных и алгебраических уравнений, при выполнении модели, решаются одним из встроенных численных методов. Сам метод и необходимая точность решения выбираются автоматически, но можно изменить предварительные установки во вкладке **Дополнительные** панели свойств объекта `Simulation`, рис. 3.8.

По умолчанию модель будет выполняться бесконечно, но можно задать условие остановки эксперимента во вкладке **Модельное время** панели свойств объекта `Simulation`, рис. 3.9.



Проведите несколько экспериментов с различными скоростями выполнения данной модели, используя кнопки останова и запуска.

Рис. 3.9

3.5. ГРАФИКИ И ДИАГРАММЫ

AnyLogic позволяет наглядно представить поведение модели, в частности, представить изменения во времени всех ее переменных. Существуют два способа просмотра графиков: с помощью окон инспекта и с помощью диаграмм.

Окна инспекта мы подробно рассмотрели в работе №1, здесь отметим лишь то, что эти графики строятся на базе автоматически создаваемых наборов данных, в которые периодически записываются новые и новые значения переменных. Если кривая Вашего графика покажется Вам не гладкой, а "рваной", то причина этого может заключаться в том, что AnyLogic недостаточно часто обновляет наборы данных новыми значениями.

Чтобы изменить частоту обновления автоматически создаваемых для переменных наборов данных перейдите на страницу свойств **Дополнительные** того класса активного объекта, на диаграмме которого находится переменная и измените период обновления данных в поле **Период,** рис. 3.10.

Для получения полнофункциональных графиков лучше воспользоваться диаграммами AnyLogic, которые позволяют динамически визуализировать данные, собираемые в результате работы модели. Набор диаграмм схож с тем, что предлагается программой MS Excel.

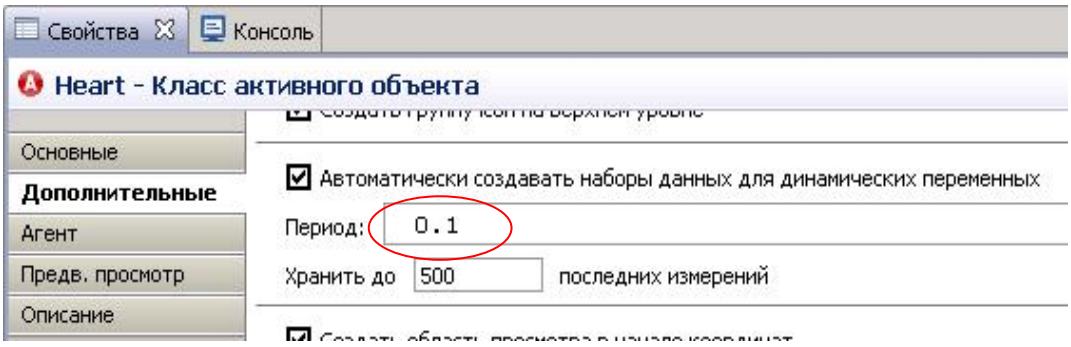


Рис. 3.10

Построим графики зависимостей переменных `b` и `x` от времени и фазовую диаграмму `b` от `x`. Перетащите элемент **График** из палитры **Статистика** в то место графического редактора, где Вы хотите нарисовать график. Перейдите на страницу **Основные** панели **Свойства**.

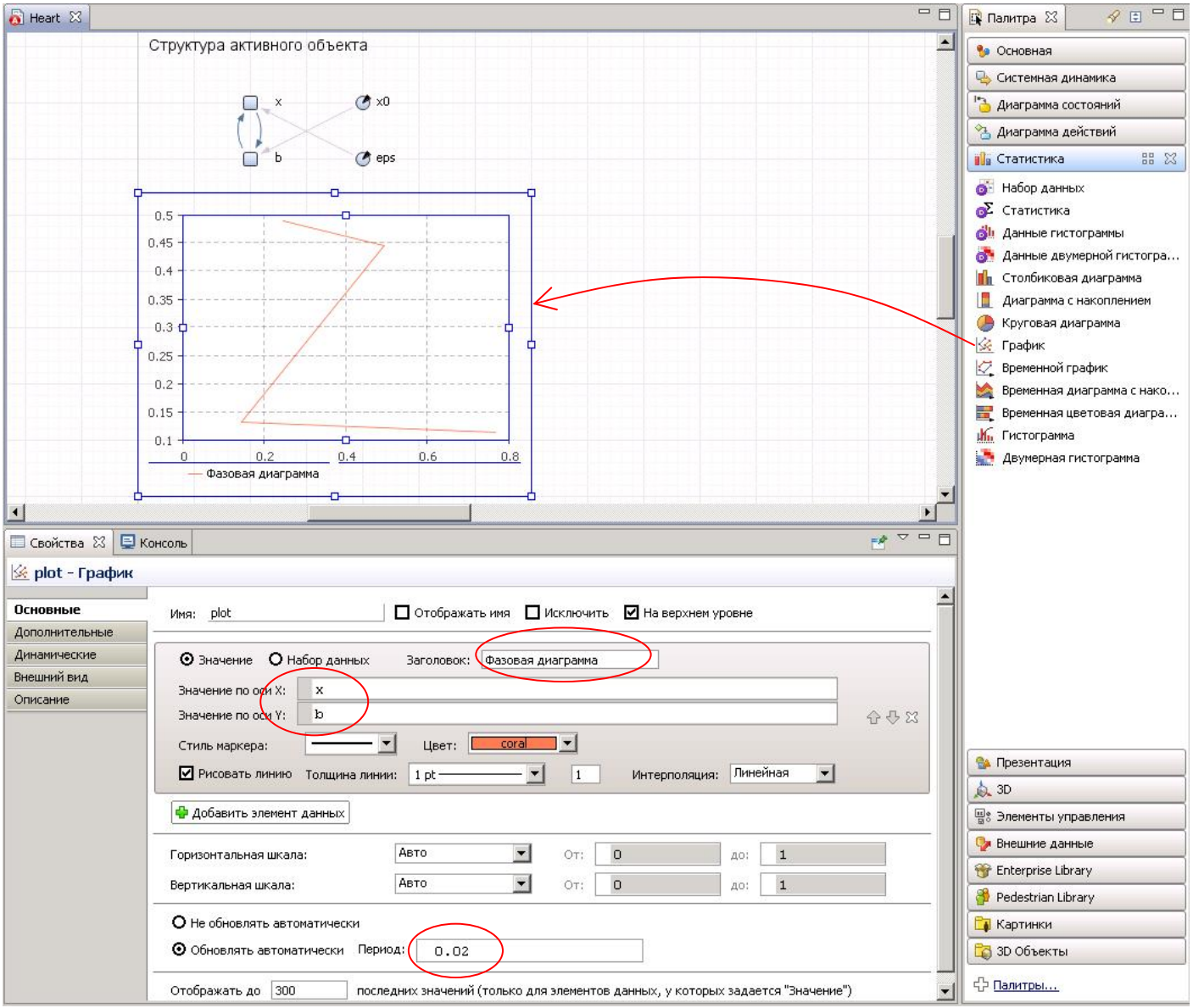


Рис. 3.11

Щелкните мышью по кнопке **Добавить элемент данных**. При этом над кнопкой появится новая секция свойств, задающая настройки нового элемента данных, который будет отображаться на данном графике. В поле **Значение по оси X** введите имя переменой X, а в поле **Значение по оси Y** – имя переменной b. Задайте заголовок `Фазовая диаграмма` для этого элемента данных в поле **Заголовок**. Для того чтобы получить гладкую диаграмму нужно уменьшить период обновления данных в поле **Период**, рис. 3.11.

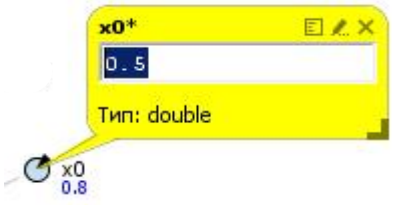


Рис. 3.12

Аналогично постройте временной график для переменных `x` и `b`, разместив их на одном графике. **Временной диапазон** укажите равным 10, рис.3.13.

AnyLogic позволяет настраивать внешний вид и функциональность диаграмм. Поэкспериментируйте с установкой цвета графиков, опциями отображения наборов данных, размерами отображаемого окна, прозрачностью, легендой, и т. п.

Проведите серию экспериментов с моделью, перезапуская ее с различными параметрами.

Для изменения параметров в процессе выполнения модели, нужно в окне инспекта этого параметра щелкнуть мышью по кнопке с карандашом в верхней части окна, рис. 3.12.

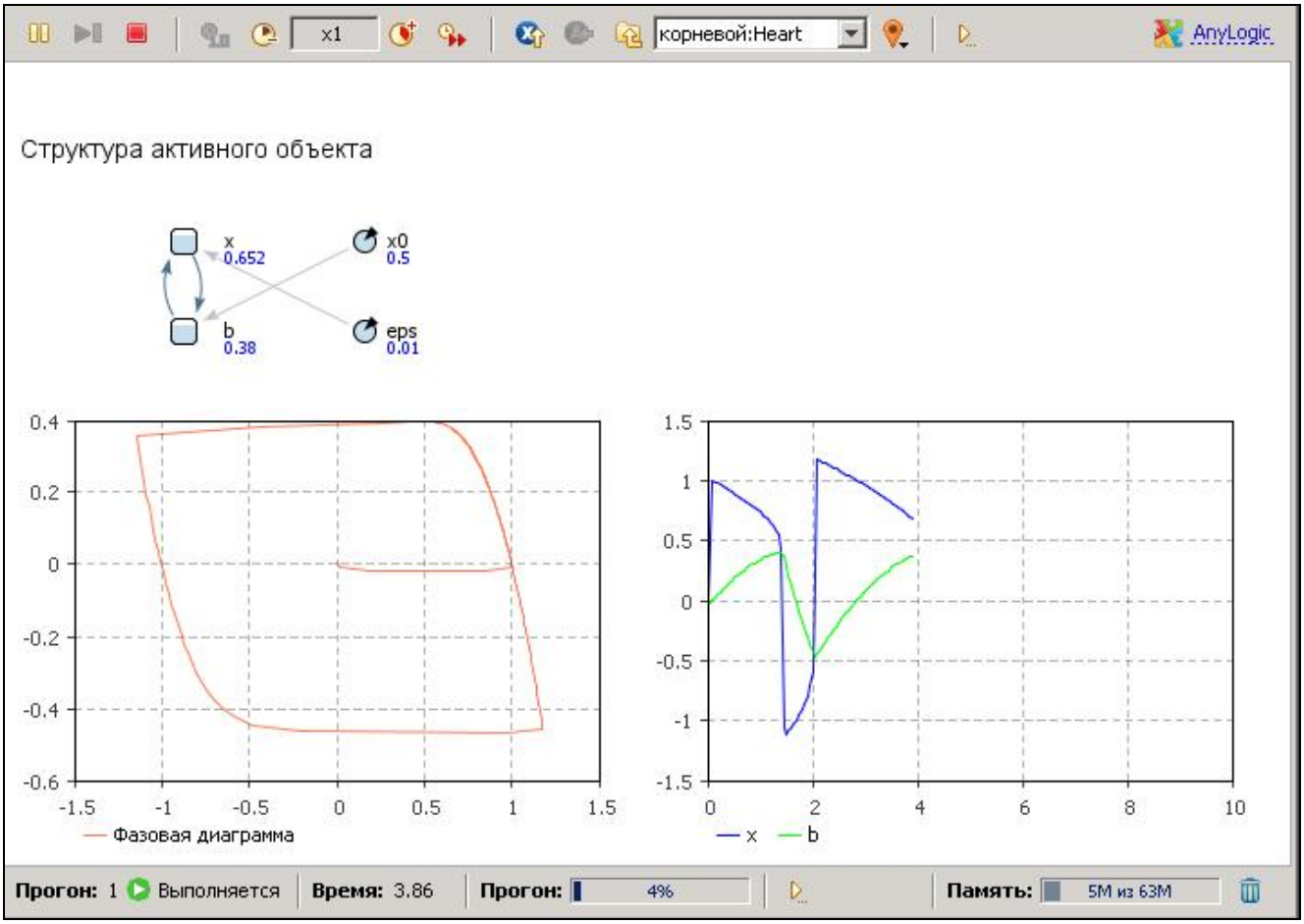


Рис. 3.13

3.6. ПРЕЗЕНТАЦИЯ МОДЕЛИ

Для лучшего понимания динамики модели и наблюдения за процессами, в AnyLogic можно строить анимированные изображения, состоящие из динамических элементов. Графические элементы называются динамическими, поскольку все их параметры - координаты, размер, цвет и даже их видимость – в процессе выполнения модели можно сделать зависимыми от переменных и параметров, которые меняются со временем при выполнении модели.

Остановите выполнение модели и вернитесь на диаграмму класса `Heart`. Прямо на диаграмме мы будем рисовать презентацию сердца.

3.6.1. ОБЪЕКТЫ В ПРЕЗЕНТАЦИИ

Построим презентацию сердца в виде изображения овала, радиус которого будет меняться. Этот радиус является функцией от значения переменной `x` модели. Для построения изображения овала перетащите мышью из вкладки **Презентация** панели **Палитра** пиктограмму **Овал** на диаграмму класса. Внизу появится окно свойств этого овала. По умолчанию имя этого объекта будет `oval`, координаты `X` и `Y` соответствуют месту, куда мы поместили овал, а радиус X и радиус Y соответствуют тому, что мы нарисовали.

3.6.2. ДИНАМИЧЕСКИЕ ПАРАМЕТРЫ ГРАФИЧЕСКИХ ОБЪЕКТОВ

Запустив модель, мы увидим, неподвижный овал, находящийся в заданном месте. В AnyLogic принята следующая концепция: каждая характеристика графического элемента имеет два значения: статическое и динамическое. Статическое значение определяет параметр (координату, угол поворота, цвет и т. п.) объекта как константу. Динамическое определяет значение этого параметра в процессе выполнения модели и может быть определено как значение любой переменной модели. Поэтому у графических объектов в панели свойств имеются вкладки **Основные** и **Дополнительные** для статических значений, и вкладка **Динамические** – для динамических значений параметров. Если динамическое значение не определено, графический объект сохранит свое статическое значение.

Выделите овал, представляющий динамику сердца. Статические значения его параметров задайте так: в панели свойств овала на вкладке **Основные** цвет заливки определите бордовым, цвет линии границы - красным; толщину линии границы установите 2.

Изменение объема сердца представим радиусами овала на вкладке **Динамические** как функцию от переменной `x`, как показано на рис. 3.14.

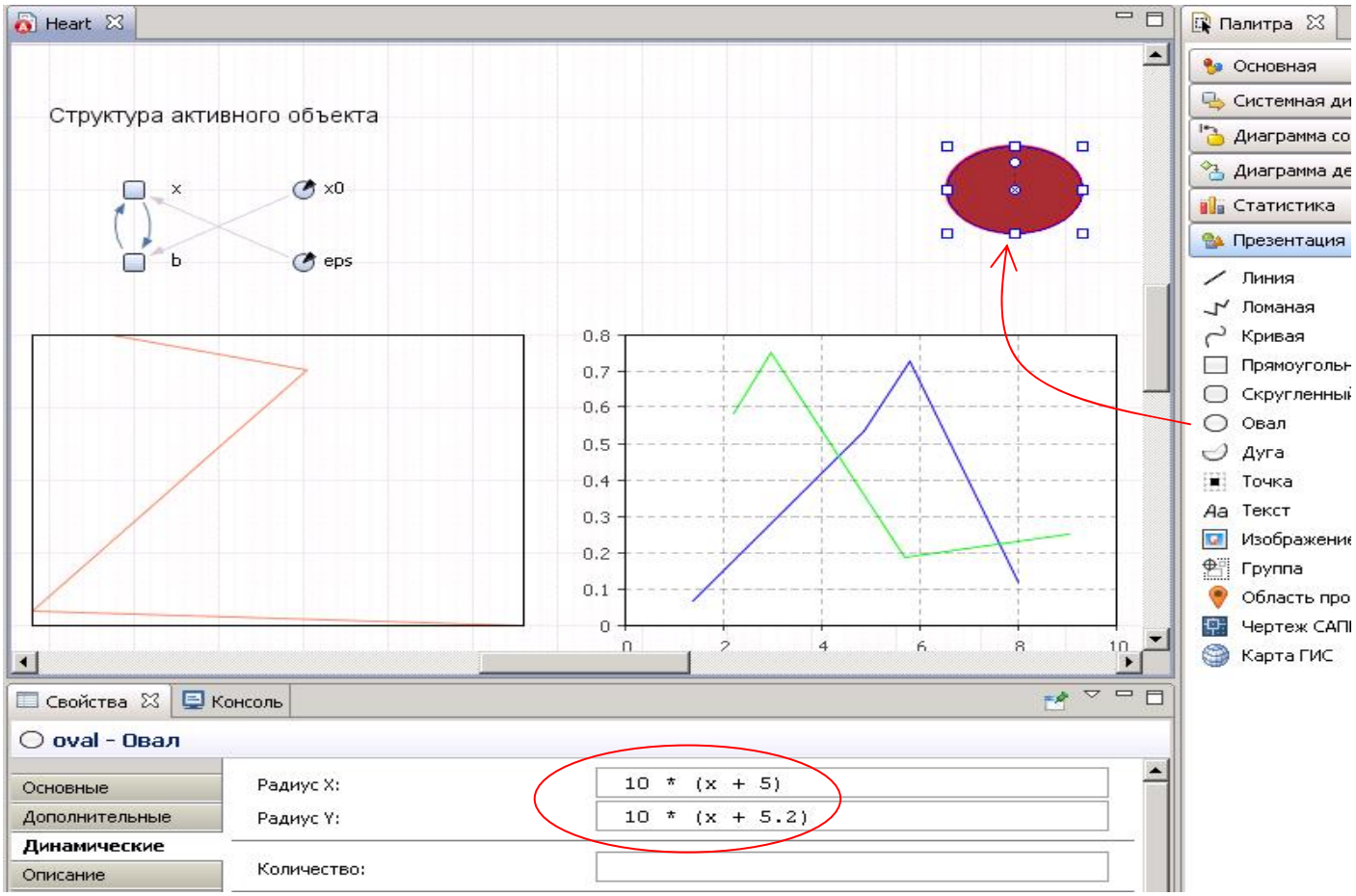


Рис. 3.14

Запустите модель. Проведите эксперименты с установкой различных параметров модели (`x0` и `eps`) и наблюдайте, как изменяется характер сердцебиения.

3.6.3. СЛАЙДЕРЫ В ПРЕЗЕНТАЦИИ

В AnyLogic существует элемент управления, позволяющий пользователю графически выбирать число из заданного диапазона значений путем перетаскивания рукоятки – бегунок.

Бегунки или их еще называют слайдеры, обычно используются для изменения значений численных переменных и параметров во время выполнения модели.

Перетащите мышью с вкладки **Элементы управления** панели **Палитра** пиктограмму **Бегунок** на диаграмму класса `Heart` между структурой и анимационным овалом. В поле **Связать с** панели свойств введите имя параметра `x0`, а минимальное и максимальное значения, которые можно регулировать слайдером, установите 0 и 1. Добавить подписи к слайдеру можно простым нажатием кнопки **Добавить метки**, рис. 3.15. Запустите модель и проверьте действие слайдера.

Аналогично сделайте слайдер для параметра `eps`, установив ограничения от 0,01 до 0,5.

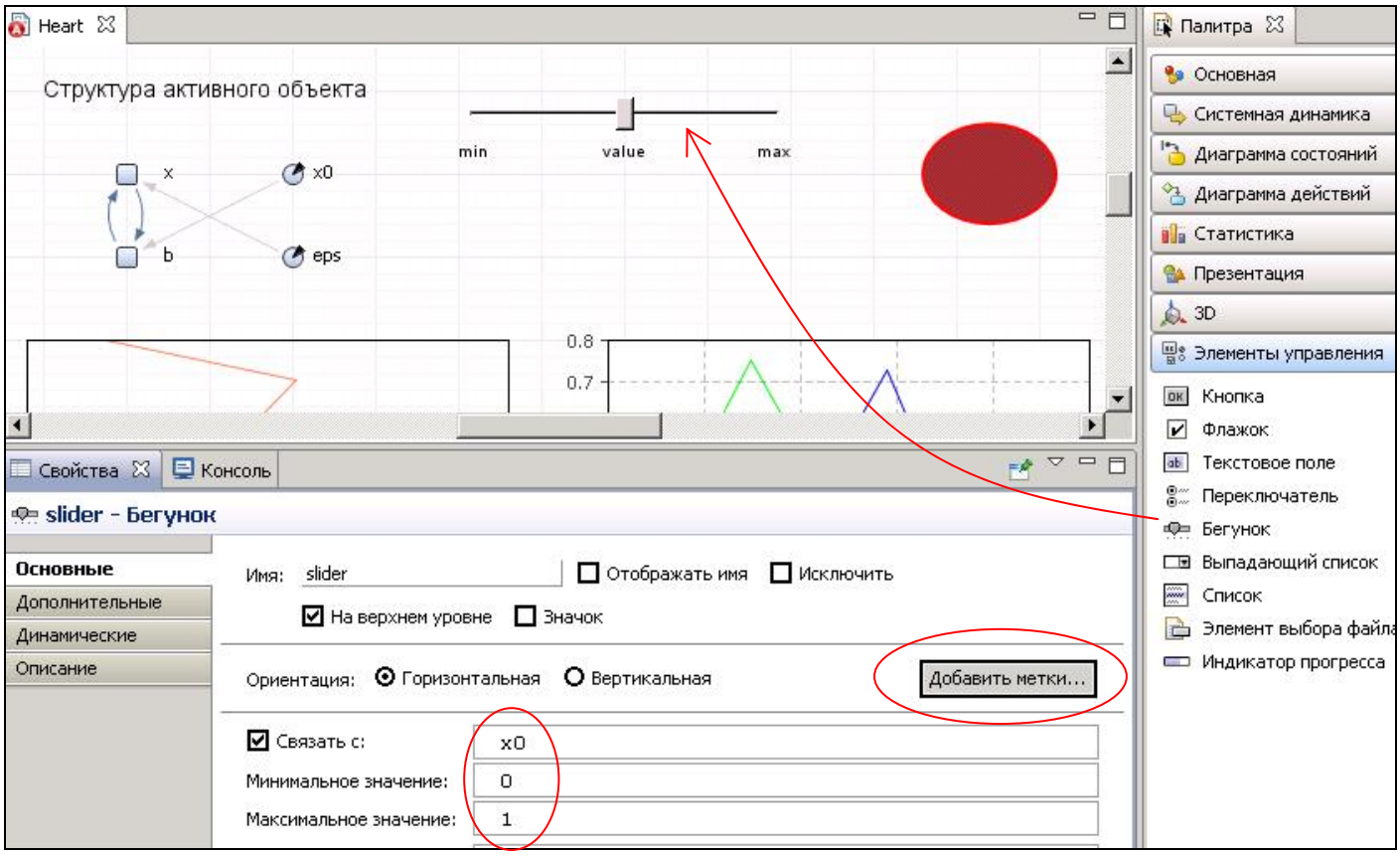


Рис. 3.15

3.6.4. ТЕКСТ В ПРЕЗЕНТАЦИИ

В верхнюю часть диаграммы класса `Heart` введите поясняющий текст. Для этого перетащите мышью пиктограмму **Текст** с вкладки **Презентация** панели **Палитра.** В поле **Текст** вкладки **Основные** введите текст `Модель сердца`, выберите шрифт **Century Gothic**, стиль **Полужирный**, размер 22. На вкладке **Дополнительные** установите координаты (X, Y) этого объекта (20, 10).

Следующий текст, содержащий пояснение к модели, введите чуть ниже и правее: `Это упрощенная модель динамики сердечных сокращений. Модель описывается двумя дифференциальными уравнениями: dx/dt=(x-x^3-b)/eps и db/dt=x-x0. Модель позволяет изменять параметры и наблюдать изменение ритма и формы пульсаций. При значении x0 = 0.6 сердце останавливается.`
Установите выравнивание по правому краю, все другие параметры текста можете оставить по умолчанию. Для того чтобы текст уместился по ширине, в нужных местах следует вставить перевод строки.

Элемент **Текст** также имеет динамические свойства, как и другие графические элементы. Это значит, что можно в процессе выполнения модели динамически изменять положение и ориентацию текста, его цвет, и даже сам текст.

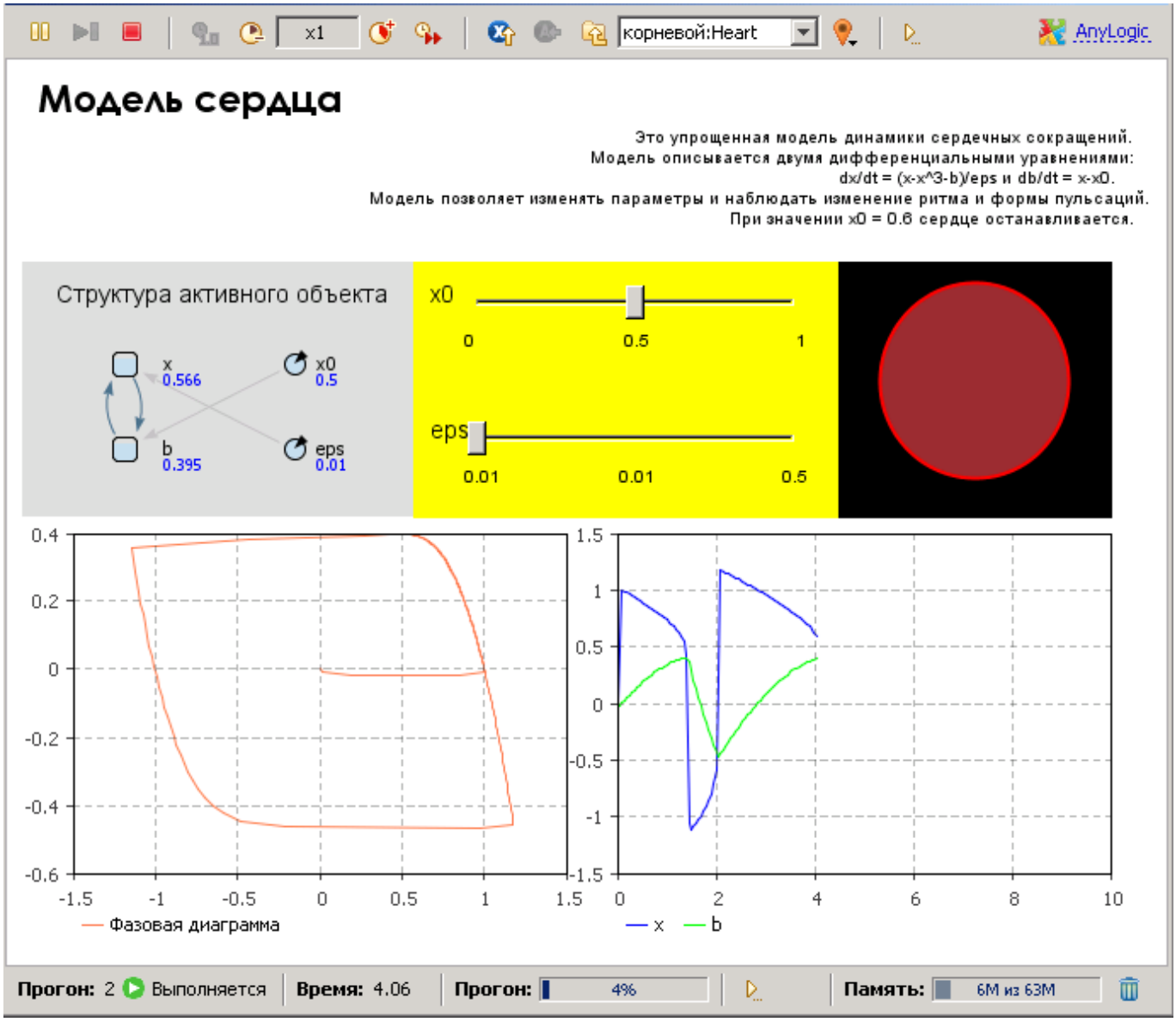


Рис. 3.16

Введите в поле презентации прямоугольники, как показано на рис. 3.16, и поместите их на задний план, чтобы они выделяли функциональные блоки построенной модели. Для того чтобы фигура была фоном и не закрывала другие изображения, в контекстном меню данного прямоугольника выполните команду **Порядок / На задний план**.

Сделайте подписи к слайдерам, чтобы было понятно, какими параметрами мы управляем.

Наша модель построена. Продемонстрируйте ее преподавателю.

3.7. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Измените презентацию сердца так, чтобы сплюснутый овал, имитирующий сердце вращался вокруг своей оси. (5+)
2. Поместите на изображение сердца текст с динамическим значением переменной X. (5)
3. Поместите в поле презентации текущее значение времени. (5)
4. Измените презентацию сердца так, чтобы овал, сжимаясь по оси, Y расширялся по оси X. (5)
5. Какие существуют способы построения графика изменения переменной? (4)
6. Как построить фазовую диаграмму? (4)
7. Как изменить период обновления графика в окне инспекта? (4)
8. Как изменять параметры модели в процессе ее выполнения? (4)
9. Как изменить условие остановки выполнения модели? (4)
10. Как выбрать численный метод решения системы обыкновенных дифференциальных уравнений? (4)
11. Как ввести в модель слайдер и связать его с переменной? (4)
12. Как изменить период обновления графика? (4)
13. Как создать подписи к слайдеру? (3)
14. Перечислите параметры графических элементов, которые могут динамически изменяться в процессе выполнения модели? (3)
15. Как создать переменную, определяемую дифференциальным уравнением? (3)
16. Как открыть график в окне инспекта? (3)
17. Как запустить компиляцию модели в программный код на языке Java? (3)
18. Как создать параметр и присвоить ему значение? (3)

ЗАДАНИЕ 4

ДИСКРЕТНО-СОБЫТИЙНАЯ МОДЕЛЬ СЧЕТЧИКА

ЦЕЛИ ЗАНЯТИЯ

В результате построения этой модели будут рассмотрены следующие новые вопросы:

- Создание нового класса активного объекта.
- События.
- Значки активного объекта.
- Порты и сообщения;
- Действия при получении сообщений.

ФОРМА ОРГАНИЗАЦИИ ЗАНЯТИЯ

Фронтальная.

СТУДЕНТ ДОЛЖЕН ЗНАТЬ

- понятия: проект, активный объект, переменная, параметр, презентация, эксперимент,
- основы алгоритмического языка Java,
- интерфейс программы AnyLogic.

СТУДЕНТ ДОЛЖЕН УМЕТЬ

- выполнять лабораторно-практическое задание №3,
- создавать модели в программе AnyLogic,

ОБЕСПЕЧЕННОСТЬ

- компьютер с установленной программой AnyLogic версии 6,
- настоящий курс лабораторно-практических работ.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

В данной работе мы построим дискретно-событийную модель средствами AnyLogic.

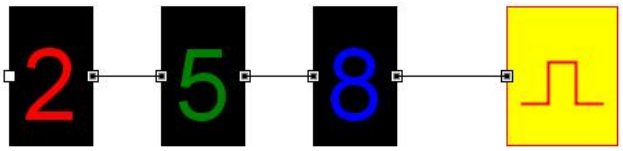
Системы называются дискретно-событийными, если изменения переменных состояния в них происходят только в явно определенные моменты времени или под влиянием явно определенных событий. Находясь в некотором состоянии, дискретная система сохраняет его до наступления очередного события, под воздействием которого переменные системы и, следовательно, ее состояние изменяются скачком. Например, при построении модели банка состояние системы может быть представлено количеством клиентов в помещении банка и числом занятых кассиров. Состояние системы изменяется, если новый клиент входит в банк или освобождается кассир, а это условно можно считать мгновенными событиями.

4.1. ДИСКРЕТНАЯ МОДЕЛЬ СЧЕТЧИКА

В счетчике генератор посылает устройству отображения какой-то сигнал ("тик"). Каждый разряд десятичного счетчика считает число пришедших на его вход "тиков" по модулю 10 и передает на свой выход сигнал переполнения после прихода на его вход каждого десятого сигнала.

При построении модели такого счетчика нужно использовать средства, характерные для моделей дискретно-событийных систем. Здесь нам достаточно трех таких средств: события, порта и передаваемых через порт сообщений.

4.2. ПОСТАНОВКА ЗАДАЧИ



Нужно построить модель трехразрядного десятичного счетчика, работающего от импульсного генератора. На рис. 4.1 представлен счетчик, насчитавший 258 импульсов от генератора.

Рис. 4.1

4.3. МОДЕЛЬ

Для реализации имитационной модели нужно построить генератор "тиков", работающий с заданной частотой, и три одинаковых десятичных разряда счетчика. Следовательно, модель должна содержать три класса активных объектов: генератор "тиков", разряд счетчика по модулю 10 и, кроме того, корневой активный объект, который будет включать в себя один экземпляр генератора и три одинаковых экземпляра одноразрядного счетчика, связанные подходящим образом.

Создайте в своей рабочей директории новый проект под названием `DCounter`. Корневой объект назовите `Model`. Модель будет состоять из нескольких подсистем, связанных между собой.

4.3.2. ПРЕДСТАВЛЕНИЕ СИГНАЛА КАК СООБЩЕНИЯ

Генератор посылает сигналы с определенной частотой разряду счетчика. Для этого предназначен специальный пакет данных – сообщение. Сообщения принимаются и посылаются через специальные элементы активных объектов – порты. Обмен сообщениями возможен только между портами, соединенными соединителями – элементами, играющими роль путей движения сообщений.

В нашей модели счетчика важен только сам факт передачи сообщений, а не содержимое. Прием сообщения будет вызывать увеличение значения разряда счетчика.

4.3.1. ГЕНЕРАТОР ТИКОВ

Постройте новый активный объект `Gen`. Для этого в панели **Проекты** щелкните правой кнопкой мыши по имени проекта `DCounter` и в появившемся контекстном меню выберите команду **Создать/Класс активного объекта**. Назовите новый класс `Gen`.

Класс активного объекта `Gen` должен посылать сообщения первому разряду счетчика с заданной частотой. Для генерации таких сообщений создадим **Событие**, для этого перетащите элемент **Событие**  из палитры **Основная** на диаграмму класса активного объекта `Gen`. В окне свойств этого события нужно оставить **Тип события** `По таймауту` без изменений, а **Режим** установить `Циклический`. Циклический режим события позволяет пользователю выполнять некоторые действия с требуемой периодичностью, например, каждое утро или ежегодно. Период срабатывания обратно-пропорционален частоте, поэтому в поле **Период** запишите `1/frequency`. В нашей модели `frequency` – это частота следования импульсов генератора, ее нужно объявить как параметр модели. Создайте в классе `Gen` параметр `frequency` со значением по умолчанию равным `2`.

4.3.1.1. ПРЕЗЕНТАЦИЯ ГЕНЕРАТОРА

Внешний вид генератора, а точнее объекта `Gen` представим прямоугольником, содержащим изображение импульса, как показано в правой части рис. 4.1. Откройте палитру **Презентация** и перетащите на диаграмму класса `Gen` прямоугольник, в свойствах укажите ширину этого прямоугольника – 80 единиц, а высоту – 100. Залейте его желтым цветом, цвет линии границы сделайте красным, а толщину линии равной 1.

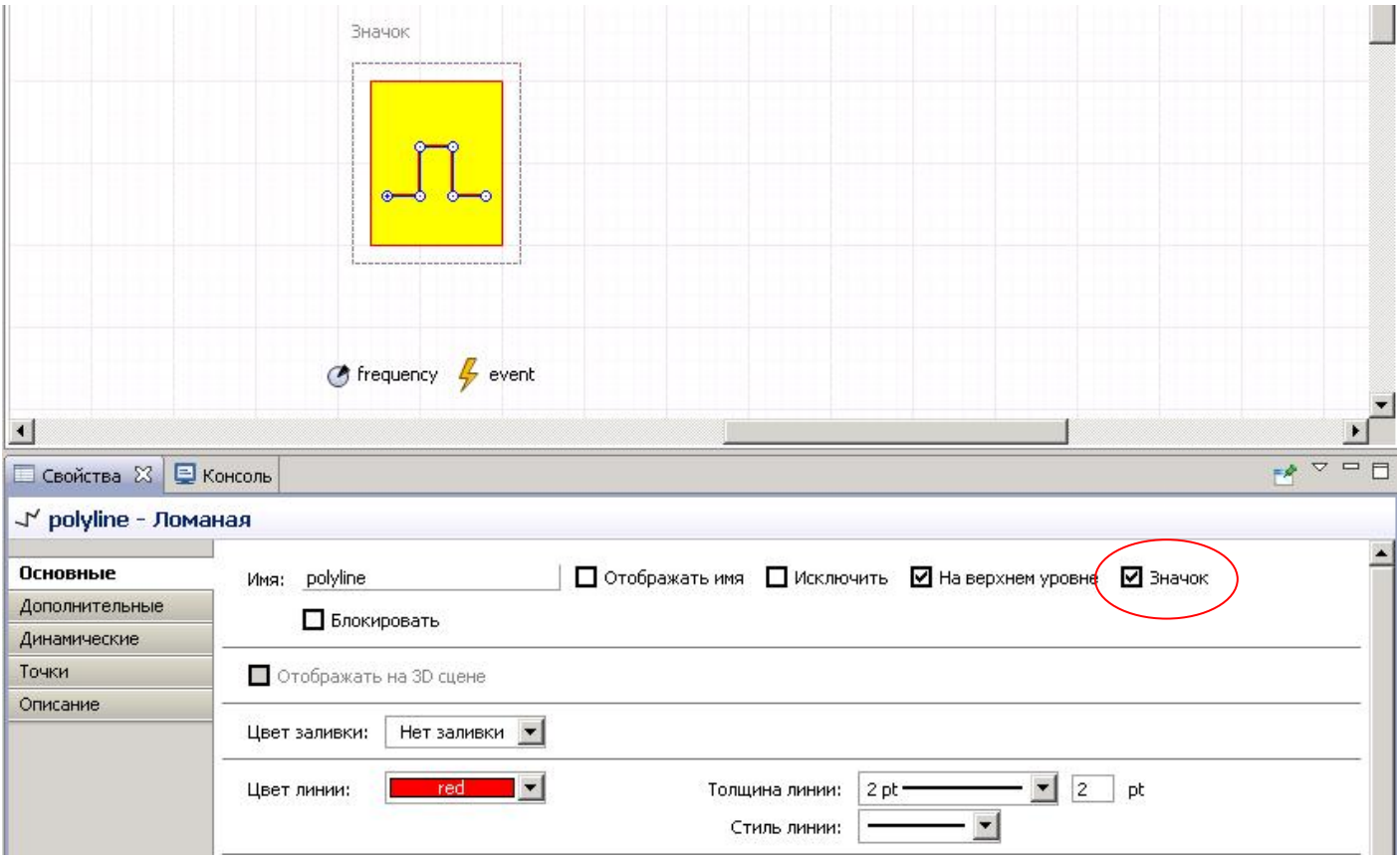
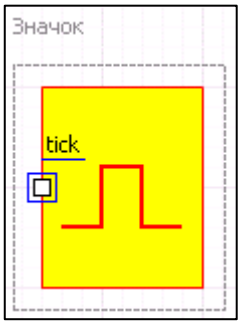


Рис. 4.2

Сделайте двойной щелчок мышью по элементу **Ломаная**  в палитре (при этом его значок должен поменяться на этот: ). Теперь Вы можете рисовать ломаную точка за точкой, последовательно щелкая мышью в тех точках диаграммы, куда Вы хотите поместить вершины ломаной. Чтобы завершить рисование, добавьте последнюю точку ломаной двойным щелчком мыши. Нарисуйте в поле прямоугольника импульс в соответствии с рис. 4.2.

Для того чтобы вложенные объекты – экземпляры класса (Gen) отображались на структурной диаграмме верхнего уровня (Model) в графическом виде и имели интерфейсные элементы – порты, нужно нарисовать для вложенного объекта значок. Мы уже нарисовали прямоугольник с импульсом внутри, для того чтобы сделать их значком выделите последовательно каждую из фигур и на вкладке **Основные** панели **Свойства** установите флажок **Значок**, как показано на рис. 4.2. Вокруг прямоугольника появится пунктирная линия и надпись Значок.



Как уже говорилось, порты играют центральную роль в механизме передачи сообщений. Сообщения посылаются и получаются портами. Порты являются двунаправленными, через них сообщения могут и посылаться, и приниматься. Перетащите элемент **Порт** из палитры **Основная** на диаграмму класса Gen, поместите его на левую сторону желтого прямоугольника и назовите этот порт tick, рис. 4.3.

Рис. 4.3

Через этот порт наш экземпляр класса активного объекта Gen может посылать и принимать сообщения от других активных объектов.

4.3.1.2. ГЕНЕРАЦИЯ ИМПУЛЬСОВ

Генератор нашей модели должен периодически с периодом 1/frequency посылать сообщения через порт tick. Ранее мы создали событие event, циклически срабатывающее с частотой frequency. При каждом срабатывании генератор должен выполнить действие - послать новое сообщение типа Object через выходной порт с именем tick. Для этого введите в поле **Действие** окна свойств события следующую строку:

```
tick.send ( new Object( ) );
```

4.3.2. РАЗРЯД СЧЕТЧИКА

Введите еще один класс активного объекта в проект – Counter. Этот класс будет моделировать разряд десятичного счетчика. Он должен иметь один целый параметр – n, в котором будет храниться значение данного разряда, и два порта: tick и overflow. Создайте параметр n целого типа в классе Counter с начальным значением = 0.

4.3.2.1. ПРЕЗЕНТАЦИЯ РАЗРЯДА СЧЕТЧИКА

Разряд счетчика представим в виде прямоугольника с цифрой разряда внутри. На диаграмме класса Counter нарисуйте прямоугольник 60x100. Залейте его и линию рамки черным цветом. Внутрь прямоугольника вставьте цифру в виде текстового символа. Для этого поместите на прямоугольник текст (кнопка Aa на палитре **Презентация**) со значением 0 в поле **Текст**, вкладки **Основные**, свойств вставленного текста. Установите параметры текста: шрифт **SansSerif**, размер 72, цвет желтый. Во вкладку **Динамические** в поле **Текст** поместите имя переменной n, значение которой требуется отображать. Отметьте галочкой выбор **Значок** в свойствах текста и свойствах черного прямоугольника, рис. 4.4.

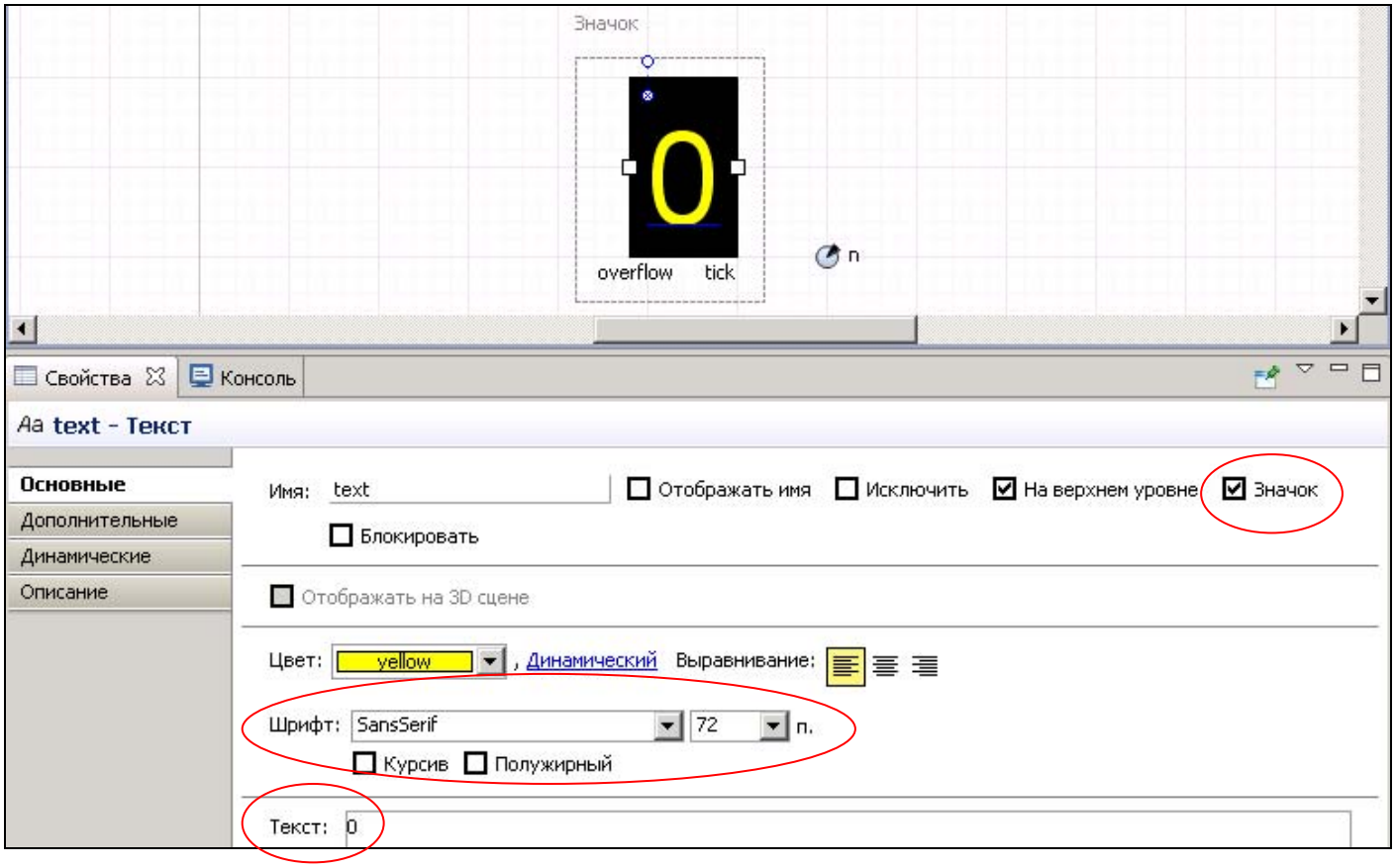


Рис. 4.4

4.3.2.2. СЧЕТ ИМПУЛЬСОВ

Создайте два порта: `tick` и `overflow` и поместите их на диаграмму класса `Counter`, как показано на рис. 4.4.

Каждый раз, когда приходит сообщение в порт `tick`, параметр `n` должен увеличиваться на единицу по модулю 10 с извещением по порту `overflow` о переполнении, если оно наступило. Это значит, что разряд счетчика увеличивает значение своего параметра `n` на 1 при приеме каждого сообщения, пришедшего через порт `tick`, кроме случая, когда `n` равно 9. В этом случае параметр `n` должен принять значение 0, а через выходной порт `overflow` будет послано сообщение. Именно это следует записать в поле **Действие при получении** в свойствах порта `tick` (рис. 4.5).

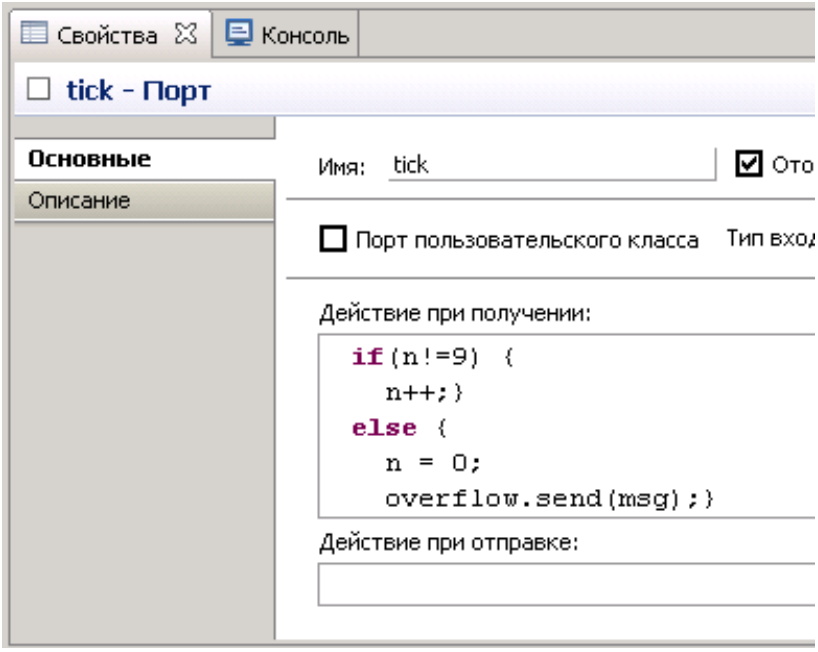


Рис. 4.5

Через порт `overflow`, таким образом, будет передаваться каждое десятое сообщение, полученное в порт `tick`.

4.3.3. КОРНЕВОЙ ОБЪЕКТ

Структура корневого объекта `Model` должна содержать один экземпляр активного объекта `Gen` и три экземпляра разряда счетчика `Counter`, соединенных по соответствующим портам.

Для создания экземпляра генератора в корневом объекте, откройте диаграмму класса активного объекта `Model` и перетащите на нее один экземпляр генератора (нажав левую клавишу мыши на имени `Gen`), а затем и три экземпляра счетчика – `Counter`. Экземпляры активных объектов получат имена по умолчанию и будут выглядеть так, как показано на рис. 4.6.

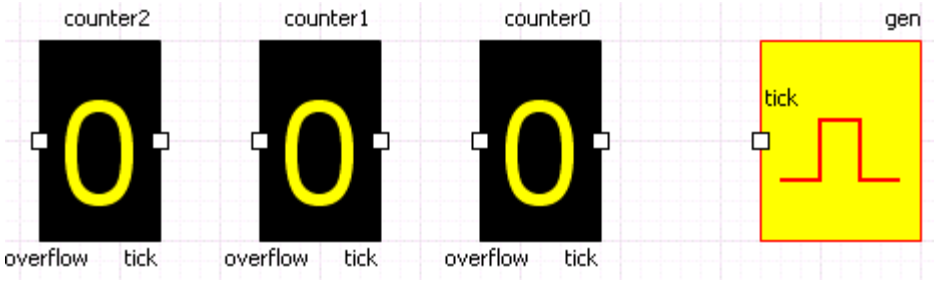
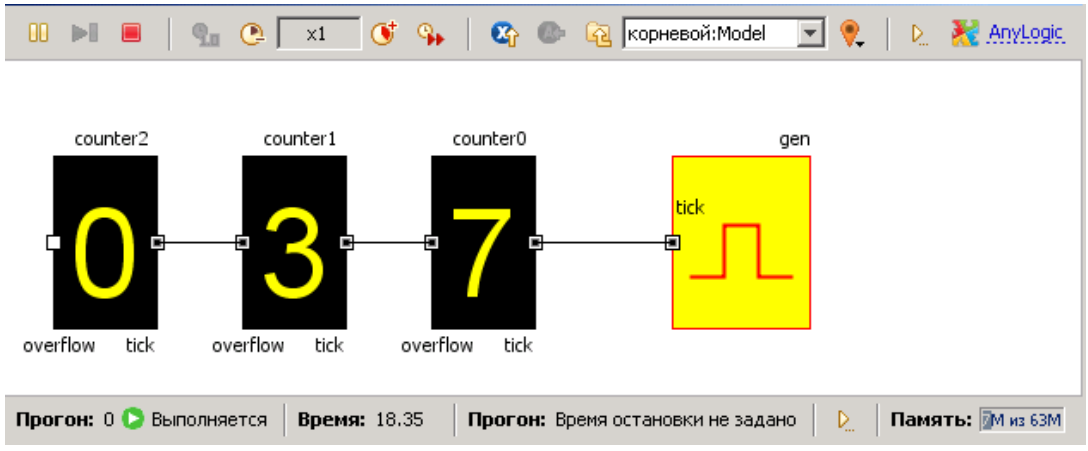


Рис. 4.6

Чтобы установить взаимодействие между объектами, Вам нужно соединить порты этих объектов с помощью соединителей. Соединитель - это линия, соединяющая два порта.

1. Сделайте двойной щелчок мышью по одному из портов.
2. Последовательно щелкните в тех местах диаграммы, где Вы хотите поместить точки изгиба соединителя (для нашей модели не обязательно).
3. Завершите процесс соединения, сделав двойной щелчок мышью по второму порту.



Запустите модель. Вы увидите изменение во времени разрядов счетчика (рис. 4.7).

Рис. 4.7

4.3.3.1. ПРЕЗЕНТАЦИЯ КОРНЕВОГО ОБЪЕКТА

Для придания созданной модели законченного вида следует добавить элементы описания и интерактивности.

Поместите в модель описание и слайдер, регулирующий частоту генератора, как показано на рис. 4.8. Продемонстрируйте свою модель преподавателю.

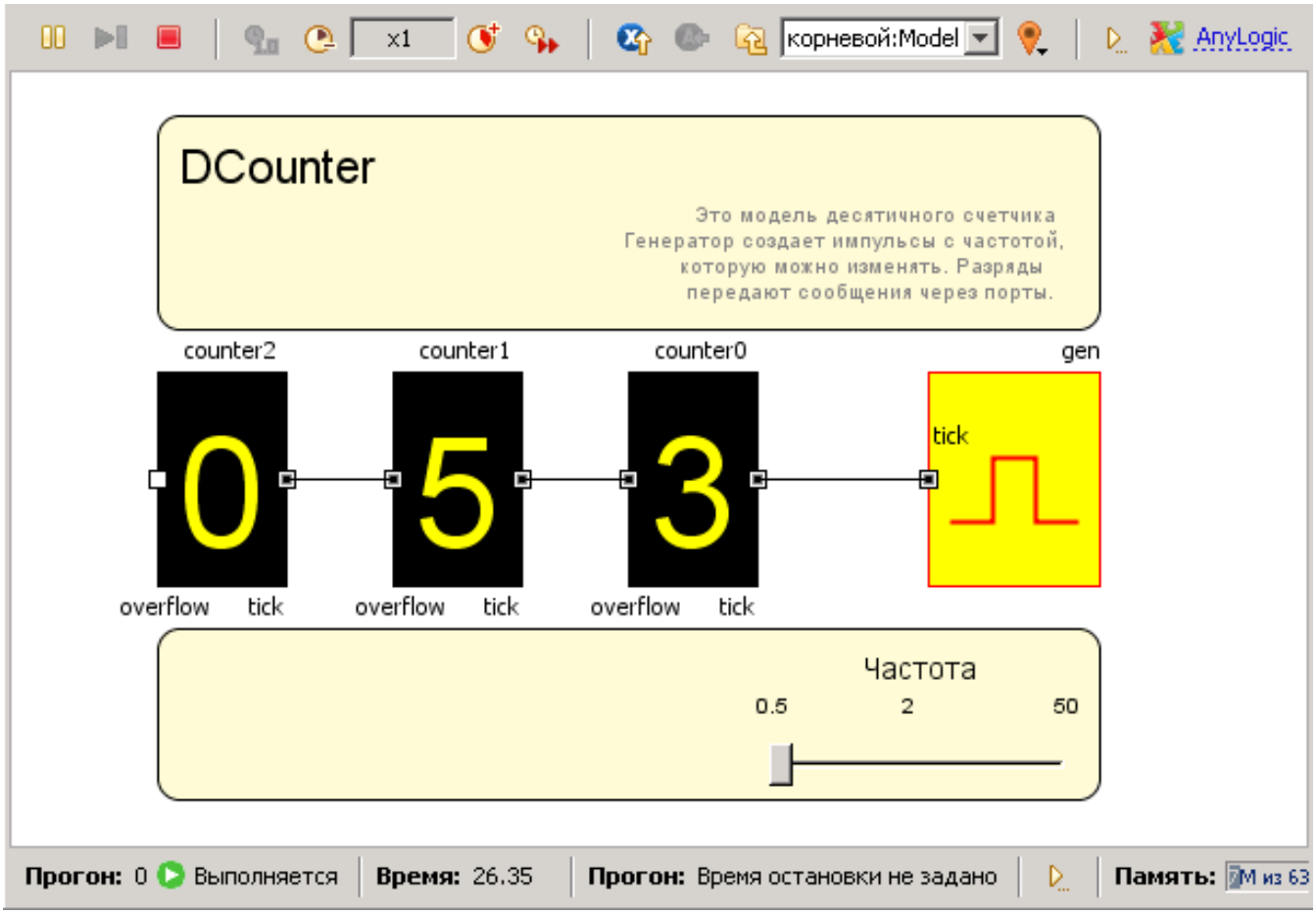


Рис. 4.8

4.4. КОНТРОЛЬНЫЕ ЗАДАНИЯ

1. Измените модель таким образом, чтобы счетчик начинал счет с произвольного числа. (5+)
2. Добавьте в модель переменную, которая будет принимать значения, показываемые счетчиком. (5+)
3. Измените модель таким образом, чтобы счетчик работал на убывание с определенного числа. (5+)
4. Измените презентацию модели таким образом, чтобы четные цифры показывались одним цветом, а нечетные - другим. (5)
5. Измените модель таким образом, чтобы параллельно основному счетчику работал второй счетчик, считающий каждый седьмой импульс. (5)
6. Измените презентацию модели таким образом, чтобы цифры мигали с частотой, которую можно изменять слайдером. (5)
7. Измените модель таким образом, чтобы цвет фона при значениях разряда счетчика = 0 был красным, а при остальных – синим. (5)
8. Измените модель таким образом, чтобы параллельно основному счетчику работал второй счетчик, считающий каждый четный импульс. (5)
9. Создайте модель 16-ричного счетчика, работающего параллельно с десятичным. (5)
10. Измените модель таким образом, чтобы параллельно основному - десятичному счетчику, работал двоичный. (4)
11. Измените модель таким образом, чтобы графическое изображение импульса генератора мигало синхронно генерации «тиков». (4)
12. Измените модель таким образом, чтобы счетчик работал на убывание. (4)

13. Измените модель таким образом, чтобы генератор создавал «тики» в случайные моменты времени. (4)
14. Измените презентацию модели таким образом, чтобы каждая цифра счетчика показывалась своим цветом. (4)
15. Дайте определение дискретно-событийной системы, приведите примеры. (3)
16. Охарактеризуйте все типы событий, реализованных в AnyLogic. (3)
17. Что такое порт, и какие элементы AnyLogic могут иметь порт? (3)
18. Добавьте еще один разряд к счетчику. (3)
19. Добавьте второй счетчик, работающий вместе с первым от одного генератора. (3)

ЗАДАНИЯ 5 – 6

СТЕЙТЧАРТЫ: МОДЕЛЬ ПЕШЕХОДНОГО ПЕРЕХОДА

ЦЕЛИ ЗАНЯТИЯ

В результате построения этой модели будут рассмотрены следующие новые вопросы:

- построение стейтчартов;
- действия при входе и выходе из состояния, иерархические состояния;
- переход по исчерпании таймаута;
- переход при получении сообщения;

ФОРМА ОРГАНИЗАЦИИ ЗАНЯТИЯ

Фронтальная.

СТУДЕНТ ДОЛЖЕН ЗНАТЬ

- понятия: сообщение, таймаут, переход,
- состояния в стейтчартах и их виды,
- основы алгоритмического языка Java,
- интерфейс программы AnyLogic.

СТУДЕНТ ДОЛЖЕН УМЕТЬ

- выполнять лабораторно-практическое задание №4,
- создавать модели в программе AnyLogic,
- использовать стейтчарты.

ОБЕСПЕЧЕННОСТЬ

- компьютер с установленной программой AnyLogic версии 6,
- настоящий курс лабораторно-практических работ.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

Построим модель регулируемого пешеходного перехода со светофором, разрешающим или запрещающим движение транспорта.

5.1. ОПИСАНИЕ ПРОБЛЕМЫ

Светофор, регулирующий движение автомобилей на пешеходном переходе, может находиться в следующих состояниях: движение транспорта разрешено (зеленый), приготовиться к запрещающему сигналу (мигающий зеленый), приготовиться к остановке (желтый), движение запрещено (красный) и приготовиться к движению (красный и желтый).

Светофор работает в автоматическом режиме. В каждом состоянии светофор находится определенный постоянный период времени.

5.2. ПОСТРОЕНИЕ МОДЕЛИ

Создайте новый проект под названием `Svetofor` и назовите класс корневого активного объекта `Model`.

Наша модель будет иметь только один активный объект, представляющий светофор, поэтому корневой объект `Model` будет единственным активным объектом нашей модели. На диаграмму класса активного объекта `Model` поместите **Начало диаграммы состояний** из панели **Диаграмма состояний** и назовите ее `Для_автомобилей`, заметьте, что AnyLogic может работать с элементами, набранными кириллицей. Перетащите мышью элемент **Состояние** под стрелочку начала диаграммы как показано на рис. 5.1.

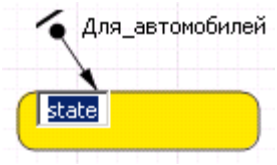


Рис. 5.1

Имя состояния, как и все другие его параметры, можно редактировать в окне его свойств.

Для того чтобы построить стейтчарт, следует использовать элементы из палитры **Диаграмма состояний**, рис. 5.2.

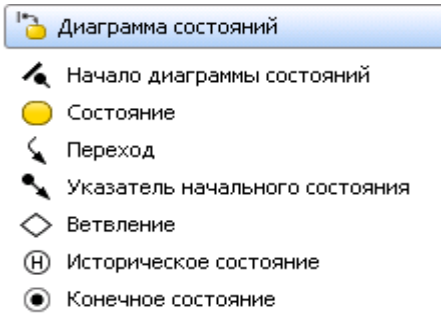


Рис. 5.2

1. определяет начальное состояние всего стейтчарта,
2. с помощью кнопки рисуются состояния (как простые, так и гиперсостояния),
3. используется для рисования переходов между состояниями,
4. определяет начальное состояние внутри сложного состояния,
5. используется для рисования состояния, являющегося "финальным" в поведении активного объекта.

Заметьте, что для любого выделенного объекта внизу появляется панель его свойств, в котором можно изменить параметры и, в частности, имя объекта, если это необходимо. Структурные ошибки при рисовании стейтчарта - повисшие переходы, дублированные указатели начального состояния и т. п. - выделяются в панели **Проекты** значком X красного цвета и записью в панели **Ошибки**. Выделенные переходы должны иметь на концах зеленые точки, если точки белые, это значит, что переход не соединен с состоянием – висит.

В соответствии с алгоритмом работы светофора помимо начального состояния в модель нужно ввести дополнительные состояния (рис. 5.3). Начальное состояние назовите `движение` – движение автомобилям разрешено (зеленый свет), затем светофор переходит в состояние `внимание` – внимание (мигающий зеленый), `медленно` – приготовиться к остановке (желтый свет), остановка транспорта `stop` – запрет движения (красный свет) и `приготовиться` – приготовиться к движению (красный и желтый свет горят одновременно).

Состояние `внимание` представим гиперсостоянием с парой переключающихся элементарных состояний: в одном из них зеленый горит (состояние `A`), в другом – нет (состояние `B`). Для построения гиперсостояния сначала создайте обычное состояние, увеличьте его (растянув мышью) и поместите внутрь другое состояние. Постройте эти

состояния и соедините их переходами, как показано на рис. 5.3.

Зададим условия срабатывания переходов. Переходы в нашем автоматическом светофоре выполняются по таймауту, т. е. по истечении интервала времени, который прошел с момента прихода системы в данное состояние.

1. В состоянии `движение` светофор находится 10 секунд,
2. затем 7 секунд зеленый сигнал мигает,
3. в состоянии `медленно` 4 секунды горит желтый,
4. в течение 10 секунд движение запрещено и
5. 4 секунды светофор находится в состоянии `приготовиться`.

В нашей модели единица модельного времени соответствует 1 секунде реального времени.

Для задания условий срабатывания переходов, выделите переход `t1`, и в поле **Происходит** оставьте без изменения вариант **По таймауту**, а в поле **По таймауту** введите 10 (рис. 5.4).

Аналогично задайте условия срабатывания других переходов. Между состояниями `A` и `B` переходы должны срабатывать через 1 секунду.

Рис. 5.3

Запустите модель. Активное в данный момент состояние подсвечивается красным. Переход, ожидающий истечения таймаута подсвечивается синим. Проведите эксперименты с моделью при различных масштабах времени.

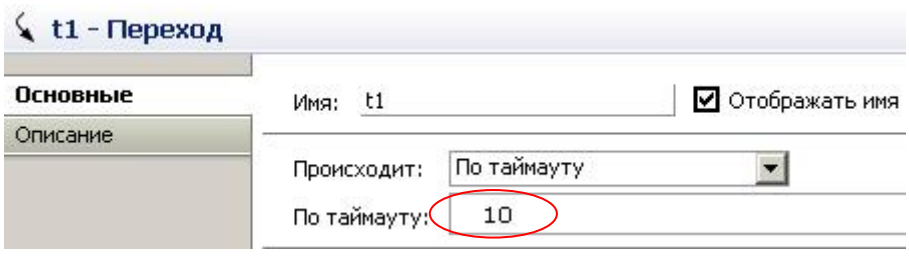
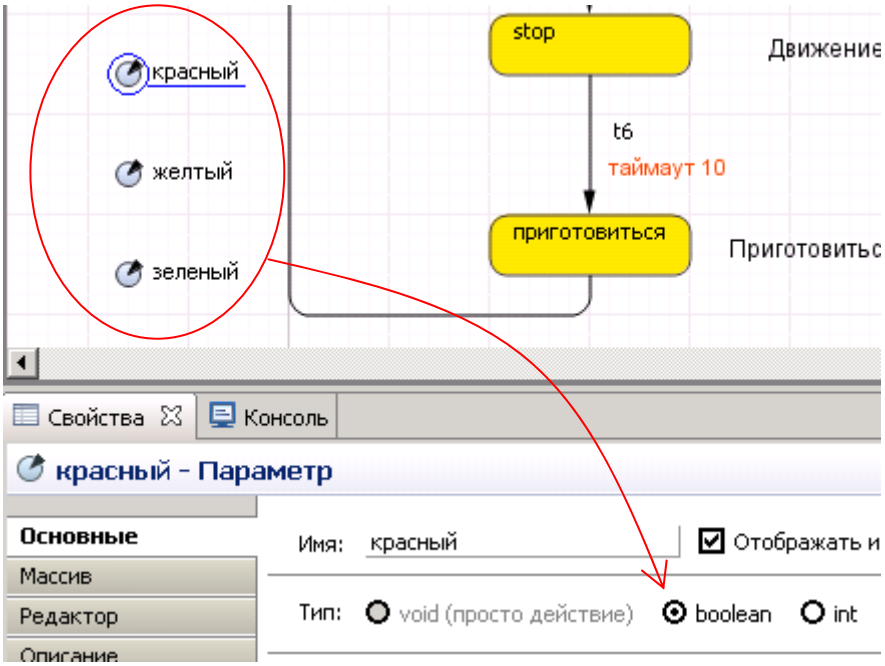


Рис. 5.4

В каждом состоянии светофора должен гореть определенный сигнал: в состоянии `движение` должен гореть зеленый, в состоянии `приготовиться` должны гореть красный и желтый одновременно и т. п. Перейдите на диаграмму класса активного объекта `Model`. Создайте три параметра логического типа: `красный`, `желтый` и `зеленый`, которые будут

принимать истинное значение тогда, когда у светофора горит соответствующий сигнал: красный, желтый или зеленый (рис. 5.5). Начальные значения этих булевых параметров можно не задавать: по умолчанию они будут равны `false`.



Мы создали стейтчарт именно для управления значениями этих параметров, каждое состояние отвечает за зажигание своего света или их комбинации. Например, в состоянии `медленно` должен гореть желтый, в состоянии `stop` должен загореться красный свет, а в состоянии `приготовиться` должны гореть красный и желтый одновременно. Запрограммируем эти действия.

Рис. 5.5

1. В свойствах состояния `движение` в поле **Действие при входе** запишите `зеленый=true;`, а в поле **Действие при выходе** запишите `зеленый = false;` (рис. 5.6).
2. То же самое запишите для состояния `B` гиперсостояния `внимание`, а у состояния `A` эти поля нужно оставить пустыми – когда светофор находится в этом состоянии, он не горит.
3. Аналогично, в состоянии `медленно` нужно включить желтый сигнал, т. е. при входе в это состояние установить параметр `желтый` в `true`, а при выходе из этого состояния установить его в `false`.
4. Для состояния `stop` аналогично опишите состояния параметра `красный`,
5. а для состояния `приготовиться` оба параметра `красный` и `желтый` нужно установить в `true` при входе, и в `false` при выходе.

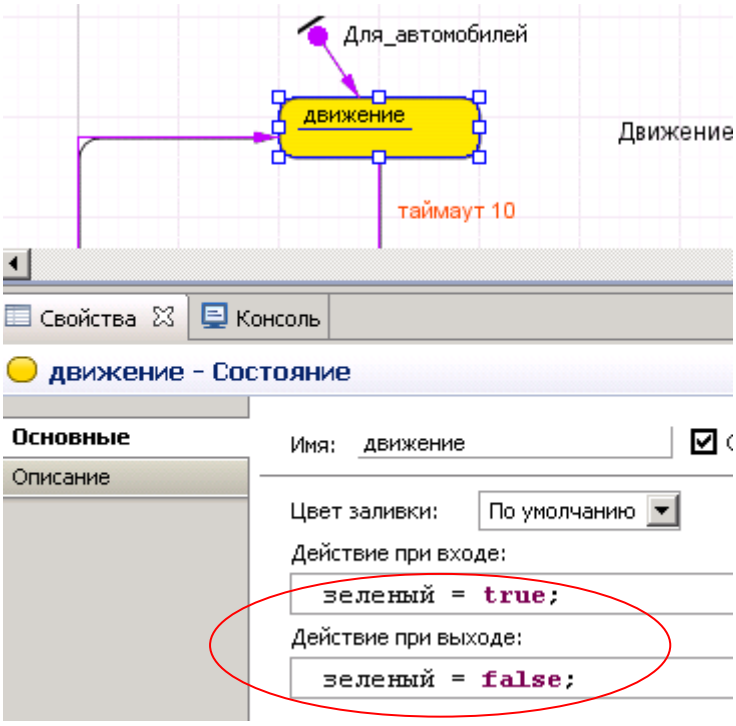


Рис. 5.6

Запустите модель на выполнение. Вы увидите, что параметры `зеленый`, `желтый` и

`красный` будут переключаться между значениями `истина` и `ложь` в соответствии с алгоритмом переключения светофора.

5.3. ПРЕЗЕНТАЦИЯ МОДЕЛИ

Презентация модели рисуется в той же диаграмме (в графическом редакторе), в которой задается и диаграмма моделируемого процесса, (рис. 5.7). Графические объекты цвета сигналов светофора в презентации имеют динамические параметры, все остальные – статические. Проезжую часть удобно нарисовать с помощью графических примитивов типа **Прямоугольник**, а светофор с помощью примитива - **Ломаная**. Сигнальные элементы светофора строятся из трех овалов, повернутых на 45 градусов (поле **Поворот** вкладки **Дополнительные** окна свойств овала).

Установим динамическое значение цвета верхнего сигнала светофора: если переменная `красный` истинна, то цвет должен быть `red` (красный), в противном случае его цвет нужно установить `gray` (серый). Это записывается следующим условным выражением на языке Java:

```
красный? red: gray
```

Цвет среднего и нижнего овалов, следует установить в поле их динамических значений соответственно так:

```
желтый? yellow: gray
```

```
зеленый? green: gray
```

red, yellow, green и gray – предопределенные константы, обозначающие стандартные цвета.

Запустите модель и проверьте ее работу.

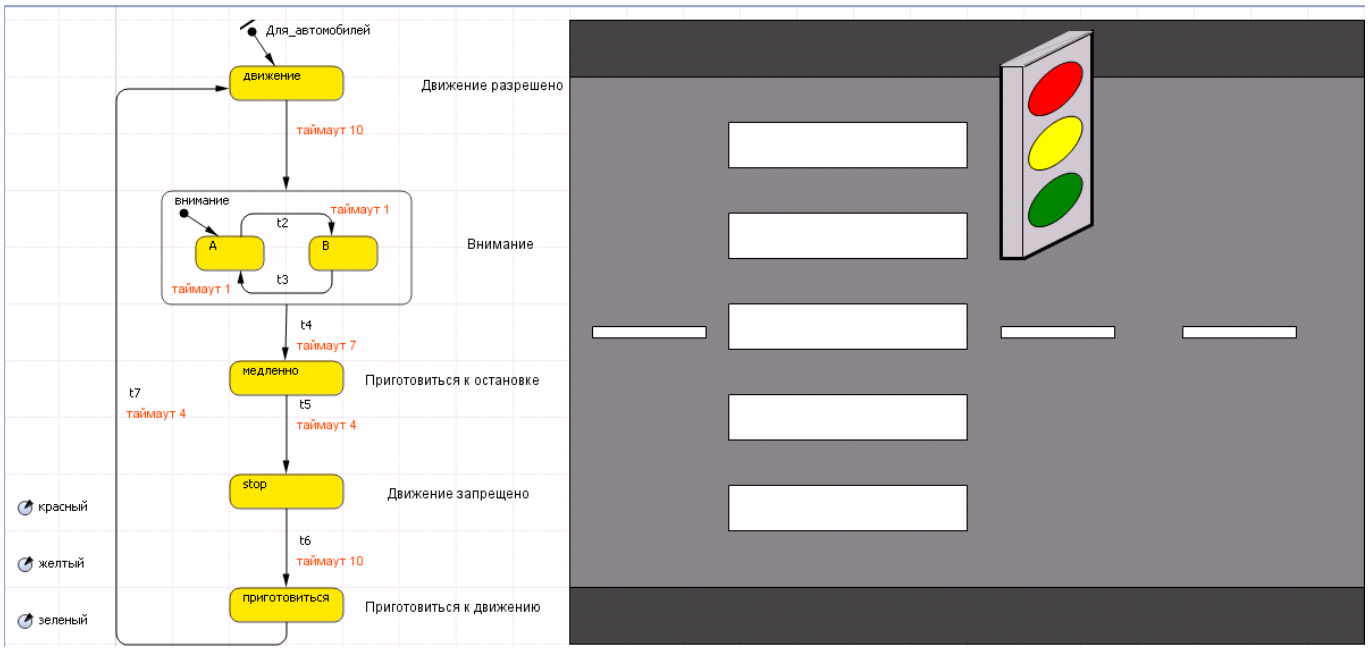


Рис. 5.7

5.4. СРАБАТЫВАНИЕ ПЕРЕХОДА ПО СИГНАЛУ

Добавим к нашей модели второй светофор, для пешеходов. Он будет иметь два сигнала, зеленый и красный, и три состояния: идите (зеленый), внимание (мигающий зеленый) и стойте (красный). Добавим в модель два булевских параметра `стойте` и `идите`, их значениями будет управлять второй стейтчарт – для пешеходов. Создадим этот стейтчарт на той же диаграмме класса `Model`, назвав его `Для_пешеходов`, (рис. 5.8).

Поскольку управление светофором пешеходов похоже на управление светофором автомобилей, новый стейтчарт можно построить копированием и изменением уже построенного стейтчарта `Для_автомобилей`. Выделите 3 состояния стейтчарта `Для_автомобилей` и вставьте их в другое место диаграммы. Эти элементы скопируются, и им автоматически будут даны уникальные имена, чтобы не было конфликта имен. Переименуйте состояния стейтчарта `Для_пешеходов`, дорисуйте недостающий переход `t12` и перенесите начало диаграммы на состояние `Стойте`, как показано на рис. 5.8.

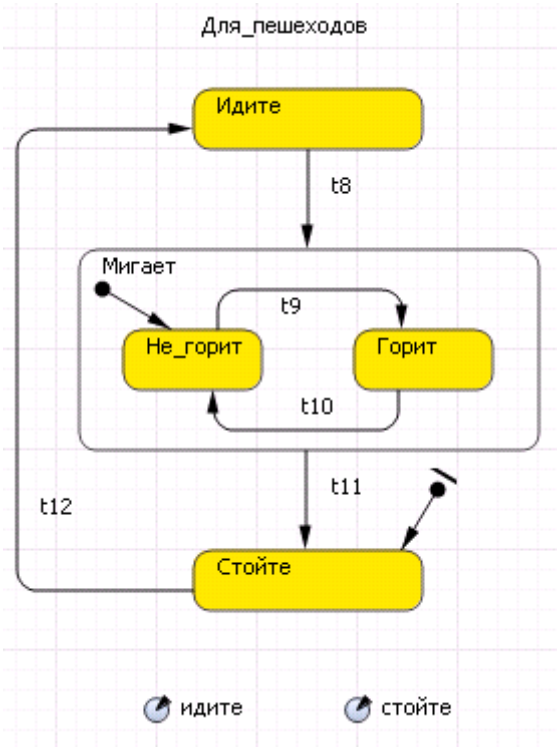


Рис. 5.8

Измените параметры в полях **Действие при входе** и **Действие при выходе** в свойствах состояний стейтчарта `Для_пешеходов`. Теперь наш стейтчарт должен управлять параметрами `идите` и `стойте`, которые, в свою очередь, будут управлять зажиганием света именно пешеходного светофора.

Настроим условия срабатывания переходов стейтчартов между состояниями. Для обеспечения безопасной работы пешеходного перехода необходимо синхронизировать срабатывания стейтчартов так, чтобы всегда, когда светофор пешеходов находится в состояниях `Идите` или `Мигает`, светофор автомобилей обязательно находился бы в состоянии `stop`. Для этого можно подобрать подходящие таймауты срабатывания переходов стейтчарта, но при каждом изменении модели, придется эти таймауты подбирать снова и снова. Более разумно синхронизировать стейтчарты, посылая специальные разрешающие сигналы из одного стейтчарта в другой.

В нашей модели стейтчарты будут обмениваться следующими сигналами: `АВТОМОБИЛИ` и `ПЕШЕХОДЫ`. В стейтчарте `Для_пешеходов` переход `t12` будет срабатывать когда получен сигнал `ПЕШЕХОДЫ`, который будет генерироваться в стейтчарте `Для_автомобилей` при переходе `t5` в состояние `stop`. В свою очередь, в стейтчарте `Для_автомобилей` переход `t6` будет срабатывать когда получен сигнал `АВТОМОБИЛИ`, который генерируется в стейтчарте `Для_пешеходов` при переходе `t11` в состояние `Стойте`, рис. 5.9.

В AnyLogic есть несколько способов передачи сообщения в диаграмму состояний. В нашей модели мы будем использовать метод `fireEvent( )`, который должен вызываться в том стейтчарте, которому предназначено сообщение. То есть, если из некоего объекта мы хотим послать сообщение стейтчарту, то нужно в этом объекте написать команду:

`стейтчарт.fireEvent(сообщение)`

Поэтому, в поле **Действие** перехода `t5` стейтчарта `Для_автомобилей` нужно вставить команду:

```
Для_пешеходов.fireEvent("ПЕШЕХОДЫ"),
```

в такое же поле перехода `t11` стейтчарта `Для_пешеходов` вставьте команду:

```
Для_автомобилей.fireEvent("АВТОМОБИЛИ")
```

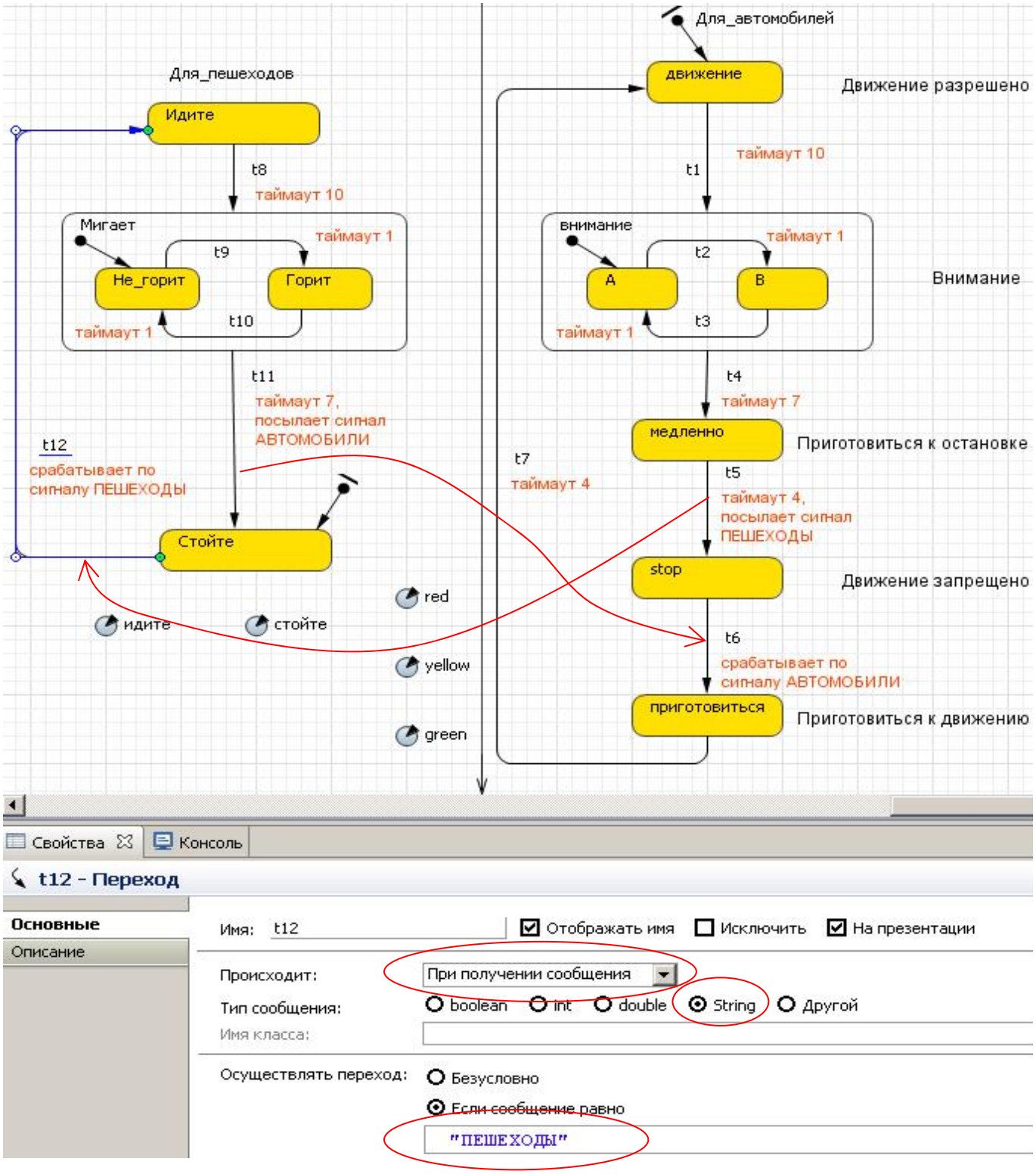


Рис. 5.9

Таким образом, каждый из светофоров будет информировать другого о своем переходе в состояние запрещения движения, как пешеходов, так и автомобилей.

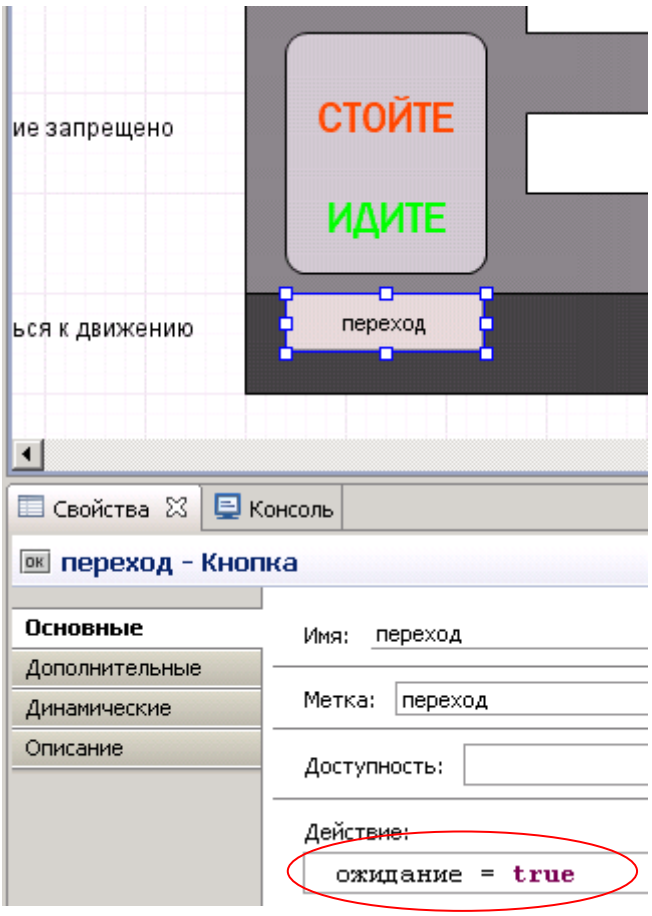
Для срабатывания перехода стейтчарта при получении нужного сообщения, в стейтчарте `Для_пешеходов` в поле **Происходит** окна свойств перехода `t12` выберите вариант **При получении сообщения**, укажите тип сообщения `String`, а в поле **Осуществлять переход** выберите `Если сообщение равно` и введите "`ПЕШЕХОДЫ`", рис. 5.9.

Аналогично, для срабатывания перехода автомобильного стейтчарта по сигналу от пешеходного стейтчарта в стейтчарте `Для_автомобилей` в поле **Происходит** окна свойств перехода `t6` выберите вариант **При получении сообщения**, укажите тип сообщения `String`, а в поле **Осуществлять переход** выберите `Если сообщение равно` и введите "`АВТОМОБИЛИ`".

Остальные переходы этих стейтчартов будут срабатывать по таймаутам, как и прежде. Проверьте по рисунку 5.9 установленные параметры переходов стейтчартов. Запустите модель на выполнение.

На презентации модели, в дополнение к светофору для автомобилей, следует нарисовать светофор для пешеходов с двумя сигналами – красной надписью СТОЙТЕ и зеленой ИДИТЕ. Динамикой цвета этих надписей будут управлять логические параметры `стоите` и `идите`, которые нужно создать на диаграмме по аналогии с параметрами `красный`, `желтый` и `зеленый`. Продемонстрируйте построенную модель преподавателю.

5.5. СРАБАТЫВАНИЕ ПЕРЕХОДА ПО УСЛОВИЮ



Если пешеходы переходят улицу достаточно редко, то автоматическое переключение светофора будет неоправданно часто прерывать движение автомобилей. В этом случае разумно установить специальную кнопку, при нажатии которой пешеходом, светофор автомобилей остановит на некоторое время движение автомашин. Для этого перетащите мышью элемент **Кнопка** с палитры **Элементы управления** на диаграмму класса активного объекта `Model` рядом с пешеходным светофором, рис. 5.10. Назовите кнопку и ее метку: **переход.**

Определите действие при нажатии на кнопку так, чтобы устанавливался в истину булевый параметр `ожидание`. Для этого нужно записать в поле **Действие** окна свойств этой кнопки команду: `ожидание = true`.

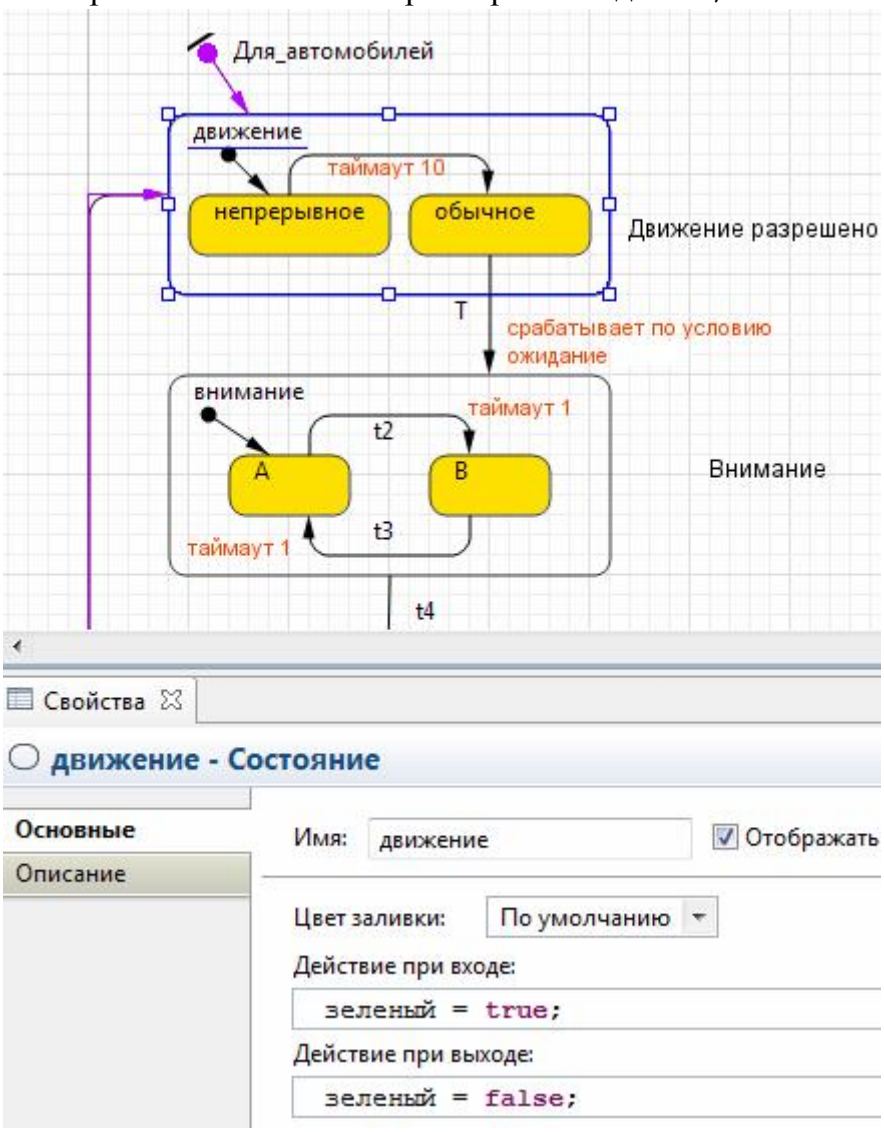
Рис. 5.10

Логический параметр ожидание, конечно же, нужно ввести на диаграмме активного объекта Model с начальным значением false. Этот параметр будет определять, собирается ли пешеход перейти дорогу. Значение этого параметра будем переводить в false каждый раз, как только пешеходный светофор перейдет в состояние Мигает. Для этого нужно записать в поле **Действие при входе** окна свойств состояния Мигает стейтчарта Для_пешеходов команду: ожидание=false.

Таким образом, при нажатии кнопки **переход,** стейтчарт автомобилей «узнает» об ожидающих на переходе пешеходах и переключится из состояния Движение в состояние Внимание и затем в состояние запрещения движения автомобилей.

В случае, если некий злоумышленник будет постоянно нажимать кнопку **переход,** это может полностью парализовать движение автомобилей. Для того чтобы этого не произошло, в стейтчарте Для_автомобилей сделайте иерархическим состояние Движение, с двумя простыми состояниями непрерывное и обычное, как показано на рис. 5.11.

В состоянии непрерывное автомобили будут двигаться до истечения таймаута 10 секунд, невзирая на состояние параметра ожидание, т.е. автомобилям будет гарантированно предоставлено некоторое время для движения, даже если кнопка **переход** будет всегда нажата. После истечения таймаута 10 секунд светофор перейдет в обычное состояние движения, которое может прерываться. Переход T из состояния обычное сработает, когда будет нажата кнопка **переход**, т.е. параметр ожидание будет истинен.



Если кнопка **переход** не нажата (т. е. параметр ожидание имеет значение false), автомобили будут продолжать движение до нажатия этой кнопки. Если нажать кнопку один или более раз, то параметр ожидание станет истинным и автомобильный светофор из состояния обычное перейдет в состояние внимание и затем остановит движение автомобилей.

Рис. 5.11

Проверьте работу модели и продемонстрируйте ее преподавателю.

5.6. КОНТРОЛЬНЫЕ ЗАДАНИЯ

1. Измените модель светофора с кнопкой таким образом, чтобы моделировалось автоматическое нажатие кнопки **ПЕРЕХОД** в случайные моменты времени в интервале от 5 секунд до 1 минуты. (5+)
2. Измените модель автоматического светофора таким образом, чтобы по кнопке **НОЧЬ** включался режим мигания желтого света, а по кнопке **ДЕНЬ** светофор переходил в нормальный режим работы. (5+)
3. Измените модель светофора с кнопкой таким образом, чтобы по кнопке **ВЫКЛ** светофор выключался (все серое), а по кнопке **ВКЛ** светофор переходил в нормальный режим работы. (5)
4. Измените презентацию модели таким образом, чтобы рядом со светофором высвечивалось время в секундах, оставшееся до смены сигнала. (5)
5. Измените презентацию модели таким образом, чтобы рядом со светофором высвечивалось время в секундах, прошедшее со смены сигнала. (5)
6. Измените модель автоматического светофора таким образом, чтобы по кнопке **СТОП** включался красный свет у всех светофоров, а по кнопке **ДВИЖЕНИЕ** светофор переходил в нормальный режим работы. (5)
7. Измените модель таким образом, чтобы регулировалось движение двух пересекающихся дорожных потоков в автоматическом режиме (без кнопки), т.е. модель двух трехцветных светофоров со следующей последовательностью сигналов: зеленый – зеленый мигающий – желтый – красный – (красный + желтый) – зеленый. (4)
8. Измените модель светофора с кнопкой таким образом, чтобы пешеход видел, нажата уже кнопка или нет, например, с помощью надписи. (4)
9. Измените модель таким образом, чтобы моделировать железнодорожный переезд: по кнопке **ПОЕЗД** включается сигнал и через 5 сек. начинает мигать сдвоенный красный светофор (попеременно правый – левый), еще через 5 сек. опускается шлагбаум. По истечении случайного промежутка времени в диапазоне от 30 до 60 сек. (прохождение поезда), переезд переводится в исходное состояние. (4)
10. Измените модель таким образом, чтобы 3-х цветный светофор переключался в полуавтоматическом режиме по нажатию кнопок регулировщиком: по кнопке **КРАСНЫЙ** начинает мигать зеленый свет, затем зажигается желтый и через несколько сек. – красный; по кнопке **ЗЕЛЕНЫЙ** зажигается одновременно красный и желтый, затем горит зеленый. (4)
11. Измените модель таким образом, чтобы частота мигания зеленого света регулировалась с помощью слайдера. (3)
12. Измените модель автоматического светофора автомобилей и пешеходов (без кнопки), удалив переходы по сигналу и подберите подходящие таймауты для согласованной работы светофоров. (3)
13. Измените модель светофора с кнопкой таким образом, чтобы с помощью слайдера можно было регулировать время, через которое повторное нажатие кнопки **ПЕРЕХОД** приведет к переключению светофора автомобилей в режим ожидания. (3)
14. Как передать сигнал от стейтчарта CT_1 стейтчарту CT_2 ? (3)
15. Измените модель таким образом, чтобы время, в течение которого разрешено движение автомобилей, было 15 секунд и перед окончанием запрещающего сигнала автомобилям мигал красный свет. (3)
16. В чем отличие  от  ? (3)
17. Измените модель таким образом, чтобы время, в течение которого разрешено

движение пешеходов, было 10 секунд и перед окончанием запрещающего сигнала пешеходам мигал красный свет. (3)

18. Как в модели реализована защита от чрезмерно-частого нажатия кнопки **ПЕРЕХОД**? (3)
19. Измените модель таким образом, чтобы зеленый сигнал светофора не мигал. (3)

ЗАДАНИЕ 7

ФИЗИЧЕСКИЙ МАЯТНИК

ЦЕЛИ ЗАНЯТИЯ

- освоение методов самостоятельного построения имитационных моделей физических систем, исходя из содержательной, концептуальной и математических постановок задачи;
- построение имитационной модели маятника.

ФОРМА ОРГАНИЗАЦИИ ЗАНЯТИЯ

Фронтальная.

СТУДЕНТ ДОЛЖЕН ЗНАТЬ

- порядок и способы построения модели,
- интерфейс программы AnyLogic.

СТУДЕНТ ДОЛЖЕН УМЕТЬ

- создавать модели в программе AnyLogic,
- создавать презентацию модели,
- запускать модель на выполнение.

ОБЕСПЕЧЕННОСТЬ

- компьютер с установленной программой AnyLogic версии 6,
- настоящий курс лабораторно-практических работ.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

В этом задании мы построим модель маятника. Колебательные движения механических систем широко распространены в технике: качания маятников, движения поршней двигателей внутреннего сгорания, колебания струн, стержней и пластин, вибрации двигателей, фундаментов и множество других подобных процессов.

7.1. СОДЕРЖАТЕЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ

Тело единичной массы прикреплено к неподвижному кронштейну с помощью нерастяжимой и несжимаемой нити длиной `l`. Исследовать колебательные движения тела.

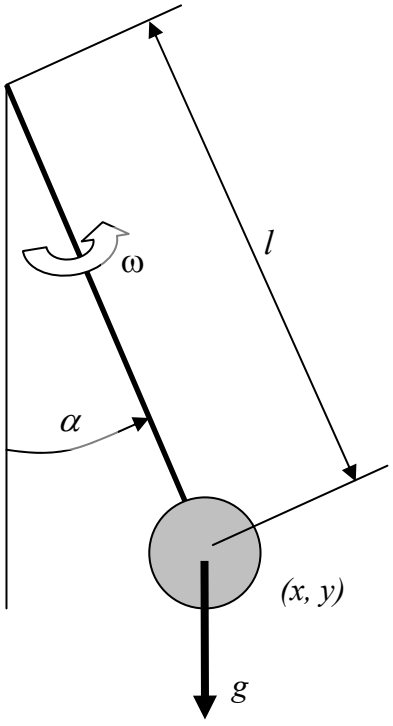


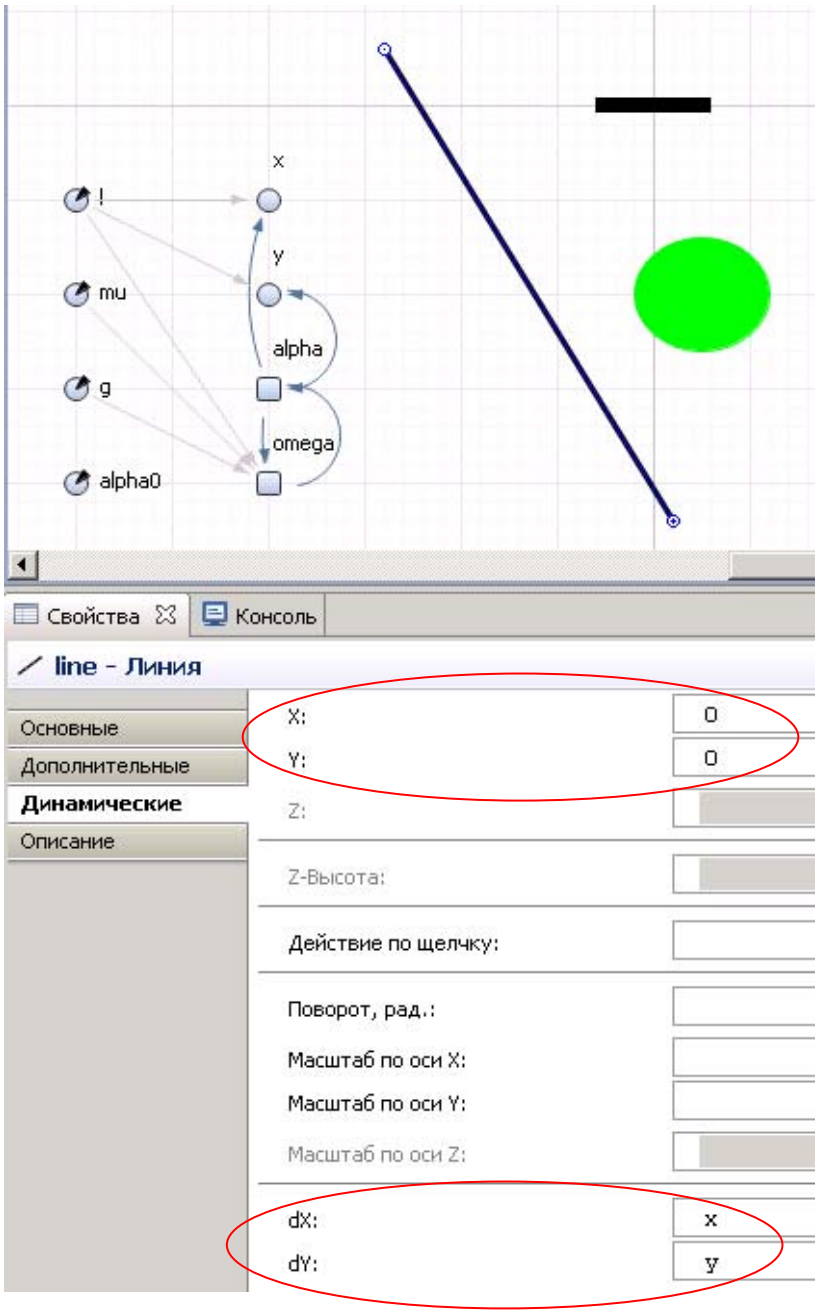
Рис. 7.1

7.2. КОНЦЕПТУАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ

Примем следующие предположения:

- объектом исследования является тело массой 1, принимаемое за материальную точку;
- движение тела подчиняется второму закону Ньютона;
- тело находится под действием трех сил: силы тяжести `mg`, реакции натяжения нити и силы сопротивления воздуха, пропорциональной квадрату скорости движения;
- тело совершает колебательные движения, так как сила тяжести `mg` уравновешивается вертикальной составляющей реакции натяжения нити;

7.3. МАТЕМАТИЧЕСКАЯ ПОСТАНОВКА ЗАДАЧИ



С математической точки зрения имеем задачу Коши.

$$\frac{d\alpha}{dt}=\omega;\qquad \frac{d\omega}{dt}=-\frac{g\cdot\sin(\alpha)}{l}-\mu\cdot\omega^2;$$
$$x=l\cdot\sin(\alpha);\qquad y=l\cdot\cos(\alpha)$$

$$\alpha(0)=\alpha 0;\qquad \omega(0)=0$$

Здесь α - текущий угол отклонения маятника от вертикали, ω - его угловая скорость, μ - коэффициент сопротивления среды (будем считать, что сопротивление среды пропорционально квадрату угловой скорости).

7.4. ОПИСАНИЕ МОДЕЛИ

Необходимо построить модель по вышеприведенным формулам. Модель должна содержать две переменных состояния `alpha` и `omega` и три параметра `l`, `mu` и `g`, а также начальное значение переменной `alpha`, которая задается параметром с именем `alpha0`. Переменные `x` и `y` задают координаты центра масс маятника, их значения можно вычислить по формулам, в соответствии со схемой рис. 7.1.

Рис. 7.2

В корневом объекте модели создайте четыре переменных: `x`, `y`, `alpha` и `omega`. Переменные `x` и `y` следует задать формулами, а `alpha` и `omega` – интегралами, в

соответствии с формулами п. 7.3. Начальное значение угловой скорости `omega` равно 0. Четыре параметра `l, mu, g` и `alpha0` так же необходимо создать на диаграмме класса активного объекта.

Обратите внимание на уравнение $\frac{d\omega}{dt} = -\frac{g \cdot \sin(\alpha)}{l} - \mu \cdot \omega^2$, при возведении ω в квадрат член уравнения $-\mu*\omega^2$ всегда будет отрицательным независимо от направления движения маятника (знака переменной ω), т.е. будет потеряно направление действия силы сопротивления, что приведет к ошибке. Необходимо видоизменить это уравнение так, чтобы знак угловой скорости не потерялся: $\frac{d\omega}{dt} = -\frac{g \cdot \sin(\alpha)}{l} - \mu \cdot \omega \cdot |\omega|$.

7.5. ПРЕЗЕНТАЦИЯ

В презентации определите четыре области. В центральной с помощью двух линий и овала нарисуйте маятник. У линии, изображающей нить, один конец всегда имеет координаты (0,0), этот конец нити привязан к неподвижной опоре. Второй конец нити имеет координаты `x` и `y`, этот конец нити привязан к маятнику.

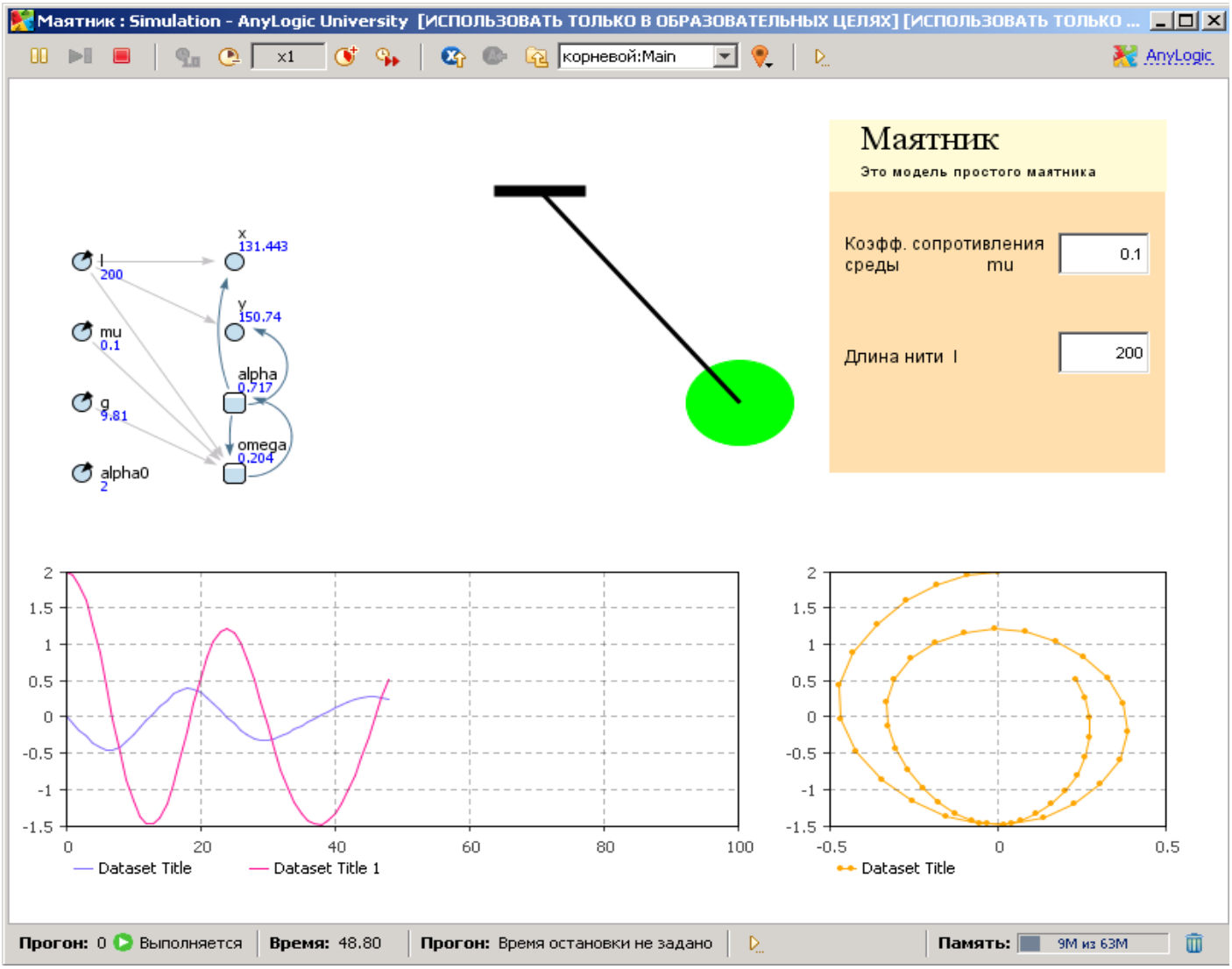


Рис. 7.3

Соответственно, на закладке **Динамические** панели свойств линии, отображающей нить, в поле значений координат начала нити **X** и **Y** введите 0, в поле значений координат конца

нити **dX** и **dY** введите величины `x` и `y`, рис.7.2. Это приведет к тому, что при работе модели концы отрезка всегда будут находиться в точках с координатами (0, 0) и (x, y). У овала во вкладке с динамическими значениями координат установите `x` и `y`.

Третья область должна содержать: название модели и небольшой поясняющий текст.

Нижняя часть третьей области должна включать два специальных поля для ввода данных: **Текстовое поле**, более известное как поле редактора или editBox, В эти поля при работе модели можно вводить новые значения параметров `l` и `mu`, изменяя их в процессе выполнения модели.

В нижней части диаграммы класса активного объекта разместите графики зависимости `alpha` и `omega` от времени, и фазовую диаграмму зависимости `alpha` от `omega`.

Постройте модель как показано на рис. 7.3 и продемонстрируйте ее преподавателю.

7.6. КОНТРОЛЬНЫЕ ЗАДАНИЯ

1. Добавьте в модель второй маятник, закрепленный в той же точке и смоделируйте упругое соударение маятников. (5++)
2. Доработайте модель таким образом, чтобы учитывался боковой ветер. (5+)
3. Измените модель таким образом, чтобы сила тяжести была направлена вдоль произвольной оси, направление которой можно ввести текстовым полем. (5+)
4. Доработайте модель таким образом, чтобы направление вектора силы тяжести совершало кругообразное движение с произвольной угловой скоростью. (5+)
5. Доработайте модель таким образом, чтобы показывалось время с момента прохождения маятником высшей точки. (5)
6. Добавьте в модель второй маятник, качающийся независимо от первого. (5)?
7. Доработайте модель таким образом, чтобы показывалось время с момента прохождения маятником низшей точки. (5)
8. Доработайте модель таким образом, чтобы моделировался отскок маятника от препятствия (стенки), расположенной перпендикулярно траектории движения маятника. Учтите потерю скорости при отскоке. (5)
9. Доработайте презентацию модели таким образом, чтобы при остановке маятника в верхней точке траектории он менял цвет, ровно на 2 секунды. (5)
10. Доработайте модель таким образом, чтобы автоматически подсчитывалось количество колебаний маятника. (5)
11. Измените модель таким образом, чтобы сила тяжести была направлена вдоль горизонтальной оси. (4)
12. Доработайте презентацию модели таким образом, чтобы при движении вниз маятник был одного цвета, а при движении вверх – другого. (4)
13. Доработайте презентацию модели таким образом, чтобы при зависании маятника в верхней точке траектории он менял цвет. (4)
14. Измените модель таким образом, чтобы сила тяжести была направлена вверх. (3)
15. Доработайте презентацию модели таким образом, чтобы при движении направо маятник был одного цвета, а при движении налево – другого. (3)
16. Доработайте модель таким образом, чтобы маятник сопровождала движущаяся рядом надпись, показывающая мгновенное значение линейной скорости. (3)
17. В чем отличие содержательной постановки задачи от концептуальной? (3)
18. Каким элементом палитры можно задать переменные, описываемые формулой, а каким – как интегралы? (3)

ЗАДАНИЕ 8

МАЯТНИК С ОГРАНИЧИТЕЛЕМ

ЦЕЛИ ЗАНЯТИЯ

- освоение методов самостоятельного построения имитационных моделей физических систем переменной структуры;
- построение имитационной модели ограниченного маятника.

ФОРМА ОРГАНИЗАЦИИ ЗАНЯТИЯ

Фронтальная.

СТУДЕНТ ДОЛЖЕН ЗНАТЬ

- порядок и способы построения модели,
- способы работы со стейтчартами,
- интерфейс программы AnyLogic.

СТУДЕНТ ДОЛЖЕН УМЕТЬ

- выполнять задание № 7.
- создавать модели в программе AnyLogic,
- создавать презентацию модели.

ОБЕСПЕЧЕННОСТЬ

- компьютер с установленной программой AnyLogic версии 6,
- настоящий курс лабораторно-практических работ.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

В этом задании необходимо построить модель динамической системы, в которой требуется учесть как непрерывные процессы (движение маятника), так и дискретные события (изменение структуры системы при встрече с ограничителем).

8.1. ОПИСАНИЕ ПРОБЛЕМЫ

Эта модель является дальнейшим развитием предыдущей модели маятника. Маятник, (см. рис. 8.1) при соприкосновении нити с ограничителем начинает закручиваться вокруг ограничителя, т.е. в модели появляется второй центр вращения. Уравнения, описывающие движение маятника без ограничителя (на нити длиной `L+l)`и с ограничителем (на нити длиной `l)`, представлены ниже:

Движение без ограничения.

$$\frac{d\alpha}{dt} = \omega; \qquad \frac{d\omega}{dt} = -\frac{g \cdot \sin(\alpha)}{L+l} - \mu \cdot \omega^2; \qquad x = (L+l) \cdot \sin(\alpha); \qquad y = (L+l) \cdot \cos(\alpha)$$

Движение при ограничении.

$$\frac{d\alpha}{dt} = \omega; \quad \frac{d\omega}{dt} = -\frac{g \cdot \sin(\alpha)}{l} - \mu \cdot \omega^2;$$

$$x = l \cdot \sin(\alpha) + L \cdot \sin(\alpha Pin);$$

$y = l \cdot \cos(\alpha) + L \cdot \cos(\alpha Pin)$, где:

α - текущий угол отклонения маятника от вертикали,
ω - его угловая скорость,
μ - коэффициент сопротивления среды, пропорциональной квадрату угловой скорости.

Начальные условия: $\alpha(0) = \alpha 0;$
$\omega(0) = 0$

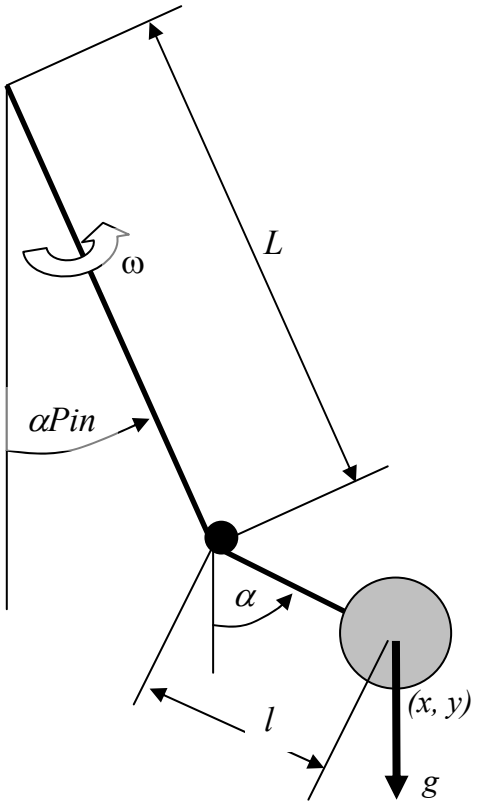


Рис. 8.1

8.2. ОПИСАНИЕ МОДЕЛИ

Маятник с ограничителем – это система, в которой происходят как непрерывное движение, так и дискретные события, изменяющие характер движения. Такая модель должна содержать уравнения, описывающие непрерывное движение, и стейтчарты, для описания дискретных событий.

Модель должна содержать четыре переменные: `x`, `y`, `alpha` и `omega`, и шесть параметров: `L`, `l`, `mu`, `alphaPin`, `g` и `alpha0`. Их физический смысл ясен из рис. 8.1. Как и в предыдущей модели, `alpha0` – начальное значение угла отклонения маятника.

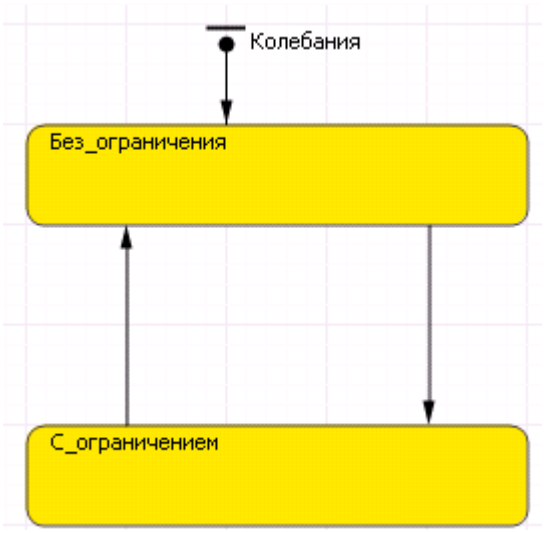


Рис. 8.2

Поведение маятника описывается стейтчартом с именем **Колебания**, рис.8.2, с двумя состояниями. В одном состоянии описывается непрерывное движение маятника без ограничения (состояние `Без_ограничения`), в другом (состояние `С_ограничением`) – описывается движение вокруг ограничителя. В зависимости от состояния, в котором находится система, переменные `x`, `y` и `omega` будут описываться различными уравнениями.

Условия перехода из одного состояния в другое, а, следовательно, изменение поведения системы определяются следующим образом. Начальное положение маятника задается его углом α0. Маятник соприкасается с ограничителем, если он движется против часовой стрелки (ω >0) и его угол равен α`Pin` или если он движется по часовой стрелке и его угол меньше, чем `-(2`π `-` α`Pin)`. Запишем это на языке Ява в условии перехода из состояния `Без_ограничения` в состояние `С_ограничением`:

```
alpha >= alphaPin && omega>0 || alpha <= -(2*PI-alphaPin) && omega <0
```

логическое И

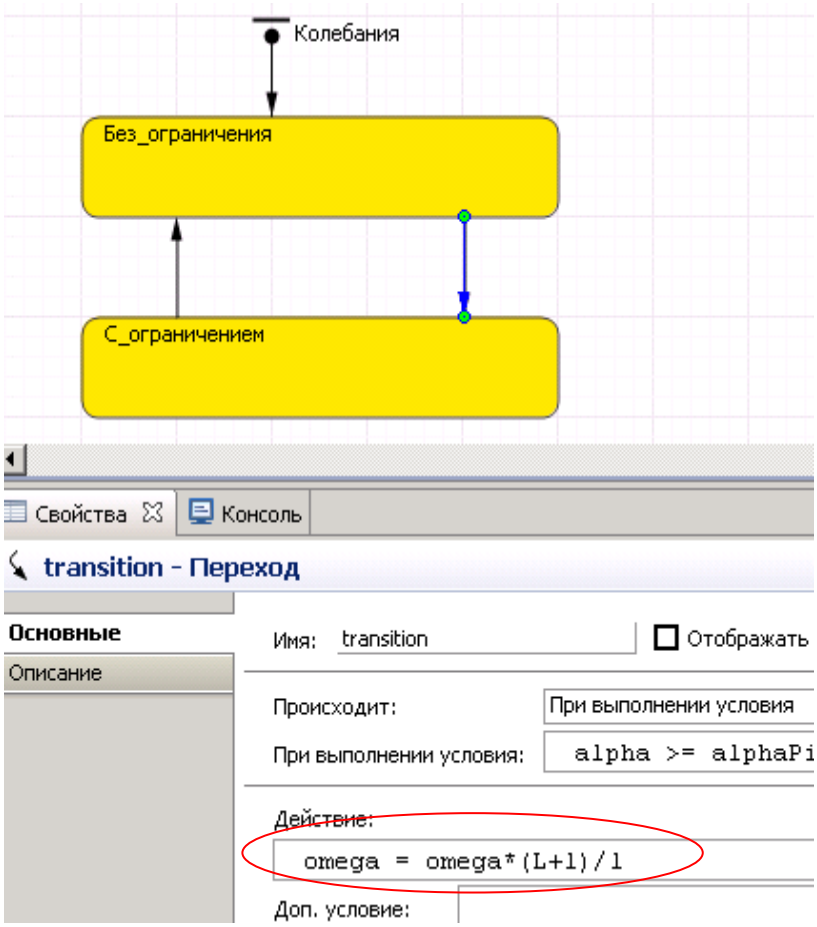
логическое ИЛИ

Условие перехода от движения с ограничением к движению без ограничения запишем аналогичным образом:

```
alpha > 0 && alpha <= alphaPin && omega < 0 ||

alpha < 0 && alpha >= -(2*PI-alphaPin) && omega > 0
```

При соприкосновении с ограничителем и изменении длины нити маятника должна сохраняться его линейная скорость, значит, справедливо соотношение:



$V_{огр} = V_{не\_огр}$ или

$omega_{огр}*l = omega_{не\_огр}*(L+l)$

или

$omega_{огр} = omega_{не\_огр}*(L+l)/l$

и

$omega_{не\_огр} = omega_{огр}*l/(L+l)$,

где: $V_{огр}$ и $V_{не\_огр}$ – линейная скорость маятника при движении с ограничителем и без ограничителя, соответственно;

$omega_{огр}$ и $omega_{не\_огр}$ –угловая скорость маятника при движении с ограничителем и без ограничителя, соответственно.

Эти соотношения следует записать в соответствующие поля **Действие** панели свойств переходов стейтчарта, рис. 8.3.

Рис. 8.3

Теперь настало время задать различные формулы для переменных `x`, `y` и `omega` в зависимости от состояния системы. Для этого можно использовать метод класса `isStateActive( )`. Если модель работает по схеме без ограничения, то вызов метода `Колебания.isStateActive(Без_ограничения)` вернет значение `true`. Если модель работает по ограниченной схеме, то вызов этого метода вернет значение `false`. Откройте свойства переменной `omega` и в поле **d(omega)/dt=** запишите следующее условное выражение:

```
Колебания.isStateActive(С_ограничением) ? -g*sin(alpha)/l -
mu*omega*abs(omega) : -g*sin(alpha)/(L+l) - mu*omega*abs(omega)
```

Если система находится в состоянии `С_ограничением`, то переменная `omega` будет

вычисляться по формуле $\frac{d\omega}{dt} = -\frac{g \cdot \sin(\alpha)}{l} - \mu \cdot \omega^2$; иначе по формуле $\frac{d\omega}{dt} = -\frac{g \cdot \sin(\alpha)}{L + l} - \mu \cdot \omega^2$.

По аналогичному принципу запишите уравнения для переменных `x` и `y`. Для переменной `alpha` – уравнение в обоих состояниях остается неизменным.

8.3. ПРЕЗЕНТАЦИЯ

Презентацию в этой модели следует построить так же, как и в предыдущей модели простого маятника. Координаты (`X, Y`) ограничителя, который задается овалом, определены так:

*X: L * sin(alphaPin) Y: L * cos(alphaPin)*

Для отрисовки нити следует использовать два отрезка. Один отрезок анимирует движение нити L, другой отрезок анимирует движение нити l.

Координаты начала нити L строятся точно так же как и в предыдущей лабораторной работе. А вот координаты конца нити L (или начала l) строятся иначе, заметьте, что в этих точках нити соединяются и их координаты будут одинаковы – это точка излома.

Координаты точки излома зависят от того по какой схеме (ограниченной или не ограниченной) работает модель в текущий момент. Предположим, что модель работает по схеме без ограничителя, т.е. нить не встречает препятствия. Тогда координату точки излома можно найти так:

$X : L \cdot \sin(\alpha)$, $Y : L \cdot \cos(\alpha)$.

Соответственно, в случае, когда модель работает по схеме с ограничителем, т.е. нить коснулась препятствия и остановилась, координата ее точки излома совпадет с координатой ограничителя:

$X : L \cdot \sin(\alpha_{пин})$, $Y : L \cdot \cos(\alpha_{пин})$

Остается только определить, по какой схеме модель работает в текущий момент. Для этого следует использовать, уже известный, метод класса `isStateActive( )`.

Для определения координаты X конца нити L запишите следующее выражение: `L*sin(Колебания.isStateActive(Без_ограничения)? alpha: alphaPin)` и аналогично для координаты Y.

Координаты конца нити *l* совпадают с координатами центра овала, т.е. они должны быть `x` и `y`, но в свойствах линии требуется указать смещение конца по осям X и Y относительно начала линии. Поэтому, в поле **dx:** динамических свойств линии L нужно указать не координату `x`, а разницу между `x` и координатой начала линии по оси X, т.е.:

```
x - L*sin(Колебания.isStateActive(Без_ограничения)? alpha: alphaPin)
```

Аналогично заполняется поле **dy:**

Достройте презентацию модели, как показано на рис. 8.4, и продемонстрируйте

построенную модель преподавателю.

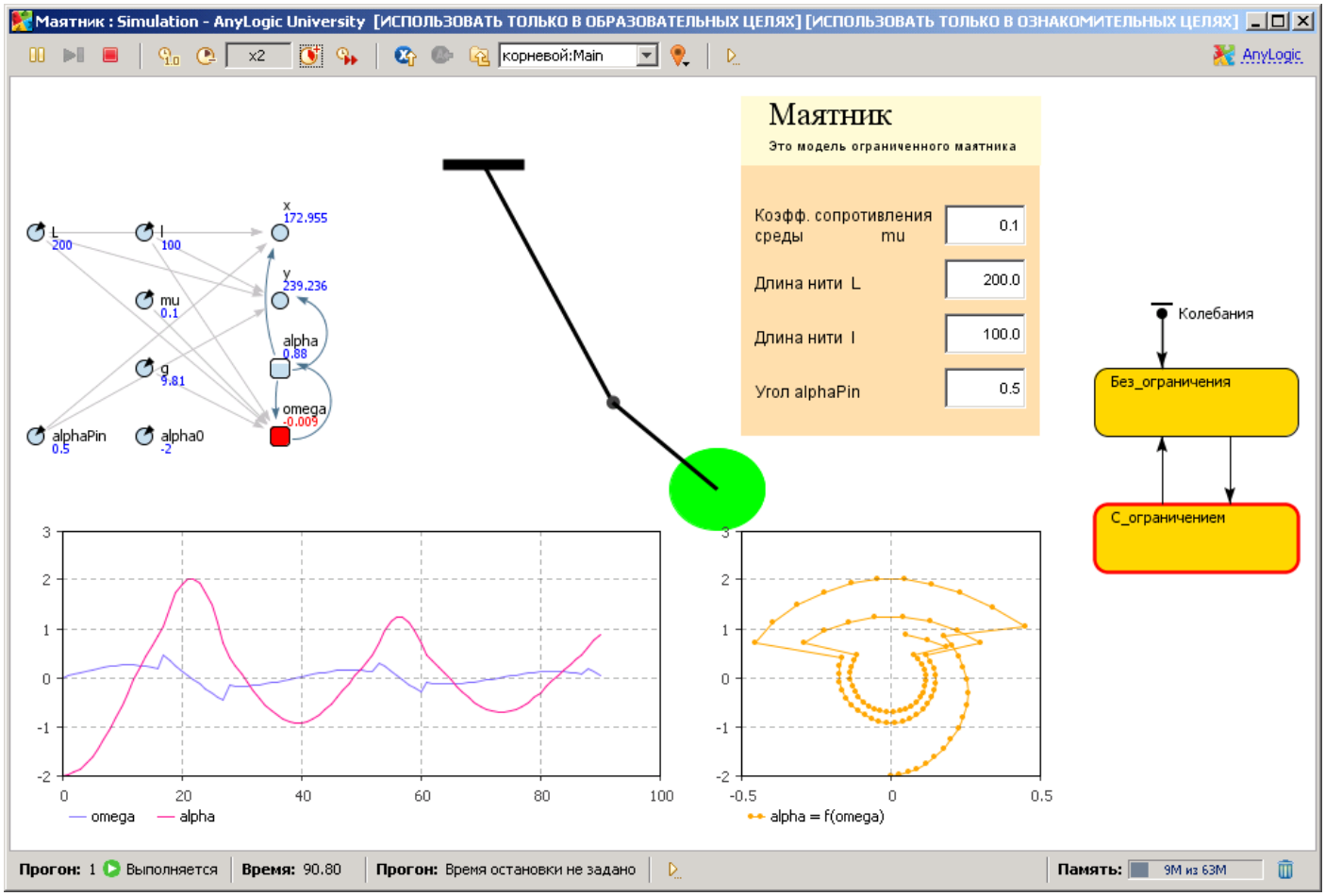
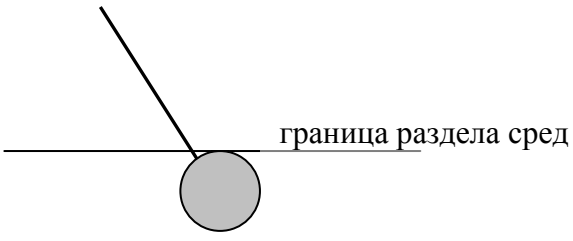


Рис. 8.4

8.4. КОНТРОЛЬНЫЕ ЗАДАНИЯ

1. Доработайте модель таким образом, чтобы при закручивании нити вокруг ограничителя учитывалось укорачивание нити на определенную длину за 1 оборот (наматывание). Учесть сохранение линейной скорости. (5+)
2. Доработайте модель таким образом, чтобы при изменении длины нити сохранялась линейная скорость маятника, а не угловая, как в базовой модели. (5+)
3. Добавьте в модель второй ограничитель, расположенный на противоположной стороне от первого. При изменении положения первого ограничителя, второй должен перемещаться зеркально. (5+)
4. Доработайте модель таким образом, чтобы в случае, когда растягивающее усилие, действующее на нить, становится отрицательным, цвет нити на презентации становится красным. (5)
5. Доработайте модель таким образом, чтобы она адекватно работала при alphaPin < 0. (5)
6. Доработайте презентацию модели таким образом, чтобы в момент касания ограничителя нитью маятник изменял свой цвет на случайный. (5)
7. Доработайте модель таким образом, чтобы маятник колебался в 2-х средах с различными коэффициентами



сопротивления, см. рис. справа. (5)

8. Доработайте модель таким образом, чтобы учитывалось изменение коэффициента сопротивления в зависимости от высоты, например, по линейному закону. (4)
9. Доработайте модель таким образом, чтобы показывалось время, прошедшее с момента касания нити ограничителя. При следующем касании показания должны сбрасываться и считаться заново. (4)
10. Доработайте модель таким образом, чтобы на презентации показывалась линейная скорость маятника только при касании нити ограничителя. (4)
11. Доработайте модель таким образом, чтобы на презентации показывался график линейного тангенциального ускорения маятника. (4)
12. Доработайте модель таким образом, чтобы показывалось время, прошедшее с момента запуска модели до касания нити ограничителя. (4)
13. Доработайте модель таким образом, чтобы показывалась длительность касания нити ограничителя. (4)
14. Доработайте презентацию модели таким образом, чтобы при движении без ограничителя высвечивалась надпись «Длинная нить», а при движении с ограничителем – «Короткая нить». (3)
15. Добавьте в модель кнопку, при нажатии которой маятник увеличивает скорость на 20%. (3)
16. Измените положение ограничителя на противоположное относительно вертикальной оси. (3)
17. Как в модели реализовано изменение структуры системы? (3)
18. С помощью какого метода можно узнать, в каком состоянии находится стейтчарт? (3)

ЗАДАНИЕ 9

ОПТИМИЗАЦИОННАЯ МОДЕЛЬ ПОЛЕТА ЯДРА

ЦЕЛИ ЗАНЯТИЯ

- освоение методов решения оптимизационных задач;
- самостоятельное построение оптимизационных моделей физических процессов;
- построение оптимизационной модели полета пушечного ядра.

ФОРМА ОРГАНИЗАЦИИ ЗАНЯТИЯ

Фронтальная.

СТУДЕНТ ДОЛЖЕН ЗНАТЬ

- технологию построения оптимизационных моделей,
- понятия: целевая функция, параметр оптимизации, оптимизационный эксперимент,
- интерфейс программы AnyLogic.

СТУДЕНТ ДОЛЖЕН УМЕТЬ

- создавать модели в программе AnyLogic.

ОБЕСПЕЧЕННОСТЬ

- компьютер с установленной программой AnyLogic версии 6,
- настоящий курс лабораторно-практических работ.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

В этой работе требуется построить оптимизационную модель полета снаряда.

9.1. СОДЕРЖАТЕЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ

Тело (ядро/снаряд) массой 1 кг выстреливается из пушки под углом α к горизонту со скоростью 100 м/с. Требуется найти оптимальный угол α, при котором достигается максимальная дальность стрельбы.

9.2. КОНЦЕПТУАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ

Примем следующие предположения:

- объектом исследования является тело массой 1 кг, принимаемое за материальную точку;
- движение тела подчиняется второму закону Ньютона;
- тело находится под действием двух сил: силы тяжести `mg` и силы сопротивления воздуха, пропорциональной квадрату скорости движения.

9.3. МАТЕМАТИЧЕСКАЯ ПОСТАНОВКА ЗАДАЧИ

С математической точки зрения имеем задачу Коши.

Модель ядра имеет четыре вещественных переменных: положение его центра `x` и `y` в координатах `X` и `Y`, и составляющие скорости `Vx` и `Vy` по этим координатам (см. рис. 9.1). Координаты являются интегралами от соответствующих скоростей:

$$\frac{dx}{dt} = Vx; \qquad \frac{dy}{dt} = Vy$$

Производная от проекций скоростей определена как:

$\frac{dVx}{dt} = -K \cdot Vx^2$; $\frac{dVy}{dt} = -g - K \cdot Vy^2$, где `K` – коэффициент сопротивления воздуха,

`g` = 9,81 – ускорение свободного падения.

Начальные условия:

`x = 0,` `y = 0;`

`Vx = V0 * cos(`α`),` `Vy = V0 * sin(`α`).`

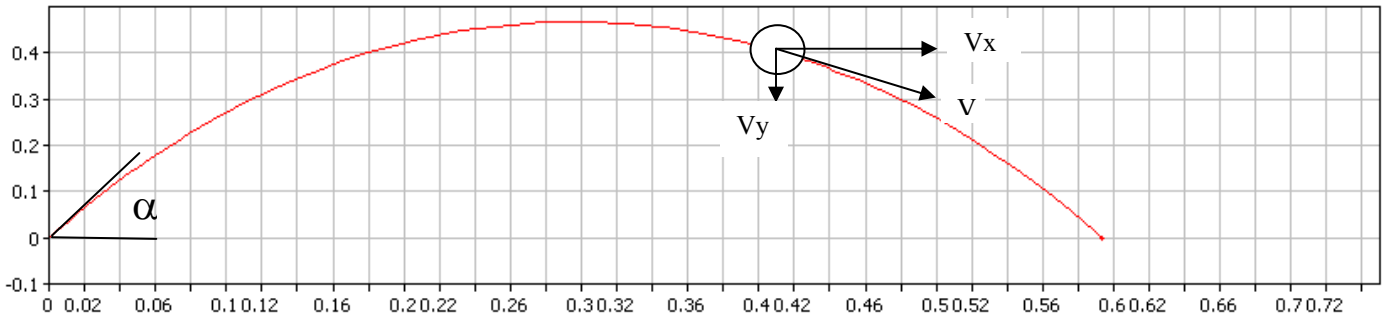


Рис. 9.1

9.4. ОПИСАНИЕ МОДЕЛИ

Представим, что наш снаряд вылетает из пушки под очень маленьким углом – почти горизонтально, тогда дальность полета снаряда получится небольшой. Напротив, если выстрелить снарядом под очень большим углом – почти вертикально, то дальность тоже получится небольшой. Очевидно, что существует оптимальный угол, при котором дальность полета будет наибольшей. Необходимо найти этот угол. Для решения задачи построим оптимизационную модель.

Оптимизация состоит из нескольких последовательных прогонов модели с различными значениями параметров, что позволяет находить значения параметров модели, соответствующие максимуму или минимуму целевой функции. В нашем случае целевая функция – дальность полета ядра. Параметр оптимизации – угол α. Для решения задачи нахождения наивыгоднейшего угла бросания следует выполнить ряд этапов:

1. Создать имитационную модель.
2. Создать оптимизационный эксперимент.
3. Задать целевой функционал (функцию, которую нужно минимизировать или максимизировать).
4. Задать оптимизационные параметры (параметры, значения которых будут меняться).
5. Задать условия остановки прогона.
6. Задать условия остановки оптимизации.
7. Запустить оптимизационный эксперимент.

9.4.1. СОЗДАНИЕ ИМИТАЦИОННОЙ МОДЕЛИ

Создайте новую модель `Polet` с чистого листа (не используя шаблоны).

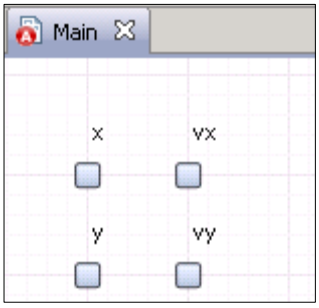


Рис. 9.2

Откройте палитру **Системная динамика**. Перенесите 4 элемента типа **Накопитель** на диаграмму класса активного объекта и переименуйте их соответственно в `X, Y, Vx` и `Vy`. Теперь в нашей модели есть 4 динамических переменных, описывающих координату и скорость перемещения снаряда, как показано на рис. 9.2.

Свяжем объявленные переменные вышеприведенными зависимостями, для этого в панели **Свойства** введите соответствующие формулы и начальные значения, как это показано на рис.9.3.

Обратите внимание на уравнение: `d(vy)/dt = -g -K*Vy`^{2}, при возведении `Vy` в квадрат член уравнения `-K*Vy`^{2} всегда будет отрицательным, независимо от направления движения снаряда, т.е. будет потеряно направление скорости движения, что приведет к ошибке. Необходимо видоизменить это уравнение так, чтобы знак скорости не потерялся:

```
d(vy)/dt = -g -K*Vy*|Vy|.
```

Аналогично нужно изменить и уравнение для `Vx`.

Теперь, когда мы описали движение снаряда, нужно объявить и присвоить значения параметрам `g, K, V`$_0$ и α. Для этого перенесите из палитры **Системная динамика** 4 элемента типа **Параметр** на диаграмму класса активного объекта, переименуйте их в `g, k, v0, a` и задайте им значения: `9,81; 0,01; 100; 1` соответственно.

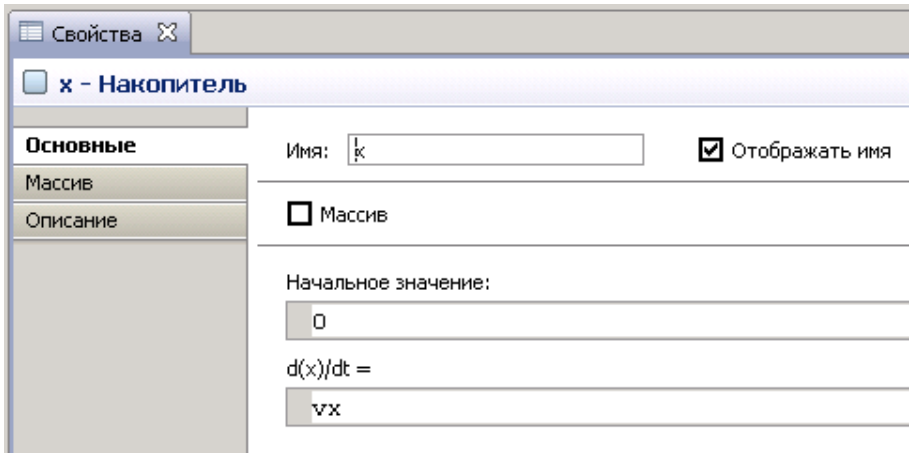
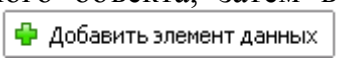


Рис. 9.3

Теперь нарисуем траекторию движения снаряда. Для этого перенесите из палитры **Статистика** элемент типа **График** на диаграмму класса активного объекта, затем в свойствах графика нажмите на кнопку: **Добавить элемент данных**. 

В поле **Значение по оси X** введите `x`, а в поле **Значение по оси Y** – `y` соответственно, как показано на рис. 9.4.

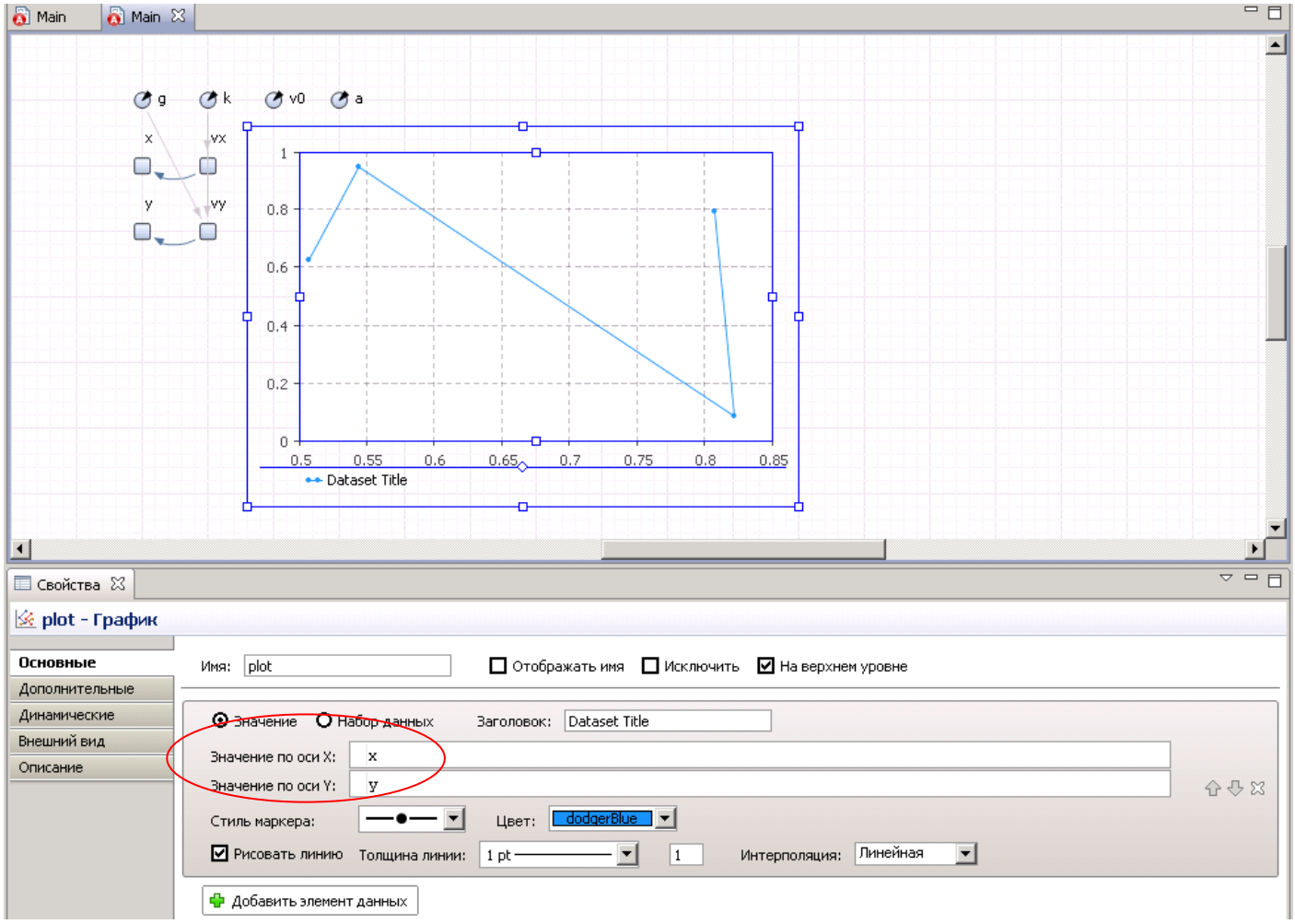
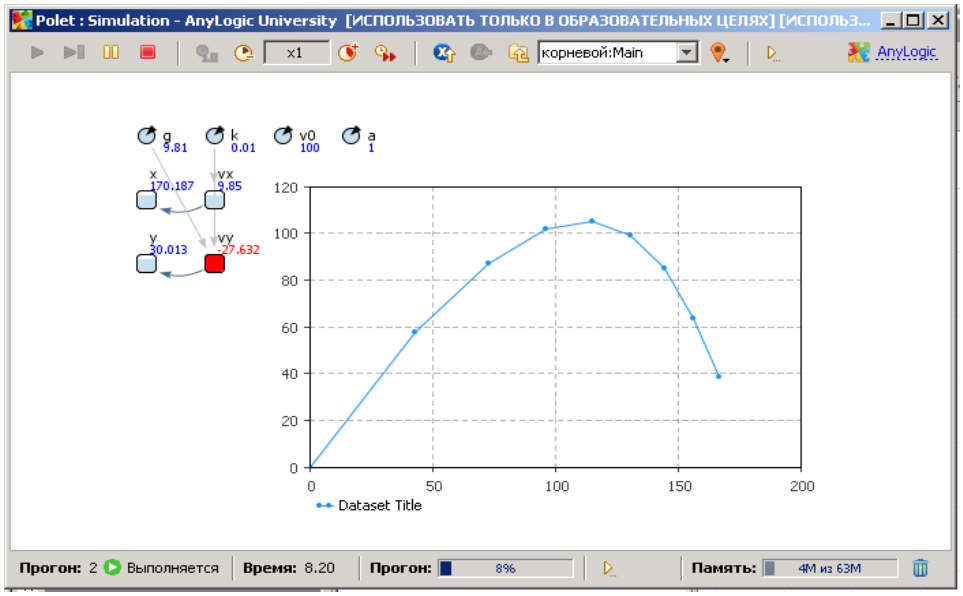


Рис. 9.4

Запустив модель на выполнение, мы увидим траекторию движения снаряда, рис. 9.5.



Визуализировав траекторию движения снаряда, мы можем оценить адекватность модели в первом приближении, а для более точной проверки нужно выполнить тестовые примеры. Но сейчас нас интересует оптимальный угол выстрела, а для этого нужно определить дальность полета.

Рис. 9.5

Введем простую переменную `D` из палитры **Основная**, в которую будем записывать дальность полета снаряда, а именно: значение переменной `x` в момент падения снаряда на землю. Отследить момент падения можно с помощью стейтчарта, который мы сейчас и построим.

С помощью палитры **Диаграмма состояний** постройте стейтчарт, как показано на рис. 13.6.

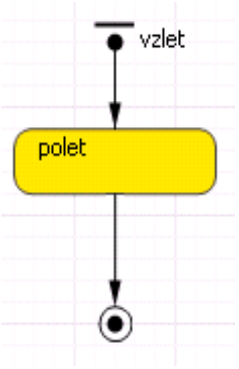


Рис. 9.6

Переход в конечное состояние должен происходить при выполнении условия `y < 0`, т.е. когда снаряд коснется земли. При этом должно выполниться действие, при котором переменная `D` получит значение дальности полета, а именно `D = x`. Сохраните проект и запустите модель для проверки ее работоспособности. Обратите внимание, что при достижении уровня земли, точнее нулевой высоты полета, наш снаряд продолжает движение. Чтобы этого не происходило нужно остановить прогон модели. Для этого в конечном состоянии в поле **Действие** впишите следующую функцию `finishSimulation()`, модель будет останавливаться.

Мы создали имитационную модель и задали условие остановки прогона, теперь приступим к созданию оптимизационного эксперимента.

9.4.2. СОЗДАНИЕ ОПТИМИЗАЦИОННОГО ЭКСПЕРИМЕНТА

Чтобы создать оптимизационный эксперимент в панели **Проекты**, щелкните правой кнопкой мыши по элементу модели и выберите **Создать|Эксперимент** из контекстного меню. Появится диалоговое окно **Новый эксперимент**. Выберите **Оптимизация** из списка **Тип эксперимента**, затем введите имя эксперимента в поле **Имя**, например `Optimum`. Остальные параметры оставьте без изменений.

Щелкните мышью по кнопке **Готово**. Мы создали оптимизационный эксперимент, теперь нужно ввести целевой функционал. Так как нам нужно максимизировать дальность полета, то в качестве целевой функции следует выбрать переменную `D`. Корневой активный объект эксперимента доступен здесь как `root`, поэтому запишем `root.D`.

Задайте условие остановки оптимизации – количество итераций `= 500`, это максимальное число итераций, доступное в учебной версии. Если выбрать опцию **Автоматическая остановка**, то оптимизация будет продолжаться до тех пор, пока значение целевой функции не перестанет существенно улучшаться (эта функция не доступна в учебной версии программы).

Затем следует задать оптимизационные параметры, т.е. параметры, значения которых будут меняться. Мы изменяем только один параметр: угол выстрела α, причем этот угол может принимать любое значение от 0 до 180^0, поэтому в таблице **Параметры** напротив параметра `a` щелкните мышью в ячейке **Тип** и выберите тип параметра `непрерывный` и установите минимальное значение – `0`, максимальное – `3.1415`, начальное значение – `1`, рис. 9.7. Нажмите кнопку **Создать интерфейс**, при этом будет создан интерфейс эксперимента – на презентацию будут добавлены элементы управления, отображающие текущий прогресс оптимизации и облегчающие управление процессом. Затем сохраните проект и запустите оптимизационный эксперимент.

9.5. ЗАПУСК МОДЕЛИ

Кнопка **Запустить оптимизацию** на презентации оптимизационного эксперимента выполняет запуск оптимизационного процесса, рис. 9.8. В правой части расположен график, визуально отображающий ход оптимизации. По оси X откладываются номера итераций, а по оси Y - **Текущее, Лучшее допустимое** и **Лучшее недопустимое** значения, найденные для каждой итерации.

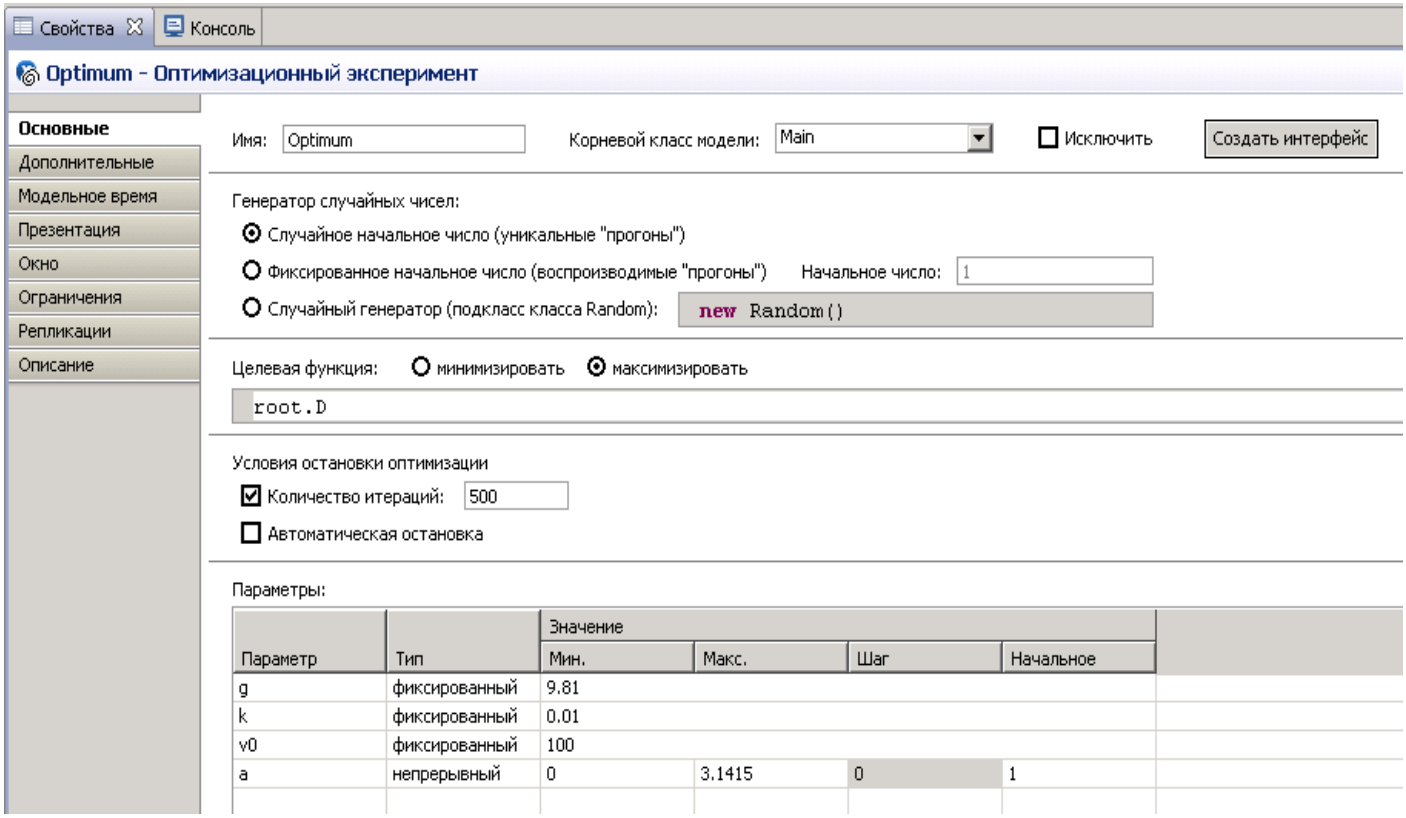


Рис. 9.7

Таблица, расположенная в левой части окна, отображает всю необходимую информацию о ходе оптимизационного процесса. В столбце **Текущее** отображаются: номер последней завершенной итерации, значение целевой функции и значения параметров, при которых оно было получено на момент окончания этой итерации.

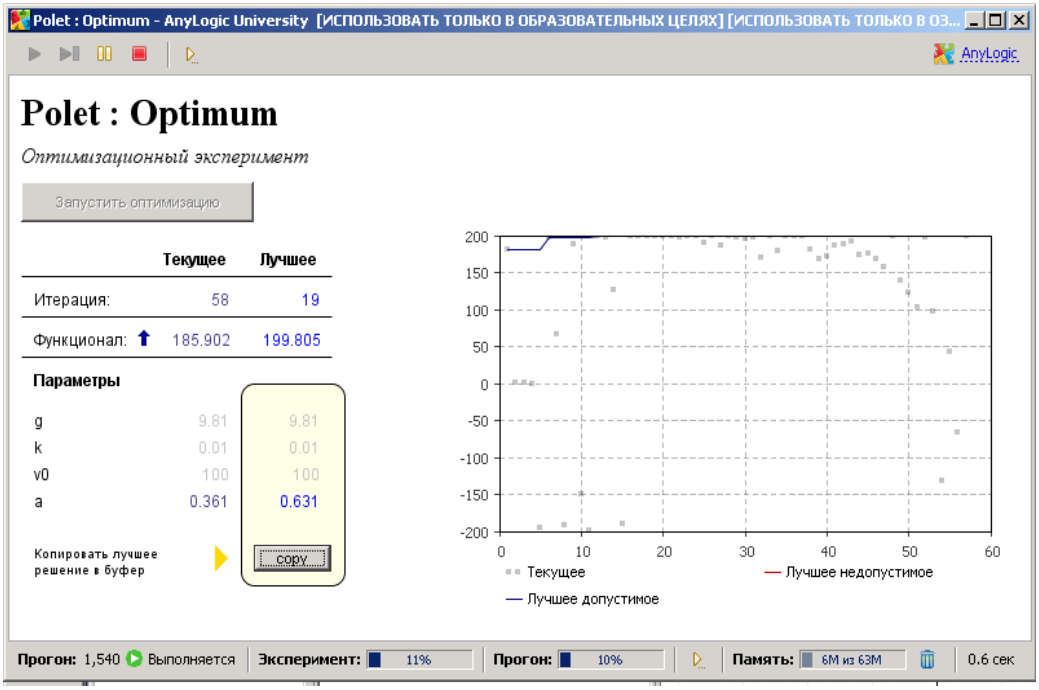


Рис. 9.8

В столбце **Лучшее** отображается та же информация для найденного решения, которое является оптимальным на данный момент.

По завершении оптимизации это решение будет считаться оптимальным найденным решением. Вы можете экспортировать полученное решение в другие эксперименты модели, щелкнув по кнопке **сору** и вставив его в дальнейшем в нужный эксперимент с помощью кнопки **Вставить из буфера**, расположенной в нижней части страницы **Основные** панели **Свойства** этого эксперимента.

Постройте модель и продемонстрируйте полученный результат преподавателю.

9.6. КОНТРОЛЬНЫЕ ЗАДАНИЯ

1. Доработайте оптимизационную модель таким образом, чтобы определить угол бросания, при котором снаряд упадет в ту же точку, при условии, что на снаряд действует встречный ветер, со скоростью 15 м/с. (5+)
2. Доработайте оптимизационную модель таким образом, чтобы учитывался встречный ветер со скоростью 10 м/с. (5)
3. Доработайте оптимизационную модель таким образом, чтобы учитывался попутный ветер со скоростью 10 м/с. (5)
4. Доработайте оптимизационную модель таким образом, чтобы учитывалась следующая зависимость коэффициента сопротивления от высоты: при $h < h_{max}/10$, Ксопр = 0,02; при $h >= h_{max}/10$, Ксопр = 0,01. (h_{max} – максимальная высота, достигаемая снарядом при отсутствии сопротивления воздуха). (5)
5. Доработайте оптимизационную модель таким образом, чтобы учитывалась подъемная сила, зависящая от квадрата скорости полета, Кпод = 0,01. (5)
6. Доработайте оптимизационную модель таким образом, чтобы в качестве целевой функции использовалась дальность броска с несколькими отскоками, сопровождающимися потерей энергии. (5)
7. Доработайте оптимизационную модель таким образом, чтобы учитывалась следующая зависимость коэффициента подъемной силы от вертикальной скорости: при наборе высоты Кпод = 0, при спуске Кпод = 0,5. (5)
8. Доработайте оптимизационную модель таким образом, чтобы учитывалась сила реактивной тяги двигателя снаряда. (5)
9. Доработайте оптимизационную модель таким образом, чтобы учитывалась сила реактивной тяги двигателя снаряда, прямо пропорциональная высоте полета. (5)
10. Доработайте оптимизационную модель таким образом, чтобы учитывалась сила реактивной тяги двигателя снаряда, действующая только в начале полета и отключаемая через 10 секунд. (5)
11. Доработайте оптимизационную модель таким образом, чтобы учитывалась сила реактивной тяги двигателя снаряда, обратно пропорциональная высоте полета. (5)
12. Доработайте оптимизационную модель таким образом, чтобы учитывалась сила реактивной тяги двигателя снаряда, обратно пропорциональная высоте полета. Целевая функция – продолжительность полета. (5)
13. Доработайте оптимизационную модель таким образом, чтобы учитывалась сила реактивной тяги двигателя снаряда, обратно пропорциональная высоте полета и всегда направленная в верхний сектор, т.е. вертикальная составляющая тяги всегда положительна. (5)
14. Доработайте оптимизационную модель таким образом, чтобы учитывалась сила реактивной тяги двигателя снаряда, обратно пропорциональная высоте полета и всегда направленная в верхний сектор, т.е. вертикальная составляющая тяги всегда

положительна. Целевая функция – продолжительность полета. (5)

15. Доработайте оптимизационную модель таким образом, чтобы учитывалась сила реактивной тяги двигателя снаряда, направленная всегда вертикально вверх и включаемая циклически, например 1 сек. – работает, 15 сек. – не работает. (5)
16. Доработайте оптимизационную модель таким образом, чтобы учитывалась следующая зависимость коэффициента сопротивления от вертикальной скорости: при наборе высоты Ксопр = 0, при спуске Ксопр = 0,1. (4)
17. Доработайте оптимизационную модель таким образом, чтобы учитывалась следующая зависимость коэффициента сопротивления от вертикальной скорости: при наборе высоты Ксопр = 0,1 при спуске Ксопр = 0. (4)

ЗАДАНИЯ 10 – 11

АГЕНТНОЕ МОДЕЛИРОВАНИЕ: БРОУНОВСКОЕ ДВИЖЕНИЕ

ЦЕЛИ ЗАНЯТИЯ

- освоение методов агентного моделирования;
- модификация агентной модели плоского соударения шаров;

ФОРМА ОРГАНИЗАЦИИ ЗАНЯТИЯ

Фронтальная.

СТУДЕНТ ДОЛЖЕН ЗНАТЬ

- технологию построения моделей,
- понятия: агент, объектно-ориентированное программирование,
- основы языка Java,
- интерфейс программы AnyLogic.

СТУДЕНТ ДОЛЖЕН УМЕТЬ

- создавать модели в программе AnyLogic;
- пользоваться справкой AnyLogic.

ОБЕСПЕЧЕННОСТЬ

- компьютер с установленной программой AnyLogic версии 6,
- модель `Шары`,
- настоящий курс лабораторно-практических работ.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

Рассмотрим динамическую систему, имитирующую столкновения: биллиардных шаров, падающих и сталкивающихся мячей или частиц при хаотичном броуновском движении. Эта модель позволяет изучить агентный подход к решению задач, связанных со взаимодействием отдельных объектов. Первым шагом при создании агентной модели является создание агентов. Агент является основным строительным блоком агентной модели. Агентная модель состоит из множества агентов и их окружения. Для каждого агента задается набор правил, согласно которым он взаимодействует с другими агентами; это взаимодействие и определяет общее поведение системы. В нашей модели агентами будут сталкивающиеся шары. Откройте модель с именем `Шары` и тщательно изучите. В следующей части задания Вам предстоит модернизировать эту модель.

10.1. ОПИСАНИЕ ПРОБЛЕМЫ

Шары одинаковой массы двигаются в ограниченном двумерном пространстве с упругим соударением без потери энергии. Столкновение шаров со стенками мы рассматривали в самой первой работе, вспомним, что новое значение скорости шара получается путем изменения знака соответствующей составляющей скорости. Рисунок 10.1 демонстрирует

столкновение двух шаров друг с другом. При центральном упругом столкновении шаров они будут обмениваться своими скоростями. В нашей модели силы трения очень малы по сравнению с упругими силами. Поэтому, при нецентральном столкновении действием сил трения можно пренебречь. Проведем через центры соприкоснувшихся шаров прямую и разложим скорость каждого шара на нормальную составляющую, направленную вдоль этой прямой, и касательную составляющую, перпендикулярную к ней. Так как, силы трения отсутствуют, изменяться будут только нормальные составляющие скорости шаров `vr` так же, как при центральном столкновении, то есть произойдет обмен нормальными составляющими скоростей, рис. 10.1.

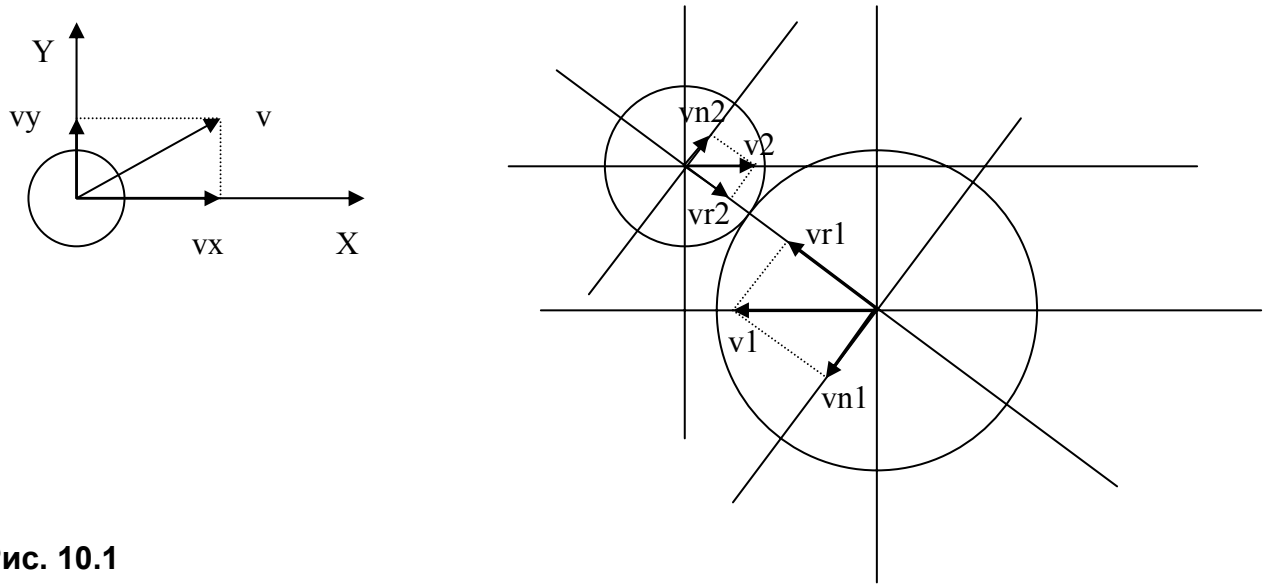


Рис. 10.1

10.2. ОПИСАНИЕ МОДЕЛИ

Модель состоит из двух классов активных объектов: сталкивающихся шаров и плоского пространства, содержащего эти шары.

10.2.1. МОДЕЛЬ ШАРА

Как и в первой работе, шары моделируются классом `Ball`, который содержит параметры, переменные и события.

`Параметры:` `r` – радиус шара, `Xmax` и `Ymax` – размеры прямоугольной области, в которой двигаются шары, `V` – начальная скорость шара, `g` – ускорение свободного падения.

Пременные: положение его центра `x` и `y` в координатах X и Y, составляющие скорости `vx` и `vy` по этим координатам (см. рис. 10.1). Три переменных описаны интегралами:

$$\frac{dx}{dt} = Vx; \qquad \frac{dy}{dt} = Vy; \qquad \frac{dVy}{dt} = g.$$

Переменная `vx` задана как простая переменная, т.к. по оси X на шары не действуют внешние силы.

Кроме того, в классе активного объекта `Ball` задана переменная `color` типа `Color`, с помощью которой можно явно задать значения цветовых компонент цвета (**Красный**, **Зеленый** и **Синий**), (значения задаются в диапазоне от 0 до 255).

Чтобы программно задать произвольный цвет для любого графического объекта, следует вызвать конструктор, например, для создания розовато-лилового цвета: `new Color(255, 127, 255)`.

Для окрашивания шаров в различные случайные цвета, будем использовать функцию `uniform_discr(255)`, принимающую случайное целое значение, распределенное равномерно между 0 и 255:

```
new Color(uniform_discr(255), uniform_discr(255), uniform_discr(255)).
```

Каждый раз при обращении к переменной `color` конструктор `new Color` будет порождать значение случайного цвета.

Поле **Действие при запуске** панели **Свойства** класса активного объекта `Ball` содержит операции, которые необходимо выполнить при порождении каждого экземпляра шара. Это – задание случайного направления движения любого порождаемого шара, а также случайные значения координат его центра, рис. 10.2.

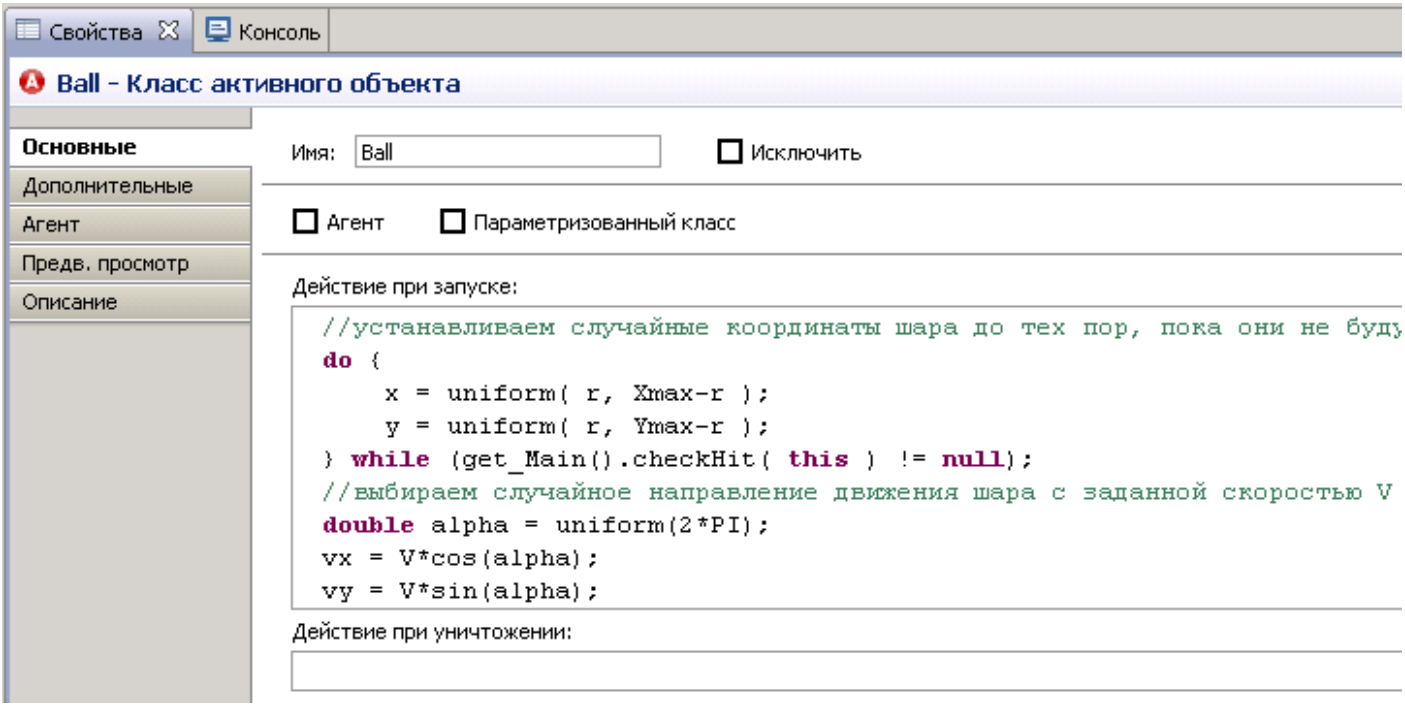


Рис. 10.2

Каждый вновь создаваемый шар будет двигаться со скоростью V в случайно выбранном направлении и все шары будут распределены равномерно в заданном пространстве, так, чтобы не попасть на уже имеющиеся шары.

10.2.2. ПОВЕДЕНИЕ ШАРА

Шары движутся в поле силы тяжести или в невесомости (если `g=0`) со скоростями `vx` и `vy`, до тех пор, пока не столкнутся с каким либо препятствием:

- вертикальной границей пространства, в котором движутся шары,
- горизонтальной границей и
- другим шаром.

Столкновение с каждым видом препятствия отслеживается специальным объектом – событием, которое выполняет определенные действия. В нашей модели определено три события, по одному на каждый тип столкновений.

Столкновение с границей происходит, когда центр шара приблизится к ней на расстояние радиуса `r` и шар движется в направлении этой границы. При наступлении события столкновения с границей, изменяется знак соответствующей составляющей скорости шара, и шар отскакивает от нее.

Событие столкновения одного шара с другим проверяется функцией `checkHit()`, которая возвращает указатель на тот шар, с которым столкнулся другой шар, или пустой указатель `null`, если столкновения не было. При столкновении шаров, их скорости `vx` и `vy` пересчитываются в соответствии с рис. 10.1., а Ява код задан в поле **Действие** окна свойств события `bounceBall`.

10.2.3. МОДЕЛЬ ПРОСТРАНСТВА

Пространство, в котором движутся шары, включает вложенный реплицированный объект типа `Ball`, с числом элементов 10. Часть параметров (`Xmax, Ymax, g, r`) класса `Ball` переопределена на странице **Параметры** панели **Свойства**, что позволяет управлять ими из корневого объекта.

Кроме того, в корневом классе `Main` определена функция `checkHit()` типа `Ball`, в которую передается указатель на проверяемый шар, а возвращается указатель на шар, с которым он столкнулся: `return b1.`

В случае если столкновения нет, функция возвращает пустой указатель: `return null;`

Функция `checkHit()` вызывается из события `bounceBall`, которое в качестве параметра передает указатель `this` на свой шар. Переменная `this` есть в каждом шаре и содержит ссылку на свой шар. В классе `Ball` мы не можем задать функцию `checkHit()`, потому что она должна иметь информацию о положении и скоростях всего множества шаров, а это множество определено в активном объекте `Main`.

10.3. ПРЕЗЕНТАЦИЯ

Шар представляется овалом с динамическими значениями: `r` радиуса, его координат `x` и `y` и цветом заливки color.

Так как в презентации класса `Ball` указано, что овал, представляющий шар, виден на верхнем уровне, то в презентации корневого объекта `Main` автоматически будут включены 10 экземпляров овалов радиуса `r` и случайно выбранным цветом. Эти овалы будут распределены равномерно в прямоугольном пространстве, и будут двигаться в различных (случайных) направлениях, имитируя движения шаров в модели.

10.3.1. ИЗМЕНЕНИЕ РАДИУСА ШАРОВ И СИЛЫ ПРИТЯЖЕНИЯ

Рассмотрим, как изменить радиусы шаров `r` во время работы модели.

У каждого шара есть свой параметр `r`, который может принимать произвольные и различные значения, т.е. радиус каждого шара может быть уникальным, отличным от радиусов других шаров. Для того чтобы сделать возможным изменение радиуса сразу всех шаров из включающего их объекта `Main`, можно использовать параметр `r`, определенный в классе `Main`. В свойствах вложенного реплицированного объекта `balls` мы переопределили параметр r, как показано на рис. 10.3. Эта связь параметра `r`, заданного в классе `Main` с параметром `r`, заданном в классе `Ball` позволяет управлять радиусами всех шаров из корневого объекта. Эта связь однонаправленная, мы можем из корневого объекта изменить все параметры `r` реплицированных объектов, но при изменении `r` в каком-нибудь реплицированном объекте, параметр `r` в корневом объекте останется неизменным.

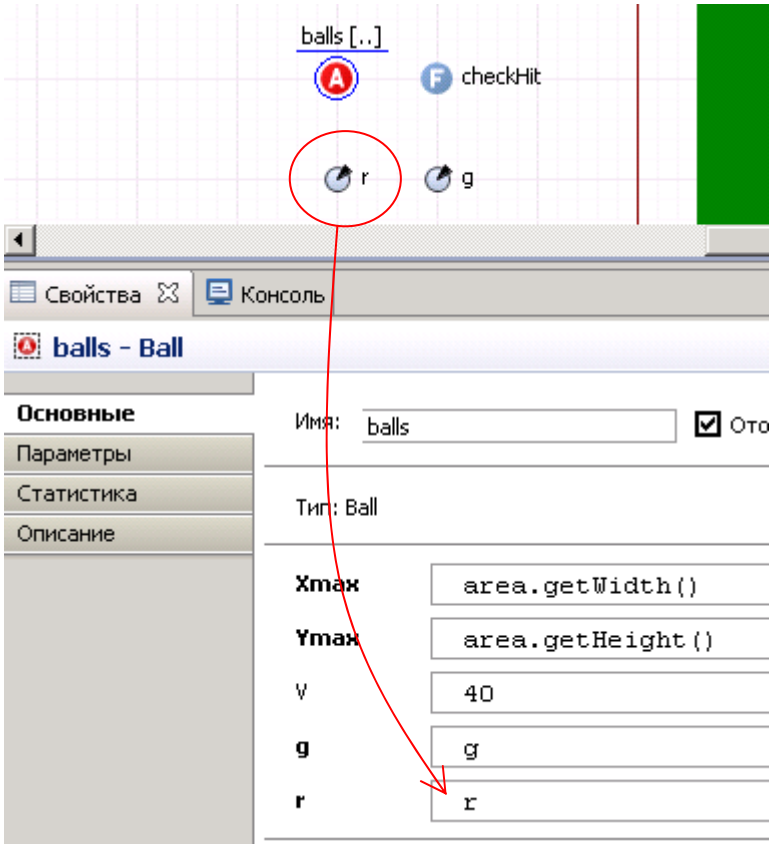


Рис. 10.3

Собственно изменение радиуса шаров реализовано с помощью слайдера, связанного с параметром `r` корневого объекта (класса `Main`). Аналогично регулируется параметр `g` – ускорение свободного падения.

10.4. ИЗМЕНЕНИЕ ЧИСЛА ШАРОВ

Для удаления случайного шара (элемента коллекции `balls`) из модели, во время ее работы, следует использовать оператор:

```
if( balls.size() > 0 ) remove_balls( balls.random() )
```

который сначала проверяет, есть ли в модели хотя бы один шар, с помощью метода `size()`, а затем, для удаления случайного шара используется метод `remove_balls()`, которому в качестве параметра передается ссылка на один из реплицированных объектов, выбранных случайным образом `balls.random()`.

Для обращения к конкретному объекту, например №3 следует использовать метод `get()`, в котором указать номер объекта (номера объектов в коллекции начинаются с 0). То есть, для того, чтобы удалить шар под №3 нужно выполнить оператор:

```
remove_balls( balls.get(3)).
```

Для создания в модели нового шара (добавления в коллекцию реплицированных объектов `balls` дополнительного элемента класса `Ball`), во время ее работы, используется метод:

`add_balls()`, без параметров.

Можно создавать экземпляры объектов и сразу же задавать значения параметров этих объектов путем вызова следующего метода:

`add_activeObjectName(parameter1, parameter2, ...)` , где `activeObjectName` – имя реплицированного объекта (в нашем случае `balls`), а параметры перечислены в том порядке, в котором они показаны на рис. 10.3.

Эти операторы записаны в поле **Действие** панелей свойств кнопок **Добавить шар** и **Удалить шар**. При каждом нажатии на соответствующую кнопку во время работы модели будет создан или удален шар.

Обратите внимание, что эти методы создаются в классе `Main`, так что они могут быть вызваны напрямую из любого места класса `Main` (например, из его Действия при запуске или из Действия события и т.д.).

При необходимости создания или удаления объекта из другого объекта, нужно вначале получить ссылку на объект `Main` с помощью метода `get_Main()`. Например, если один шар порождает другой, то следует написать такой код в объекте "родителя":

```
get_Main().add_balls()
```

Другой часто используемый случай: элемент реплицированного объекта (шар) должен уничтожить сам себя:

`get_Main().remove_balls( this )` , здесь параметр `this` – это ссылка на самого себя.

-

10.5. ИЗМЕНЕНИЕ МОДЕЛИ ШАРЫ

Выполните задание своего варианта и продемонстрируйте преподавателю.

10.5.1. НЕКОТОРЫЕ ПОЯСНЕНИЯ К ЗАДАНИЯМ

1. Обратите внимание на то, что в исходной модели, в функции `checkHit()`, определяющей столкнутся шары или нет, предполагается, что все шары имеют одинаковый радиус `r`. Следует видоизменить эту функцию таким образом, чтобы при вычислении возможности столкновения шаров учитывалось, что они могут иметь разный размер.

2. В вариантах, в которых требуется моделировать поведение шаров двух типов, например синих и красных, не следует создавать 2 класса активных объектов (для каждого типа шаров). Вместо этого целесообразно объявить параметр, определяющий тип шара и в зависимости от этого параметра задавать внешний вид и поведение шаров.

3. Узнать номер реплицированного элемента из коллекции (номер шара, под которым он стоит в коллекции) можно с помощью метода `getIndex()`.

Вариант 1. Требуется модернизировать модель следующим образом:

1. Рабочая зона должна быть поделена на 2 части, соответственно верхнюю и нижнюю.
2. В верхней зоне может одновременно находиться не более N шаров. Значение N должно регулироваться слайдером в диапазоне от 0 до 10.
3. Шары могут свободно покидать верхнюю зону, но войти в нее могут, только если в этой зоне число шаров меньше N.
4. В презентации модели отразить количество шаров в верхней зоне. (5)

Вариант 2. Требуется модернизировать модель следующим образом:

1. При старте модели создаются шары только 3-х основных цветов: красный, зеленый и синий.
2. При соударениях шаров друг с другом должен изменяться их цвет на усредненный по следующему правилу: интенсивность каждой компоненты цвета первого шара уменьшается на 10%, и к ним добавляются по 10% от компонент второго шара.
    a. Например, сталкиваются 2 шара. 1-й шар окрашен в светло желтый цвет (компоненты RGB имеют следующие значения: 150, 150, 0). 2-й шар окрашен в темно серый цвет (компоненты RGB имеют следующие значения: 10, 10, 10).
    b. После столкновения 1-й шар окрасится в цвет, определяемый так: красная компонента = 150 – 15 + 1 = 136. Зеленая компонента = 150 – 15 + 1 = 136. Синяя компонента = 0 – 0 + 1 = 1.
    c. 2-й шар окрасится в цвет, определяемый так: красная компонента = 10 – 1 + 15 = 24. Зеленая компонента = 10 – 1 + 15 = 24. Синяя компонента = 10 – 1 + 0 = 9.
3. Рядом с каждым шаром должна быть надпись, указывающая на значения компонент R, G, и B. (5)

Вариант 3. Требуется доработать модель следующим образом:

1. В модели 2 вида шаров: большие (`r` = 15) и маленькие (`r` = 5). При соударении *разных* шаров должен появиться еще один шар, случайного размера (15 или 5 единиц), с вероятностью обратно пропорциональной общему количеству шаров. Зависимость вероятности успешного появления нового шара задана таблицей.

Кол-во шаров	Вероятность появления
<=10	1
12	0.9
14	0.8
16	0.5
18	0.2
>=20	0

2. При соударении *одинаковых* шаров должен исчезать шар с меньшей скоростью с вероятностью обратно пропорциональной количеству шаров данного вида. Зависимость вероятности успешного исчезновения шаров задана таблицей.

Кол-во шаров	Вероятность исчезновения
<=2	0
4	0.1
6	0.2
8	0.4
10	0.8
>=16	1

3. Презентация модели должна показывать текущее количество шаров каждого вида. (5)

Вариант 4. Требуется модернизировать модель следующим образом:

1. При соударениях шаров друг с другом должен изменяться их радиус `r` по следующему алгоритму: радиус того шара, у которого скорость после соударения будет больше, должен увеличиться в 2 раза, а радиус шара с меньшей скоростью должен уменьшиться вдвое.
2. При увеличении радиуса шара до 40 единиц, шар должен исчезать.
3. При уменьшении радиуса шара до 2,5 единиц – шар должен превратиться в 2 шара нормальной величины (`r` = 15 единиц).
4. Шары должны быть пронумерованы, номера шаров не должны изменяться и повторяться. Номер должен быть виден на презентации рядом с шаром. (5)

Вариант 5. Требуется модернизировать модель следующим образом:

1. При старте модели создаются шары только 3-х основных цветов: красный, зеленый и синий.
2. При соударениях шаров друг с другом должен изменяться их цвет по следующему правилу:
    a. Сравнивается интенсивность каждой компоненты цвета обоих шаров.
    b. Если интенсивности компонент разные, то компонента имевшая значение = 0, получит значение = 255, а компонента, имевшая значение 255 не изменится.
    c. Если интенсивности компонент одинаковые, то компонента имевшая значение = 0 не изменится, а компонента, имевшая значение 255, получит значение = 0.
    d. Например, сталкиваются 2 шара. 1-й шар окрашен в пурпурный цвет (компоненты RGB имеют следующие значения: 255, 0, 255). 2-й шар окрашен в голубой цвет (компоненты RGB имеют следующие значения: 0, 255, 255).
    e. После столкновения оба шара окрасятся в цвет, определяемый так: красная компонента = 255. Зеленая компонента = 255. Синяя компонента = 0. Получится желтый цвет.
3. Шар, получивший белый цвет делится на 2 шара случайных основных цветов, т.е. белый шар, так же как и новый шар получает случайный основной цвет.
4. Шар, получивший черный цвет удаляется.
5. Рядом с каждым шаром должна быть надпись, указывающая на значения компонент R, G, и B. (5)

Вариант 6. Требуется модернизировать модель следующим образом:

1. Рабочая зона должна быть поделена на 2 части, соответственно левую и правую.
2. Шары должны иметь свои уникальные номера, видимые на презентации.
3. Шары с четным номером могут пересекать разделительную линию только в направлении слева направо.
4. Шары с нечетным номером могут пересекать разделительную линию только в направлении справа налево.
5. Кнопка «добавить шар» должна создавать шары с уникальным номером.
6. В презентации модели отразить количество шаров в левой и правой частях по отдельности. (5)

Вариант 7. Требуется модернизировать модель следующим образом:

1. Рабочая зона должна быть поделена на 2 части, соответственно левую и правую.
2. При прохождении разделительной линии слева направо шары исчезают с вероятностью 0,3.
3. При прохождении разделительной линии справа налево появляется 3 новых шара с вероятностью 0,1.
4. В презентации модели отразить общее число шаров и количество шаров, прошедших разделительную линию в том или ином направлении по отдельности. (4)

Вариант 8. Требуется модернизировать модель следующим образом:

1. После каждого соударения со стенками должен изменяться цвет шара по следующему правилу:
    a. При соударении с вертикальными стенками каждая компонента RGB-цвета увеличивается на N единиц. По достижении величины 255 единиц, компонента больше не увеличивается.
    b. При соударении с горизонтальными стенками каждая компонента RGB-цвета уменьшается на N единиц. По достижении величины 0 единиц, компонента больше не уменьшается.
2. В презентации сделать слайдер, изменяющий число N в диапазоне от 10 до 50.
3. Шар, получивший черный цвет исчезает.
4. Шар, получивший белый цвет порождает новый шар красного цвета, а сам меняет цвет на синий.
5. В презентации модели отразить общее количество шаров. Рядом с каждым шаром сделать надпись, указывающую на значения компонент R, G, и B. (4)

Вариант 9. Требуется модернизировать модель следующим образом:

1. Шары должны изменять свой цвет по следующему правилу:
    a. Величина каждой компоненты RGB уменьшается на N единиц в секунду.
    b. При достижении нуля компонента RGB более не уменьшается.
2. Шары черного цвета исчезают. При исчезновении шара, с вероятностью 0,5 появляются 2 шара случайного цвета и со случайным значением скорости. Один шар появляется в правом верхнем углу, другой – в левом верхнем.
3. В презентации модели отразить общее количество шаров. Рядом с каждым шаром сделать надпись, указывающую на значения компонент R, G, и B. (4)

Вариант 10. Требуется модернизировать модель следующим образом:

1. Каждый шар должен иметь свой уникальный, неизменный номер, видимый на презентации.
2. При столкновении шаров одинаковой четности (четный – четный или нечетный – нечетный) должен появиться еще один шар в левом нижнем углу рабочего пространства.
3. При столкновении шаров разной четности (четный – нечетный) должен исчезнуть тот шар, сумма RGB-компонент цвета у которого больше.
4. В презентации модели отразить общее количество шаров. Рядом с каждым шаром сделать надпись, указывающую на значения компонент R, G, и B. (4)

Вариант 11. Требуется доработать модель следующим образом:

1. В модели 2 вида шаров - красные и желтые. При столкновении красного шара с желтым, желтый шар исчезает.
2. Время жизни красных шаров 15 сек (сделать слайдер с диапазоном от 5 до 50 сек). По истечении срока жизни красный шар исчезает. Срок жизни красного шара обнуляется при столкновении с желтым.
3. Красные шары делятся при столкновении с красными с вероятностью 0.3 (при делении получается новый красный шар, со случайной скоростью и координатами).
4. Желтые шары делятся аналогично при столкновении с желтыми с вероятностью 0.1. (4)

Вариант 12. Требуется доработать модель следующим образом:

1. При соударении друг с другом появлялся еще один шар с вероятностью обратно пропорциональной объему популяции шаров. Зависимость вероятности успешного деления шаров задана таблицей.

Кол-во шаров	Вероятность деления
<=10	1
12	0.9
14	0.8
16	0.5
18	0.2
>=20	0

2. Время жизни шаров определяется нормальным законом распределения с мат.ожиданием 10 сек (сделать слайдер с диапазоном от 5 до 50 сек) и средним квадратическим отклонением 3 сек.
3. Презентация модели должна показывать текущее количество шаров. (4)

Вариант 13. Требуется модернизировать модель следующим образом:

1. При соударениях шаров друг с другом должен изменяться их радиус `r` по следующему алгоритму: радиус того шара, который в момент соударения находится выше, должен увеличиться на 50%, а радиус нижнего шара должен уменьшиться на 50%.
2. При увеличении радиуса шара до 40 единиц, шар должен исчезать.
3. При уменьшении радиуса шара до 4 единиц – шар должен превратиться в 2 шара,

радиусом 15, при этом новый шар должен возникнуть в правом верхнем углу.
4. Рядом с каждым шаром должна быть надпись, указывающая на значения компонент R, G, и B. (4)

Вариант 14. Требуется доработать модель таким образом, чтобы при ударе о левую стенку шар раздваивался (т.е. создавалась его копия), при этом скорость нового шара должна быть такая же, как у старого по абсолютной величине и направлению. Начальная координата нового шара должна быть в *центре* рабочего пространства. При ударе о правую стенку шары должны исчезать с вероятностью 0.3. Модель должна показывать текущее количество шаров и абсолютную скорость каждого шара. (4)

Вариант 15. Требуется доработать модель следующим образом:

1. В модели 2 вида шаров – синие и желтые. При столкновении шаров разного цвета шар с б*о*льшей скоростью исчезает с вероятностью 0,3.
2. Если шар не исчезнет после третьего столкновения с шаром другого цвета, то появляются еще 2 шара. Скорости вновь созданных шаров = 0. Координаты – случайные. Цвет – как у прародителя.
3. В презентации модели показывать, сколько столкновений перенес каждый шар с шаром другого цвета. (4)

Вариант 16. Требуется доработать модель следующим образом:

1. В модели 2 вида шаров – голубые и пурпурные. При столкновении шаров разного цвета, исчезает шар с меньшей скоростью.
2. Время жизни шаров 15 сек. По истечении срока жизни шары исчезают. Срок жизни шаров обнуляется при столкновении с шаром другого цвета.
3. При столкновении шаров одного цвета создается новый шар с вероятностью 0.5. Этот шар возникает того же цвета, со случайными координатами и направлением движения. Скорость нового шара равна сумме скоростей столкнувшихся шаров.
4. В презентации модели показывать время жизни каждого шара. (4)

Вариант 17. Требуется модернизировать модель следующим образом:

1. При старте модели создаются шары только 3-х основных цветов: красный, зеленый и синий.
2. При столкновениях шаров одинакового цвета исчезает тот из них, скорость которого больше.
3. При столкновениях шаров разных цветов появляется шар случайного основного цвета в центре рабочей области.
4. В презентации модели отразить количество красных, зеленых и синих шаров по отдельности. (4)

Вариант 18. Требуется модернизировать модель следующим образом:

1. Шары должны исчезать после двух, следующих друг за другом, соударений с вертикальными стенками, соударения шара с шаром не учитывать.
2. После двух, следующих друг за другом, соударений с горизонтальными стенками должен появиться новый шар в верхнем левом углу.
3. В презентации модели отразить количество шаров, около каждого шара показать количество соударений со стенками (отдельно с вертикальными, отдельно с

горизонтальными). (4)

Вариант 19. Требуется модернизировать модель следующим образом:

1. После трех, следующих друг за другом, соударений с горизонтальными стенками должен появиться новый шар в произвольном месте, с произвольной скоростью. Соударения шара с шаром не учитывать.
2. Шары должны исчезать после истечения времени жизни шаров. Время жизни шаров на старте модели = 15 сек. и должно регулироваться слайдером в диапазоне от 5 до 60 сек.
3. В презентации модели отразить общее количество шаров, около каждого шара показать количество соударений с горизонтальными стенками. (4)

Вариант 20. Требуется модернизировать модель следующим образом:

1. Рабочая зона должна быть поделена на 2 части, соответственно верхнюю и нижнюю.
2. Шары в модели имеют случайный размер (радиус шаров находится в диапазоне от 4 до 40 единиц).
3. При соударении шаров в верхней зоне исчезает тот шар, сумма RGB-компонент цвета у которого меньше.
4. При соударении шаров в нижней зоне появляется новый шар в случайном месте, при этом радиус нового шара равен полусумме радиусов столкнувшихся шаров.
5. В презентации модели отразить общее количество шаров. Рядом с каждым шаром указать его радиус.(4)

Вариант 21. Требуется модернизировать модель следующим образом:

1. После некоторого количества (N), следующих друг за другом, соударений с горизонтальными стенками должен появиться новый шар в произвольном месте, с произвольной величиной скорости и вертикальным направлением движения. Соударения шар-шар не учитывать.
2. В презентации сделать слайдер, изменяющий число N в диапазоне от 1 до 10.
3. При соударении с вертикальными стенками шары должны исчезать.
4. В презентации модели отразить общее количество шаров, около каждого шара показать количество соударений с горизонтальными стенками. (4)

Вариант 22. Требуется модернизировать модель следующим образом:

1. При соударениях шаров друг с другом должен изменяться их радиус `r` по следующему алгоритму: радиус того шара, у которого скорость после соударения будет меньше должен увеличиться в 2 раза, а радиус шара с б*о*льшей скоростью должен уменьшиться вдвое.
2. При увеличении радиуса шара до 60 единиц, шар должен исчезать (лопаться).
3. При уменьшении радиуса шара до 1,5 единиц – шар должен превратиться в 2 шара нормальной величины. (4)

Вариант 23. Требуется модернизировать модель следующим образом:

1. При старте модели создаются шары только 3-х основных цветов: красный, зеленый и синий.
2. При столкновениях шаров одинакового цвета появляется шар того же цвета в левом

нижнем углу.
3. При столкновениях шаров разных цветов исчезает тот из них, скорость которого меньше.
4. В презентации модели отразить количество красных, зеленых и синих шаров по отдельности. (4)

Вариант 24. Требуется доработать модель следующим образом:

1. В модели 2 вида шаров - красные и синие. При столкновении красного шара с синим, синий шар исчезает с вероятностью 0,2.
2. Если синий шар не исчезнет после третьего столкновения с красным, то он делится на 2 синих шара. Скорость вновь созданного шара = 0. Координаты – случайные.
3. В презентации модели показывать (цифрой на шаре), сколько столкновений перенес каждый синий шар с красным. При делении шара счетчик столкновений должен обнуляться. (4)

Вариант 25. Требуется доработать модель таким образом, чтобы при ударе о землю возникал новый шар.

1. Скорость нового шара должна иметь случайное направление.
2. Начальная координата нового шара должна быть в центре рабочего пространства. При ударе о боковые стенки шары должны исчезать.
3. Модель должна показывать текущее количество шаров. (4)

Вариант 26. Требуется доработать модель таким образом, чтобы при ударе о стенки радиус шара уменьшался вдвое, когда радиус станет меньше 2 единиц, шар должен исчезнуть. Модель должна показывать текущее количество шаров в виде графика. (3)

Вариант 27. Требуется доработать модель таким образом, чтобы при каждом третьем ударе о стенки радиус шара увеличивался вдвое, когда радиус станет больше 80 единиц, шар должен исчезнуть. Модель должна показывать текущее количество соударений со стенками для каждого шара. (3)

ЗАДАНИЕ 12

МОДЕЛЬ СЕГРЕГАЦИИ Т.ШЕЛЛИНГА

ЦЕЛИ ЗАНЯТИЯ

- совершенствование методов агентного моделирования;

ФОРМА ОРГАНИЗАЦИИ ЗАНЯТИЯ

Фронтальная.

СТУДЕНТ ДОЛЖЕН ЗНАТЬ

- технологию построения моделей,
- понятия: агент, среда, вложенный объект,
- основы языка Java,
- интерфейс программы AnyLogic.

СТУДЕНТ ДОЛЖЕН УМЕТЬ

- создавать агентные модели в программе AnyLogic;
- пользоваться справкой AnyLogic.

ОБЕСПЕЧЕННОСТЬ

- компьютер с установленной программой AnyLogic версии 6,
- настоящий курс лабораторно-практических работ.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

С точки зрения практического применения агентное моделирование можно определить как метод имитационного моделирования, исследующий поведение отдельных агентов и то, как это поведение определяет поведение всей системы в целом. При создании агентной модели, разработчик задает параметры агентов (это могут быть люди, компании, активы, проекты, транспортные средства, города, животные и т.д.), определяет их поведение, помещает их в некую окружающую среду, устанавливает возможные связи, после чего запускает моделирование. Индивидуальное поведение каждого агента образует глобальное поведение моделируемой системы.

12.1. ОПИСАНИЕ ПРОБЛЕМЫ

Модель сегрегации Шеллинга – классическая агентная модель системы, способной к самоорганизации. Две социальные группы, например, две различные расы; мужчины и женщины; курящие и некурящие и т.д. помещаются на шахматную доску, где каждая клетка представляет собой дом, в котором может находиться не больше одного человека.

Человек счастлив, если определенный процент его соседей принадлежит к той же социальной группе. Счастливые люди остаются на месте, в то время как несчастливые перемещаются на другие свободные места.

Шеллинг установил, что если задать "условия счастья", при которых отдается предпочтение сегрегации, то доска быстро примет вид сегрегированного шаблона расположения. Удивительно, но полная сегрегация установится даже в том случае, если индивидуумы лишь слегка склонны к тому, чтобы иметь соседей одного с ними социального типа.

Требуется построить модель сегрегации Шеллинга с возможностью интерактивного изменения "условия счастья" по ходу выполнения модели, изучить ее поведение и выполнить необходимые доработки.

12.2. МОДЕЛЬ

Создайте новую модель **Сегрегация** на основе шаблона **Агентное моделирование**. Начальное количество агентов установите `9000`. Выберите тип пространства `Дискретное`, остальные параметры оставьте без изменения.

12.2.1. КЛАСС АКТИВНОГО ОБЪЕКТА – PERSON

Откройте диаграмму класса `Person`, вы увидите небольшой силуэт человека в начале координат. Увеличьте масштаб для упрощения работы, выделите мышью этот силуэт и удалите его. Нарисуйте в начале координат синий квадрат. Во вкладке **Дополнительные** панели свойств этого квадрата укажите расположение по осям X и Y: 0 и 0, а ширину и высоту: 3 и 3, соответственно.

Перейдите на вкладку **Динамические** и в поле **Цвет заливки** напишите: `color`.

Теперь, нужно объявить этот параметр `color` типа `Color` со значением по умолчанию:

```
randomTrue( 0.5 ) ? Color.blue : Color.green
```

Как Вы помните, это условное выражение присвоит начальное значение параметру `color`, с вероятностью 50%, либо синий, либо зеленый цвет.

Из постановки проблемы видно, что каждый агент может быть «счастлив» и оставаться на месте, или «несчастлив» и стремиться переехать в другое место. Поэтому, в классе `Person` нужно объявить простую переменную `счастье` типа `boolean`. На этом с агентом пока закончим и перейдем к конструированию класса `Main`.

12.2.2. КЛАСС АКТИВНОГО ОБЪЕКТА – MAIN

Откройте диаграмму класса `Main`, вы увидите вложенный реплицированный объект `people`, объект `environment` и вложенную презентацию (небольшой квадрат), рис. 12.1.

В свойствах вложенного объекта `people` мы увидим, что он связан со средой `environment` и, что в его основе лежит класс `Person`. Количество агентов можно изменить в поле **Количество**.

В свойствах вложенного объекта `environment`, на вкладке Основные, следует поставить галочку в поле **Выполнять шаги**. На вкладке **Дополнительные** мы увидим, что наше пространство **Дискретное2D**, ширина, высота, число строк и столбцов – в соответствии с

выбранными параметрами шаблона. **Тип соседства** – `Мурово`, то есть соседями считаются агенты, располагающиеся в 8 соседних ячейках, в отличие от соседства `Евклидово`, где соседями считаются агенты, располагающиеся в 4 соседних ячейках.

Объявим параметр `порог`, который будет управлять допустимым процентом соседей, принадлежащих к той же социальной группе, со значением по умолчанию `0,7`.

Для наглядного представления процесса сегрегации создайте презентацию модели, как показано на рис. 12.2. Размер прямоугольника презентации 500 на 650, цвет – зеленый. Размер прямоугольника рабочего поля 402 на 402, расположение 48 по осям X и Y, цвет – blanchedAlmond. Для того, чтобы расположение агентов совпало с рабочим полем, переместите презентацию агента (небольшой квадрат) в позицию 50 по осям X и Y.

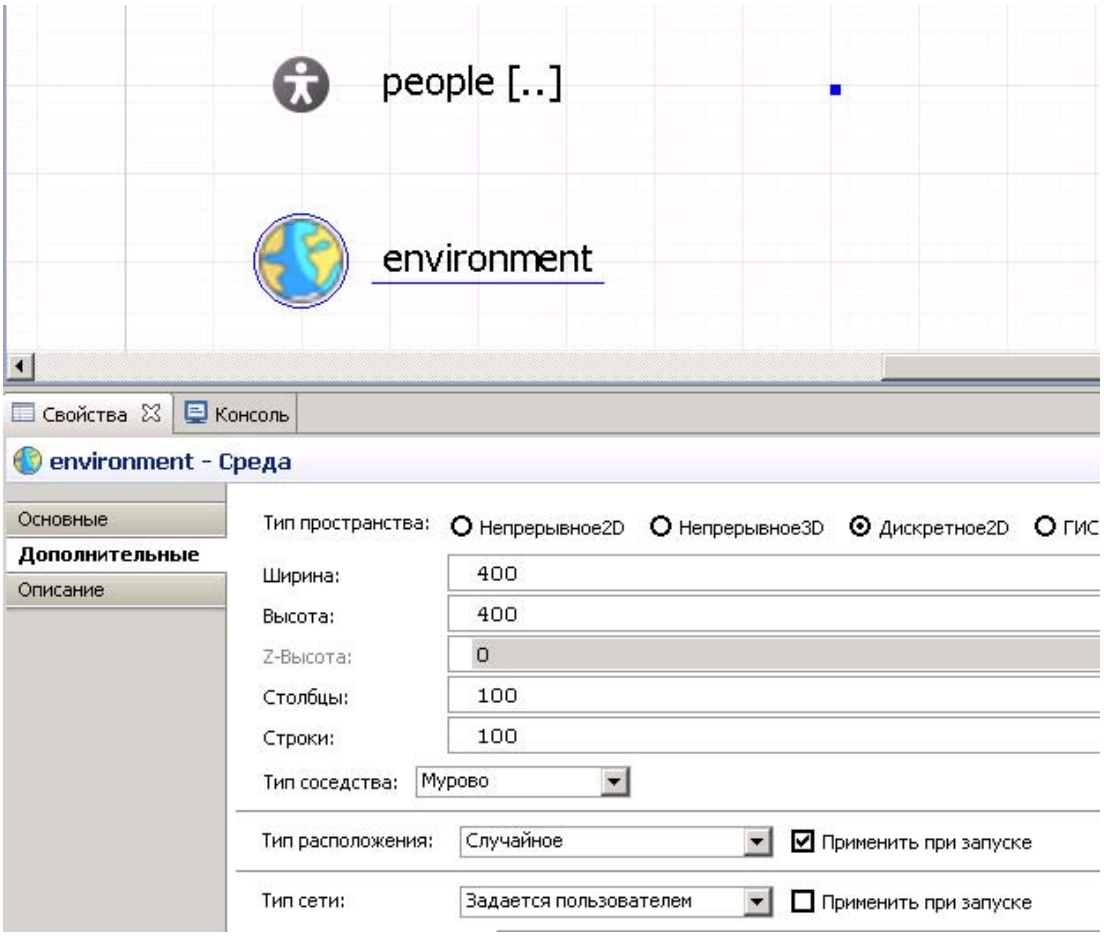


Рис. 12.1

Для управления параметром `порог` создайте слайдер в нижней части презентации, установите пределы регулировки от 0 до 1. Для дополнительного удобства добавим кнопки, передвигающие бегунок слайдера на 1% вверх или вниз. В поле **Действие:** кнопки «–» впишите следующий код:

```
порог = limitMin( 0, порог - 0.01 );

setControlValueToDefault( slider );
```

для кнопки «+» первая строка должна выглядеть иначе:

```
порог = limitMax( порог + 0.01, 1 );
```

Здесь метод `limitMin` возвращает `порог - 0.01`, если он больше или равен 0, соответственно `limitMax` возвращает `порог + 0.01`, если он меньше или равен 1.

Метод `setControlValueToDefault` устанавливает бегунок слайдера в соответствии с измененным параметром `порог`.

В верхней части презентации напишите текст: `Предпочтение иметь соседей того же самого цвета:`, а немного правее введите текст, показывающий значение параметра `порог`. Для этого во вкладке **Динамические** в поле **Текст:** введите:

`format(порог * 100) + "%"` – это позволит избавиться от десятичных значений и добавит знак процентов к числу.

Для того чтобы презентация открывалась в окне, совпадающем по размерам с нашей презентацией, откройте свойства эксперимента `Simulation` и на вкладке **Окно** измените ширину и высоту на значения `500` и `650`, соответственно. На вкладке **Презентация** выберите режим `Виртуальное время` – модель будет выполняться с максимальной скоростью.

12.2.3. ФУНКЦИОНИРОВАНИЕ МОДЕЛИ

Для того чтобы наша модель работала, нужно написать, что делать агентам на каждом шаге. Откройте свойства класса активного объекта `Person` и перейдите на вкладку **Агент**. В поле **Действие перед выполнением шага** напишите следующий код на языке Ява:

```
int рядом = 0;

Agent[] соседи = getNeighbors(); // строим массив агентов - соседей
if( соседи == null ) {
     счастье = true;             // нет соседей - это тоже хорошо
     return;}                    // выход

for( Agent a : соседи )          // перебираем весь массив соседей
     if(((Person)a).color == color)//сравниваем цвет соседа со своим
          рядом++;               // если цвет совпал - суммируем

//счастлив, если процент того же цвета больше/равен заданному порогу:
счастье = (рядом >= get_Main().порог * соседи.length);
```

В поле **Действие на шаге** напишите следующий код на языке Ява:

```
if(!счастье && randomTrue(0.3))//если не счастлив, то с вероятн. 0,3
     jumpToRandomEmptyCell();  // переход на другую свободную клетку
```

Комментарии можно не вводить.

12.2.4. СБОР СТАТИСТИКИ

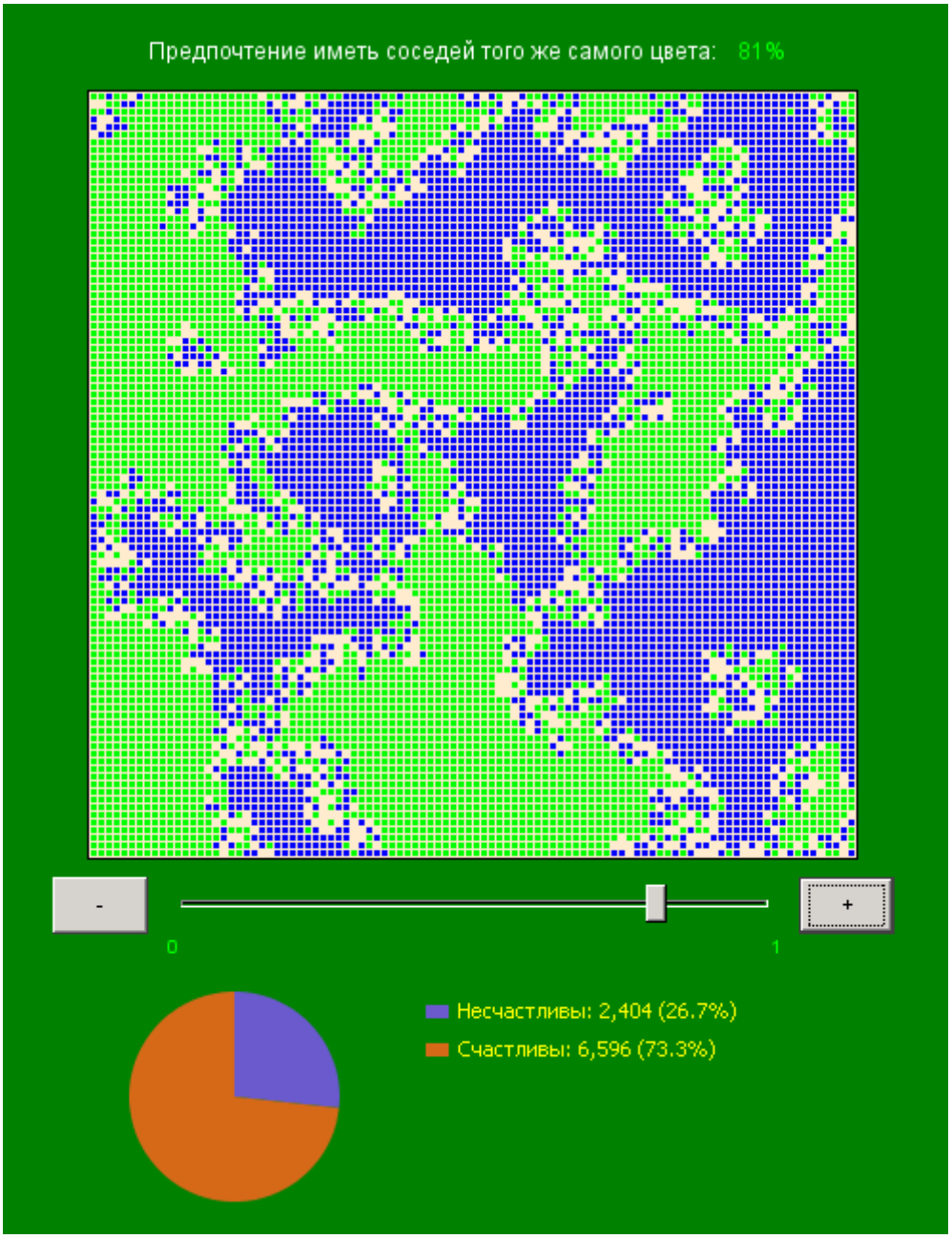
Реплицированные объекты поддерживают удобный способ сбора статистики. Поддерживаются следующие типы функций сбора статистики:

- количество,
- сумма,
- среднее,
- минимальное,
- максимальное.

Откройте свойства реплицированного объекта `people` и перейдите на вкладку **Статистика**. Нажмите на кнопку **Добавить ф-ю сбора статистики**. В появившиеся поля введите:

- Имя: `people_Сч`
- Тип: `кол-во`
- Условие: `item.счастье`

Текущий элемент реплицированного объекта (т.е. тот, по которому в текущий момент проходит итерационный цикл функции сбора статистики) доступен в этих полях как `item`.



В приведенном примере мы будем подсчитывать число «счастливых» людей, т.е. агентов у которых переменная `счастье` имеет значение `true`.

Заданную функцию сбора статистики можно будет вызывать, как и любую другую функцию: написав ее имя, после которого должны следовать круглые скобки. То есть, чтобы произвести сбор статистики с помощью заданной выше функции, нужно будет написать `people_Сч()`.

Аналогичным образом добавьте функцию сбора статистики `people_Не` для «несчастных» людей.

Рис. 12.2

Для отображения статистических данных можно использовать элементы из палитры **Статистика**. Перенесите на презентацию элемент **Круговая диаграмма**. В свойствах на вкладке **Основные** нажмите на кнопку **Добавить элемент данных** и в появившиеся поля введите:

- Заголовок: `Несчастливы`
- Значение: `people.people_He()`
- Цвет на Ваше усмотрение

Добавьте аналогичным образом еще один элемент данных для «счастливых» людей и перейдите на вкладку **Внешний вид**. Здесь выберите **Расположение** легенды справа от диаграммы, а **Цвет текста**: желтый, рис. 12.2.

Запустите модель, изучите ее поведение и выполните необходимые доработки в соответствии с заданием.

12.3. КОНТРОЛЬНЫЕ ЗАДАНИЯ

Для выполнения задания необходимо использовать справочную систему AnyLogic, в частности статьи: **Агенты, Дискретное пространство, Среда** и д.р.

1. Доработайте модель таким образом, чтобы переезд на другое место для 1-го типа происходил, если соседи справа-сверху и справа-снизу – другие, а для 2-го типа, если соседи слева-сверху и слева-снизу – другие. (5)
2. Доработайте модель таким образом, чтобы порог, сдерживающий людей от переезда на другое место, понижался в зависимости от времени проживания. После переезда порог – высокий, потом постепенно снижается и вынуждает к новому переезду. (5)
3. Доработайте модель таким образом, чтобы производился подсчет, и показывалось число неудовлетворенных агентов без использования функции сбора статистики. (5)
4. Добавьте в модель текстовое поле для выбора агента и флажок. При установленном флажке выбранный агент циклически перемещается на свободную ячейку. (5)
5. С помощью функции сбора статистики постройте временной график среднего времени «проживания» людей на одном месте. (5)
6. С помощью функции сбора статистики постройте столбиковую диаграмму, показывающую в каждом столбце число агентов с определенным количеством соседей (от 0 до 8). (5)
7. С помощью функции сбора статистики постройте столбиковую диаграмму, показывающую в каждом столбце число агентов с определенным количеством соседей того же типа (от 0 до 8). (5)
8. С помощью функции сбора статистики постройте столбиковую диаграмму, показывающую в каждом столбце число агентов с определенным количеством соседей противоположного типа (от 0 до 8). (5)
9. Доработайте модель таким образом, чтобы в моделировании участвовало 3 типа агентов, и на диаграмме показывалось сколько агентов имеют соседей только своего типа, сколько – одного другого, сколько двух других и сколько всех трех типов. (5)
10. Доработайте модель таким образом, чтобы переезд на другое место происходил, если сосед сверху – другой. (4)
11. Доработайте модель таким образом, чтобы переезд на другое место происходил, если сосед справа-сверху – другой. (4)
12. Добавьте в модель кнопку, при нажатии которой, случайно выбранный агент переедет на другое место. (4)
13. Доработайте модель таким образом, чтобы переезд на другое место происходил, если соседи справа-сверху и слева-сверху – другие. (4)
14. Доработайте модель таким образом, чтобы переезд на другое место происходил, если сосед справа – другой. (4)

15. Как установить синхронизацию действий агентов? (3)
16. С помощью функции сбора статистики постройте график зависимости количества «счастливых» и «несчастных» людей от времени. (3)
17. Какие преимущества и недостатки дает объявление реплицированных объектов агентами? (3)

ЗАДАНИЕ 15-16

ПЕШЕХОДНАЯ БИБЛИОТЕКА

ЦЕЛИ ЗАНЯТИЯ

- освоение методов дискретно-событийного моделирования;
- знакомство с Пешеходной библиотекой;
- построение модели детского спортивно-оздоровительного лагеря.

ФОРМА ОРГАНИЗАЦИИ ЗАНЯТИЯ

Фронтальная.

СТУДЕНТ ДОЛЖЕН ЗНАТЬ

- понятия: дискретно-событийная система, параметры объекта;
- интерфейс программы AnyLogic.

СТУДЕНТ ДОЛЖЕН УМЕТЬ

- создавать модели в программе AnyLogic,

ОБЕСПЕЧЕННОСТЬ

- компьютер с установленной программой AnyLogic версии 6.6,
- настоящий курс лабораторно-практических работ.

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

О ПЕШЕХОДНОЙ БИБЛИОТЕКЕ

Пешеходная библиотека является высокоуровневой библиотекой для моделирования движения пешеходов в физическом пространстве. Она позволяет моделировать здания, в которых движутся пешеходы (станции метро, стадионы, музеи), улицы, парки отдыха и т.д. В моделях, созданных с помощью Пешеходной библиотеки, пешеходы движутся в непрерывном пространстве, реагируя на различные виды препятствий в виде стен и других пешеходов.

Пешеходная библиотека AnyLogic наглядно визуализирует моделируемый процесс с помощью анимации и позволяет отслеживать плотность пешеходов в различных областях модели для того, чтобы убедиться в способности системы справиться с потенциальным ростом нагрузки, вычислить время пребывания пешеходов в каких-то определенных участках модели, выявить возможные проблемы, возникающие при перепланировке интерьера здания, и т.д.

15.1. ПОСТАНОВКА ЗАДАЧИ

Создадим простейшую модель, моделирующую движение детей/подростков (назовем их условно пионерами) при заселении в начале смены в спортивно-оздоровительный лагерь.

Перед тем, как расселиться по комнатам, пионеры проходят медосмотр. Пионеры, прошедшие медосмотр собираются в зоне ожидания, а затем, получив приглашение, расходятся по комнатам в зависимости от номера отряда. Эта модель демонстрирует, как моделировать поток пешеходов и простейшие сервисы в Пешеходной библиотеке AnyLogic.

15.2. МОДЕЛЬ

Создайте новую модель **Пионерлагерь** без использования шаблона. В свойствах модели на вкладке **Основные** в поле **Единицы модельного времени** выберите `Секунды`.

15.2.1. АНИМАЦИЯ МОДЕЛИ

Начнем с анимации модели, для того, чтобы графически задать объекты среды модели. Для этого добавим изображение плана лагеря. Мы не будем рисовать план в графическом редакторе, а просто вставим уже готовое изображение. Для этого откройте закладку **Презентация** панели **Палитра**. Перетащите элемент **Изображение** из палитры **Презентация** на диаграмму класса активного объекта. Задайте свойства изображения в панели **Свойства**. Щелкните мышью по кнопке **Добавить** и выберите файл изображения плана лагеря `atlanta_shema.jpg`. Вы увидите добавленное изображение в области предварительного просмотра на панели Свойства, рис. 15.1.

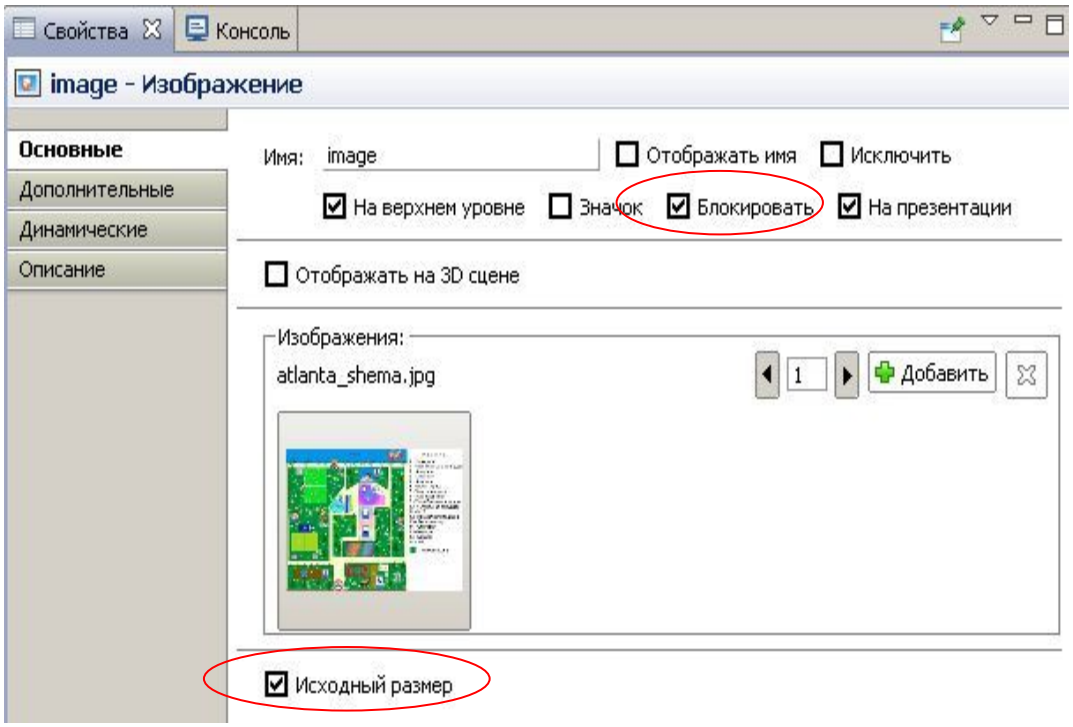


Рис. 15.1

Установите флажок **Исходный размер** для того, чтобы сохранить исходный размер изображения, а также защитите изображение от редактирования, установив флажок **Блокировать**. Вы не сможете выбрать заблокированную фигуру в графическом редакторе до тех пор, пока Вы не снимете с нее блокировку.

Теперь нужно нарисовать на анимации объекты моделируемой среды. Чтобы было легче рисовать, лучше отключить сетку и увеличить масштаб анимации с помощью соответствующих кнопок панели инструментов, рис. 15.2

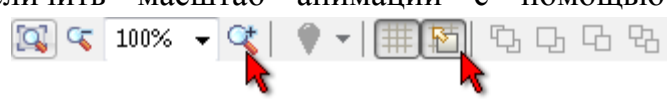


Рис. 15.2

Вначале мы нарисуем границу моделируемого нами пространства, играющую роль стен и ограждений.

Обведите с помощью ломаных линий границы интересующей нас части лагеря, газоны, здания и т.д., как показано на рисунке 15.3. Эти линии будут задавать границу моделируемой среды, но сначала их нужно объединить в одну группу, и эту группу указать в параметре **Стены** соответствующего библиотечного объекта.

Добавьте только что нарисованную ломаную в группу. Щелкните по ней правой кнопкой мыши (при этом она должна подсветиться синим цветом) и выберите **Группировка/ Создать группу** из контекстного меню, см. рис. 15.3.

Рис. 15.3

Для добавления другой ломаной в группу выделите ее мышью (ломаная подсветится синим цветом) в контекстном меню выберите **Группировка/Добавить в существующую группу**, появится кружок с перекрестием – значок группы, рис. 15.4. Выберите его и ломаная войдет в состав данной группы. Проделайте эти операции со всеми ломаными, ограничивающими перемещение пионеров.

Рис. 15.4

Нарисуйте с помощью элемента **Линия** вход в модель и выход из модели для пионеров, как показано на рисунке 15.5. Назовите их `вход` и `вход_в_корпус` соответственно. Обе линии должны находиться полностью внутри фигуры, задающей границу моделируемой среды.

Рис. 15.5

15.2.2. ДИАГРАММА ПРОЦЕССА

Теперь создадим диаграмму моделируемого процесса движения пионеров. С помощью диаграммы процесса в моделях пешеходной динамики задается поведение пешеходов. Диаграмма процесса в AnyLogic создается путем добавления объектов библиотеки из палитры на диаграмму класса активного объекта, соединения их портов и изменения значений свойств объектов в соответствии с требованиями модели.

Попадающие в моделируемую систему пешеходы будут последовательно проходить по блокам диаграммы процесса.

Помимо объектов, составляющих диаграмму процесса, модели Пешеходной библиотеки состоят из объектов, моделирующих объекты среды (стены, различные области, сервисы, очереди и т.д.). Чтобы задать объект среды, Вы должны вначале графически нарисовать его на анимации, а затем добавить соответствующий объект библиотеки на структурную диаграмму активного класса модели и задать необходимые свойства этого объекта.

Включите привязку к сетке. Добавьте на диаграмму класса `Main` следующие объекты Пешеходной библиотеки:

- `pedSource`
- `pedGoTo`
- `pedSink`
- `pedConfiguration`
- `pedGround`.

Разместите объекты так, как показано на рисунке 15.6, и соедините их порты.

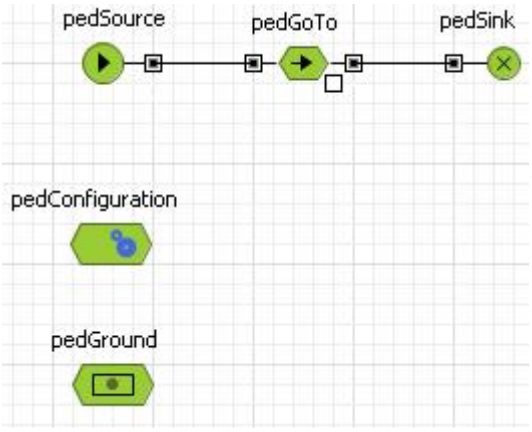


Рис. 15.6

Согласно принятым стандартам, блоки в диаграмме процесса обычно располагаются цепочкой слева направо, представляя собой последовательную очередность операций, которые будут производиться над пешеходом.

Теперь необходимо сконфигурировать каждый добавленный нами библиотечный объект.

Объект `PedGround` позволяет задавать двумерное пространство в моделируемой среде, представляющее собой «этаж», т.е. поверхность, по которой будут перемещаться пешеходы. Этажи могут быть ограничены какой-то стеной или быть неограниченными. Стены - это объекты, которые пешеходы не могут пересекать. Стены являются частью этажа, то есть одна стена не может быть использована несколькими этажами. В образовательной версии AnyLogic число этажей ограничено одним этажом.

Откройте закладку **Основные** панели свойств объекта `PedGround` и введите в поле **Стены (группа, необязательный)** имя нашей группы: `group`, созданной ранее именно для этой цели, рис.15.7.

Объект `PedSource` создает пешеходов. Обычно он используется в качестве начальной точки диаграммы процесса, формализующей поток пешеходов. В нашем примере он моделирует приход пионеров в лагерь.

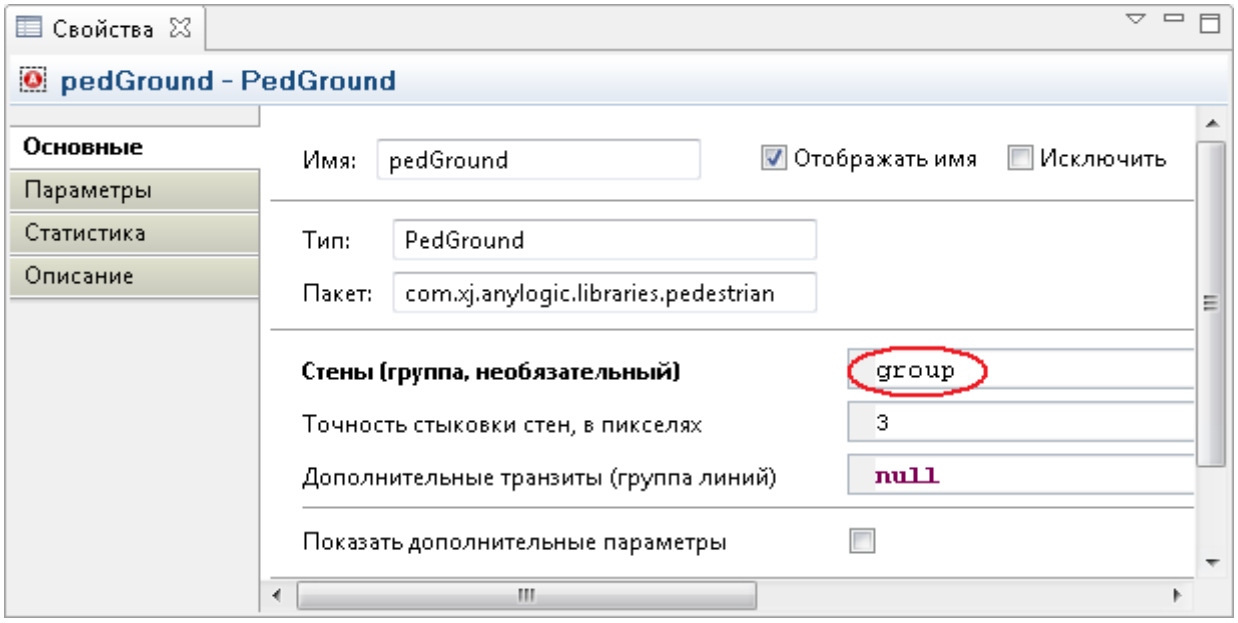


Рис. 15.7

Укажите имя объекта `PedGround`, задающего этаж, на который будут прибывать пионеры. В поле **Этаж (PedGround)** введите имя добавленного Вами ранее объекта **PedGround**: `pedGround`, рис. 15.8.

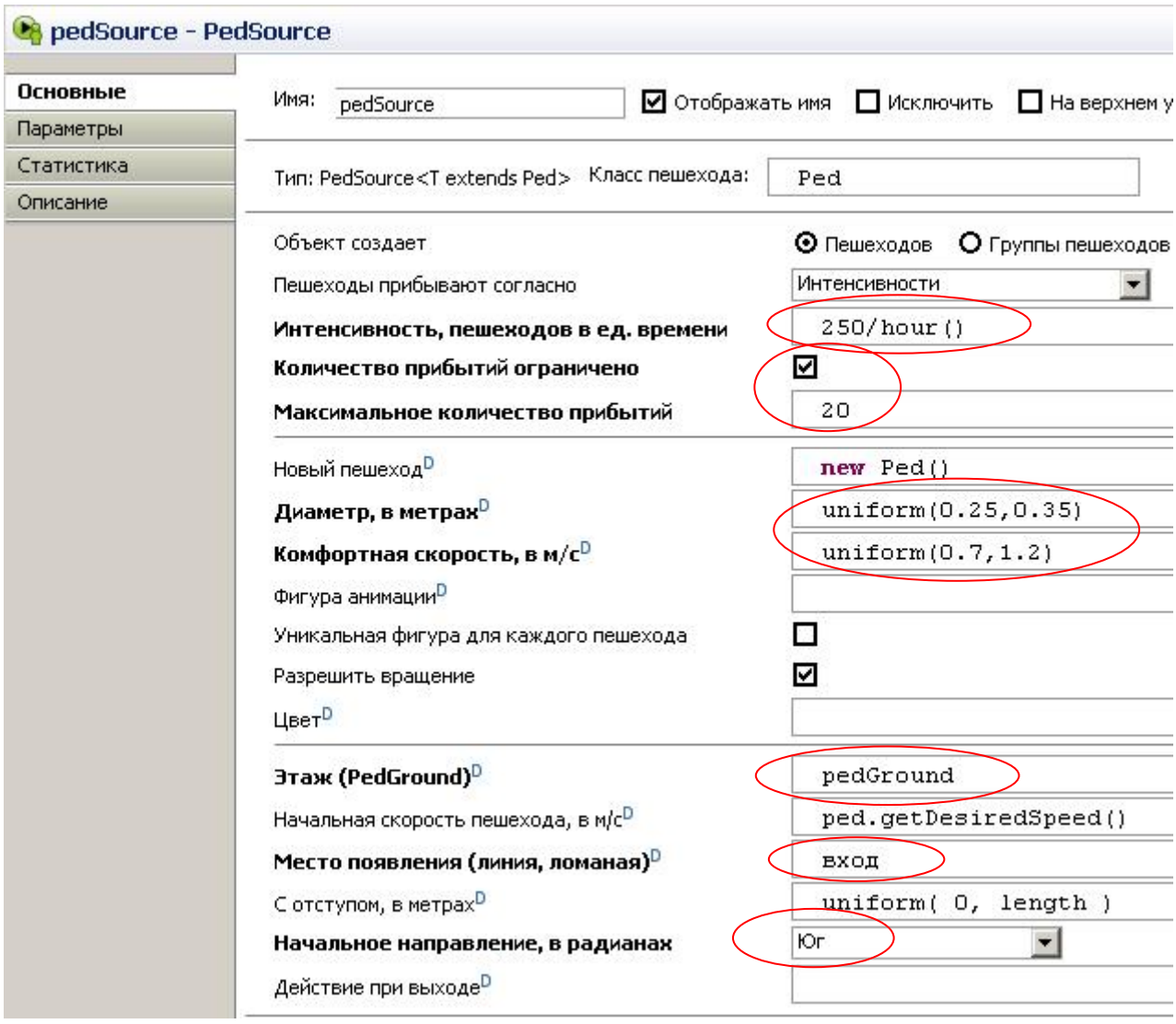


Рис. 15.8

Задайте место на этаже, где будут появляться пионеры. Место их появления на этаже может быть задано линией или ломаной. Введите `вход` (имя линии, нарисованной нами ранее для этой цели) в поле **Место появления (линия, ломаная)**. Теперь наши пионеры будут появляться в случайно выбранной точке линии `вход`. Для задания первоначального направления движения можно в пункте **Первоначальное направление, в радианах** выбрать `Юг`.

Интенсивность, пешеходов в ед. времени – установите `250/hour()`, т.е. пионеры будут прибывать в лагерь с интенсивностью 250 человек в час.

Поставьте галочку напротив пункта **Количество прибытий ограничено**, и укажите `20` в следующем пункте: **Максимальное количество прибытий**, т.е. прибудет 20 пионеров.

Поскольку наши пионеры – дети, скорректируем их размер, для этого в поле **Диаметр, в метрах введите:** `uniform(0.25,0.35)`, а среднюю скорость их перемещения установим в поле **Комфортная скорость, в м/с** с помощью функции `uniform(0.7,1.2)`, рис. 15.8. Следующий объект в созданной нами диаграмме процесса – `PedGoTo`. Этот объект моделирует перемещение пешеходов из текущего местоположения в другое (заданное параметром этого объекта). С помощью этого объекта мы будем моделировать то, как пионеры перемещаются от входа в ворота лагеря к жилому корпусу.

Задайте то место, куда будут перемещаться пешеходы, достигшие этого блока в диаграмме процесса. Такое место может быть задано линией или точкой, нарисованной на презентации. На данный момент мы хотим, чтобы пионеры, вошедшие в ворота, сразу двигались к жилому корпусу. Поэтому введите `вход_в_корпус` (имя линии, нарисованной нами на плане у входа в жилой корпус) в поле **Цель (точка, линия),** рис. 15.9. Теперь для того, чтобы покинуть моделируемую среду, пешеходы должны будут вначале дойти до заданной области выхода.

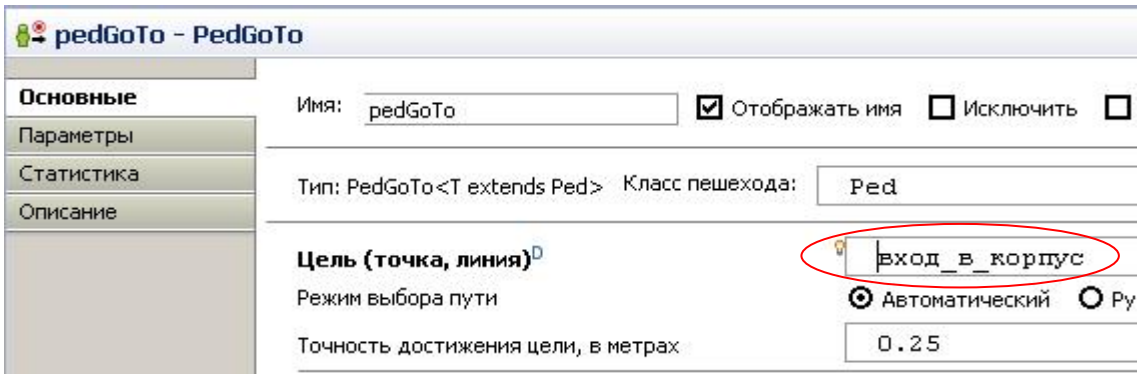


Рис. 15.9

Оставьте заданные по умолчанию свойства объекта `PedSink` без изменений. Этот объект удаляет поступивших в него пешеходов из моделируемой среды. Обычно объект используется в качестве конечной точки диаграммы процесса.

Объект `PedConfiguration` позволяет изменять общие настройки диаграммы процесса движения пешеходов, что отражается на производительности модели.

Сбросьте флажок **Скрыть фигуры среды**, чтобы не отображать на анимации фигуры, которые использовались для задания моделируемой среды (стен, различных областей, сервисов и т.д.), рис. 15.10.

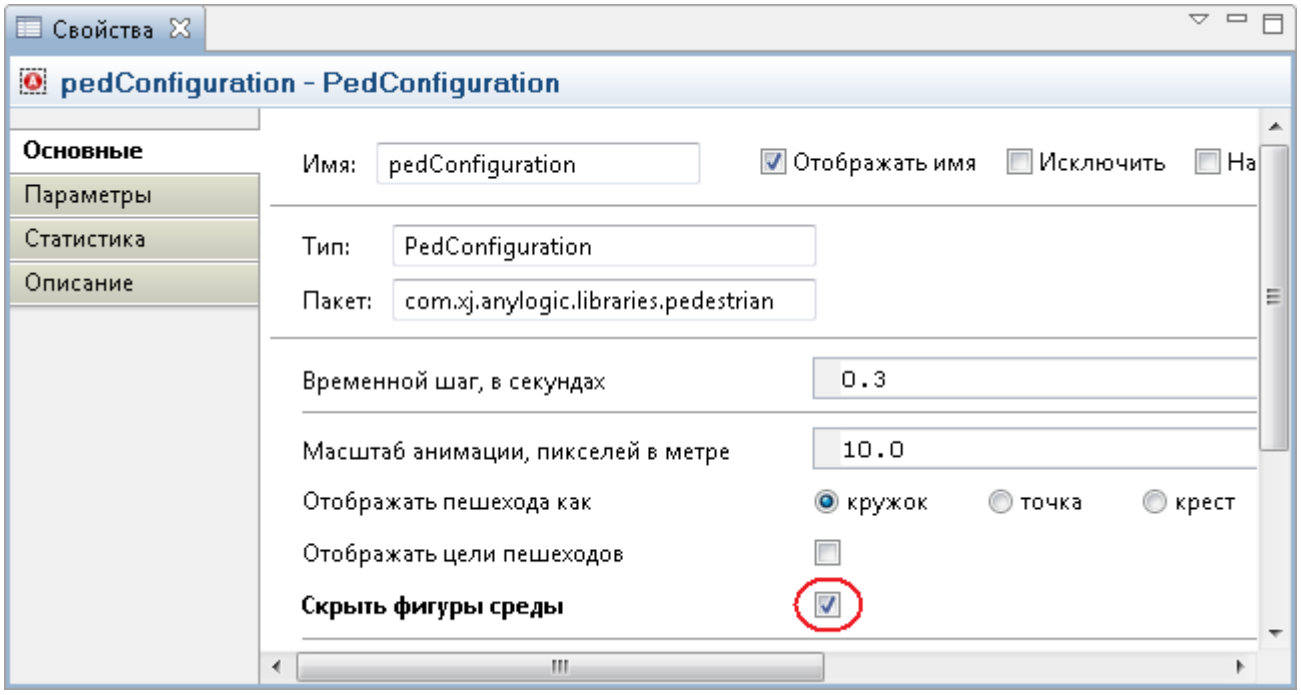


Рис. 15.10

Запустив модель, мы увидим движение пионеров от ворот лагеря до жилого корпуса. Одновременно можно наблюдать за перемещением заявок по структурной диаграмме.

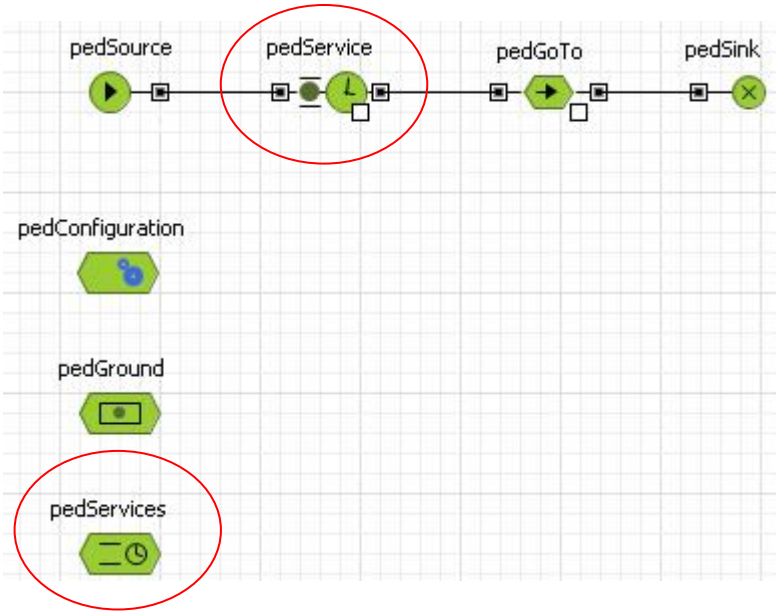
15.2.3. МОДЕЛИРОВАНИЕ МЕДПУНКТА

Построив простейшую пешеходную модель детского оздоровительного лагеря, будем наполнять ее функциональностью. На данном этапе нам нужно смоделировать медпункт, через который должны пройти все пионеры, заселяющиеся в лагерь.

Нарисуйте линию, которая будет графически задавать место осмотра пионеров в медпункте нашей модели. Создайте группу для этой линии, назвав ее `Медпункт`.

Рис. 15.11

Нарисуйте ломаную, которая будет графически задавать очередь в медпункт и поместите ее в другую группу, которую назовите `Мед_очередь`, рис. 15.11. Очень важно, где находится начальная точка ломаной, т.к. эта точка соответствует началу очереди. Поэтому ее нужно поместить рядом с медпунктом.



Теперь необходимо внести небольшие изменения в диаграмму процесса.

Добавьте в модель объекты `PedServices` и `PedService` и соедините последний, как показано на рис. 15.12.

Рис. 15.12

Объект `PedServices` задает группу одинаковых физических объектов обслуживания (например, несколько турникетов или автоматов по продаже билетов). Объект позволяет задавать очереди и сервисы в любой комбинации и задавать правила выбора сервисов – какую очередь выбрать, какой сервис выбрать, к какой очереди должен обращаться сервис, может ли сервис обслуживать несколько очередей и т.д.).

Задав такой объект, Вы можете ссылаться на него в диаграмме моделируемого Вами процесса с помощью блока `PedService`.

Впишите в поле **Сервисы (группа линий)** название группы, задающей линию сервиса (в нашем случае это `Медпункт`).

Введите имя группы, задающей очередь к сервису, в поле **Очереди (группа линий, ломаных)**: `Мед_очередь`.

Выберите **Тип сервиса**: `Точечный`. В Пешеходной библиотеке есть два типа сервисов: протяженные и точечные. Протяженные сервисы используются тогда, когда пешеходы должны пройти вдоль заданной линии сервиса, от начальной точки до конечной. Точечные сервисы используются, когда для обслуживания пешеход должен просто подойти к любой точке фигуры, задающей сервис. В нашем примере медпункт моделируются точечным сервисом.

Укажите **Время задержки**: `uniform(20.0*second(), 60.0*second())`. Это значит, что пионеры задержатся в медпункте от 20 до 60 секунд, рис. 15.13.

Объект `PedService` добавляется в диаграмму процесса, чтобы промоделировать, как пешеходы обслуживаются в сервисе, заданном объектом `PedServices`.

В поле **Сервис (PedServices)** укажите имя объекта `PedServices`, задающего сервис, через который должны пройти пионеры (медпункт). Введите здесь `PedServices`.

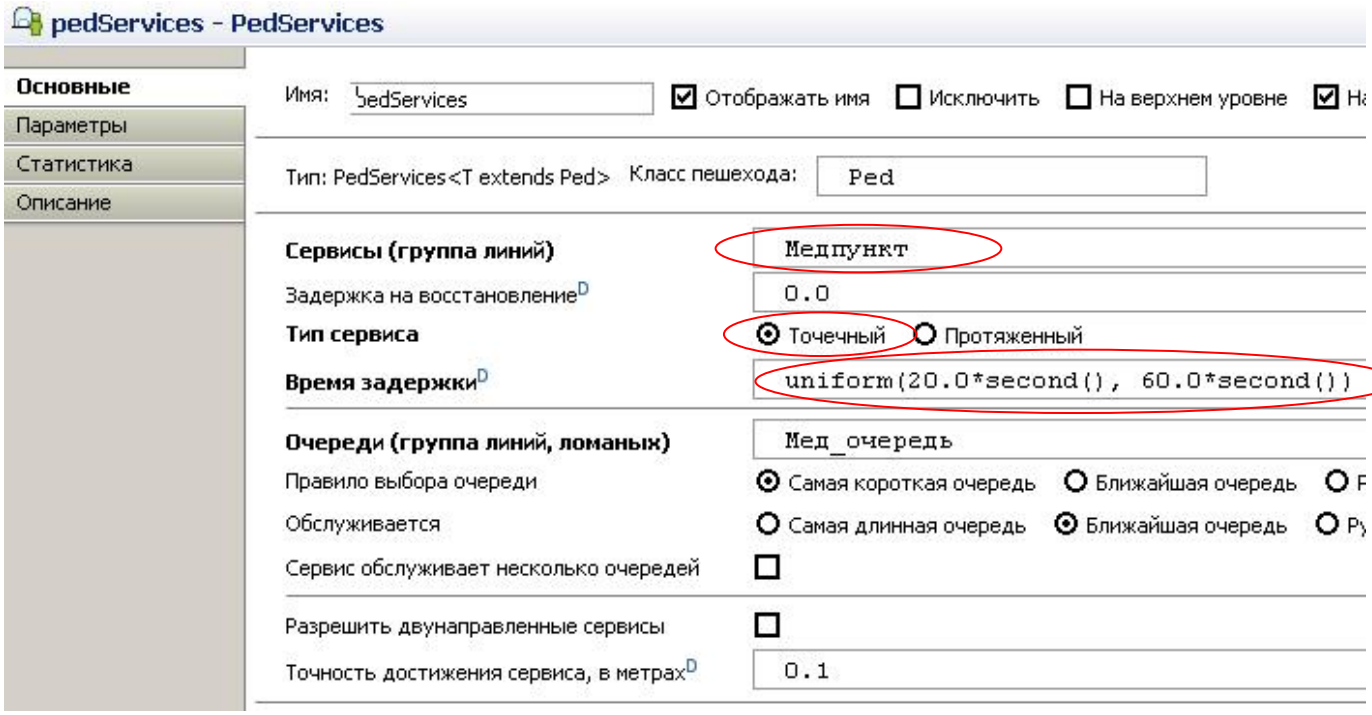


Рис. 15.13

Запустите модель и проследите за ее поведением с помощью анимации. Вы можете увидеть, что теперь пионеры проходят через медпункт, перед которым образуется очередь.

Некоторых пионеров медпункт может не допустить в лагерь, например по причине отсутствия медсправки. Чтобы смоделировать эту ситуацию используем объект `PedSelectOutput`.

Объект `PedSelectOutput` является блоком принятия решения Пешеходной библиотеки. Пешеход, вошедший в блок `PedSelectOutput`, будет перенаправляться в один из пяти выходных портов в зависимости от заданных для этих портов коэффициентов предпочтения.

Объект `PedSelectOutput` будет перенаправлять пионеров без медицинского допуска на выход, а пионеров с допуском – к жилому корпусу.

Добавьте на диаграмму объекты `PedSelectOutput` и `PedGoTo`, и соедините их, как показано на рис. 15.14.

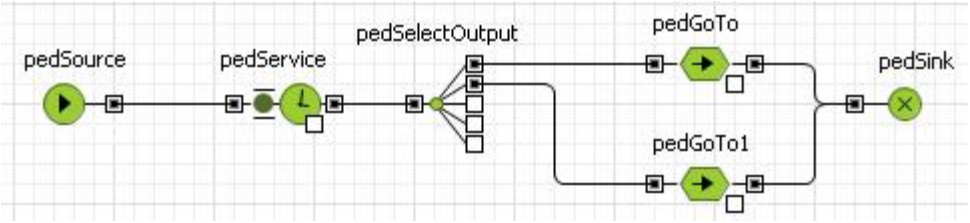


Рис. 15.14

Измените свойства объекта `pedSelectOutput`:

- **Имя**: `pedSelectOutput_med`;
- коэффициенты предпочтения для пионеров, имеющих мед.допуск (**Коэфф. предпочтения 1**) и не имеющих мед.допуска (**Коэфф. предпочтения 2**) установите `93` и `7`, соответственно, рис. 15.15. То есть, около 7% от общего числа пионеров не пройдет медосмотр.

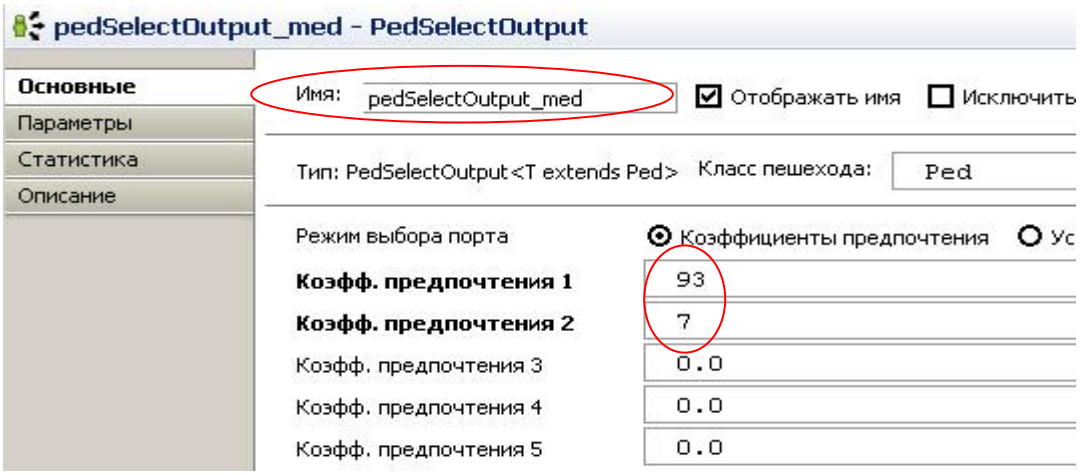


Рис. 15.15

Измените свойства объекта `pedGoTo1`:

- **Имя**: `На_выход`;
- **Цель (точка, линия)**: `вход`.

Запустив модель, мы увидим, что некоторые пионеры не прошли медосмотр и направились к выходу.

15.2.4. МОДЕЛЬНОЕ ВРЕМЯ

В Пешеходной библиотеке, предоставляемой для учебных заведений, есть ряд ограничений, среди которых ограничение на длительность эксперимента – 1 час. Для того чтобы программа не выдавала ошибку ограничим выполнение модели временем 60 минут.

В панели **Проекты**, выделите эксперимент `Simulation:Main` щелчком мыши, перейдите на страницу **Модельное время** панели **Свойства**. Здесь задаются установки модельного времени. Для того, чтобы модель останавливалась в заданное время выберите из выпадающего списка **Остановить** пункт `В заданное время`. Так как по умолчанию в качестве единиц модельного времени заданы секунды, можно просто установить конечное время 3600 единиц (секунд) в поле **Конечное время**.

Зададим интервал моделирования с помощью календарных дат, для этого установите флажок **Использовать календарь,** выберите из выпадающего списка **Остановить** пункт `В заданную дату` и установите начальную и конечную дату моделирования с помощью элементов управления **Начальная дата** и **Конечная дата**: `01.07.2012` в `10:00:00` и `01.07.2012` в `11:00:00` соответственно, рис.15.16.

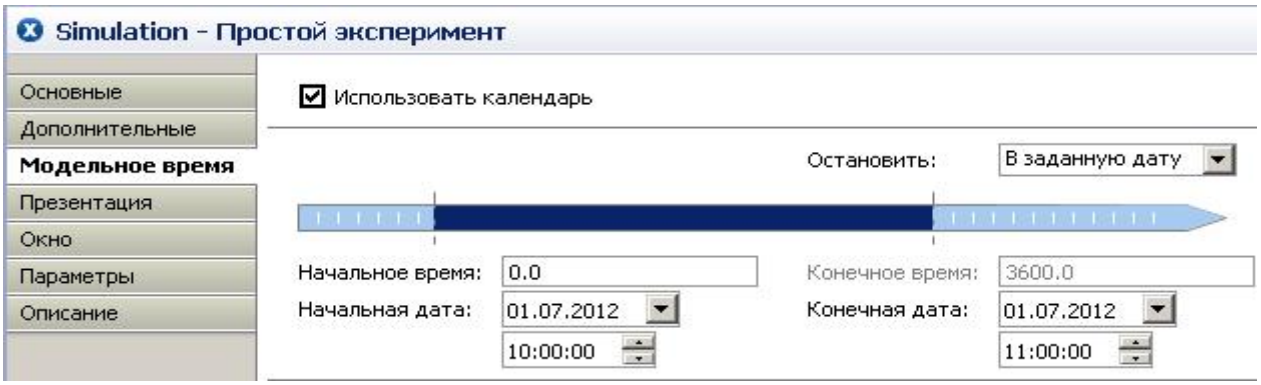


Рис. 15.16

Проверьте правильность работы модели.

15.2.5. ПЕРСОНАЛИЗАЦИЯ ПЕШЕХОДОВ

Так как пионеры сгруппированы в отряды, то мы хотели бы отобразить на анимации модели пионеров каждого отряда своим цветом. Для этого нужно в класс Ped, описывающий пешеходов, добавить дополнительное поле, содержащее номер отряда. Существующие классы в AnyLogic нельзя менять, но их можно переопределить, использовав как родительский класс.

Создадим новый класс `Pioner`, наследующий все поля и методы класса `Ped`, но имеющий дополнительное поле `division`, в котором будет храниться номер отряда.

Щелкните правой кнопкой мыши по самому верхнему пункту дерева **Проекты** и выберите **Создать | Java класс**. Введите имя класса `Pioner` и выберите `Ped` как родительский (базовый) класс (доступен в выпадающем списке). Щелкните мышью по кнопке **Далее**.

Задайте имя и тип создаваемого нами поля: `division` / `int`. Оставьте флажки **Создать конструктор** и **Создать метод toString()** установленными. Щелкните мышью по кнопке **Готово**.

Вы увидите Java редактор, в котором будет отображен автоматически сгенерированный код созданного класса. В нем ничего менять не нужно, просто закройте это окно.

После того как мы создали новый класс пешеходов его нужно ввести в модель. Для этого в объекте `PedSource` поля **Класс пешехода** и **Новый пешеход** замените `Ped` на `Pioner`, рис.15.17.

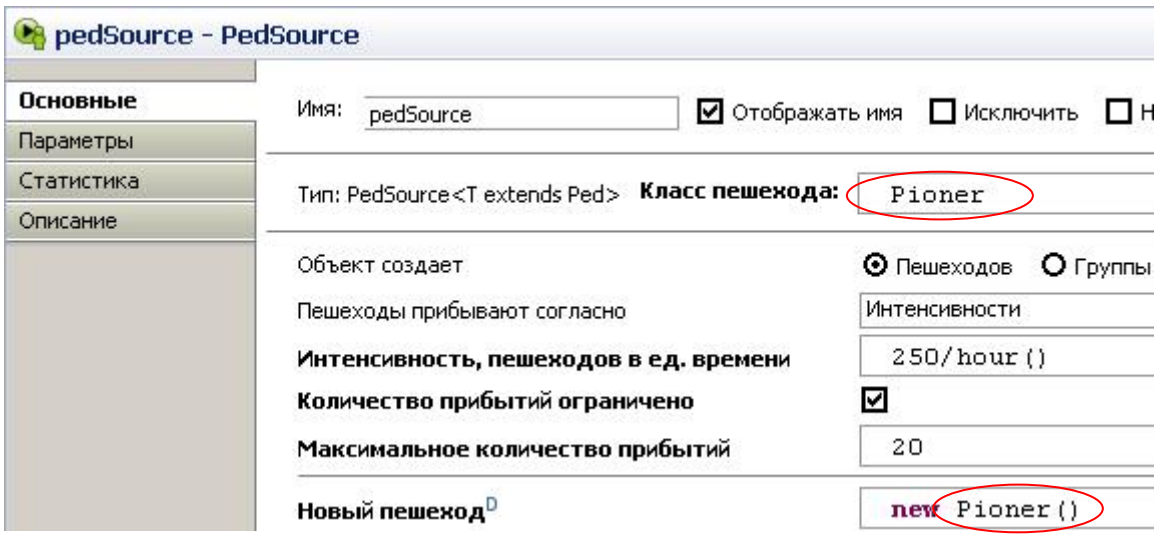


Рис. 15.17

Таким образом, в нашей модели будут создаваться объекты класса `Pioner`, имеющие поле `division` (для хранения номера отряда), наследующие все поля и методы своего предка – `Ped`.

Теперь нужно каждому пионеру назначить номер отряда, а каждому отряду для наглядности присвоить свой цвет. Пусть у нас в лагере будет 3 отряда с примерно одинаковым числом пионеров. Запишите в поле **Действие при выходе** объекта

PedSource следующий код:

```
if (uniform() < 0.33){ped.division =1; ped.setColor(Color.blue);} else
if (uniform() < 0.66){ped.division = 2; ped.setColor(Color.red);} else
                     {ped.division = 3; ped.setColor(Color.green);}
```

Объект PedSource при создании пешеходов типа Pioner будет каждому из них присваивать номер отряда (1, 2 или 3) и цвет (синий, красный или зеленый).

Запустите модель и убедитесь в ее безошибочной работе.

15.2.6. МОДЕЛИРОВАНИЕ ЗОНЫ ОЖИДАНИЯ

В нашей модели лагерь открывается в 10 часов утра, а медпункт начинает работу в 10:20. Создадим зону ожидания, где пионеры будут ждать приглашения на медосмотр.

Для этого будем использовать объект PedWait, который заставляет пешеходов перейти в заданное место и ожидать там в течение определенного периода времени, а также объект PedArea, задающий область ожидания или накладывающий ограничения на скорость пешеходов.

Добавьте в модель объекты PedArea и PedWait и соедините последний, как показано на рис. 15.18.

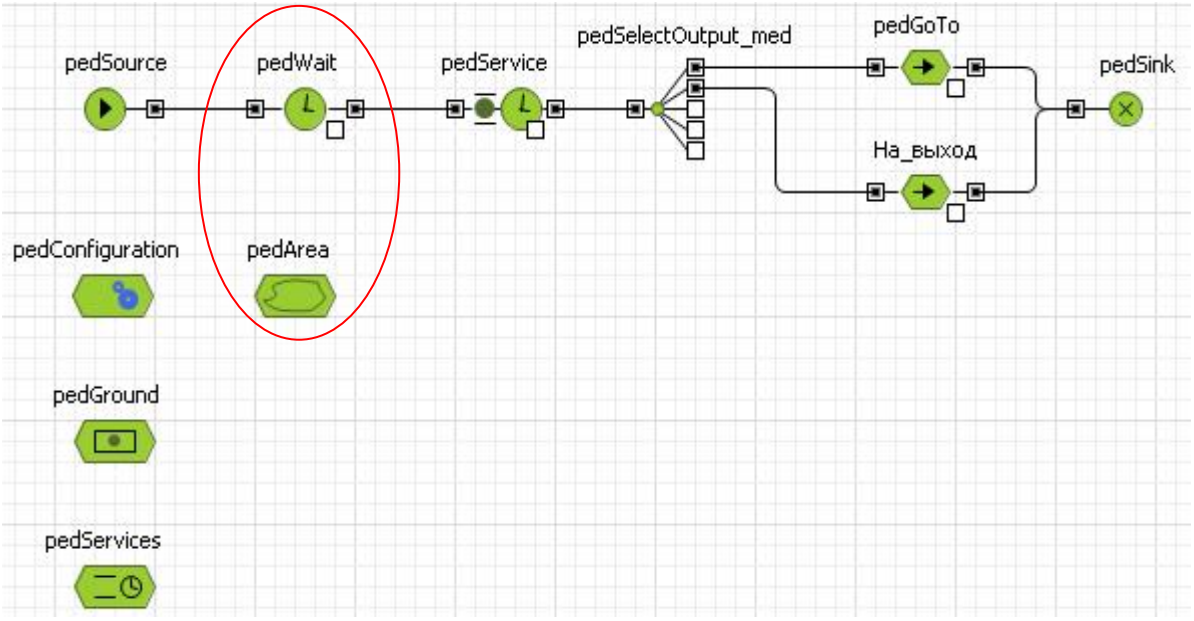


Рис. 15.18

Нарисуйте на анимации прямоугольник и назовите его ожидание_медосмотра, рис. 15.19.

Укажите в поле **Фигура (овал, прямоугольник, замкнутая ломаная)** объекта PedArea название прямоугольника, в котором пешеходы будут ожидать сигнала: ожидание_медосмотра, а в поле **Этаж**: pedGround.

Измените свойства объекта PedWait:

- **Имя**: pedWait_med;

- **Область ожидания (PedArea):** `pedArea;`
- **Тип**: `Ручной (по вызову метода free()).`

Если запустить модель, то мы увидим, что пионеры не будут проходить в медпункт, а будут ожидать около входа приглашения на медосмотр. Такое приглашение можно реализовать с помощью события, которое будет срабатывать в определенное время, и вызывать в объекте `PedWait` метод `freeAll()`, который прерывает выполнение команды для всех пешеходов и заставляет их покинуть объект через порт `OUT`.

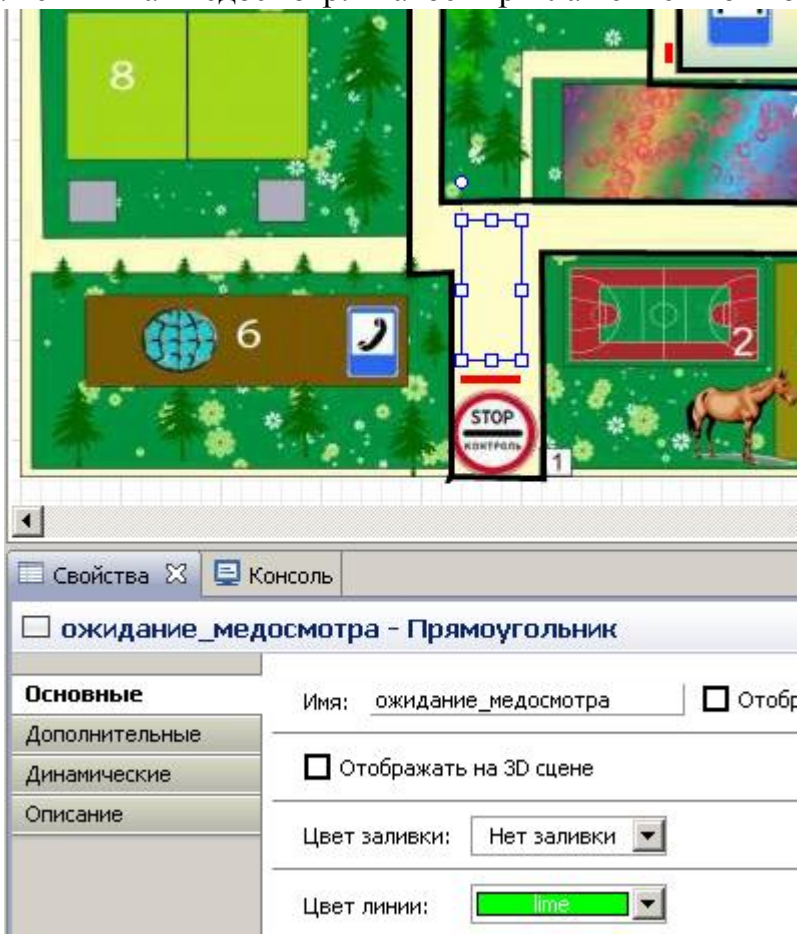


Рис. 15.19

Перенесите на диаграмму класса объект событие и измените свойства объекта `event`:

- **Режим**: `Циклический`;
- **Время первого срабатывания (абсолютное)**: `01.07.2012 10:05:00`;
- **Период**: `600`;
- **Действие**: `pedWait_med.freeAll().`

Запустите модель и убедитесь, что все пионеры прошли (или не прошли) медосмотр.

15.2.7. РАСПРЕДЕЛЕНИЕ ПИОНЕРОВ ПО КОМНАТАМ

После успешного прохождения медосмотра пионеры должны собраться возле концертной площадки и, получив приглашение, проследовать в комнату, соответствующую номеру отряда.

Как и в предыдущем случае, мы сделаем зону ожидания, где объект `PedWait` будет задерживать пионеров, но освобождать он будет не всех, а только соответствующих определенному номеру отряда. Приглашение пройти в комнату в нашей модели будет подаваться вручную с помощью кнопки.

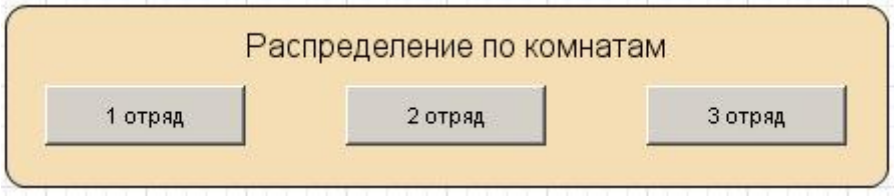
Создайте зону ожидания: `ожидание_распределения` возле концертной площадки с помощью замкнутой ломаной линии. Помните, что нельзя выходить за границы, отведенной для пешеходов зоны.

Добавьте на диаграмму класса объект `PedArea`, назовите его `pedArea_rasp` и свяжите с зоной ожидания `ожидание_распределения`.

Добавьте на диаграмму класса объект `PedWait` между объектами `pedSelectOutput_med`

и pedGoTo, назовите его pedWait_rasp, свяжите с pedArea_rasp и укажите тип: Ручной (по вызову метода free()).

Запустите модель и убедитесь, что все пионеры, прошедшие медосмотр собрались возле концертной площадки.



Создайте 3 кнопки, как показано на рис. 15.20. **Рис. 15.20**

В свойствах кнопки 1 в поле Действие напишите программный код на языке Ява:

```
for(Pioner p: pedArea_rasp.peds)
                          {if(p.division==1) pedWait_rasp.free( p ); };
```

Обратите внимание, что мы используем более простую в использовании "улучшенную" версию цикла **for**. Эта форма используется для прохождения (итерирования) по элементам массивов и коллекций. В нашем случае мы обратимся ко всем пешеходам, находящимся в зоне pedArea_rasp и проверим у каждого объекта типа Pioner поле division. Если это поле равно 1, то происходит вызов метода free() для этого конкретного пешехода и он покидает зону ожидания.

Несмотря на то, что пешеходы в моделируемом нами процессе имеют тип Pioner, объекты на блок-схеме будут продолжать считать их объектами класса Ped, и не позволят нам явно обращаться к полю division, которое есть только у класса Pioner. Чтобы сделать возможным доступ к собственным полям пешехода в динамических параметрах объектов блок-схемы следует указать имя класса пешехода в поле **Класс пешехода** этого объекта. Для этого в поле **Класс пешехода** объекта pedArea_rasp замените Ped на Pioner.

Напишите аналогичный код в свойствах кнопок 2 и 3, запустите модель и убедитесь, что пионеры покидают зону ожидания по соответствующей команде.

Теперь настало время распределить пионеров по комнатам. Для этого будем использовать уже знакомый нам объект PedSelectOutput, который следует добавить между объектами pedWait_rasp и pedGoTo, а также два новых объекта PedGoTo. Добавьте и соедините эти объекты как показано на рис. 15.21.

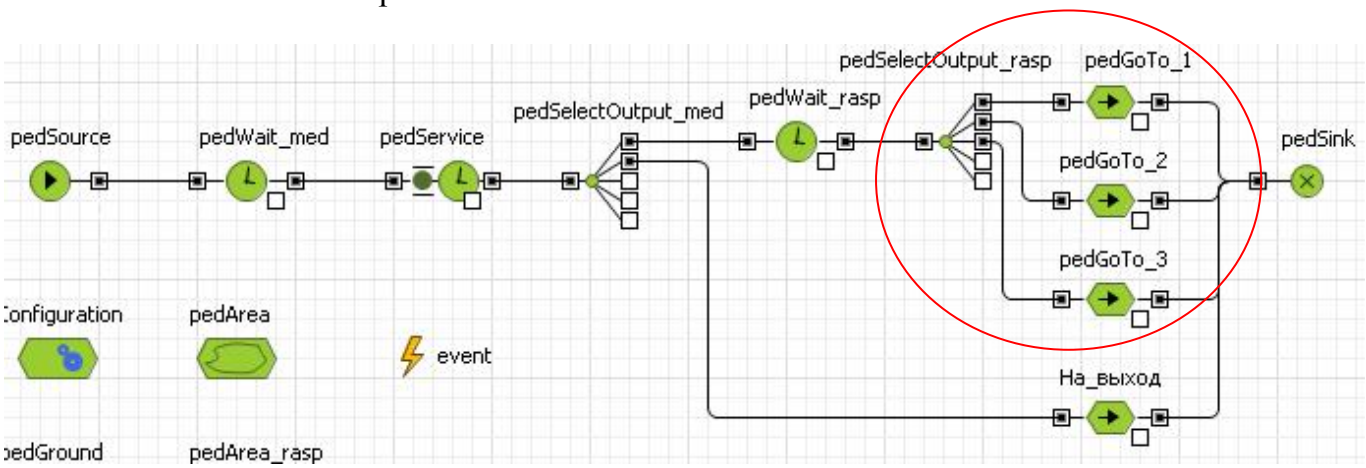


Рис. 15.21

Измените свойства объекта `pedSelectOutput`:

- **Имя**: `pedSelectOutput_rasp`;
- **Режим выбора порта:** `Условия (проверяются последовательно);`
- **Условие 1**: `((Pioner)ped).division == 1;`
- **Условие 2**: `((Pioner)ped).division == 2;`
- **Условие 3**: `((Pioner)ped).division == 3.`

Объект `pedSelectOutput` проверяет поле `division` у каждого пешехода и, в зависимости от значения этого поля, будет направлять их на тот или иной выход.

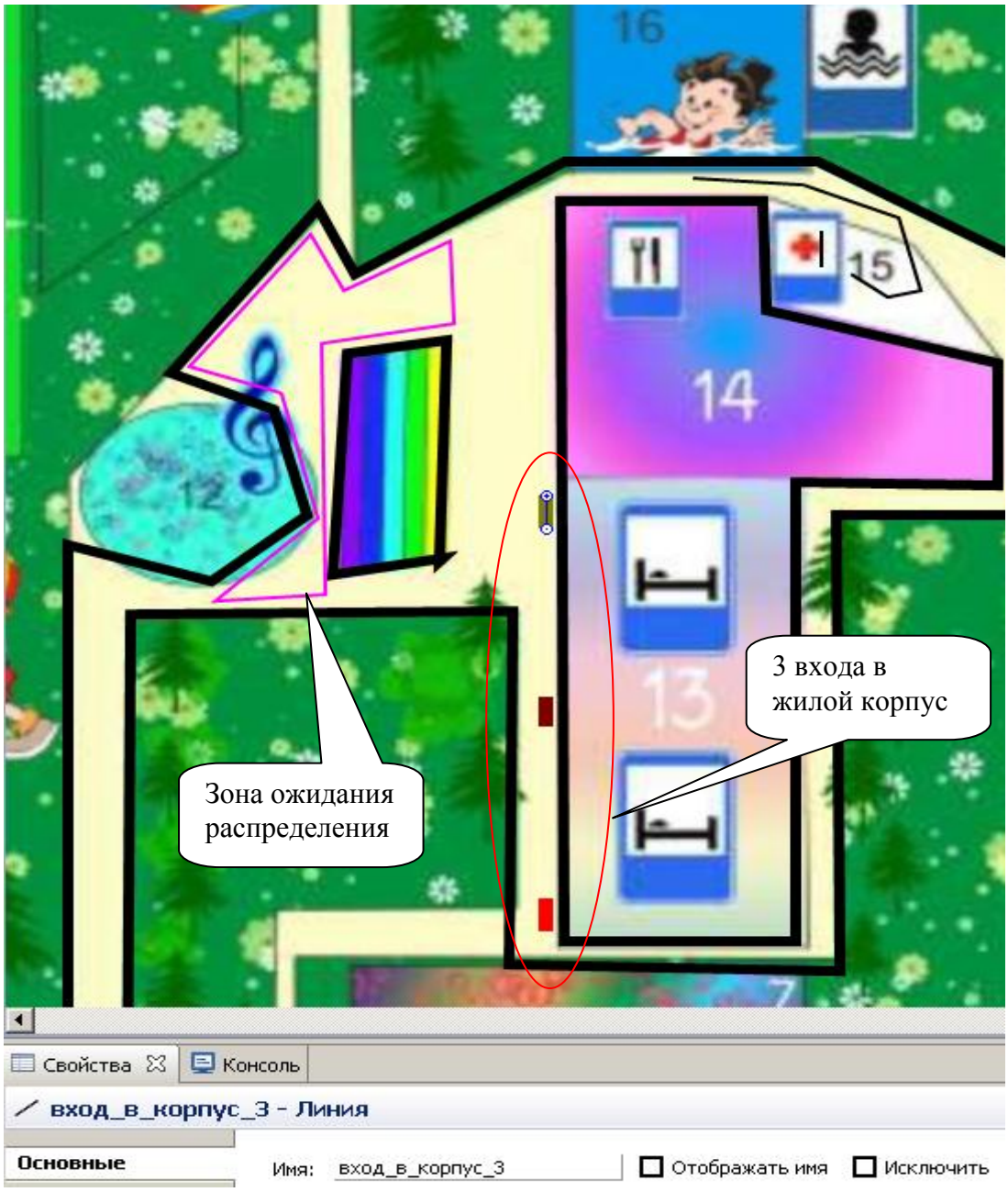


Рис. 15.22

Измените свойства объекта `pedGoTo`:

- **Имя**: `pedGoTo_1.`

Измените свойства объекта `pedGoTo1`:

- **Имя**: `pedGoTo_2;`

- **Цель (точка, линия)**: `вход_в_корпус_2`.

Измените свойства объекта `pedGoTo2`:

- **Имя**: `pedGoTo_3`;
- **Цель (точка, линия)**: `вход_в_корпус_3`.

На анимации нарисуйте две линии, которые будут символизировать входы в жилой корпус для второго и третьего отрядов, рис. 15.22. Назовите эти линии `вход_в_корпус_2` и `вход_в_корпус_3`.

Запустите модель и убедитесь, что пионеры расходятся по комнатам в зависимости от номера отряда.

15.3. КОНТРОЛЬНЫЕ ЗАДАНИЯ

1. Доработайте модель таким образом, чтобы при нажатии на кнопки 1, 2 и 3 пионеры проследовали на футбольную площадку, на баскетбольную площадку и на пляж. (4)
2. Добавьте в модель кнопку, при нажатии которой все пионеры, прошедшие медосмотр, пойдут в столовую. (4)
3. Добавьте в модель 3 параметра, значения которых будут отражать число пионеров в каждом из отрядов. (4)
4. Доработайте модель таким образом, чтобы можно было указать путь следования пионеров в медпункт мимо концертной площадки или мимо котельной, например с помощью переключателя или кнопки. (5)

ЛИТЕРАТУРА

1. Введение в математическое моделирование: уч.пособие/под ред. П.В.Трусова. – изд.:Логос, 2004, 440с.
2. Карпов Ю.Г. Имитационное моделирование систем. Введение в моделирование с AnyLogic 5.0. – СПб.: БХВ-Петербург, 2005. – 400 с.: ил.
3. Карпов Ю.Г. Изучение современных парадигм информационного моделирования в среде AnyLogic // Компьютерные инструменты в образовании. - СПб.: Изд-во ЦПО "Информатизация образования", 2005, N12, С. 03-14.
4. Копыльцов А.В. Компьютерное моделирование: сферы и границы применения. Практикум. Учебное пособие для 10-11 классов общеобразовательных учреждений. – СПб: «СМИО Пресс», 2005. – 144 с.
5. Лабораторный практикум по дисциплине «компьютерное моделирование» с использованием программы AnyLogic / А.Е.Осоргин. – Самара: СГПУ, 2008.

ИНТЕРНЕТ-РЕСУРСЫ

1. http://www.xjtek.ru – Официальный сайт разработчика системы AnyLogic. Дистрибутивы, примеры моделей, руководства, статьи и другая информация.
2. http://www.gpss.ru/ - сайт, посвященный имитационному моделированию систем.
3. http://headwire.narod.ru/ - здесь собраны самые разные примеры имитационных моделей, построенных в среде AnyLogic.
4. http://www.exponenta.ru/soft/Others/mvs/mvs.asp - здесь представлена альтернативная система компьютерного моделирования VMS: дистрибутив, руководство, примеры моделей, примеры уроков и другие материалы.